ISSN 2320 - 2602

Volume 7 No.9, September 2018
International Journal of Advances in Computer Science and Technology
Available Online at http://www.warse.org/IJACST/static/pdf/file/ijacst01792018.pdf
https://doi.org/10.30534/ijacst/2018/01792018

Efficient Simple Object Access Protocol (SOAP) Messaging for Mobile Devices in Android Platform

Talaam K. Obadiah[1], George O. Okeyo[2], Michael W. Kimwele[3]
[1] Jomo Kenyatta University of Agriculture and Technology, Kenya, obadiahtalaam@gmail.com
[2] Jomo Kenyatta University of Agriculture and Technology, Kenya, gokeyo@jkuat.ac.ke
[3] Jomo Kenyatta University of Agriculture and Technology, Kenya, kimwele@icsit.jkuat.ac.ke

ABSTRACT

Service Oriented Architecture, and its most common implementation method Web services, has not seen widespread use on wireless mobile systems and smart devices that are characterized by less computational resources such as small computing devices and limited power and wireless networks characteristics like low bandwidth which is often ad hoc and unreliable. Web services are commonly realized on computer systems where processing resources and network bandwidth are not a limitation.

Android is one of the largest open-source operating systems for smart devices, but lacks native support for the SOAP protocol. Google has shown, to date, little interest in adding a SOAP library to Android. This could be because they would rather support the current trends in Web Services toward REST-based services, and using JSON as a data encapsulation format or using XMPP for messaging. However, this is a conjecture subject to future research.

SOAP is the backbone protocol of Web services hence this thesis will focus in supporting SOAP on the android platform. We will explore and compare different transport protocols and compression techniques in order to achieve an efficient technique for SOAP messaging. The experiment will be done on mobile broadband (second, third and fourth generations) and Wi-Fi to examine the effects the different combinations has on CPU load and battery usage of the Android device, and the network load.

Key words: Android, SOAP, SOA, web services, XML, compression techniques and transport protocols,

1 INTRODUCTION

Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. Central to SOA is the principle that functionality should be broken down into locally stand-alone services, complete with explicitly defined and described interfaces. This principle means that the SOA concept is well suited for building federations of systems, as each system can be developed and operated independently, while at the same time enabling interoperability as long as the systems comply with the agreed upon interfaces. SOA as a concept can be realized using a number of different technologies, the most common being Web services [1].

Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. Web services are usually realized on computer systems where processing resources and network bandwidth 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)[2]. In such cases reducing the size and frequency of messages and using transport mechanisms that are tolerant of such conditions can help mitigate the effects of the limited conditions.

The Mobile phone industry is enjoying an escalating growth all over the world. Smart phones have become a part of our daily lives and more than 80% of the world population today owns a mobile phone. There have been significant advances in the mobile device space during the last decade. They now have better CPU and memory capabilities, embedded hardware such as camera and Wi-Fi, and in-built sensors such as GPS, accelerometer, gyroscope and magnetic field sensor, making them usable in versatile scenarios. We also have achieved higher data transmission rates for mobiles with the advances in 3G, 4G and Wi-Fi, paving way for mobile commerce and location based services. The advances in the mobiles and the adaptation of component based SOA everywhere have made the space for mobile web services.

While the advances in the mobiles are significant and they are also being used as service providers, they still have certain limitations. Battery life is one space where the advances are not sufficient. Mobile battery still lasts only for about one or two hours, if used for continuous computing. Wireless charging of mobiles is a good solution to deal with the problem [3]. Even though computing power and memory capacities of these 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 is used which

relies on the Extensible Markup Language (XML) Information Set 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 smart devices, mainly on how to compress the messages being sent, but also looking at different ways of sending the messages.

2 RELATED WORK

The integration of Web Services with mobile devices has many useful benefits. It supports automatic and autonomous self-configuring distributed systems without interfering with the main functionality of the mobile host which is making phone calls. An important motivation that leads us to this research is the fact that mobile devices have limited battery life and need for wireless environment which is problematic considering the heavy weight SOAP parsers to process the requests. Hence, the need of compressing the SOAP messages and exploring other protocols other than HTTP/TCP.
In this chapter we will present Android, Service Oriented Architecture, Web services and SOAP. We will describe different protocols for SOAP, a third party SOAP library for Android (KSOAP2) and compression techniques that focuses on compressing XML.

2.1 Android

There is a tremendous potential in developing smarter mobile devices that are more aware of its owner's location and preferences. Android being an open source platform can be described as a complete set of software for mobile devices; it delivers an operating system, middleware and key mobile applications. Android is built on a Linux kernel version, though it 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.

Android doesn't offer native support for consumption of Web services, but exists useful libraries like AndroidSOAP, WSClient++ and ksoap2-android which permits Android applications that in an easy and efficient way to consume Web services based on SOAP. This libraries are third-party distributed as free source, optimized for Android [7].

2.2 Service Oriented Architecture

Service Oriented Architecture (SOA) realized by Web services technology, provides seamless information exchange based on different policies and loose coupling of its components. In a military domain it enables making sensitive information resources available in the form of services, which can be discovered and used by all mission participants that do not need to be aware of these services in advance.
The most mature technology for implementing SOA, recommended by NATO and widely applied in the commercial sector, is Web services. Web services are described by a wide range of standards that deal with different aspects of their realization, transport, orchestration, semantics, etc. They provide the means to build a very flexible environment that is able to dynamically link different system components to each other. These standards are based on the eXtensible Markup Language (XML) and have been designed to operate in high bandwidth links [4].

