Improving 3G Performance for the Mobile Internet

Erik Lundsten

KTH, Royal Institute of Technology
Department of Microelectronics and Information Technology

Master of Science Thesis
Performed at Center for Wireless Systems, KTH
and Telia Research

Examiner:

Prof. Gunnar Karlsson

Advisor:

Anders Dahlén, PhD.

Telia Research AB

Abstract

TCP has some properties that make it inefficient when used as a transport protocol for wireless links. It has been the subject of many research projects and a number of solutions have been suggested. Most of these proposed solutions are trying to improve TCP's performance in general without looking at a specific technology.

UMTS, the universal mobile telecommunication system, is a new standard for mobile networks. The UMTS radio access network is called UTRAN, UMTS radio access network, and it uses WCDMA, wideband code division multiple access, as its radio access method. When constructing the radio network it would be beneficial if a high error rate could be used for packet-based services. However, such a high error rate would affect the performance of TCP.

The introduction of retransmission mechanisms in the radio link control layer reduces the error rates of UMTS. An ensuing problem however is that the delay will vary. The delay or reordering of data due to the retransmissions may cause TCP to underutilize the radio link.

TCP's design is based on the assumption that transmission errors occur rarely. Hence, TCP assumes that all packet losses are due to congestion and it cannot tell congestion from loss due to error. When packet loss or packet reordering occurs due to errors on the wireless link, TCP interprets this as congestion and limits the sending rate. This leads to an underutilization of the radio link.

This thesis reviews and investigates a few suggested solutions to the underutilization problem. The solutions are of different character: TCP can be changed to handle wireless communication better, but it is not the only way to mitigate the problem. The RLC, radio link control, could be configured to deal better with the problem, and improved radio links can also be a used.

The most important proposals are: Eifel, TCP Westwood, Split TCP and RLC configurations. They are examined and simulated using a model of the UMTS RLC, implemented in NS2. In-sequence and out-of-sequence delivery in the RLC are tested, and the effect of different radio block sizes is examined. We also gauge how well the suggested solutions handle spurious timeouts and fast retransmissions. For small file transfers the improvement in performance is measured when the initial window is increased. The aim when conducting these simulations is to find the most suitable solutions for reducing the underutilization.

The main result from this study is that there is a severe underutilization for small IP packets in combination with high transfer speeds. The utilization is even lower when small radio blocks are used and some solution is clearly needed. Generally, the in-sequence delivery option of the radio link should be used to deal with the problems. However, when small radio blocks are used an additional solution is needed. Split TCP is found the best in terms of performance but Eifel is also worth considering.

Acknowledgments

During my work with my master thesis I have had the opportunity to obtain both helpful advices and useful support from several persons. I would like to thank both Telia Research AB and Wireless@KTH for providing me with the means needed to complete my work. I am especially grateful to my supervisor Anders Dahlén for making such a commitment to my work and for rewarding discussions. I also would like to thank Gunnar Karlsson for his valuable advices and for being my examiner.

Table of Content

List of Figures

List of Tables

1 Introduction

The Internet has mainly been interconnecting stationary computers connected by wired links. This is about to change since increasingly more communication is done from mobile clients requiring wireless communication. This will generate new possibilities both for business and the technology development. One way of accessing the Internet wirelessly is to use UMTS, the universal mobile telecommunication system. UMTS is a third generation mobile system and allows packet-based IP communication between hosts connected to the Internet.

Since large geographical areas must be covered with UMTS radio access points, the price of the network will be extremely important for the success of the system. This implies that the operator would like to use as few access points (base stations) as possible without suffering reduced quality or capacity. Hence, maximization of the utilization of an access point is of greatest importance. Sparsely distributed access points may yield high bit error rate, BER, which in turn might have implications for the performance of connections using TCP. This problem has been described in [8], [9] and [11]. Although it has been studied earlier, no one has investigated and compared the solutions examined here to any wider extent.

TCP, the transport control protocol, is the most commonly used protocol for reliable data transfers over the Internet. TCP provides a reliable connection-oriented service that many popular applications utilize. TCP has undergone a few changes since its introduction, mostly related to performance. The underlying media has been assumed to be reliable with low error rate, which is not valid for wireless links.

Unreliable radio links that connect the mobile host to the base station can cause TCP to perform unnecessary retransmissions. TCP can also reduce its transmission speed due to impairments in the radio link. Short file transfers also present a problem since the TCP connection does not utilize all of the available bandwidth in the start-up period of a connection. UMTS will worsen the problems by its large round trip time, RTT, since TCP then takes more time to recover from loss. Furthermore, TCP will have problems due to handovers and connectivity loss, but those two issues are beyond the scope of this study.

It is vital for the industry to find ways to increase TCP performance for wireless links. Quite a few proposals are available but some are of experimental character and will not fulfill the requirements of the industry. The solutions considered in this thesis are TCP Westwood, Eifel and Split TCP. TCP Westwood estimates the available bandwidth and with that information as a basis, it does not reduce the transfer rate more then necessary. Eifel provides functionality for detecting spurious retransmissions to avoid reductions in sending rate. Split TCP divides the connection into two parts, one over the wireless part and one over the wired part. For small files, the impact of increasing the initial window is also studied. The aim of this thesis is to investigate the aforementioned solutions and quantify how well they mitigate the problems.

To achieve this, we implement a model of the UMTS RLC in NS2 and carry out simulations with large and small files. In the simulations the RLC will be working in out-of-sequence or in in-sequence delivery mode. We consider both small and large IP packets as well as small and large radio blocks. TCP Westwood, Eifel and Split TCP are included in the simulations and their ability to improve the performance is measured.

We summarize TCP and describe its features that are related to congestion control in Section 2.1. The general characteristics of the wireless media that are relevant for transport protocols are described in Section 2.2, and more specific aspects for running TCP over UMTS are described in Sections 2.3 and 2.4. This study also introduces some other solutions to the problems in Section 3.1 and we discuss their effects and applicability in section 4. Section 5 presents how the work was done and simulation results can be found in Section 5.3 to 5.5. A short analysis of the results is given in Section 6 with conclusions in Section 7. An appendix gives a description of the important parts for this work of NS2, network simulator 2, is given. It also describes the implementation of the system.

2 Background

2.1 TCP Overview

2.1.1 Introduction

TCP has been available since the mid 1970's; it was used in combination with IP in the ARPA-net. The properties that made TCP successful were its speed, its simplicity and the fact that it was easy to implement. It is also the most used protocol on the Internet nowadays with well over 90 % of the total amount of traffic carried [13].

The sender side of TCP divides the data into segments and sends each segment separately. The receiver side of the connection controls that the segments are assembled in the right order. The size of the segments are important, as we will see later, and the sender generally wants to use as large segments as possible. The receiver announces the maximum segment size, MSS that it is willing to accept; the sender will try to use segments of that size. Unfortunately there might be limits in the maximum size of segments that can be sent to the receiver, and intermediate routers may fragment the segments. Usage of MTU path discovery is recommended to find out how big segments that the network can handle [23]. If it is not used, TCP will have to rely on the minimum size that is guaranteed (536 bytes) in order to avoid fragmentation. In general, TCP benefits from having as large segments as possible, and if Ethernet is used the maximum segment size is usually 1460 bytes.

TCP is in its basic form a rather simple protocol that provides reliability over a non-reliable network, e.g. the Internet. To accomplish this it relies on acknowledgment of data; i.e., the receiver acknowledges data that are received properly. The basic technique that TCP is based on for sending data in an efficient way is a sliding window. The sliding window provides maximal usage of the link by allowing segments of data to be sent before receiving acknowledgments. As illustrated in Figure 1, datagrams are always in transit.

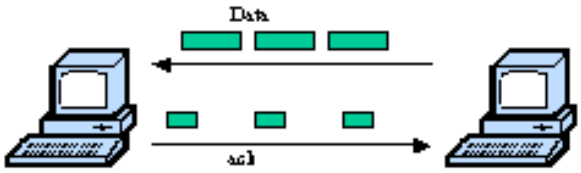

Figure 1 Data is always in transit.

Since TCP is widely used in large networks, it is a major contributor to the total amount of traffic. Due to this fact it is important that TCP does not flood the network with more traffic than the network can handle. That would not only be a problem for the network, but it would also limit the performance of each individual connection. To avoid this problem, congestion control mechanisms are included in TCP [1].

Since the receiver is not able to handle an unlimited amount of data in a short time, it needs to inform the sender know much data it can handle at the moment. This is done by window announcements: the receiver simply tells the sender how many bytes of data it can handle for the time being. This is called the advertised window and it may change over

time, since the receiver may at one time have a full buffer and at another time an empty buffer. The sender may not send more data than the receiver has announced.

An RTT, round trip time, measurement is taken by the sender to monitor the state of the network in form of delay. The RTT is used to calculate for how long it is reasonable to wait for an acknowledgment from the receiver. The receiver can save bandwidth by sending an acknowledgment together with a data segment. This is called piggybacking and acknowledgments are sometimes delayed until there is data to send. TCP is required to send an acknowledgment for every other segment received even if there is no data to send. But in order for TCP to exchange data, the connection first needs to be set up. This is done in the connection establishment.

2.1.2 Connection Establishment

In order for two processes to communicate using TCP, they first need to negotiate the starting parameters of the connection. The first step in the procedure of setting up a TCP connection between two processes is the three-way-handshake, seen in Figure 2.

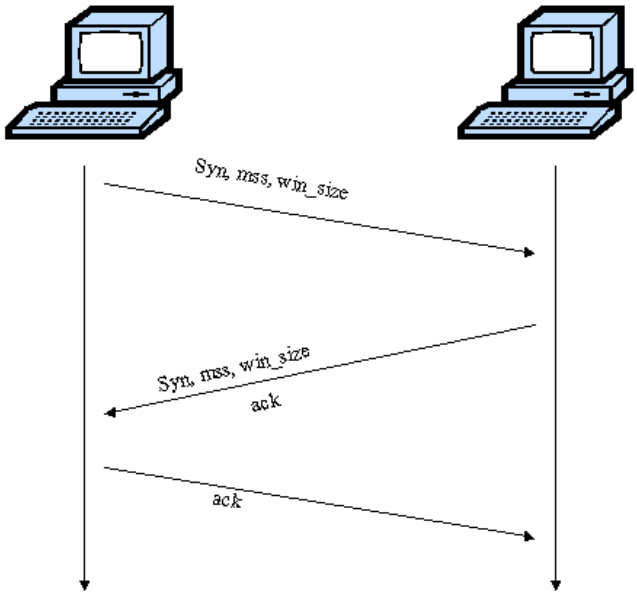

Figure 2 The three-way-handshake

The initiator first sends a SYN, synchronize, with a value of the MSS and a window size advertisement. The other party responds with an acknowledgment, ACK, and a window size advertisement. This is usually sent in the same message by using piggybacking. Attached to the SYN is also information about the MSS. When the initiator receives this information it sends an acknowledgment as an answer, the acknowledgment is received and the processes are then ready to communicate.

2.1.3 Slow Start

TCP is required to start sending the data slowly in order to prevent congestion. Thus, TCP starts out with sending only one segment of data sent. A state variable in TCP keeps track of the amount of data that can be sent without receiving acknowledgments. The variable is called congestion window and it has a central role in TCP's functionality. The congestion window is used to control the transfer rate of the sender including the slow start. The advertised window from the receiver can of course not be exceeded even if the congestion window is larger. The maximum of the congestion window and the advertised window is called the *send window*, and it determines how many segments the sender may send before receiving an acknowledgment.

TCP starts out by setting the congestion window to an initial size, usually one segment[1]. However, as soon as TCP receives the ACK for the first segment, the congestion window is increased by one segment. Now two segments can be sent, and when the corresponding acknowledgments arrive the congestion window is incremented by two segments. For every packet acknowledged the congestion window is increased by one segment. This will result in an exponential growth of the congestion window and it is called slow start. The slow start continues until the advertised window or the slow start threshold is reached. Congestion will also stop the slow start, as we will describe later.

2.1.4 Congestion Control

Congestion control is an important function in TCP. By this, TCP can detect congestion in the network by examining the received acknowledgments. When congestion occurs in the network, TCP can find out about it in two different ways. First the retransmission timer, RTO, can timeout and thereby signal lost packets. Second, the sender may receive a number of acknowledgments for the same packet. It is an indication of congestion, since every TCP implementation must send duplicate acknowledgments as soon as it receives packets out of order or if one or several packets are lost but the subsequent packets are correctly received. The reason for packets to arrive out of order is often packet loss: one packet is lost but the subsequent packets arrive correctly. However, reordering could also be the result of packets being routed different ways from the sender to the receiver.

A TCP-connection might encounter congestion and TCP responds by decreasing the transfer rate when this happens. How much TCP backs off is dependent on the way the congestion is discovered. If there was a RTO timeout, TCP sets the congestion window to size one and re-enters slow start. A threshold is used in order to know for how long to continue with the slow start. TCP will continue in the slow start phase until the congestion window reaches that threshold. It is set to half the size of the congestion window's former value after a timeout. This is a rather drastic way to limit the amount of data and another method, called fast recovery, is used when acknowledgments are received in duplicate.

2.1.5 Fast Retransmit

Fast retransmit is used to reduce the recovery time after a packet loss. When TCP receives a number of duplicate acknowledgments, DUPACKS, it responds by retransmitting the segment that followed the last acknowledged one, i.e., the last packet that was received correctly in sequence. The congestion window is halved when the acknowledgment for the

[1] The standard [1] allows an initial window size of maximum two segments.

retransmitted segment is received. After halving the window, the window is increased according to congestion avoidance. The data flow will recover faster, due to the window management, and thereby decrease the recovery time. More details about the fast retransmit can be found in [1].

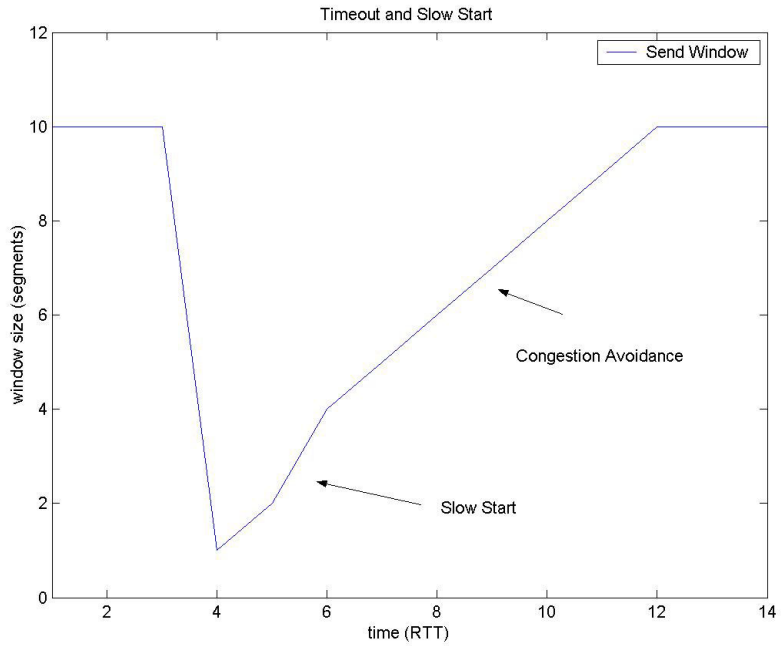

Figure 3 The size of the send window for TCP in slow start

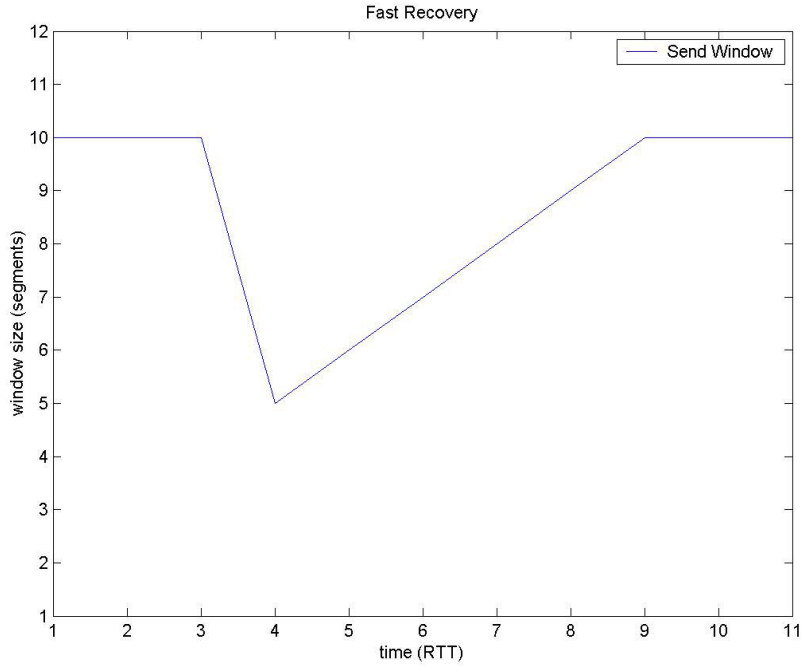

Figure 4 The size of the send window when TCP is doing a fast recovery

In Figure 3, we can see that the congestion window is set to size one after a timeout occurred. This can be compared to a fast recovery as seen in Figure 4, where the window is halved. The cost in time for recovering from congestion, by either slow start or fast

recovery, is substantial since the time in the figures is measured in RTTs. If the RTT is large, the effect seen in Figure 3 and Figure 4 will have a greater impact on performance. For more information about TCP refer to [12] and [15].

