Efficient Transport in 2.5G3G Wireless Wide Area Networks

Andrei Gurtov

Licentiate Thesis
Department of Computer Science
UNIVERSITY OF HELSINKI

UNIVERSITY OF HELSINKI – HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET

Faculty – Tiedekunta/Osasto – Fakultet/Sektion	Department – Laitos – Institution
Science	Computer Science

Author – Tekijä – Författare
Andrei Gurtov

Title – Työn nimi – Arbetets title
Efficient Transport in 2.5G3G Wireless Wide Area Networks

Subject – Oppiaine – Läroämne
Computer Science

Level – Työn laji – Arbetets art	Month and Year – Aika – Datum	Number of pages – Sivumäärä – Sidoantal
Licentiate Thesis	September 2002	28 + 120 p.

Abstract– Tiivistelmä – Referat

Multibillion investments are committed to licenses and infrastructure of second (2.5G) and third (3G) generation wireless wide area networks to deliver multimedia services to mobile users. Interactive, conversational, streaming and background applications use transport protocols to communicate over unreliable wireless links. Achieving efficient transport in 2.5G3G networks implies meeting QoS requirements of applications while preserving radio resources, battery power and friendliness to other flows in the Internet.

In this thesis, existing and emerging wireless wide area networks are examined through measurements, simulations and emulation. Such events as delay spikes, bandwidth oscillation and connectivity outages are difficult to prevent in the heterogeneous and dynamic wireless environment. For instance, delay spikes can be caused by handovers, higher priority voice calls or persistent loss recovery at the link layer. Furthermore, link characteristics can change by an order of magnitude when a user switches between 2.5G and 3G networks. Such disruptive events can cause delivery of stale real-time data, spurious TCP timeouts and low utilization of the wireless link. Therefore, achieving efficient transport in this environment demands coordinated efforts from the radio network and from the end-to-end transport protocol.

We introduce a framework to deal with these challenges in a resource-efficient way while minimizing QoS violations seen by applications. The framework includes enhancements in the radio network and in end-to-end transport. In the radio network, we deal with resource management, link level retransmissions, inefficient cross-layer interactions, active queue management and user charging.

We study end-to-end transport of real-time and non-real-time data. For non-real-time data, TCP is a highly suitable transport protocol when profiled with state-of-the-art features and when its robustness to delay spikes is improved. In this thesis, response of different TCPs to delay spikes is measured; algorithms to alleviate negative performance effects of spurious TCP timeouts are presented and evaluated.

Delay spikes in the network can often make real-time data useless to the receiver. For real-time transport we suggest using a transport protocol that incorporates explicit lifetime into packet headers. The link layer discards stale data instead of transmitting them over the radio link. This preserves radio resources and battery power of wireless users. However, we found that discarding stale data can incorrectly trigger congestion control at the end systems.

Finally, we propose a receiver congestion manager to utilize locally available information on priority of flows and available bandwidth at the mobile client.

Computing Reviews Classification:
C.2.1 (Network Architecture and Design): Wireless Communication,
C.4 (Performance of Systems)

Keywords – Avainsanat – Nyckelord
Wireless networks, mobile computing, transport protocols

Where deposited – Säilytyspaikka – Förvaringställe
Library of the Dept. of Computer Science, Report C–2002–42

Additional information– Muita tietoja – Övriga uppgifter

Acknowledgments

Gaining hands-on wireless networking experience while working in Sonera Corp. was important to complete this work. I would like to acknowledge support from Sami Grönberg, Ville Saarikoski, Mika Raitola, Sami Ala-Luukko, Heimo Laamanen and all colleagues at the Cellular Systems Development department. In particular, Matti Passoja, Olli Aalto and Jouni Korhonen contributed to this work through many hours of measurements and tracing.

I would like to thank Prof. Kimmo Raatikainen for supporting this work at the Department of Computer Science. Pasi Sarolahti and Markku Kojo provided many useful comments on TCP.

Dr. Reiner Ludwig from Ericsson Research was my advisor and a beacon of high quality research standards. Dr. Michael Meyer, Dr. Roger Kalden, and Hannes Ekström provided valuable details on GPRS and UMTS networks.

Many thanks to the IETF community, especially to the PILC and Transport Area working groups, for many friendly discussions and social events; to the Linux networking community and foremost to Alexey Kuznetsov for many rewarding emails; to graduate students and to Prof. Randy Katz at the University of California at Berkeley for inspiring discussions and for arranging two summer schools.

Completing this work would not be possible without love and patience of my wife Anastasia.

Contents

Overview o-1

[P1] J. Korhonen, O. Aalto, A. Gurtov, H. Laamanen, Measured Performance of GSM HSCSD and GPRS. In Proceedings of the IEEE Conference on Communications, June 2001. p-1

[P2] A. Gurtov, Effect of Delays on TCP Performance, IFIP Personal Wireless Communications, August 2001. p-6

[P3] 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, September 2001. p-24

[P4] A. Gurtov, Making TCP Robust Against Delay Spikes, University of Helsinki, Department of Computer Science, Series of Publications C, No C-2001-53, November 2001. p-40

[P5] A. Gurtov, R. Ludwig, Evaluating the Eifel Algorithm for TCP in a GPRS network, In Proceedings of European Wireless, February 2002. p-57

[P6] H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov, F. Khafizov, TCP over Second (2.5G) and Third (3G) Generation Wireless Networks, RFCxxxx. p-64

[P7] A. Gurtov, M. Passoja, O. Aalto, M. Raitola , Multi-Layer Protocol Tracing in a GPRS Network. IEEE Vehicular Technology Conference, September 2002. p-92

[P8] A. Gurtov, R. Ludwig, Responding to Spurious Timeouts in TCP, submitted for publication. p-97

[P9] A. Gurtov, R. Ludwig, Exploiting Packet Lifetime for Efficient Real-Time Transport, submitted for publication. p-109

List of Acronyms

2.5G3G	Extended Second and Third Generation
3GPP	Third Generation Partnership project
ACK	Acknowledgment
ADU	Application Data Unit
AIMD	Additive Increase Multiplicative Decrease
ALF	Application Layer Framing
API	Application Programming Interface
ARQ	Automatic Repeat Request
BSC	Base Station Controller
BTS	Base Transceiver Station
CDMA	Code Division Multiple Access
CRC	Cyclic Redundancy Check
DCCP	Datagram Congestion Control Protocol
DupThresh	Duplicate Acknowledgment Threshold
DUPACK	Duplicate Acknowledgement
ECN	Explicit Congestion Notification
EDGE	Enhanced Data for GSM Evolution
ETSI	European Telecommunications Standards Institute
FACK	Forward Acknowledgment
FEC	Forward Error Correction
FTP	File Transfer Protocol
GGSN	Gateway GPRS Serving Node
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HSCSD	High Speed Circuit Switch Data
HTTP	Hyper Text Transfer Protocol
IETF	Internet Engineering Task Force
IFIP	International Federation for Information Processing
IP	Internet Protocol
IS-96	Interim Standard #96
LLC	Logical Link Control
MPEG	Motion Picture Experts Group
MS	Mobile Station
MTU	Maximum Transmission Unit
NRT	None Real Time
NS2	Network Simulator version 2
NTT DoCoMo	Nippon Telephone & Telegraph, Do Communications over the Mobile network
PDC-P	Personal Digital Cellular Packet-switched
PILC	Performance Implications of Link Characteristics
QoS	Quality of Service
RED	Random Early Detection
RFC	Request for Comments
RLC	Radio Link Control
RNC	Radio Network Controller
RT	Real Time
RTO	Retransmission Timeout
RTP	Real Time Protocol
RTT	Round Trip Time
SACK	Selective Acknowledgment
SCTP	Stream Control Transmission Protocol
SGSN	Serving GPRS Support Node

SR	Selectively Reliable
TBF	Temporal Block Flow
TCP	Transmission Control Protocol
TFRC	TCP-Friendly Rate Control
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
WAP	Wireless Application Protocol
WCDMA	Wideband Code Division Multiple Access
WLAN	Wireless Local Area Network
WWAN	Wireless Wide Area Network
VoIP	Voice over Internet Protocol

Overview: Contents

1 Introduction **1**
2 Background **3**
2.1 Applications ... 3
2.2 Transport Protocols ... 4
2.2.1 Reliable Transport ... 4
2.2.2 Unreliable Transport ... 5
2.3 2.5G3G Networks ... 5
2.3.1 Deployments ... 5
2.3.2 Link Characteristics ... 6
2.4 The Problem: Efficient Transport ... 8
2.4.1 Radio Network Challenges ... 8
2.4.2 End-to-End Transport Challenges ... 9
2.5 Related Work ... 10
2.5.1 Cross-Layer Interactions ... 10
2.5.2 Adaptive Transport ... 11
2.5.3 Overlay Networks and Intermittent Connectivity ... 11
2.5.4 Congestion Control ... 11
3 Methodology **12**
3.1 Simulation ... 12
3.2 Emulation ... 12
3.3 Measurements ... 13
3.4 Modelling Assumptions ... 14
4 Radio Network Optimizations **14**
4.1 Active Queue Management ... 14
4.2 Cross-Layer Interactions ... 15
4.3 Efficiency at the Link Layer ... 16
4.4 New Charging Principles ... 17
5 End-to-End Optimizations **18**
5.1 The General TCP Profile for 2.5G3G Networks ... 18
5.2 Responding to Spurious Timeouts in TCP ... 19
5.3 Avoiding Spurious Timeouts in TCP Loss Recovery ... 19
5.4 Exploiting Packet Lifetime for Real-Time Transport ... 20
5.5 Receiver Congestion Manager ... 21
6 Research History **22**
7 Conclusions and Future Work **23**
References **24**

1 Introduction

Wireless data access for nomadic users is the key enabling technology for the future Internet [30]. Therefore, existing second-generation wireless wide area networks such as GSM [40] are upgraded to support packet switched radio access. The upgraded networks are referred to as 2.5G and are in extending use today. Wide deployment of third generation systems is several years ahead. The transition to 3G is expected to be a gradual process. Initially, 3G will be deployed to introduce high capacity and high-speed access in densely populated areas. Mobile users with multimode terminals will be able to utilize existing coverage of 2.5G systems in the rest of the territory. Operators have invested tremendous amounts of money into 3G spectrum licenses and no less is required to create the new network infrastructure. With these commitments in mind it is crucial for the industry and society to deliver advanced data services to mobile users in an efficient way.

The primary goal of this study is to explore efficient end-to-end transport of user data in a heterogeneous and dynamic environment of 2.5G3G networks. Our secondary goal is to provide the user with means to prioritize among ongoing data transfers and to examine requirements for fair charging in 2.5G3G networks. Applications, the fixed network and the radio network are designed by independent groups of people often unaware of the effect of their solutions on the other layers. Therefore, it is an additional challenge to converge these distinct views to provide a well-functioning system.

Efficient transport requires addressing the challenges on the application, network and radio bearer level as illustrated in Figure 1. At the application layer the requirement is to satisfy demands on Quality of Service such as low response time, high throughput, and reliability [55]. To guarantee a long-term Internet stability, data flows must follow congestion control principles [15]. Scarce radio bandwidth and battery power require that duplicate or stale data are not sent over a wireless link. We argue that due to the nature of the wireless medium, QoS violations cannot be totally avoided in 2.5G3G networks. It is the task of a transport protocol to minimize the negative impact of events such as delay spikes or data losses on the application and at the same time avoid loading the network with unnecessary transmissions.

We perform multi-layer tracing [P7], simulation [19] and emulation [P3] of 2.5G3G systems. Some of the known problems in a wireless environment such as bursty data losses, long latency, overbuffering and intermittent connectivity [35] are still present in 2.5G3G networks. New problems include delay spikes resulting from cell reselections, resource pre-emption [P2] and bandwidth oscillation due to on-demand allocation of radio resources [65]. An inter-system vertical handover in addition to a delay spike and packet losses can often cause an order-of-magnitude change in bandwidth and latency of the data link [59], [7], because a user moves from a wider area and slower radio network into a smaller area and faster overlay network. In such a dynamic and

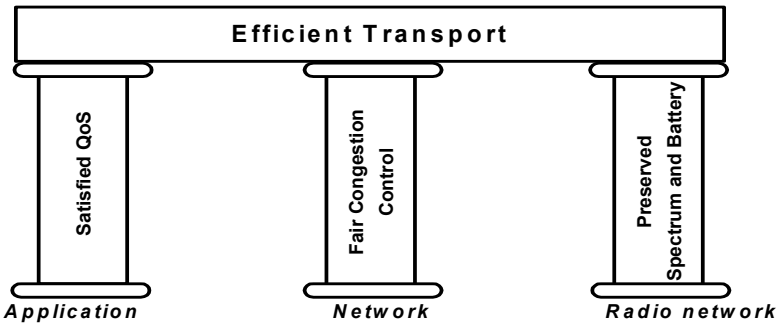


Figure 1. Requirements for efficient transport in wireless networks.

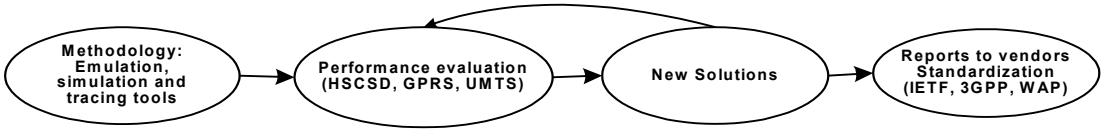


Figure 2. A path to achieve efficient transport in 2.5G3G networks.

heterogeneous environment, loss of efficiency easily occurs due to unnecessary retransmissions by a reliable transport protocol such as TCP or delivery of stale data by a protocol with real-time constrains.

The contribution and the structure of this thesis is summarized in Figure 2. On the methodology side, the contribution is in creating new tools and extending the existing ones. The tools for network measuring and tracing are critical to identify performance problems. Currently deployed protocols and networks are rich in implementation flaws and we attempt to filter out fundamental issues behind poor performance. After we understand the problems, we can suggest new solutions and evaluate existing ones. Finally, the solutions need to be standardized or submitted to network vendors to ensure their deployment.

Figure 3 shows our contributions in more detail. We first describe modifications at the end host improving efficiency of end-to-end transport. The improvements are located in Internet servers used for delivery of reliable non-real-time data and unreliable real-time data. Today TCP is the most widely used reliable transport protocol accounting for more than 90% of total Internet traffic [8]. TCP is a mature and robust protocol highly suitable for dominant applications such as WWW, FTP, remote login and many other non-real-time (NRT) Internet applications. We show that state-of-the-art TCP [P6] performs well in 2.5G3G networks in general, although there are some corner cases such as robustness to delay spikes that would benefit from further study. We improve robustness of TCP to delay spikes and bandwidth oscillation with response mechanisms to spurious timeouts [P8].

There is a growing demand for real-time (RT) applications such as video telephony, audio streaming, or stock quote updates. Such applications often favour timeliness instead of perfect reliability. We assume that the Selectively Reliable Real-time Transport Protocol (SR-RTP) on top of the Datagram Congestion Control Protocol (DCCP) is used for delivery of real-time data; the protocols are described in Section 2.2 on page 4. Real-time data is often useless to the receiver if delayed beyond a certain limit in the network. We suggest discarding stale data already in the radio network to improve efficiency by preserving wireless bandwidth and battery power. The data lifetime can be available in the wireless network as a part of the negotiated QoS profile or, alternatively, by attaching the real-time server and the base station to a global clock.

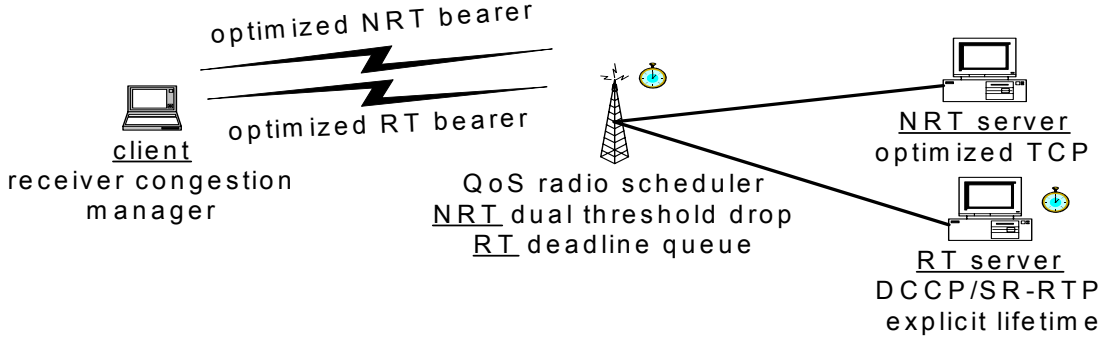


Figure 3. A framework for achieving efficient non real-time (NRT) and real-time (RT) transport in 2.5G3G networks.

At the mobile host, to give the end user a capability to prioritize among ongoing transfers, we propose a receiver congestion manager. The receiver congestion manager utilizes locally available information such as link bandwidth, hints on imminent handovers from the link layer and priority of transfers from the user to selectively throttle real-time and non-real-time flows.

We study active queue management and behavior of link-layer protocols in the radio network. A task of link buffering for non-real-time flows such as TCP is to smooth traffic burstiness and varying transmission delay at the link. At the same time, negative effects such as high queueing delay and heavy data losses due to congestion need to be avoided. We propose a simple Dual Threshold Drop algorithm that fulfils these goals in an environment with only a few concurrent flows [18]. At the radio link level, we suggest several protocol enhancements based on live network tracing that help avoiding unnecessary retransmissions of data frames. We also study cross-layer interactions of end-to-end protocols with resource allocation and scheduling at the link layer. Finally, we propose new charging principles that could improve overall system capacity by rewarding social behavior of users.

The rest of the thesis overview is organized as follows. Section 2 provides the necessary background on applications, transport protocols and 2.5G3G networks. Furthermore, it defines the problem of efficient transport in detail and reviews related work. In Section 3 we introduce new measurement, simulation and emulation tools we have developed and used. Section 4 describes optimizations in the radio access network that includes enhanced operation of the link protocols, active queue management and scheduling. Section 5 concentrates on solutions at the end hosts. That includes TCP improvements, applying explicit packet lifetime to real-time transport protocols, and the receiver congestion manager. Section 6 documents the research history of publications included into this thesis. Finally, Section 7 sums up our main results and outlines future work.

2 Background

2.1 Applications

The first component of achieving efficient transport is satisfying the QoS requirements for applications. It was suggested to divide 2.5G3G applications into four classes: background, streaming, interactive and conversational [55].

Examples of **background** applications include file transfer and email download. Background applications normally require reliable data delivery but are not highly sensitive to interpacket jitter and can also tolerate high RTT delay. Thus, the main performance goal for such applications is high throughput or the inverse, short download time. TCP is a highly suitable protocol for such applications.

Interactive applications include web browsing and remote terminal access. Users of such applications expect an immediate result of their actions. Therefore, the most important characteristic for interactive applications is low response time. Interactive applications often exhibit ‘click from page to page’ behavior. For example, when the user selects a new Web link, the ongoing transaction is aborted and data buffered in the network become obsolete.

Streaming applications playback video or audio contents without waiting for the entire download to complete. Such applications typically buffer data at the receiver to accommodate jitter in packet delivery. However, the buffer size available at the receiver is often limited due to memory

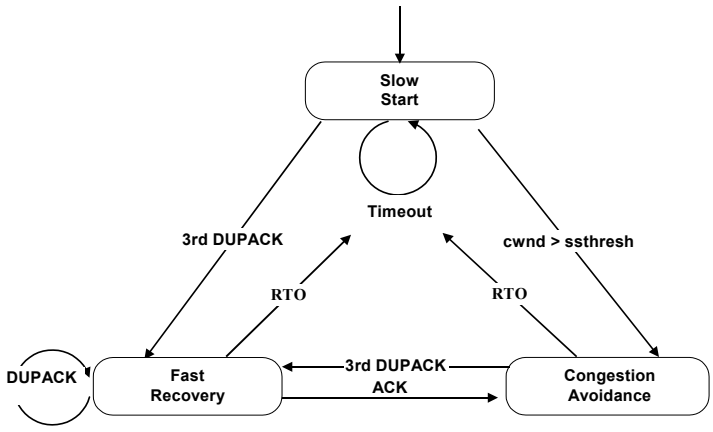


Figure 4. Loss recovery and congestion control in TCP. A steady connection state is in congestion avoidance.

size limitations in mobile devices and a possible change of the play point by the user. Streaming applications do not require perfect reliability but favor on-time delivery. In fact, data packets may only be useful for a streaming application before the receiver playback buffer becomes empty. A transport protocol is most attractive for streaming applications if it provides partial reliability by performing retransmissions only within the allowed time frame before the buffer is depleted.

Conversational traffic is also referred to as voice over IP (VoIP) and includes bi-directional video and audio. Conversational traffic does not necessarily require high data rates, but it has stringent requirements on latency and delay jitter. The acceptable one-way latency is 100-300 ms which often leaves no time for loss recovery through retransmissions.

Other real-time services such as stock quotes or news ticker updates often present a mixture of properties of streaming and conversational traffic classes.

2.2 Transport Protocols

2.2.1 Reliable Transport

The **Transmission Control Protocol (TCP)** [47], [54] is the most widely used transport protocol in the Internet. TCP provides applications with reliable byte-oriented delivery of data on the top of the Internet Protocol (IP). Figure 4 shows functions of loss recovery and congestion control in TCP. A TCP sender transmits data in segments and expects acknowledgments (ACKs) from the receiver. TCP has two basic mechanisms for recovery of lost data segments. The first mechanism is a retransmission timer at the sender, which expires when no new ACKs arrive for the duration of the retransmission timeout (RTO). RTO is set up dynamically based on the current round trip time (RTT) and its variation [28]. The second mechanism is a fast retransmit algorithm followed by a loss recovery phase. Fast retransmit is triggered when three Duplicate Acknowledgments (DUPACKs) arrive to the sender. DUPACKs are generated by the receiver as a response to out-of-order segments. A fast retransmit algorithm often allows quicker recovery from a lost segment than via a retransmission timeout.

The congestion control is required for estimation of available bandwidth in the network and fair co-existence with other flows [15]. Congestion control in TCP is tightly connected with loss recovery because a packet loss is taken as an indication of congestion in the network [28]. A TCP connection begins with the slow start until a packet loss is detected or the slow start threshold is reached. In slow start, TCP approximately doubles the load on the network in every RTT. In congestion avoidance, TCP enters an additive increase phase raising the load approximately by one

segment per an acknowledged congestion window. Upon a fast retransmit, the congestion window is halved. After a timeout the estimate of available network capacity (the congestion window) is set to one packet.

TCP has an important property of self-clocking also known as the packet conservation principle [28]. In the equilibrium condition each arriving ACK indicates that a segment has left the network and triggers a transmission of a new segment. To prevent a fast sender from overflowing a slow receiver, TCP implements flow control based on a sliding window. When the total size of outstanding segments, segments in flight (FlightSize), reaches the window advertised by the receiver, further transmission of new segments is blocked until an ACK arrives.

When a delay spike in the network exceeds the current value of the retransmit timer, a timeout occurs. The TCP sender retransmits the oldest outstanding segment. Since the original segment or the corresponding ACK is only delayed but not lost, the retransmission is unnecessary and the timeout is said to be spurious. TCP suffers from a retransmission ambiguity problem [29]. ACKs bear no information that would allow the TCP sender to distinguish an ACK for the original segment from that for the retransmission. Therefore, the sender interprets the ACK generated by the receiver in response to the original segment as corresponding to the retransmitted segment. This leads to unnecessary go-back-N retransmission behavior and violation of the packet conservation principle [33].

2.2.2 Unreliable Transport

The **User Datagram Protocol (UDP)** [46] is the basic unreliable transport protocol. It includes only the essential transport mechanisms such as port numbers and a checksum.

The **Real-time Transport Protocol (RTP)** [51] is an application-layer protocol based on the Application Layer Framing (ALF) [9] concept. RTP supports sequence numbers, timestamps and the media source identifiers which are common features required for real-time applications. By itself RTP makes no guarantees on data delivery. However, there are extensions such as Selectively Reliable RTP (SR-RTP) [11] that allow for selective retransmissions and congestion control.

The **Datagram Congestion Control Protocol (DCCP)** [38] is a new connection-oriented protocol that uses ACKs for congestion control. However, DCCP does not provide any reliability. The most important feature of DCCP is TCP-friendly congestion control (**TFRC**) [16] that achieves smoother bandwidth adaptation than the Additive Increase Multiple Decrease (AIMD) algorithm used in TCP [28]. Therefore, DCCP is a suitable protocol for applications that cannot tolerate rapid variation in throughput.

We generally assume in this thesis that SR-RTP on top of DCCP is used as a real-time transport protocol.

2.3 2.5G3G Networks

2.3.1 Deployments

The second generation cellular systems are commonly referred to as 2G. The 2G phase began in the 1990s when digital voice encoding had replaced analog systems (1G). 2G systems are based on various radio technologies including frequency-, code- and time- division multiple access. Examples of 2G systems include GSM (Europe), PDC (Japan), and IS-95 (USA). Data links provided by 2G systems are mostly circuit-switched and have a transmission speed of 10-20 kbps uplink and downlink. Demand for higher data rates, instant connectivity, charging based on data volume rather than on connection time, and lack of radio spectrum allocated for 2G led to the introduction of 2.5G (GPRS, EDGE, PDC-P) and 3G (UMTS, cdma2000) systems. A taxonomy of mobile

radio networks can be found in [58].

The integrated GPRS-UMTS infrastructure is shown in Figure 5. The General Packet Radio Service (GPRS) [6] is an extension to GSM that provides higher data rates and 'always on' capability that allows users to remain connected to the network also during the idle periods. GPRS is based on a packet-switched technology, that permits efficient sharing of radio resources among users. Built on top of the GSM radio interface, GPRS allows a user to utilize multiple GSM timeslots which increases the data rate available to the user up to 40 kbps. GPRS supports handovers of ongoing data connections. The Radio Link Control (RLC) protocol provides recovery of error losses on the radio link between the mobile station (MS) and the Base Station Controller (BSC). The Logical Link Control (LLC) protocol spans from the mobile station to the Serving GPRS Support Node (2G-SGSN) and retransmits lost data primary during handovers. The GPRS Gateway Support Node (GGSN) serves as a router connecting the GPRS network to the Internet and via dedicated links to enterprise users or service providers. A GPRS handover is called a cell reselection as the mobile terminal is responsible for activating the cell change procedure.

High-Speed Circuit-Switched Data (HSCSD) is an extension for GSM that allows the user to utilize multiple timeslots creating a fast circuit-switched link. Enhanced Data for GSM Evolution (EDGE) is a new modulation technique and new channel coding that increases the bandwidth of the radio link. EDGE applied to GPRS can threefold the data rate for a single user up to 120 kbps.

The radio technology of Universal Mobile Telecommunication System (UMTS) in Europe and Japan and cdma2000 in USA is based on code division multiple access allowing for higher data rates and higher spectrum utilization than 2G systems. 3G systems provide both packet-switched and circuit-switched connectivity. A UMTS core network in an evolved version of GPRS and some network elements, for example GGSN, can be shared between UMTS and GPRS as shown in Figure 5. 3G-SGSN has only a limited functionality compared to 2G-SGSN, because many functions are moved to the radio access network composed of Radio Network Controllers (RNC). We will use the term base station to refer both to BSC and RNC.

The transition to 3G is expected to be a gradual process. Initially, 3G will be deployed for high capacity and high speed access in densely populated areas. Mobile users with multimode terminals will be able to utilize existing coverage of 2.5G systems in the rest of the territory. An inter-system handover between 2.5G and 3G is also called a vertical handover because a user moves from a wider area and slower radio network into a smaller area and faster overlay network [7], [59].

2.3.2 Link Characteristics

This section describes user-visible performance aspects of 2.5G3G links [P1], [P6].

Data rates. The main incentives for transition from 2G to 2.5G to 3G are increased data rates for users and greater network capacity for operators. 2.5G systems have data rates of 10-20 kbps in uplink and 10-40 kbps in downlink. 3G systems are expected to have bit rates around 64 kbps in uplink and 384 kbps in downlink. The per user data rate is dynamic and depends on the number of users in the cell, amount of voice traffic and amount of radio resources reserved for packet-switched traffic. Furthermore, a user located on a cell boundary cannot obtain as high a data rate as close to the base station due to rapidly increasing error rates and interference. The link data rate can oscillate rapidly due to switching of a high-speed radio channel among multiple users [65].

Asymmetry. 2.5G3G systems have built-in asymmetry in uplink and downlink data rates. The uplink data rate is limited by the battery power consumption and complexity limitations of mobile terminals. Bandwidth available in the downlink direction is typically 3-6 times higher than in uplink. This asymmetry is well-suited for mobile users who are more likely to consume the information than produce it.

Latency. The latency of 2.5G3G links is high due to interleaving on the radio link and process-

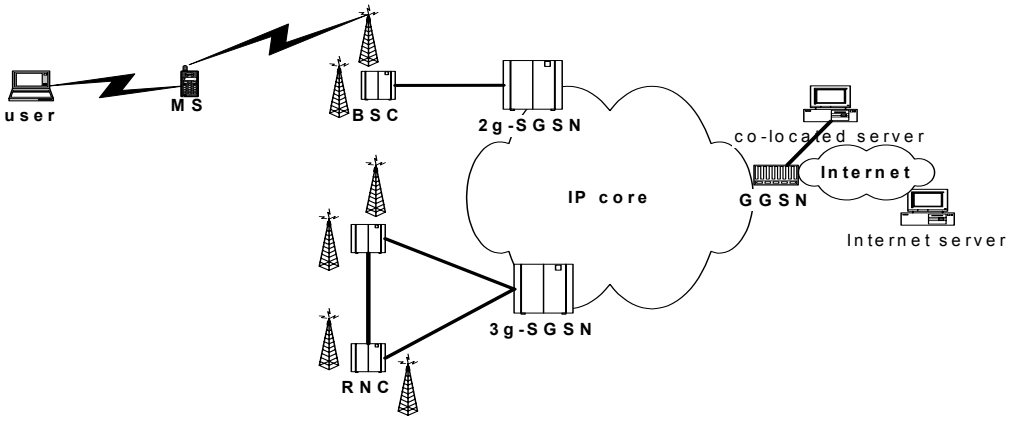


Figure 5. A combined UMTS and GPRS network architecture. The laptop is connected to the mobile terminal using an access link such as IrDA, Bluetooth, or a serial cable [58].

ing delays in networking equipment. A typical RTT on an unloaded link varies between a few hundred milliseconds to more than one second.

Delay spikes. 2.5G3G links are likely to have delay spikes significantly exceeding the typical RTT due to following reasons. (1) A delay spike can be the result of link layer recovery during an outage due to temporal loss of radio coverage, for example, while driving into a tunnel or stepping into an elevator. (2) During a handover the mobile terminal may have to perform some time-consuming actions before data can be transmitted in a new cell. Some wide area wireless networks try to internally re-route packets from the old to the new base station at the expense of additional delay. (3) Blocking by high-priority traffic may occur when an arriving circuit-switch call or higher priority data user temporarily preempts the radio channel..

Error losses. In general, 2.5G3G systems have a low rate of error losses thanks to link-level retransmissions. However, link layer recovery introduces variable delays in data delivery which may not be acceptable for real-time flows. Therefore, the link should be flow adaptive, i.e. perform retransmissions only for flows that require reliable data delivery [35]. 2.5G3G systems can suffer from congestion-unrelated losses during handovers.

Intersystem handovers. As mentioned in Section 2.3.1, it is likely that 3G systems will be used as a 'hot spot' technology while 2.5G systems will provide lower speed data service on the rest of territory. A mobile user can roam between 2.5G and 3G networks while keeping ongoing TCP connections. Figure 6 (a) shows the envisaged operation in the integrated environment of WLAN, UMTS and GPRS. The first challenge for seamless mobility is a high delay spike caused by a vertical handover to ongoing flows. Loss from a few to all outstanding packets during a handover presents a second challenge. Furthermore, the link characteristics after a handover can be radically different from the old cell. The bandwidth and latency can change by two orders of magnitude, for example from 20 kbps bandwidth in GPRS to over 2 Mbps in WLAN.

Intermittent connectivity. In case the user does not have multimode terminals, or the overlay coverage is not available in the area, the data access is only available sporadically as shown in Figure 6 (b). A further development of this concept relies on very high speed access available only in a limited number of spots called Infostations [26].

2.4 The Problem: Efficient Transport

2.4.1 Radio Network Challenges

This section describes challenges to efficient end-to-end transport in the radio access and core part of 2.5G3G networks. The fundamental resources in the radio network is battery capacity at the mobile terminal and the radio spectrum. Secondary resources are computational capacity of networking elements and limited bandwidth of last-mile links from the core network to the base stations. Last-mile links are often point-to-point microwave radio links.

Radio resource allocation. Although 2.5G3G networks provide packet-switched connectivity, at the link layer they still require setting a temporal radio resource allocation before user data can be transmitted [58]. A decision on how long to maintain this link-level resource allocation affects transport efficiency. It is a research problem to find an optimal allocation algorithm based on anticipated end-to-end protocol behavior and user workload.

Radio link control protocols typically operate in a selective acknowledged mode. The important issue is how frequent link protocol ACKs are sent and on which radio channel. There are involved trade-offs between preserving radio resources to avoiding large delays in transmission. The retransmission policy at the link protocol layer is also important for timely recovery and avoidance of unnecessary retransmits.

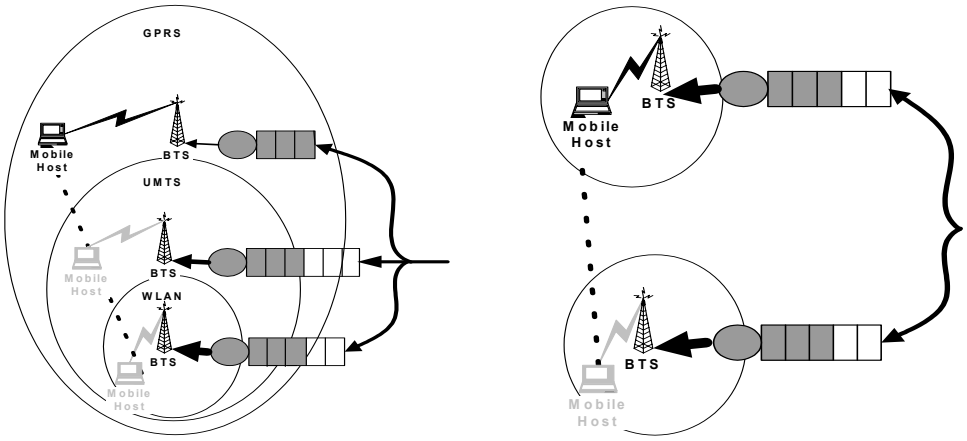
Cross-layer interactions. Radio networks include multiple protocol layers that may interact inefficiently. One example of inefficient interaction could be data fragmentation. There should be a clear understanding of how IP packets are fragmented into link layer frames and radio blocks. Fragmentation policies affect performance of loss recovery, scheduling as well as header overhead. Another aspect is competition among multiple protocol layers for error recovery. Some radio networks, for example GPRS, include more than one protocol level capable of error recovery in the radio network in addition to end-to-end recovery at the transport layer.

Scheduling. Radio scheduling is crucial to achieve differentiation among flows to fulfil QoS requirements. Although priority scheduling ensures that the most important data gets transmitted first, lengthy delays of background TCP flows should be avoided as it may cause spurious timeouts and unnecessary retransmissions. The amount of allocated radio resources should take into account the current traffic activity of the user. The allocation is typically controlled by the radio network that has no information on how much data a user wants to transmit in the near future. A situation is possible, when abundant resources are allocated to a user.

Buffer management. The end-to-end transport protocols treat the network as a black box and attempt to estimate the appropriate transmission rate based on data losses due to buffer overflows. Therefore, adequate buffering is important to enable efficient transport. An overly small buffer results in link underutilization, inability to handle bandwidth variation and an insufficient window for error recovery [54]. An overly large buffer increases link RTT, which is particularly harmful for real-time traffic. Deploying ‘smarts’ in the base station such as deleting stale real-time data, duplicate packets or data from aborted transfers can significantly improve the efficiency. Additional functions are often required from buffer management such as policing of real-time and shaping of non-real-time flows.

Additional issues that are not directly in our scope include protocol header compression and mobility issues.

Compression. Sending less data over the radio link is more efficient. Traditional compression protocols such as Van Jacobson’s header compression do not work well with a high level of data losses in a wireless environment [35]. The Robust Header Compression working group in IETF is developing new solutions suitable for 2.5G3G networks [24].



(a) Vertical handovers between wireless overlay networks

(b) Intermittent connectivity e.g. to high-speed Infostations.

Figure 6. Varying connectivity presents challenges for efficient transport.

IP mobility and context transfer. Handovers in 2.5G3G networks are so-called 'layer 2' handovers because the upper layers including IP and TCP protocols remain unaware of it. IP-level solutions are required, for example, for WLAN-GPRS mobility. IP mobility solutions include paging to locate dormant nodes and discovery of new access routers. These issues are addressed by the Seamoby working group in IETF [23]. During vertical handovers it can be beneficial to transfer buffered data or protocol state, such as the header compressor state, from the old to the new network.

2.4.2 End-to-End Transport Challenges

A task of the transport protocol is to meet QoS requirements of applications by facing difficult network characteristics. This section describes challenges for efficient end-to-end transport as seen by transport protocols at the communicating hosts [P1], [P2].

The well-known problems in wireless networking are *error losses* and *utilization of high-bandwidth delay paths*. These problems have been extensively studied [4]. Losses on wireless links often do not relate to congestion, but occur due to corruption at the radio link or due to handovers. Such losses may cause lengthy retransmission timeouts and underutilization of the radio link. On high-speed networks of high latency ramping up to an appropriately large transmission window takes a long time for TCP. Using state-of-the-art TCP and reducing the network latency whenever possible lowers the negative effect of these problems.

Vertical handovers and intermittent connectivity shown in Figure 6 (a) and (b) can cause the following anomalies:

- A delay spike
- A delay increase due to decreased bandwidth
- A bandwidth-delay product change.

For transport protocols such events cause the following problems:

A **spurious timeout** is a timeout that would not occur had the sender waited longer. The transport protocols that rely on timeouts to signal loss data will falsely deduce a loss event in such case. Spurious timeouts lead to unnecessary retransmissions that can violate the packet conservation principle. *Inefficient cross-layer interactions* can occur when error recovery at the link layer is

seen by the transport protocol as a delay spike. Spurious timeouts can also occur after an abrupt increase in the link RTT due to a sudden decrease of link rate, for example, due to release of a high-speed radio channel.

A link is **overbuffered** if it persistently has a longer queue than required for its efficient utilization. In static network conditions, overbuffering usually occurs when a redundantly large buffer is allocated in the network. **Underutilization** is typical in the initial phase of a TCP connection due to slow start or due to non-congestion related losses. In general, overbuffering or underutilization appear when an estimate of the network capacity made by the transport protocol does not match the real network capacity. When the network conditions change at once, for example due to a vertical handover, the amount of outstanding data can be radically different than optimally required for efficiently using the network. When a connection is moved from a slow to fast cell, it can underutilize the available bandwidth, since in the congestion avoidance phase TCP increases the sending rate slowly. Changing from a fast to slow cell is handled well by TCP due to a self-clocking mechanism. However, a large TCP window used in a faster cell can create the overbuffering problem in the slower cell. Intersystem handovers can cause performance problems for ongoing TCP connections as many features (e.g. window scaling) are negotiated at the connection establishment and cannot be changed later in the connection lifetime.

Genuine retransmission timeouts occur when a large amount of data is lost in the network, for instance due to a vertical handover. Alternatively, a retransmission is lost, which cannot be typically recovered without a timeout by existing transport protocols.

A **dependability** issue arises when a delay spike lasts for minutes and is more appropriately referred to as an outage. In this case the transport protocol can often exceed its maximum number of retransmissions and give up. Ideally, the persistency of the transport protocol is controlled by the application by specifying the maximum time it is willing to wait for data delivery.

2.5 Related Work

This section provides a high-level description of the latest directions in the transport research area. A good overview of TCP performance problems and improvements in wireless environments can be found in [4], [35], [63] and in our previous work [18]. A description of Internet QoS mechanisms is given in [60] and of wireless QoS aspects in [58], [64].

2.5.1 Cross-Layer Interactions

A recent work on eliminating inefficient cross-layer interactions in wireless networks [35] is based on measurements of GSM circuit-switched data links. On the radio network level the major contribution was a concept of flow-adaptive links that change their characteristics based on QoS requirements of a flow. This concept is partly implemented in 2.5G3G networks. At the transport layer, two major contributions were the Eifel algorithm [33] for detecting spurious timeouts and a new more accurate retransmission timer for TCP. We extend the results [35] to the new environment of 2.5G3G packet-switched networks.

Loss recovery by a reliable link layer protocol is seen as delay jitter by the transport layer. There exists a possibility of competing error recovery when the transport protocol decides that outstanding segments were lost and retransmits them. In fact, these segments can be still being recovered by the link layer. In such a case the timeout in the transport protocol is spurious, as simply waiting longer would allow for the link layer to complete the recovery. Spurious TCP timeouts present two problems as described in Section 2.2.1. The Eifel algorithm stores the timestamp of the first TCP retransmission after a timeout. By comparison with the timestamp of the first ACK received after the timeout, the algorithm detects whether the timeout has been spurious. After a spurious timeout,

transmission is resumed with a next unsent segment and the congestion control state is restored. The TCP sender returns to the equilibrium state as before the timeout.

An alternative response algorithm to spurious timeouts in TCP has been proposed [49]. A different example of interaction of TCP with link-layer resource allocation causing bandwidth oscillation is studied in [65]. The feasibility of cross-layer interactions of TCP lower layers in GPRS is studied in [39].

2.5.2 Adaptive Transport

Adaptive Video Streaming work [12] focuses on developing an open-source, widely accepted streaming video application. The Internet imposes packet loss on data, which can severely hamper the quality of a compressed bit stream with interdependencies. Available bandwidth and delays on the Internet are variable, which causes problems for an application that wants to play out received data at a constant rate. A streaming application should adjust its sending rate and the quality of the transmitted bit stream in accordance with these changes. The SR-RTP software selectively recovers the most important lost data within its lifetime [11] and adapts to the varying network conditions using the congestion manager [5].

Many of current multimedia applications and protocols including RTP originate from the Columbia University [52]. RTP and related protocols are further developed and standardized by an IETF working group on audio and video transport [25].

The Stream Control Transmission Protocol (SCTP) [43] is a recently introduced protocol primarily for transport of signalling over IP networks. The most important difference compared to TCP is the ability to deliver user messages within multiple streams and network-level fault tolerance through support of multihoming. SCTP is built on similar congestion control algorithms as TCP.

2.5.3 Overlay Networks and Intermittent Connectivity

The BARWAN research project [7] developed a scalable architecture that can support wireless access across multiple overlay networks while delivering high levels of end-to-end performance to applications. It allows mobile applications to seamlessly operate with network connectivity that provides best bandwidth/battery use and price trade-off in the current location. The project generated many well-known contributions, such as the Snoop protocol for TCP-aware link layer loss recovery [4].

A standard TCP connection is bound to the IP interface and address on which it was initiated. This presents a problem for mobile hosts that often acquire a new IP address on the move, for instance using Mobile IP. Alternatively, a user may want to switch from an office desktop to a portable laptop while maintaining an ongoing file transfer. These problems are tackled in the Migrate project [53]. Migrate supports address changes and disconnectivity handling for legacy applications through a shim session layer and provides a set of system primitives for new mobility-aware applications.

2.5.4 Congestion Control

The Congestion Manager (CM) [5] is an end-to-end framework for congestion control and management, bandwidth sharing, independent of specific transport protocols, for example TCP, and applications. Its end-system architecture enables logically different flows such as multiple concurrent Web downloads, concurrent audio and video streams to adapt to congestion, share network information, and share varying available bandwidth well. Rather than have each stream act in isolation and thereby adversely interact with the others, the CM maintains host- and domain-specific

path information, and orchestrates all transmissions. The CM's internal algorithms ensure social and stable network behavior; its API enables a variety of applications and transport protocols to adapt to congestion and varying bandwidth.

Unsuitability of TCP-like AIMD congestion control for real-time flows triggered developing TCP-friendly rate control (TFRC) algorithms [16], [61]. TCP can introduce an arbitrary delay because of its reliability and in-order delivery requirements; thus, the applications such as on-line gaming and streaming use UDP instead. This growth of long-lived non-congestion-controlled traffic, relative to congestion-controlled traffic, poses a real threat to the Internet stability. TFRC is friendly to TCP in the long run, but avoids abrupt changes in the transmission rate which makes it attractive for real-time applications.

3 Methodology

3.1 Simulation

Simulation is an ideal approach for preliminary evaluation of protocol design, as it allows a rapid exploration of a wide parameter space. The Network Simulator (NS2) [57] is a de facto standard tool for evaluation of Internet protocols. NS2 provides a reproducible and controllable environment with reference protocol implementations. We have contributed new modules implementing our protocol modifications and a delay spike generator [19].

For experiments we use a simple topology shown in Figure 7. The end points implement the transport protocol under study. A 2.5G3G link is represented as a link with variable bandwidth, packet losses and periodic delay spikes. The bottleneck queue preceding the link is controlled by a queue management algorithm such as drop-tail or RED [13]. The details of configuration differ from case to case, please refer to the methodology part of the publications.

3.2 Emulation

Emulators allow researchers to create network topologies and conditions that are difficult to achieve in a controlled and reproducible manner in production networks. Furthermore, emulators allow experimenting with real prototype protocols and applications. The principal difference between simulation and emulation is that an emulator replaces one component of the system, in

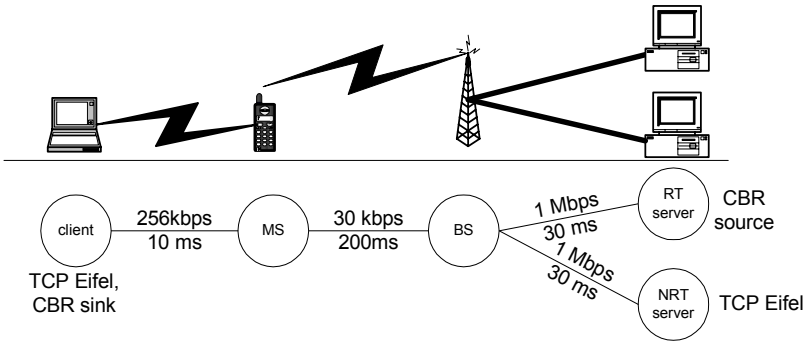


Figure 7. Simulation setup in NS2. A Constant Bit Rate (CBR) source represents a real-time application.

our case the communication network, in a way that enables the system to practically operate in real-time. Simulation, on the other hand, presents an abstract model of the complete system executed in virtual time.

Available network emulators imitate delay, packet drop, and queue management commonly present in computer networks. In wireless networks, the network characteristics can change drastically due to the movement of mobile terminals. A key functionality that is missing in existing emulators such as DummyNet [50] or ONE [1] is ability to change network characteristics during an experiment. For effective evaluation of 2.5G3G networks, changing sets of parameters in the course of an experiment is important to model the user mobility. An emulator developed for trace-based modelling of wireless networks best suits for reproducing characteristics of existing networks, but provides a limited capability to study networks only in the design phase [42].

We developed an emulator named Seawind that suits well for modelling next generation wireless wide area networks [P3]. Execution of a fine-grain real-time event schedule on off-the-shelf computer platforms is challenging. A study of the issues related to real-time emulation in Linux describes possible methods to achieve accurate results [20]. Figure 8 depicts a test setup used during Seawind experiments. The mobile host and fixed host are running a real implementation of the transport protocol or application under study. Seawind intercepts data traffic between two hosts and imitates limited bandwidth, latency, delay spikes and packet losses.

3.3 Measurements

A powerful multi-layer tracing methodology was introduced in [32] to study the GSM data transmission. We adapt this methodology to perform multi-layer tracing in a GPRS test network. Tracing at radio and link layers in a GPRS network is done using a commercial tool NetHawk [41] as shown in Figure 9. NetHawk is capable of recording packets of GPRS protocols for a particular user in BSC and SGSN. End-to-end protocol behavior is captured using the tcpdump utility. We measured end-to-end performance in a live GPRS network while driving in a urban environment in the Helsinki area. The exact test configuration including types of mobile terminals can be found in [P7]. Throughput measurements are performed using the ttcp tool and latency using the standard ping utility.

We developed a number of techniques to estimate the size of bottleneck network buffer and delay jitter. Such estimates are necessary since detailed characteristics of commercial networking equipment are often unavailable. The buffer size is calculated according to the amount of outstanding packets at the time the first packet loss occurs in a TCP connection with a large receiver window [18]. Delay jitter is measured by generating a stream of small packets sent at the regular

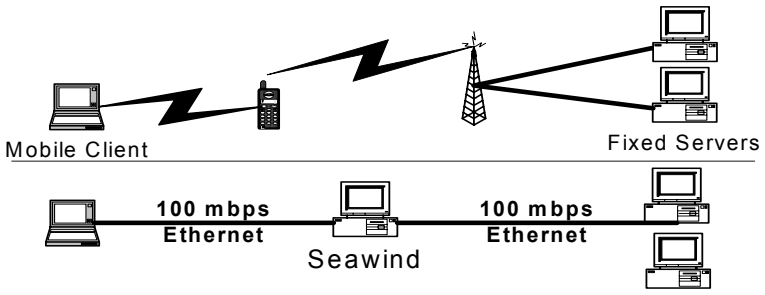


Figure 8. Measurement setup using the Seawind real-time emulator.

interval using an rtptools utility package.

3.4 Modelling Assumptions

In most tests we use a uni-directional bulk TCP transfer as workload. It is a simple, well-understood and commonly used type of workload. During measurements of response time for transactional traffic such as in [P1], request and reply messages are sent of a size estimated from logs of a local web proxy. For latency measurements, the standard ping program provides a simple and widely used workload.

For mobile users and operators battery power consumption and radio resource preservation are often as important as the throughput across the wireless link. Therefore, throughput (calculated based on all data transmitted by the sender including retransmissions) and goodput (calculated based on useful data arrived to the receiver) are equally important performance metrics. Additional metrics are used when necessary for a specific experiment. For example, in [P8] we also give the average number of spurious and genuine timeouts for each connection to indicate how susceptible a TCP modification is to timeouts.