The essential part of Web services is the interact relationship between a **Service provider** and **Service requestor**. This is the Web Service. **Service provider** is the component that implements the web service and informs its existence to other requester by publishing its interface and access information in the service registry. Sometimes, the service requestor wanting to use a Web service does not know the location of it. Hence, **Discover agencies** are responsible for the availability of both interface and implementation access information for the Web service to any service requester. **Service requester** searches the service within the discover agencies to find its service provider then connect to the latter using specific communication protocol [5].

2.3 SOAP

Simple Object Access Protocol (SOAP) is an XML-based messaging protocol that is platform free, transport free and the operating system free because of the usage of HTTP and Extensible Markup Language (XML) as its core technologies.

There are two kinds of SOAP requests; the first one is Remote Procedure Call (RPC) format request alike to new distributed architectures. The format is typically synchronous; client sends message and pauses to get any response and/or fault message back from server. The second format type of SOAP request is document type request. In this situation, full XML document is supplied to/from client and private server privately by SOAP message and vice versa [6].

It defines a set of rules for structuring messages that can be used for simple one-way messaging but is particularly useful for performing RPC-style (Remote Procedure Call) request-response dialogues. It is not tied to any particular transport protocol though HTTP is popular.

Theoretically, the clients and servers in these dialogues can be running on any platform and written in any language as long as they can formulate and understand SOAP messages. As such it is an important building block for developing distributed applications that exploit functionality published as services over an intranet or the internet.

Rather than define a new transport protocol, SOAP works on existing transports, such as HTTP, Simple Mail Transfer Protocol (SMTP), and Advanced Message Queuing Protocol (AMQP). SOAP message has a very simple structure. 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.
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].

2.4 SOAP WEB SERVICES

The term web service implies "something" accessible on the "web" that gives you a "service." Web services applications can be implemented with different technologies such as SOAP, discussed in this thesis, or REST. Web service is a technology that can be used for implementing clients and services based on a SOA, achieving interoperability between different systems.

Distributed software has been around for a long time, but, unlike existing distributed systems, SOAP web services are adapted to the Web. The default network protocol is HTTP, a well-known and robust stateless protocol. Although other protocols such as SMTP, TCP, UDP, AMQP can also be used, which forms the basis of our research. Web Services technology recognizes mobile computing as an area to which it should expand. Through integration, Web Services enable pervasive accessibility by allowing for user mobility as it overcomes the physical location constraints of conventional computing. However, mobile computing also requires a technology that connects mobile systems to a conventional distributed computing environment.

Web services are the proven way towards implementation of a "Service Oriented Architecture". Advancement in mobile device technology has motivated researchers to explore the possibilities of effectively hosting web services over mobile devices, and thereby trying to realize service oriented systems in mobile environments. There has been substantial work towards enabling mobile devices to host web services. An important aspect of service oriented systems, "service discovery" described above, however, remains a challenge in mobile environments. Several challenges specific to hosting web services over mobile devices need to be taken into account in such service discovery mechanisms. These include, but are not limited to battery and network constraints, limited computational power of mobile devices. Moreover, such dynamic mobile services are prone to uncertainty (owing to network outage, battery issues, physical damage) and frequent changes in functionality (primarily owing to the change of context), and hence make frequent service updates a necessity to effectively function as web-services.

Despite the fact that the condition of mobile computing has so much improved in recent years, applying current Web Service communication models to mobile computing may result in unacceptable performance overheads. This potential problem comes from two factors. First, the encoding and decoding of verbose XML-based SOAP messages consumes resources. Therefore Web Service participants, particularly mobile clients, will suffer from poor performance. Second, the performance and quality gap between wireless and wired communication will not close quickly. It is caused by the mobile environment's constraints like limited processor speed, limited battery lifetime, and slow unreliable and intermit connection [8].

2.5 Compression Techniques

Data compression is a process by which a file (Text, Audio, and Video) may be transformed to another (compressed) file, such that the original file may be fully recovered from the original file without any loss of actual information [9].

Data Compression is divided into two parts: Lossy and Lossless Data Compression. Data compression utilities are critical in helping achieve energy-efficient data communication, reducing communication latencies, and making effective use of available storage. The general goal of data compression is to reduce the number of bits needed to represent information. Whereas lossy compression approximates the original data, lossless compression enables the exact reconstruction of the original data by the decompressor. Lossy compression is not relevant in the SOAP and XML context. In this thesis therefore we focused on lossless compression, which is crucial for data such as program code, text input, images. The compression includes Deflate Compression (GZip, zlib) and XML-specific compression techniques.

Deflate Compression (GZip, zlib)

GZip is an optimized, lossless, and open source compression utility created to be a general replacement of existing compression techniques. It has been widely used to optimize traffic flow and optimization is achieved by requiring only a single pass through the file without the need for backwards seeking, and does so without knowledge about the input media type, or file size. The result of GZip is a file renamed with the .gz extension [12].GZip is a variation of LZ77 algorithm, which works by looking for duplicated strings in the data. The second and subsequent occurrences of a string is then replaced by a pointer to the first occurrence. Moreover, GZip applies Huffman coding in order to to assign shorter codes to more frequent characters or strings. Because of this, GZip provides a smaller file size as a result [10].

Because of this, we used GZip in this thesis.

XML-specific compression solutions

Augeri 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 [11].

In Teixeira's paper two algorithms for XML documents compression were discussed: Schema-aware algorithm and Hybrid algorithm. These were compared to WAP Binary Extensible Markup Language (WBXML), XMill and Efficient XML Interchange (EXI), considering the metrics compression rate and compression time. Although no method

was good enough in all requirements, among the conclusions were that EXI reached the best compression rate [13].

