UiO : Department of Informatics
University of Oslo

Efficient SOAP messaging for Android

Master's thesis

Dag Ove Eggum

2 May 2014

Abstract

The concept of Service Oriented Architecture, and its most common implementation method Web services, has not seen widespread use on wireless mobile systems and smart devices. NATO seeks to incorporate these communication standards, and military research and development groups aim to utilize Commercial Off-The-Shelf devices because of cost and versatility. Android is one of the largest open-source operating systems for smart devices, but lacks native support for the SOAP protocol. SOAP is the backbone protocol of Web services, but has a large overhead due to its XML structure.

This thesis expands the third-party SOAP library ksoap2-android with the possibilities of using different transport protocols other than HTTP/TCP and using compression to reduce the size of SOAP messages. The additional transport protocols are UDP and AMPQ, and the compression tools added are gzip and EXIficient, an XML-specific tool that implements the Efficient XML Interchange format. The expanded ksoap2-android library was used in a Web service client application installed on an unrooted Samsung Galaxy tablet using the 4.2.2 version of the Android operating system. The Web service client was tested against different Web services with different transport and compression combinations, using a proxy server to adapt the messages to a COTS server. The testing was done over both mobile broadband and Wi-Fi to examine the effects the different combinations had on CPU load and battery usage of the Android device, and the network load.

The testing showed that while EXIficient compressed slightly better than gzip, it caused a much greater CPU load and battery usage that gzip, causing the expenses to absorb the profits. Both UDP and the AMPQ implementation RabbitMQ performed better than HTTP, especially when focusing on achieving a higher goodput. This thesis concluded that using gzip together with RabbitMQ is the better option when it comes to reducing network overhead while simultaneously maximizing battery lifetime of reliable SOAP communication on an Android device.

Preface

This master thesis is written at the Department of Electrical Engineering of Bergen University College (HIB) in collaboration with the Department of Informatics of University of Oslo (UIO) in 2013/2014. The Norwegian Defence Research Establishment (FFI) provided the thesis topic.

I would like to use this opportunity to thank my academic supervisors at FFI, Dr. Frank Trethan Johnsen and Cand.Scient. Trude Hafsøe Bloebaum, for guidance, support and suggestions during the work with this thesis. Thanks also go to my supervisor at UIO, Prof. Dr. Josef Noll, and my supervisor at HIB, Prof. Dr. Knut Øvsthus.

Furthermore, I would like to thank Birthe Marie Roang for her proofreading effort.

Bergen, May 2014

Dag Ove Eggum

Table of Content

Abstract i

Preface iii

Table of Contents v

List of Abbreviations viii

1 Introduction 1

1.1 Motivation 2

1.2 Scenarios 2

1.3 Problem Statement 3

1.4 Scope and Technological Challenges 4

1.5 Research Method 5

1.6 Contribution 6

1.7 Outline of Remainder of Thesis 6

2 Background / State Of The Art 7

2.1 Android 7

2.1.1 An open-source mobile operating system 7

2.1.2 Android devices in military settings 9

2.1.3 Requirements specification 10

2.2 Web Services 11

2.3 Service Oriented Architecture 12

2.4 The SOAP Protocol 13

2.4.1 SOAP limitations 14

2.5 SOAP Optimizations: Compressing Data 15

2.5.1 General data compression 15

2.5.2 XML-specific compression. 16

2.5.3 Survey of comparisons of Binary XML solutions 18

2.6 SOAP Optimizations: Different Transport Method 19

2.6.1 Transport protocols for SOAP 19

2.6.2 Surveys of alternative transport protocols for SOAP 23

2.7 SOAP Library for Android 24
2.8 Using a Proxy Server 24
3 Design & Implementation 25
3.1 Optimizing 25
3.1.1 Compression techniques 25
3.1.2 Transport layer protocols 25
3.1.3 Proxy 26
3.2 Design 27
3.2.1 Main architecture 27
3.2.2 ksoap2-android 29
3.2.3 The Web service client 31
3.2.4 The proxy server 32
3.3 Implementation 34
3.3.1 Changes in ksoap2-android 34
4 Testing and Evaluation 43
4.1 Profiling for Android 43
4.1.1 Method profiling 44
4.1.2 Network traffic tool 45
4.2 Testing 47
4.2.1 Hello Web service 48
4.2.2 Upload NFFI data Web service 49
4.2.3 Exchange Picture Web service 50
4.3 Test Measurements 51
4.3.1 CPU load 51
4.3.2 Battery usage 51
4.3.3 Network load 51
4.4 Test Results 52
4.4.1 Test 1 & 2: Hello Web service 52
4.4.2 Test 3 and 4: Upload NFFI data Web service 53
4.4.3 Test 5 and 6: Exchange Picture Web service 54
4.4.4 Compression results 55

4.5 Evaluation 56
4.5.1 CPU load 56
4.5.2 Battery usage 58
4.5.3 Network load 59
4.5.4 Goodput 61
4.5.5 Comparing gzip and EXIficient compression 64
4.6 Summary 66
5 Conclusion and Future Work 67
5.1 Conclusion 67
5.2 Future Work 68
References 69
Appendix A – Attempt at enabling SCTP on Android 75

List of Abbreviations

Abbreviation	Meaning
ADT	Android Development Tools
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
APP	Application
ASN.1	Abstract Syntax Notation One
BSD	Berkeley Software Distribution
CEI	Collective Environment Interpretation
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
DARPA	Defence Advanced Research Projects Agency
DDMS	Dalvik Debug Monitor Server
ECN	Encoding Control Notation
EXI	Efficient XML Interchange
FFI	Norwegian Defence Research Establishment
GPS	Global Positioning System
HIB	Bergen University College
HTTP	Hypertext Transfer Protocol
HSPDA	High-Speed Downlink Packet Access
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IP	Internet Protocol
ISO	International Organization for Standardization
ITU-T	International Telecommunication Union-Telecommunication
JAXB	Java Architecture for XML Binding
JPEG	Joint Photographic Experts Group
MEP	Message Exchange Pattern
MIDP	Mobile Information Device Profile
NATO	North Atlantic Treaty Organization
NDK	Native Development Kit
NFFI	NATO Friendly Force Information
NIST	National Institute of Standards and Technology
OS	Operating System
PDF	Portable Document File
PNG	Portable Network Graphics
PNT	Position, Navigation and Timing
PPP	Point-to-Point Protocol
PSTN	Public Switched Telephone Network

R&D	Research and Development
REST	Representational State Transfer
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SCTP	Stream Control Transmission Protocol
SDK	Software Development Kit
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
STANAG	Standardization Agreement
TCP	Transmission Control Protocol
TransApps	Transformative Apps
UDP	User Datagram Protocol
UIO	University of Oslo
UMTS	Universal Mobile Telecommunications System
URL	Uniform Resource Locator
US	United States
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WBXML	WAP Binary XML
WSDL	Web Services Description Language
XML	Extensible Markup Language

1 INTRODUCTION

Service Oriented Architecture (SOA) [2] is a software architectural design pattern for constructing and deploying application functionality based on loosely coupled components (see Section 2.3 for further information on SOA). Web services [3] is a much-used technology for implementing applications based on a SOA, and achieving interoperability between different systems (see Section 2.3 for further information on Web services).

Web services are usually realized on computer systems where processing resources and network bandwidth[1] are not a limitation, and haven't been widely employed to mobile systems that are characterized by less computational resources (e.g. small computing devices and limited power), and wireless networks characteristics (e.g. low bandwidth, often ad hoc and unreliable).

Even though computing power and memory capacities of mobile devices are constantly improving, the dependency on battery power and wireless networks calls for improved solutions when implementing SOA on wireless systems. To interact with Web services, SOAP [4] messages are used (see Section 2.4 for further information on SOAP). SOAP relies on the Extensible Markup Language (XML) Information Set [5] for its message format. XML has a large information overhead, which is a challenge in the context of mobile devices.

Much research has been done and is still being conducted on how to enable Web services in the world of mobile devices, mainly on how to compress the messages being sent, but also looking at different ways of sending the messages. The topic is especially interesting for military use, since SOA and Web services are among open and accepted standards that should be used by the North Atlantic Treaty Organization (NATO) in the future [6].

[1] Bandwidth has several meanings, dependent on the context. This thesis uses bandwidth in the computer-networking context, where bandwidth is a measurement of data communication, expressed in bits, bytes or kilobytes per second, unlike in signal processing where bandwidth is the difference between upper and lower frequencies.

1.1 MOTIVATION

Today's smart devices are like Swiss Army Knives of the information age. They can be used as cell phones, video cameras, GPS trackers, sensors, and have the possibility to access the Internet using mobile or Wi-Fi [7] technology. Compared to traditional military communications equipment they are very light and handy, and easily carried by a single person. It is also of importance that unlike traditional military information systems, they are mass-produced and come at a reasonable price.

Android is an operating system (OS) for mobile platforms that has become very popular. Unlike other major mobile OS's like iOS and Windows Mobile, Android is an open source platform. Being open source is important for both the Web service community and the military research and development (R&D) community, in regards of information sharing and security considerations. In addition, many different smart device companies employ Android, meaning that Android platforms comes in a wide price range. These reasons make Android an interesting choice when exploring the possibilities of enabling Web services on a mobile device.

Recent trends within NATO seek to leverage the rapid pace of Commercial Off-The-Shelf (COTS) platforms for military use. A challenge in this context is the security aspect of using commercial platforms, both in regards to software and hardware. The United States (US) military has large resources, enabling them to review and modify commercial platforms in order to use them in accordance with their military security specifications. Smaller NATO countries, like Norway, does not have as much resources, but are also experimenting (for example with commercial software like Web services).

To reduce development time and cost to deploy systems, the goal is to make use of COTS components when possible, integrating custom and military-off-the-shelf components as necessary.

1.2 SCENARIOS

A military use-case scenario of using Web services on a mobile device could consist of using the device's sensors and mobility to report information back to a central or headquarter. As NATO is moving into network-centric warfare the need for information sharing is growing.

In Simanta et al. [8] a set of prototypes are described which demonstrate the use of SOA in tactical environments in which users are employing handheld devices to obtain situational awareness data. Situational awareness can be described as the level of knowledge of a given situation in regard of time and space, and how others and owns actions might affect it. In

military operations, more accurate and timely information enables decision makers to make better decisions. This information can could be text, pictures, video, map annotations, observations etc.

Another scenario would be to use the GPS receiver in devices to enable Blue Forces Tracking, a military term for providing military commanders and forces with location information about friendly (and sometimes hostile) military forces. In NATO, the colour blue typically denotes friendly forces. A Blue Force Tracking system could be as simple as a handheld device periodically sending location data back to base using Web services, not even requiring a response or acknowledgement.

NATO Friendly Force Information (NFFI) is a standard for Blue Forces Tracking, developed for use in Afghanistan. NFFI is an XML-based NATO standardized agreement, and consists of a message definition and message protocols [9]. Enabling Web services on a mobile device would enable sending standardized NFFI messages using COTS equipment, for example a smart phone.

1.3 Problem Statement

Web services have not seen widespread use on mobile devices. While being mobile has many advantages, it also has drawbacks like dependency on battery power and the need of a wireless network. Especially the battery power can be problematic, and solutions to provide more effective Web services (as reducing the size of exchanged data) can save power usage. Reducing the vendor cost of using privately owned mobile links is also an important aspect.

When developing an effective app[2] for mobile devices many aspects need consideration. In this thesis, the focus will be on how to most efficiently handle and send SOAP messages using compression techniques and various transport protocols, and isolate this perspective separate from other mobile device challenges.

The two main parameters to be examined are bandwidth usage and battery consumption. Bandwidth usage is important in a military setting, since good mobile coverage and high transmission rates are rarely the case in military operations. In addition, sending data over the mobile network often comes with an economic cost. Battery usage is equally important, since a soldier out in the field has to focus on other things than constantly recharging his device.

[2] App is short for application, often used in the context of mobile applications.

1.4 Scope and Technological Challenges

Many papers and theses on how to enable or optimize Web services on mobile platforms exist. In Johnsen et al. [10] potential gains from replacing the Hypertext Transfer Protocol (HTTP) [11] and the Transmission Control Protocol (TCP) [12] with alternative transport protocols for Web services in limited capacity networks were investigated, and in Johnsrud's thesis [13] different XML compression techniques were tested on a mobile device in order to reduce the size of SOAP messages.

The focus of this thesis is on how a combination of using different compression algorithms and a different transport protocol instead of HTTP/TCP may improve the battery life time and bandwidth usage of a mobile device employing Web services.

This thesis uses an Android tablet as mobile platform for Web services. One important reason for choosing Android is that it has become the dominant operating system for mobile smartphones (Figure 1.1), and is growing in the tablet market.

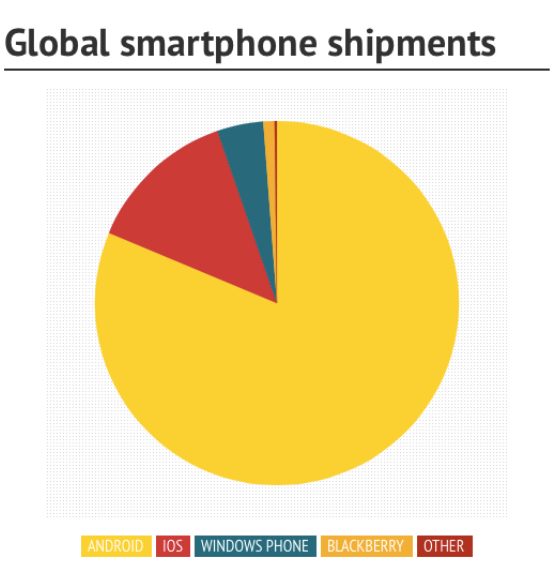

Figure 1-1 Smartphone market share Q3 2013 [1]

One of the self-imposed limitations of this thesis is that the Android device is unrooted[3]. This is because rooting the device would make it harder to use in military settings, because of security issues.

A challenge when it comes to deploying and testing Web service functionality on an Android device is that the official Android API (Application Programming Interface) does not offer a SOAP library, which is essential for Web services communication (for more on SOAP and Web services see Section 2.2). A third-party SOAP library therefore needs to be included.

This thesis focuses on *aspects* of the technical possibilities *related to SOAP on the Android platform*. Though security issues need to be addressed in a production system, security concerns are beyond the scope of this thesis.

[3] Android rooting is the process of allowing users of smartphones, tablets, and other devices running the Android mobile operating system to attain privileged control (known as "root access") within Android's sub-system.

1.5 Research Method

There exist several scientific approaches for the computer science discipline. In Denning et al. [14] the authors define three main scientific approaches for the computer science discipline: theory, abstraction and design. These approaches all follow an iterative process, but the process steps differ from approach to approach.

The **theory approach** is based on mathematics, and consists of the following four stages: 1) Characterize and define the objects of study, 2) form hypothesized theorems of relationships between the objects, 3) determine the truth of each relationship by means of proofs, and 4) interpret the results.

The **abstraction approach** is based on the experimental scientific method, and consists of the following four stages: 1) Form hypothesis, 2) construct a model and make predictions, 3) design, accomplish and measure experiments, and 4) analyse the results.

Finally, the **design approach** is based on engineering. This process also consists of four stages: 1) Perform requirements analysis, 2) derive a specification based on the requirements, 3) design and implement the system, 4) test the system. In the engineering approach the hypothesis is that, the system fulfils the specification and thereby meets the requirements.

This thesis follows the design approach. Section 1.4 (Scope and Technological Challenges) and Chapter 2 (Background / State Of The Art) cover the first step of performing the requirement analysis; whereas the second step of deriving a specification is stated in Section 2.1.3 (Requirements specification). The design and implementation of the system is included in Chapter 3 (Design and Implementation), and the results of testing comes in Chapter 4 (Testing and Evaluation).

1.6 CONTRIBUTION

This thesis aims to contribute to the ongoing effort to optimize SOAP communication, specifically for use on smart devices with Android. The main part of this contribution will be to evaluate using alternative transport mechanisms, as well as different compression techniques, up against the bandwidth usage and battery usage. The goal is to recommend a path or solution for supporting SOAP on Android.

In order to facilitate the above-mentioned goal, a third party SOAP library is used as the basis, and expanded with the possibility for different transport mechanisms and compression techniques.

The resulting library will be given back to the open source community in the spirit of open source programming.

1.7 OUTLINE OF REMAINDER OF THESIS

Chapter 2 covers an introduction to Android, Service Oriented Architecture, SOAP, Web services, different transport solutions and compression techniques, and a third party SOAP library for Android.

Chapter 3 describes the technical solution of this thesis, and design choices made in order to support the goal of the thesis.

Chapter 4 presents profiling tools, the test scenarios and finally presents the results.

Chapter 5 consists of the conclusion, and suggestions for future work.

2 BACKGROUND / STATE OF THE ART

Military R&D centres have investigated the possibility of using smart devices in military settings for years, and Android devices are no exception.

XML is used to define the entire suit of Web services standards. Optimizing XML has been a matter of research for many years, since XML is recognized by much overhead/metadata. The benefits of XML include being both human-readable and machine-readable, and can be used for documents as well as representing arbitrary data structures.

This chapter presents Android, the operating system used in this thesis, as well as an introduction to Service Oriented Architecture, Web services and SOAP. Different transport solutions for SOAP are described, in addition to a short survey of compression techniques focusing on compressing XML. At the end of this chapter, a third party SOAP library for Android is presented.

2.1 ANDROID

Android is an operating system for mobile platforms, which has become in widespread use the last few years.

2.1.1 An open-source mobile operating system

Android, Inc. was founded in 2003, aiming to develop “smarter mobile devices that are more aware of its owner’s location and preferences” [15]. In 2005 Google acquired Android Inc., and in 2007 the Open Handset Alliance (a consortium of technology companies, including Google) announced “the development of Android, the first truly open and comprehensive platform for mobile devices” [16]. Android can be described as a complete set of software for mobile devices; it delivers an operating system, middleware and key mobile applications.

Unlike other major mobile OS’s like iOS and Windows Mobile, Android is an open source platform. Although based on other open source technologies, like Linux and Java, Android differs slightly from these in various ways. Figure 2-1 shows the system architecture of Android.

Android is built on a Linux kernel, but does not include a full set of Linux utilities. The reason for choosing Linux was the memory and process management Linux offers, in addition to the

permission-based security model and support for shared libraries. Another reason was that Linux already was open source.

Android's libraries, Bionic libc, is a derivation of the C library BSD (Berkeley Software Distribution), optimized for embedded use. The reason for developing a different version of C library code was size, CPU speed and licencing reasons. It is not compatible with the common GNU C Library (glibc) [17].

Android's virtual machine, in which the applications run, is called Dalvik. Android applications are written in Java, and then compiled to byte code before they are converted to Dalvik-compatible .dex files at build time. Dalvik is a custom-made virtual machine. The Java used in Android is not entirely similar to the Standard Edition Java Platform, instead it uses a subset of Apache Harmony [18], an open source Java implementation developed by the Apache Software Foundation.

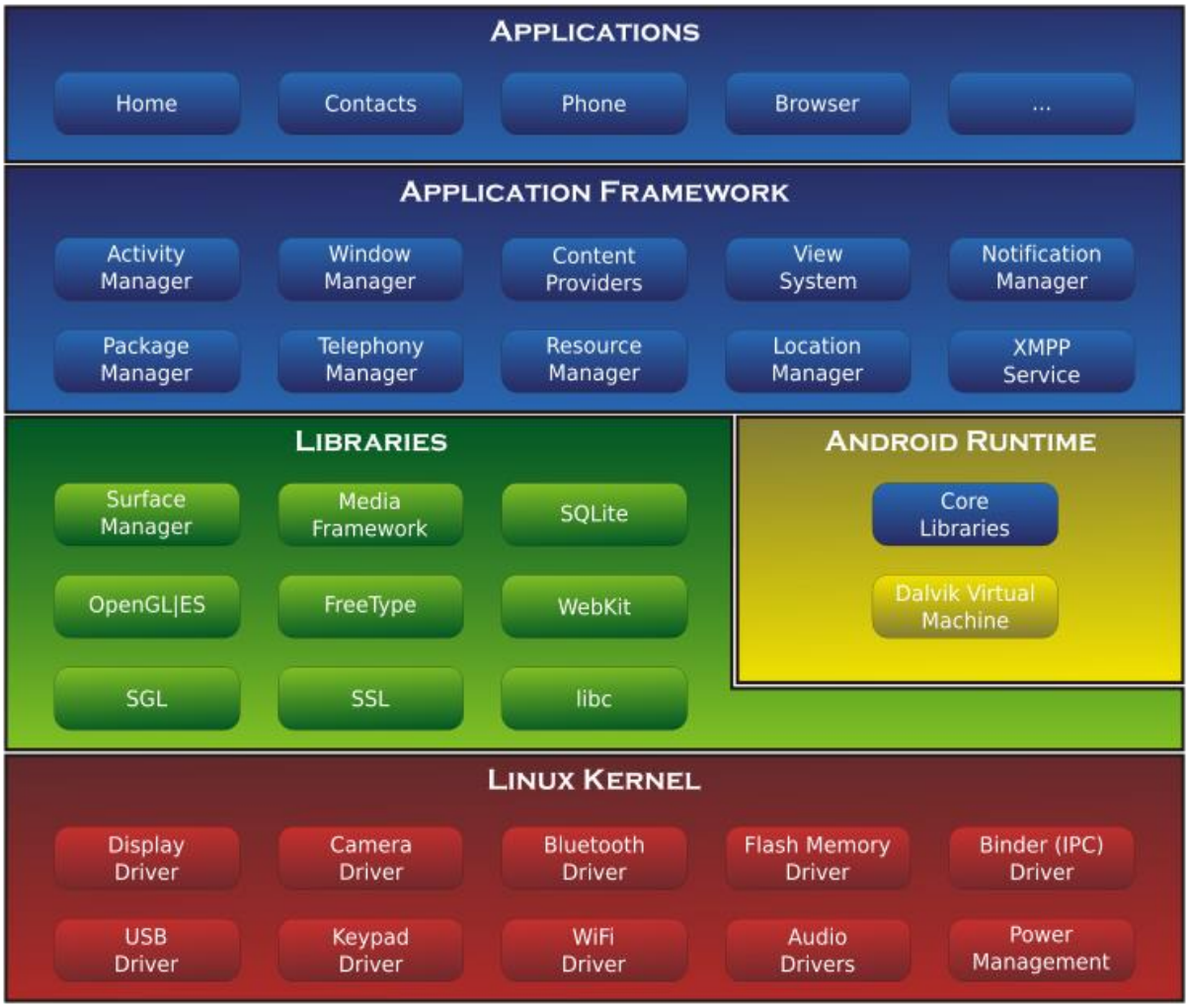

Figure 2-1 Android System Architecture [17]

2.1.2 Android devices in military settings

The US military has already started to incorporate Android smart devices. In the US Army they have specified interfaces and infrastructure for PNT-enabled (Position, Navigation and Timing) applications on COTS devices running Android, enabling them to communicate with military-grade PNT devices. Prototype implementations have successfully demonstrated Android applications using PNT data from external positioning service receivers [19].