We make typical modelling assumptions that TCP connections are long-lived and there is no congestion in the opposite direction. Determining the extent to which these assumptions hold in the Internet is a hard problem [17]. For example, HTTP connections commonly transfer only 10 kilobytes of data ([31], p. 380). However, we suppose that conclusions based on our assumptions generally hold in 2.5G3G networks.

4 Radio Network Optimizations

This section presents a number of optimizations developed during a measurement study in a test GPRS network [P7]. The issues appear to be general enough to be considered for other 2.5G3G networks as well.

4.1 Active Queue Management

Radio networks often have a large buffer with drop-tail policy which has been shown to perform poorly [32]. Our GPRS measurements indicated the downlink bottleneck buffer of approximately

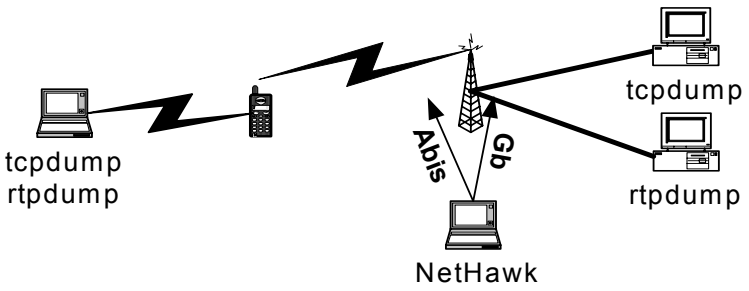


Figure 9. Multi-layer protocol tracing in a test GPRS network at Abis and Gb interfaces.

50 KB [P7]. This is several times larger than is required to smooth traffic burstiness and utilize the link. In our earlier work [18] we studied an effect of applying the Random Early Detection (RED) algorithm [13] in the emulated radio network. The major design goals for RED were maintaining a normally small packet queue in the router but allowing for short-term traffic bursts and avoiding synchronization among flows.

When only a few connections share the bottleneck queue, RED is not efficient to prevent the start-up buffer overshoot and avoid overbuffering. The main reason is that the moving average of the queue size used to calculate a dropping probability in RED reacts too slowly to the exponential TCP slow start.

Therefore, we suggested the following buffering scheme. The idea is to define a soft threshold in the router queue, close to the delay-bandwidth product of the link. The hard threshold can be two-three times larger than the soft limit. When the queue size reaches the soft threshold, 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 threshold is reached, the router drops all arriving packets. Extending this algorithm to work well for multiple concurrent connections requires a counter or a timer-based mechanism to determine when to drop a packet from another flow. Some heuristics that favor connections with small packets can be implemented to protect interactive flows. An evaluation of this algorithm shows improved performance of TCP and TFRC flows [10].

An active Internet user often aborts ongoing TCP connections, for example, by clicking from page to page in a web browser. Due to the significant amount of buffering found in 2.5G3G networks, packets from aborted TCP connections are still transmitted to the user that wastes resources. When a TCP connection is aborted at the receiver, an arriving data segment on that connection is discarded and a reset packet is sent to the sender. For example, GPRS measurements show that after aborting a file download, data packets are still unnecessary delivered to the mobile host during 30 s. This problem can also be partly solved by limiting the bottleneck buffer space to the necessary minimum or throttling flows from the receiver as proposed in Chapter 5.5. Here we describe a complete solution located at the bottleneck buffer in the base station.

The TCP receiver generates reset (RST) segments in response to arriving packets on a aborted connection ([54], p. 247). We suggest intercepting such RST packets in the bottleneck network queue located before the wireless link. Then, buffered packets in the opposite direction that belong to the aborted connection are deleted [21]. The performance gain from such an approach could be significant and the method works robustly for all TCP applications including WWW, FTP, and email.

The method is implemented as follows. The network node located before the bottleneck link (we assume that the wireless link is the bottleneck in most cases) examines headers of all TCP segments and looks for the 'RST' reset bit set. When it notices such a packet, it removes all buffered packets in the opposite direction belonging to the same TCP connection as identified by the IP source and destination address together with the TCP source and destination port. The node then forwards the RST packet further to the destination so that the TCP sender aborts the connection. In case the RST packet gets lost after that, the sender will retransmit a TCP segment that will not be dropped at the node according to the algorithm. Instead, it will generate another RST packet at the receiver that gets forwarded to the sender. In case of a repeated loss of a data or RST segments the algorithm relies on the TCP sender’s retransmission timeout.

4.2 Cross-Layer Interactions

This section discusses resolving four types of undesired cross-layer interactions [P7].

Fragmentation and reassembly. Lower protocol layers in radio networks often fragment (and

reassemble) IP packets into smaller units. Fragmentation is done to achieve better throughput in presence of data corruption. A smaller size can also provide a finer-grain scheduling. However, a small fragment size also increases the header overhead that wastes radio resources and possibly decreases the user throughput. The fragment size is often chosen without considering typical packet sizes at the upper layers. For instance, in GPRS measurements the typical IP MTU of 1500 bytes gives slightly lower throughput than that of 1480 bytes. Tracing at the Gb interface showed inefficient fragmentation at the LLC layer. The maximum LLC frame size was configured to 500 bytes, thus IP packets were fragmented into four frames, with the fourth frame only a few bytes long. Sending small frames reduces efficiency due to higher header overhead.

Competing error recovery. The presence of multiple protocol layers in the radio network (for example RLC and LLC in GPRS) creates a possibility for competing error recovery between these protocols. We did not observe any such cases in GPRS multi-layer tracing because our network was operating with unacknowledged LLC. However, the upper protocol layer should ensure that the lower protocol has discarded the data before retransmission. It can be achieved with less persistence at a lower layer than a typical value of a retransmit timeout at the upper layer. Alternatively, a flush-buffer message can be sent to the lower layer when a timeout occurs on the upper layer.

Channel allocation. A radio channel in a packet-switched network is often allocated on demand. Until the interval between data packet arrivals to the link is less than the channel release timeout, the resource allocation is triggered frequently. Keeping the radio channel for the time equal to average link RTT could avoid this problem. The challenge is that the packet interarrival time depends on traffic characteristics. It may be lower for bulk TCP connections and higher for interactive connections such as carrying HTTP traffic. The network configuration should be tuned according to the prevailing type of traffic, for example, separately for each traffic class described in Section 2.1 on page 3.

During GPRS measurements we observed oscillations in RTT of subsequent ping packets of approximately 200 ms [P7]. Prior to transmitting user data, a Mobile Station (MS) must activate a Temporal Block Flow (TBF) toward BSC. MS contends on an ALOHA-style random access channel to receive a resource allocation from the network. According to release 97 specifications, TBF should be turned down immediately when the data buffer in the base station becomes empty [56]. Such a policy interacts badly with the TCP layer by increasing the RTT, as every segment and ACK may trigger setup of a new TBF. Keeping TBF for longer periods has been suggested [62] and is reflected in Enhanced GPRS specifications. The extended TBF release decreases the minimum RTT seen by TCP by more than a hundred milliseconds and reduces the signaling load. However, the BSC is unaware if an MS with an active TBF has any data to transmit and therefore also has to schedule idle MSs, thus wasting radio resources. Furthermore, the number of simultaneous TBFs is limited and postponing the TBF release may prevent data transmission by other MSs.

Priority blocking. During GPRS measurements we observed situations when a scheduler in the radio network caused spurious timeouts in TCP connections. The reason was found to be in a scheduling algorithm allocating a long timeslot for one data flow, which starved other flows. To avoid this problem, the scheduler should assign weights and a switching interval to flows appropriately. Blocking caused by voice calls is more difficult to address, as most mobile terminals are only capable of either a data or voice connection at a time.

4.3 Efficiency at the Link Layer

By tracing data transfers in GPRS we have identified the following problems related to tradeoffs between higher throughput and an efficient use of resource [P7]. These appear to be general issues worth considering for any selective repeat link protocol.

Idle retransmissions. The first issue is premature retransmissions when there are unacknowledged blocks outstanding on the RLC layer but no new blocks to transmit. The RLC sender retransmits unacknowledged blocks in round robin until an ACK is received. On one hand, it increases the probability of data blocks to get through the radio link. On the other hand, it may waste radio resources and battery power of an MS. However, the MS has no knowledge whether there is new data in BSC to be sent and therefore has to decode the assigned timeslots anyway, which consumes the battery power as well. Avoiding such retransmissions when other users have data to transmit would prevent waste of radio resources.

Redundant retransmissions. The second issue is multiple retransmissions of lost blocks when an RLC receiver generates multiple ACKs in response to lost data. Such ACKs indicate the same lost blocks and therefore can trigger unnecessary retransmissions at the RLC sender. The sender normally retransmits lost radio blocks already on a first ACK indicating their loss. However, the sender cannot tell if a next ACK indicating the same blocks as missing is generated before the retransmissions arrived to the receiver or that the retransmissions were lost. A timer at the RLC receiver could prevent repeated retransmission of blocks approximately for one RLC RTT ([58], p. 355). This gives enough time for the retransmissions to arrive and be acknowledged. Alternatively, the receiver can generate ACKs less frequently.

ACK frequency. According to specifications the radio network controls generation of ACKs and schedules ACKs in uplink and downlink 'when needed' [56]. Less frequent ACKs preserve radio resources and battery power, but increase the probability of stalling the window. We suggest that a BSC during uplink transfers sends a selective ACK immediately or shortly after a missing block is detected [45]. On the other hand, when all blocks are received correctly, infrequent ACKs suffice. In downlink transfers, a BSC can poll an MS for ACKs more frequently when the link quality is poor.

Amount of allocated radio resources. Radio resources should be allocated for the user when needed, i.e. when there is enough data to be sent in the given direction. It is not always the case; we observed in GPRS that redundant downlink resources are allocated during an uplink data transfer using TCP. Three timeslots are assigned for an MS in the downlink direction. Only small-size TCP ACKs are sent in downlink, which leaves a large part of assigned resources unused. Amount of required resources could be determined by monitoring the recent traffic volume or amount of buffered data for an MS. We suggest allocating only the minimum resources (e.g. one timeslot) per MS by default. If the initially assigned radio resources are fully utilized, the MS should be allocated more resources to support its sending rate. If the MS decreases the transmission rate, the amount of allocated resources should be decreased.

4.4 New Charging Principles

It is a current practice to charge users in a packet-switched wireless network based on the amount of bytes that cross a gateway connecting the wireless network to an external packet network. With this approach the user pays for data discarded within the wireless network, for example due to buffer overflows. On the other hand, the user (in this context a user refers to a human user, the TCP/IP stack in user's laptop, or radio protocols in the user's mobile terminal) is not motivated to utilize the radio network efficiently as retransmissions over the radio interface are free. We suggest that data delivery in poor radio conditions, mobility and delivering of bursty traffic should be seen as additional services provided by the operator. These services present an additional value to the customer and a cost in radio resources to the operator. Therefore, they can be charged more than a basic data delivery service to match the actual use of the radio interface by user data and signalling. This proposal solves both problems present while charging at the network gateway.

Overcharging. The bottleneck buffer queue in the downlink direction is located downstream

from the router where charging data is collected. Therefore, if data arrives from the external network at twice the rate that the wireless link can handle, half of the packets are lost due to buffer overflow. The user is paying for data not received. When charging counters are located in the radio network this situation is avoided.

Undercharging. If charging counters are collected before the radio link, the user is not interested in improving the efficiency of the radio resource use. For instance, the user is not interested to move closer to a base station (it could be visible as a tower or located by a field strength indicator in a mobile terminal), as retransmissions done by the radio link are free although they consume radio resources. Furthermore, such activities as mobility or allocation of a radio channel due to bursty traffic create plenty of signalling loading control channels and possibly degrading service provided to other users. Thus, charging for the use of the radio link may motivate users to switch to a coding scheme that requires less retransmissions although provides a lower data rate, move closer to a base station, download files in a single cell, and deploy applications producing less bursty traffic. These incentives increase the capacity of a radio network, which is good for other users and the operator.

Shortcomings of the proposed approach are in increased complexity of the charging system and in user acceptance of an idea of paying more for receiving worse service. Furthermore, data transmission in poor radio conditions or on the move often has a direct cost to the user in increased response times and lower throughput that might already provide enough incentive to transfer data in more favorable conditions.

In summary, charging is a powerful tool to stimulate useful behavior from the user and from the operator side. For instance, the operator could be obligated to charge less for data retransmitted by the transport protocol such as TCP or RTP. Consequently, the operator would be more interested in providing a higher quality link with less losses and delay jitter to the user.

5 End-to-End Optimizations

This section describes improvements at the end-to-end transport layer. First, we propose three TCP enhancements. The general TCP profile for 2.5G3G is a starting point for our further study. TCP robustness to delay jitter is improved in the normal transmission state and during a loss recovery phase. Second, we show how deadline-based queueing at the link layer can be utilized by a real-time transport protocol. Finally, we propose a receiver congestion manager which controls non-real-time and real-time flows at the mobile host to prioritize among them and reduce congestion losses.

5.1 The General TCP Profile for 2.5G3G Networks

Operators who have control over handset configuration, such as NTT DoCoMo, as well as standardization organizations, such as WAP Forum, who wish to adapt TCP for use in 2.5G3G networks would benefit from the recommendations on a wireless TCP profile. A document [P6] defines and motivates use of state-of-the-art standard-track TCP features found in modern TCP stacks. These TCP features are widely available, can be used safely in the Internet, and include:

- A large initial window
- A window scale option

- The Limited Transmit algorithm [3]
- Discovery of the path Maximum Transfer Unit (MTU)
- Selective Acknowledgments (SACK) [38]
- Explicit Congestion Notification (ECN) [48]
- A timestamp option [27]
- Disabling TCP/IP header compression which is not robust to packet losses.

We found that the TCP timestamp option increases accuracy of RTT measurements in bandwidth-limited networks and decreases likelihood of spurious timeouts [P4]. According to previous work, the timestamp option was not considered useful for a general use in the Internet [2].

5.2 Responding to Spurious Timeouts in TCP

A basic TCP response to spurious timeouts was suggested by Ludwig [33]. When a timeout occurs, the Eifel algorithm at the sender stores current values of the slow start threshold and the congestion window. Upon detecting a spurious timeout, the sender can restore them and resume transmission with the next unsent segment. We improved the Eifel response to achieve the following goals:

- Efficient recovery from packet losses
- Appropriate restoration of the congestion control state
- Adapting the retransmit timer to avoid further timeouts.

The results are published in [P5], [P8]. Our proposed response of a TCP sender to a spurious timeout is as follows. After a spurious timeout the transmission always resumes with the next unsent segment and TCP always restores the congestion window. We also recommend restoring the slow start threshold, but optionally the threshold can be limited to the congestion window. The TCP sender uses Limited Transmit [3] together with Forward Acknowledgments (FACK) [37] or NewReno-SACK [P8] and adapts the RTO using the back-off counter. We believe this response is efficient and robust under a wide range of networking conditions and increases throughput and goodput. Over links with a high delay-bandwidth product such as in a 3G or satellite environment we obtained up to 350% gain in throughput compared to TCP without the Eifel algorithm [P8].

5.3 Avoiding Spurious Timeouts in TCP Loss Recovery

The Eifel algorithm can be applied for detection of spurious timeouts when a TCP connection is in the slow start or congestion avoidance phase as illustrated in Figure 4 on page 4. However, spurious timeouts can also occur in the fast recovery phase that starts when a fast retransmit is triggered by DUPACKs (upon reaching the threshold, DupThresh, which is commonly set to three DUPACKs) and ends when the foremost segment outstanding on a first DUPACK (the ‘recovery point’) is acknowledged [28].

A spurious timeout in a fast recovery phase causes an unnecessary reduction of congestion control parameters and unnecessary retransmission of segments that in certain cases can even destabilize the TCP connection. The latter is possible because unnecessary retransmissions cause a long series of DUPACKs sent in response to duplicate segments. The TCP sender in this situation can experience repeating spurious timeouts.

The problem of spurious timeouts in the fast recovery phase can be solved by a more conserva-

tive restart policy for the retransmit timer and a careful retransmission scheme. On the other hand, the fast recovery phase in a situation of a real packet loss must not be significantly prolonged. These two conflicting goals are addressed by the following algorithm [P4]:

- When receiving a DUPACK, restart the retransmit timer if the number of received DUPACKs is not more than DupThresh
- After exceeding the DupThresh, restart the timer on DUPACKs only when an advanced loss recovery scheme is implemented which is capable of recovering lost retransmissions
- If a timeout does occur during a DUPACK series, ignore further arriving DUPACKs

The first two steps ensure that spurious timeouts do not occur very frequently in the recovery phase, while the third one ensures that a spurious timeout has only little negative effect. Recovery of lost retransmissions in the second step can be implemented by counting DUPACKs or utilizing the information from SACK blocks.

5.4 Exploiting Packet Lifetime for Real-Time Transport

Achieving timely delivery and avoidance of unnecessary transmissions is difficult purely on the end-to-end basis, especially when such disruptive events as delay spikes are unavoidable in the network. The end points have little information about the current network conditions, about how much data is still in the network and where it is buffered. The lack of information causes two different problems [P9]. First, in wireless networks the transport protocol can prematurely assume that outstanding segments were lost and retransmit them, while the original segments are still held by the link layer. A concurrent retransmission by the link and the transport protocol layer is called competing error recovery ([35], p. 34). Second, the application can re-generate a fresh version of a data object rendering the old object buffered in the network obsolete and blocking the way for the new data.

A possible approach to these two problems is to minimize the amount of data kept in the network by configuring an appropriately small router buffer size. However, a large delay-bandwidth product, a large number of hops, and rapidly changing bandwidth in the network can put a limit on the lowest buffer size. Thus, it is desirable to control for how long the network is allowed to buffer the data. Carrying the packet lifetime that controls the link level persistency solves the first problem. We believe this further advances the idea of differentiated treatment of packets at the flow-adaptive link layer [36]. This solves the problem of competing error recovery, because by the time the transport protocol performs a retransmission, it can be sure that the link layer has already given up on the packet in question. The second problem is solved when applications provide the transport protocol with the lifetime of a data object, so that the transport layer struggles to deliver the object within its lifetime but discards it afterwards. This prevents obsolete application data in the network to block the way for a newly generated data objects.

The proposed solution can be useful, for instance, for streaming applications. Such applications may not require perfect reliability, but cannot tolerate high delays variation in data delivery as it causes a buffer depletion and discontinued playback. The option of increasing the size of the playback buffer is not always available due to memory and battery power limitations of portable devices. On the other hand, delivering live streaming limits the amount of possible delay introduced by a large buffer. Often, a streaming application would greatly benefit if several attempts are made by the transport protocol to recover lost data, as long as the buffer is not depleted.

Communicating data lifetime to the wireless link is essential for detection of stale data. Some of existing wireless networks already allow a wireless host to signal the maximum buffering delay [55]. Alternatively, IP packets can carry an expiration deadline in the IP timestamp option. This is

convenient when the real-time server determines data lifetime, but requires a global clock for the server and the base station. A transport protocol sets packet lifetime to the minimum of application provided lifetime and the retransmission timeout value thus eliminating duplicate delivery but allowing for selective recovery of packet losses.

5.5 Receiver Congestion Manager

QoS requirements of applications need to be mapped into transmission capabilities available at the lower layers. Classifying packets into QoS classes in the network purely on IP or TCP headers can be difficult. For instance, the HTTP protocol can be used to transport nearly any kind of traffic including background downloads, interactive web browsing and streaming [31]. Furthermore, a user may temporary put a higher priority to a bulk data application, which would normally be considered low-priority background traffic in the network. Therefore, the user should have control over the priority of active transfers, perhaps through an interactive tool that would display and prioritize currently active data flows.

We suggest a congestion manager at a data receiver which is located behind a slow access link [22]. It utilizes locally available information such as the bandwidth of the access link and priorities of data flows from the user. It could improve QoS by reducing congestion losses, preventing excessive buffering in the network, reducing the response time for interactive applications and preserving performance when handovers occur. For TCP connections, the receiver sets the aggregate window slightly above the bandwidth-delay product of the access link. The receiver then allocates a part of that aggregate window to each TCP connection based on its priority. This allows giving more bandwidth to high priority connections and avoids overbuffering and congestion losses. It is possible because the rate of each flow is limited to a fraction of bandwidth of the bottleneck link. Furthermore, it allows a faster adaptation during inter-system handovers as follows. When a TCP sender is limited by the receiver window, it can still increase the congestion window. Therefore, by increasing the advertised window after switching to a high-speed network, the receiver can cause a burst at the TCP sender that will allow utilizing the new link.

For UDP receivers, it is possible to deliberately drop packets to cause throttling of flows. As an alternative to the receiver window, the receiver can also use an Explicit Congestion Notification (ECN) [48] for keeping the flows at the desired share of the access link bandwidth. ECN can also be used to control UDP flows that implement some form of congestion control. However, ECN does not allow triggering bursts for TCP such as with the receiver window.

The proposed solution assumes that the receiver knows the rate of an access link. When it is not known, it can be estimated, for example, using Additive Increase Multiplicative Decrease (AIMD) algorithm, similar to the bandwidth estimation at the TCP sender. The ideas presented in this section are preliminary as we performed only an initial evaluation of the proposal.

6 Research History

This section describes the contribution of the author and main results for each publication included into this thesis. The first publicly available performance measurement of GPRS is presented in [P1]. Measurements show on the average a 50% degradation in throughput and occasional loss of connectivity during bulk TCP transfers due to user mobility. The author's contribution is in performing measurement tests and writing the analysis part of the paper. In [P2] reasons for the performance problems are explained. First, we measured the duration of cell reselections in GPRS, which is found to be the main source of TCP misbehavior. Cell reselections can cause an outage in the data transmission as high as 15 sec. The GPRS environment is reproduced in the Seawind emulator to study the effect of such delay spikes on Linux, FreeBSD and Windows TCP implementations. The results show that all TCPs exhibit so-called go-back-N behavior after a spurious timeout resulting from a delay spike. The New Reno algorithm [14] is shown to worsen TCP recovery from spurious timeouts. Furthermore, FreeBSD and Windows TCP have even presented unstable behavior after a spurious timeout producing a chain of unnecessary retransmissions. The entire study was performed by the author.

The Seawind emulator is presented in detail in [P3]. Seawind is a tool that allows emulating the transmission paths provided by cellular wireless networks and allows testing of real protocol implementations. The author's contribution is in the design and implementation of the core of the emulator including event scheduling, real-time execution, delay and packet loss generation, and in being the key author of the paper.

Further work on TCP concentrated on improving robustness to delay spikes. In [P4], the TCP error recovery phase is enhanced by appropriate restarting of the retransmit timer and suppressing unnecessary retransmits. In combination with existing recovery methods, the proposed scheme allows avoiding the retransmission timeouts caused by lost retransmissions. Furthermore, this study illustrates the usefulness of timestamps [27] for TCP connections over bandwidth-limited links. The entire study was performed by the author.

In [P5], we evaluated the Eifel algorithm [33] for detection and recovery from spurious timeouts in a simulated GPRS environment. In the environment without packet losses, an improvement of 20% in goodput and 12% in throughput can be achieved. Surprisingly, when packet losses are present, Reno with Eifel performed worse than Reno due to poor recovery from packet losses. The main contribution of the paper resides in this finding and in showing that using standard enhancements such as SACK [38] and Limited Transmit [3] can largely overcome this difficultly. The entire study was performed by the author.

The general TCP profile for 2.5G3G networks is specified in [P6]. The author's contribution is in writing part of the introduction, Section 2 reviewing link characteristics of 2.5G3G networks, Section 3.3 describing the GPRS architecture, and Section 4.8 advocating the use of timestamps. The GPRS protocol stack operation is traced in [P7]. Author's contribution is in performing part of the measurements, analysis of all the measurement data, and writing the paper.

Both [P8] and [P9] present on-going work performed by the author. In [P8] the goal is to finalize the response to spurious TCP timeouts for the Eifel algorithm and show that it is useful and safe for widespread deployment. A side contribution of this study is in extending the NS simulator with a delay generator tool and implementation of the Eifel algorithm [19]. In [P9] the goal is to exploit lifetime data available for real-time traffic to avoid unnecessary transmissions of stale data over an expensive wireless link.

7 Conclusions and Future Work

Achieving efficient transport in 2.5G3G networks implies meeting QoS requirements of applications while preserving radio resources, battery power and friendliness to other flows in the Internet. Such events as high delay spikes, bandwidth oscillation and connectivity outages are difficult to prevent in the 2.5G3G environment. Therefore, achieving efficient transport demands coordinated efforts from the radio network and the end-to-end transport protocol. We have introduced a framework that allows achieving efficient transport for both non-real-time and real-time flows.

The concept of flow-adaptive links together with the Eifel algorithm for reducing inefficient cross-layer interactions [35] has been an important step towards efficient transport. We studied three new types of inefficient cross-layer interactions including on-demand channel allocation, fragmentation and scheduling at the link layer. We explained how these inefficient interactions can be resolved in 2.5G3G networks. Ideally, these mechanisms should be tailored to the pattern of the user traffic. We showed that the current practice of placing a large drop-tail buffer before the wireless link is poor, as it allows for high latency and unnecessary delivery of stale data. We calculated the optimal buffer size for 2.5G3G networks and proposed active queue management algorithms to prevent poor TCP behavior. We extended the concept of flow-adaptive links to include the explicit lifetime of data packets. This allows the link error recovery protocol to perform a limited amount of retransmissions until the packet is still worth delivering to the receiver. On the other hand, stale data is removed from the queue, thus preserving radio spectrum and battery power.

At the end-to-end transport layer, our work focused on TCP optimization and efficient delivery of real-time data. A proposed general TCP profile for 2.5G3G networks is used in the WAP v2 specifications and in TCP implementations in mobile phones by NTT DoCoMo. We enhanced the Eifel response to spurious TCP timeouts making it more robust to packet losses and repeated delay spikes. In addition, we improved TCP robustness to delay spikes during the loss recovery phase. We showed that with support from the link-layer, a real-time transport protocol can eliminate unnecessary delivery of stale data over the wireless link. We studied interactions of expiration losses at the link layer with existing end-to-end congestion control algorithms. Furthermore, necessary adjustments to the calculation of the retransmission timer at the transport protocol are explained.

Finally, in heterogeneous network environments the end systems cannot always rely on the network to provide the required QoS. The network may not be able to appropriately prioritize among flows or provide adequate buffering. Furthermore, only the consumer of the data, i.e. the user, can properly assign priorities for concurrent data flows. Therefore, we suggest a Receiver Congestion Manager to selectively throttle data flows at the mobile host. Our preliminary results suggest that it can prioritize among flows, reduce congestion losses, prevent excessive buffering in the network, reduce the response time for interactive applications and preserve performance when handovers occur.

Our future work on GPRS will include a study of network-controlled cell reselections, measurements of throughput and battery lifetime under varying radio conditions and network load, and performance of streaming. UMTS measurements will commence as soon as the first terminals become available. Implementing QoS in the GPRS and UMTS network is also in our plans. We are further developing a more accurate TCP retransmission timer based on [34]. The accuracy of a timer assumes a reduced fraction of spurious timeouts and moderate waiting time for unavoidable timeouts.

References

[1] M. Allman, A. Caldwell, and S. Ostermann. ONE: The Ohio Network Emulator. Technical Report TR-19972, School of Electrical Engineering and Computer Science, Ohio University, August 1997.

[2] M. Allman, V. Paxson, On Estimating End-to-End Network Path Properties, In Proceedings of ACM SIGCOMM'99, September 1999.

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

[4] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, R. H. Katz, A Comparison of Mechanisms for Improving TCP Performance over Wireless Links, IEEE/ACM Transactions on Networking, Vol 5, No 6, pages 756-769, December 1997.

[5] H. Balakrishnan, H. Rahul, S. Seshan, An Integrated Congestion Management Architecture for Internet Hosts, In Proceedings of ACM SIGCOMM'99, September 1999.

[6] G. Brasche, B. Walke. Concepts, services and protocols of the new GSM phase 2+ General Packet Radio Service, IEEE Communications Magazine, Vol 35, No 8, pages 94-104, August 1997.

[7] E. Brewer, R. H. Katz, E. Amir, H. Balakrishnan, Y. Chawathe, A. Fox, S. Gribble, T. Hodes, G. Nguyen, V. Padmanabhan, M. Stemm, S. Seshan, and T. Henderson, A Network Architecture for Heterogeneous Mobile Computing, IEEE Personal Communications Magazine, Vol 5, No 5, pages 8-24, October 1998.

[8] Caida, Traffic Workload Overview, http://www.caida.org/, June 2002.

[9] D. D. Clark, D. L. Tennenhouse. Architectural considerations for a new generation of protocols, In Proceedings of ACM SIGCOMM'90, August 1990.

[10] H. Ekstrom, R. Ludwig, Active Queue Magement for Links with Low Statistical Multiplexing, submitted for publication.

[11] N. Feamster and H. Balakrishnan, Packet Loss Recovery for Streaming Video, 12th International Packet Video Workshop, Pittsburgh, PA, April 2002.

[12] N. Feamster, H. Balakrishnan, Adaptive Video Streaming, http://nms.lcs.mit.edu/projects/videocm/, August 2002.

[13] S. Floyd, V. Jacobson, Random early detection gateways for congestion avoidance, IEEE/ACM Transactions on Networking, Vol 1, No 4, pages 397-413, August 1993.

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

[15] S. Floyd, K. Fall, Promoting the Use of End-to-End Congestion Control in the Internet, IEEE/ACM Transactions on Networking, Vol 7, No 4, pages 458-472, August 1999.

[16] S. Floyd, M. Handley, J. Padhye, J. Widmer, Equation-Based Congestion Control for Unicast Applications, In Proceedings of ACM SIGCOMM'00, August 2000.

[17] S. Floyd, V. Paxson, Difficulties in Simulating the Internet, IEEE/ACM Transactions on Networking, Vol 9, No 4, pages 392-403, August 2001.

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

[19] A. Gurtov, Extensions of NS2, http://www.cs.helsinki.fi/u/gurtov/ns/, August 2002.

[20] A. Gurtov, Technical Issues of Real-Time Simulation on Linux, In Proceedings of Finnish Data Processing Week'99, June 1999.

[21] A. Gurtov, M. Raitola, Deleting packets from aborted TCP connections, a patent application filed by Sonera, June 2002.

[22] A. Gurtov, A method to achieve QoS at the mobile data receiver, a patent application filed by Sonera, September 2002.

[23] IETF, Context Transfer, Handoff Candidate Discovery, and Dormant Mode Host Alerting , http://www.ietf.org/ html.charters/seamoby-charter.html, July 2002.

[24] IETF, Robust Header Compression, http://www.ietf.org/ html.charters/rohc-charter.html, July 2002.

[25] IETF, Audio/Video Transport Working Group, http://www.ietf.org/html.charters/avt-charter.html, July 2002.

[26] J. Irvine, D. Pesch, D. Robertson, D. Girma, Efficient UMTS Data Service Provision using INFOSTATIONS, In Proceedings of IEEE Vehicular Technology Conference'98, May 1998

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

[28] V. Jacobson, Congestion avoidance and control, In Proceedings of ACM SIGCOMM '88, August 1988.

[29] P. Karn, C. Partridge, Improving Round-Trip Time Estimates in Reliable Transport Protocols, In Proceedings of ACM SIGCOMM'87, August 1987.

[30] L. Kleinrock, Breaking Loose, Communications of the ACM, Vol 44, No 9, pages 41-45, September 2001.

[31] B Krishnamurthy, J. Rexford, Web Protocols and Practice: HTTP/1.1, Networking Proto-

cols, Caching, and Traffic Measurement, Addison-Wesley, May 2001.

[32] R. Ludwig, B. Rathonyi, A. Konrad, K. Oden, A. Joseph, Multi-Layer Tracing of TCP over a Reliable Wireless Link, In Proceedings of ACM SIGMETRICS'99, May 1999.

[33] R. Ludwig, R. H. Katz, The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions, ACM Computer Communication Review, Vol 30, No 1, January 2000.

[34] R. Ludwig, K. Sklower, The Eifel Retransmission Timer, ACM Computer Communications Review, Vol 30, No 3, July 2000.

[35] R. Ludwig, Eliminating Inefficient Cross-Layer Interactions in Wireless Networking, Doctoral Dissertation, Aachen University of Technology, Germany, April 2000.

[36] R. Ludwig, A. Konrad, A. D. Joseph, R. H. Katz, Optimizing the End-to-End Performance of Reliable Flows over Wireless Links, ACM/Baltzer Wireless Networks, Vol 8, No 2/3, March 2002.

[37] M. Mathis, J. Mahdavi, Forward Acknowledgment: Refining TCP Congestion Control, In Proceedings of ACM SIGCOMM'96, August 1996.

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

[39] M. Meyer, TCP Performance over GPRS, In Proceedings of IEEE Wireless Communications and Networking Conference'99, September 1999.

[40] M. Mouly, M.-B. Pautet, The GSM System for Mobile Communications, Cell&Sys, 1992.

[41] NetHawk, http://www.nethawk.fi, August 2002.

[42] G. T. Nguyen, R. H. Katz, B. D. Noble, M. Satyanarayanan, A Trace-based Approach for Modeling Wireless Channel Behavior, In Proceedings of Winter Simulation Conference, December 1996.

[43] L. Ong, J. Yoakum, An Introduction to the Stream Control Transmission Protocol (SCTP), RFC3286, May 2002.

[44] S. Parker, C. Schmechel, Some Testing Tools for TCP Implementors, RFC 2398, August 1998.

[45] M. Passoja, M. Raitola, O. Aalto, A. Gurtov, Improvement for RLC uplink acknowledgements of GPRS, a patent application filed by Sonera, January 2002.

[46] J. Postel, User Datagram Protocol, RFC768, August 1980.

[47] J. Postel, Transmission Control Protocol - DARPA Internet Program Protocol Specification, RFC 793, September 1981.

[48] K. Ramakrishnan, S. Floyd, D. Black, The Addition of Explicit Congestion Notification

(ECN) to IP, RFC 3168, September 2001.

[49] P. Sarolahti, M. Kojo, and K. Raatikainen, F-RTO: A New Recovery Algorithm for TCP Retransmission Timeouts, University of Helsinki, Department of Computer Science, Technical Report C-2002-07, February 2002.

[50] L. Rizzo, DummyNet: a simple approach to the evaluation of network protocols, ACM Computer Communication Review, Vol 27, No 1, pages 31-41, January 1997.

[51] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, RTP: A Transport Protocol for Real-Time Applications, RFC1889, January 1996.

[52] H. Schulzrinne, Internet Telephony and Multimedia Services, http://www.cs.columbia.edu/IRT/imm/, August 2002.

[53] A. C. Snoeren, J. Salz, D. G. Andersen, H. Balakrishnan, F. Kaashoek, The Migrate Internet Mobility Project, http://nms.lcs.mit.edu/projects/migrate/, August 2002.

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

[55] The Third Generation Partnership Project, QoS Concept and Architecture, TS 23.107 V5.4.0, http://www.3gpp.org/specs/specs.htm, March 2002.

[56] The Third Generation Partnership Project, TS 04.60 V8.6.0 Mobile Station (MS) - Base Station System (BSS) interface; Radio Link Control/ Medium Access Control (RLC/ MAC) protocol, http://www.3gpp.org/specs/specs.htm, March 2002.

[57] UCB/LBNL/VINT, The NS2 network simulator, http://www.isi.edu/nsnam/ns/, August 2002.

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

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

[60] Z. Wang, Internet QoS: Architectures and Mechanisms for Quality of Service, Morgan Kaufmann, 2001.

[61] WebTP, A User-Centric Framework for WebTransport, http://webtp.eecs.berkeley.edu/, December 2000.

[62] H. Wiemann, A. Schieder, H. Ekström, Enhanced TBF Features in GERAN, In Proceedings of the Forth International Symposium on Wireless Personal Multimedia Communications, Septemeber 2001.

[63] G. Xylomenos, G. Polyzos, P. Mahonen, M. Saaranen, TCP Performance Issues over Wireless Links, IEEE Communications Magazine, Vol. 39, No 4, pages 52-58, April 2001.

[64] G. Xylomenos, G. Polyzos, Quality of Service Support over Multi-Service Wireless Internet Links, Computer Networks, Vol 37, No 5, pages 601-615, July 2001.

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

Measured Performance of GSM HSCSD and GPRS

Jouni Korhonen
Olli Aalto
Andrei Gurtov
Heimo Laamanen
Sonera Corporation
P.O.Box 970
00051 Helsinki, Finland

Abstract – **In this paper we present results of measurements on the performance of GSM HSCSD and GPRS data transmission. We used a measurement tool that we have developed to study the performance of various wireless links as perceived by nomadic applications using the TCP protocol. The results show that in stationary connections the throughput and response time were stable and, in general, close to the theoretical values. However, the throughput and response time varied a lot when connections were used in motion. One of the reasons is that TCP was not capable to adapt itself properly to the variability of QoS of HSCSD and GPRS, and therefore, it did a lot of unnecessary retransmissions causing performance slowdown. The performance of HSCSD was better than the performance of GPRS. Reliability was adequate in stationary connections, but in moving connections there were unwanted disconnections or long pauses in data transfer.**

I. INTRODUCTION

The basic GSM data services [1] have been in the market for about 6 years. However, they have not reached wide scale usage. Some of the basic reasons are low throughput and dependability of GSM data transmission. ETSI [2] has planned a technology path to higher throughput and more dependable services. High Speed Circuit Switched Data (HSCSD) [3, 4] and 14400 bps channel coding (144CC) [5] were the first steps in the path. General Packet Radio Services (GPRS) [6] is entering the market, and Universal Mobile Telecommunications Services (UMTS) [7] will succeed them within next 2-3 years.

HSCSD offers higher transfer rates by combining two or more time slots. It implements a flexible time slot allocation scheme. The allocation of time slots depends on the following factors: end user's subscription, air capacity, and network load. HSCSD uses an Automatic Link Adaptation scheme (ALA) for the best channel coding at each occasion. GPRS enhances GSM data services to a packet switched data transmission. GPRS has also a flexible time slot allocation scheme and ALA. In the first phase, GPRS uses CS-1 (9.05 kbps line rate) and CS-2 (13.4 kbps line rate) channel codings [15].

We made this study to understand the performance of HSCSD and GPRS from the viewpoint of nomadic applications. In the study we used a performance measurement tool (Wireless Link Tester – WLT) [11] that we have developed to study the performance of various wireless links. WLT implements a client – server model and uses TCP as its transport protocol. With WLT we can measure the throughput, round-trip time, and reliability of a wireless link. We made the measurements on HSCSD using the public GSM network and on GPRS using the public GPRS network in Helsinki surroundings operated by Sonera.

Our major results are the following:

1. Generally, in stationary connections the performance was stable meaning that the variation in both the throughput and round-trip time was small and there were only few disconnections [11].
2. In moving connections, the variation in both the throughput and the round-trip time was high, and there were several disconnections or long pauses in data transfer. Disconnections happened on the average every 11th-12th minute. In addition, there were cases where the round-trip time was exceptionally long and the throughput was very low.
3. The measurements indicated that TCP was not capable to adapt itself properly to the variability of QoS of HSCSD and GPRS. Therefore, TCP did unnecessary retransmissions causing significant performance slowdown.

The rest of the paper is organized as follows: Section 2 discusses HSCSD and GPRS. Section 3 describes our measurements including WLT, measurement setup, and traffic loads. We then present the results of the measurements in Section 4. Finally we summarize the study in Section 5.

II. HSCSD AND GPRS

HSCSD consists of two separate technologies: multislot capability and modified channel coding scheme. The former provides the usage of several parallel time slots per user thus increasing the user data rate respectively. The first phase of HSCSD specifications allows the usage of 4+4 time slots, but in practice, the mobile equipment manufacturers implement more limited versions of that. At the beginning of HSCSD services, the maximum number of time slots is mostly limited

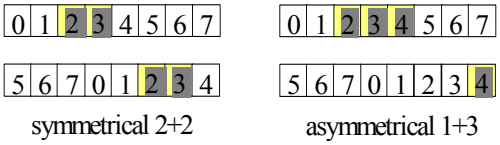


Fig. 1. Example of symmetric and asymmetric HSCSD multi slot-scheme

to 1+3 and 2+2 (Fig. 1). The specifications define asymmetric traffic in such a way, that the amount of time slots in uplink direction cannot exceed the number of time slots in downlink direction.

The 144CC scheme provides a 14.4 kbps line rate instead of the original 9.6 kbps line rate. In order to achieve higher line rates there is a more efficient puncturing method used, which on the other hand decreases the radio interface error correction performance. This means, that 144CC cannot be used, when there are a lot of noise and interference affecting the quality of the radio signal.

The specifications define upgrading and downgrading of air interface resources. It means that the amount of parallel time slots can vary between one and the maximum defined value. Up- and downgrading may occur when the signal level, interference level or capacity varies during a connection. The system can also use ALA, which means that the data rate can be 14.4 kbps or 9.6 kbps for an individual time slot depending on the radio path conditions. Both up- and downgrading of the resources and ALA can happen during a data transfer. Even though, the specifications allow the up- and downgrading and ALA, the combinations might be limited depending on the network and mobile terminal capabilities.

In the first phase, the maximum data rate of HSCSD is limited to 64.0 kbps due to the A-interface [4]. Depending on the connection type and the infrastructure capabilities of the network, the maximum user rate can be 38.4 kbps using ISDN V.110 protocol, and 57.6 kbps using ISDN V.120 protocol.

GPRS maintains the GSM Base Station Subsystem (BSS) access technologies with some modifications to the allocation of air interface resources. In contrary to HSCSD, time slots are allocated for a terminal when needed and up to eight terminals may share one time slot. GPRS uses a flexible time slot allocation scheme with up to 8+8 time slots. GPRS has four channel coding schemes (CS-1 9.05 kbps, CS-2 13.4 kbps, CS-3 15.6 kbps, and CS-4 21.4 kbps below the RLC level). In practice CS-1 provides about 8.0 kbps data rate and CS-2 about 12.0 kbps data rate [16]. Different coding has different performance and error resiliency characteristics. Most important new components to support packet switched data transmission in the Network Subsystem are: the Serving GPRS Support Node (SGSN), the Gateway GPRS Support Node (GGSN), and the Border Gateway (BG) [12, 13].

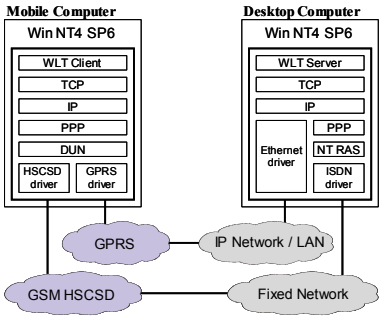


Fig. 2. Configuration of the measurement tool

III. MEASUREMENTS

In this section, we describe our measurement tool, measurement setup, and traffic workloads used in the study.

A. *Measurement Tool*

WLT is designed to measure the throughput and round-trip time of a wireless link. WLT also collects TCP segment counters and link disconnections.

No data compressions (Van Jacobson [8], V.42bis [9], etc.) are used, so the actual throughput – not improved by data compressions – is measured.

The Dialup Networking (DUN) and Remote Access Service (RAS) of Windows NT4SP6a are used to establish the data link connection between the mobile computer and the desktop computer. The configuration of the measurement tool is shown in Fig. 2.

B. *Measurement Setup*

For HSCSD measurements we used 1+3 time slot allocation with 144CC and ALA in the air interface and V.110 protocol in the ISDN interface between MSC IWU and the desktop computer. Fig. 3 shows the HSCSD test configuration. For GPRS measurements we used 1+2 time slot allocation with CS-2 because that was the best the mobile terminal was able to support. During live GPRS tests the ALA functionality was disabled in the GPRS network. The requested reliability class was “Unacknowledged GTP and LLC; Acknowledged RLC, Protected data”. Fig. 4 shows the GPRS test configuration.

We have used following TCP parameters: MSS of 536 bytes and the receiver window of 16 kilobytes. No other TCP options (SACK, timestamps, etc.) were enabled.

C. *Traffic Workloads*

We measured two types of workloads: bulk data transfers

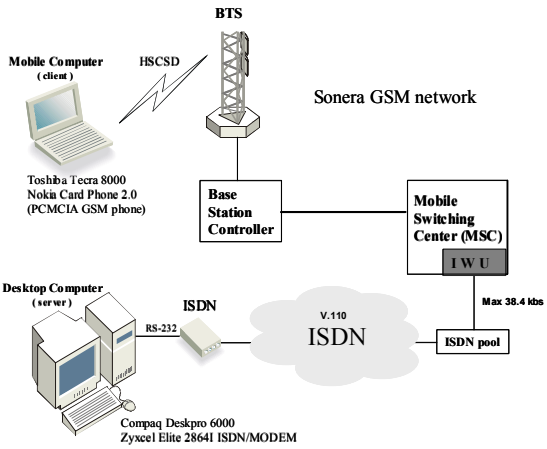


Fig. 3. HSCSD test configuration

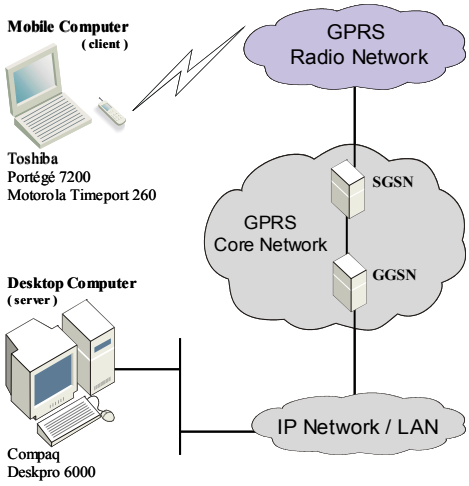


Fig. 4. A model of GPRS used in test

and request – reply transfers. Bulk data transfer simulates for example FTP transfers or a retrieval of large image. The bulk transfer sequences consisted of 40 repetitions of 150000 bytes data transfers.

The request – reply transfers simulate HTTP transactions. The test sequences are described in Table 1. The sizes for request – reply measurements are based on a real HTTP trace collected from one Sonera's Internet service access point and its transparent proxy. The HTTP trace was collected during one week and contains almost two billion requests. The selection criteria for the message sizes of the measurements were the frequencies of message sizes categorized by appropriate ranges. We intentionally rejected browser's cache hits and huge downloads. This scales minimum reply sizes up and maximum reply sizes down than those of typical Web browsing; however, the overall response time of Web browsing can be estimated from our results.

IV. Results of Measurements

In this section, we present the results of the measurements.

A. HSCSD Bulk Data Transfers

Stationary connections had generally a stable throughput [11]. Measured throughput for moving connections is illustrated in Fig. 5.

The uplink throughput was the following: maximum 23.3 kbps, minimum 4.1 kbps, and average 9.6 kbps. We experienced some variation in the throughput and few significant peaks. The cause for variation was mostly ALA. The peaks can be explained by our network configuration; the tested network allocated 2+2 time slots instead of 1+3, if there was no capacity to provide three downlink time slots. About 18% of uplink test sequences had significant amount of TCP retransmissions – in some cases even all segments were resent.

The downlink throughput was the following: maximum 27.9 kbps, minimum 13.4 kbps, and average 22.7 kbps. There was less variation in downlink than in uplink throughput. We also experienced less TCP retransmissions; 5% of test sequences had over 1/3 of all segments resent.

TABLE 1
TEST SEQUENCES IN REQUEST - REPLY TRANSFERS

Number of Tests	Average Request Message (bytes)	Average Reply Message (bytes)
19*120	280	499
19*120	348	4758
19*120	539	5070

B. HSCSD Request – Reply Transfers

Response times for each request – reply test sequences are illustrated in Fig. 6. The performance was good; though, the variation increased considerably when the request – reply data size increased. Small transactions (280/499) had the following round-trip times: the average 1.0 sec, the minimum 0.8 sec, and the maximum 7.2 sec. The average round-trip time for medium size transactions (348/4758) was 2.7 sec, the minimum was 2.1 sec, and the maximum was 13.3 sec, whereas for large size transactions (539/5070) the average round-trip time was 3.1 sec, the minimum was 2.3 sec, and the maximum was 26.8 sec. Although there were variations, they were less than half of those of GPRS. Over 60% of the test sequences had only a few TCP retransmissions. But there were some large size (539/5070) test sequences where over 1/3 of segments was resent.

C. GPRS Bulk Data Transfers

Uplink bulk data transfers in GPRS (Fig. 7) were more problematic than downlink, which was similar to HSCSD.

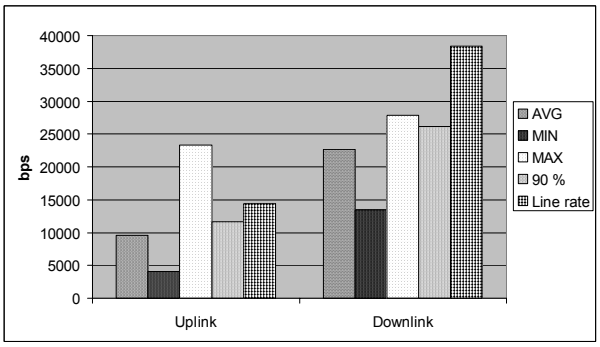


Fig. 5. Throughput of HSCSD (38*150000 bytes transfers)

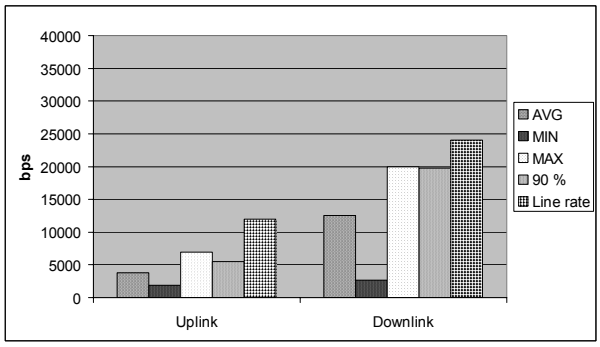


Fig. 7. Throughput of GPRS (40*150000 bytes transfers)

Almost all uplink data transfers have retransmitted segments, in some cases the amount of retransmissions have exceeded the size of transferred user data. Only a small part of retransmissions was recovering lost segments, the larger part of retransmissions was unnecessary. We could observe this by studying the receiver-side TCP traces and comparing the number of segments sent and received. The maximum throughput in the uplink direction was 7 kbps, which is significantly less than the data rate of 12 kbps, even with TCP/IP header overhead included. Some transfers took a very long time to complete. The lowest throughput was only 1.8 kbps, which is more than six times less than the data rate. The average uplink throughput in GPRS of 3.8 kbps was less than a half of HSCSD.

