Efficient SOAP messaging for Android

Frank T. Johnsen and Trude H. Bloebaum
Norwegian Defence Research Establishment (FFI), Kjeller, Norway
University Graduate Center at Kjeller (UNIK), Kjeller, Norway

Dag Ove Eggum
Bergen University College
Bergen, Norway

Abstract—**Service-Oriented Architecture (SOA) is a software architectural design pattern for constructing and deploying application functionality based on loosely coupled components. Web services is a much-used technology for implementing applications following SOA principles, and achieving interoperability between different systems. 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). 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. In this paper we focus on supporting SOAP on the Android platform in an efficient manner. By efficient, we mean maximizing battery life while minimizing network use. We explore and compare different transport protocols and compression techniques in order to achieve these goals. This work was performed in context of the NATO IST-118 "SOA recommendations for disadvantaged grids in the tactical domain" group.**

I. Introduction

During the last few years mobile devices have been getting both smarter and cheaper, leading to a formidable increase in the number of smart devices in common use. This development is also of interest for defense applications, as current civil technology provides a powerful and highly portable communication platform. Civil technology can be leveraged in use cases where properties like high volume and acquiring low-cost devices are more important than having rugged, high-assurance special-purpose military hardware. There are several recent proposals for leveraging civil smart devices for military purposes (see e.g., [1], [15]). However, it is important to keep the direction NATO is taking with regard to interoperability in mind, namely leveraging the Service-Oriented Architecture (SOA) principles. This approach dates back to the NATO Network Enabled Capability (NNEC) Feasibility Study (FS) of 2005 [4], which identified Web services[1] as the key enabler for NNEC. The next step was the standardization work of the Core Enterprise Services Working Group (CESWG), which produced an overview of the identified Core Enterprise Services (CES). The final work of the CESWG was the SOA baseline [5], in which they pointed to specific standards to use when realizing a selected subset of the CES (e.g., using W3C's Web services as the message oriented middleware). Following the CESWG, which finished its work by the end of 2011, the work has continued along two paths: The NATO C3 Taxonomy continues to refine and detail the overall classification of services (including the CES), whereas the standards stack that was identified by CESWG is being further detailed and expanded upon in NATO's ongoing work in defining Service Interface Profiles (SIPs). SIPs can be considered foundational for the Connected Forces Initiative (CFI) and Federated Mission Networking (FMN), and some SIPs are already part of the NATO FMN Implementation Plan (NFIP). Current SIPs address solutions mainly for fixed infrastructure networks, and do not address specific issues that can arise on the tactical level.[2] This, however, is the focus of the ongoing NATO group IST-118 "SOA recommendations for disadvantaged grids in the tactical domain". The work presented in this paper was performed in context of IST-118, the follow-on group of IST-090 which focused on "SOA challenges for real-time and disadvantaged grids" (see [2], [10] for more on the IST-090/118 groups).

Web services, the middleware technology most com-

[1]The term "Web services" is often used on a general basis, to mean any kind of service built using approaches encountered on the Web. However, in the context of the NNEC FS, "Web services" follows the World Wide Web Consortium's definition [21], i.e., Web services realized using XML as the format, SOAP as the message protocol, and WSDL as the service interface.

[2]The NATO C3 Taxonomy, existing SIPs, and the current iteration of NFIP documents are available at http://tide.act.nato.int/tidepedia (access requires a TIDEpedia account).

monly used for implementing SOA based systems, is normally deployed in networks where the communication channels have high capacity. This has enabled a design that supports machine-to-machine interactions by prioritizing clarity, by including context information in messages, over compact information representation. In addition, most Web service implementations rely on Web technologies, such as HTTP over TCP, which work well in infrastructure networks. Relying on these Web technologies is however a challenge in mobile networks [17]. If applied for defense use, the smart devices may be connected to the tactical edge. Typically, one must rely on the wireless communication the devices can provide then. Also, we must keep the current set of NATO standards in mind, in order to ease interoperability with systems in the infrastructure.

The mobile platform with the currently largest user base is Android. This platform supports a number of different Web technologies, but has no native support for the Web service message format, SOAP. Support for this protocol is available in the form of third party libraries, but these libraries often rely on the same underlying Web technologies as those used in infrastructure networks. Most of the devices running the Android operating system rely exclusively on wireless transmission of information, and are thus more likely to encounter adverse network conditions such as longer response times, limited bandwidth or communication disruptions. Adapting the transport of Web service messages to consume less network resources may help reduce the impact these network limitations have on the user experience when accessing Web services from mobile devices, while still enabling access to the information and functionality these services represent.