Another example from the US is Transformative Apps (TransApps). This is a DARPA (Defence Advanced Research Projects Agency) funded program whose goal is to develop a range of militarily relevant software applications to enhance the operational-effectiveness of military personnel on (and off) the battlefield. Evaluation of 50+ tactically relevant applications operating on numerous Android-powered platforms has been conducted by the National Institute of Standards and Technology's (NIST) [20].

An interesting example from Norway is the CEI-system (Collective Environment Interpretation) [21], a "social tactical reporting system" written for Android, intended to strengthen collective understanding and interpretation of situations. The Norwegian Defence Research Establishment (FFI) developed it to demonstrate smartphone technologies and related technologies with relevance for the Norwegian Defence. The system consists of an Android mobile application, a server, a web-application and a small scripting language.

The CEI-application is a map application intended for smartphones and tablets. The application allows users to share positions and observations with text, pictures and other elements in a uniform way to other users of the CEI-service. It has been tested by the Norwegian Home Guard on several occasions, most recently at "Øvelse Hovedstad" in September 2013. The application has interesting functionality, which can be useful for different parts of the Norwegian Defence, especially to demonstrate the usefulness and the possibilities of mobile technology. The usability of such a system increases proportionally with the battery life span of the mobile unit.

Users of the Android CEI-application can share positions and observations to the rest of the CEI-system, using several device sensors to provide accurate information: Compass, GPS, accelerometer, Wi-Fi, mobile network location and camera. The CEI-server exposes an API based on Representational State Transfer (REST), which the clients can reach over HTTP to share information. REST is an alternatively architectural style for distributed hypermedia systems in regards to SOAP and Web services [22].

Among the drawbacks of using REST instead of SOAP and Web services is a lack of interoperability against NATO standards such as NFFI, forcing the use of proprietary solutions. Enabling the possibility of using SOAP on Android devices could increase the interoperability of Android against other NATO systems and standards.

2.1.3 Requirements specification

From the discussions in Section 1.4 (Scope and Technological Challenges), and up until now in this chapter, a list of requirements can be identified. The resulting system must be able to use relevant open standards, as well as be compatible with COTS services. Four premises can also be stated from 1.4; the platform should be Android, the platform should be unrooted, the network load should be minimized and battery lifetime should be maximized. The goal of this thesis is to implement library support for Android that fulfils the demands stated in Table 2-1 System requirements. The rest of this chapter presents relevant information used as basis for designing a solution (for the design, see chapter 3).

Property	**Implication**	**Importance**
Support SOAP on Android	Further develop existing library	Premise
Minimize network load	Evaluate optimization techniques	Premise
Maximize battery lifetime	Evaluate battery usage of optimization techniques	Premise
Maintain security and integrity	Device unrooted	Premise
Support COTS services	Must be able to connect to COTS server	Requirement
Support relevant open standards	Use Web services specifications	Requirement

Table 2-1 System requirements

2.2 WEB SERVICES

As mentioned in the introduction, a Web service is a technology that can be used for implementing clients and services based on a SOA, achieving interoperability between different systems (see Section 2.3 for more details of SOA).

The *Reference model for service oriented architecture 1.0, OASIS standard, October 2006* [2] defines a **service** as:

*"**A service is a mechanism** to enable access to resources, where the access is provided **using a prescribed interface** and is exercised consistent with constraints and policies as **specified by the service description**."*

When a service is accessible over the Internet, it is called a **Web service**. Figure 2-2 shows a simple illustration of a Web service. The *Web Services Glossary* [23] defines a Web service as:

*"**A Web service** is a software system designed to support interoperable machine-to-machine interaction over a network. It has an **interface** described in a machine-processable format (specifically **WSDL**). Other systems **interact** with the Web service in a manner prescribed by its description using **SOAP-messages**, typically conveyed using HTTP with an **XML serialization** in conjunction with other Web-related standards."*

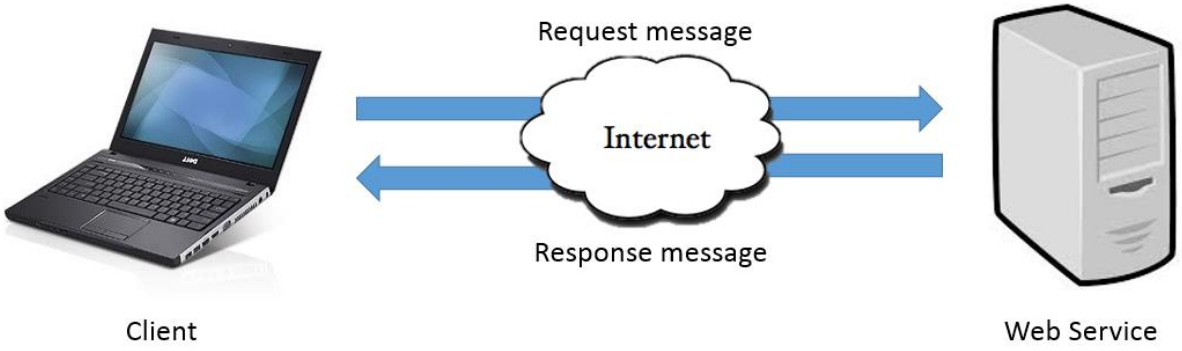

Figure 2-2 A Web service

2.3 SERVICE ORIENTED ARCHITECTURE

The *Reference model for service oriented architecture 1.0, OASIS standard, October 2006* [2] defines SOA like this:

"Service Oriented Architecture is a paradigm *for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations."*

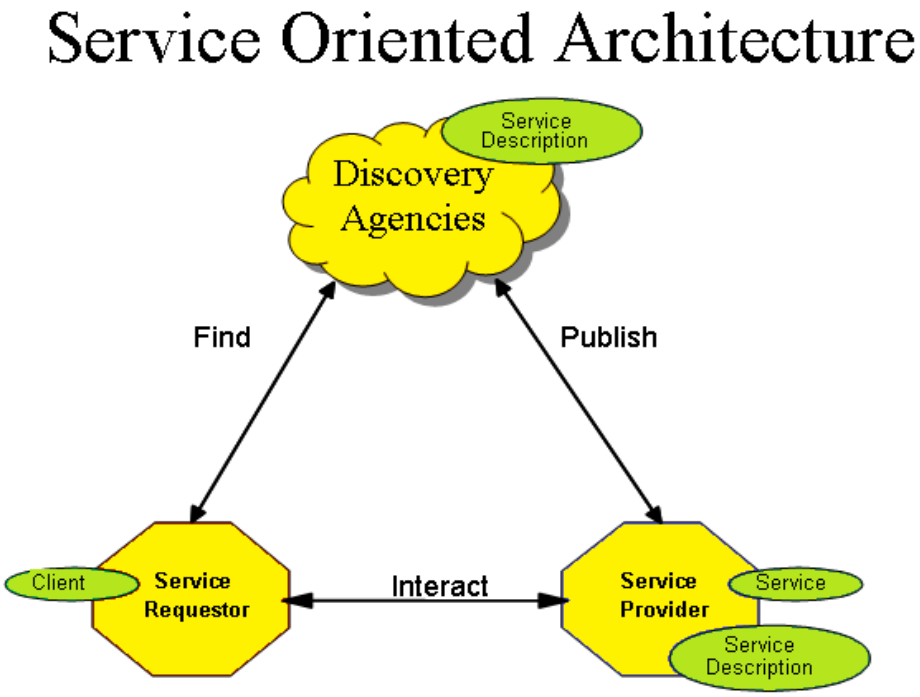

Figure 2-3 Service Oriented Architecture [3]

Figure 2-3 shows a typical illustration of SOA. The essential part of Web services is the relationship between the **service provider** and the **service requestor** (the client). A service's actual implementation is often hidden from the service requestor. What is shown is information and behaviour models exposed through the service interface, and the information required by service requestors to determine whether a given service is appropriate for their needs. The interaction between the service requestor and the service provider is the essential defining element of Web services [24].

The service provider offers a service, accessible over the Internet using common protocols like HTTP and XML to facilitate the interaction. The service provided could be application-

components like currency conversion, weather reports, language translation, database queries, as well as connecting existing software; exchanging data between different applications and different platforms.

In some cases, the service requestor wanting to use a Web service does not know the location of it. Before the service requestor can find a specific Web service offered by a service provider, the service provider must publish the Web service (see Figure 2-3). The Web service is published as a XML document called *Web Services Description Language* (WSDL) [25]. A WSDL is written in XML, and it specifies the location of the service and the operations (or methods) the service exposes.

When the service requestor knows the location and parameter of the service (found in the WSDL), it can request to use the Web service offered. The interaction between the service requestor and the service provider is in most cases facilitated by HTTP transportation of XML-encoded messages, using the SOAP protocol.

2.4 THE SOAP PROTOCOL

The tutorial *Unravelling the Web services web: an introduction to SOAP, WSDL, and UDDI* presents a simple introduction to the SOAP protocol [26]. SOAP [4] is an XML-based protocol for messaging and remote procedure calls (RPCs). Rather than define a new transport protocol, SOAP works on existing transports, such as HTTP, Simple Mail Transfer Protocol (SMTP) [27], and Advanced Message Queuing Protocol (AMQP) [28]. At its core, a SOAP message has a very simple structure: an XML element with two child elements, one of which contains the header and the other the body. The header contents and body elements are themselves XML. Figure 2-4 shows a SOAP envelope's structure.

```
<SOAP:Envelope xmlns:SOAP=
            «http://schemas.xmlsoap.org/soap/envelope/»>
   <SOAP:Header>
            <!- content of header goes here ->
   </SOAP:Header>
   <SOAP:Body>
            <!- content of body goes here ->
   </SOAP:Body>
</SOAP:Envelope>
```

Figure 2-4 SOAP envelope

At the basic functionality level, SOAP works as a simple messaging protocol. SOAP messages are in Web service context predominantly carried by HTTP requests and responses. Figure 2-5 shows a SOAP message carried by HTTP.

```
POST /travelservice
SOAPAction: “http://www.acme-travel.com/checkin”
Content-Type: text/xml; charset=“utf-8”
Content-Length: nnnn

<SOAP:Envelope xmlns:SOAP=
            «http://schemas.xmlsoap.org/soap/envelope/»>
   <SOAP:Header>
            <!- content of header goes here ->
   </SOAP:Heaader>
   <SOAP:Body>
            <!- content of body goes here ->
   </SOAP:Body>
</SOAP:Envelope>
```

Figure 2-5 SOAP message carried by HTTP

The HTTP headers are above the SOAP:Envelope element. The POST header shows that the message uses HTTP POST, which web browsers also use to submit forms. Following the POST header is an optional SOAPAction header that indicates the messages’ intended purpose. If a response follows the request, the HTTP response would be of type text/xml, as declared in the Content-Type header, and could contain a SOAP message. Alternatively, the recipient could deliver the response message later (asynchronously).

2.4.1 SOAP limitations

SOAP relies on the XML format. XML defines a set of rules for encoding documents in a format that is both human-readable and machine-readable, and is a platform-independent and well-supported technology. Nonetheless, the very feature that makes Web services and SOAP universally usable, namely the adoption XML, may make it difficult to reach the performance levels required by large-scale processes and applications.

A major performance bottleneck resides in SOAP message processing. SOAP communication produces considerable network traffic, and causes higher latency than competing technologies. This problem applies especially to wireless communication networks with their

relatively low bandwidth and high latency. Another issue is that generation and parsing of SOAP messages and their conversion to-and-from in-memory application data can be computationally very expensive. [29]

2.5 SOAP Optimizations: Compressing Data

Since SOAP messages are large and contain much overhead, compressing these messages is an obvious way to reduce the size of the messages. This, in turn, may make SOAP communication more effective. There are roughly two different ways to compress SOAP messages, general data compression and XML-specific compression.

2.5.1 General data compression

General data compression involves encoding information using fewer bits than the original representation. The compressed data is then decompressed, yielding the original data. Compression is often divided into lossless (no information lost in the compression/ decompression) and lossy (discarding some of the data). Lossless compression exploits repeating patterns or redundant information in order to express the same information in a more efficient way. Lossy compression is used for minimizing the size of data when the consequence of losing some of the information is not critical, for instance compressing music or pictures. Lossy compression is not relevant in the SOAP and XML context.

2.5.1.1 Deflate Compression (gzip, zlib)

Deflate [30] is a popular data compression method that was originally used in the well-known Zip [31] and gzip [32] software. It has since been adopted by many applications, among them the HTTP protocol, the PPP (Point-to-Point compression control protocol) [33], PNG (Portable Network Graphics) [34], and Adobe's PDF (Portable Document File) [35]. Deflate was developed by Philip Katz as part of the Zip file format. Both the Zip format and the Deflate method are in the public domain, which allows implementations such as Info-ZIP's Zip and Unzip to appear on a number of platforms.

The most notable implementation of Deflate is zlib [36], a portable and free compression library designed and implemented to be free of patents and licensing requirements. This

library implements the zlib and gzip file formats, which are at the core of most Deflate applications, including the popular gzip software [37].

The gzip/zlib compression is very popular, and most computer platforms have implemented it. Gzip provides better compression rate (40-50%) and freedom from patented algorithms [38]. Because of these properties, gzip will be used in this thesis.

2.5.2 XML-specific compression.

XML-specific compression may use ***binary XML*** as basis. Binary XML is a more compact way of representing XML. A basic example of binary XML could be a semicolon-separated string. The XML document in Figure 2-6 as expressed in binary XML would be:

;Dag;Ove;Eggum;h136577@stud.hib.no;Hetlevikstraumen 5;5173;Loddefjord.

```
<contacts>
    <firstName>Dag</firstName>
    <middleName>Ove</middleName>
    <lastName>Eggum</lastName>
    <phoneNumber>95903263</phoneNumber>
    <eMail>h136577@stud.hib.no</eMail>
    <Address>Hetlevikstraumen 5</Address>
    <PostalCode>5173</PostalCode>
    <Town>Loddefjord</Town>
</contacts>
```

Figure 2-6 Binary XML

This would be a more compact way to represent the data, but it would not be self-describing in the way well-formed XML is. Unless both the sender and receiver of the binary XML representation agree on how to convert it back to valid XML, interoperability is lost. One of the major challenges for binary XML has been to create a single, widely adopted standard for binary XML.

One of the advantages of binary XML is that it is possible to work on it without decompressing. Below is a short description of some of the most important standards and experimental solutions.

2.5.2.1 Efficient XML Interchange

Efficient XML Interchange (EXI) [39] is a standard for XML representation that was designed to work well for a broad range of applications. The EXI format is derived from the AgileDelta Efficient XML format, and is built on open standards [40]. The World Wide Web Consortium (W3C) adopted it as a W3C Recommendation on 10 March 2011.

EXI is schema[4] "informed", meaning it can utilize available schema information to improve compactness and performance, but does not depend on accurate, complete or current schemas to work. EXI also offers a set of fidelity options, each of which independently enables or disables the format's capacity for the preservation (or preservation level) of a certain type of information item. This is useful for applications that do not require the entire XML feature set and would prefer to eliminate the overhead associated with unused features [39]. Disabling the preservation of information may affect interoperability with other systems.

2.5.2.2 Fast Infoset

Fast Infoset [41] is a standard for XML representation published by the International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) and International Telecommunication Union-Telecommunication Standardization Sector (ITU-T) using binary encodings. These binary encodings are specified using the Abstract Syntax Notation One (ASN.1) notation and the ASN.1 Encoding Control Notation (ECN) [42].

2.5.2.3 XMill

XMill [43] is a compression tool specially targeted at compressing XML, and its implementation is a result of work done at the AT&T Labs Research in New Jersey, USA, in 1999. XMill incorporates and combines existing compressors in order to apply them to heterogeneous XML data: it uses zlib, the library function for gzip, a collection of data type specific compressors for simple data types, and user defined compressors for application specific data types [44].

[4] An XML schema is a description of a specific type of XML document, expressed in terms of constraints on the structure and content of documents of that type.

2.5.2.4 WAP Binary XML (WBXML)

WBXML [45] defines a compact binary representation of XML. The binary XML content format is designed to reduce the transmission size of XML documents, allowing more effective use of XML data on narrowband communication channels. It was developed by the Wireless Application Protocol (WAP) Forum and is now maintained by the Open Mobile Alliance.

2.5.3 Survey of comparisons of Binary XML solutions

The 2007 study *An Analysis of XML Compression Efficiency* [46] tested a multitude of ways to compress XML, focusing on the compressed file sizes and execution times. Among its conclusions were that in most instances a general-purpose compressor should be used, although if maximum parsing and compression speed was needed an XML-specific compressor might be useful. The results indicated that binary format was best applied to small files.

In *New Approaches for XML Data Compression* of 2012 two algorithms for XML documents compression were proposed: Schema-aware algorithm and Hybrid algorithm [47]. These were compared to WBXML, XMill and EXI, considering the metrics compression rate and compression time. Among the conclusions were that EXI reached the best compression rate.

The paper *Why Use Efficient XML Interchange Instead of Fast Infoset* of 2013 presents EXI and Fast Infoset (FI) as the best XML compressors [48]. The paper endeavoured to evaluate the performance of both compressors based on parameters such as memory utilization, CPU usage time, compression ratio, decompression ratio, and the compressed file sizes. The authors concluded that EXI schema informed mode compression delivers superior results compared to other FI compression technique; “EXI is better performer than FI.” Because of these results, this thesis will test EXI as the XML-specific compressor.

2.6 SOAP OPTIMIZATIONS: DIFFERENT TRANSPORT METHOD

The most common way to transport SOAP messages is using HTTP, but other transport methods may be used as well.

2.6.1 Transport protocols for SOAP

One of the characteristics of SOAP is neutrality; SOAP enables exchange of SOAP messages using any transport protocol, such as HTTP, SMTP, TCP, or UDP. The formal set of rules for carrying a SOAP message within or on top of another underlying protocol for the purpose of exchange is called a **binding**. The *SOAP Protocol Binding Framework* [49] provides general rules for the specification of protocol bindings; the framework also describes the relationship between bindings and SOAP nodes that implement these bindings.

2.6.1.1 SOAP-over-HTTP/TCP

HTTP/TCP [11, 12] is the most commonly used transport protocol for sending SOAP messages (Figure 2-7). An important reason for this is that all Internet browsers and servers support HTTP. TCP is one of the core protocols of the Internet protocol suite (IP) [50], and is so common that the entire suite is often called TCP/IP.

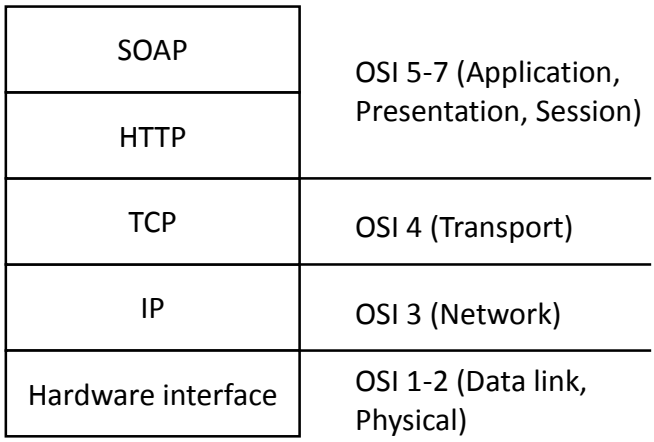

Figure 2-7 SOAP-over-HTTP/TCP

Before data can be sent over a TCP connection, a three-way handshake between a client and a server has to occur. This is to ensure that both parties acknowledge the TCP connection and are ready to send/receive data. TCP transmission is ordered, the packets are numbered so that the destination host can rearrange the packets according to sequence number. TCP also ensures retransmission of lost packets, checks for error, and offers flow and congestion control.

The flow control limits the rate a sender transfers data to guarantee that the receiver is not overwhelmed by packets, while the congestion control tries to balance how much data is sent to avoid congesting (exceeding) the network capacity.

2.6.1.2 SOAP-over-SMTP

Using SMTP [27] to exchange SOAP messages (Figure 2-8) is mentioned more often as an illustration of the possibility to use different transport protocols, than it is actually used. In SOAP-over-SMTP binding, SOAP messages are piggybacked on SMTP packets.

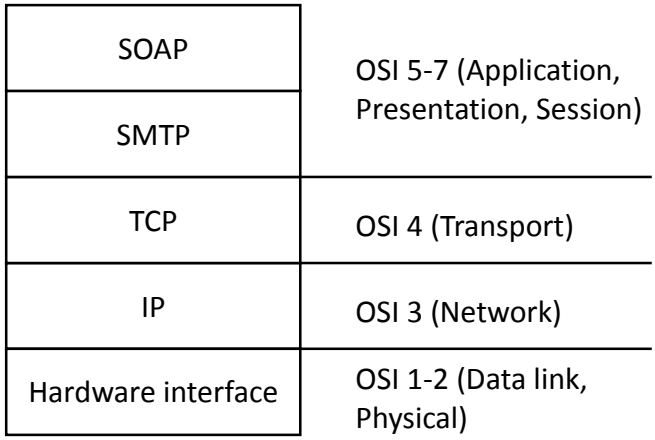

Figure 2-8 SOAP-over-SMTP

2.6.1.3 SOAP-over-TCP

SOAP-over-TCP [12] (Figure 2-9) differs from SOAP-over-HTTP in that the SOAP message is stored directly in the data part of the TCP packet, not including the HTTP metadata. There is no official standard for binding SOAP directly with TCP, but various specifications have been developed [51].

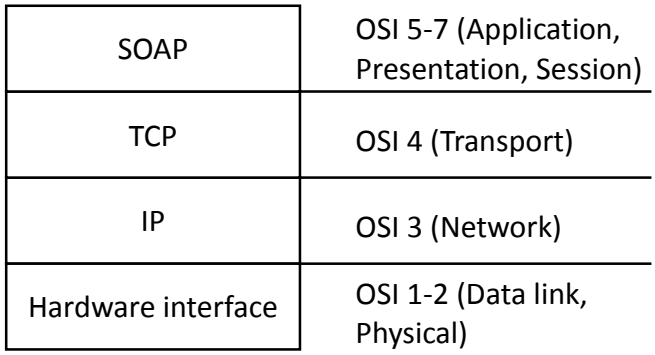

Figure 2-9 SOAP-over-TCP

2.6.1.4 SOAP-over-UDP

The User Datagram Protocol (UDP) [52] is another core transport protocol used on the internet, but it is a much simpler protocol than TCP, using a simple transmission model with much fewer transport mechanisms. UDP has no three-way handshake like TCP, and offers no guarantee of delivery, ordering of packets or duplicate protection.