In the downlink direction, only a few transfers have had retransmissions and throughput was good (results cannot be directly compared to HSCSD, since GPRS tests have had two timeslots allocated versus three for HSCSD). Downlink tests were stable; 90% of tests have had throughput close to the maximum of 20 kbps, which still is 17% lower than the data rate of 24 kbps. However, the lowest throughput of 2.6 kbps was almost ten times less than the data rate.

D. GPRS Request – Reply Transfers

On the average, all transactions showed satisfactory performance over GPRS (Fig. 8). However, the maximum response time was extremely high, in the order of several minutes. For small transactions (260/499), we noticed only a few retransmissions, but the minimum response time of 1.5 seconds was almost twice as large as for HSCSD. For medium (348/4758) and large (539/5070) transactions, the minimum response time was approximately 30 % larger than for HSCSD and 90 % of transactions took approximately twice longer to complete over GPRS than over HSCSD. Approximately 10% of segments were retransmitted.

E. Reliability

In general, in the case of HSCSD the reliability of stationary connections was adequate for nomadic end users. However, in moving connections there were disconnections caused by handover failures, air interface failures, and remote equipment failures. On the average, there was a disconnection every 11th - 12th minute. In the case of GPRS, in moving connections there were several cases, where transfer was paused for exceptionally long time. One of the reasons was that GPRS is still in earlier phase, and thus, there were software faults causing performance slow down.

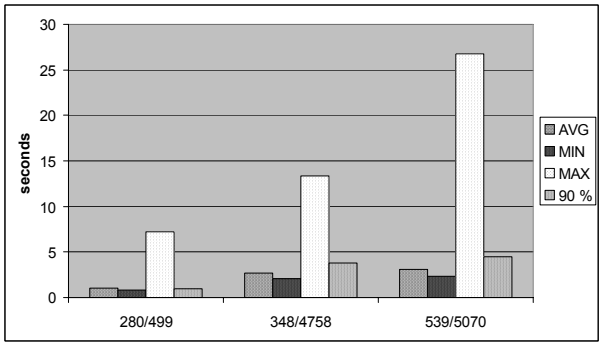


Fig. 6. Response times of HSCSD request – reply transfers

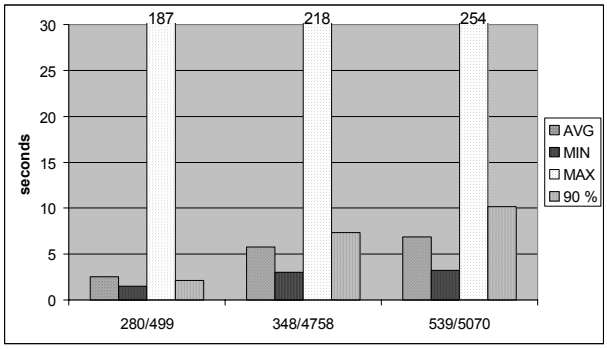


Fig. 8. Response times of GPRS request – reply transfers

V. SUMMARY

In general, in good radio signal quality environment, both HSCSD and GPRS provided significantly better throughput and response time than the basic GSM data service. The throughput and round-trip time in stationary connections were stable. However, in moving connections they may change a lot; for example, the throughput changed between 23.2 kbps and 2.4 kbps and response time changed between 2.3 sec and 26.8 sec. One of the reasons to the high variance was the behavior of TCP. TCP could not cope properly with the characteristics of HSCSD and GPRS data transmission, thus causing performance slow down by doing a lot of unnecessary retransmissions. The problems with TCP are discussed, for example, in [10]. IETF has started to address these problems [14].

HSCSD provided better performance than GPRS does. GPRS had higher variance in performance than HSCSD. The high variance needs to be addressed when developing nomadic applications over GPRS.

The reliability in stationary connections is adequate. But the reliability in moving connections – a disconnection every 11th-12th minute (HSCSD) or long pauses in data transfer (GPRS) – may create problems, for example, in long web browsing sessions or in long file transfers. Therefore, the reliability of moving connections may create problems, if a distributed application cannot cope properly disconnections or long pauses.

Future studies will include performance measurements using simulated UMTS.

REFERENCES

[1] Michel Mouly, Marie-Bernadette Pautet, "The GSM System for Mobile Communications", 1992.

[2] European Telecommunications Standards Institute, WWW.ETSI.ORG, checked 19.2.2001.

[3] GSM Technical Specification, "GSM 02.34, High Speed Circuit Switched Data (HSCSD), Stage 1", Version 5.2.0. ETSI, July 1997.

[4] GSM Technical Specification, "GSM 03.34, High Speed Circuit Switched Data (HSCSD), Stage 2+", Version 5.2.0. ETSI, May 1999.

[5] GSM Technical Specification, "GSM 10.14 version 1.0.1, Digital cellular telecommunications system (Phase 2+), System Overview for 14.4 kbit/s Work Item", ETSI, 1997.

[6] GSM Technical Specification, "GSM 02.60 version 6.1.0, GPRS Service Description, Stage 1", ETSI 1998.

[7] WWW.3GPP.ORG, checked 19.2.2001.

[8] Van Jacobson, "Compressing TCP/IP Headers for Low-Speed Serial Links", RFC 1144, 1990.

[9] ITU-T, "Recommendation V.42bis: Data Compression Procedures for Data Circuit Terminating Equipment (DCE) Using Error Correction Procedures", ITU-R, January 1990.

[10] Reiner Ludwig, and Randy H. Katz, "The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions", *Computer Communication Review*, Vol 30, Number 1, January 2000.

[11] H. Laamanen, J. Penttinen, and J. Laukkanen, "Measured Performance of GSM High Speed Circuit Switched Data Link", *BAS2000*, 5th Computer Networks Symposium, June 2000.

[12] GSM Technical Specification, "GSM 03.60 version 7.4.0, Digital cellular telecommunications system (Phase 2+), General Packet Radio Service (GPRS) Service Description – Stage 2", ETSI, 2000.

[13] Jian Cai, David Goodman, "General Packet Radio Service in GSM", *IEEE Communications Magazine*, October 1997.

[14] G. Montenegro, et al, "Long Thin Networks", RFC 2757, January 2000.

[15] GSM Technical Specification, "GSM 03.64 version 7.1.0, Digital cellular telecommunications system (Phase 2+), Overall description of the GPRS radio interface, Stage 2", ETSI 1999

[16] M. Mayer, "TCP Performance over GPRS", *IEEE Wireless Communications and Networking Conference 1999*, New Orleans, LA, September 21-24, 1999

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.

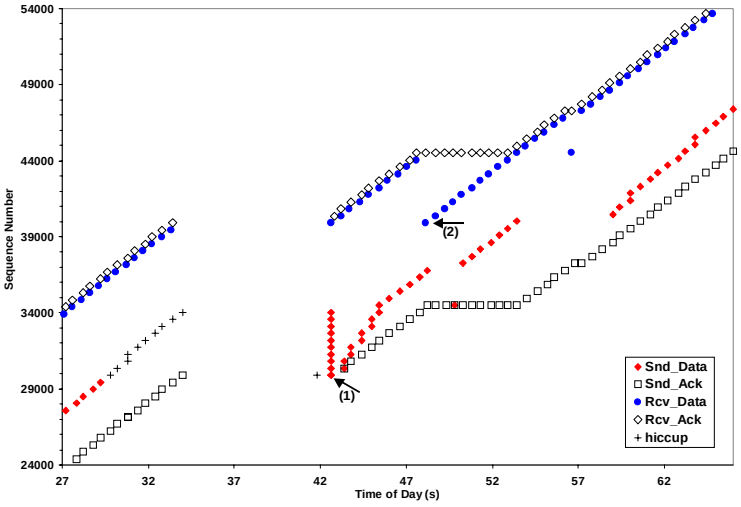


Figure 1. Reaction of TCP on a 13-second delay (sender and receiver traces) [15].

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.

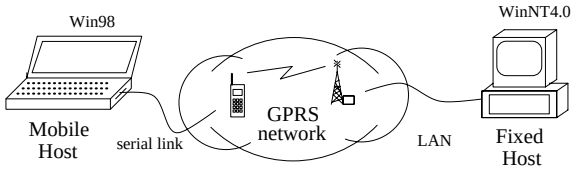


Figure 2. Configuration of measurements in the live GPRS network.

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.

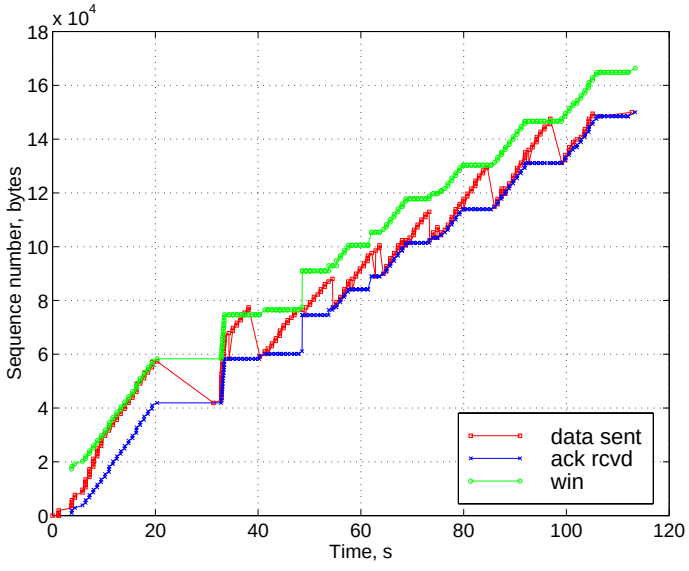


Figure 3. A delay during a downlink bulk TCP transfer in the GPRS network.

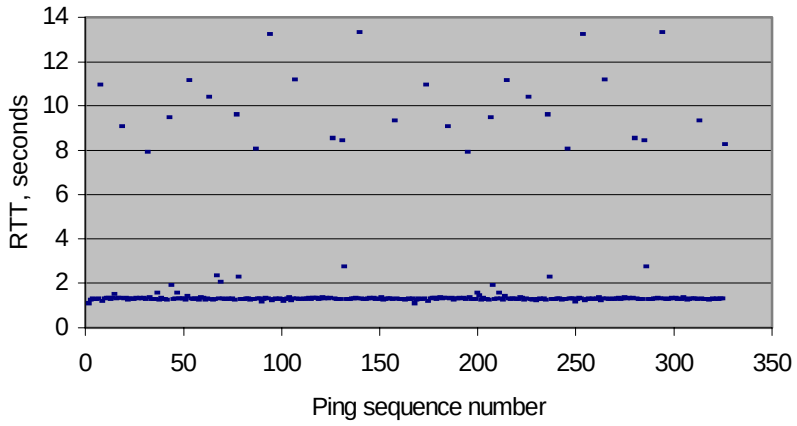


Figure 4. RTT spikes in GPRS when cell reselection is forced from the mobile terminal.

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

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

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 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

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.

3 This idea was generated by Reiner Ludwig in private communication

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, 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.

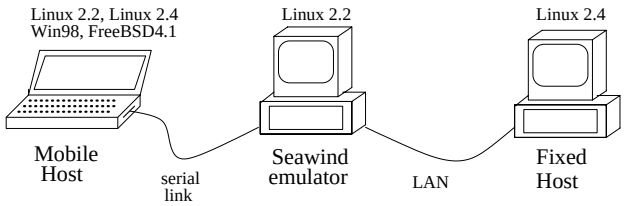


Figure 5. Configuration of measurements with an emulated W-WAN link.

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.

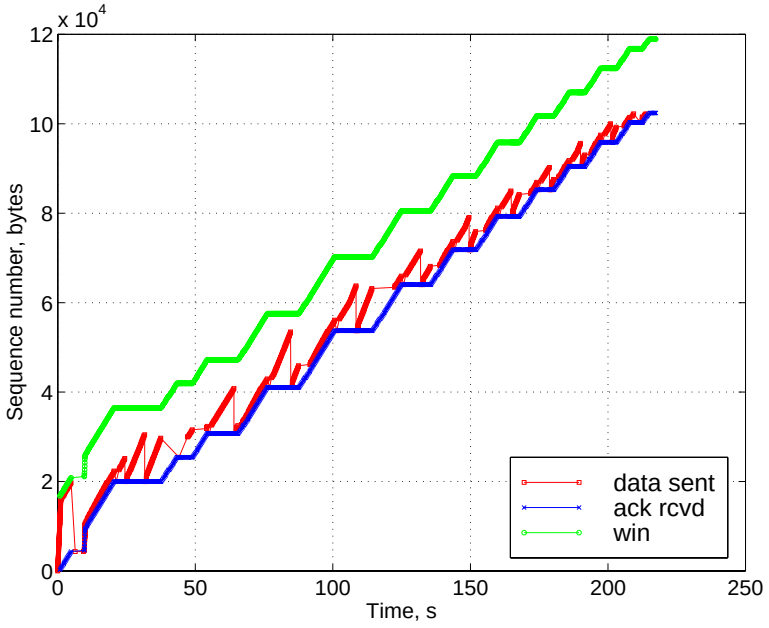


Figure 6. Recovery of TCP in FreeBSD 4.1 from a spurious timeout (complete connection).

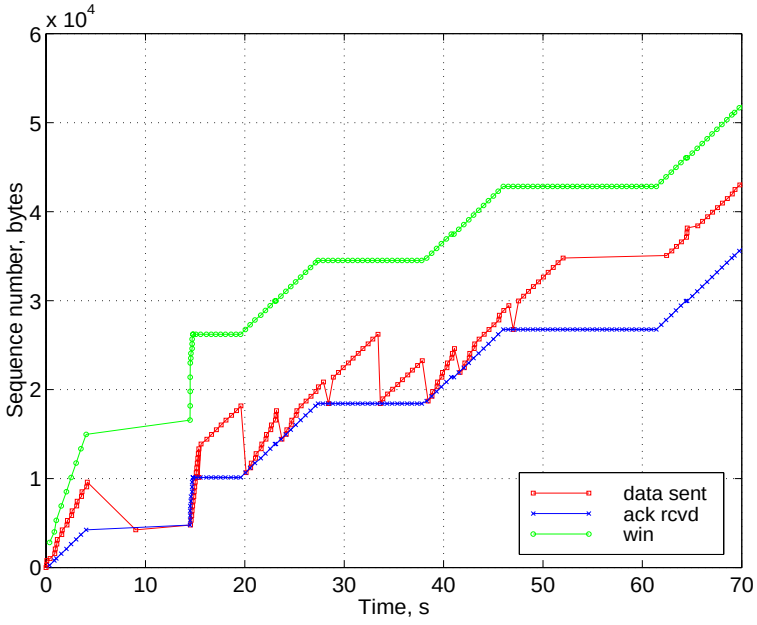


Figure 7. Recovery of TCP in Windows 98 from a spurious timeout (zoomed, total connection time 191 seconds).

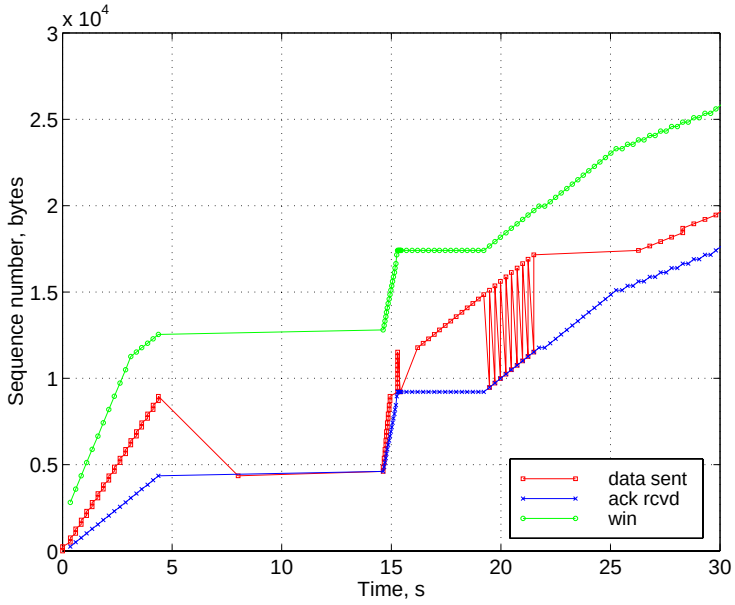


Figure 8. Recovery of TCP in Linux 2.2.12 from a spurious timeout (zoomed, total connection time 115 seconds).

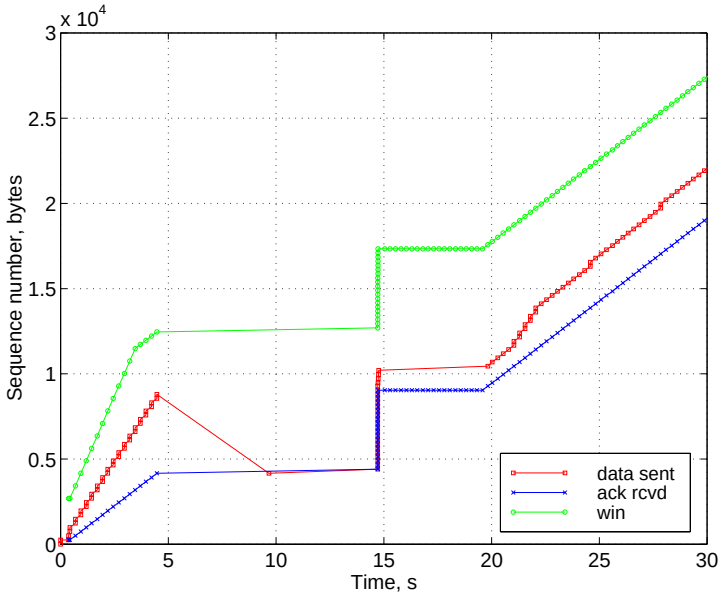


Figure 9. Recovery of TCP in Linux 2.4.0 from a spurious timeout (zoomed, total connection time 114 seconds).

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/IP 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 RFC 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. CONCLUSION

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.

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.

[13] 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-ludwig.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--154, 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, 31(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, 11(6):10--23, November/December 1997.

Seawind: a Wireless Network Emulator

Markku Kojo, Andrei Gurtov, Jukka Manner,
Pasi Sarolahti, Timo Alanko, and Kimmo Raatikainen
University of Helsinki, Department of Computer Science
P.O.Box 26, FIN-00014 UNIVERSITY OF HELSINKI, Finland
{markku.kojo,andrei.gurtov,jukka.manner,pasi.sarolahti,
timo.alanko,kimmo.raatikainen}@cs.helsinki.fi

Abstract

Behavior of current communication protocols as well as current and future networked applications is of fundamental importance for technical and commercial success of Mobile Internet. The forthcoming wireless Wide-Area Networks, such as GPRS and UMTS, are quite complex and network operators have a large set of parameters to tune the transfer performance of these networks. In this situation it is of great value to be able to execute practical experiments. The Seawind emulation software introduced in this paper enables measurements of protocol implementations in modeled networking environments. The Seawind software provides a rich set of ways to define transfer characteristics including delays and errors. The software has also means to conduct large sets of experiments in an automatic fashion. In addition, tools of analyzing measurement data has been integrated into the Seawind software.

1 Introduction

Nowadays Wireless Wide Area Networks (WWAN) are widely used by mobile users to access data services. New mobile data networks, as for example the General Packet Radio Service (GPRS) [6, 8], and future third generation mobile communication systems [32] are expected to provide a high-speed packet data access suitable for a wide range of Internet services. However, wireless links represent a different communication environment than the wireline Internet. Hence, protocols and applications not particularly designed for wireless links often require enhancements in order to achieve reasonable performance in a wireless environment [4, 17].

Evaluating such enhancements over a real data link or network is often costly; if a system is only in a development stage, the evaluation may be impossible. A frequently asked question is whether next-generation wireless networks could provide multimedia services that meet the end-user expectations. Network emulation is a convenient tool to examine how existing multimedia applications behave. An emulator can also be used in usability studies involving real end-users. The main difference to a field trial is that an existing network is replaced by a model describing characteristics of transfer, delay and error behavior, for example. An emulation also allows controlling the network characteristics and reproduce the environment. On the other hand, the problems of emulation include the accuracy of the model; parameters drawn from real-world phenomena and properties are always estimates.

In this paper we describe the Seawind emulator and present a case study that demonstrates its practical utility. The primary target of Seawind is performance studies of real protocols and applications as seen by the end user in the present and future wireless networks. Although Seawind was developed for modeling wireless networks, GPRS in the first hand, the Seawind emulator is rather generic and it can be used in modeling a wide range of networks.

Emulation is a compromise between two other possible approaches in performance evaluation; between simulation and measurements using a testbed [2]. The main advantage of a network emulator is that the performance of actual implementations of protocols and applications can be examined. This is a clear advantage, for example, over most of the TCP performance studies that rely on the abstract TCP model available in the NS simulator [14]. However, most end users – if not all – connected to the Internet use TCP implementations that are not necessarily close to the one found in the NS simulator (for example, those in Windows or Linux). Therefore, the NS simulator or other simulators having their own TCP implementations do not allow a network operator to tune the networking parameters so that the performance is optimized for real end users and their applications. Furthermore, a real-time emulator provides answers to ”what-if” type of questions. It also allows back-engineering parameters of closed networking implementations.

Emulation studies can be time-consuming because the duration of an experimental session is determined by the speed of the modeled network. In order to enhance the usability of the emulator, the Seawind emulator supports an automatic set up of tests and collection of a sufficient number of test repetitions for statistically valid results. Therefore, the experiments can be run overnight and during weekends without any human intervention. The Seawind package also provides basic tools for statistical analysis and for graphical presentation of results. We use Seawind on the Linux operating system. The Seawind software runs in the user space and can be easily ported to any Unix operating system.

Several extensive performance studies have been made using Seawind. TCP performance has been studied in [13, 25] and GPRS performance in [18]. Seawind was also used in the Monads demonstration at MOBICOM 2000.

The rest of the paper is organized as follows. After a brief summary of related work, we discuss, in Section 2, common characteristics of wireless links. We also derive the requirements for a network emulator taking those features into account. In Section 3 we describe the Seawind architecture. We present the structure of the Seawind simulation Process that is the core of the Seawind emulator. In Section 4 we discuss the features of Seawind that are important in emulation of any wireless network. In Section 5 we present how we validated the Seawind emulator. Finally, a case study is described in Section 6 in order to illustrate the practical value of Seawind.

2 Related Work

Simulation of communication networks is an active research area. A wealth of different simulators are found worldwide; most of them are freely available while some are commercial products. The software tools for network simulations can be divided into two categories: discrete event simulators and real-time simulator or emulators, as we call them. Many simulation tools are discrete event simulators that operate in virtual time. These simulators have their own abstract implementations for modeling different links, protocols, and even applications. Probably the most well-known discrete event simulators are NS [14] and the commercial simulator

OPNET [31]. The Monarch Project at Carnegie Mellow University has created a set of wireless and mobile extensions to NS that provide a more detailed model of the physical and link layer behavior of a wireless network and allow arbitrary movement of nodes within the network [7]. Other network simulation tools include MobSim [27], SWimNet [5] and GloMoSim [33]. A parallel environment for the simulation of mobile wireless network systems, based on the parallel simulation language Maisie [3], has been presented in [26].

Discrete event simulators are great tools for overall network performance simulations and other more theoretical testing. These cannot, however, be used with actual protocol implementations and applications, unless the implementations are ported to the simulation package. During a product implementation and test, intermediate versions of the software emerge from time to time, and it would not be feasible to port each version to the simulation environment for testing. In addition, simulations cannot give a real-time view of how a user would experience some service using a new application, protocol, or network. Therefore, real-time tests and actual protocol implementation studies require a real-time simulator.

Real-time simulators or emulators allow researchers to create network topologies and conditions, which are difficult to achieve in a reproducible manner on production networks, or to perform real-time tests with various prototype protocols and products, for example. Such an emulator environment is well controlled and reproducible.

The common nominator within the emulators available in this category is that they provide different delay, packet drop, and queue-handling functionality in order to simulate some communication medium or network. NIST Net [11] is implemented as a kernel module extension to the Linux operating system and an X Window System-based user interface application. NIST Net provides parameters such as delay, packet drop probability, fraction of duplicated packets, and link bandwidth. Dummynet [24] is a similar tool, implemented as a FreeBSD Unix kernel patch in the IP stack. Dummynet works by intercepting packets on their way through the protocol stack; it uses parameters similar to the ones in NIST Net to affect the flow of packets. A third similar kind of emulator is the Ohio Network Emulator, ONE [1]. ONE uses three parameters to simulate a communications network, namely a transmission delay, a propagation delay, and packet queues.

The functionality offered in these emulators enable the simulation of a variety of different links, networks, and protocols. However, parameters such as delay, bandwidth, and queue sizes are not enough for all simulation purposes. A key functionality that is missing in the above emulators is the lack of timed events and changes of the simulated network environments. Especially in wireless networks, the network characteristics can change drastically due to the movement of mobile terminals and even the present weather conditions. To expand the area of studies that can be performed with an emulator, changes in the simulated environment and other timed events, such as handovers, should be provided by an emulator. In addition, the portability of the emulator to other platforms is more complicated in the emulators mentioned above since those have been tightly coupled with specific operating system kernels.

3 Wireless Network Characteristics

In this section we present a summary of properties of wireless links that present challenges for efficient data communication. Wireless links typically have relatively low bandwidths, high latency and high error rates. We discuss how these properties relate to the requirements for the network emulator.

Slow, asymmetric and changing line rate. The line rate of a wireless WAN link does not often exceed some tens of kilobits per second. Such a link speed is typical also for dial-up modem users. For some wireless links, the line rate can vary over time, due to a change in the amount of radio resources assigned to the user or change of the channel encoding scheme. The line rates may be asymmetric, for example when using certain types of satellite links or GPRS. Thus, the emulator should provide the desired line rate by delaying data packets and provide means to emulate changes in the line rate, in both directions independently. For the majority of W-WANs a rate up to 100 kbps is sufficient. However, modeling future broadband wireless networks will require line rates at least up to 2 Mbps.

High latency and variable delays. The propagation delay of wireless links is typically high. The delay comes from the special transmission schemas on the wireless link and from the processing delays of the link hardware. For example, the Global System for Mobile Communications (GSM) uses interleaving of data on the radio link to reduce the effect of error bursts, and this introduces a latency of 90 ms independent of packet size [20]. Additional latency in using a GSM data service is caused by the connection to the Internet Service Provider (ISP) and the processing time within the GSM system. The total one-way latency in GSM sums up to 200-300 ms[1]. The emulator should correctly model this delay by adding a propagation delay to each packet. Variable delays may appear on a wireless link due to a number of reasons, for example Link-level ARQ recovery, radio resource (re-)allocation and handovers to mention a few. There should be a possibility to add such random delays to a packet flow.

Error losses. Some wireless links impose a significant amount of data corruption due to transmission errors. The error rate depends on the current radio conditions and the strength of the channel coding schema. For example, in the transparent GSM data service the residual bit error rate (BER) of the link is allowed to be as high as 10^{-3} after the Forward Error Correction (FEC) [21]. Radio conditions can vary greatly. In the ideal conditions all protocol data units (PDU) are delivered correctly, and in the worst case nothing can be correctly sent over the link. For accuracy of emulations and ease of use the emulator should be able to drop packets on a per-packet basis or using a bit error probability. The transmission error on the link can be seen by the upper layers as a delay in data delivery (reliable link level), loss of a PDU (error detection in link layer) or as a corrupted data packet (transparent link layer). The emulator should provide all three cases.

Congestion losses at the bottleneck queue, over-buffering. 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. Routers play a significant role because congestion data losses are most likely to happen at the bottleneck queue. A limited number of buffers can be allocated in a last-hop router per user. The emulator should contain a queue at the emulated bottleneck link and provide means to limit the queue size in terms of bytes and number of packets. Optionally, a timer would be used to limit the time a packet can be buffered. New queue management algorithms and drop policies should be easily attached.

[1]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.

Handovers. In a cellular radio network the mobility is accomplished by changing the access point that serves the user, according to the user's current location. The handover process may cause data losses and a drastic change in the service provided, when the user moves from a less busy to a more occupied service area. Modeling a handover would cause a change in a number of simulation parameters at once. For example, in a packet radio network, a new service area may have more users using the same shared medium than in the previous location; the user will notice this as less available bandwidth after the handover. In addition, the handover process itself may cause a delay or loss in data delivery. The emulator should be able to allow changing a set of parameters at the same point of time to emulate changes in network service.

Link blackouts. Wireless links are prone to temporal interruptions of service. A typical example is loss of radio coverage; this might happen due to driving in a tunnel or moving away from the serving access point. Blackouts cause a situation when, for a period of time, no user data is successfully transmitted through the link. If QoS support is implemented in the wireless network, higher priority packet traffic can as well cause blocking of lower priority traffic. The emulator should provide means to specify the timing of blackout periods and the handling of PDUs during that time, for example, whether to drop or store the PDUs during the blackout.

Finally, a number of features would enhance the usability of the emulator. It is desirable that the emulator could be running in user space on an unmodified operating system. Binding the emulator in the operating system kernel is not desirable because it complicates the portability of the emulator between different operating systems and even between different operating system versions. The emulator must provide accurate execution of all timed events and notify if some scheduled event slipped past a certain threshold value and would thus affect the result of the test run. The emulator should have an easy to use user interface to enable its use by a wide range of specialists unfamiliar with its implementation details.

4 Seawind Architecture

The fundamental architecture behind the Seawind emulator is shown in Figure 1. Seawind intercepts the traffic flow between the client and the server transparently to the endpoint hosts. The desired link characteristics are emulated by delaying, dropping and modifying packets in the flow. The socket API and the protocol implementations in the client and the server need not be changed. The client and server can be directly connected to Seawind (e.g. by a serial cable), or they can be located anywhere in the network. For example, the latter option is useful, if the researcher wants to experiment with a data transfer over an emulated wireless link to a server located in the global Internet.

We have been mostly interested in studying the TCP/IP protocols and the behavior of TCP-based applications. However, Seawind can be used with any data flow, for example with traffic produced by the WAP protocol [19]. This generic approach allows comparing the performance of different protocols (e.g. WAP and TCP) or different implementations of the same protocol (e.g. Windows TCP and Linux TCP) under the same emulated network characteristics.

4.1 Seawind components

Figure 2 illustrates the Seawind components, which are used in setting up the test runs and running successive performance tests with various parameters automatically. The core of the

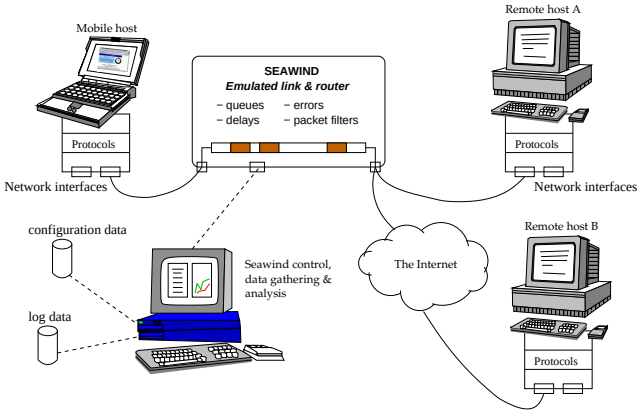


Figure 1: Structure of Seawind.

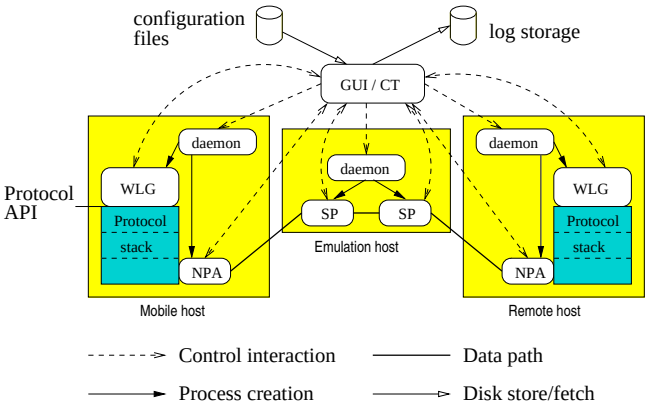


Figure 2: Architecture of the Seawind emulator.

emulation is the *Simulation process (SP)* that cause delays and packet losses to emulate the target link or the target network with various queues and buffers. Before describing the SP functionality in detail, we briefly introduce the other Seawind components shown in Figure 2.

The user sets up the tests using a *Control Tool (CT)* with a graphical user interface. With the control tool the user creates a number of entities called *replication sets*. A replication set defines the workload used in the performance tests and the parameters of the emulated network. For each replication set the user also gives the number of test run repetitions (replications) to be made with the given parameters. After the user has defined a sequence of replication sets to be tested, he may save the parameter settings for later use and start the test run with the given sequence of replication sets.

A replication set configuration consists of a *network configuration* and a *workload configuration* which are set up independently. Any combination of workload and network configurations can be selected to be repeatedly tested in a replication set. Workload configuration defines the tools that are used for generating the workload for the test and the parameters for the tool. Any external tool or script can be used as a workload generator. For example, the `ttcp` tool [30] can be used with Seawind to generate simple bulk data. Seawind also works in manual mode, in which the user may launch arbitrary, possibly interactive applications for generating the workload (e.g. a web browser and a http server), which communicate through the network emulated by Seawind. Network configuration consists of parameters defining the characteristics of the

emulated network and the *Network Protocol Adapter (NPA)* configuration, which we describe below.

In addition to the CT, Seawind uses a number of other processes to set up the tests. CT controls the creation and the cleanup of the processes in the beginning and in the end of each replication set. The processes may be distributed into multiple hosts to avoid having multiple resource-consuming processes running on the same host. To control the processes there is a *Seawind-daemon* process running on each host in the background. A Seawind-daemon creates the processes needed in a test run according to the requests from CT and terminates them after the test run.

The task of the NPA is to capture network packets from the endpoint host and forward the packets to the SP. For example, a NPA that captures IP traffic creates a dedicated network interface from which it captures packets, and adds a route to the created network interface in the IP routing table. There are various ways for doing this. One alternative is to use *Point-to-Point Protocol (PPP)* [28] that uses a pseudo-terminal device connected to NPA. Secondly, some operating systems support virtual network interfaces that deliver the received packets to and from the user-space applications. Finally, it is possible to capture packets directly from the Ethernet. We have implementations for all above-mentioned variants.

After the NPAs have been started and the simulation pipeline is properly initialized, Seawind starts the *workload generators (WLGs)* at both ends of the simulation pipeline. Workload generator can be any conventional tool that generates network traffic. No modifications need to be made, because the NPAs capture the network traffic transparently to the WLGs. For example, when IP traffic is used, the workload generator can be any tool that generates IP packets using the standard application interface (e.g. Berkeley sockets). The packets that are transmitted to the specified IP address are routed to the network interface connected to NPA and furthermore to SP. At the receiving end the NPA delivers the packet to the IP protocol using the created network interface.

The architecture presented above allows replacing any of the WLG, NPA or even SP components by alternative implementations. It is also possible to read the emulation data from a serial port, which makes it possible to connect an arbitrary machine to the emulation host using a null-modem cable. For example, we have used this facility to connect Windows hosts to Seawind. Furthermore, the receiving end NPA does not have to be attached to the endpoint host, but it can optionally be used to forward packets from SP to the network between the SP and the endpoint host. Thus, any Internet host can be used as an endpoint in the performance tests, making it possible to create a realistic model of communicating to a host in the Internet over the emulated link.

4.2 Pipelining

A mobile network typically includes several logical entities that affect the overall performance and throughput seen by the end user. For example, from the GPRS architecture [8] we can identify three possible emulated components: the base station subsystem (BSS) including the wireless link, the Serving GPRS Support Node which acts as a last-hop router in the GPRS network and the Gateway GPRS Support Node, which is the gateway router in the mobile terminal's home GPRS network.

As a single SP is often used to model a single network element with a link, to model a network path with multiple network elements, Seawind allows connecting several SPs together to form a simulation pipeline. Data packets are forwarded between SPs, and the last SP in the

pipeline sends the packet out to the destination.

Some links use flow control at the link level. This means that the rate at which the packets are transmitted from one network element to another is controlled by the receiving network element. Seawind supports flow control between SPs and between NPAs by using a sliding window based algorithm.

4.3 Channel model

While computer networks become increasingly complex, the principle of having a set of routers (switches, etc.) interconnected by communication links remains the same. In Seawind, a single Simulation Process models an outgoing link, optionally attached to a network node with buffering and a specified queue management policy. Several SPs can be pipelined to represent a path through the network between the client and the server.

The emulated link is modeled as a direction-specific *channel*, which is maintained separately for the uplink (towards the fixed end) and for the downlink (towards the mobile end) traffic. The downlink and uplink channels are largely independent, with an exception of some special events (for example a blackout). The model of the channel is shown in Figure 3. Packets arriving to the emulator are placed into the input queue. The maximum queue length can be limited in terms of bytes or packets. Different packet-drop policies can be applied on the queue (e.g. the traditional tail-drop or RED active queue management [9]). For example, the input queue can be used to model a queue in a router, and thus inspect the effect of congestion and congestion-based losses at a network node.

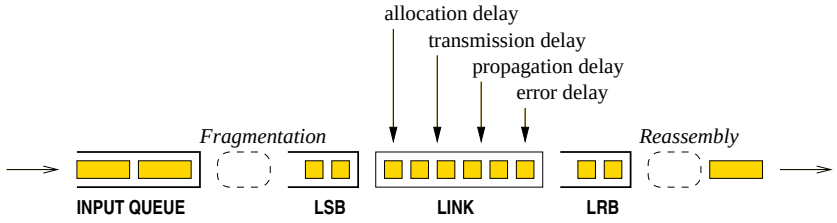


Figure 3: The channel model.

Some link protocols (e.g. Radio Link Control (RLC) [23] in GSM) fragment PDUs of the upper protocol layer as a part of internal operation. The fragmentation unit before the link and the reassembly unit after the link allow to logically fragment the data packet into smaller pieces for the purpose of the different events performed during the emulations. The actual size of the transmitted data transmitted by lower-level protocols can increase due to protocol overhead (e.g. added header) or decrease due to compression. This is also taken into account in the calculations.

The channel model also includes *Link Send Buffer (LSB)* and *Link Receive Buffer (LRB)* to model the send and receive buffers that are present with real links. The link send buffer is used to store the frames to be transmitted to the link, and the link receive buffer is used to store frames at the receiving end until all pieces of a fragmented unit have arrived, allowing reassembly. The link receive buffer is also needed to store out-of-order frames, when a link layer Automatic Repeat Request (ARQ) mechanism is used for retransmitting corrupted or missing frames. The size of these buffers should be large enough allowing the ARQ sliding window protocol to keep the link fully utilized. These buffers may significantly increase the buffering capacity of the link.

Before data can be delivered over the link, the radio resources often have to be allocated first. The delay can be rather high due to possible contention or even queueing for resources. In

the current model the *allocation delay* is triggered when a data unit arrives to an empty queue. Once the resources are allocated, data units 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 the data unit is completed, a *propagation delay* is issued for the unit.

Transmission errors on a link are modeled by specifying a probability that is evaluated on per-packet basis or on per-bit basis. If a data unit is corrupted, the following actions depend on the desired error mode. In the *corrupt* mode the data unit is forwarded in the channel with the corrupted bits. In the *drop* mode the corrupted data unit is dropped by Seawind (i.e. the link layer receiver is assumed to detect the transmission error). In the *delay* model the data unit is delayed for a user-specified amount of time. This can be used to emulate a link ARQ protocol retransmitting the corrupted data unit to recover from transmission errors. In such a case the upper protocol layers experience only an excessive delay for the affected packet.

5 Seawind Features

5.1 Emulation

Protocol filters. A protocol filter is a protocol-specific module that is implemented separately for each protocol (e.g. TCP, WAP) used with Seawind. New protocol filters can be easily added using the interface provided. A protocol filter has two functions: *packet recognizer* recognizes the packet boundaries from the incoming data and populates Seawind structures used in different emulation calculations. Usually the packet recognizer is based on the link layer protocol used for transmitting the Seawind packets (e.g. PPP used over an asynchronous link). Another function of the protocol filter is the *packet printer*, which scans the required information from the protocol headers and stores it to a log file. This information later allows combining the packet trace with the SP event log to determine various events, like the reason for a packet loss and to measure round-trip times. For example, when using TCP/IP protocols the packet outputter uses an output format which is compatible with the `tcpdump` [15] tool to allow interoperability with existing tools used for analysis.

State changes. The user can define multiple sets of parameters (states) that are changed during a test run. For example, the available bandwidth, error properties and delay properties can be changed simultaneously according to the given time interval distribution. This feature can be used to model changes in the mobile communication environment, e.g. due to handoffs. The state is changed synchronously at all SPs used in the emulation. Seawind also provides an interface to trigger state changes from an external program, thus providing a flexible way for creating a wide range of mobility scenarios.

Random distributions. A wide variety of random distributions are included in Seawind to model different kinds of network properties. The list includes the basic distributions (e.g. uniform, exponential, Cauchy) and a two-state Markov distribution. Additionally, any arbitrary distribution can be stored in a file to be used by different parameters during the emulation. Seawind uses its own random number generator to avoid being affected by the biased random number generators that some operating systems may have.

Parallel workload generators. Seawind allows using an arbitrary number of WLGs in automatically run tests and in manually executed tests. Starting the workload generating tools manually is straightforward, as user may launch any number of applications using the command shell, and Seawind transparently captures the data generated by the applications from the network device interface. In automatic operation the user may enter a number of WLG definitions with a starting time relative to the beginning of the test run. This makes it possible to inspect competing traffic flows over the network emulated by Seawind.

Multiple queues. The user data packets may belong to different priority classes. Multiple queues are present in SP to hold user packets of different priority and background load packets (Figure 4). A number of algorithms are given for each queue. The classifier algorithm assigns a specific queue to an arrived packet. The queue management algorithm (e.g. RED [9]) actively marks packets based on the averaged queue length. The drop policy algorithm (e.g. head drop) discards packets that exceed state queue size or length limits. Finally, packets can be marked using *Explicit Congestion Notification (ECN)* [10]. All algorithms have well-defined interfaces so new implementations can be easily added. We currently have a basic implementation of the single-priority traffic, but in the near future we will implement more classifier algorithms for handling multiple queues.

Background load. In addition to the main workload captured by the NPA components, Seawind provides a framework for generating virtual *background load (BGL)*, which affects the internal queueing and delay calculations of SP components. With this feature, the user may create a flexible model of the effects caused by other users in the emulated access network. Figure 4 illustrates the background load framework and how it can be used with multiple Seawind input queues. A *BGL generator* can be attached to any SP in the simulation pipeline to generate packets according to the defined model. Because the BGL packets exist only virtually, the information about BGL packets is forwarded to the next SP using a dedicated BGL channel. The BGL can be consumed by any SP in the pipeline, but it is not forwarded to the NPAs.

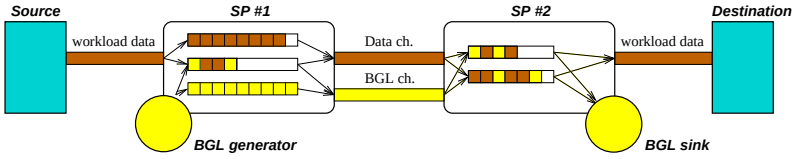


Figure 4: Background load generators with two pipelined SPs with multiple queues.

5.2 Output analysis

The CT collects the log output from various Seawind components and stores them on the disc for further analysis. Two kinds of logs are generated by the Seawind components. *Filter log* is generated by the NPAs and SPs. It contains information about the network packets that have traversed through Seawind. For example, when monitoring IP packets, Seawind uses `tcpdump` for this purpose. The filter log is created by the protocol filter described above and is protocol dependent. *Seawind log* contains Seawind-specific information of the test run. SP stores the information about the events such as delays or losses on each data unit. For each event a timestamp is stored to make it possible to verify that the events have been performed on time. Seawind

log is also collected from NPAs and WLGs. The contents of those Seawind logs depend on the mechanism used in NPA or on the tool used as WLG.

During a test run there are usually a large amount of log information generated. Therefore it is essential to have tools and scripts for analyzing the logs. For a filter log containing tcpdump-compatible information about IP packets this is easy, as there are a wide variety of publicly available tools such as `tcptrace` [22] available.

Currently our scripts collect information about various time measurements, throughput, number of retransmissions, number of packet losses and fairness (using Jain's fairness index [16]). Optionally, the information about different TCP variables in the Linux kernel, such as congestion window size or RTT/RTO estimates, can also be stored to be coupled with the other statistics. We also plan to enhance the graph plotting scripts to show various Seawind or Linux TCP events such as delays, packet losses and retransmission timeouts.

6 Implementation Issues

Developing a real-time emulator for an operating system and network environment that do not guarantee real-time response is not straightforward [12]. An off-the-shelf personal computer and Unix OS are not designed for real-time use, have coarse timer resolution, and are prone to delays caused by the I/O (a disk or network access). Especially in a multi-process environment, keeping a real-time schedule is hard, because processes have to compete for the system resources.

We have not set any absolute real-time requirement for the response times of Seawind events, as it would be impossible to guarantee the required response time in the general case. However, in this section we introduce how we try to ensure that the Seawind response times are accurate enough for performance tests and how we monitor the accuracy of emulation.

Simulation process. We have enhanced the timing accuracy of the events by waking up SP a configurable amount of milliseconds prior the event is due and wait in a busy loop until the actual event time is reached. Before running the performance tests, the user can take a few preliminary runs to adjust the timing estimator for his environment, if the default value is not good for some reason. By distributing the Seawind processes we wanted to make it possible to run the *simulation process* in a lightly loaded host in order to avoid competition of the system resources.

After each event Seawind takes a timestamp from the system clock and stores it to the log. If a threshold value given by the user is exceeded, a warning message is printed so that the user can discard the results for the particular test. However, if the timing estimator is correctly set and there are no other resource-consuming processes running, this occurs very rarely. In our experience the accuracy of Seawind remains within 1 ms with rare exceptions.

During the test runs Seawind avoids unnecessary I/O access, which could cause harmful context switches. It only reads and writes the workload and background load to and from its neighboring processes. The configuration file is read before the test starts and the log is only stored in the memory buffers during the test. After the test is over, the memory buffer is flushed on the disk.

Communication. The inter-process communication between Seawind components need to be performed in a timely manner to ensure correct emulation results. Seawind uses TCP connections between the neighboring components. This is an obvious selection, because the compo-

nents can be distributed into multiple hosts located anywhere in the network, and the connection is required to be reliable. However, certain TCP features, namely the slow start and Nagle's algorithm [29], may cause unwanted delays in the delivery of workload data.

The packet size for the workload data should be selected to be small enough to fit in a single TCP segment in order to avoid the effect of slow start on the transmission rate. Usually this is the case, as the packet size is selected to be small on the emulated slow link, and on the other hand, Seawind is often used over Ethernet using 1500-byte frames. We have disabled Nagle's algorithm from the TCP connections used in the internal Seawind communication in order to allow TCP sender to transmit the TCP segments without delaying them. Finally, it is assumed that the link used to transmit the packets between NPAs and SPs is substantially faster than the emulated link. For example, 100 Mbps Ethernet is sufficient when emulating line rates up to 2 Mbps.

7 Case Study

We now show an example of how to study the behavior of a real TCP implementation by using Seawind. Figure 5 illustrates the target environment we are modeling and how it is emulated using Seawind. In this test case we assume a wireless last-hop link with a bandwidth of 9600 bps and a last-hop router with a buffer size for 7 network packets. The last-hop router is located on the same 10 Mbps LAN with the remote end host. Additionally, there are link buffers for four packets at both ends of the wireless link. Thus, the sending link buffer extends the total buffering capacity to 11 packets. These network properties are close to what GSM data has, for example. We do 20 replications of this test case.

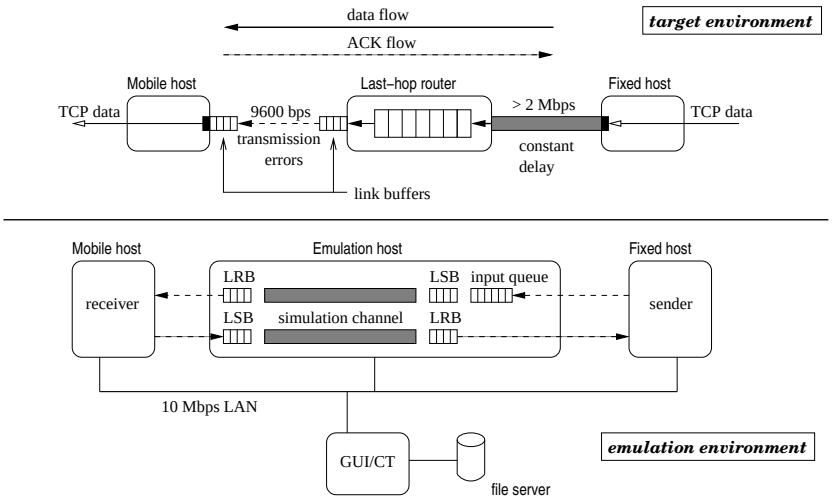


Figure 5: The emulated environment and its setup in Seawind.

In our scenario the wireless link is prone to transmission errors. The transmission errors are assumed to be detected and the corrupted packets are dropped. In our model the packet-drop probability is 1 % for the first 40 seconds of the test run. After 40 seconds the link quality decreases (e.g. the mobile user moves to a location with a weaker radio link quality) and the packet drop probability decreases to 10 %.

In Figure 5 we can see how the emulation is configured to use three hosts. One of the hosts acts as the mobile-end receiver, one of the hosts is the fixed end sender and one host is dedicated to the real-time emulation. Table 1 summarizes the Seawind parameters that were used by the Seawind SP to model the link at the emulation host. The router buffer is modeled with an input queue that drops the packets that do not fit in the queue using the *tail-drop* algorithm. The scenario is modeled with two distinct states in the Seawind state machine, one state for the first 40 seconds and another state for the rest of the test. We have left out from the table the Seawind parameters regarding the features that were not used in this test case. The workload we are using in this test is a bulk transfer of 100 KB using a single TCP connection over the IP protocol.