2.2 Wireless Links

Wireless communication use many of the protocols designed for wired links, e.g. TCP. TCP will not perform as well in wireless environments where the bit error rate is much higher. Different wireless technologies have different characteristics, but a few properties are common and they will have an impact on TCP's performance.

High bit error rate is maybe the most important factor that can limit the utilization of the link. When high enough, it can cause all communication to fail. One big challenge in wireless communication is to minimize the bit error rate, BER. However, it is expensive to build networks with low BER and therefore it is important to find a way to make the upper layers unaware of the data loss that the high BER cause. This is where retransmission over the wireless link can be of use.

Retransmission on link level can hide the losses but at the cost of increase in delay variation. If the link encounters an error during the transmission, it may try to resend the data and thereby causing delay. The delay will vary since retransmissions only happen occasionally, due to random loss. This may have an impact on the overlying protocols if those are dependent on the delay of the link. One might argue that it would be sufficient only to do end-to-end retransmissions. But it turns out that a simple retransmission on the link level will in general lead to improvement in efficiency.

In order to do the retransmission, the sender-side of the link will have to buffer the outgoing data until it can determine that the data has been received correctly. Furthermore, the receiver will obviously need processing capacity for detecting errors and order retransmissions. This introduces extra complexity, which in turn leads to extra cost.

Variable bandwidth in the connection between sender and receiver is another important factor. Bandwidth variation can occur due to a few different reasons. The most common reason in ordinary wireless communication is reduced quality in the radio environment. This could happen due to interference or other circumstances that makes the conditions for receiving signals worse. Generally speaking: the longer the range, the lower the bandwidth. Bandwidth can also vary in systems that implements priorities, one user with low priority may have to wait for another user with higher priority. Furthermore, users in wireless networks often share an access point with other users through a shared channel. This may result in additional delays and short periods without connectivity due to the fact that several users cannot access the network simultaneously. Depending on the application being used, this may become a problem.

Asymmetric bandwidth is often used when providing Internet access for the end-user. It is based on the assumption that end-users often download more than they upload. Some transport protocols may have trouble handling this when the asymmetry is too big but TCP will not have trouble as long as the asymmetry is in the range of 3 to 6 times [3]. Asymmetric bandwidth is used in many other environments then the wireless, e.g. ADSL, but it nevertheless presents a potential problem.

2.3 UMTS Overview

2.3.1 Introduction

UMTS, the universal mobile telecommunication system, is defined by 3GPP, 3rd generation partnership project agreement, and defines a set of systems for providing many communication services. The standard specifies services from ordinary telephone calls, and associated services, as well as packet based data communication and connectivity to the Internet. The packet-based communication is the main difference from older mobile networks, such as GSM. UMTS has been updated since its first definition and the standardization has led to several releases. The main difference with newer releases compared to older ones, is the number of features and services provided. The available releases are R99, R4 and R5 while R6 is still under development. For more information about the UMTS refer to [14].

UMTS makes use of a radio access system to provide connectivity for the users. The radio access system is called UTRAN, which stands for UMTS terrestrial radio access network. The access network makes use of WCDMA, wideband code-division multiple access, as its radio interface.

Quality of service, QoS, is also provided by UMTS. Support for different needs of QoS is available, since there are different needs in different applications. UMTS realizes this and defines several classes for QoS traffic: Background, Interactive, Real-Time Streaming and Real-time Conversational. These are used in different situation to enable the applications to get the most out of the UMTS transport. Real-time classes are defined in UMTS to serve time-critical applications, which require small delay and delay variations. In other situations the need for reliability (correct data) makes the real-time class infeasible to use, and instead data transfer that incorporate radio link retransmissions should be used.

Besides QoS, best effort service is also provided. When providing best effort, there exist several modes in which the radio link can be used. They are the transparent, the acknowledged and the unacknowledged mode. The acknowledged mode is used for TCP traffic to provide a reliable link, but at the cost of retransmissions. (See Section 2.2.1) The use of the acknowledge mode is possible since most applications using TCP are not time critical.

2.3.2 Radio Protocols

The radio protocols of UMTS have to provide different services for different users and applications. Therefore, the radio protocols have to be designed in such way that they can be used with great flexibility.

UMTS radio protocols are designed in a three-layer model. Layer one and two are mainly used for the data transfer while layer three, the radio resource control, contributes by providing utilities for connection establishment, configuration of the lower interfaces etc.

As previously mentioned, UMTS makes use of WCDMA as its radio interface. The main advantage of WDCMA over the radio interfaces used today is the speed. It increases the speed up to 2 megabits per second in local area access mode and 384 kilobits per second in wide area access mode. The idea is to differentiate the capacity so that higher speeds can be used where the access points are capable of handling it, i.e., where the radio conditions are good.

Level two protocols are PDCP, BMC, RLC and MAC. The RLC, radio link control, handles the control of one logical channel. The link layer control also handles the

retransmission if the channel operates in acknowledged mode, which is the case for traffic like TCP. Retransmission on the link level is activated when bit errors cause a radio frame to be discarded. More about UMTS and the RLC can be found in Section 5.1.

2.4 TCP over UMTS

2.4.1 Known Problems

The main problem with applying TCP as the transport protocol over wireless links is that TCP does not have the capability to discover that packet losses are caused by transmission errors. TCP treats all losses as an indication of congestion, since it is designed to be used in an environment with low error rate. TCP would benefit from being able to tell the difference between losses caused by congestion and transmission losses.

Another problem is when data arrives out of order and thereby triggers TCP to activate the fast retransmission algorithm. If the reordering of packets is caused by retransmission on link level, it is a spurious TCP retransmission. If only the receiver had waited a little longer the packet would have arrived, due to the retransmission provided by the link layer. This leads to that unnecessary TCP retransmissions are issued causing a waste of capacity. The scenario is described in [11].

Wireless networks such as UMTS and GPRS can be characterized as so-called long thin networks, which means that they have moderate bandwidth in combination with high delays. The bandwidth delay product, BDP, is a way to characterize the topology in terms of how much data the pipe can hold and how long delays it introduces. BDP is defined as the bandwidth multiplied with the end-to-end delay. It can be defined both for networks as well as for single links and it is of rather high value in UMTS. A high bandwidth delay product will cause much information to stay in the network for some time during the transfer. This will have effects on the communication between the sender and the receiver since signaling is delayed. This can therefore limit the performance of communication that relies on such signaling. In [4], much of these effects are described.

For TCP, high RTT causes a high retransmission cost, since more time is wasted during the slow start and the congestion avoidance phase. It takes longer for the window size to reach its normal level again, and when the congestion window is small not all of the available capacity is used.

Mobile clients who are using wireless access to some network may not only experience reduced capacity, they may even become temporarily disconnected. This is the nature of wireless communication and must be considered when design choices are made. Although disconnection is unwanted it may not be possible to avoid and it is good if the time without connectivity does not introduce problems for TCP. Moreover, handovers can cause similar performance degradation. However, these problems will not be considered here.

Since TCP does not use all of the available bandwidth in the slow start phase, small files can cause the utilization of the link to become very low. If the TCP connection experience problems, e.g. reordering of packets or packet loss, during the slow start the utilization will be even worse. Moreover, UMTS has a rather high RTT and it increases with high retransmission rate. Hence, transferring small files with TCP over UMTS will be extra sensitive. In [8] an analytical model for TCP transfers is constructed. The author realizes

that TCP's slow start is a vital part of the total time when small files are being transferred and takes that into account when constructing the model.

TCP uses the sliding window approach as described in Section 2.1.1. However, in order to fully utilize the available network capacity, the send window needs to be larger than the BDP. If the receiver has a limited amount of buffer space, it cannot advertise as big window as needed to fully utilize the capacity provided by the network. This will also lead to underutilization.

We have in this section pointed out and described a few problems and they can be summarized in the following list:

- The inability to differentiate between congestion and radio loss
- Reordering of packets
- Long delays
- Limited bandwidth
- Small file transfers
- Congestion handling
- Bandwidth variation
- Limited buffer size
- Handle time without connectivity
- Handovers

In this report we focus on high radio block error rates that introduce packet reordering and high delay variations. Also, the performance for small file transfers is studied.

2.4.2 Traffic to Mobile Clients

How much the wireless environment affects TCP is mostly dependent on the traffic that traverses the wireless link. The behavior of the users of the new mobile terminals will decide what the traffic pattern will look like, and thereby also implicitly affect the utilization of the wireless media. We choose not to study any specific traffic pattern but instead we look at large file transfers. Moreover, small files of different sizes are also used in the simulations.

3 Proposed Solutions

3.1 The Solutions

3.1.1 Introduction

There are quite a few things that could be done to improve the performance of TCP over wireless media. The remedies can be categorized into sections according to the procedure they use to solve the problem. The solutions are all trying to improve the throughput of TCP on wireless links but some also improve TCP's performance in general.

First, we have the ones that suggest modifications to the parameters of TCP. One of the most obvious is the increase of the initial congestion window. This will lead to that TCP will start up faster and thereby less time is wasted in the start up phase in which the link is not fully utilized. There are several other modifications of this kind, but often TCP's performance is increased for wired links as well; they are not specific for the wireless area.

Secondly, we have the solutions that try to change the topology. One way of changing the topology is to split the TCP connection into two parts; this approach is called Split-TCP and one implementation is I-TCP [7]. Another way is to analyze the traffic at the boundary between the wired and wireless segment, and from that information modify the traffic to increase performance as Snoop [2] does. Since these solutions are specialized, they will not improve TCP's performance in general and they also increase the complexity.

Finally, we have the approaches that propose modifications to TCP itself. These solutions change the TCP algorithm in different ways, in order to improve performance over wireless links. Often they also improve performance in wired environments. They can make TCP substantially more complex depending on how TCP is changed. Eifel and TCP Westwood falls under this category.

Besides these, the radio link will also have a major impact on how well TCP performs. HSDPA is a new radio link that has not yet reached the market but it looks promising in mitigating some of the problems associated with wireless communication. Moreover, the configuration of the present radio links affects the performance of TCP.

3.1.2 Split TCP and I-TCP

Split TCP is based on the fact that different parts of the path from the sender to the receiver have different characteristics, and by dividing the path into two sections each section can be optimized. Split TCP is illustrated in Figure 5, and in the wireless context the TCP connection is usually split at the base station. Splitting at the base station will result in having different TCP-sessions over the wireless link and the wired part of the network.

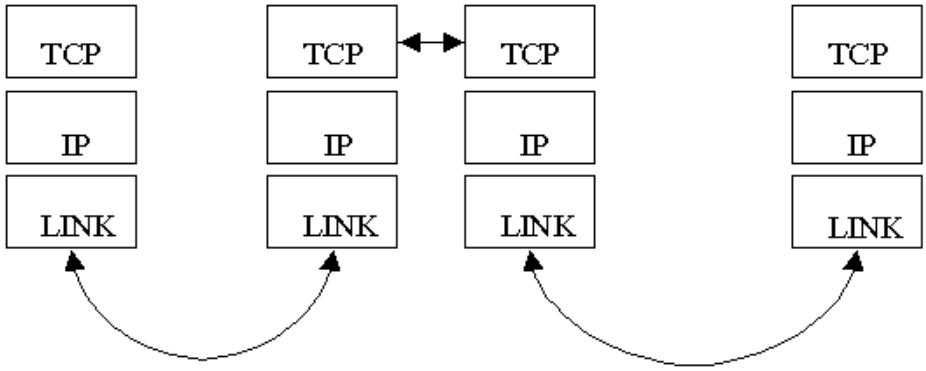

Figure 5 The idea behind Split TCP

I-TCP [7] is one design and implementation of the split TCP semantics. Besides defining how the splitting should be implemented, I-TCP also describes how to handle handovers. Handovers is beyond the scope of this report, but they also have an effect on TCP's performance. The main advantage of Split TCP is that retransmissions and errors on the wireless link will not cause TCP to issue end-to-end retransmission. Since the TCP connection is divided into two parts, the two connections can be highly optimized for the environment present at each part of the connection. Prominent in this context is the fact that each part will have a lower RTT, which implies that the TCP transfer rate recovers faster after congestion handling.

3.1.3 Snoop

Snoop [2] introduces extra functionality to a node right before the wireless link and to the mobile client. Snoops works by examining the TCP header and can thereby be considered as a proxy technology since the underlying layers are not transparent. It inspects traffic that flows through the base station and by looking at the TCP packets and applying a set of rules for handling retransmissions and other events, Snoop improves the performance of TCP. The improvement is the result of the fact that Snoop is preventing TCP from doing unnecessary end-to-end retransmissions, but instead retransmits IP packets only over the wireless link. This differs from the link-level retransmission described in Section 2.2 where the retransmission is independent if TCP. Snoop is a TCP aware retransmission scheme.

The main improvement in performance is due to TCP not reducing its congestion window, because the snoop agent residing in the base station apprehends duplicate acknowledgments, that are due to loss at the radio link, and thereby restrict the receiver from initiating a fast retransmit. To accomplish this, Snoop caches packets until the receiver has acknowledged them.

Snoop works by intercepting all messages that passes through the base station and after inspecting them taking the appropriate action. Both acknowledgments and data packets are intercepted. Snoop discards spurious ACKs from the mobile client. The reason that Snoop can distinguish between real ACKs and spurious ACKs is that Snoop keeps track of the already acknowledged data. Furthermore, Snoop filters out duplicate ACKs and retransmit only locally if it is needed.

Snoop also deals with data transfers from the mobile host to a fixed host. For doing so Snoop introduces NACKs, negative acknowledgments, which are based upon the SACK option of TCP. The NACKs are used for requesting retransmission from the mobile client without forcing TCP to activate the congestion control. This will lead to a better performance but it requires the mobile client to have support for SACK.

3.1.4 Eifel

Eifel [5] is a modification of TCP that improves throughput performance by a different management of the congestion window. Eifel introduces new functionality letting TCP to see when a fast retransmission or a time-out is spurious. A spurious fast retransmission or time-out is not detectable, by standard TCP, due to the retransmission ambiguity (explained in next paragraph). By eliminating this ambiguity, Eifel can manage the congestion window with greater accuracy.

The retransmission ambiguity is illustrated in Figure 6. When the sender receives duplicate acknowledgments, or gets a time-out for a packet arriving late it issues a retransmission of the packet missing. The packet is re-sent and eventually an acknowledgment for that packet is received. When the ACK is received the sender does not know which packet it corresponds to. Did the receiver send the ACK when the original packet arrived or was it sent when the retransmitted packet arrived? Since no information about this is available the sender must assume that the original packet was lost and that the ACK corresponds to the retransmitted packet.

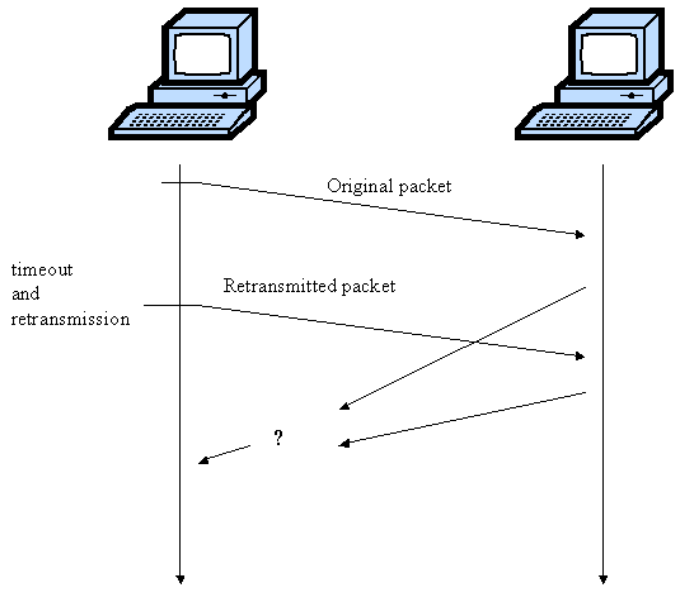

Figure 6 The retransmission ambiguity

When the sender assumes that the original packet was lost it has to assume that it was lost due to congestion and therefore issues window management. If the acknowledgment in fact corresponded to the original packet, the congestion window will have been decreased unnecessarily, since there was no congestion. In fact, the packet was not even lost.

