Effect of Delays on TCP Performance

Andrei Gurtov
Cellular Systems Development, Sonera Corporation

Key words: TCP, delay, GPRS, Eifel.

Abstract: This paper has several contributions. First, we report that long sudden delays during data transfers are not uncommon in the GPRS wireless WAN. Long sudden delays can lead to spurious TCP timeouts and unnecessary retransmissions. Second, we show that the New Reno algorithm increases the penalty of spurious TCP timeouts and that an aggressive TCP retransmission timer may trigger a chain of spurious retransmissions. Third, we test how four widely deployed TCP implementations recover from a spurious timeout and notice that two of them have severe problems to recover. Finally, we discuss several existing ways to alleviate the problems.

1. INTRODUCTION

The number of nomadic users that access the Internet using wireless technology grows rapidly. Nowadays Wireless Wide Area Networks (W-WAN) are the primary means for nomadic users to access the data services. With all advantages, mobile computing introduces an environment quite different from the one found in fixed networks due to scarce radio bandwidth and intermittent connectivity. Data services provided by W-WANs allow for a rather low link speed; error losses, changing line rate and variable delays present additional challenges for an efficient data transport. The Global System for Mobile Communications (GSM) is a widely successful effort to build a W-WAN system with millions of users in Europe and worldwide [20, 25]. GSM data, High Speed Circuit Switch Data (HSCSD), and General Packet Radio Service (GPRS) [4] are data transmission services offered by GSM.

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 [29]. 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, high latency and long sudden delays occurring on the wireless link may confuse TCP and yield throughput far from the available line rate. Some wireless networks try to hide all data losses from the sender by performing link-level retransmissions, seamless mobility and deep buffering inside the network. The reliability comes at the cost of variable delays in data transmission that can create problems for TCP. While optimizing TCP for a wireless environment has been an active research area for the last few years, not much attention has been paid to ensure that TCP implementations react well to long sudden delays, since sudden delays are not typical in fixed networks.

The rest of the paper is organized as follows. In Section 2 we give the background information on the TCP protocol and list possible sources of delays in W-WANs. In Section 3 we describe the reaction of TCP to large sudden delays. In Section 4 the effect of the New Reno algorithm and the TCP retransmission timer on spurious TCP timeouts is considered. In Section 5 we test how four widely deployed TCP implementations recover from spurious timeouts. Section 6 discusses several existing methods to strengthen TCP against spurious timeouts.

2. BACKGROUND

2.1 TCP

The Transmission Control Protocol (TCP) [24, 3, 2] 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 [3]. 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 [22]. 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 [28]. When the total size of outstanding segments, segments in flight (FlightSize), exceeds the window advertised by the receiver, further transmission of new segments is 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 [10]. 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 [2]. 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 windows. 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.

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 [26]. 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 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 inflating 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.

2.2 Sources of delays

The latency of W-WAN links is typically close to a second, which is several times higher than on a typical route in the wireline Internet. TCP adapts well to this type of more or less constant delay. This is because the TCP retransmission timeout is updated dynamically based on the smoothed mean and variance of RTT samples of the connection. The TCP retransmission timer can be considered overly conservative [17]. However, TCP does not react well to large delays (several times the usual RTT) that occur suddenly. Without a chance to adapt its retransmission timer to such a delay, TCP has to assume that outstanding segments were lost and retransmits them. There is a number of possible reasons for such type of delays in W-WANs.

Link-level error recovery. This is the most widely known source of varying delays as it presents a possibility for competing of link-level and end-to-end protocols in recovering error losses on a data link [14]. For example, an error burst requiring a high number of link-level retransmissions may be caused by a partial loss of the radio signal while driving into a tunnel. If the persistence of link-level error recovery exceeds the typical RTO of TCP over the given connection, spurious timeouts may result. Although studies report that cases of competing error recovery are infrequent in the basic GSM data service [16], the situation may be different for other W-WANs. Some wireless networks can include two or more layers of protocols capable of error recovery at the link level. For example, the GPRS wireless network includes both the Radio Link Control (RLC) protocol operating with small-sized frames at the lower level and the Logical Link Control (LLC) protocol operating with IP datagrams at the higher level [4]. Both protocols can be used in reliable or unreliable mode and the maximum number of retransmissions can be set as a network parameter. Optimally configuring the persistence of individual protocol sublayers may be a difficult task for a network operator. Thus, the TCP protocol in not guaranteed against operation over a highly persistent link introducing long delays in data transfers.