Table 1: Seawind parameters used in the case study.

Parameter Name	Value
input queue length	7 pkts
queue overflow handling	drop
queue drop policy	tail-drop
link send buffer size	4 pkts
link receive buffer size	4 pkts
transmission rate	9600 bps
propagation delay	200 ms
error handling	drop
packet error probability	state 1: 0.01, state 2: 0.10

Table 2: Summary of measurements.

Metric	Value
Elapsed time, 10th percentile	153.27 s.
Elapsed time, median	170.51 s.
Elapsed time, 90th percentile	196.40 s.
Throughput, median	601 Bps
Rexmitted pkts, median	65
Dropped pkts, median	47

After the 20 replications of the test have been run, Seawind has generated the logs of the test runs. First, we can have a look at the summary of the measurement results, which are shown in Table 2. The shown values are measured from the sending end TCP log. The table shows the median of the selected metrics. Additionally, 10- and 90- percentiles are shown for the elapsed time to illustrate the level of variability in the results. It is also possible to have a separate look at the statistics of each of the 20 replications. As every packet is logged with timestamps, protocol information and related Seawind events, measuring different kinds of metrics and performing different kinds of analysis is only a matter of having suitable scripts for the purpose.

After inspecting the general statistics for the replication set, the user can have a detailed look at what happens at the packet level. One way to do this is to generate a time-sequence diagram of the TCP segments, which is shown in Figure 6. When comparing the time-sequence diagram to the the Seawind event log, we can have an understanding of what happened during the test run.

There are only two corruption losses before the error rate changes after 40 seconds. These two packet losses occur in the beginning of the test and they cause the TCP sender to adjust its *slow start threshold* and enter congestion avoidance, in other words, reduce its sending rate. Thus, the last-hop router buffer load increases moderately, and there are no congestion-related losses until 35 seconds have passed. After the error rate has changed, there are 44 packet losses due to emulated transmission errors. Because of the higher loss rate, the TCP sender keeps transmitting at a low rate and the router buffer queue does not overflow for the rest of the test run.

We used Linux kernel version 2.4.0 at the endpoint hosts. Therefore the phenomena shown in the trace would really occur, if the Linux machine in question is used in the environment similar to what was modeled here.

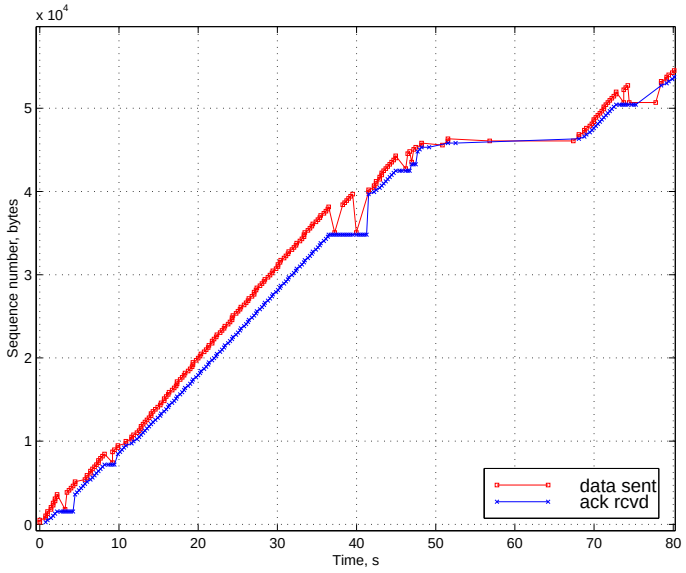


Figure 6: A time-sequence graph of the TCP segments in a test run.

8 Concluding Remarks

This paper presented a wireless network emulator called Seawind. The emulation approach allows performance evaluation of existing implementations of protocols and applications over a wide range of network characteristics. User scenarios that are difficult to reproduce in existing wireless networks or impossible when the network is only in the design phase can be easily presented in the emulator. Distinguishable features of the Seawind network emulator are its wireless-oriented design, portability, easy extendibility and an extensive environment of scripts and tools for the automatic set up of tests and analysis of results. The practical utility of Seawind is demonstrated by a case study and a number of studies beyond this paper. We have experimented with different operating systems and discovered a number of implementation specific features, of which some did not conform to the RFC specifications. We believe that slow links are an environment which have not been considered carefully enough when designing and testing the different implementations of TCP and other protocols. Therefore, we believe that Seawind is a valuable tool for testing the protocol implementations in different networking environments in a controllable fashion.

References

[1] M. Allman, A. Caldwell, and S. Ostermann. ONE: The Ohio Network Emulator. Technical Report TR-19972, School of Electrical Engineering and Computer Science, Ohio University, August 1997.

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

[3] R. Bagrodia and W-L. Liao. Maisie: A language for design of efficient discrete-event simulations. *IEEE Transactions on Software Engineering*, April 1994.

[4] 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.

[5] A. Boukerche, S.K. Das, A. Fabbri, and O. Yildiz. Exploiting model independence for parallel PCS network simulation. In *13th workshop of Parallel and Distributed Simulation 1999*, pages 166–173, May 1999.

[6] 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.

[7] J. Brosh, D. A. Maltz, D. B. Johnson, Y. Hu, and J. Jetcheve. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, pages 85–97, Dallas, Texas, October 1998.

[8] J. Cai and D. J. Goodman. General packet radio service in GSM. *IEEE Communications Magazine*, pages 122–131, October 1997.

[9] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[10] S. Floyd and K. K. Ramakrishnan. A proposal to add explicit congestion notification (ECN) to IP. IETF RFC 2481, January 1999.

[11] NIST Internetworking Technology Group. NIST net network emulation package. *http://www.antd.nist.gov/itg/nistnet/*, June 2000.

[12] A. Gurtov. Technical Issues of Real-Time Network Emulation in Linux. In *Proceedings of FDPW*, June 1999. Available at: http://www.cs.Helsinki.FI/u/gurtov/papers/.

[13] 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/.

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

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

[16] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. John Wiley & Sons, 1991.

[17] 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.

[18] 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.

[19] N. Leavitt. Will WAP deliver the wireless internet. *IEEE Computer*, 33(5):16–20, May 2000.

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

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

[22] S. Ostermann. `tcptrace`. Available at: http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html.

[23] M. Rahnema. Overview of the GSM system and protocol architecture. *IEEE Communications Magazine*, 31(4):92–100, April 1993.

[24] L. Rizzo. Dummynet: A simple approach to the evaluation of network protocols. 27(1):31–41, January 1997.

[25] P. Sarolahti. Performance analysis of TCP improvements for congested reliable wireless links. Master's thesis, Department of Computer Science, University of Helsinki, February 2001. Available at: http://www.cs.Helsinki.FI/group/iwtcp/papers/.

[26] J. Short, R. Bagrodia, and L. Kleinrock. Mobile wireless network system simulation. In *Proceedings of the First Annual International Conference on Mobile Computing and Networking (MOBICOM)*, pages 195–209, Berkeley, CA USA, November 1995.

[27] Simulation Laboratory (SimLab). MobSim++. Web page: http://www.it.kth.se/labs/sim/demoVisTool/mobsimdemo.html, October 1995.

[28] W. Simpson. The point-to-point protocol (PPP). IETF RFC 1661, July 1994.

[29] W. Stevens. *TCP/IP Illustrated, Volume 1; The Protocols*. Addison Wesley, 1995.

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

[31] OPNET Technologies. OPNET Modeler. Commercial, Information at: http://www.mil3.com/products/modeler/home.html, 2001.

[32] B. H. Walke. *Mobile Radio Networks: Networking and Protocols*. John Wiley, first edition, 1999.

[33] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: a library for parallel simulation of large-scale wireless networks. In *Proceedings of the 12th Workshop on Parallel and Distributed Simulations (PADS '98)*, May 1998. Available at: http://pcl.cs.ucla.edu/projects/glomosim/documents.html.

University of Helsinki
Department of Computer Science
Series of Publications C, No. C-2001-53

Making TCP Robust Against Delay Spikes

Andrei Gurtov

Helsinki, November 2001

Report C-2001-53

University of Helsinki
Department of Computer Science
P. O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, FINLAND

Making TCP Robust Against Delay Spikes

Andrei Gurtov

Department of Computer Science, University of Helsinki
Report C-2001-53
November 2001
14 pages

Abstract. This document discusses optimizations for a TCP sender that are most helpful in the presence of delays spikes, but are seemingly suitable for general deployment. The motivation for this work is increasing popularity of links (e.g. provided by cellular networks) that have delay spikes exceeding the usual link latency by several times. The effect of a delay spike on TCP Tahoe, Reno, NewReno and SACK is described. The document recommends timing every segment and restarting the retransmit timer to achieve a more conservative RTO estimate. Furthermore, it discusses how a series of DUPACKs should be treated.

Key Words: TCP, cellular networks, delay spike, spurious timeout

CR Classification: C.2.1

Contents

1. Introduction **1**

2. Sources of delay spikes **2**

3. Delays and interactions with TCP mechanisms **2**

4. Making TCP robust against delay spikes **5**

4.1 Restarting the retransmit timer 5

4.2 Timing every segment 6

4.3 Treating a DUPACK series 8

4.4 Ignoring DUPACKs for oldest outstanding segment after RTO 11

5. Conclusions **12**

References **13**

1. Introduction

The increasing number of users access the Internet via data links provided by cellular Wide Area Wireless Networks (W-WANs). Due to link outages, handovers, and priority blocking delay spikes in order of tens of seconds can occur leading to spurious TCP timeouts and unnecessary retransmissions.

Making TCP robust against delay spikes may include mechanisms on signalling [LU01a], detection [LU01b] and response [LG01] [BA01c] to spurious timeouts. Recommendations made in this documents are expected to be suitable for general deployment even when these mechanisms are implemented.

This document proposes several mechanisms to increase the conservativeness of the TCP retransmission timer. It has a positive effect of making it more tolerable to delays spikes. However, a more conservative RTO timer also has the drawback of a lengthy recovery in case the RTO has not been spurious, i.e. occurred due to segment losses [AP99]. To avoid this performance drawback, non-spurious RTOs should be avoided as much as possible. Thus, all relevant mechanisms that reduce a probability of RTO in presence of packet losses are recommended. Two most important such mechanisms are SACK [RFC2018] [BA01a] and the Limited Transmit algorithm [RFC3042]. The New Reno algorithm [RFC2582] may be used by the TCP sender when the SACK option is not available on the connection. A list of other experimental methods for enhancing loss recovery and avoiding non-spurious RTOs can be found in [RFC3155].

An important observation is that for W-WAN users and operators the battery power consumption and radio resource usage are often as important as the throughput of the link. This suggests that the amount of data sent over the wireless link should be minimized even at the trade-off of extra delay in data delivery. With regard to this document it means that TCP features that avoid unnecessary packet retransmissions have an extra value.

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119]. A segment that acknowledges new data is referred as an ACK. The DupThresh corresponds to the number of DUPACKs necessary to trigger the fast retransmit algorithm, the default value is three DUPACKs.

2. Sources of delay spikes

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 why delay spikes in order of tens of seconds can occur [GU01a].

(1) A long delay spike can be a result of link layer recovery from a link outage due to temporal loss of radio coverage for example while driving into a tunnel or stepping into an elevator.

(2) During a handover the mobile terminal may have to perform some time-consuming actions before data can be transmitted in a new cell. 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.

(3) Blocking by high-priority traffic may occur when an arriving circuit-switch call or higher priority data user temporally preempts the radio channel.

Delay spikes in the Internet can occur for example due to routing changes, but are less frequent than in cellular networks [AP99]. Delay spikes are often coupled with increased likelihood of packets losses in the network. In addition, the network path conditions can change heavily after a delay, for example the available bandwidth can shrink tenfold, after a handover from a fast to a slower cell.

3. Delays and interactions with TCP mechanisms

This section briefly summarizes reaction of a conformant TCP to a delay spike. The Reno description is borrowed from [LK00]. Description of Tahoe, New Reno and SACK is based on experiments in the NS2 simulator [NS] version 2.1b8. The experiment included inserting a delay approximately three times the RTT in the beginning of the TCP connection without data losses. The line rate was 9.6 kbps and the latency 300 ms. Traces are available at [GU01b].

When a sudden delay that exceeds the current value of the TCP retransmission timer occurs in the data transfer, TCP times out and retransmits the oldest outstanding segment, as shown in Figure 1. Since data segments are delayed but not lost, the retransmission is unnecessary and the timeout is spurious. The sender interprets the ACK generated by the receiver in response to a delayed segment 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 a retransmission policy is called go-back-N since the sender forgets about all segments it has earlier transmitted. When retransmitted segments arrive to the receiver they generate DUPACKs since the original segments have already been delivered. When the threshold of three DUPACKs is reached at the sender, 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. From this point, TCP behavior depends on the flavor of the TCP implementation.

TCP receiver sends DUPACKs in response to out-of-order segments. In other words, a DUPACK series appears due to unnecessary retransmissions, if segments have been duplicated by the network, or due to a packet loss. A DUPACK series triggers a fast retransmit when the DupThresh is reached, unless not prevented by the ”bug fix” or SACK information (Section 4.3).

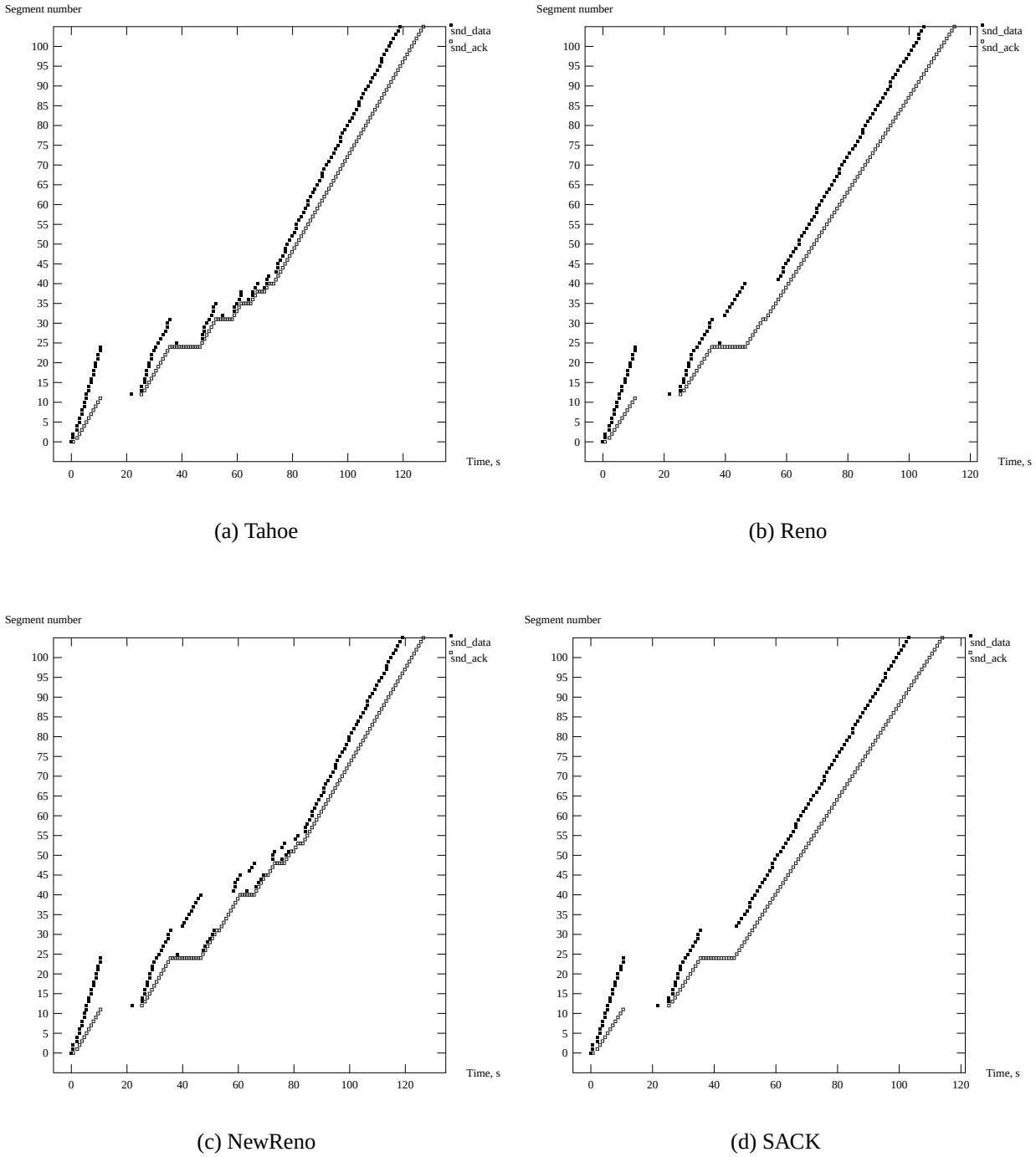
TCP Tahoe in Figure 1(a) ignores arriving DUPACKs after the fast retransmit, as it does not implement fast recovery. However, when partial ACKs start to arrive, Tahoe retransmits outstanding segments unnecessary using the slow start algorithm. This in turn leads to a sequence of DUPACKs causing a fast retransmit and repeating the cycle until the flight size is reduced to the point when the fast retransmit cannot be triggered anymore. After that, the connection proceeds normally slowly increasing the window in congestion avoidance.

TCP Reno in Figure 1(b) enters the fast recovery phase after the false fast retransmit and does not perform any additional unnecessary retransmissions unless the RTO timer expired during fast recovery (Section 4.4)

ACKs arriving after false fast retransmit are partial because they are not confirming the reception of the foremost outstanding segment at that time. The New Reno [RFC2582] algorithm retransmits the presumably missing segment at each new partial acknowledgment in Figure 1(c). A new DUPACK series is triggered by these unnecessary retransmissions, in a similar way as for Tahoe. This continues over and over until too few packets are in flight to trigger a spurious fast retransmit.

Fortunately, preventing the first false fast retransmit after the spurious timeout by the ”bug fix” (Section 4.3) also solves the problem of continues unnecessary retransmits for Tahoe and New Reno.

TCP SACK in Figure 1(d) can avoid the false fast retransmit but cannot avoid the go-back-N behavior. The information on the retransmitted segments during go-back-N comes only in DUPACKs. Using the D-SACK extension of SACK (duplicate-SACK) allows reporting to the sender



(a) Tahoe

(b) Reno

(c) NewReno

(d) SACK

Figure 1: Response to a spurious timeout by different TCPs. Timestamps are enabled and the ”bug fix” is disabled.

the sequence number of a packet that triggered a DUPACK [RFC2883]. D-SACK info comes too late to avoid go-back-N retransmissions, but it can be used to learn about unnecessary retransmissions [BA01b] and adapt the for the future.

In our experience, the impact of delay spikes on real-world TCPs is worse than have been described above. Almost every TCP tested against a delay spike revealed implementation bugs [GU01b]. One goal of this document is to make the TCP developers aware of the negative effect of delay spikes.

4. Making TCP robust against delay spikes

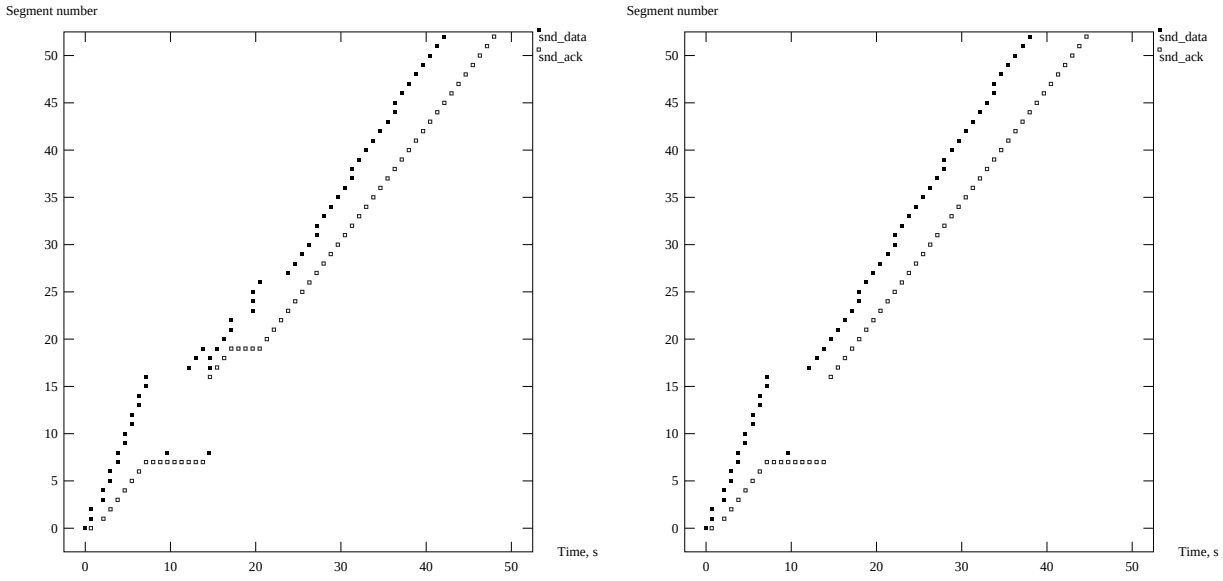
4.1 Restarting the retransmit timer

The obvious way to reduce the number of spurious RTOs in the presence of long sudden delays is to make the RTO timer more conservative than [RFC2988], which recommends restarting RTO only upon an ACK that acknowledges new data. Restarting the RTO timer also when a segment is retransmitted or upon a DUPACK is clearly more conservative approach which is explicitly allowed by [RFC2581].

Restarting the retransmit timer after performing fast retransmit gives a TCP sender more time to wait for the retransmitted segment to be acknowledged. Otherwise, spurious RTO can occur during fast recovery as shown in Figure 2(a), even when no delay spike is present and the RTT samples are collected frequently. Figure 2(b) shows that restarting the retransmit timer when the fast retransmit is triggered prevents the spurious timeout. Restarting the retransmit timer on fast retransmit is a common implementation strategy among existing TCPs. The limited transmit algorithm can increase fast recovery by two DUPACKs, thus raising the likelihood of a spurious RTO, especially if the retransmit timer is not restarted.

Restarting the retransmit timer on DUPACKs is discussed in more detail in Section 4.3. The general argument to consider here is that TCP would not want to timeout while it gets some feedback that segments are being delivered by the network. In addition, DUPACKs could have useful SACK information.

Restarting the retransmit timer on partial ACKs is discussed in [RFC2582]. The Impatient variant of NewReno restarts the retransmit timer only on the first partial ACK, while the Slow-but-Steady variant upon each partial ACK. The Impatient version may timeout during a lengthy fast recovery, and proceed with the go-back-N and slow start. This may result in a quicker recovery when a large number of segments is lost. The Slow-but-Steady version can stay in fast recovery for



(a) Without (b) With

Figure 2: Fast retransmit due to a lost segment with and without reset of the retransmit timer. TCP Reno with timestamps enabled.

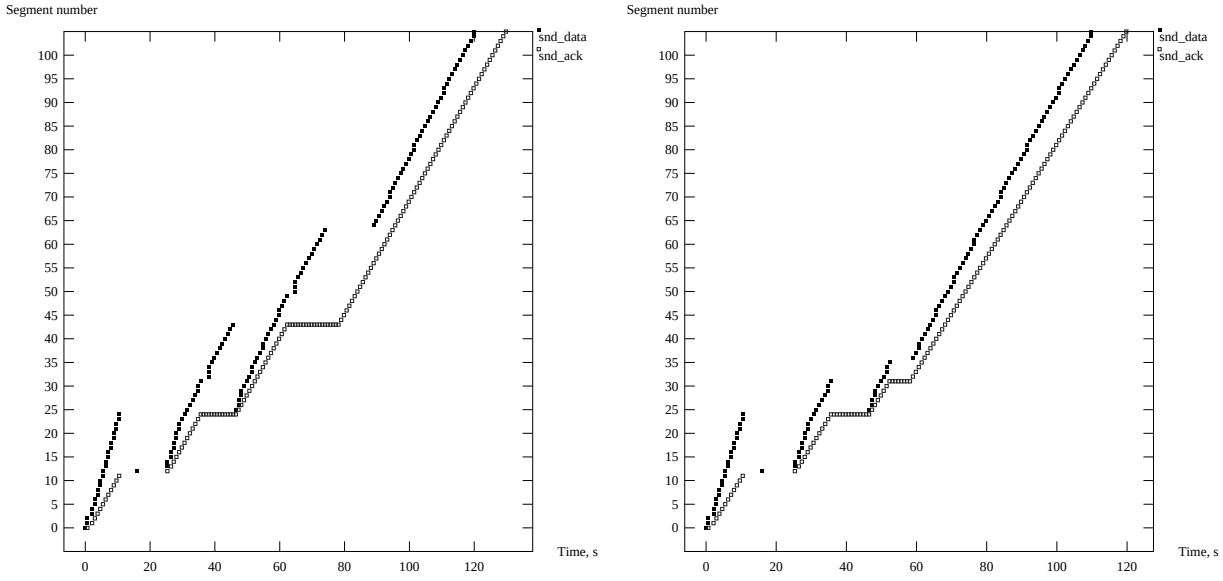
a long time, however avoiding unnecessary retransmissions which are likely during go-back-N. If a delay spike occurs during the fast recovery phase, the Impatient version is more likely experience a spurious timeout. In the presence of delay spikes and when unnecessary retransmissions are costly, the TCP MAY prefer the Slow-but-Steady version, that is restarting the timer on each DUPACK.

Recommendation: The retransmit timer SHOULD be restarted after fast retransmit. TCP MAY restart the retransmit timer on partial ACKs when unnecessary retransmissions are costly and delay spikes are likely.

4.2 Timing every segment

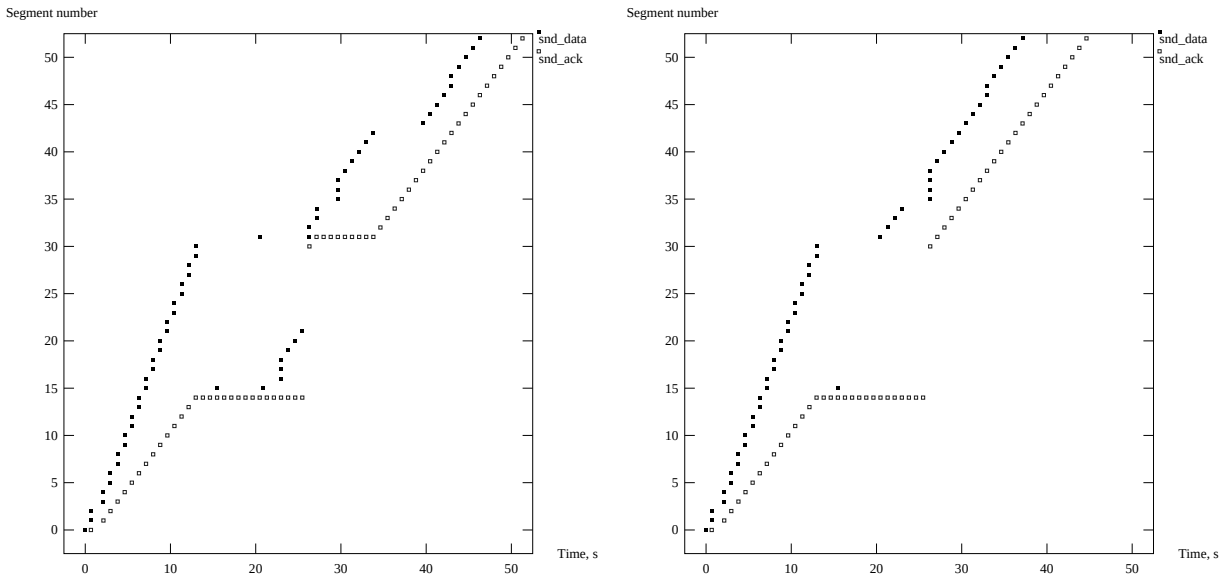
Traditionally, TCPs have been collecting one RTT sample per a window of data [Jac88]. This can lead to underestimating the link RTT and spurious RTOs.

During the slow start phase the queuing delay is increasing rapidly and the RTO value applied just before a new RTT sample is collected may underestimate the current RTT. Increasing the RTTvar coefficient from two to four prevents a spurious timeout during the slow start [Jac88]



(a) Reno (b) SACK

Figure 3: Response to a spurious timeout when timestamps are disabled.



(a) Without (b) With

Figure 4: Fast retransmit in TCP Reno due to a lost segment with and without timestamps.

assuming no delay spikes are present. Still, the standard RTO timer [RFC2988] may exceed the link RTT only by a small edge allowing even a small RTT spike to cause a spurious timeout.

In congestion avoidance, a spurious RTO without a delay spike is still possible with the standard RTO timer, however, only if a very large window is used on a bandwidth-limited link [Jac88]. Unfortunately, overbuffering is seems to be a frequent case for slow links [LU99].

Timing every segment eliminates the effect of lagging RTO behind a rapidly increasing link RTT, thus decreasing the likelihood of a spurious timeout. Timing every segment can be implemented with or without the timestamp option [RFC1323]. Using the timestamp option has the advantage in allowing use of retransmitted segments for RTT measurement which is otherwise blocked by the Karn's algorithm [KP87].

Experiments using NS show that the RTT spike tolerated by TCP without a spurious timeout could be twice higher when every segment is used for RTT estimation [GU01b]. Figure 3(a) and 3(b) illustrate the response to the same delay spike as in Figure 1(b) and 1(d) when the RTT sample is collected only once per window. The retransmit timer is clearly more aggressive without timestamps, as the first retransmission occurs 4 secs earlier. Furthermore, both Reno and SACK experience a second spurious RTO during a DUPACK sequence when timestamps are not used.

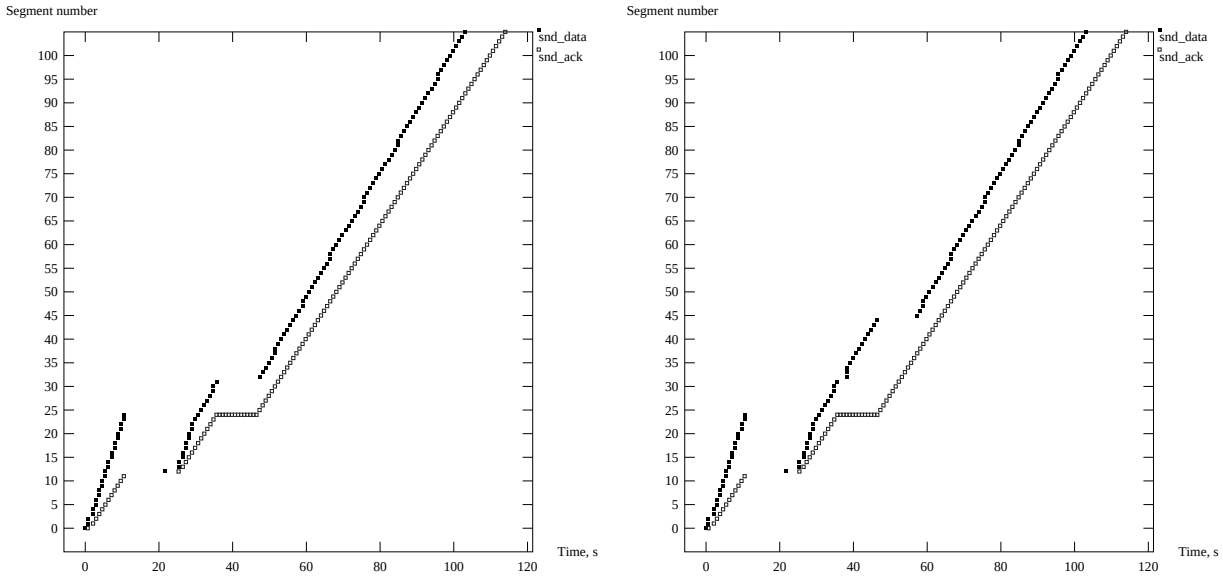
Figure 4(a) shows another example when the spurious RTO occurs during fast recovery. Waiting for 5 secs more would avoid the timeout in this example. Using the timestamp option in Figure 4(b) allows to complete the fast recovery phase without a spurious timeout.

Recommendation: TCP SHOULD collect RTT samples more frequently than once per RTT to decrease the likelihood of a spurious RTO.

4.3 Treating a DUPACK series

Despite of a conservative retransmit timer, a spurious RTO can still occur. The resulting go-back-N behavior produces a large number of DUPACKs triggered by unnecessary retransmissions. The DUPACK series can cause a spurious fast retransmit and a spurious RTO.

Without SACK support on the connection, the receiver has no knowledge whether a DUPACK has been due to a unnecessary retransmission or due to a lost segment. To prevent unnecessary fast retransmits after a RTO, a "bug fix" has been suggested [RFC2582]. The "bug fix" disables fast retransmits until all segments outstanding at the time when RTO occurred are acknowledged. 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. A spurious timeout without



(a) Tahoe

(b) Reno and NewReno

Figure 5: Response to a spurious timeout when the ”bug fix” is enabled.

lost segments presents exactly the situation when DUPACKs arrive for the foremost outstanding segment. The careful version is recommended [RFC2582]. Response to a spurious timeout with the careful ”bug fix” is shown in Figure 5(a) and 5(b).

When SACK is supported by the connection, receiving a DUPACK without a SACK block or with a D-SACK block pointing below the cumulative ACK indicates that the DUPACK was triggered by a unnecessary retransmission (excluding a pathological case when the receiver has agreed to use SACK but does not send SACK info). Thus, there is no reason to enter the loss recovery phase, and the same behavior should be followed as without SACK when the fast retransmit is prevented by the ”bug fix”.

At the moment, the safest way is to follow [RFC2582] and ignore DUPACKs not covering the highest outstanding segment. Ignoring DUPACKs means no segment (re)transmission or changes to congestion control state. However, if a TCP sender simply ignores DUPACKs arriving when fast retransmit is disabled, it can lead to loosing the ACK clock. Whether incoming DUPACKs can be used to trigger transmission of segments is an open problem, and below we provide some facts to be considered. Note, that for DUPACKs before DupThresh, transmission of segments is covered by the limited transmit algorithm. Using the limited transmit algorithm when the ”bug fix”

is enabled shares the considerations below.

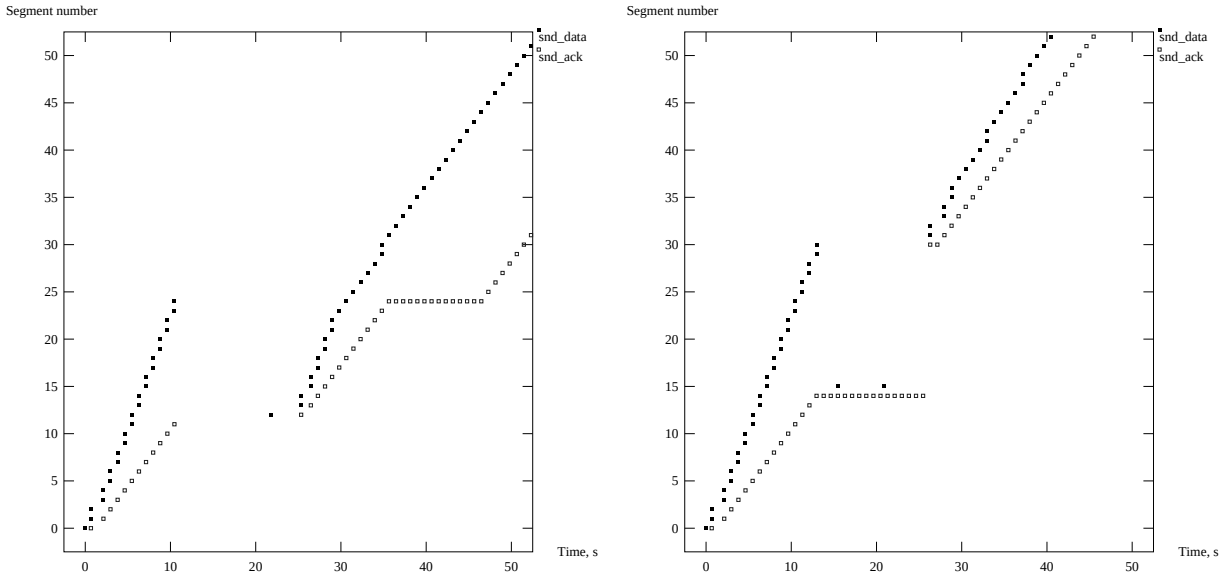
TCP Reno in NS2 uses DUPACKs to rigger transmission of segments during 37-50 secs as shown in Figure 5(b). When an ACK arrives at 50 sec, the congestion window is deflated that produces a pause in packet transmission during 50-58 sec. The alternative behavior when the congestion window is not inflated is shown in Figure 5(a). Correspondingly, segments are not transmitted upon DUPACKs and there is no pause due to deflating of the congestion window when an ACK arrives at 50 sec. There is no clear benefit in either approach, as illustrated by the equal download time for both connections. A possible modification is shown in Figure 6(a) when DUPACKs trigger transmission of segments and the congestion window is not deflated. However, this algorithm produces unstable behavior when evaluated in environment with congestion losses and therefore cannot be recommended. In opposite, transmitting a new segment every *second* DUPACK (similar to the Rate-Halving algorithm) reduces the transmission rate, but still preserves the ACK clock. Experiments show that this approach is stable in presence of congestion losses and improves the throughput.

If segments are transmitted on DUPACKs for the segment below the highest outstanding segment, they are retransmissions. If these retransmissions happen to be unnecessary, a new DUPACK series is created later. On the other hand, segments transmitted upon DUPACKs for the highest outstanding segment are new transmissions. Thus, one option would be to allow transmitting segments on DUPACKs only for the highest outstanding segment.

The retransmit timer may be restarted safely upon DUPACKs if no segments are transmitted after DupThresh. Restarting the timer decreases likelihood of spurious RTOs during a DUPACK series when delay spikes occur. However, timing every segment and restarting the timer on reaching DupThresh seem to provide a conservative enough retransmit timer in many cases. Restarting the retransmit timer on DUPACKs can lead to a lengthy recovery when the segment was lost.

The reason for banning transmission of a segment AND restarting the retransmit timer on DUPACKs is a situation when the last outstanding segment is lost. The receiver will keep sending DUPACKs until the lost segment is received. A newly transmitted segment on a DUPACK will trigger another DUPACK in the future creating an endless loop. The retransmit timer in this case serves as an back-up way to interrupt the loop. With SACK, this problem can appear only in a pathological case when the receiver does not report losses.

Recommendation: TCP without SACK SHOULD implement the careful version of the ”bug fix”. TCP with SACK SHOULD NOT enter the loss recovery phase when DUPACKs do not have a SACK block indicating a lost segment. TCP MAY restart the retransmit timer upon receiving a DUPACK. Transmitting segments upon DUPACKs above the oldest outstanding segment is an



(a) DUPACKs increase the congestion window

(b) DUPACKs for the oldest outstanding segment are ignored after RTO (timestamps are disabled)

Figure 6: Effect of DUPACKs when the ”bug fix” is enabled.

open issue. However, TCP MUST NOT transmit a new segment and restart the retransmit timer for DUPACKs above the DupThresh.

4.4 Ignoring DUPACKs for oldest outstanding segment after RTO

TCP can time out during a series of DUPACKs, either during fast recovery phase as shown in Figure 2(a) and 4(a), or after a large number of unnecessary retransmissions as shown in Figure 4(a). If DUPACKs are delayed or lost, RTO can occur despite of restarting the retransmit timer upon DUPACKs. RTO in this situation may or may not be spurious; the recommendation in this section applies in both cases.

The proper behavior after RTO is specified in [RFC2581], that is to retransmit the oldest outstanding segment, wait for an ACK and back-off the RTO timer if it expires again. However, a fairly common behavior among TCPs is to use DUPACKs arriving after RTO to inflate the congestion window and clock out retransmissions of segments, as happens in Figure 4(a) at 25 secs. This behavior is observed at least in current versions of FreeBSD, Windows, and NS2 TCPs. This behavior can be a special case of the problem as unnecessary fast retransmits which is discussed in

Table 1: Summary of recommendations.

Mechanism	Use	SACK	Section
Restarting the retransmit timer			
after DUPACK for DupThresh	SHOULD	both	4.1
after partial ACK	MAY	both	4.1
after DUPACK above DupThresh	MAY	both	4.3
Timing every segment	SHOULD	both	4.2
Careful ”bug fix”	SHOULD	w/o	4.3
No recovery on DUPACKs without loss info	SHOULD	with	4.3
New segment on every second DUPACK (without restarting the retransmit timer)	MAY	both	4.3
Ignoring DUPACKs after RTO	SHOULD	both	4.4

Section 4.3. In this situation the fast retransmit does not make sense, since only a single segment retransmitted after RTO is assumed to be outstanding, and cannot cause enough DUPACKs to trigger the fast retransmit. Furthermore, since RTO is taken as an indication of severe congestion, it is unwise to retransmit segments on DUPACKs after RTO without getting any feedback for the first retransmission.

It has been observed with real TCPs that transmitting segments on DUPACKs after RTO can lead to a series of spurious timeouts as follows. TCP times out during a long DUPACK series caused by go-back-N retransmissions. After RTO, DUPACKs are triggering unnecessary retransmission of segments; resulting DUPACKs in the future cause RTO again. When DUPACKs are ignored after RTO as shown in Figure 6(b), a spurious RTO during a DUPACK series can only lead to unnecessary reduction of the congestion window and slow start threshold, but does not produce a series of spurious RTOs.

Recommendation: TCP SHOULD ignore DUPACKs for the oldest outstanding segment after RTO.

5. Conclusions

A TCP connection can experience delay spikes due to various reasons like handovers, priority blocking, temporal link outages or route changes. We described the response of Tahoe, Reno, New Reno and SACK TCP to a spurious timeout resulting from a delay spike. We have studied the behavior of the retransmit timer and ways to treat a DUPACK series using the NS simulator. The resulting summary of recommendations is shown in Table 1.

Acknowledgements

Many thanks to Reiner Ludwig, Mark Allman, and Sally Floyd for discussions on the contents of this document and to Timo Alanko for supporting this work.

References

[AP99] M. Allman and V. Paxson, On Estimating End-to-End Network Path Properties, ACM SIGCOMM '99, September 1999, Cambridge, MA.

[BA01a] E. Blanton, M. Allman. A Conservative SACK-based Loss Recovery Algorithm for TCP. Internet-Draft draft-allman-tcp-sack-07.txt, July 2001, work in progress.

[BA01b] E. Blanton, M. Allman. Using TCP DSACKs and SCTP Duplicate TSNs to Detect Spurious Retransmissions, August 2001, work in progress.

[BA01c] E. Blanton, M. Allman, Adjusting the Duplicate ACK Threshold to Avoid Spurious Retransmits, work in progress, July 2001.

[Jac88] V. Jacobson, "Congestion Avoidance and Control", In proceedings of ACM SIGCOMM'88, 1988.

[GU01a] A. Gurtov, Effect of Delays on TCP Performance, In Proceedings of IFIP Personal Wireless Communications, August 2001.

[GU01b] A. Gurtov, Traces of TCP connections experiencing a delay spike, http://www.cs.helsinki.fi/u/gurtov/tcp/, November 2001.

[RFC1323] V. Jacobson, R. Braden, D. Borman, TCP Extensions for High Performance, RFC 1323, May 1992.

[RFC2018] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, TCP Selective Acknowledgement Options, RFC 2018, October 1996.

[RFC2119] S. Bradner, Key words for use in RFCs to Indicate Requirement Levels, RFC 2119, March 1997.

[RFC2026] S. Bradner. The Internet Standards Process – Revision 3, RFC 2026, October 1996

[RFC2581] M. Allman, V. Paxson, W. Stevens, TCP Congestion Control, RFC 2581, April 1999.

[RFC2582] S. Floyd and T. Henderson. The NewReno modification to TCP's fast recovery algorithm. IETF RFC 2582, April 1999.

[RFC2883] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky, A. Romanow, An Extension to the Selective Acknowledgement (SACK) Option for TCP, RFC 2883, July 2000.

[RFC2988] V. Paxson, M. Allman, Computing TCP's Retransmission Timer, RFC 2988, November 2000.

[RFC3042] Allman, M., Balakrishnan, H. and S. Floyd, Enhancing TCP's Loss Recovery Using Limited Transmit, RFC 3042, January, 2001.

[RFC3155] S.Dawkins, G. Montenegro, M. Kojo, V. Magret, N. Vaidya. End-to-end Performance Implications of Links with Errors, RFC3155, August 2001.

[KP87] P. Karn, C. Partridge, Improving Round-Trip Time Estimates in Reliable Transport Protocols, In Proceedings of ACM SIGCOMM 87.

[LK00] R. Ludwig, R. H. Katz, The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions, ACM Computer Communication Review, Vol. 30, No. 1, January 2000.

[LU99] 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, May 1999.

[LG01] R. Ludwig, A. Gurtov, Responding to Spurious Timeouts in TCP, work in progress, November 2001.

[LU01a] R. Ludwig, TCP Retransmit (RXT) Flag, work in progress, November 2001.

[LU01b] R. Ludwig, The Eifel Algorithm for TCP, work in progress, November 2001.

[NS] ISI, Network Simulator 2, http://www.isi.edu/nsnam/ns

Evaluating the Eifel Algorithm for TCP in a GPRS Network

Andrei Gurtov
University of Helsinki – Finland
e-mail: Andrei.Gurtov@cs.Helsinki.FI

Reiner Ludwig
Ericsson Research – Germany
e-mail: Reiner.Ludwig@Ericsson.com

ABSTRACT
Large and sudden variations in packet transmission delays are often unavoidable in GPRS. This may cause spurious timeouts in TCP. Spurious timeouts affect TCP performance in two ways: (1) the TCP sender unnecessarily reduces its load, and (2) the TCP sender is forced into a go-back-N retransmission mode. The Eifel algorithm avoids these consequences. We evaluate the performance of the Eifel algorithm for TCP Reno, NewReno and SACK in a simulated GPRS network. We use throughput and goodput as equally important performance metrics. In all our simulations, we find that the Eifel algorithm improves goodput; in some cases by up to 20 percent. When complemented with an efficient loss recovery scheme (SACK or NewReno), we find that the Eifel algorithm also improves bulk data download times in all our simulations; in some cases by up to 12 percent.

1. INTRODUCTION

An increasing number of mobile users access the Internet via data links provided by cellular wide area wireless networks such as the General Packet Radio Service (GPRS) [1]. GPRS is a packet-switched extension of the Global System for Mobile communications (GSM). While it is being actively deployed globally at the time of writing, GPRS is already operational in many countries. GPRS incorporates many physical and link layer techniques including different forward error correction schemes, Automatic Repeat reQuest (ARQ), power control, and frequency hopping that typically ensure a "smooth" data transmission. Nevertheless, large and sudden variations in packet transmission delays are often unavoidable. This often creates a problem for end-to-end protocols. In particular, the Transmission Control Protocol (TCP) [11] is not sufficiently robust to cope with such delay variations.

Our previous work discusses possible sources of delay spikes in the GPRS network [5]. Possible events that may cause suspension of a TCP connection on the order of seconds are link outages, handovers and radio resource preemption. Link outages can result from a transient loss of radio coverage, e.g., while driving through a tunnel or when using an elevator. During a handover the mobile terminal may have to perform time-consuming operations before data can be transmitted in the new cell. Blocking by high-priority traffic may occur when an arriving voice call or higher priority data user temporally preempts the radio channel.

Such events do not necessarily cause packet losses since GPRS implements a rather persistent link layer retransmission scheme. However, the sudden delay spikes can cause TCP to timeout prematurely, and perform unnecessary retransmissions. This paper presents a quantitative evaluation of the Eifel algorithm for TCP [4] within the context of GPRS. The Eifel algorithm is a mechanism to detect and respond to spurious timeouts and spurious fast retransmits in TCP. We simulate bulk data connections of TCP Reno, NewReno [8] and SACK [2] while generating delay spikes that are typical for those caused by cell reselections in a GPRS network. Although the study could have been done in a live GPRS network, it would have been difficult to reproduce exactly the same sequences of cell reselections, and thus difficult to estimate the effect of Eifel. Additionally, it is difficult to find a flawless TCP implementation. We therefore used the NS2 [3] simulator. For that, we implemented a module that simulates a GPRS link and is able to replay traces of delay variations based on measurements taken in a live GPRS network.

An important observation is that for mobile users and operators the battery power consumption and radio resource usage are often as important as the download time over the wireless link. This suggests that the amount of data sent over the wireless link should be minimized. Thus, in our evaluating of the Eifel algorithm in GPRS, we use the download time and the goodput, i.e., the ratio of useful over total data transmitted, as equally important performance metrics.

The rest of the paper is organized as follows. In Section 2, we describe the cell reselection mechanism in GPRS. In Section 3, we explain the effect of delay spikes on TCP, and how the Eifel algorithm makes the TCP sender robust against the potentially resulting spurious timeouts. In Section 4, we describe the methodology and assumptions underlying our analysis. Our results are presented in Section 5. Section 6 concludes the paper and outlines our plans for future work.

2. CELL RESELECTION IN GPRS

In GPRS, the mobile terminal selects the serving cell. This is different from the basic circuit-switched GSM data service where the network controls the transfer of on-going data calls between cells [6][7].

The cell reselection process causes a delay, and sometimes packet losses in active data flows. The total The total delay consists of the radio channel access delay in the Base Station Subsystem (BSS), and the delay caused by mobility management procedures in the core network; more precisely the Serving GPRS Support Node (SGSN).

Changing between cells that belong to the same base station controller can be typically done within the BSS without involving the SGSN, which reduces the delay. Some events in the network may cause cell reselection to be aborted and later restarted which significantly increases the delay. According to the GPRS specifications, a cell reselection should be completed within a few seconds. However, in a live GPRS network we observed that it can take any time from a few to a few tens of seconds.

The frequency of cell reselections is to a large extent determined by the speed of a user's movement and the size of cells. For example, driving in an urban area may cause frequent cell reselections. A typical interval between cell reselections in such a case is around a minute, but can be as small as few tens of seconds in densely populated environment.

