TCP Performance in the Presence of Congestion and Corruption Losses

Andrei V. Gurtov

Department of Computer Science, University of Helsinki
Department of Computer Science, University of Petrozavodsk

P.O. Box 26, FIN-00014 UNIVERSITY OF HELSINKI, Finland

E-mail: gurtov@cs.helsinki.fi

Abstract

The wireless environment of slow and lossy links presents a challenge for efficient data transport. We have performed an experimental evaluation of TCP in an emulated wireless environment. We consider a network model including a lossy wireless link and a last-hop router with a limited-size buffer. We have explored how well the state-of-art TCP perform, identified key reasons behind the behavior, and measured the effect of different optimizations. We experimented with TCP connections with different values of the initial window, receiver window, with or without SACK and New Reno over the emulated network with different error rates and buffer sizes. The experimental data is obtained with a state-of-art TCP implementation of the Linux operating system and a real-time network emulator Seawind. Our main result is a comparative study and analysis of different TCP optimizations.

1 Introduction

The number of nomadic users that access the Internet using wireless technology grows rapidly. Soon, in the upcoming era of mobile computing, every portable device will have a wireless interface and an IP address. With all advantages, mobile computing introduces an environment quite different from the one found in fixed networks, with limitations that come from physical properties of the wireless medium. The scarce radio bandwidth allows for a rather low link speed; miscellaneous external factors like fading of the radio signal may cause loss of data on the radio path. We believe that in the future, wireless connections will be widely used, but they will remain a different environment from wireline networks. In this work we focus on Wireless Wide Are Networks (WWANs). A typical WWAN is a cellular phone systems also capable of data transmission.

Many popular Internet applications including World-Wide Web (WWW), File Transfer Protocol (FTP) and email require reliable data delivery over the network. The Transmission Control Protocol (TCP) is the most widely used transport protocol for this purpose; traffic studies in the Internet report that the dominant fraction of the traffic belongs to TCP [30]. TCP was designed and tuned to perform well in fixed networks, where the key functionality is to utilize the available bandwidth and avoid overloading the network. However, nomadic users want to run their favorite applications that are built on TCP over a wireless connection, as well. Packet losses due to transmission errors, a long latency and sudden delays occurring on the wireless link may confuse TCP and yield a throughput far from the available line rate. Optimizing TCP for a wireless environment has been an active research area for the last few years. An excellent state-of-art classification of related work can be found in [16]. We are working in the area of pure transport layer solutions based on on modifications of TCP solely at the end points of a connection. This approach retains the end-to-end TCP connection semantics, but enhances the TCP protocol to make it perform better in the wireless environment. A description of TCP optimizations that we use in this paper can be found in [14].

This paper presents an experimental evaluation of TCP in an emulated wireless environment. We consider a network model including a lossy wireless link and a last-hop router with a limited-size buffer. Our goal is to explore how well the state-of-art TCP performs in this environment, what are the key reasons behind the behavior, and what is the effect of different TCP optimizations. We experiment with multiple error rates and buffer sizes over TCP connections with different optimizations. In the experiments the network is represented with a real-time network emulator Seawind and the real data communication using TCP. We have used the state-of-art TCP implementation of the Linux OS. Our main result is a comparative study of performance of different TCP optimizations. We also present a list of detected implementation faults, discuss anomalies in performance and give a detailed analysis of interesting cases.

The rest of the paper is organized as follows: in Section 2 we describe the Transmission Control Protocol, the assumed network architecture, the properties of wireless links and review the related work. In Section 3 we give specific performance problems we focus on. Section 4 specifies the network and workload model. In Section 5 we present our measurement setup and in Section 6 we illustrate and analyze the results of our experiments.

2 TCP over wireless links

2.1 Transmission Control Protocol

The Transmission Control Protocol (TCP) [23, 9, 5] is the most used transport protocol in the Internet. TCP provides applications with reliable byte-oriented delivery of data on the top of the Internet Protocol (IP). TCP sends user data in *segments* not exceeding the Maximum Segment Size (MSS) of the connection. Each byte of the data is assigned a unique sequence number. The receiver sends an acknowledgment (ACK) upon reception of a segment. TCP acknowledgments are cumulative; the sender has no information whether some of the data beyond the acknowledged byte has been received. TCP has an important property of *self-clocking*; in the equilibrium condition each arriving ACK triggers a transmission of a new segment. Data are not always delivered to TCP in a continuous way; the network can lose, duplicate or re-order packets. Arrived bytes that do not begin at the number of the next unacknowledged byte are called *out-of-order* data. As a response to out-of-order segments, TCP sends *duplicate acknowledgments* (DUPACK) that curry the same acknowledgment number as the previous ACK. In combination with a retransmission timeout (RTO) on the sender side, ACKs provide reliable data delivery [9]. The retransmission timer is set up based on the smoothed *round trip time* (RTT) and its variation. RTO is *backed off* exponentially at each unsuccessful retransmit of the segment [21]. When RTO expires, data transmission is controlled by the slow start algorithm described below. To prevent a fast sender from overflowing a slow receiver, TCP implements the flow control based on a *sliding window* [29]. When the total size of outstanding

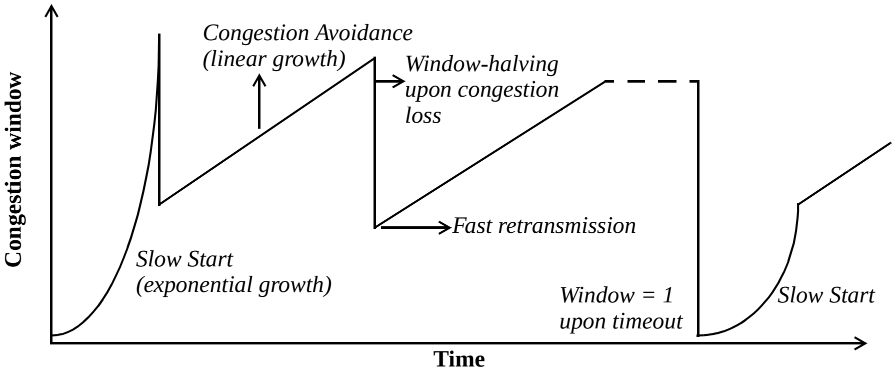


Figure 1: Congestion control in TCP [6].

segments, *segments in flight* (FlightSize), exceeds the window advertised by the receiver, blocked until ACK that opens the window arrives.