Handovers. During a handover the mobile terminal may have to perform some time-consuming actions before data can be transmitted in a new cell. These include, for example, collection of signal quality, transmitting it to the new base station, authentication, etc. Many W-WANs in such a case try to provide seamless mobility, that is internally re-route packets from the old to

the new base station at the expense of additional delay. As the result, the data transfer can be suspended for tens of seconds.

Blocking by high-priority traffic. When packet-switched and voice calls have to co-exist in a W-WAN network, in most cases the network operator assigns a higher priority to voice calls. An incoming voice call can temporarily preempt radio resources from packet-switched traffic, thus causing a delay in data transfer in the order of tens of seconds. Currently the support for Quality of Service (QoS) is being introduced into packet-switched W-WANs. Interactive traffic, for example web browsing, may have higher priority over best-effort bulk data transfers. There can be situations when the lower-priority traffic is delayed when higher-priority connections become active.

3. TCP AND DELAYS

3.1 TCP reaction on delays

As large sudden delays have not been a concern in the past, the only detailed study is presented in [15]. Here we briefly summarize results of that study. First, however, the method to visualize TCP traces has to be described. TCP traces in this paper are collected using tcpdump [11] program and presented as a time-sequence plot. The highest sequence number of a data segment versus capture time is plotted, compared to the acknowledgment number and advertised window for an ACK. TCP traces collected at the sender and at the receiver look differently, as can be seen in Figure 1. In the rest of the paper we present only sender-side plots, as they provide a better picture of the behavior of the TCP connection. However, receiver-side traces are used to verify that all segments have arrived to the receiver and not lost in the network. More details on displaying TCP traces can be found in [14, p. 52].

When a sudden delay that exceeds the current value of TCP retransmission timer occurs in the data transfer, TCP times out and retransmits the oldest outstanding segment. Since data segments are delayed but not lost, the retransmission is unnecessary and the timeout is spurious. A spurious TCP timeout is shown in Figure 1 taken from [15]. The delay in this test was generated by the hiccup tool, and delayed segments are marked with + in the plot. The first retransmission that happens at the 42nd second is also delayed. The sender interprets the ACK generated by the receiver in response to delayed segment (1) as related to the retransmission, not the original segment. This happens due to the retransmission ambiguity problem as the ACK bears no information which segment, original or retransmitted, has generated it. Encouraged by arriving ACKs, TCP retransmits all outstanding segments using the slow start algorithm. Also, a number of new

segments allowed by the congestion window are transmitted. Such retransmission policy is refereed to as go-back-N since the sender forgets about all segments it has earlier transmitted.

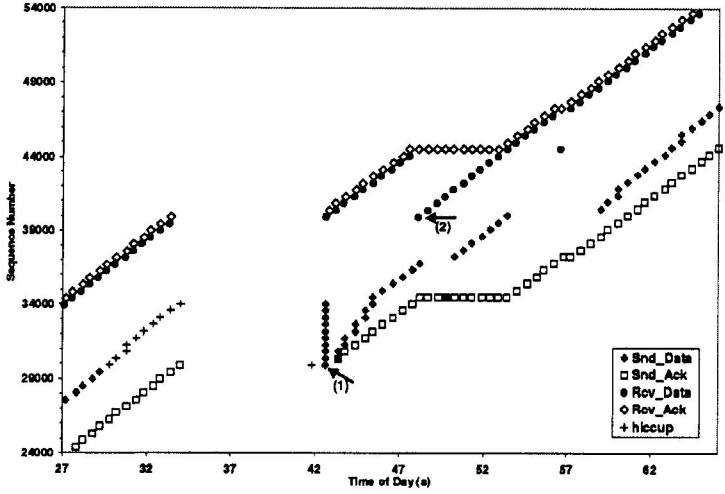

Figure 1. Reaction of TCP on a 13-second delay (sender and receiver traces) [15].

At time (2) retransmitted segments arrive to the receiver and generate DUPACKs as the original segments have already been delivered. When the threshold of three DUPACKs is reached at the sender on the 50th second, a spurious fast retransmit is triggered. The presumably missing segment is retransmitted and the congestion window is reduced that causes a pause in transmission of new segments (about 5 seconds starting from the 54th second) before the connection returns to normal. The BSDi3.0 TCP implementation used in this experiment has recovered relatively well from a spurious timeout.

3.2 Delays in a live network

During evaluation of the new GPRS wireless packet-switched data service [13] we frequently observed pauses during bulk data transfers using TCP. Tests were performed in a public GPRS network operated by Sonera in stationary conditions and while driving is Helsinki surroundings. The test configuration is shown in Figure 2.