Eifel enables TCP to determine what packet the ACK corresponds to, i.e., if it corresponds to the original or the retransmitted packet. This is done by using timestamps and by recording the time when the retransmitted packet is sent. When TCP issues the retransmission, it saves the time and then sends the packet. A timestamp, containing the same time as recorded by the sender, is attached to the retransmitted packet. When the receiver receives a packet it copies the timestamp from the incoming packet to the outgoing ACK. This enables the sender to see the time when the acknowledged packet was sent. By looking at the timestamp attached to the ACK Eifel knows if it corresponds to the original packet or not. If the timestamp recorded at the retransmission is newer than the time in the acknowledgment's timestamp, it must be the original packet that is being acknowledged. If instead the acknowledgment's timestamp is newer or equal, the sender knows that it originates from the retransmitted packet.

When TCP detects that the ACK corresponds to the original packet, by using Eifel, there is no need to reduce the amount of segments that can be sent since there was not any congestion. This lets TCP use the same size of the congestion window as it was using before the retransmission occurred and by this eliminating the slow start and congestion avoidance phases. This leads to increased throughput. Eifel of course also has the capability to detect spurious timeouts [19], which also leads to improved performance when timeouts are common.

The timestamp option is used to separate the ambiguity when starting after a retransmission. Since the TCP timestamp is only an option it is not included in older TCP implementations. Eifel is backwards compatible with standard TCP. This enables the modification to be introduced incrementally in the network.

3.1.5 TCP Westwood

TCP Westwood [6] is a server side modification to TCP that tries to estimate the bandwidth in use in order to adjust the congestion window and the slow start threshold in an effective way. TCP Westwood falls into the category of solutions that changes the behavior of TCP. The authors of TCP Westwood argue that it can be seen as a natural evolution of TCP and compares the change from TCP Tahoe to TCP Reno with the change from TCP Reno to TCP Westwood.

The improvement in throughput is the result of changes in the handling of the congestion window in the case of a retransmission. TCP Westwood estimates the bandwidth available and makes use of that information when setting the slow start threshold and the congestion window size.

The estimation of the bandwidth is based on samples that TCP Westwood collects from every ACK received. When an acknowledgment is received the estimated bandwidth is calculated. This may seem simple and straightforward but in order to filter out fast fluctuations, the samples are passed through a discrete low pass filter. The filter takes into consideration how much data the ACK acknowledges as well as the time elapsed since the last sample. The time sample is used in order to weight old values lower compared to

newer ones. All this adds up to give TCP Westwood a reasonable estimate of the bandwidth available, even when the network is congested.

The new threshold will be the estimated bandwidth multiplied by the RTT. The idea is to allow the sender to send as much data as the estimated delay bandwidth product, i.e. using all the link's or network's available capacity.

TCP retransmissions can, as previously mentioned, either be caused by receiving duplicate acknowledgments or by a timeout. In the case of duplicate acknowledgments, TCP Westwood sets the slow start threshold to a new value, based on the bandwidth estimation. TCP only enters congestion avoidance if the current congestion window is larger that the new threshold. This makes sense since we do not need to reduce the rate if there is more bandwidth available than what we are using. If the retransmission was caused by a timeout, the congestion window is set to one segment and a slow start is issued. This is similar to what TCP Reno does but with the difference that the slow start threshold is calculated using a new mechanism.

3.1.6 HSDPA

HSDPA, high-speed downlink packet access, is a technology that will serve as bearer for UMTS in the future. This will improve both quality and speed resulting in data rates well up in the megabit range. When WCDMA is used with HSDPA, services can make use of a speed as high as 8 Mbit/s. The delay will also decrease due to reduction in the amount of interleaving.

HSDPA uses a faster and more advanced link layer retransmission scheme than "older" links, to compensate for the relatively high loss rate. The main advantage over "older" links is its speed both in transfer rate and in response time. This will lead to different effects on TCP's congestion control and hopefully, by its quicker reaction to loss, reduce the problem with TCP over wireless substantially.

In [10] it is shown that HSDPA, using its shared channel capacity, mitigates the throughput problem as well as increases the utilization of the system. Besides providing better effectiveness for TCP, HSDPA also increase the total system performance. The improvement for the end-user is not thoroughly analyzed but indications point to a significant increase in performance. The performance of HSDPA is also studied in [26].

3.1.7 Other Solutions

There are, as mentioned earlier, numerous approaches for trying to solve problems related to TCP over wireless, besides the ones presented in the previous sections. Often they share the same underlying semantics as the solutions presented above, and thereby also share many of the performance enhancing attributes.

TCP SACK, Selective Acknowledgment, [20] provides functionality for a selective-acknowledgment scheme instead of the incremental acknowledgments usually used. Performance is especially improved when several packets in the same window are lost. T/TCP, TCP for transactions, [21] can improve the performance when transferring small files since it reduces the set-up time of a TCP connection.

Optimizing the parameters of TCP can also be a good way to improve the performance. Increasing the MSS as well as widening the initial window size mitigates the throughput

problem [9]. Furthermore, increasing the threshold for duplicate acknowledgments will decrease the probability for spurious retransmissions but at the cost of longer time before congestion is detected. In order to be able to choose the most suitable solution in a specific context, a comparison is needed.

In UMTS the mobile host can either deliver the received packets to the IP layer as soon as they arrive or the packets can be sorted and delivered in the order they were received at the base station. Sorting the packets might improve performance when the packet reordering due to radio link retransmissions is extensive.

3.2 Comparison of the Proposed Solutions

The different solutions are based on totally different ideas, as we understand from the previous sections. Therefore they can be difficult to compare. While TCP Westwood and Eifel are modifications to TCP, HSDPA is a new type of radio link. Split TCP is as described a way to isolate the characteristic of the wireless link from the rest of the communication path. Snoop is somehow similar to Split TCP but it also resembles the link layer retransmission scheme.

The main advantage of TCP Westwood is that it gives rather good improvements in wired situations as well as in wireless without changing the end-to-end semantics of TCP. In [6], huge improvements can be shown but this can also be the result of TCP Westwood being more aggressive. Since congestion is unwanted, are more aggressive approaches really the way to improve performance over wireless links? Furthermore, the migration from TCP Reno to TCP Westwood will take a long time since every TCP implementation will have to be changed.

The Eifel algorithm is a rather clean modification of TCP. Although it is less complex than TCP Westwood, which uses bandwidth estimation, it requires the host to use the timestamp option. Since not all hosts are required to use the timestamp option it is not possible to use Eifel everywhere. Eifel shares the migration problem with TCP Westwood and will not be used in every implementation in a long time.

Snoop shares some properties with Split TCP but keeps the end-to-end semantics. Thereby it escapes the criticism that Split TCP has got due to the fact that it does break the end-to-end semantics. Split TCP has another flaw: every packet that is sent across the TCP connection has to go through the TCP-stack four times. That's twice as many times as when using ordinary end-to-end TCP. This is the result of the back-to-back TCP stacks at the base station and may introduce some extra delay depending on the processing power available at the intermediate node. Furthermore, it is obvious that the base station needs to have an implementation of TCP that otherwise would not be necessary. All this leads to these approaches not always being advisable to use.

HSDPA does not really have any obvious disadvantages besides its cost. One property that can complicate the evaluation of HSDPA is that it can use shared channel communication. The scheduling algorithm that is responsible for distributing the right to send data, has a major impact on how TCP will perform. Since TCP is sensitive to variations in delay, a scheduling algorithm that leave one user without connectivity for some time will affect TCP in a negative way. Furthermore, HSDPA is not in use today and it is not clear when it will be implemented into the mobile networks.

The simplest form of solution is the optimization of TCP's parameters as the initial congestion window size and the MSS. These optimizations can be used together with other modifications of TCP, and the combination might very well mitigate the problem. In order to have a big MSS, path MTU discovery is important to use. Increasing the window size that is used after a timeout will allow TCP to reach a high data rate faster and thus limiting the cost of a retransmission.

Table 1 The presented solutions and the problems they address

	Differentiate between Congestion and Loss	Spurious Retransmissions and Timeouts	Small Files	Congestion
Split TCP	X	X	X	X
Snoop	X	X		
Eifel		X		
TCPW		X		X
HSDPA		X	X	X
Parameter opt.		X	X	X

A summary of the solutions and what problems they address is seen in Table 1. The figure should be considered as a guideline, since it does not say how well the different solutions mitigate the problems. Furthermore, there may be difficulties implementing the solutions, as well as side effects. Depending on which problem is considered the worst, different solutions could be recommended.

4 Discussion of the Proposed Solutions

Much work in this area has already been done, as previously seen. The problem is that the solutions will behave differently in different networks (UMTS, GPRS, WLAN) and a general rule of which solution to use is not applicable. All solutions may not even be possible to use depending on how much freedom there is to change the system. Several solutions introduce extra complexity in order to improve performance and this seems to be the general tradeoff. How much complexity can then be accepted for a certain performance improvement?

The suggested modifications to TCP will definitely reduce the problem but in different ways. Also the cost of implementation differs. Since both TCP Westwood and the Eifel algorithm are backwards compatible, they will work in a partially changed environment. If the support for these solutions is not widespread they will of course not work in many situations.

The split connection forces a "TCP-proxy" into the system and introduces all the problems associated with that. HSDPA is a new type of radio link that introduces different behaviour for the connection to the clients. This will at the same time change the underlying layer for TCP and thereby change its behaviour.

Security issues cannot be neglected when looking at how well the solutions can be implemented and how well they improve performance. If the mobile client wishes to communicate with a server on the Internet using e.g. IPSec, it is extremely important that the solution does not interfere with such security protocols. For instance, Split TCP and Snoop do not allow end-to-end IPSec.

Simulations done with the different solutions show a big improvement over the versions of TCP that are mainly used today (TCP Reno and successors). Most of the simulations and tests have considered WLAN as means of communication. WLAN has different characteristics than UMTS and that may lead to different conclusions. Moreover, the case considered is often to transfer a big file from a server on the Internet to the mobile client. This is a reasonable assumption apart from one thing: the file size may be small. If this is the case, more of the total transfer period will be spent in the start up period and thereby limiting the improvement. Furthermore, the traffic pattern, e.g. the packet sizes and the traffic distribution, is an important factor when judging how well a solution will mitigate the problem.

The eventual introduction of IPv6 will also have a role in the effectiveness of using TCP over wireless media. IPv6 has bigger packets in general and the minimum size of the MTU is substantially larger.

More testing and evaluation is needed in the context of UMTS since the characteristics of UTRAN will have a major impact on the performance of TCP. The link layer has retransmission and it is necessary to examine how this affects the suggested solutions. Other factors of the link used for UMTS radio access might also impact TCP's performance over the wireless link.

5 Method and Results

A few of the proposed solutions, described in the previous sections, have been simulated in a modeled environment. Before these simulations could be done and the result can be presented, the model of the radio link must first be presented. We must however first present how the UMTS RLC, radio link control, works. Section 5.1 will describe some relevant parts of UMTS in order to give a general understanding. The following section will describe the model and point out its most important features.

5.1 UMTS

5.1.1 UMTS – UE, UTRAN and CN

We need a bit more knowledge in the area of data transmission in UMTS in order to be able to build the model. UMTS is comprised of several logical network elements as described in the background section. There are three main parts: UE, the user equipment, UTRAN, the UMTS terrestrial radio access network, and CN, the core network. The architecture of the system is in many ways similar to the one used for GSM. A more comprehensive description of UMTS is available in [14].

The user equipment provides the interface to the user and the radio interface that connects the UE with the UTRAN. This interface is called Uu. The UTRAN defines how users access the network but also how they will be connected to the core network. Furthermore, the UTRAN also defines various radio-related functions. The core network is the internal structure of UMTS and communicates with UTRAN over the Iu interface. The core network's switching and routing is of limited interest here, since this study will only deal with the interface for transferring data wirelessly from and to the mobile host.

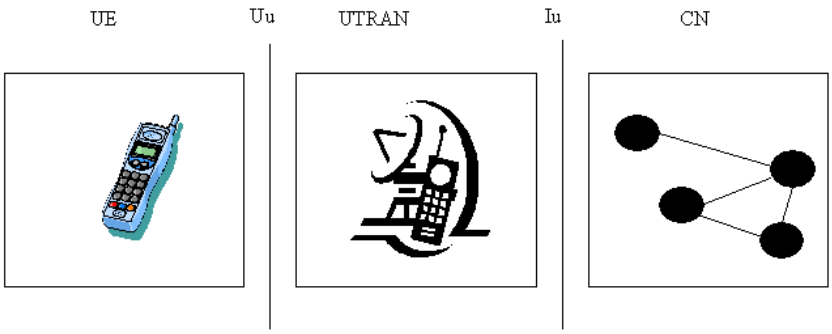

Figure 7 How the logical elements are interconnected

The logical network elements are interconnected by interfaces according to the structure seen in Figure 7. Another dimension is that the interfaces are divided into two planes: The user plane and the control plane. This is done to distinguish between the data related to controlling the transmission and the actual user data.

The system is divided into several logical elements, as described. These logical elements are in turn made up of smaller components.

5.1.2 Components of the UE, UTRAN and CN

The UTRAN has two main components: RNC and Node B. While RNC, the radio network controller, controls the radio resources, Node B functions as a base station. Figure 8, shows how RNC and Node B are interconnected. Every RNC can have several Node B. Furthermore, different radio network controllers can communicate with each other through the Iur interface.

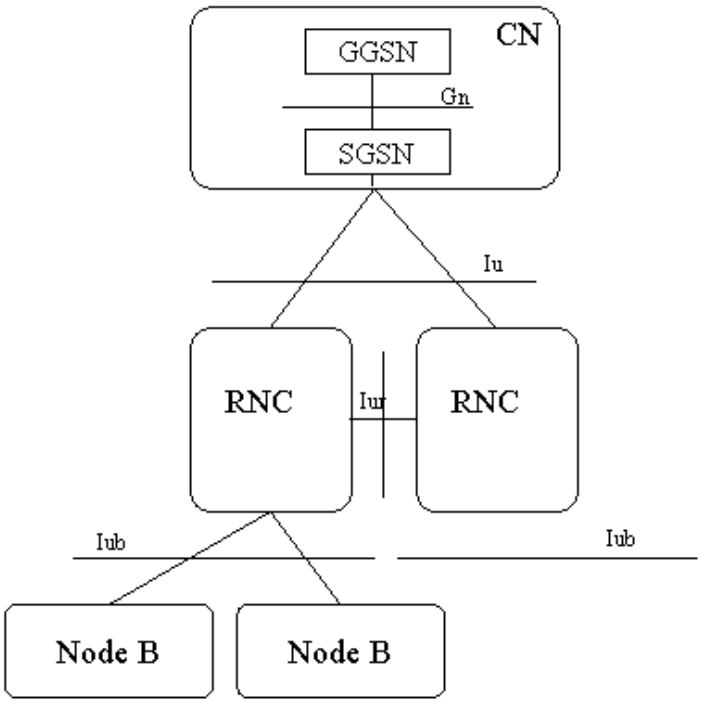

Figure 8 How RNC and Node B are interconnected

The core network uses many of the same components as GSM and GPRS. The most important components are:

- HLR, home location register, is a database located in the user's home network that stores the user's service profile.
- MSC, mobile services switching center
- VLR, visitor location register
- GMSC, gateway MSC
- SGSN, serving GPRS support node
- GGSN, gateway GPRS support node

External components can also be connected to the UMTS core network. These components can be divided into two groups from the core network's point of view: the CS (circuit switched connections) group and the PS (packet service group). Internet is one example of a PS while PSTN is an example of a CS.

The UE is made up of two main components: USIM and ME. The USIM, UMTS subscriber identity module, is implemented as a smart card and holds identity information about the user. Moreover, it performs authentication and encryption. The ME, mobile equipment, is simply the terminal used by the user to access the UMTS network over the Uu interface. The RLC resides in the ME and in the RNC.

5.1.3 RLC

The radio link control's main task is to control the data transmission over the wireless link [18]. The RLC also interconnects the UE and the UTRAN. It has a variety of capabilities but the most interesting in this context are the ones related to transmission.

The RLC resides above both the MAC and the physical layer in the protocol stack. It is able to handle transmission errors. These errors are a direct result from the unreliable communication that the wireless physical layer provides. Since the conditions for receiving and transmitting radio signals vary over time, UMTS needs a way to compensate for the changes. Adjusting the power level of the transmission can actually control the error rate. Doing this results in a more constant error rate than would have been the case otherwise.

