Performance improvement of a Body Area Network interconnect protocol

Master's Thesis

Performed at the Application Protocol Systems (APS) group
Department of Computer Science, University of Twente, The Netherlands

By: Daniël F. Knoppel

Supervisors:

Prof.dr.ir. D. Konstantas
Dr.ir. A.T. van Halteren
Dr.ir. I. Widya

Timeframe:

November 2002 - 2003

Samenvatting

(for English version of the abstract, see next page)

Het MobiHealth project is een Europees project dat werkt vanuit de visie dat de bewegingsvrijheid van patiënten wiens gezondheidstoestand toezicht behoeft zo min mogelijk beperkt moet worden. Het systeem dat ontwikkeld is ter bevordering van deze doelstelling maakt gebruik van draagbare lichtgewicht meetapparatuur (Body Area Networks) in combinatie met nieuwe generatie draadloze technologieën. Het zogenaamde "Body Area Network interconnect protocol" is verantwoordelijk voor het overbrengen van meetgegevens via een draadloze verbinding vanaf een Body Area Network naar een locatie die toegankelijk is vanuit ziekenhuizen.

Deze scriptie bevestigt het vermoeden dat het Body Area Network interconnect protocol gebrekkig presteert met een (draadloze) GPRS verbinding. Op zijn best benut het protocol minder dan eenderde van de capaciteit van de verbinding en levert meetgegevens af met ongeveer 5 tot 11 keer meer vertraging dan de vertraging die de GPRS verbinding gemiddeld veroorzaakt. In combinatie met de toch al lage capaciteit van de GPRS verbinding heeft dit ernstige gevolgen voor de real-time meetcapaciteiten van het Body Area Network.

Middels een proces van analyse waarbij er onder andere prestatiemetingen zijn verricht in een realistische (niet-gesimuleerde) netwerkomgeving zijn er een aantal belangrijke factoren vastgesteld waar het merendeel van de gebrekkige prestaties aan te wijden is. Vervolgens zijn er een stel verbeteringen ontworpen en geïmplementeerd in het daadwerkelijke interconnect protocol.

Het resultaat is dat het verbeterde Body Area Network interconnect protocol in staat is om ongeveer vier keer meer data per seconde over de draadloze verbinding te transporteren met ruwweg 30% minder vertraging dan zijn voorganger. Het nuttig gebruik van de draadloze verbinding is gestegen naar 66% en wordt verwacht te stijgen naar 97% met een andere GPRS aanbieder en GPRS modem. Een bijeffect is dat de nieuw toegevoegde data compressie resulteert in een belangrijke kostenbesparing van meer dan 40%.

Verwacht mag worden dat de gerealiseerde verbetering van het Body Area Network interconnect protocol resulteert in een sterk vergroot bereik van patiënten die mogelijk baat hebben bij het MobiHealth systeem.

Abstract

The MobiHealth project is a European project that envisages the delivery of healthcare services to patients who require health monitoring without restricting their freedom to move around. The system developed for this purpose employs wearable lightweight monitoring equipment (Body Area Networks) in combination with next generation wireless technologies. The so-called "Body Area Network interconnect protocol" is responsible for transferring vital sign measurements from a Body Area Network over a wireless connection to a location that is accessible from healthcare centers.

This thesis confirms the suspected poor performance of the Body Area Network interconnect protocol on a (wireless) GPRS link. It effectively uses less than one third of the link capacity at best and delivers vital sign measurements with around 5 to 11 times more delay than the average GPRS link latency introduces. Combined with the already low capacity of the GPRS link, this has grave consequences for the real-time monitoring capabilities of the Body Area Network.

Through a process of analysis that includes performance measurements in a realistic (non-simulated) network setup, a number of major performance drawbacks were identified that are responsible for most of the observed performance loss. Subsequently, a set of improvements were designed and implemented in the actual interconnect protocol.

The result is that the improved Body Area Network interconnect protocol is able to transfer about four times more data per second over the wireless link, with roughly 30% less delay than its predecessor. Its utilization of the wireless link has risen to 66%, which is expected to grow to 97% with a different GPRS provider and GPRS modem. As a side effect, the newly introduced data compression results in significant cost savings of more than 40%.

It is expected that the implemented improvements to the Body Area Network interconnect protocol result in a much larger range of patients that can potentially benefit from the MobiHealth system.

Acknowledgements

I would like to thank a number of people for helping me out in various ways with this masters’ thesis. Grateful acknowledgements go out to my immediate supervisors Aart van Halteren and Ing Widya for their myriad of helpful comments and insights. For support with the MobiHealth system implementation, I thank George Koprinkov and Nickolay Dokovsky. Fellow researcher Katarzyna deserves a thank-you for the interesting and valuable discussions we had. And last but not least, I would like to thank my family and friends, in particular Annemarie, for continued support and ensuring my general well being.

Thanks, all ☺

Table of Contents

Abbreviations

2G	Second generation wireless technology (GSM)
2.5G	Wireless technology that represents an intermediate step between 2G and 3G (GPRS)
3G	Third generation wireless technology (UMTS)
BAN	Body Area Network
BANip	Body Area Network interconnect protocol
BDU	BAN data unit
BDL	BAN Data Latency (of BAN data units transmitted the BANip)
BESys	BackEnd System
Bps	Bytes per second
CPI	Critical Performance Indicator
GGSN	Gateway GPRS Support Node
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
KB	Kilobyte (1024 bytes = 1 KB)
kbps	kilobits per second (1 kbps = 1000 bits/sec = 125 Bps)
LAN	Local Area Network
MBU	Mobile Base Unit
MS	Mobile Station
MTU	Maximum Transmission Unit
NTP	Network Time Protocol
PKT	Packetizer Setting (BANip)
QoS	Quality of Service
RFC	Request For Comments (IETF issued document)
SGSN	Serving GPRS Support Node
SH	Surrogate Host
TBF	Temporary Block Flow
TC	Transfer Capacity (of the BANip)
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
URL	Universal Resource Locator
WSB	Wireless Service Broker

1. Introduction

With the advancement of wireless technology, new applications become possible in the healthcare sector. Removing the restrictions imposed by wires and cables enables patients to benefit from increased mobility. For example, ambulant patients would normally have to remain at a healthcare center if they require regular health monitoring, even though they are not confined to a hospital bed. With a wireless monitoring device it becomes possible for them to return to the comfort of their own home. Such an application not only improves the patient's quality of life, but also benefits healthcare insurers when it comes to disease and care related costs. A healthcare center saves on its expenses, e.g. food and housing, by enabling patients to spend more of their time away from the center.

These, and other mobile health applications are the topic of investigation for the MobiHealth project. This project envisages healthcare services for patients without restricting their freedom to move around. As a part of the project, the so-called MobiHealth *Body Area Network* (BAN) has been developed to enable remote monitoring of ambulant patients. The BAN is worn by a patient and basically consists of a set of lightweight devices that monitor and wirelessly transmit certain biosignals (vital signs) to a BackEnd System (BESys). Healthcare centers can then retrieve this data over a reliable wired connection. Figure 1.1 shows the basic concept of the MobiHealth System with an example set of devices that can be used for the implementation.

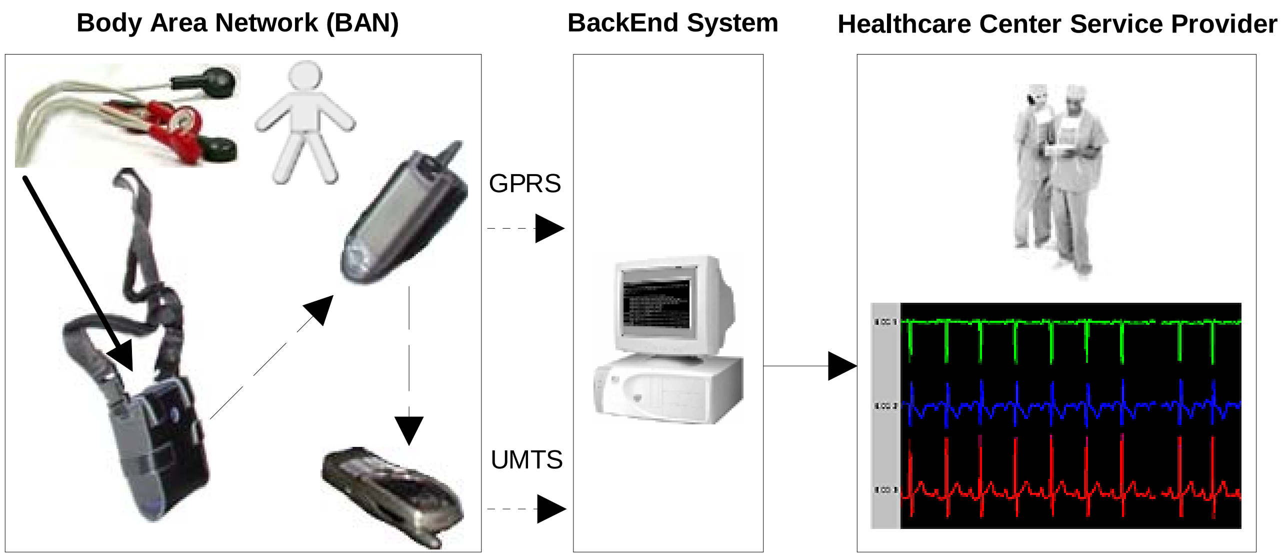


Figure 1.1: MobiHealth System example configuration

Unfortunately, the reference design and implementation of the application protocol that interconnects the BAN with the BackEnd System shows significant performance problems. In the configuration used, the protocol operates more than four times slower than may be expected from the capabilities of the applied wireless technology, resulting in major transfer delays.

This thesis represents the work done as part of a graduation project on improving the performance of the interconnecting application protocol, conducted at the University of Twente in The Netherlands. The main results presented are the following:

- An analysis of the causes for the performance problems in the reference implementation
- The design of an improved interconnect protocol
- An implementation of the design mentioned above
- The performance improvement that the new design offers, relative to the reference interconnect protocol

The remainder of this introductory chapter is organized as follows. Section 1.1 describes the MobiHealth project in more detail, and elaborates on the role and components of a MobiHealth BAN. The application protocol that interconnects the BAN with the BackEnd System is called the Body Area Network interconnect protocol (BANip), and is further introduced in Section 1.2. This thesis focuses on resolving BANip performance problems, which are outlined in Section 1.3. The objectives are formulated through a number of research questions, listed in Section 1.4. Section 1.5 describes the general approach taken to answer the research questions.

The remaining chapters are organized as follows:

- Chapter 2 describes low-level technologies such as the General Packet Radio Service (GPRS) and Internet protocols that are required for the operation of the BANip. The BANip itself and the HyperText Transfer Protocol (HTTP) it uses are also described.
- Chapter 3 provides a complete high-level performance assessment of the reference BAN interconnect protocol. This includes a definition of critical performance indicators (CPIs) and a set of performance measurements.
- An in-depth protocol performance analysis is presented in Chapter 4, complemented with various performance measurements.
- Chapter 5 presents the design and implementation of an improved BAN interconnect protocol, based on the results of the performance assessment and analysis. The performance of the improved BAN interconnect protocol is also measured and compared to the performance measurements of the reference BAN interconnect protocol.
- Chapter 6 presents the main conclusions of this thesis.

1.1 The MobiHealth Project

This section provides some background information on the MobiHealth project in the first subsection. The core components of the health monitoring system developed as part of the MobiHealth project are explained in the two subsections following that. The last subsection describes the typical operation of the MobiHealth system.

1.1.1 What is MobiHealth?

MobiHealth is a mobile healthcare project. An important part of the project is to develop an infrastructure that allows professionals at healthcare centers to monitor, on a near real-time basis, vital signs from patients who are not physically there (but at home or on the move, for instance).

The MobiHealth[1] consortium unites 14 partners from five European countries. Funded by the European Commission, the MobiHealth project has been investigating the technical feasibility and potential benefits of mobile and wireless telematics systems for the support of mobile health applications since its inception in May 2002. MobiHealth aims at developing new mobile value-added services in the area of healthcare, which will be tested in several different field trials. The MobiHealth consortium consists of partners such as hospitals and medical service providers, universities, mobile network operators, mobile application service providers and mobile infrastructure and hardware suppliers.

1.1.2 Body Area Network

A key aspect of the MobiHealth system is that it enables near real-time information exchange with patients who are not physically present at a healthcare center. More specifically, patients can wear a lightweight wireless Body Area Network (BAN) that mainly measures and transmits biosignals to a remote healthcare center. The BAN is a core component of the MobiHealth system, and is organized as follows: (see Figure 1.2 below)

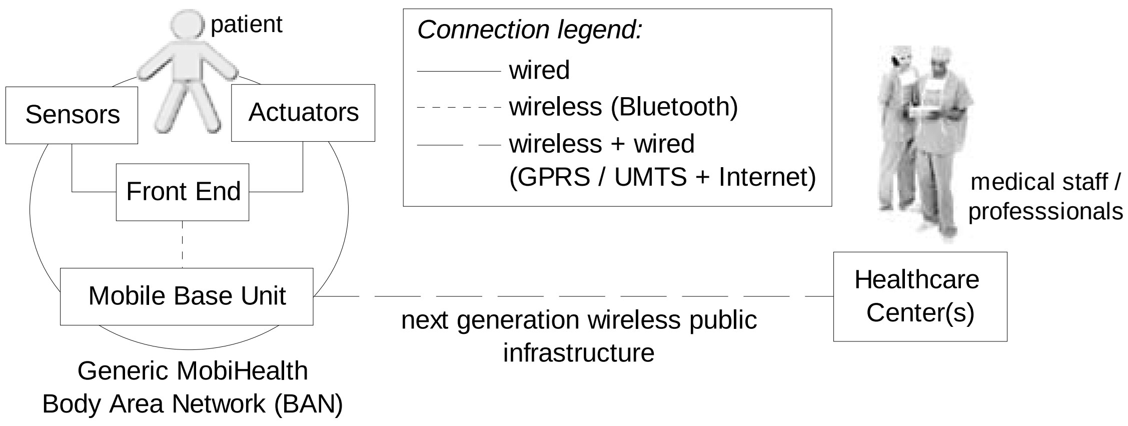

patient
Sensors
Actuators
Front End
Mobile Base Unit
Generic MobiHealth Body Area Network (BAN)
Connection legend:
wired
wireless (Bluetooth)
wireless + wired (GPRS / UMTS + Internet)
medical staff / professsionals
Healthcare Center(s)
next generation wireless public infrastructure

Figure 1.2: Body Area Network in MobiHealth: topology

[1] See also the MobiHealth website: www.mobihealth.org

The MobiHealth BAN consists of the following three major components:

1. **An array of sensors and / or actuators**
   This component is highly customized to suit the requirements for monitoring patients whom it will serve. For example, a diabetic patient BAN might contain sensors to measure blood glucose levels and actuators that can inject insulin. Other examples of sensors that could be part of a specific BAN include [Mwp302]: a temperature sensor, pulse & oxygen saturation sensor, drop (impact) sensor and more complex sensors such as electrocardiogram (ECG) and electroencephalograph (EEG) sensors.

2. **A Front End**
   Sensors are plugged into ports on a device called the Front End. Through each port, the Front End supplies the attached sensor with power. Analog signals received through the port are also converted to digital values and mapped to a channel. The channels are multiplexed and transmitted via a Bluetooth[2] wireless connection. Potentially, the Front End will also be able to control actuators, but this option is not yet implemented in the reference system.

3. **A Mobile Base Unit (MBU)**
   The MBU has several functions within the BAN. As the control unit of the BAN, the MBU starts biosignal measurements through the Front End, via Bluetooth. The MBU also receives digitized biosignals from the Front End and processes these for further transportation. Finally, the MBU initiates (long distance) wireless data transfers to transmit the collected data to healthcare center(s). In a sense the MBU is the gateway of the BAN to the Internet (and ultimately the healthcare center). Examples of MBUs include a cellphone supporting Java or a Personal Digital Assistant (PDA) with a General Packet Radio System (GPRS) extension.

A distributed interaction model [Viss99] of the BAN is shown in Figure 1.3.

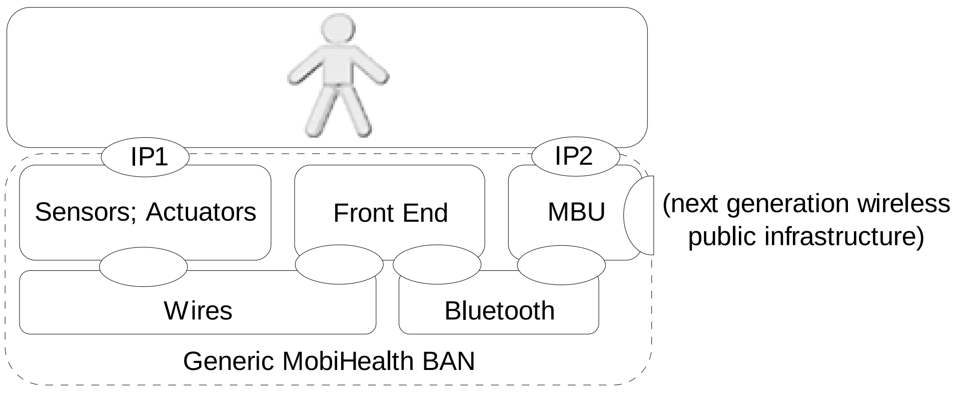


Figure 1.3: Body Area Network in MobiHealth: entities and interaction points

A patient has two interaction points with the BAN (IP1 and IP2). The primary function of the BAN includes monitoring the patient through the available sensors and potentially affecting the patient through actuators. These interactions are modeled through IP1. There is also a

[2] A specification for short range (meters) wireless communication. See also the Bluetooth Wireless Info Site: www.bluetooth.com

different type of interactions possible with the BAN. The MBU has a graphical user interface (GUI) that allows its users to exert control over the BAN. The interaction through the GUI is represented by IP2. Besides or instead of the patient, medical staff may also be assigned to perform the role of controller interacting with the MBU GUI.

There are many possible applications for the MobiHealth BAN. With a MobiHealth BAN strapped on, ambulant patients who require monitoring can be discharged earlier from a healthcare center. The patient has much more freedom to pursue daily life activities or be around the comfort of their own home, factors that are known to have a positive effect on the healing process. The increased freedom is especially beneficial for patients who need to be monitored for long periods of time, rather than for “just” a few hours or days. This works both ways. The healthcare center saves on its expenses by not having to hospitalize the patient.

Another example of an application of the MobiHealth BAN is in the area of the emergency services. Instead of strapping a customized BAN to a predetermined patient, the system can be installed in, or carried with, a vehicle of the emergency services. When (unfortunately not “if”) a serious accident occurs, ambulance personnel can use the BAN to measure and relay vital information ahead of the emergency vehicle to a hospital. At the hospital, the information can immediately be assessed by a trauma team, which can thus start necessary preparations (such as preparing an operating room and, if appropriate, calling a neuro-surgeon) before the patient has even arrived. In short, the MobiHealth BAN may be used to decrease the time that elapses before a severely injured patient is treated, thereby improving the patient’s chance of recovery.

Other example applications are the trials that can be found on the MobiHealth website (www.mobihealth.org).

1.1.3 The BackEnd System

Besides Body Area Networks, the MobiHealth system requires a number of other components to realize its primary concept: to make biosignals measured from ambulant patients available to healthcare centers on a near real-time basis. In Figure 1.4, the BackEnd System is introduced. The dashed arrow highlights the high level path of the information flow between patient and medical professional(s).

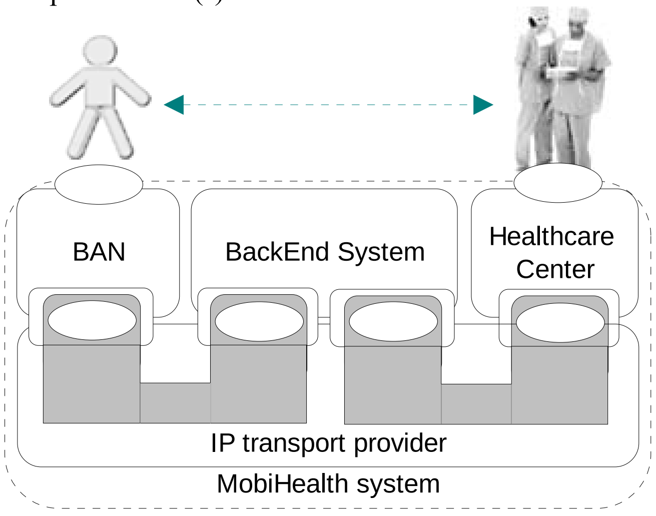


Figure 1.4: MobiHealth system overview: intermediate BackEnd System

Figure 1.4 shows that the BackEnd System is an intermediate system between Body Area Networks and healthcare centers. The two gray areas denote remote interaction functions. The essence is to convey that components of the BAN and healthcare centers do not interact directly, only implicitly through the BackEnd System.

Two important reasons why the BackEnd System is necessary are:

1. Resource requirements for the BAN components should be kept to a minimum because they are scarce. For example, battery and processing power are limited, as is the available bandwidth in the wireless environment. Tasks should be delegated away from the BAN as much as possible. Performing these tasks on the BackEnd System rather than in healthcare centers allows the latter to focus on their primary task of receiving and interpreting biosignals measured by BANs. Moreover, the delegated tasks do not have to be duplicated for every participating healthcare center but can be implemented just once in the BackEnd System, thereby reducing manageability and scalability issues.

2. Due to the nature of the reference BAN technology, the BAN control unit (the MBU) is required to push its data towards receivers, rather than being able to wait for those interested to collect (pull) the data. By acting as an intermediate storage buffer, the BackEnd System prevents data from the BAN from being pushed directly into a secured hospital environment at will. Another advantage of this intermediate storage, in line with the task delegation referred to by the previous paragraph, is that data can be retransmitted to other healthcare centers without further burdening the BAN.

In the future, the BackEnd System can also be expanded to host additional services. For example, an alarm component could monitor the status of certain vital signs and send an urgent notification to the pager of a healthcare professional in case a certain threshold is reached. Such services simplify the requirements on the hospital infrastructure for monitoring ambulant patients.

1.1.4 MobiHealth System Operation

Figure 1.5 shows how the BAN and BackEnd System fit in the network topology of the MobiHealth system. Once again, the dashed arrows highlight the high level path of the information flow between patient and medical professional(s).

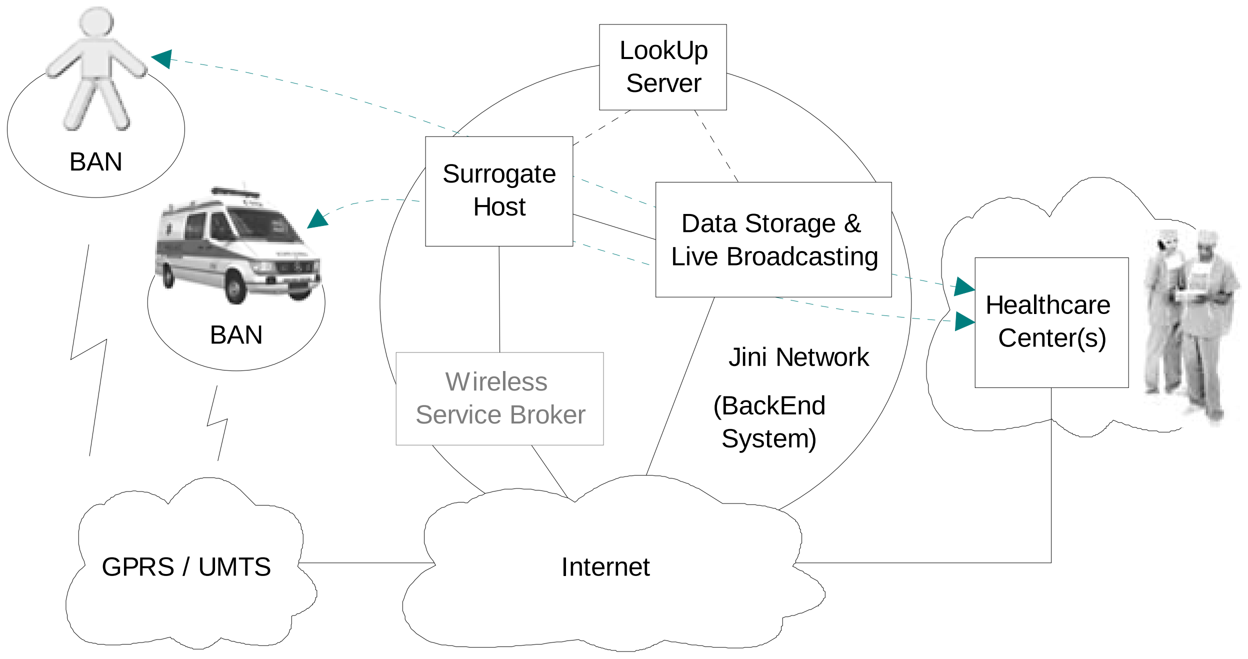


Figure 1.5: MobiHealth system overview: network topology

Data from the BANs proceeds to the receivers (healthcare centers in the figure) through components in the network topology as follows:

- The MBUs within the various BANs each have wireless access to the Internet through the General Packet Radio System (GPRS). In the near future, there will also be support for the Universal Mobile Tele-communications System (UMTS). UMTS is a wireless transmission system that can reach higher speeds than GPRS. MBUs are thus able to transmit sensor data through GPRS, over the Internet, towards the BackEnd System.

- Every MBU is connected via a Wireless Service Broker (WSB) to a Surrogate Host (SH). The WSB it is set up as a proxy that is transparent to the MBU and SH. Its responsibilities include taking care of MBU authentication, so that only authorized MBUs can connect to the SH on the BackEnd. Even when connected, all traffic between the MBU and SH must proceed through the WSB.

- SHs spawn representatives of MBUs within the BackEnd System. Because an SH does not need to be mobile, it can be a more powerful and well-connected system. While an SH may represent more than one MBU, it does not necessarily represent *all* the MBUs. So it is possible to have multiple SHs distributed in various geographical locations. Because the MBU representatives are the source of sensor data within the BackEnd System, other components must have a way to locate ("discover") them. To facilitate this, all participating components are part of a so-called Jini[3] Network.

[3] See also the Jini community website: www.jini.org

- Jini calls for all the MBU representatives to register themselves as "resource providers" at a LookUp Server. Now, even when the components that wish to subscribe to a certain resource are themselves geographically distributed, as long as they are aware of the LookUp Server, they can still locate resource providers.

- In the reference implementation, two types of components within the BackEnd System can discover and subscribe to sensor data flows. These are: a data storage component and a live broadcasting component. A healthcare center could, for instance, consult the data storage component at the end of the day and get a detailed overview of how its patient's situation progressed over the course of that day. In addition, it could be subscribed to a live broadcasting component that relays just enough information for the healthcare center to determine if an emergency situation is imminent.

This concludes the description of how measured biosignals progress from a BAN all the way to a healthcare center.

The following section describes the part of the MobiHealth system with which this research thesis is concerned in more detail.

1.2 Focus within MobiHealth

This thesis focuses on performance problems of the *Body Area Network interconnect protocol* (BANip). The BANip is basically the protocol that connects the MBU present on a BAN to a Surrogate Host on the BackEnd System. To describe in more detail what the BANip is, the definition of *interconnect protocol* is first examined.

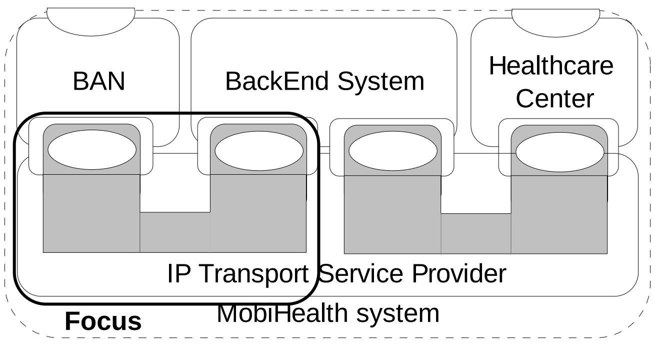


Figure 1.6: MobiHealth system part examined in this thesis

The term *interconnect protocol,* used in the MobiHealth project, originates from the world of Jini. It is defined by the Jini Surrogate Architecture Specification [JiniSA]. This specification addresses the problem of attaching devices to a Jini network that are unable to meet the demands for being part of a Jini network. Because the definition of interconnect protocol relies on basic Jini and Jini Surrogate Architecture concepts, these concepts are described first before the definition is discussed further.

- **Jini** is a distributed system that extends the Java application environment from a single virtual machine to a network of machines. It is based on the idea of federating groups of users and the resources required by those users [JiniArch]. Within MobiHealth, sensor data from the BAN can be seen as a resource. The components within the BackEnd System that collect these resources and set them up for retrieval by healthcare centers can be seen as (resource) users. Consequently, Jini provides a way to organize and manage how BackEnd System components discover BAN features and acquire data from the BANs.

- The **Jini Surrogate Architecture** was devised to resolve the problem of allowing devices to participate in a Jini network without imposing the strict demands on hardware and software associated with Jini network participation. This problem exists in the MobiHealth system, because the applied Jini approach creates unacceptable resource requirements for BAN components if they were to be incorporated directly into the Jini network. The Surrogate Architecture allows a more powerful system to represent devices that do not satisfy the usual requirements for participating in a Jini network. Such a system is called a *surrogate host*. To apply the Surrogate Architecture solution, one of the components of the BackEnd System acts as a surrogate host for BANs (essentially MBUs).

- MBUs implicitly become part of the Jini network by establishing a so called **interconnect** with a surrogate host on the BackEnd System. An interconnect is defined as *the logical and physical connection between a device and its surrogate host* (it is possible to have more than one interconnect over the same physical connection) [JiniSA]. Once the interconnect is established, the surrogate host downloads a *surrogate object* (a collection of Java code) that will serve as the representative of the MBU. Figure 1.7 shows an interconnected MBU.

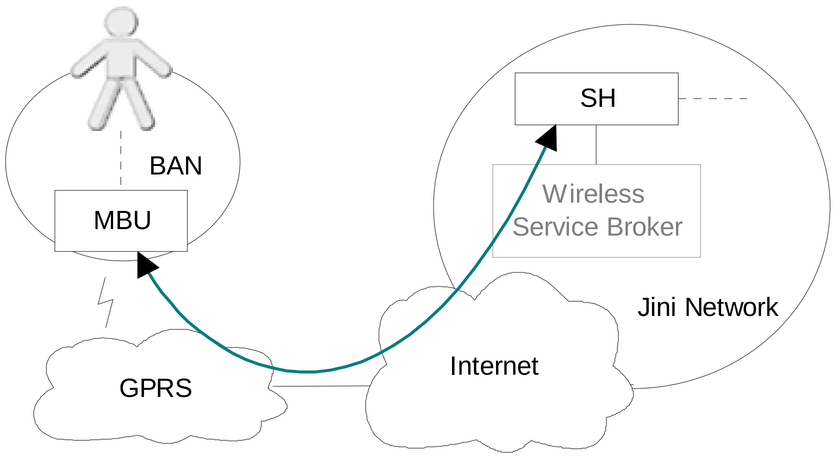


Figure 1.7: Interconnect between MBU and its Surrogate Host

An **interconnect protocol** can now be defined as a protocol that includes interconnect-specific mechanisms for [JiniSA]:
1. discovery[4] (MBU and Surrogate Host finding each other),
2. retrieval of the surrogate (downloading), and
3. liveness (alive is still operating).

This definition partly introduces the functionality of the **BANip**. It is responsible for providing the three mechanisms mentioned above for MBU - SH interconnects. Another important task of the BANip is to provide a data transfer service over the interconnect so that data from the front end (sensor array) can travel to the surrogate object (running on the Surrogate Host), which ultimately represents the BAN in the Jini network. Figure 1.8 shows a decomposition of the service the BANip provides over the MBU - SH interconnect. The area of focus marked with a rounded square in Figure 1.6 is marked again in Figure 1.8.

[4] Discovery in this context is unrelated to the Jini discovery and join protocols, which are used for locating and subscribing to resources that are already part of the Jini network

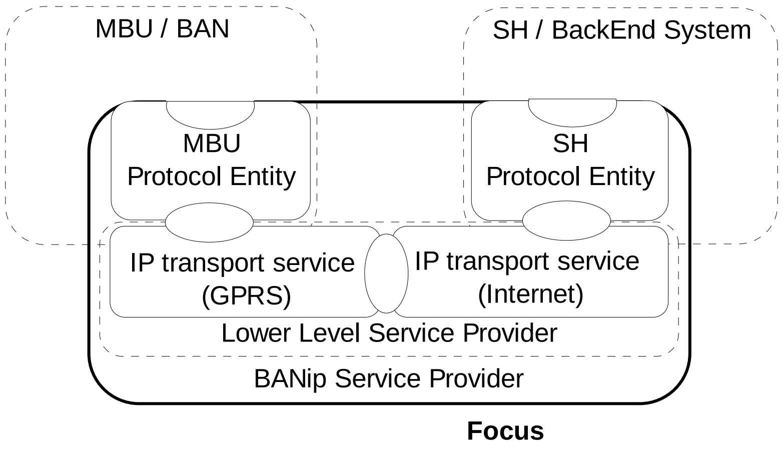


Figure 1.8: BANip service decomposition (MBU - SH interconnect)

N.B.: The Wireless Service Broker is not modeled separately in the figure above even though it lies within the area marked as the focus of this thesis. Because it is a proxy host that is transparent to both the MBU and Surrogate Host, it was left out of this introductory diagram in order to avoid unnecessary complexity. Since it can conceivably be a source of performance problems, it will be described and examined in the following chapters along with the rest of the system in the focused part.

Figure 1.9 shows the protocol stack associated with the reference BANip service. To the left, International Organization for Standardization (ISO) Open Systems Interconnection (OSI) reference model layers are shown.

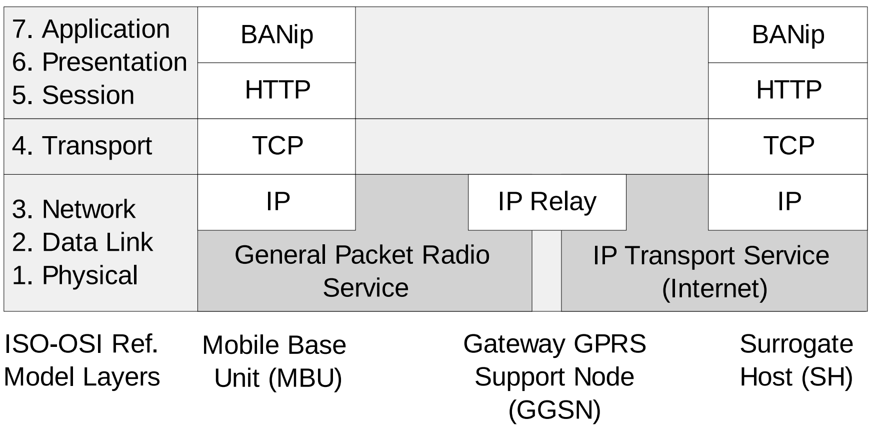


Figure 1.9: BANip service provider: protocol stack

The MBU and SH protocol entities from Figure 1.8 correspond with the top three protocols drawn in the protocol stack. The operation of these protocols can be summarized as follows:

- The BANip protocol on the MBU and Surrogate Host communicate by exchanging their protocol data through the HyperText Transmission Protocol (HTTP) [RFC1945], [RFC2616].

- HTTP utilizes the Transmission Control Protocol (TCP) [RFC793] to exchange its requests, which provides reliable data transfers between the two hosts.

- The Surrogate Host has an Internet Protocol (IP) [RFC791] interface available to it at the Network layer. The MBU also has access to an IP interface because it is specifically supported by GPRS operator's network services. The TCP in both the MBU and SH delivers its data to these interfaces for transmission. On the MBU, IP datagrams are transmitted over a GPRS link into the GPRS operator network. This network uses a Gateway GPRS Support Node (GGSN) to relay IP datagrams to the Internet. From the GGSN, MBU data is routed through the Internet towards its destination, the Surrogate Host. When utilizing TCP, the Surrogate Host may also send replies back to the MBU.

A more detailed description of the total stack and an explanation of the mechanisms at work in the interconnect between MBUs and surrogate hosts can be found in Chapter 2.

The next section highlights performance issues with the reference interconnect protocol.

1.3 Performance Issues

The MobiHealth system relies on 2.5G (GPRS) and 3G (UMTS) wireless technology to significantly enhance the mobility of ambulant patients who require health monitoring. The transfer rates supported by GPRS are relatively low compared to the transfer rates that can be achieved over modest wired links. Because each biosignal measured in a BAN results in a certain volume of data that should be displayed in near real-time at healthcare centers, it is clearly very important to have a system that can make efficient use of the limited transfer rate.

The reference MobiHealth BANip is responsible for transferring measured biosignals over the wireless link from a BAN to the BackEnd System, from where the data can be distributed further without the limitations imposed by the wireless link to the patient BAN. The reference BANip appears to suffer from unexplained delays, possibly originating from reduced transfer speeds or overhead.

One of the major efforts initially performed in this thesis was to define the BANip performance in a measurable way and execute measurement sessions in order to verify the existence of significant performance problems (see Chapter 3). The results confirm the suspicions of degraded performance:

- Compared to the transfer rate of 26.8 kbps expected from the GPRS connection (dual slot CS-2), the BANip only achieved 5.8 kbps on average. That is more than four times slower.

- At best, the BANip achieved 8.5 kbps, but this causes biosignal data to arrive at the BackEnd System with an average delay of 11.5 seconds, while the average transit delay of individual data packets on the GPRS connection is only expected to be around 0.6-2 seconds. So, the delay is more than five times greater than expected.

This thesis identifies and resolves performance drawbacks that are connected to this performance degradation. The exact objectives are formulated through a number of research questions. These are described in the next section.

1.4 Research Questions

This thesis deals with a number of research questions. The objective of the thesis leads to the following main research question:

- How can the performance of a Body Area Network (BAN) interconnect protocol (spanning a wireless and wired wide area network) be improved?

Derived from this main research question are three research questions and several sub questions.

Before performance can be improved, it must first be clear how performance is assessed. Therefore, the first derived research question is:

1. Which critical performance indicators (CPIs) are used to assess the BAN interconnect protocol?

The answer to this question provides a theoretical basis for the performance assessment of the BAN interconnect protocol. Because any performance improvements to the reference (existing) BAN interconnect protocol that are presented in this thesis must also be quantified, the second research question is:

2. How can these CPIs be measured most accurately in practice, i.e. not through simulations?

    - Which test setup should be used to closely represent actual network conditions in which the BAN interconnect protocol will operate?

    - Which test set should be used to closely represent actual usage of the BAN interconnect protocol?

The performance of the BAN interconnect protocol is influenced by many different factors. The effect of these factors can be determined by measuring instance values of the CPIs. The reference BAN interconnect protocol must be examined for possible improvements. The third derived research question covers this requirement in relationship to the CPIs:

3. How can the CPIs be positively influenced, thus improving performance?

    - Which significant negative influences on the CPIs (i.e. performance drawbacks) can be identified in the reference BAN interconnect protocol stack and its elements?

    - Which changes to the reference BAN interconnect protocol stack and its elements have a positive influence on the CPIs?

The approach taken to answer the research questions is presented in the last section of this chapter, Section 1.5.

1.5 Approach

The general approach to answering the research questions, ultimately resulting in the implementation of a modified interconnect protocol that contains measurable improvements, is reflected in the structure of the chapters. The whole process can be divided into a set of three main phases with specific objectives:

1. **Background research and clarification of the focus**
   → Acquire extended knowledge of the research area
   → Limit the focus and set up goals

2. **Analysis of the reference BANip performance**
   → Define the notion of performance
   → Measure the performance of the reference BANip (black box)
   → Perform a theoretical and practical layered (white box) analysis to identify performance drawbacks

3. **Proposing, implementing and verifying improvements**
   → Determine changes in the BANip that should theoretically resolve performance drawbacks or generally improve performance
   → Implement changes and measure the actual relative performance improvement
   → Discuss remaining unresolved performance drawbacks or possible improvements

This chapter and Chapter 2 (Technology Overview) contain the results of the preparatory phase 1.

The first objective of phase 2 is essentially equal to answering the first research question: “Which critical performance indicators (CPIs) are used to assess the BAN interconnect protocol?”.

For the second phase 2 objective, it is necessary that the second research question is answered: “How can these CPIs be measured most accurately in practice, i.e. not through simulations?”. Both answers can be found in Chapter 3 (Interconnect Protocol Performance Assessment).

The third phase 2 objective reflects the philosophy that it is necessary to analyze and test individual elements that affect the BANip performance in order to properly identify significant performance drawbacks. It maps directly to the first part of the third research question: “Which significant negative influences on the CPIs (i.e. performance drawbacks) can be identified in the reference BAN interconnect protocol stack and its elements?”. Chapter 4 (Interconnect Protocol Element Analysis) is dedicated to this objective and answers the corresponding first part of the third research question.

The last research question: “How can the CPIs be positively influenced, thus improving performance?”, which is partially answered in phase 2, receives a finalized answer in phase 3. Chapter 5 (Improved Interconnect Protocol) covers this phase.

The most important findings are reiterated in Chapter 6 (Conclusions).

2. Technology Overview

This chapter provides an overview of important technologies used in the part of the MobiHealth system that this thesis focuses on. These are also shown in the BANip service provider protocol stack of Figure 1.9.

Section 2.1 is dedicated to the introduction of the General Packet Radio Service (GPRS), used to support long distance wireless transfers for the BANip. The Internet Protocol (IP) provides the actual (unreliable) transport service for data from the MBU through the GPRS operator network and then through Internet, to the Surrogate Host. It is described in Section 2.2. Section 2.3 introduces the Transmission Control Protocol (TCP) that operates on top of IP to provide a reliable data transport service with flow control. Finally, Section 2.4 and 2.5 describe the HyperText Transfer Protocol (HTTP) and Body Area Network interconnect protocol (BANip) used for application layer functionality, such as continuously transferring biosignals from MBU to SH.

2.1 General Packet Radio Service (GPRS)

This section is divided into two subsections. The first subsection provides an overview of the General Packet Radio Service. The second subsection describes the GPRS protocol architecture and elements.

2.1.1 Overview

GPRS is a European Telecommunication Standards Institute (ETSI) standard [EN260]. Overall, it is a complex standard based on the widespread Global System for Mobile communications (GSM) standard for mobile telephony.

In GSM networks, a permanent connection is established for its users for the duration of their call. Each call connection occupies precious resources in the frequency spectrum available for GSM, whether there is speech data to be transmitted or not. It is obviously inefficient for the connection to occupy a fixed amount of resources equal to those needed if the two people in the call were talking at the same time, non-stop from call start to call end. This permanent connection scheme is even more inefficient for Internet applications such as web browsing, which is characterized by bursts of data to be transferred followed by periods of inactivity (in which the user looks at the downloaded web page).

GPRS extends GSM with the same packet switched technology that makes the modern day Internet so efficient. For GPRS connections, frequency resources are allocated on the fly when the data is ready for transmission. Since there is no fixed amount of frequency resources allocated for GPRS connections, it is not only possible to decrease this allocation when they are not transmitting data, but also to increase it beyond what a GSM connection offers (9.6 kbit/s). This enables higher maximum transfer rates (a theoretical maximum of 171 kbit/s, in practice currently about 50 kbit/s), a second important advantage of GPRS.

Existing GSM networks can be upgraded to support GPRS functionality. The resulting coexistence of technologies is show in Figure 2.1.