Typically, blackout periods had some packet losses causing a performance slow down. However, here we are most interested in cases when packets are not lost, but delivered after a long delay. We have observed several such cases, one example is shown in Figure 3. Approximately 20 seconds after beginning of the connection, there is a 13-

second sudden delay. No segments are actually lost during the delay, but the TCP connection cannot recover from the delay for its lifetime unnecessary retransmitting many segments. Our preliminary measurement results indicate that the handover delay in the live GPRS network can exceed 10 seconds and thus is a likely reason for spurious TCP timeouts. Figure 4 shows the handover delay measured between two cells using the standard ping program with 32-byte packets. While in general the round trip time is stable at 1.3 seconds, it soars up to 10 seconds and more every time when the cell reselection is forced from the mobile terminal.

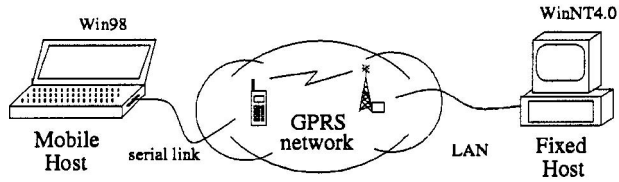

Figure 2. Configuration of measurements in the live GPRS network.

While every effort should be done to identify and remove sources of such delay spikes, it is unlikely that they can be completely eliminated. Furthermore, it is hardly possible to avoid spurious TCP timeouts occurring at such delay spikes, as it would require an extremely conservative TCP retransmission timer hampering the recovery of lost data. Thus, it is actual to ensure that existing and future TCP implementations recover reasonably from spurious timeouts.

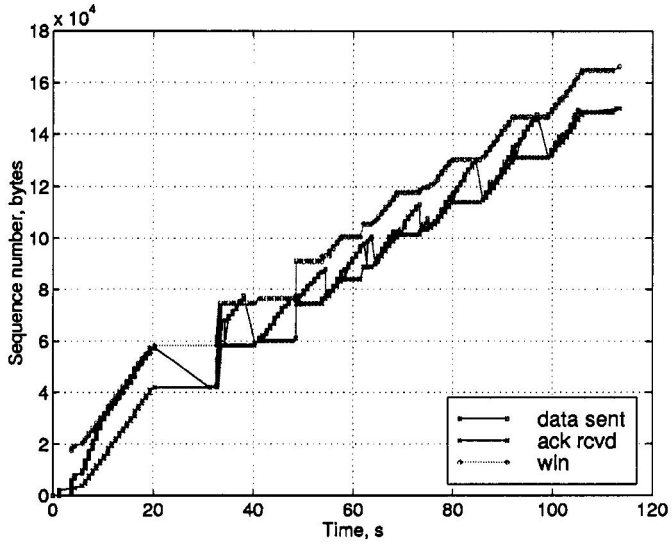

Figure 3. A delay during a downlink bulk TCP transfer in the GPRS network.

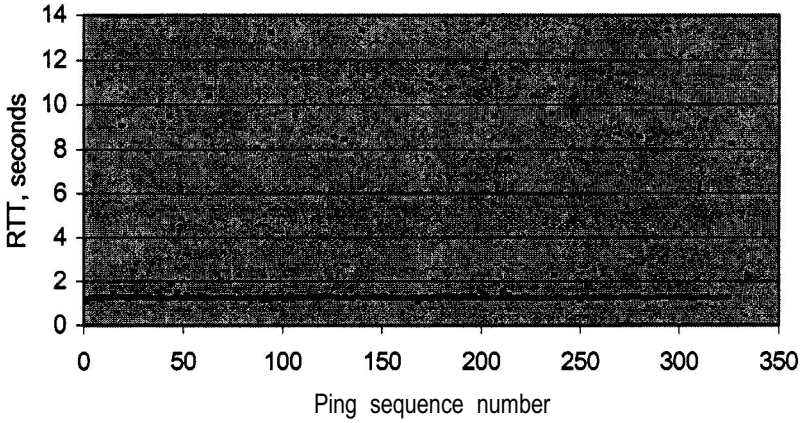

Figure 4. RTT spikes in GPRS when cell reselection is forced from the mobile terminal.

4. INTERACTION WITH TCP ALGORITHMS

This section examines the effect of several TCP algorithms on recovery from a spurious retransmission timeout.[1]

4.1 Effect of New Reno

New Reno [6] 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 [5]. New Reno can recover from multiple losses at the rate of one packet per round trip time. If during 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.