The RLC has three different modes, which it can operate in to provide data transmission for different applications. They are: transparent, unacknowledged and acknowledged mode. When operating in the transparent mode, RLC simply forwards the incoming data without adding any extra functionality. The unacknowledged and the acknowledged mode share many things related to control but only the acknowledged mode provides retransmission of data. However, the unacknowledged mode provides an error free delivery of packets to upper layers. This is achieved by discarding erroneous packets so that only error free packets are delivered.

In Figure 9, it can be seen that the RLC layer fits between the MAC layer and the RRC, the radio resource control layer in the control plane. Also seen in Figure 9 is that the RLC is located in the UE and in the RNC. This leads to two things: Node B has to be traversed by every radio block and no active retransmission is done between the base station (Node B) and the mobile host.

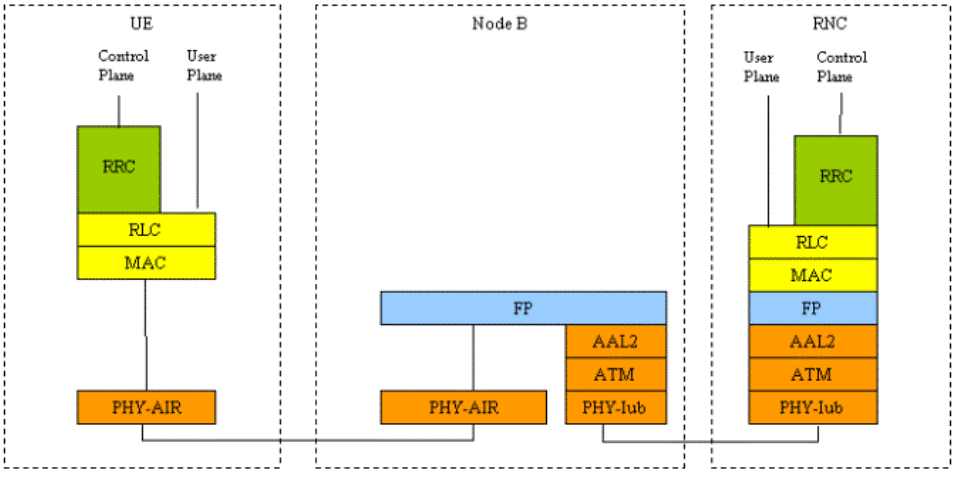

Figure 9 The protocol stack related to RLC

When the RLC is used for TCP traffic, the acknowledged mode is the recommended choice. It is always recommended to use a local retransmission scheme whenever possible.

The main focus and interest in this context is therefore the acknowledged mode, since we want to study and maximize TCP's performance over UMTS.

The RLC interacts with the networking protocol directly in the user plane and the data are passed from, e.g. IP to the RLC. The IP-packets are from the RLC's point of view called SDUs, service data units. The SDUs are segmented into PDUs, protocol data units, and passed on to the MAC-layer. To achieve higher utilization of the link, concatenation is used if the SDU does not fill the last PDU fully.

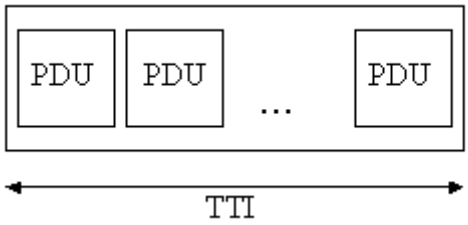

Figure 10 PU, PDU and TTI relation

The TTI, transmission time interarrival, defines how often data should be sent from the RLC layer. The amount of data that can be transmitted during one TTI is decided by the bandwidth and of course the length of the TTI. Hence, this determines the number of PDUs in one TTI, and the PDU size is set to the amount of data in a TTI of the lowest possible bit rate for the service in question. During one TTI the bits can be scrambled in order to obtain a better tolerance to variations in radio conditions.

When the RLC is operating in acknowledged mode, erroneous PDUs are retransmitted. If the receiver receives a PDU that turns out to be corrupted by bit errors, it must request the sender to retransmit that PDU. The receiver lets the sender know of an incorrect block by sending a status report. All this is handled by ARQ, automatic repeat request. In the RLC, the automatic requests are implemented by status messages. Exactly how the status messages are used is not well defined and manufactures have some freedom. The most straightforward method is for the receiver to send a status message when a received PDU is erroneous. Another way of doing this is for the sender to ask the receiver at a certain interval if everything is ok. That delay will affect upper layers, e.g. TCP, and it is important for the ARQ to have as short response times as possible.

Error detection is done using CRC, cyclic redundancy check. The CRC is calculated over the entire PDU and the RLC is thereby able to detect errors and order retransmissions.

The use of retransmissions has drawbacks. The main problem with using acknowledged mode is that it introduces extra delay. When a radio block is retransmitted, it cannot be delivered to the upper layer before it is correctly received. The delay will vary since it might take several retransmissions before the radio block is correctly received. This variation in delay is the cost for the guarantee of error free delivery. There is functionality in RLC for doing a tradeoff between retransmissions and delay. This is accomplished by providing a threshold for how many times the RLC will retransmit a specific block before giving up.

Another option is whether the SDUs should be delivered in-sequence or out-of-sequence. The simplest option of the two is the out-of-sequence delivery. It simply delivers a SDU when all PDUs associated with it have been received. The in-sequence delivery never delivers a SDU before all preceding SDU's has been delivered. The later option causes the

variation in delay to increase. Using RLC in acknowledged mode in combination with using in-sequence delivery gives the highest variation in terms of delay. There is a major drawback if out-of-sequence delivery is used: the possibility for IP packets to be reordered.

There are of course more details of how the RLC works and they can be found in [14]. The functionalities that the RLC provide can be summarized in the following points:

- Segmentation and reassembly
- Concatenation
- Padding
- Transfer of user data
- Error correction
- In-sequence delivery of higher layer PDUs
- Detection of duplicate
- Flow control
- Sequence number check
- Protocol error detection and recovery
- Encryption
- Suspend/resume function

5.2 Simulation Model

5.2.1 Introduction

A model of the radio link is needed to evaluate the possible performance improvements resulting from the different solutions presented in Section 3.1. The model allows tests to be conducted without actually implementing the proposed solutions in a real system. A model is by definition a simplification and an abstraction of the reality, and thereby the results from the experiments involving the model should only be seen as a guideline.

5.2.2 Modeling the UMTS RLC

When modeling the radio link control some abstractions are made. The actual physical layer is not modeled; the model is entirely on the RLC level. The model will of course take into consideration the errors that the physical layer will introduce. However, the block error rate is kept at a predefined level by adjusting the power of the transmission. Therefore it is assumed that the distribution of errors is uniform when considering blocks of data. This assumption is justified by the fact that increasing the transmission power will reduce the error rate. There is of course a limit to the extent the power can be increased without interfering too much with other users and base stations. This is, however, not a problem addressed here, since we assume that the network is constructed in such a way that increasing the power is possible for adjusting the error rate to the predefined level.

The segmentation used in the RLC is also considered in the model. IP packets are split into PDUs in the radio link control layer. There are two cases to consider when IP packets are segmented into PDUs: Either the IP packet size is larger than the PDU size or the reverse. In either case the model handles this as seen in Figure 11.

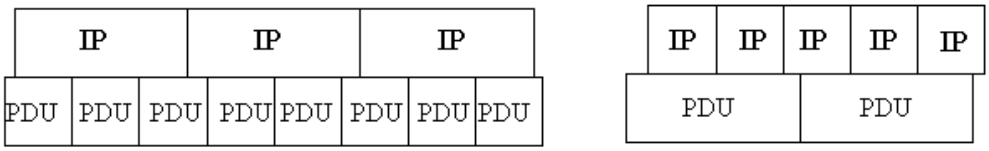

Figure 11 The IP-packets are segmented into protocol data units

After the segmentation of IP packets, the radio blocks (PDUs) are transmitted over the wireless link using the MAC and physical layer. As described previously the TTI, transmission time interval, in addition to the transfer speed determines how many PDUs that can be transferred during one TTI.

Radio blocks are retransmitted if they are found to be erroneous, when the model uses acknowledged mode. Blocks being retransmitted are given a higher priority than those who are sent for the first time and are therefore resent immediately after an error indication. It is critical to do so since the transmission time, including retransmissions, for each PDU is crucial for the performance of the link.

5.2.3 Assumptions and Description

Parameters and details must also be specified in addition to the description of how the RLC and the wireless link are modeled. The assumptions made in this context are important and they must be handled with care in order to realize an accurate model. Some of the most important attributes of the model and the surrounding area are described in this section.

The RLC can deliver data to the upper layers in-sequence or out-of-sequence, as described in Section 5.1.3. The choice of this has relevance to the simulation results, since if in-sequence delivery is chosen the delay will increase when packets have to wait for other delayed packets. If, on the other hand, out-of-sequence delivery is chosen there might be reordering of packets.

In the model we assume a specific level of errors to be used in every simulation. We are especially interested in comparing a 10% BLER with a low BLER, possibly 0 or 1%. The errors are on the RLC level and the BLER value indicates the probability that a radio block (PDU) is determined as erroneous by the receiver. For instance if a 10% BLER is used, on average every tenth radio block needs retransmission.

The model will not set a limit for how long the RLC will try to resend a packet. Since the probability of loss is assumed to be around 10 %, the probability for a packet not being correctly received after *n* retransmissions is *10^(-n)*. Obviously there will only be a very small number of packets needing as high as three or four retransmissions. The gain from setting a threshold is not always obvious since it might cause a timeout for TCP, due to a lost packet.

The cost of a retransmission is of greatest importance and will have a big effect on the results. It takes time to process a PDU through the protocol stack and must be considerate. This also includes the CRC calculation and CRC check. A reasonable assumption is that it takes 20 ms every time that the PDU has to be processed from a protocol point of view. There are, of course, two end-points and there will consequently be a delay of 40 ms only for protocol processing. Furthermore, the status report message needs 20 ms to be sent. All this time adds up to a total of 60 ms for a packet to be retransmitted, when using a TTI of 20 ms.

The model of the UMTS core network and the Internet are greatly simplified. We assume that no packet losses occur over the fixed network. If such packet losses were used in the model they can make the effects on TCP even greater, but the main thing here is to investigate the effects that the radio link cause.

Different transmission channels are available in UMTS. The channel type examined here is the dedicated channel. It means that every transport flow has its own channel. This can be seen in contrast to the shared channel strategies that let several flows share one channel. To avoid conflicts in that case, scheduling is used. Furthermore, the channel described in the model is the downlink channel.

5.2.4 Simulation Topology

The topology of the simulation is of greatest importance. It will affect the results so we have chosen a simple model to get a clearer picture of how the radio link affects TCP without the interference of other factors.

The topology comprises three nodes, each with its own representation as seen in Figure 12. The leftmost node represents the mobile host. This could in the reality be a UMTS terminal in form of a PDA or a mobile phone. The node in the middle represents the base station (Node B in UMTS) while the rightmost node represents a server residing on the Internet.

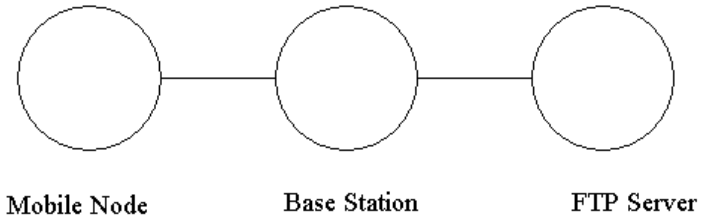

Figure 12 The simulation topology

The delay is one of the most important characteristics of the model and it must be selected with care since the performance of TCP is totally dependent on it. The delay of UMTS (CN and UTRAN) combined with the Internet is assumed to be around 135 ms one-way. Out of these 135 ms, 60 are accounted for the delay in UMTS while the rest (75 ms) represents the Internet delay. The delay in UMTS are in turn made up of two parts: the delay resulting from the core network are assumed to be 20ms, while the delay for the radio access network is assumed to be 40 ms.

The transmission speed of the radio link model is adjustable and it ranges from 64 to 384 kbps. How often the different speeds are actually used is not specified, but it is dependent of the quantity of users sharing the same cell. The more users, the lower the speed.

A quantification of the problems with high error rate will be made under the assumption that a large file is downloaded is shown in Section 5.5. A few proposed solutions will be tested in order to investigate their ability to reduce the problem. Downloads of small files will be examined in Section 5.4, while Section 5.5 gives the result from using smaller PDUs.

5.3 Simulations of Large File Transfers

Simulations are done in order to characterize and examine to what extent TCP is affected by the high radio block error rate. All simulations are done in NS2, network simulator version 2 [17]. NS2 is described in detail in Appendix 1. The radio link model is implemented as a module in the simulation environment of NS2. A description of the implementation and its usage can be found in Appendix 2.

To see how the performance, of the transmission, is affected when downloading data using TCP, a large file of 16 Mbyte is downloaded from the Internet to the mobile host. By using such big file, the steady state of TCP can be studied. The time for this transfer is measured, and from this data the utilization is calculated. The parameter values of the features described in Section 5.2.2 are shown in Table 2. Of course some other parameters, e.g. packet size, also need to be specified but since they will be different for every simulations they are specified in conjunction with each individual simulation. The parameters listed in Table 2 are used in all simulations described in this section.

Table 2 Parameter values

Parameter	Value
TTI	20 ms
Retransmission delay	60 ms
PDU size	1 TTI
UL Error Rate	0%
DL Error Rate	0% vs. 10%
Network delay	135 ms
File Size	16 Mbyte

The speeds 384, 128 and 64 kbps are considered. The reason for choosing these specific speeds is that they will initially be the most commonly used in UMTS.

The maximum send window is set so that the capacity of the link is fully utilized when the window reaches its maximum. The BDP indicates how much data can be in transfer at the same time at the maximum capacity, and the window size is set to a higher value. To use an unlimited maximum window would be unrealistic due to implementation issues, e.g. receiver buffer size. The maximum window sizes used here have been verified as large enough for full link utilization of an error free link.

The amount of data being transferred includes the size of the IP header (usually 40 bytes). If, e.g. a packet is 1500 bytes, the amount of data accounted for is also 1500 bytes, even though 40 bytes in the reality is used for the IP header. This approach makes the difference in overhead from IP transparent. This is reasonable, since we are only interested in the utilization of the link and not the amount of overhead that IP headers introduce. Hence, this is how the bit rates 384, 128 and 64 kbps should be interpreted, i.e., as IP layer throughput.

5.3.1 TCP Reno with and without Radio Block Errors

TCP Reno is one of the most common implementations of TCP and it has the behavior described in Section 2.1. To understand how big the reduced utilization is, simulations without any modification to TCP are needed. The objective of these simulations is to examine how much extra transferring time the 10% BLER in UMTS causes when the Reno implementation of TCP is used. The speeds 384, 128 and 64 kbps have been simulated.

Table 3 Increase in download time for different speeds and packet sizes

Speed (kbps)	Packet Size (bytes)	Max Window (segments)	Increase in Time (%)
384	576	25	239
384	1500	14	25
128	576	11	19
128	1500	7	11,3
64	576	9	11,1
64	1500	6	11,4

The best possible utilization still takes 11% more time when 10% BLER is used, as seen in Table 3. These 11% arise from the 10% radio block retransmissions, and are not due to TCP congestion control. This effect is considered a small price to pay in relation to the cost savings of building a mobile network with high BLER instead of a lower BLER (1%).

The results show that 64 kbps does not cause any problems for TCP's congestion control. Neither does 128 kbps in combination with 1500 bytes packet size. The cases when 64 kbps speed is used are considered to be satisfactory and no extra investigation to improve the performance is therefore needed.

The cases that cause problems are when the 384 kbps channel is used and when 128 kbps is used in combination with small packets. These results correspond well with earlier results [11]. Using a speed of 384 kbps and a packet size of 576 bytes shows a very significant increase in download time.

A new calculation is made to show the underutilization when the RLC retransmissions are not considered (that introduces 11% increase), and only the effect of the TCP congestion control is considered. In Table 4 the modified values are shown. They are calculated as follows: (transfer time-(10% of transfer time))/ideal time.

Table 4 Increase in download time with the retransmission time of RLC excluded

Speed (kbps)	Packet Size (bytes)	Max Window (segments)	Increase in Time (%)
384	576	25	183,18
384	1500	14	12,63
128	576	11	7,1
128	1500	7	0,1

The degradation in the performance is large in the case of 384 kbps while still not small enough to be neglected at the speed of 128 kbps, as seen in Table 4,. In the following simulations the result that will be shown will include the 11%. So results should be compared to Table 3, and the reader should bear in mind that 11 % is the optimal value.

5.3.2 TCP Reno Congestion Window

The retransmissions can cause IP-packets to be reordered as described in Section 5.2.5. If a packet is delayed long enough, three or more packets may pass the retransmitted packet, forcing TCP to do a fast retransmission. The receivers advertised window will affect the simulations since the sender's window is limited by the minimum of the congestion window and the receiver's advertised window. In all simulations the advertised window is set as small as possible but without limiting the throughput when an error free link is used. Simulations have been made to verify the choice of window size.