Early in its evolution, TCP was enhanced by *congestion control* mechanisms to protect the network against the incoming traffic that exceeds its capacity [15]. A TCP connection starts with a *slow-start* phase by sending out the *initial window* number of segments. The current congestion control standard allows the initial window of one or two segments [5]. During the slow start, the transmission rate is increased exponentially. The purpose of the slow start algorithm is to get the "ACK clock" running and to determine the available capacity in the network. A *congestion window* (cwnd) is a current estimation of the available capacity in the network. At any point of time, the sender is allowed to have no more segments outstanding than the minimum of the advertised and congestion window. Upon reception of an acknowledgment, the congestion window is increased by one, thus the sender is allowed to transmit the number of acknowledged segments plus one. This roughly doubles the congestion window per RTT. The slow start ends when a segment loss is detected or when the congestion window reaches the *slow-start threshold* (ssthresh). When the slow start threshold is exceeded, the sender is in the *congestion avoidance* phase and increases the congestion window roughly by one segment per RTT. When a segment loss is detected, it is taken as a sign of congestion and the load on the network is decreased. The slow start threshold is set to the half of the current congestion window. After a retransmission timeout, the congestion window is set to one segment and the sender proceeds with the slow start. Figure 1 shows a possible behavior of the congestion window for a TCP connection.

TCP recovery was enhanced by the *fast retransmit* and *fast recovery* algorithms to avoid waiting for a retransmit timeout every time a segment is lost [27]. Recall that DUPACKs are sent as a response to out-of-order segments. Because the network may re-order or duplicate packets, reception of a single DUPACK is not sufficient to conclude a segment loss. A threshold of three DUPACKs was chosen as a compromise between the danger of a spurious loss detection and a timely loss recovery. Upon the reception of three DUPACKs, the fast retransmit algorithm is triggered. The DUPACKed segment is considered lost and is retransmitted. At the same time congestion control measures are taken; the congestion window is halved. The fast recovery algorithm controls the transmission of new data until a non-duplicate ACK is received. The fast recovery algorithm treats each additional arriving DUPACK as an indication that a segment has left the network. This allows to inflate the congestion window temporarily by one MSS per each DUPACK. When the congestion window is inflated enough, each arriving DUPACK triggers a transmission of a new segment, thus the ACK clock is preserved. When a non-duplicate ACK arrives, the fast recovery is completed and the congestion window is deflated.

New Reno [13] is a small but important modification to the TCP fast recovery algorithm. “Normal” fast recovery suffers from timeouts when multiple packets are lost from the same flight of segments [12]. New Reno can recover from multiple losses at the rate of one packet per round trip time. If during the fast recovery the first non-duplicate ACK does not acknowledge all outstanding data prior to the fast retransmit, such an ACK is called a *partial acknowledgment*. The New Reno algorithm is based on an observation that a partial acknowledgment is a strong indication that another segment was also lost. During the *recovery phase* New Reno retransmits the presumably missing segment and transmits new data if the congestion window allows it. The recovery phase ends when all segments outstanding before the fast retransmit are acknowledged or the retransmission timer expires.

RFCs describing the TCP protocol leave many issues unspecified and

TCP implementations differ in how they behave under similar conditions. For a long time, the reference implementation has been Reno TCP found in the Unix BSD4.3 operating system [31]. Modern TCP implementations differ significantly from Reno. The current family of BSD OSes is derived from Unix BSD4.4 with TCP-Lite implementation [17]. For the baseline in our analysis we wanted to select a state-of-art TCP implementation that is both widely used in the Internet and has the source code available for analysis and modification. We chose Linux as a popular operating system with the source code available. Due to a large amount of independent developers interested in Linux, implementations of new features are quick to appear for Linux. The TCP implementation in earlier versions of Linux had problems with conforming to standards [20]. We have detected, evaluated and corrected a number of misbehavior problems. We believe that after these fixes we obtained a TCP that behaves reasonably with regard to standards. A recent work gives the requirements for a TCP implementation to be used for TCP research [4]. Our baseline TCP (described in detail in [14]) satisfies these requirements.

2.2 Properties of Wireless Links

Many wireless links are *slow*, have high latency and may have high error rates. These link characteristics adversely affect the TCP performance. The line rate of a wireless link may not exceed some tens of kilobits per second. The *latency*, the propagation delay, of wireless links is typically high. The latency comes from the special transmission schemas and processing delays the network equipment. The total one-way latency in GSM sums up to 200-300 ms. Note, that we do not include the transmission delay into the link latency. Thus the *round-trip time* is defined as the sum of transmission and propagation delays in both directions. Some wireless links impose a significant amount of data corruption due to transmission *errors*. For example, in the transparent GSM data service the residual bit error rate (BER) of the link can be as high as 10^{-3} after the Forward Error Correction (FEC) [19]. The delay-bandwidth product is an important characteristic of a network [26]. It defines the minimum size of data in flight to utilize the available network bandwidth, the pipe capacity. Networks with a large delay-bandwidth product, for example including satellite links, demand special attention from the transport protocol. For example, the slow start phase of TCP can be time-consuming in such networks [3]. In our environment, the delay-bandwidth product is small, close to one kilobyte. In the slow start, the pipe capacity is filled already after one-two RTTs .