The description of a TCP spurious timeout in Section 3.1 is missing an important point if the New Reno algorithm is implemented at the sender. When the first DUPACK in Figure 1 arrives, the sender has six segments outstanding. Non-duplicate acknowledgments start to arrive at the 55th second. From the point of view of the New Reno algorithm, these acknowledgments are partial because they are not confirming the reception of the foremost outstanding segment at that time. Thus, the algorithm retransmits the presumably missing segment at each new partial acknowledgment. In this case, the number of unnecessary retransmitted segments increases from 10 to 15, thus increasing the penalty of a spurious timeout by half. The practical example of spurious New Reno retransmissions is shown in Figure 8 and discussed later in the paper.

Multiple fast retransmits in the context of segment losses are considered in [6, Section 5]. Here we extend the discussion to TCP recovery after a spurious timeout. A variant of New Reno, which does not transmit new segments on partial ACKs[2], could further worsen the recovery. Note that segments retransmitted on partial ACKs also generate DUPACKs for the foremost outstanding segment. When later on three DUPACKs arrive to the

[1] Ideas discussed in this section were presented on the end-to-end interest list in August 1-3, 2000

[2] RFC2582 permits sending new segments on partial ACKs if allowed by the congestion window. The available space in the congestion window depends on whether retransmitted segments are counted into it, which is not clearly said in RFC2581. Not sending new segments on partial ACKs seems to be a more conformant version of New Reno.

sender, another false fast retransmit is triggered followed by New Reno retransmissions. This would continue over and over until too few packets are in flight to trigger a spurious fast retransmit[3]. Fortunately, preventing the first false fast retransmit after the spurious timeout makes this problem a non-issue.

4.2 Spurious timeouts during fast recovery

A long delay in a TCP transfer triggers a spurious timeout, go-back-N retransmissions and a spurious fast retransmit. TCP implementations with an aggressive TCP retransmission timer may time out one or more times during the fast recovery while DUPACKs generated by go-back-N retransmissions are arriving. Such a spurious timeout causes more damage than just unnecessary retransmitting the outstanding segments. Retransmissions create a new series of DUPACKs that can be long enough to cause another spurious RTO. Such behavior continues through the connection life time until no more segments are left in flight to trigger a new spurious timeout. We suggest that this problem is independent of the problem of spurious New Reno retransmissions and multiple spurious fast retransmits discussed above. That is, preventing them does not necessary prevent a chain of spurious RTOs.

A practical example of such behavior is shown in Figure 6 and discussed later in this paper. A TCP connection experiencing such problems can send all data over two or three times, while not a single packet is actually lost. To avoid spurious timeouts during fast recovery, it may be useful to reset the retransmission timer when a DUPACK arrives, which is not currently considered in [22]. Absence of spurious timeouts during fast recovery would stop the chain of spurious retransmissions. We have to note, however, that the RTO can expire only during the exceptionally long fast recovery, especially if the RTO has been backed off during the preceding delay. Furthermore, resetting the RTO on DUPACKs makes it more conservative thus delaying recovery of lost segments in some scenarios.

5. TEST OF TCP IMPLEMENTATIONS

5.1 Test setup

The Software Emulator for Analyzing Wireless Data transfers (Seawind) [12] replaces the actual wireless link with a model that allows examining TCP behavior in a configurable and controlled environment. Indeed,

[3] This idea was generated by Reiner Ludwig in private communication.

capturing an exact scenario where a delay is present on the real data link is difficult, and performance comparison of different TCP implementations may not be valid. Seawind reproduces the basic properties of a wireless data link such as the limited line rate and a large propagation delay, and also allows placing a long sudden delay at a certain time in the connection. Seawind intercepts the flow of packets between a mobile host and a server and delays IP packets emulating the effect of a wireless link. An advantage of the emulation approach over tests for example with the NS simulator [9] is in ability to experiment with and compare the performance of different existing TCP implementations.

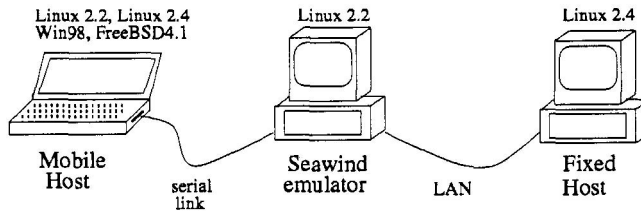

Figure 5. Configuration of measurements with an emulated W-WAN link.

We have configured Seawind to emulate a link with similar characteristics as provided by data services of GSM. The line rate was set to 9600 bps, the propagation delay to 300 ms. Further on, a 10-second delay is introduced after 5 seconds of the beginning of the TCP connection. For testing we have selected four major TCP implementations that together form a larger part of all deployed TCPs: FreeBSD 4.1, Windows 98, Linux 2.2, and Linux 2.4. The TCP implementation in Linux has undergone a major revision from the kernel release 2.2 to 2.4. The Linux distribution was RedHat 6.2.