The increase in download time is a direct result of TCP interpreting the reordering as congestion and limits the send window accordingly. In the following figures the send window over time is shown to visualize TCP's performance degradation.

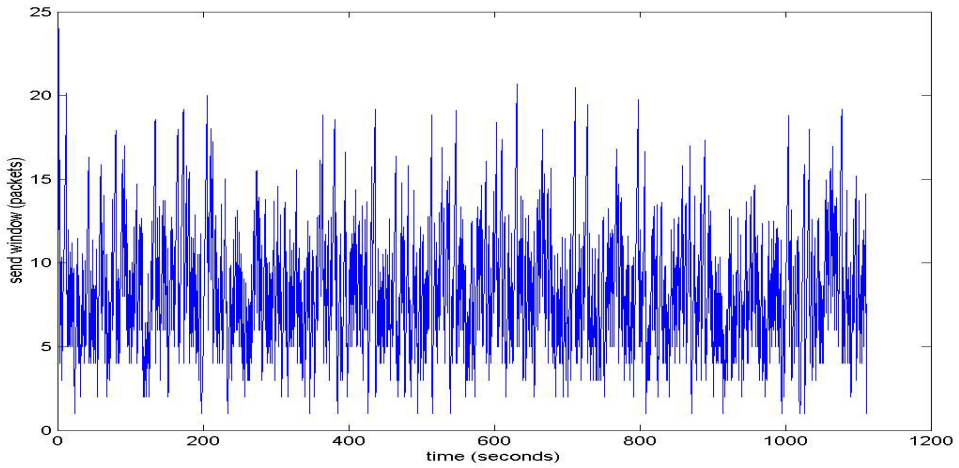

Figure 13 Send window for 384 kbps with a packet size of 576 bytes

Figure 13 shows that the congestion window varies a lot. The degradation of the utilization is obvious since the link is only fully utilized then the congestion window is near the maximum send window of 25 segments.

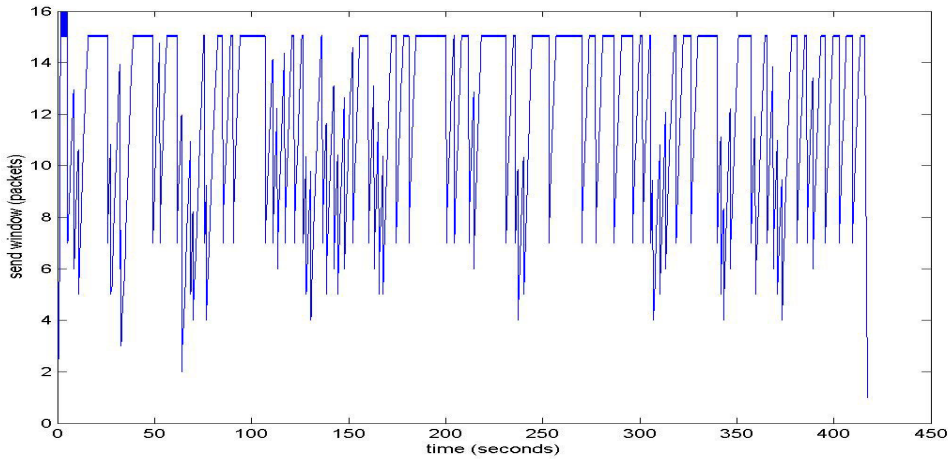

Figure 14 Send window for 384 kbps with a packet size of 1500 bytes

Here in Figure 14, the effect is not as big but the throughput degradation is still as big as twelve percent. The horizontal sections of the plot, at the value of 15, indicate that the link is fully utilized at some times.

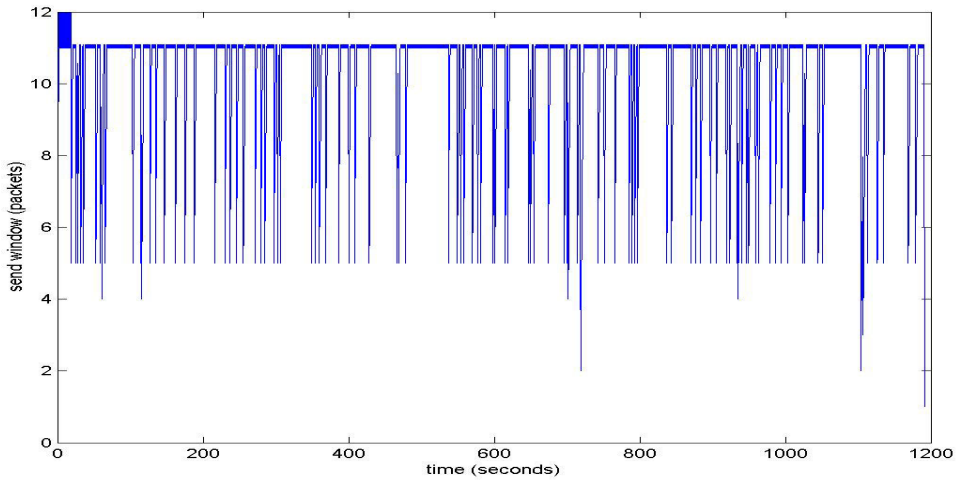

Figure 15 Send window for 128 kbps with a packet size of 576 bytes

In Figure 15, the influence of TCP's congestion control on the utilization is not as big as in the previous two figures. The effect corresponds to an increase in download time of seven percent. The figure also serves as a good illustration of the congestion window being halved when a fast retransmission occurs.

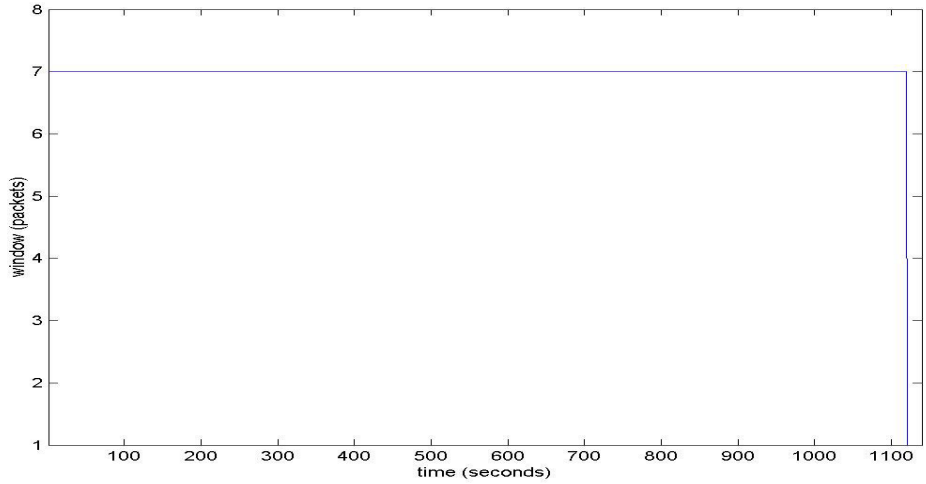

Figure 16 Send window for 128 kbps with a packet size of 1500 bytes

When TCP do not do any fast retransmission due to imparities in the link, the congestion window will not be decreased at all. In Figure 16, the utilization is at the maximum level and the send window is fixed at seven segments all through the transfer[2].

The send window alternates the most when the packet sizes are small and the transfer speed is high as seen in Figure 13 to Figure 16. The fluctuation of the send window seen in the figures, correspond well to the measured download time. TCP's handling of the congestion window, due to high BLER, causes the link not to be utilized to the extent that one would desire.

While the problem is severe at 384 kbps, almost no effect can be seen when using a transfer speed of 64 kbps. In the search of a solution to the underutilization problem, only the link speed and packet size combinations with a problem are considered (and therefore simulated).

5.3.3 Eifel and Westwood

To address the problem of underutilization of the radio link, quantified in the previous section, the solutions described in Section 3.1 are used. As we remember from Section 3.1.4, Eifel improves the performance of TCP when spurious fast retransmissions are common. This is exactly the problem that the UMTS radio link causes, under our assumptions.

TCP Westwood on the other hand does not attempt to detect false fast retransmissions; instead it changes the limitation of the congestion window for all fast retransmissions. This is done by bandwidth estimation as described in Section 3.2.5. The result is that Westwood does not reduce the window as much as Reno and thereby it improves the utilization.

Both the simulation of Eifel and TCP Westwood is made under the same assumptions as the simulations using TCP Reno. In Table 5, the results obtained using Eifel can be seen, while Table 6 shows the results obtained using Westwood.

[2] Except for an initial slow start phase, which is to short too be seen in the figure.

Table 5 Increase in transfer time using TCP with Eifel

Speed (kbps)	Packet Size (bytes)	Max Window (segments)	Increase in Time (%)
384	576	25	34,2
384	1500	14	13,6
128	576	11	12,5

Eifel almost eliminates the TCP problem in the case of a 128 kbps link as seen in Table 5. The highest speed (384 kbps) still cause an increase in download time due to TCP, but not to the same extent. Note that the increase in transfer time in the case of 384 kbps with a packet size of 576 bytes has decreased from roughly 240% to 34%.

The results when using TCP Westwood can be seen in Table 6. The results are roughly the same as for Eifel.

Table 6 Increase in transfer time using TCP Westwood

Speed (kbps)	Packet Size (bytes)	Max Window (segments)	Increase in Time (%)
384	576	25	33,6
384	1500	14	12,8
128	576	11	13,8

TCP Westwood and Eifel do not prevent TCP retransmissions, which are rather frequent, but instead try to limit the effects from them. Thus, much of the 34% are due to unnecessary IP packet retransmissions and the window only limits the throughput to a fairly low extent.

5.3.3 Split TCP

One of the main reasons for Reno's bad performance is the big delay over Internet and the UMTS network, since the TCP window recovers slowly from congestion when the RTT is large. Split TCP, described in Section 3.1.2, can actually help to reduce the effects by reducing the delay seen by TCP. Each TCP flow will have a smaller RTT since the window handling will be separated over the two parts of the path. Hence, the TCP window can recover faster by introducing a Split TCP scheme.

There is no support for Split TCP in NS2[3], but a simplification still enables the simulation to be completed. Simply removing the delay over the fixed network gives a good approximation, since the limiting factor, over long time, is the wireless link. Depending on where the TCP connection is split, different fixed network delays should be considered. It is reasonable to place the TCP proxy between the RNC and the SGSN. This is assumed

[3] No Split TCP add-no module for NS2 has been found either.

here and the delay from the radio link's physical layer to the TCP proxy is assumed to be 30 ms.

Table 7 Increase in transfer time using Split TCP

Speed (kbps)	Packet Size (bytes)	Max Window (segments)	Increase in Time (%)
384	576	25	75,6
384	1500	14	12,5
128	576	11	11,6

The data from simulations using Split TCP are shown in Table 7. The result from the case when 384 kbps is used in combination with a packet size of 576 bytes is rather good, compared to the original results using Reno, but not as god as with Eifel and Westwood.

5.3.4 In-order Delivery

The SDUs, in this case the IP packets, can be delivered to the mobile node's IP layer in the same order that they are received at the sending base station. This is in contrast to delivering them in the order that they are actually received without errors. The risk that the radio link reorders IP-packets are eliminated, but at the cost of larger variation in the delay. Many packets can be delayed by only one erroneous packet since every packet received during a RLC retransmission has to wait for the retransmitted packet.

The Reno implementation is used to quantify the problems of using in-order delivery. The results from using the facility to deliver SDUs in order can be seen in Table 8. Again, the case of 384 kbps in combination with 576 bytes packet size causes the most problems. The increase in time is 53%. As with the other solutions, the lower speeds do not cause any significant problems.

Table 8 Increase in transfer time using in-sequence delivery

Speed (kbps)	Packet Size (bytes)	Max Window (segments)	Increase in Time (%)
384	576	25	52,87
384	1500	14	12,57
128	576	11	11,12

5.3.5 Comparison of the Modifications to TCP

A comparison of the accumulated data transferred is shown in Figure 17. The quantity measured is the amount of data received. The comparison is for the speed of 384 kbps in combination with a packet size of 576 bytes, i.e. the worst case. The rate of data received is almost the same for Eifel and TCP Westwood while Reno is far behind. Split TCP manages to reduce the problem significantly but as seen the result is not as good as for in-sequence delivery.

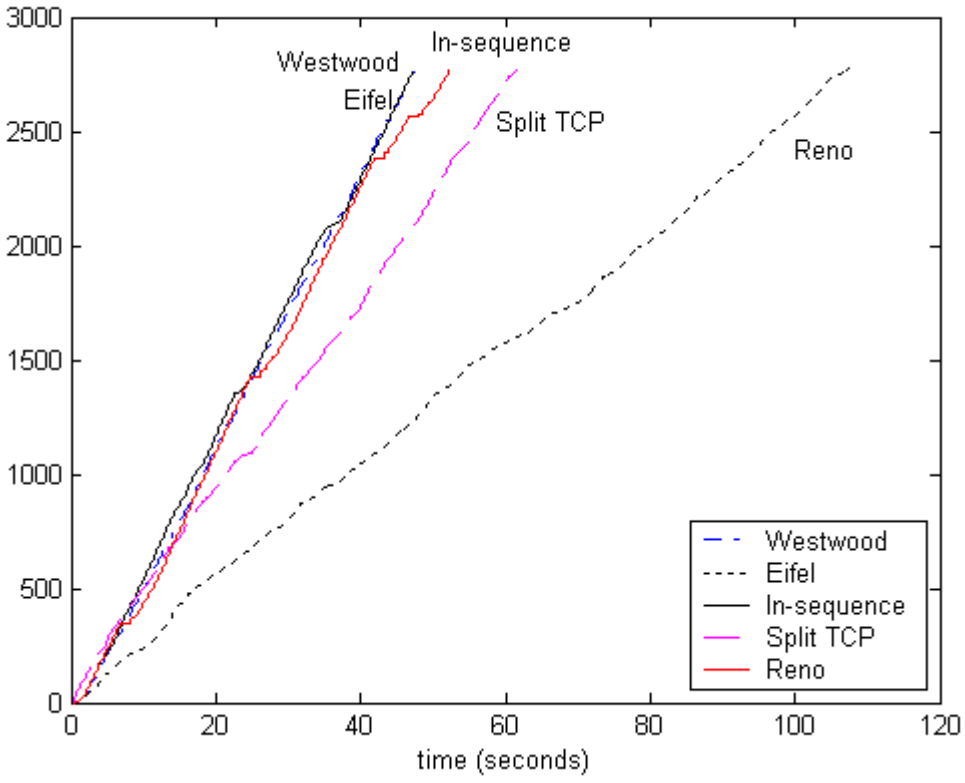

Figure 17 Comparison of the data received and acknowledged for different TCP versions

5.4 Simulations of Small File Transfers

Another important scenario to study is when a user wishes to download a small file. This case differs in many ways from the case of a large file. The most important difference is that a bigger proportion of the connection time will be spent either setting up the connection or in the slow start phase. To quantify the problem, simulations of small file transfers are needed.

5.4.1 Quantifying the Problem

By small files we mean sizes of less than 200 kB. Which size in this range that are the most common can be discussed but 100 kB seams reasonable. This file can for instance be a web page being downloaded with HTTP 1.1. If an older version of HTTP is used, the objects in the web page would be downloaded separately, each using its own TCP connection and the file sizes would then be smaller. This would cause the downlink utilization to be even worse.

In the following simulations the receiving part of the TCP connection is set to acknowledge every segment. Note that most TCP implementations acknowledge every second segment, which results in a even slower initial TCP window increase.

Table 9 Increase in download time of 100 kByte for different speeds and packet sizes compared for 10% error rate vs. an error free link

Speed (kbps)	Packet Size (bytes)	Max Window (segments)	Increase in Time (%)
384	576	25	94,5
384	1500	14	17,3
128	576	11	16,6
128	1500	7	10,2
64	576	9	11,5
64	1500	6	11,6

The figures in Table 9 are the results from calculating the mean value of twenty simulations. The variation in the results from the individual simulations is quite large. The results will be affected by the time when the retransmissions occur since the simulations are short (only 100kByte). As seen in Table 9, the problems in terms of performance degradation are not as significant as with larger files. The figures does not show the whole truth, since they are only indicating how much longer the transfer takes when the link has 10% errors and uses retransmissions versus an error free link. The actual utilization is low even in the first case, when the link is error free, and this is mainly due to the slow start phase being large in relation to the total transfer time. Furthermore, the time to set up the connection, using the three-way-handshake, is a significant part of the total connection time. The time before data can be sent is about one RTT. Even when the speed of the link is 384 kbps, the actual transfer speed is only around 250 kbps for an error free link as seen in Table 10. The table shows the average transfer speeds with 10% error rate and also without errors.