2.3 Network Architecture

Rather than selecting one particular network architecture and developing a detailed model that would reflect the behavior of this network we attempt to build a generic model that would be suitable for all wireless networks with similar characteristics. We are interested in the issue how a nomadic user can use Internet services via a wireless network. In a scenario shown in Figure 2, the wireless network plays the role of an access network from the Internet point of view. It is also possible for a nomadic user to exchange data with another mobile user, so that two wireless links are present on the data path. We do not consider such configuration is this thesis, assuming that the access to a remote host in the Internet would be the dominant case.

The wireless link is often the bottleneck in the path of a data flow, because fixed networks are fast and reliable compared to the capabilities of the wireless link. When data packets flow from the relatively fast Internet to the slow wireless link they are buffered in the last-hop router which connects the wireless link to the Internet. This router plays a significant role in the end-to-end TCP performance because congestion data losses are most likely to happen at the bottleneck queue. A limited number of buffers can be allocated in the last-hop router per user. This buffer space is shared among connections of the same user, but there is no interference between the connections of different users. A similar network architecture was considered, when three buffers are available per user [25].

The wireless link in our environment imposes corruption losses. We assume that all data with transmission errors are detected and discarded at the wireless link. We also assume no error recovery and no variable delays on the link. Thus, different patterns of link errors is the only non-deterministic element in our environment. The Global System for Mobile Communication (GSM) is a widely successful effort to build a WWAN system with millions of users in Europe and worldwide [19, 24]. It maps well to our generic model shown in Figure 2.

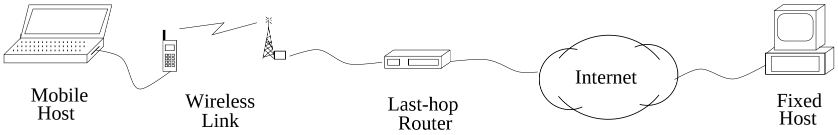


Figure 2: Network Architecture.

3 Problem Description

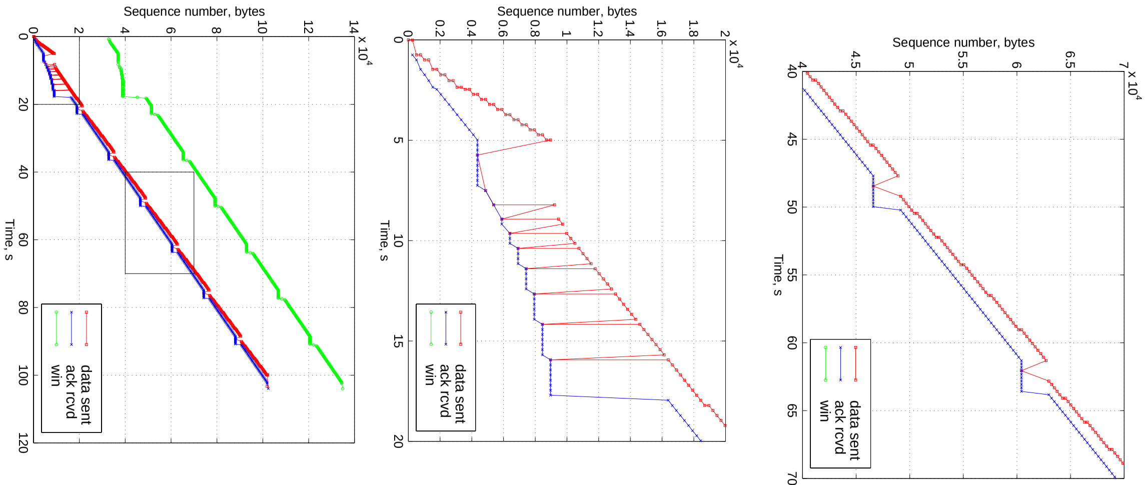
In this section we outline the specific problems of TCP over wireless links we focus on in the rest of the thesis.

3.1 Congestion Losses

In this section we discuss the occurrence of congestion losses and their effect on TCP. We use the term *congestion* for the time period when many packet losses occur due to a buffer overflow, even in the case of a single connection. We first look at a typical TCP connection over a limited-size buffer, but in an error-free environment. Figure 3 shows a baseline TCP connection when the buffer space is limited to seven packets. Two phases of the connection are clearly visible. In the first phase, which lasts approximately 20 s, the connection starts up aggressively, creates congestion, loses a large number of packets and recovers them. We call this phase *the start-up buffer overflow* hereinafter. In the second phase, the connection proceeds smoothly with the periodic loss of a packet. This is referred to as the *steady state* of the connection. During this phase the connection goes through periodic congestion avoidance cycles following the linear increase – multiplicative decrease policy [27].

Start-up buffer overflow. Let us look at the start-up buffer overflow which is also known as the slow-start overshoot [18]. Figure 3(b) zooms on the start-up buffer overflow. Ten segments are lost and retransmitted. The important points to notice on the figure are: when congestion occurs, when the first packet loss is detected, and how segment losses are recovered. Questions about the start-up buffer overflow are "why does it happen", "what is the negative effect", and "how can it be prevented".