TCP parameters in the experiment were as follows: MSS of 256 bytes, the receiver window of 16 kilobytes, and the SACK option disabled. The New Reno algorithm was enabled on Linux, disabled in FreeBSD and of unknown status in Windows 98. A bulk data transfer sending 100 kilobytes of data from the mobile to the fixed host was generated by ttcp [27].

5.2 Test results

During the delay, the behavior of tested TCPs was generally in accordance with Figure 1. The TCP timer expires approximately after five seconds of the delay and the oldest outstanding segment is retransmitted. When ACKs to delayed segments arrive, all outstanding segments are retransmitted according to the go-back-N policy. From this point, tested TCP implementations differ in behavior.

FreeBSD 4.1 (Figure 6) showed particularly poor performance. The huge initial window (discussed in Section 5.3) leads to a large number of

outstanding segments when the delay occurs. The go-back-N retransmissions generated enough DUPACKs so that after a spurious fast retransmit, the RTO has expired two times. Upon each timeout, the outstanding segments are retransmitted although no non-duplicate ACKs have arrived, which is a clear implementation fault. Extensive number of unnecessary retransmissions has triggered a series of spurious fast retransmits and timeouts collapsing the throughput. Double or triple transmissions of segments are separated by idle times due to congestion avoidance procedures done at each timeout and fast retransmit. This continues until the point when not enough segments are in flight to trigger another spurious timeout.

Windows 98 (Figure 7) is having similar problems after a delay as FreeBSD, although a small initial window is used. A large number of spurious retransmissions in this case can be caused by the incorrect RTO computation or by a strange version of the New Reno algorithm, since segments were retransmitted after reception of the first partial ACK. At the point of time between the 50^{th} and 60^{th} second, TCP has stopped retransmitting segments due to unknown reasons. After that the connection proceeds normally. In live network tests we have observed the behavior similar to shown in Figure 3 in a major part of the traces; data segments were transmitted several times over the link resulting in inadequate throughput.

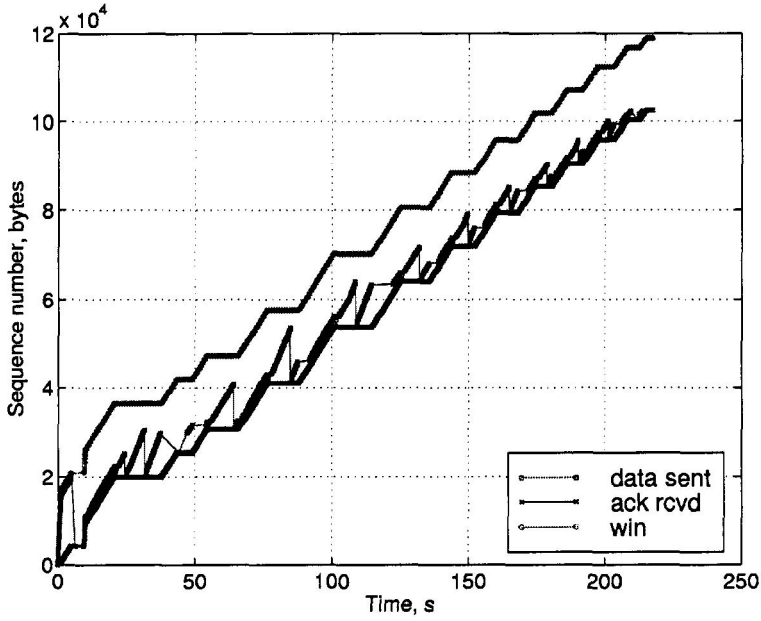

Figure 6. Recovery of TCP in FreeBSD 4.1 from a spurious timeout (complete connection).