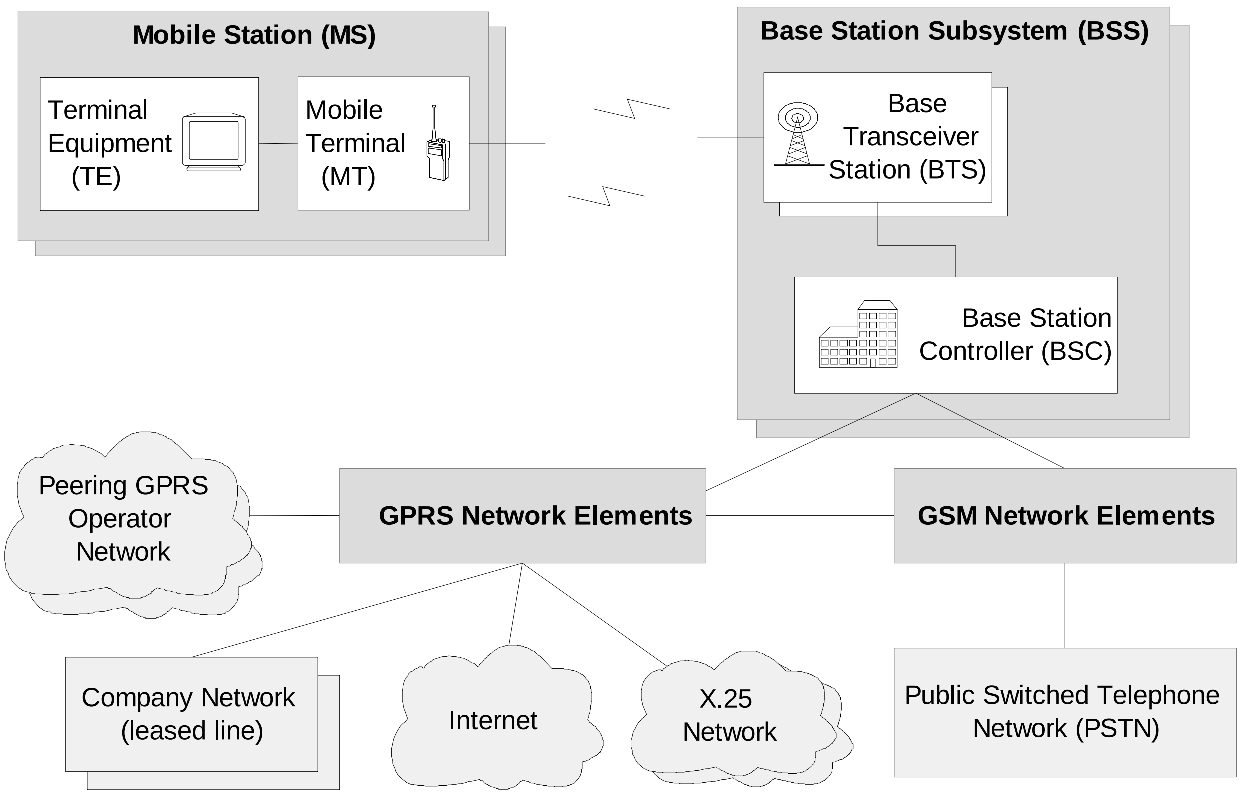

Mobile Station (MS)
Terminal Equipment (TE)
Mobile Terminal (MT)
Base Station Subsystem (BSS)
Base Transceiver Station (BTS)
Base Station Controller (BSC)
Peering GPRS Operator Network
GPRS Network Elements
GSM Network Elements
Company Network (leased line)
Internet
X.25 Network
Public Switched Telephone Network (PSTN)

Figure 2.1: GSM network with GPRS upgrade

A GSM network enhanced with GPRS consist of a Base Station Subsystem (BSS) and GPRS and GSM network elements. Mobile Stations (MSs) can exchange data with the BSS through a radio link. GPRS and GSM network elements both make us of the same BSS, and even the same radio frequency spectrum. If a GPRS call is received, it is routed to the GPRS network elements and from there, a number of destinations including the Internet are possible.

A Mobile Station consists of Terminal Equipment (TE) and a Mobile Terminal (MT). TE is the equipment with which the user interacts, such as a laptop. The MobiHealth project employs an iPAQ, which is a handheld computer. The MT is a device responsible for the radio communications with the (usually nearest) Base Transceiver Station (BTS). Example MTs used in the MobiHealth project are include a wireless expansion pack for the iPAQ and a GPRS capable cellphone. Each BTS covers a so-called cell around it, a limited range within which it handles a limited number of MSs (as permitted by the radio frequency resources allocated to that BTS). To cover large geographical areas, multiple BTSs are set up with slightly overlapping cells. A Base Station Controller (BSC) is used to manage these BTSs, making sure for instance than adjacent BTSs use separate parts of the GSM radio frequency spectrum to avoid interference.

In case an MS enters an area where its “home” GPRS operator does not have any BTS coverage it could try to connect to BTSs owned by other operators. This is called roaming. If the other operator the MS is trying to access has a network connection with its home operator (peering networks), its traffic can be routed to that operator and then on to several destinations again.

Figure 2.2 below shows the GPRS and GSM network elements in more detail.

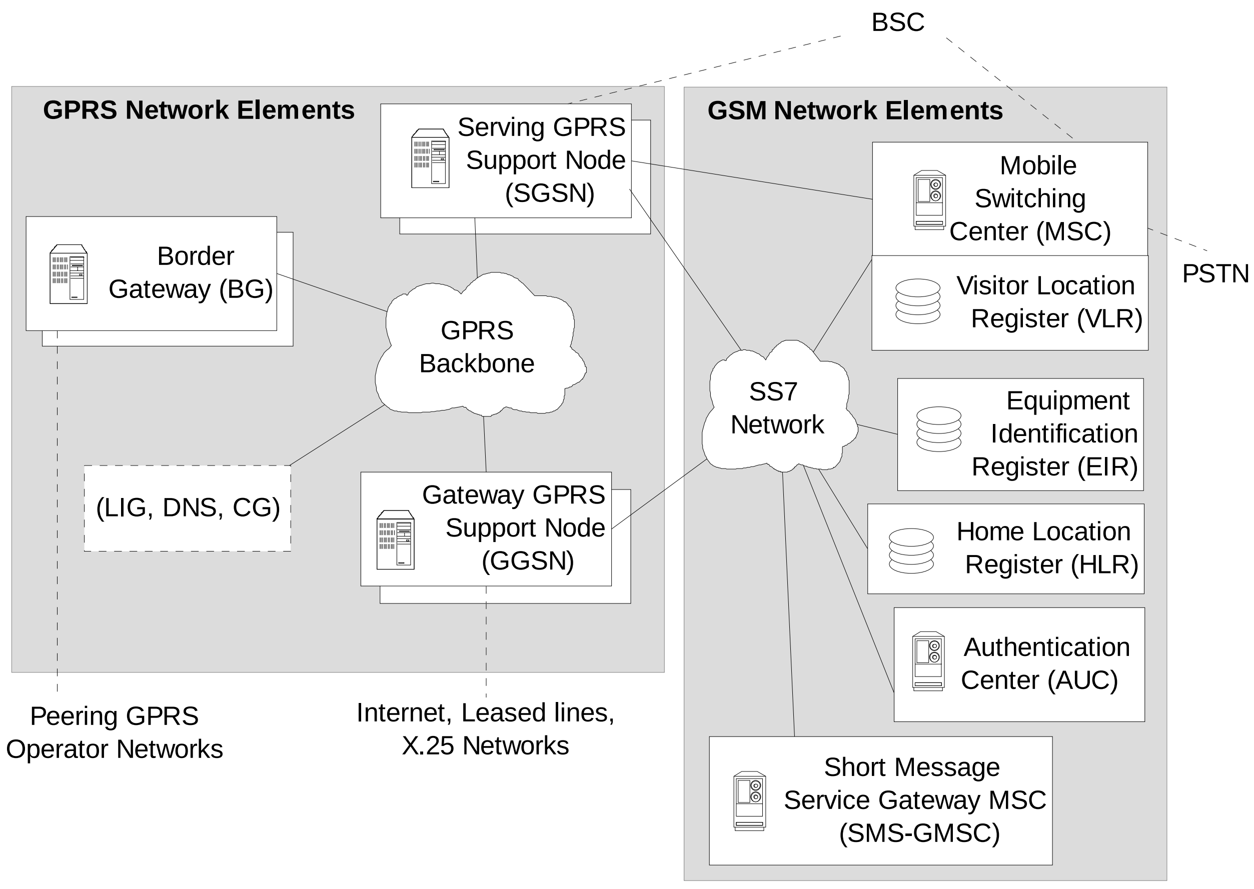

BSC
GPRS Network Elements
Serving GPRS Support Node (SGSN)
GSM Network Elements
Mobile Switching Center (MSC)
PSTN
Border Gateway (BG)
Visitor Location Register (VLR)
GPRS Backbone
SS7 Network
Equipment Identification Register (EIR)
(LIG, DNS, CG)
Gateway GPRS Support Node (GGSN)
Home Location Register (HLR)
Authentication Center (AUC)
Peering GPRS Operator Networks
Internet, Leased lines, X.25 Networks
Short Message Service Gateway MSC (SMS-GMSC)

Figure 2.2: GSM and GPRS enabling network elements

There are two key elements that enable a GSM network to support GPRS. These are the Serving GPRS Support Node (SGSN) and the Gateway GPRS Support Node (GGSN) [EN360].

The SGSN is at the same hierarchical level as the Mobile Switching Center (MSC). It implements support for a number of functions:

- network access control (e.g. authentication, authorization)
- packet routing and transfer (e.g. relay, compression, ciphering)
- mobility management (keeping track of the location of the MS)
- logical link management (link establishment, maintenance and release)
- radio resource management.(communicating with the BSS)

In short, a SGSN allows MSs to access core GPRS services. It relays MS data towards its destination and queues inbound data for MSs.

The GGSN comes into play when a MS needs to access external data networks. It is attached to the internal GPRS (usually IP) backbone and provides interworking functions to external networks. When encapsulated data is received from a SGSN (ultimately from a MS), it is decapsulated and routed to the appropriate destination net. When data is received for a MS, it is encapsulated and routed over the backbone to the SGSN that "knows" the MS in question. Similar to the SGSNs, there can also be multiple GGSNs.

Examples of additional hosts that can be introduced on the GPRS backbone include:

- Border Gateway (BG)
  Connecting to peering GPRS operator networks (e.g. for roaming agreements)

- Charging Gateway (CG)
  Keeping track of user data volumes for billing purposes (although the SGSN can also perform this function)

- Domain Name Server (DNS)
  Providing a faster lookup of hostnames to IP addresses for MSs

- Lawful Interception Gateway (LIG)
  Enabling law enforcement agencies to listen in on data transmissions of particular MSs

2.1.2 Protocol Architecture

Figure 1.9 shows the BANip protocol stack with the General Packet Radio Service as a "black box". It spans the bottom three OSI reference model layers. The protocol stack shown in Figure 2.3 provides a more detailed insight into the way GPRS is designed with respect to these layers.
N.B.: Some black boxes remain, out of relevancy considerations

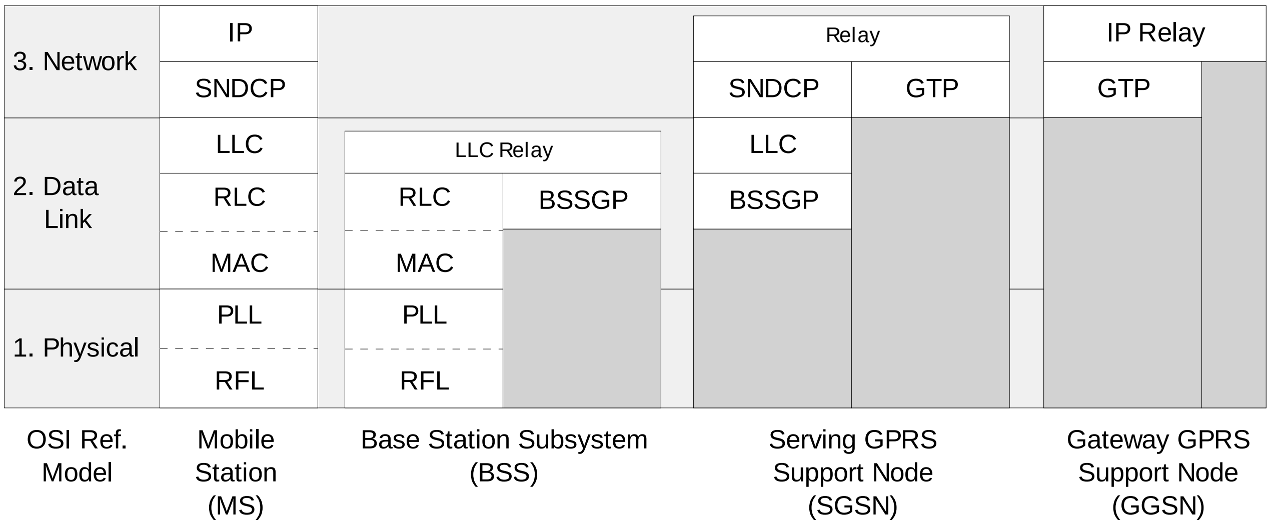


Figure 2.3: GPRS elements protocol stack

The functionality of the layers is described below from the point of view of a Mobile Station, lowest layer first.

- **RFL - physical Radio Frequency subLayer**

The **RFL** provides physical waveform modulation / demodulation. The signal latency at this level is comparable to wired links (as opposed to the latency on wireless links such as satellite links).

Figure 2.4 shows how radio frequencies are allocated for the transmission of the waveforms. Note that these are frequencies allocated for GSM and that GPRS transmissions actually make use of the same frequency spectrum as GSM.

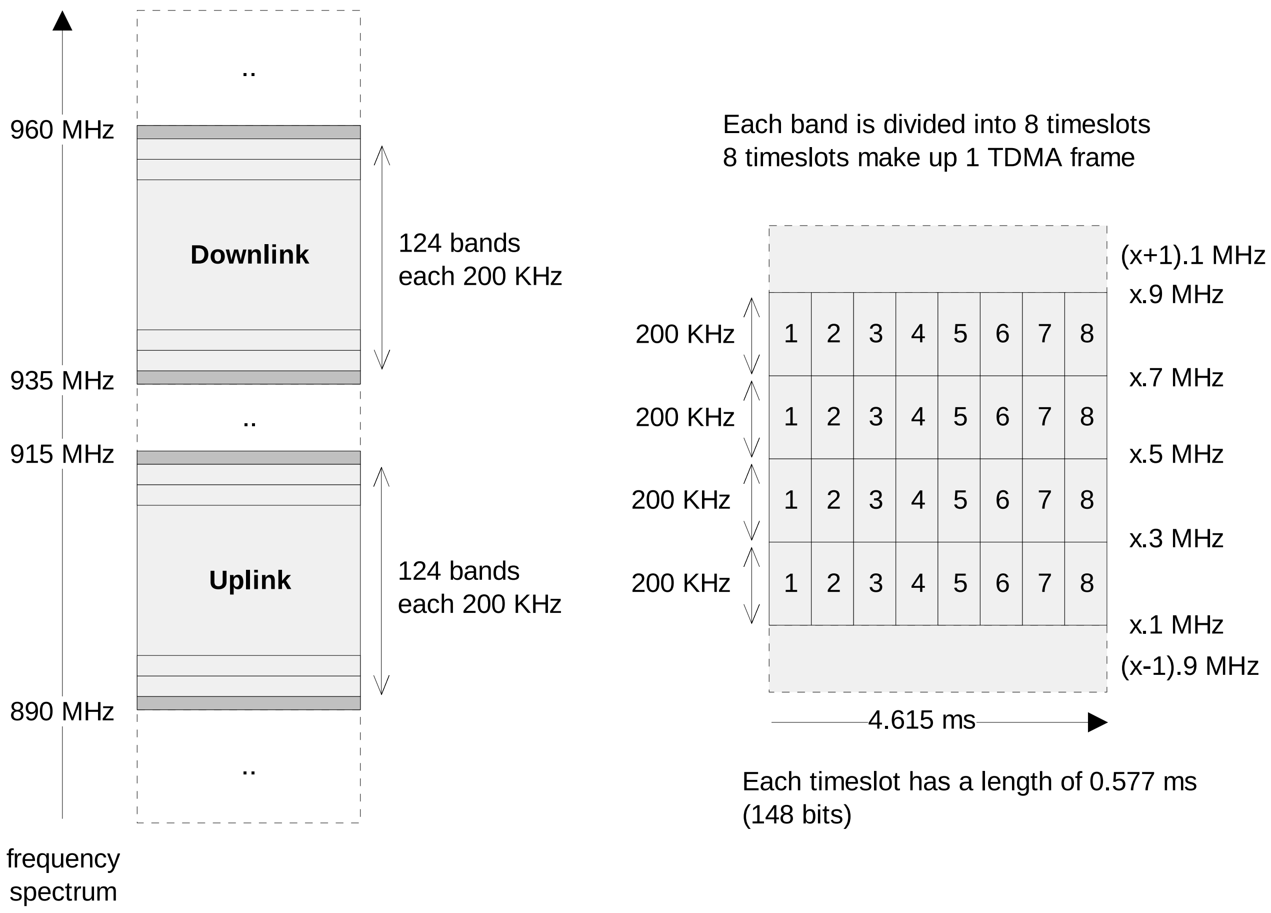


Figure 2.4: GSM/GPRS frequency usage and the concept of timeslots

The left part of the figure shows that a range of frequencies around the 900 MHz[5] band are reserved for GSM/GPRS transmissions. The frequency range 890 MHz – 915 MHz is reserved for uplink data transfer, which is data transfer from the MS to the BSS. 935 MHz – 960 MHz is allocated for downlink data, from the BSS to the MS.

In order for multiple users in the same cell to share downlink and uplink access, a combination of Frequency Division Multiple Access (FDMA) and Time Division Multiple Access (TDMA) is used.

[5] This band varies: in Europe 900 MHz and 1800 MHz are used while the United States has allocated frequencies around the 1900 MHz band. Some Eastern European countries use the frequencies around 450 MHz

The FDMA scheme does exactly as the name suggests: both the uplink and downlink frequency ranges are divided into 124 frequency bands (of 200 KHz each). That allows 124 users to continuously and simultaneously transmit data over their assigned uplink and downlink frequency.

The actual frequencies used for each of the GSM/GPRS bands can be calculated as follows. On the outer edges of the uplink and downlink ranges, there are 100 KHz separation bands. Consequently, the formula that relates frequency band and actual transmission frequency is:
Freq = BASE_MHz + 100 KHz + (200 * BAND_Nr / 2) KHz

For example, the transmission frequencies for the uplink and downlink transfers on band 1 are:
Uplink_Freq = 890 MHz + 100 KHz + (200 * 1 / 2) KHz = 890.2 MHz
Downlink_Freq = 935 MHz + 100 KHz + (200 * 1 / 2) KHz = 935.2 MHz
These frequencies represent the center of the two 200 KHz bands (890.1 MHz - 890.3 MHz and 935.1 MHz - 935.3 MHz), as can also be seen in Figure 2.4.

The second scheme, TDMA, further increases the amount of concurrent users. Each of the 248 frequency bands is divided in time. The idea behind using TDMA is that not every user requires access to “their” frequency band 100% of the time. For GSM this is true because of the quality of speech used. Sampled bits of sound information do need to be sent periodically, but in-between each transmission there is a pause long enough for other users to transmit their own samples. For GPRS this is even more obvious, because the whole system is based on the idea that users are not sending / receiving all the time, but rather transmit every so often, which leaves room for other users to share the frequency.

The right part of Figure 2.4 shows how TDMA is applied. The frequencies are divided into TDMA-frames. Every 4.615 ms a new TDMA-frame begins. Each TDMA-frame is divided into eight equal-length timeslots. Both GSM and GPRS users can make use of these timeslots to transmit (or receive, if it is a downlink frequency) data. The difference between the two is that GSM users require one timeslot in each TDMA-frame on both uplink and downlink frequencies, whereas GPRS users do not get any set amount of pre-allocated timeslots. They must request slots on the uplink when they have data to transmit, and are assigned slots in the downlink when data has arrived for them (uplink and downlink slot assignments are independent from each other).

As mentioned in Figure 2.4, each timeslot has a length of 0.577 ms, during which 148 bits of data are transmitted. As shown in Figure 2.5, only 114 of these bits contain data from higher layers (payload). The rest is required so that the receiver can determine unambiguously which data is sent during which timeslot.

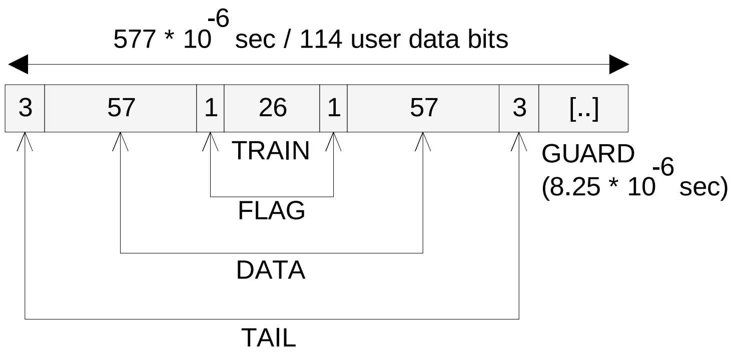


Figure 2.5: Usage of one timeslot

▪ **PLL - Physical Link subLayer**

The GPRS **PLL** offers a data transfer service through RFL timeslots. It is responsible for issues like detecting physical layer congestion and adding data redundancy. The latter has a profound impact on the speed at which the PLL can transmit higher layer data (payload). Figure 2.6 shows why.

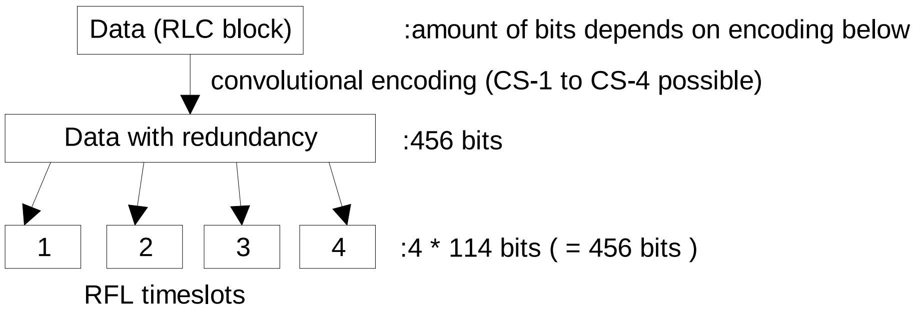

Data (RLC block)
:amount of bits depends on encoding below
convolutional encoding (CS-1 to CS-4 possible)
Data with redundancy
:456 bits
1
2
3
4
:4 * 114 bits (= 456 bits)
RFL timeslots

Figure 2.6: Adding data redundancy

There are four so-called “coding schemes” available for adding data redundancy. These are named CS-1 to CS-4. Each coding scheme represents a convolutional encoding that maps a certain amount of data onto 456 bits (4 timeslots). CS-1 provides the most error detection and correction possibilities while CS-4 offers virtually no opportunity to do this. Consequently, CS-1 requires more redundant data to be present in each of the 456 bits, so less unique data (infobits) can be encoded. The table below shows how many infobits each convolutional encoding can fit in 456 bits.

Coding Scheme	**Infobits per 456 bits**	**Theoretical max. 1-slot[6] throughput (kbps)**
CS-1	181	9.05
CS-2	268	13.40
CS-3	312	15.60
CS-4	428	21.40

[6] 1-slot throughput means: the achievable throughput when 1 slot per TDMA frame is available

Encoding less infobits per 456 bits to be transmitted results in a lower maximum throughput of infobits. So, a coding scheme with higher data redundancy leads to a lower maximum throughput, as is also indicated by the table.

For wireless links, CS-2 is probably the most commonly used coding scheme because of the effectivity of its performance given the average bit error rate (BER) of the link. CS-1 would provide too much redundancy, artificially slowing itself to a lower actual maximum throughput than CS-2, while CS-3 might provide too little redundancy, leading to a slowdown from retransmissions that more than negates its theoretical throughput advantage over CS-2.

- **MAC: Medium Access Control**

The **MAC** sublayer provides contention resolution between (uplink) radio channel access attempts, arbitration between multiple service requests from different MSs and medium allocation to individual MSs in response to service requests.

A slotted ALOHA mechanism is used for this functionality. It has 3 phases on the uplink (MS to BTS) and 2 on the downlink (BTS to MS):
Uplink phases:

1. contention - MSs “fighting” for radio resources to transmit data to BTS
2. notification - which MS may use a certain frequency, and when
3. transmission- actual data transfer

Downlink phases:

1. notification - which MS the data on the downlink is destined for
2. transmission- actual data transfer

The relationship from BTS to MS is a one to many relationship. While MS must “fight” to use a certain (uplink) radio frequency for transmitting data to the BTS, the BTS need only to signal a certain MS to listen for its data on the downlink. Consequently, there is no need for contention resolution on the downlink.

- **RLC: Radio Link Control**

The **RLC** sublayer provides a radio-technique-dependent highly reliable link. Data from higher layers is segmented into payload for RLC blocks, as shown in Figure 2.7.

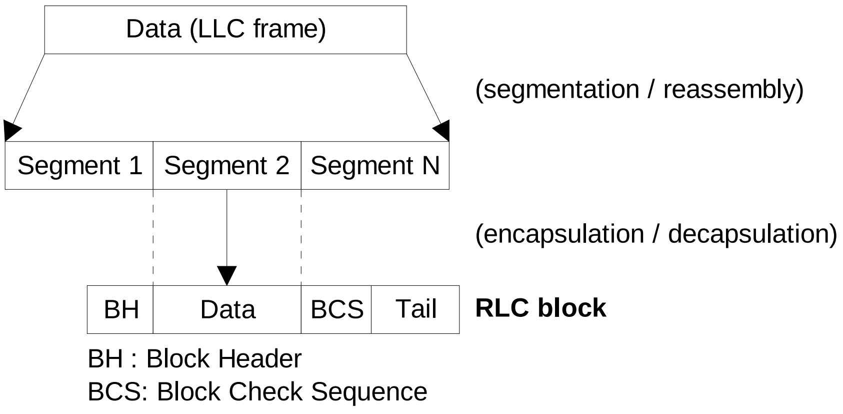

Data (LLC frame)
(segmentation / reassembly)
Segment 1
Segment 2
Segment N
(encapsulation / decapsulation)
BH
Data
BCS
Tail
RLC block
BH : Block Header
BCS: Block Check Sequence

Figure 2.7: Data segmentation/reassembly for RLC blocks

Reliability is achieved through Block Level Retransmissions (BLR).

Whenever RLC blocks are ready for transmission, a Temporary Block Flow (TBF) must be established, indicating the amount of RLC blocks to be transferred[7]. Once the TBF is established, RLC blocks can be transferred using a certain amount of slots per TDMA frame. A GPRS link may use more than one uplink and downlink slot per TDMA frame. GPRS devices are divided in twelve multislot classes, indicating their support for actually utilizing multiple slots. The classes are listed in the table below.

Multislot class	**Max. downlink slots**	**Max. uplink slots**	**Max. active slots**
1	1	1	2
2	2	1	3
3	2	2	3
4	3	1	4
5	2	2	4
6	3	2	4
7	3	3	4
8	4	1	5
9	3	2	5
10	4	2	5
11	4	3	5
12	4	4	5

A class 3 device for example supports using 2 downlink slots and 1 uplink slot per TDMA frame for a total of 3 active slots. Alternatively the device can also utilize 1 downlink slot and 2 uplink slots at the same time. Note that the actual granting of the slots is still dependent on

[7] Devices are prohibited to request transmission for more RLC blocks than they have available in their buffer, though some GPRS operators may allow this number to remain unspecified.

the GPRS operator network policies and traffic intensity. The multislot class only indicates device capabilities. So while a class 12 device could handle, for instance, 4 uplink slots and 1 downlink slot at the same time, it might only be able to use 1 up- and downlink slot in total on a busy network.

- **LLC: Logical Link Control**

A highly reliable logical link between a MS and a SGSN is provided by the **LLC** sublayer. The abstraction from a physical link into a logical link means that the LLC supports multiple unique connections. In practice, these separate connections are used for mobility management, the Short Message Service (SMS) and various SubNetwork Dependent Convergence Protocol (SNDCP) Quality of Service (QoS) classes.

The LLC sublayer employs flow control based on a sliding window (and QoS criteria), as well as ciphering for confidentiality. A Frame Check Sequence (FCS) allows data errors to be detected. If the optionally Automatic Repeat reQuest (ARQ) mechanism is enabled, retransmissions are used to correct these errors. Figure 2.8 shows the layout of an LLC frame.

- **SNDCP: SubNetwork Dependent Convergence Protocol**

The SNDCP is used to transfer data packets between the MS and SGSN. The underlying LLC sublayer provides one virtual logical connections. SNDCP multiplexes packets from different protocols onto this connection and has other functionalities such as TCP/IP header compression, data compression (e.g. V42.bis, also used in analog modems) and data segmentation for packets larger than the maximum LLC frame data size (see Figure 2.8).

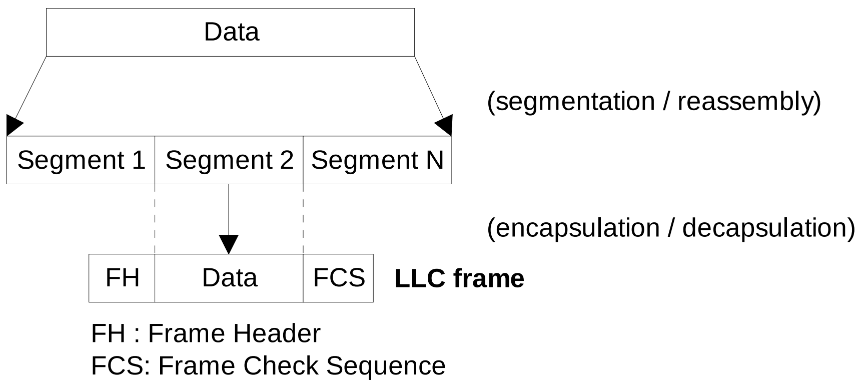

Data
(segmentation / reassembly)
Segment 1
Segment 2
Segment N
(encapsulation / decapsulation)
FH
Data
FCS
LLC frame
FH : Frame Header
FCS: Frame Check Sequence

Figure 2.8: Data segmentation/reassembly for LLC frames

- **IP: Internet Protocol**

GPRS offers support for IP, one of the fundamental Internet protocols. It is described further in the next section.

2.2 Internet Protocol (IP)

The Internet Protocol (IP) is one of the fundamental protocols in the Internet. Specified by [RFC791], it is a network layer protocol that is supported in one form or another by the different networks that make up the Internet. Any pair of hosts that is connected to the Internet should be able to exchange IP datagrams, assuming there are no network policies preventing either of them from exchanging data with the other. Routers, also fundamental parts of the Internet, have the task of finding the route that the IP datagram should take from one host to another. Individual IP datagrams are not guaranteed to reach their destination (for example, they might be dropped by a congested router), nor are they guaranteed to arrive in the same order as they were sent.

Because the IP layer provides a universal interface, it effectively hides the type of the networks its datagrams traverse. In the MobiHealth project, the surrogate host cannot know for sure that MBU datagrams it receives indeed originated from a mobile station with a wireless connection to GPRS network or from a host connected through a wired network[8].

What the IP layer cannot hide, however, are the characteristics of the networks crossed (in terms of throughput and latency). This means that these will manifest themselves this layer. Since all networks in the Internet are unequal with respect to lower layer characteristics, this is generally not a problem for the higher layer protocols, because they are designed with these differences in mind. However, some networks are more unequal than other networks. As section 2.1 shows, the characteristics of wireless (GPRS) networks differ vastly from the majority of the wired networks making up the Internet. This unexpected difference has been shown to break some IP based protocols such as the Transmission Control Protocol (TCP) used in the interconnect between MBU and its Surrogate Host (see the next chapter, 2.3).

There is a second disadvantage to the network independency of IP. Datagrams travelling across the interconnect are influenced by the characteristics of both the wireless (GPRS) network and wired networks which are part of the Internet. Without special facilities along the route the interconnect covers, hosts on either side cannot tell which forces are at work. If, in a stream of datagrams, the next datagram does not arrive as fast as the others did, does that mean it is just being delayed by the wireless network due to (perhaps) atmospheric interference, or was it dropped by a congested router on the wired network? Should the receiver wait a little more for the datagram to arrive or should it stop waiting and ask for retransmission right away?

Despite these disadvantages, the Internet Protocol has proven itself in over two decades to be robust and flexible, and remains something the Internet cannot do without.

[8] Most GPRS operators do not even give out Internet IP addresses to Mobile Terminals, so the Surrogate Host will see the IP address of the GGSN as the source IP, which has a wired connection.

2.3 Transmission Control Protocol (TCP)

The Transmission Control Protocol (TCP) was designed to provide a reliable end-to-end byte stream when the underlying network is unreliable ([Tane96] p.552). Moreover, TCP was designed to work over an *internetwork*, which may be a combination of different network topologies, bandwidths, latencies, packet sizes to which it can dynamically adjust.

TCP has been around for over two decades now; the Internet Engineering Task Force (IETF) has specifications of TCP as a standard as well as extensions. See Figure 2.9.

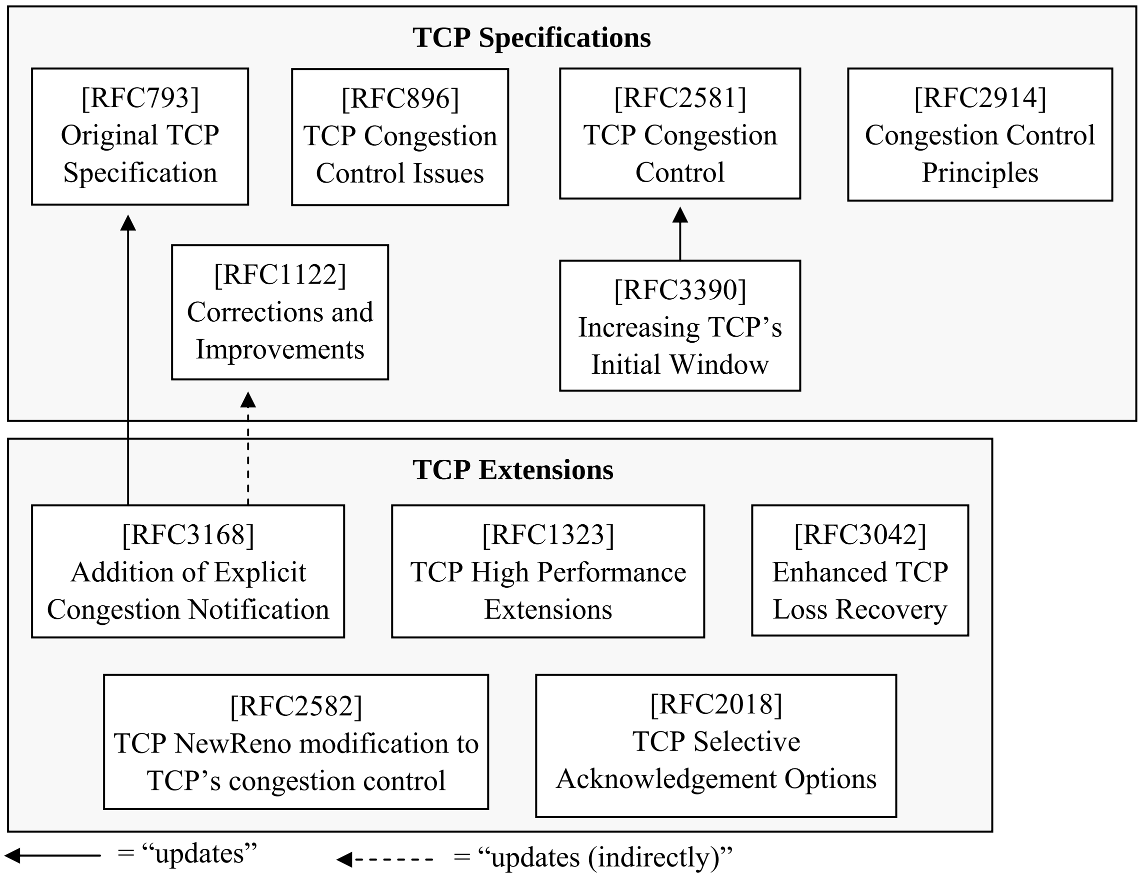

TCP Specifications
[RFC793] Original TCP Specification
[RFC896] TCP Congestion Control Issues
[RFC2581] TCP Congestion Control
[RFC2914] Congestion Control Principles
[RFC1122] Corrections and Improvements
[RFC3390] Increasing TCP's Initial Window
TCP Extensions
[RFC3168] Addition of Explicit Congestion Notification
[RFC1323] TCP High Performance Extensions
[RFC3042] Enhanced TCP Loss Recovery
[RFC2582] TCP NewReno modification to TCP's congestion control
[RFC2018] TCP Selective Acknowledgement Options
= "updates"
= "updates (indirectly)"

Figure 2.9: Internet Engineering Task Force & Transmission Control Protocol

After being introduced as a standard in [RFC793] in1981, TCP has become one of the most widely used transport layer Internet protocols.

To communicate through the TCP protocol, two hosts must set up a TCP connection between them. This connection is a full duplex connection, meaning that data flows both ways. It can also be thought of as two simplex connections, since either side can "close" the data flow direction originating from them at any time.

Individual *segments* containing user data are sent over the connection. A segment is a packet of data consisting of TCP header information and optionally, user data. To make the connection reliable, the receiver acknowledges data from every segment.

Figure 2.10 and Figure 2.11 show the typical TCP connection establishment and connection closing sequences. Time progresses from left to right in the figures.

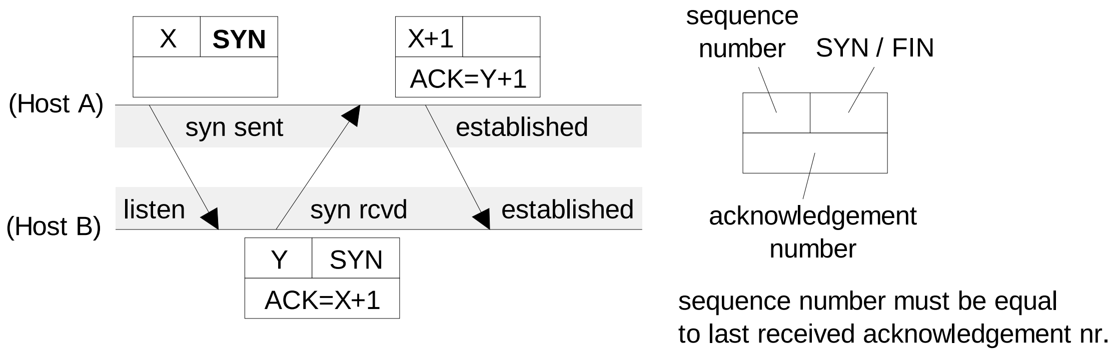


Figure 2.10: TCP connection establishment

Host A initiates connection to Host B. TCP's connection establishment is a "three way handshake" between two hosts. Segments containing SYN must have been transmitted both ways, and both of them must be acknowledged for the connection to be fully established. Segments will be automatically retransmitted if the acknowledgement does not arrive within a short time. Over a fully established connection, segments with user data can be exchanged.

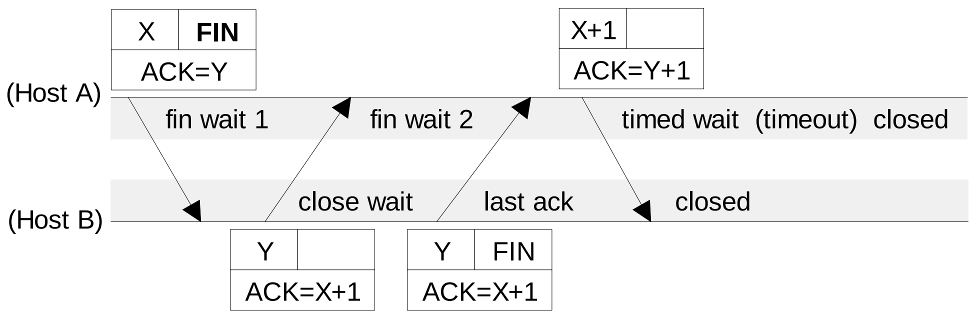


Figure 2.11: TCP connection closing

Host A initiates disconnection from Host B. In the same way that SYNs are exchanged during connection establishment, FINs must now be exchanged. After B receives the FIN sent by A, it acknowledges it and continues transmitting its own segments until it is ready to close the connection as well. When it's ready, its own FIN is sent. Host A then needs a timed wait before definitely closing (and discarding) the connection because the last ACK it sends could be lost. If that happens, a retransmitted ACK from B is to be expected, and in order to process it correctly (and retransmit its own ACK) A will need its connection information to still be intact.

While the connection is established, segments may be exchanged. To control the flow of info, a *sliding window* is used. The host on either side of the connection advertises a window size. The window size is the maximum amount of bytes[9] that the advertising host is prepared to accept, starting from the location of the next expected byte. In this way, if a receiving host's buffers become saturated with incoming data, it can slow the sending host down by advertising a smaller window (or stop it completely with a zero sized window). Note that by

[9] Many explanations use segments to explain the sliding window protocol for simplicity. In reality, bytes are used, though some TCP implementations only specify byte sizes that are multiples of segments.

delaying acknowledgements the receiving host can also slow down the sender, but by doing this the receiver risks the sender's retransmission timer expiring. This retransmission timeout (RTO) will cause the sender to retransmit the segment it sent. Since no new information is sent there is no additional strain on the receiver's buffers, but the retransmission itself is obviously a waste of bandwidth.

Besides the capability of the receiver to process data, the link capacities and traffic intensity on the network path between the sender and receiver are also important. Ideally, the combined traffic intensity on the path should be less or equal than its capacity. When the combined transmission rate exceeds path capacity, data will start to fill router buffers until they too become saturated, after which data drops must occur. Packet queuing (sometimes) and packet drops cause retransmissions that further saturate the network. Eventually, "congestion collapse" [RFC896], [RFC2914] occurs, which is basically a serious decrease in usage efficiency (of the capacity of the network). It is therefore in the interest of protocols to use congestion control. TCP uses a so-called *congestion window* [RFC2581] in addition to the sliding window. How much data is actually sent out is controlled by the minimum of the two window sizes.

TCP uses four intertwined congestion control algorithms: slow start, congestion avoidance, fast retransmit and fast recovery. These are shown in Figure 2.12.