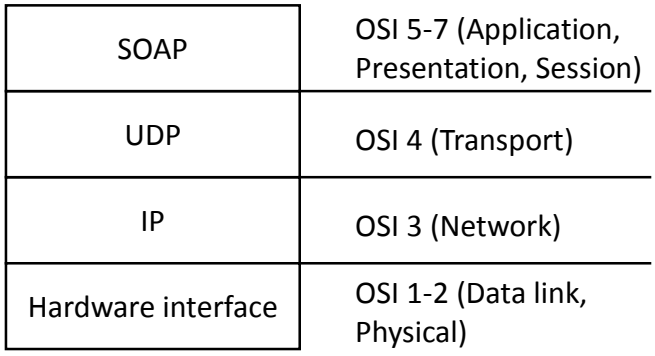

Figure 2-10 SOAP-over-UDP

SOAP-over-UDP (Figure 2-10) is an OASIS Standard intended to send SOAP envelopes in UDP user datagrams, supporting one-way MEP (message-exchange patterns), request-response MEP and multicast transmission [53].

Benefits from using UDP over TCP are that UDP does not require a connection to be established before sending a packet, reducing setup time associated with sending a message. UDP datagrams have less overhead than TCP packets, making UDP more suitable for networks with low bandwidth available, or transmitting time critical data (such as real-time voice/video). UDP supports multicasting, which opens up the possibility to create push[5]-based or publish/subscribe-based Web services [54].

2.6.1.5 SOAP-over-SCTP

The Stream Control Transmission Protocol (SCTP) [55] is a reliable transport protocol operating on top of a connectionless packet network such as IP (like TCP or UDP does) (Figure 2-11). Despite being designed to transport PSTN (Public Switched Telephone Network) signalling messages over IP networks, SCTP is capable of broader applications. It

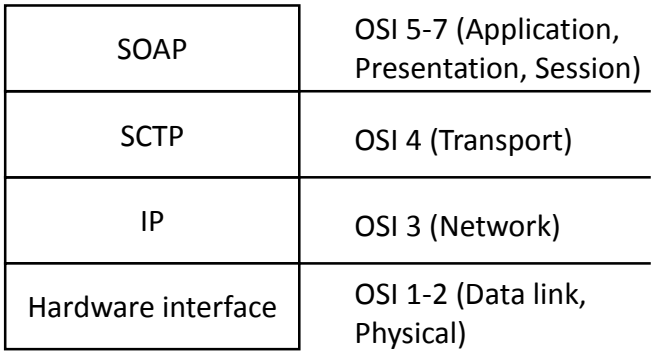

Figure 2-11 SOAP-over-SCTP

[5] Push describes a style of Internet-based communication where the request for a given transaction is initiated by the publisher or central server.

offers functionality from both TCP and UDP, in that it is message-oriented like UDP, but ensures reliable, in-sequence transport of messages with congestion control like TCP [56]. SOAP-over-SCTP is an interesting approach, but it is not standardized.

2.6.1.6 SOAP-over-AMQP

AMQP [28] is an open standard for passing messages between applications. It is an application layer protocol designed to support a variety of messaging applications and communication patterns, and assumes an underlying reliable transport layer protocol such as TCP. The Advanced Message Queuing Protocol (AMQP) Technical Committee is working towards defining AMQP as a ubiquitous, secure, reliable and open Internet protocol for message handling [28].

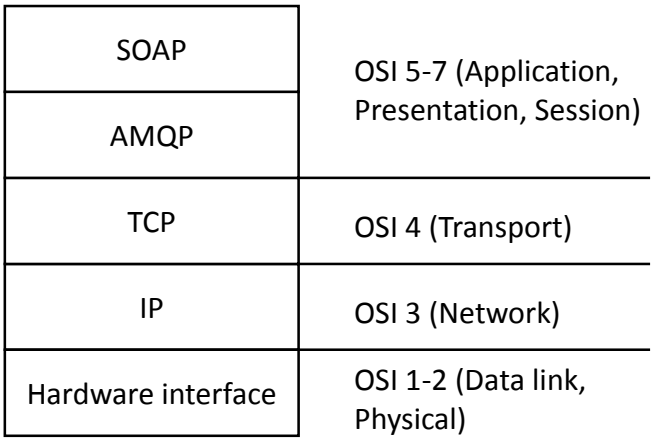

Figure 2-12 SOAP-over-AMQP

In [10] AMQP was described as a protocol which could be suitable for Web services in disruptive environment, due to its reliability when facing network disruptions. This solution is not an official standard yet, but a specification is in the pipeline [57].

2.6.1.7 Problems with SOAP-over-HTTP/TCP

As mentioned earlier, using SOAP-over-HTTP/TCP is the most common and standard way when using Web services. TCP performs well in traditional networks where packet losses occur mostly because of congestion. However, networks with wireless and other lossy links also suffer from significant losses due to a higher Bit Error Rate (BER) and handoffs.

The congestion avoidance mechanism in TCP assumes packet losses are always due to congestion. However, in a wireless environment, packet losses can also be due to disconnections and transmission errors. The congestion control and avoidance algorithms in TCP results in degrading the end-to-end performance of TCP in such networks [58]. This problem increases when using TCP in mobile Ad-hoc networks, adding link failures due to mobility [59].

TCP requires a connection to be established before any data can be transmitted. As data is received, acknowledgement packets are sent. This lead to additional overhead, which may not be justifiable when bandwidth and client power are limited, and reliable transmission of packets is not required.

SOAP messages that carry only small amounts of data can finish transmitting while the TCP connection is in its slow start phase, resulting in poor utilization of the available bandwidth. This problem is more severe in wireless environments due to high round trip time.

2.6.2 Surveys of alternative transport protocols for SOAP

A benchmark [6] of the performance of different underlying transport protocols for SOAP in wireless environments was presented in *On the Performance of Web Services* [54]. Here it was shown that SOAP-over-HTTP and SOAP-over-TCP are not well suited for wireless applications, and lead to high latency and high transmission overhead. Using SOAP-over-UDP instead was studied, and results showed that SOAP-over-UDP provided a throughput up to six times higher than SOAP-over-HTTP in wireless settings. Using UDP instead of HTTP/TCP also reduced transmission overhead by more than 30%.

Johnsen et al.'s study [10] also investigated using alternative transport protocols to convey SOAP messages in order to both reduce the bandwidth requirement and meet the challenges related to frequent disruptions in wireless network characterized by low bandwidth, variable throughput, unreliable connectivity and energy constraints. This study considered these protocols relevant for testing at that time: TCP, UDP, SCTP, and AMQP. Among the results was that UDP performed well compared to the other protocols with small payloads for large bandwidths. It also stated that SCTP was a promising new transport protocol, performing better than TCP in many cases. Based on these results this thesis will test the same protocols used in [10] on Android.

[6] A benchmark is in this context a test used to compare performance of hardware and/or software.

2.7 SOAP Library for Android

The Android API does not contain a SOAP library for creating or interpreting SOAP messages, meaning that a third-party library needs to be added for this. There exist several non-official SOAP libraries aimed for working on Android (AndroidSOAP [60], WSClient++ [61], ksoap2-android [62]), but as they often are created and maintained on a voluntary basis, some tend to be outdated while others require payment to use. This thesis will use the SOAP library provided by the ksoap2-android project [62] since it was actively maintained at the time of choosing and also has been described in similar testing with SOAP on Android.

The ksoap2-android project provides a lightweight SOAP client library for the Android platform [62]. It is a fork of the kSOAP2 library [63] optimized for the Android platform, but also works on other platforms using Java. ksoap2-android provides an API for creating SOAP envelopes in the XML format, thus making an Android application capable of interacting with a Web service.

Future Information Communication Technology and Applications [64] proposes ksoap2-android as an open source SOAP API with small footprint implementation of XML, aimed at developing applications for the Android platform. ksoap2-android was also tested in *Web Services for Handheld Tactical Systems* [8], where ksoap2-android was modified to support UDP on the Android platform in order to transmit video data.

2.8 Using a Proxy Server

A proxy server is a system or an application situated between a client and a server, acting as an intermediary node that relays traffic between the client and the server. Proxies add structure and encapsulation to distributed systems, and are often used for caching data, firewalling and adapting content. Proxies can also be used for adapting different types of communication traffic to a COTS server, which is relevant for this thesis.

In *IST-118 – SOA recommendations for Disadvantaged Grids in the Tactical Domain [65]* Johnsen et al. argue that proxies should be used to apply SOA;

“Recommendations from that group include employing optimizations such as removing the dependency on end-to-end connections, addressing network heterogeneity, and reducing the network traffic overhead of Web services. The group suggested introducing proxies to implement these optimizations, in an attempt to provide a separation of concerns between proprietary enhancements and COTS services and clients “

3 DESIGN & IMPLEMENTATION

This chapter presents the design and implementation of the Web service app created for testing the ksoap2-android library on an Android unit.

3.1 OPTIMIZING

The design and implementation aim to meet the demands stated in Section 2.1.3 (Requirements specification). This thesis will test a combination of changing the transport protocol and implementing compressing methods. The main objective of the testing is to demonstrate ways to optimize the ksoap2-android library for use on mobile devices, fulfilling the premise of supporting SOAP on Android as stated in the requirements specification. The testing should measure the effects these changes have on the mobile device used for testing, and the network load.

3.1.1 Compression techniques

The two compression methods chosen to compare against uncompressed transmission are gzip and EXI. The reason for not including more compression tools is that many of them are not adapted to the Android platform. Including gzip and EXI is linked to the premises of minimizing network load and maximizing battery lifetime (see Section 2.1.3).

The reason for choosing gzip is that it is widely used for file compression and decompression, and one of the most efficient general compression tools (see Section 2.5.1.1). Both Java and Android have embedded classes for writing and reading compressed data in the gzip file format.

The reason for choosing EXI is that it is one of the most prominent binary XML efforts to encode XML documents in a binary data format, and has performed best in several comparisons (see Section 2.5.3). EXI is not part of either Java or Android, and therefore has to be added as a third-party library.

3.1.2 Transport layer protocols

The three SOAP transportation methods chosen for comparison are HTTP/TCP, SOAP-over-UDP and SOAP-over-AMQP. SCTP would also be interesting to test, but as of the time of writing, Android has not yet made it available in the official API [66]. Including SOAP-over-

UDP and SOAP-over-AMQP is linked to the premises of minimizing network load and maximizing battery lifetime (see Section 2.1.3).

The reason for choosing HTTP/TCP is that it is the most common way of sending SOAP messages. Since HTTP and TCP are in such widespread use on the Web, both Java and Android has good support for them.

The reason for choosing SOAP-over-UDP is that it has been reported to give a higher throughput and less transmission overhead than HTTP/TCP in wireless settings (see Section 2.6.2). UDP is also a very common protocol, and is supported in both Java and Android.

The reason for choosing SOAP-over-AMQP is that it could be suitable for use in disruptive environments, and therefore is interesting to include. In this thesis, a third-party library will be included for implementing SOAP-over-AMQP.

Initially, SCTP was supposed to be tested alongside the above-mentioned protocols. SCTP has been successfully implemented as a third-party library on Android in experimental test cases [67-69]. An attempt to do the same was done in this thesis as well, but with unsuccessful results. Appendix A describes this attempt.

3.1.3 Proxy

Utilizing a proxy server is useful for providing a link between the provider and the consumer of a service. A proxy server will be used in this thesis to enable the client to be able to connect to the COTS server using the Web services specification, as stated in the requirements specification. This approach was discussed briefly in Section 2.8.

3.2 DESIGN

The design is constructed to meet the requirements specification discussed in Section 2.1.3.

3.2.1 Main architecture

The WS Client is implemented as an Android app, running on a Samsung Galaxy Tab 2 (GT-P5100). The Android version used in this thesis is 4.2.2. The tablet has the possibility to transfer data over both mobile broadband and Wi-Fi. Figures 3-1 and 3-2 show the system architectures for both transmission alternatives.

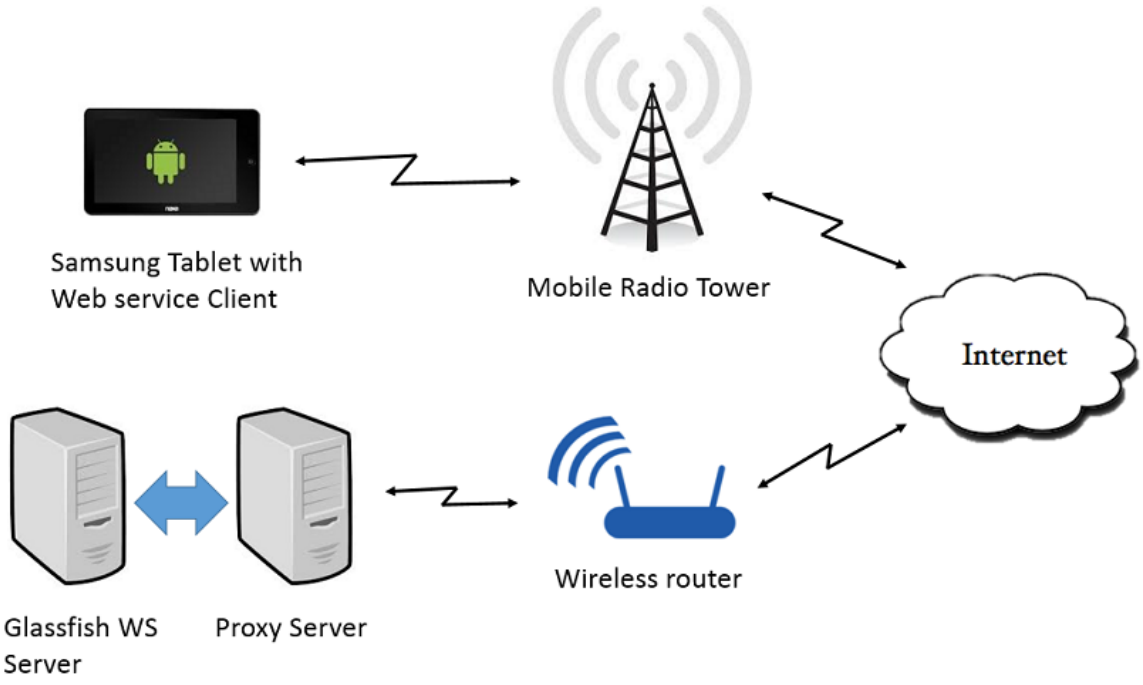

Figure 3-1 System architecture using mobile broadband

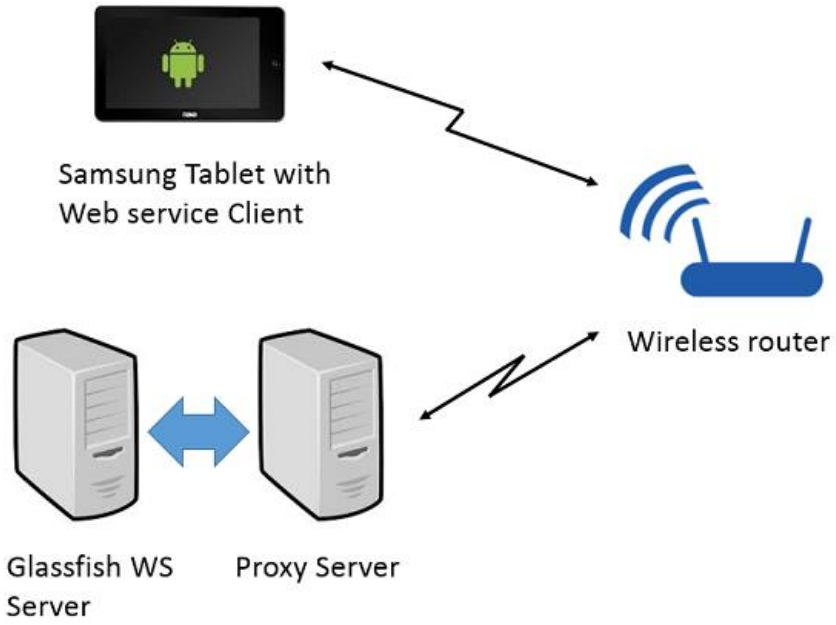

Figure 3-2 System architecture using Wi-Fi

The mobile network is provided by Telenor, in an area with mobile broadband a few kilometres outside Bergen, Norway. The wireless router is a Jensen Air:Link 89300 LongRange Extreme-N. The internet connection of the wireless router provides 25 Mbps download, and 5 Mbps upload.

The proxy server handles all the compression and use of different transport protocols, and communicates with the Web service server in uncompressed HTTP. The Web services used for testing are hosted on a Glassfish server. Both servers are installed on the same computer, an ASUS Notebook N56V running Windows 8.1. Table 3.1 describes the hardware specifications for the Android device and the computer hosting the servers.

Model	**Samsung Galaxy Tab 2**	**ASUS N56VZ**
Processor	Dual-core 1.0 GHz	4-Core 2.4 GHz
RAM	1 GB	8 GB
Storage	16/32 GB	750 GB
WLAN	Wi-Fi 802.11 a/b/g/n, Wi-Fi Direct, dual-band, Wi-Fi hotspot	Wi-Fi 802.11 b/g/n
HSDPA[7]	21 Mbps	N/A

Table 3-1 Hardware specifications

[7] High-Speed Downlink Packet Access (HSDPA) is an enhanced third-generation mobile-telephony communications protocol, which allows networks based on Universal Mobile Telecommunications System (UMTS) to have higher data-transfer speeds and capacity.

3.2.2 ksoap2-android

ksoap2-android is a lightweight SOAP client library for the Android platform. This thesis uses version 3.0.0. It uses the XML parser kXML2 [70] to parse the SOAP messages. A parser is a software component that takes input data and builds a data structure; in this case, kXML2 takes XML as its input and parses it to Java objects. kXML2 is a pull parser [71], which means that is parses a document as a series of items which are read in sequence, only proceeding at the user's command. kXML is small, designed for constrained environments such as Applets, Personal Java or MIDP (Mobile Information Device Profile) devices [70].

Figure 3-3 shows a simplified class diagram for ksoap2-android. The main class starts a Web service call with creating a *SoapEnvelope* before it uses the method *call* in HttpTransportSE to send the request. In the *call* method, an instance of the class ServiceConnection is created. The instance of ServiceConnection creates a HTTP connection to the server hosting the Web service. The response from the Web service is then parsed into the *SoapEnvelope*, from which the main class then can collect the response. ksoap2-android contain many more classes and is more complex, but this illustrates a basic Web service call.

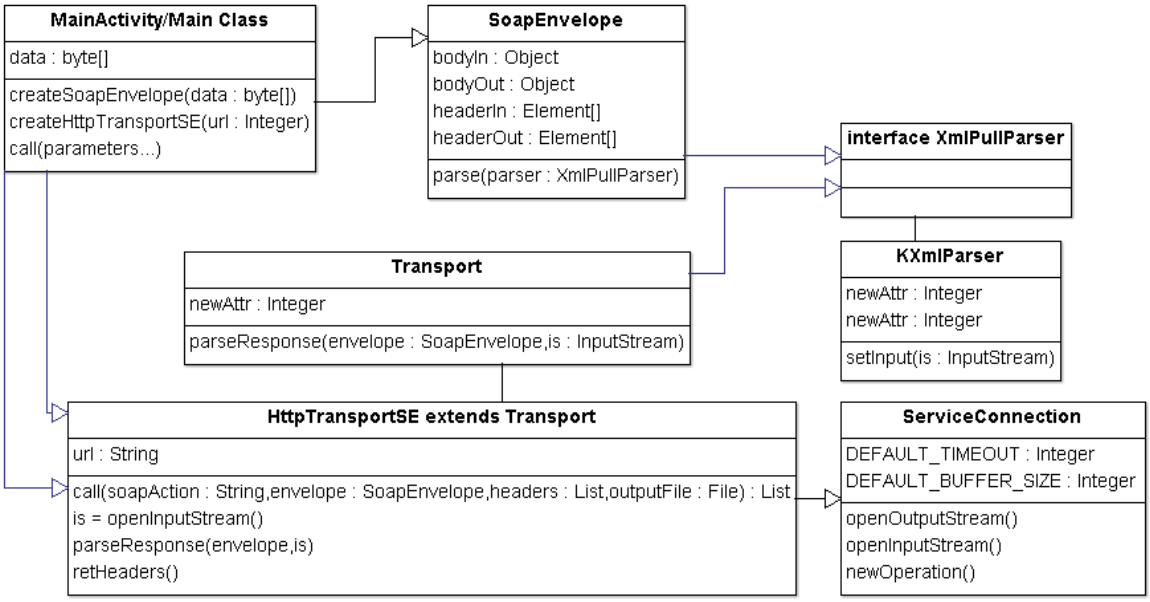

Figure 3-3 Simplified Class Diagram of ksoap2-android

Despite being able to create and interpret SOAP messages, ksoap2-android is not a full SOAP library, since it is missing several useful features. It does not support the WS-Addressing standard [72]. WS-Addressing provides transport-neutral mechanisms to address a message to a Web services. Instead of relying on network-level transport to convey routing

information, a message utilizing WS-Addressing includes routing data in a standardized SOAP header. WS-Addressing helps identify Web service endpoints and secure end-to-end endpoint identification in messages.

ksoap2-android does not support stub and skeleton code generation from a WSDL [25] specification to Java code. “Stubs and skeleton” is a standard mechanism used in Java Remote Method Invocation (RMI) [73] for communicating with remote objects. In a distributed computing environment, a *skeleton* stands for a server side object, and *stub* stands for a client side object, both participating in distributed object communication. The stub serializes the client requests to a SOAP format message, according to the WSDL definition. This automatic serialization is not supported in ksoap2-android and must therefore be done manually.

Both Android and ksoap2-android are missing the Java Architecture for XML Binding (JAXB) [74]. JAXB allows Java developers to map Java classes to XML representations, but when using ksoap2-android this must be done manually. This can cause extensive coding if the Java classes to be mapped are large and complex.

These deficiencies are not necessarily a drawback for this thesis, since its goal is to make the *exchange* of SOAP messages more effective.

3.2.3 The Web service client

The Web service client is an Android app running on a Samsung Galaxy tablet, using the ksoap2-android library to send and receive SOAP messages. The app was created using Android Development Tools (ADT) [75], a plugin for Eclipse [76]. The ADT/Eclipse version used is build number v22.3.0-887826. Eclipse is an integrated development environment (IDE), which is a software application that provides facilities and programming tools for software development.

Android supports both HTTP and UDP, so no third-party programs were needed here. The third transport method, AMQP, is not a part of the Android API and needed to be added. To implement this RabbitMQ [77] was chosen. The RabbitMQ version used is version 3.2.3. RabbitMQ is a messaging system based on the AMQP standard, written in Java. The reason for choosing it was that it is free and open source, and was easy to implement in both the app and the proxy server.

ksoap2-android originally only had the possibility to decode gzip-encoded responses. In this testing the ability to also encode the requests was added using the gzip classes found embedded in Java and Android. To add the choice of compressing according to the EXI standard, *EXIficient* [78] was chosen. EXIficient is an open source implementation of the EXI format written in the Java programming language, the version used is 0.9.2.