Table 10 Average transfer speeds when downloading a 100 kByte file

Link Speed (kbps)	Packet Size (bytes)	Actual Speed 0% errors (kbps)	Actual Speed 10% errors (kbps)
384	576	224,09	107,61
384	1500	251,57	215,34
128	576	109,73	94,06
128	1500	113,96	104,17
64	576	61,11	54,89
64	1500	61,53	55

In conclusion, using a low BLER instead of 10% would roughly increase throughput 10-20 % but at a high network infrastructure cost. However, lowering BLER for 384 kbps and small packets is much more beneficial.

5.4.2 Increasing the Initial Congestion Window

Increasing the initial congestion window is one way of mitigating the problem of the big proportion spent in slow start. This will allow the start up to go faster and thereby increase the utilization. It has been discussed by how many segments the initial window should be increased. Note that there is a negative side of increasing the initial window size: The traffic will be burstier. The window should at least be increased to two segments, which is allowed but there are also suggestions for allowing a size of three or four. In [1], the formula min(4 SMSS, max(2 SMSS, 4380bytes)) is presented but it is not a part of the standard. Note that the receiver as well as the media can limit SMSS, sender maximum segment size. The intuition behind the formula is that more segments can be sent initially if the segments are smaller. The formula gives an initial window of 4 segments for 536 bytes SMSS and a window of size 3 for 1460 bytes SMSS.

The results from simulating a download of a 100 kByte file using an initial window size of 3 and 4 are shown in Table 11,. The improvement is measured in decreased download time when using an initial window size of three or four instead of one.

Table 11 Decreased download time for a 100 kByte file, due to increased initial window. Increasing the window from 1 to 3 or 4 segments

Speed	Packet Size	Init Window size	Improvement
384	576	3	6,47%
384	1500	3	17,38%
128	576	3	5,74%
128	1500	3	5,66%
384	576	4	5,75%
384	1500	4	21,32%
128	576	4	7,85%
128	1500	4	6,65%

The largest gain can be made when the slow start phase is significant in combination with a large packet size. The difference in improvement is not that big between an initial window of three and four segments.

The following two figures are presented to illustrate how the initial window affects the transfers. The first figure, Figure 18, shows the accumulated data during a transfer when an error free link is used. Figure 19 shows an example of traces when an error rate of 10% is used. The speed of 384 kbps and a packet size of 576 kBytes are used in the simulations. As seen in the first figure, the difference lies in how fast the connection is able to reach its maximum speed. Increasing the window size has largest effect on files really small files, e.g., 20 kB as seen in Figure 18.

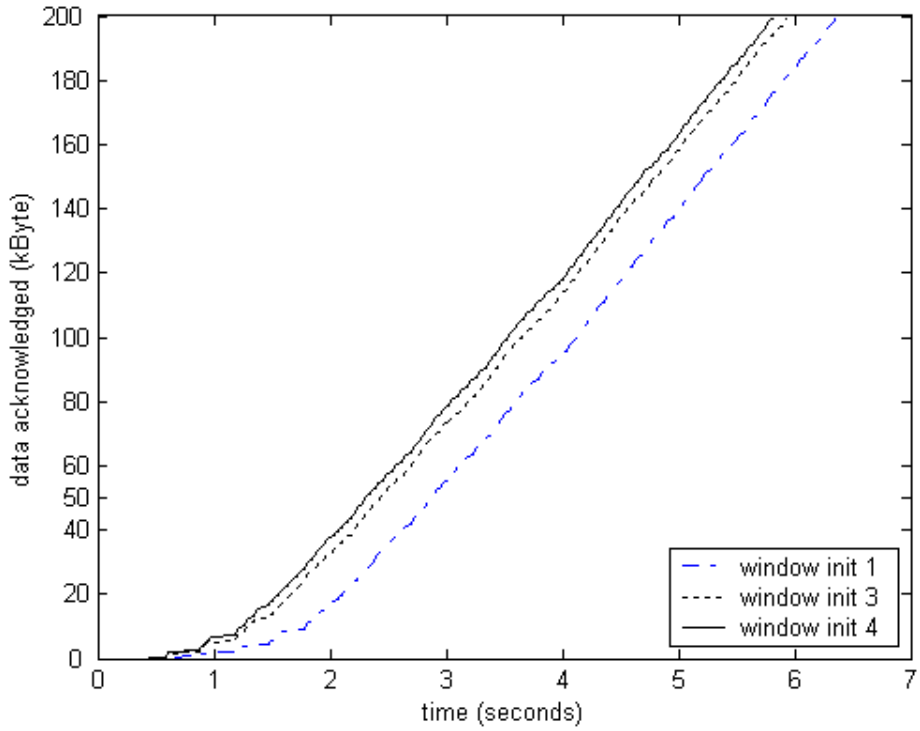

Figure 18 Ideal simulation with no errors showing the difference between an initial window of 1, 3 and 4 segments

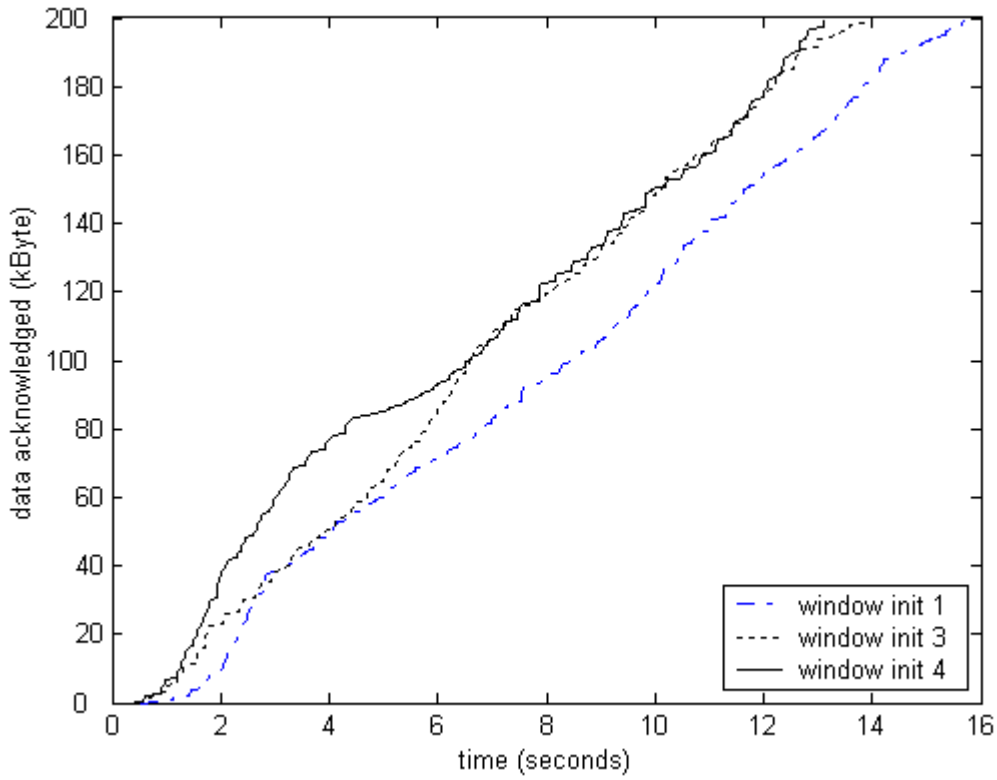

Figure 19 Trace of simulations of small files for an initial window of 1, 3 and 4 segments with 10% BLER on the radio link

Figure 19 shows only one simulation trace of each initial window size. The actual transfer speed of the link varies considerably between the simulations. The reason for this is the random errors that occur over the wireless link. The numerical results shown in the tables are the mean over 20 simulations.

5.4.3 Different File Sizes in combination with different Initial Windows

Two other cases are also simulated in order to study how the results are affected by the file size. The new file sizes are 50 kByte and 200 kByte and are chosen to assess the improvements. Table 12 and Table 13 show the result from 50 kByte and 200 kByte files, respectively. Only the window size of three was used to study the improvement since the gain from increasing the window from three to four segments is rather small: see Figure 18.

Table 12 Decrease in download time for a 50 kByte file, due to increased initial window

Speed	Packet Size	Init Window size	Improvement
384	576	3	10,60%
384	1500	3	20,95%
128	576	3	6,86%
128	1500	3	10,13%

Table 13 Decrease in download time for a 200 kByte file, due to increased initial window

Speed	PacketSize	Init Window size	Improvement
384	576	3	3,36%
384	1500	3	1,98%
128	576	3	4,07%
128	1500	3	3,43%

Small files in combination with large packet sizes is the case where it is most beneficial to use larger initial window. This is no surprise, but the improvement in transfer speed of 20% is quite large. For 200 kByte files the increased initial window gives only a minor improvement.

5.5 Simulations using Multiple PDUs in one TTI

Along with the assumptions made in the previous simulations, there is the possibility for one TTI to hold several PDUs. Using PDUs as small as 40 bytes is suggested. Considering the amount of data that fits into one TTI when the speed is 128 kbps, there will be a large amount of PDUs in every TTI. The retransmission probability for each PDU is still the same since the error rate for them are still considered to be around 10%. But since more PDUs will make up an SDU the probability that an SDU will be delayed increases, causing a higher delay. To quantify the throughput problems when using multiple PDUs, simulations of large file transfers are again carried out.

5.5.1 In-order Delivery vs. Out-of-order Delivery

When using the multiple PDU approach there is a choice whether to use in-sequence or out-of-sequence delivery. As we remember from earlier, the in-sequence delivery delivers IP-packets in the same order as they are received at the base station while the out-of-sequence delivers each packet as soon as they are received correctly. Due to extensive retransmissions on the link, the packets may be substantially delayed when using in-sequence delivery or extensively disordered if out-of-sequence delivery is used.

The first case to study is when out-of-sequence is used in combination with 40 byte PDUs using the Reno implementation of TCP. Table 14 shows the results from these simulations. The results for the higher speeds are terribly bad.

Table 14 Increase in download time for different speeds and packet sizes using 40 byte PDU

Speed (kbps)	Packet Size (bytes)	Max Window (segments)	Increase in Time (%)
384	576	25	676
384	1500	14	388
128	576	11	184
128	1500	7	18,7
64	576	9	11,4
64	1500	6	11

One possible solution to the bad performance is to use in-sequence delivery. Although it does not solve the underutilization problem, it reduces the effect to a large extent. It can be seen that there is only a major problem with 384 kbps and 576 Bytes packets in Table 15.

Table 15 Increase in download time for different speeds and packet sizes using 40 byte PDU and in-sequence delivery

Speed (kbps)	Packet Size (bytes)	Max Window (segments)	Increase in Time (%)
384	576	25	305
384	1500	14	14,3
128	576	11	18,7
128	1500	7	11,5
64	576	9	11
64	1500	6	11

The bad result in out-of-sequence delivery is due to IP packet reordering. The reason for this extensive reordering is that RLC retransmissions occur and many packets will need several retransmissions before they are ready for delivery. If on the same time a few packets of higher sequence number pass by such a significantly delayed packet it causes a fast retransmission. TCP's inability to tell the difference to reordering on the wireless link and congestion is also here the root of the problem.

The introduction of in-sequence delivery does not solve the utilization problem even though it removes the fast retransmissions. Similarly to when in-order delivery is used in combination with PDUs of the size of a whole TTI, the case of smaller PDUs also cause timeouts to TCP. The timeouts occur most often when the speed of 384 kbps is used.

5.5.2 In-sequence Delivery using TCP Westwood, Eifel and Split TCP

In-sequence delivery managed to reduce the problem of using small PDUs substantially. However, for the speed of 384 kbps, in combination with a packet size of 576 bytes, the problem is still significant. An increase of some 300 % is not an acceptable price to pay for the 10 % BLER used.

The suggested solutions may be used to try to mitigate the low utilization when using in-sequence delivery in combination with small PDUs. The reduced transfer time are quantified by simulation of TCP Westwood, Eifel and Split TCP. We show a comparison of the different solutions in Figure 20.

The first solution we look at is Westwood. Again, we download of a large file and compare the result to using Reno with 0% and 10% error rate, respectively. TCP Westwood does actually reduce the problem, and the increase in download time versus Reno with 0% BLER is 120 %, to be compared with 305 % for Reno with 10% BLER. While still a quite large number, the reduction is noteworthy.

The next solution considered is the Split TCP approach. We do the same assumption now for Split TCP as previously and set the delay to 30 ms over the wireless part of the path. A reduction of the recovery time of the TCP's congestion window is achieved, and as we see in the result from the simulations that the utilization is now almost as good as it can be when using 10 %.

Eifel is also considered and the simulations indicate that Eifel is better than Westwood, but still gives worse performance than the Split TCP's performance. An increase of 50 % compared to Reno 0% BLER leaves the Eifel at a rather large increase even though the link utilization is greatly improved.

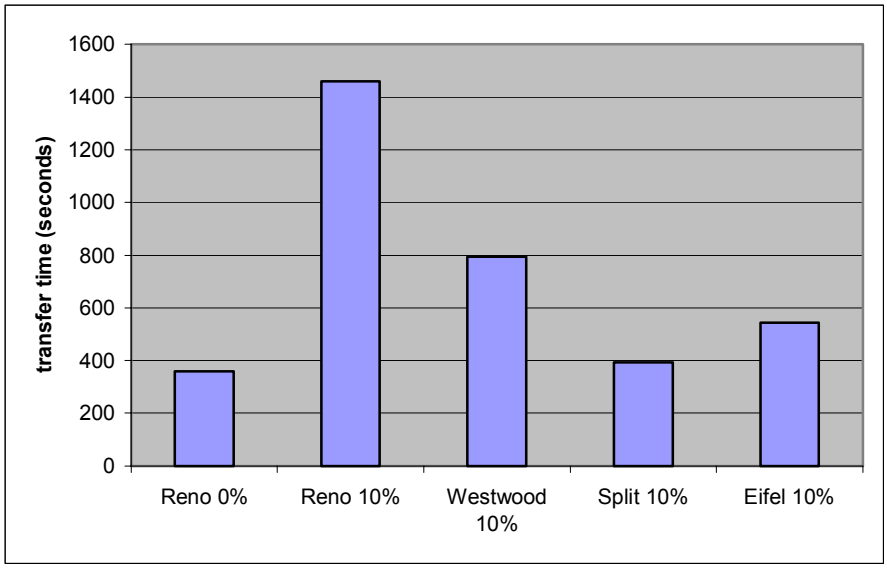

Figure 20 Comparison of different solutions for multiple PDU at the speed of 384 kbps in combination with 576 bytes packet size

In conclusion, the best result is achieved when Split TCP is used, but Eifel also gives a rather good improvement. The reason for Split TCP's success in combination with in-sequence delivery while it was not as good with out-of-sequence delivery, is most probably the result of the RTO. Split TCP is not affected as much by a higher variation in the RTT as it is by the reordering of the packet. Split TCP manages to evade timeouts by better estimation of the RTO threshold. Since the delay variation is greater in comparison to the fixed delay, TCP does not set the RTO as low as it would have done otherwise.

6 Analysis

6.1 Introduction

We must first realize that some of the solutions have totally different semantics and as a result from this they can be hard to compare. This is especially true if more factors than the simulation results are considered. Not only do they give different results, they also have different cost for introduction into the UMTS system.

6.2 Discussion

All simulations done to investigate the problems are based on the same model. The model includes the Internet, CN and UTRAN but is rather simplified. The delay is as previously mentioned set to 135 ms in order to account for routing in the Internet and also the traversal of the UMTS network. Packet losses on the Internet were not included in the model. This may have impact on the result but since packet losses are in general rare on the Internet, the model is relatively accurate. Another simplification is that the uplink is assumed to be error free. This is not the case in reality and if uplink errors were introduced the results would probably be even worse. The model is of course an abstraction but the most important part in this context, the RLC, is relatively accurately modeled.

The decrease in performance seen in the simulations comes from two different areas: First, the retransmission of radio blocks limits the throughput, since a retransmitted radio block uses capacity that otherwise could have been used by another block. Second, the TCP congestion control limits the utilization by restricting the sending of enough data to fill the pipe. Furthermore, TCP retransmissions also "steal" capacity. The first reason (that the RLC retransmissions "steal" capacity) is not something that can be solved without lowering the BLER and it is considered to be a small price to pay related to the financial benefits that can be made in the network construction. The capacity of each base station is utilized better if the BLER is increased because the transmission power for each user is reduced. This implies that there will be more capacity in the network if the TCP problem could be solved. Hence, different solutions to the TCP problem are investigated.

The results from the simulations show that there is a great gain in the utilization of the link when using either Eifel or Westwood. The benefit from using these is greatly dependent on how bad the utilization is in the first place. The largest improvement can be seen when the utilization was really poor when using Reno.