To show that a delay spike caused by a cell reselection can indeed trigger a spurious timeout in TCP, we have performed a simple test. We took a laptop running Linux (RedHat version 6.2) connected via a Motorola Timeport GPRS phone to a live GPRS network. By forcing cell reselections from the phone and recording the TCP behavior using `tcpdump` [12], we have obtained the TCP trace plot shown in Figure 1. More details on reading TCP trace plots can be found in [4]. The first cell reselection occurs at 510 s when the TCP connection is in the slow start phase and takes 7 seconds. The second one occurs at 550 s during the congestion avoidance phase and takes 8 seconds. In both cases, TCP experiences a spurious timeout and performs unnecessary retransmissions.

3. THE EIFEL ALGORITHM

The Eifel algorithm proposed in [4] makes the TCP sender robust against spurious timeouts and packet reordering. In this section, we only explain the Eifel algorithm in the context of spurious timeouts. When a delay spike exceeds the current value of TCP's retransmission timer, a timeout occurs, and the TCP sender retransmits the oldest outstanding segment. If that segment or the corresponding ACK is only delayed but not lost, that retransmission was unnecessary and the timeout is said to be spurious. Figure 2 shows a spurious timeout for Reno TCP produced using the NS2 simulator. The receiver trace is offset by 25 segments to prevent an overlap with the sender trace. We enhanced the `hiccup` tool [9] to generate the delays in this test. The first retransmission is sent at second 6, and is also delayed. The sender interprets the ACK generated by the receiver in response to the original segment as corresponding to the retransmission. This happens because TCP ACKs bear no information that would allow the TCP sender to distinguish an ACK for the original segment from that for the retransmission. Likewise the TCP sender misinterprets the following original ACKs, and retransmits all outstanding

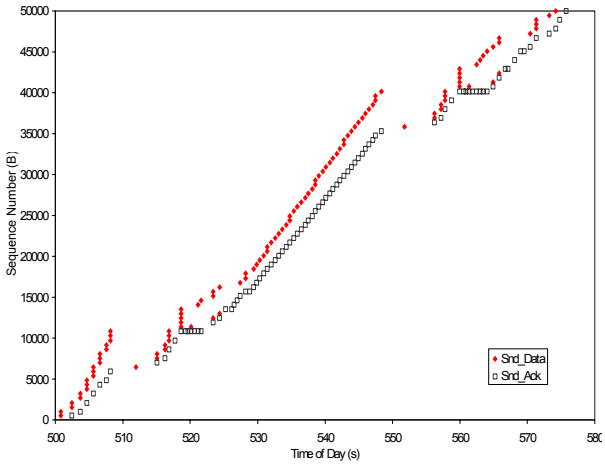


Figure 1. A TCP trace shows spurious retransmissions caused by two cell reselections in a live GPRS network.

segments using the slow start algorithm. Also, a number of new segments allowed by the congestion window are transmitted in this phase.

At second 8 the retransmitted segments arrive at the TCP receiver and generate DUPACKs [13]. When the threshold of three DUPACKs is reached at the sender, a spurious fast retransmit is triggered since the TCP sender does not implement the careful version of the fast retransmit algorithm [8].

Figure 3 illustrates the operation of the Eifel algorithm in the event of a spurious timeout. The Eifel algorithm stores the timestamp of the first retransmission occuring at second 6. The first ACK that acknowledges the retransmission at second 8 carries a timestamp of 3 s which is when the original transmission of the corresponding segment took place. By comparison with the timestamp stored for the retransmission (6 s) the Eifel algorithm detects that the timeout was spurious.

The response to a spurious timeout in the original study [4] resumes transmission with the next unsent segment. How the congestion control state is reversed depends on the number of subsequent spurious timeouts. After the first timeout, the sender restores the slow start threshold and the congestion window to the values before the timeout. Figure 3 shows this situation when a delay spike occurs in the beginning of the connection in the slow start phase. After detecting the spurious timeout at second 8, the slow start phase continues. The behavior after two subsequent timeouts is shown later in Figure 7. In this case, the slow start threshold is set to the previous value of congestion window, which itself is left halved. In that case, a TCP sender ignores some of the original ACKs after a spurious timeout until the congestion window has sufficiently increased. After three and more subsequent spurious timeouts the congestion control state is not reversed at all.

In this paper, we evaluate the Eifel algorithm as proposed in [4] leaving the study of various enchancments to the response part for future work.

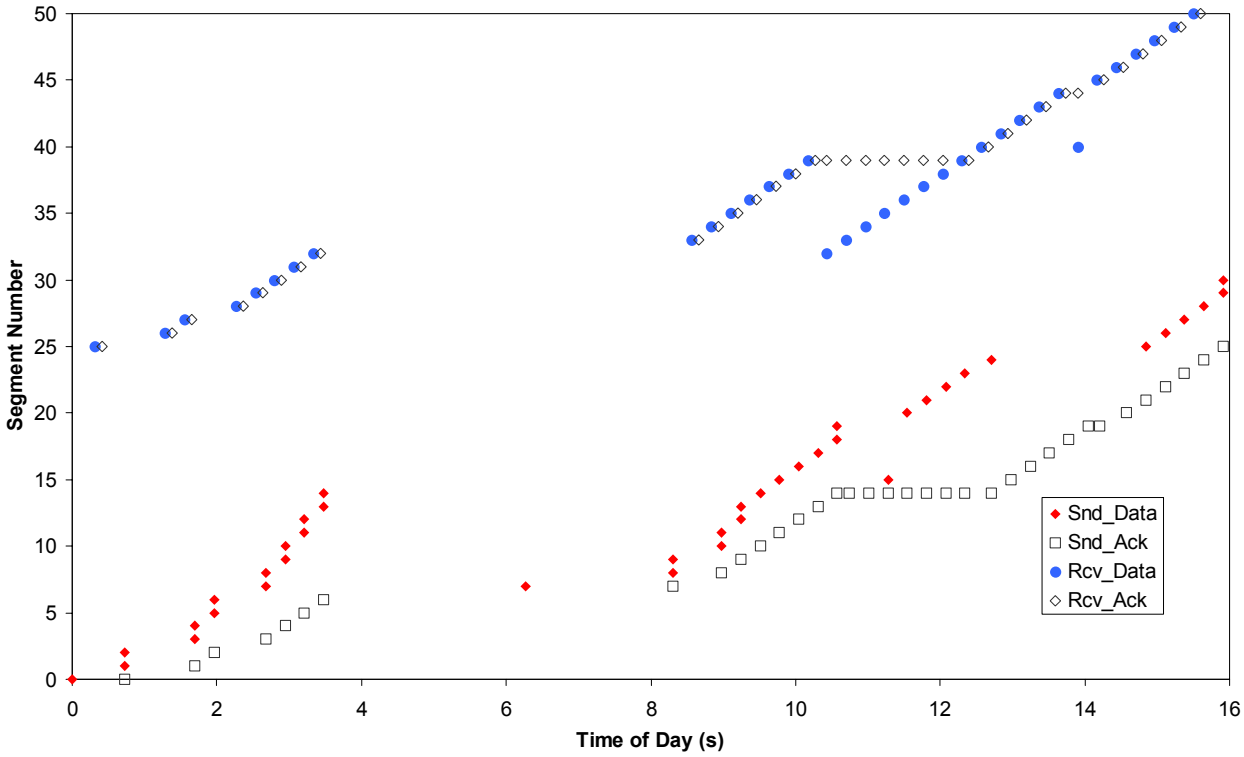


Figure 2. Spurious timeout of TCP Reno due to a 5 s delay spike.

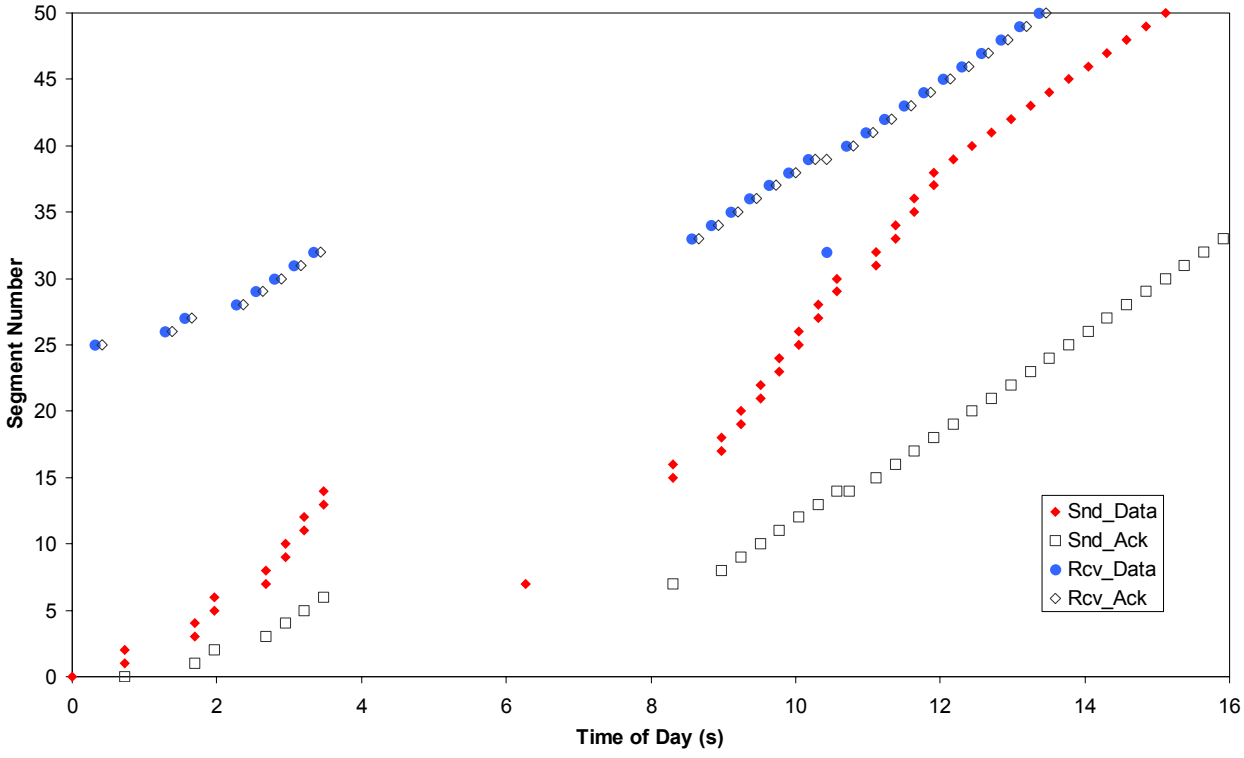


Figure 3. Spurious timeout of TCP Reno with Eifel due to a 5 s delay spike.

4. SETUP OF EXPERIMENTS

We evaluate the Eifel algorithm in three scenarios: easy, mediocre, and difficult. For each scenario, we assume different intervals between cell reselections where the intervals are drawn from a uniform distribution between a minimum and a maximum interval as shown in Table 1. The interval times have been chosen in a way that for a typical download time of 120 s on average one, two and three delay spikes occur per connection for the easy, mediocre and difficult scenario, respectively. We further assume that the time to complete a cell reselection is uniformly distributed between 3 and 15 seconds.

Table 1. Interval between Cell Reselections for three Scenarios.

	Min (s)	Max (s)
Easy	80	140
Mediocre	40	80
Difficult	20	40

For our experiments, we used the one-way models of Reno, NewReno and SACK TCP with delayed acknowledgments in NS2, and a slightly adapted version of the Eifel implementation from [9]. The test configuration is shown in Figure 4. It contains two nodes and a link with a drop-tail queue. The full-duplex link has a rate of 30 kbps and a one-way latency of 300 ms. The `hiccup` tool generates delays on the link. We have improved `hiccup` to suspend data flow in *both* directions and *after* the queue.

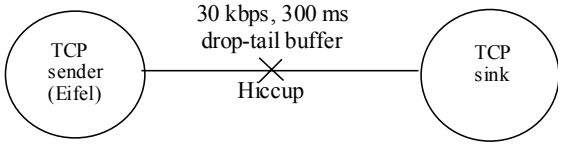


Figure 4. Test configuration in NS2.

A single test is based on a TCP connection transferring 300 KB of bulk data. Each test has been repeated a hundred times to ensure sound statistics. We experimented with a bottleneck buffer size of 10 KB that causes congestion losses, and a size of 100 KB that is large enough to fit the receiver window of data, and thus avoids any losses. We used the default settings for all parameters in the simulator, except for disabling the "bug fix" [8] and enabling the TCP Timestamps option [15].

5. RESULTS OF EXPERIMENTS

Figure 5 and Table 2 depict the average download times for the various bulk data transfers, while Figure 6 and Table 3 show the goodput results. As goodput we defined the ratio of the minimum number of segment required for completing the data transfer to the actual number of segments transmitted. Results are based on TCP Reno (R), NewReno (N), and SACK (S) with and without the Eifel algorithm. As expected, the download time increases and goodput decreases from the easy to the difficult scenario with growing frequency of delay spikes.

In an environment without congestion losses (100 KB buffer), Eifel reduces the download time and the number of unnecessarily retransmitted segments in all scenarios and for all three TCP flavors. Interestingly, NewReno without Eifel suffers from a large number of unnecessary retransmissions and increased download time compared to TCP Reno and SACK [5]. The goodput for all three TCP flavors with Eifel is close to 100 percent in all scenarios. This is a significant improvement compared to 87 percent of Reno and SACK or 80 percent of NewReno in the difficult scenario. The download time reduction ranges from 4 percent in the easy scenario to 12 percent in the difficult scenario.

In case of the small bottleneck buffer size (10 KB), Eifel improves the goodput for all three TCP flavors in all scenarios, and reduces the download time of NewReno and SACK. The goodput of NewReno and SACK with Eifel is close to 100 percent in all scenarios, compared to about 90 percent without Eifel in the difficult scenario. Eifel reduces the download time for NewReno and SACK by up to 8 percent. Although Reno with Eifel shows an improvement in goodput of several percent compared to pure Reno, the download time is notably increased in all scenarios. To explain this unexpected result we have closely examined packet traces of such connections. Apparently, Reno with Eifel suffers from lengthy non-spurious timeouts caused by packet losses.

Figure 7 shows an example of the poor performance of Reno with Eifel when packet losses occur. The first timeout at second 30 is caused by a delay and is spurious. The Eifel algorithm successfully detects the spurious timeout, and resumes transmission with the next unsent segment at second 36. In this case, some of the original segments were lost due to a buffer overflow. DUPACKs for the first lost segment start to arrive at second 37 *below* the highest outstanding segment. The "bug fix" [8] is disabled, and thus the fast retransmit is triggered at second 38. However, due to multiple losses the sender experiences a second timeout at second 61. That timeout is not spurious. However, the RTO at that time is huge, as it is calculated from the timestamps in the delayed segments. Additionally, the RTO may still be backed-off after the first timeout (It is not quite clear what the requirement level is in [10] for resetting the back-off counter once a new RTT sample is collected. We have instrumented TCP to reset the counter). When the retransmit timer finally expires at second 61 lost segments are recovered and normal transmission resumes. Without Eifel, Reno often avoids the second non-spurious timeout as it retransmits all outstanding segments in go-back-N, including the lost ones. Disabling the "bug fix" (as we have done in our tests) helps to recover without a non-spurious timeout when one, and sometimes two, segments are lost. However, in order to provide good TCP performance in environments with delay spikes and high loss rate, the Eifel algorithm should be coupled with efficient loss recovery schemes like SACK and Limited Transmit [14].

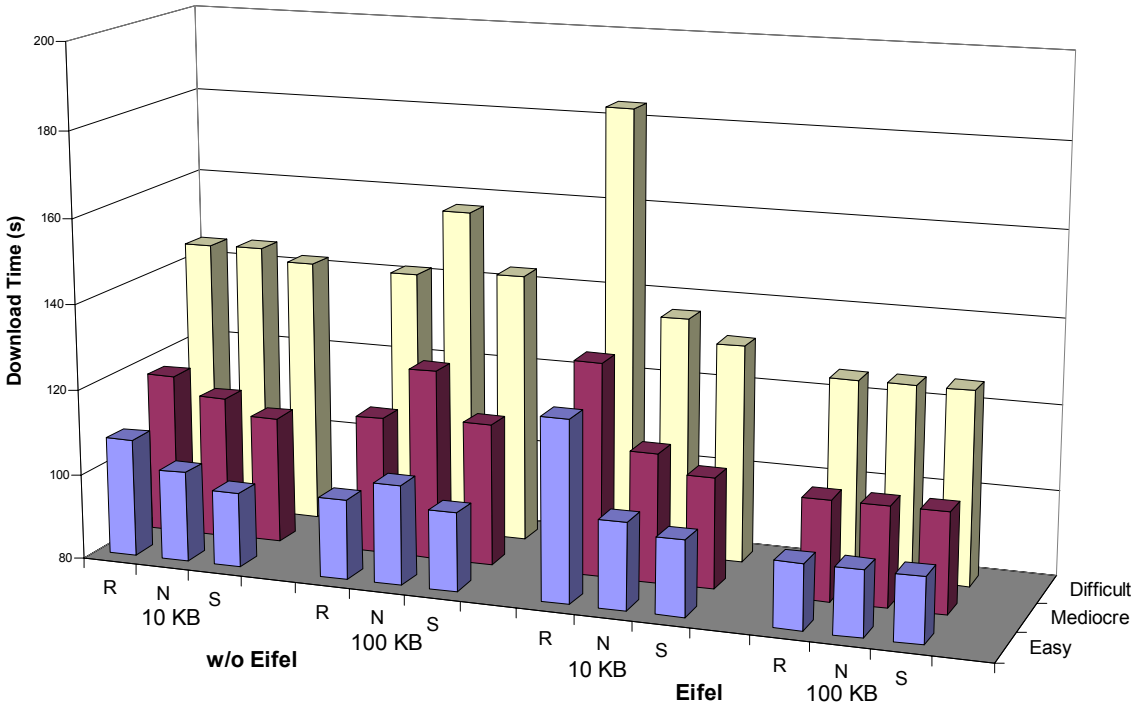


Figure 5. Download time of Reno (R), NewReno (N) and SACK (S).

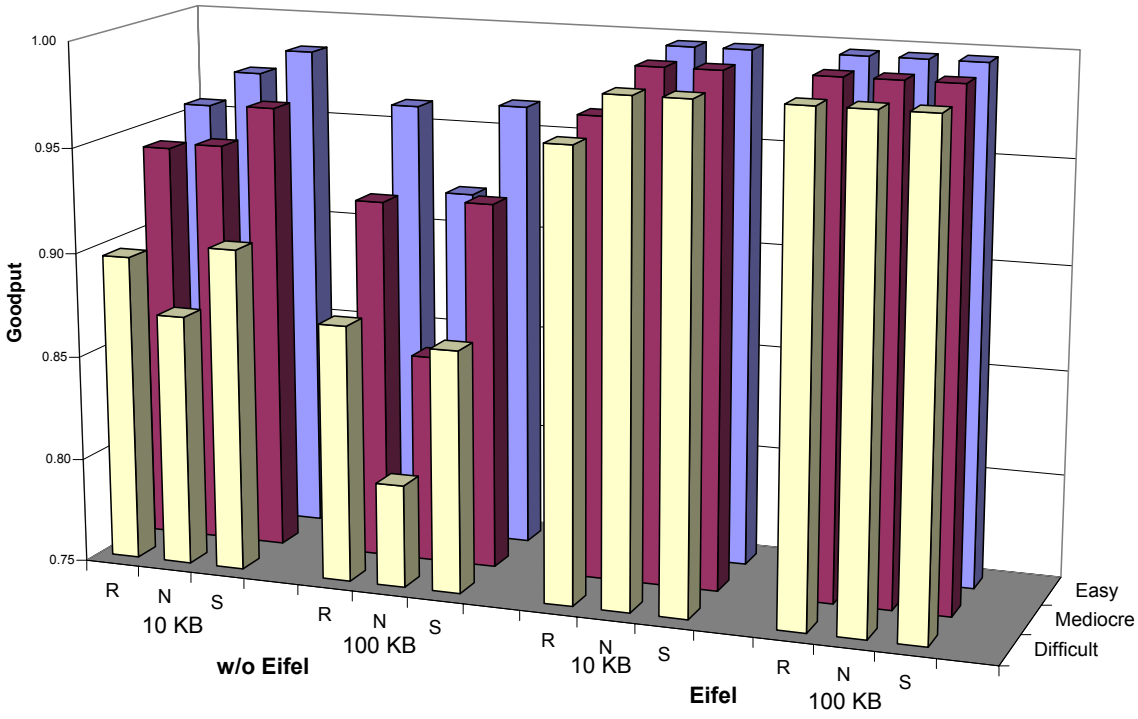


Figure 6. Goodput of Reno (R), NewReno (N) and SACK (S).

Table 2. Download time (s) of Reno (R), NewReno (N) and SACK (S).

Eifel	NO						YES					
Buffer	10KB			100KB			10KB			100KB		
TCP	R	N	S	R	N	S	R	N	S	R	N	S
Easy	108	101	98	99	103	98	122	100	98	95	95	95
Mediocre	118	113	110	112	125	113	130	110	106	103	103	103
Difficult	145	145	142	142	157	143	184	136	131	125	125	125

Table 3. Goodput of Reno (R), NewReno (N) and SACK (S).

Eifel	NO						YES					
Buffer	10KB			100KB			10KB			100KB		
TCP	R	N	S	R	N	S	R	N	S	R	N	S
Easy	0.96	0.97	0.98	0.96	0.92	0.96	0.97	1.00	1.00	1.00	1.00	1.00
Mediocre	0.94	0.94	0.96	0.92	0.85	0.93	0.97	0.99	0.99	0.99	0.99	0.99
Difficult	0.90	0.87	0.90	0.87	0.80	0.87	0.96	0.99	0.99	0.99	0.99	0.99

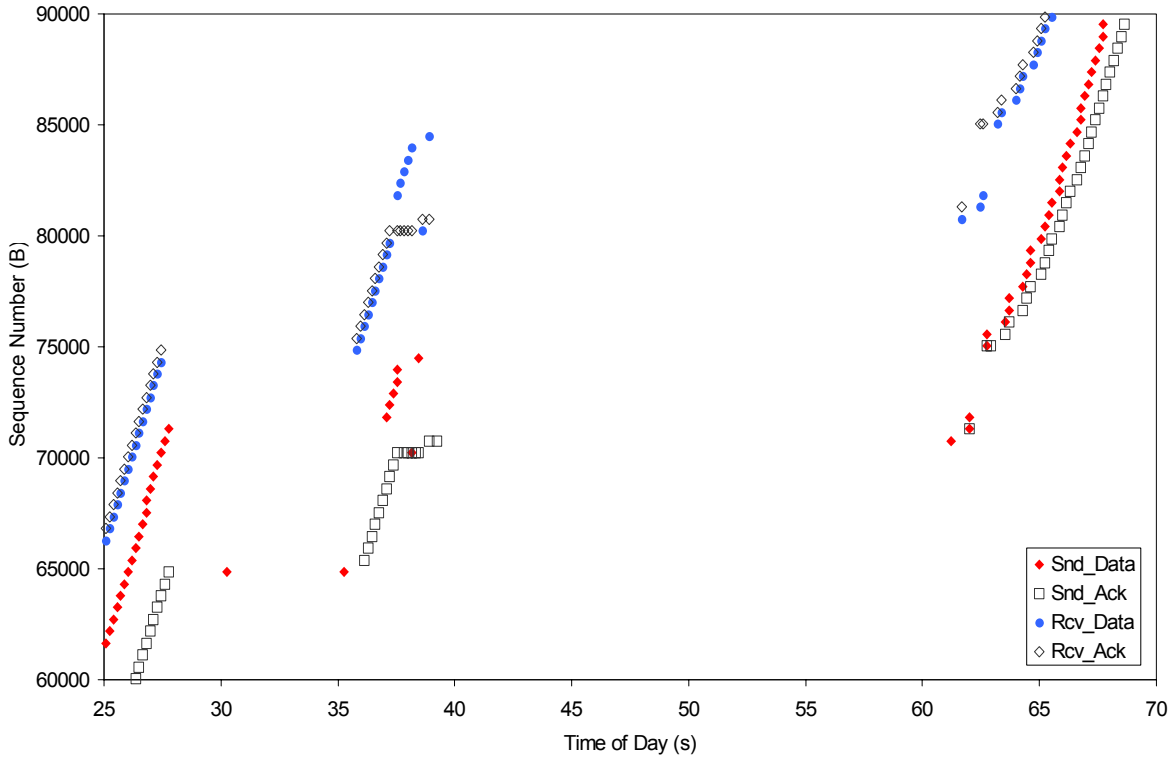


Figure 7. Performance problem of Reno with Eifel when packet losses are present.

6. CONCLUSIONS

We have studied the Eifel algorithm as defined in [4] for TCP Reno, NewReno and SACK in a simulated GPRS network. Large and sudden variations in packet transmission delays are often unavoidable in GPRS potentially causing spurious timeouts in TCP. For example, cell reselections may cause such delay variations. We simulated different scenarios with on average between one and three cell reselections taking place over the course of a two minutes bulk data transfer.

For mobile users and operators the battery power consumption and radio resource usage are often as important as the throughput across the wireless link. We therefore used throughput (download times) and goodput as equally important performance metrics. The bottleneck queue was assumed to be within the GPRS network. In bandwidth-dominated systems such as GPRS, the size of the bottleneck queue can greatly impact TCP's performance. We used two different sizes of the simulated drop-tail queue to capture this impact.

In case of a bottleneck queue that is sufficiently large to accommodate the maximum receiver window of a TCP connection, the Eifel algorithm improves the performance for all TCP flavors in all three scenarios. It reduces download times by up to 12 percent, and increases goodput by up to 20 percent. Bottleneck queues of such a size are often found in real GPRS networks.

In case of a smaller bottleneck queues, congestion losses may occur, and hence the TCP connection becomes network-limited. In that case, the Eifel algorithm still improves goodput by up to 10 percent for all TCP flavors in all three scenarios. For SACK and NewReno it also improves download times by up to 8 percent in all three scenarios.

Unexpectedly, TCP Reno yielded a considerable increase in download times when the Eifel algorithm was enabled and the bottleneck queue was small. We found that the reason for that were non-spurious timeouts with huge RTOs that typically follow a spurious timeout when packets from the outstanding flight were in fact lost due to congestion. From that we conclude that the Eifel algorithm is ideally complemented with an efficient SACK- or NewReno-based loss recovery scheme.

Our future work will focus on studying various modifications to the response part of the Eifel algorithm including how to reverse congestion control state, and how to adapt the round-trip time estimators.

REFERENCES

[1] 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.

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

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

[4] R. Ludwig and R. H. Katz. The Eifel algorithm: Making TCP robust against spurious retransmissions. ACM Computer Communication Review, 30(1), January 2000.

[5] A. Gurtov, Effect of Delays on TCP Performance, In Proceedings of IFIP Personal Wireless Communications, 2001.

[6] 3GPP TS 05.08 Radio subsystem link control, 2001.

[7] ETSI GSM 04.08, Mobile radio interface; Layer 3 specification.

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

[9] M. Schläger, NS TCP Eifel Page, http://www-tkn.ee.tu-berlin.de/~morten/eifel/ns-eifel.html

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

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

[12] S. McCanne and V. Jacobson, The BSD Packet Filter: A New Architecture for User-Level Packet Capture, In Proceedings of the 1993 Winter USENIX Conference.

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

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

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

Network Working Group
Internet-Draft
Expires: December 30, 2002

H. Inamura (editor)
NTT DoCoMo, Inc.
G. Montenegro (editor)
Sun Microsystems Laboratories, Europe
R. Ludwig
Ericsson Research
A. Gurtov
Sonera
F. Khafizov
Nortel Networks
July 1, 2002

TCP over Second (2.5G) and Third (3G) Generation Wireless Networks
draft-ietf-pilc-2.5g3g-10

Status of this Memo

Copyright Notice

Abstract

This document describes a profile for optimizing TCP over second (2.5G) and third (3G) generation wireless networks. We describe the relevant characteristics of 2.5G and 3G networks, and specific

features of example deployments of such networks. We then recommend TCP optimization mechanisms and discuss open issues. The configuration options in this document are commonly found in modern TCP stacks, and are widely available standards-track mechanisms that the community considers safe for use on the general Internet.

Table of Contents

1. Introduction 3
2. 2.5G and 3G Link Characteristics 5
2.1 Latency 5
2.2 Data Rates 5
2.3 Asymmetry 6
2.4 Delay Spikes 6
2.5 Packet Loss Due to Corruption 7
2.6 Intersystem Handovers 7
2.7 Bandwidth Oscillation 7
3. 2.5G and 3G Deployments 9
3.1 2.5G Technologies: GPRS, HSCSD, and EDGE 9
3.2 W-CDMA 9
3.3 CDMA2000 10
4. TCP over 2.5G and 3G 12
4.1 Appropriate Window Size (Sender & Receiver) 12
4.2 Increased Initial Window (Sender) 12
4.3 Limited Transmit (Sender) 13
4.4 IP MTU Larger than Default 13
4.5 Path MTU Discovery (Sender & Intermediate Routers) 14
4.6 Selective Acknowledgments (Sender & Receiver) 14
4.7 Explicit Congestion Notification (Sender, Receiver & Intermediate Routers) 14
4.8 TCP Timestamps Option (Sender & Receiver) 15
4.9 Disabling RFC1144 TCP/IP Header Compression (Wireless Host) 16
4.10 Summary 17
5. Open Issues 18
6. Security Considerations 20
7. IANA Considerations 21
8. Acknowledgements 22
References 23
Authors' Addresses 27
Full Copyright Statement 29

1. Introduction

The second generation cellular systems are commonly referred to as 2G. The 2G phase began in the 1990s when digital voice encoding had replaced analog systems (1G). 2G systems are based on various radio technologies including frequency-, code- and time- division multiple access. Examples of 2G systems include GSM (Europe), PDC (Japan), and IS-95 (USA). Data links provided by 2G systems are mostly circuit-switched and have transmission speeds of 10-20 kbps uplink and downlink. Demand for higher data rates, instant availability and data volume-based charging, as well as lack of radio spectrum allocated for 2G led to the introduction of 2.5G (GPRS, EDGE, PDC-P) and 3G (Wideband CDMA, cdma2000) systems.

Radio technology for both Wideband CDMA (W-CDMA) (Europe, Japan) and cdma2000 (US, South Korea) is based on code division multiple access allowing for higher data rates and more efficient spectrum utilization than 2G systems. 3G systems provide both packet-switched and circuit-switched connectivity in order to address the quality of service requirements of conversational, interactive, streaming, and bulk transfer applications. The transition to 3G is expected to be a gradual process. Initially, 3G will be deployed to introduce high capacity and high speed access in densely populated areas. Mobile users with multimode terminals will be able to utilize existing coverage of 2.5G systems on the rest of territory.

Much development and deployment activity has centered around 2.5G and 3G technologies. Along with objectives like increased capacity for voice channels, a primary motivation for these is data communication, and, in particular, Internet access. Accordingly, key issues are TCP performance and the several techniques which can be applied to optimize it over different wireless environments[1].

This document proposes a profile of such techniques, (particularly effective for use with 2.5G and 3G wireless networks). The configuration options in this document are commonly found in modern TCP stacks, and are widely available IETF standards-track mechanisms that the community has judged to be safe on the general Internet (that is, even in predominantly non-wireless scenarios). Furthermore, this document makes one set of recommendations that covers both 2.5G and 3G networks. One common set is warranted, because 2.5G and 3G networks share similar challenges to TCP performance (see Section 2).

Two example applications of the recommendations in this document are:

- The WAP Forum [12] (part of the Open Mobile Alliance [13] as of June 2002) is an industry association that has developed standards

for wireless information and telephony services on digital mobile phones. In order to address WAP functionality for higher speed networks such as 2.5G and 3G networks, and to aim at convergence with Internet standards, the WAP Forum thoroughly revised its specifications. The resultant version 2.0 [18] adopts TCP as its transport protocol, and recommends TCP optimization mechanisms closely aligned with those described in this document.

- I-mode[24] is a wireless Internet service deployed on handsets in Japan. The newer version of i-mode runs on FOMA [25], an implementation of W-CDMA. I-mode over FOMA deploys the profile of TCP described in this document.

This document is structured as follows: Section 2 reviews the link layer characteristics of 2.5G/3G networks; Section 3 gives an overview of some specific 2.5G/3G technologies like W-CDMA, cdma2000 and GPRS; Section 4 recommends mechanisms and configuration options for TCP implementations used in 2.5G/3G networks, including a summary in chart form at the end of the section; finally, Section 5 discusses some open issues.

2. 2.5G and 3G Link Characteristics

Link layer characteristics of 2.5G/3G networks have significant effects on TCP performance. In this section we present various aspects of link characteristics unique to the 2.5G/3G networks.

2.1 Latency

The latency of 2.5G/3G links is high mostly due to the extensive processing required at the physical layer of those networks, e.g., for FEC and interleaving, and due to transmission delays in the radio access network [55]. A typical RTT varies between a few hundred milliseconds and one second. The associated radio channels suffer from difficult propagation environments. Hence, powerful but complex physical layer techniques need to be applied to provide high capacity in a wide coverage area in a resource efficient way. Hopefully, rapid improvements in all areas of wireless networks ranging from radio layer techniques over signal processing to system architecture will ultimately also lead to reduced delays in 3G wireless systems.

2.2 Data Rates

The main incentives for transition from 2G to 2.5G to 3G are the increase in voice capacity and in data rates for the users. 2.5G systems have data rates of 10-20 kbps in uplink and 10-40 kbps in downlink. Initial 3G systems are expected to have bit rates around 64 kbps in uplink and 384 kbps in downlink. Considering the resulting bandwidth-delay product (BDP) of around 1-5 KB for 2.5G and 8-50 KB for 3G, 2.5G links can be considered LTNs (Long Thin Networks [1]), and 3G links approach LFNs (Long Fat Networks [4], as exemplified by some satellite networks [45]). For good TCP performance both LFNs and LTNs require maintaining a large enough window of outstanding data. For LFNs, utilizing the available network bandwidth is of particular concern. LTNs need a sufficiently large window for efficient loss recovery. In particular, the fast retransmit algorithm cannot be triggered if the window is less than four segments. This leads to a lengthy recovery through retransmission timeouts. The Limited Transmit algorithm RFC3042 [27] helps avoid the deleterious effects of timeouts on connections with small windows. Nevertheless, making full use of the SACK RFC2018 [5] information for loss recovery in both LFNs and LTNs may require twice the window otherwise sufficient to utilize the available bandwidth.

This document recommends only standard mechanisms suitable both for LTNs and LFNs, and to any network in general. However, experimental mechanisms suggested in Section 5 can be targeted either for LTNs [1] or LFNs [45].

Data rates are dynamic due to effects from other users and from mobility. Arriving and departing users can reduce or increase the available bandwidth in a cell. Increasing the distance from the base station decreases the link bandwidth due to reduced link quality. Finally, by simply moving into another cell the user can experience a sudden change in available bandwidth. For example, if upon changing cells a connection experiences a sudden increase in available bandwidth, it can underutilize it, because during congestion avoidance TCP increases the sending rate slowly. Changing from a fast to a slow cell normally is handled well by TCP due to the self-clocking property. However, a sudden increase in RTT in this case can cause a spurious TCP timeout as described in Section 2.7. In addition, a large TCP window used in the fast cell can create congestion resulting in overbuffering in the slow cell.

2.3 Asymmetry

2.5G/3G systems may run asymmetric uplink and downlink data rates. The uplink data rate is limited by battery power consumption and complexity limitations of mobile terminals. However, the asymmetry does not exceed 3-6 times, and can be tolerated by TCP without the need for techniques like ACK congestion control or ACK filtering [46]. Accordingly, this document does not include recommendations meant for such highly asymmetric networks.

2.4 Delay Spikes

A delay spike is a sudden increase in the latency of the communication path. 2.5G/3G links are likely to experience delay spikes exceeding the typical RTT by several times due to the following reasons.

1. A long delay spike can occur during link layer recovery from a link outage due to temporal loss of radio coverage, for example, while driving into a tunnel or within an elevator.

2. During a handover the mobile terminal and the new base station must exchange messages and perform some other time-consuming actions before data can be transmitted in a new cell.

3. Many wide area wireless networks provide seamless mobility by internally re-routing packets from the old to the new base station which may cause extra delay.

4. Blocking by high-priority traffic may occur when an arriving circuit-switched call or higher priority data temporarily preempts the radio channel. This happens because most current terminals are not able to handle a voice call and a data

connection simultaneously and suspend the data connection in this case.

5. Additionally, a scheduler in the radio network can suspend a low-priority data transfer to give the radio channel to higher priority users.

Delay spikes can cause spurious TCP timeouts, unnecessary retransmissions and a multiplicative decrease in the congestion window size.

2.5 Packet Loss Due to Corruption

Even in the face of a high probability of physical layer frame errors, 2.5G/3G systems have a low rate of packet losses thanks to link-level retransmissions. Justification for link layer ARQ is discussed in [10], [7], [41]. In general, link layer ARQ and FEC can provide a packet service with a negligibly small probability of undetected errors (failures of the link CRC), and a low level of loss (non-delivery) for the upper layer traffic, e.g., IP. The loss rate of IP packets is low due to the ARQ, but the recovery at the link layer appears as delay jitter to the higher layers lengthening the computed RTO value.

2.6 Intersystem Handovers

In the initial phase of deployment, 3G systems will be used as a 'hot spot' technology in high population areas, while 2.5G systems will provide lower speed data service elsewhere. This creates an environment where a mobile user can roam between 2.5G and 3G networks while keeping ongoing TCP connections. The inter-system handover is likely to trigger a high delay spike (Section 2.4), and can result in data loss. Additional problems arise because of context transfer, which is out of scope of this document, but is being addressed elsewhere in the IETF in activities addressing seamless mobility [47].

Intersystem handovers can adversely affect ongoing TCP connections since features may only be negotiated at connection establishment and cannot be changed later. After an intersystem handover, the network characteristics may be radically different, and, in fact, may be negatively affected by the initial configuration. This point argues against premature optimization by the TCP implementation.

2.7 Bandwidth Oscillation

Given the limited RF spectrum, satisfying the high data rate needs of 2.5G/3G wireless systems requires dynamic resource sharing among

concurrent data users. Various scheduling mechanisms can be deployed in order to maximize resource utilization. If multiple users wish to transfer large amounts of data at the same time, the scheduler may have to repeatedly allocate and de-allocate resources for each user. We refer to periodic allocation and release of high-speed channels as Bandwidth Oscillation. Bandwidth Oscillation effects such as spurious retransmissions were identified elsewhere (e.g., [17]) as factors that degrade throughput. There are research studies [48], [50], which show that in some cases Bandwidth Oscillation can be the single most important factor in reducing throughput. For fixed TCP parameters the achievable throughput depends on the pattern of resource allocation. When the frequency of resource allocation and de-allocation is sufficiently high, there is no throughput degradation. However, increasing the frequency of resource allocation/de-allocation may come at the expense of increased signaling, and, therefore, may not be desirable. Standards for 3G wireless technologies provide mechanisms that can be used to combat the adverse effects of Bandwidth Oscillation. It is the consensus of the PILC Working Group that the best approach for avoiding adverse effects of Bandwidth Oscillation is proper wireless sub-network design [10].

3. 2.5G and 3G Deployments

This section provides further details on specific 2.5G/3G technologies, namely, Wideband CDMA (W-CDMA), cdma2000 and GPRS. Other documents discuss the underlying technologies in more detail. For example, ARQ and FEC are discussed in [10], while further justification for link layer ARQ is discussed in [7], [41].

3.1 2.5G Technologies: GPRS, HSCSD, and EDGE

High Speed Circuit-Switched Data (HSCSD) and General Packet Radio Service (GPRS) are extensions of GSM providing high data rates for a user. Both extensions were developed first by ETSI and later by 3GPP. In GSM, a user is assigned one timeslot downlink and one uplink. HSCSD allocates multiple timeslots to a user creating a fast circuit-switched link. GPRS is based on packet-switched technology that allows efficient sharing of radio resources among users and always-on capability. Several terminals can share timeslots. A GPRS network uses an updated base station subsystem of GSM as the access network; the GPRS core network includes Serving GPRS Support Nodes (SGSN) and Gateway GPRS Support Nodes (GGSN). The RLC protocol operating between a base station controller and a terminal provides ARQ capability over the radio link. The Logical Link Control (LLC) protocol between the SGSN and the terminal also has an ARQ capability utilized during handovers.

Enhanced Data for Global Evolution (EDGE) uses a new modulation technique and new channel coding that increases throughput and capacity of the radio link. EDGE applied to GPRS (EGPRS) or HSCSD (ECSD) can increase the data rate threefold for a single user.

3.2 W-CDMA

The International Telecommunication Union (ITU) has selected Wideband Code Division Multiple Access (W-CDMA) as one of the global telecom systems for the IMT-2000 3G mobile communications standard. W-CDMA specifications are created in the 3rd Generation Partnership Project (3GPP).

The link layer characteristics of the 3G network which have the largest effect on TCP performance over the link are error controlling schemes such as layer two ARQ (L2 ARQ) and FEC (forward error correction).

W-CDMA (Wideband CDMA) uses RLC (Radio Link Control) [2], a Selective Repeat and sliding window ARQ. RLC uses protocol data units (PDUs) with a 16 bit RLC header. The size of the PDUs may vary. Typically, 336 bit PDUs are implemented [25]. This is the unit for link layer

retransmission. The IP packet is fragmented into PDUs for transmission by RLC. (For more fragmentation discussion, see Section 4.4.)

In W-CDMA, one to twelve PDUs (RLC frames) constitute one FEC frame, the actual size of which depends on link conditions and bandwidth allocation. The FEC frame is the unit of interleaving. This accumulation of PDUs for FEC adds part of the latency mentioned in Section 2.1.

For reliable transfer, RLC has an acknowledged mode for PDU retransmission. RLC uses checkpoint ARQ [2]. Using "status report" type acknowledgments: the poll bit in the header explicitly solicits the peer for a status report containing the sequence number that the peer acknowledged. The use of the poll bit is controlled by timers and by the size of available buffer space in RLC. Also, when the peer detects a gap between sequence numbers in received frames, it can issue a status report to invoke retransmission. RLC preserves the order of packet delivery.

The maximum number of retransmissions is a configurable RLC parameter that is specified by RRC [32] (Radio Resource Controller) through RLC connection initialization. The RRC can set the maximum number of retransmissions (up to a maximum of 40). Therefore, RLC can be described as an ARQ that can be configured for either HIGH-PERSISTENCE or LOW-PERSISTENCE, not PERFECT-PERSISTENCE, according to the terminology in [7].

Since the RRC manages RLC connection state, Bandwidth Oscillation (Section 2.7) can be eliminated by the RRC's keeping RF resource on an RLC connection with data in its queue. This avoids resource de-allocation in the middle of transferring data.

In summary, the link layer ARQ and FEC can provide a packet service with a negligibly small probability of undetected error (failure of the link CRC), and a low level of loss (non-delivery) for the upper layer traffic, i.e. IP. Retransmission of PDUs by ARQ introduces latency and delay jitter to the IP flow. This is why the transport layer sees the underlying W-CDMA network as a network with a relatively large BDP (Bandwidth-Delay Product) of up to 50 KB for the 384 kbps radio bearer.

3.3 CDMA2000

One of the Terrestrial Radio Interface standards for 3G wireless systems, proposed under the International Mobile Telecommunications-2000 umbrella, is cdma2000 [52]. It employs Multi-Carrier Code Division Multiple Access (CDMA) technology with a single-carrier RF

bandwidth of 1.25 MHz. cdma2000 evolved from IS-95 [53], a 2G standard based on CDMA technology. The first phase of cdma2000 utilizes a single carrier and is designed to double the voice capacity of existing CDMA (IS-95) networks and to support always-on data transmission speeds of up to 316.8 kbps. At the physical layer, the standard allows transmission in 5,10,20,40 or 80 ms time frames. Various orthogonal (Walsh) codes are used for channel identification and to achieve higher data rates.

Radio Link Protocol Type 3 (RLP) [54] is used with a cdma2000 Traffic Channel to support CDMA data services. RLP provides an octet stream transport service and is unaware of higher layer framing. There are several RLP frame formats. RLP frame formats with higher payload were designed for higher data rates. Depending on the channel speed, one or more RLP frames can be transmitted in a single physical layer frame.

RLP can substantially decrease the error rate exhibited by CDMA traffic channels [49]. When transferring data, RLP is a pure NAK-based finite selective repeat protocol. The receiver does not acknowledge successfully received data frames. If one or more RLP data frames are missing, the receiving RLP makes several attempts (called NAK rounds) to recover them by sending one or more NAK control frames to the transmitter. Each NAK frame must be sent in a separate physical layer frame. When RLP supplies the last NAK control frame of a particular NAK round, a retransmission timer is set. If the missing frame is not received when the timer expires, RLP may try another NAK round. RLP may not recover all missing frames. If after all RLP rounds, a frame is still missing, RLP supplies data with a missing frame to the higher layer protocols.

4. TCP over 2.5G and 3G

What follows is a set of recommendations for configuration parameters for protocol stacks which will be used to support TCP connections over 2.5G and 3G wireless networks. Some of these recommendations imply special configuration

- at the data receiver (frequently a stack at or near the wireless device),
- at the data sender (frequently a host in the Internet or possibly a gateway or proxy at the edge of a wireless network), or
- at both.

These configuration options are commonly available IETF standards-track mechanisms considered safe on the general Internet. System administrators are cautioned, however, that increasing the MTU size (Section 4.4) and disabling RFC1144 header compression (Section 4.9) could affect host efficiency, and that changing such parameters should be done with care.

4.1 Appropriate Window Size (Sender & Receiver)

TCP over 2.5G/3G should support appropriate window sizes based on the Bandwidth Delay Product (BDP) of the end-to-end path (see Section 2.2). The TCP specification [37] limits the receiver window size to 64 KB. If the end-to-end BDP is expected to be larger than 64 KB, the window scale option [4] can be used to overcome that limitation. Many operating systems by default use small TCP receive and send buffers around 16KB. Therefore, even for a BDP below 64 KB, the default buffer size setting should be increased at the sender and at the receiver to allow a large enough window.

4.2 Increased Initial Window (Sender)

TCP controls its transmit rate using the congestion window mechanism. The traditional initial window value of one segment, coupled with the delayed ACK mechanism [57] implies unecessary idle times in the initial phase of the connection, including the delayed ACK timeout (typically 200 ms, but potentially as much as 500 ms) [8]. Senders can avoid this by using a larger initial window. At the time of this writing, an increased initial window setting of two segments is already approved [3], and and a further increase up to four segments (not to exceed roughly 4 KB) has been proposed to become a standards track RFC [8]. Experiments with increased initial windows and related measurements have shown (1) that it is safe to deploy this mechanism (i.e. it does not lead to congestion collapse), and (2)

that it is especially effective for the transmission of a few TCP segments' worth of data (which is the behavior commonly seen in such applications as Internet-enabled mobile wireless devices). For large data transfers, on the other hand, the effect of this mechanism is negligible.

TCP over 2.5G/3G SHOULD set the initial CWND (congestion window) according to Equation 1 in [8]:

min (4*MSS, max (2*MSS, 4380 bytes))

This increases the permitted initial window from one to between two and four segments (not to exceed approximately 4 KB).

4.3 Limited Transmit (Sender)

RFC3042 [27], Limited Transmit, extends Fast Retransmit/Fast Recovery for TCP connections with small congestion windows that are not likely to generate the three duplicate acknowledgements required to trigger Fast Retransmit [3]. If a sender has previously unsent data queued for transmission, the limited transmit mechanism calls for sending a new data segment in response to each of the first two duplicate acknowledgments that arrive at the sender. This mechanism is effective when the congestion window size is small or if a large number of segments in a window are lost. This may avoid some retransmissions due to TCP timeouts. In particular, some studies [27] have shown that over half of a busy server's retransmissions were due to RTO expiration (as opposed to Fast Retransmit), and that roughly 25% of those could have been avoided using Limited Transmit. Similar to the discussion in Section 4.2, this mechanism is useful for small amounts of data to be transmitted. TCP over 2.5G/3G implementations SHOULD implement Limited Transmit.

4.4 IP MTU Larger than Default

The maximum size of an IP datagram supported by a link layer is the MTU (Maximum Transfer Unit). The link layer may, in turn, fragment IP datagrams into PDUs. For example, on links with high error rates, a smaller link PDU size increases the chance of successful transmission. With layer two ARQ and transparent link layer fragmentation, the network layer can enjoy a larger MTU even in a relatively high BER (Bit Error Rate) condition. Without these features in the link, a smaller MTU is suggested.

TCP over 2.5G/3G should allow freedom for designers to choose MTU values ranging from small values (such as 576 bytes) to a large value that is supported by the type of link in use (such as 1500 bytes for IP packets on Ethernet). Given that the window is counted in units

of segments, a larger MTU allows TCP to increase the congestion window faster [9]. Hence, designers are generally encouraged to choose larger values. These may exceed the default IP MTU values of 576 bytes for IPv4 RFC1191 [19] and 1280 bytes for IPv6 [59]. While this recommendation is applicable to 3G networks, operation over 2.5G networks should exercise caution as per the recommendations in RFC3150 [9].

4.5 Path MTU Discovery (Sender & Intermediate Routers)

Path MTU discovery allows a sender to determine the maximum end-to-end transmission unit (without IP fragmentation) for a given routing path. RFC1191 [19] and RFC1981 [21] describe the MTU discovery procedure for IPv4 and IPv6, respectively. This allows TCP senders to employ larger segment sizes (without causing IP layer fragmentation) instead of assuming the small default MTU. TCP over 2.5G/3G implementations should implement Path MTU Discovery. Path MTU Discovery requires intermediate routers to support the generation of the necessary ICMP messages. RFC1435 [20] provides recommendations that may be relevant for some router implementations.

4.6 Selective Acknowledgments (Sender & Receiver)

The selective acknowledgment option (SACK), RFC2018 [5], is effective when multiple TCP segments are lost in a single TCP window[11]. In particular, if the end-to-end path has a large BDP and a high packet loss rate, the probability of multiple segment losses in a single window of data increases. In such cases, SACK provides robustness beyond TCP-Tahoe and TCP-Reno [6]. TCP over 2.5G/3G SHOULD support SACK.