3.2.4 The proxy server

The proxy server is written in Java, using the IDE Eclipse, and provides a public IP address and several ports for the Web service client to contact. The proxy server consists of a simple HTTP server [79] from Oracle, as well as several UDP servers and message queues.

Figure 3-4 illustrates the role of the proxy server. When the proxy server receives a compressed SOAP message from the client, the proxy server decompresses the message, makes an uncompressed exchange with the Web service server, and compresses the response before sending it back to the client. The horizontal lines represent the transition to and from the wireless communication for both the client and the server.

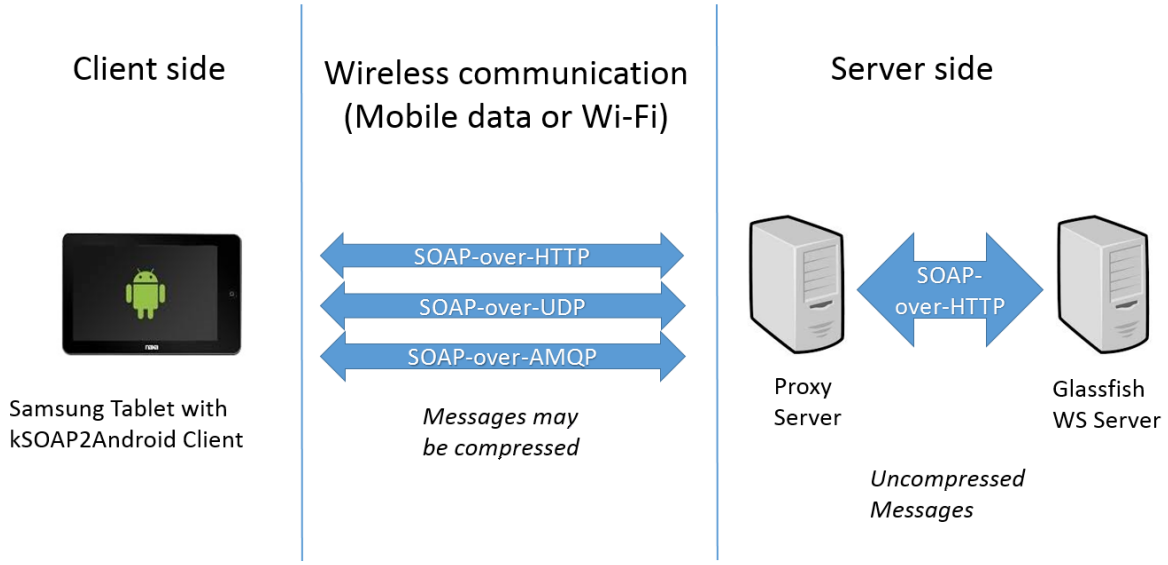

Figure 3-4 The role of the proxy server

The reason for adding the proxy server between the Web service server and the Web service client was to simplify the software coding when dealing with different transport protocols and compression techniques, thus achieving interoperability between the client using ksoap2-android and the Web service server. The proxy server ensures compatibility with the COTS server, meeting the requirements specification in Section 2.1.3. A more detailed view of how the proxy server works is illustrated in Figure 3.5.

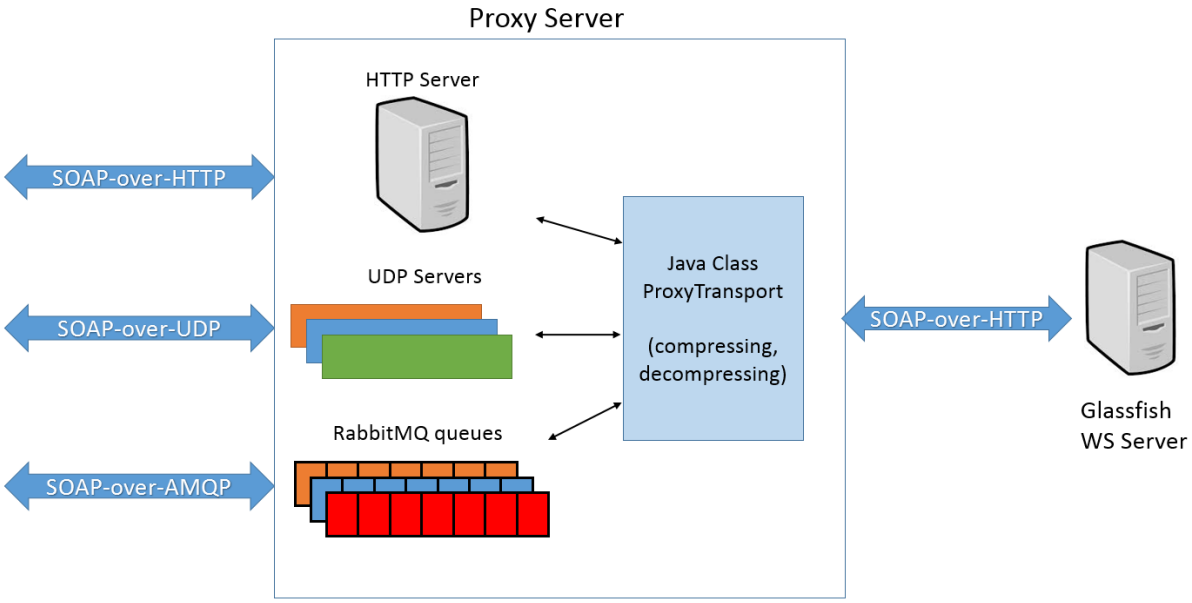

Figure 3-5 Proxy server internals

The HTTP server listens for incoming HTTP connections from the Web service client. It relays the SOAP requests and responses with the Web service server in uncompressed HTTP.

There are three UDP servers (one for each compression mode) listening on different ports in the proxy server. They are constructed using Datagramsockets, which exchanges datagram packets with Datagramsockets in the Web service client.

There are also three different message queues (one for each compression mode) in the proxy server. The queues themselves are actually held by the instance of RabbitMQ installed on the computer, but the code that uses these queues to publish and consume messages are part of the proxy server.

To communicate with the Web service server all the different servers use a class ProxyTransport, which takes a byte array as an input instead of a SOAP envelope. ProxyTransport was created using HttpTransportSE as a template, and does the compression and decompression, if any, and ultimately contacts the Web service.

3.3 IMPLEMENTATION

This section describes how the ksoap2-android was changed to match the test goals, and shortly how the app was constructed. The complete code can be found in the data provided together with this thesis.

3.3.1 Changes in ksoap2-android

The ksoap2-android library offers a Java API to create and read SOAP messages, without much support for compression (it has some functionality to decode gzip), and send these over HTTP to a Web service server. The basic SOAP call in ksoap2-android is illustrated in Figure 3-6: An instance of the class HttpTransportSE is created with a given Uniform Resource Locator[8] (URL), and a method call to HttpTransportSE causes it to make a HTTP request to the Web service server at the given URL, and modifies the SoapEnvelope with the Web service response. The main thread creates the SoapEnvelope using other ksoap2-android classes.

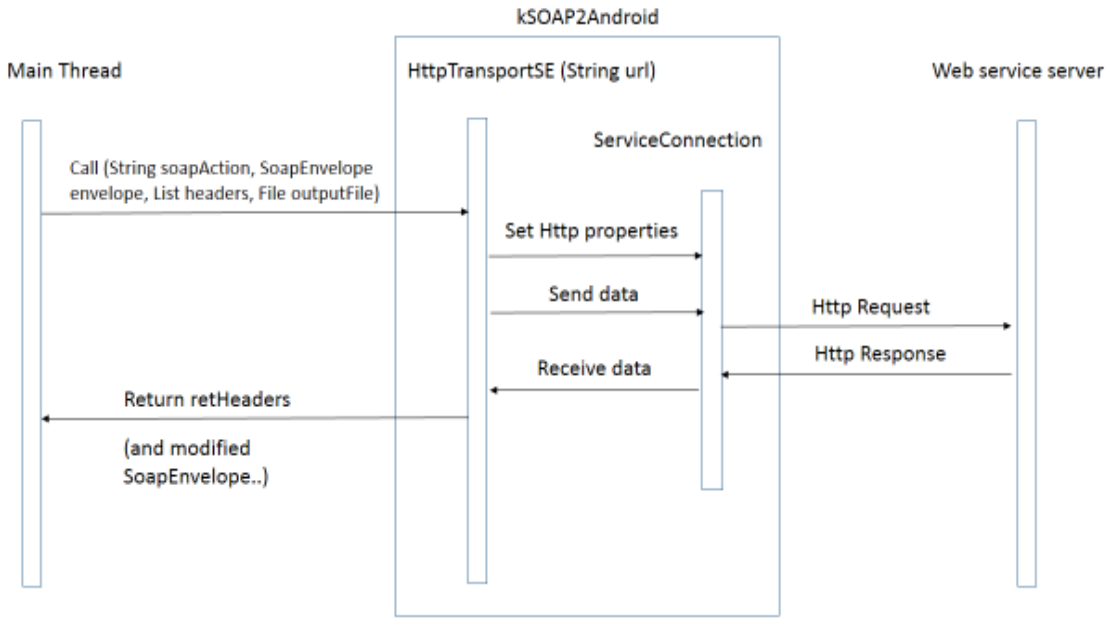

Figure 3-6 HttpTransportSE

[8] A uniform resource locator is a specific character string that constitutes a reference to a resource. An example of a typical URL would be "http://www.vg.no/".

To expand the possibilities of transport methods and compression techniques three more classes were added: HTTPTransport, UDPTransport and MQTransport. These were constructed using the original HttpTransportSE as a template.

3.3.1.1 UDP implementation

The original ksoap2-android uses a HTTP/TCP connection to send the SOAP message to the server. In order to implement the possibility to send SOAP over UDP, a class UDPTransport was created, using HttpTransportSE as a template (see Figure 3-7).

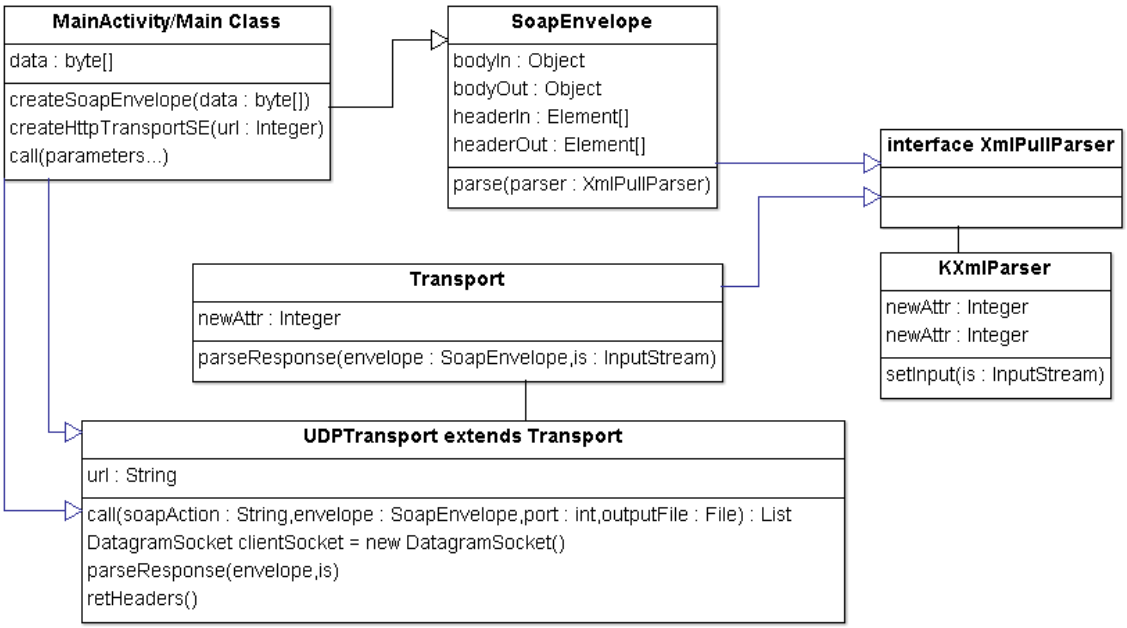

Figure 3-7 Class UDPTransport

Instead of using ksoap2-android's ServiceConnection to create a HTTP connection, UDPTransport creates a UDP socket, which exchanges datagrams with the UDP sockets in the proxy server. As illustrated in Figure 3-8 an Integer port is included in the call made from the main thread, indicating what port to use. This is due to a design choice on the server side of defining different UDP ports in regards of what compression is used.

Since UDP does not include header data the same way that HTTP does, the correct way would be use WS-Addressing to include routing data in the SOAP header. Since neither Android nor ksoap2-android supports WS-Addressing (see Section 3.2.2) assigning different ports is a workaround to this challenge.

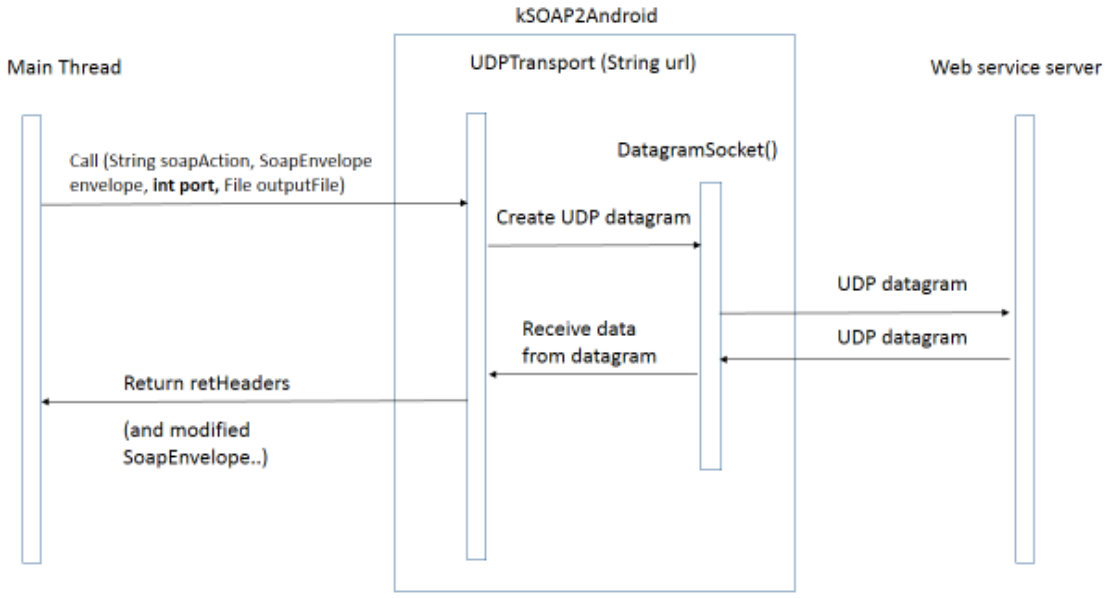

Figure 3-8 UDPTransport

3.3.1.2 RabbitMQ implementation

Tying RabbitMQ together with ksoap2-android was done in a similar way as with UDP. A class MQTransport was created, using HttpTransportSE as a template (see Figure 3-9).

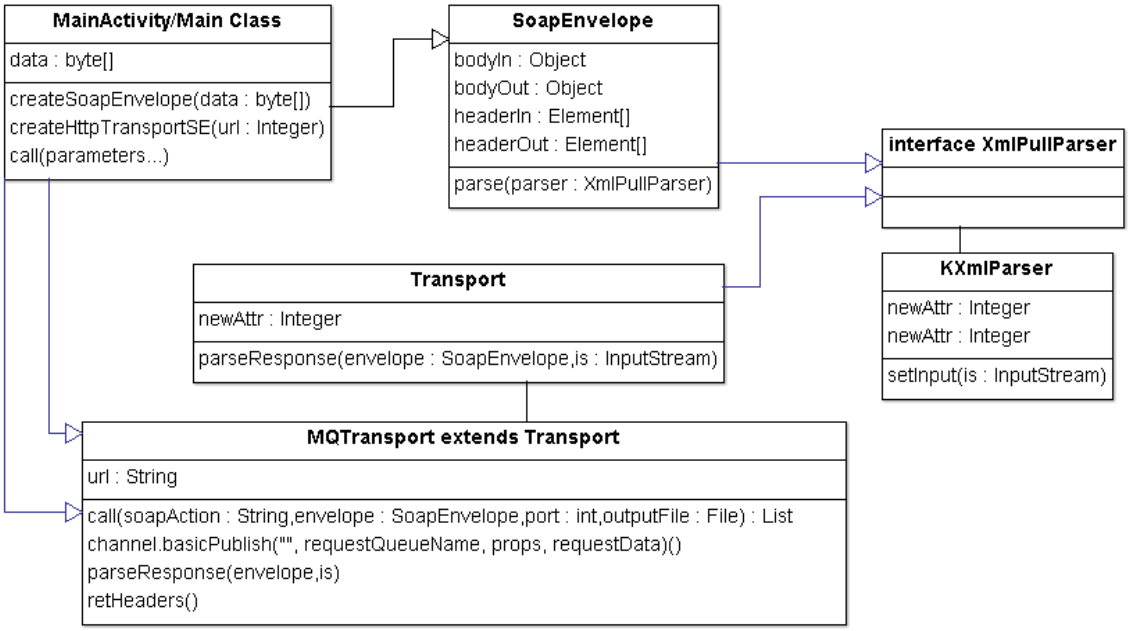

Figure 3-9 Class MQTransport

RabbitMQ is a message broker, meaning it accepts, stores, and forwards messages (see Figure 3-10). These messages are stored in what is called a message queue. A program that sends messages to the broker is called a producer (P), and the program receiving the messages is called a consumer (C). One advantage of using message brokers is that many producers can send messages that go to one queue, and many consumers can try to receive messages from one queue.

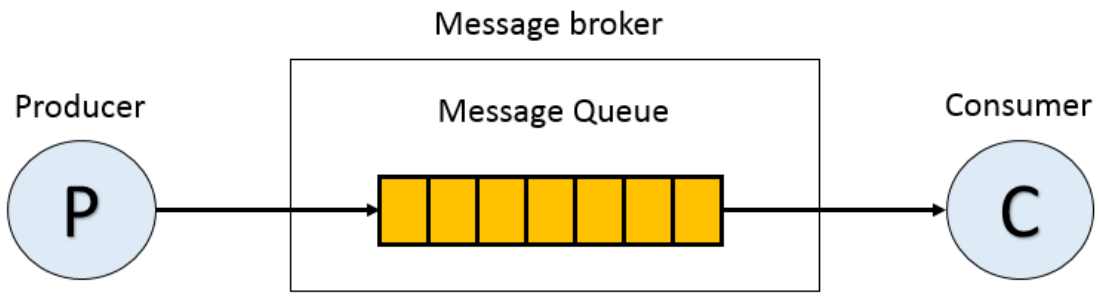

Figure 3-10 Message broker

In this thesis, the Web service client and the proxy server will both act as producers and consumers, when exchanging SOAP messages. The message broker, RabbitMQ, is installed on the same computer as the proxy server.

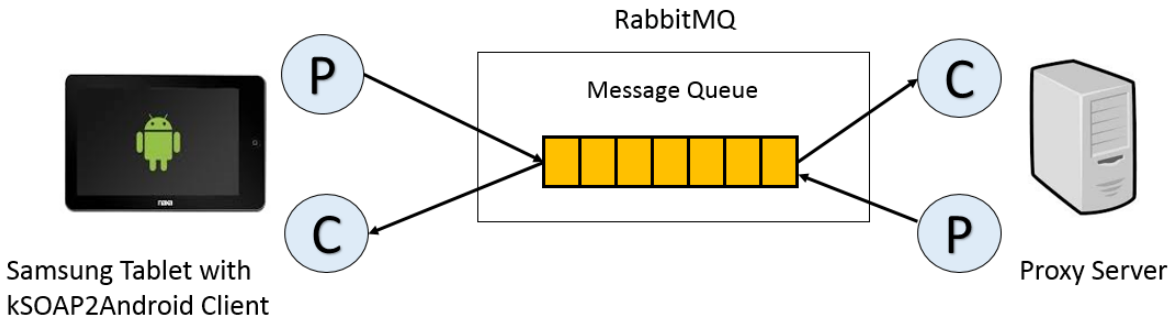

Figure 3-11 RabbitMQ

Instead of using the ksoap2-android's ServiceConnection to create a HTTP connection, MQTransport creates a connection (over TCP) to RabbitMQ. It then publishes the SOAP message to the message queue, and waits for the response from the queue.

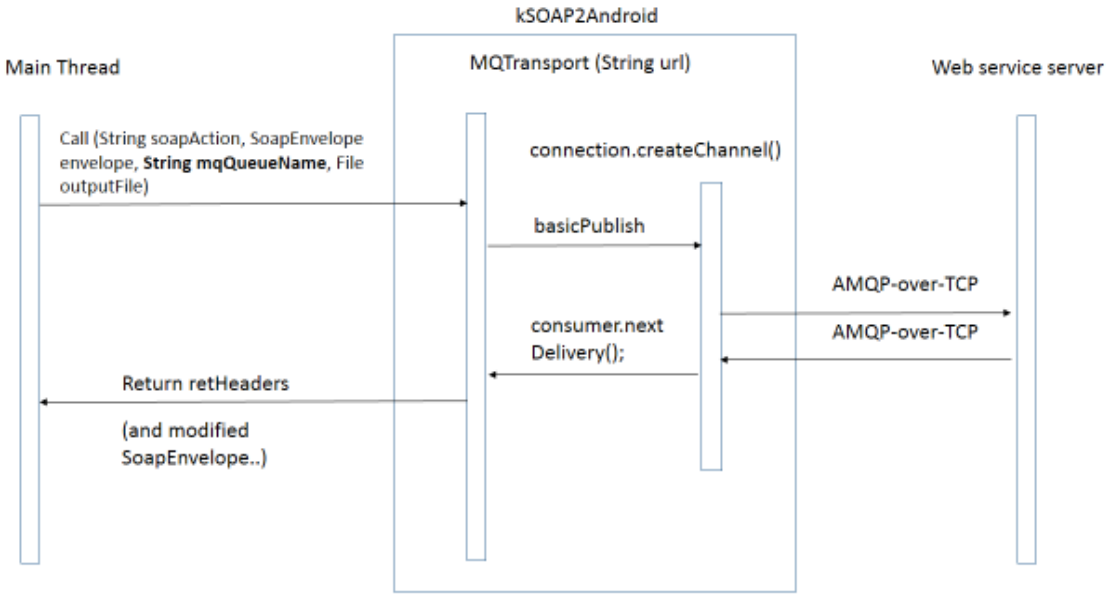

Figure 3-12 MQTransport

3.3.1.3 Gzip implementation

Gzip is supported both in the Java API and in the Android API. The original class HttpTransportSE in ksoap2-android had support for decoding gzip if the Web service response was gzipped, but lacked the functionality to encode request data with gzip. Including this possibility consisted of adding a few programming lines in the newly created HTTPTransport, UDPTransport and MQTransport.