EXI schema informed mode compression delivers superior results compared to other FI compression technique; In essence "EXI is better performer than FI" [14].
EXI format removes redundant tags and values from XML documents and encodes numeric content in a binary format. This format delivers significant file size savings and processing efficiencies. For XML-based data, a doubling of bandwidth potential is achievable and CPU burdens minimized when EXI is applied. Additional findings indicate that traditional binary data formats converted to an XML format can be smaller than their native binary format after the application of EXI [12]

Giving credence to the comparisons above, in this thesis we tested EXI as the XML-specific compressor.

2.6 Transport protocols for SOAP

SOAP enables exchange of SOAP messages using a variety of underlying protocols. One of the characteristics of SOAP is neutrality; SOAP enables exchange of SOAP messages using any transport protocol, such as HTTP, SMTP, TCP, or User Datagram Protocol (UDP) [2].
The formal set of rules for carrying a SOAP message within or on top of another protocol (underlying protocol) for the purpose of exchange is called a binding. The SOAP Protocol Binding Framework provides general rules for the specification of protocol bindings; the framework also describes the relationship between bindings and SOAP nodes that implement those bindings.

SOAP-over-HTTP

Over the Internet, HTTP is the protocol that is most widely used for SOAP binding. Because HTTP is one of the core protocols of the Internet and is widely supported by Web servers, SOAP-over-HTTP is the only concrete binding specification defined in the SOAP binding framework proposal. Because it is often allowed to pass through firewalls, it is a convenient candidate for transporting SOAP. The recommended version of HTTP for SOAP binding is HTTP/1.1 and all SOAP implementations provide this binding.

A SOAP message can be transported using HTTP by encapsulating the SOAP request into the message body of a HTTP GET or HTTP POST. Similarly, a SOAP response can be encapsulated into the body of a HTTP response. HTTP binding provides reliable message transport, flow and congestion control.
However, SOAP over- HTTP has some drawbacks. This includes:

- Does not support peer-to-peer messaging exchange between SOAP nodes.
- The response time is generally higher when using HTTP as the transport protocol for SOAP because of the three-way handshake process occurring at the TCP layer between a client and a server. This is to ensure that all the parties acknowledge the connection and are ready to transmit data.

SOAP-over-SMTP

SOAP-over-email binding is only presented in the specification as an example to demonstrate the realization of the SOAP binding framework. In the SOAP-over-email binding, SOAP messages are piggybacked on SMTP packets.

SOAP-over- TCP

In order to transport a SOAP message using TCP as a direct underlying protocol, a SOAP message is stored in the data octets part of a TCP packet. There is not yet any official specification for SOAP binding with TCP, however, Apache Axis (10) and Microsoft Web Service Enhancement (WSE) 2.0 (108) include APIs that enable the sending of SOAP messages via TCP.

SOAP-over- UDP

SOAP-over-UDP is an OASIS standard covering the publication of SOAP messages over UDP transport protocol, providing for One-Way and Request-Response message patterns. Unlike TCP, it does not provide any flow-control mechanism and only guarantees best-effort delivery of packets. Packets delivered by UDP may be duplicated, arrive out of sequence or not even reach their destination at all. However, because of its simplicity, UDP provides a number of benefits over TCP which includes:

- UDP does not require a connection to be established before sending a packet. Each UDP datagram carries its own destination address and is routed independently of other packets. This reduces the setup time associated with sending a message.
- UDP packets are smaller than TCP packets. The UDP header is only eight bytes in length, in comparison to the TCP header which is at least 20 bytes in length.
- UDP supports multicasting opening up the opportunity to create push-based and publish/subscribe Web services, where SOAP messages or notifications are sent to multiple clients periodically or triggered by an event.

Given this advantages, we examined UDP as an alternative protocol.

SOAP-over- AMQP

AMQP is a binary, application layer protocol, designed to efficiently support a wide variety of messaging applications and communication patterns. It can utilize different transport protocols but it assumes an underlying reliable transport protocol such as TCP. AMQP provides asynchronous publish/subscribe communication with messaging. Its main advantage is its store-and-forward feature that ensures reliability even after network disruptions. It ensures reliability with the message-delivery guarantees of at most once, at least once and exactly once. Security is handled with the use of the TLS/SSL protocols over TCP. AMQP has low success rate at low bandwidths, but it increases as bandwidth increases. Comparing AMQP with the aforementioned REST, AMQP can send a larger amount of messages per second. An AMQP

environment with 2,000 users spread across five continents can process 300 million messages per day. For example, JPMorgan which is an American banking and financial services company uses AMQP to send 1 billion messages per day [15].There are various implementations of AMQP. This includes Apache Qpid, Apache ActiveMQ and RabbitMQ. In this thesis however, we used the RabbitMQ implementation. Other implementations can give different results subject to future research.

SOAP-over-SCTP

SCTP (Stream Control Transmission Protocol) is a standard protocol (RFC 2960) developed by the Internet Engineering Task Force (IETF) for transmitting multiple streams of data at the same time between two end points that have established a connection in a network. Sometimes referred to as "next generation TCP" (Transmission Control Protocol) - or TCPng. SCTP was originally designed as a protocol for telephony signaling over IP networks. It 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. A telephone connection requires that signaling information (which controls the connection) be sent along with voice and other data at the same time. SCTP also is intended to make it easier to manage connections over a wireless network and to manage the transmission of multimedia data [16].

SCTP has been implemented for all major operating systems and its most important enhancements are multi-homing and multi-streaming.