Both TCP Westwood and Eifel give similar results but the improvement comes from different modifications to TCP. Eifel adjusts its window to the old size after finding out that the retransmission was unnecessary. TCP Westwood on the other hand estimates the available bandwidth and does not limit the congestion window as much as TCP Reno. Since the radio link is the limiting factor in terms of bandwidth, Westwood uses almost all of that bandwidth. The advantage of Westwood is that it improves performance when there is congestion in the network as well. This is accomplished by a rather big modification of TCP. Eifel's main advantages are its simplicity and the fact that is already implemented in actual operating systems (e.g. Linux) [19]. Furthermore, Eifel is advancing through the standardization [28].

Split TCP also gives a rather good improvement in performance. This is dependent on the fact that the RTT for the TCP connection over the wireless link is smaller. Depending on

where the TCP connection is split, different results will be achieved. We have made the reasonable assumption that the split should be placed between the RNC and the SGSN. The closer to the mobile host that the split is done, the lower the RTT. Lowering the RTT will result in a better performance. An evaluation of the performance of I-TCP in general can be found in [27]. It is evident that not only the usage of a Split TCP approach is important, but the actual implementation and its associated algorithms are also of outmost importance.

The introduction of in-sequence delivery of SDUs gives a totally different characteristic of the TCP connections behavior. The reordering of IP packets are no longer present, but instead the risk for TCP timeouts is introduced. The timeouts are triggered by a sudden change in RTT, caused by a PDU being retransmitted several times. If the threshold for timeouts would be higher, the problem would not be as big. However, raising the threshold would have implications on the performance of TCP in general, e.g. in severe congestion situations.

Multiple PDUs in one TTI gives interesting results. The performance is really bad if a PDU size of 40 bytes and out-of-sequence delivery are used. The result can be improved by using in-sequence delivery. However, even when the in-sequence delivery is used there is still a big underutilization, and to tackle this some of the suggested solutions are again considered. Both Eifel and Westwood manages to reduce the increase in download time significantly but the best solution in this context is Split TCP, which almost eliminates the problem. Eifel is better than Westwood in dealing with the spurious timeouts triggered by the radio link. One thing is clear, the problems that arise from using small PDU needs attention and some solution ought to be used for high speeds.

For small files the utilization is overall low. Ten percent BLER gives a poor performance but since the performance is poor even without errors another strategy than lowering the BLER is needed. Increasing the initial window is generally good and the result from this is satisfactory for 64 kbps and somewhat good at a speed of 128 kbps. Increasing the window from 3 to 4 segments provides only a minor improvement, and a window size of 4 is not considered to be worth the price of higher congestion probability. A general observation is that increasing the window gives a higher increase in performance when the packet size is large. The 384 kbps speed needs some additional measure since there is a big underutilization In-sequence delivery may improve the performance even for small files (this has not been tested in this study), but errors on the uplink, acknowledging every second packet and Internet congestion may imply that 128 and 64 kbps radio bearers also needs additional attention.

There are of course other suggestions available to how TCP's problems of handling the imparments of the radio link. One very interesting suggestion is to change the threshold for fast retransmissions dynamically. Different schemes for updating the threshold are investigated in [22]. Moreover, changing the way the retransmission timer is calculated may also be worth investigating. The current method obviously has its limitations [24].

From the UMTS network operator's point of view, not only the simulation results are interesting. Maybe even more important is the cost and feasibility for introducing these solutions into the network. The modifications to TCP are very attractive since both Eifel and Westwood improves the performance significantly while no cost in term of equipment is needed (from the network operators perspective). The problem with this is that all TCP implementations have to support the algorithms. Both Eifel and TCP Westwood are

applied to the sender side of the TCP connection. In the case studied, i.e. when a file is downloaded, the modifications are only needed to the server side of the connection[4]. Although the upgrade to TCP variants addressing these problems may be done in the future, the network operator has little means to force the migration to be done as soon as possible.

One advantage of Split TCP is that it can be deployed immediately into the network since the TCP sender and the TCP receiver, i.e., the parts that the operator has little or no control of, do not need any updating. Split TCP also has some drawbacks. For instance, in the case of handovers, exchange of information between TCP proxies might be needed. Furthermore, IPSec will not work on an end-to-end basis. Therefore, the network operator cannot deploy the Split TCP without a moderate amount of effort.

Changing the configuration of the RLC is on the other hand something that the network operator can do at a low cost. The parameters that could be changed are the delivery order, the TTI and the selection of how small the PDUs will be. There are of course some restrictions on how these parameters can be changed and consideration has to be made to radio transmission issues as well as to optimize for TCP. Note that TTI and PDU sizes will affect the required signal power and thereby create more radio interference, which in turn affects the capacity of the network. This is a complicated situation with many factors involved. However, the most important parameter that this thesis is based on is the BLER. 10 % is attractive when dimensioning the network.

[4] For Eifel also the receiver might need to be updated if it does not support the time stamp option.

7 Conclusions

The different problems associated with using TCP over wireless links, as the one used in UMTS, have been both discussed and quantified in this thesis. The implications of a high BLER on TCP's congestion control are in fact the core of the problem. In the previous section we have discussed the results and the problems, and from this discussion we now point out some conclusions.

The wireless link is controlled by the RLC. The RLC has several important tasks to perform. The two possibly most important tasks are the segmentation of IP packets into radio blocks and the process of resending those blocks until they are correctly received. Furthermore, the RLC has an option to deliver IP packets in-sequence or out-of-sequence. Because of the RLC's and the wireless media's important characteristics, we have modeled and implemented them into the simulations environment of NS2.

First, simulations to quantify the problems were conducted. We saw that there were severe problems, especially at high speeds and with small IP packets. Due to these problems we tested a few proposed solutions to see how well they serve as a remedy. The tested solutions are: TCP Westwood, Eifel, Split TCP and also some RLC parameter optimizations. Small files are also used for simulation and the size of the initial window affect on the performance is quantified.

7.1 Results

To quantify the problems, TCP Reno is used for simulations over the wireless UMTS link model. The radio block error rate is set to 10 % and a large file is downloaded. The results from these simulations show that there are problems that TCP does not handle well. The worst results are obtained at the speed of 384 kbps in combination with small packet sizes. However, the lower speeds does not reach the maximum utilization either, due to the fact that TCP perceives the packet reordering, caused by the high BLER, as congestion. However, the lower speeds do not at all present a problem to the same extent. For instance, using 10 % BLER over a 64 kbps channel does not present any underutilization at all (only radio link retransmissions steal capacity). 128 kbps is also relatively free from implications and can be used without any extra measures if a small underutilization can be accepted.

It becomes apparent that the choice of using in-sequence or out-of-sequence delivery is of greatest importance when looking at how the network can be adjusted to deal with the problems in a good way. In general, the in-sequence delivery ought to be used since it gives better results for all transfer speeds. It is also recommended to use PDUs of the same size as the TTI when looking from a utilization point of view.

Short file transfers have different characteristics than long transfers, since a major part of the connection time will be in the start up phase. From the simulations of this scenario we come to the conclusion that a large initial window is preferable. Increasing the window size to three segments improves the performance, but not to the extent one would desire.

To deal with the problems arising from the high BLER, different TCP approaches are used (in addition to adjusting the delivery order). Eifel and TCP Westwood gives similar results generally. Eifel handles spurious timeouts better then Westwood, but Westwood on the other hand is believed to handle congestions in a better way. A Split TCP approach is also tested and it is found to out-perform Eifel and Westwood when the risk for spurious

timeouts is high. Moreover, Split TCP might also be of interest for short file transfers, especially in connection with high bit rates.

It is important to use as large IP packets as possible. In every case studied and simulated during this work the performance is improved when using large instead of small packets. Making sure that the largest packet size possible is used, by utilizing the path MTU, is stressed as extremely important.

7.2 Future Work

There are still things to study even though the area has been studied in detail and simulations have been conducted. The limited time as well as the limited form that a thesis represents have caused us to leave out some areas that are both relevant and interesting. Much work in the area of TCP over wireless media is still being done, e.g. [25]. Some of the following points may be worth investigating.

- HSDPA. The introduction of HSDPA into the network contributes with a great deal of new possibilities and issues. The main things to look at here are to see how the shared channel affects TCP and how the scheduling could be implemented.

- Split TCP for small files. The use of Split TCP for small files could improve the performance. The performance are however dependent on the implementation of the TCP proxy and thereby it would be very interesting to look at how the actual implementation of the proxy affects the performance. Caching and window management can affect the performance. Snoop could also be investigated as an alternative.

- System utilization. Considering the view of the system utilization and not only the link utilization would greatly improve the possibility for making the right decisions and recommendations. Issues like flow interaction are considered to be important. For instance many simultaneous TCP sessions, belonging to several users, can be active over the same link.

- Implementation details, i.e. how the solutions should be deployed. The recommendations in this report are based on simulations. If solutions are to be implemented there are great deals of practical issues that need attention. Hardware and design issues are of greatest importance.

- Handovers between different base stations is also important to consider. It is especially important to examine the handover time. This time might cause TCP to timeout or in some other way decrease the performance.

8 References

[1] Allman, Paxson and Stevens. **TCP Congestion Control.** *IETF RFC2581* 1999

[2] Balakrishnan, Seshan, Amir and Katz. **Improving TCP/IP Performance over Wireless Networks.** *ACM MOBICOM*, November 1995

[3] Inamura, Montenegro, Ludwig, Gurtov, Khafizov. **TCP over Second (2.5G) and Third (3G) Generation Wireless Networks** *IETF draft-ietf-pilc-2.5g3g-08*, 2002

[4] Montenegro G., Dawkins S., Kojo M., Magret V. and Vaidya N. **Long Thin Networks**, *IETF RFC2757*, 2000

[5] Ludwig R. and Katz R. **The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions**. *SIGCOMM Computer Communication Review*, 2000

[6] Mascolo, Casetti **TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links**. *ACM SIGMOBILE*, 2001

[7] Bakre A. and Badrinath B. R. **I-TCP: Indirect TCP for Mobile Hosts** *IEEE (ICDCS),* May 1995

[8] Peisa, Meyer. **Analytical Model for TCP File Transfers over UMTS**. *3Gwireless'01* 2001

[9] Sachs J. **Mobile Internet – Performance Issues Beyond the Radio Interface**. *10th Aachen Symposium on Signal Theory*, 2001

[10] Peisa J. and Englund E. **TCP Performance over HS-DSCH.** *IEEE VTC02*, 2002

[11] Dahlén A and Ernström P. **TCP over UMTS.** *RVK02*, 2002

[12] Comer Douglas E. **Internetworking with TCP/IP - Volume 1: Principles, Protocols, and Architectures** *(4th edition), Prentice Hall*, 2000.

[13] CAIDA, **Traffic Workload Overview**, http://www.caida.org/outreach/recources /learn/trafficworkload/tcpudp.xml, September 2002

[14] Holma H., Toskala A. **WCDMA for UMTS** *Wiley* ISBN 0-471-72051-8, 2000

[15] Stevens W.R **TCP/IP Illustrated, Volume 1 (the Protocols)** *Addison Wesley* ISBN 0201633469, 1994

[16] Stroustrup B **The C++ Programming Language** *Addison Wesley* ISBN 0-201-70073-5, 2000

[17] **The NS Manual**, http://www.isi.edu/nsnam/ns/doc/index.html, September 2002

[18] **RLC protocol specification** *3GPP TS 25.322*, *Release 1999*

[19] Gurtov A., Ludwig R. **Responding to Spurious Timeouts in TCP**, Submitted Aug 2002

[20] Mathis M., Mahdavi J.**TCP Selective Acknowledgment Options**, IETF *RFC 2018*, 1996

[21] Braden R **T/TCP – TCP Extensions for Transactions Functional Specification** *RFC 1644*, July 1994

[22] Blanton E., Allman M. **On making RCP More Robust to Packet Reordering**, July 24, 2001

[23] Mogul G., Deering S **Path MTU Discovery**, IETF *RFC 1191*, November 1990.

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

[25] Vangala S., Labrador M. **Performance of TCP over Wireless Networks with the Snoop protocol**, *In the proceedings of the 27th annual IEEE conference on local computer networks (LCN 2002)* November 2002.

[26] Furuskär A. et al. **Performance of WCDMA High Speed Packets Data**, *In Proceedings of IEEE VTC 2002 (Spring)*, May 2002.

[27] Bakre A., Badrinath B. R. **Implementation and Performance Evaluation of Indirect TCP**, *IEEE Transactions on computers, vol46, no3*, March 1997.

[28] Ludwig R., Meyer M. **The Eifel Algorithm for TCP**, Internet draft: draft-ietf-tsvwg-tcp-eifel-alg-03, IETF, Feb 2002.

Appendices

Appendix A – The Network Simulator

An introduction to the network simulator, NS2, and its features is given in the following sections. We will focus on the issues needed to implement the RLC model. Hence, the description of NS2 is not complete.

A.1 Discrete Event Simulation

Simulation is used to investigate how the proposed solutions mitigate the problems associated with TCP over wireless media. Discrete event simulation is suitable to use since the network simulated is IP. There are two main advantages with using simulation for the research and evaluation. First, simulations it is often inexpensive and provides an easy way to examine how different assumptions in the model affect the result, without having to implement the model in a real system. Second, the implementation of the model is often much less time consuming than the implementation of a real system. Furthermore, discrete event simulations can be very fast executing and effective, since only the actual events have to be simulated, not the time in between.

A discrete event simulation can be seen as the system moving from one state to another in discrete steps. Discrete time simulation is suitable since we wish to simulate a packet-based network. One thing common for all simulation is that it mimics the behaviour of the real world. Of course some characteristics may differ between different implementations and simulations.

There are a few simulators for simulating IP networks available. Two of the most well known is Opnet and NS2. They are a bit different in their approaches but still they provide almost the same functionality. However, Opnet is a bit more focused on larger systems while NS2 is excellent for simulation smaller topologies.

A.2 Introduction to NS2

Network simulator version 2, NS2, is a discrete event simulator developed at UC Berkely for simulating local as well as wide area networks [17]. The type of networks simulated is mainly IP but other types, e.g. ATM, can also be simulated with add-on modules. One of the most important strengths of NS2 is that it is widely used, mainly in the academic community. There are a great variety of extensions to NS2 available. The fact that it is free licensed encourages users to share information about bugs and features and the development is in progress continuously.

The simulator is mainly written is C++ [16] and it is implemented according to the object oriented paradigm. All behaviour of links, nodes and other functionality is implemented in C++ but NS2 uses an interpreter as a front-end. The interpreter enables users to set up a topology in a rather smooth manner, instead of having to edit C++ code and recompile for every new simulation. The front-end is written in OTcl, object tool command language, which is an interpreted language with support for objects. The objects in C++ are linked to the OTcl objects in such way that the behavior of the model objects in C++ can be controlled from OTcl objects. This approach of has the benefit of both the powerful and fast executing features of C++ but also the flexibility that a script language like tcl provides. There are of course drawbacks with the shared object approach, e.g.

special procedures are needed in order to make the C++ objects visible to the tcl part of the system.

One major problem with NS2 is that it is rather poorly documented. This leads to problems for the one who want to learn how to use NS in a more advanced way. The only way to fully understand it is to look at the source code itself. Unfortunately, it is really necessary to fully understand the functionality when setting up more complicated topologies.

A.3 Design and Implementation

As mentioned in the previous section NS2 makes use of a shared object implementation. As seen in Figure 21, every object in Otcl has a corresponding object in C++. Since the objects are shared, their attributes can be accessed from both the tcl and the C++ interface.

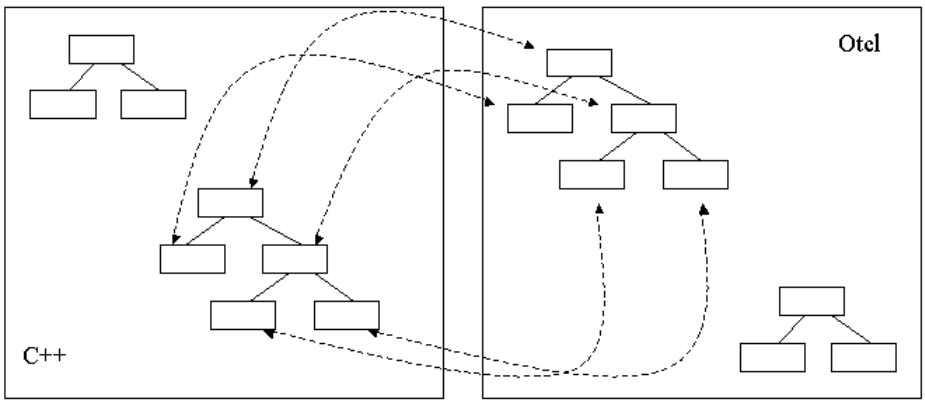

Figure 21 The shared object design used in NS

A.4 Schedulers and Events

One of the most important functionalities in NS2 is the scheduling of events. The scheduler is the core of the simulation and provides functionality for handling events and making sure that they execute the correct code at the right time. The scheduler class has a great variety in its features, but one of the most useful is the one providing a way to schedule your own events.

```
schedule(Handler*, Event*, double delay)
```

There are three main types of schedulers but they differ only in how they are implemented. The types are: calendar, heap and list. There is also a real-time scheduler but it uses the list implementation. The real-time scheduler tries to synchronize simulation time with real time events. This can be very useful when NS2 is used as a component in a larger environment where other parts are not simulated but instead may be real world equipment connected to e.g. the Internet.