The optimal router buffer size. The maximum size of the queue in the router has a significant effect on the connection. A router buffer,



(a) Complete connection

(b) Start-up zoom

(c) Steady-state zoom

Figure 3: An error-free TCP connection over the router buffer of seven packets.

which is too small, can result in a smaller FlightSize than needed by TCP to recover well from packet losses. The size, which is too large, leads to the heavy start-up buffer overflow and overbuffering. One paper has estimated 1.5*RTT*bandwidth as the optimal value for the buffer size [17].

Overbuffering. The situation when significantly more packets are in flight than is required to fill the available network capacity is called *overbuffering*. Overbuffering does not necessarily cause congestion. If the number of packets injected into the network equals the number of packets leaving the network, no congestion take place. However, having a large number of packets buffered in the network has several drawbacks [17]. If buffers in the network are full, there is no capacity left to accommodate traffic bursts. Some applications using TCP generate bursty traffic. In addition, the TCP protocol itself can inject packets in bursts. Another drawback is a poor service for interactive applications, because the end-to-end delay on the overbuffered path can be huge. Finally, the data in the network can become stale, when a user aborts the data transfer, for example using a stop button in a web browser. Due to these reasons overbuffering should be avoided.

Fair sharing of resources. Tail-drop routers are known to have problems with sharing the bandwidth between connections in a fair way [8]. When two or more TCP connections share the same router buffer, one connection can starve while other connections monopolize the resources. This situation is referred to as *lock-out* and occurs due to timing effects. We would like to avoid this problem in our environment.

3.2 Corruption Losses

Performance problems of TCP at the presence of error losses are well known [7]. Upon a loss detection, TCP always reduces the transmission rate, as the reason of the packet loss, congestion or corruption, is not known. When the level of error losses is low, they do not have a notable effect on the performance. At the moderate level of error losses, TCP underestimated the available network bandwidth. When the level of error losses is high, most of time the connection is idle waiting for a retransmission timeout to expire. In the worst case, the connection is terminated, when the maximum number of retransmission is exceeded. Not only the rate of the error losses is important, but also the burstiness [17].

In general, TCP suffers more when errors are bursty rather when they are uniformly distributed. Recommendations for using the TCP algorithms and control parameters at the presence of error losses is given in [11]. We have identified three patterns of error losses to be studied.

Single Errors. Normally, single-packet error drops do not have a significant effect on the TCP behavior, except for a few special cases. We will try to locate such interesting cases and analyze them.

Random errors. We will try to identify levels of the uniformly-distributed packet error rate when error losses have no effect on performance, when the link bandwidth is underestimated, when most time is spent in RTOs and when the connection is terminated. We will study how different TCP optimizations affect the performance for varying size of the router buffer and the error loss rate.

Burst errors. It is interesting to study the effect of burst errors on TCP. An error loss rate of one percent does not normally affect TCP performance, if uniformly distributed. However, the same error rate when errors occur in clusters can adversely impact performance. We expect TCP to perform badly during an error burst; also performance after the burst ends can be hampered.

3.3 OS-Related Problems

State-of-art TCP performance. Most of the related TCP research concentrated on evaluation of TCP performance of the Reno TCP implementation or an abstract TCP model in the *ns* simulator [22]. However, from the point of view of an end user, it is much more important how their currently installed operating system performs under the given conditions. The upcoming Linux version 2.4 differs from Reno in many ways. Thus, it is actual to evaluate how the state-of-art TCP implementation performs on wireless links.

Conformance of Linux. There is a stereotype among researchers about the TCP implementation in Linux, that it does not conform to standards. Indeed, earlier releases of the Linux kernel showed malicious behavior and were even named as an incoming danger to the Internet [20]. The Linux networking code has undergone significant changes since version 1.0, and a large number of independent developers have verified and improved Linux. Today, when Linux is widely used on Internet servers, it is actual to locate and fix the remaining inconsistencies with TCP standards produced by IETF.

4 Performance Model

In this section we describe the network model for the network architecture depicted in Figure 2 on page 8. The network model is implemented in a real-time emulator. The model of downlink and uplink channels is shown in Figure 4. The last-hop router is modeled as a queue. The wireless link is modeled as a combination of the transmission and propagation delays; error losses are modeled as packet drops. The uplink and downlink directions in our model are independent.

In the downlink direction, packets arriving to the emulator are placed in the queue. The maximum queue length can be limited; when an overflow happens, packets are tail-dropped. The RED algorithm can be used to actively control the queue length. Packets are taken from the head of the queue one-by-one for "transmission" over the link. The length of the transmission delay is computed according to the line rate and the packet size. When the transmission delay for a packet is completed, the packet is moved to the propagation delay node. The length of the propagation delay is the same for all packets independently of packet size. Several packets can be in the propagation delay node simultaneously. Error losses are modeled by dropping packets after the propagation delay. If a packet was not dropped, it is sent out from the emulator.

On the uplink direction, the transmission and propagation delay nodes are used in the same way as for downlink. We assume no queueing in the uplink direction. With our workload model the chance of two or more packets (i.e. acknowledgments) to be queued in the uplink direction is negligible. Error losses are modeled in the same way as for downlink. We assume the link rate of 9600 bps and the propagation delay of 200 ms hereinafter. Our error model assumes that all corrupted packets are detected and discarded on the wireless link, that is, no corrupted packets are delivered to the IP layer. The packet drop probability is independent of the packet size. This may be considered inaccurate because, for example, the loss rate of small ACKs is the same as of large packets. However, this is the case, for example, when acknowledgments are piggybacked to large data packets. The type of workload used for evaluation of different