Multi-homing enables the respective endpoints to communicate over multiple IP addresses and network interfaces, hence systems with multiple interfaces can use one over the other without having to wait. **Multi-streaming** is a technique employed to avoid head-of-line blocking by splitting control and data into separate streams. Each message sent to a data stream can have a different final destination, but each must maintain message boundaries. With multi-streaming only the affected stream would be blocked; the other streams are allowed to continue to flow.

Johnsen et al.'s [4] study 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 though was left out because was not official [4].

Based on these results this thesis tested the same protocols used in on Android in both advantaged and disadvantaged networks.

2.7 SOAP Libraries for Android

As mentioned earlier the android platform does not have native support for SOAP. Hence a third-party library needs to be added. There exist several non-official SOAP libraries aimed for working on Android including AndroidSOAP, WSClient++ and ksoap2-android.

All these libraries are often created and maintained on a voluntary basis, some tend to be outdated while others require payment to use. Therefore, in our thesis we used the ksoap2-android project since it's widely used, recently updated and actively maintained library. The ksoap2-android project provides a lightweight and efficient SOAP client library for the Android platform. This is good for constrained devices such as mobile devices. 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. However, can also work well in other platforms [2]. It is an open source SOAP API with small footprint implementation of XML, aimed at developing applications for the Android platform [17].

3 METHODOLOGY

In this research, we used the Experimental Approach**.** We carried out an experiment to evaluate how using GZIP and EXI compressions, different protocols like UDP, AMQP other than HTTP/TCP will affect the battery life and bandwidth usage of the android device. We performed the experiment on both advantaged and disadvantaged networks to determine the efficiency of the technique to be used to send a SOAP Message.

3.1 Experiment

After establishing the research procedure, we proceeded to carry out the actual experiment. This involved application of both software and hardware tools. The figures below show the setup of our experiment.

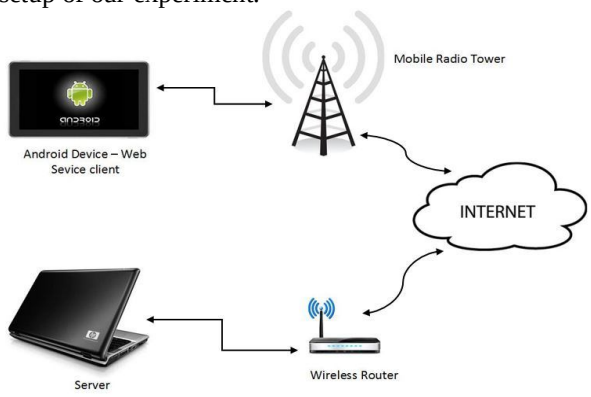

Figure 3.2 Experiment Diagram Using Wi-Fi

We had an android device (TAB A) and a server (a laptop Lenovo L460 series with 8GB RAM and 500GB HDD.) which hosted our services and other additional softwares like wireshark and editors like Android Studio. (Figure 3.1 and 3.2 in section 3.2 above shows the experiment diagrams using a modem and using Wi-fi respectively).

We extended KSOAP2 to accommodate the additional protocols. The android device received the service from the server then send it back to the server. WireShack was used to

test bandwidth usage. (More of this is described in section 3.3 below).

Hardware & software tools

1. Client(Android Device)
   - Android 5.0.2
   - 1.0Ghz Dual core processor or higher
   - 1GB RAM
   - 16GB Internal Memory
   - 6000 mAh Battery capacity
2. Server
   - 2.0Ghz Quad core or higher
   - 4GB RAM
   - 128GB Storage
   - Wi-Fi 802.11 b/g/n
   - Wireshark - installed on the server.
   - RabbitMQ – installed on the server.
3. Modem

3.2 Data Collection

In this section we described the tools used for data collection that is profiling tools for android and the network analyser (Wireshark) which will help us capture the network traffic. We also describe the various tests (Web services) we will use in our experiment.

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). The same is integrated in Android Studio and it provides port-forwarding services, screen capture on the device, thread and heap information on the device, logcat, process, and radio state information, incoming call and SMS spoofing, location data spoofing, and more.

DDMS in android studio

Method profiling is a means to track certain metrics about a method, such as number of calls, execution time, and time spent executing the method. 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 that you create by using the debug class to log tracing information in your code and it helps to debug the application and profile its performance. Traceview visualizes the application in two panels, the timeline panel and the profile panel.

Another way of measuring CPU load is to measure how much time a method in the program spends before it finishes. This time can be logged to a file for that test and then compared with running the same method using other parameters. In this thesis the parameters would be the different compression methods (No compression, GZip and EXI) and different transport methods (HTTP, AMQP and UDP).

Network traffic tool

Wireshark is the world's foremost and widely-used network protocol analyzer. It lets you see what's happening on your network at a microscopic level. Figure 3.10 and Figure 3.11 shows Wireshark's graphical front end and graphical illustration respectively.

3.3 Web Service

We will use the Hello Web service for testing purposes. We tested different transport protocols and different compression techniques against the "Hello Web service" over both mobile (In second, third and fourth Generations) and Wi-fi (with an average download speed of 0.22 Mbps and upload speed of 8.44Mbps)

- Test 1: "Hello Web service" over mobile network (In 2G,3G and 4G)
- Test 2: "Hello Web service" over Wi-Fi

More tests can be done with several Web services (for example TempConvert Web service, File uploads or simply Exchange Picture Web Service etc) so as to vary between large and small SOAP messages, as well as having both XML and non-XML payloads. However, since timelines for this research is limited, we just used the "Hello Web service". The testing was done under normal conditions as artificial packet loss or bad network connection were not added. In addition to the Web service testing, the size of the compressed files were measured in order to compare the compression of GZip and EXI.