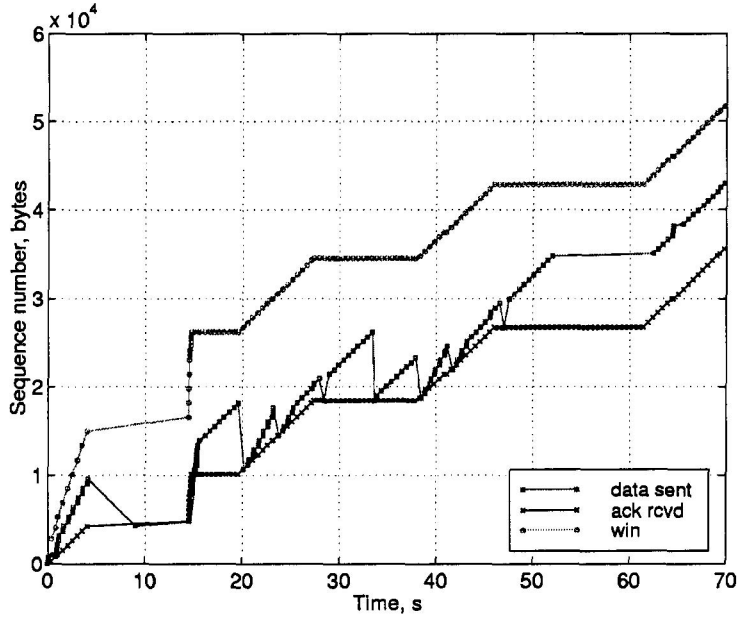

Figure 7. Recovery of TCP in Windows 98 from a spurious timeout (zoomed, total connection time 191 seconds).

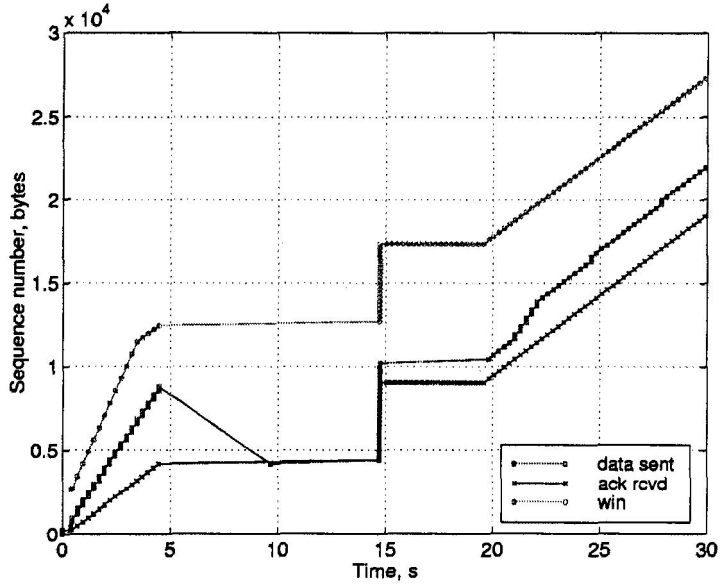

Figure 8. Recovery of TCP in Linux 2.2.12 from a spurious timeout (zoomed, total connection time 115 seconds).

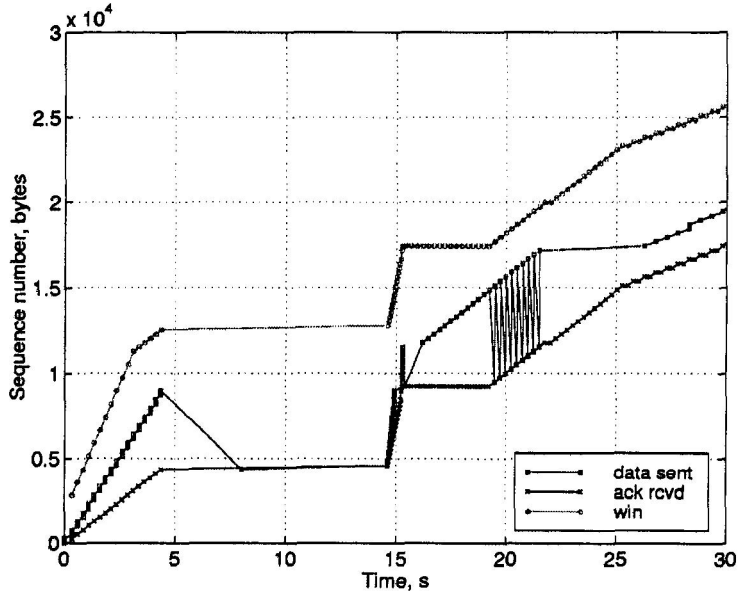

Figure 9. Recovery of TCP in Linux 2.4.0 from a spurious timeout (zoomed, total connection time 114 seconds).

Linux 2.2 (Figure 8) recovers relatively well from the spurious timeout. Arriving DUPACKs trigger a spurious fast retransmit followed by additional retransmissions caused by the New Reno algorithm. One old and one new segment is transmitted per each partial ACK thus resulting in nine additional unnecessary retransmissions. A 10-second pause after partial ACKs is caused by congestion avoidance procedures. However, from this point the connection proceeds normally without additional retransmissions.