3.3.1.4 EXIficient implementation

EXIficient is an open source implementation of the EXI format specification written in the Java programming language. The EXIficient version used is 0.9.2, and was downloaded from [80]. Since EXIficient is written in Java, it was possible to run it on Android without any noteworthy changes. However, the EXIficient library depends on Xerces, and Xerces initially gave some problems for Android.

Xerces is Apache's collection of software libraries for parsing, validating, serializing and manipulating XML. The Xerces version that came with the EXIficient library did not work properly on the Android platform (since Android and Java has diverged somewhat), therefore a modified version of Xerces was needed. Luckily, a modified version already existed, and was found at [81]. The modified ksoap2-android library of this thesis uses this version.

In order to implement the EXIficient library within ksoap2-android, a class called ExiJava was created. This class handles the compression and decompression, and is called from the either HTTPTransport, UDPTransport or MQTransport.

As stated in Section 2.5.2.1, the EXI format offers a set of fidelity options, each of which preserves or abandons certain elements of the XML feature set, eliminating overhead associated with unused features. In ExiJava, the fidelity option is set to the ***strict*** option, meaning that namespace prefixes and comments are not preserved. This was done in order to obtain a better compression result.

The resulting ksoap2-android class diagram (after adding the different compression techniques to HTTPTransport, UDPTransport and MQTransport) can be found in Figure 3-13.

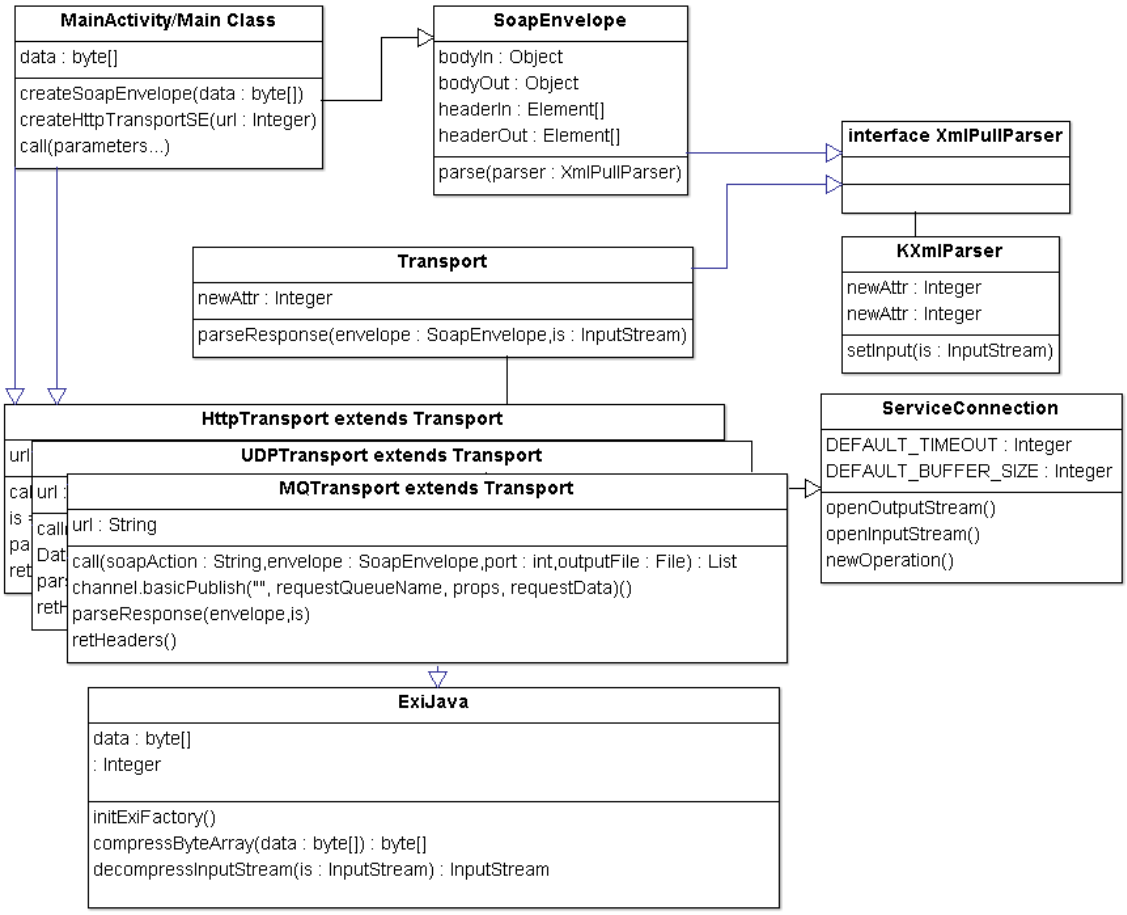

Figure 3-13 Resulting ksoap2-android class diagram

4 TESTING AND EVALUATION

This chapter describes the profiling tools available for Android, the test scenarios and finally presents the results.

4.1 PROFILING FOR ANDROID

In software engineering, profiling is a form of program analysis that measures different parameters of a software program. Common profiling parameters includes how much memory is used, how much CPU time is used, frequency and duration of function calls et cetera. Profiling is a way to aid program optimization.

Android has its own debugging tool used for software profiling called Dalvik Debug Monitor Server (DDMS). It is integrated in the Eclipse IDE as well as included in the Android Software Development Kit (SDK). DDMS is a useful tool for Android app developers, and provides port-forwarding services, screen captures of the device, thread information, heap information, processes, radio state information, and more. Figure 4-1 shows a typical DDMS screen in Eclipse.

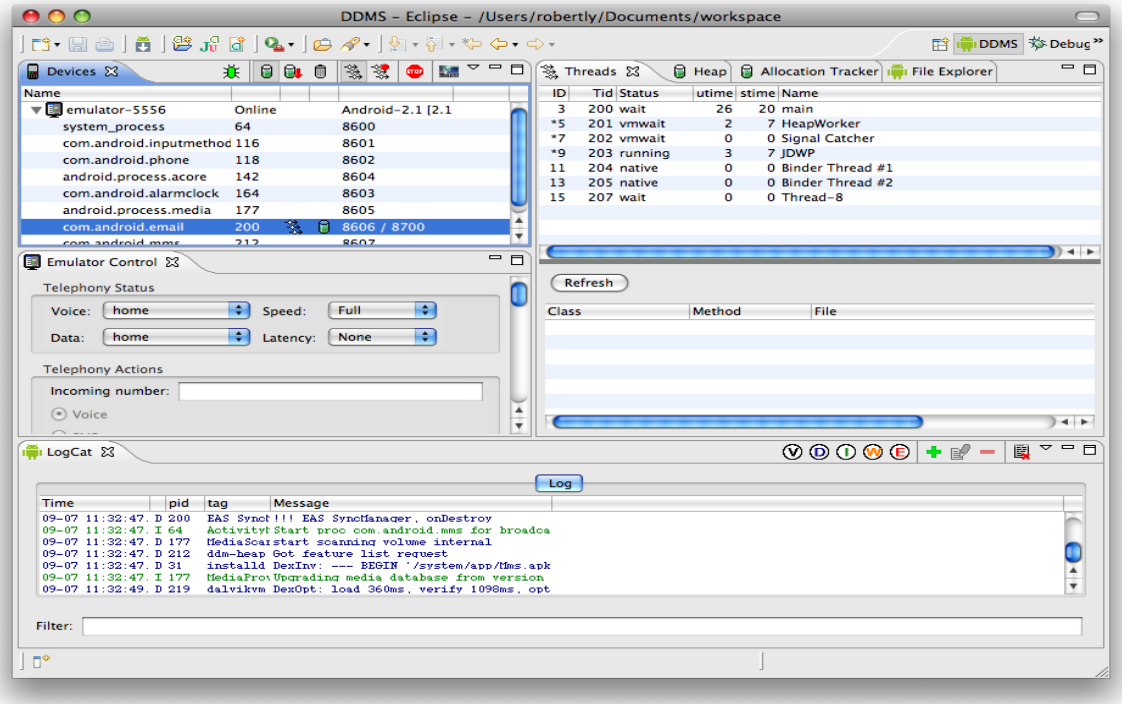

Figure 4-1 DDMS graphical front-end [82]

4.1.1 Method profiling

Method profiling is a means to track certain metrics about a method, such as number of calls and execution time. To do this DDMS needs to be told when to start method profiling, and when to stop. After the profiling DDMS will open a **Traceview** with the profiling information collected. Traceview is a graphical viewer for execution logs, which can help debug an app and profile its performance. Traceview visualizes the application in two panels: The Timeline panel and the Profile panel.

In the Timeline panel (Figure 4-2), each thread's execution is shown in its own row, with time increasing to the right. Each method is shown in different colours. The thin lines underneath the first row show the extent (entry to exit) of all the calls to the selected method.

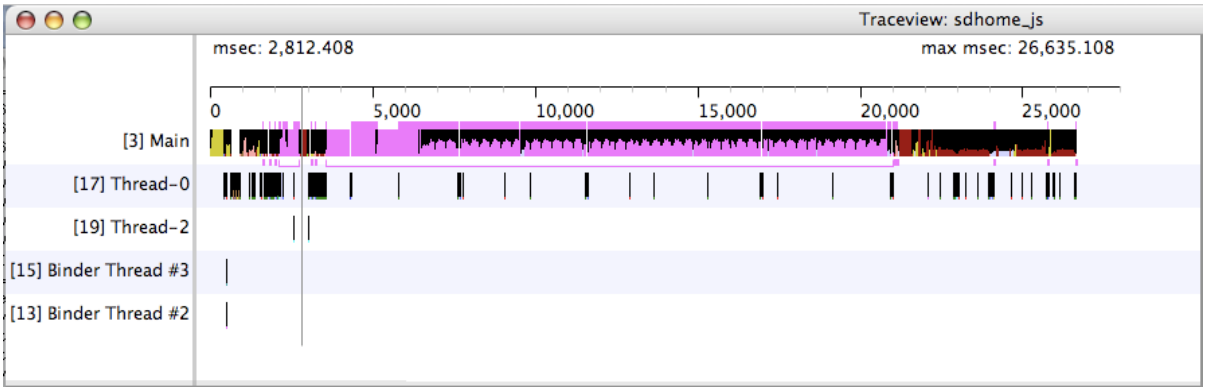

Figure 4-2 DDMS Timeline panel [82]

The profile panel (Figure 4-3) shows a summary of all the time spent in a method. The panel shows both the inclusive and exclusive times (as well as the percentage of the total time). Exclusive time is the time spent in the method. Inclusive time is the time spent in the method, and the time spent in any called functions.

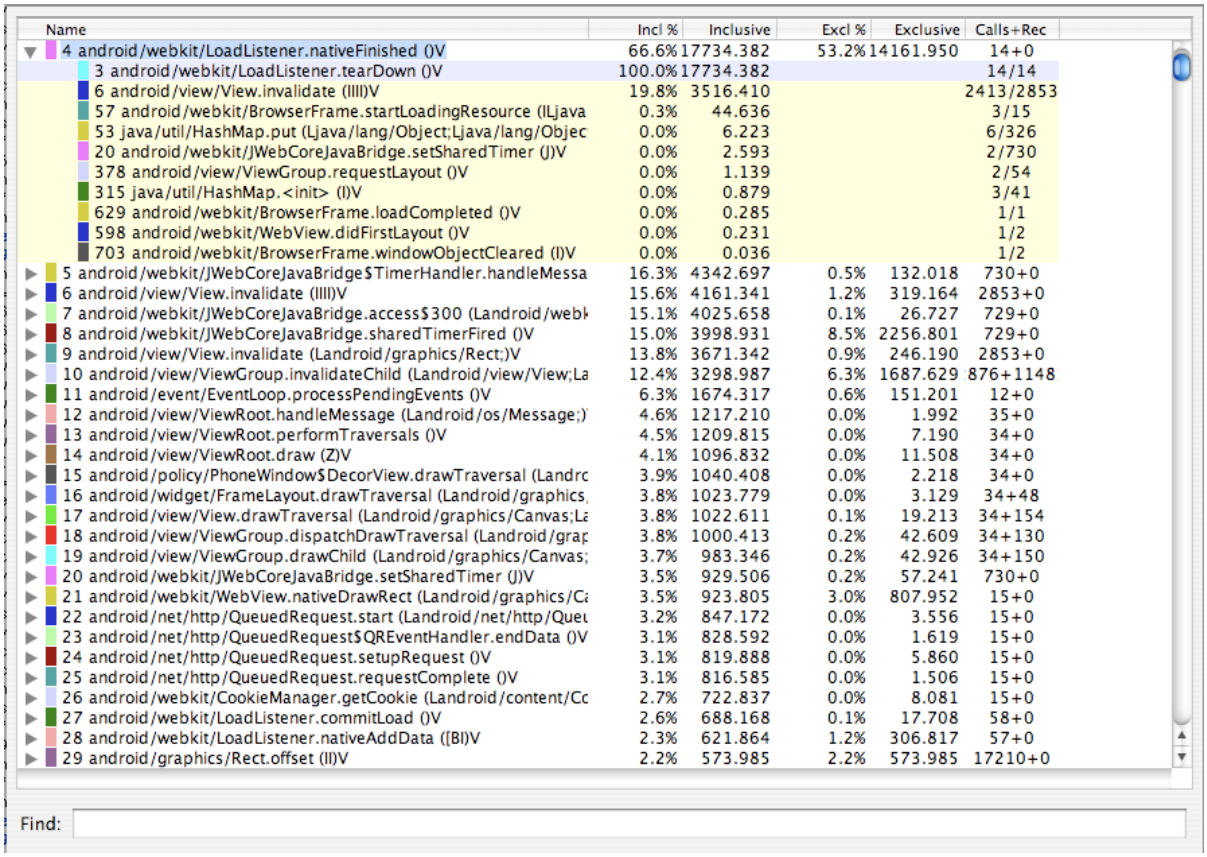

Figure 4-3 DDMS Profile panel [82]

The method profiling described above provides very detailed information, which amounts to even more if the same method call is performed several times. An alternative way of measuring CPU load is to measure how much time a method in the program spends before it finishes. This time can then be compared with running the same method with other parameters (in our testing these parameters would be the different compression methods and different transport methods).

4.1.2 Network traffic tool

Wireshark [83] is a network protocol analyser that can capture the traffic running on a computer network, both wired and wireless. It is freely available as open source, and is released under the GNU General Public License version 2. Wireshark has a graphical front-end, plus some integrated sorting and filtering options (see Figure 4-4 and Figure 4-5).

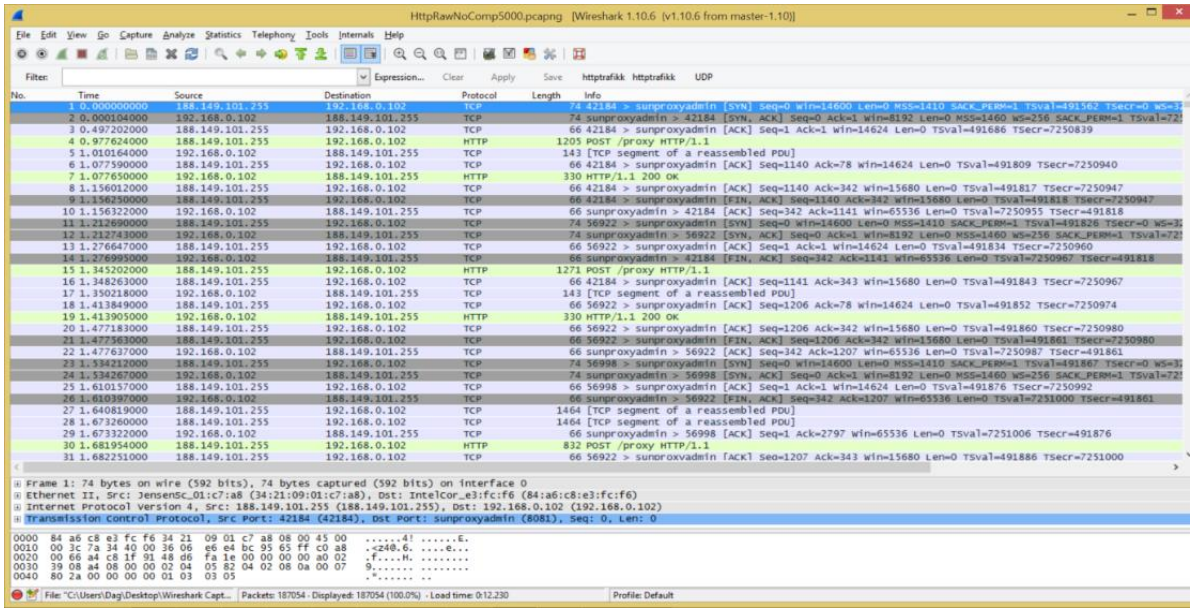

Figure 4-4 Wireshark graphical front-end

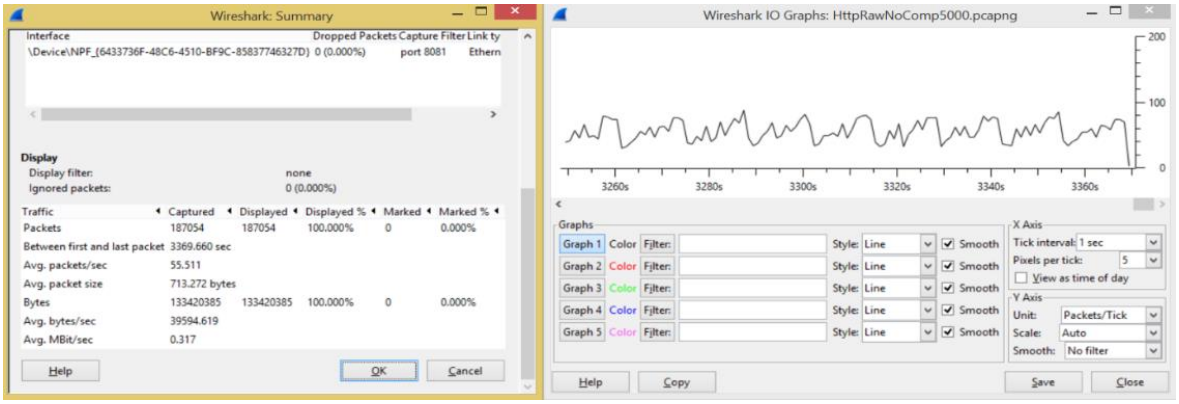

Figure 4-5 Wireshark summary and graph illustration

4.2 TESTING

Three different Web services were created for testing purposes (see Sections 4.2.1 – 4.2.3 for detailed descriptions). These Web services were created using the Netbeans IDE (version 7.1.2). Each different configuration of transport and compression mode in the Web service client were tested against these three Web services, over both the mobile network and Wi-Fi, giving six different tests:

- Test 1: “Hello Web service” over mobile network
- Test 2: “Hello Web service” over Wi-Fi
- Test 3: “Upload NFFI data Web service” over mobile network
- Test 4: “Upload NFFI data Web service” over Wi-Fi
- Test 5: “Exchange Picture Web service” over mobile network
- Test 6: “Exchange Picture Web service” over Wi-Fi

The reason for creating several Web services was to vary between large and small SOAP messages, as well as having both XML and non-XML payloads. The testing was done under normal conditions, as artificial packet loss or bad network connection were not added.

When UDP and RabbitMQ were used as the transport method during the testing, fabricated routing data was included in the SOAP header of the request message, to simulate the use of WS-Addressing.

In addition to the Web service testing, the size of the compressed files were measured in order to compare the compression of gzip and EXIficient.

4.2.1 Hello Web service

In this Web service, the client sends a small request with a String “Name” to a Hello Web service hosted on the Glassfish server, which replies with a String “Hello Name !”.