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 4.0, which replies with a String "Hello Name!".

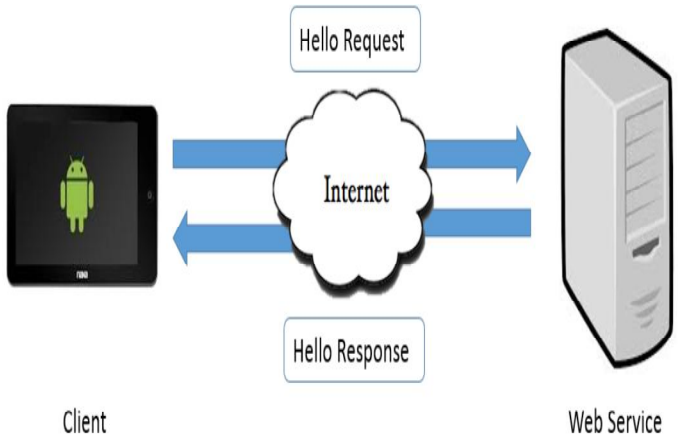

Figure 3.7 Hello Web Service

The test begins with the client passing a string "Name", marshalling it into a SOAP envelope (possibly compressing it) and sending it to the server. The server sends the same string data in the reply to the client. Upon receiving the reply, the client unmarshals the data (decompressing if needed) into

a new string, which it saves on the device memory card. This procedure is then repeated for the duration of the test.

3.4 Test Parameters

The testing as described in section 3.3.3 will be done in both over wi-fi and over mobile network (2G, 3G and 4G) and we will measure the differences in the following variables:

1. Battery Level
2. Network traffic in the form of the total amount of data sent over the network and goodput (the number of useful information bits delivered by the network to a certain destination per unit of time). In goodput the amount of data considered excludes protocol overhead bits as well as retransmitted data packets.
3. CPU load caused by different compression techniques.

Battery level

The percentage battery drop is recorded before and after each test run. The battery level is measured calling the battery level programmatically from the Android system. Without knowing the battery status of a device, a web developer must design the web application with an assumption of sufficient battery level for the task at hand. This means the battery of a device may exhaust faster than desired because web developers are unable to make decisions based on the battery status. Given knowledge of the battery status, web developers are able to craft web content and applications which are power-efficient, thereby leading to improved user experience.

Network Load.

As described in section 3.3.2 we will use Wireshark to monitor traffic generated by the tests, measuring the total Megabytes transceived as described in section 3.3.3.

CPU load

The time spent on marshalling and unmarshalling was measured (in milliseconds) to show the effect each compression tool and on different networks both advantaged and disadvantaged has on the CPU load. Compared to DDMS Method profiling this method is simple and gives sufficient results as far as this this is concerned.

4 RESULTS AND DISCUSSIONS

The Hello Web service is called 180,000 times for each protocol in each test. This done on both wi-fi and mobile broadband. Below are the results from each of the studied protocols that is HTTP, UDP and RabbitMQ (implementation of AMQP). The results for 4G network is not discussed as it's slightly similar to that of 3G.

Measuring the battery drop for the different configurations was difficult, and many calls had to be done to see an effect on the battery. It's also impossible to measure the battery level with decimals. The Android API only offers an Integer value of the battery, making the ordeal more imprecise than we would have desired. There is no apparent way to determine if a drop of 3% in battery level is in reality 3.0% or 3.7%. The results for the battery usage from the calls done over Wi-Fi are not presented here since they have the same size. There was so significant change in the battery level. More tests can be done here with an increase in the number of calls.

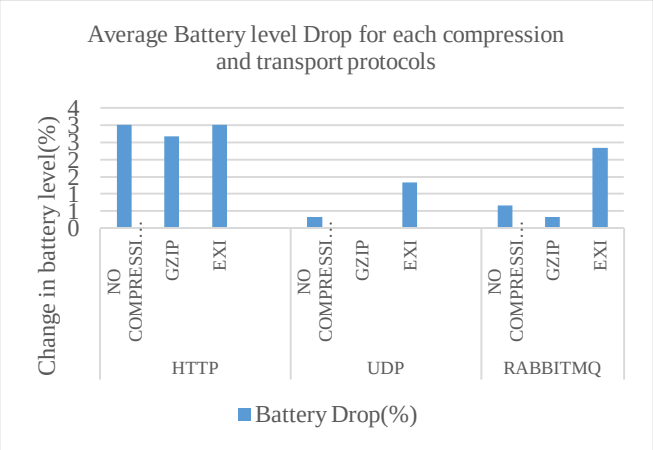

Figure 4-1 Average Battery level Drop for each compression and transport protocols.

Figure 4-1 shows that EXIficient consumes more battery power than where there is no compression and in GZip compression. Both UDP and RabbitMQ consume significantly less battery power as compared to when HTTP is used. Though close to No compression, GZip consumes less battery.

We measured the time spent on marshalling and unmarshalling to give an impression of the CPU load of the different compression methods. The figures below show the results for both mobile broadband and wireless networks.