Linux 2.4 (Figure 9) is similar to Linux 2.2, except that there is no spurious fast retransmit and no spurious New Reno retransmissions. When we have noticed the problem in Linux 2.2, it was reported to developers and corrected in the kernel release 2.4 by implementing the careful version of the restriction described in Section 6.

5.3 TCP implementation faults

Here we present a number of TCP "features" that were noticed while examining TCP traces during measurements in the live GPRS network and during tests with Seawind. Some of these implementation problems are already discussed in [23, 21].

A huge initial window. We were surprised to observe that FreeBSD used the initial window of 16 kilobytes instead of one or two segments currently allowed [2]. Apparently, the initial window is set to this value when the sender and receiver are located in the same IP subnetwork. This is clearly not desired when the TCP connection is established over a slow PPP link, as it has been in our case. The default initial window can be set to a proper value by changing a system parameter.

Receiver window overruns. We can see in Figure 3 that the TCP sender exceeds the receiver advertised window in two places, at 38th and 97th second. We have observed this with Windows 98 and NT in many of examined TCP traces. Exceeding the advertise window is not allowed and may create additional performance problems with TCP implementations that discard TCP segments outside their advertised window.

An incorrect retransmission timeout. One part of the problem with excessive spurious retransmissions on Windows may be caused by incorrect calculation of the retransmission timer. On the official support page of Microsoft, the problem is described as follows [19]. Windows 95, Windows 98, Windows NT4.0 TCP/JP may retransmit packets prematurely as the retransmit timer is computed incorrectly because of a math error. This can result in unnecessary retransmissions and lower throughput over high-delay networks.

Early fast retransmit. In order to avoid spurious fast retransmits when packets are re-ordered in the network, TCP is required to collect a threshold of three DUPACKs before a presumably missing segment is retransmitted

[2]. Examining TCP traces showed that Windows 98 always has been triggering a fast retransmit already after the second DUPACK, while Windows NT after the first DUPACK.

6. METHODS TO STRENGTHEN TCP AGAINST SPURIOUS TIMEOUTS

Careful version of *EWC 2582.* Due to performance considerations, fast retransmits in TCP should be disabled after an RTO until all packets transmitted earlier are acknowledged [6]. A less careful version of this restriction allows the fast retransmit when DUPACKs arrive for the foremost outstanding packet, while a more careful version does not. As we can see in Figure 1, a spurious timeout presents exactly the situation when DUPACKs arrive for the foremost outstanding segment. Using the more careful version of the restriction [6, Section 5] would not allow a spurious fast retransmit in the case of spurious timeouts limiting the penalty to go-back-N retransmissions. However, there is still a possibility of a second spurious RTO if the retransmission timer is not reset upon DUPACKs. Another scenario unrelated to spurious timeouts where the more careful version avoids unnecessary retransmissions is given in [8, p. 69]. This empirical evidence suggests that the careful version should be implemented in all TCPs.

D-SACK. The Selective Acknowledgment option (SACK) [18] in TCP allows conveying the information to the sender on exactly which packets are missing at the receiver. The D-SACK extension of SACK (duplicate-SACK) allows reporting the sequence number of a packet that triggered a DUPACK [7]. The sender can use this information to determine whether the segment has been unnecessary retransmitted and avoid further unnecessary retransmissions. In our case a spurious fast retransmit and New Reno retransmits can be avoided. However, D-SACK does not help to prevent the go-back-N behavior after a spurious timeout, as the retransmission ambiguity problem is not resolved.

Eifel. The Eifel algorithm [15] is using a timestamp option to distinguish between original data and retransmissions. It resolves the retransmission ambiguity problem and entirely avoids unnecessary retransmissions after a spurious timeout. When an ACK after the first retransmitted segment is received, its timestamp is compared to the timestamp of the first retransmission. If the ACK is older than the retransmission, the ACK was generated by the original transmission and the timeout was spurious. The Eifel algorithm allows to detect and abort spurious retransmissions, thus preventing the go-back-N behavior and spurious fast retransmits.

Packet lifetime. Assigning appropriate lifetime to packets inside a W-WAN network and discarding expired packets makes TCP to apply loss recovery mechanisms in an intended way, which is more efficient than recovery from a spurious timeout. For example, GPRS specifications support packet lifetime in the network components [1], A long delay exceeding RTO of TCP would also exceed the lifetime of packets in the network. The TCP timeout ceases to be spurious, as it leads to retransmission of discarded segments. No DUPACKs are generated in this case avoiding problems seen in TCP traces.