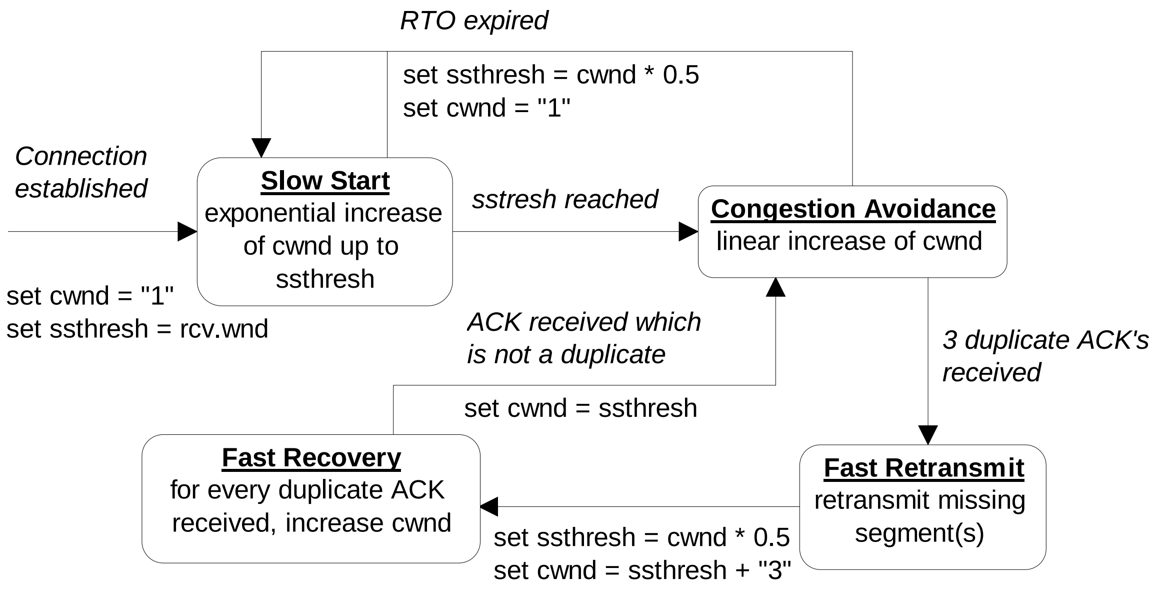

RTO expired
set ssthresh = cwnd * 0.5
set cwnd = "1"
Connection established
set cwnd = "1"
set ssthresh = rcv.wnd
Slow Start
exponential increase of cwnd up to ssthresh
sstresh reached
Congestion Avoidance
linear increase of cwnd
ACK received which is not a duplicate
set cwnd = ssthresh
3 duplicate ACK's received
Fast Recovery
for every duplicate ACK received, increase cwnd
Fast Retransmit
retransmit missing segment(s)
set ssthresh = cwnd * 0.5
set cwnd = ssthresh + "3"

Figure 2.12: TCP congestion control algorithms (TCP Reno)

These algorithms operate as follows. Starting with no knowledge about the congestion state of the connection, the slow start phase initializes the congestion window (cwnd) to a small amount of bytes[10] (depicted as "1" in Figure 2.12) and probes for the capacity of the network by exponentially increasing cwnd. This process will continue until either an RTO occurs or

[10] According to the TCP specification, "small amount of bytes" must be less than, or equal to 2 times the size of the largest segment the sender can transmit (usually set to the maximum transfer unit (MTU) of the medium which is 1500 bytes for ethernet connections). "small amount of bytes" must also fit in two segments or less.

cwnd grows beyond the slow start threshold (ssthresh). This ssthresh can initially be an arbitrarily high number, but is usually set to the receivers advertised window (rcv.wnd).

RTOs are taken as an indication of network congestion. In the first case, ssthresh was not reached and its value must be set to half the congestion window at the time of the RTO (but never less than the maximum initial congestion window). The slow start phase is then restarted “from scratch” (while keeping the ssthresh adjustment).

As soon as cwnd manages to reach ssthresh, the congestion avoidance phase is initiated. This phase slows down the cwnd increase, providing a more conservative (linear) “probing for additional network capacity”. An RTO in this phase results in the same behavior as during the slow start phase, namely the adjustment of ssthresh and restart of the slow start phase.

The TCP version which used only the above two mechanisms is called OldTahoe. The two remaining algorithms were devised to allow TCP to recover from losses more efficiently than by using the RTO and slow start “from scratch”. The first, fast retransmit, is contained in TCP Tahoe. The addition of a fast recovery phase was done in TCP Reno. Reno is one of the most popular versions in the current Internet. TCP NewReno contains some modifications to the state transition from Reno’s fast recovery.

Fast retransmit works as follows. Suppose that the receiver has already received S_0 and that the order in which the following two segments, S_1 and S_2, are delivered is reversed. Upon reception of S_0 the receiver acknowledged to the sender that it has received up to S_0 contiguous segments. S_2 then arrives, but since S_1 has not arrived yet, the receiver has no choice but to acknowledge that it has only received up to S_0 contiguous segments. This is the exact same acknowledgement, so the sender will receive a duplicate acknowledgement. If *three* such duplicate acknowledgements occur in a row, the sender presumes S_1 to be lost because it is unlikely that the order of the packets is altered so drastically. The retransmission of S_1 after three duplicate acknowledgements is called a fast retransmission, because the sender did not wait for the RTO to occur first, thereby saving time. Similar to when an RTO occurs, ssthresh is set to half of the current congestion window, but the congestion window itself is only reduced to ssthresh plus three segments. The three-segment addition comes from the argument that three (duplicate) acknowledgements also indicate that three segments beyond S_1 have already left the network, so that there is probably “room” for three more.

This is the start of the fast recovery phase, in which the congestion window will be further “inflated” for every additional duplicate acknowledgement received, until a non-duplicate acknowledgement is received. This means that new segments may be transmitted if the inflated congestion window and the receiver window allow it. Once a non-duplicate acknowledgement is received, cwnd is “deflated” back to ssthresh and the congestion avoidance algorithm is invoked once again.

This concludes the technical introduction to the Transmission Control Protocol.

2.4 HyperText Transfer Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems [RFC2616]. It is a generic, stateless protocol that can be used for many tasks beyond its use for hypertext. The Wireless Application Protocol (WAP), a de-facto standard for the presentation and delivery of wireless services on wireless terminals ([WAPv1], [WAPv2]), builds on HTTP 1.1 as the protocol for regulating data exchanges. With this in mind, the MobiHealth system also uses HTTP 1.1 on the wireless connection between MBU and SH. Additionally, a HTTP proxy at the “entrance” of the BackEnd System (where the SH resides) authenticates every MBU interconnect. The HTTP proxy is known as the *Wireless Service Broker* (WSB). The whole system is shown in Figure 2.13.

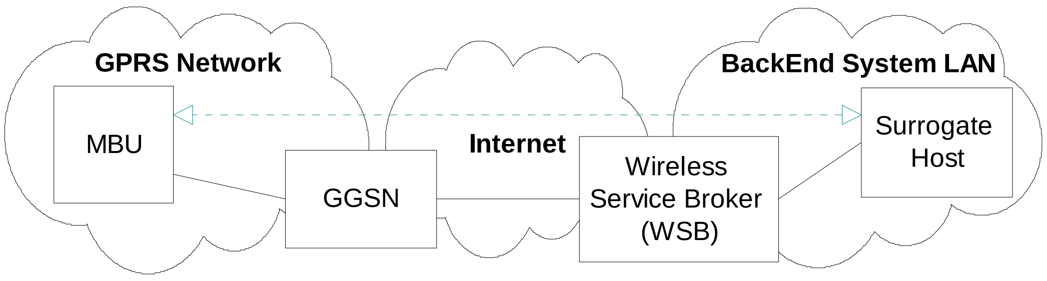


Figure 2.13: The MBU and SH communicate through the WSB (a HTTP proxy)

The MBU should only be able to contact the SH through the WSB for the authentication mechanism to function properly. The MBU must initiate this contact because it will most likely be assigned a private (LAN) address, thereby making it impossible for applications outside the GPRS network to initiate contact to it. Thus, the only way for the MBU and SH to communicate with each other is through the WSB, after the MBU has initiated the contact.

The original HTTP specification is [RFC1945]. Within a year, the protocol specification was updated to version 1.1 [RFC2068] to incorporate important extensions. Even though this specification had a number of problems and ambiguous statements that were fixed a few years later in [RFC2616], the version remained the same, and it still is at the time of this writing.

The basic HTTP 1.1 paradigm is that of request and response. The client, in this case the MBU, sends a HTTP request to the server, which is the surrogate running on the SH. This request carries data from the higher layer BAN interconnect protocol. The surrogate must send a response after that, and then the process can be repeated for the next set of data from the BANip.

2.5 Body Area Network Interconnect Protocol (BANip)

As explained in Section 1.2, the Body Area Network interconnect protocol is responsible for the communication between a MBU and its Surrogate Host (SH).

The main task of the BANip is to transfer biosignals from a MBU to a SH, accompanied by metadata such as type and timing information. The biosignals are measured from a patient through a Front End that is connected to the MBU. The combination of biosignals and metadata will be referred to as *BAN data*. The performance of the BANip is closely related to how it transfers BAN data. Figure 3.1 shows the BANip at work.

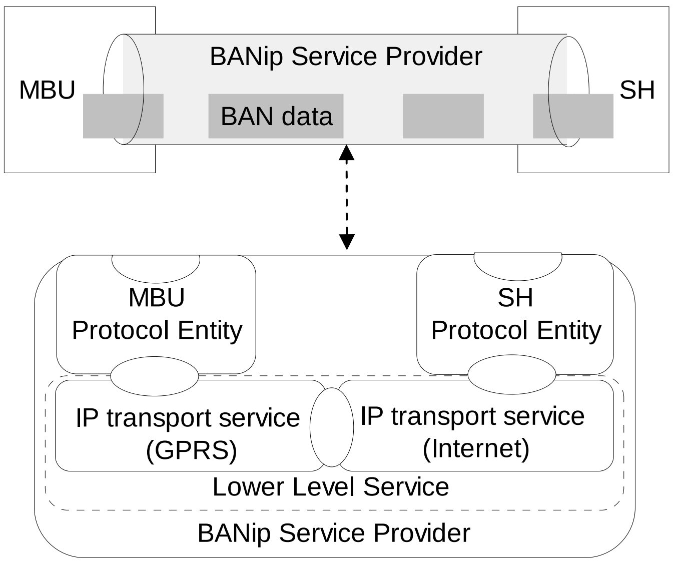


Figure 2.14: BANip transporting BAN data between MBU and SH

There are two general phases that can be distinguished in the operation of the BANip. These are shown in Figure 2.15. The first phase is a setup phase during which the MBU BANip protocol entity arranges a dedicated communication peer in the SH. In the second "data transfer" phase this peer is contacted so that BAN data can be transferred to it. The two phases are described in more detail below.

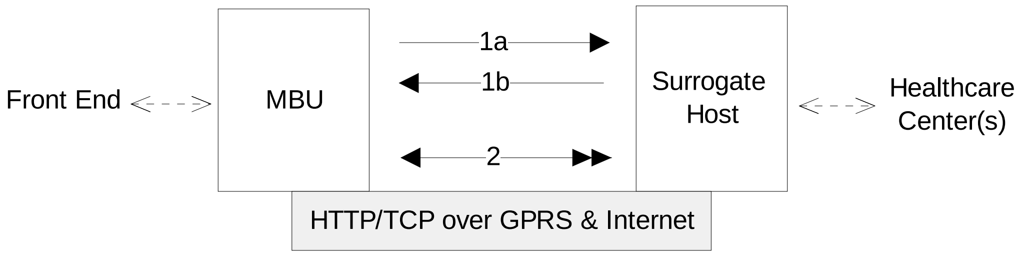


Figure 2.15: BANip session overview

- **Phase 1: setup**
  There are two notable parts in the setup phase. In the context of the Jini system, the MBU must be represented by a surrogate (Java class) running on the SH. Initially, this surrogate is not present on the SH so the first part involves getting it there. The MBU registers with the SH and transmits a URL that specifies the location where the MBU surrogate can be

downloaded from. Once the surrogate is retrieved and activated by the SH, it waits for the MBU to contact it. In the second part of the setup phase, the SH relays a URL to the MBU containing the location of the activated MBU surrogate.

- **Phase 2: data transfer**
  Having been informed on how to contact the surrogate, the MBU establishes a logical connection with its surrogate in phase two. The MBU can then transfer BAN data gathered from the Front End to its surrogate over this connection. To meet the requirements for a Jini interconnect protocol (discovery, surrogate retrieval, liveness), the MBU also sends periodic keepalives to its surrogate. If something bad happens to the surrogate or SH, the MBU will not be able to successfully transmit either keepalives or data, so it will know the interconnect is no longer "alive", functional. If the MBU does not report either data or keepalives within a set amount of time to the surrogate, it will also know that the interconnect is no longer alive. Therefore, the keepalive mechanism is sufficient to enable both MBU and surrogate to know whether the interconnect is still functional. If the interconnect breaks down, the surrogate is cleaned up by the SH after a certain amount of time. If the MBU deregisters from the surrogate, the latter is cleaned up right away.

Figure 2.16 shows how data from the Front End (sensor array) is packaged and transmitted to the surrogate running on the SH.

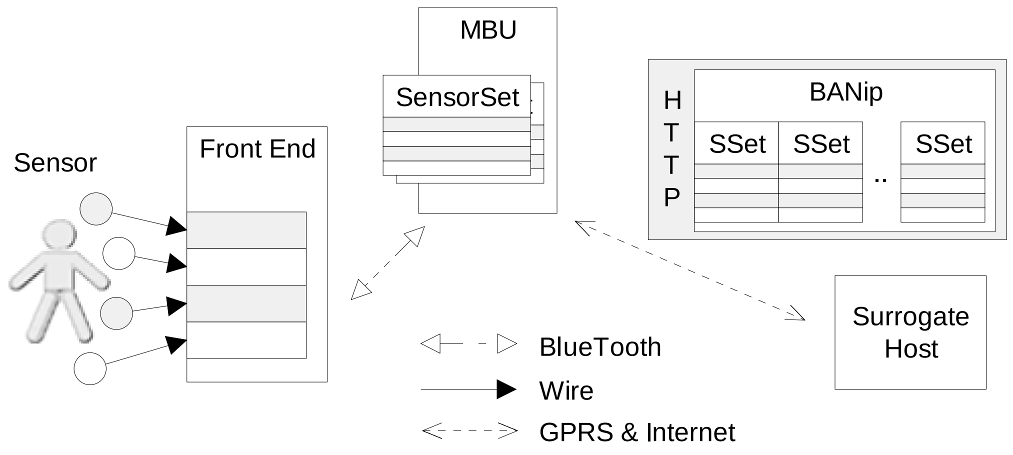


Figure 2.16: BANip data transfer

Sensor objects represent sensors connected to the Front End. Every time a sensor reading is requested, all sensor objects are filled with the current value of the sensor they represent. The MBU groups these sensor objects together in a 30-byte *SensorSet*, usually called *BAN data unit* in this thesis. This means that within every SensorSet, sensor objects (and the sensor readings they represent) are synchronized to the same instance of time. SensorSet groups are then transferred to from MBU to SH as content in HTTP requests.

There is one more mechanism at work between the MBU and SH that should be described. The BANip implements a so-called *transcoding chain*, as shown in Figure 2.17

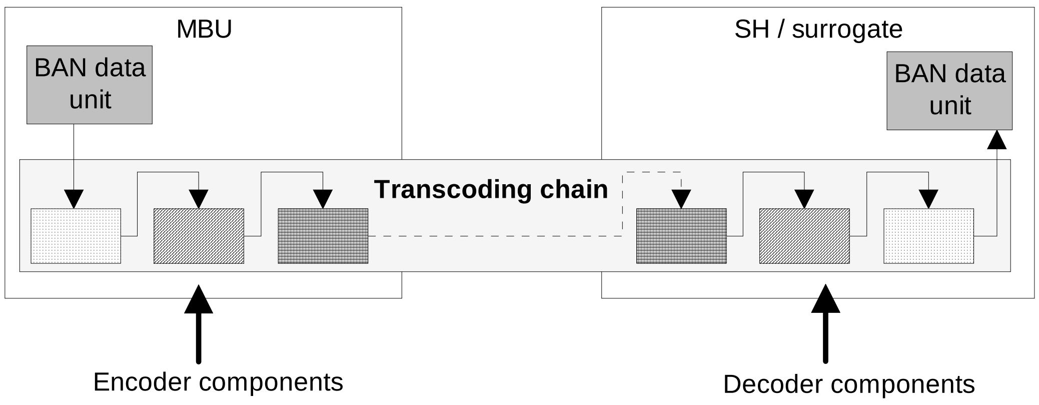


Figure 2.17: BANip transcoding chain

A transcoding chain is a chain of customizable components through which BAN data units proceed in sequence before they are handed over to HTTP for transfer. Each component consists of a ComponentEncoder and ComponentDecoder part, which reside at the MBU and SH respectively. An important component that is referred to throughout this thesis is the so-called *packetizer*. Implemented as the last Encoder (and first Decoder), the packetizer aggregates a (configurable) number of BAN data units into groups for more efficient transfer.

The number of BAN data units aggregated in each group will be referred to as the **packetizer setting**. The packetizer setting can be set in the range of 1 to 255 BAN data units per group in the reference BANip, which is equal to groups of 30 to 7650 bytes.

The BANip has no built-in Quality of Service (QoS) differentiation. This means that all BAN data units are transferred in the order that they are offered to the BANip, so there is no concept of priority that one might have expected in such a system. Note also that the SensorSet time synchronization mechanism implies that each sensor must be sampled at the same frequency[11]. In other words, sensors will always be sampled at the maximum of the various sample rates needed by sensors in the set. For example, if one sensor requires a sampling rate of 256 Hz while another sensor requires only 64 Hz, both sensors will be sampled at 256 Hz.

[11] Currently, the following four frequencies are supported: 64, 128, 256 and 512 Hz

3. Interconnect Protocol Performance Assessment

This chapter presents the performance criteria that are used to assess the Body Area Network interconnect protocol (BANip), in the form of *Critical Performance Indicators* (CPIs). The CPIs are defined in the first section. The performance of the reference BANip is also quantified. Section 3.2 introduces the strategy used to measure instance values of the defined CPIs. The test setup in which actual measurements are made is specified in Section 3.3. Finally, Section 3.4 presents the results of the CPI tests, representing the quantification of the performance of the reference BANip.

3.1 Critical Performance Indicators

This section introduces a definition of performance as required for the BANip Service Provider[12] performance assessment. Also necessary is a set of performance indicators to express the performance of the BANip numerically. Performance indicators that measure an essential element of performance are *Critical Performance Indicators* (CPIs).

Ultimately, the performance of the entire MobiHealth system is what matters to the users. There are many possible definitions of performance. The primary objective of the system is to allow professionals at healthcare centers to monitor biosignals from patients who are not physically there, on a near real-time basis. In this thesis, performance is seen as how well the transporting functionality of the system works towards the primary objective (without taking into account security issues for example). From the perspective of the professionals monitoring patients, the following system performance considerations are very important:

1. In general, measurements lose value with time. That is why it is important that measured biosignals from the patient are viewable at the healthcare center as soon as possible, especially in emergency situations.

2. While the general guidelines are that biosignal measurements should be viewable at the healthcare center as soon as possible, some types of biosignals may be more time-critical than others in comparison. For Healthcare Center professionals it is not just important that the whole package of biosignals measured from the patient is viewable as soon as possible, but also that the more time-critical biosignals have priority over less time-critical biosignals. Certainly, in a well performing system, highly time-critical biosignal measurements should not be delayed significantly by measurements that are less time constrained.

3. The transport of biosignal measurements from a remote location to the healthcare center not only affects the time at which the data can be viewed, but it can also affect the quality[13] of the biosignal measurements. For example, by reducing the quality of data, the time that elapses before measured biosignals become viewable at the healthcare center

[12] Strictly speaking, the concept BANip Service Provider encompasses more than just the BAN interconnect protocol. However, in terms of performance the BANip performance is equal to BANip Service Provider performance when viewed from a higher layer.

[13] Quality in this context refers to the integrity and sampling rate of biosignal measurements

also decreases. However, it is important that the quality of a certain biosignal remains at or above a level that is sufficient for interpretation by medical professionals.

These considerations are the guidelines for the assessment of the BANip performance because it is part of the MobiHealth system and plays an important role in transporting biosignals from patient to healthcare center. The guidelines are used to establish the main CPIs for the BANip in the following subsections.

3.1.1 BAN Data Latency

BAN Data Latency (BDL) is the end-to-end delay for a unit of BAN data travelling between MBU and SH through the components of the BANip Service Provider. A unit of BAN data is the smallest amount of BAN data that causes relevant updates on monitoring equipment. Technically that would be every new value for a graph displaying biosignal measurements. Considering the fact that biosignal measurements from different sensors are atomically grouped together in SensorSets (see Chapter 2.5), it makes more sense to use the size of one SensorSet as the size for a unit of BAN data.

The motivation for using BDL in the performance assessment of the BANip stems from consideration 1, which is concerned with the time it takes BAN data[14] to progress from patient to healthcare center(s) through the whole MobiHealth system. This time is the sum of the time BAN data spends in the different parts of the system, so the time BAN data spends in the BANip Service Provider is part of this sum. Figure 3.1 shows the BANip Service Provider modeled as a pipe through which all BAN data must travel. The BDL for a unit of BAN data is equal to the total time the unit spends in this pipe. That is, the time that elapses from the moment the BANip Service Provider accepts a unit of BAN data until the BAN data unit is fully delivered at the SH.

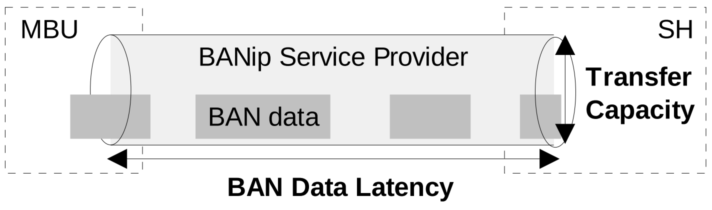


Figure 3.1: BANip transporting BAN data between MBU and SH

The average BDL is used as a CPI. This average is calculated by summing the individual BDL of *n* BAN data units and then dividing the sum by *n*.

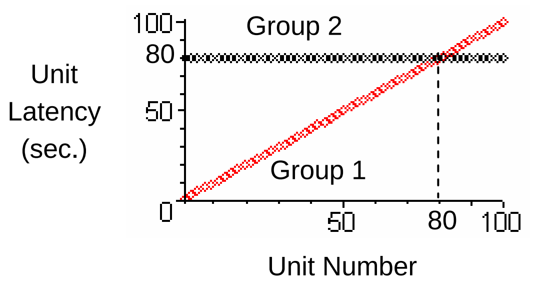
Initially it may seem sufficient to use the average BDL as the only critical performance indicator to represent performance consideration 1. Section 3.1.2 explains why an additional CPI must be defined.

[14] As described in Section 2.5, BAN data consists of biosignals and metadata

3.1.2 Transfer Capacity

Besides BAN Data Latency, consideration 1 requires the definition of an additional CPI. The following example demonstrates the need for the indicator *Transfer Capacity* (TC), which is defined as the maximum amount of BAN data the BANip can accept per second without building up a continuously growing queue, in bytes per second. Implicitly, this is strongly related to the maximum amount of BAN data that the BANip can deliver at the receiver every second.

Assume that there is a set of 100 BAN data units to be transferred. Now suppose that there are two groups who have each designed their own version of the BANip. With the BANip of group 1, it takes 100 seconds before the last unit is transferred. Group 2's BANip only takes 80 seconds to transfer all 100 units. Without looking at the average BDL for both groups, it seems like group 2 has the better performing BANip because they have completed the transfer in the shortest amount of time. There is a catch, however. Figure 3.2 shows the time it took for each unit to be delivered by the two BANip versions.



Displayed functions are:
(unit number ∈ [1,2..100])
Group 1: latency = unit number
Group 2: latency = 80

Figure 3.2: End to end delay of BAN data units

From Figure 3.2 the average BDL can be calculated for both groups. For group 1 this yields: (1 + 100)/2 = 50.5 seconds. The BDL for all the units of group 2 is 80 seconds, so their average BDL is also 80 seconds. This means that according to the BDL CPI, not group 2 but group 1 has the better performing BANip, because - on average - it delivers individual BAN data units almost 30 seconds faster. Which group really has the BANip with better performance?

A review of the designs reveals that group 1's BANip transfers 1 data unit every second, whereas group 2 implemented a compression algorithm so that the same amount of data can be transferred in less time. Even though the compression technique allows group 2 to average 0.25 data unit per second more than group 1, the healthcare center cannot access any of the individual data units until the last one has been transferred, which results in the high 80-second average.

On one hand, group 1 delivers 79% of the BAN data units to the healthcare center with less delay than group 2, which is justly reflected by the average BDL of group 1's BANip. On the other hand, when faced with a continuous stream of 1.20 units per second for a few hours, group 1's BANip would be significantly outperformed by group 2's BANip. After the first hour, group 1's BANip would already have built up a 12-minute queue of data units, whereas group 2's BANip would still be delivering data units within 2 minutes.

For a fair comparison to be possible, the TC of the BANip over a certain amount of time should be measured in addition to the average BDL.

3.1.3 Exact Delivery

Performance considerations 2 and 3 in the introduction of this section can be combined into the issue of Quality of Service (QoS). Biosignals could be assigned a certain QoS class. To cover both performance considerations, such a QoS class could define a value for the allowed quality reduction and a value for the priority level[15] of the biosignal measurements in this class. In the reference BANip, there exists only one QoS class. The BANip is not allowed to change the quality of BAN data, and since there are no other QoS classes, there is no difference in priority level for biosignal measurements. While the introduction of new QoS classes would most likely improve the performance of the BANip in this area, designing, implementing and testing a BANip QoS scheme is outside the scope of this thesis. However, even when there are no improvements in this area, it is still important that improvements in other areas do not cause any quality reduction or deviation from the first-in first-out treatment of BAN data. Because these are not variables, it is not very useful to define CPIs for them. Rather, the BANip containing improvements must continue to meet the design requirement that it conveys BAN data unmodified and in order.

[15] in terms of "which data should be delivered first"

3.2 CPI Measurement Strategy

Transfer Capacity (TC) can be measured by inserting a dummy sensor into the MobiHealth System (see Figure 3.3) that generates a bulk data stream. If the dummy sensor generates more data per second than the BANip can handle, the receiving end observes the BANip TC. With an "overproduction" of BAN data, the BANip is never allowed to "rest", thus forced to operate at its maximum capacity. Overproduction can be guaranteed for example by setting the generation rate beyond the maximum physical capacity of the tight link.

By starting a timer at the receiver as soon as the first BAN data has arrived, and noting from that point on how much BAN data bytes arrives, the TC is measured. The average TC of the BANip is the amount of arrived bytes divided by the elapsed time.

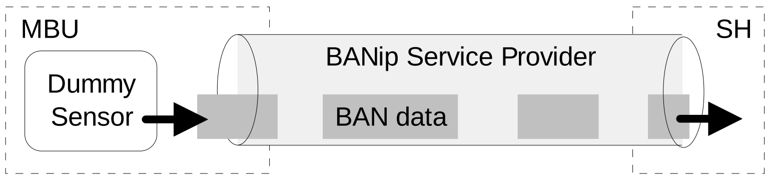


Figure 3.3: Dummy sensor for measuring CPIs

Measuring **BAN Data Latency (BDL)** is a bit more involved. There are two issues that require particular attention: the problem of measuring the time BAN data is underway and the need for rate control. These are discussed below:

1. Latency is often gauged by looking at round-trip time (RTT), because this is an easy and accurate way of measuring latency. The disadvantage of using this method is that both upstream and downstream delay characteristics are merged together. The link asymmetry poses further difficulties in establishing the end-to-end latency of BAN data, because the forward leg of the "trip" may be completed faster than the return leg, or the other way around. The approach taken to avoid these difficulties is to synchronize the clocks of sender and receiver with a time server using the Network Time Protocol (NTP) [RFC1305]. BDL can be measured by inserting timestamps in BAN data units. The BDL of a BAN data unit is calculated as the receiver clock value at the moment the data unit is delivered minus its timestamp value. The average BDL of a group of BAN data units is the sum of the BDL values of these data units divided by the number of data units.

2. BDL will start to increase significantly if the rate at which the stream of test data is generated exceeds the TC of the BANip. Worse, this increase is not so interesting to measure because it can be predicted by comparing the BANip TC and dummy sensor data generation rate. For example, if the BANip TC is 1000 Bps and the dummy sensor generates 2000 Bps, the BDL will increase by 1 second every second. In order to avoid this queuing delay, the dummy sensor should not generate data at a rate that exceeds the average TC. The BDL is measured at two rates: high and medium load conditions. To simulate high load conditions, the rate is set slightly below the average TC. Setting the rate at half the average TC simulates medium load conditions.

To verify **Exact Delivery**, a dummy sensor sends data from a well-known binary file. The receiver (SH) stores all delivered data in a new file. The exact delivery requirement is met if the two files are exactly the same after the transfer is completed.

Lastly, there is an issue for both TC and BDL regarding the setting of the BANip packetizer (see Chapter 2.5). At a lower setting, the packetizer groups less BAN data units together for every transmission. Smaller groups take less time to transfer and the receiver gets faster updates of BAN data, thus resulting in a lower average BDL. However, the BANip transfers smaller groups less efficiently, which results in a lower average TC. The main testing effort is focused on an average packetizer setting which favors neither high TC nor low BDL. Additional tests with a low and a high packetizer setting are conducted to demonstrate the BANip performance with these settings.

3.3 Test Setup

By design, the MobiHealth system can run on a variety of hardware devices. The test setup consists of a selected set of hardware. The most important consideration for this selection is that the test setup should closely represent the environment in which the MobiHealth system is expected to operate. More specifically, the test setup for the BANip Service Provider should not introduce any new or different bottlenecks.

The BANip Service Provider performance tests are conducted in The Netherlands. Figure 3.4 shows the test setup. All tests conducted as part of this thesis utilize this setup.

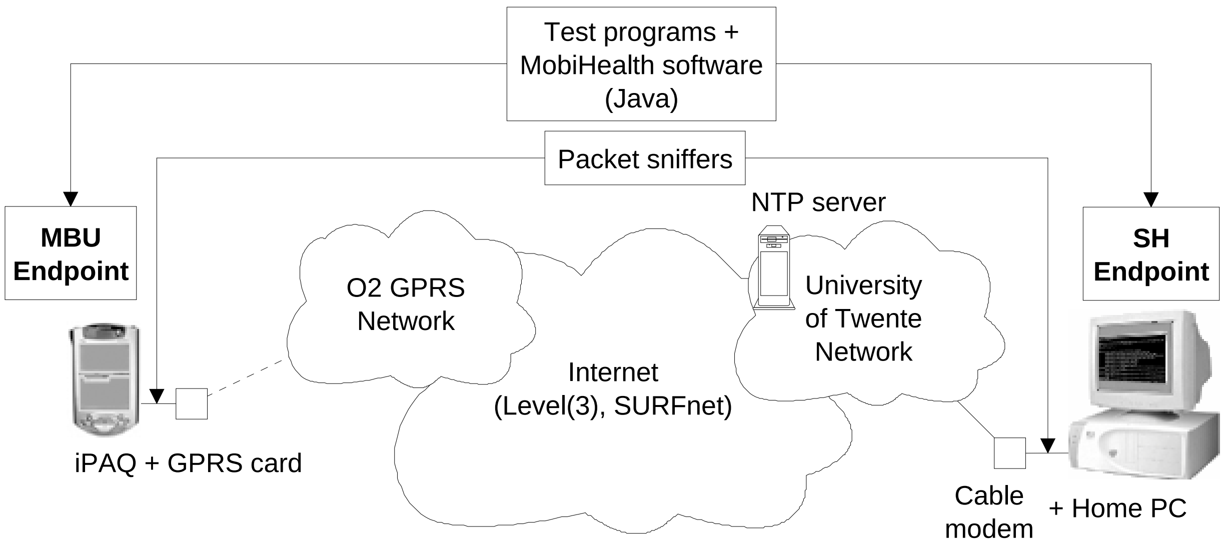


Figure 3.4: General test setup

The following two sections discuss the hardware and software components of the setup.

3.3.1 Hardware Components

An exact specification of the network hardware in the endpoints of the test setup can be found in Appendix A: Test setup specifications. The different parts of the test setup shown in Figure 3.4 are discussed below:

- An iPAQ handheld computer running MobiHealth software takes on the role of Mobile Base Unit (MBU). It is equipped with a GPRS modem for wireless access. The same devices are used in MobiHealth field trials.

- Authorization to use the GPRS services of a particular GPRS operator is gained through a Subscriber Identity Module (SIM) that is placed in the modem. All tests in this thesis that require GPRS make use of a SIM of the telecom company O_2[16].
- An average PC with cable access to the Internet represents the Surrogate Host (SH). Both the power of the PC and the cable network speed are more than sufficient for running the MobiHealth software and supporting an MBU connection.

[16] O_2 (www.telfort.nl) is set to undergo a name change back to Telfort, which it carried previously

- The MBU has Internet access through a gateway (the GGSN) in the O_2 GPRS network, while the SH uses a gateway in the network of the University of Twente.

- Because the Internet is a rather opaque collection of networks, it introduces uncertainties in controlled tests. To provide more clarity, the network tool traceroute is used to examine the route between the MBU and SH.

  The Amsterdam Internet Exchange (AMS-IX) is an important point in The Netherlands where Internet Service Providers (ISPs) exchange Internet traffic. Generally, ISPs route customer traffic to such a point where it can be routed on to peering (adjacent) networks. With the traceroute tool it was verified that traffic between the MBU and SH indeed passes the AMS-IX. This means that the route from MBU to AMS-IX is fairly independent from the BackEnd ISP or placement of the SH, but inevitably tied with the choice of O_2 as the wireless Internet gateway. Therefore, as long as O_2 is a reference for the operational environment of the MobiHealth system, only the route from the AMS-IX to the custom SH used for tests in this thesis differs slightly from the route used in the reference MobiHealth system.

  On the path between the AMS-IX and SH are high-speed networks registered to SURFnet and Level(3) Communications. The gigabit routers in these networks are not expected to pose any lack of network capacity problems at all.

- As mentioned in Section 3.2, the clocks of the MBU and SH are synchronized to a time server. An NTP server on the network of the University of Twente was the time server of choice. The level of synchronization between the MBU and SH clocks can be determined in two ways:

  1. After an NTP update, the NTP client reports how much the clock was adjusted. When both clocks are consistently adjusted by less than 10 milliseconds forwards or backwards, it is highly likely that the clocks are within 20 milliseconds of each other.

  2. The second method for determining the level of synchronization between the two clocks involves sending a time-stamped packet both ways between the two hosts. The first host sends a packet with timestamp T_1. The receiving host adds its own timestamp T_2 to the packet and returns the total back to the first host. The first hosts receives the packet at time point T_1 + RoundTripTime. Assuming the two clocks are fully synchronized, T_2 must fall somewhere within the interval $< T_1, T_1 +$ RoundTripTime $>$. Thus, the deviation between the two clocks can be determined with a maximum error of RoundTripTime. Attaching the iPAQ to the USB port of the PC yields a RoundTripTime of less than 10ms, so the deviation between the clocks can be determined fairly accurately.

  Either method can be used when appropriate. Various tests have different accuracy requirements, but the difference between the clock values was kept below 50 ms[17] unless specified otherwise.

[17] This includes the clock skew caused by drifting clocks. The drift of the MBU clock relative to the SH clock was determined to be less than 3 ms per minute.

- [O2LSpec] gives an indication of latency characteristics of the O_2 GPRS network. The tables below illustrate a breakdown of the round-trip latency associated with the transmission and reception of a **500 byte IP packet** in a system employing **1 uplink and 2 downlink timeslots**.

Latency if a Temporary Block Flow[18] (TBF) has not yet been established:

Latency Element	**Uplink Latency (1 timeslot)**	**Downlink Latency (2 timeslots)**
MS (MBU) Delay	215 ms	65 ms
TBF Establishment avg.	530 ms	1000 ms
variability	320 - 750 ms	290 - 1700 ms
Over the Air Delay	480 ms	260 ms
SGSN/GGSN Latency	20 ms	20 ms
Total (average)	**1.3 second**	**1.3 second**

The cells marked with a gray background are latencies that are the same with or without TBF establishment. Once a TBF is established, the average latency decreases:

Latency Element	**Uplink Latency (1 timeslot)**	**Downlink Latency (2 timeslots)**
MS (MBU) Delay	110 ms	65 ms
Over the Air Delay	480 ms	260 ms
SGSN/GGSN Latency	20 ms	20 ms
Total (average)	**0.6 second**	**0.4 second**

Note that TBF establishments result in a quite formidable additional delay of around 0.7 sec.

3.3.2 Software Components

Apart from hardware, there are also software components in the test setup displayed in Figure 3.4:

- The MobiHealth software. In the timeframe of this thesis the MobiHealth software was in the process of undergoing the transformation from prototype to full implementation, so it was necessary to "freeze" a certain version of the software for testing and comparison. Only essential bug fixes were still applied before the final test run.

- In the in-depth BANip Service Provider analysis of Chapter 4, Java test programs are also used in addition to, or instead of the MobiHealth software. They are further discussed and specified there.

- IP packets at both endpoints are recorded with packet sniffers to keep an exact log of "what went across the wire".

Appendix A: Test setup specifications contains a detailed specification of the software components used during the tests.

[18] A host can only request a TBF for data it has pending for transmission. It cannot start transmitting until the TBF is established, and the TBF only lasts for as long as the host keeps transmitting.

3.4 Reference Interconnect Protocol Performance

3.4.1 Transfer Capacity

The first CPI to be measured is the Transfer Capacity (TC). The requirement for TC tests is that the transmission rate of the dummy sensor exceeds the TC of the BANip configuration during the entire test. A few probing tests with the dummy sensor transmitting at increasing rates established that the average TC would probably not exceed 1500 Bps, so a "safe" dummy sensor transmission rate of double that (3000 Bps) was chosen. This is a rate that is also close to the maximum physical capacity of the tight link (dual slot CS-2 = 3350 Bps). A series of **six tests** are executed to measure TC with an average packetizer setting of groups of **140** BAN data units (groups of 140 * 30 = 4200 Bytes). Each test transfers between 90.000 and 100.000 Bytes from MBU to SH. Two additional tests measure the TC with packetizer settings of **16** (480 B) and **250** (7500 B) units per group, i.e. near the minimum and maximum setting. The table below summarizes the TC test results.

Test Number	**Transfer Capacity (Bytes per second)**		
	PKT = 140	**PKT = 16**	**PKT = 250**
1	700	172	1002
2	722	(not tested)	1112
3	730		(not tested)
4	727		
5	719		
6	720		
Test average	**720 Bps**	**170 Bps**	**1060 Bps**

3.4.2 BAN Data Latency

According to the measurement strategy in Section 3.2, the BDL CPI is measured at high and medium load. Considering the TC at the packetizer setting of 140 units per group, high load is defined as **650 Bps** and medium load as **350 Bps**. For the packetizer setting of 250, high load is **1000 Bps** and medium load **500 Bps**. The TC with a packetizer setting of 16 is so low that only a high load of **150 Bps** is tested. The following table summarizes these settings.

Packetizer setting	**High load (Bps)**	**Medium load (Bps)**
16	150	(not tested)
140	650	350
250	1000	500

Before the BDL test results are presented, a testing problem is discussed that falsifies test results if left unchecked. For the test it is necessary that time-stamped BAN data units are offered to the BANip Service Provider at a certain fixed rate. The dummy sensor used for this purpose runs, like the BANip Service Provider, on the iPAQ. During test runs it was established that the iPAQ MobiHealth software experiences system wide temporary blockades. These are most likely caused by the GPRS hardware, as the blockades consistently occur right after a group of BAN data units is queued for transmission. Figure 3.5 shows how the blockades affect the timestamps of the generated BAN data units.

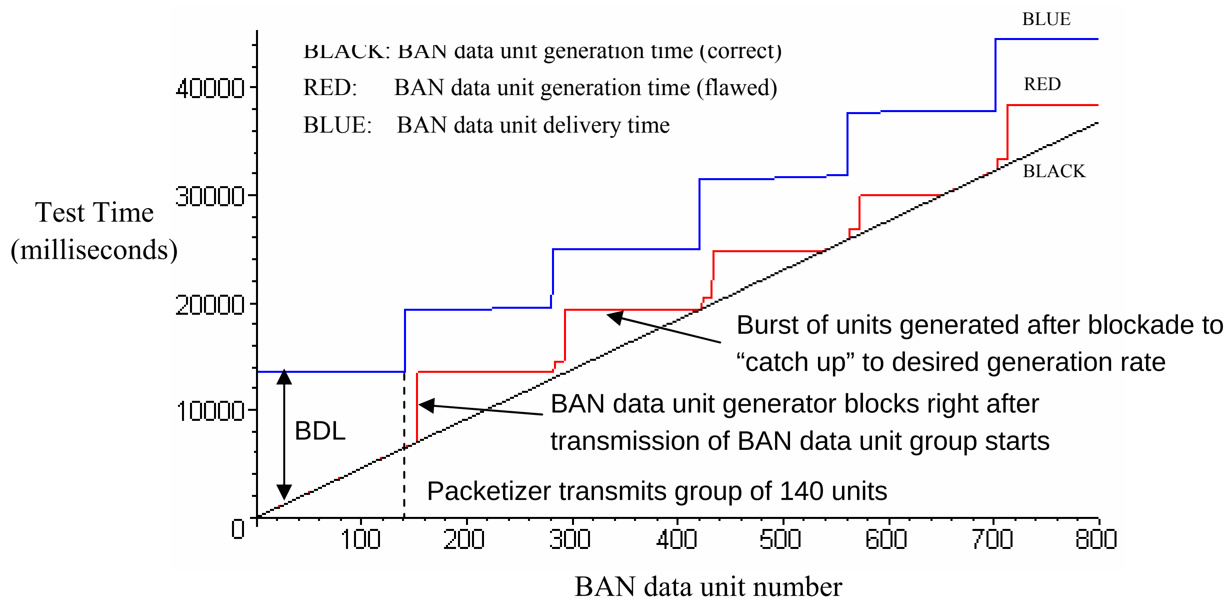


Figure 3.5: BDL testing problem

The horizontal difference in the generation time and delivery time graphs of Figure 3.5 is the BDL. The red graph represents the timestamps that were put in the BAN data units at the time they were generated and clearly shows the irregularities caused by the blockades. A fixed generation rate should result in a straight (diagonal) line. The deviation causes the BDL to be perceived much lower than it really is. The following analogy explains the difference between the red and black graph. Suppose that the BANip Service Provider is a post office. Customers regularly deposit letters (BAN data) in its mailbox. Even when the mailbox is not emptied right away (e.g. because the BANip is still busy transmitting other data), the time a letter spends in the mailbox should count towards the total servicing time of the letter. The red graph is an example of time-stamping letters at the moment the mailbox is emptied, whereas in the black graph, letters are properly time-stamped at the moment they enter the mailbox.

The solution to the blocking problem is as follows. A file with time-stamped BAN data units is generated beforehand. The timestamps are calculated so that every second contains a fixed amount of BAN data units. If the desired rate were 60 Bytes per second for example, assuming a BAN data unit is 30 Bytes, the sequence of the first five timestamps would be 0, 500, 1000, 1500, 2000 (milliseconds). The file is read in a rate controlled way by the dummy sensor so that it can never give the BANip Service Provider units that "are not supposed to be in the mailbox yet", but can only play catch-up after a blockade. This solution of simulating timestamps is very effective, as demonstrated by the black graph in Figure 3.5. It precisely overlaps the red graph, which contains real-time generated timestamps, in places that are not affected by blockades (such as the first 140 units).

Figure 3.6 shows a graph containing each calculated BDL of a set of 2500 BAN data units. It is the graph that results from subtracting the black graph from the blue graph (Figure 3.5).