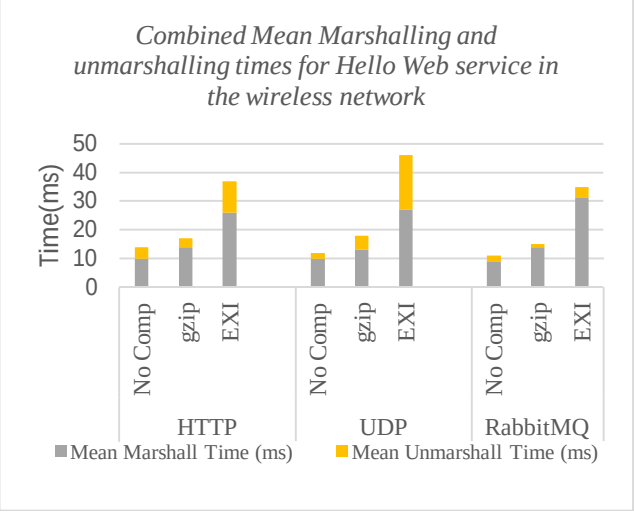

Figure 4-2 Combined Mean Marshalling and unmarshalling times for Hello Web service in the wireless network.

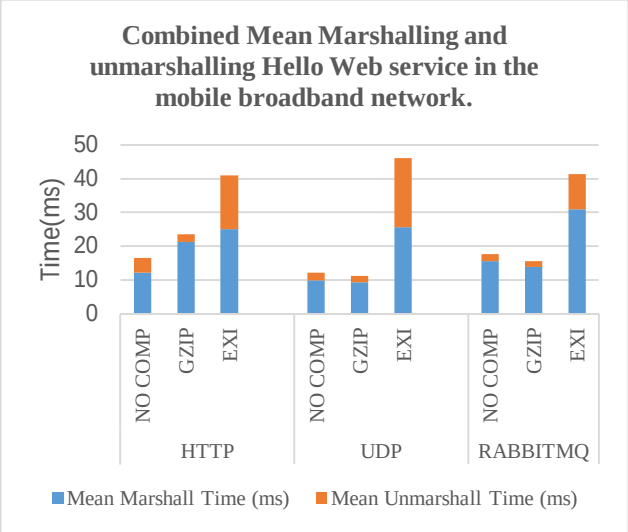

Figure 4-3 Combined Mean Marshalling and unmarshalling times for Hello Web service in the mobile broadband network.

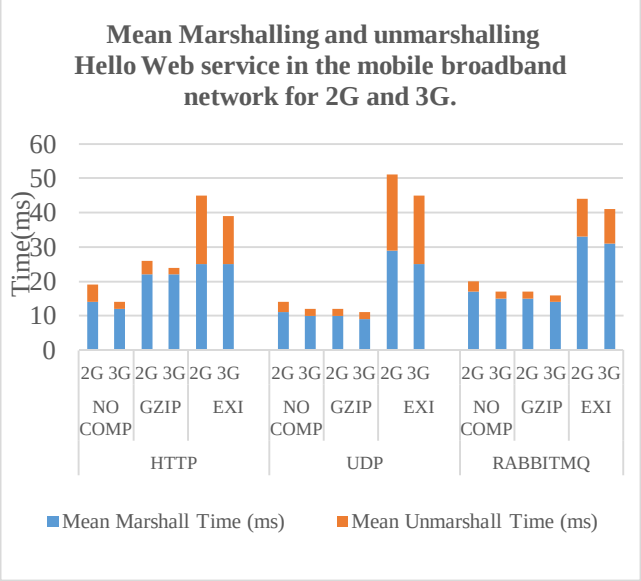

Figure 4-4 Mean Marshalling and unmarshalling times for Hello Web service in the mobile broadband network for 2G and 3G.

Figures 4-2, 4-3 and 4-4 shows that the marshalling and unmarshalling time is very high when EXIficient compression is used. It can be seen that EXIficient causes much more CPU load since the more the time it takes for marshalling and unmarshalling the higher the CPU load.
Figure 4-4 shows that there is less CPU load when the network speed improves. In 3G the CPU load is lower than in 2G. Results for 4G are not presented since it's approximately the same as that of 4G.

The total amount of data sent over the network is presented combining the requests and responses of all Web service calls. This is done for both wi-fi and mobile network. The results for 4G are not presented as they are slightly similar to that of 4G.

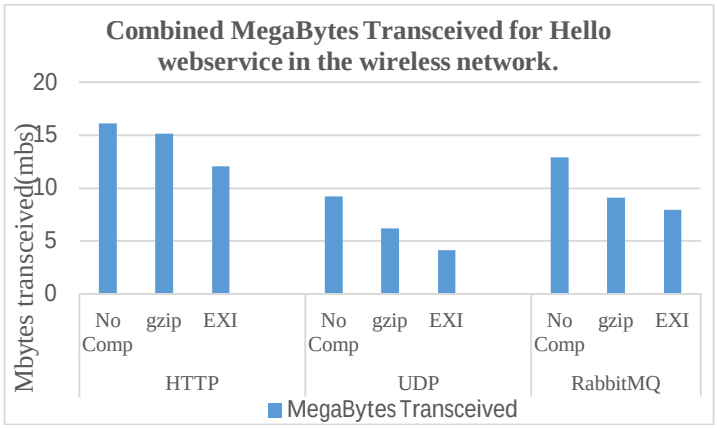

Figure 4-4 Combined MegaBytes Transceived for Hello webservice in the wireless network.