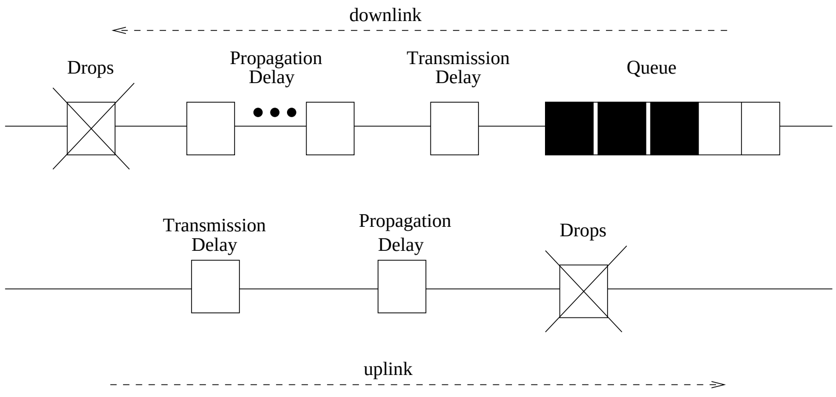


Figure 4: The model of downlink and uplink data channels.

solutions has a significant effect on the results. In most of our tests we use a single unidirectional bulk data transfer as the workload. In the limited set of tests we use two such transfers.

Table 1: Features in the baseline TCP.

Feature	Availability
Fast Retransmit, Fast Recovery	ON
New Reno	ON
Initial Window Size, segments	2
SACK	OFF
MSS, bytes	256
Timestamps	OFF
Delayed Acks	ON
Advertised Window, kilobytes	32
PPP Compression	OFF
Control Block Interdependence	OFF

Recent studies of the Internet traffic indicate that both the New Reno algorithm and the Selective Acknowledgment (SACK) option are widely used nowadays [2]. We have decided to include the New Reno algorithm into the baseline, but leave SACK as one of optimizations. This corresponds to the current practise and makes easier comparisons with the related work. Table 1 presents a list of relevant parameters that we assume in the baseline TCP if not mentioned otherwise. More details and justification behind such a choice are give in [14].

5 Experimental Design

5.1 Test Environment

In this section we describe our test environment: how the network model shown in Figure 4 on page 13 is realized in our emulator, and how the workload generator (TTCP) is positioned. Figure 5 shows the protocol layering in our setup. The workload source, the Seawind emulator [1] and the workload sink are each located on a separate computer in Ethernet LAN (802.3). The test TCP traffic is encapsulated into a regular TCP/IP connection. Seawind runs on a normal Linux workstation, as it gets the test TCP traffic from a standard socket interface.

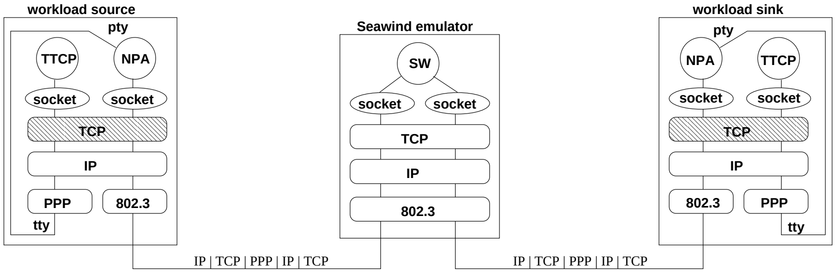


Figure 5: Seawind protocol stack. The modified TCP code is dashed.

The Seawind emulator implements the network model shown in Figure 4 by delaying and dropping TCP segments in real time. The downlink and uplink channels are parameterized independently. The maximum length of the queue in packets is controlled by a parameter; in the "unlimited" mode the length is only bound by the available memory. For the purpose of our experiments, it can be considered infinite. One packet is considered currently "in transmission" and is not counted into the queue limit. Through the rest of the thesis we assume that the "router buffer size" does not include this packet. Link errors can be emulated as a fixed pattern (e.g. 5th and 12th packets are dropped) or with a specified dropping probability per each packet.

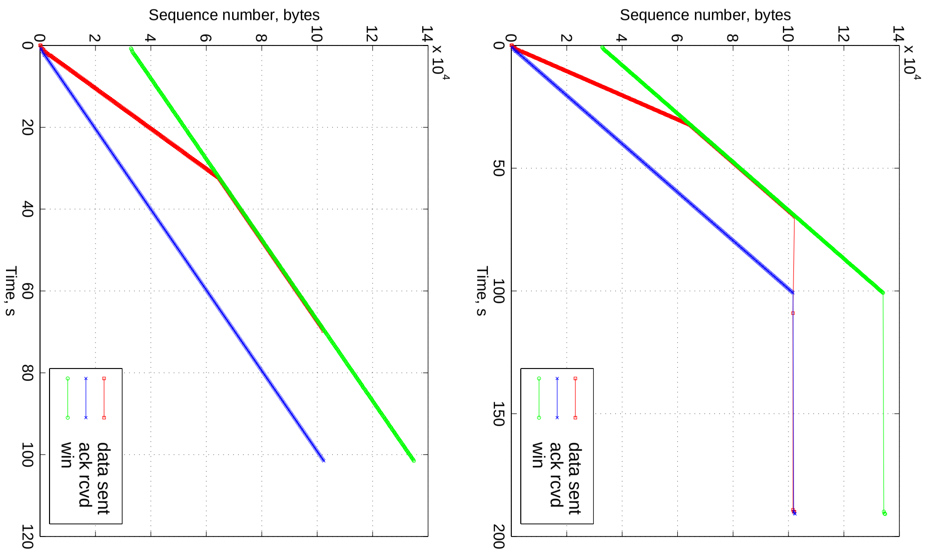
The workload source and sink computers use the TCP implementation under study. The Point-to-Point (PPP) protocol is used as a link service for a TCP connection under study. This corresponds to a real-world situation, as most dial-up users employ PPP. We have disabled all kinds of header and data compression, as well as escaping of control characters in PPP. The PPP/IP/TCP traffic is forwarded by the Network Protocol Adapter (NPA) via a TCP/IP connection to the Seawind emulator.