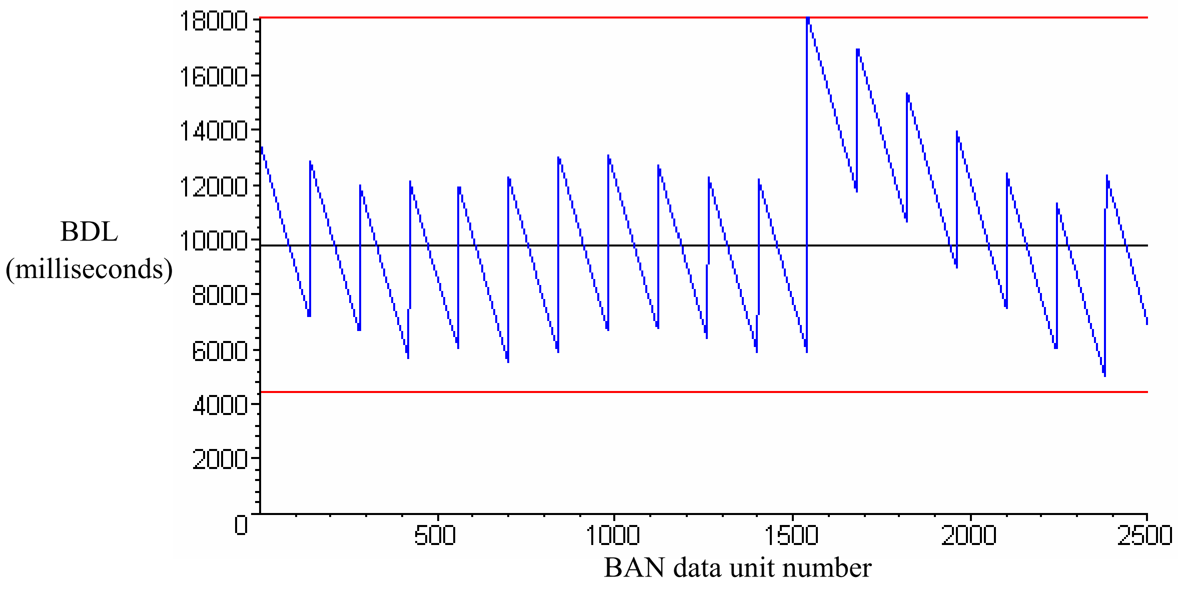


Figure 3.6: Example BDL graph

Figure 3.6 also shows horizontal lines of the maximum, average and minimum BDL of the graph. The sawtooth pattern is a logical result of the packetizer. The top of the "up leg" is always the BDL of the first unit of the group, while the bottom of the "down leg" is the BDL of the last unit of the group. The large jump in BDL around unit 1500 is the result of a temporary reduction in capacity that could have been caused by, for example, retransmissions after a burst of errors in the wireless transfer. After the jump the BDL decreases slowly as the BANip uses extra capacity[19] to "catch up" on the small queue of units caused by the jump.

The table on the next page summarizes the averages collected from 15 BDL tests, as well as maximum and minimum values measured in each test. Note that the testing effort focuses on the packetizer setting of 140. The more "extreme" packetizer settings of 16 and 250 are only briefly tested to show BDL behavior of the BANip Service Provider in these cases.

[19] The BANip itself is not rate limited, only the dummy sensor that generates data for the BANip to transfer

Test #	**BAN Data Latency: average (max. / min.) seconds**				
	PKT = 140		**PKT = 16**	**PKT = 250**	
	650 Bps	**350 Bps**	**150 Bps**	**1000 Bps**	**500 Bps**
1	8.8 (13.2 / 3.8)	12.3 (19.6 / 5.5)	5.5 (9.5 / 3.2)	11.6 (16.9 / 5.6)	16.8 (31.6 / 7.1)
2	8.6 (13.4 / 3.8)	12.6 (21.9 / 5.6)	(not tested)	(not tested)	(not tested)
3	10.2 (17.0 / 4.4)	12.6 (19.7 / 6.0)			
4	9.1 (14.2 / 4.3)	12.7 (23.0 / 6.0)			
5	9.2 (13.8 / 4.3)	12.5 (19.8 / 4.7)			
6	9.8 (18.1 / 4.4)	11.7 (18.8 / 4.5)			
Test average	**9.5 (15.0 / 4.0)**	**12.5 (20.5 / 5.5)**	**5.5 (9.5 / 3.0)**	**11.5 (17.0 / 5.5)**	**17.0 (31.5 / 7.0)**

N.B. The test averages were rounded to the nearest 0.5 sec. to keep the accuracy in line with the scale and significance of BDL values.

3.4.3 Summary and Discussion

Figure 3.7 summarizes the BAN Data Latency and Transfer Capacity test results in a more intuitive way.

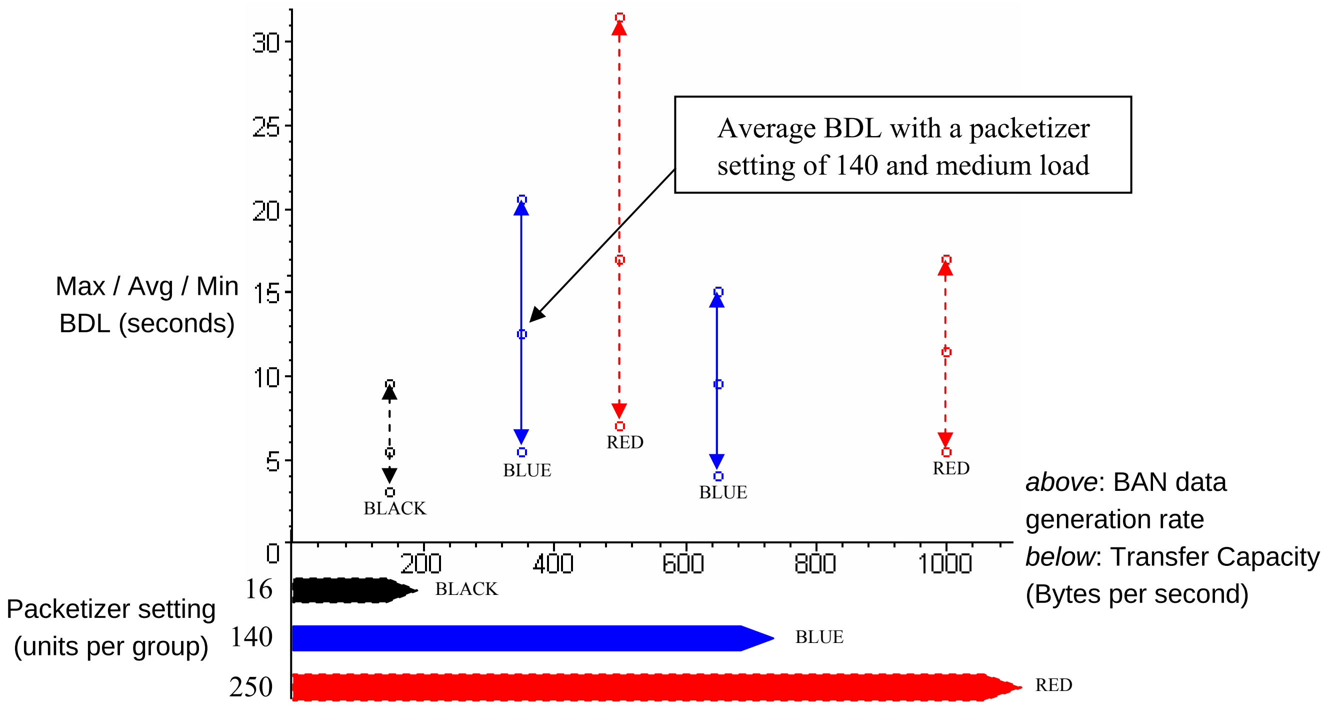


Figure 3.7: Performance test results

A (two-sided) vertical arrow represents the maximum, average and minimum measured BDL for a certain packetizer and BAN data generation rate setting. Different colors indicate a different packetizer setting. The red and black arrows are dotted because they are not backed up with as many tests as the blue arrows. The horizontal block arrows each represent the TC for a certain packetizer setting. Vertical arrows of the same color cannot indicate stable BDL values if they are outside the horizontal range of the block arrow of the same color, because the BDL will grow to infinity if the TC is exceeded.

There appear to be various trends in the test results displayed in Figure 3.7. The most obvious trend is that the BANip has a higher Transfer Capacity when the packetizer is set to group more data units together. Opposite of this rising TC performance trend are the BDL performance trends. The amount of time the average BAN data unit lags behind “reality” only increases in the measurements with higher packetizer settings. As long as the load on the BANip is high, BDL performance deteriorates relatively slowly. The leftmost red and blue arrow show a significant deterioration in BDL performance as the load decreases.

The general conclusion from Figure 3.7 is that there is a tradeoff between TC and BDL performance of the BANip. The best combined performance seems to occur when the packetizer is set so that the BANip has just enough capacity to meet the most recent load requirements.
The second conclusion is that even with the packetizer setting that favors Transfer Capacity the most, the BANip only achieves a throughput of about 1060 bytes per second. Compared to the fact that the throughput limit of the utilized GPRS uplink is close to 2 slot CS-2, which is 1000 * 2 * 13.40 / 8 = 3350 bytes per second, it is clear that the BANip fails to efficiently utilize even one third of the maximum throughput.

This concludes the chapter that deals with defining and quantifying the performance of the reference BANip Service Provider. The next chapter contains an in-depth analysis of the BANip Service Provider. The analysis examines the contributions of different elements in the BANip Service Provider to the test results summarized in this section.

4. Interconnect Protocol Element Analysis

This chapter presents the analysis of elements of the interconnect protocol. As explained in Section 1.2, the BANip is an application protocol that relies on several other protocols and services to function. All of these components affect each other's performance. Therefore, the analysis of the performance of the BANip should also include an analysis of underlying components.

Section 4.1 defines the scope of the analysis, while Section 4.2 describes the general approach for analyzing elements. The results of the actual analysis are presented in Section 4.3 through 4.6, one section for each layer. Finally, important results and conclusions are summarized in Section 4.7. It contains an answer to the research question: "Which significant negative influences on the CPIs (i.e. performance drawbacks) can be identified in the reference BAN interconnect protocol stack and its elements?".

4.1 Scope

The BANip Service Provider shown in Figure 4.1 contains the protocol elements that are present in the reference BANip protocol stack.

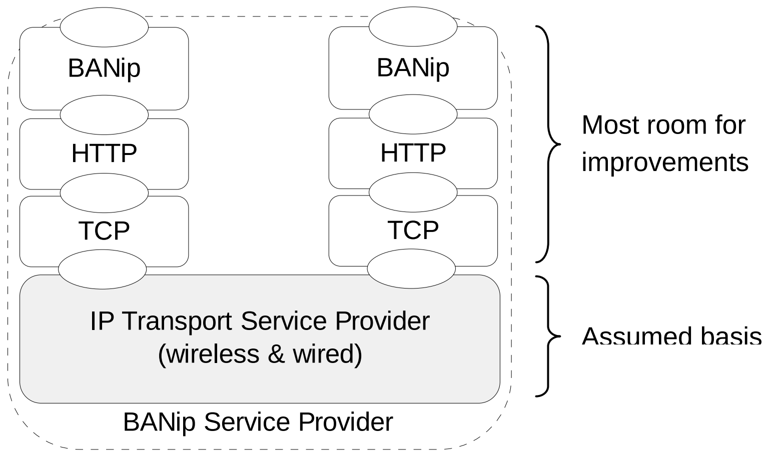
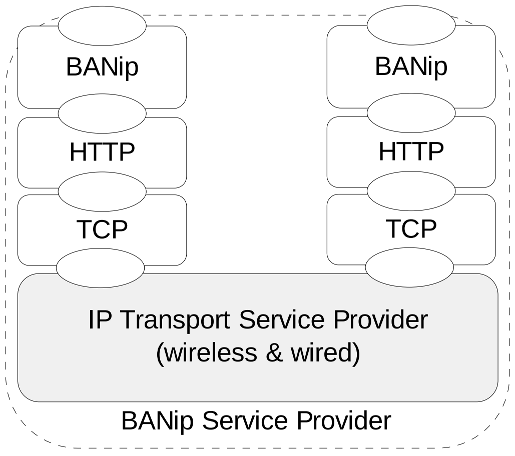

BANip
BANip
HTTP
HTTP
TCP
TCP
IP Transport Service Provider (wireless & wired)
BANip Service Provider
Most room for improvements
Assumed basis

Figure 4.1: Scope of BANip analysis

Improving the BANip performance by making modifications to the protocols operating at and below the IP layer is beyond the scope of this thesis. They are considered to be part of an IP Transport Service that falls under the responsibility of Internet Service Providers (ISPs) and Internet Backbone Providers. As Figure 4.1 shows, the protocols above the IP layer offer the most room for improvements.

4.2 Approach

The BAN Data Latency (BDL) and Transfer Capacity (TC) CPIs defined in the previous chapter enable the assessment the BANip Service Provider as a whole. Figure 4.2 shows that the measured values of the CPIs are composed of contributions of individual BANip Service Provider elements.



BAN Data Latency
- each element adds to the BDL of a unit of BAN data travelling through

Transfer Capacity
- capacity of each element is limited by lower level elements

Figure 4.2: Elements affecting the CPIs

As a BAN data unit proceeds through various elements, each element adds to that unit's BDL. In terms of Transfer Capacity, the performance of elements is limited by that of lower level elements. For example, if the wireless part of the IP Transport Service Provider only allows a flow of up to 1000 bytes per second, the TCP element would not be able to transmit at a higher rate, which limits the amount of payload the TCP element can process every second without a queue buildup.

The general approach for identifying performance drawbacks in the BANip Service Provider elements is to:

- analyze how each element contributes to the CPI values,
- evaluate whether this contribution differs significantly from what can reasonably be expected in the test setup and if so,
- identify performance drawbacks responsible for the difference.

Sections 4.3 through 4.6 present the element analysis in a bottom-up fashion. Elements on higher layers may exhibit performance drawbacks that stem from performance drawbacks in lower layers. The bottom-up approach increases the chance that the cause of performance drawbacks is identified as closely as possible.

4.3 IP Transport Service Performance Drawbacks

The IP Transport Service carries BAN data from the MBU to the SH over various networks and computer systems. The speed at which IP packets can move through the IP Transport Service as well as the latency characteristics are largely determined by these networks and computer systems. Their influence is analyzed through a model that is discussed below.

A network path between two hosts is a sequence of network links. Each link has a certain capacity, the maximum rate at which data can be transmitted over the link. Figure 4.3 illustrates the fact that the capacity of the network path is limited to the capacity of the link with the least capacity, which [Jain02] defines as the *narrow* link.

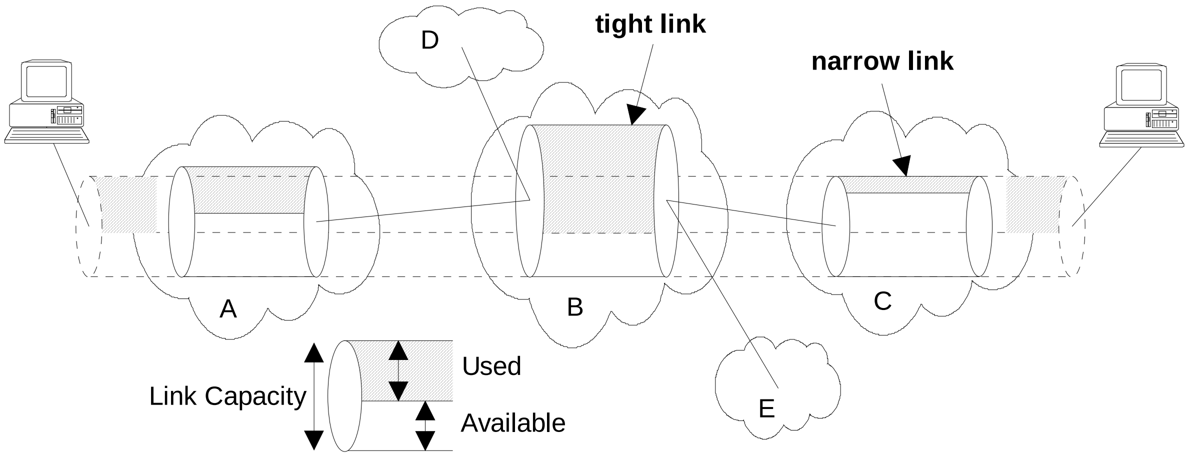


Figure 4.3: Narrow and tight link in a network path

Each link processes data up to its capacity and may be used by many different sources. In Figure 4.3 the link in network B processes data travelling between A and C as well as D and E. It has a greater capacity than the link in network C, but more of that capacity is being used. The amount of capacity that is still available on the link in network B in this example is less than the available capacity of any of the other links in the network path. [Jain02] defines such links to be *tight* links. Note that available capacity is the additional amount of link bandwidth that can be used without reducing the rate of existing traffic flows.

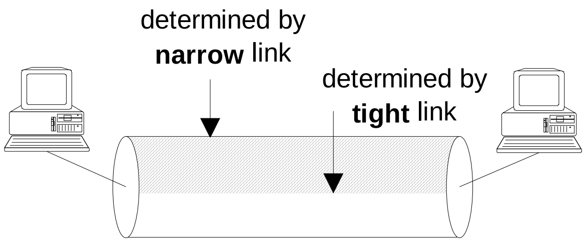


Figure 4.4: End-to-end abstraction

Figure 4.4 shows the abstraction that is possible with the concepts of narrow and tight link. The narrow link on a network path presents a relatively static upper bound to the capacity of the network path, the speed at which IP packets can be transferred over the path. The tight link determines how much of this capacity is available. Transport protocols that operate between two hosts can be congestion-aware or congestion-unaware. The end-to-end abstraction translates to different capacity limitations for each of these two types:

1. For congestion-aware transport protocols such as TCP, available capacity is an approximate upper boundary.

2. Congestion-unaware protocols such as UDP can artificially increase the amount of available capacity they can use. By trying to use more than the available capacity, congestion-aware protocols that use the same tight link are likely to reduce their transfer rate. Thus, the upper boundary for congestion-unaware protocols is not necessarily the available capacity, but rather somewhere between the available capacity and narrow link capacity in the end-to-end abstraction.

Figure 4.5 shows that in the setup used by the MobiHealth system, the narrow and tight link are assumed to be equal to the wireless (GPRS) link. The motivation behind this assumption is discussed below.

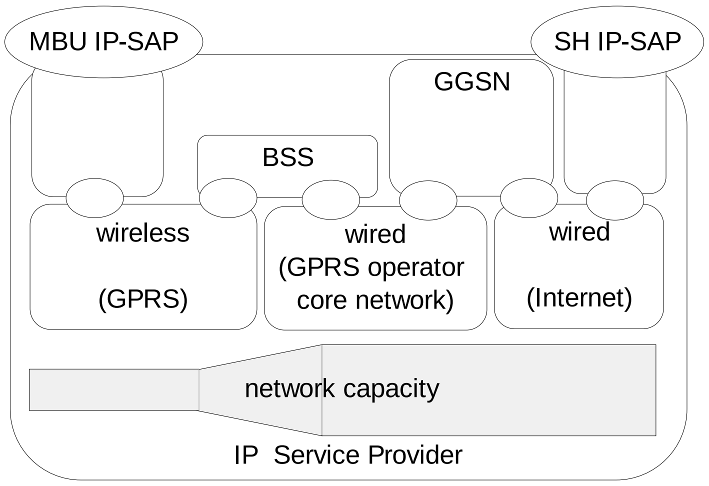


Figure 4.5: Bottleneck link identified (narrow and tight)[20]

The absolute maximum transfer rate of the GPRS link is limited by the particular coding scheme being used. The coding scheme with the fastest associated transfer rate (CS-4) and current maximum amount of assigned slots (4) yields a mere 4*21.40 = 85.6 kbit/s and is easily outclassed by a basic 10 Mbit/s wired link in the area of speed, reliability and latency. The wireless link is therefore definitely the narrow link.

Because of the relatively low "cost for speed" of wired network links (compared to the cost of using wireless links), it is assumed that the GPRS operator core network will easily be able to handle the traffic generated by its users even at peak usage times. In turn, the Internet path leads through core routers, which should not be very impressed with the maximum of 21.40 kbit/s per MBU either. The GPRS link is therefore also deemed the tight link[21].

In terms of latency, the GPRS link is also a major factor. As Section 3.3.1 shows, the GPRS link introduces round-trip times of - on average - at least 600+400 = 1000 ms for near-MTU

[20] The SGSN is omitted; it resides within the wired GPRS operator core network

[21] In the test setup presented in Chapter 3, Section 3.3, it is obvious that the wireless link is indeed also the tight link

size packets. The network path from the SH to somewhere close to the GGSN can be measured using ICMP Echo packets of the same size as the IP packets in the table in Section 3.3.1 (500 Bytes). The result was a median[22] of less than 50 ms (round-trip time), which is less than 5% of the minimum round-trip time of the GPRS link.

According to this model, the majority of the contribution of this layer to the Transfer Capacity and BAN Data Latency CPIs comes from the characteristics of the wireless link. As long as the following two assumptions hold, it is safe to focus the analysis of performance drawbacks in this layer on the GPRS link:

1. The rest of the network path is healthy[23] and does not pose any additional capacity limits.

2. The rest of the links introduce a relatively small amount of additional latency (<50 ms round-trip).

Section 4.3.1 presents the theoretical analysis of the GPRS link. The actual performance of the link in the reference setup is measured thereafter. The performance measurement strategy is presented in Section 4.3.2, while Section 4.3.3 presents and discusses the results of the measurements.

[22] To prevent unreasonable skewing of the round-trip time estimate by the sometimes very low priority treatment of ICMP packets, the 50th percentile (also known as median) was calculated

[23] Healthy Internet paths should operate at loss rates below 1% ([RFC2680] pg. 12)

4.3.1 GPRS Performance Drawbacks

Each of the layers of the GPRS stack performs functions that may transform or mitigate performance drawbacks from other layers, or introduce new performance drawbacks. Figure 4.6 shows a schematic of conditions and mechanisms on each layer from bottom to top related to important performance drawbacks. Resulting new or changed performance drawbacks are displayed on the right. The Figure is discussed below.

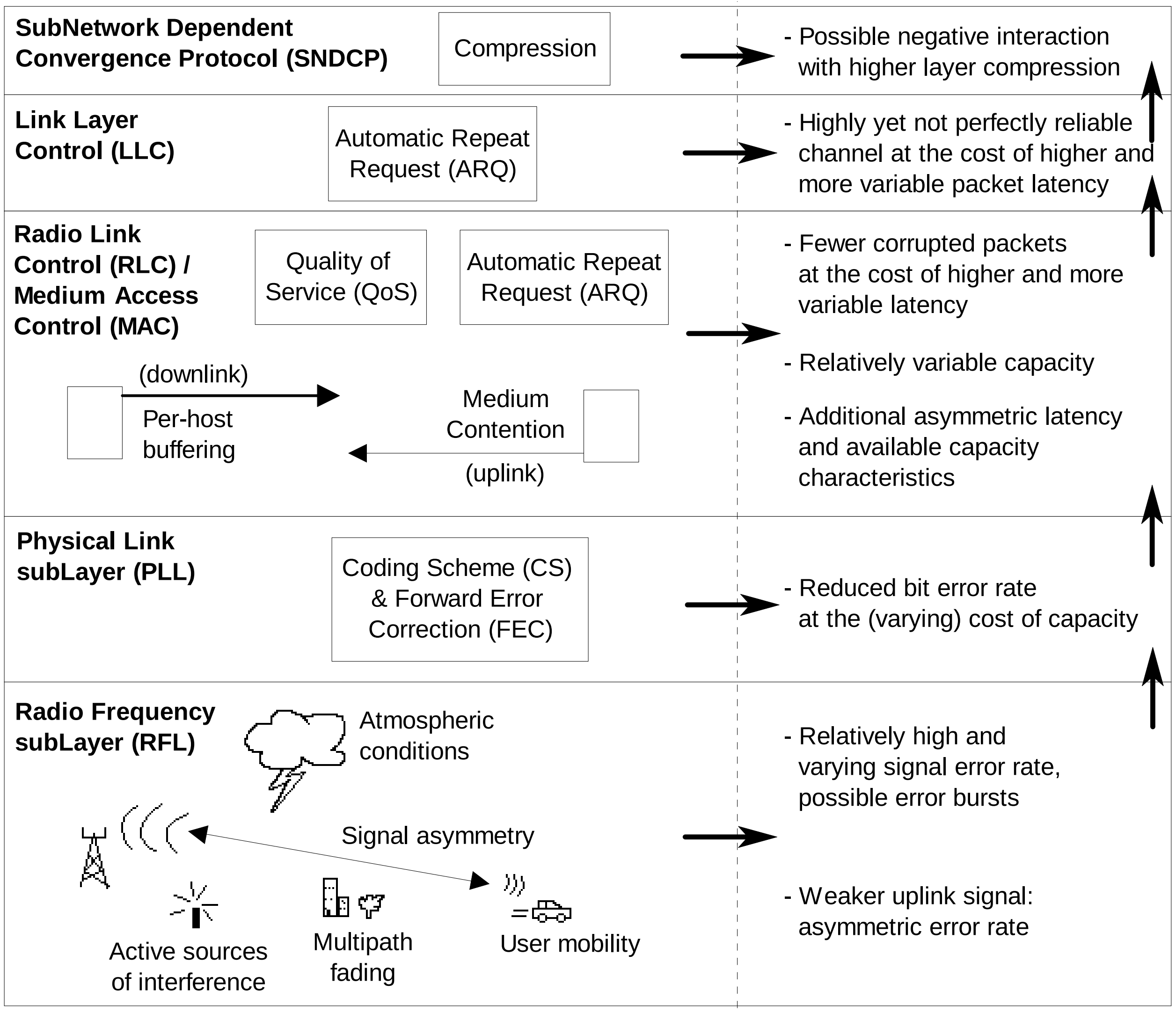

SubNetwork Dependent Convergence Protocol (SNDCP)
Compression
- Possible negative interaction with higher layer compression
Link Layer Control (LLC)
Automatic Repeat Request (ARQ)
- Highly yet not perfectly reliable channel at the cost of higher and more variable packet latency
Radio Link Control (RLC) / Medium Access Control (MAC)
Quality of Service (QoS)
Automatic Repeat Request (ARQ)
- Fewer corrupted packets at the cost of higher and more variable latency
- Relatively variable capacity
(downlink)
Per-host buffering
Medium Contention
(uplink)
- Additional asymmetric latency and available capacity characteristics
Physical Link subLayer (PLL)
Coding Scheme (CS) & Forward Error Correction (FEC)
- Reduced bit error rate at the (varying) cost of capacity
Radio Frequency subLayer (RFL)
Atmospheric conditions
Signal asymmetry
Active sources of interference
Multipath fading
User mobility
- Relatively high and varying signal error rate, possible error bursts
- Weaker uplink signal: asymmetric error rate

Figure 4.6: Bottom-up schematic of GPRS performance drawbacks

Radio signals generated in the lowest layer, the RFL, travel through the air. These signals are affected by a number of environmental and external factors, as Figure 4.6 shows. As a result, the error rate of signals travelling through the air is relatively high compared to signals travelling through cables. The error rate also varies because the environment is very dynamic due to objects moving, atmospheric conditions changing, etc. Furthermore, the Mobile Station (MS) usually does not have same amount of transmitting power as the Base Transceiver Station (BTS), which leads to an asymmetric error rate.

This layer is capable of transmitting bits at approximately 260 kbit/s[24] over a GSM channel, of which about 200 kbit/s[25] are for payload from higher layers. Figure 4.7 shows these limitations in relationship with higher layer channel capacity limitations.

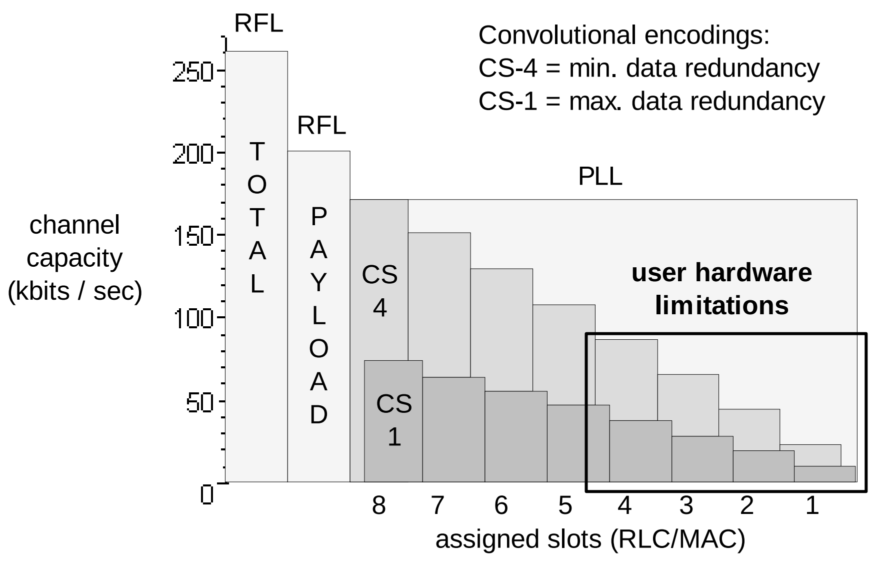


Figure 4.7: Channel capacity limitations

On the PLL, the high error rate is countered by adding redundant data, which also results in a reduced channel capacity. When using the coding scheme that adds the least amount of redundant data, CS-4, a continuous rate of 8 timeslots * 21.40 kbit/s = 171.20 kbit/s of payload from higher layers can be transmitted. With CS-1, this rate is further reduced to 8 timeslots * 9.05 kbit/s = 72.40 kbit/s.

Both calculations assume that on the RLC/MAC layer, a single user can use a full channel (8 slots per TDMA frame). In practice, the current GPRS devices only support using a total of 5 slots per TDMA frame, of which no more than 4 can be used in the same direction (uplink / downlink). Figure 4.8 shows a zoomed version of the area in Figure 4.7 that users are limited to by GPRS hardware.

In the particular setup used during this project, CS-1 or CS-2 is used on the GPRS operator network. Figure 4.8 also shows that in combination with a GPRS card that has a two-slot uplink capability, this results in an expected uplink capacity of around 18.10 – 26.80 kbit/s. With the capability of four downlink slots and the fact that only a fraction of the total traffic between MBU and SH travels this way, the uplink capacity represents the biggest performance drawback.

[24] 1 GSM slot lasts 577 microseconds and consists of 148 bits transmitted + 8.25 microseconds guard time, which means 148 * 10^{-3} / (577 – 8.25) * 10^{-6} ~ 260 kbit/s

[25] 114 out of 148 bits are for payload, so 114 * 260 / 148 ~ 200 kbit/s

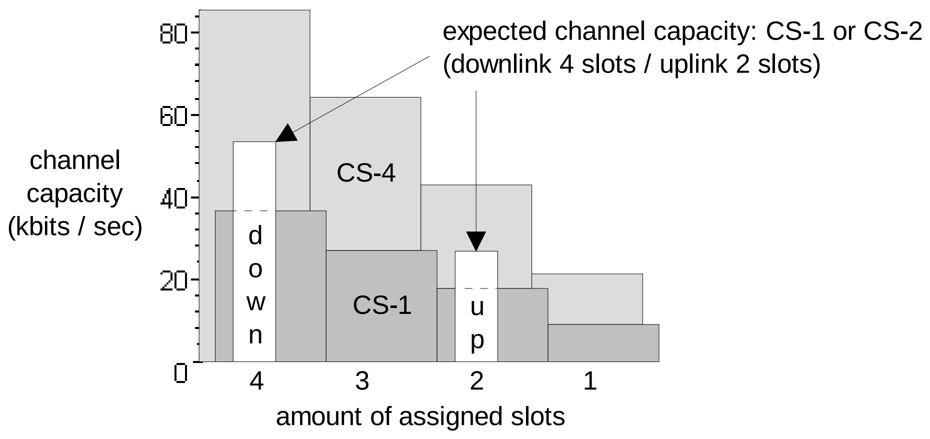


Figure 4.8: Expected capacity within practical limitations

Besides the limits that the GPRS hardware capabilities and coding schemes impose on the capacity of the RLC/MAC layer, there is another important limit. Once the cell a user is located in approaches the limit of its capacity in terms of available channels and slots, users will start to lose slot allocations in accordance with the Quality of Service class they have negotiated. The model introduced in the beginning of this Section offers a way to model this extra limitation. Figure 4.9 does this for the uplink channel.

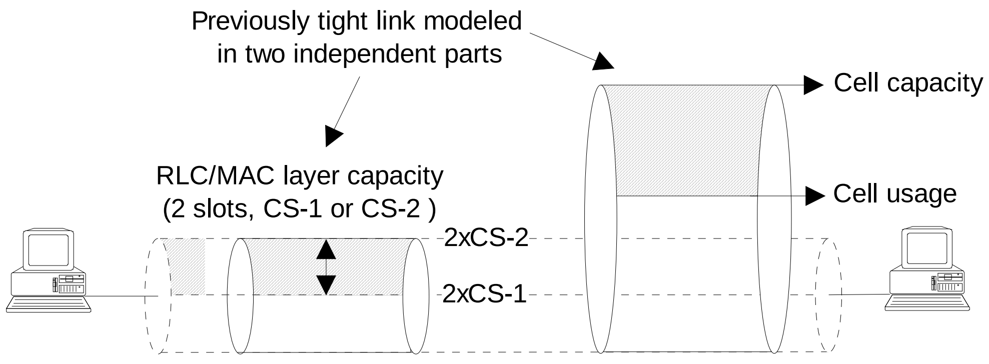


Figure 4.9: Uplink channel capacity model

The left part is the “tight link” in every case where the available capacity (uplink spectrum) of the cell exceeds 2 slots[26]. As long as this situation prevails, the available capacity is equal to the layer capacity because no other Mobile Stations can use the assigned slots. When cell usage approaches cell capacity, higher layers (LLC and up) are confronted with sharp capacity changes. Layer capacity may be halved or doubled (or even zeroed) within a few milliseconds depending on whether slots are granted this time around or not. Cell handovers may also cause these sharp changes. Different cells have different capacity and usage characteristics.

[26] Presumably, the majority of the cells of any reasonable GSM/GPRS operator should not block any GSM calls and therefore remain safely away from such “close calls” as “just 2 slots of capacity left”, so the left part is likely to be the tight link in most cases

The asymmetric properties of the wireless connection are increased by the fact that slot assignments on the uplink are unrelated to slot assignments on the downlink. This is an efficient mechanism, but it does confront higher layers with more asymmetry. In much the same way, the difference in medium access between the MS and BTS is necessary but introduces independently varying latency and capacity characteristics.

The Automatic Repeat reQuest (ARQ) mechanisms in the RLC/MAC and LLC layer trade latency for high link reliability. This increases both the average latency and the latency variance because the bit error rate on the channel is not constant. Also, because the LLC layer does not guarantee that corrupted packets are always retransferred, and because it is the uppermost GPRS layer that corrects frame errors, the wireless link can introduce packet loss that is not related to congestion[27].

The last mechanism depicted in Figure 4.6 to be discussed is the optional compression done on the SNDCP layer. Since compression is costly in terms of required processing power and thus energy, implementations at this level might purposely be tweaked to a lower compression performance. On top of that, this layer has (and should have) no idea of what type of payload is being offered by higher layers so that a general type of compression is most likely being applied. SNDCP compression could conceivably even increase the amount of data to be transferred when higher layers offer carefully compressed payload. Care should be taken if SNDCP compression is enabled.

This concludes the theoretical analysis of the contribution of GPRS layers to the CPI values in the form of capacity and latency. The combined effects of performance drawbacks in these layers on the IP layer are summarized in the table below.

Accumulated effects on IP layer (compared to wired links)
High average latency (O_2 : uplink 600 - 1300 ms, downlink 400 - 1300 ms)
High latency variance (O_2 : uplink TBF adds 320 - 750 ms, downlink TBF adds 290 - 1700 ms)
Asymmetric latency characteristics
Asymmetric capacity characteristics
Possible non-congestion-related losses
Possible clustered losses
Possible negative compression effects
Possible sharp capacity changes, but stable available capacity on the RLC/MAC layer most of the time
Uplink has the worst capacity and latency characteristics

[27] Because the Internet is based around wired links which are extremely reliable, Internet protocols commonly regard packet loss as an indication of congestion

4.3.2 Performance Measurement Strategy

The performance of the IP layer is measured in order to supplement the theoretical analysis with practical material for reference and verification. This subsection presents the measurement strategy used to gather this material. Note that the objective of the performance measurement is not to provide an in-depth IP layer performance characterization, but rather to measure the general behavior so that possible major performance drawbacks can be identified. Also, the main measurement effort is focused on the uplink because it was profiled as the main performance bottleneck in the previous subsection.

The IP Performance Metrics Working Group[28] (IPPM WG) of the Internet Engineering Task Force (IETF) has developed a set of standard metrics that can be applied to the quality, performance, and reliability of Internet data delivery services. The metrics are listed below. Note that they use terminology defined by a general framework [RFC2330].

Metric	**What it determines**	**Reference(s)**
(1) Connectivity	Whether pairs of hosts (IP addresses) can exchange packets	[RFC2678]
(2) One-way delay (3) One-way loss	Time it takes an IP packet to travel from source to destination and loss of IP packets during their journey	[RFC2679] [RFC2680]
(4) Round-trip delay (5) Round-trip loss	Same as above, only now the characteristics of the return path are captured by the metrics	[RFC2681]
(6) Delay variation	Often indicated with the (now deprecated) term ”jitter”: how the end to end delay of individual packets varies	[RFC3393]
(7) Loss patterns	The patterns in which losses occur, e.g. bursts	[RFC3357]
(8) Packet reordering	Packet Reordering Metric for IPPM (draft)	[RFCPRO]
(9) Bulk transport capacity	Bulk Transport Capacity (BTC) is a measure of a network's ability to transfer significant quantities of data with a single congestion-aware transport connection (e.g., TCP)	[RFC3148]
(10) Link bandwidth capacity	The total amount of traffic a link can handle	[NTOOL]

A subset of relevant metrics was used as a reference for the IP layer performance measurement. The selection is discussed below:

- **Packet Latency (delay)**

In terms of latency, round-trip delay (4) is easier to measure than one-way delay (2). However, it does not allow for any distinction between uplink and downlink latency characteristics. To measure just the uplink characteristics, (2) must be used. This can be achieved by inserting timestamps in a flow of packets travelling from the MBU to the SH, in much the same way that time stamping was used for BDL measurements in Chapter 3. The sender and receiver clock are time synchronized through an NTP time server. A synchronize operation precedes every test measurement. With a custom built application, the clock skew between MBU and SH clock was determined to be insignificant (about 1 ms per minute). Delay variation (6) is loosely observed during measurements by recording the latency of

[28] See also: the group's homepage at http://www.ietf.org/html.charters/ippm-charter.html

individual test packets. An average is calculated so that test results can be compared with the network latency characteristics specified by the GPRS operator whose infrastructure is used.

- **Packet Loss**

One-way loss (3) rather than round-trip loss (5) is used because it is necessary to distinguish between the downlink and uplink path due to the high degree of asymmetry. By inserting sequence numbers in a flow of packets travelling over the uplink path, packet loss can be detected. Packets are collected for the duration of the test run. When the test is over, missing sequence numbers are assumed to have been lost. Precise measurements for loss patterns (7) were unnecessary because the random loss rate appeared to be very low (<1%) in general[29].

- **Capacity**

In terms of capacity the objective is to measure the *achievable bandwidth*. Achievable bandwidth is defined as the throughput that is achieved between two hosts over a network path given a specific set of conditions such as transmission protocol, host hardware and software configuration. The most important limits to achievable bandwidth are determined by the capacity and utilization of the tight link on the network path in combination with the capabilities of the hardware used by each host to access the network path. Although many techniques to measure link capacity (10) exist, none of these can be used to determine the capacity of the GPRS link. The uplink capacity model displayed by Figure 4.9 shows why. Cell capacity varies wildly per operator, and even per individual cell. As a result, achievable bandwidth needs to be measured similarly to bulk transport capacity (9). However, (9) requires a congestion-aware protocol, and the behavior of the IP layer is more closely matched by using a protocol that does not worry about flooding the network. Figure 4.10 presents the strategy to measure achievable bandwidth using a rate-controlled stream of packets.

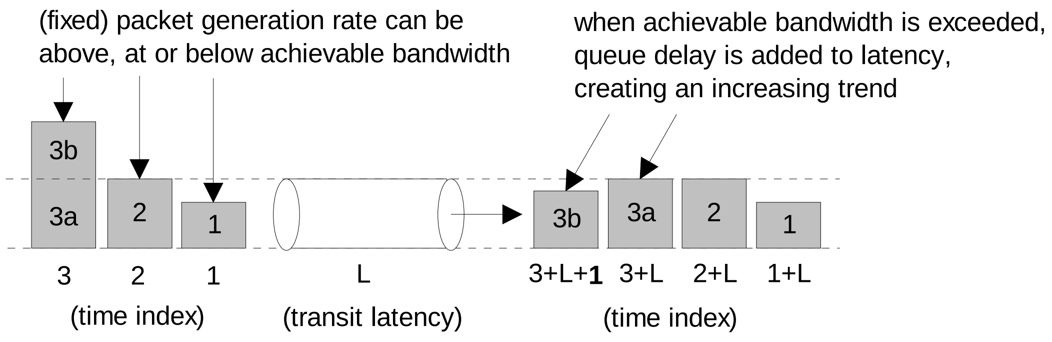

(fixed) packet generation rate can be above, at or below achievable bandwidth
when achievable bandwidth is exceeded, queue delay is added to latency, creating an increasing trend
3b
3a
2
1
3b
3a
2
1
3
2
1
L
3+L+1
3+L
2+L
1+L
(time index)
(transit latency)
(time index)

Figure 4.10: Rate controlled stream for achievable bandwidth measurements

With this strategy it is clear that as long as the packet generation rate is set above or at the achievable bandwidth, the receiver can determine the throughput that was achieved from the amount of data that arrives over a certain time interval. In practice though, severe problems occur with the MBU GPRS component when the UDP sender packet generation rate consistently exceeds the achievable bandwidth. [Yuch03] experiences similar problems, e.g. the GPRS component not transmitting any data anymore until a hardware reset is done. This creates an interesting paradox. The packet generation rate must be set exactly to the

[29] Some UDP measurements showed a lot of packet loss, but in all cases this was the result of packets being discarded due to the (fixed rate) sender overflowing the GPRS component. Measurements with TCP don't overflow the GPRS component and confirm that less than 1% packet loss occurs.

achievable bandwidth, because setting it lower causes an achievable bandwidth underestimation, while setting it higher causes severe test problems. Yet if the value of the achievable bandwidth were known before the test is executed, the test would not be necessary at all.

Figure 4.10 also shows a way to solve this problem by using latency trends. As long as the capacity of the underlying medium is greater than the load being generated, packet latency should remain relatively stable, i.e. only affected by one-way delay and delay variation. As soon as the generated load exceeds the capacity of the underlying medium, a queue of data will start to form. This queue will grow to infinity if the generated load remains above the capacity. Packets are time stamped before entering the queue, so the latency will also grow to infinity in this situation. If the load decreases below the capacity, the queue will gradually drain until it is empty again. In short, an increasing latency trend signifies load > capacity, stable latency signifies load = capacity and a decreasing latency trend signifies load < capacity.

Figure 4.11 shows a test run with a UDP bulk data stream transmitted over the uplink at three different packet generation rates: below, above, and roughly equal to the achievable bandwidth.