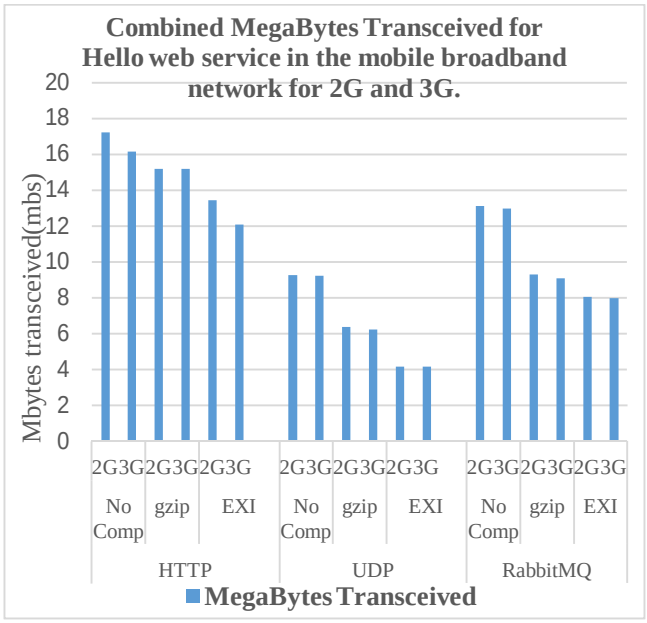

Figure 4-5 Combined MegaBytes Transceived for Hello webservice in the mobile broadband network for 2G and 3G.

Figures 4-4 and 4-5 show that EXIficient has the lowest data transceived. This means that it compresses better compared to GZip though the difference is less using RabbitMQ. There is no much difference in compression when using 2G or 3G.

Comparing GZip and EXI Compression techniques

This section elaborates the size of the original file compared to the GZip and EXIficient compression files.

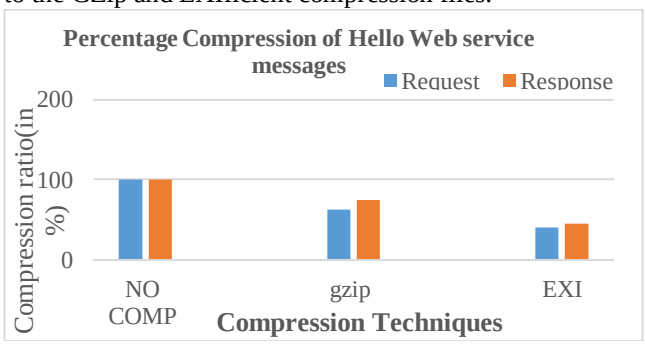

Figure 4-6 Percentage Compression of Hello web service messages

Compression reduces the size of data being transferred hence reduces the time for messaging. EXIficient compresses much better than GZip with the Hello web service. Figure 4-6 shows that the size of the compressed data with EXI is the lower compared to GZip. The compression ratio is also much lower compared to GZip.

In this thesis, goodput is how fast the exchange of SOAP messages are in megabits per second.

5 OPTIMIZING SOAP MESSAGING IN MOBILE DEVICES – PROPOSED APPROACH

5.1 Compression techniques

Compression improves bandwidth utilization and response time of SOAP messages. The two compression techniques used for this thesis included: GZip compression and EXI compression

From Figure 4-1 GZip consumes less battery as compared to EXIficient and No Compression. Figures 4-2, 4-3 and 4-4 show that marshalling and unmarshalling time is lower when GZip compression is used hence less CPU load.

From our results and discussion, EXIficient performed poorly with respect to the CPU load. From Figures 4-2, 4-3 and 4-4 marshalling and unmarshalling time is very high when EXIficient compression is used hence causes much more CPU load since the more the time it takes for marshalling and unmarshalling the higher the CPU load. Although figures 4-4 and figure 4-5 show that EXIficient has the lowest data transceived. Meaning that it compresses much better than GZip when using the Hello web service though the difference is less when using RabbitMQ.

Because of more CPU load, EXIficient compression consumes more battery hence we advise not to be used in devices with less processing capabilities.

5.2 Transport protocols

SOAP over HTTP - Since HTTP is one of the core protocols of the Internet and is widely supported by Web servers, SOAP-over-HTTP is the only concrete binding specification defined in the SOAP binding framework proposal. From our experiments HTTP performed poorly as compared to UDP and AMQP (the RabbitMQ implementation) as it consumes more battery power, has more CPU load and network load and in terms of goodput, it was the lowest.

The response time is generally higher when using HTTP as the transport protocol for SOAP because of the three-way handshake process occurring at the TCP layer between a client and a server.

SOAP over UDP - UDP consumes less battery power as compared to when HTTP and AMQP (RabbitMQ implementation) is used. From figures 4-2, 4-3 and 4-4 the mean marshalling and unmarshalling time is lowest when the UDP protocol is used with No compression and with GZip compression though highest when EXIficient compression is used. More tests can be done with heavier messages.

UDP had the highest goodput compared to HTTP and RabbitMQ although the difference is less comparing with that of RabbitMQ.

SOAP over AMQP - RabbitMQ consume less battery power as compared to when HTTP is used. From figures 4-2, 4-3 and 4-4 the mean marshalling and unmarshalling time is for RabbitMQ is lower than that of HTTP with No compression and with GZip compression though high when EXIficient compression is used. RabbitMQ had the higher goodput compared to the most widely used protocol HTTP.

5.3 Combination of Compression technique and Transport protocol

Both UDP and RabbitMQ consume significantly less battery power as compared to when HTTP is used.

In terms of the network load, figures 4-4 and 4-5 show that EXIficient compresses better compared to GZip though the difference is not bigger using RabbitMQ because it has the lowest data transceived. Figures 4-7, 4-8 and 4-9 shows that when measuring goodput, UDP and RabbitMQ are better compared to HTTP.

In summary, when we compare different combinations of the transport protocols (HTTP, UDP and RabbitMQ) with compression techniques (GZip and EXI) both in wireless and mobile broadband (advantaged (3G, 4G) and disadvantaged (2G) networks), using GZip together with AMQP (RabbitMQ implementation to be specific) is better than all the other combinations for a reliable connection. GZip consumes less battery and has less CPU load compared to EXIficient.