In the absence of SACK feature, the TCP should use NewReno RFC2582 [38].

4.7 Explicit Congestion Notification (Sender, Receiver & Intermediate Routers)

Explicit Congestion Notification, RFC3168 [23], allows a TCP receiver to inform the sender of congestion in the network by setting the ECN-Echo flag upon receiving an IP packet marked with the CE bit(s). The TCP sender will then reduce its congestion window. Thus, the use of ECN is believed to provide performance benefits [22], [40]. RFC3168 [23] also places requirements on intermediate routers (e.g. active queue management and setting of the CE bit(s) in the IP header to indicate congestion). Therefore, the potential improvement in performance can only be achieved when ECN capable routers are deployed along the path. TCP over 2.5G/3G SHOULD support ECN.

4.8 TCP Timestamps Option (Sender & Receiver)

Traditionally, TCPs collect one RTT sample per window of data [37], [57]. This can lead to an underestimation of the RTT, and spurious timeouts on paths in which the packet transmission delay dominates the RTT. This holds despite a conservative retransmit timer such as the one specified in RFC2988 [31]. TCP connections with large windows may benefit from more frequent RTT samples provided with timestamps by adapting quicker to changing network conditions [4]. However, there is some empirical evidence that for TCPs with an RFC2988 timer [31], timestamps provide little or no benefits on backbone Internet paths [56]. Using the TCP Timestamps option has the advantage that retransmitted segments can be used for RTT measurement, which is otherwise forbidden by Karn's algorithm [39], [4]. Furthermore, the TCP Timestamps option is the basis for detecting spurious retransmits using the Eifel algorithm [17].

A 2.5/3G link (layer) is dedicated to a single host. It therefore only experiences a low degree of statistical multiplexing between different flows. Also, the packet transmission and queueing delays of a 2.5/3G link often dominate the path's RTT. This already results in large RTT variations as packets fill the queue while a TCP sender probes for more bandwidth, or as packets drain from the queue while a TCP sender reduces its load in response to a packet loss. In addition, the delay spikes across a 2.5/3G link (see Section 2.4) may often exceed the end-to-end RTT. The thus resulting large variations in the path's RTT may often cause spurious timeouts.

When running TCP in such an environment, it is therefore advantageous to sample the path's RTT more often than only once per RTT. This allows the TCP sender to track changes in the RTT more closely. In particular, a TCP sender can react more quickly to sudden increases of the RTT by sooner updating the RTO to a more conservative value. The TCP Timestamps option [4] provides this capability, allowing the TCP sender to sample the RTT from every segment that is acknowledged. Using timestamps in the mentioned scenario leads to a more conservative TCP retransmission timer and reduces the risk of triggering spurious timeouts [42], [48], [50], [58].

There are two problematic issues with using timestamps:

- 12 bytes of overhead are introduced by carrying the TCP Timestamps option and padding in the TCP header. For a small MTU size, it can present a considerable overhead. For example, for an MTU of 296 bytes the added overhead is 4%. For an MTU of 1500 bytes, the added overhead is only 0.8%.

- Current TCP header compression schemes are limited in their

handling of the TCP options field. For RFC2507 [35], any change in the options field (caused by timestamps or SACK, for example) renders the entire field uncompressible (leaving the TCP/IP header itself compressible, however). Even worse, for RFC1144 [34] such a change in the options field effectively disables TCP/IP header compression altogether. This is the case when a connection uses the TCP Timestamps option. That option field is used both in the data and the ACK path, and its value typically changes from one packet to the next. The IETF is currently specifying a robust TCP/IP header compression scheme with better support for TCP options [16].

The original definition of the timestamps option [4] specifies that duplicate segments below cumulative ACK do not update the cached timestamp value at the receiver. This may lead to overestimating of RTT for retransmitted segments. A possible solution [44] allows the receiver to use a more recent timestamp from a duplicate segment. However, this suggestion allows for spoofing attacks against the TCP receiver. Therefore, careful consideration is needed in implementing this solution.

Recommendation: TCP SHOULD use the TCP Timestamps option. It allows for better RTT estimation, reduces the risk of spurious timeouts, and enables the detection of spurious retransmits using the Eifel algorithm.

4.9 Disabling RFC1144 TCP/IP Header Compression (Wireless Host)

It is well known (and has been shown with experimental data) that RFC1144 [34] TCP header compression does not perform well in the presence of packet losses [40], [48]. If a wireless link error is not recovered, it will cause TCP segment loss between the compressor and decompressor, and then RFC1144 header compression does not allow TCP to take advantage of Fast Retransmit Fast Recovery mechanism. The RFC1144 header compression algorithm does not transmit the entire TCP/IP headers, but only the changes in the headers of consecutive segments. Therefore, loss of a single TCP segment on the link causes the transmitting and receiving TCP sequence numbers to fall out of synchronization. Hence, when a TCP segment is lost after the compressor, the decompressor will generate false TCP headers. Consequently, the TCP receiver will discard all remaining packets in the current window because of a checksum error. This continues until the compressor receives the first retransmission which is forwarded uncompressed to synchronize the decompressor [34].

As previously recommended in RFC3150 [9], RFC1144 header compression SHOULD NOT be enabled unless the packet loss probability between the compressor and decompressor is very low. Actually, enabling the

Timestamps Option effectively accomplishes the same thing (see Section 4.8). Other header compression schemes like RFC2507 [35] and Robust Header Compression [33] are meant to address deficiencies in RFC1144 header compression. At the time of this writing, the IETF was working on multiple extensions to Robust Header Compression (negotiating Robust Header Compression over PPP, compressing TCP options, etc) [51].

4.10 Summary

Items	Comments
Appropriate Window Size	(sender & receiver) based on end-to-end BDP
Window Scale Option [RFC1323]	(sender & receiver) Window size > 64KB
Increased Initial Window [RFC TBA]	(sender) CWND = min (4*MSS, max (2*MSS, 4380 bytes))
Limited Transmit [RFC3042]	(sender)
IP MTU larger than Default	more applicable to 3G
Path MTU Discovery [RFC1191,RFC1981]	(sender & intermediate routers)
Selective Acknowledgment option (SACK) [RFC2018]	(sender & receiver)
Explicit Congestion Notification(ECN) [RFC3168]	(sender, receiver & intermediate routers)
Timestamps Option [RFC1323, R.T.Braden's ID]	(sender & receiver)
Disabling RFC1144 TCP/IP Header Compression [RFC1144]	(wireless host)

5. Open Issues

This section outlines additional mechanisms and parameter settings that may increase end-to-end performance when running TCP across 2.5G/3G networks. Note, that apart from the discussion of the RTO's initial value, those mechanisms and parameter settings are not part of any standards track RFC at the time of this writing. Therefore, they cannot be recommended for the Internet in general.

Other mechanisms for increasing TCP performance include enhanced TCP/IP header compression schemes [16], and active queue management RFC2309 [15], link layer retransmission schemes [10], and caching packets during transient link outages to retransmit them locally when the link is restored to operation [10].

Shortcomings of existing TCP/IP header compression schemes (RFC1144 [34], RFC2507 [35]) are that they do not compress headers of handshaking packets (SYNs and FINs), and that they lack proper handling of TCP option fields (e.g., SACK or timestamps) (see Section 4.8). Although RFC3095 [33] does not yet address this issue, the IETF is developing improved TCP/IP header compression schemes, including better handling of TCP options such as timestamps and selective acknowledgements. Especially, if many short-lived TCP connections run across the link, the compression of the handshaking packets may greatly improve the overall header compression ratio.

Implementing active queue management is attractive for a number of reasons as outlined in RFC2309 [15]. One important benefit for 2.5G/3G networks, is that it minimizes the amount of potentially stale data that may be queued in the network ("clicking from page to page" before the download of the previous page is complete). Avoiding the transmission of stale data across the 2.5G/3G radio link saves transmission (battery) power, and increases the ratio of useful data over total data transmitted. Another important benefit of active queue management for 2.5G/3G networks, is that it reduces the risk of a spurious timeout for the first data segment as outlined below.

Since 2.5G/3G networks are commonly characterized by high delays, avoiding unecessary round-trip times is particularly attractive. This is specially beneficial for short-lived, transactional (request/response-style) TCP sessions that typically result from browsing the Web from a smart phone. However, existing solutions such as T/TCP RFC1644 [14], have not been adopted due to known security concerns [30].

Spurious timeouts, packet re-ordering, and packet duplication may reduce TCP's performance. Thus, making TCP more robust against those events is desirable. Solutions to this problem have been proposed

[17], [26], [36], and standardization work within the IETF is ongoing at the time of writing. Those solutions include reverting congestion control state after such an event has been detected, and adapting the retransmission timer and duplicate acknowledgement threshold. The deployment of such solutions may be particularly beneficial when running TCP across wireless networks because wireless access links may often be subject to handovers and resource preemption, or the mobile transmitter may traverse through a radio coverage hole. Such disrupting events may easily trigger a spurious timeout despite a conservative retransmission timer. Also, the mobility mechanisms of some wireless networks may cause packet duplication.

The algorithm for computing TCP's retransmission timer is specified in RFC2988 [31]. The standard specifies that the initial setting of the retransmission timeout value (RTO) should not be less than 3 seconds. This value might be too low when running TCP across 2.5G/3G networks. In addition to its high latencies, those networks may be run at bit rates of as low as about 10 kb/s which results in large packet transmission delays. In this case, the RTT for the first data segment may easily exceed the initial TCP retransmission timer setting of 3 seconds. This would then cause a spurious timeout for that segment. Hence, in such situations it may be advisable to set TCP's initial RTO to a value larger than 3 seconds. Furthermore, due to the potentially large packet transmission delays, a TCP sender might choose to refrain from initializing its RTO from the RTT measured for the SYN, but instead take the RTT measured for the first data segment.

Some of the recommendations in RFC2988 [31] are optional, and are not followed by all TCP implementations. Specifically, some TCP stacks allow a minimum RTO less than the recommended value of 1 second (section 2.4 of [31]), and some implementations do not implement the recommended restart of the RTO timer when an ACK is received (section 5.3 of [31]). Some experiments [48], [50], have shown that in the face of bandwidth oscillation, using the recommended minimum RTO value of 1 sec (along with the also recommended initial RTO of 3 sec) reduces the number of spurious retransmissions as compared to using small minimum RTO values of 200 or 400 ms. Furthermore, TCP stacks that restart the retransmission timer when an ACK is received experience far less spurious retransmissions than implementations that do not restart the RTO timer when an ACK is received. Therefore, at the time of this writing, it seems preferable for TCP implementations used in 3G wireless data transmission to comply with all recommendations of RFC2988.

6. Security Considerations

In 2.5G/3G wireless networks, data is transmitted as ciphertext over the air and as cleartext between the Radio Access Network (RAN) and the core network. IP security RFC2401 [29] or TLS RFC2246 [28] can be deployed by user devices for end-to-end security. The use of a transport gateway introduces conflicts with IPsec; however TLS can be used in such architectures.

7. IANA Considerations

This document has been written assuming that Larger Initial Windows will be a proposed standard soon. Once this happens, the related text in this document should be updated (section 4.2, the table in section 4.10 and reference entry [8]). At that point this section should be deleted.

8. Acknowledgements

The authors would like to acknowledge the contribution to the text from the following individuals:

Max Hata, NTT DoCoMo, Inc. (hata@mml.yrp.nttdocomo.co.jp)

Masahiro Hara, Fujitsu, Inc. (mhara@FLAB.FUJITSU.CO.JP)

Joby James, Motorola, Inc. (joby@MIEL.MOT.COM)

William Gilliam, Hewlett-Packard Company (wag@cup.hp.com)

Alan Hameed, Fujitsu FNC, Inc. (Alan.Hameed@fnc.fujitsu.com)

Rodrigo Garces (rgarces2000@yahoo.com)

Peter Ford, Microsoft (peterf@Exchange.Microsoft.com)

Fergus Wills, Openwave (fergus.wills@openwave.com)

Michael Meyer (Michael.Meyer@eed.ericsson.se)

The authors gratefully acknowledge the valuable advice from the following individuals:

Gorry Fairhurst (gorry@erg.abdn.ac.uk)

Mark Allman (mallman@grc.nasa.gov)

Aaron Falk (falk@ISI.EDU)

References

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

[2] Third Generation Partnership Project, "RLC Protocol Specification (3G TS 25.322:)", 1999.

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

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

[5] Mathis, M., Mahdavi, J., Floyd, S. and R. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

[6] Fall, K. and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", Computer Communication Review, 26(3) , July 1996.

[7] Fairhurst, G. and L. Wood, "Link ARQ issues for IP traffic", Internet draft , March 2002, <http://www.ietf.org/internet-drafts/draft-ietf-pilc-link-arq-issues-04.txt>.

[8] Allman, M., Floyd, S. and C. Partridge, "Increased TCP's Initial Window", Internet draft (currently under IESG review), April 2002, <http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-initwin-03.txt>.

[9] Dawkins, S., Montenegro, G., Kojo, M. and V. Magret, "End-to-end Performance Implications of Slow Links", RFC 3150/BCP 48, July 2001.

[10] Karn, P., "Advice for Internet Subnetwork Designers", Internet draft , May 2002, <http://www.ietf.org/internet-drafts/draft-ietf-pilc-link-design-11.txt>.

[11] Dawkins, S., Montenegro, G., Magret, V., Vaidya, N. and M. Kojo, "End-to-end Performance Implications of Links with Errors", RFC 3135/BCP 50, August 2001.

[12] Wireless Application Protocol, "WAP Specifications", 2002, <http://www.wapforum.org>.

[13] Open Mobile Alliance, "Open Mobile Alliance", 2002, <http://www.openmobilealliance.org/>.

[14] Braden, R., "T/TCP -- TCP Extensions for Transactions", RFC 1644, July 1994.

[15] Braden, R., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J. and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.

[16] IETF, "Robust Header Compression", 2001, <http://www.ietf.org/html.charters/rohc-charter.html>.

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

[18] Wireless Application Protocol, "WAP Wireless Profiled TCP", WAP-225-TCP-20010331-a, April 2001, <http://www.wapforum.com/what/technical.htm>.

[19] Mogul, J. and S. Deering, "Path MTU Discovery", RFC 1191, November 1990.

[20] Knowles, S., "IESG Advice from Experience with Path MTU Discovery", RFC 1435, March 1993.

[21] McCann, J., Deering, S. and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.

[22] Hadi Salim, J. and U. Ahmed, "Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks", RFC 2884, july 2000.

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

[24] NTT DoCoMo Technical Journal, "Special Issue on i-mode Service", October 1999.

[25] NTT DoCoMo Technical Journal, "Special Article on IMT-2000 Services", September 2001.

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

[27] Allman, M., Balakrishnan, H. and S. Floyd, "Enhancing TCP's

Loss Recovery Using Limited Transmit", RFC 3042, January 2001.

[28] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

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

[30] de Vivo, M., O. de Vivo, G., Koeneke, R. and G. Isern, "Internet Vulnerabilities Related to TCP/IP and T/TCP", ACM Computer Communication Review 29(1), January 1999.

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

[32] Third Generation Partnership Project, "RRC Protocol Specification (3GPP TS 25.331:)", September 2001.

[33] Bormann, C., Burmeister, C., Degermark, M., Fukushima, H., Hannu, H., Jonsson, L-E., Hakenberg, R., Koren, T., Le, K., Liu, Z., Martensson, A., Miyazaki, A., Svanbro, K., Wiebke, T., Yoshimura, T. and H. Zheng, "RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed", RFC 3095, July 2001.

[34] Jacobson, V., "Compressing TCP/IP Headers for Low-Speed Serial Links,", RFC 1144, Feb 1990.

[35] Degermark, M., Nordgren, B. and S. Pink, "IP Header Compression", RFC 2507, February 1999.

[36] Blanton, E. and M. Allman, "On Making TCP More Robust to Packet Reordering", ACM Computer Communication Review 32(1), January 2002, <http://roland.grc.nasa.gov/~mallman/papers/tcp-reorder-ccr.ps>.

[37] Postel, J., "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, September 1981.

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

[39] Karn, P. and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols", ACM SIGCOMM 87, 1987.

[40] Ludwig, R., Rathonyi, B., Konrad, A. and A. Joseph, "Multi-layer tracing of TCP over a reliable wireless link", ACM SIGMETRICS 99, May 1999.

[41] Ludwig, R., Konrad, A., Joseph, A. and R. Katz, "Optimizing the End-to-End Performance of Reliable Flows over Wireless Links", Kluwer/ACM Wireless Networks Journal Vol. 8, Nos. 2/3, pp. 289-299, March-May 2002.

[42] Gurtov, A., "Making TCP Robust Against Delay Spikes", University of Helsinki, Department of Computer Science, Series of Publications C, C-2001-53, Nov 2001, <http://www.cs.helsinki.fi/u/gurtov/papers/report01.html>.

[43] Stevens, W., "TCP/IP Illustrated, Volume 1; The Protocols,", Addison Wesley , 1995.

[44] Braden, R., "TCP Extensions for High Performance: An Update", Internet draft , Jun 1993, <http://www.kohala.com/start/tcplw-extensions.txt>.

[45] Allman, M., Dawkins, S., Glover, D., Griner, J., Tran, D., Henderson, T., Heidemann, J., Touch, J., Kruse, H., Ostermann, S., Ostermann, S., Scott, K. and J. Semke, "Ongoing TCP Researh Related to Satellites", RFC 2760, Feb 2000.

[46] Balakrishnan, H., Padmanabhan, V., Fairhurst, G. and M. Sooriyabandara, "TCP Performance Implications of Network Asymmetry", Internet draft , November 2001, <http://www.ietf.org/internet-drafts/draft-ietf-pilc-asym-07.txt>.

[47] Kempf, J., "Problem Description: Reasons For Performing Context Transfers Between Nodes in an IP Access Network", Internet draft , November 2001, <http://www.ietf.org/internet-drafts/draft-ietf-seamoby-context-transfer-problem-stat-04.txt>.

[48] Khafizov, F. and M. Yavuz, "Running TCP over IS-2000", Proc. of IEEE ICC 2002.

[49] Khafizov, F. and M. Yavuz, "Analytical Model of RLP in IS-2000 CDMA Networks", Proc. of IEEE Vehicular Technology Conference , September 2002.

[50] Yavuz, M. and F. Khafizov, "TCP over Wireless Links with Variable Bandwidth", Proc. of IEEE Vehicular Technology Conference , September 2002.

[51] Bormann, C., "Robust Header Compression (ROHC) over PPP", RFC 3241, April 2002.

[52] TIA/EIA/cdma2000, "Mobile Station - Base Station Compatibility Standard for Dual-Mode Wideband Spread Spectrum Cellular

Systems", Washington: Telecommunication Industry Association , 1999.

[53] TIA/EIA/IS-95 Rev A, "Mobile Station - Base Station Compatibility Standard for Dual-Mode Wideband Spread Spectrum Cellular Systems", Washington: Telecommunication Industry Association , 1995.

[54] TIA/EIA/IS-707-A-2.10, "Data Service Options for Spread Spectrum Systems: Radio Link Protocol Type 3", January 2000.

[55] Dahlman, E., Beming, P., Knutsson, J., Ovesjo, F., Persson, M. and C. Roobol, "WCDMA - The Radio Interface for Future Mobile Multimedia Communications", IEEE Trans. on Vehicular Technology, vol. 47, no. 4, pp. 1105-1118 , November 1998.

[56] Allman, M. and V. Paxson, "On Estimating End-to-End Network Path Properties", ACM SIGCOMM 99, September 1999.

[57] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[58] Gurtov, A. and R. Ludwig, "Making TCP Robust Against Delay Spikes", draft-gurtov-tsvwg-tcp-delay-spikes-00 (work in progress), February 2002.

[59] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

Authors' Addresses

Hiroshi Inamura
NTT DoCoMo, Inc.
3-5 Hikarinooka
Yokosuka Shi, Kanagawa Ken 239-8536
Japan

EMail: inamura@mml.yrp.nttdocomo.co.jp
URI: http://www.nttdocomo.co.jp/

Gabriel Montenegro
Sun Microsystems Laboratories, Europe
29, chemin du Vieux Chene
38240 Meylan
France

EMail: gab@sun.com

Reiner Ludwig
Ericsson Research
Ericsson Allee 1
52134 Herzogenrath
Germany

EMail: Reiner.Ludwig@Ericsson.com

Andrei Gurtov
Sonera
P.O. Box 970, FIN-00051
Helsinki,
Finland

EMail: andrei.gurtov@sonera.com
URI: http://www.cs.helsinki.fi/u/gurtov/

Farid Khafizov
Nortel Networks
2201 Lakeside Blvd
Richardson, TX 75082,
USA

EMail: faridk@nortelnetworks.com

Multi-Layer Protocol Tracing in a GPRS Network

Andrei Gurtov, Matti Passoja, Olli Aalto, Mika Raitola
Cellular Systems Development
Sonera
Helsinki, Finland

***Abstract*—This paper presents a performance evaluation of GPRS accomplished by combination of measurement at the end hosts and tracing inside the network. The multi-layer tracing approach allows not only observing, but also understanding the network performance. With end-to-end measurements we assess data rates, latency, and buffering experienced by users in a live GPRS network. Comparing the results to our previous measurements shows a notable improvement in the network and terminals over past two years. Mobility tests while driving in the urban environment quantify the interval, duration and data loss caused by cell reselections. In the test lab, multi-layer tracing of radio, link and transport protocols gives a closer picture of GPRS performance. For instance, TCP interacts inefficiently with resource allocation at the RLC layer and fragmentation at the LLC layer. Finally, we illustrate delay spikes and data losses during a cell reselection by tracing of signaling messages during a cell update and routing area update procedures.**

Keywords—performance, wireless, link layer, TCP, protocol

I. INTRODUCTION

General Packet Radio Service (GPRS) [1] is a packet-switched wireless wide area network being deployed worldwide. Performance evaluation of GPRS is an active research area. In particular, TCP performance [3], buffering [4], scheduling [9] and mobility procedures have been studied through analytical analysis and simulation. We present a performance study based on measurement data collected in live and test GPRS networks [11]. Some of the issues presented in this paper are affected by implementation details and the standardization status of the available network and terminal equipment. Therefore, we do not claim that the results apply for *all* GPRS users in general.

In the first part of the paper we evaluate performance of data transmission in live GPRS network from the end user point of view. Uplink and downlink throughput, round trip time, buffer size, delay spikes and data losses are characterized by end-to-end measurements. Comparing with our earlier results [2], data rates, latency and reliability are notably improved. For instance, the maximum downlink user throughput increased from 27 kbps to 43 kbps. As expected, we have not detected error losses at the transport layer due to the reliable link layer protocol in GPRS. Instead, we find delay spikes and bursty losses due to mobility procedures.

Sufficient buffering is crucial to achieve efficient multiplexing of bursty user traffic over the radio link. However, our buffer measurement suggests that a GPRS network may overbuffer user data. Too large buffers cause negative effects such as high round trip time and delivery of stale data when the user "clicks from page to page" [6]. From the TCP point of view, the optimal buffer size should be slightly above the bandwidth-delay product of the network [10].

A powerful multi-layer tracing methodology is introduced in [6] to study the GSM data transmission. We perform multi-layer tracing in a GPRS test network. We found that if the frame size at the logical link layer is not configured to match with a typical IP packet size, small fragments are inefficiently transmitted. Furthermore, we confirm that the radio resource allocation can interact inefficiently with TCP [7] and, on the other hand, that competing error recovery between radio link and TCP is uncommon [3]. At the radio layer, we observed unnecessary retransmissions due to trade-offs in the acknowledgment and retransmission policy.

During mobility tests we measure frequency, duration and data loss of cell reselections in GPRS. In the test lab we trace the signaling procedures and end-to-end TCP behavior. In the test lab the duration of cell reselections is typically below 5 s, there as in the live network it may take up to twenty seconds.

The rest of the paper is organized as follows. In Section 2 the relevant aspects of data transmission over GPRS are presented. Section 3 describes the configuration of our GPRS

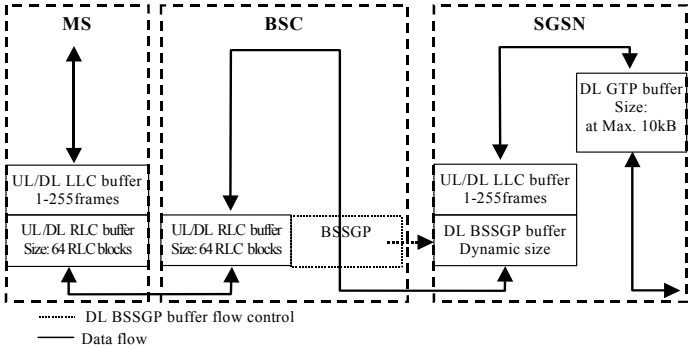


Figure 1. Buffering of user data in GPRS.

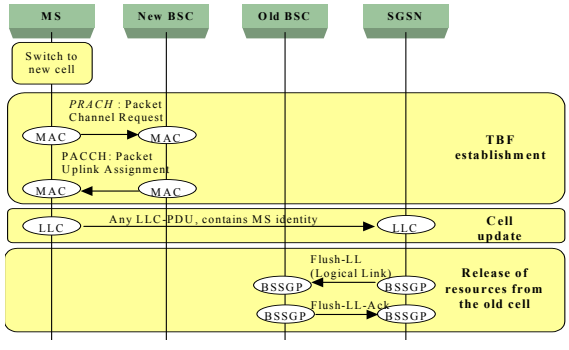


Figure 2. The cell update procedure in GPRS.

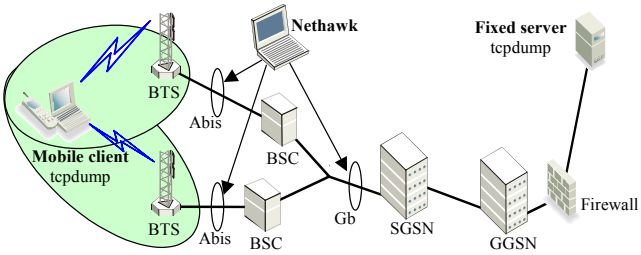


Figure 3. Measurement setup in the test lab.

Vendor	Model	Timeslots (UL+DL)	Multislot class	Inter-face
Ericsson	T39m	1+3	4	Serial
Ericsson	R520m	1+3	4	Serial
Nokia	N8310	2+3	6	IrDA
Motorola	T280	1+4	8	USB

Figure 4. Mobile terminals used for measurements.

network and setup of measurement. Section 4 reports end-to-end and tracing results. Finally, Section 5 sums up the main findings and outlines the future work.

II. THE GPRS NETWORK

Figure 1 and 3 illustrate the data transmission path of GPRS. The relevant network elements for us are the Mobile Station (MS), Base Transceiver Station (BTS), Base Station Controller (BSC), Serving GPRS Support Node (SGSN) and Gateway GPRS Support Node (GGSN). BSC handles the medium access and radio resource scheduling, as well as data transmission toward MS over the Abis interface. SGSN handles mobility and controls the data flow toward BSC over the Gb interface. GGSN provides connectivity to external packet networks. A firewall shields the GPRS network from the rest of the Internet. A detailed overview of the GPRS system can be found in [1].

A. The GPRS Protocol Stack

Figure 1 shows the protocol stack of the user data transmission plane of GPRS. The Radio Link Control (RLC) protocol provides acknowledged or unacknowledged data transfer between MS and BSC in uplink (UL) and downlink (DL) directions. The Logical Link Control (LLC) protocol provides acknowledged and unacknowledged mode between MS and SGSN. The Base Station Subsystem GPRS (BSSGP) protocol controls the data flow between BSC and SGSN. Finally, GPRS Tunneling Protocol (GTP) encapsulates user packets for delivery between SGSN and GGSN.

Medium Access Control (MAC) manages sharing of radio resources among multiple users. MS can utilize several radio timeslots simultaneously to increase the data rate and decrease the transmission latency. The multislot class of MS determines the maximum number of timeslots in uplink and downlink. Before transmitting user data, MS must activate a Temporal Block Flow (TBF) toward BSC. MSs contend on an ALOHA-style random access channel to receive a resource allocation from the network. Optionally, a second stage is used for extending the assignment if MS is not satisfied with allocated resources.

RLC operates on small (20 to 50 bytes) blocks of user data that are encoded with one of a coding schemes (CS-1 to CS-4 for basic GPRS) to provide Forward Error Correction (FEC). RLC uses wrapping sequence numbers for blocks in the range of 0-127 and a sliding window of 64 blocks. In an acknowledged mode, a bitmap of received block is used to retransmit missing blocks. In contrary to TCP, RLC ACKs are sent on a separate control channel and cannot be piggybacked onto the reverse data traffic. BSC controls the acknowledgment frequency by sending ACKs in downlink or polling MS for ACKs in uplink. As we will see in Section IV, frequency of ACKs and retransmission policy at the sender are important to avoid unnecessary retransmissions in RLC. A situation when too many outstanding unacknowledged blocks prevent advancing of the sliding window should be avoided. Then, the window is *stalled*, and no new data blocks can be transmitted.

LLC provides a retransmission capability between MS and SGSN, and is supposed to recover losses caused by mobility. However, most GPRS networks nowadays operate in the unacknowledged LLC mode. LLC fragments and reassembles user packets if they exceed the maximum size. It can be configured up to 1556 bytes.

B. Buffering of User Data in GPRS

Figure 1 illustrates buffering of user data in GPRS. Buffering is performed at multiple protocol layers, but a corresponding buffer is used only if the protocol operates in the acknowledged mode. In our measurements, reliable RLC and unreliable LLC modes are used, thus the only enabled buffers are at the RLC and BSSGP layer. In downlink, LLC frames are stored in the BSSGP buffer in SGSN prior to transmission over the Gb interface. Although the buffer is located in SGSN, it is controlled by the BSSGP function in BSC. This enables BSC to adjust the data flow rate from SGSN in order to match it with available radio resources to prevent an overflow of the RLC buffer. Therefore, BSSGP buffer can be seen as an extension of the RLC buffer. The RLC buffer size is 64-128 RLC blocks or up to 6 kilobytes of the user data. Using multiple timeslots the content of the RLC buffer can be transmitted in a few hundreds ms. Therefore, multiple retransmissions can easily stall the window. This problem is corrected in Enhanced GPRS where the RLC buffer size can be 64 – 1024 blocks [5].

C. Cell Reselection

In the release 97 GPRS the mobile terminal selects the serving cell. This is different from circuit-switched GSM data where a handover is controlled by the network. In the simplest case when the user changes the serving cell while staying in the same routing area, a cell update procedure is performed. A *routing area* is a group of cell arranged together to balance between signaling overhead and positioning of MS. When the

new cell belongs to a different routing area, the cell reselection involves more signaling, especially if GSM-specific location information is updated as well. Finally, the most complicated case concerns an inter-SGSN handover. However, it is expected to be a rare event and therefore we have not measured it.

Figure 2 shows signaling required to accomplish a cell update. First, MS makes a cell reselection decision based on tracking signal power of surrounding cells. After synchronizing at the frequency in the new cell, MS starts a random access procedure to acquire radio resources. Then, MS starts transmitting data in the new cell. When SGSN receives an LLC frame with a new cell identity, it internally updates the MS location. Finally, SGSN signals the old cell using BSSGP protocol to release any resource reservations for MS and discard buffered data.

Shortcomings of the mobile-driven cell reselection are widely recognized and improvements are being standardized in 3GPP. A Network Controlled Cell Change (NCCC) will make the BSC responsible for a cell change decision. NCCC will eliminate unnecessary cell reselections currently performed by stationary MSs. A Network Assisted Cell Change (NACC) should reduce the delay and data losses seen by a moving MS.

III. MEASUREMENT ARRANGEMENTS

At the time of measurements, Sonera's GPRS network is implementing the 3GPP release 97. Figure 4 lists mobile terminals used for measurements. The GPRS network was able to support the maximum number of time slots defined by the terminal multislot class. All terminals are forced to use CS-2 encoding, as it provides better throughput with only a small loss over CS-1 in error recovery. Usage of CS-3 and 4 is currently not possible due to capacity limitations at the Gb interface. The network uses unacknowledged LLC. The Van Jacobson header compression is disabled due to poor performance in presence of packet losses, high computing burden on the network and lack of support from terminals.

For end-to-end throughput measurements, we use a tool generating bulk transfers over TCP. For measuring latency, we use a standard ping program. The NetHawk tracing tool records data traffic and signaling messages at the Abis or Gb interface as shown in Figure 3. Using an engineering mode available in some terminals it is possible to see an identifier of the serving cell, as well as force a cell reselection to one of surrounding cells. Finally, we use tcpdump to record TCP traces at the end hosts.

IV. MEASUREMENT RESULTS

A. Throughput, Latency and Buffering

Figure 5 and 6 show downlink and uplink throughput measured with four different terminals. In addition to minimum, average and maximum throughput observed over 40 replications, graphs also show the line rate computed based on the available number of timeslots and coding scheme, and the maximum TCP throughput taking into account TCP/IP header overhead. The multislot class of the terminal chiefly determined the throughput; the data rate per slot was approximately the same for all terminals. In downlink the maximum measured value is 43 kbps achieved by T280 terminal using four timeslots. In uplink, the maximum value is 21 kbps by N8310 using two timeslots. This is a notable improvement over earlier measurements [2]; at that time the maximum downlink throughput was 20 kbps and only 7 kbps in uplink.

In general, setting a larger IP MTU at the end hosts resulted in higher throughput. For instance, increasing the MTU from 576 bytes up to 1480 bytes improves throughput by one percent due to reduced TCP/IP header overhead. However, the MTU of 1500 bytes gave slightly lower throughput than of 1480 bytes. Tracing at the Gb interface showed inefficient fragmentation at the LLC layer. The maximum LLC frame size was configured to 500 bytes, thus 1500-byte IP packets were fragmented into four frames, with the fourth frame only a few bytes long. Sending plenty of small frames reduces efficiency due to higher header overhead. In the acknowledged LLC mode, using a smaller frame size than 1556 bytes can be beneficial by achieving finer grain retransmissions. However, for the unacknowledged mode we do not see a compelling reason to reduce the maximum LLC size.

We measured RTT of a GPRS link using 32-byte pings. RTT varies depending on the terminal and a serving cell in the range of 500-1100 ms on the unloaded link. A typical value is 700 ms. The minimum RTT improved since our earlier measurements [2] approximately by 200 ms. Interestingly, we observed regular oscillations in RTT when every second ping gets roughly 100 ms higher RTT than the other. This effect seems to relate to radio resource allocation as explained in the next section.

To estimate the buffer size of the GPRS link, we started a bulk TCP transfer with a sufficiently large window (200KB) to overflow the bottleneck buffer. The amount of outstanding data when the first loss occurs reflects the size of a drop-tail buffer. Based on specifications, we expected to see an approximately 10 kilobytes buffer. However, measurements indicate the downlink buffer of 50 kilobytes. Apparently, GPRS implementations include additional backlog buffers not present in standards. The per-user buffer size for GPRS downlink is optimally 5-10 packets since the bandwidth-delay product of a GPRS link does not exceed 5 kilobytes.

The uplink buffer measurement indicated the buffer size in terminals in the range of 3 to 30 kilobytes. A terminal (not listed in Figure 4) having only a 3-kilobyte buffer showed throughput of one-third of other terminals for an uplink bulk transfer. The reason was in repeating TCP retransmission timeouts. TCP needs at least three buffers in the network to utilize the fast retransmit algorithm [10]. On the other hand, the buffer of 30 kilobyte is excessive as it allows for unacceptably high link round-trip time and unnecessary delivery of data from aborted TCP connections.

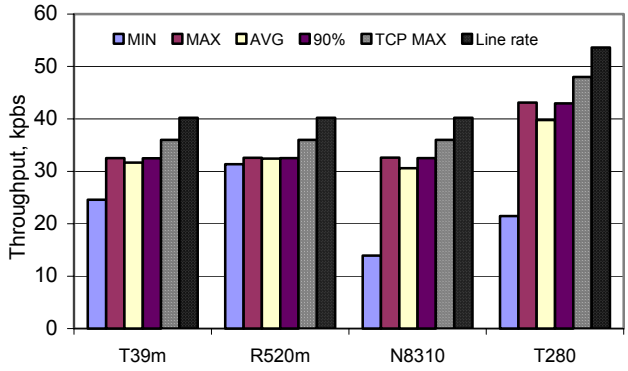


Figure 5. Bilk TCP throughput in GPRS downlink.

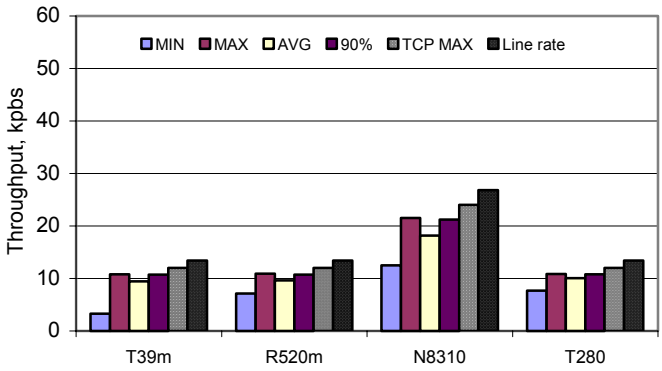


Figure 6. Bulk TCP throughput in GPRS uplink.

B. Tracing at the RLC Layer

Tracing at the Abis interface illustrates several interesting details on functioning of the RLC protocol. The first problem is allocation and release of TBF. According to release 97 specifications, TBF should be turned down immediately when the data buffer empties. Such a policy increases TCP RTT since every segment and ACK may trigger setup of a new TBF. Keeping TBF for longer periods has been suggested [7] and is reflected in Enhanced GPRS specifications [5]. The extended TBF release decreases RTT seen by TCP by more than a hundred milliseconds and reduces the signaling load. However, BSC is unaware if MS with an active TBF has any data to transmit and has to schedule also idle MSs, which wastes radio resources. Furthermore, the number of simultaneous TBFs is limited and postponing the TBF release can prevent data transmission by other MSs. Figure 7 shows an RLC trace of a downlink TCP transfer. In uplink, TCP ACKs are sent in groups of two on separate TBFs. The graph also shows an increase in TCP RTT caused by signaling to set up TBF for every TCP ACK in uplink. The network transmits dummy RLC blocks downlink to keep up TBF for instance at 11.5-12 s.

Another interesting case is premature retransmissions when there are unacknowledged blocks at the RLC layer but no new blocks to transmit. The RLC sender retransmits unacknowledged blocks in round robin until an ACK is received [8]. It can be seen in Figure 7 for instance at 10.7 s. On one hand, it increases the probability of data blocks to get through the radio link. On the other side, it may waste radio resources and battery power of MS. However, MS has no knowledge whether there is new data in BSC to be sent and therefore has to decode assigned timeslots anyway. It consumes the battery power as well. Avoiding such retransmissions when other users have data to transmit would prevent waste of radio resources.

At times the RLC sender retransmits lost blocks several times unnecessarily. It happens when the RLC receiver generates several ACKs before the first retransmission has arrived to it. Such ACKs will indicate the same lost segments and therefore can trigger unnecessary retransmissions at the RLC sender. A timer at the RLC receiver could prevent repeated retransmission of blocks approximately for one RLC RTT [8]. This gives enough time for the first retransmission to arrive and be acknowledged. Alternatively, the receiver can generate ACKs less frequently.

According to specifications, BSC schedules ACKs in uplink and downlink "when needed" [5]. For instance, BSC in Figure 7 requests an ACK for every tenth block. Less frequent ACKs preserve radio resources and battery power, but increase probability of stalling the window. We suggest that BSC during uplink transfers sends a selective ACK immediately or shortly after a missing block is detected. On the other hand, when all blocks are received correctly, infrequent ACKs suffice. In downlink transfers, BSC can poll MS for ACKs more frequently when the link quality drops down.

C. Mobility Measurements

We measured frequency, length and data loss of cell reselections in a live GPRS network. In the test network we recorded multi-layer traces of cell reselections.

Tests in the live network were performed while driving in downtown Helsinki. Cell reselections occurred at irregular intervals on the average every 40-70 s. The interval depends on the route and speed, but cell reselections occurring even in stationary conditions are not uncommon. Cell reselections suspended the data transfer by 3 to 15 s with most cases below 5 s. There were a few exceptions when a failed cell reselection made the link unusable for two minutes. By examining receiver TCP traces it is possible to calculate the number of lost segments during cell reselections. In downlink direction a large number and sometimes all outstanding packets were lost. However, we also observed cases where data segments were not lost but just delayed. In uplink, data loss was less common.

In the test network we traced LLC and TCP layers during cell reselections with and without a routing area update. At the LLC layer cell reselections cause a delay spike in uplink and downlink transfers of 2-4 s with cell update procedure only. The delay increases to 4-5 s when a routing area update is also triggered. We recorded signaling messages when a routing area update was performed between BSCs from two different vendors. The message exchange lasted typically less than two seconds.

During cell reselections about ten TCP segments were lost in downlink and none or one segment in uplink. The difference is due to the fact that in uplink transfers buffered data can be easily sent in the new cell there as in downlink direction data

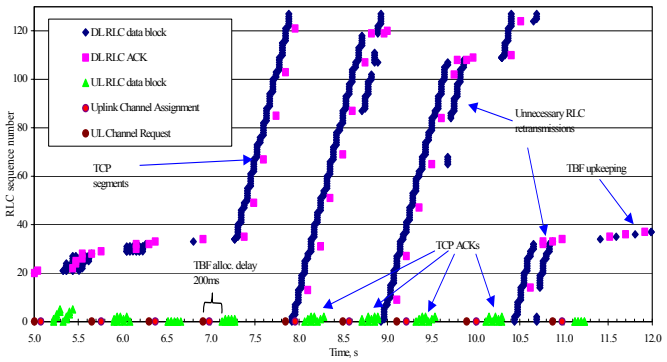


Figure 7. Interactions of TCP with channel allocation at the RLC layer.

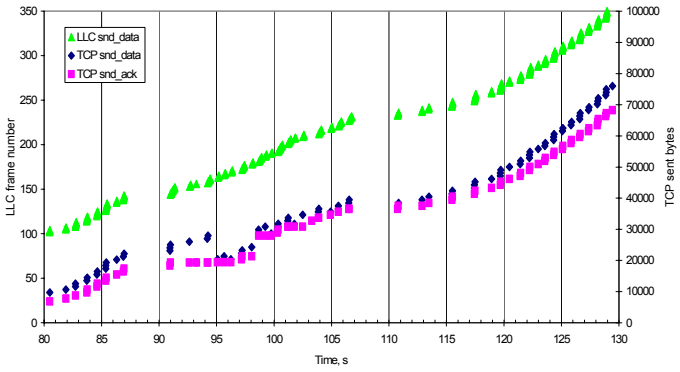


Figure 8. Effect of cell reselections on TCP and LLC in a downlink transfer.

need to be transferred to a new cell, which is rarely done. Therefore, in downlink cell reselections are seen by TCP as loss bursts that can cause lengthy timeouts and underutilization of the radio link in a new cell. On the contrary, for uplink transfers cell reselections are seen by TCP as delay spikes that can cause spurious timeouts [6]. Figure 8 illustrates two cell reselections during a downlink transfer. At the LLC layer a visible break due to cell reselection is approximately 5 s. However, it takes 5 to 10 s. more for TCP to retransmit lost segments.

We found three implementation problems in the mechanism of cell reselection. First, one or two LLC frames sent in the old cell before a cell reselection were unnecessary retransmitted in the new cell due to a bug in MS. Second, SGSN sent the FLUSH-LL message to the *new* cell instead of the old cell causing waste of resources in the old cell and loss of data in the new cell. Third, dummy LLC frames caused an outage in the LLC data transmission. Dummy frames are required for completing the cell reselection in case when no data is available for transmission in MS.

V. Conclusions and Future Work

We measured performance of GPRS in stationary and mobile operation. The maximum downlink TCP throughput was 43 kbps and 21 kbps in uplink. The typical RTT of the unloaded link is around 0.7 s. We have estimated a 50 kilobytes downlink buffer available for a single GPRS user. It exceeds the optimal value by several times and allows for undesired effects such as inflated RTT and delivery of stale data. On the other hand, one terminal had only a two-packet buffer in the uplink direction and showed throughput of one-third of the normal. TCP needs at least three buffers per connection for efficient loss recovery.

By combining end-to-end tracing with tracing performed within the network, we observed several undesired cross-layer interactions between RLC, LLC and TCP. In particular, the slow start phase of TCP and delayed ACKs interact badly with radio resource allocation at the RLC layer. At the LLC layer, mismatch between the maximum size of data units in LLC and in TCP results into inefficient fragmentation. Finally, we showed situations when RLC retransmits data unnecessary.

While driving in an urban area we observed cell reselections to occur roughly every minute and to last for five seconds. In downlink, most of outstanding data gets lost during a cell reselection. It takes 5 to 10 s. for TCP to recover lost data. In the uplink, cell reselections often do not cause data loss, but instead are seen as a delay spike by the upper layers. In this case, TCP can experience a spurious timeout and retransmit the outstanding data unnecessarily [12].

Our future work on GPRS will include testing network-controlled cell reselections, measuring throughput and battery consumption under varying radio conditions and network load. We also plan to evaluate performance of real-time streaming applications and quality of service mechanisms.

Acknowledgements

We thank Roger Kalden, Michael Meyer, Janne Peisa, Hannes Ekström and Reiner Ludwig for useful comments that improved the paper significantly.

References

[1] 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.

[2] J. Korhonen, O. Aalto, A. Gurtov, H. Laamanen, Measured Performance of GSM HSCSD and GPRS, IEEE ICC, June 2001.

[3] M. Meyer, TCP Performance over GPRS, IEEE WCNC, September 1999.

[4] Ho, J.; Zhu, Y.; Madhavapeddy, S., Throughput and buffer analysis for GSM General Packet Radio Service (GPRS), IEEE WCNC, vol. 3, 1999.

[5] 3GPP TS 04.60 V8.6.0 Mobile Station (MS) - Base Station System (BSS) interface; Radio Link Control/ Medium Access Control (RLC/MAC) protocol (2000). Release 99.

[6] 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, May 1999.

[7] H. Wiemann, A. Schieder, H. Ekström, Enhanced TBF Features in GERAN, WPMC'01, Aalborg, Denmark, Septemeber 2001.

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

[9] J. Sau, C. Scholefield, Scheduling and quality of service in the General Packet Radio Service, IEEE ICUPC, 1998.

[10] W. Stevens, TCP/IP Illustrated, Volume 1; The Protocols, Addison Wesley, 1995.

[11] M. Passoja, Effects of Cell Reselection to the Performance of GPRS, Diploma Thesis, University of Oulu, September 2001.

[12] A. Gurtov, R. Ludwig, Evaluating the Eifel Algorithm for TCP in a GPRS network, In Proceedings of European Wireless, Florence, Italy, February 2002.

Responding to Spurious Timeouts in TCP

Andrei Gurtov
University of Helsinki
Finland

Reiner Ludwig
Ericsson Research
Herzogenrath, Germany

Abstract - **Delays on Internet paths, especially including wireless links, can be highly variable. On the other hand, a current trend for modern TCPs is to deploy a fine-grain retransmission timer with a lower minimum timeout value than suggested by RFC2988. Spurious TCP timeouts cause unnecessary retransmissions and congestion control backoff. The Eifel algorithm detects spurious TCP timeouts and recovers by restoring the connection state saved before the timeout. This paper presents an enhanced version of Eifel response and illustrates its performance benefits on paths with a high delay-bandwidth product. The refinements concern the following issues (1) an efficient operation in presence of packet losses (2) appropriate restoring of congestion control (3) adapting the retransmit timer to avoid further spurious timeouts. In our simulations applying the Eifel algorithm on paths with a high delay-bandwidth product can increase throughput by up to 250% and at the same decrease the load on the network by 3%. Eifel also shows adequate performance on heavily congested paths.**

I. Introduction

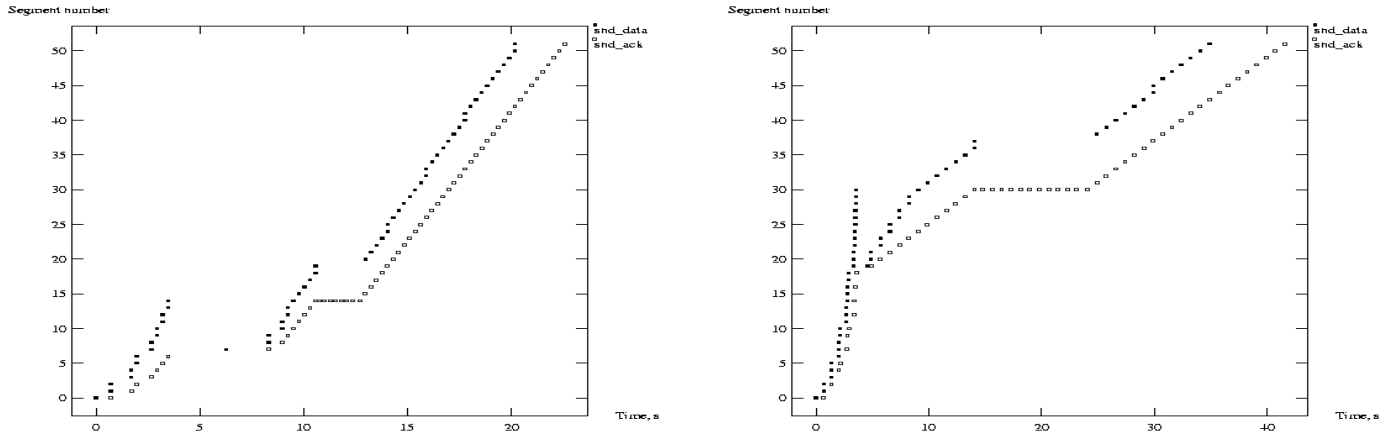
Recent measurement studies [9] show that TCP [36] maintains its position as the dominant transport protocol in the Internet. TCP is a stable, mature, and probably the most thoroughly tested protocol of its kind. Nevertheless, there are some corner cases where TCP could still be improved. The problem of *spurious timeouts*, i.e., timeouts that would not have occurred had the sender waited "long enough", is an example of the above mentioned corner cases.