The packet class is a sub-class derived from the event class. The classes are therefore in many ways similar and in fact almost all events are packets being sent and received. Each event can be scheduled individually using the method show above. When an event is ready for dispatch, the handler passed to the scheduler is called. The handler executes its code and takes appropriate action before passing the event (packet) on.

The format of a packet can be seen in Figure 22. All the headers that are available in NS2 are in fact present in the packet representation. This is in contrast to real IP packets where only the headers that are used are present. To compensate for this, the size of the packet in not at all dependent on these headers, instead the size of the packet is set in a special field in the cmn part of the header. The data field in the packet are most of the time not used at all. This results in that only headers are passed around in the network without actually carrying any data. Since NS is only a simulator this is reasonable, and everything that is dependent on the size of the packet, uses the size indicated by the number in the cmn field in the header.

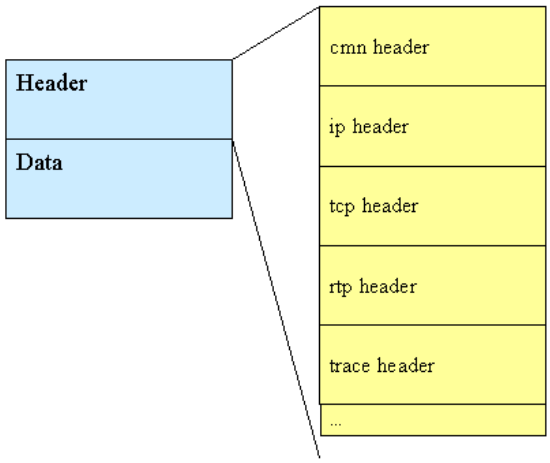

Figure 22 The internal structure of a packet in NS2

A.5 Nodes and Links

The first thing to set up before doing a simulation is the topology. It often comprise of nodes and links. The links interconnect the nodes and one node can have multiple incoming and outgoing links. To determine what incoming packet should be sent out to what outgoing link an internal "routing" is used. Classifiers who have the capacity to determine what to do with an incoming packet implement this routing. The classifier used in unicast is the address classifier and it can use IP addresses to forward the packet to the correct destination.

Each object that ever handles a packet has one thing in common: the way that the packet is received. A packet is always received through the method recv(Packet* P, Handle* H). P is the packet being transferred and H is a reference to a handler. The handler is a pointer to the object that will execute if the receiver chooses to call it. The handler can be, and is in some cases, used to send information in the opposite direction from what the packet is travelling.

Links are also a very fundamental element of the simulation topology and comprises of several components. The most important parts are the queue, linkdelay and the ttl module. While the linkdelay introduces a delay corresponding to the transfer time over the link, the queue simply queues incoming packets before sending them on the link. The ttl module is responsible for calculating the new time to live value for every packet. All packets must pass these modules before being passed on.

It might be strange that the queuing is done in the link instead of in the node (router) but there is only a small difference. All queuing of packets are normally done in the incoming stream of a router, in NS2 this is done in the link instead. The main disadvantage of the strategy used in NS2 is that if a node has several incoming queues the node has no way of doing advanced queue management taking into account the characteristics of all queues. The reason for this is simply that the different queues are located in different links.

A.6 Agents

A simulation without traffic is not worth much so NS of course has a way to insert traffic into he network. The means for this is agents. There are many types of agents but they have one thing in common: they all operate at an end-point of the network, i.e. they work as terminals. Furthermore, they have the responsibility of constructing and destroying packets.

There are several agents supporting various protocols implemented in NS2. One agent is the TCP agent. It sends out TCP traffic and is actually only a sender side implementation of TCP. For receiving TCP traffic and responding in a correct way there is need for an agent handling this. The agent who does this is called the TCP Sink. This division of TCP into one sender and one receiver is really a simplification and later there were added functionality to have both sides act as both sender and receiver. This new agent is called Full TCP. There is also support for other types of traffic. TCP may be the dominant in the number of variants supported but there is also support for UDP as well as RTP.

The TCP and UDP agents do not produce data. Instead a traffic generator needs to be connected to the agents. The least complicated one is the CBR, constant bit rate, but there are many others as well. FTP is implemented as a traffic generator and when connected to TCP it can be configured to send a specific amount of data to another agent in the topology.

A.7 Tracing

There is built-in support for tracing the data flow generated in the simulation in NS2. The tracing is done by writing data, describing various events, to a text file. Since the simulator is used to simulate networks, most things that have to do with packets are written to the file. If the packet is associated with a specific protocol, e.g., TCP, data from the protocol header can also be traced.

Besides tracing packet related events the simulator can also be configured to trace other things. This manually configurable tracing utility is one very useful feature when working with TCP since all TCP's internal variables can be traced. One example of this feature is illustrated when the congestion window of TCP is traced. It shows in a straightforward way how the congestion window grows or shrinks as a reaction to packets arriving. Below, a sample taken from a trace file can be seen.

The trace file may seem confusing at first, but once you understand the meaning of the columns and symbols it is really straightforward. The leftmost column indicates what type of event that occurred; the next column is the time when that event occurred. After this there is two columns containing information about the source and destination addresses for the packet transmitted. The next thing to come, still going left to right, is the type of packet followed by the packet size.

```
r 0.534048 3 0 ack 40 ------- 2 3.0 2.0 2 5
+ 0.534048 0 1 ack 40 ------- 2 3.0 2.0 2 5
- 0.534048 0 1 ack 40 ------- 2 3.0 2.0 2 5
+ 0.541776 2 1 tcp 1500 ----- 2 2.0 3.0 3 6
- 0.541776 2 1 tcp 1500 ----- 2 2.0 3.0 3 6
- 0.544176 2 1 tcp 1500 ----- 2 2.0 3.0 4 7
r 0.554112 0 1 ack 40 ------- 2 3.0 2.0 2 5
+ 0.554112 1 2 ack 40 ------- 2 3.0 2.0 2 5
- 0.554112 1 2 ack 40 ------- 2 3.0 2.0 2 5
```

The most common symbols in the leftmost column are the r, + and -. The r symbol represents the event that a packet is received at a node, while + and – symbols are related to queuing. The + means that a packet was inserted in a queue while – means that a packet was taken from the queue.

Even when one understand the trace file it can be hard to se the big picture. That's where a visualization tool can be very useful. NS2 has its own graphical visualization tool called NAM, network animator.

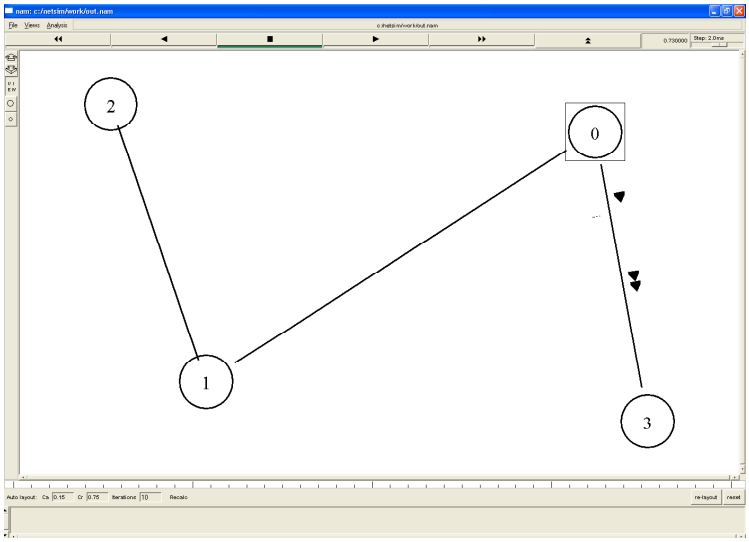

Figure 23 NAM the network Animator

NAM shows not only the topology but also packet flow and can be very useful for following the traffic. In Figure 23, a simple topology can be seen as well as some packets that are in transit. Unfortunately you cannot see any internal state event though you chose to trace it in the log file.

Appendix B – Implementation

In the following sub-sections the design and implementation of the RLC model is described. The model is implemented as a module for NS2. Readers only interested in using and configuring the module should skip to Section A.2.5.

B.1 Introduction

The implementation of the model is done with NS2 as the target simulation platform. Since NS2 does not have support for UMTS radio links, a new module had to be implemented. There is support for wireless simulation in NS2, but the focus is mainly on WLAN and the RLC behaviour of UMTS not supported. However, implementing the complete UMTS RLC support in NS2, including the physical layer, would be extensive work and is really not needed to simulate the behaviour of UMTS RLC. Instead the model is implemented as a modification of a link.

As mentioned in section A.1.5, every link has an internal queue, responsible for holding the packets while they are waiting for their turn to traverse the link. The name of this class is Queue and it is used as base for the implementation of the RLC model. The functionality of the original queue is limited and provides only functionality for inserting packets in one end and extracting them at the other end. Our model replaces the internal functionality of the queue to mimic the behaviour of RLC. Since we are only interested in building a model that from the outside behaves like RLC, this is a reasonable design. Since NS2 is object oriented, our implementation also follows this paradigm.

B.2 The Implementation and its Features

The implementation is a realization of the theoretical model described in section 5.2, and much of the features are the same. The central feature of the model is that packets are segmented into radio blocks, PDUs. The radio blocks are then transmitted and retransmitted according to the RLC algorithms.

The radio blocks are after their creation inserted into a queue for radio blocks. When a radio block is sent, the next one is taken from the head of the queue and sent over the radio link. The actual sending is implemented as delaying the radio block for a while and when the block is ready, a handler takes care of the responsibility for making the correct thing happen.

B.3 Classes

To implement the functionality there are three main classes that providing the necessary functionality. The most important is the EasyLink class that is a subclass of Queue. Since the queue interacts with the link delay, EasyLink also needs to do so in an appropriate way. Below, a list of the classes in the model can be seen.

The model comprises the following classes:

- `PacketHolder`
- `PHList`
- `DelayHandler`
- `ResumeHandler`
- `RetransmitHandler`
- `RadioBlockQueue`
- `EasyLink`

- `SortPac`

B.4 A Packets Route through the Model

To illustrate and hopefully give a better understanding of the how the implementation works, an example is now given. The scenario starts with a packet being received at the link and ends with that the packet is sent on.

The packet enters the system when EasyLink's recv() method is called. In recv the packet is "segmented" into radio blocks. Actually, the packet is not segmented but radio blocks are allocated and a mapping from that IP packet to the allocated radio blocks is constructed. There are a number of things that are considered when the mapping is done. The most obvious one is if the IP packet fits into one radio block or if several blocks are needed. Furthermore, there is a check to find out if there is room in an earlier block so that concatenation can be done.

When the mapping is done, both the IP packet and the mapping to radio blocks are stored in a PacketHolder for later use. The radio blocks are now put in a queue, waiting to be "sent" over the radio link. If no radio block is in transfer, a new block is scheduled for sending. When the block has been in transit for long enough time (the amount of time is of course dependent on the links speed) the DelayHandler is invoked by the scheduler. The code in DelayHandler now calls the EasyLinks method passon(). In passon() a decision is made whether the block has an error or not. This is done with a random variable. If the block is free from errors, the mapping from radio blocks to IP packets is consulted to make a decision if any packets are ready. If, on the other hand, the block is considered to be faulty, a retransmission is made.

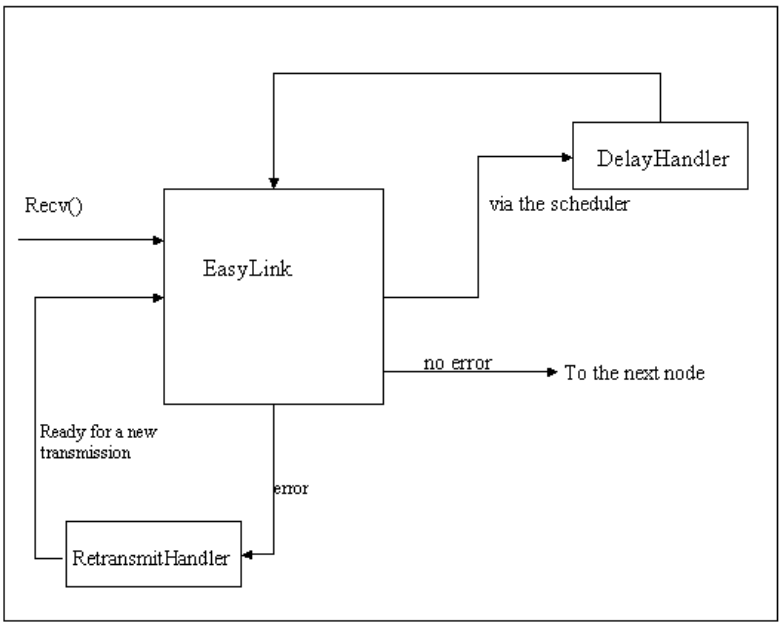

Figure 24 Diagram over the function of the simulation model

When a retransmission issue has been processed long enough (retransmission time) it is inserted in the queue where packets that not yet have been sent are contained. To follow the models indication that the retransmitted blocks should have a higher priority, they are inserted at the front of the queue. This procedure is repeated until every radio block associated with the incoming IP-packet are transferred without errors. The packet is then

sent on the next item, usually a node. Of course several packets can be handled at the same time causing a radio block from one packet to wait for another radio block from another packet. This type of interaction in normal and are considered being a part of the model since the real RLC will have the same behaviour.

B.5 Usage

In order to be able to use the implementation of the model for simulations a few prerequirements must first be fulfilled. First of all a NS2 package must be installed and configured. Furthermore, the link model must be inserted into the NS2 environment and compiled. When all this is done, it is time to set up the simulation topology and configure the parameters of the model.

The topology set-up is not described here since it is well documented in the NS manual [17]. However, we will describe the configuration of the parameters in the model.

To start with we need to set up a link and tell it that our EasyLink should be used as queue. This is done by using the built in simplex-link in NS and specifying that our model should be used as a queue in that link.

```
$ns simplex-link $n1 $n0 3Mb 3ms EasyLink
```

This code results in a connection between node n1 and node n0, where EasyLink is used in the direction from n1 to n0. Besides the possibility to set up the link, EasyLink also has a few parameters that can be set. These parameters are central in the model and thereby affect the simulation. Below a short initialization code can be seen. The parameters set corresponds to the parameters in the model.

```
Queue/EasyLink set tti_ 0.02
Queue/EasyLink set retransdelay_ 0.059
Queue/EasyLink set speed_ 128000
Queue/EasyLink set errorrate_ 10
```

When the link is configured it is ready to be used. Please note that omitting the initialization of the parameters will result in default values that may not correspond to the desired configuration.

List of Abbreviations

3G	Third Generation (Mobile Networks)
3GPP	Third-Generation Partnership Project
ACK	Acknowledgment
ADSL	Asynchronous Data Subscriber Line
BDP	Bandwidth Delay Product
BPS	Bits Per Second
BER	Bit Error Rate
BLER	Block Error Rate
BMC	Broadcast/Multicast Control
BPS	Bits Per Second
CN	Core Network
DUPACKS	Duplicate Acknowledements
GPRS	General Packet Radio Service
GSM	Global System for Mobile telecommunications
GMSC	Gateway MSC
GGSN	Gateway GPRS Support Node
HLR	Home Location Register
HSDPA	High Speed Downlink Packet Access
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
MAC	Medium Access Layer
MSS	Maximum Segment Size
MSC	Mobile Service Switching Centre
MTU	Maximum Transmission Unit
NS2	Network Simulator (version 2)
OPNET	Optimum Network Performance
PDU	Protocol Data Unit
PDCP	Packet Data Convergence Protocol
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RB	Radio Block
RLC	Radio Link Control
RNC	Radio Network Controller

RTT	Round Trip Time
RTO	Retransmission Time-Out
SACK	Selective Acknowledgments
SDU	Service Data Unit
SGSN	Serving GPRS Support Node
SMSS	Sender Maximum Segment Size
SYN	Synchronizing Segment
TCP	Transport Control Protocol
TTI	Transmission Time Interval
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
UTRAN	UMTS Radio Access Network
VLR	Visitor Location Register
WCDMA	Wideband Code Division Multiple Access
WLAN	Wireless Local Area Network