However, if no reliable connection is required, then using UDP protocol together with GZip compression is the best.

6 CONCLUSION

Our main goal was to find out how to efficiently send and receive SOAP messages in the android platform. In this thesis we explored and compared different ways to transport and compress SOAP messages in both wireless and advantaged (3G, 4G) and disadvantaged (2G) networks in order to give recommendations on how to achieve this. We extended ksoap2-android library to allow android support the different transport and compression methods. Our tests included exchanging SOAP messages with payloads consisting of text strings.

From our tests EXIficient performed poorly with respect to the CPU load and consumes more battery since the marshalling and unmarshalling times of EXIficient were much higher than when GZip or no compression are used. Universal Datagram Protocol (UDP and AMQP preserve more battery life than HTTP does.

A combination of **GZip** with **AMQP** (RabbitMQ implementation to be specific) performs better than all the other combinations for a **reliable connection**. However, if **no reliable connection** is required, then a combination of **UDP** together with **GZip** is the best choice.

7 FUTURE WORK

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.

It will be interesting to test SOAP-over-SCTP because of its important enhancements of multi-homing and multi-streaming. Android has not yet made it available in the official API. Security issues will need to be addressed in the future since we used third party library, KSOAP2 with additional code.There exists other SOAP Optimization techniques that were not covered in this thesis which can be considered in future research. This includes: client caching algorithms and SOAP Parsing.

Further testing with the proposed solution presented in this thesis is also possible, with for example adding more web services with higher payloads (e.g. image uploads) and also making more calls to the web services. The addition of more web services with much bigger SOAP messages might have different results.

8 REFERENCES

1. Bloebaum, T. H., Johnsen, F. T., Brannsten, M. R., Alcaraz-Calero, J., Wang, Q., & Nightingale, J. (2016, May). **Recommendations for realizing SOAP publish/subscribe in tactical networks.** In Military Communications and Information Systems (ICMCIS), 2016 International Conference on (pp. 1-8). IEEE.
2. Eggum, D. O. (2014). **Efficient SOAP messaging for Android.**
3. Srirama, S. N. (2017). **Mobile web and cloud services enabling Internet of Things.** CSI Transactions on ICT, 1-9. https://doi.org/10.1007/s40012-016-0139-3
4. Johnsen, F. T., Bloebaum, T. H., & Eggum, D. O. (2015, May). **Efficient SOAP messaging for Android.** In Military Communications and Information Systems (ICMCIS), 2015 International Conference on (pp. 1-9). IEEE.
5. AbdAllah, M. M., & Mahjoub, W. H. (2013). **A Quick Introduction to SOA.Software Engineering Competence Center.**
6. Mohsin, A., Asghar, S., & Naeem, T. (2016, December). Intelligent security cycle: **A rule based run time malicious code detection technique for SOAP messages.** In Multi-Topic Conference (INMIC), 2016 19th International (pp. 1-10). IEEE.
7. Shabani, I., Sejdiu, B., & Jasharaj, F. (2015). **Consuming Web Services on Android Mobile Platform for Finding Parking Lots.** University of Prishtina, Republic of Kosovo, IJACSA, 6(2).
8. Hamad, H., Saad, M., & Abed, R. (2010). **Performance Evaluation of RESTful Web Services for Mobile Devices.** Int. Arab J. e-Technol., 1(3), 72-78.
9. Sidhu, A. S., & Garg, M. (2014). **Research Paper on Text Data Compression Algorithm using Hybrid Approach.** International Journal of Computer Science and Mobile Computing, 3(12), 01-10.
10. Boonkrong, S., & Dinh, P. C. (2015, October). **Reducing battery consumption of data polling and pushing techniques on Android using GZip.** In Information Technology and Electrical Engineering (ICITEE), 2015 7th International Conference on (pp. 565-570). IEEE.
11. Augeri, C.J., et al. **An Analysis of XML Compression Efficiency in 2007 Workshop on Experimental Computer Science (ExpCS)**. 2007. New York, NY, USA.
12. Snyder, S. L. (2010). **Efficient XML Interchange (EXI) compression and performance benefits: development, implementation and evaluation.** NAVAL POSTGRADUATE SCHOOL MONTEREY CA.
13. Teixeira, M. A., Miani, R. S., Breda, G. D., Zarpelão, B. B., & de Souza Mendes, L. (2012). **New Approaches for XML Data Compression.** In WEBIST (pp. 233-237).
14. Jaiswal, G., & Mishra, M. (2013, February). **Why use Efficient XML Interchange instead of Fast Infoset. In Advance Computing Conference (IACC)**, 2013 IEEE 3rd International (pp. 925-930). IEEE.
15. Karagiannis, V., Chatzimisios, P., Vazquez-Gallego, F., & Alonso-Zarate, J. (2015). **A survey on application layer protocols for the internet of things.** Transaction on IoT and Cloud Computing, 3(1), 11-17.
16. Belkhode, V. V., & Dakhane, D. M. (2014). **UDP-Based Multi-Stream Communication Protocol Using NS2.** International Journal on Recent and Innovation Trends in Computing and Communication, 2(3).
17. Shen, Z., Man, K. L., Liang, H. N., Zhang, N., Fleming, C., Afolabi, D. O., & Poon, S. H. (2013). **A light mobile web service framework based on axis2.** In Future Information Communication Technology and Applications (pp. 977-985). Springer Netherlands. https://doi.org/10.1007/978-94-007-6516-0_107