Internet measurements show highly variable delays on some paths resulting for example from route flipping [2][3][7]. Another measurement study reports occasional delay jitter of several seconds on slow dial-up connections due to link-layer error recovery by a modem [28]. Especially on wireless links mechanisms such as error recovery, mobility, on-demand resource allocation and priority scheduling can cause high delay variation [18]. For example, we measured delay spikes in the order of several seconds occurring frequently in a wireless cellular network due to cell changes [27][19]. Another study reports abrupt changes in link RTT resulting from on-demand allocation of a high-speed radio channel [44]. Mobile users start utilizing heterogeneous overlay networks for Internet access. Currently inter-network handovers are extremely challenging in terms of delay jitter and data loss, and it seems unlikely that such disruptions can be completely avoided in the near future [42].

A sudden delay increase can cause a spurious TCP timeout which presents two problems. First, outstanding segments are retransmitted unnecessarily. Second, the congestion control [22][1] is falsely triggered. The Eifel algorithm suggested in [29] uses the TCP timestamp option [23] to reliably detect spurious timeouts and prevent these two problems.

The role of recovering from spurious timeouts is growing as many modern TCPs, for instance Linux 2.4, start to use a finer-grain timer and a low minimum retransmit timeout value (e.g. 200 ms) [38]. The recent stable kernel release of Linux 2.4 includes the Eifel algorithm. At the same time Eifel is advancing through the standardization process in the IETF [31]. Therefore, it is important to assure that the algorithm is efficient and safe for wide deployment in the Internet. The goal of this paper is to refine the response part of the algorithm and to demonstrate its utility in the environment of a high delay-bandwidth product. We show that Eifel can potentially have significant performance benefits for TCP that justifies efforts and additional complexity in its development in order to produce a well-working and robust solution to the problem of spurious timeouts.

The rest of the paper is organized as follows. In Section 2 we review the problem of spurious timeouts, the Eifel algorithm and related work. In Section 3 we motivate and explain modifications to the Eifel response algorithm. Section 4 describes our modelling approach and evaluates the new response for paths with a high bandwidth-delay product. Finally, Section 6 presents conclusions and plans for the future work.



(a) Triggered by a delay spike.

(b) Triggered by a bandwidth change.

Figure 1. Spurious retransmission timeouts in TCP. SACK and timestamps are enabled.

II. Spurious Timeouts in TCP

In this section, we provide a detailed description of how spurious timeouts affect TCP's protocol operation. We assume that the reader is familiar with the basics of TCP [39].

A. Definition of a Spurious Timeout

A *retransmission timer* is a prediction of the upper limit of the RTT. In common TCP implementations, an adaptive retransmission timer accounts for RTT variations [22]. A spurious timeout occurs when the RTT suddenly increases, to the extent that it exceeds the retransmission timer that had been determined a priori. RTT can quickly return back to normal for example in case of a handover-triggered delay spike. RTT stays high when available bandwidth of the bottleneck link has suddenly decreased.

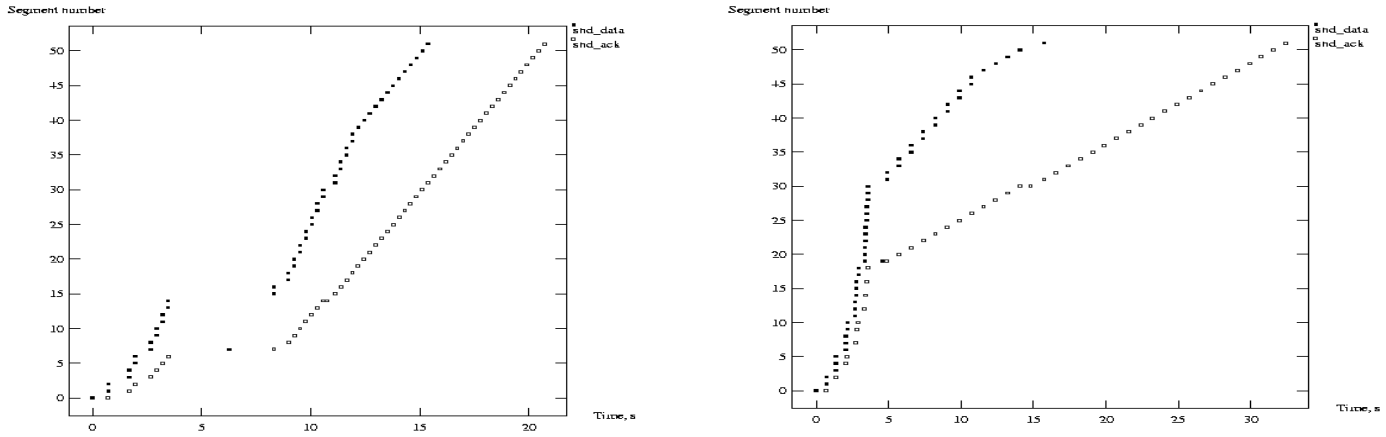
On a spurious timeout TCP assumes that all outstanding segments are lost and retransmits them unnecessarily as shown in Figure 1 (a) (the trace is recorded in NS2 over a 30 kbps link). It was shown that the go-back-N retransmission behaviour triggered by spurious timeouts has a root: the retransmission ambiguity [25], i.e., a TCP sender's inability to distinguish an ACK for the original transmission of a segment from the ACK for its retransmission. Shortly after the timeout ACKs for the *original* transmissions return to the TCP sender. On receipt of the first ACK after the timeout, the sender must interpret this ACK as acknowledging the retransmission, and must assume that all other outstanding segments have also been lost. Thus, the sender enters the slow start phase, and retransmits all outstanding segments in this fashion. The go-back-N retransmission triggers the next problem: the receiver generates a DUPACK for every segment received more than once. The receiver has to do that because it must assume that its original ACKs had been lost. This may trigger a spurious fast retransmit at the sender.

Another major problem is a distortion of congestion control. On one hand, the congestion window and slow start threshold are reduced after a spurious timeout. The reduction is unnecessary as no data loss has yet been detected that would indicate congestion in the network. On the other hand, TCP makes an assumption that all outstanding segments were lost and left the network. In fact, they are probably still located in the bottleneck queue. Therefore, go-back-N retransmissions performed in slow start lead to aggressive sender behaviour. That is, while the original transmissions are draining from the queue, the retransmissions get sent at twice the link rate (assuming the receiver generates an ACK for each segment). This behaviour violates the 'packet conservation' principle [22] and can cause *real* packet losses due to congestion [29]. After a spurious timeout TCP should behave in a way that does not cause short-term congestion and does not underutilize the link in the long run.

Figure 1 (b) shows a spurious timeout resulting from a bandwidth change. The available bandwidth of a bottleneck link is reduced from 300 kbps down to 10 kbps. Such rapid bandwidth changes can occur due to on-demand allocation and release of a high speed radio channel [26]. The link RTT is increased from less than 100 ms to almost 2 s which causes a spurious TCP timeout. The sender's response is largely the same as in case of a delay spike.

B. The Eifel Algorithm

Eliminating the retransmission ambiguity requires extra information in ACKs that the sender can use to unambiguously distinguish an ACK for the original transmission of



(a) Triggered by a delay spike.

(b) Triggered by a bandwidth change.

Figure 2. TCP sender's response to a spurious timeout with the Eifel algorithm.

a segment from that of a retransmission. The TCP timestamp option provides exactly what we need. When using the timestamp option the TCP sender writes the current value of a "timestamp clock" into the header of each outgoing segment. The receiver then echos those timestamps in the corresponding ACKs according to the rules defined in [23]. Eliminating the retransmission ambiguity is then implemented as follows. The sender always stores the timestamp of the *first* retransmission triggered by an expiration of the retransmission timer. In our implementation, we call that timestamp *ts_first_rexmit*. Then, when the first ACK that acknowledges the retransmission arrives, the sender compares the timestamp of that ACK with ts_first_rexmit. If it is smaller than ts_first_rexmit, this indicates that the retransmission was spurious.

A case when a timeout occurs due to lost ACKs has been a subject of some discussion. When receiving a duplicate segment below the cumulative ACK some TCPs update a cached timestamp [8], and some do not [23]. If a TCP sender receives the timestamp from the original segment after a timeout, it deduces that the timeout was spurious. Therefore, if the receiver echoes the original timestamp in response to duplicate segments as the current standard defines [23], then a timeout due to lost ACKs is considered spurious. If the receiver echoes timestamps from retransmissions, the timeout is not considered spurious. We believe that timeouts due to lost ACKs should be considered spurious. Restoring the congestion control state in this situation is justified, since there is no loss and therefore no congestion on the forward path.

Including the 12 bytes TCP timestamp option field in every segment and ACK might seems heavy weight. The advantage of using the timestamp option is that this scheme is already a proposed standard and that it is widely deployed [3]. Existing TCP/IP header compression schemes [10] [24] do not support compression of TCP options, but there is ongoing work to enable it [21]. Furthermore, there is some evidence that timestamps are useful in general on bandwidth-limited paths [17].

C. *Related Work*

In [16] we evaluated performance of the original Eifel response [29] for a mobile user in a General Packet Radio Service (GPRS) wide-area cellular network [41]. We used simulation to obtain a controllable environment and reference TCP implementations. We also confirmed the results with smaller scale tests in a live GPRS network [18]. In the lossless scenario, the Eifel algorithm improved the performance for all TCP flavours under varying frequency of delay spikes. It reduced download times by up to 12 percent, and increased goodput by up to 20 percent. These gains are valuable on a slow GPRS link, but in relative numbers they are moderate. All the improvement is coming from eliminating duplicate delivery of packets, but not from unnecessary reduction of the congestion window. In this paper, we evaluate Eifel on high capacity links where unnecessary reduction of the congestion window has greater impact. Another result of [16] was that in a scenario with congestion losses Eifel can suffer from non-spurious timeouts, but using efficient loss recovery algorithms such as SACK [33][6] and Limited Transmit [4] improves the situation.

A study of a cdma-2000 wireless wide area network reports 'bandwidth oscillation' due to switching of a high-speed radio channel between several users [26]. The link bandwidth changes frequently between 300 kbps and 10 kbps increasing the link RTT beyond the estimate of the TCP retransmission timer and triggering spurious TCP timeouts. A further study reports that increasing the TCP

window helps to decrease the number of spurious timeouts [44]. It is achieved with larger network buffers and a larger TCP receiver window.

Eifel is not the only possible way to detect spurious timeouts. The response algorithm described in this paper can be applied also with other detection algorithms. For instance, a following heuristic was suggested in [2]. Whenever a TCP retransmits due to a timeout, it measures T, the time from the retransmission until the next ACK arrives. If T is less than the minimum RTT measured so far, then arguably the ACK was already in transit when the retransmission occurred, and the timeout was spurious. If the ACK only comes later than the minimum RTT, then likely the timeout was necessary.

Waiting for the receiver to signal in DUPACKs that is has correctly received duplicate segments, as proposed in [13], would be too late to prevent the unnecessary retransmissions during the go-back-N behaviour. However, this information can be used for restoring congestion control state afterwards and for adapting the retransmit timer.

A "Forward RTO Recovery" (F-RTO) algorithm [37] for recovering from TCP timeouts is a TCP sender only algorithm that does not require any TCP options to operate. After retransmitting the first unacknowledged segment triggered by a timeout, the F-RTO algorithm at a TCP sender monitors the incoming acknowledgements to determine whether the timeout was spurious and to decide whether to send new segments or retransmit unacknowledged segments. The algorithm starts by transmitting new segments after a timeout and reverts to standard go-back-N only if a DUPACK is received. Otherwise, the timeout is considered spurious and the sender continues transmitting new data. F-RTO cannot properly classify timeout under packet re-ordering [5] or when no new data is available for transmission. In such cases it uses the standard TCP behaviour.

The Eifel algorithm does not concern with spurious timeouts that occur during fast recovery. In [17] we propose restarting the timer on DUPACKs and using a method for recovering lost retransmissions as protection against spurious timeouts during a DUPACK series and in the recovery phase.

III. The Sender's Response

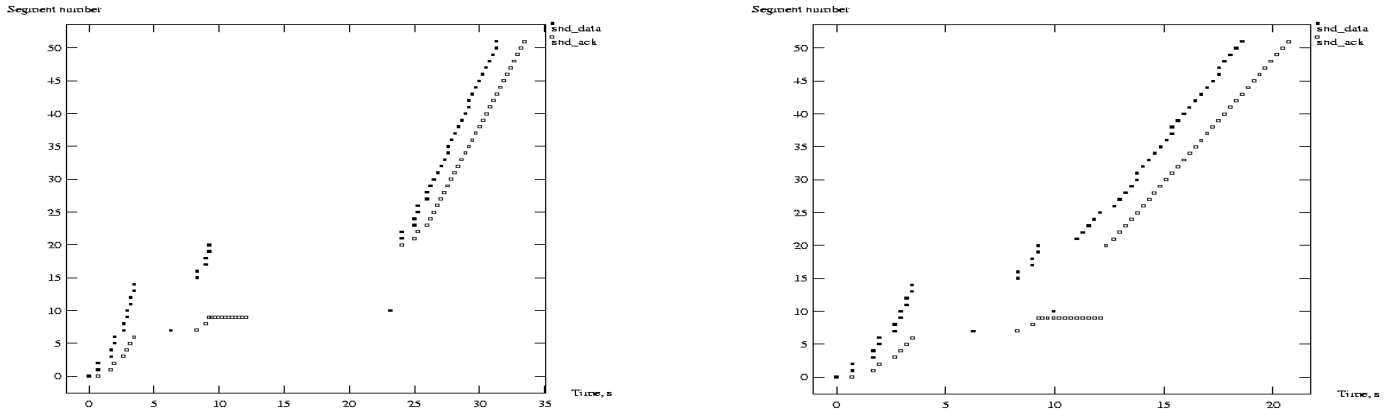
When a timeout occurs, the Eifel algorithm at the sender stores the current values of the slow start threshold and the congestion window. Upon detecting a spurious timeout, the sender can restore them and resume transmission with the next unsent segment as shown in Figure 2. This section defines enhancements to this basic response algorithm proposed in [29].

A. Efficient Recovery from Packet Losses

The original proposal simply specified that the transmission after detecting a spurious timeout always resumes with the next unsent segment [29]. This works fine when none of delayed segments are lost. In reality, delay spikes are often coupled with data losses, for instance during a handover [19]. In the extreme case, all but the oldest outstanding segment are lost. In such a situation, Eifel still detects a spurious timeout, while recovery using the standard go-back-N would be faster. Simply transmitting new data in this case leads to a second non-spurious timeout. However, it is difficult to select a transmit policy on a first ACK after a timeout since there is no information available on the amount of lost data. Therefore, we still believe in resuming transmission with the next unsent segment while relying on efficient loss recovery algorithms to cope with data losses.

In [16] we show that allowing fast retransmits for outstanding segments [12], using Limited Transmit [4] and SACK [33][6] largely solves the problem of poor performance with packet losses. In this section we start by illustrating these cases and suggest even more robust recovery methods.

A single lost segment. This simple case illustrates the response when one of delayed segments is also lost. Figure 3 (a) shows Reno that blocks fast retransmit until the recovery point (snd_max, the highest sequence number transmitted before the timeout) is acknowledged because the 'bug fix' is enabled [12]. TCP sender has to wait for a second non-spurious timeout to recover this lost segment. The RTO value is large as it is calculated from delayed segments (and even may still be backed-off). The reason to block fast retransmit until the recovery point is possibility of a DUPACK series from unnecessarily retransmitted segments during go-back-N. However, as transmission is resumed by the Eifel algorithm with the next unsent segment, there is no unnecessary retransmissions and thus a DUPACK series can only indicate a lost segment. There is no reason to block fast retransmit in such a case. As Figure 3 (b) shows, Reno successfully recovers from a lost segment with fast retransmit when the bug fix is disabled. Disabling bug fix applies also to NewReno and SACK



(a) Fast retransmit is prevented by the 'bug fix'.

(b) Fast retransmit is allowed when the 'bug fix' is disabled.

Figure 3. Response of TCP-Reno with Eifel to a spurious timeout. A single segment is lost.

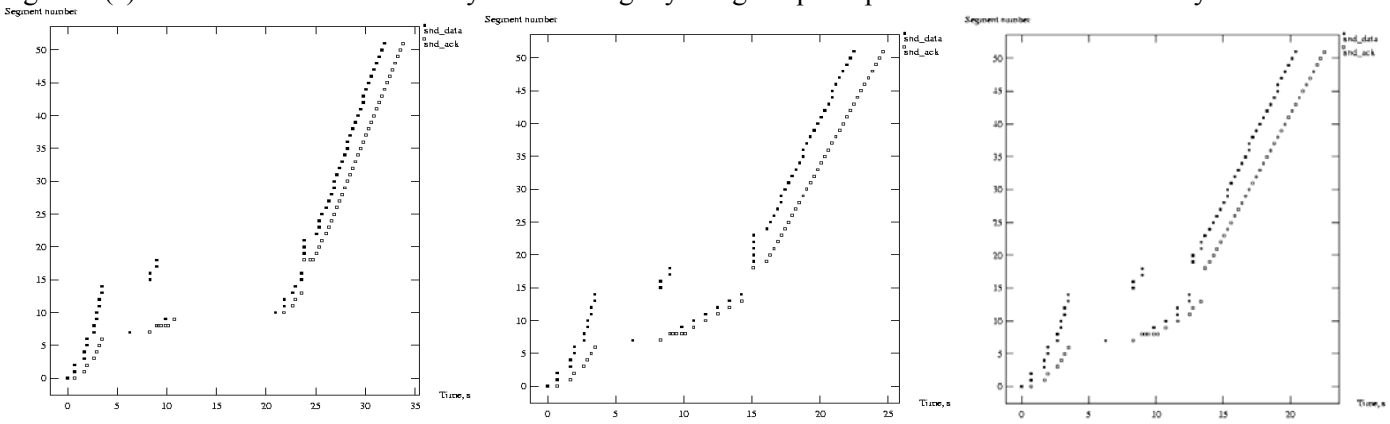
TCPs as they still need three DUPACKs to enter the fast recovery phase.

Loss of all but one segment. A worst-case spurious timeout occurs when all outstanding segments are lost except for the oldest segment that is delayed. We describe response of Reno, NewReno and Reno-SACK in this scenario. It is well recognized that Reno usually experiences a timeout when multiple segments are lost from the same window [11]. Reno with Eifel is not an exception; when several of delayed packets are lost, the timeout is inevitable. Figure 4 (a) shows response of Reno. Even with many lost segments the fast retransmit is still possible since new transmitted segments cause DUPACKs. Limited Transmit allows to trigger fast retransmit even with a flight size of one segment assuming no further losses. After a second, necessary timeout, Reno recovers other lost segments using go-back-N. Figure 4 (b) shows response of NewReno in the same scenario. NewReno recovers steadily one segment per RTT and avoids the second timeout. Figure 4 (c) shows Reno-SACK recovery which is slightly faster than NewReno. Thus, using the standard-track mechanisms it is possible to achieve an efficient operation in many cases.

Loss of segments and ACKs. There are although cases when Reno-SACK cannot recover without a necessary timeout. Such situations appear when there are large 'holes' in the receiver window or a few ACKs are lost. The sender cannot retransmits segments as it is limited by the congestion window. The Reno-SACK scheme is conservative because it considers segments reported missing by the receiver to be still outstanding in the network.

The Forward Acknowledgment algorithm [32], on the other hand, considers that missing packets left the network and can recover without a necessary timeout in such cases. FACK is not standardized by IETF due to concerns on operation in presence of packet re-ordering, but is used by some TCPs, for example Linux 2.4 [38].

Therefore, we studied if necessary timeouts for Reno-SACK could be avoided with a simple modification retaining the principles of conservative recovery. We found that

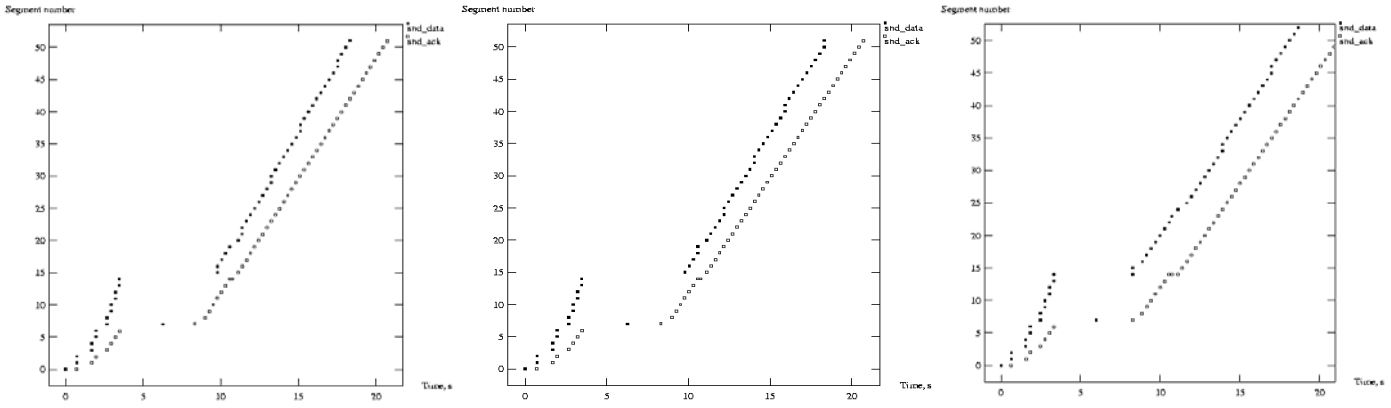


(a) Reno.

(b) NewReno.

(c) SACK.

Figure 4. Response of TCPs with Eifel to a spurious timeout. All of delayed segment but one are lost.



(a) ssthresh=cwnd_old/2, cwnd=ssthresh. (b) ssthresh=cwnd_old/2, cwnd=1. (c) ssthresh=cwnd_old, cwnd=ssthresh.

Figure 5. Restoring the congestion control state when a spurious timeout occurs in slow start.

a major part of necessary timeouts of Reno-SACK is due to lack of retransmissions on partial ACKs. NewReno retransmits a packet on every partial ACK and is widely used in the Internet [34]. We combined the NewReno and SACK so that the sender always retransmits a packet on a partial ACK. These retransmissions are accounted into the pipe estimate and therefore in the long run the NewReno-SACK sender retains fairness while avoiding retransmission timeouts more effectively. Although we believe it is a safe modification for general use, as an extra precaution it is possible to enable it only after a spurious timeout and disable it when the recovery point (snd_max) is acknowledged.

B. Restoring the Congestion Control State

In section Section 2.A we described problems with congestion control experienced by conventional TCP after a spurious timeout. We explain how these problems are resolved by Eifel.

The original paper [29] proposed the following options for restoring of the congestion control state:

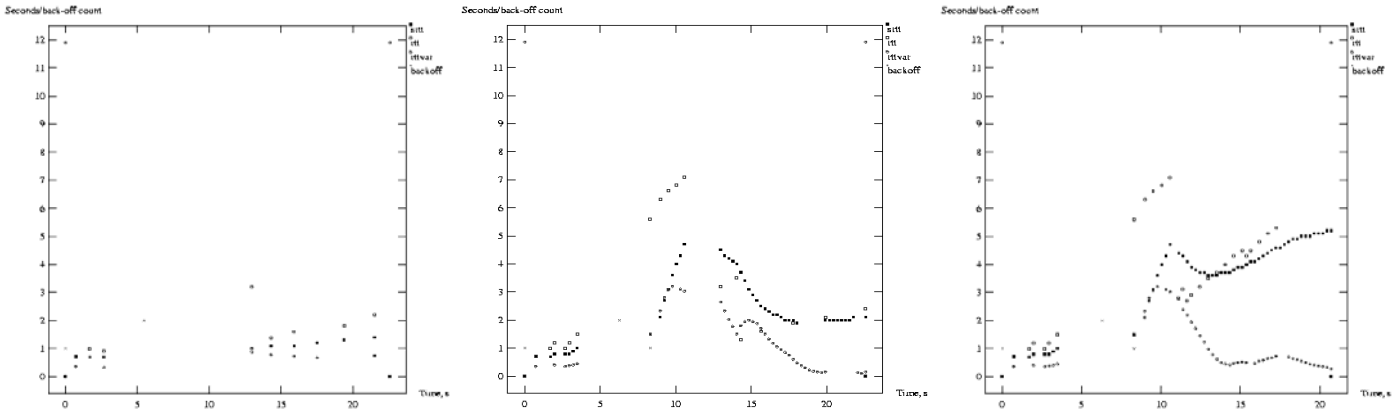
1. ssthresh=ssthresh_old, cwnd=cwnd_old
2. ssthresh=cwnd_old/2, cwnd=ssthresh
3. ssthresh=cwnd_old/2, cwnd=1

The first option, complete restoration is achieved by setting both the slow start threshold and the congestion window to values stored before the timeout. The second option, partial restoration is achieved by setting the slow start threshold to half of congestion window before the timeout as done normally by TCP. However, instead of leaving the congestion window at one segment after the timeout, it is set to the new value of the slow start threshold. Therefore, the connection always continues in congestion avoidance in this case. The third option is not to restore the congestion control state, i.e. leave the slow start threshold at half of the old congestion window and the congestion window at one segment.

The first option is used only after a single spurious timeout. The second option is used after two subsequent timeouts, and the third option, which is the default TCP behaviour, is used after three or more timeouts. However, we have not found any practical evidence that the length of a delay reflects the change of characteristics in the network.

A question was raised whether restoring the congestion control state after a spurious timeout can cause undesirable bursty TCP behaviour. This is not the case because at the same time the Eifel algorithm resumes transmission with the next unsent segment (snd_nxt=snd_max) which also restores the estimate of the flight size [1][43]. As TCP only transmits data when the congestion window is greater than the flight size, no burst is produced when both parameters are restored from the equilibrium state. However, we emphasize that TCPs such as Linux 2.4 [38] which determine the flight size in a different way than BSD must explicitly restore it after a spurious timeout.

Lack of restoring of the slow start threshold can lead to severe unnecessary underutilization of a link if a spurious timeout occurs in an early phase of slow start. Additionally, we found that options 2 and 3 are prone to timeouts. If the congestion window is not restored fully, the sender cannot transmit on returning ACKs as shown in Figure 5 (a) and (b) because the flight size estimate is larger than the congestion window. Experiments in Section 4 indicate that with options 2 and 3 the TCP sender is prone to non-spurious timeouts. This decreases throughput and actually increases the load on the network due to a greater number of unnecessary retransmissions. Given this fact and that full restoring of the congestion control state does



(a) Reno. (b) Reno with timestamps. (c) Reno with timestamps and Eifel.

Figure 6. RTO dynamics after a spurious timeout.

not cause bursts, applying the option 1 seems to be an attractive choice.

We did not find much difference between the option 2 and 3 for links with a moderate delay-bandwidth product. With the option 2 the sender continues in congestion avoidance incrementing the congestion window only by a single segment per window. With the option 3, the sender is in slow start and increments the congestion window by one segment per ACK. Therefore, it reaches the same size as used by the option 2 quickly.

We suggest a forth option to restore the congestion control state

4. ssthresh=cwnd_old, cwnd=ssthresh,

where the slow start threshold is set to the old value of the congestion window, and the congestion window is fully restored. This allows the sender to immediately resume transmission on ACKs as shown in Figure 5 (c). However, the sender is forced to continue in congestion avoidance which may lead to underutilization on high-delay bandwidth paths. A variation of this approach would restore the slow start threshold but only when no loss has yet been detected during the connection.

The response when a spurious timeout occurs in congestion avoidance is similar. The sender resumes in congestion avoidance which prevents a short-term congestion problem caused by slow start go-back-N retransmissions in conventional TCP.

C. Adapting the Retransmit Timer

With traditional TCP, a sender that uses a too aggressive retransmit timer has to pay the price (i.e. slow down) after a spurious timeout. Presumably this discourages developing too aggressive retransmission timers and preserves the network from duplicate retransmissions that do no useful work. Therefore, some modification to the retransmit timer that makes it more conservative after a spurious timeout is needed. This section discusses various approaches to adapting the retransmit timer after a spurious timeout. Note, that in order to increase conservativeness of the retransmit timer, the TCP sender must be robust to packet losses. Otherwise, the sender will suffer excessively from waiting for necessary timeouts. TCP-FACK and NewReno-SACK suggested in Section 3.A seem to be sufficiently robust to packet losses.

Figure 6 shows RTO parameters of TCP-Reno, TCP-Reno with timestamps, and TCP-Reno with timestamps and the Eifel algorithm after a spurious timeout. The RTO parameters of Reno without timestamps remain nearly at the same level as before the timeout. In other words, TCP does not learn much from a delay spike. This situation is explained by the Karn's algorithm as collecting of RTT samples from retransmitted segments is denied due to the retransmission ambiguity problem [25]. Therefore, during the go-back-N behaviour no RTT samples can be collected, but RTO is kept backed off. A spurious fast retransmit present in some TCPs after go-back-N can still delay obtaining a valid RTT sample. Once a new RTT sample is collected, SRTT and RTTVAR are recalculated from the new sample and the back off counter is reset. The RTO value basically returns at the level before the delay spike. Note, that timing every segment without the timestamp option does not improve the situation, as delayed segments still cannot be timed due to retransmission ambiguity.

Figure 6 (b) shows behaviour of RTO parameters for Reno with timestamps. It is less aggressive than RTO computed without timestamps due to delayed segments used for RTT sampling. Immediately after a timeout when original ACKs are arriving, the RTO becomes very high. A pause in the graph between 10-13 s is due to arriving DUPACKs which cannot be used for RTT sampling [23]. RTO stabilizes at the new level approximately 10 s after

the spurious timeout. This might be a too quick decay to protect the sender from spurious timeouts in the future. Making SRTT and RTTVAR weights adaptive to the frequency of RTT sampling as suggested in [30] can solve this problem.

Figure 6 (c) shows the RTO dynamics of TCP-Reno with Eifel. The timer naturally uses timestamps since they are required for the Eifel detection algorithm. Already this fact makes TCP with Eifel more conservative than widely used Reno without timestamps. Furthermore, no idle period in RTO updates due to DUPACKs such as in Figure 6 (b) is present here. Also, the RTO with Eifel does not decay as quickly after the timeout. This is because Eifel restores the congestion control state and gets more data outstanding in the network. Higher RTT in such a case makes the timer less prone to spurious timeouts [26].

In summary, using a conservative RTO such as suggested in [35] with timestamps provides a sufficient protection against excessive spurious timeouts in many cases. That timer is restarted on ACKs and is limited to the minimum RTO value of 1 s.

Further adapting the timer may include the following options:

1. Re-seed RTO after a spurious timeout
2. Reset the back-off counter only on a necessary timeout
3. Increase the minimum RTO

The first option is implemented by using the first RTT sample obtained with timestamps from delayed segments to re-initialize SRTT and RTTVAR variables and restart the timer. The second option is to keep the back-off counter at the level set during a spurious timeout and reset it back only on a necessary timeout. The third option is to perform additive increase of the minimum RTO value on each spurious timeout and reset it back to the default value on a necessary timeout.

IV. Performance Evaluation

A. Methodology

We choose simulation to have a reproducible and controllable environment with reference TCP implementations. A simple 'dumb bell' topology has a bottleneck link with 2 Mbps and with high latency of 150 ms. Such characteristics are typical for satellite links and third generation wireless wide area networks [41]. In all measurements the TCP timestamp option [23] was enabled and the MSS was 1000 bytes. The receiver advertised a window of 150000 bytes and the bottleneck queue was Drop-Tail with 75 buffers. We used one-way models of Reno, NewReno, Reno-SACK and FACK TCP with delayed acknowledgments. We implemented our modified Eifel response algorithm and made it available at [20].

Figure 7. Measurement setup in NS2.

We also implemented a tool to trigger delay spikes in both directions and after the queue. The length of delay spikes is uniformly distributed between 3 and 15 s; they occur at the interval of 20-40 s. Shorter delay spikes would suffice to trigger spurious timeouts in our tests, but we decided to use the typical values experienced by a cellular network user driving in an urban area [19]. In all tests a conservative timer with a minimum limit on RTO of 1 s [35] is applied. In one test, the minimum RTO value is increased by 1 s after each spurious timeout and reset back to 1 s after a necessary timeout. If not mentioned otherwise, the Eifel algorithm restores the congestion control state fully.

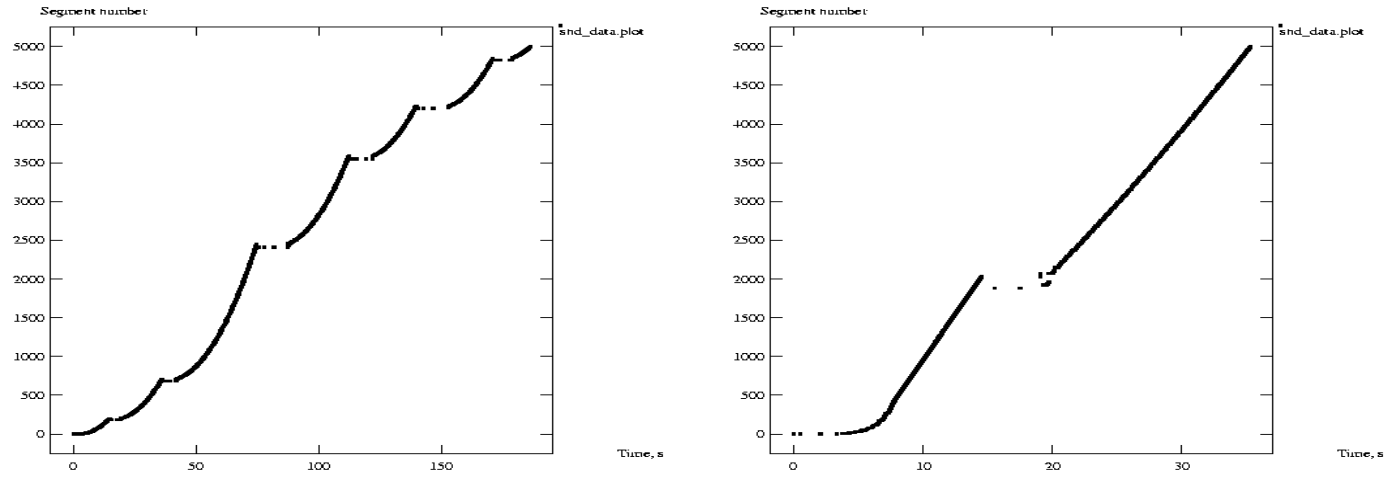
In the first set of tests we use a single TCP connection transferring 5 MB of data. In the second set of tests we also add a competing constant bit rate flow running at 1 Mbps. It congests the link especially during delay spikes.

TCP studies such as presented in [32] are typically limited to qualitative analysis that shows the behaviour of only a few TCP connections. We reuse transport agents in NS2 which allows to run many test repetitions in a reasonable time and provide quantitative results. Values given in the next section are averages over 100 repetitions.

B. Results

For mobile users and operators the battery power consumption and radio resource preservation are often as important as the throughput across the wireless link. We therefore used throughput (download times) and goodput (number of segments) as equally important performance metrics. We also give the average number of spurious and necessary timeouts for each connection to indicate how susceptible a TCP modification is to timeouts.

In the first test, we use Reno-SACK over a link without other traffic. Table 1 show results with and without Eifel. Applying the Eifel algorithm gives 254% increase in throughput and at the same time requires 3% less segments to complete the connection. Most of improvement in



(a) Reno-SACK.

(b) Reno-SACK with Eifel.

Figure 8. Effect of Eifel on Reno-SACK on a uncongested link.

throughput comes in this case from restoring of the congestion control state after spurious timeouts. Figure 8 (a) shows that Reno-SACK reduces the congestion window and performs go-back-N on every spurious timeout. Enabling the Eifel algorithm in Figure 8 (b) allows the connection to increase the congestion window until a segment loss is detected at 20 s. The number of spurious timeouts is decreased due to shorter connection lifetime.

TABLE 1. EFFECT OF EIFEL ON RENO-SACK ON A UNCONGESTED LINK.

TCP	Eifel	Time, s	Segments sent	Spurious RTOs	Necessary RTOs
Reno-SACK	Off	138	5234	4.68	0.00
Reno-SACK	On	39	5088	1.37	0.03

Table 2 shows results for Reno-SACK over a congested link with the same delay jitter model. It is a worse-case scenario for Eifel, as often all segments but one are lost during a delay spike. The standard go-back-N recovery would recover lost segments faster in such a case. The Eifel retransmits lost segments using the SACK recovery phase. The congestion window and slow start threshold are fully restored after a spurious timeout, but are reduced again upon detecting of a packet loss.

Reno-SACK with Eifel has 73% longer download time in this case due to a large number of necessary timeouts. Timeouts typically occur when the TCP sender enters the fast recovery phase but cannot retransmit lost segments due to large 'holes' in the receiver window. Figure 9 (a) shows three such timeouts at 100 s, 200 s, and 250 s. NewReno-SACK corrects this problem by recovering at least one segment per RTT and allows Eifel to achieve higher throughput and goodput. In Figure 9 (b) no necessary timeouts are present. TCP-FACK with Eifel achieves 43% reduction in download time over Reno-SACK even in such harsh conditions. Figure 9 (c) shows that FACK avoids necessary timeouts and recovers from packet losses faster than NewReno-SACK.

TABLE 2. EFFECT OF EIFEL ON TCPS OVER A CONGESTED LINK.

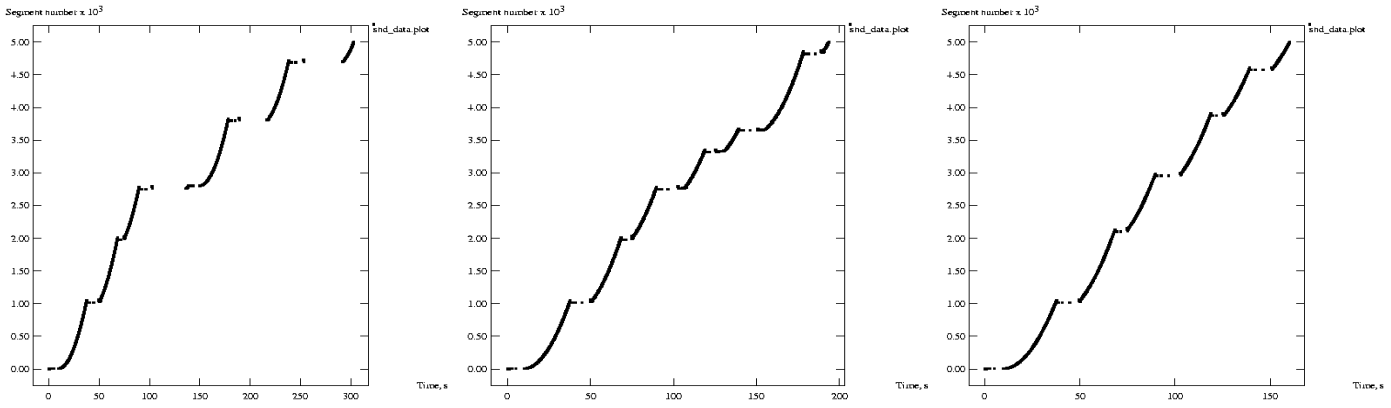
TCP	Eifel	Time, s	Segments sent	Spurious RTOs	Necessary RTOs
Reno-SACK	Off	191	5251	5.79	0.68
	On	331	5237	6.02	4.98
NR-SACK	Off	191	5251	5.78	0.69
	On	146	5192	4.35	0.57
FACK	Off	191	5251	5.74	0.70
	On	108	5225	3.24	0.38

Table 3 shows performance of FACK with Eifel under different options (described in Section 3.B) of restoring the congestion control state. Options 2 and 3 perform poorly in terms of throughput and goodput. Not only the download time is several times higher than for the option 1, but also more unnecessary retransmissions are sent wasting the network capacity. Therefore, using these options does not seem attractive. The option 4 achieves the same throughput as the option 1 which fully restores the congestion control state. Thus, the option 4 may be used by an extra careful sender which does not want to use the option 1.

TABLE 3. FACK WITH EIFEL ON A CONGESTED PATH WITH VARYING RESTORATION OF THE CONGESTION CONTROL STATE.

CC restore	Time, s	Segments sent	Spurious RTOs	Necessary RTOs
option 1	108	5225	3.24	0.38
option 2	540	5325	8.48	8.43
option 3	912	5558	11.19	14.68
option 4	109	5226	3.26	0.38

Table 4 shows results of experiments with adapting the retransmission timer. Re-seeding the timer with a new sample after a spurious timeout does not have any effect in



(a) Reno-SACK with Eifel. (b) NewReno-SACK with Eifel. (c) FACK with Eifel.

Figure 9. Effect of Eifel on TCPs over a congested link.

this scenario. The RTO is already high after a timeout but re-reseeding it does not help to prevent its fast descent. Using the back-off approach reduces the number of spurious timeouts by 40% with only a small decrease in throughput. This might be an attractive option to use. Increasing the minimum RTO is slightly less effective in this scenario than using the back-off counter.

The effect of proposed RTO adaptation methods could be different for other delay scenarios. For example, when the link bandwidth oscillates the delay jitter typically only slightly exceeds the RTO, there as in the scenario we have studied, RTO is exceeded significantly. Finally, adaptation techniques could be more effective if learnt characteristics of the path would be shared between TCP connections to the same destination [40].

TABLE 4. FACK WITH EIFEL ON A CONGESTED PATH WITH DIFFERENT RTO ADAPTATION TECHNIQUES.

RTO adapt.	CC restore	Time, s	Segments sent	Spurious RTOs	Necessary RTOs
std	option 1	109	5225	3.24	0.38
reseed		109	5225	3.24	0.38
back-off		113	5166	1.92	0.40
min++		114	5168	2.41	0.43

C. *Discussion*

We believe that TCP with the Eifel algorithm is friendly to other TCPs as the basic congestion control mechanisms triggered on a packet loss are unmodified. It gains the capacity underutilized by other TCPs and reduces the rate fairly upon congestion in the same way as other TCPs. Adding competing TCP connections to our experiments did not show any surprises.

The experiments were re-run with setting Adaptive RED [15] with automatic configuration of parameters as the bottleneck queue instead of Drop-Tail. The conclusions made based on the Drop-Tail measurements still hold and Eifel showed equal or better performance. However, in general TCP throughput was from slightly to many times lower than in case of a Drop-Tail queue. We interpret this as an artifact of our test setup with a low degree of statistical multiplexing and presence of a competing constant bit rate flow unresponsive to congestion.

We made typical modelling assumptions that TCP connections are long-lived and there is no congestion in the opposite direction. Determining the extend to which these assumptions hold in the Internet is a hard problem [14]. Wide-scale Internet measurements of TCP with the Eifel algorithm would be useful, but they are difficult to obtain, share and reproduce.

Formally assessing performance gains of applying the Eifel algorithm is difficult as the result depends on too many factors [29]. It could be from nothing to several hundred percent depending on the frequency of delay spikes, path characteristics, the retransmission timer and type of workload. The best case for Eifel occurs when one of the segments in the initial window experiences a spurious timeout on a high-delay bandwidth path. In such a case, the TCP connection stays in congestion avoidance and is likely to use only a small fraction of the available bandwidth. A TCP connection with the Eifel algorithm will continue the slow start until a segment loss indicating a real need to slow down.

The Eifel algorithm is robust to packet losses caused by data corruption, but does not perform more aggressively than traditional TCP as it still relies on a segment loss as an indication of congestion.

V. CONCLUSION AND FUTURE WORK

Delays in the Internet, especially over wireless links can be highly variable. For instance, handovers in cellular networks or on-demand allocation of a high-speed radio

channel to a wireless user can cause delay jitter beyond a normal RTT of the link. At the same time, modern TCPs such as in Linux 2.4 use a precise timer (with 10 ms granularity) and put a low limit on the minimum RTO (200 ms) [38]. Therefore, the problem of spurious timeouts in TCP is important to resolve. The Eifel algorithm uses the TCP timestamp option to robustly detect spurious retransmission timeouts. The Eifel response prevents unnecessary retransmissions and congestion control back-off performed by conventional TCP. All the required modifications are at the TCP sender.

This paper shows that in a broadband environment applying the Eifel algorithm can give up to 250% increase in throughput and at the same time decrease the load on the network by 3%. We show that the original response suggested for Eifel [29] could be further improved. In a scenario with heavy congestion, TCP with Eifel suffers from necessary timeouts even with Reno-SACK and Limited Transmit. Eifel performs well with FACK, but it may not be always used due to concerns in presence of packet reordering. Therefore, we suggested combining the NewReno and SACK algorithms in a single TCP. NewReno-SACK avoids retransmission timeouts present for Reno-SACK due to large 'holes' in the receiver window. Eifel with NewReno-SACK works well even under heavy packet losses and is presumably safe to use in the Internet.

We show that full restoration of the congestion control state does not lead to bursty behaviour. Furthermore, partial or lack of restoring of the congestion window reduces the throughput and loads the network with a greater number of unnecessary retransmissions. This is because if only the flight size is restored the sender cannot transmit segments on arriving ACKs which makes it prone to timeouts. We suggested a new option for partially restoring of the congestion control state which seems to perform as well as full restoration but is more conservative.

We studied a number of techniques for adapting of RTO to avoid further spurious timeouts. TCP with the Eifel algorithm uses samples from delayed segments to update RTO. It alone provides a more conservative timer than TCP-Reno without timestamps. However, additional methods for learning from a spurious timeout may be desirable. Re-seeding the timer with a new sample is ineffective in the scenario we used. Increasing the exponential back-off counter decreases the number of spurious timeouts by 40% with only a small decrease in throughput. Increasing the minimum RTO works slightly worse than the back-off method. Therefore, it is reasonable to implement one of the latter techniques with the Eifel algorithm. However, either FACK or NewReno-SACK are required at the same time to avoid low throughput due to a large number of necessary timeouts.

The final response of a TCP sender to a spurious timeout is as follows. After the timeout the transmission always resumes with the next unsent segment.TCP always restores the congestion window. We also recommend restoring the slow start threshold, but optionally it can be limited to the congestion window. The TCP sender uses Limited Transmit together with FACK or NewReno-SACK and adapts the RTO using the back-off counter. We believe this response is efficient and robust under a wide range of networking conditions.

We implemented this new response and a new tool for delay generation in NS2. The code is publicly available [20] and we are working on integrating it into an official NS2 release. We enhanced the simulator to enable re-use of transport agents in the same run which allows running a large number of repetitions in a reasonable time.

In the future work we plan to evaluate behaviour of other retransmit timers, such as the Eifel timer [30] in presence of highly variable delays.

ACKNOWLEDGMENTS

We thank Mark Allman, Pasi Sarolahti, Alexey Kuznetsov, and Farid Khafizov for constructive criticism and helpful suggestions.

REFERENCES

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

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

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

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

[5] J.C.R. Bennett, C. Partridge, N. Shectman, Packet Reordering is Not Pathological Network Behavior, *IEEE/ACM Transactions on Networking*, December 1999.

[6] E. Blanton, M. Allman, K. Fall, *A Conservative SACK-based Loss Recovery Algorithm for TCP*, draft-allman-tcp-sack-11.txt, work in progress, July 2002.

[7] J. C. Bolot, Characterizing End-to-End Packet Delay and Loss in the Internet, *Journal of High Speed Networks*, vol. 2(3), pp. 289--298, September 1993.

[8] R. Braden, *TCP Extensions for High Performance: An Update*, unpublished, http://www.kohala.com/start/tcplw-extensions.txt, June 1993.

[9] CAIDA, *Traffic Workload Overview*, http://www.caida.org/outreach/resources/learn/trafficworkload/tcpudp.xml, July 2002.

[10] M. Degermark, B. Nordgren, S. Pink, *IP Header Compression*, RFC 2507, February 1999.

[11] K. Fall, S. Floyd, Simulation-based Comparisons of Tahoe, Reno, and SACK TCP, *ACM Computer Communication Review*, vol. 26(3), July 1996.

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

[13] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky, A. Romanow, *An Extension to the Selective Acknowledgement (SACK) Option for TCP*, RFC 2883, July 2000

[14] S. Floyd, V. Paxson, Difficulties in Simulating the Internet, *IEEE/ACM Transactions on Networking*, vol 9(4), pp. 392-403, August 2001.

[15] S. Floyd, R. Gummadi, S. Shenker, Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management, unpublished, August 2001.

[16] A. Gurtov, R. Ludwig, Evaluating the Eifel Algorithm for TCP in a GPRS network, *European Wireless*, February 2002.

[17] A. Gurtov, *Making TCP Robust Against Delay Spikes*, University of Helsinki, Department of Computer Science, Technical Report C-2001-53, November 2001.

[18] A. Gurtov, Effect of Delays on TCP Performance, *IFIP Personal Wireless Communications*, August 2001.

[19] A. Gurtov, M. Passoja, O. Aalto, M. Raitola, Multilayer Protocol Tracing in a GPRS Network, *IEEE Vehicular Technology Conference*, September 2002.

[20] A. Gurtov, *Implementation of the Eifel algorithm for NS2*, http://www.cs.helsinki.fi/u/gurtov/ns, June 2002.

[21] IETF, *Robust Header Compression*, http://www.ietf.org/html.charters/rohc-charter.html, July 2002.

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

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

[24] V. Jacobson, *Compressing TCP/IP Headers for Low-Speed Serial Links*, RFC 1144, February 1990.

[25] P. Karn, C. Partridge, Improving Round-Trip Time Estimates in Reliable Transport Protocols, *ACM SIGCOMM*, August 1987.

[26] F. Khafizov, M. Yavuz, Running TCP over IS-2000, *IEEE Conference on Communications*, April 2002.

[27] J. Korhonen, O. Aalto, A. Gurtov, H. Laamanen, Measured Performance of GSM HSCSD and GPRS, *IEEE Conference on Communications*, June 2001.

[28] D. Loguinov and H. Radha, Measurement Study of Low-bitrate Internet Video Streaming, *ACM SIGCOMM Internet Measurement Workshop*, November 2001.

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

[30] R. Ludwig, K. Sklower, The Eifel Retransmission Timer, *ACM Computer Communication Review*, vol. 30(1), July 2000.

[31] R. Ludwig, M. Meyer, *The Eifel Algorithm for TCP*, draft-ietf-tsvwg-tcp-eifel-alg-03.txt, work in progress.