7. CONCLUSIONS

Long sudden delays in data transfers are not typical in wireline networks and their effect on the TCP protocol has not been extensively studied. Such delays are not uncommon in W-WANs due to link-level retransmissions, handovers and temporal resource preemption by high-priority packet traffic or voice calls. As an example, we give preliminary measurements of the cell reselection delay in the GPRS network. We have shown that the New Reno algorithm increases the amount of retransmissions after the spurious timeout roughly by half and that an aggressive retransmission timer may expire during the fast recovery generating a chain of spurious retransmissions. We have tested four widespread TCP implementations in their reaction to a long delay. All four implementations, FreeBSD 4.1, Windows 98, Linux 2.2 and Linux 2.4 have exhibited go-back-N retransmissions while the following behavior has been different. FreeBSD and Windows are recovering especially poorly, partly due to implementation problems that we also have listed. We have to highlight the importance of the Eifel algorithm as the only known solution that prevents go-back-N retransmissions thus nearly altogether eliminating the penalty of a spurious TCP timeout. We recommend that all TCPs implement the careful version of New Reno, as it prevents many of the discussed problems. Resetting the retransmission timer upon a reception of DUPACK would avoid spurious timeouts during fast recovery, but we have to study the effect of this modification in more detail. Future work will include examination of delay sources in the GPRS and UMTS wireless networks, as well as designing new algorithms to improve the response of TCP to long sudden delays.

8. ACKNOWLEDGMENTS

Many thanks to Reiner Ludwig, Sally Floyd, Mark Allman, Alexey Kuznecov, Rod Ragland, Venkat Venkatsubra, Pasi Sarolahti for their comments and suggestions on the material presented in this paper. Experiment with the live GPRS network could not be done without Olli Aalto and Heimo Laamanen. I am grateful to Timo Alanko for checking the paper and invaluable advice on research methodology.

REFERENCES

[1] BSS GPRS protocol (BSSGP). 3GPP TS 08.18 V8.4.0, October 2000.

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

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

[4] G. Brasche and B. Walke. Concepts, services and protocols of the new GSM phase 2+ general packet radio service. IEEE Communications Magazine, pages 94-- 104, August 1997.

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

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

[7] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. An extension to the selective acknowledgment (SACK) option for TCP. IETF RFC 2883, July 2000.

[8] A. Gurtov. TCP performance in presence of congestion and corruption losses. Master's thesis, Department of Computer Science, University of Helsinki, December 2000. Available at: http://www. cs.Helsinki.FI/group/iwtcp/papers/.

[9] ISI at University of South California. Network simulator 2. Available at: http://www.isi.edu/nsnam/ns/.

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

[11] V. Jacobson, C. Leres, and S. McCanne. tcpdump. Available at http://ee.lbl.gov/, June 1997.

[12] M. Kojo, A. Gurtov, J. Mannner, P. Sarolahti, T. Alanko, and K. Raatikainen. Seawind: a wireless network emulator. Submitted to MMB 2001.

[I3] J. Korhonen, O. Aalto, A. Gurtov, and H. Laamanen. Measured performance of GSM HSCSD and GPRS. In Proceedings of the IEEE International Conference on Communications, 2001. To appear.

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

[15] R. Ludwig and R. H. Katz. The Eifel algorithm: Making TCP robust against spurious retransmissions. ACM Computer Communication Review, 30(1), January 2000. Available at: http://www. acm. org/sigcomm/ccr/archive/2000/jan00/ccr-200001 -1udwig.html.

[16] R. Ludwig, B. Rathonyi, A. Konrad, K. Oden, and A. Joseph. Multi-layer tracing of TCP over a reliable wireless link. In Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computing Systems (SIGMETRICS-99),

volume 27,1 of SIGMETRICS Performance Evaluation Review, pages 144--1 54, New York, May 1—4 1999. ACM Press.

[17] R. Ludwig and K. Sklower. The Eifel retransmission timer. ACM Computer Communication Review, 30(3), July 2000.

[18] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. IETF RFC 2018, October 1996. Standards Track.

[19] Microsoft. TCP/IP may retransmit packets prematurely. Available at: http://support.microsoft.com/support/kb/articles/Q236/9/26.ASP.

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

[21] 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.

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

[23] V. Paxson, M. Allman, S. Dawson, W. Fenner, J. Griner, I. Heavens, K. Lahey, J. Semke, and B. Volz. Known TCP implementation problems. IETF RFC 2988, Mar. 1999.

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

[25] M. Rahnema. Overview of the GSM system and protocol architecture. IEEE Communications Magazine, 3 1 (4):92-- 100, April 1993.

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

[27] 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.

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

[29] K. Thompson, G. J. Miller, and R. Wilder. Wide-area internet traffic patterns and characteristics. IEEE Network, 1 1(6): 10--23, November/December 1997.