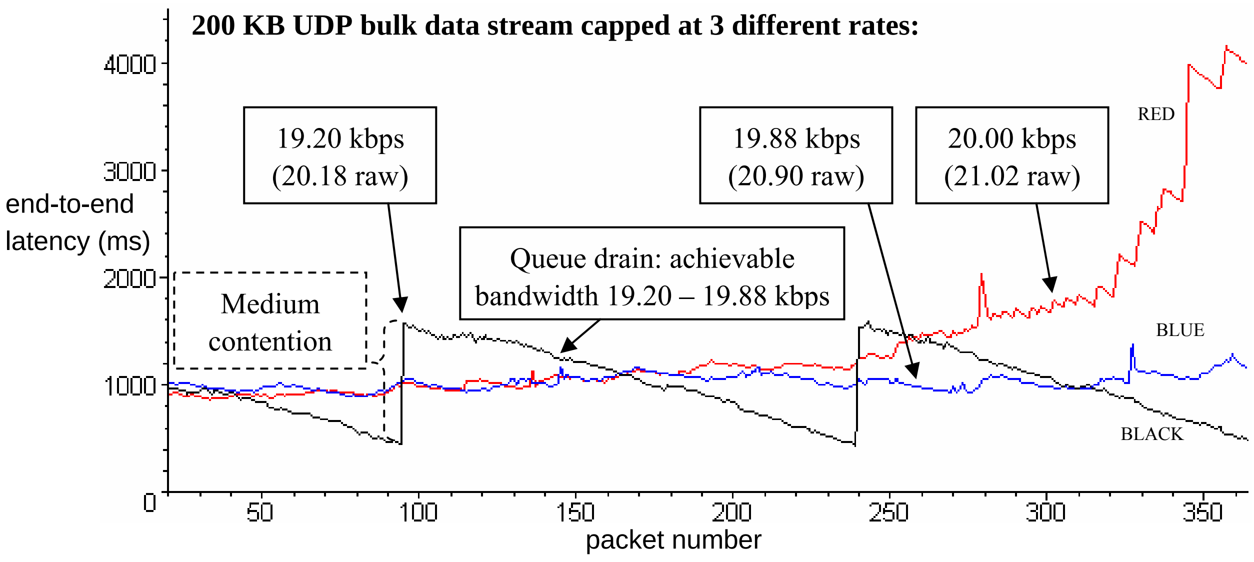


Figure 4.11: Typical latency behavior around maximum achievable bandwidth

Each latency graph has two numbers associated with it. The top number denotes the load generated by the packet generation rate in terms of UDP payload bits. The second (raw) number denotes the generated load if the IP and UDP packet header overhead is taken into account. This overhead is calculated as follows. The Maximum Transfer Unit (MTU) on the MBU is set to 576 bytes, meaning that the maximum size of an IP packet is 576 bytes. The IP and UDP headers are 20 and 8 bytes respectively, which leaves 576 – 28 = 548 bytes for UDP payload. The test program fills all 548 bytes of each packet, so the "raw" load including overhead can be calculated as (UDP payload load / 548) * 576.

The three tests were executed within a minute of each other, long enough to keep them from interfering with each other and short enough to experience approximately the same network conditions. A number of observations can be made from the test graphs:

- The red graph, the latency graph of the stream with the largest packet generation rate, shows an increasing trend (see also Figure 4.12). This means that the IP layer transmitted less than 20 kilobits of UDP payload per second, or less than 21.02 kbps including UDP and IP overhead.

- When the packet generation rate was slightly lowered to 19.88 kbps, the latency remained fairly stable during the test, meaning that this setting approached the load the IP layer was able to handle fairly closely.

- Further lowering the packet generation rate introduces an artifact of the GPRS medium: Temporary Block Flow (TBF) establishment. In short, mobile stations must go through a medium contention resolution phase in which a TBF is acquired that remains active for as long as the mobile station keeps transmitting[30]. As soon as the send buffer empties the TBF must be reacquired, causing the sharp latency jumps visible in the black graph. During the contention resolution phase, a small queue of time-stamped packets builds up. The queue drains slowly as soon after a TBF is reacquired because the packet generation rate is apparently below the medium capacity, which results in a progressive decrease in packet latency. The result is the sawtooth pattern with a stabilizing average latency (see Figure 4.12).

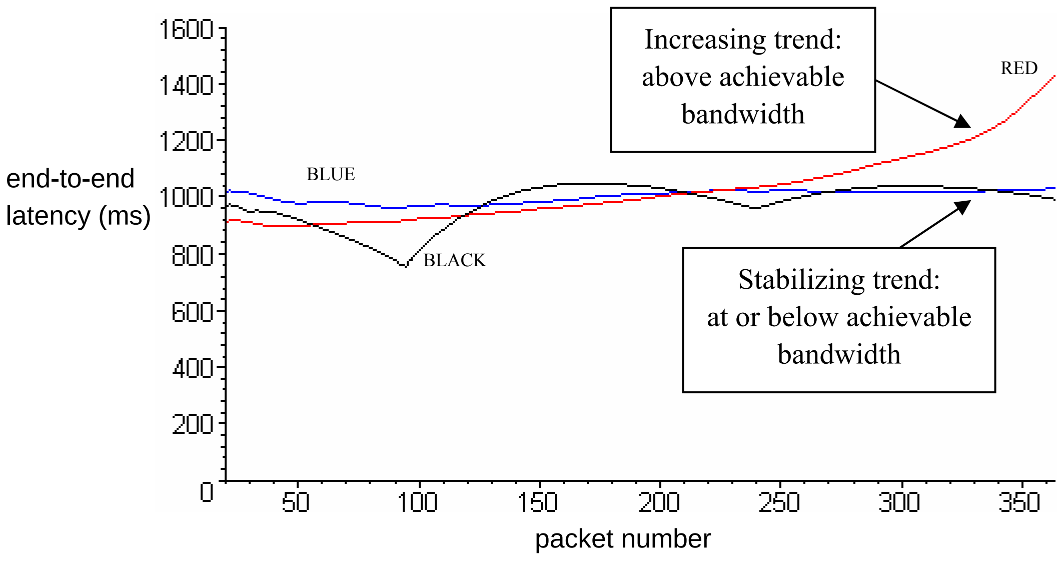


Figure 4.12: Averaged versions of the graphs in Figure 4.11

- Through the previous observations it is clear that the achievable bandwidth in the time period of the tests was between 19.20 and 20.00 kbps. The GPRS artifact causing the sawtooth pattern can be used to estimate the achievable bandwidth even more accurately. If the additional delay caused by the medium contention phase and the amount of packets it takes the medium to drain the queue caused by the delay are known, the required “overcapacity” can be calculated. The formula for this is the following:

  R_{ref} = rate in bytes per second at which data was generated

[30] This is operator dependent; operators may also choose to limit the validity of TBFs to the amount of data that a mobile station has ready to send at the time that the TBF is first requested

B_{drain} = total number of bytes sent in-between two medium contention phases
T_{cont} = delay in ms that the first medium contention phase adds
R_{medium} = rate in kilobits per second at which data was actually transmitted

$R_{medium} = 8 * B_{drain} / ((1000 * B_{drain} / R_{ref}) - T_{cont})$

The formula can be understood as follows. $1000 * B_{drain} / R_{ref}$ equals the time in milliseconds that it takes all the packets between two medium contention phases to be generated. The medium cannot transmit during T_{cont} milliseconds, because it is still acquiring a TBF. Dividing B_{drain} by the packet generation time minus T_{cont} yields the rate (in bytes*10^3 per sec) at which the bytes must have actually been sent. The result is multiplied by 8 to convert it to kbps.

For example, the first full sawtooth in the black graph shows that T_{cont} = 1500 – 400 = 1100 ms. The black graph has a packet generation rate of 19.20 kbps, so R_{ref} = 1000 * 19.2 / 8 = 2400 bytes per second. In between the two contention phases, 240 – 95 = 145 packets are sent. Each packet contains 548 bytes of payload, so B_{drain} = 145 * 548 = 79460 bytes. The formula yields: R_{medium} = 8 * 79460 / ((1000 * 79460 / 2400) – 1100) = **19.86 kbps** (raw: 20.88 kbps). This is a fairly accurate estimate of the **achievable bandwidth** during the test.

- **Integrated test method**

An integrated test method was devised so that all three characteristics (latency, loss, achievable bandwidth) can be measured at the same time. A Java application running on the MBU generates packets at a fixed rate and transmits them to an application running on the SH. Each packet contains a timestamp and a sequence number.

Before any measurements are made it is important to determine whether compression is being used in the lower layers. The negotiation phase during the GPRS connection establishment should provide the necessary information, but just to be sure two sets of each three tests are conducted. During the first three tests, a stream of zeroes is transferred at high speed. The second set of tests transfers bytes from a compressed file. If the average transfer rates of the two sets do not differ significantly, compression is not being used and tests can be conducted safely without being influenced by packet content.

Two packet generation rates are tested. The first rate is set slightly above the approximate expected achievable bandwidth to demonstrate a definitive upper bound to the achievable bandwidth (visible through an increasing latency trend). The second rate roughly equals the expected achievable bandwidth and should demonstrate through stable latency that the rate is indeed a good estimate of the real achievable bandwidth. Alternatively, the formula to determine R_{medium} above can be used in combination with a packet generation rate below the expected achievable bandwidth, but this method is less resistant to temporary interferences, and the added accuracy is also not required for these achievable bandwidth “probes”.

4.3.3 Performance Measurement Results

First, it is determined that there is no compression mechanism active on the path between the MBU and SH. The integrated test results are presented and discussed after that. For the test setup see Chapter 3.3.

In the script used by the ppp daemon to set up a connection, all options regarding compression (specified in the table below) are turned off.

```
novj
nobsdcomp
novjccomp
nopcomp
noaccomp
```

This compression would probably only be applied on the serial connection between the ppp daemon and the wireless pack so it should not make a difference either way, but the options are still turned off. There are no further indications of any compression being used, and the compression test results listed below testify that it is indeed turned off.

	Content = zeroes	**Content = ZIP file**
Throughput tests (Bps)	2609	2584
	2602	2606
	2586	2604
Average	**2599 Bps**	**2598 Bps**
Latency tests (ms)	849	855
	637	763
	1028	882
Average	**838 ms**	**833 ms**

A series of 24 tests were executed on random days and times to provide an indication of the performance of the IP layer in the test setup. In each test 200 KB of UDP payload is transmitted by a Java application running on the MBU. Latency, packet loss and achievable bandwidth statistics are collected by the SH.

A problem with the slot assignment occurred during testing. In about half the cases only 1 slot instead of the expected 2 slots were available on the uplink. This would be normal behavior if the cell were operating near its capacity, but Section 4.3.4 shows why it is unlikely that this was the responsible factor. To avoid any dependency on the source of the problem, the test set was divided into two separate sets: 12 tests in which 1 only slot was available and 12 tests during which 2 slots remained available.

Each of the two test sets consists of two sets of tests at a certain packet generation rate. As discussed in the previous subsection the first rate is set slightly above expected achievable bandwidth, while the second rate is set roughly at the expected achievable bandwidth. The tables below display the results of all four test sets. IP and UDP header overhead is accounted for in “raw” numbers (raw = 576 * payload / 548). Also note that the median is calculated to prevent extreme values from skewing the test results. Due to the even number of tests, the median is the arithmetic mean of the two middle values, e.g. mean[1,0,0,32,0,1] = (0 + 1)/2.

1 slot achievable bandwidth tests				
Packet Generation Rate Bps raw / (payload)	**Test #**	**Avg. Bps arrived (raw)**	**Avg. Latency (ms)**	**Packets lost**
1418 (1350)	1	1317	3772	1
	2	1296	4346	0
	3	1350	2778	0
	4[31]	1267	3469	32
	5	1364	2551	0
	6	1372	2422	1
	Median	**1334**	**3124**	**< 1**
1366 (1300)	1	1366	1073	0
	2	1354	860	0
	3	1359	892	0
	4	1362	886	0
	5	1363	824	1
	6	1361	1013	1
	Median	**1362**	**889**	**0**

2 slot achievable bandwidth tests				
Packet Generation Rate Bps raw / (payload)	**Test #**	**Avg. Bps arrived (raw)**	**Avg. Latency (ms)**	**Packets lost**
2628 (2500)	1	2073	3247	17
	2	1424	10410	124
	3	1684	7705	81
	4	2369	2214	0
	5	2300	2496	0
	6	2127	3619	13
	Median	**2100**	**3433**	**15**
2611 (2484)	1	2400	1454	30
	2	2595	1222	1
	3	2230	3012	8
	4	2613	950	0
	5	2607	989	0
	6	2612	618	0
	Median	**2601**	**1106**	**< 1**

Note that the achieved throughput rates are above the rate that is possible with CS-1, so that CS-2 must be the coding scheme in effect.

The results indicate that the packet generation rate that is chosen slightly above the expected achievable bandwidth is indeed consistently above the achievable bandwidth when noting the accompanying high average latency values. These high averages are actually the result of the predicted increasing latency trend. On the same note, the low latency averages that go with the packet generation rate that is chosen to roughly match the achievable bandwidth indicate

[31] GPRS component stopped transmitting before the end of the test, presumably because of overload

stable latency graphs. In a few cases the average is elevated a bit, which means that the estimates for 1 and 2 slot achievable bandwidth are probably slightly on the high side.

Remarkable is that the overall throughput is significantly degraded if the generated load exceeds the achievable bandwidth even by a relatively small amount, especially in the two slot tests. Rising latency and eventually packet loss are logical effects, but throughput degradation is not. The size of the queue waiting for transmission should not matter to the amount of data that can be transmitted through the IP layer. If anything, a larger queue should allow the MBU GPRS component to retain the its TBF, thereby avoiding the costly medium contention phase. As noted earlier on, the GPRS component seems to have some sort of buffering problem and the only way to avoid it is to avoid building up a queue (by not exceeding the achievable bandwidth).

In all cases where the achievable bandwidth was not exceeded, packet loss remained very low (less than one packet). Most of the packet losses in the results were the result of the system preventing excessive queues (and the associated exploding latency) by discarding packets, rather than losses caused by medium errors or congestion.

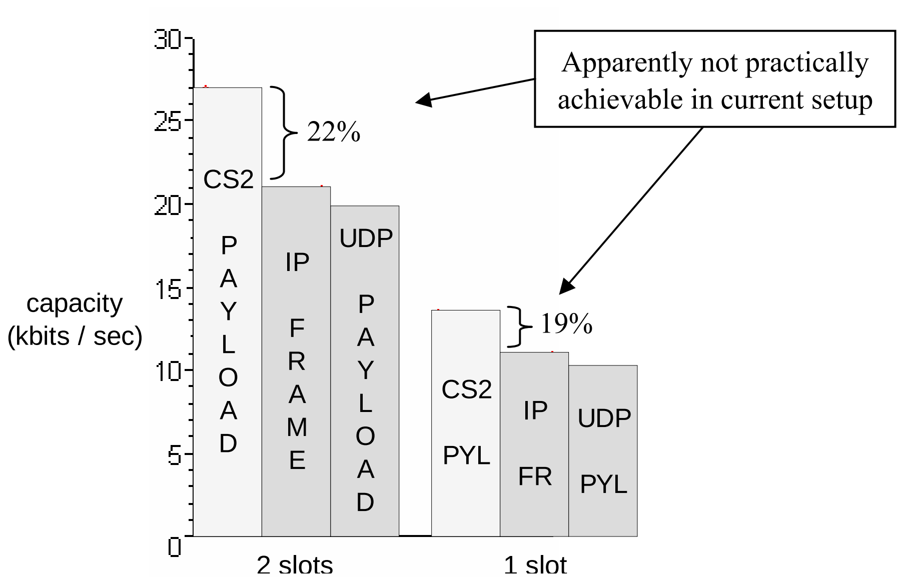


Figure 4.13: IP layer performance measurement results

Figure 4.13 shows a comparison between the achieved bandwidth and the expected capacity of lower layers that was determined in Section 4.3.1. According to the achievable bandwidth measurements when just 1 slot is available, only approximately 10.90 kbps (1362 Bps) is achievable (in terms of IP frames), instead of 13.40 kbps. This means that 19% of the capacity of the lower layers cannot be utilized or is lost on some unknown overhead. Similarly, if an extra slot is granted, 22% of the capacity cannot be used or is lost. The significance of these numbers is that they put a clear limit on the “raw bandwidth” usage of higher layers, which includes the BANip. Raw bandwidth is the amount of bytes per second used by complete IP frames.

The following section discusses the problem with slot assignments experienced during the measurements of this section.

4.3.4 GPRS Slot Problem

According to the tight link model of Section 4.3.1, the achievable bandwidth described in Section 4.3.2 should remain fairly stable around what is achievable with 2 slots (CS-2) assigned under the assumption that there is enough capacity in the cell. During testing runs the achievable bandwidth seemed to fluctuate a lot between 1-slot and 2-slot achievable bandwidth between tests, but rarely during tests. For example, suppose that three tests of two minutes each are executed in sequence, with a 15 second interval. Now suppose that test one and three are allocated only 1 slot for the entire duration of the test, while test two receives 2 slots for as long as the test lasts. A possible conclusion is that during test one and three, the cell was loaded so heavily that a second slot could not be assigned. That means that during the exact two minutes that test two lasted, with 30 seconds of added uncertainty (the pauses before and after test two) the cell load was decreased. The assumption that this is a random coincidence became less likely as the phenomenon occurred with an increasing number of tests. Adding to the mystery, 1-slot assignments were common even during the quiet hours of the night when the cell load can be expected to be minimal.

Further research into the phenomenon lead to the discovery of a strange second phenomenon. In all cases where only 1 slot was assigned it turned out to be possible to raise slot assignment to 2 for the duration of the test by using a simple “pause trick”. Whenever the achievable bandwidth seemed to be “stuck” with one slot, a temporary break was initiated. No more data was queued for transfer until the GPRS component finished transmitting the current queue of data[32]. Then, the program is resumed and it immediately receives 2 slots after the medium contention phase is completed once again. Figure 4.14 shows the results of 2 tests using the UDP protocol. To prove that the phenomenon is not related to the use of UDP or the test method, the same results were also acquired through tests using the TCP protocol (see Figure 4.15) and a different test method.

[32] Because transmissions on the GPRS frequency cause an audible disturbance in speaker and microphone equipment, it can be determined exactly when the GPRS component has finished transmitting.

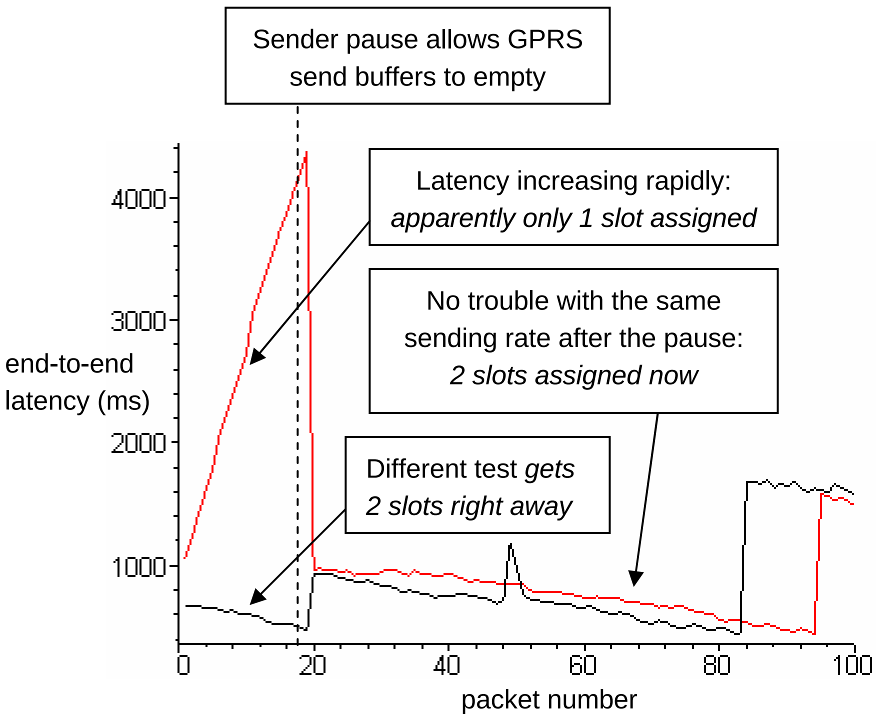


Figure 4.14: UDP test: slot gain after pause

Figure 4.14 displays two UDP tests that used the same fixed packet generation rate, somewhere slightly below 2 slot (CS-2) achievable bandwidth. During one of the tests packet latency initially rises rapidly. This indicates that only one slot is assigned, because the medium is rapidly building up a queue. After the pause trick, latency no longer rises rapidly. Moreover, a sawtooth pattern emerges, indicating medium under-usage. It is clear that 2 slots must have been assigned immediately after the pause. Comparing the graph to a graph of a different test that did get 2 slots assigned right away it is even more obvious that the pause trick caused a jump from an assignment of 1 slot to 2.

The TCP tests displayed in Figure 4.15 do not use any fixed rate packet generation. Furthermore, slot assignment is judged by looking at throughput rather than latency trends. The pause trick immediately causes an approximate doubling of the measured throughput for the remainder of the test, again indicating a jump from an assignment of 1 slot to 2.

A possible explanation for the slot problem is that slot assignment occurs only once, during the medium contention phase. This assumption would make it much more likely that the slot assignments as experienced in the tests reflect actual network load. However, slot downgrades have been observed in some TCP tests (without the test having been stopped), which would invalidate the assumption.

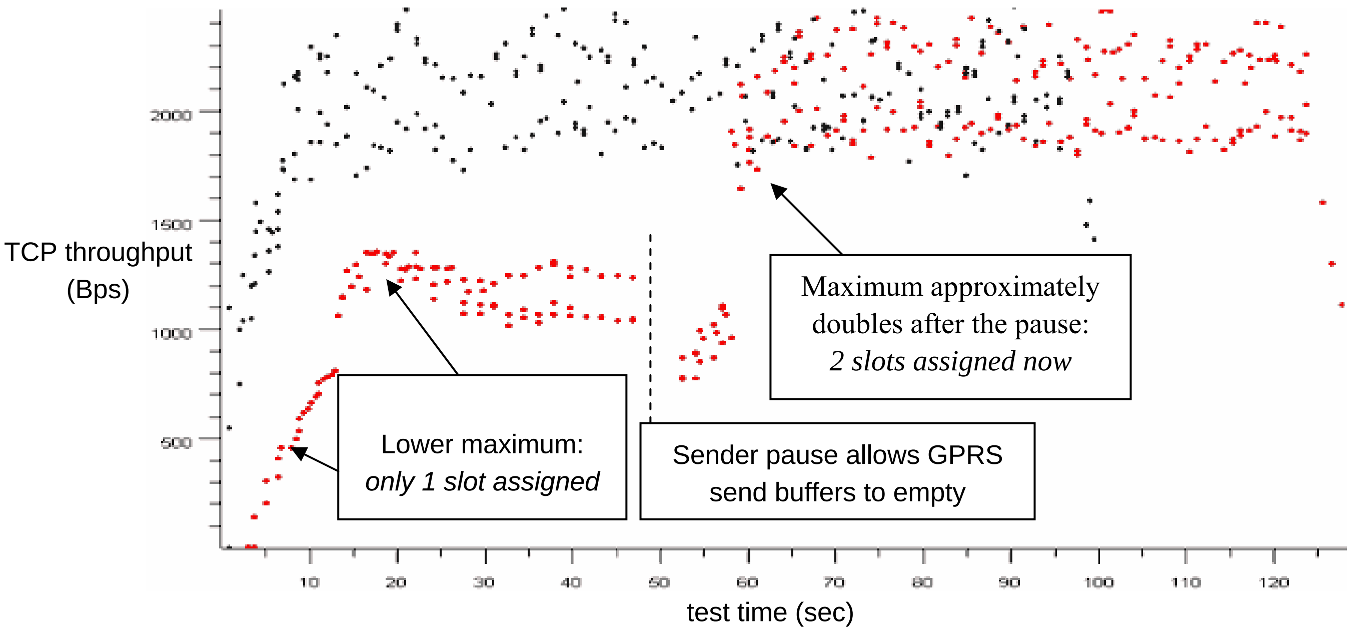


Figure 4.15: TCP test: slot gain after pause

As the phenomenon remains unexplained, the final hypothesis is that there is a problem with the GPRS component in the sense that it fails to adapt to new slot requirements on the uplink for as long as the transmission part is busy with user data.

To avoid negative influences of this phenomenon on the tests, the pause trick was not used and a distinction was made between tests that were assigned either just 1 or 2 CS-2 slots. More importantly, judging by the transfer capacity measured during the high level BANip tests in Chapter 3, it is not unlikely that slot assignment consistently remained “stuck” at one rather than two, which would be a significant performance drawback.

4.4 TCP Performance Drawbacks

The TCP protocol was designed and fine tuned with wired networks in mind. Therefore, this section focuses on the TCP drawbacks that occur as a result from the usage of a wireless network component in the setup of the BAN interconnect protocol.

Due to assumptions of usage on a wired network, it is not surprising that wireless network components present TCP with problems that require custom solutions. One of the problems that is easiest to see is that on wired links, packet loss usually means that the link is congested, whereas on a wireless link it could also mean that the medium lost the packet due to corruption rather than congestion. Because TCP is built to avoid congestion it will needlessly lower its transmission rate when such a corruption related loss occurs. The performance drawbacks identified in the wireless link (Section 4.3.1) are translated into TCP performance drawbacks below, keeping in mind that the largest volume of data travels from MBU to SH (MBU uplink channel).

- **<u>High average latency</u>**
  → Slow bandwidth probing
  When a TCP connection is established, the TCP protocol entity attempts to probe the throughput that can be reached without causing network congestion (see Section 2.3). The speed with which the throughput probe approaches this "target" throughput is tightly coupled with the round-trip delay of the connection because the next increase can only come after a new packet acknowledgement is received. Given the fact that the average round-trip delay on a single GPRS link is more than ten times larger than the average round-trip delay on a basic wired network path on the Internet, it is clear that the bandwidth probe will be significantly slower if a GPRS link is involved.

  → Slower loss detection & recovery
  Other slowdowns also occur due to high latency. Because TCP uses acknowledgements to detect losses, it needs at least one round-trip time to know whether a packet arrived, and preferably more just in case a packet or acknowledgement is temporarily delayed. The higher latency forces more time to pass before TCP can know about, and recover from a loss. Because TCP guarantees in-sequence delivery, the receiving TCP protocol entity cannot deliver any further data received after packet loss is detected until a successful recovery has occurred, which means that this data incurs additional latency.

  → Higher Bandwidth Delay Product (BDP)
  The BDP is calculated by taking into account the capacity as well as propagation delay of a link or path. Higher latency means that the BDP is higher, meaning that the amount of data that can be underway without an acknowledgement arriving is greater. Consequently, if the TCP send window (of the MBU) is not large enough, the connection will not be able to fully use the available capacity of the tight link.

- **<u>High latency variance; Possible non-congestion-related losses; Possible clustered losses</u>**
  → Congestion Window (cwnd) over correction
  A sudden jump in round-trip delay can trigger TCP's loss detection routine because the time in which TCP expects an acknowledgement for the packet to arrive (the round-trip timeout, RTO) has elapsed. Paradoxally, the round-trip delay jump could be caused by the lower layer ARQ mechanisms recovering a corrupted packet, e.g. by retransmission. So

lower layer recovery could trigger TCP loss detection and needless retransmissions. An additional inefficiency is caused by the fact that TCP asserts that these losses are a sign of congestion. In order to avoid congestion TCP reduces its congestion window, which results in link capacity under usage in this case.

→ **Premature termination of "slow" start**
Misidentification of packet losses and falsely concluding that all packet losses are indications of congestion (mentioned in the previous paragraph) can cause additional problems when the TCP protocol entity is still in the slow start phase. In the slow start phase bandwidth usage can be increased exponentially, whereas in the phase after that (congestion avoidance), bandwidth usage only increases linearly. Activation of the packet loss detection routine before bandwidth usage actually causes congestion results in a premature termination of the slow start phase. This means that it will take longer for the TCP protocol entity to utilize the approximate full capacity of the link.

→ **Slower loss recovery**
Clustered losses are not only present on wireless links but may also occur on wired links. An example of why TCP may suffer significantly from loss clustering can be found in [Fall96].

- **<u>Asymmetric latency characteristics; Asymmetric capacity characteristics (Uplink has the worst capacity and latency characteristics)</u>**
  Asymmetric characteristics problems are more pronounced in the traditional client/server setup where the largest volume of data flows over the downlink channel of a connected client (i.e. the MBU). In the MobiHealth system, the situation is reversed, so that the MBU uplink carries the largest volume of data. The achievable throughput over the uplink channel is not likely to suffer from a lack of capacity of the downlink channel because this capacity is usually higher.

- **<u>Possible negative compression effects</u>**
  As discussed in Section 4.3.3, link layer compression is disabled or does not cause any noticeable effects.

- **<u>Possible sharp capacity changes, but stable available capacity on the RLC/MAC layer most of the time</u>**
  → **Inefficient bandwidth utilization**
  Sharp capacity changes, and most notably sharp capacity increases can cause inefficiencies. While GPRS slot assignment changes can theoretically occur within milliseconds, the TCP protocol entity needs seconds to adjust to the new available throughput level due to the high latency of the GPRS link.

4.4.1 Performance Measurement Strategy

Similar to the performance measurement strategy of the IP layer (Section 4.3.2), the objective of the TCP performance measurement is to help identify significant performance drawbacks caused by the TCP protocol in the MobiHealth system configuration.

Any significant performance drawbacks affecting a TCP connection should be clearly visible in a graph that charts the throughput of a bulk transfer over the connection. Packets are captured at the MBU (sender) and SH (receiver), as the test setup in Section 3.3 specifies. Ethereal includes a packet capture analysis tool that can plot the throughput of a TCP connection. It is generally more useful to plot the throughput of the receiver than that of the sender, because only the receiver observes the combined effects of the network on transmitted data packets. Figure 4.16 shows a typical plot of a receiver packet capture. Two effects are immediately noticeable: **decreased throughput during connection start and when packet loss is detected**. Graph interpretation pitfalls must first be discussed however.

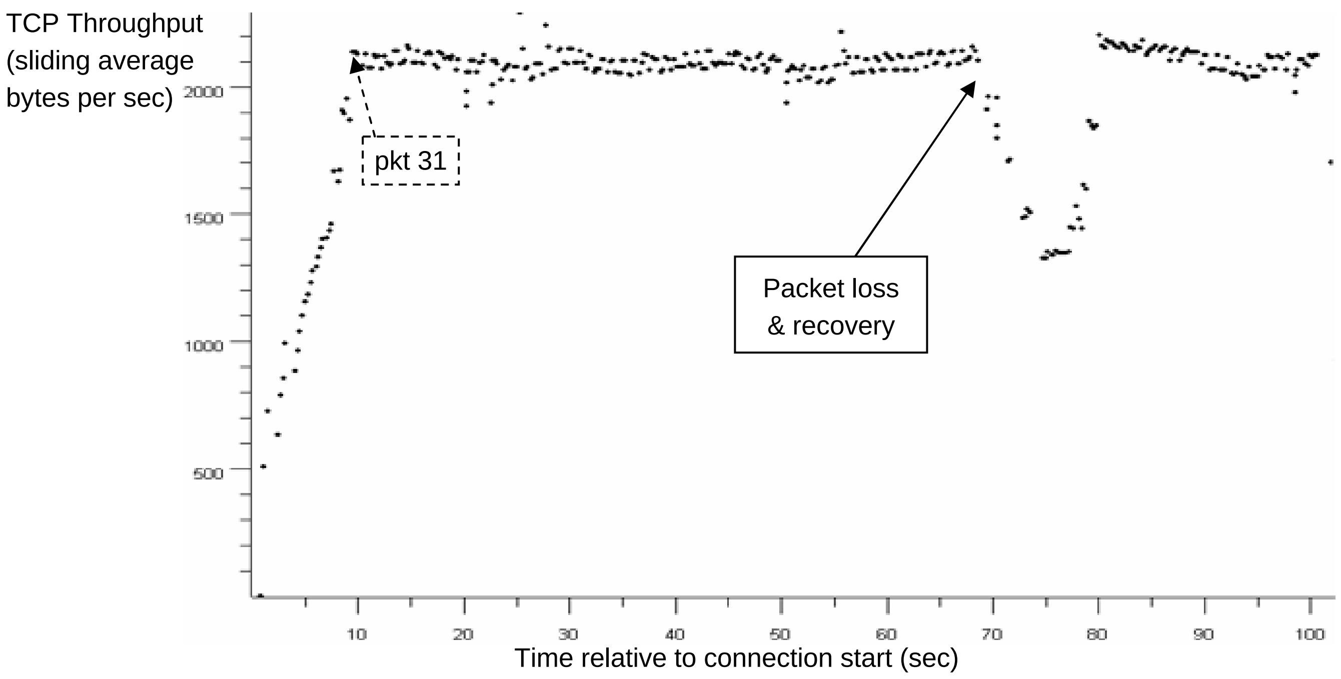


Figure 4.16: Typical ethereal TCP throughput graph (receiver)

The x value of each dot represents the arrival time of a packet. From the ethereal source code, packet capture and resulting graph it was determined that the y value of each dot is equal to the throughput achieved averaged over the last 22 packets. Each dot can be assigned a packet number n (starting at 1). x(n) would be the arrival time of the n-th packet.

The TCP protocol entity is verified to transmit packets with a constant payload size of 524 bytes during the whole bulk transfer, so generally speaking: $y(n) = (22 * 524) / (x(n) – x(n-22))$. Taking into account that the first two packets are SYN and ACK with 0 bytes payload, the formula would hold for $n > 24$.

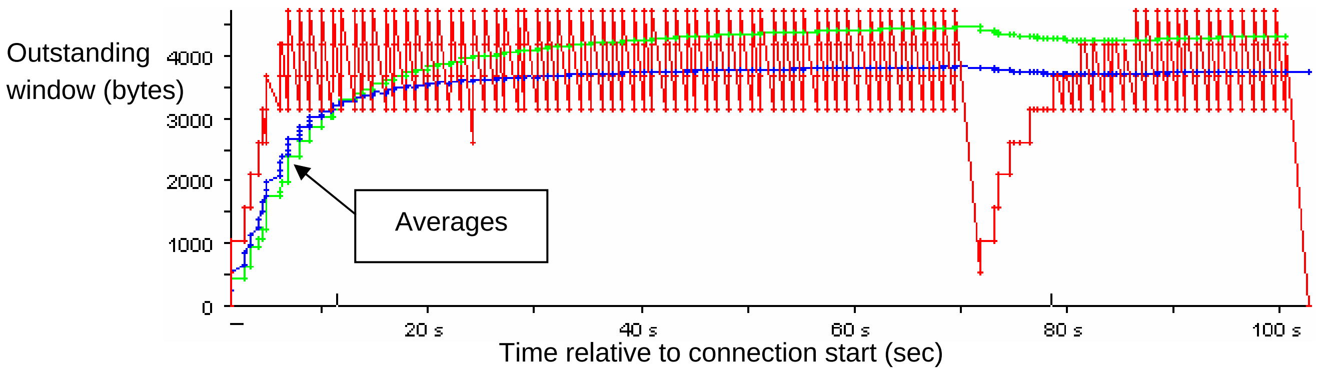


Figure 4.17: tcptrace outstanding window graph (sender)

It is important to realize that averaging over 22 packets causes the throughput displayed in the graph to always lag some time behind the actual throughput. In Figure 4.16 it looks like the maximum throughput is not reached until packet 31 arrives at x(31) = 9.5 sec. According to Figure 4.17, the maximum constant amount of outstanding data at the sender is already reached around 5-6 sec.

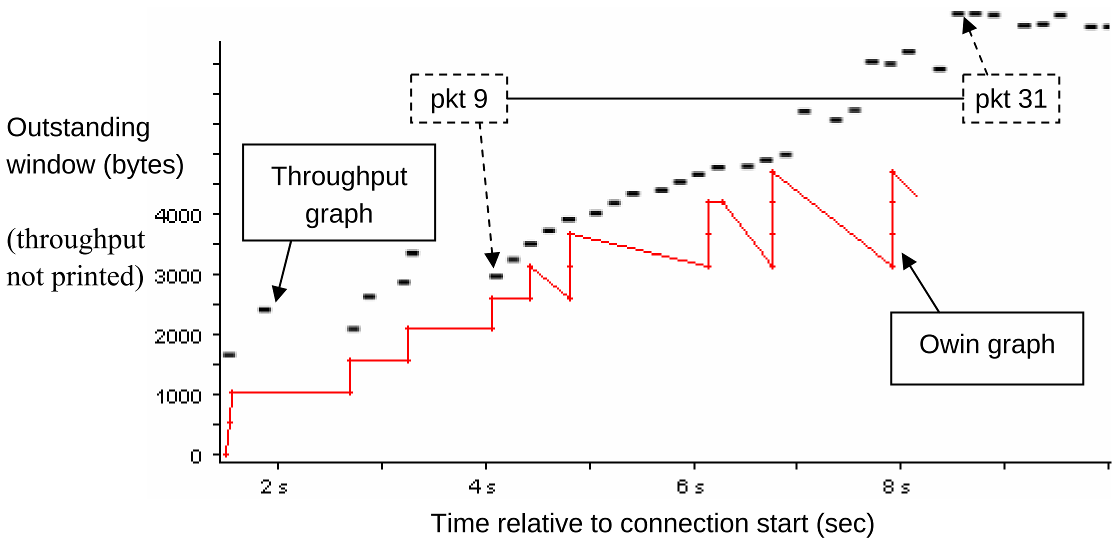


Figure 4.18: Sender outstanding window vs. receiver perceived throughput

It is possible to get a better estimate of point at which the throughput of the TCP connection first stabilizes around a maximum value. Figure 4.18 shows a combination of the start of the graphs in Figure 4.16 and Figure 4.17. The first point that is displayed very close to the maximum throughput value is point 31. Because the maximum throughput is so stable during this transfer, it is safe to conclude that the 22-point-average at point 31 can only be roughly equal to this maximum value if all points in the average contribute the approximate same maximum amount. In other words, at point 31-22 = 9 the maximum throughput is already reached. According to the receiver packet capture, x(9) equals about 4 sec.

These example graph calculations demonstrate that care must be taken when interpreting ethereal throughput graphs. It is necessary to take into account the lagging effect the averaging creates when determining how much time elapses before the TCP connection is fully[33] utilized. In the first 4 seconds, the average throughput is about 885 Bytes / sec. (can be read directly from the graph), which is not far below half the maximum measured throughput.

The second noticeable effect on throughput occurs when packet loss is detected. Figure 4.19 shows a time sequence diagram of the sender around the time the packet loss is detected to confirm a packet was indeed lost. Indeed, the figure shows that the sender received three duplicate acknowledgements and should therefore trigger a retransmission. To make sure that the sender only retransmit the packet that was actually lost, the receiver uses selective acknowledgements to indicate segments beyond the lost packet that do not require retransmission.

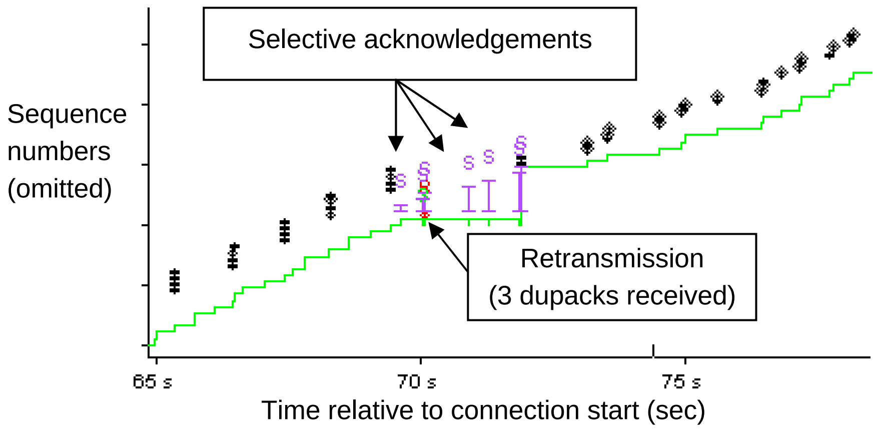


Figure 4.19: Time-sequence diagram near packet loss

Looking back at Figure 4.16, the first packet near the detected loss that causes the 22-point-average throughput to decrease is $n_1 = 269$, located at $x(n_1) = 69.4$ sec. The first packet at which the average has “recovered back to the maximum value” is n = 301. Following the same reasoning as with the start of the connection, the actual throughput has recovered at $n_2 = 301-22 = 279$ already, with $x(n_2) = 74.8$ sec. This means that the total time during which the connection was affected by the loss is 74.8 - 69.4 = 5.4 sec. Furthermore, it can be calculated that during this period the average throughput was a little less than half the maximum measured throughput: (279-269)*524 / 5.4 = 970 Bytes / sec.

The calculations in the previous part of this section provide a method to extract several important statistics from ethereal (receiver) throughput graphs:

1. Maximum achieved throughput values, assuming bulk transfer was measured and achievable bandwidth remained relatively stable for periods of at least 22*524 = 11528 Bytes of data transmitted (in general, achievable bandwidth appeared much more stable than that).

2. The amount of seconds the TCP flow required to approach the maximum measured throughput from the moment the connection is established.

[33] Fully utilized at least, in the sense that utilization does not increase for the rest of the transfer. It does not necessarily mean that the achievable bandwidth is fully utilized.

3. The amount of seconds (time interval) the TCP flow was affected by packet losses.

4. The average throughput over any given interval.

With these statistics the practical effect of the majority of the performance drawbacks listed in the introduction of Section 4.4 can be examined without requiring detailed data from the TCP protocol entity implemented in the operating system.

The only drawback that is very difficult to examine in the current setup is how TCP behaves during sharp capacity changes because as mentioned, the achievable bandwidth remains very stable most of the time.

4.4.2 Performance Measurement Results

The performance drawback concerning a send window that is possibly too small for the link BDP can be resolved without testing. The TCP send window is configured to 7 KB during the tests executed below, whereas the link BDP only requires approximately 2 * CS-2 bandwidth * link latency[34] = 2 * 1675 * 1.2 = 4 KB. When only 1 slot is available, BDP only decreases further below 4 KB. However, tests with various send window sizes uncovered a TCP send buffer problem (Section 4.4.3)

The performance measurement result tables below summarize the information gathered from twelve tests. The first table contains information extracted from six packet captures. The second set of tests (2 slots) could not be completed due to difficulties with slot assignment during packet capture testing. The table only lists values gathered through manual calculation with timestamps inserted in the data.

- The overall average throughput (for payload) is deduced through tcptrace.

- Peak (payload) throughput was measured manually from the graph as the approximate average of long stretches (>20 seconds) of high values.

- The time to full utilization, packet loss and loss influence were determined as demonstrated in the previous (strategy) subsection.

- The loss influence in throughput is calculated as the average throughput during the loss influence interval vs. the peak throughput. For example, 11.4; 5.0 | 90%; 84% means that the first loss degraded the throughput to 90% of the peak throughput for 11.4 seconds, while the second loss degraded the throughput to 84% of the peak throughput for 5.0 seconds.

[34] See [O2LSpec]

200 KB TCP transfer (1 slot, uplink)						
Test	**Overall throughput**	**Peak throughput**	**Time to full utilization**	**Packets lost**	**Loss influence**	
					Time	**Throughput**
1	1086	± 1130	2.9	-	-	-
2	1099		1.7	-	-	-
3	1115		2.8	-	-	-
4	1098		2.9	2	11.4; 5.0	90%; 84%
5	1122		2.9	-	-	-
6	1039		2.9	2	20.0; 6.6	77%; 28%
Avg.	**1093 Bps**	**1130 Bps**	**2.7 sec.**	**<1**	**10.8 sec.**	**70%**

200 KB TCP transfer (2 slots, uplink)	
Test	**Overall throughput**
1	1987
2	1981
3	1965
4	1920
5	1903
6	1993
Avg.	**1958 Bps**

Through these measurements it is possible to assess the theoretical performance drawbacks described in the previous subsection in the tested MBU/SH configuration.

- The “slowness” of the bandwidth probe in detecting the approximate achievable throughput is almost 3 seconds when 1 slot is available. During this time there is bandwidth under usage. In the case of 2 slots the exponential phase of the bandwidth probe will last longer, so the overall slowness should be less than double the amount of seconds for 1 slot. No bandwidth probe results were recorded for the 2 slot tests, but three ad hoc measurements indicated a probe slowness of about 4-5 seconds, which seems to confirm the theory.