[32] M. Mathis, J. Mahdavi, Forward Acknowledgment: Refining TCP Congestion Control, *ACM SIGCOMM,* August 1996.

[33] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, *TCP Selective Acknowledgement Option*s, RFC 2018, October 1996.

[34] J. Padhye, S. Floyd, On Inferring TCP Behavior, *ACM SIGCOMM*, August 2001.

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

[36] J. Postel, *Transmission Control Protocol*, RFC793, September 1981.

[37] P. Sarolahti, M. Kojo, and K. Raatikainen, *F-RTO: A New Recovery Algorithm for TCP Retransmission Timeouts*, University of Helsinki, Department of Computer Science, Technical Report C-2002-07, February 2002.

[38] P. Sarolahti, A. Kuznetsov, Congestion Control in Linux TCP, *USENIX Annual Technical Conference*, June 2002.

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

[40] J. Touch, *TCP Control Block Interdependence*, RFC 2140, April, 1997.

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

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

[43] G. R. Wright, W. R. Stevens, *TCP/IP Illustrated, Volume 2 (The Implementation)*, Addison Wesley, January 1995.

[44] M. Yavuz, F. Khafizov, TCP over Wireless Links with Variable Bandwidth, *IEEE Vehicular Technology Conference*, September 2002.

Exploiting Packet Lifetime for Efficient Real-Time Transport

Andrei Gurtov
University of Helsinki

Reiner Ludwig
Ericsson Research

Abstract – **Achieving in-deadline delivery while avoiding unnecessary transmissions is difficult purely on the end-to-end basis, especially when such disruptive events as delays spikes are unavoidable in the network. The end points have little knowledge about the current network conditions, about how much data is still in the network and where it is buffered. The lack of information and control causes two different problems. First, in wireless networks the transport protocol can prematurely assume that outstanding packets were lost and retransmit them, while the original segments still being hold by the link layer causing the problem of competing error recovery. Second, the application can re-generate a fresh version of a data object rendering the old object buffered in the network obsolete and blocking the way for the new data. Thus, it is desirable to control for how long the network is allowed to buffer the data. Carrying the packet lifetime that controls the link level persistency. It solves the problem of competing error recovery, because by the time when the transport protocol performs a retransmission, it can be sure that the link layer has already given up on the packet in question. We believe this further advances the idea of differentiated treatment of packets at the flow-adaptive link layer. Furthermore, the application provides the transport protocol with lifetime of a data object, so that the transport layer struggles to deliver the object within its lifetime and but discards it afterwards. This prevents obsolete application data in the network to block the way for a newly generated data objects. The suggested solution requires the sender and the bottleneck router be synchronized in clocks.**

I. INTRODUCTION

It becomes increasingly evident that two widely deployed Internet protocols, TCP and UDP, cannot satisfy demands of newly emerging applications. TCP implements the ultimate reliability retransmitting data potentially forever. (Some TCPs define a maximum number if retransmissions and reset the connection after reaching the limit. However, the application has no control over this limit). UDP, on the other hand, provides a purely best effort service, without any guarantees or feedback on delivery of application data. Many of existing and emerging applications have a finer grain demands on reliability and timeliness of data delivery. In this paper, we consider a class of applications that can benefit from a data object only if it is delivered with in a given time period. One example of such an application is a news ticker (NT) that generates news updates at a regular interval and has no interest in the old news object once a newer update is generated. A different example is a file transfer (FT) started by a user who wants a file in the remaining 5 minutes before the meeting or not at all. Note, that in for news ticker the object lifetime is determined by the sender, and for a file transfer by the receiver. We do not address applications that require bounded delay, but also full reliability, for example telnet.

While meeting the requirements of applications like NT and FT is a challenge in fixed networks, it becomes even more difficult in wireless environment. Wireless network typically present a highly dynamic environment where link characteristics such as available bandwidth can change quickly and long sudden delays due to link outages, handovers and radio resource blocking are possible (experiments with 2.5G and 3G systems show that these problems are not going to disappear quickly). On the other hand, *any transmission on a wireless link directly translates into the terminal battery power consumption and increased interference to other users*. Thus, reducing the amount of unnecessary transmitted data (data which is of no value to the receiving application either because it has been already delivered once, or because it has expired) over the wireless link is a top priority task.

Achieving in-deadline delivery and avoidance of unnecessary transmissions is difficult purely on the end-to-end basis. The end points have little knowledge about the current network conditions, about how much data is still in the network and where it is buffered. The lack of information causes two different problems. First, in wireless networks the transport protocol can prematurely assume that outstanding packets were lost and retransmit them, while the original segments still being hold by the link layer causing the problem of competing error recovery [20]. Second, the application can re-generate a fresh version of a data object rendering the old object buffered in the network obsolete and blocking the way for the new data. A possible approach to these two problems is to minimize the amount of data kept in the network by configuring an appropriately small router buffer size. However, a large delay-bandwidth product, large number of hops and rapidly changing bandwidth in the network can put a limit on the lowest possible buffer size. Thus, it is desirable to control for how long the network is allowed to buffer the data. Carrying the packet lifetime that controls the link level persistency solves the first problem**.** We believe this further advances the idea of differentiated

treatment of packets at the flow-adaptive link layer [19]. This *solves the problem of competing error recovery*, because by the time when the transport protocol performs a retransmission, it can be sure that the link layer has already given up on the packet in question. The second problem is solved when applications provide the transport protocol with lifetime of a data object, so that the transport layer struggles to deliver the object within its lifetime and but discards it afterwards. This *prevents obsolete application data in the network to block the way for a newly generated data objects*. The both suggested solutions require that the sender and the bottleneck router be synchronized in clocks. It may be a realistic assumption, as for example 3G systems require synchronized clocks and GPS is commonly implemented in mobile terminals.

While the above solutions can be partly applied to existing protocols (e.g. TCP) we went on to design a new transport protocol to fully explore the idea of delivering of an application object within its lifetime with avoiding unnecessary data transmissions. The Advanced Transport Protocol (ATP) is based on the following key principles: the application level framing (ALF), modular network stack design, TCP-friendly congestion control and the concept of active networks.

The revolutionary concept of ALF [1] agues that only the application has a complete knowledge of the required data delivery service, and the application should be given full control in deciding how to cope with such events as delayed or lost data packets. Our protocol and the application communicate in terms of ADUs. Our contribution to the ALF principle is that *by giving the ADU lifetime the application clearly signals its demands to the transport protocol*. It is widely recognized that a retransmission of the complete ADU once a fragment of it is lost (for example due to congestion) is inefficient. Thus, selective retransmission of ADU fragments is often included to improve the performance. We suggest setting the *fragment lifetime to the current RTO estimate, which is less than ADU lifetime* to avoid the problem of competing error recovery.

The principle of modular design of protocol stacks (spoken by John Wroclawski in the 51st IETF meeting) argues that future protocols should be composed of reusable blocks each providing some basic feature like congestion control, connection establishment, window and flow control, and selective retransmissions [7]. Such approach makes it possible to quickly produce new protocols accustomed for the needs of a given application and puts the burden off the application designer (to reinvent the buggy wheel). The principle of conformant congestion control says that in order to avoid congestion collapse in the future Internet all protocols must implement congestion control, which is fair to TCP [21]. We study the *behavior of congestion control algorithms in the presence of expiration data losses*.

Our work does not violate the end-to-end principle as data packets remain unmodified during transit in network and routers on the path do not have to snoop into headers above the network layer. However, we refer to the idea of active networks [2] that speaks for including more intelligence into the network. The paper suggests that network routers should be more actively involved into packet delivery process, potentially replacing a packet with a capsule concept. Capsules contain a program code that specifies how the application data should be treated. Such approach opens versatile horizons by tailoring the network behavior to the actual demands of the application.

The rest of the paper is organized as follows. In Section 2 we briefly describe several protocols based on the ALF concept or including support for packet lifetime. Section 3 outlines the architecture of our new protocol. In Section 4 we explore how our ideas influence the behavior of existing applications and congestion control algorithms. In Section 5 we evaluate the quantitative performance benefits of ATP in *realistic and highly dynamic network environment* with long sudden delays and data losses. Finally, Section 6 concludes the paper and provides grounds for future work.

II. RELATED WORK

The idea of using globally synchronized clocks at the end points and network routers is exploited in [11]. The authors suggest an improvement to earliest-deadline-first scheduling algorithm that takes into account the remaining number of hops to the destination. This work has some serious drawbacks: it assumes no packet losses due to errors or congestion, does not study interaction with congestion control, does not work on reducing unnecessary transmissions, does not study the problems of competing error recovery and obsolete application data, and does not present a real protocol to experiment with.

Another work [12] develops a new way to increase efficiency of radio resource usage in a cellular radio network using the idea of globally synchronized clocks. The effect is achieved by delaying transmission of a packet in poor radio conditions if its deadline allows waiting until the link quality becomes better.

There are several transport protocols that are based on the ALF principle and are relevant to our work. Design of RTP [3] has been strongly based on the ALF concept by leaving as much functionality as possible to the application and providing only bare-bone facilities at the transport layer: multiplexing, timestamps, sequence numbers and source identifiers. RTP is a purely best-effort protocol that does not have built-in reliability or congestion control. RTCP protocol is used to provide the receiver feedback about the perceived quality of service and TFRC [4] could be used as a

congestion control mechanism. Currently five different retransmission schemes are proposed for RTP, including a version integrated with Congestion Manager and ALF [22] [23].

The Xpress Transport Protocol (XTP) [6] is a transport layer protocol designed to provide a wide range of communication services built on the concept that orthogonal protocol mechanisms can be combined to produce appropriate paradigms within the same basic framework. Rather than using a separate protocol for each type of communication, XTP's protocol options and control of the packet exchange patterns allow the application to create appropriate paradigms such as reliable datagrams, transactions, and unreliable streams. Error control, flow control, and rate control are each configured to the needs of the communication. In other words, XTP recognizes the limitations of all-or-nothing service available from TCP and UDP protocol and let the application choose finer grain service. XTP is a high-speed protocol standardized by ISO. The mechanisms of XTP are specifically designed for high bandwidth, low delay and low errors rate communication.

The HPF protocol [10] supports packet flows with different quality-of-service requirements on reliability, priority and deadlines in the same transport connection. The following are the key features of HPF: HPF supports heterogeneous packet flows with different reliability, priority and timing (delay) requirements in the same transport connection. HPF decouples the congestion control and reliability mechanisms (that are integrated in TCP) in order to support congestion control for unreliable and heterogeneous packet flows. HPF supports application-level framing, and provides APIs for applications to specify the priority, reliability and timing requirements of each frame. HPF enables the use of application-specified priorities as hints for network routers to preferentially drop low-priority packets during congestion. This ensures that `important data' gets through with high priority over unimportant data during congestion. However, the authors do not evaluate the benefits of using the deadline information in routers.

In [13] authors suggest the Image Transport Protocol (ITP) for image transmission over loss-prone congested or wireless networks. ITP runs over UDP, incorporates receiver-driven selective reliability, and uses the Congestion Manager (CM) to adapt to network congestion. ITP improves user-perceived latency using out-of-order ADU delivery, achieving significantly better interactive performance.

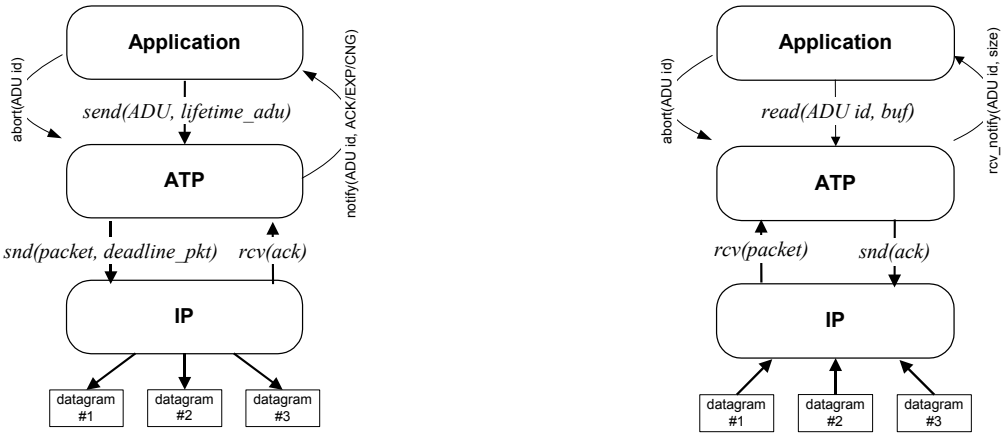
Scalable Reliable Multicast [5] is another important realization of the ALF concept. Methods for retransmitting missing data for streaming applications are evaluated in [24].

The Eifel algorithm [9] presents a competitive end-to-end solution for the problem of competing error recovery. The algorithm is based on resolving the retransmission ambiguity problem by comparing the timestamp in the returning ACK with a stored timestamp of the retransmission. If the returning ACK acknowledges the original transmission, not the retransmission, then the go-back-N behavior and congestion control reaction are aborted. The benefits of the solution presented in this paper are in further reduction of unnecessary retransmissions over Eifel for the case when the sender experiences several RTO back offs. Also, end-to-end solutions are not applicable when the ADU lifetime is comparable with path RTT leaving no time for end-to-end actions. In addition, Eifel has not yet been considered in terms of real-time applications like the news ticker or streaming.

III. PROTOCOL ARCHITECTURE

A. Interface between applications and ATP

ATP is a transport protocol operating on top of IP providing a simplex data delivery service. The application



(a) ATP sender (b) ATP receiver

Figure 1. The interaction of ATP with the application and network layer.

communicates with ATP in terms of complete ADUs and their lifetime, as shown in Figure 1. The sender application passes an ADU to ATP and gets a unique ADU id. Then, ATP struggles to deliver ADU over the network within its specified lifetime, however not giving any guarantees of in-time delivery. ATP notifies the receiving application of ADU delivery status. The ADU size may exceed the size of IP MTU, and ATP performs fragmentation and reassembly for ADU transmission. However, it may not be feasible for an application to pass an arbitrary sized ADU, as the receiver protocol entity is required to buffer the complete ADU before informing the application. For example, a file transfer application could want to transfer a 1 GB file as a single ADU. From the practical point of view it is not possible, since such a big ADU would not fit into protocol buffers. Thus, the sender application must limit the ADU size to the size of the available buffer space at the receiver protocol. In the given example, an ADU size could correspond to one disk sector or track size. How to negotiate the available buffer size is an open issue.

An open issue is whether ATP should be connection-oriented, following a TCP-like client-server model, or it should be connectionless like UDP. In case of a connection-oriented service, the client has to exchange SYNs and FINs with the server. The data transfer in the connection can go either direction and can include multiple ADUs. However, connection establishment causes overhead and there is so far no clear demand of connection-oriented service for ATP. Thus, for now we assume that ATP is connectionless. If the ADU transfer is sender-initiated (like the news ticker), the sender can just transmit the ADU to the known port of the receiver. The receiver application in this case does not know about the ongoing ADU transfer until the complete ADU is received, does not have ADU id and cannot abort the transfer of the given ADU. The application can only unbind from the known port that will cause an ICMP unreachable message to be returned for all ADUs going to the port. However, if the transfer is receiver-initiated (like a file transfer), the receiver first has to send a request with ADU URL to the sender and can supply ADU id in the request. Thus, the receiver is aware of ADU transfer before it is completed and can abort it by giving the ADU is to ATP.

ATP uses a callback mechanism to inform the application on the status of ADU delivery (the application implements a well-known function name as the callback handler). The status can include a successful delivery of ADU (ok), or indication that the ADU has expired and is deleted from the network (exp). After receiving a negative status report, the application may decide to re-send a fresh version of ADU is necessary. In addition, ATP supports a congestion notification callback to the application (cng), which is triggered if a major loss event is observed in the network. The application then can reduce the load on the network, for example by increasing the interval between news updates.

Before commencing an ADU transmission, and also possibly during the transmission, ATP can use an estimate of the available network capacity provided by the congestion control algorithm to evaluate whether it is feasible to deliver the ADU within its lifetime. If not, the ADU transfer can be aborted.

Both the sender and the receiver application can abort the ADU transfer by calling the ATP with the corresponding ADU id.

B. ATP protocol functionality

We reuse as many components as possible from existing protocols: de-multiplexing (port numbers), checksum (header and partly payload), connection establishment (may not be required for the protocol but facilitates operation through firewalls), flow control (coupled with congestion control).

A tentative header format is shown in Figure 2. This assumes that ATP operates on top of UDP. The flags field contains the ATP version number (2 bits), the ACK flag (the packet is acknowledgment), ABR (a request to abort the ADU transmission and to discard all packets belonging to this ADU), EXP (set by the router in subsequent packets after a discarding of an expired packet to avoid congestion control reaction to missing packet), and reserved bits.

ATP does not number each data byte, like TCP, but instead numbers fixed-size fragments serially within one ADU.

The header contains fields for ADU and for this given fragment. The congestion control parameters depend on the algorithm used, for example loss rate and RTT estimates for TFRC. The receiver window is replaced by ECN in a way that if receiver is unable to process packets at the rate they arrive (which is improbable) it can set the CE flag in ACK to instruct sender to slow down.

Fragment deadline (in IP)

Flags | Checksum
ADU id
ADU length
ADU deadline
Fragment offset
CC parameters
SACK (in ACK)

Figure 2. ATP header fields (each row is 32 bits).

When ADU fragments get lost due to error or congestion it would be inefficient to retransmit the whole ADU. To avoid it, ATP uses SACK information (a vector that indicates which fragments of ADU are received and which are missing) whether possible to retransmit the missing fragments. The retransmission algorithm is similar to conservative SACK recovery in TCP, which makes ATP robust against packet reordering in network.

However, as a backup, ATP keeps a RTO timer, which is computed similarly to TCP (or Eifel timer). Deadline of fragments is set to current time plus RTO (see section IV.B for recently encountered problems). A heavy congestion or a long delay can cause all packets to be discarded at the router. The lack of ACKs allows the RTO timer to fire. After that, fragments are retransmitted from the oldest outstanding, according to the pace allowed by the congestion control algorithm. Here our main contribution comes to play, that is resolving the link-layer and transport-layer competing error recovery. Since all fragments after RTO have been deleted from the network, there is not going to be a situation when both original and retransmitted fragments are located in the network buffers.

ATP receiver sends an ACK for every Nth packet, where N can be discussed with the sender, or after a delayed ACK timer expires. Every ACK contains a SACK vector displaying which packets have been received. The ATP sender needs some mean to calculate the RTT to set the RTO, and it is done is some clever way. If the receiver has better knowledge of ADU lifetime, it sets the ADU deadline field to indicate this to the sender. The receiver can also set the fragment deadline in ACKs to its RTT estimate to prevent ACK queuing problem on highly asymmetric links. Also, a burst of ACKs delivered to the sender after a delay is prevented, which saves bandwidth and may prevent a false congestion control alarm.

C. Router behavior

In order to prevent unnecessary transmissions, the router should detect and discard expired packets. Recall, that we have assumed synchronized clocks at the sender, the router and possibly the receiver. The router should check the packet lifetime at least before transmission it. Preferably, the router should check its all packets in the queue every time a packet is sent or received to make more space in the queue for newly arriving valid packets. If the router has knowledge on the link RTT or the remaining number of hops to the destination, the router can estimate whether a packet can be delivered before its deadline. If not, such packet can be discarded even if it is valid at the time of transmission. Potentially, the deadline information available in the router can be benefited in many more advances ways than just discarding of expired packets. For example, the Earliest Deadline First algorithm is suggested in the related work.

An important question is whether the router has to discard expired packets or transmit them at a lower priority. In the related work the second option is used, which is wasting wireless resources. Delivering expired packets makes sense only if the application may still somehow benefit from expired ADUs. However, delivering expired packets can prevent problems with congestion control, which requires more research.

As an optimization, the router can drop all packets from the same ADU after a single packet is dropped. This is useful when transport layer retransmissions are disabled; since such ADU would anyway be discarded at the receiver. The router can check if packet lifetime is equal to ADU lifetime. If yes, there is no time for retransmitting ADU fragments and the whole ADU can be dropped in the router. Also, ADU ids are required for the router to detect packets belonging to the same ADU. This optimization is not useful when packet losses occur due to expiration, because in this case the following packets after the lost one also most probably will expire and will be discarded automatically. However, an ADU drop feature is useful when packet losses are present due to congestion or corruption. If ADU drop mode is enabled at the router, those packets that are sent before the dropped one are also wasting wireless resources. Perhaps the router can first collect all packets belonging to ADU before start transmitting the first packet. This is problematic for large ADUs and is potentially dangerous because packets can expire while waiting for other ADU packets.

A method to synchronize clocks at the router and the hosts is required. It is not nowadays uncommon to have GPS clocks in support nodes in a cellular network. Also, NTP protocol is widely used in the Internet. The older way is the WWVB clocks discussed in RFC 778 and [17]. The required accuracy of clock synchronization is in order of tens of milliseconds. If the router clock gets out-of-sync it can have bad effect on ATP. If the clock lags back compared to the sender clock, then packets do not expire at the router and ATP functionality is not triggered. The worse case if when the clock is much ahead of the sender clock. In this case the router will discard all packets preventing any data delivery to the receiver.

D. Congestion Control

Stability of the Internet is based on assumption that all transport protocols must implement the congestion control. In the absent of reliable signaling method using which the network can report a congestion to the sender, the fundamental assumption is that when a packet loss is detected and no explicit information is available on the reason of such loss, it must be treated as an implicit congestion indication.

ATP uses ECN whether possible to signal congestion to avoid unnecessary packet drops in the network. Also, ECN can be used for flow control to replace TCP's receiver window. The ultimate congestion signal is detection of a lost packet via ACK vector or RTO. ATP transmits packets at the rate allowed by the congestion control mechanism. If it does not allow sending the ADU within its lifetime, ADU expires in ATP buffer, and the application is informed.

The congestion control method can be selected by the application. At least three methods can be used:

AIMD. This is TCP-like congestion control suitable for bulk transfer applications not sensible to large variations in send rate. ATP should not be sensible to such variations as long as ADU gets delivered in time, but AIMD can have negative impact on the wireless link utilization. Congestion control is window-based with ACK frequency is per two packets with maximum delay of 500 ms.

TRFC. This is a method that avoids large variations in sending rate by reacting to congestion signals less rapidly and ramping up slowly when no congestion is detected. Congestion control is rate-based with one ACK roughly per RTT.

CM. Integrating with congestion manager allows sharing the common bottleneck bandwidth more fairly with other flows from the same host.

ATP has also a capability for 'automatic' congestion control. A growing queue at the router can leads to packet expiration before the queue actually overflows thus resulting into a congestion signal to the sender and reduction in the sending rate and consequently the queue size.

It has to be experimentally confirmed which congestion control method is best suited for ATP in general. In particular, since every packet expiration and drop is treated as a congestion signal, this may lead to reduced goodput if packet expiration is caused by a delay on the link. A possible alternative is just to transmit packet headers from expired packets or set EXP bit after an expiration drop in the router to avoid congestion control for expiration losses.

E. Implementation Issues

While ATP is best implemented as an independent transport protocol, some of its features can be added to existing protocols. Below there is a list of issues arising when adding ATP functionality to existing protocols.

TCP. A separate connection is required per ADU. Packet lifetime is set to TCP RTO+RTT, and TCP is modified to retransmit lost packets indefinitely (currently TCPs do only a limited number of retransmissions). In addition, a kill timer is started which resets the TCP connection if it does not complete before ADU lifetime. This approach is well suitable for larger ADU size, like the FTP example, but may cause too much overhead for small ADUs like news ticker.

UDP includes only basic transport mechanisms such as port numbers and checksum. UDP can provide good basic services for ATP. ATP then requires addition of ACKs for recovering lost packets and for congestion control. IP deadline option is set based on RTO estimate from ACKs.

DCP has connection establishment and ACKs that are used for congestion control. However, DCP does not provide any data reliability. Thus, ATP would need to use the Ack Vector Feature of DCP to retransmit lost packets. Packet lifetime needs to be set to the RTO estimate obtained by ATP code. Also, DCP has no ALF so it has to be implemented in ATP layer. DCP may be the best basic platform for ATP.

ITP. This protocol contains a custom header that contains all the fields that ATP needs and ALF support However ITP is not having any real-time support, and thus ATP is a very good complement for ITP. ITP has timestamp and RTO estimate in the header, the packet lifetime is set to RTO estimate.

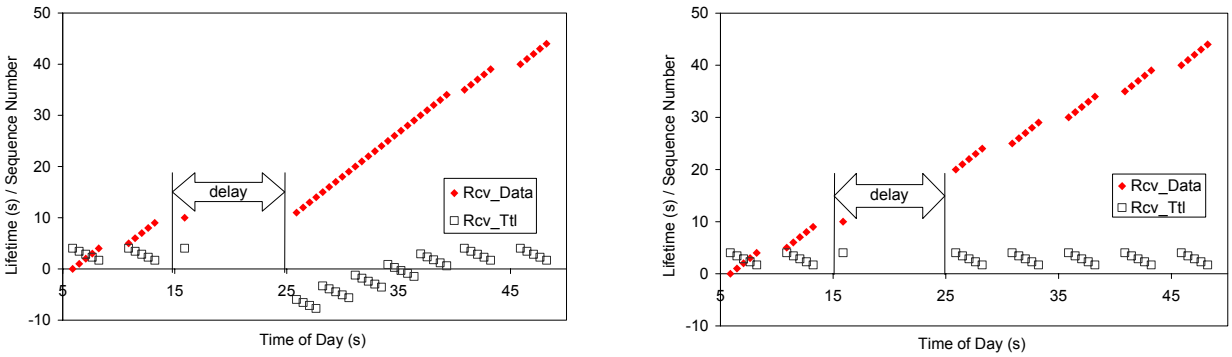
HPF is a TCP-based protocol and supports ALF with ADU lifetime. ATP can be implemented on top of HPF by setting the packet deadline to RTO value to recover congestion losses.

RTP can be used as the base protocol for ATP with an existing extension that supports selective retransmissions and congestion control.

The full version ATP has to be implemented on NS and Unix for experimentation. Which protocol to use as the starting point while implementing ATP depends on the availability of the base protocol for different platforms (Table 1). In addition, a preferable congestion control algorithm should be also available for given platform.

Table 1. Implementations of protocols for Network Simulator, Linux and FreeBSD

	NS	Linux	FreeBSD
DCP	No	Planned	No
ITP	No	Yes	No
HPF	No	Yes	No
RTP (w/o extensions)	Yes	Yes	Yes
TFRC	Yes	Yes (user-level)	Yes (user-level)
CM	No	Yes	No
AIMD	Yes	Yes	Yes



(a) Without deadline support at the router

(b) With deadline support at the router.

Figure 3: Reaction of the News ticker application to a 10-sec sudden delay.

F. How to carry packet deadline information

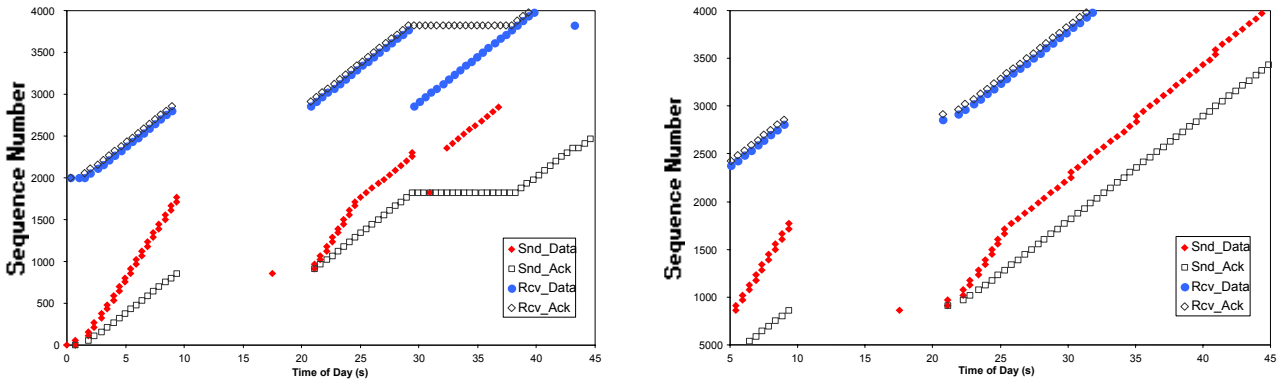
The way in which the expiration time (the deadline) is carried in packet headers is of a vital importance if only ATP is to become any real protocol. It is best done in the IP header, because then all protocols above the IP can use it and router do not have to snoop into the upper protocol headers. We have identified three possibilities.

1) *An IP timestamp option plus scaled TTL IP field*. The IP timestamp option is defined in [17][18]. It occupies 8 bytes, and contains a 32-bit time-of-day timestamp in milliseconds modulo 24 from midnight UT. It is unclear how widely this option is supported by routers and hosts. The IP TTL is a 1-byte field that originally was supposed to carry a packet lifetime in seconds. However, in practice the routers just decrement this field by one on each hop. Simply setting the TTL field to packet lifetime does not provide enough resolution and can lead to ATP-unaware routers to prematurely drop the packet in a likely case when a packet is forwarded in less than a second. A possible solution is to set TTL to hundreds of milliseconds, which enhances the resolution to acceptable values and also ATP-unaware routers can safely decrease TTL by one. This gives a range of ADU lifetimes from 100 ms to 25.6 s, which is probably suitable for a news ticker but not for a file transfer.
2) *A custom format of the IP timestamp option*. The RFC [18] defines that if the high-order bit is set, then any custom format for time can be used. We can employ this by setting the IP timestamp to the packet expiration time and set the high bit, so that ATP-unaware routers do not confuse the option. This gives a maximum ADU lifetime of 12 hours, which is probably long enough for most of applications. This method seems to be the best.
3) *A transport layer timestamp*. Alternatives to IP-level are timestamps sometimes readily available in the transport headers. The advantage of this is that no overhead is needed to carry timestamps in two places. At least TCP, RTP, ITP and DCP have timestamps in the header. However, TCP timestamps for example are not related to real time, but just are incrementing counters.

In NS the IP TTL is a 32-bit value, which made our tests simple: we just set TTL to packet deadline. It is not very accurate as it does not reflect the overhead caused by carrying additional bytes, but it does give some understanding on protocol behavior.

IV. EFFECT OF ATP ON PROTOCOL BEHAVIOR

In this section we discuss the effect of deadline support in the router. For this purpose we have been using a simple extension in NS2 to include the deadline information into TTL field of IP packets. The remaining packet lifetime must be at least FixTtl seconds (which is an estimation of link one-way latency) before transmission over the bottleneck link to prevent unnecessary transmission of packets that get expired before reaching the receiver. Otherwise a packet is considered expired and is dropped.



(a) Without deadline support at the router (b) With delay support at the router

Figure 4. Reaction of TCP to a 10-second sudden delay

A. News ticker on top of UDP without congestion control

The News Ticker (NT) application is assumed to transmit a news update (a fixed-size ADU) at some regular interval. Once a new tick is available, the old one is considered outdated and should be discarded. If ADU is larger than the MTU provided by the network, then a tick is sent as multiple packets. The ADU lifetime and packet lifetime is set to news tick update interval. For our tests we have been using a simple NT implementation on top of UDP which is sending 5KB ADU every 5 seconds. MTU size is 700 bytes. FixTtl in this test is set to 0.6 s (which is the time to deliver a 700-byte packet over 9.6 kbps link).

Figure 3 (a) shows the effect of a 10-second delay on the delivery of NT data at the receiver. The rcv_data shows the sequence number of arriving packets at the receiver, and the rcv_ttl shows the remaining lifetime of the arrived packet. Negative values in this case mean that the packet is outdated and should be discarded. When the delay starts at 16 sec, no ADU is delivered until delay ends, but newly arriving ADUs get queued in the buffer. When the delay ends at 26 sec, for the next 15 sec the link delivers only the expired ADUs. The backlog of expired ADUs in this case blocks the fresh ADU to be delivered to the user. Further on, transmission of expired ADUs wastes the wireless bandwidth.

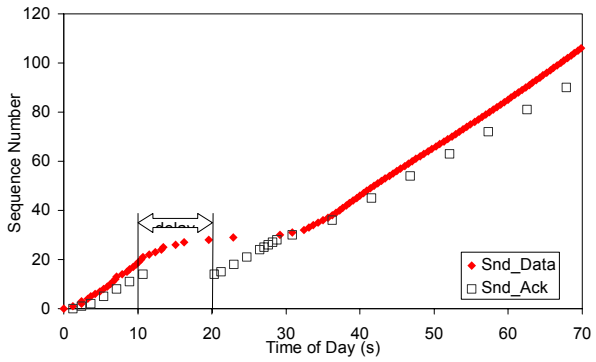
Figure 3 (b) shows the similar experiment when the router is capable to detect and discard the packets with passed deadline information. Immediately when the delay ends, fresh ADUs are delivered to the receiver. Further on, no expired packets are unnecessary sent over the wireless link saving the resources.

In this experiment we have been using a large buffer size (100 KB) in the router. In case a small (e.g. 10 KB) buffer is used, the ATP has less benefit, because ADUs get dropped during the delay due to a buffer overflow. Also, we assumed no congestion control at the NT application, which should have slowed down while not getting any feedback during a delay.

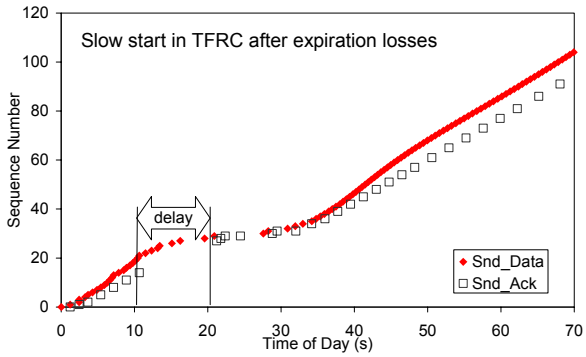
There can be situations when the NT sender transmits ADU with a faster rate when could be handled by the bottleneck link. If this is the case, a queue forms at the router and all ADUs are delivered expired to the receiver. Including deadline support at the router does not help in this situation, because a single missing fragment leads to the whole ADU being dropped at the receiver, as the ADU short lifetime does not allow retransmissions fragment. The sender should handle this situation by checking if ADU can be sent within RTT (latency-limited) and against bandwidth estimates provided by CC algorithm (bandwidth -limited).

B. Bulk transfer using TCP

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 4 (a) produced using FullTcp. Sequence numbers in the receiver trace are offset by 20000 bytes to prevent an overlap with sequence numbers of the sender. The delay in generated between 10th and 20th second in this test. The first retransmission that happens at the 17th second is also delayed. The sender interprets the ACK generated by the receiver in response to the delayed segment as related to the retransmission, not the original segment. This happens due to the retransmission ambiguity problem as the ACK bears no information on 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 a retransmission policy is refereed to as go-back-N since the sender forgets about all segments it has earlier transmitted. At 28th second retransmitted segments arrive to the receiver and generate DUPACKs as the original segments have already been delivered. When the threshold



(a) Without deadline support at the router.



(b) With deadline support at the router.

Figure 5. Reaction TFRC to a long delay without deadline support at the router.

of three DUPACKs is reached at the sender a spurious fast retransmit is triggered, as the sender does not implement the bug fix. Further on, new segments are clocked out by arriving DUPACKs.

Figure 4 (b) shows the effect of dropping of expired segments. At the time when the sender times out and retransmits the first segment at 17th second, all segments previously transmitted are dropped at the sender. If the sender would time out several times, the previous retransmissions are also dropped. When the delay ends, the sender immediately gets an ACK for the latest retransmitted segment, and continues retransmitting segments using the slow start. Since all originally transmitted segments are dropped, there are no negative side effects on TCP. However, the slow start performed by the TCP sender may become time-consuming on networks with large bandwidth delay product increasing the download time. In this case the bandwidth delay product is small and the pipe is filled already after two RTTs after the delay ends.

Setting a correct lifetime setting is difficult for TCP since it is unpredictable when the RTO timer will actually expire. First, the actual expiration of RTO in TCP is offset by the RTT as the timer is restarted upon every ACK. It is even worse since the RTO at the time when segment N is sent (at that time segment N-M is the oldest outstanding segment) is not the RTO that will eventually be used to restart REXMT when N has become the oldest outstanding segment. It is the RTO calculated from the ACK for segment N-1 that will be used to retransmit segment N should it be lost. If the packet lifetime is set to RTO then segments get wrongly expired during the slow start due to a large queuing delay. If the lifetime is set to RTO+RTT then if the delay is just above RTO, then packets do not get expired and there are unnecessary retransmissions. Both cases can happen for the same connection. *A fundamental problem identified here is that a reliable transport protocol does not know the RTO of a packet at the time of its transmission. A possible solution to this problem is to set the packet lifetime to RTO+TTL and to use Eifel as a back-up way to prevent unnecessary retransmissions.*

C. TFRC

In this simple test we have been using a TFRC implementation (after fixing some bugs) in NS that sends bulk data without any reliability provided.

Figure 5 (a) shows the reaction of TFRC sender to a 10-sec delay. As expected due to a lack of feedback during the delay, the sender gradually slows down to eventually transmit one packet per RTT. When the delay ends, a burst of ACKs that were delayed comes to the sender. It takes about 15 sec after delay ends for the sender to get to normal transmission rate. However, since the receiver reports no loss events, the sender does no congestion control reductions and continues in congestion avoidance.

Figure 5 (b) shows the same case when the router drops the expired packets and ACKs. At the sender lifetime in packets is set to the RTO estimate. At the receiver lifetime of ACKs is set to RTT estimate plus 3 sec, to allow for a reasonable lifetime in the beginning of connection until the RTO estimate is unknown at the receiver. Until the delay ends, the sender behavior is identical to (a), as expected. However, since ACKs report multiple loss events at the receiver, the sender performs the slow start. In this example, performing slow start does not result in decreased download time, as the connection ramps up quickly and ACK arrive more frequently compared to case (a). In some tests we have actually seen (b) to achieve 20% better throughput than (a), mostly due to the fact that due to packet expirations and unreliable service less data had to be transmitted over the bottleneck link.

Our tests also have indicated that TFRC is rather sensible to the pattern of packet and ACK loss that may trigger lengthy idle periods in transmission. This needs further investigation.

	Min, s	Max, s
Easy	80	140
Mediocre	40	80
Difficult	20	40

Table 2. The interval between delays in three scenarios.

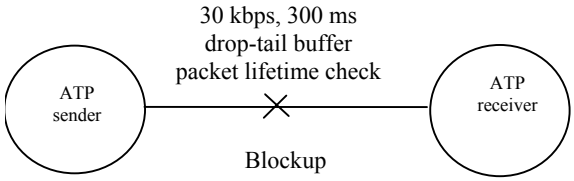


Figure 6. Test configuration in NS2.

V. PERFORMANCE EVALUATION

To illustrate the performance benefits ATP may have for the news ticker (subsection B) and file transfer application (subsection C) we have used a simple ATP implementation in the NS2 simulator and described in subsection A. For a more comprehensive ATP evaluation in presence of competing traffic and random losses (subsection D), a complete ATP implementation is required. The complete ATP implementation could be done on top of one of existing protocols (Table 1) for NS, Linux or FreeBSD.

A. Setup of Experiments

Our goal is to evaluate ATP in the realistic environment close to observed in the live networks. Three scenarios with different intervals of delays (cell reselections) are shown in Table 2. The interval times are selected in a way that for a typical download time of 120 s, correspondingly one, two and three delays occur per connection for the "easy", "mediocre" and "difficult" scenario on average. The length of the next cell reselection event is generated randomly from the uniform distribution with these parameters. We assume the length of the cell reselection delay to be uniformly distributed between 3 and 15 seconds.

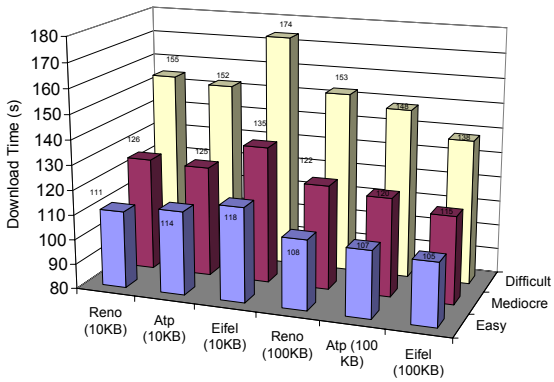
The test configuration in Figure 6 contains two nodes and a link with a drop-tail queue. The full-duplex link has the rate of 30 kbps and one-way latency of 300 ms. The `blockup` module inserts delays on the link. We have done two sets of tests with the bottleneck buffer firstly set to 10 KB, and secondly to 100 KB (which is large enough to avoid congestion losses).

First, we planned using the hiccup tool to provide delays on the link. However, in real life the delays block the transfer in both directions at once, and hiccup could not provide this type of a delay. Thus, we have implemented a new tool 'blockup' that allows to implement bi-directional outage after the link queue (although the packet which is being transmitted on the link when the delay starts is allowed to complete the transmission).

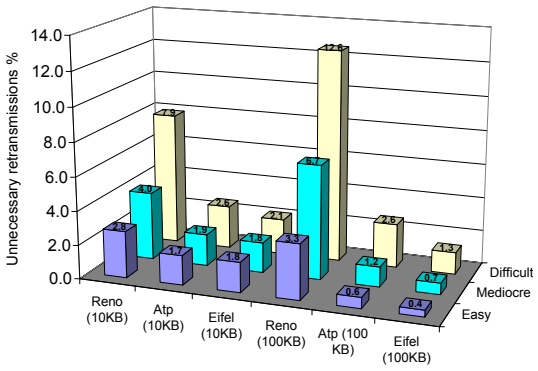
B. TCP/ATP comparison

For experiments we have been using a FullTcp model of Reno-TCP. A single test is based on a TCP connection transferring 300 KB of bulk data. Tests include TCP connection establishment and termination. Each test has been repeated a hundred times to ensure sound statistics. We have used default values for all FullTcp parameters, except for the "bug fix" that has been disabled and the timestamp option that has been enabled. ATP in this test equals Reno TCP with TTL=RTO+RTT and dropping of expired packets in the router.

The results are shown in Figure 7, ATP does not have much benefit over Reno or Eifel due to the problem of correct packet lifetime setting (results looked better with hiccup). For a small buffer size, ATP reduces the download time slightly over Reno and Eifel, and decreases the number of unnecessary retransmissions over Reno. For large buffer,



(a) Average download time



(b) Average ratio of segments sent unnecessary over wireless link

Figure 7. Results for Reno and ATP (the bottleneck buffer size in parenthesis).

ATP also reduces download time and retransmissions, but not as good as Eifel. The achieved benefits looks like not enough to cover the trouble of syncing clocks.

C. UDP/ATP comparison

This tests attempt to evaluate quantitative benefits of ATP for the news ticker application described in section IV.A. FixTtl in this test is set to 0.6 s (which is the time to deliver a 700-byte packet over 30 kbps 300ms link). In this test ATP means UDP with packet lifetime set to ADU transmit interval (5 s) and packet lifetime checking in the router. Figure 8 (a) shows how many valid packets are delivered to the receiver. ATP increases this value considerably both for small and large buffer. The ratio of unnecessary transmitted packets over the bottleneck link is shown in Figure 8 (b). It is zero for ATP in all cases (if FixTtl is set correctly), while can be as large as 27% for UDP.

The current results are given in terms of packets, not ADUs, which is not quite accurate, because if a single packet is missing, the whole ADU is dropped at the receiver. When technically implemented, the results should show the ratio of ADU delivered valid to the receiver.

D. ATP evaluation with congestion control and data losses

This section will provide performance results in NS or the Internet with a complete ATP implementation, and should be done after ATP design issues are agreed. The application can be the news ticker with TFRC congestion control. The ATP performance is compared to performance of UDP and TCP SACK. The communication environment should be highly dynamic with sudden delays, link speed changes, and congestion losses introduced by competing traffic.

Two test set can be done

1) ATP sender performs selective retransmissions of missing ADU fragments (for example due to short ADU lifetime). In this case, the router can drop a complete ADU once a single fragment is dropped, since the ADU will be anyway discarded at the receiver.
2) ATP sender performs selective retransmissions of missing ADU fragments. In this case the router should drop only expired fragments.

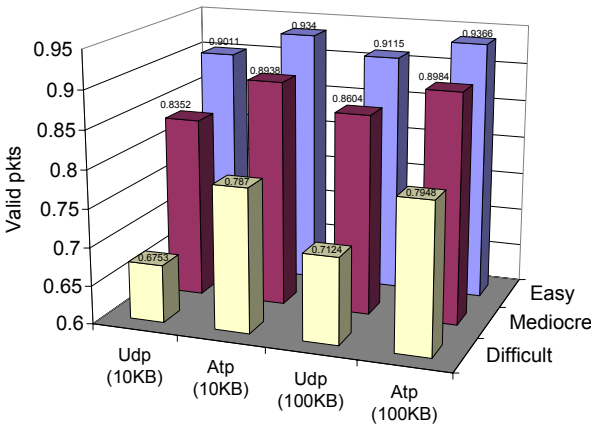
VI. CONCLUSION

At the moment the amount of arising problems seems to exceed the amount of benefits ATP can provide, at least for TCP-like protocols. The problems are in difficulties with setting packet lifetime correctly (the actual RTO is unknown when packet is transmitted), interaction with congestion control, syncing router and host clocks, overhead in the router.

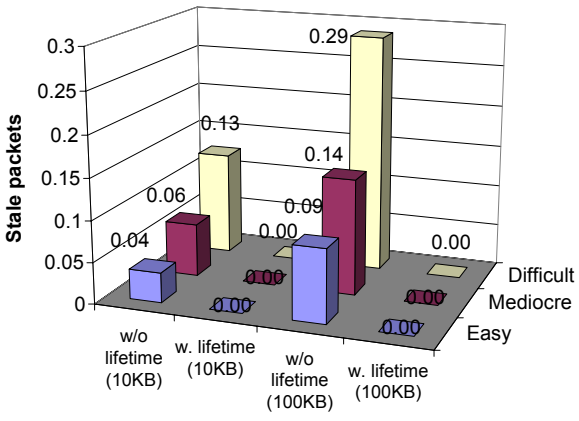
Hopefully, the ideas presented in this paper can be better suited for streaming or conversational traffic, web browsing, interactive gaming and other ‘more real-time’ applications than bulk transfer using TCP.

The packet lifetime information can be used in a better way rather than just dropping expired packets, for example by doing the earliest deadline first scheduling. Priority to ADUs can be added later so that the router can schedule ADUs accordingly.

To enable proper operation of ATP with IPsec, the protocol fields inspected by an ATP-aware router should not be encrypted.



(a) Ratio of packets delivered in time



(b) Ratio of packets sent unnecessary over the bottleneck link

Figure 8. Results for UDP and ATP (the bottleneck buffer size in parenthesis).

p-119

REFERENCES

[1] D. D. Clark and D. Tennenhouse, Architectural Considerations for a New Generation of Protocols, Proc. of the ACM SIGCOMM '90 Conference, September, 1990.

[2] D. L. Tennenhouse, D. J. Wetherall, Towards an Active Network Architecture, Computer Communication Review, a publication of ACM SIGCOMM, volume 26, number 2, April 1996.

[3] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, RTP: A Transport Protocol for Real-Time Applications, RFC 1889, January 1996.

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

[5] Floyd, S., Jacobson, V., Liu, C., McCanne, S., and Zhang, L., A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing, IEEE/ACM Transactions on Networking, December 1997, Volume 5, Number 6, pp. 784-803.

[6] A.C. Weaver, Xpress transport protocol specification v4.0. XTP Forum, March 1995.

[7] I. Chrisment, D. Kaplan, and C. Diot, Design, Automated Implementation and Evaluation of an ALF Communication Architecture, in IEEE Journal of Selected Area Communication, Volume 16, Number 3, April 1998, pp 332-344

[8] S. Bailey, ULP framing for TCP, draft-ietf-tsvwg-tcp-ulp-frame-00.txt, work in progress.

[9] Reiner Ludwig, Randy H. Katz, The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions. Appears in ACM Computer Communications Review, Vol. 30, No. 1, January 2000.

[10] J. R. Li, S. Ha and V. Bharghavan, ``A Transport Protocol For Heterogeneous Packet Flows.'' IEEE Infocom '99, New York, NY. March 1999.

[11] Wong, J.W.; Liu, Y.E., Deadline based network resource management, In Proceedings of the Ninth International Conference on Computer Communications and Networks, 2000, pages 264 -268.

[12] Lee, K.S., Zarki, M.E., Scheduling real-time traffic in IP-based cellular networks The 11th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, Volume 2, 2000, pages 1202 –1206

[13] Raman, S., Balakrishnan, H., Srinivasan, M. An image transport protocol for the Internet. 2000 International Conference on Network Protocols, 2000, pages 209 -219.

[14] M. Schläger, NS TCP Eifel Page, http://www-tkn.ee.tu-berlin.de/~morten/eifel/ns-eifel.html

[15] A. Gurtov, R. Ludwig. Evaluating the Eifel Algorithm for TCP in a GPRS network. Submitted to European Wireless'2002.

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

[17] Zaw-Sing Su, A SPECIFICATION OF THE INTERNET PROTOCOL (IP) TIMESTAMP OPTION, RFC 781, May 1981

[18] J. Postel, Internet Protocol (IP), RFC-791, September 1981.

[19] Ludwig R. and Rathonyi B. Link Layer Enhancements for TCP/IP over GSM In Proceedings of IEEE INFOCOM 1999.

[20] Ludwig R., Rathonyi B., Konrad A., Oden K., and Joseph A., Multi-layer Tracing of TCP over a Reliable Wireless Link. In Proceedings of ACM SIGMETRICS 1999.

[21] Floyd, S, Congestion Control Principles. RFC 2914, Best Current Practice, September 2000.

[22] Balakrishnan, H., et al., SR-RTP Software Library/Toolkit, http://nms.lcs.mit.edu/software/videocm/main.html

[23] N. Feamster, H. Balakrishnan , Packet Loss Recovery for Streaming Video, 12th International Packet Video Workshop, April 2002.

[24] D. Loguinov and H. Radha, Retransmission Schemes for Streaming Internet Multimedia: Evaluation Model and Performance Analysis, ACM SIGCOMM Computer Communication Review (CCR), vol. 32, no. 2, April 2002.