In addition to the potential performance benefits, adapting SOAP transport to minimize the use of network resources may also have a cost benefit. Many users either pay for the exact amount of data transferred, or have a limited data plan included in their subscriptions. There are several different approaches to achieving this reduction in network resource consumption, including data compression and the use of more efficient transport mechanisms. Some of these methods will require increased processing on the mobile device, which will impact power consumption. Battery power is a limited resource on mobile devices, which might counter some of the benefits gained from reducing network usage.

In this paper we focus on supporting SOAP on the Android platform in an efficient manner. By efficient, we mean maximizing battery life while minimizing network use. We explore and compare different transport protocols and compression techniques in order to achieve these goals.

The remainder of this paper is organized as follows: First, we discuss the overall requirements we adhered to in our work in Section II. Section III gives an overview of related work. Our software design and implementation are discussed in Section IV. The solution evaluation and test results are covered in Section V. Finally, Section VI concludes the paper.

II. Requirements

Our overall goal is to support SOAP on Android in an efficient manner. Since several unofficial SOAP libraries for Android already exist, we think that continuing development on one of these rather than developing a new library from scratch is preferable.

Efficiency can encompass many things, but we have chosen to focus on two major issues related to using mobile devices: Battery lifetime and network load. The different ways we investigate efficient SOAP transmission on Android in this paper are quantified in relation to these aspects.

For business applications it is important that the security and integrity of the devices used are maintained. Thus, we do not consider any techniques in this paper that would require the device to be *rooted*. Rooting is the process of allowing users of smartphones, tablets, and other devices running the Android mobile operating system to attain unauthorized privileged control (i.e., "root access") within Android's sub-system.

The strength of Web services lie in the strong focus on standardization and profiling for that middleware technology. Following the standards ensures that clients and services from different vendors can be used seamlessly and interchangeably. Thus, we require that all optimizations can be made to work with a COTS[3] application server. The overall goal is not reached unless a SOAP client on Android can access a Web service on a COTS server regardless of the optimizations applied.

The requirements identified are summarized in Table I. Next we discuss related work.

[3]Recent trends within NATO seek to leverage Commercial Off-The-Shelf (COTS) products for military use.

TABLE I. REQUIREMENTS

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

III. RELATED WORK

Werner et al. [22] have identified bandwidth use as one of the major issues when using Web services. The authors investigated transmitting SOAP over several different transport protocols. The study identified SOAP over UDP as the approach with the least overhead.

Along the same lines, Phan et al. [14] have also considered different transport protocols for SOAP. That study provides an evaluation of SOAP over HTTP, TCP, and UDP. Their findings support those of Werner et al., showing that SOAP over UDP has the lowest overhead but, due to the unreliable nature of UDP, the solution may suffer from high packet loss rates.

Tari et al. [17] evaluated the performance of different underlying transport protocols for SOAP in a wireless environment. The study showed that SOAP-over-HTTP and SOAP-over-TCP were seemingly not well suited for wireless applications, and led to high latencies and high transmission overhead. Conversely, 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 percent.

Augeri et al. [3] tested a multitude of ways to compress XML, focusing on the compressed file sizes and execution times. Among their conclusions were that in most instances a general-purpose compressor (e.g., gzip) should be used, although if maximum parsing and compression speed was needed an XML-specific compressor might be useful. Because of these observations we wanted to include both a general-purpose solution (i.e., gzip) as well as an XML-specific compressor for our evaluation on Android.

Efficient XML Interchange (EXI) [20] is a standard for XML representation that was designed to work well for a broad range of applications. In "New Approaches for XML Data Compression" [18] several different algorithms for XML document compression were compared considering the metrics compression rate and compression time. Among the conclusions from that study were that EXI achieved the best compression rate.

Fast Infoset [7] (FI) 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 encoding. The paper "Why Use Efficient XML Interchange Instead of Fast Infoset" [8] presents EXI and FI as the best XML compressors. The paper evaluated the performance of both techniques based on parameters such as memory utilization, CPU usage time, compression ratio, and the compressed file sizes. The authors concluded that EXI schema informed mode compression delivers superior results compared to other techniques. Because of these results, we focus on EXI as the XML-conscious compression method in this paper.

In our previous work [12] we have shown that it is possible to utilize a mobile phone as a Web service client using SOAP, and that we can reduce the communication overhead by using compression. We used actual Universal Mobile Telecommunications System (UMTS) and General Packet Radio Service (GPRS) network connections. We tried different compression algorithms, and found that compression should be employed in order to lower the monetary costs involved when using mobile phones. Following this we performed a study [9] of four different protocols (TCP, UDP, SCTP, and AMQP) for SOAP. We evaluated the protocols in an emulator under typical networking conditions, and found that TCP and SCTP exhibited similar results, with AMQP being slightly worse due to a high amount of signaling traffic. Like the other mentioned studies we found that SOAP over UDP was most efficient provided there was no need for reliable communication.