- Bandwidth utilization also drops whenever packet loss is detected. On average, the utilization is reduced to 70% for nearly 11 seconds if a single packet is lost. Multiple packet losses per segment did not occur in the measurements. Note that selective acknowledgements played a profound role in reducing the negative effects of packet losses.

- The peak and average throughput can be compared to the IP layer measurements to determine the impact of the TCP control loops. For a direct comparison to be possible, the TCP payload throughput values must first be converted to values that include TCP and IP frame headers. Because TCP is stream oriented, it is usually difficult to tell how exactly payload is mapped to IP frames during a transfer. In this case all packets are recorded so that tcptrace can be used to determine the average segment size: between 506 – 509 bytes (out of 524 max.). The throughput in terms of IP frames can therefore be estimated with

576 * TCP payload throughput / 507.5.

Throughput measurement	**1 slot**		**2 slots**	
	payload	**IP frames**	**payload**	**IP frames**
IP (achievable)	-	10.90 kbps	-	20.81 kbps
TCP (peak)	9.04 kbps	10.26 kbps	-	-
TCP (avg)	8.74 kbps	9.92 kbps	15.66 kbps	17.78 kbps

Over 94% of the achievable bandwidth is utilized at peak throughput when 1 slot is available. Throughput averages around 91% and 85% of the achievable bandwidth when 1 and 2 slots are assigned, respectively. This includes the startup phase during which the TCP protocol entity attempts to estimate the achievable bandwidth and loss detection and recovery.

A final note on the measurements is that the following circumstances were not encountered, and therefore not examined for significant drawbacks:
→ changing assignment of slots during a measurement
→ clustered losses (multiple losses within a single segment)
→ losses falsely detected due to sudden large latency jumps

The following section discusses the TCP send buffer problem that was uncovered through some tests in this section.

4.4.3 TCP Send Buffer Problem

The TCP send buffer size on the iPAQ is set to 32 KB by default. As discussed in the previous two subsections, setting this buffer size below the BDP of the connection would affect TCP throughput negatively. Increasing the send buffer size beyond the BDP only means that the TCP protocol entity can deal efficiently with higher BDPs as well. The BDP for one and two slot mode is below 7 KB. That means that any send buffer size setting from 7 KB to the default of 32 KB should yield roughly equal throughput results. The problem uncovered is that larger send buffer size settings cause a progressively higher throughput performance drop (up to about 28%).

The following table lists the results of 20 tests with various send buffer size settings. The size of the send buffer can be adjusted in steps of 1088 bytes (i.e. size = multiplier * 1088).

Test #	**TCP send buffer size (multiplier)**			
	32640 (30)	**16320 (15)**	**9792 (9)**	**7616 (7)**
1	1364	1601	1773	(see previous subsection)
2	1688	1556	1525	
3	1293	1476	1776	
4	1619	1546	1767	
5	1189	1398	1905	
Avg.	**1430 Bps**	**1510 Bps**	**1749 Bps**	**1985 Bps**

To gain more insight into the problem, a set of 10 tests were also run with different GPRS hardware. Instead of used the attached GPRS wireless pack, the iPAQ was paired with a

Nokia 6310i phone via Bluetooth. Furthermore, the SIM module from the wireless pack was inserted into the phone to test with the same GPRS network. The test results are listed in the table below.

Test #	**TCP send buffer size (multiplier)**		
	32640 (30)	**X**	**7616 (7)**
1	2499		2426
2	2461		2265
3	2476		2503
4	2368		2473
5	2418		2307
Avg.	**2444 Bps**		**2395 Bps**

Figure 4.20 shows the results of both tables graphically. The iPAQ wireless pack obviously has problems with larger send buffer sizes. The phone does not suffer from the send buffer problem and performs as it should theoretically perform. Moreover, the maximum 2-slot throughput of the phone is roughly 459 Bps or 23% higher than the maximum of the wireless pack.

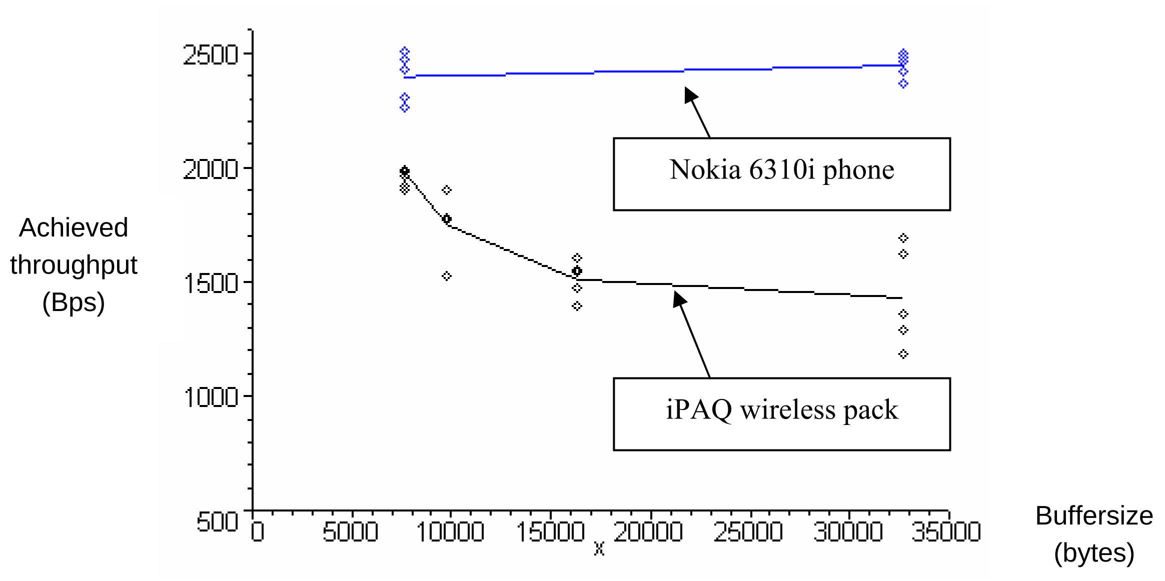


Figure 4.20: TCP send buffer tests

These tests lead to the following two conclusions:

1. The iPAQ wireless pack (in combination with iPAQ hardware and Linux drivers) has a TCP send buffer problem that is not present in other hardware.

2. Either because of this inefficient buffering performance or additional unknown inefficiencies, the iPAQ wireless pack is outperformed by different hardware by 23%.

To avoid the negative influence of the send buffer problem as much as possible, the send buffer size is limited to 7 KB in every measurement test that uses TCP with the exception of the reference BANip performance tests in Chapter 3.4. This is because the reference performance should include the effects of all performance drawbacks whereas improvements should resolve as many drawbacks as possible.

4.5 HTTP Performance Drawbacks

The first thing to note about the use of HTTP in the BANip protocol stack is that it is known to add many restrictions and performance drawbacks, yet remains a required element[35]. The purpose of this section is to identify and analyze the biggest performance drawbacks that stem from the use of HTTP in the reference configuration.

The main issue is that HTTP creates additional overhead in terms of data and time. Increased overhead in these areas translates directly into a lower Transfer Capacity and higher BAN Data Latency.

Section 4.5.1 discusses the way the reference implementation of the BANip utilizes HTTP and the overhead that is to be expected from this usage. The reference implementation does not use some important performance increasing options that HTTP version 1.1 provides:

- Section 4.5.2 examines the amount of HTTP overhead that remains if the first of these options, using **persistent connections**, were to be used by the reference implementation.

- The remaining overhead if the option of **chunked transfer coding** were used is examined in Section 4.5.3.

N.B.: This section consists mainly of a theoretical overhead analysis. Persistent connections and chunked transfer codings are not part of the reference BANip implementation so they were not tested in practice for this performance drawback analysis. However, they are implemented as part of the improved BANip due to the positive results of this analysis, so their effect is measured through the improvement measurements (see Section 5.3.3).

4.5.1 Overhead in Reference Implementation

A typical HTTP session consists of a client request and a server response. The role of client is played by the MBU, whereas the SH adopts the server role. To calculate the result of one request and response in terms of overhead, the way that HTTP affects the lower layer transport protocol (TCP) must be taken into account.

The table below shows the effect that each phase in the HTTP session has on TCP, and how this translates to the size and amount of IP frames that must be sent.
IPH and TCPH are the size of the standard IP and TCP packet header, whereas TCP PYL indicates any extra bytes that must be carried by the packet.

HTTP Phase	**TCP Result**	**Direction MBU ↔ SH**	**Bytes in IP frame**		
			IPH	**TCPH**	**TCP PYL**
CONNECT	TCP SYN	→	20	32	8
	TCP SYN/ACK	←	20	32	12
	TCP ACK	→	20	32	-
REQUEST	TCP DATA / ACK	→	20	32	H_1 + HP
RESPONSE	TCP DATA / ACK	←	20	32	H_2
	TCP ACK	→	20	32	-

[35] In order to achieve WAP 2.0 compliance

DISCONNECT	TCP FIN	←	20	32	-
	TCP FIN/ACK	→	20	32	-
	TCP ACK	←	20	32	-

The table contains three variables. H_1 and H_2 are the size of HTTP headers in those packets, whereas HP is the size of the HTTP payload in the request. When the BANip uses the HTTP layer, HP would be equal to the size of the BAN data being transported.

The size of HTTP headers varies, but for the sake of calculation the following estimates are used for H_1 and H_2:

H_1:

```
POST_HTTP/1.1<CRLF>          = 15
Host:XX.X.X.X<CRLF>          = 15
Content-Length:X<CRLF>       = 20
<CRLF>                       =  2
--------------------------------
                Total:       52 bytes
```

H_2:

```
HTTP/1.1_200_OK<CRLF>        = 17
Connection:close<CRLF>       = 18
<CRLF>                       =  2
--------------------------------
                Total:       37 bytes
```

The XX.X.X.X field in H_1 denotes the minimal length of an IP address. The X for content-length also represents the minimal amount of space a number can take up. Note that H_1 and H_2 are minimal header size estimates.

HP is a true variable in the sense that it should not be estimated but rather used as an input variable of the function that estimates overhead. This formula is given in the table below:

Direction MBU <-> SH	**Cost (IP frames)**	**Cost (IP frame bytes)**
→	5	5 * (20 + 32) + 8 + (H_1 = 52) + HP = **320 + HP**
←	4	4 * (20 + 32) + 12 +(H_2 = 37) = **257**

The formulas are only valid estimations when the assumption holds that the request consists of one packet total. Since the size of an IP packet in the configuration of the system is limited to 576 bytes total, HP should be at most 576 – 20 – 32 – H_1 = 472 bytes. If HP is larger, more packets and acknowledgements are necessary. The formulas can be adjusted to accommodate for this.

- The total number of packets to be sent for an arbitrary HP > 0, where base_cost_nr is the minimum number of packets that must be sent when HP = 0, is:
  N(base_cost_nr, HP) = base_cost_nr + nearest_higher_integer ((HP – 472) / 524)

  Explanation:
  The first packet contains up to 576 – 32 – 20 – H_1 = 472 bytes of HTTP payload.
  Following packets can contain 576 – 32 – 20 = 524 bytes of payload, because the header H_1 needs to be sent only once.
  When HP is up to 472 bytes, (HP – 472) / 524 will result in a negative number larger than –1. The nearest_higher_integer is zero, making N equal to base_cost for HP ≤ 472.
  If HP = 473, the division results in a number slightly higher than 0, and thus N = base_cost + 1

- The total number of bytes to be transferred over the uplink for an arbitrary HP > 0, where base_cost_p is the minimum amount of bytes payload that will be sent when HP = 0, is:
  BU(base_cost_p, base_cost_nr, HP)=base_cost_p+N(base_cost_nr, HP)*(20+32) +HP

  Explanation:
  N(base_cost_nr, HP) * (20+32) calculates the total amount of bytes that must be sent containing IP + TCP headers. Only base_cost_b and HP must then be added to get the total amount of bytes to be transferred. For example, using HP ≤ 472, base_cost_nr = 5, base_cost_p = 52 + 8, the formula yields the same outcome as calculated in the IP frame cost table above: 60 + 5 * (20+32) + HP = 320 + HP.

- The downlink cost formula is basically the same as the uplink formula, except that no HTTP payload is transmitted in this direction:
  BD(base_cost_p, base_cost_nr, HP) = base_cost_p + N(base_cost_nr, HP)*(20+32)

The adjusted formulas, valid for all HP are:

Direction MBU <-> SH	**Cost (IP frames)**	**Cost (IP frame bytes)**
→	N(5, HP)	BU(8+52, 5, HP) = **BU(60, 5, HP)**
←	N(4, HP)	BD(12+37 , 4, HP) = **BD(49, 4, HP)**

There is one more formula that matters for estimating the effect that HTTP usage has on the overhead in size. The current formulas can provide a close estimate for the amount of packets and bytes that will need to be transmitted to carry a certain payload of BAN data (HP). However, if pure TCP were used to transmit the same payload, there would also be a certain amount of header overhead. To estimate the real performance drawback that HTTP usage by the reference framework implies, the ratio between the total HTTP cost and TCP-only cost should be determined. For instance, if the cost to transport BAN Data payloads of 5000 bytes each were 6000 bytes “through the air” for TCP-only and 6500 bytes for HTTP enabled, then 6000/6500 = 92% of the cost would be justified. In other words, 100-92 = 8% of the total cost would be specifically induced by the usage of HTTP.

The formulas for TCP-only traffic are kept simple. The TCP connection is assumed be already set up and last across multiple payload transfers[36].

- The total number of packets sent over either the uplink or downlink for a specific payload HP is equal to:
  NT(HP) = nearest_higher_integer (HP / 524)

Explanation:
Each packet can contain 576 – 32 – 20 = 524 bytes of payload. The required amount of packets is the payload divided by this number. Since the amount of packets must be an integer, partial packets count as 1. This is achieved by rounding up.

- The total number of bytes to be transferred over the uplink is equal to:
  TU(HP) = NT(HP) * (32+20) + HP

- Over the downlink:
  TD(HP) = NT(HP) * (32+20)

Figure 4.21 shows the total overhead for HTTP and TCP-only as a percentage of the overall amount of data to be transferred. The displayed formulas are:

- HTTP:
  100 – 100 * HP / (BU(60, 5, HP) + BD(49, 4, HP))

- TCP-only:
  100 – 100 * HP / (TU(HP) + TD(HP))

Explanation:
The upload and download cost are added to get the total cost. Dividing 100 * HP by the total cost results in a percentage that indicates how much of the total cost is pure BAN data payload. The remainder is overhead, so 100 minus payload percentage equals the percentage of the total cost that is overhead.

[36] It might not seem fair to compare like this, but in reality the TCP setup/teardown of such a transfer would be negligable as is assumed here.

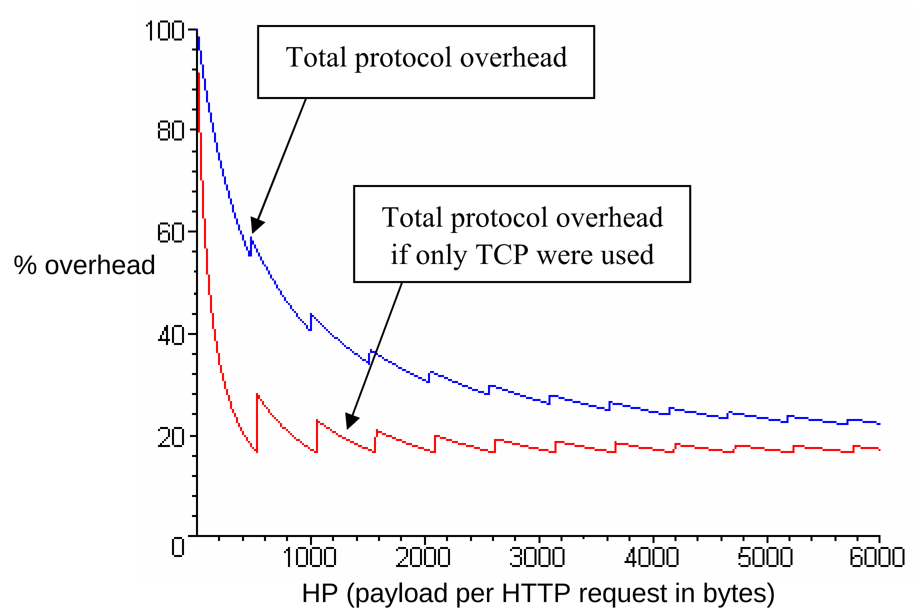


Figure 4.21: Protocol overhead vs. BAN data payload size

The two total protocol overhead graphs can be interpreted as follows: If the BANip passed a request to transfer HP bytes from MBU to SH, then the matching percentage on the y-axis indicates how much of the bytes (IP frames) that will actually be transmitted are pure header overhead. For example, if the BANip requests transmission of 5 standard 30 byte SensorSets (=150 bytes), 79% of the bytes that flow between MBU and SH on the IP layer are protocol headers if HTTP is used. 41% of the exchanged bytes would be protocol headers if only TCP were used.

Figure 4.22 represents the ratio between the HTTP and TCP-only overhead, as discussed earlier. Plotted is the formula:

100 – 100 * (TU(HP) + TD(HP)) / (BU(60, 5, HP) + BD(49, 4, HP))

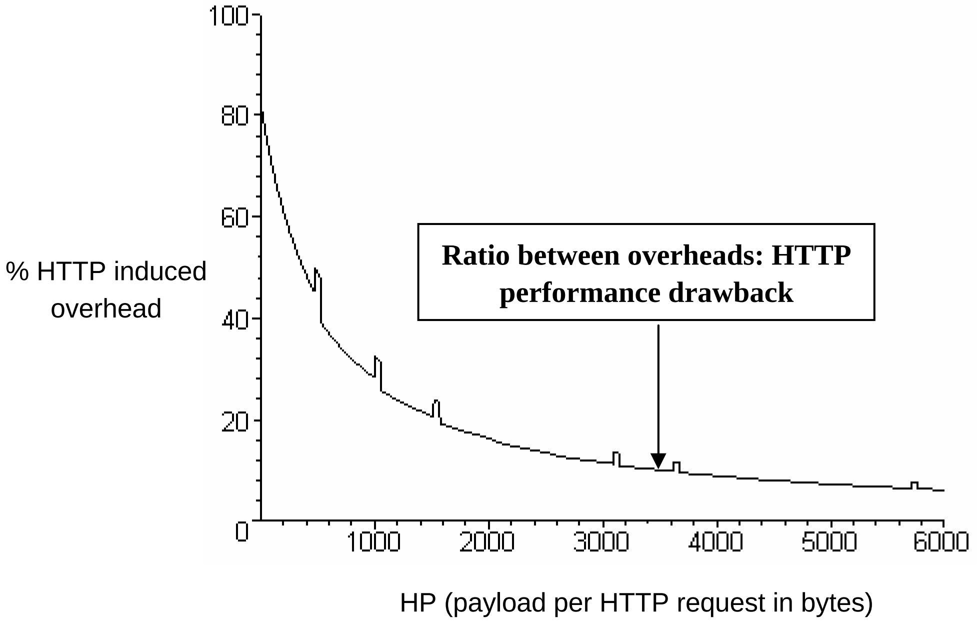


Figure 4.22: HTTP induced protocol overhead

The graph can be interpreted as follows: The percentage on the y-axis shows how much of the overhead created by HTTP was not necessary in the case where only TCP is used. For 150 payload bytes, HTTP causes 727 bytes to be transmitted while a TCP-only solution requires 254 bytes. That means that 254 out of 727 bytes or 35% are “justified”, while 65% are specifically added by the use of HTTP. At 1000 bytes per BANip payload transfer request the HTTP induced overhead decreases to 32%, and at 4000 bytes the overhead is only 9% compared to what is necessary with the TCP only solution.

Every byte takes time to transfer. Therefore, knowing the additional overhead that HTTP causes in terms of bytes gives an indication of what effective throughput decrease HTTP causes. There are more factors that negatively affect throughput when using HTTP however. For instance, the fact that every HTTP REQUEST must wait at least a full round-trip time before the connection is established. Another issue is that a stream of BAN data is segmented into HTTP REQUESTs that each await the HTTP RESPONSE, whereas TCP segments can be sent continuously (because the congestion window allows for a delayed reception of replies / acknowledgements). It has been found extremely difficult to express these additional influences in formulas in order to give a reliable prediction of the loss in throughput that is to be expected from the usage of HTTP. This is mostly because the formulas for the additional influences require models of the characteristics of the channel.

In summary, even though the true performance drawback associated purely with the use of HTTP cannot be predicted accurately with the calculations done in this subsection, it is still clear that small payloads per HTTP request will cause large performance drawbacks due to header overhead. Larger payloads reduce the header overhead but decrease the responsiveness (the receiver can not deliver any payload until it has been fully received). Also, larger payloads do not resolve additional issues that exist due to the HTTP request-response paradigm and constantly repeated connect / disconnect operations.

4.5.2 Persistent Connections

A traffic log of the “conversation” between the MBU and SH reveals that while HTTP 1.1 [RFC2616] is advertised for every exchange, the connections are always closed and reopened between subsequent exchanges (i.e. HTTP 1.0 behavior). HTTP 1.1 defines the possibility of keeping the connection open (persistent) between exchanges to avoid the cost of recurring connect / disconnect operations. This difference is displayed in the table below.

HTTP *without* persistent connections	**HTTP *with* persistent connections**
1. CONNECT	1. CONNECT
2. EXCHANGE (A)	2. EXCHANGE (A)
3. DISCONNECT	... EXCHANGE (B)
4. CONNECT	n. DISCONNECT
5. EXCHANGE (B)	
6. DISCONNECT	
...	

Clearly, HTTP without persistent connections incurs constant CONNECT / DISCONNECT overhead. In order to assess the difference in overhead, the cost of ongoing exchanges is examined. For HTTP without persistent connections there is no difference between the cost for a one-time exchange and an (ongoing) exchange that is part of a series of exchanges. The cost for an ongoing exchange using HTTP with persistent connections is displayed in the table below.

HTTP Phase	**TCP Result**	**Direction MBU ↔ SH**	**Bytes in IP frame**		
			IPH	**TCPH**	**TCP PYL**
REQUEST	TCP DATA / ACK	→	20	32	H_1 + HP
RESPONSE	TCP DATA / ACK	←	20	32	H_2

The cost formulas determined in the previous subsection can be reused to describe the cost for the behavior with a persistent connection, as the table below shows. For a more in-depth description of the cost formulas and graphs plotted from them (Figure 4.23 and Figure 4.24), see the previous subsection (Section 4.5.2).

Direction MBU <-> SH	**Cost (IP frames)**	**Cost (IP frame bytes)**
→	N(1, HP)	BU(52, 1, HP)
←	N(1, HP)	BD(37, 1, HP)

Figure 4.23 shows the cost comparison graph from the previous subsection with a new graph plotted for the cost of HTTP over a persistent connection.

The extra plotted formula is:

- HTTP (persistent connection):
  100 – 100 * HP / (BU(52, 1, HP) + BD(37, 1, HP))

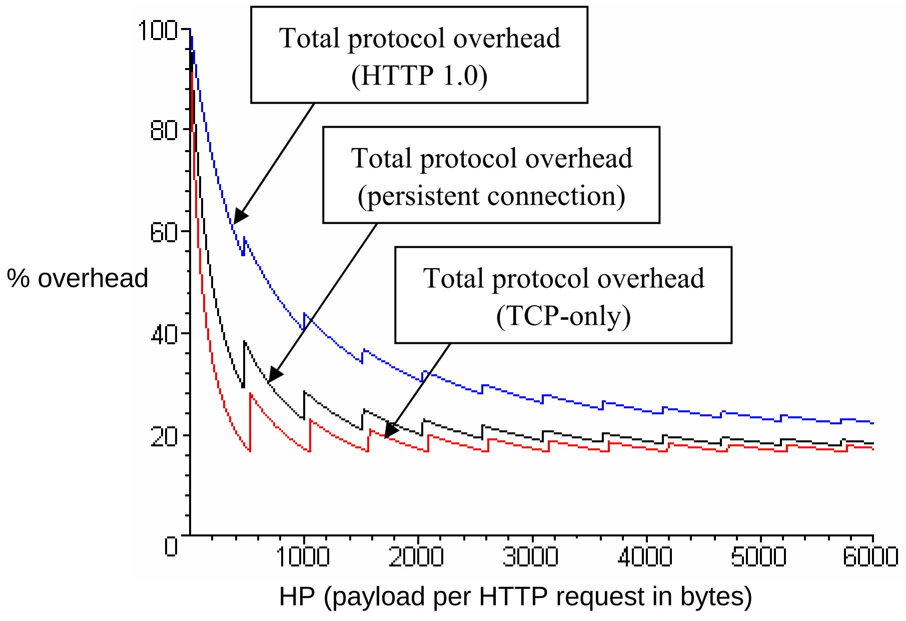


Figure 4.23: Protocol overhead vs. BAN data payload size

Finally, Figure 4.24 shows the ratio between the persistent and TCP-only graph.

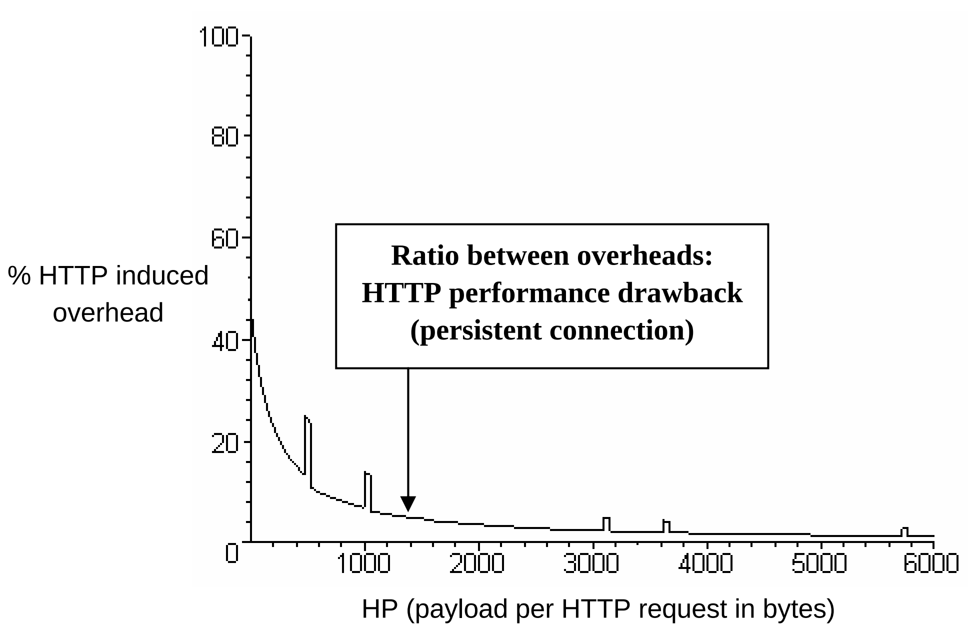


Figure 4.24: HTTP induced protocol overhead (persistent connection)

From the graphs it is clear that the usage of a persistent connection eliminates a lot of overhead that is associated with pure HTTP 1.0, especially at medium HTTP REQUEST sizes and smaller. Some other performance drawbacks such as the need to reestablish the connection and the associated waiting periods are also no longer present. The performance drawback that still exists in this configuration however, is the waiting period that occurs after every request. A new request cannot be sent until the response is fully received so that the transfer of data still stalls slightly after every request.

4.5.3 Chunked Transfer Coding

The chunked coding allows dynamically produced content to be transferred along with the information necessary for the recipient to verify that it has received the full message [RFC2616]. This coding modifies the body of a message (HTTP payload) in order to transfer it as a series of chunks, each with its own size indicator.

The following table displays a typical exchange using a request with a chunked transfer coding.

Phase	**What**	**Direction MBU <-> SH**	**Bytes in IP frame**
CONNECT	TCP SYN	→	60
	TCP SYN/ACK	←	64
	TCP ACK	→	52
EXCHANGE	TCP[HTTP REQ]	→	$52 + H_1 + C_1$
	TCP[HTTP CHUNK]	→	$52 + C_2$
	TCP[HTTP CHUNK]	→	..
	...	→	52 + Cn
	TCP[HTTP RESP]	←	$52 + H_2$
	TCP ACK	→	52
DISCONNECT	TCP FIN	←	52
	TCP FIN/ACK	→	52
	TCP ACK	←	52

Where C_1..Cn should each be defined as 6 + Psizechars + P, as can be seen below:

```
<CHUNK_PAYLOADSIZE_HEX><CRLF>    = Psizechars + 2
<CHUNK PAYLOAD><CRLF>            = P + 2
<CRLF>                           = 2
```

There are a number of ways in which a chunked transfer coding can be used. For instance, if the coding is merely used to segment the HTTP REQUESTs of the previous two subsections, then the only real difference would be that the chunks add extra data (overhead) and allow updates to be delivered to the receiver more periodically (increased responsiveness).

A different usage of the chunked transfer coding is proposed. The stream of BAN data is mapped onto a single request with a chunked transfer coding that may continue indefinitely. There is one large advantage and one large disadvantage to this mapping. The disadvantage is that the receiver of the chunks (SH) loses the ability to transmit data to the sender (MBU) via HTTP RESPONSEs, though establishing a second HTTP connection dedicated for reverse traffic easily solves this. The advantage of the mapping is that it basically encompasses the same advantages associated with using persistent connections but without the performance drawbacks related to the HTTP request / response paradigm. Because the MBU HTTP protocol entity remains in the HTTP REQUEST phase for as long as it wishes to continue transmitting BAN data in chunks, it does not need to wait for a HTTP RESPONSE.

It is not necessary to build new cost formulas for the chosen mapping because there is very little difference with the HTTP persistent connection cost as described in the previous subsection. Each chunk contains a small additional overhead of Psizechars + 2 bytes. Figure 4.25 shows the overhead versus chunk size. The overhead is insignificant (<1%) if chunk sizes of a little over 720 bytes (24 grouped BAN data units) or more are used. With a very minimal grouping of 2 BAN data units per chunk, the overhead is less than 12%.

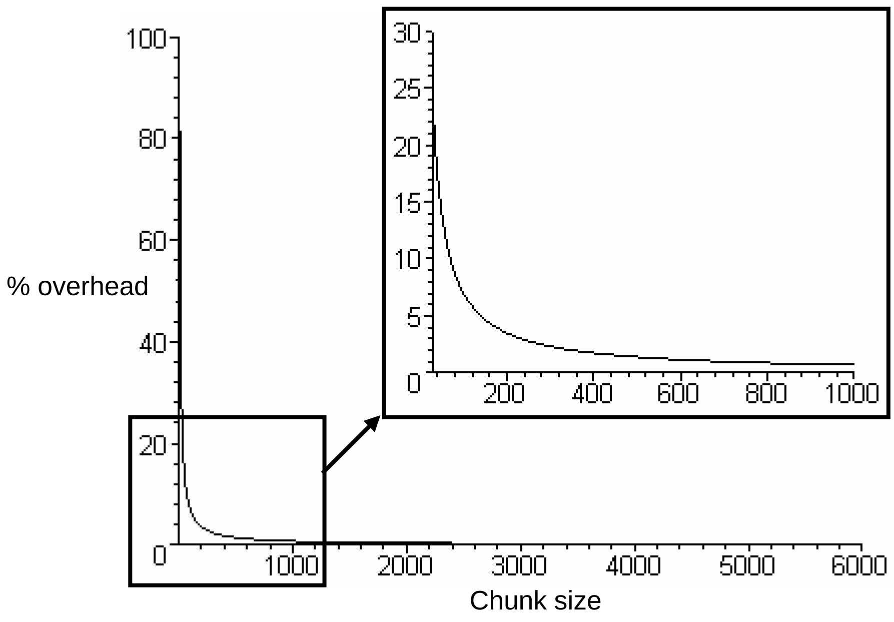


Figure 4.25: Chunk overhead

Besides the fact that the chunked transfer coding introduces a relatively insignificant byte overhead, the proposed way of mapping the stream of BAN data units changes the request / response behavior into something that closely resembles a normal bulk data transfer. This means that there are no significant performance drawbacks to be expected from using a chunked transfer coding that did not already exist in the lower layer TCP protocol. The main drawback that is created by using this mapping is a functional one, i.e. the loss of a previously available SH to MBU data channel. As stated, with a second HTTP connection this drawback can easily be compensated.

4.6 BANip Performance Drawbacks

The BANip redirects the task of transferring BAN data to a HTTP protocol entity. Internally however, the BANip uses buffering to aggregate 30 byte BAN data units into groups for more efficient transfer. The packetizer (PKT) setting determines how many units are aggregated into each group that is passed on as a whole to the HTTP protocol entity. Each first unit of a group must wait until the last unit of the group is collected before transfer of the group can commence. This means that the average BAN Data Latency (BDL) of units in the group is increased by half the delta time between first and last unit of the group by the packetizer "holding delay". Figure 4.26 shows the average BDL values recorded through the BANip performance measurement with the influence of the packetizer holding delay visualized.

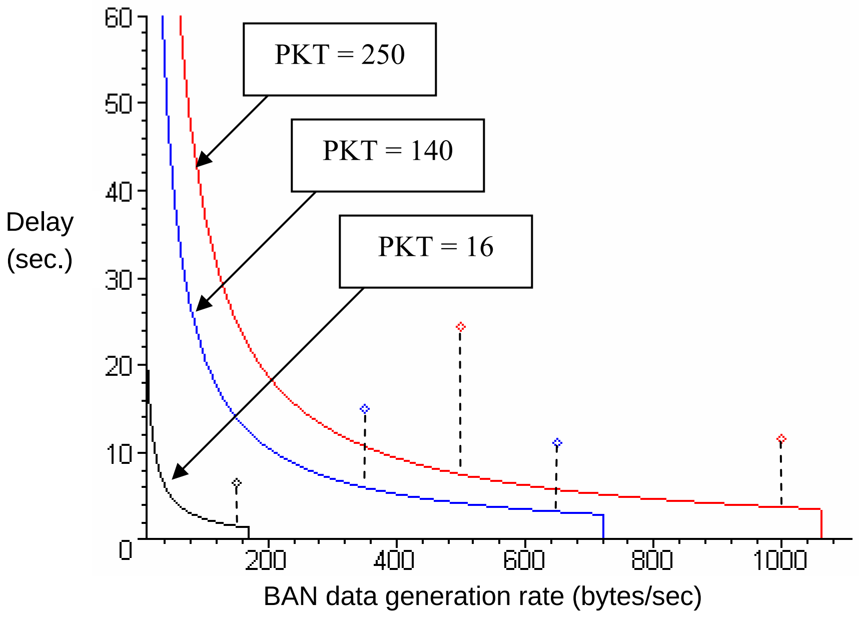


Figure 4.26: Packetizer holding delay influence on average BDL

An example in interpreting the graph is the following case:
The second blue dot at x = 650 indicates that for a packetizer setting of 140 * 30 = 4200 bytes per group, individual BAN data units had an average BDL of 11 seconds, of which a little over 3 seconds were caused by the packetizer holding delay (according to the curve beneath it).

The part of average BDL values that is not caused by the packetizer holding delay is indicated by dotted lines. The height of dotted line in the example case is equal to about 11 – 3 = 8 seconds. The BANip passes grouped units straight on to the HTTP protocol entity, so the BANip should not be responsible for any further significant delays. The time represented by the dotted lines is therefore the average time used by the actual transfer of the BAN data to the SH. In case the transfer does not proceed as fast as new BAN data units are collected, there will naturally be an additional queuing time within the dotted line. This was not the case during the tests (as verified by latency trends).

The value of Figure 4.26 is that it shows the main performance drawback with regard to BDL that can be expected from the BANip if a certain packetizer setting is chosen, given a certain arrival rate of new BAN data units (at the MBU, from the sensor(s)). Clearly, a low arrival rate of BAN data units is severely detrimental for their average BDL, the time that each unit will lag behind the time it was measured by the time it arrives at the SH. This is also counter-intuitive, as one would expect the BANip to yield faster transfer results if the load decreases but the opposite happens. Thus, implicitly, the fixed value of the packetizer represents a second significant performance drawback. It is a negative (exponentially increasing) influence on the average BDL as the BAN data unit load decreases linearly.

4.7 Summary and Conclusions

This section summarizes the potential performance drawbacks in the IP, TCP, HTTP and BANip protocol layers that were uncovered by the performance analysis and measurements in this chapter.

UDP and TCP performance measurements uncovered significant throughput degradations. The results of the measurements are summarized in Figure 4.27.

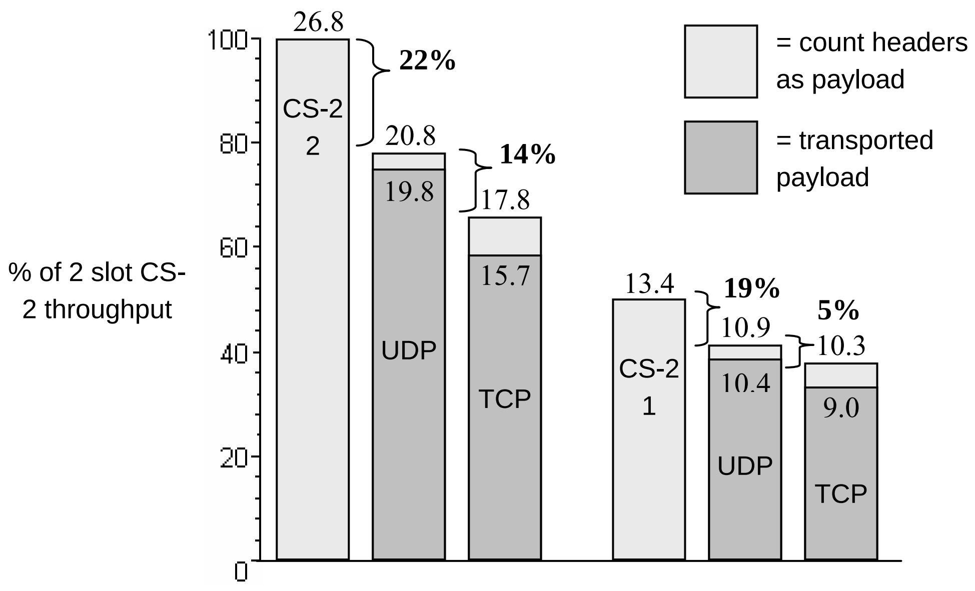


Figure 4.27: Throughput performance degradations

The numbers above and inside the bars are throughput in kbps. The two CS2 bars represent the theoretical throughput that can be achieved when 2 slots or just 1 slot are assigned during a transfer. The UDP bars show the average amount of throughput that could actually be achieved. The lighter part of the bar counts all bytes in the IP frames sent on behalf of the protocol as payload (for GPRS to transmit). The TCP bars are similar to the UDP bars, but obviously include the effects of TCP control loops (e.g. flow control and loss recovery).

Conclusions from the displayed results in combination with the results of the analysis in this chapter are discussed below.

- **UDP measured performance degradation**

The UDP performance measurements indicate that, on average, 20% of the projected CS-2 throughput remains unattainable with the reference system configuration. Measurements with different hardware (Nokia 6310i phone via Bluetooth instead of iPAQ wireless pack suggest that the problem is not hardware related. However, undocumented measurements with the different hardware and a different GPRS operator network lead to the hypothesis that the problem lies with the GPRS operator network.

Two other performance drawbacks that have a significant impact on the UDP measurements were identified:

1. Consistently exceeding the achievable throughput by even a little results in an additional significant degradation of the throughput. This was demonstrated by a series of (two slot available) UDP tests that differed only 0.6% in generated load, but resulted in average throughput values that differ 20%[37]. In some cases, the throughput overload even caused the GPRS component to stop transmitting any data at all, which was verified by "listening in" on the activity of the GPRS connection by means of a microphone. This phenomenon was also experienced in [Yuch03].

2. A strange phenomenon seems to be present in the way slots become available for the GPRS component. While it remains unexplained, the final hypothesis is that there is a problem with the GPRS component in the sense that it fails to adapt to new slot requirements on the uplink for as long as the transmission part is busy with user data. This results in transfers that seem to remain "stuck" with the particular slot assignment at the beginning of the transfer, which is a potential major performance drawback for a system that continuously transmits data.

- **TCP measured performance degradation**

Realistically, TCP should not be expected to achieve a higher throughput than the finely tuned UDP performance measurements achieved. Therefore, it is important to compare the TCP measurement results to the UDP measurement results instead of just to the theoretical CS-2 throughput. Early TCP performance measurements indicated that roughly 38% of the UDP-measured achievable throughput could not be achieved with TCP. After limiting the TCP send buffer, new measurements were performed. These are the measurement results displayed in Figure 4.27. They indicate that around 5 - 14% of the UDP-measured achievable throughput is not achievable for TCP. Measurements with different hardware (Nokia 6310i phone via Bluetooth instead of iPAQ wireless pack) achieved very close to 100% of the UDP-measured achievable throughput. This leads to the conclusion that in addition to the potential drawback that the GPRS operator network represents, the GPRS hardware also represents a potential significant drawback.

In addition to the measured performance drawbacks, the following potential performance drawbacks were identified through an analysis of the effects of the wireless link on TCP.

Accumulated effects on IP layer (compared to wired links)
High average latency (O_2 : uplink 600 - 1300 ms, downlink 400 - 1300 ms)
High latency variance (O_2 : uplink TBF adds 320 - 750 ms, downlink TBF adds 290 - 1700 ms)
Asymmetric latency characteristics
Asymmetric capacity characteristics
Possible non-congestion-related losses
Possible clustered losses
Possible negative compression effects
Possible sharp capacity changes, but stable available capacity on the RLC/MAC layer most of the time
Uplink has the worst capacity and latency characteristics

[37] Coincidentally, this is the same number as the unattainable CS-2 throughput. Note however that the "overload" 20% occurs *in addition to* the "unattainable" 20%, as the latter always occurred.

Note that the following circumstances may introduce additional performance drawbacks in TCP, but they were not encountered, and therefore not examined for significant drawbacks:
→ changing assignment of slots during a measurement
→ clustered losses (multiple losses within a single segment)
→ losses falsely detected due to sudden large latency jumps

- **HTTP performance drawbacks**

Despite the protocol overhead and limitations that the usage of HTTP introduces, it remains a required component in the BANip protocol stack. Two types of overhead identified in this layer represent particularly large performance drawbacks:

1. The overhead of constantly setting up and tearing down underlying TCP connections

2. The overhead introduced by the request-response paradigm

It is difficult to provide a theoretical estimate of the extent to which these performance drawbacks cause loss of throughput and delays. Sections 5.3.1 and 5.4.1 present a motivated estimate that suggests a possible performance improvement of 7% and 98% for eliminating performance drawback 1 and 2, respectively.

- **BANip performance drawbacks**

The grouped transfer of BAN data units is more efficient than transferring individual BAN data units separately. The BANip packetizer takes care of this. However, because groups are comprised of real-time BAN data units, the first unit of the group must wait for the last unit of the group to be measured before transfer of the entire group can *begin*. This delay is called the packetizer holding delay. Depending on the size of the group and how fast new BAN data units are measured, the packetizer holding delay can heavily affect the average BDL of BAN data units. The delay increases exponentially as the load (BAN data units measured per second) on the BANip *decreases*.

5. Improved Interconnect Protocol

This chapter presents the design, implementation and performance measurements of improvements to the reference interconnect protocol. These improvements are based on results presented in previous chapters. Chapter 3 provides the theoretical and practical means to measure and compare BANip performance, while Chapter 4 identifies major performance drawbacks in various elements of the BANip protocol stack.

The goals for the improved interconnect protocol are discussed in Section 5.1.

Each of the Sections 5.2 through 5.5 presents specific improvements in the form of a design and implementation. Each design is preceded by a table that summarizes the performance drawbacks that are discussed and resolved, as shown below.