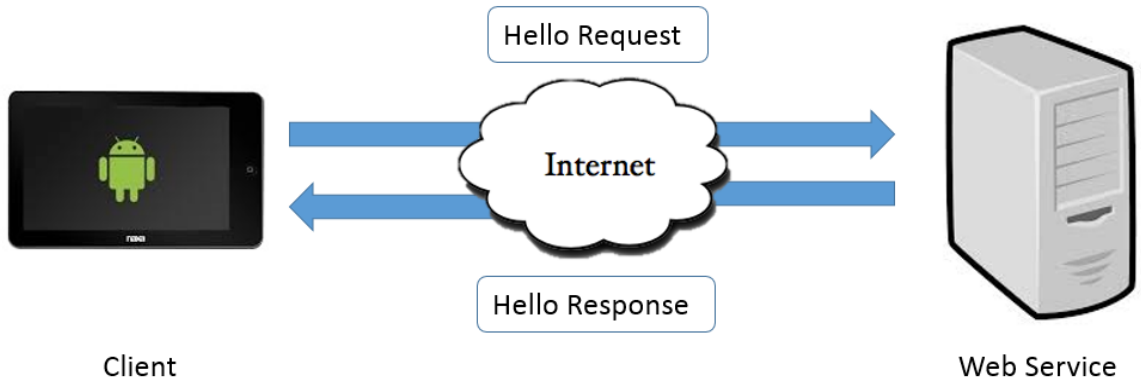

Figure 4-6 Hello Test

```
<v:Envelope
xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns:d="http://www.w3.org/2001/XMLSchema"
xmlns:c="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:v="http://schemas.xmlsoap.org/soap/envelope/"><v:Header />
  <v:Body>
    <n0:hello id="o0" c:root="1" xmlns:n0="http://eggum/">
      <name>Dag</name>
    </n0:hello>
  </v:Body>
</v:Envelope>
```

Figure 4-7 Hello request

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelopexmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:helloResponse xmlns:ns2="http://eggum/">
      <return>Hello Dag !</return>
    </ns2:helloResponse>
  </S:Body>
</S:Envelope>
```

Figure 4-8 Hello response

4.2.2 Upload NFFI data Web service

The data to be used in this test is a set of NFFI files (see Section 1.3 Scenario) used in a simulated exercise by FFI. NFFI is an *Interoperability Standard Interface* in the XML format, which may contain tracking, position, and status information about military units. The reason for using this data is that it simulates what data might be sent in a real military setting using Web services.

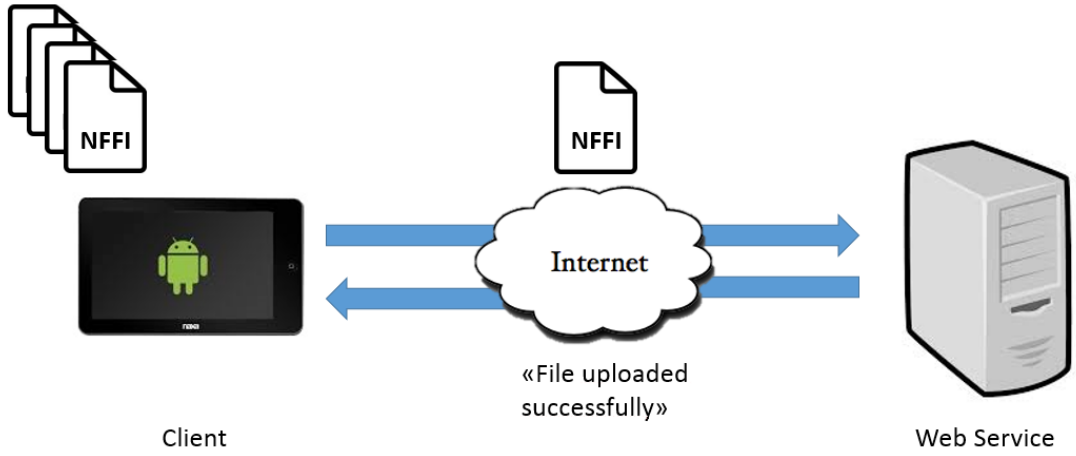

Figure 4-9: NFFI Test

The set consists of 20 different NFFI-files, which varies in size from 1 kB to 39 kB, and has a mean size of 15.8 kB. The files were sent repetitively; after sending file number 20, file number 1 was sent (and so on). Figure 4-10 shows the smallest file:

```
<NFFIMessage xmlns="urn:nato:fft:protocols:nffi13">
    <track>
        <positionalData>
            <trackSource>
                <sourceSystem>
                    <system>AFTS</system>
                </sourceSystem>
                <transponderId>RAY</transponderId>
            </trackSource>
            <dateTime>20070216094723</dateTime>
                <coordinates>
                    <latitude>35.4734755555556</latitude>
                    <longitude>67.5143733333333</longitude>
                </coordinates>
        </positionalData>
    </track>
</NFFIMessage>
```

Figure 4-10 Small NFFI-file used in testing

4.2.3 Exchange Picture Web service

In this test a single JPEG [84] picture file (approx. 40kB), previously taken with the camera of the Android device, is being sent back and forth between the client and the server for each Web service call. The reason for sending a JPEG file is to test both ksoap2-android and the compression techniques on non-XML data.

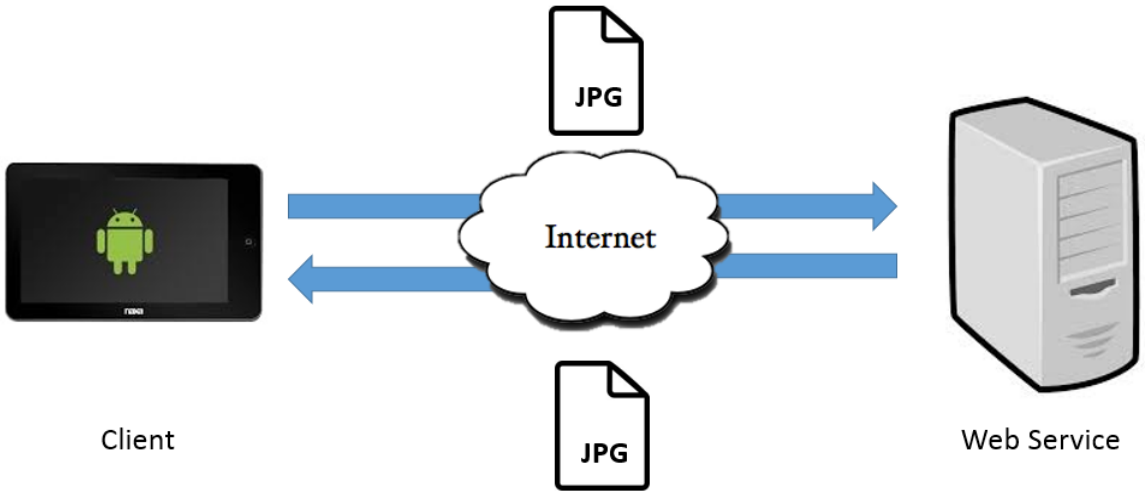

Figure 4-11 Picture Test

The test begins with the client getting the picture file from its storage, marshalling it into a SOAP envelope (possibly compressing it) and sending it to the server. The server sends the same picture data in the reply to the client. Upon receiving the reply, the client unmarshals the data (decompressing if needed) into a new picture file, which it saves on the device memory card. This procedure is then repeated for the duration of the test.

4.3 TEST MEASUREMENTS

The tests will measure differences in these three variables:

- CPU load caused by the different compression methods
- Battery usage
- Network load in the form of total amount of data sent over the network, and goodput[9].

The tests will be done both over Wi-Fi and over the mobile network.

4.3.1 CPU load

Of the two methods discussed in Section 4.1.1 (using DDMS Method profiling versus measuring how much time a method use) the second one was used in the test part of this thesis. In these tests the marshalling [10] and unmarshalling [11] times was measured in milliseconds, to give an impression of the CPU load of the different compression methods. This choice was done because this method is simpler and still gives sufficient results for this thesis. The compression time is included in the marshalling measurement, and the decompression time is included in the unmarshalling measurement.

4.3.2 Battery usage

It is not practical to monitor the battery status continuously; instead, the battery level was recorded before and after each test run. The battery levels was measured calling the battery level programmatically from the Android system.

4.3.3 Network load

Wireshark (as described in Section 4.1.2) were used to monitor the wireless traffic generated by the tests, measuring the total amount of data sent over the network.

---

[9] Goodput is the number of useful information bits delivered by the network to a certain destination per unit of time, in this thesis goodput is defined as how fast the exchange of SOAP envelopes are in megabits per second (Mbps).

[10] Marshalling is the process of transforming the memory representation of an object to a data format suitable for storage or transmission.

[11] Unmarshalling is the process of transforming data into the memory representation of an object.

4.4 TEST RESULTS

4.4.1 Test 1 & 2: Hello Web service

The Hello Web service was called 10 000 times for each configuration in each test, 60 000 calls in total.

	Test 1 - Mobile Network				**Test 2 - Wi-Fi**		
HTTP	***No c.***	***gzip***	***EXI***		***No c.***	***gzip***	***EXI***
Total Time (min)	61	56	60		6	6	9
Battery Drop (%)	3	3	3		0	0	1
Mean Marshall Time (ms)	12	21	31		12	15	27
Mean Roundtrip [12] Time (ms)	350	312	307		18	15	15
Mean Unmarshall Time ms)	6	2	15		4	3	11
Mean Call Time (ms)	368	335	353		34	33	53
Mbytes Transceived	16.95	15.23	13.72		16.92	15.21	13.69
Goodput (Mbps)	0.52	0.57	0.53		4.77	5.45	3.45
UDP	***No c.***	***gzip***	***EXI***		***No c.***	***gzip***	***EXI***
Total Time (min)	25	20	26		4	5	10
Battery Drop (%)	1	1	1		0	0	0
Mean Marshall Time (ms)	11	10	27		12	11	32
Mean Roundtrip Time (ms)	131	107	98		9	10	8
Mean Unmarshall Time (ms)	3	2	21		1	3	17
Mean Call Time (ms)	145	119	146		22	24	57
Mbytes Transceived	9.1	5.58	4.02		9.1	5.58	4.02
Goodput (Mbps)	1.29	1.55	1.2		7.12	6.54	3.17
RabbitMQ	***No c.***	***gzip***	***EXI***		***No c.***	***gzip***	***EXI***
Total Time (min)	30	25	25		5	5	9
Battery Drop (%)	1	1	2		1	0	0
Mean Marshall Time (ms)	15	15	33		11	16	33
Mean Roundtrip Time (ms)	154	134	105		10	9	9
Mean Unmarshall Time (ms)	1	2	11		3	1	6
Mean Call Time (ms)	171	151	149		24	26	48
Mbytes Transceived	12.77	9.2	7.91		12.6	9.19	7.87
Goodput (Mbps)	1.08	1.24	1.26		6.58	6.54	3.86

Table 4-1 Results from tests 1 & 2

[12] Roundtrip time is in this thesis defined as the time the app spends on transmitting the data, waiting for response, and reading the response.

4.4.2 Test 3 and 4: Upload NFFI data Web service

The Upload NFFI data Web service was called 10 000 times for each configuration in each test, 60 000 calls in total.

	Test 3 - Mobile Network				**Test 4 - Wi-Fi**		
HTTP	***No c.***	***gzip***	***EXI***		***No c.***	***gzip***	***EXI***
Total Time (min)	86	63	108		15	14	51
Battery Drop (%)	4	3	8		3	1	6
Mean Marshall Time (ms)	49	48	272		43	47	264
Mean Roundtrip Time (ms)	481	306	358		42	18	25
Mean Unmarshall Time ms)	2	2	14		5	3	8
Mean Call Time (ms)	512	356	644		90	68	297
Mbytes Transceived	266.93	46.26	42.1		264.5	45.92	41.82
Goodput (Mbps)	0.3	0.36	0.27		1.97	2.31	0.63
UDP	***No c.***	***gzip***	***EXI***		***No c.***	***gzip***	***EXI***
Total Time (min)	63	52	100		12	12	54
Battery Drop (%)	5	3	7		3	1	3
Mean Marshall Time (ms)	45	49	256		43	47	269
Mean Roundtrip Time (ms)	306	245	214		17	10	21
Mean Unmarshall Time (ms)	2	3	31		1	3	23
Mean Call Time (ms)	353	297	501		61	60	313
Mbytes Transceived	247.39	35.04	31.26		247.41	35.02	31.13
Goodput (Mbps)	0.51	0.61	0.29		2.73	2.73	0.6
RabbitMQ	***No c.***	***gzip***	***EXI***		***No c.***	***gzip***	***EXI***
Total Time (min)	55	40	79		12	12	57
Battery Drop (%)	4	3	7		1	1	4
Mean Marshall Time (ms)	47	50	292		45	49	277
Mean Roundtrip Time (ms)	269	173	157		22	13	47
Mean Unmarshall Time (ms)	2	3	11		1	2	7
Mean Call Time (ms)	318	226	460		68	64	331
Mbytes Transceived	264.01	40.47	36.29		259.92	40.04	36.21
Goodput (Mbps)	0.58	0.79	0.41		2.58	2.64	0.56

Table 4-2 Results from tests 3 & 4

4.4.3 Test 5 and 6: Exchange Picture Web service

The Exchange Picture Web service was called 5000 times for each configuration in each test, 30 000 calls in total.

	Test 5 - Mobile Network				Test 6 - Wi-Fi		
HTTP	***No c.***	***gzip***	***EXI***		***No c.***	***gzip***	***EXI***
Total Time (min)	84	80	136		17	18	77
Battery Drop (%)	4	6	8		1	2	5
Mean Marshall Time (ms)	75	105	645		77	93	640
Mean Roundtrip Time (ms)	801	795	1039		60	61	101
Mean Unmarshall Time (ms)	75	42	151		45	48	135
Mean Call Time (ms)	951	942	1835		182	202	876
Mbytes Transceived	613.18	460.33	456.42		610.1	456.92	452.99
Goodput (Mbps)	0.9	0.9	0.48		4.35	4.05	0.98
UDP	***No c.***	***gzip***	***EXI***		***No c.***	***gzip***	***EXI***
Total Time (min)	75	63	113		33	26	74
Battery Drop (%)	4	4	9		1	1	5
Mean Marshall Time (ms)	86	96	664		80	94	640
Mean Roundtrip Time (ms)	743	575	576		64	65	89
Mean Unmarshall Time (ms)	34	40	89		34	41	95
Mean Call Time (ms)	863	711	1329		178	200	824
Mbytes Transceived	576.87	429.87	425.58		585.23	433.03	426.36
Goodput (Mbps)	1	1.19	0.66		2.27	2.86	1.02
RabbitMQ	***No c.***	***gzip***	***EXI***		***No c.***	***gzip***	***EXI***
Total Time (min)	72	69	122		17	16	72
Battery Drop (%)	4	5	10		1	1	4
Mean Marshall Time (ms)	78	111	634		80	95	623
Mean Roundtrip Time (ms)	712	656	735		78	43	161
Mean Unmarshall Time (ms)	42	42	80		34	41	74
Mean Call Time (ms)	832	809	1449		192	179	858
Mbytes Transceived	608.45	453.51	450.81		599.49	446.34	447
Goodput (Mbps)	1.04	1.08	0.61		4.23	4.53	1.03

Table 4-3 Results from tests 5 & 6

4.4.4 Compression results

These compression results apply for using HTTP, when using UDP and RabbitMQ the original file sizes are a few bytes bigger because of the simulated WS-Addressing header at described in Section 4.2.

Hello Web service					
File	**No Comp (bytes)**	**gzip (bytes)**	**gzip (%)**	**EXI (bytes)**	**EXI (%)**
Request	336	199	59.2	128	48.5
Response	225	172	76.4	102	45.3

Table 4-4 Compression result of Hello Web service

Upload NFFI data Web service					
File no	**No Comp (bytes)**	**gzip (bytes)**	**gzip (%)**	**EXI (bytes)**	**EXI (%)**
1	878	562	64	481	54.8
2	944	618	65.5	540	57.2
3	3300	1171	35.5	1057	32
4	3406	1207	35.4	1091	32
5	5172	1962	37.9	1819	35.2
6	10310	2111	20.5	1864	18.1
7	19258	2541	13.2	2251	11.7
8	20498	2386	11.6	2142	10.4
9	20532	2945	14.3	2587	12.6
10	24128	2281	9.5	2070	8.6
11	24326	2166	8.9	1964	8.1
12	24450	2164	8.9	1974	8.1
13	24536	2558	10.4	2316	9.4
14	25920	3421	13.2	3102	12
15	28698	4236	14.8	3814	13.3
16	34166	4922	14.4	4469	13.1
17	36130	4099	11.3	3598	10
18	55070	6584	12	5886	10.7
19	55070	6496	11.8	5804	10.5
20	55070	6584	12	5886	10.7
NFFI Response	264	197	74.6	128	48.5

Table 4-5 Compression result of Upload NFFI data Web service

Exchange Picture Web service					
File	**No Comp (bytes)**	**gzip (bytes)**	**gzip (%)**	**EXI (bytes)**	**EXI (%)**
Upload	57195	42678	74.6	42000	73.4
Download	55188	41141	74.5	41041	74.4

Table 4-6 Compression result of Exchange Picture Web service

4.5 EVALUATION

The test results are not as concise as envisioned. Wireless networks are in their nature volatile, and even when trying to create stable conditions for testing one is bound to experience variations.

4.5.1 CPU load

The time spent on marshalling and unmarshalling was measured to give an impression of the CPU load of the different compression methods. The following figures are based on the results from mobile broadband testing, since the marshalling/unmarshalling times of mobile broadband were relatively similar to the Wi-Fi results.

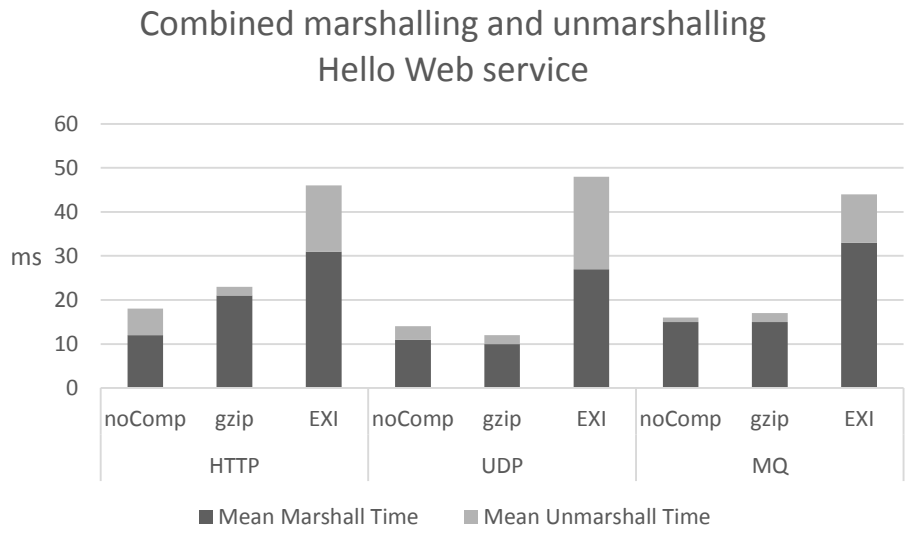

Figure 4-12 Combined Marshalling and unmarshalling Hello Web service

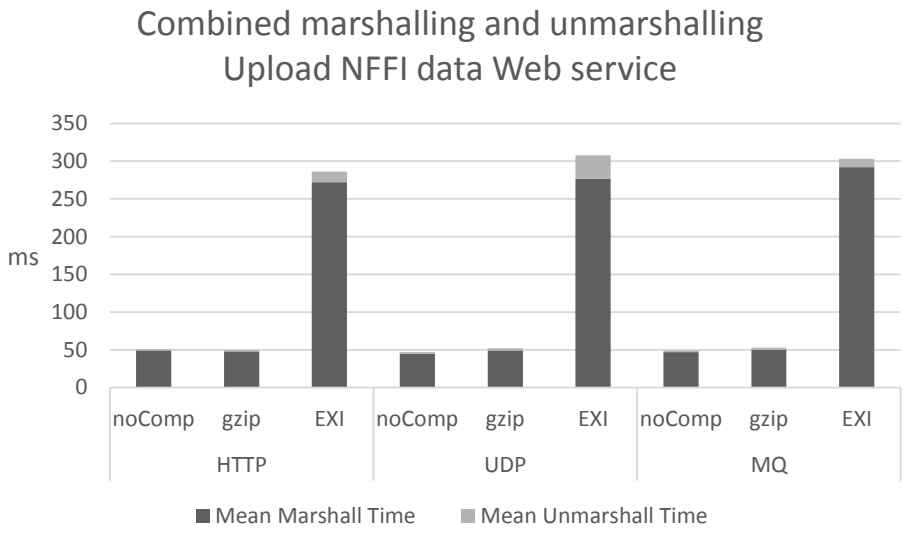

Figure 4-13 Combined Marshalling and unmarshalling Upload NFFI data Web service

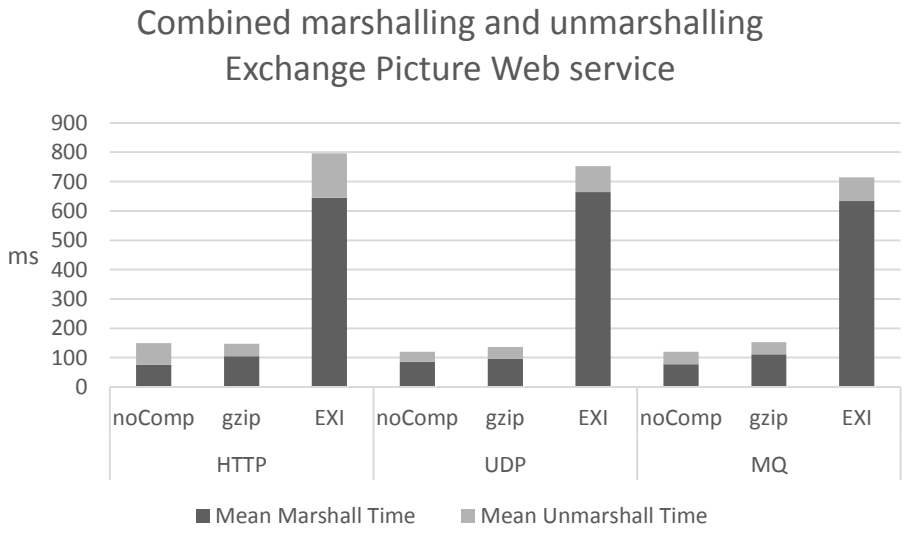

Figure 4-14 Combined marshalling and unmarshalling Exchange Picture Web service

From figures 4-12, 4-13 and 4-14 it is clear that the marshalling and unmarshalling times increase when compressing/decompressing with EXIficient. This can be seen in all tests, but is especially easy to see in the NFFI and Picture tests where the data amounts are greater. This can be interpreted as EXIficient being more CPU intensive than using no compression or gzip, since the CPU spends more time on marshalling and unmarshalling.

4.5.2 Battery usage

Measuring the battery drop for the different configurations was difficult, and many calls had to be done to see an effect on the battery. Another challenge was that it is not possible to measure the battery level with decimals. The Android API only offers an Integer value of the battery, making the ordeal more imprecise than wanted. There is no apparent way to determine if a drop of 2% in battery level is in reality 2.0% or 2.9%.

Figure 4-15 illustrates the battery drop of all six tests combined, differencing between the compression and transport parameters.

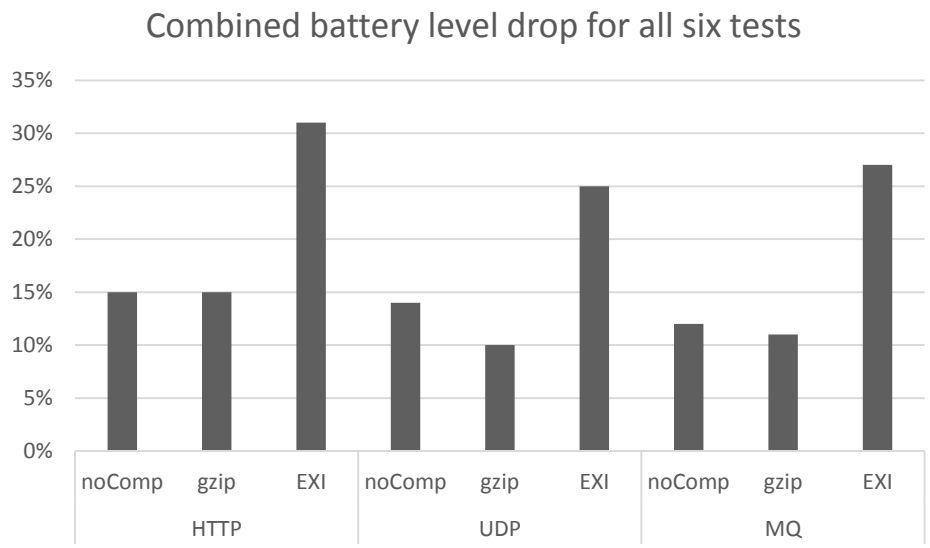

Figure 4-15 Combined battery level drop for all six tests

Figure 4-15 shows a clear trend of EXIficient using more battery than the No compression and gzip options. The differences between No compression and gzip are small, compared with EXIficient. The figure also shows that UDP and MQ generally drains less battery than HTTP does.

4.5.3 Network load

In the following figures, the total amount of data sent over the mobile network is presented in percentages, combining the requests and responses of all Web service calls. The results from the calls done over Wi-Fi are not presented since they have approximately the same size.

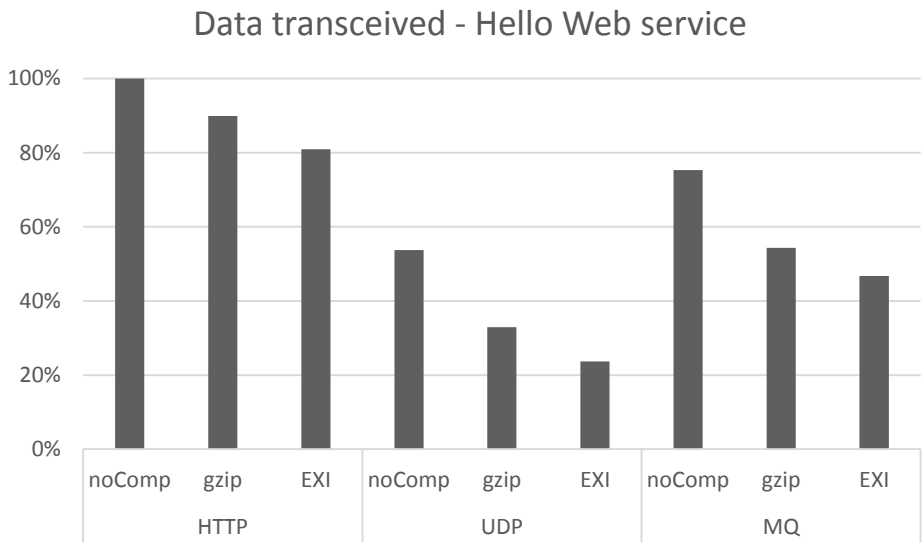

Figure 4-16 Data transceived - Hello Web service

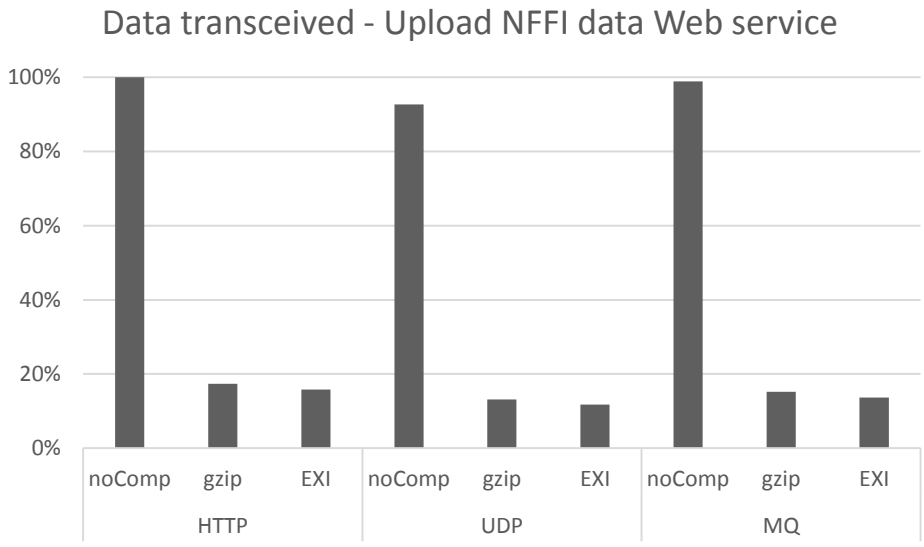

Figure 4-17 Data transceived - Upload NFFI data Web service

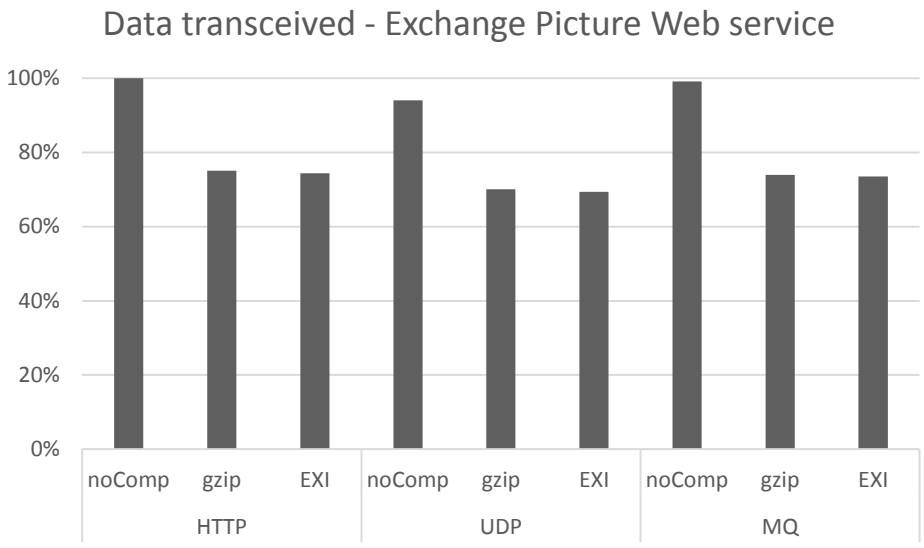

Figure 4-18 Data transceived - Exchange Picture Web service

Figure 4-16 shows that EXIficient compresses better than gzip when the request and response messages are small. In Figure 4-17 and Figure 4-18, where the data amount is larger, there is no big difference between using EXIficient or gzip.

The figures also show that using a different transport method than HTTP does not affect the total amount of data much when the messages are large (Figure 4-17 and Figure 4-18).

4.5.4 Goodput

In these tests, goodput is defined as how fast the exchange of SOAP envelopes are in megabits per second (Mbps).

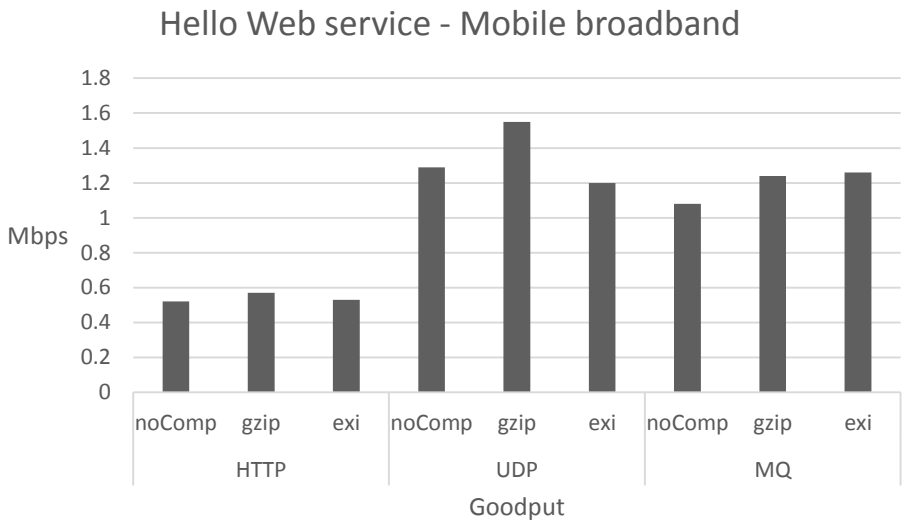

Figure 4-19 Goodput Hello Web service - Mobile broadband

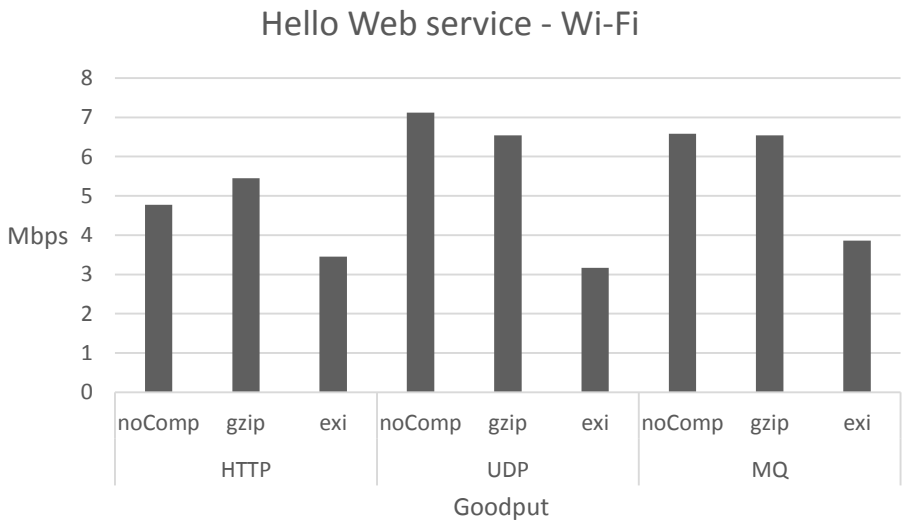

Figure 4-20 Goodput Hello Web service - Wi-Fi

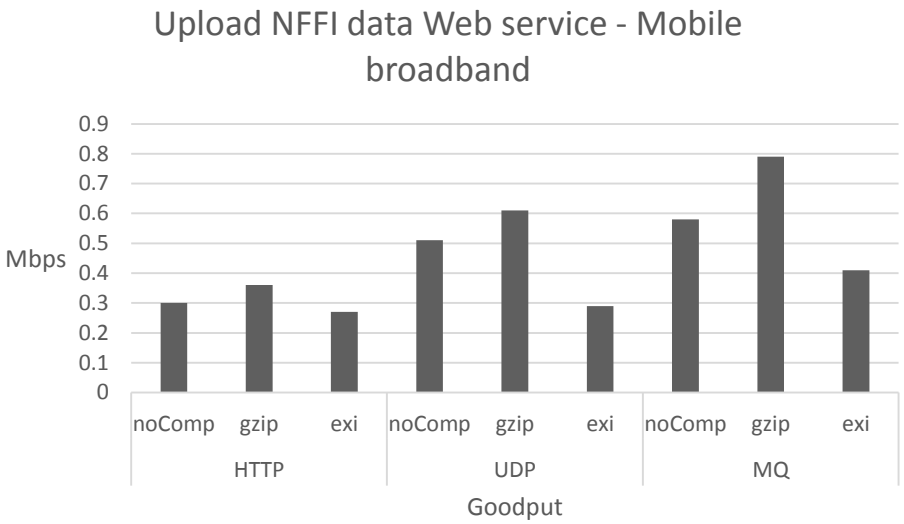

Figure 4-21 Goodput Upload NFFI data Web service - Mobile broadband

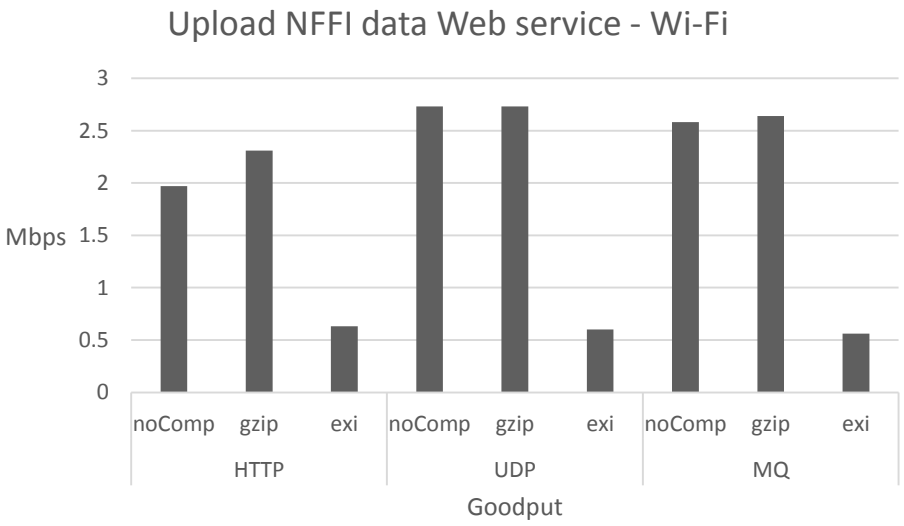

Figure 4-22 Goodput Upload NFFI data Web service - Wi-Fi

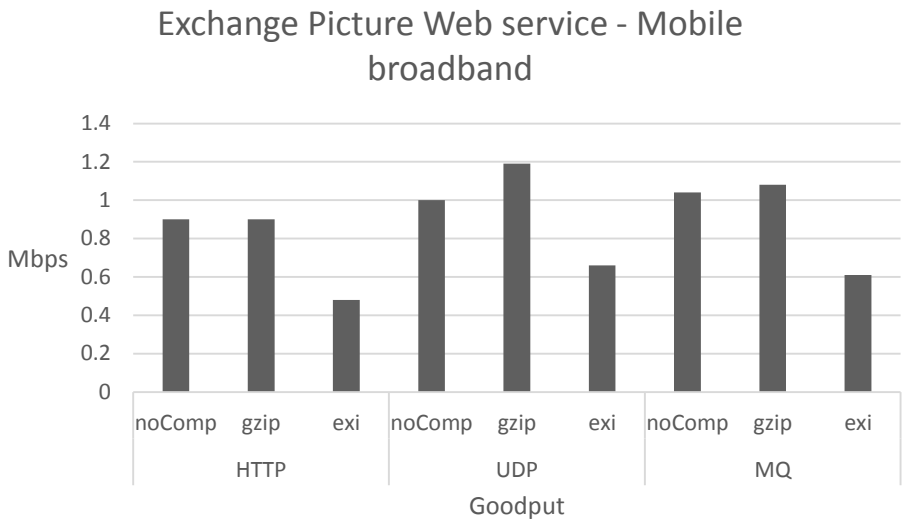

Figure 4-23 Goodput Exchange Picture Web service - Mobile broadband

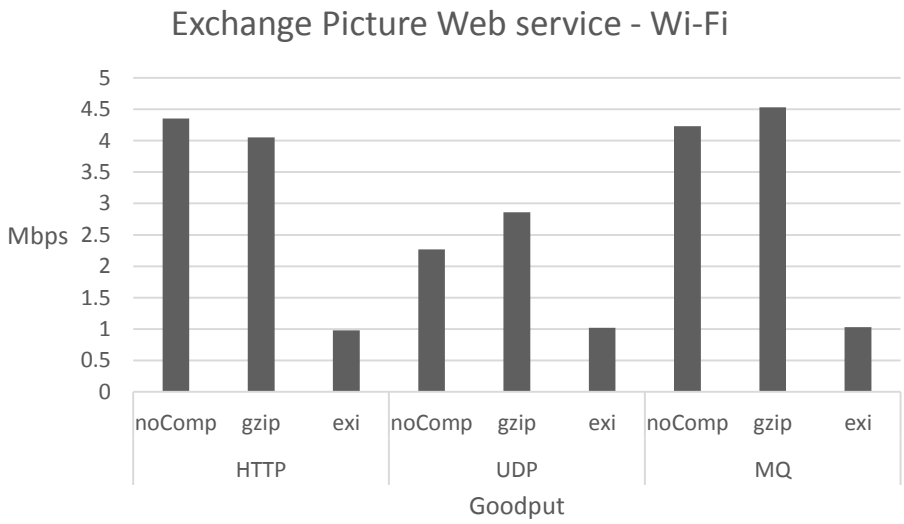

Figure 4-24 Goodput Exchange Picture Web service - Wi-Fi

When measuring goodput, UDP and RabbitMQ are generally more effective than HTTP. The UDP results shown in Figure 4-24 is deviating from the rest of the results. The reason for this is that calling the Exchange Picture Web service with UDP over Wi-Fi caused many dropped packets, causing the UDP results in this test to degrade. The reason this happened might be bad programming, or perhaps more likely, because of packet loss due to congestion in the networks.

4.5.5 Comparing gzip and EXIficient compression

This section illustrated the size of the original files compared to the gzip- and EXIficient-compressed files.

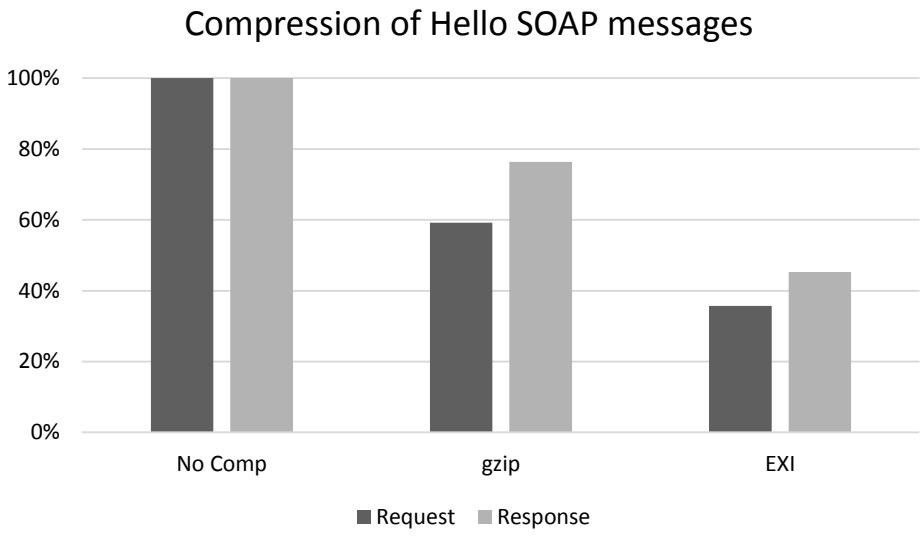

Figure 4-25 Compression of Hello SOAP messages

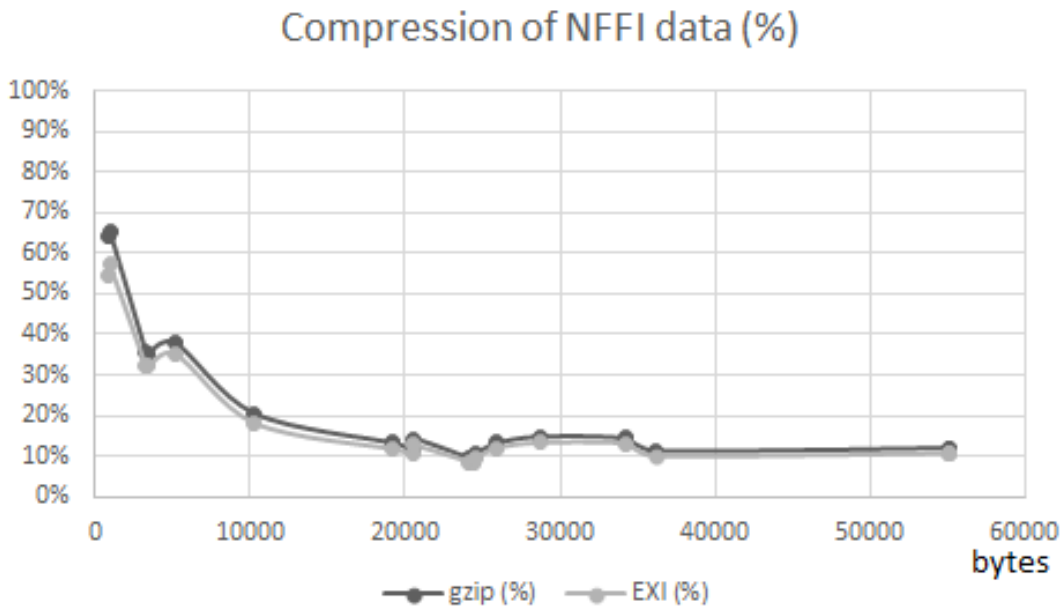

Figure 4-26 Compression of NFFI data

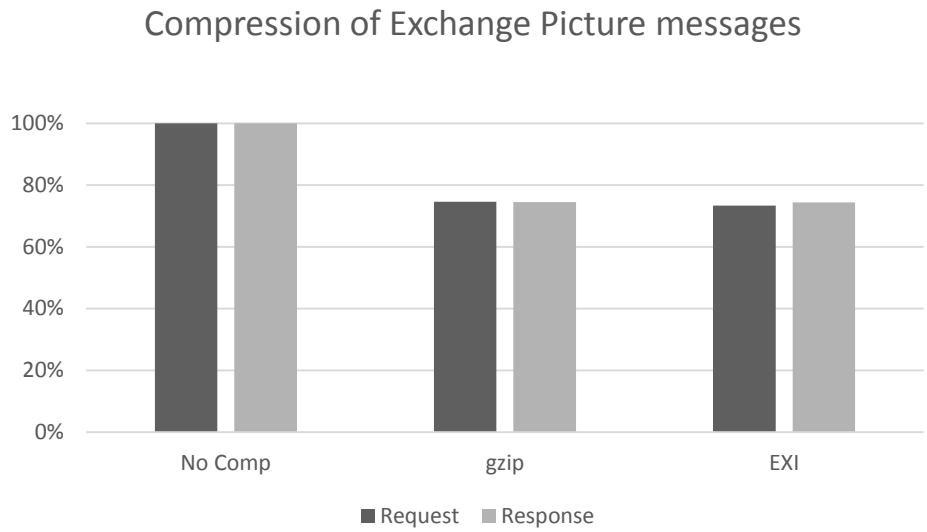

Figure 4-27 Compression of Exchange Picture messages

Figure 4-25 shows that EXIficient compresses much better than gzip when the SOAP messages are small, such as in the Hello Web service. Figure 4-26 shows that when the SOAP messages are bigger, such as in the Upload NFFI data Web service, the difference is not that big. Figure 4-27 shows that in the case of the JPG picture gzip and EXIficient had roughly the same performance level.

4.6 SUMMARY

This section discusses the observed results in regards to the requirements specification in 2.1.3.

EXIficient performed badly in regards of CPU load, the marshalling and unmarshalling times of EXIficient were much higher than when using gzip or no compression.

When focusing on maximizing battery lifetime EXIficient suffered greatly compared to using gzip or not compressing at all. The results also show that UDP and AMQP preserve more battery life than HTTP does.

When focusing on minimizing network load, using either gzip or EXIficient to compress and decompress greatly minimized the data amount (to less than 20%) when dealing with large XML files (Upload NFFI data Web service). The effect was not so good when dealing with small XML files (Hello Web service) or with non-XML data (Exchange Picture Web service). Since the JPG format is a compressed format, trying to compress it a second time did not cause a big effect.

The test results also show that UDP and AMQP performed generally better than HTTP did, especially when considering goodput.

5 CONCLUSION AND FUTURE WORK

This chapter presents the conclusion of this thesis, as well as suggestions for future work.

5.1 CONCLUSION

The goal of this thesis is to contribute to the ongoing effort to optimize SOAP communication, specifically for use Android platforms. The main part of this contribution is to evaluate the use of alternative transport mechanisms, as well as different compression techniques, up against bandwidth usage and battery usage. The goal is to recommend a path or solution for supporting SOAP on Android.

The results in this thesis show that using a different transport mechanism other than the standard HTTP/TCP, like UDP or AMQP, can both reduce battery usage and help increase the goodput when using Android as a Web service client.

When it comes to compression and decompression of SOAP messages, the results show that both gzip and EXIficient help reduce the network load, but EXIficient demands more CPU time and battery power than gzip does. It can be argued that the resource usage EXIficient demands can be too high compared to the size reduction EXIficient offers, at least in the experiment of this thesis. Gzip compresses almost as well as EXIficient, using far less CPU time and battery power.

Thus, the recommendation of this thesis is that using gzip together with RabbitMQ is the better option when it comes to reducing network overhead while simultaneously maximizing battery lifetime.

ksoap2-android proves to be a functional and lightweight SOAP library that can be altered to support different transport mechanisms and compression techniques on Android. The code changes of ksoap2-android in this thesis will be offered to the ksoap2-android community, since it is a user-driven community dependent on individual contributions.

The test log files, Wireshark captures and data used for measurement can be found in the data provided together with the thesis.

5.2 Future Work

While ksoap2android works as a SOAP library for the Android platform, among the drawbacks are its lack of support for JAXB and WS-Addressing, requiring the programmer to do more manual programming compared to other Java-based SOAP libraries. Still, as long as Android does not provide a SOAP library of its own in the near future, ksoap2-android is a viable option. ksoap2-android should be expanded with WS-Addressing to support other transport protocols, and should have an alternative to JAXB in order to be more user-friendly.

Implementing SCTP on Android is achievable (see Appendix A), and would be interesting to compare to other transport protocols when possible.

Further testing with the solution presented in this thesis is also possible, with for example adding worse network conditions with more errors and disruptions.

REFERENCES

1. Koetsier, J., *Android captures record 81% global market share, Windows Phone is 'fastest growing'*. 2013 [Accessed 28 April 2014]; Published by: Venturebeat. Available from: http://venturebeat.com/2013/10/31/android-captures-record-81-global-market-share-windows-phone-is-fastest-growing/
2. OASIS, *Reference Model for Service Oriented Architecture 1.0*. 2006 [Accessed 28 April 2014]; Published by: OASIS. Available from: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf
3. W3C Web Services Architecture Working Group, *Web Services Architecture*. 2004 [Accessed 28 April 2014]; Published by: World Wide Web Consortium. Available from: http://www.w3.org/TR/ws-arch/
4. World Wide Web Consortium, *SOAP Version 1.2*. 2007 [Accessed 28 April 2014]; Published by: World Wide Web Consortium. Available from: http://www.w3.org/TR/soap12-part1/
5. World Wide Web Consortium, *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. 2008 [Accessed 28 April 2014]; Published by: World Wide Web Consortium. Available from: http://www.w3.org/TR/REC-xml/