This paper builds on these ideas, and takes the concept further, in that we investigate efficient SOAP support on the Android platform. As opposed to the previous work, we investigate both compression and different transport protocols in conjunction on an actual device here to study their mutual effects on battery and network usage.

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 for Android: AndroidSOAP [6], WSClient++ [13], and ksoap2-android [19] to name a few prominent ones. We chose to leverage the library from the ksoap2-android project since it was being actively maintained at the time, seemed to have a large user community, and finally because an evaluation by Shen et al. [16] has shown that ksoap2-android has a very efficient and small footprint implementation of XML. We think this makes the library a good choice as a basis for further optimizations as well.

IV. Design and implementation

A. Software involved

The two compression methods chosen to compare with uncompressed SOAP transmission are gzip and EXI. The reason for choosing gzip is that it is widely used for file compression and decompression, and it is one of the most efficient general compression tools. 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 as we discussed in Section III. EXI is not part of either Java or Android, and therefore has to be added as a third-party library. We chose EXIficient, the open source implementation of EXI, for this purpose. The EXIficient version we used was 0.9.2.

The three SOAP transportation methods chosen for comparison were HTTP/TCP, SOAP-over-UDP and SOAP-over-AMQP. 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 have 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. 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 paper, we leverage the third-party library RabbitMQ for implementing SOAP-over-AMQP. The RabbitMQ library used was version 3.2.3.

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. Enabling SCTP would require rooting the device at this point, thus breaking the requirements we discussed in Section II. Thus, we left experiments with SCTP on Android for future work.

TABLE II. Modified ksoap2-android library functionality overview.

Compression methods
Bidirectional uncompressed SOAP
Bidirectional gzip compressed SOAP
Bidirectional EXI compressed SOAP
SOAP transports
HTTP/TCP
UDP
AMQP

B. Modifying ksoap2-android

The ksoap2-android library supported uncompressed SOAP communication over HTTP/TCP. Also, it contained functionality to receive a gzipped response, but was unable to create a gzipped request. We modified the library so that it would support the following in addition to the existing functionality: Gzip and EXI compressed SOAP requests and responses, as well as using UDP and AMQP as transport layers. All compression methods and transports can be used interchangeably. The complete capabilities of our modified ksoap2-android library is shown in Table II.

C. Retaining compatibility with a COTS application 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, since they can provide a separation of concerns between proprietary enhancements and COTS services. Leveraging gateways/proxys for interoperability (often called interoperability points) is in key with the NATO mindset, and is discussed further in [4], [11].

Figure 1 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 2 shows the details of the inner workings of the proxy. Here we see that different transport protocol alternatives are mapped to the respective library. Each

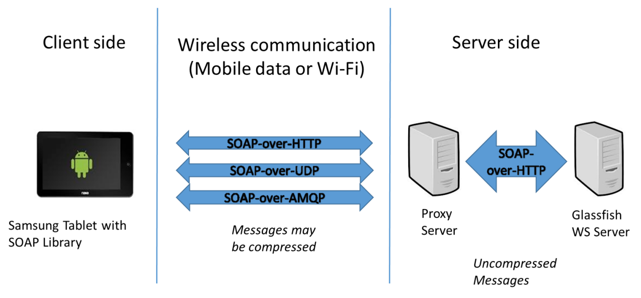

Fig. 1. The role of the proxy server

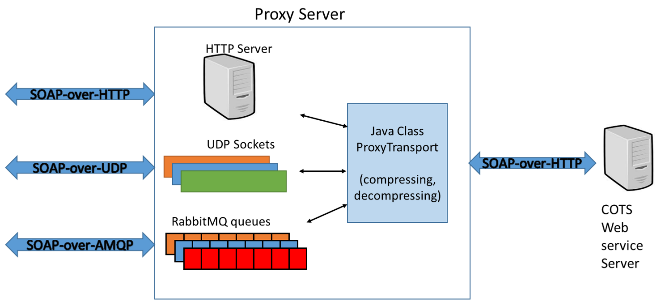

Fig. 2. Detailed overview of the proxy server

transport alternative can also support all three compression alternatives. In the back-end the proxy uses uncompressed SOAP over HTTP/TCP to access the Web services in the application server, thus ensuring compatibility with COTS Web services.

The ksoap2-modifications and our proxy server have both been released as open source and are available at https://github.com/dagove.

V. Evaluation

We evaluated the different approaches using two different setups. In one case, illustrated in Figure 3, we used the mobile network from the Android device, accessing the Web service across the Internet and in an application server hosted on a wireless network. In the second test case, shown in Figure 4, the Android device was present on the same wireless network as the server, accessing it directly.

For each configuration we tested using three different services:

1) A simple request/response service, where both request and response contains a short text message.