Design summary			
Performance drawback	**Solution**	**Estimated improvement**	**Status**

Each solution in the table is accompanied by an estimated throughput improvement and a status. A status of “design improvement” (DI) indicates that the option is part of the implementation of the improved interconnect protocol. A “recommendation”(R) is argued to resolve certain performance drawbacks, but not implemented[38].

In addition to a design and implementation, Sections 5.2 through 5.5 also contain a performance measurement strategy and performance measurement results to determine the actual effect of improvements.

Section 5.6 summarizes the results of the improvements and examines the significant relative improvement with respect to the reference interconnect protocol.

Finally, Section 5.7 discusses the potential room for further improvements.

[38] Some solutions are technically beyond the scope of this thesis

5.1 Goals

The main goal of this chapter is to provide an answer to the following research question (defined in Chapter 1.4):

- How can the critical performance indicators [of a Body Area Network interconnect protocol] be positively influenced, thus improving performance?

More specifically, the aim is to present improvements that reduce the effect of existing major performance drawbacks (negative CPI influences) or represent significant positive CPI influences. Improvements consist of both a design and an implementation. Note that improvements must be measurable, which means that any possible performance drawbacks that could not be properly verified through measurements are beyond the scope this thesis. For example, the negative effects of loss clustering could not be measured because packet losses were relatively uncommon (<1%) on the network path used for tests. Performance drawbacks due to loss clustering are thus out of scope, even though they might become significant if the network conditions change drastically (e.g. if a different GPRS operator network is used)[39].

Derived from the main goal, the goals for the development of an improved interconnect protocol are to:

- propose changes in the design of the reference BANip protocol and protocol stack elements that should lead to measurable performance improvements (e.g. by reducing the effect of existing major performance drawbacks),

- present an implementation of the proposed design changes,

- present measurements[40] of the performance of the alterations and

- determine the improvement.

Note that the IP layer and anything below are an assumed basis, so the scope of the improvements is limited to reducing major TCP, HTTP and BANip layer performance drawbacks.

As mentioned in the introduction of this chapter, these goals are discussed for each improvement in Sections 5.2 through 5.5, while the overall improvement is discussed in Section 5.6.

[39] Even though this limitation is inevitable for this thesis, the approach used to analyze performance drawbacks is still largely independent of network conditions.

[40] BANip measurements are focused around a packetizer (see Section 2.5) setting of 140 BAN data units per group (about halfway between the minimum and maximum value) which represents a compromise between the importance of TC and BDL performance

5.2 TCP Option Tweaking

This section discusses the design, implementation and measured performance improvement of TCP related modifications.

5.2.1 Design

Design summary			
Performance drawback	**Solution**	**Estimated improvement**	**Status**
Initial window size	Increase to 4 * MSS	(unknown)	R
GPRS hardware	Upgrade firmware or replace with different type	~71%	R
TCP send buffer size	Limit to 7 KB	~38%	DI

TCP is known to exhibit performance drawbacks when there are one or more wireless links present on the network path (see Section 4.4). There are a number of options for dealing with this:

1. Most TCP drawbacks are related to flow control problems, so it is possible to replace TCP by a protocol without flow control, such as UDP.

2. There are efforts to modify the design of TCP so that it performs better when wireless links are present on the network path. Examples are Westwood+ TCP [TCPWW] and Wireless TCP [WTCP]. The implementation on the system could be replaced by one of these modified versions.

3. TCP offers a number of parameters that may be adjusted, for example send and receive buffer sizes. Various optional extensions have also been added to TCP through the years. Selective acknowledgements (SACKs) for example reduce the amount of data that must be retransmitted after packet loss is detected, and have been known to increase TCP performance over GPRS ([Chak02], pg.4). Tweaking these options can potentially reduce the performance drawback of TCP over a wireless link.

There is a problem with the first solution. Even though the achievable bandwidth appeared fairly stable on the reference network, there is no guarantee for this. Additionally, the MobiHealth system must operate in different countries on the networks of different wireless network operators, with different capabilities (GPRS and UMTS) and even different hardware. In short, the solution would not scale well because flow control is a necessary tool in utilizing dynamically varying bandwidth. Although custom built flow control mechanisms can be added, doing this would probably result in a lot of reinventing the wheel without the guarantee that a suitable solution for the same problems TCP faces would be found. Had there been a known fixed amount of bandwidth in all cases, then the first solution might have been more appropriate, but as it stands it was not considered to be a viable solution.

When comparing solution two and three it is clear that the latter requires less effort and is less experimental. Acquiring modified TCP versions not commonly used can be difficult, and manually replacing Linux kernel code could potentially result in a TCP implementation that does not work as intended. Hence, the third solution was investigated for the initial design. Performance measurements show that TCP behaves reasonably well (see Section 4.4.2) after minimal tweaking, so solution two was not investigated any further. Solution three is described below.

[WPTCP] describes how TCP should be configured so that it more efficiently copes with wireless links on the network path. The reconfigured version is dubbed *Wireless Profiled TCP* (WPTCP, not to be confused with [WTCP]). The following TCP options and values are specified for WPTCP:

TCP option	**Recommended value**
Selective Acknowledgements (SACK) [RFC2018]	On
Window Scale [RFC1323]	On
Timestamps [RFC1323]	On
Disable Path MTU Discovery [RFC1191]; [RFC1981]	Off
Explicit Congestion Notification (ECN) [RFC3168]	Off

Large window size based on Bandwidth Delay Product
Bytes of window reserved for buffering overhead
Application part of receive buffer set to ¼.
Receiver Buffer Size (min, default, max)
	
	31
	2
	4096 87380 174760

Larger initial window [RFC3390]	4 * MSS[41]

It appears that in the reference MobiHealth system, all of these options except for the large initial window are already set to the recommended value. However, [RFC3390] does not specify how this option should be set on a Linux system. Some time was unsuccessfully spent looking for a way to set this option (the same problem was encountered in [WPTCP]). Because larger initial window sizes mostly benefit short lived flows, and given the fact that this design proposes improvements that eliminate the need for short-lived flows, the option was not researched any further.

In summary, TCP was verified to be wireless profiled with the exception of one option (larger initial window) that does not significantly affect the improved interconnect protocol. It is tempting to assume after this verification that no other option tweaking should lead to any significant improvements of TCP performance. However, measurements uncovered that changing the default setting of an option that is not specified in [WPTCP] may lead to as much as 38% improved[42] TCP throughput (see Section 4.4.3).

The option in question is the setting of the TCP send buffer, whereby smaller buffer sizes (down to 7 KB) lead to a progressive performance improvement. The fact that the setting does not significantly impact throughput when different GPRS hardware (but the same TCP protocol entity and network) is used seems to point to a problem with a deeper origin, in the GPRS hardware or kernel drivers. The theory for this problem is as follows:

- Observations in Section 4.3.3 suggest that the GPRS hardware suffers significant throughput loss when the achievable throughput is exceeded.

- Setting a smaller send buffer size basically limits the TCP uplink congestion window. [Chak02] shows that this results in significantly lower round-trip delays, which makes

[41] On the MBU the MTU is set to 576 bytes, so the MSS is 576 – 20 (IP) – 32 (TCP) = 524 bytes

[42] Or, when viewed as a drawback, the default setting equals a 28% decrease from the altered setting

TCP more responsive to changing network conditions. The technique used in [Chak02] is called TCP window clamping, a procedure whereby the TCP congestion window is fixed to a certain value. Measurement results in [Chak02] indicate a decrease of the round-trip times of around 80% if the congestion window is clamped to 10 KB for a GPRS device using 4 slots[43].

- Consequently, it is theoretically possible that the reduced send buffer size allows TCP to react quickly at the first sign of decreasing throughput, thereby significantly reducing the negative effect of "achievable throughput overloads" seemingly caused by the GPRS hardware.

A partial solution to the problem is therefore to set the maximum TCP send buffer size to a conservative value of about 7 KB. As mentioned, this setting improves performance by about 38% over the default settings of the buffer for the reference system setup. This is an acceptable solution for as long as the uplink Bandwidth Delay Product (BDP) does not change significantly (i.e. uplink slot capability of the GPRS hardware remains at 2 slots or lower and the latency characteristics of packets on the GPRS network remain roughly the same).

If the uplink BDP does change significantly, the congestion window limitation might be too tight. TCP's congestion window must be able to grow at least equal to the link BDP, or throughput loss will occur. This is also why it is better to treat the problem closer to its origin:

- A possible solution is to upgrade the firmware of the GPRS hardware (iPAQ wireless pack). Unconfirmed measurement results suggest that this approach to eliminating the buffer problem is at least partially successful.

- Another solution is to switch to a different type of hardware, for example to the alternative hardware tested in Section 4.4.3 (using a phone as the GPRS modem). The measurements in that section indicate that with this hardware the TCP protocol entity achieves over 94% of its expected maximum achievable throughput, an improvement of about 71%.

Even with the conservative send buffer setting, which improves throughput by about 38%, the TCP protocol entity can still only achieve up to 76% of its expected maximum achievable throughput. With this observation in mind, either one of the two solutions listed above definitely seem to be more favorable. This thesis aims at finding solutions for the hardware used in the reference setup however, so the partial solution of limiting the send buffer size is part of the design of the improved interconnect protocol, while the favorable solutions remain recommended.

5.2.2 Implementation

A number of TCP options have been checked and verified to be set according to their recommended values. On the Linux operating system installed on the iPAQ, each option is linked to a file (see Appendix A: Test setup specifications) in the /proc directory structure.

[43] That's double the amount the GPRS hardware used for this thesis supports on the bulk transfer link

The table below lists these options, the specific location of the file that is linked to each option, as well as the value the option is set to. The value of options can be displayed or adjusted by executing '*cat file*' or '*echo setting > file*', respectively.

TCP option	**File (/proc/sys/net/ipv4)**	**Value**
Selective Acknowledgements	tcp_sack	1 (= on)
Window Scaling	tcp_window_scaling	1 (= on)
Timestamps	tcp_timestamps	1 (= on)
Disable Path MTU Discovery	ip_no_pmtu_disc	0 (= off)
Explicit Congestion Notification	tcp_ecn	0 (= off)
Bytes of window reserved for buffering overhead	tcp_app_win	31
Application part of receive buffer set to ¼.	tcp_adv_win_scale	2
Receiver Buffer Size (min, default, max)	tcp_rmem	4096 87380 174760

According to the design, the TCP send buffer size option needs to be adjusted to the value displayed in the table below.

TCP option	**File (/proc/sys/net/ipv4)**	**Value**
Send Buffer Size (min, default, max)	tcp_wmem	4096 7616 7616

This value is reduced from the default value, but has been measured to increase the average throughput in the reference system setup.

5.2.3 Performance Measurement Strategy

No additional performance measurements are executed for the TCP option tweaks made in the design. The only setting that was changed in the reference system configuration is the TCP send buffer size. It is limited to 7 KB. The performance impact of this altered setting on the TCP protocol is already measured and discussed in Section 4.4.3. Furthermore, the setting is used in every set of measurements except the reference BANip measurements because the default setting causes more unreliable behavior.

5.2.4 Performance Measurement Results

As mentioned in the previous subsection, the TCP send buffer improvement is already measured and incorporated in all other improved interconnect protocol measurements. Section 4.4.2 contains the measurement results of the TCP send buffer improvement itself. The average measured throughput (TCP payload) was 1958 Bps.

5.3 HTTP: Persistent Connections

This section discusses the design, implementation and measured performance improvement of applying HTTP persistent connections.

5.3.1 Design

Design summary			
Performance drawback	**Solution**	**Estimated improvement**	**Status**
Separate TCP connection for every HTTP request	Use HTTP persistent connections	at least ~7% (Figure 5.1)	DI

The use of HTTP in itself adds drawbacks to any protocol built on top of it in the form of overhead and restrictions to the way the protocol can operate (in the sense that it is bound to the request-response paradigm). Section 4.5.2 reveals that the HTTP protocol entities of the MBU and SH establish a separate TCP connection for every HTTP request. This behavior is associated with the older HTTP 1.0 standard [RFC1945]. In the HTTP 1.1 standard [RFC2616] the possibility of reusing the same TCP connection is defined, thus avoiding the overhead of constantly tearing down and setting up TCP connections. Also, a lasting connection allows the underlying TCP protocol entity to more efficiently estimate the congestion state of the network, because a new estimate must be made for each new connection. Figure 5.1 shows an estimate of the protocol overhead reduction as a result of using HTTP persistent connections.

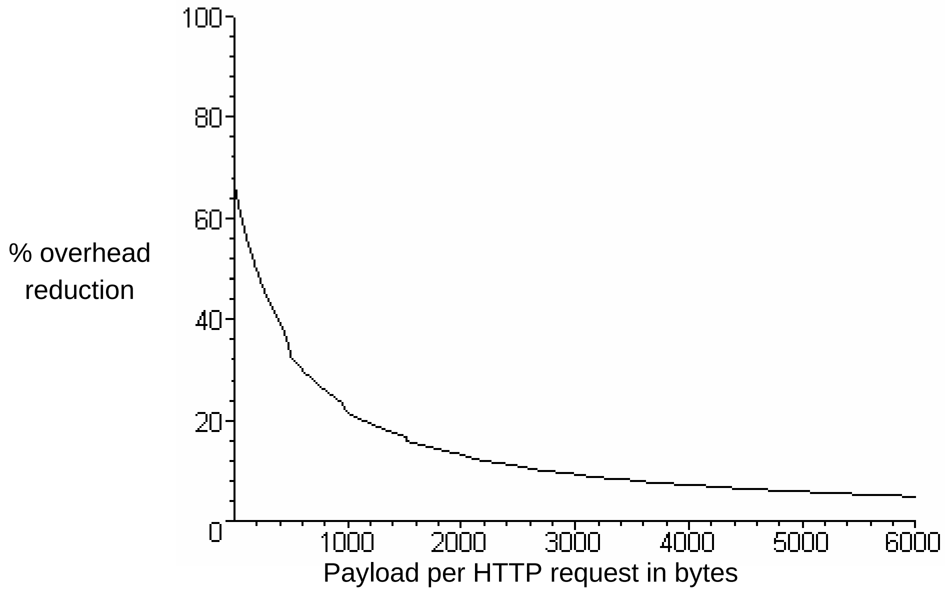


Figure 5.1: Persistent connection improvement due to overhead reduction

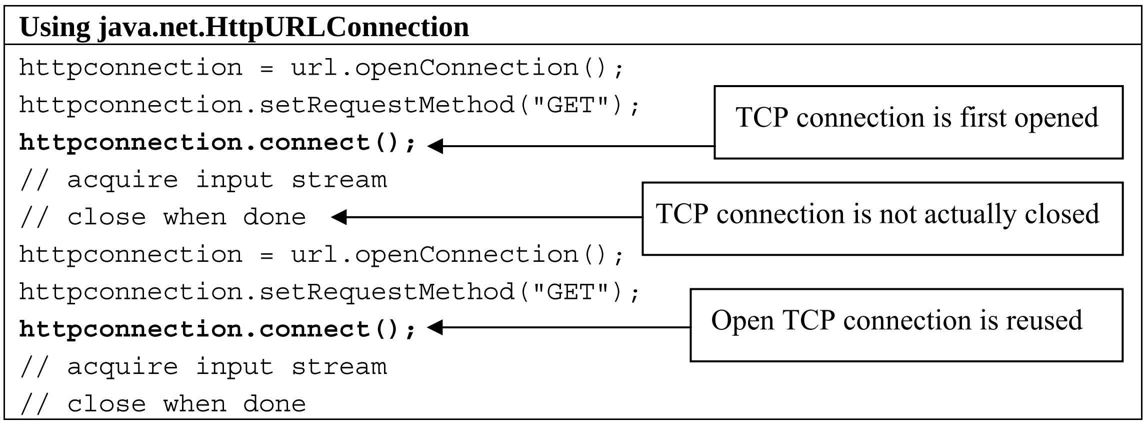
The graph represents a comparison between the total HTTP induced overhead and the remaining overhead if the connection is made persistent (see Section 4.5.2). The average BANip packetizer setting (140 BAN data units per group) generates content for HTTP requests of about 140 * 30 = 4200 bytes each. In Figure 5.1 the estimated improvement for content sizes of 4200 bytes is about 7%. Additional improvement is expected because TCP can more efficiently determine the congestion state of the network, but it is very difficult to predict how much. The measurement section of this chapter determines the collective improvement of this change.

The required change in the design is fairly straightforward. [RFC2616] dictates that HTTP 1.1 protocol entities should assume the other side wishes to maintain the connection until an explicit “Connection: close” header is received. The header should no longer be sent and the MBU and SH HTTP protocol entity should be verified to keep the connection open after this change.

5.3.2 Implementation

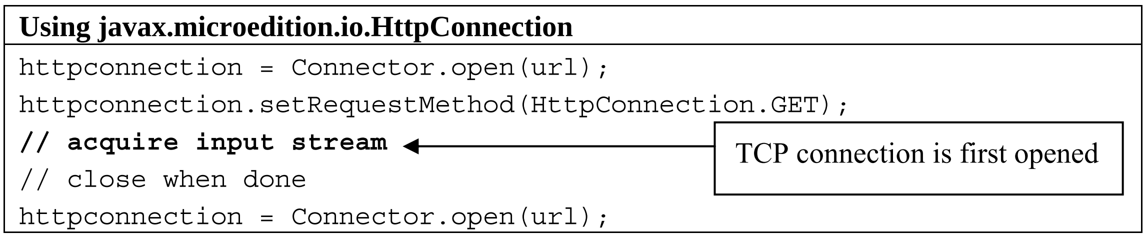
The reference MobiHealth implementation uses the SUN reference framework implementation for HTTP communication (*javax.microedition.io.HttpConnection)*. This implementation announces itself as a HTTP 1.1 implementation, so according to [RFC2616], it SHOULD implement support for persistent connections. SHOULD directives may be ignored if there “exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.” [RFC2119]. The java documentation accompanying the class does not mention any such reasons, yet MobiHealth engineers reported the inability to get the class to maintain a persistent connection. The following research provides the definitive answer, namely that the option is not supported by the SUN reference framework implementation for HTTP communication.

The class *HttpURLConnection* from java.net is used as a reference of an implementation that properly supports persistent connections. The following table shows how persistent connections are used in this class:


Using java.net.HttpURLConnection

```
httpconnection = url.openConnection();
httpconnection.setRequestMethod("GET");
httpconnection.connect();
// acquire input stream
// close when done
httpconnection = url.openConnection();
httpconnection.setRequestMethod("GET");
httpconnection.connect();
// acquire input stream
// close when done
```
TCP connection is first opened
TCP connection is not actually closed
Open TCP connection is reused

The HttpConnection class from the SUN reference framework has a slightly different way of being invoked, but the structure strongly resembles that of *HttpURLConnection*:


Using javax.microedition.io.HttpConnection

```
httpconnection = Connector.open(url);
httpconnection.setRequestMethod(HttpConnection.GET);
// acquire input stream
// close when done
httpconnection = Connector.open(url);
```

```
httpconnection.setRequestMethod(HttpConnection.GET);
// acquire input stream ◄──────────── New TCP connection is used!
// close when done
```

The preliminary conclusion is that the (microedition)HttpConnection does not support persistent connections as would be expected from the way the (java.net)HttpURLConnection class works.

The documentation for both implementations provides the definitive answer.

Class	**Documentation relevant to persistent connections**
HttpURLConnection	Each HttpURLConnection instance is used to make a single request but the underlying network connection to the HTTP server may be transparently shared by other instances.
HttpConnection	HTTP is a request-response protocol in which the parameters of request must be set before the request is sent. The connection exists in one of two states: 1. **Setup**, in which the connection has not been made to the server. 2. **Connected**, in which the connection has been made, request parameters have been sent and the response is expected. The following methods may be invoked only in the **Setup** state: • setRequestMethod • setRequestProperty

According to the documentation, once the connected state is entered, no new request parameters may be passed to the class. This means that it is impossible for an application to do a GET of some URL, and then a GET of a different URL using one and the same (microedition)HttpConnection. Because the practical test verifies that using a new HttpConnection (even to GET the same URL) sets up a new TCP connection rather than reusing the already active connection, there appears to be no way to send a new request over the same TCP connection. Thus, the implementation of the (microedition)HttpConnection offers no support for persistent connections and is thus conditionally HTTP 1.1 compliant (rather than unconditionally compliant).

Possible ways to get support for persistent connections are:

1. Replace (microedition)HttpConnection with a different HTTP class that does have support for persistent connections

2. Replace HttpConnection with a custom TCP based implementation that implements required behavior for the BANip

3. Modify the implementation of HttpConnection itself

While option 2 offers a lot of controllability, it is also a lot of work, and not to mention a lot of "reinventing the wheel". At the point where this decision needed to be made, there were a lot of uncertainties regarding which classes were "allowed" in the MobiHealth framework, e.g. the availability of java.net on target MobiHealth platforms. This means that while option

1 probably offers the most readily usable solution, there is a risk that it cannot be used as a permanent solution. An advantage of option 3 is that it requires the least amount of changes it the complex MobiHealth framework code. It only involves adjusting the connection variables so that they are reused, which is less work than finding and replacing every instance of the class with a different class and. Different method names and parameters would present further work in the MobiHealth framework code if option 1 is chosen. Hence, option 3 is chosen.

The implementation of the reference SUN/Java CDC (microedition) implementation of the HTTP protocol resides within the Generic Connection Framework (GCF) [GCF03]. The GCF is basically a Java 2 microedition application programmer interface (J2ME API) that provides broad support for connection types. Whenever a user application tries to open a URL (through http://), the Connector class reroutes this call to a Protocol.java file that implements HTTP support and takes care of the request. This file, included as “cdc/io/j2me/http/Protocol.java” in the MobiHealth source code tree, is the file that is modified to support persistent connections.

The most important change is the introduction of a connection state variable and the removal of the limitation that request parameters may only be specified once for the HttpConnection. The state variable can only be in one of two different states:

1. The request parameters and payload must be set

2. The request has been sent, and a reply must be retrieved

After the reply is retrieved in state 2, the state changes back to state 1. If the connection is lost, a new HttpConnection class instance must be created by the user application, in this case MobiHealth framework source code, to establish a new connection.

Other changes involve state checks, i.e. which methods can be called in which state, and some bug fixes:

- removed superfluous carrier return + line feed (CRLF) at the end of a request in accordance with the HTTP 1.1 specification

- fixed connection open/close reference counters going negative

The MobiHealth MBU source code that uses the HttpConnection class can be adapted fairly easily to use the improved functionality. Where previously a new HttpConnection was forcibly recreated, a single connection variable now suffices. No other changes were required.

5.3.3 Performance Measurement Strategy

The HTTP layer improvements offered by persistent connections are largely overlapped by the chunked transfer codings improvements in the next section (Section 5.4). That is, when the MBU uses a chunked transfer coding, it automatically benefits from improvements that a persistent connection offers, because the first request (and with it, the connection) remains active for as long as the MBU wishes.

Support for persistent connections was also implemented to use for BANip connections that require a two-way communication channel, i.e. connections for which the chunked coding improvement would be functionally undesirable. With this in mind, persistent connection measurements only focus on the achievable throughput. The BAN data unit transfers should make use of the far more improved chunked transfer coding rather than just a persistent connection.

5.3.4 Performance Measurement Results

The results of the persistent connection measurements are displayed in the table below. Each measurement involves transferring a total of 30 requests or 30 * 6000 = 175 KB of BAN data. The packetizer is set to 200 BAN data units per group (6000 bytes). The measurements are conducted with the actual (modified) MobiHealth framework implementation and improved HttpConnection class.

Test #	**Average payload throughput (Bps)**
1	1264
2	1118
3	1035
4	1224
5	1215
6	1182
Avg.	**1173 Bps**

5.4 HTTP: Chunked Transfer Coding

This section discusses the design, implementation and measured performance improvement of using the HTTP chunked transfer coding extension.

5.4.1 Design

Design summary			
Performance drawback	**Solution**	**Estimated improvement**	**Status**
Wait for response after every request	Use chunked transfer coding and use second HTTP session for SH to MBU data	~98%	DI

The request-response paradigm represents a performance drawback that is not resolved by making the connection persistent. After transmitting a full HTTP request, the sender must go idle and wait for the request to arrive at the receiver, and for a response to arrive back. With an average BANip packetizer setting of 140 BAN data units per group, the MBU-SH data transfer is interrupted by an idle period of at least one round-trip time every 4200 bytes. Moreover, when the transfer resumes after every idle period it may incur two additional penalties: about 700 ms for the GPRS device to reacquire uplink data transfer slots (see Section 3.3.1) and a reduced throughput for about 3 more seconds before the TCP congestion window grows large enough again to allow full utilization of the uplink channel (see Section 4.4.2).

A possible solution for these constant interruptions is to use very large HTTP request content lengths, because interruptions occur only in between requests. In addition to that, the calculations in Section 4.5.1 and Section 4.5.2 show that the protocol header overhead caused by HTTP is relatively small when large request content lengths[44] are used. Indeed, the transfer of the request content itself does not incur any HTTP overhead because it is passed on directly to the underlying TCP protocol entity, so larger request contents result in a proportionally smaller total HTTP overhead, which improves throughput. However, there are some important arguments against using this method:

- The end-to-end delay of BAN data units in the content increases along with the length of the content in the HTTP request. This occurs because technically, the sender HTTP protocol entity must first determine the content length before it can transmit the content. In other words, all content must be buffered before the HTTP request can be sent so that each BAN data unit is delayed until the last unit of the total content arrives.

- Assuming the implementation of the sender HTTP protocol entity can be modified, it is possible to "hack" the sender so that it specifies a near-infinite content length and transmits a stream of partial content to avoid increasing the BAN data unit end-to-end delay. Although possible, this is not desirable behavior for a number of reasons. First of all, the sender would be specifying that it has data that does not even exist yet. The solution is also not very portable, as it would need to be reimplemented on other MBU platforms. Furthermore, since the HTTP 1.1 standard [RFC2616] does not specify a

[44] around 6% overhead with content lengths of 6000 bytes and higher

requirement for the receiving (or proxy) HTTP entities to deliver any content at all before the full request content length has been received. This means that even with a modified sender implementation, the receiver is still at the mercy of the unspecified buffering behavior of the WSB proxy (see Section 2.4) and receiving HTTP protocol entity.

A better solution is to use a *chunked transfer coding* [RFC2616]. Conceptually, this technique is similar to setting an infinite content length. The major difference is that the sender does not (and should not) specify the content length in the request. Rather, the sender segments data in so-called chunks which each contain a small[45] header specifying their length. This technique allows the sender to continue transmitting data chunks indefinitely without having to wait for receiver responses, because the receiver may only send a response after the sender specifically halts the stream of chunks with a zero size chunk (because only then the HTTP request is finished).

The improved design is as follows:

- The sender BANip protocol entity should have a way to indicate to the HTTP protocol entity that it wishes to use a chunked transfer coding, e.g. by requesting an output stream without specifying a content length.

- The sender BANip protocol entity should also be aware of how chunks must be passed to the HTTP protocol entity, e.g. by flushing an output stream.

- For each chunk a small overhead occurs (see Section 4.5.3), so it is wise to not use chunks that are very small. The overhead ranges from 12% down to 1% for chunks containing groups of 2 – 24 BAN data units. It is probably not very useful to use chunk sizes smaller than the packetizer group setting either, since the receiving BANip protocol entity will not deliver any BAN data units before the whole group has arrived.

- The MBU can send chunks indefinitely without having to wait for any reply, and may choose to cleanly end the “stream of chunks” by specifying a zero size chunk.

N.B.: it is conceivable that some implementations will attempt to buffer chunks. However, because it is more natural to buffer in terms of single chunks since the amount of chunks that will follow from the sender is unknown. So it is probably safe to assume that good implementations will refrain from buffering multiple chunks or offer a way to turn the chunk buffering behavior off.
The chunked coding technique causes the transfer of BAN data through HTTP to resemble an ordinary (one way) TCP data transfer. A rough estimate for the throughput improvement is therefore the difference between the maximum throughput achieved with TCP (see Section 4.4.3) and the reference BANip TC (see Section 3.4.1). With the default TCP send buffer size setting, the estimated throughput improvement is (1430 – 720) / 720 = 98%.

This improvement can be used in conjunction with persistent connections, but because the HTTP response phase is theoretically never reached (unless the MBU must stop transmitting data altogether), the connection is never closed whether it is persistent or not. So, technically, using persistent connections does not increase the percentage of throughput improvement any

[45] The per-chunk overhead is less than 1% if the chunk size exceeds 720 bytes, see Section 4.5.3

further with this solution.

Finally, the improved design does introduce a functional drawback. Because the SH can no longer send regular responses to the MBU, it can also not piggyback any data back to the MBU. A second HTTP connection easily compensates this drawback, and if SH to MBU chunking is used the additional load on the MBU uplink channel is negligible.

5.4.2 Implementation

The HTTP protocol implementation used in the MobiHealth framework, javax.microedition.io.HttpConnection, is already described in Section 5.3.2. The class supports receiving chunked transfer codings from a server but unfortunately does not support client side chunked transfer codings.

The set of modifications necessary to improve the interconnect protocol with chunked transfer codings is twofold. First of all, the HTTP protocol entity must implement support for this type of coding. Secondly, the MobiHealth BAN and SH source code must be modified to not only make use of this transfer coding, but also to change any mechanism that used the reverse (SH to MBU) data flow direction.

The second part is determined to be out of scope for this thesis. Rather, the likely improvement is demonstrated by means of a relatively simple proof of concept implementation, consisting of a custom sender and receiver. The sender and receiver call upon the modified (microedition)HttpConnection class that was already modified to support persistent connections and is now improved to support chunked transfer codings.

The proof of concept sender must be able to inform the HttpConnection class that it requires the “Transfer coding: chunked” header. The sender should not need any special purpose functions to specify chunk lengths. This is reflected in the implementation by using the existing flush() method as the signal that the currently queued payload should be sent as a chunk. The final chunk in a series of chunks must be indicated by a zero length chunk. Technically, this can be accomplished by not queuing any data in between two calls to flush().

A third state is added to the existing two HttpConnection states so that the class can determine what it should send. The third state is an intermediate state in which the request headers have already been sent, but no zero length chunk has been specified by the sender yet. After the last chunk has been sent, HttpConnection proceeds to the next state, in which a reply must be read.

Various other changes to HttpConnection include fixes to chunked reading. For example, if the receiver of a chunked request or response calls the method available(), it now automatically reads the size of the next chunk rather than returning 0 bytes available. Consequently, 0 bytes available is only returned when the last chunk has been read.

The HttpConnection class is now suitable for using chunked transfer codings from both client to server and server to client.

5.4.3 Performance Measurement Strategy

The average TC and BDL of chunked BAN data unit transfers are determined for a comparison with the reference BANip measurements. The strategy to measure these two CPIs is equal to the strategy used for the reference measurements (see Section 3.2).

5.4.4 Performance Measurement Results

According to the strategy for measuring TC and BDL, the TC is first determined. As mentioned before, the measurement effort is focused mainly around a packetizer setting of 140 BAN data units / group. The measurements are made with a proof of concept client and server that emulate the MobiHealth BAN interconnect protocol.

Test Number	**Transfer Capacity (Bytes per second)**
	PKT = 140
1	1784
2	1738
3	1579
4	1684
5	1656
6	1819
Test average	**1710 Bps**

According to the determined TC, high load (~90% TC) and medium load (~50% TC) are defined as follows (PKT = 140):

- **High load:** 1600 Bps
- **Medium load:** 900 Bps

For each of these load settings, BDL values are determined from BAN data units generated at that rate. Because the blocking problem that is described in Section 3.4.2 also occurs in the proof of concept system, the same solution is used to correct the BDL measurements. Rather than time-stamping the BAN data units real-time, the units are time-stamped relatively to time index 0 at the specific rate they would normally be generated at. During the measurement, the proof of concept client executes its regular rate control function, but rather than using the time of wakeup to time-stamp the BAN data units, the timestamps are read from the prepared file. This solves the problem of the rate control function being woken up many seconds too late and consequently bursting timestamps. Using the timestamps from the file produces a smooth generation time graph, as required for correct measurements.

Test #	**BAN Data Latency: average (max. / min.) seconds**	
	PKT = 140	
	1600 Bps	**900 Bps**
1	8.2 (13.1 / 3.8)	7.8 (13.0 / 3.9)
2	11.7 (18.2 / 4.9)	6.3 (9.6 / 3.8)
3	8.2 (11.7 / 4.8)	6.5 (10.5 / 3.9)
4	7.2 (10.4 / 3.9)	6.4 (9.4 / 3.8)
5	8.2 (11.1 / 4.6)	6.4 (9.2 / 3.9)
6	6.6 (9.2 / 3.8)	7.1 (11.1 / 3.9)
Test average	**8.5 (12.5 / 4.5)**	**7.0 (10.5 / 4.0)**

N.B. The test averages were rounded to the nearest 0.5 sec. to keep the accuracy in line with the scale and significance of BDL values.

5.5 Data Compression

This section discusses the design, implementation and measured performance improvement of using data compression.

5.5.1 Design

This performance improvement does not resolve any specific performance drawbacks, so the entry is not present in the table.

Design summary		
Improvement	**Estimated improvement**	**Status**
End-to-end throughput increase or end-to-end latency decrease due to a reduction in the volume of data that needs to be transferred over the network	~33%	DI

Data compression is a technique commonly used to reduce the volume of the data that needs to be transported between two end points. It takes less time to transfer smaller volumes of data, so a performance improvement can be expected either with respect to the TC (more data per second with the same link utilization) or BDL (smaller transfer time for the same amount of data).

There are number of general considerations for how and when to apply data compression:

- In general, the larger the blocks of data that are compressed at the same time, the more efficient an algorithm can compress the data. This principle is used for instance by the RAR compression algorithm. It has a “solid archive” option that allows it to group all files into one large chunk of data to be compressed, rather than separately having to compress each file. Traditionally, compression on a Linux operating system is also achieved by first grouping a set of files into one so-called tarball, which is then processed by the GZip compression algorithm.

- Also, the more that is known about the type of data to be compressed, the better the compression results potentially are. RAR again uses this principle through an option named “multimedia compression”, which instructs it to use a specifically optimized algorithm for compressing files with multimedia content.

- The advantages of compressing data (in a lossless way) that is already compressed are more often than not outweighed by the cost. In some cases, this kind of compression strategy actually results in an increase in the data volume.

Going by every one of these considerations, it is a much better idea to build a compression component for the BANip layer than to use a very general compression algorithm in the GPRS protocol stack[46]. The BANip controls the largest groups of data and is most aware of the type of the data. Thus, the best compression results can probably be achieved there.

[46] Sections 4.3.2 and 4.3.3 show that this general compression algorithm is not used in the reference BANip

The improved design features a new compression/decompression component in the BANip transcoding chain (see Section 2.5). For optimal results, the compression part of the component is inserted as the last component in the chain so that it processes BAN data unit groups outputted by the packetizer rather than individual BAN data units. Naturally, compression results will be limited at lower packetizer settings.

The periodic character of most of the sensor data should provide good compression results and thus reduce required bandwidth. The following example provides a guess of the improvement that compression could achieve. The GZip compression module built into the web server Apache has been reported to speed up downloads for slow connections (28k8 / 14k4 modems) by 30-35% for mixed content [ApGZip]. The mixed content consisted of text and images, which could perhaps be compared to the output from trivial and more advanced sensors. Above all, it is an encouraging estimate for implementing compression.

The only concern regarding compression is that it might significantly load the central processing unit (CPU) of the MBU so that its battery power will be exhausted much sooner. Fortunately this is not the case. In practice, the total CPU load remains below about 10% even with the compression algorithm set to “maximum compression”.

5.5.2 Implementation

The data compression / decompression component can be added to the existing BANip transcoding chain without any modifications to the MobiHealth source code. Once the component is built and specified in the SH configuration file (madison.prop), the Encoder (compression) part is automatically inserted in the MBU transcoding chain, while the SH gets the Decoder (decompression) part in its transcoding chain.

To separate the data compression component from other transcoding chain components, the two parts are named DeflateEncoder and DeflateDecoder. They can be built fairly simple with the use of the package java.util.zip.Deflater. Note that Zip is not used because it is basically a deflater with extra overhead for supporting file archives. The data that the DeflateEncoder receives in every update from the previous component in the chain (in this case the PacketEncoder) is compressed and passed on. Once transferred over HTTP to the SH, the DeflateDecoder reverses the Deflate compression (Inflates) and delivers the data to the rest of the Decoder chain.

A final remark; the DeflateDecoder also generates statistics regarding achieved compression ratios so that the compression achieved for each individual group of BAN data units can be determined, as well as the average compression ratio.

5.5.3 Performance Measurement Strategy

The critical performance indicator of a data compression algorithm is the volume reduction that the algorithm can accomplish with certain sets of data. Since the volume of the data to be transferred has a direct impact on the throughput, it acts as a sort of (inverted) throughput performance multiplier. The compression ratio that the new “Deflate” transcoder component can achieve when applied to actual sensor data is measured. If this ratio is 40% for example, meaning that 40% of the original data volume was “compressed away” (i.e. out of the 100% original data volume, only 60% remains after compression), then an additional 40% of data can be transferred in the exactly same time as the original set of data would have been transferred. As such, the performance improvement would be calculated as 40% on top of any other improvements.

Obviously, before the algorithm can be tested, a suitable collection of actual sensor data must be selected. Candidate files containing sensor data were taken from a BAN data repository (BDR) with a target size of about 100-300 KB. A selection of five files is listed in the table below. They are labeled M1 through M5 for easy reference.

Measurement mappings	
Datafile	**Source file**
M1	4G26DW34107C_924030002_start_13-43__13-5-2003_stop_13-50__13-5-2003
M2	4G22DW3480CD_123456_start_21-48__27-3-2003_stop_21-52__27-3-2003
M3	4G2ADW34N06H_924030003_start_16-24__28-3-2003_stop_16-29__28-3-2003
M4	4G26DW34107C_924030002_start_17-32__16-5-2003_stop_17-40__16-5-2003
M5	4G26DW34107C_924030002_start_20-51__16-6-2003_stop_20-55__16-6-2003

The contents of each file were visually inspected with the signal analysis software package Portilab [TMSI]. This was done to ensure that the total selection contains a wide variety of different sensor signals.

For each of the recordings, one or more channels[47] contained no sensor data. The table below lists, for every file, the total number of channels in the measurement as well as how many of those are significant (because they received sensor input). The static% is the percentage of channels that did not receive sensor data (but were fixed to a certain number, usually zero). Also listed are the size of every file, the duration in minutes:seconds of the measurement stored in the file, and what kind of sensors were attached to the significant channels.

Measurement descriptions			
Datafile	**Nr. of channels Total / Significant (Static%)**	**Size / Duration**	**Channel Sources (sensors)**
M1	6 / 1 (83%)	88 KB / 1:39	Sawtooth
M2	14 / 4 (71%)	388 KB / 3:26	Sawtooth; EMG; ECG; Press/flow
M3	10 / 5 (50%)	276 KB / 4:03	Sawtooth; Thorax; Pleth; HeartR; SaO_2
M4	6 / 3 (50%)	192 KB / 3:38	Sawtooth; ExG1; ExG2
M5	6 / 5 (17%)	149 KB / 2:48	Sawtooth; ExG1; ExG2; ExG3; ExG4

N.B.: As mentioned above, the data files were selected based on a visual inspection in order to ensure a mix of signals with different characteristics (static values, highly dynamic values, repetitive patterns). The exact type of the attached sensors and correlation with each other are out of scope; the object is to provide an educated guess of the average performance improvement to be expected from the Deflate component. Purely informational: a sawtooth is just that, a signal used to verify the continuity of the received data; HeartRate and SaO_2 are averaged values of the beats per minute (BPM) of the heart and oxygen saturation% of the blood; Pleth graphs display the pulse and ExG sensors measure electrical activity in the body.

[47] A channel in this context refers to a single sensor input port. Empty channels are the result of channel filtering or when there is simply no sensor attached to that specific port

The strategy for measuring the compression that the deflate component achieves with this data is as follows:

1. A dummy sensor is attached to the BANip. The dummy sensor reads the files and inputs BAN data units.

2. The data then proceeds through the transcoding chain of encoders and decoders, which causes the DeflateDecoder to output statistics about the compression of the data. These statistics are noted.

3. The *significant compression* is calculated from the noted statistics. The calculation method is explained below.

The purpose of the calculating significant compression is as follows. There are channels present in the recorded sensor data that are unused (always produce zero or a static value). Because this kind of data can be compressed extremely well, and because unused channels would most likely not even be transmitted during actual usage (filtered out), the initial compression statistics outputted by the DeflateDecoder are too optimistic.

The following calculation is used to eliminate most of the bias that the unused channels cause.

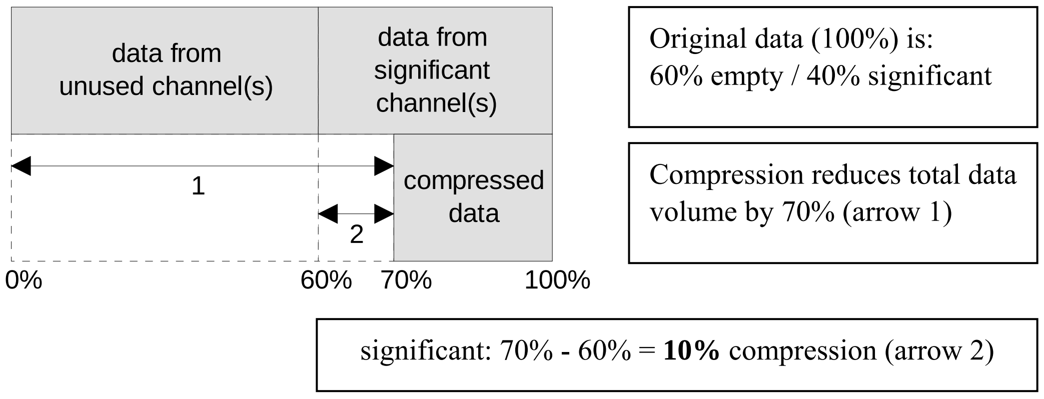

data from unused channel(s)
data from significant channel(s)
1
2
compressed data
0%
60%
70%
100%
Original data (100%) is: 60% empty / 40% significant
Compression reduces total data volume by 70% (arrow 1)
significant: 70% - 60% = **10%** compression (arrow 2)

Figure 5.2: Example of determining significant compression

Let the total size of recorded data from all channels be 100%. This amount is then compressed to a certain percentage. For example, if the size of the data after compression is only 30% of the original, the compression component will report an achieved compression of 100-30 = 70%. This number corresponds to the arrow labeled 1 in Figure 5.2.

The optimistic bias caused by the data from unused channels can be totally eliminated by shrinking the arrow labeled 1 by the percentage this data occupies in the original data, in this case 60%. This yields the arrow labeled 2 (70% - 60%). In a sense, arrow 2 pretends that only data from significant channels was offered for compression so that it only measures how much the significant data could be reduced in volume / compressed.

This is not totally fair to the compression algorithm because the unused channel data, even if it were a row of zeroes, could never be compressed 100%. So in practice, if all unused channels are filtered out before reaching the DeflateEncoder, the achieved compression will be slightly better than the significant compression calculated here.

Summarizing, a set of carefully selected data is used to provide a general insight into the effectivity of the Deflate data compression component. A full-scale test is beyond the scope of the thesis, but the test results should enable some general conclusions and predictions.

5.5.4 Performance Measurement Results

The results of the compression measurements are now presented. Each of the 5 selected test files is read by the "Dumy" component and inserted into the transcoding chain for transmission from MBU to Surrogate Host. The data proceeds through the transcoding chain in parts of 30 bytes (one BAN data unit). Every 135 parts are grouped by the packetizer in the transcoding chain, creating packets of 135*30 = 4050 Bytes each. These packets are then offered to the Deflate component for compression.