We use a modified *TTCP* tool for generating traffic for TCP connections. TTCP is a popular public domain tool for testing the end-to-end throughput by sending a high volume of data over the network [28]. TTCP is commonly used as a workload generator for bulk data transfers. We have made several extensions to TTCP to make it more suitable to our needs. In our tests we used 400 writes of 256-byte data blocks, which results in a 100-kilobyte transfer.

6 Measurement Results and Analysis

In the first set of tests we have shown that an unlimited buffer size in the router is not desirable. It creates the overbuffering problem and worsens the recovery from sudden data losses. Figure 6(a) shows the behavior of the baseline TCP when no congestion or error losses are present. The achieved throughput of 1002 bytes per second (Bps) is close to the maximum taking into account TCP/IP/PPP header overhead and the line rate of 1200 Bps. When the FlightSize equals the receiver window (32 kilobytes), 127 segments are queued in the network. In our environment, where the FlightSize of a few segments is sufficient for utilizing the pipe capacity, this is undesirable for reasons discussed in Section 3.1. We will add here that the measured RTT also includes the queuing time and thus is highly inflated with regard to the actual RTT of the link.

An adaptive link layer can change the strength of the radio channel coding when the number of transmission errors on the link changes [17]. A stronger coding schema allows to reduce the packet loss rate over the link



(a) No segment losses.

(b) Three last segments and the first retransmission are lost.

Figure 6: TCP behavior with the unlimited router buffer size.

at the expense of the reduced line rate. A complete lack of segment losses creates the overbuffering problem and is not desirable in our environment. In order to keep the FlightSize at the optimal level, TCP needs a low rate of segment losses. Losses due to congestion at the router buffer and losses due to link errors are treated in the same way by TCP. Thus a link can provide a low level of error losses with a beneficial effect on TCP. For an adaptive link it means that the channel coding can be kept as weak as possible to maximize the line rate, leaving a low level of packet losses to be noticed and corrected by TCP.

Figure 7 shows that a range of the router buffer sizes of 3-12 packets gives the optimal performance on an error-free link. Variations in throughput for the buffer size of 3-12 packets have a simple explanation. If a congestion control cycle happens to occur at the end of a connection, it causes a retransmission timeout. For example, the connection suffers from the RTO for the six-packet router buffer, but does not for the five-packet buffer. Whether RTO occurs or not for the given router buffer size depends on the amount of data sent over the connection. Starting with the buffer size of 15 packets, performance decreases. The start-up buffer overflow for a 15-packet buffer already lasts for 40 seconds.

We have located and analyzed the cases where a loss of a single packet significantly affects the performance of TCP. In case segment losses *do* suddenly happen on an overbuffered link (with a large router buffer), the recovery time is long, as shown in Figure 6(b). Four last segments in a connection are lost (three original and the first retransmission). The retransmission of a lost segment happens only after 40 seconds after its loss. Another interesting example is a packet loss in the beginning of a connection that may actually have a positive effect. The packet loss triggers congestion avoidance measures; the start-up buffer overflow is avoided, and the connection proceeds smoothly for its lifetime. More examples can be found in [14].

Our largest group of tests is with random errors on the link. The error rate was set to 2, 5, and 10 % with a queue limit of 3, 5, 7, 10, and 20 packets. Table 2 lists the optimizations we have experimented with. We found that the optimal buffer size to be 7-10 packets. Throughput of the baseline TCP is adequate at a 2 % error rate, but is only half of the line rate for a 10 % loss rate. TCP with SACK performed significantly better than other modifications under all conditions, especially at higher loss

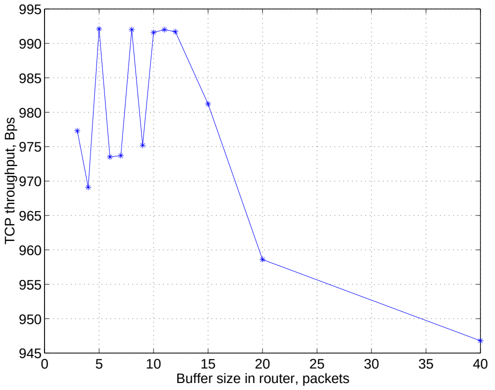


Figure 7: Throughput vs. buffer size for the baseline TCP over an error-free link.

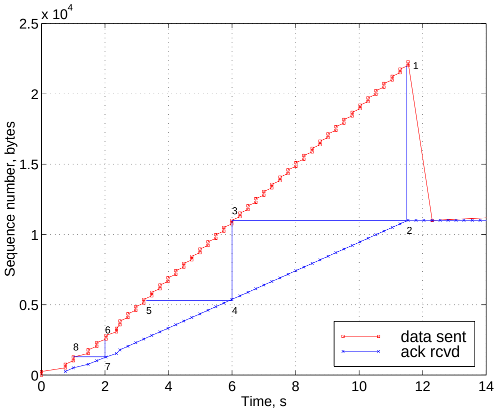


Figure 8: Analysis of the start-up buffer overflow. The baseline TCP over a 20-packet buffer.