6. Busch, J., *Deploying Web Services: Bringing NNEC To The Edge*. 2006 [Accessed 28 April 2014]; Published by: NATO Consultation, Command and Control Agency. Available from: https://www.google.no/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0CDMQFjAB&url=http%3A%2F%2Fnc3a.info%2FP%2Fpres%2FDeploying%2520Web%2520Services%2520for%2520MCC%25202006%2520-%2520distributable.ppt&ei=Eds5UrKiA82P4gSTs4HADw&usg=AFQjCNHfB4OTEJ1foZm9jv75HXZQRb0pYA&bvm=bv.52288139,d.bGE
7. Wi-Fi Alliance, *Wi-Fi*. 2014 [Accessed 28 April 2014]; Published by: Wi-Fi Alliance. Available from: http://www.wi-fi.org/who-we-are
8. Simanta, S., D. Plakosh, and E. Morris, *Web Services for Handheld Tactical Systems* in *Systems Conference (SysCon), 2011 IEEE International*. 4-7 April 2011: Montreal, QC, Canada.
9. Johnsen, F.T., et al., *Interoperable service discovery: Experiments at Combined Endeavor 2009* Forsvarets forskningsinstitutt/Norwegian Defence Research Establishment (FFI), 2009/01934.
10. Johnsen, F.T., et al., *Evaluation of Transport Protocols for Web Services*, in *Military Communications and Information Systems Conference (MCC)*. 7-9 Oct. 2013: St.-Malo, France.
11. Fielding, R., *Hypertext Transfer Protocol -- HTTP/1.1*. 1999 [Accessed 28 April 2014]; Published by: Internet Engineering Task Force. Available from: http://tools.ietf.org/html/rfc2616
12. Information Sciences Institute University of Southern California, *Transmission Control Protocol*. 1981 [Accessed 28 April 2014]; Published by: The Internet Engineering Task Force. Available from: http://tools.ietf.org/html/rfc793
13. Johnsrud, L., *Efficient Web Services on Mobile Devices*, Master's thesis at the Department of Telematics, Norwegian University of Science and Technology, 2007
14. Denning, P.J., et al., *Computing as a discipline.* Magazine Communications of the ACM, 1989. Volume: 32(1): p. 9-23.
15. Elgin, B., *Google Buys Android for Its Mobile Arsenal*. 2005 [Accessed 28 April 2014]; Published by: Bloomberg Businessweek. Available from: http://www.webcitation.org/5wk7sIvVb

16. Open Handset Alliance, *Industry Leaders Announce Open Platform for Mobile Devices*. Open Handset Alliance: http://www.openhandsetalliance.com/press_110507.html.
17. Android Engineering Application & Consulting Services Team, *Android Anatomy Physiology*. 2012 [Accessed 28 April 2014]; Published by: Android Engineering Application & Consulting Services Team. Available from: http://androidteam.googlecode.com/files/Anatomy-Physiology-of-an-Android.pdf
18. Delap, S., *Google's Android SDK Bypasses Java ME in Favor of Java Lite and Apache Harmony*. 2007 [Accessed 28 April 2014]; Published by: Infoq. Available from: http://www.infoq.com/news/2007/11/android-java
19. Mason, T.R., C.H. Weaver, and M.A. Camacho, *Position, navigation, and timing in the Common Operating Environment: Prototyping the PNT User equipment Modernization Architecture*, in *Military Communications Conference, 2012*. 29 Oct. -1 Nov. 2012: Orlando, FL, USA.
20. Weiss, B.A., et al. *Performance assessments of Android-powered military applications operating on tactical handheld devices*. in *Mobile Multimedia/Image Processing, Security, and Applications 2013*. April 29, 2013. Baltimore, Maryland, USA.
21. Karlsen, L.H. and B.K. Reitan, *CEI - et sosialt taktisk rapporteringssystem: Teknisk beskrivelse av Android klient for smarttelefon og nettbrettstøttet til CEI-systemet* Forsvarets forskningsinstitutt (FFI), 2014/00526
22. Fielding, R.T., *Architectural Styles and the Design of Network-based Software Architectures*, Doctoral thesis at the University of California, Irvine, 2000
23. Web Services Architecture Working Group, *Web Services Glossary*. 2004 [Accessed 28 April 2014]; Published by: World Wide Web Consortium. Available from: http://www.w3.org/TR/ws-gloss/
24. Berners-Lee, T., *Web Services - Program Integration across Application and Organization boundaries*. 2009 [Accessed 28 April 2014]; Published by: World Wide Web Consortium. Available from: http://www.w3.org/DesignIssues/WebServices.html
25. W3Schools, *Introduction to WSDL*. 2014 [Accessed 28 April 2014]; Published by: World Wide Web Consortium. Available from: http://www.w3schools.com/webservices/ws_wsdl_intro.asp
26. Curbera, F., et al., *Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI.* Internet Computing, IEEE March-April 2002. Volume: 6(2): p. 86 - 93.
27. Klensin, J., *Simple Mail Transfer Protocol*. 2008 [Accessed 28 April 2014]; Published by: The Internet Engineering Task Force. Available from: http://tools.ietf.org/html/rfc5321
28. OASIS, *Advanced Message Queuing Protocol (AMQP) Technical Committee*. 2014 [Accessed 28 April 2014]; Published by: OASIS. Available from: https://www.oasis-open.org/committees/amqp/charter.php
29. Tekli, J.M., et al., *SOAP Processing Performance and Enhancement.* Services Computing, IEEE Transactions on 2012. Volume: 5(3): p. 387 - 403.
30. Deutsch, P., *DEFLATE Compressed Data Format Specification version 1.3*. 1996 [Accessed 28 April 2014]; Published by: Internet Engineering Task Force. Available from: http://tools.ietf.org/html/rfc1951
31. Katz, P., *ZIP File Format, Version 6.2.0 (PKWARE)*. 2012 [Accessed 28 April 2014]; Published by: Sustainability of Digital Formats Planning for Library of Congress Collections. Available from: http://www.digitalpreservation.gov/formats/fdd/fdd000355.shtml
32. Deutsch, P., *gzip file format specification version 4.3*. 1996 [Accessed 28 April 2014]; Published by: Internet Engineering Task Force. Available from: http://tools.ietf.org/html/rfc1952

33. Rand, D., *The PPP Compression Control Protocol (CCP)*. 1996 [Accessed 28 April 2014]; Published by: Internet Engineering Task Force. Available from: http://tools.ietf.org/html/rfc1962
34. Boutell, T., *PNG (Portable Network Graphics) Specification Version 1.0*. 1997 [Accessed 28 April 2014]; Published by: Internet Engineering Task Force. Available from: http://tools.ietf.org/html/rfc2083
35. Adobe Systems Incorporated, *Adobe Portable Document Format*. 2006 [Accessed 28 April 2014]; Published by: Adobe Systems Incorporated. Available from: http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf
36. Deutsch, P., *zlib Compressed Data Format Specification version 3.3*. 1996 [Accessed 28 April 2014]; Published by: Internet Engineering Task Force. Available from: http://tools.ietf.org/html/rfc1950
37. Salomon, D., *A Concise Introduction to Data Compression*. 2007: Springer. ISBN: 1848000723, 9781848000728
38. Nair, S.S., *XML compression techniques: A survey.* Journal of Computer and System Sciences, 2009. Volume: 75(5).
39. World Wide Web Consortium, *Efficient XML Interchange (EXI) Format 1.0*. 2011 [Accessed 28 April 2014]; Published by: World Wide Web Consortium. Available from: http://www.w3.org/TR/exi/
40. AgileDelta, *Lightning-Fast Delivery of XML to More Devices in More Locations*. 2011 [Accessed 28 April 2014]; Published by: AgileDelta. Available from: http://www.agiledelta.com/product_efx.html
41. International Organization for Standardization, *Fast Infoset*. 2007 [Accessed 28 April 2014]; Published by: International Organization for Standardization. Available from: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=41327
42. Telecommunication Standardization Sector, *Fast Infoset*. 2010 [Accessed 4 Feb. 2014]; Published by: International Telecommunications Union. Available from: http://www.itu.int/ITU-T/asn1/xml/finf.htm
43. XMILL Inc., *XMILL*. 2001 [Accessed 28 April 2014]; Published by: XMILL Inc. Available from: http://homes.cs.washington.edu/~suciu/XMLTK/xmill/www/
44. Suciu, D. and H. Liefke, *XMill: An Efficient Compressor for XML*. 2004 [Accessed 4 Feb. 2014]; Published by: XMill. Available from: http://www.liefke.com/hartmut/xmill/xmill.html
45. World Wide Web Consortium, *WAP Binary XML Content Format*. 1999 [Accessed 28 April 2014]; Published by: Open Mobile Alliance. Available from: http://www.w3.org/TR/wbxml/
46. Augeri, C.J., et al. *An Analysis of XML Compression Efficiency*. in *2007 Workshop on Experimental Computer Science (ExpCS)*. 2007. New York, NY, USA.
47. Teixeira, M.A., et al., *New Approaches for XML Data Compression*, in *International Conference on Web Information Systems and Technologies*. 2012: Paper presented at the meeting of the WEBIST, 2012. p. 233-237.
48. Jaiswal, G. and M. Mishra, *Why use Efficient XML Interchange instead of Fast Infoset*, in *Advance Computing Conference (IACC), 2013 IEEE 3rd International*. 22-23 Feb. 2013: Ghaziabad, India. p. 925 - 930.
49. World Wide Web Consortium, *SOAP Protocol Binding Framework*. 2007 [Accessed 2 Feb. 2014]; Published by: World Wide Web Consortium. Available from: http://www.w3.org/TR/soap12-part1/#transpbindframew

50. Information Sciences Institute University of Southern California, *Internet Protocol*. 1981 [Accessed 28 April 2014]; Published by: The Internet Engineering Task Force. Available from: http://www.ietf.org/rfc/rfc791.txt
51. Microsoft Developer Network, *One-Way Method Invocation Using SOAP Over TCP*. 2014 [Accessed 28 April 2014]; Published by: Microsoft. Available from: http://msdn.microsoft.com/en-us/library/cc237457.aspx
52. Postel, J., *User Datagram Protocol*. 1980 [Accessed 28 April 2014]; Published by: The Internet Engineering Task Force. Available from: http://www.ietf.org/rfc/rfc768.txt
53. OASIS, *SOAP-over-UDP Version 1.1*. 2009 [Accessed 28 April 2014]; Published by: OASIS. Available from: http://docs.oasis-open.org/ws-dd/soapoverudp/1.1/os/wsdd-soapoverudp-1.1-spec-os.html
54. Tari, Z., et al., *Benchmarking SOAP Binding*, in *On the Performance of Web Services*. 2011. p. 35-58.ISBN: 978-1-4614-1930-3
55. Stewart, R., *Stream Control Transmission Protocol*. 2000 [Accessed 28 April 2014]; Published by: Internet Engineering Task Force. Available from: http://www.ietf.org/rfc/rfc2960.txt
56. Stewart, R., *Stream Control Transmission Protocol*. 2007 [Accessed 2 Feb. 2014]; Published by: Internet Engineering Task Force. Available from: http://tools.ietf.org/html/rfc4960
57. OASIS Advanced Message Queuing Protocol (AMQP) Bindings and Mappings (AMQP-BINDMAP) Technical Committee, *SOAP Binding to Advanced Message Queuing Protocol (AMQP) Transport Version 1.0*. 2013 [Accessed 28 April 2014]; Published by: OASIS. Available from: https://www.oasis-open.org/committees/download.php/50611/amqp-soap-v1.0-wd01.doc
58. Balakrishnan, H., et al., *A comparison of mechanisms for improving TCP performance over wireless links.* IEEE/ACM Transactions on Networking, 1997. Volume: 5(6).
59. Holland, G. and N. Vaidya, *Analysis of TCP performance over mobile ad hoc networks.* Wireless Networks - Selected Papers from Mobicom'99, 2002. Volume: 8.
60. Gábor, A., *AndroidSOAP*. 2014 [Accessed 28 April 2014]; Published by: Confluence. Available from: http://wiki.javaforum.hu/display/ANDROIDSOAP/Home
61. NeuroSpeech Inc, *WSClient++*. 2014 [Accessed 28 April 2014]; Published by: NeuroSpeech Inc. Available from: http://wsclient.neurospeech.com/wsclient/java-android-blackberry/
62. The ksoap2-android project, *ksoap2-android*. 2014 [Accessed 28 April 2014]; Published by: The ksoap2-android project. Available from: https://code.google.com/p/ksoap2-android/
63. Geeknet, I., *kSOAP2*. 2009 [Accessed 28 April 2014]; Published by: SourceForge. Available from: http://ksoap2.sourceforge.net/index.shtml
64. Shen, Z., et al., *A Light Mobile Web Service Framework Based on Axis2*, in *Future Information Communication Technology and Applications*, J.T.K. Hoe-Kyung Jung, Tony Sahama, Chung-Huang Yang, Editor. 2013.ISBN: 978-94-007-6515-3
65. Johnsen, F.T., et al., *IST-118 – SOA recommendations for Disadvantaged Grids in the Tactical Domain* in *18th Internation Command and Control Research and Technology Symposium*. 19-21 June 2013: Alexandria, VA, USA.
66. Android Open Source Project - Issue Tracker, *Android Open Source Project - Issue 3272: Support SCTP* 2013 [Accessed 28 April 2014]; Published by: Android. Available from: https://code.google.com/p/android/issues/detail?id=3272
67. Soderman, P., et al., *Sub-second transport layer vertical handover using mSCTP in android mobile devices*, in *Wireless Communication Systems (ISWCS), 2012 International Symposium on*. 28-31 Aug. 2012: Paris, France.

68. Tachibana, A. and T. Hasegawa, *A deployable scheme of CMT-SCTP with off-the-shelf android smartphones*, in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 IEEE 8th International Conference on*. 8-10 Oct. 2012: Barcelona, Spain.
69. Ege, R.K., *Securing Video Delivery to the Android Platform.* Journal of Systemics, Cybernetics & Informatics, 2012. Volume: 10(5).
70. Haustein, S., *About kXML*. 2005 [Accessed 28 April 2014]; Published by: SourceForge.net. Available from: http://kxml.sourceforge.net/about.shtml
71. The Apache Software Foundation, *What is Pull Parsing?* 2012 [Accessed 28 April 2014]; Published by: The Apache Software Foundation. Available from: http://ws.apache.org/axiom/userguide/ch01.html
72. Box, D., *Web Services Addressing (WS-Addressing)*. 2004 [Accessed 28 April 2014]; Published by: World Wide Web Consortium. Available from: http://www.w3.org/Submission/ws-addressing/
73. Oracle, *The Java Remote Method Invocation (RMI)*. 2014 [Accessed 28 April 2014]; Published by: Oracle. Available from: http://docs.oracle.com/javase/tutorial/rmi/
74. Java Community Process, *JAXB*. 2014 [Accessed 28 April 2014]; Published by: Java Community Process. Available from: https://jcp.org/en/jsr/detail?id=222
75. Android, *Android Development Tools*. 2014 [Accessed 28 April 2014]; Published by: Android. Available from: http://developer.android.com/tools/sdk/eclipse-adt.html
76. The Eclipse Foundation, *Eclipse*. 2014 [Accessed 28 April 2014]; Published by: The Eclipse Foundation. Available from: https://www.eclipse.org/
77. RabbitMQ, *Downloading and Installing RabbitMQ*. 2014 [Accessed 28 April 2014]; Published by: Pivotal. Available from: https://www.rabbitmq.com/download.html
78. Peintner, D., *EXIficient*. 2013 [Accessed 28 April 2014]; Published by: Sourceforge.net. Available from: http://exificient.sourceforge.net/
79. Oracle, *Class HttpServer*. 2005 [Accessed 28 April 2014]; JavaSE 7:[Published by: Oracle. Available from: http://docs.oracle.com/javase/7/docs/jre/api/net/httpserver/spec/com/sun/net/httpserver/HttpServer.html
80. Siemens AG, *exificient-0.9.2-bundle*. 2014 [Accessed 28 April 2014]; Published by: Sourceforge. Available from: http://sourceforge.net/projects/exificient/files/exificient/0.9.2/exificient-0.9.2-bundle.zip/stats/map
81. Flores, M., M. Balao, and F. Faggiani, *Xerces for Android*. 2014 [Accessed 28 April 2014]; Published by: Google Project Hosting. Available from: https://code.google.com/p/xerces-for-android/
82. Android, *Using DDMS*. 2014 [Accessed 28 April 2014]; Published by: Android. Available from: http://developer.android.com/tools/debugging/ddms.html
83. Combs, G., *Wireshark*. 2014 [Accessed 28 April 2014]; Published by: Wireshark. Available from: http://www.wireshark.org/
84. Hamiltion, E., *JPEG File Interchange Format Version 1.02*. 1992 [Accessed 28 April 2014]; Published by: Joint Photographic Experts Group. Available from: http://www.jpeg.org/public/jfif.pdf
85. Kommunikationsnetz Franken, *The SCTP library (sctplib)*. 2014 [Accessed 28 April 2014]; Published by: Kommunikationsnetz Franken. Available from: http://www.sctp.de/sctp-download.html

86. Sctp-refimpl, *usrsctp*. 2014 [Accessed 28 April 2014]; Published by: Sctp-refimpl. Available from: https://code.google.com/p/sctp-refimpl/source/browse/#svn%2Ftrunk%2FKERN%2Fusrsctp
87. Simula School of Research and Innovation AS, *A state of the art userland stack for SCTP*. 2014 [Accessed 28 April 2014]; Published by: Simula. Available from: https://simula.no/education/ssri/master-opportunities/available-master-topics/copy_of_communication-systems/a-state-of-the-art-userland-stack-for-sctp

APPENDIX A – ATTEMPT AT ENABLING SCTP ON ANDROID

In the time of writing Android does not offer the ability to use the Stream Control Transmission Protocol. The Linux kernel has had built-in support for the SCTP protocol since the 2.6 kernel series. The first Android version was built on the Linux kernel version 2.6.26, meaning that the developers behind Android made a choice not to include SCTP as an alternative transport layer protocol, and has stood by that decision to this day.

The Android Open Source Project maintains a public issue tracker where you can report bugs and request features for the core Android software stack. In 2009 a request was made to the issue tracker to offer SCTP on the Android platform, but five years later the status has not changed from "new", and nobody from Android have replied to the request [66].

It is possible to enable SCTP on Android by rebuilding the Linux kernel [13] on an Android device. Because SCTP is an interesting alternative to TCP and UDP, several experiments have been done implementing and testing SCTP on Android. *Securing Video Delivery to the Android Platform* [67] proposes using SCTP on Android for vertical handover between heterogeneous wireless networks like Wi-Fi and cellular 3G and 4G networks. *A Deployable Scheme of CMT-SCTP with Off-the-Shelf Android Smartphones* [68] also describes using SCTP on Android for seamless handover between networks. *Sub-Second Transport Layer Vertical Handover Using mSCTP in Android Mobile Devices* [69] describes a prototype Android app which accesses multimedia data over SCTP.

The experiment done in [68] is of interest because SCTP on Android "*…is accomplished by neither embedding special modules nor relying on super-user (root) privileges on off-the-shelf Android…*". The middleware used was based on the *sctplib* [85], a SCTP library implementation written in the programming language C.

Because the sctplib library is written in C, it cannot be added directly the way libraries written in Java (for example ksoap-2android, RabbitMQ, EXIficient) can. Android refers to its C libraries as *native code*. In order to implement native code such as C and C++ in an Android app a toolset called Android Native Development Kit (NDK) needs to be used. When adding a third-party library written in C or C++, NDK needs to compile the library into required machine code.

Trying to compile the latest version (1.0.15) of sctplib into the Android application used in this thesis failed. The reason for that is that sctplib is written for Unix and Linux platforms,

[13] Rebuilding a kernel means downloading the kernel source code, making any wanted changes in the code, compiling the code, and finally installing the kernel. Since Linux is open source, this is popular among programmers and developers to do.

while Android uses its own derivation of C libraries (see Section 2.1.1). This difference resulted in problems with missing classes and data constants.

Contacting the authors of sctplib, they provided some guidance and support via e-mail to resolve the problems, but ultimately the attempt had to be stopped because the sctplib library requires opening a raw socket[14]. This is not possible to do without rooting the Android device, and not rooting the device was one of the premises stated in the requirements specification. The authors of sctplib also suggested trying *usrsctp [86]*, a different SCTP library, but trying to implement usrsctp also resulted in similar problems as sctplib did.

Trying to contact the authors of [68] for assistance was fruitless, as they did not reply to my e-mails. In the end, the limitations on time and resources forced the attempt to include SCTP in this thesis to be suspended.

Given more time and resources, a manageable SCTP library for Android could most likely be created. There is certainly interest in the SCTP protocol. Actually, the authors of sctplib mentioned that there is an offer for a student project to extend the usrsctp API [87].

[14] A raw socket is an internet socket that allows direct sending and receiving of Internet Protocol packets without any protocol-specific transport layer formatting.