For every file, the average compression ratio is noted. After compression, the file size is reduced by the percentage listed in the column labeled "Average compression ratio". For example, 97% means that after compression the file size is 100-97 = 3% of the original. Significant compression is determined as described in the measurement strategy.

Compression test results			
Datafile	**Total KB processed**	**Average compression ratio**	**Significant compression ratio**
M1	90 KB	97%	82%
M2	401 KB	83%	41%
M3	282 KB	70%	40%
M4	196 KB	60%	20%
M5	151 KB	25%	10%
Avg.	**224 KB**	**67%**	**39%**

N.B.: Because the packetizer inserts a header in every packet, the total amount of bytes processed by the Deflate component is slightly larger than the actual file size.

According to the "significant compression" column, compression ratios of around 10% ~ 80% should be expected. This is a very large range that should be commented on.

The highest compression ratio (82%) was achieved on a sawtooth, i.e. a highly repetitive signal that is easy to compress, but not a typical signal for a biosignal sensor. So this compression result seems far too optimistic. On the other hand, a typical SaO_2 graph is much more repetitive than the sawtooth, which means that it would be even easier to compress. In this case, high compression ratios are no unrealistic expectation.

Looking at the instance where compression performs the worst, highly variable sensor data is recorded in four out of five significant channels present in the test (ExG1 through ExG4). This type of data is indeed difficult to compress, resulting in the measured low compression effectivity. This result warns that instead of very high compression ratios, some sensor configurations might also lead to very low compression ratios.

Files M2 and M3 contain recordings of a mixture of signals, both easy and difficult to compress. Interestingly enough, no two sensors in M2 and M3 are the same, yet their compression ratios are very close (41% and 40%). Additionally, the average of the results is also close to this (39%).

5.6 Summary and Comparison

This section summarizes and compares the CPI measurements of the improved BANip to the measurements of the reference BANip (see Chapter 3). The measured effects of individual improvements are also evaluated with respect to the estimations of their effect.

Figure 5.3 shows a comparison between the results of the reference and improved interconnect protocol CPI measurements. The main measurement effort is focused on the performance of the BANip with a packetizer setting of 140 BAN data units per group, so only these measurements are displayed. Lastly, the average packetizer holding delay is plotted. This is the delay an average BAN data unit incurred from having to wait in a BANip buffer for the rest of its group to be generated.

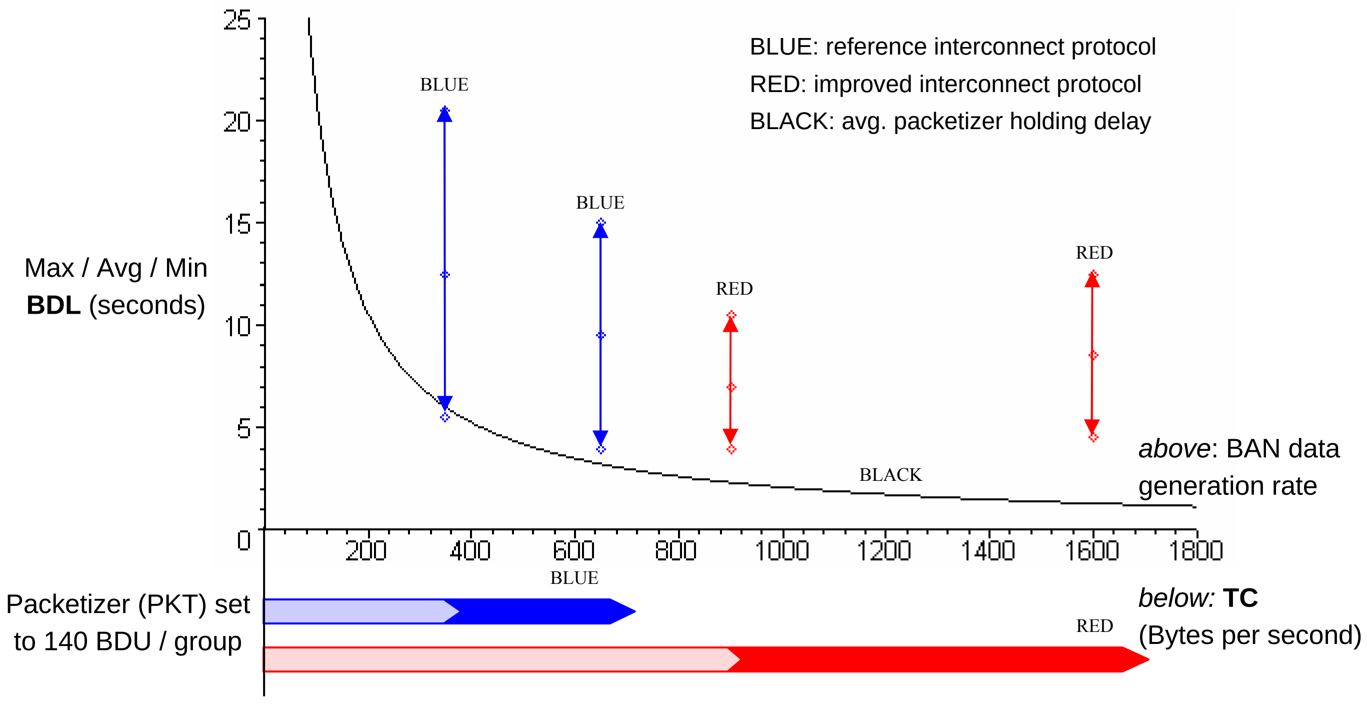


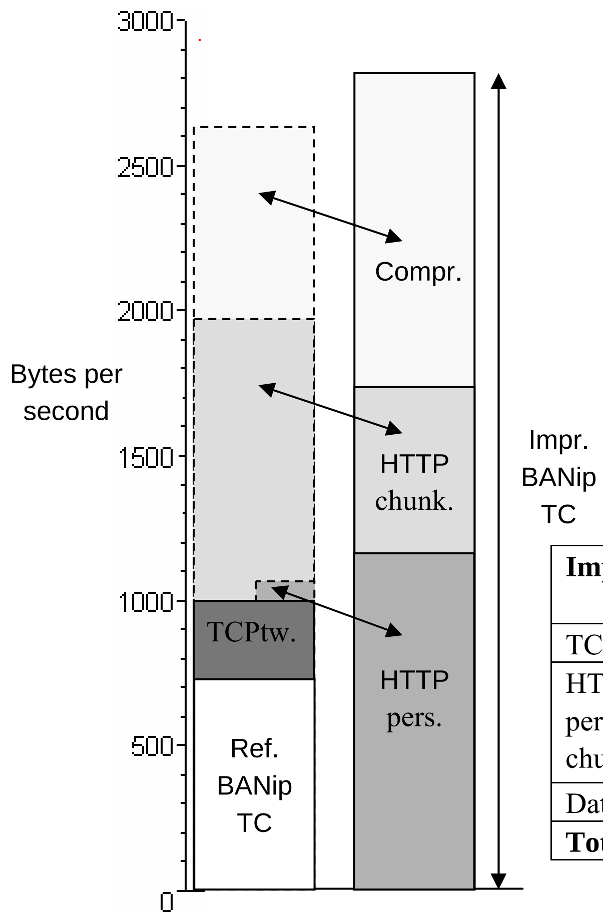
Figure 5.3: Reference and improved interconnect protocol CPI measurements

Starting with the transfer capacity, the red bar shows that the TC of the improved BANip more than doubles that of the reference BANip, achieving an almost 1000 bytes per second higher transfer rate, or a **138%** improvement. The lighter parts of the TC bars mark 50% (medium load). Note that the higher transfer rate does *not* include the improvements gained from compression, because compression can be applied in different ways that have a different impact on the BDL. This will be discussed further after Figure 5.3 is fully explained.

Each of the BDL measurements was executed at around 50% and 90% load of the TC of the respective BANip. For example, the leftmost blue and red arrow represent the results of the measurements at medium (50%) load. The average BDL of each of the results is visible as a circle near the center of every arrow. The average packetizer holding delay is directly part of this value. The leftmost arrow shows that on average, a BAN data unit sent through a medium loaded reference BANip has an average BDL of 12.5 seconds, out of which 8 seconds were spent in the packetizer buffer. Note that the minimum and maximum BDL should not be compared to the black (average packetizer holding delay)[48].

The conclusion from Figure 5.3 is that the improved interconnect protocol (without data compression) indeed possesses a significantly (138%) improved Transfer Capacity without any detrimental effects to the BAN Data Latency. On the contrary, BDL values at comparable load percentages are also slightly improved, even if it were just because the effect of the packetizer holding delay is slightly less noticeable at higher physical loads. The only effect that stands out in Figure 5.3 is the fact that the average BDL value of the 90% load measurements with the improved BANip is *higher* than the 50% load BDL average, in contrast to the reference BANip where this is reversed. The explanation for this result is that there was some time (months) between the analysis and measurements of the protocol stack and the CPI measurements of the final design of the improved interconnect protocol. During this time, it seems that the packet loss rate increased slightly (e.g. 3-5 losses per measurement rather than 1 or less). At a 50% load, the improved BANip may use the remaining 50% "overcapacity" to prevent a BAN data unit queue buildup. At 90% load, some queue buildup occurred, causing slightly increased BDL values. Nonetheless, these increased values are still lower than the average 90% load BDL values of the reference BANip.

A more in-depth summary of the improvements follows.



Improvement	TC gain	
	estimated	**measured**
TCP tweak (send buffer)	-	38%
HTTP: persistent connections (OR) chunked transfer coding	 7% 98%	 18% 72%
Data Compression	33%	64%
Total	**263%**	**289%**

Figure 5.4: Measured improvements vs. expectations (PKT = 140 BDU / group)

[48] The minimum BDL value is not affected by the packetizer holding delay because it originates from the last units that make up a group, while the maximum BDL should be compared to a graph of the maximum packetizer holding delay.

Figure 5.4 and the accompanying table show the Transfer Capacity of the reference and improved BANip, as well as the projected (estimated beforehand) and measured effect of all improvements. Projected improvements are drawn on the left, on top of the reference BANip, while measured improvements are represented in the right vertical bar.

Note that the estimated improvement of both HTTP persistent connections and the HTTP chunked transfer coding are calculated relative to the same base TC reference point, because the HTTP chunked transfer coding should already include the positive effects seen with persistent connections whether they are enabled or not. For this reason the square of the estimated improvement with HTTP persistent connections is drawn half as wide, so that the bottom of the HTTP chunked transfer coding improvement estimate touches the same base.

The estimated and measured improvements drawn in Figure 5.4 appear to be relatively close, except for the compression results. The fact that the improvement for HTTP persistent connections turned out to be slightly higher than 7% was expected, since the 7% was calculated only from the reduction in overhead in terms of volume. The chunked transfer coding performance gain is unfortunately somewhat less than expected. This is attributed to the packet loss increase on the reference network in the later stages of this research, as described above Figure 5.4. The higher compression TC gain is discussed below.

Figure 5.4 incorporates the data compression improvement (contrary to Figure 5.3) as a theoretical extension of the TC of the BANip. This raises an important subject for discussion, namely the new relationship between the Transfer Capacity, packetizer setting and effectivity of the compression algorithm. First, the calculation of the 64% TC gain is explained. In the improved BANip the per-group overhead has been significantly reduced. Consequently, the packetizer setting of a certain amount of BAN data units per group should have a significantly reduced impact on the Transfer Capacity. Assuming for a moment that the setting no longer limits the TC at all, the following calculation can be made:

- Let the Deflate component achieve a 39% compression ratio on blocks of N bytes so that the remaining volume of bytes is (100% - 39%) * N = 0.61 * N

- The remaining volume of bytes matches the TC exactly, i.e. 0.61 * N = TC

- The result is that N = TC / 0.61 = TC * 1.64

Technically, this means that the BANip with this Deflate component can process 64% of additional data without exceeding its measured TC. When the Deflate component is integrated in the BANip, the TC of the BANip would rise 64%. This number is the TC gain displayed in Figure 5.4 as a result of data compression.

However, the effectivity of the Deflate component is related to the value of N. In general, larger values for N yield better compression results. N in turn, is related to the packetizer setting, because the Deflate component operates on the output of the packetizer. Consequently, even though the original relationship of the packetizer setting with the Transfer Capacity may have diminished, a new relationship is created through the Deflate component, whereby higher packetizer settings once again are expected to lead to higher TC values.

The exact relationships between the effectiveness of data compression with the value of N and with certain configurations of sensors lie outside the scope of this thesis. For N = 4050 bytes, an average compression ratio of 39% was achieved with the output of a wide variety of sensor types, which equals a 64% TC gain. In general, the performance of the Deflate component is strongly related to the set of sensor types in a particular BAN.

Further measurements in this area were considered to be out of scope. A final remark on this topic is that the percentage of TC gain at certain compression ratios is independent of the height of the TC. Figure 5.5 shows a plot of the nature of the relationship between the compression ratio and TC gain. Note that for the 39% compression ratio mentioned above, the graph indeed displays 64% TC gain.

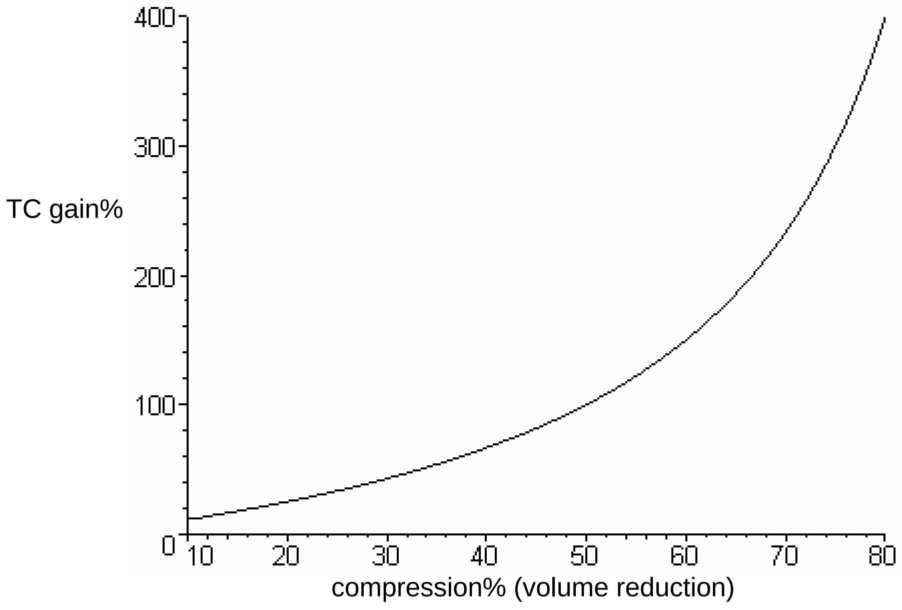


Figure 5.5: Relationship between compression and TC gain

The next section examines the BANip measurements in contrast with the capabilities of various layers to determine possible room for further improvements.

5.7 Remaining Room for Improvement

This section discusses remaining room for improvement in the following two ways:

1. Section 5.7.1 presents recommendations for dealing with remaining major performance drawbacks that were identified but could not be resolved in the reference system setup.

2. Additionally, Section 5.7.2 puts the BANip measurement results into perspective with theoretical and practical limitations to determine how much room there is for further improvement with respect to link capacity and utilization. Some remarkable findings are discussed that could potentially increase the TC of the BANip by another 66%.

5.7.1 Remaining Major Performance Drawbacks

This subsection discusses any remaining major performance drawbacks that were not resolved and the reason(s) why. For each of the unresolved performance drawbacks, measures are recommended that might reduce the impact of these drawbacks. In some cases it is possible to eliminate drawbacks with measures that fall outside of the scope of this thesis. The remaining performance drawbacks are listed and discussed below.

- **Unattainable 20% raw bandwidth**

IP performance measurements in Chapter 4 indicate clear limits for the achievable throughput of 81% and 78% for 1-slot and 2-slot CS-2, respectively.

Section 4.7 presents the theory that this limitation might be a result of the particular (O_2) GPRS operator network used, because the limitation does not seem to occur with a different (Vodafone) network. Note that these networks were fairly similar in terms of average latency (<1.3 sec) and low packet loss (<1%).

Switching to a different GPRS operator network could eliminate this drawback, but verifying this lies outside of the scope of this thesis.

- **Throughput degradation and connection stalls**

The problem described as a “GPRS hick-up” in [Yuch03] also seems to occur in the measurement environment used in this thesis. If the GPRS connection is “overloaded”, i.e. the achievable throughput is exceeded, the total average throughput may significantly deteriorate up to a point where the connection may “stall” entirely for an unknown[49] amount of time. Section 4.7 describes a manifestation of the throughput deterioration (20% less for consistently exceeding achievable throughput by 0.6%). Connection stalls also seemed to occur on at least half of the connections if no packet were transferred at all for a period of about 5 seconds. This time period was tested by generating traffic every X seconds and iteratively raising X from one second until the delay interval caused the connection to stall.

[49] The maximum time spent waiting for the connection to carry data again is about 5 minutes, without success.

Each time a stall occurred, the GPRS modem had to be disconnected and rebooted before reconnecting. In this light, it seems like the modem might have crashed somehow from the overload. However, [Yuch03] reports seeing the “hick-up” with different hardware in different networks too, which would suggest that it is not so much a problem of the hardware or network. The final hypothesis remains with the less “mysterious” initial theory of the GPRS modem experiencing buffering problems.

The 5-second stall could perhaps be explained as a network operator policy, although the stall does not occur consistently. Some connections lasted for dozens of minutes. Besides, GPRS is considered an “always on” technology as in the sense that the modem remains “always connected”, whether it has any data to transmit or not. The final hypothesis for this problem is that the GPRS operator network might require periodic link layer “LCP echoes” to confirm that the wireless connection is still intact, and that this policy is perhaps related to the time of day.

Fortunately, the effect of both drawbacks is already reduced in the reference MobiHealth BANip. The use of TCP, which uses careful bandwidth probing techniques, has a mitigating effect on connection stalls. Although the large send buffer size in the reference MobiHealth BANip TCP entity did seem to cause a slow reaction time of TCP to overload effects, no connection stalls were observed in tests that used TCP. Coincidentally, the BANip also transmits high level “keepalives” every 5 seconds to satisfy the liveness detection requirements of the Jini surrogate architecture. These keepalives would mitigate the possible “5+ second timeout” policy of the GPRS network operator, although timeouts could still occur between the activation of the GPRS connection and the user activation of the MBU software (includes the BANip). Support for link layer echoes is reported to have been implemented during the writing of this thesis, so this would be a good solution against connection stalls from timeouts.

- **Slot assignment fixation**

A very strange phenomenon regarding slot assignments occurred during measurements, whereby the assignment of a second slot could consistently be “forced” by letting the data transfer become idle for a second or two. Viewed another way, active uploads often remained “stuck” with the assignment of only 1 slot, but consistently got an extra slot right away if the short pause was initiated. Note that this might seems like a method for “stealing slots”, but consider that even in the deep night where the network load is not expected to limit slot the assignments, 1-slot assignments were quite common.

As argued in Section 4.7, the final hypothesis is that there might be a problem with the GPRS component in the sense that it fails to adapt to new slot requirements on the uplink for as long as the transmission part is busy with user data.

This means that possible solutions are to change or perhaps upgrade the GPRS hardware. The problem represents a significant issue. The approach taken in this thesis attempts to circumvent the issue by dividing measurements into a “consistently 1 slot available” and “consistently 2 slots available” class. Above TCP level this division becomes increasingly harder to make because HTTP introduces its own pauses in the transfer and slot assignments are not so easy to derive from throughput graphs. In summary, the strange effects of idle time on the slot negotiation introduce uncertainties in the analysis of performance drawbacks and evaluation of performance improvements, for which there is no clear single solution. The "best effort" solution is to group measurements with consistent results and judge the higher layer (HTTP, BANip) performance independently of actual slot assignments.

- **TCP initial window size**

The measured time required for a TCP flow to reach "full" 1-slot utilization starting from a new connection is 3 seconds on average. In the reference BANip, where every group of BAN data was sent over a new connection, this represents a major performance drawback.

While the usage of persistent connections in the improved BANip eliminates the constant TCP connection setup and teardown, the performance drawback still exists, though it is likely no longer significant. Still, if a way can be found to increase the initial TCP window size (specified in [RFC3390]), it might prove to be a slight performance improvement for the time to "full" utilization.

- **Extreme packetizer delays**

The delay of individual BAN data units increases exponentially as the load (measured in new BAN data units per second) on the BANip *decreases*.

This drawback is caused by the static nature of the BANip packetizer setting, i.e. it cannot be changed in run-time.

Though no solution to this problem was actually implemented in the BANip, this thesis recommends that a specific time limit be added to the packetizer to prevent it from buffering BAN data for an extraordinary amount of time. This time limit would be established based on the latency demands on the specific application of a BAN. Alternatively, the packetizer could scale the buffered group size according to the load (amount of new BAN data units measured every second) and the capability of the wireless connection. Though the latter solution is more difficult to implement, it would improve the BDL values over the whole load spectrum.

5.7.2 Measured Room for Improvement

Figure 5.6 shows results from the layered analysis (see Chapter 4) in combination with results from this chapter and Chapter 3, as well as some undocumented measurements that support the theory that there is still much room for improvement.

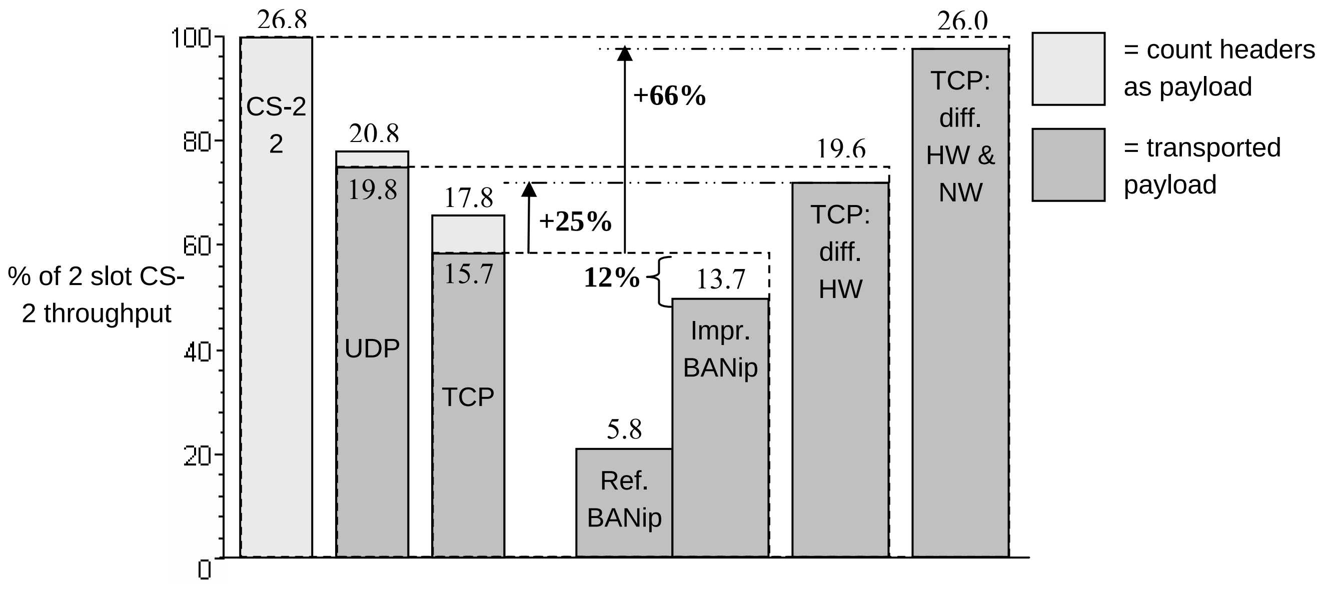


Figure 5.6: Remaining room for improvement

The leftmost bar shows the theoretical maximum throughput of a GPRS link that is assigned 2 slots (CS-2 coding). The other bars, representing various throughput measurements, cannot climb past this maximum. The dashed lines indicate these limitations. The numbers above or inside bars represent throughput in kilobits/sec.

Section 4.7 already concluded that the first two (UDP and TCP) measurements represent significant performance drawbacks of 22% and 14% respectively, which could not be resolved. The reference and improved BANip TC (without compression) cannot realistically exceed the TCP throughput bar, since both protocol versions implicitly employ TCP. In the improved BANip, the TCP connection is used much more efficiently. Further improvements may only hope to eliminate some 12% of inefficiency, resulting in a TC gain of about (15.7-13.7) / 13.7 = 15% of the current TC (without compression).

TCP measurements conducted with different GPRS hardware and a different GPRS operator network shed some light on the (remaining) UDP / TCP measured performance drawbacks. Section 4.7 describes that after substituting the iPAQ wireless pack with a Nokia 6310i phone, the TCP measured performance drawback is almost entirely eliminated. It is likely that the BANip would benefit directly from the 25% throughput gain that results from this GPRS hardware change. Similarly, measurements (not documented in this thesis) with the Nokia 6310i phone on a different GPRS operator network show that it is possible to resolve the last of the remaining significant throughput related performance drawbacks. The measured TCP throughput closely approaches the theoretical maximum, potentially increasing the TC of the BANip by 66%.

The table below summarizes the room for improvement in terms of Transfer Capacity.

Condition	**Room for improvement% (BANip TC)**
BANip inefficiency resolved	15%
GPRS hardware related drawback resolved	25%
Both GPRS hardware and GPRS network operator related drawbacks resolved	66%

6. Conclusions and Future Work

Section 6.1 presents the main achievements and conclusions of this thesis. Section 6.2 discusses potentially interesting subjects for future research.

6.1 Achievements

This thesis contains the results of research into the suspected poor performance of a Body Area Network interconnect protocol developed as part of the MobiHealth project. The objectives were to not only identify significant performance drawbacks, but also to resolve them in the actual implementation and verify the improvements in realistic (non-simulated) network conditions.

To achieve this, two so-called critical performance indicators were first defined to allow the performance of the BANip to be measured:

1. BAN Data Latency (BDL) – the end to end delay of the units of data the BANip transfers

2. Transfer Capacity (TC) – the maximum amount of data the BANip can transfer every second without building up a continuously growing queue

By devising and executing a set of black box measurements in a realistic network setup, the poor performance of the BANip was confirmed: the highest TC achieved by the BANip indicates that less than a third of the theoretical bandwidth of the available 2 slot GPRS connection was utilized, while average BDL values show that data units are delivered with 5 to 11 times more delay than the average GPRS link latency introduces.

The major challenge of overcoming a multitude of uncertainties and hidden dependencies was taken on by performing a comprehensive and systematic analysis of elements of the BANip protocol stack (including IP, TCP and HTTP protocol entities). The analysis includes a set of general IP and TCP layer performance measurement strategies developed and executed in order to help pinpoint performance drawbacks.

The analysis revealed a number of significant performance drawbacks responsible for the poor performance of the BANip. Subsequently, a set of improvements were designed and implemented in the complex MobiHealth system.

Performance measurements of the improved system show that the following results have been achieved:

- ➢ The uncovered performance drawbacks that lie within the scope of this thesis have been nearly entirely resolved: the improved BANip utilizes 88% of the (TCP) throughput that it can reasonably be expected to use

- ➢ The total performance gain in terms of the Transfer Capacity of the BANip is enhanced by data compression to approximately 4x the speed of the reference BANip

- The fact that data compression reduces the volume of data by 39% results in a similar amount of cost savings on the GPRS connection. The total amount of cost savings are further elevated by a reduced protocol overhead

- On average, the improved BANip also introduces roughly 30% less delay (BDL). Since the MBU-SH connection accounts for the majority of delay, the end-to-end delay of BAN to healthcare center has most likely been reduced by a similar amount.

Lastly, a set of identified performance drawbacks that lies beyond the scope of this thesis imposes a general limitation on the utilization of the wireless GPRS connection by the TCP protocol entity. Only 59% of the theoretical throughput could be achieved during TCP measurements. Since the BANip implicitly uses TCP, the limitation also significantly affects the BANip Transfer Capacity. Measurements from this thesis and related work[50] combined suggest that the achievable throughput can be increased to 97% by changing or upgrading the GPRS hardware and switching to a different GPRS provider. This represents a possible additional BANip TC improvement of 66%.

In conclusion, this thesis has identified and resolved significant performance drawbacks in a Body Area Network interconnect protocol, and contains recommendations to eliminate remaining performance drawbacks.

[50] Basically an FTP transfer test with a different GPRS provider and GPRS hardware

6.2 Future Work

There are a number of subjects that are potentially interesting for future work. On a short-term basis, the following subjects could be researched:

- The projections this thesis makes of significant improvements with different GPRS hardware and provider could be verified with measurements.

- In the improved BANip without compression, the setting of the packetizer should no longer have any significant impact on TC (if set to 24 units per group or more, the header overhead of using the chunked transfer coding is below 1%). The packetizer setting does still affect the BDL, so the exact BDL advantage of a low setting could be measured. This advantage could later be weighed against the advantages of high packetizer settings for the effectivity of data compression.

The following subjects were considered interesting yet with a higher degree of difficulty:

- The measurements in this thesis were executed out of necessity with a wireless device that remained relatively static. Since actual patients are likely to be more mobile, a set of new measurements could be executed with a moving device to uncover possible performance drawbacks that only occur in such an environment.

- The relationship between the effectiveness of compression and a given set of sensor types could be determined. The size of the blocks that are offered to the compression algorithm also matters in this equation.

- A Quality of Service scheme could be implemented in the BANip to increase user perceived performance. For example, if some points from a certain graph could be discarded without affecting the ability of a medical professional to correctly interpret it, then a highly loaded BANip (or a bad connection) could use the throughput normally required for these points for other data. A different example of QoS is that data could be prioritized. If data prioritization ensures that throughput is first allocated to very time-critical signals, then any remaining throughput can be used efficiently for data that is less time-critical. In this way, if the quality of the wireless link decreases, the decreased throughput will correctly affect the less-critical signals first.

References

This is the list of referenced books and documents. Additionally, if an online document was used the length (in pages) and web location (URL) are given.

Note: In case a URL becomes invalid, one possibility is to use the search engine Google[51] that keeps a cache of web documents. If the URL is cached, it might be possible to retrieve the document even if the original website is gone. Another possibility is to look for the document title through CiteSeer[52].

[ApGZip] John Giannandrea, Eric Bina; "*performance: HTTP Compression*"; 1998.
URL: www.mozilla.org/projects/apache/gzip/

[Chak02] Rajiv Chakravorty, Joel Cartwright, Ian Pratt; "*Practical experience with TCP over GPRS*"; University of Cambridge Computer Laboratory; 2002.
URL: www.cl.cam.ac.uk/users/rc277/gprs.html

[EN260] ETSI; "*Digital cellular telecommunications system (Phase 2+) (GSM); General Packet Radio Service (GPRS); Service description; Stage 1*"; GSM 02.60 version 7.5.0 (release 1998); 31 pages; August 2000.
URL: webapp.etsi.org/WorkProgram/Expert/QueryForm.asp
(work item reference RTS/TSGS-010260Q7R2)

[EN360] ETSI; "*Digital cellular telecommunications system (Phase 2+) (GSM); General Packet Radio Service (GPRS); Service description; Stage 2*"; GSM 03.60 version 7.4.1 (release 1998); 117 pages; September 2000.
URL: webapp.etsi.org/WorkProgram/Expert/QueryForm.asp
(work item reference REN/TSGS-020360Q7R2)

[EN460] ETSI; "*Digital cellular telecommunications system (Phase 2+) (GSM); General Packet Radio Service (GPRS); Mobile Station (MS) - Base Station System (BSS) interface; Radio Link Control/Medium Access Control (RLC/MAC) protocol*"; GSM 04.60 version 7.5.0 (release 1998); 216 pages; October 2000.
URL: webapp.etsi.org/WorkProgram/Expert/QueryForm.asp
(work item reference RTS/SMG-020460Q7R4)

[Fall96] Kevin Fall, Sally Floyd; "*Simulation-based comparisons of Tahoe, Reno, and SACK TCP*"; Lawrence Berkeley National Laboratory; 1996.
URL: aciri.org/floyd/papers/sacks.ps.Z

[GCF03] C. Enrique Ortiz; "*The Generic Connection Framework*"; August 2003.
URL: wireless.java.sun.com/midp/articles/genericframework/

[51] Google (www.google.com) was founded in 1998 by Larry Page and Sergey Brin, two Stanford Ph.D. candidates, who developed a technologically advanced method for finding information on the Internet.
[52] CiteSeer (citeseer.nj.nec.com/cs) is a digital library for scientific literature.

[Jain02] Manish Jain, Constantinos Dovrolis; "*End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput*"; appeared at ACM SIGCOMM; 14 pages; August 2002.
URL: www.cc.gatech.edu/fac/Constantinos.Dovrolis/Papers/sigcomm02.ps.gz

[JiniArch] Sun Microsystems;*"Jini™ Architecture Specification"*; version 1.2; (online specification, amount of pages not available); December 2001.
URL: java.sun.com/products/jini/

[JiniSA] Sun Microsystems;*"Jini™ Technology Surrogate Architecture Specification"*; version 1.0DraftStandard; 34 pages; July 2001.
URL: surrogate.jini.org/sa.pdf

[Mwp302] Rafael Gómez Bertrand; *"WP3 meeting: 2.5 - 3G Communication Infrastructure"*; Internal report of the MobiHealth Work Package 3 group; 27 pages; September 2002.
URL: www.mobihealth.org/html/details/workpackages/wp3/WP3meeting.ppt

[NTOOL] Network Measurements Working Group; "*A Hierarchy of Network Measurement Tool Properties*"; 2003.
URL: www-didc.lbl.gov/NMWG/NM-WG-tools.html

[O2LSpec] O2 technologies; "*GPRS latency factors*"; Table illustrating breakdown of round-trip latency; 2003.
URL:www.sourceo2.com/O2_Developers/O2_technologies/GPRS/Technical_overview/gprs_latency_factors_diagram.htm

[Tane96] Andrew Tanenbaum; "*Computer Networks*" *(3rd ed.)*; Prentice-Hall; New Jersey (USA); ISBN 0-13-394248-1; 1996.

[TCPWW] Saverio Mascolo et al.; "*Westwood+ TCP*"; Politecnico di Bari.
URL: www-ictserv.poliba.it/mascolo/tcp%20westwood/homeW.htm

[TMSI] Twente Medical Systems International; "*Software products overview*" (Portilab); 2003.
URL: www.tmsi.com/pages/s-specifications.htm

[Viss99] Chris Vissers et. al; *"The architectural design of distributed systems"*; Reader for The Design of Telematics Systems; University of Twente (Enschede, The Netherlands); January 1999.

[WAPv1] WAP Forum; "*Wireless Application Protocol*"; version 1.0 white paper; June 2000.
URL: www.wapforum.org/what/WAP_white_pages.pdf

[WAPv2] WAP Forum; "*Wireless Application Protocol: WAP 2.0*"; version 2.0 technical white paper; January 2002.
URL: www.wapforum.org/what/WAPWhite_Paper1.pdf

[WPTCP] LogicaCMG; "*memo regarding Wireless Profiled TCP (W-TCP) options and configuration*"; Based on WAP 2.0 Wireless Profiled TCP specification; 2003.

[WTCP] Karu Ratnam and Ibrahim Matta; "*WTCP: An Efficient Transmission Control Protocol for Networks with Wireless Links*"; The Illinois Mobile Environments Laboratory (TIMELY).
URL: timely.crhc.uiuc.edu/Projects/wtcp/wtcp.html

[Yuch03] Alex Yu Chen, Richard Bults, Katarzyna Wac; "*Performance analysis of TCP and UDP protocols over GPRS Network*"; MobiHealth Project WP3; 2003.

References - Internet Drafts

- *General*

[RFC2119] S. Bradner; "*Key words for use in RFCs to Indicate Requirement Levels*"; IETF RFC 2119 (Best Current Practice); 3 pages; March 1997.
URL: www.ietf.org/rfc/rfc2119.txt

- *Internet Protocol (IP) related*

[RFC791] J. Postel; "Internet Protocol"; IETF RFC 791 (Standard); 45 pages; September 1981.
URL: www.ietf.org/rfc/rfc791.txt

[RFC1191] J.C. Mogul, S.E. Deering; "*Path MTU discovery*"; IETF RFC 1191 (Draft Standard); 19 pages; November 1990.
URL: www.ietf.org/rfc/rfc1191.txt

[RFC1981] J. McCann, S. Deering, J. Mogul; "*Path MTU Discovery for IP version 6*"; IETF RFC 1981 (Proposed Standard); 15 pages; August 1996.
URL: www.ietf.org/rfc/rfc1981.txt

[RFC3168] K. Ramakrishnan, S. Floyd, D. Black; "*The Addition of Explicit Congestion Notification (ECN) to IP*"; IETF RFC 3168 (Proposed Standard); 63 pages; September 2001.
URL: www.ietf.org/rfc/rfc3168.txt

- *Transmission Control Protocol (TCP) related*

[RFC793] J. Postel; "*Transmission Control Protocol Protocol*"; IETF RFC 793 (Standard); 85 pages; September 1981.
URL: www.ietf.org/rfc/rfc793.txt

[RFC896] J. Nagle; "*Congestion Control in IP/TCP Internetworks*"; IETF RFC 896 (status unknown); 8 pages; January 1984.
URL: www.ietf.org/rfc/rfc896.txt

[RFC1122] R. Braden; "*Requirements for Internet Hosts -- Communication Layers*"; IETF RFC 1122 (Standard); 116 pages; October 1989.
URL: www.ietf.org/rfc/rfc1122.txt

[RFC1305] D.L. Mills; "*Network Time Protocol (Version 3) -- Specification, Implementation and Analysis*"; IETF RFC 1305 (Draft Standard); 98 pages; March 1992.
URL: www.ietf.org/rfc/rfc1305.txt

[RFC1323] V. Jacobson, R. Braden, D. Borman; ”*TCP Extensions for High Performance*”; IETF RFC 1323 (Proposed Standard); 37 pages; May 1992.
URL: www.ietf.org/rfc/rfc1323.txt

[RFC2018] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow; “*TCP Selective Acknowledgment Options*”; IETF RFC 2018 (Proposed Standard); 12 pages; October 1996.
URL: www.ietf.org/rfc/rfc2018.txt

[RFC2581] M. Allman, V. Paxson, W. Stevens; “*TCP Congestion Control*”; IETF RFC 2581 (Proposed Standard); 14 pages; April 1999.
URL: www.ietf.org/rfc/rfc2581.txt

[RFC2582] S. Floyd, T. Henderson; “*The NewReno Modification to TCP's Fast Recovery Algorithm*”; IETF RFC 2582 (Experimental); 12 pages; April 1999.
URL: www.ietf.org/rfc/rfc2582.txt

[RFC2680] G. Almes, S. Kalidindi, M. Zekauskas; “*A One-way Packet Loss Metric for IPPM*”; IETF RFC 2680 (Proposed Standard); 15 pages; September 1999.
URL: www.ietf.org/rfc/rfc2680.txt

[RFC2914] S. Floyd; “*Congestion Control Principles*”; IETF RFC 2914 (Best Current Practice); 17 pages; September 2000.
URL: www.ietf.org/rfc/rfc2914.txt

[RFC3042] M. Allman, H. Balakrishnan, S. Floyd; “*Enhancing TCP's Loss Recovery Using Limited Transmit*”; IETF RFC 3042 (Proposed Standard); 9 pages; January 2001.
URL: www.ietf.org/rfc/rfc3042.txt

[RFC3390] M. Allman, S. Floyd, C. Partridge; “*Increasing TCP's Initial Window*”; IETF RFC 3390 (Proposed Standard); 15 pages; October 2002.
URL: www.ietf.org/rfc/rfc3390.txt

- *Hypertext Transfer Protocol (HTTP) related*

[RFC1945] T. Berners-Lee, R. Fielding, H. Frystyk; “*Hypertext Transfer Protocol -- HTTP/1.0*”; IETF RFC 1945 (Informational); 60 pages; May 1996.
URL: www.ietf.org/rfc/rfc1945.txt

[RFC2068] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee; “*Hypertext Transfer Protocol -- HTTP/1.1*”; IETF RFC 2068 (Proposed Standard); 162 pages; January 1997.
URL: www.ietf.org/rfc/rfc2068.txt

[RFC2616] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee; “*Hypertext Transfer Protocol -- HTTP/1.1*”; IETF RFC 2616 (Draft Standard); 176 pages; June 1999.
URL: www.ietf.org/rfc/rfc2616.txt

- *Performance related*

[RFC2330] V. Paxson, G. Almes, J. Mahdavi, M. Mathis; "*Framework for IP Performance Metrics*"; IETF RFC 2330 (Informational); 40 pages; May 1998.
URL: www.ietf.org/rfc/rfc2330.txt

[RFC2678] J. Mahdavi, V. Paxson; "*IPPM Metrics for Measuring Connectivity*"; IETF RFC 2678 (Proposed Standard); 10 pages; September 1999.
URL: www.ietf.org/rfc/rfc2678.txt

[RFC2679] G. Almes, S. Kalidindi, M. Zekauskas; "*A One-way Delay Metric for IPPM*"; IETF RFC 2679 (Proposed Standard); 20 pages; September 1999.
URL: www.ietf.org/rfc/rfc2679.txt

[RFC2680] G. Almes, S. Kalidindi, M. Zekauskas; "*A One-way Packet Loss Metric for IPPM*"; IETF RFC 2680 (Proposed Standard); 15 pages; September 1999.
URL: www.ietf.org/rfc/rfc2680.txt

[RFC2681] G. Almes, S. Kalidindi, M. Zekauskas; "*A Round-trip Delay Metric for IPPM*"; IETF RFC 2681 (Proposed Standard); 20 pages; September 1999.
URL: www.ietf.org/rfc/rfc2681.txt

[RFC3148] M. Mathis, M. Allman; "*A Framework for Defining Empirical Bulk Transfer Capacity Metrics*"; IETF RFC 3148 (Informational); 16 pages July 2001.
URL: www.ietf.org/rfc/rfc3148.txt

[RFC3357] R. Koodli, R. Ravikanth; "*One-way Loss Pattern Sample Metrics*"; IETF RFC 3357 (Informational); 15 pages; August 2002.
URL: www.ietf.org/rfc/rfc3357.txt

[RFC3393] C. Demichelis, P. Chimento; "*IP Packet Delay Variation Metric for IPPM*"; IETF RFC 3393 (Proposed Standard); 21 pages; November 2002.
URL: www.ietf.org/rfc/rfc3393.txt

[RFCPRO] IETF Network Working Group; "*Packet Reordering Metric for IPPM*" (DRAFT)
URL: www.ietf.org/html.charters/ippm-charter.html

Appendix A: Test setup specifications

Software specifications	
Item	**Specification**
MBU (iPAQ) operating system	Familiar Linux kernel 2.4.18-rmk3
SH operating system	Microsoft Windows 2000 Service Pack 4
MobiHealth reference software version (CVS tag)	Release_2_0_1
MBU packet sniffer	tcpdump v3.6 using libpcap v0.6
SH packet sniffer	ethereal v0.9.11 using winpcap v3.01 alpha

Network hardware specifications	
Item	**Specification**
MBU GPRS card	Compaq Wireless Pack for GSM/GPRS; Firmware build version 776 Device is multislot class 10, CS1-4 support, Class B GPRS (simultaneous attach to GSM and GPRS, operation on one at a time), Type 1 GPRS (no simultaneous transmit and receive)
O_2 GPRS interface	Uses CS-1 or CS-2 so 9.05 - 13.40 kbps per slot 1 slot guaranteed, 2 slot policy unknown
SH connection to cable modem	100 Mbps PCI Network Interface Card (NIC) through 10 Mbps HUB (effectively a 10 Mbps connection from SH to cable modem)
COM21 cable modem	512 kbps downstream / 64 kbps upstream