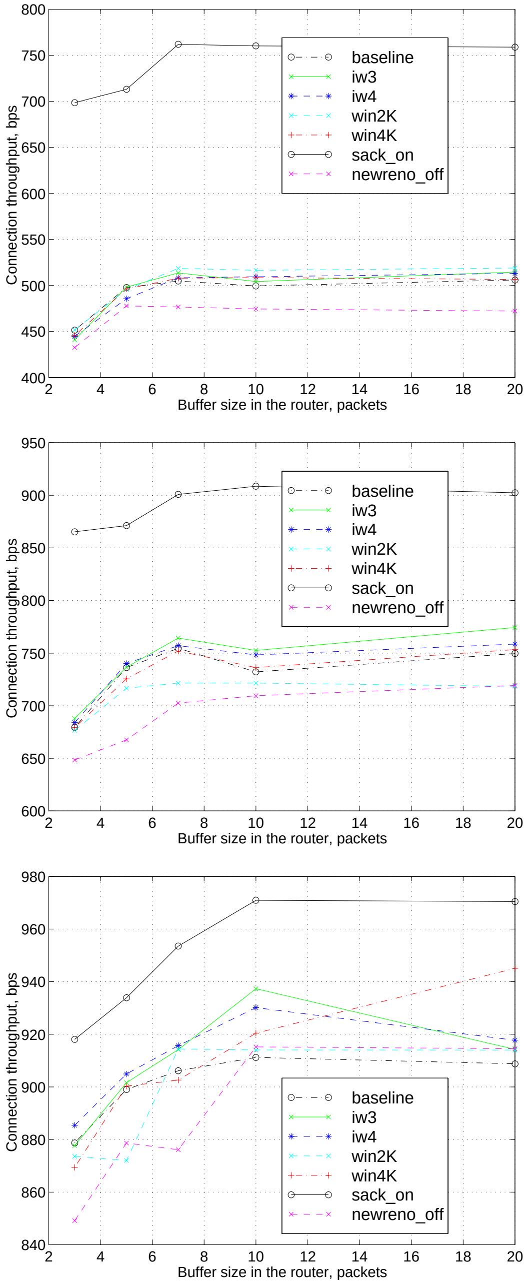
Table 2: TCP optimizations tested with random errors.

Label	iw segm.	win bytes	sack on/off	newreno on/off	mss bytes
baseline	2	32696	off	on	256
iw3	3				
iw4	4				
win2K		2048			
win4K		3840			
sack_on			on		
newreno_off				off	
mss536					536

rates. The increased initial window gives slightly better throughput than the baseline. A small receiver window (2 kilobytes) decreases throughput, especially for smaller buffer sizes. A moderate window (4 kilobytes) is beneficial for larger buffers. Disabling New Reno is helpful at a low error rate and larger buffer sizes; in other cases it is worse. In general, our results are coherent with a previous evaluation of Reno, New Reno, and SACK TCP at the presence of error losses [12].

We have also studied what time it takes to transmit the first 15 kilobytes of data in the bulk data connections. In this way we can estimate the performance of a transaction-type traffic. The performance picture is different than for whole connections. The optimal buffer size varies with error rates and TCP optimizations. A limited receiver window and disabled New Reno are quite helpful at the low error loss rate. SACK performs better than other modifications in this case, as well.

We have studied the effect of an error burst on the TCP connection. Typically, little or no data gets through during the burst already at a 20 % packet loss rate. After the bursts ends, the transmission is resumed immediately, except when RTO was backed off several times during the burst. In the later case the connection is idle approximately for half of the burst length after the link quality returns to normal. The likelihood of the RTO back off is increased with the buffer size. This is because most packets sent during the burst are retransmissions, but not the new data.



(a) Good radio conditions (2 % error rate)

(b) Mediocre radio conditions (5 % error rate)

(c) Poor radio conditions (10 % error rate)

Figure 9: Throughput of TCP at the end of connections. Note the different scale.

In such a case no valid RTT sample can be collected and RTO is more likely backed off several times. Thus a smaller buffer size is preferable for a link where error bursts are possible.

We found that RED worsens the performance when only a single TCP connection is present. This is because the moving average of the queue size does not react timely to the start-up buffer overflow, and late packet drops only worsen the recovery. For two concurrent TCP connections, RED improves the throughput and the fairness among the connections, but only for large buffer sizes (20, 40 packets). We have provided the detailed analysis of the start-up buffer overflow and have suggested using a dual threshold drop policy to prevent it. However, its implementation and evaluation is left for future work. A deployment of the Explicit Congestion Notification (ECN) could make RED more attractive in our environment, because ECN avoids congestion-related losses. The implementation of ECN in our emulator and a performance evaluation is left for future work.

Here we present the detailed analysis of the start-up buffer overflow shown in Figure 8. The segment marked *1*, is the last segment transmitted before the overflow is detected after the third DUPACK (*2*) for the lost segment (*3*). The number of segments between *1* and *2* is the FlightSize when a packet loss is detected, it is about twice as large as the router buffer. Approximately, every second segment from this flight is lost due to the buffer overflow. The time between points *2* and *3* shows the current RTT of the link, it is about six seconds. The number of segments between *3* and *5* is the FlightSize at the moment when the first loss occurs. Thus, the segment marked *5* is the latest segment to be dropped by an active queue management algorithm, so that a packet loss is detected before point *3*. When a loss of *5* would be detected, the FlightSize is not grown anymore and additional losses are prevented. The number of segments between points *6* and *7* is the minimum FlightSize to trigger the fast retransmit, four segments. Thus, segment *8* is the earliest segment of the connection, which loss would be recovered by the fast retransmit. The segments before that can be recovered only by the retransmission timeout. Thus, if we drop a segment between points *5* and *6* we avoid the buffer overflow at the cost of a single packet drop. It is better to select a packet closer to point *5* to avoid underutilization of the link. A practical implementation of such a policy could define a soft queue limit in the router, for example ten packets. The hard limit can be two-three times larger than the soft limit. When the current queue size reaches the soft limit, a single packet is dropped. When the TCP sender detects a packet loss, it decreases the transmission rate and the buffer overflow is prevented. If the hard queue limit is reached, the router drops all arriving packets. Extending this algorithm to work well for a few concurrent connections requires a counter or a timer-based mechanism to determine when to drop another packet in case the load is not decreased, i.e. a previously dropped packet was not from the most aggressive connection. Some heuristics that favors connections with small packets can be implemented to protect interactive flows. The suggested algorithm is similar to the Dual Threshold Early Packet Discard [10].

We have collected some empirical evidence suggesting that the more careful version of the "bug fix" for preventing multiple fast retransmits should be implemented in all TCPs. In the first scenario, multiple fast retransmits are caused by a long delay on the link and a spurious timeout and in the second scenario, by a loss of a block of segments in the middle of the flight. New Reno adversely affects the performance at the presence of multiple fast retransmits.

7 Conclusion

We have performed an experimental evaluation of the state-of-art TCP implementation in the emulated wireless environment. We have addressed the problem of start-up buffer overflow, determined a range of optimal buffer size in the last-hop router, demonstrated the negative effect of overbuffering and the effect of different patterns of error losses on TCP performance. We have also experimented with RED algorithm in the last-hop router and did not find it useful in our environment.

References

[1] T. Alanko, A. Gurtov, M. Kojo, and J. Manner. Seawind: Software requirements document. University of Helsinki, Department of Computer Science, September 1998.

[2] M. Allman. A web server's view of the transport layer. *ACM Computer Communication Review*, 30(5), October 2000.

[3] M. Allman, S. Dawkins, D. Glover, J. Griner, D. Tran, T. Henderson, J. Heidemann, J. Touch, H. Kruse, S. Ostermann, K. Scott, and J. Semke. Ongoing TCP research related to satellites. IETF RFC 2760, 2000.

[4] M. Allman and A. Falk. On the effective evaluation of TCP. *ACM Computer Communication Review*, 5(29), Oct. 1999.

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

[6] H. Balakrishnan. *Challenges to Reliable Data Transport over Heterogeneous Wireless Networks.* PhD thesis, Computer Science Division, Univ. of California at Berkeley, Berkeley, CA, Aug. 1998.

[7] H. Balakrishnan, R. Katz, V. Padnamabhan, and S. Seshan. Improving performance of TCP over wireless networks. Technical report, Texas A&M University, 1996.

[8] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on queue management and congestion avoidance in the Internet. IETF RFC 2309, Apr. 1998.

[9] R. Braden. Requirements for internet hosts – communication layers. IETF RFC 1122, October 1989.

[10] R. Cohen and Y. Hamo. Balanced packet discard for improving TCP performance in ATM networks. Tel Aviv, Israel, Mar. 2000.

[11] S. Dawkins, G. Montenegro, M. Kojo, V. Magret, and N. Vaidya. End-to-end performance implications of links with errors. Internet draft "draft-ietf-pilc-error-06.txt", November 2000. Work in progress.

[12] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *ACM Computer Communication Review*, July 1996.

[13] S. Floyd and T. Henderson. The NewReno modification to TCP's fast recovery algorithm. IETF RFC 2582, April 1999.

[14] A. Gurtov. TCP performance in presence of congestion and corruption losses. Master's thesis, Department of Computer Science, University of Helsinki, December 2000.

[15] V. Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM '88*, pages 314–329, August 1988.

[16] R. Ludwig. A case for flow-adaptive wireless links. Technical Report CSD-99-1053, University of California, Berkeley, 1999.

[17] R. Ludwig. *Eliminating Inefficient Cross-Layer Interactions in Wireless Networking*. PhD thesis, Aachen University of Technology, April 2000.

[18] M. Mathis and J. Mahdavi. Forward acknowledgement: Refining TCP congestion control. In *Proceedings of ACM SIGCOMM '96*, volume 26, October 1996.

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

[20] V. Paxson. Automated packet trace analysis of TCP implementations. In *Proceedings of the ACM SIGCOMM Conference: Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM-97)*, volume 27 of *Computer Communication Review*, pages 167–180, Cannes, France, Sept. 14–18 1997. ACM Press.

[21] V. Paxson and M. Allman. Computing TCP's retransmission timer. IETF RFC 2988, November 2000. Standards Track.

[22] K. Poduri and K. Nichols. Simulation studies of increased initial TCP window size. IETF RFC 2415, September 1998.

[23] J. Postel. Transmission control protocol. IETF RFC 793, 1981. Standard.

[24] M. Rahnema. Overview of the GSM system and protocol architecture. *IEEE Communications Magazine*, 31:92–100, April 1993.

[25] T. Shepard and C. Partridge. When TCP starts up with four packets into only three buffers. IETF RFC 2416, Sept. 1998.

[26] W. Stallings. *Data and Computer Communications.* Prentice-Hall, sixth edition, 2000.

[27] W. Stevens. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. IETF RFC 2001, Jan. 1997.

[28] R. H. Stine. FYI on a network management tool catalog: Tools for monitoring and debugging TCP/IP internets and interconnected devices. IETF RFC 1147, Apr. 1990.

[29] A. S. Tanenbaum. *Computer Networks.* Prentice-Hall International, 1996.

[30] K. Thompson, G. J. Miller, and R. Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network*, 11(6):10–23, November/December 1997.

[31] G. Wright and W. Stevens. *TCP/IP Illustrated, Volume 2; The Implementation.* Addison Wesley, 1995.