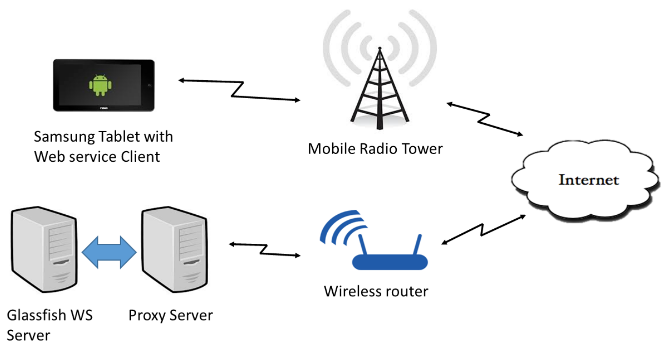

Fig. 3. Tests using the mobile network on the Android device

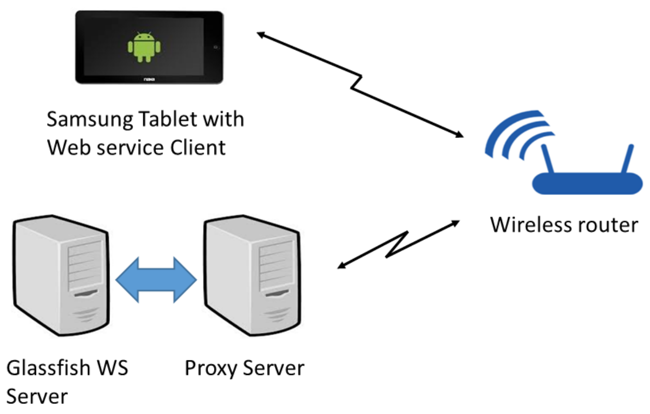

Fig. 4. Tests using the wireless network on the Android device

2) A simple picture service, where the Android device and server exchange a JPEG-formatted file.
3) An NFFI[4] reporting service, where the device sends NFFI-formatted positions to the server.

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

The Web services used for testing were hosted on a Glassfish server. Our own developed proxy server was used as an intermediary to interface our Android client application with the corresponding COTS Web services. Both servers were installed on the same computer, an ASUS Notebook N56V running Windows 8.1. The notebook specifications were as follows:

[4]NATO Friendly Force Information (NFFI) is a NATO standard (STANAG 5527) for reporting the position of friendly forces.

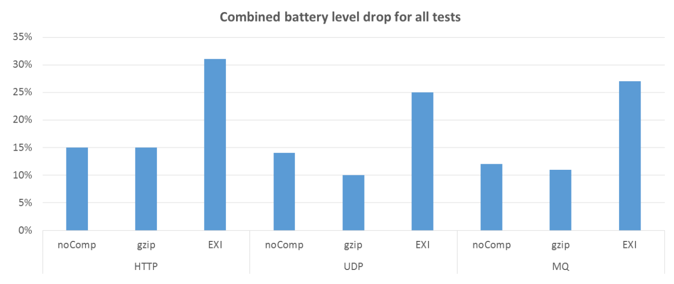

Fig. 5. Battery usage tests. The Y-axis represents the percentage drop in battery capacity, meaning that a lower number indicates better battery life.

- 4-Core 2.4 GHz CPU
- 8 GB RAM
- 750 GB HDD
- Wi-Fi 802.11 b/g/n support

We used a Samsung Galaxy Tab 2 (GT-P5100) for testing purposes. The tablet specifications were as follows:

- Dual-core 1.0 GHz CPU
- 1 GB RAM
- 16 GB storage
- 802.11 a/b/g/n, Wi-Fi Direct, and HSPDA[5] support.
- Standard battery, Li-ion 7000 mAh
- Android 4.2.2

The tests were executed for a duration of several hours, constituting of 150 000 requests. This was done to minimize the impact of any scheduled, time-limited inherent background processes in the operating system on the overall measured results. Figure 5 shows the aggregated results from all tests related to battery life. The Y-axis indicates the percentage drop in battery capacity. This means that a lower number indicates longer battery life. We see that the general trend is that EXI compression has a negative impact on battery life compared to using gzip or no compression, regardless of the transport mechanism used.

Figure 6 shows the aggregated CPU load for each service type. These results corroborate the battery usage tests, in that we see that EXI in general is much more demanding of CPU resources than the other alternatives.

Figure 7 shows the network load for the different services using different compression methods and transport alternatives.

Figures 8 and 9 show the goodput[6] for the tests using mobile broadband and the wireless network, respectively.

EXIficient performed badly with respect to 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 no compression at all. The results also show that UDP and RabbitMQ 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 percent) when dealing with large XML files (i.e., the NFFI Web service). The effect was not so pronounced when dealing with small XML files (i.e., the simple Web service) or with non-XML data (i.e., the 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, which was to be expected. The test results also show that UDP and RabbitMQ performed generally better than HTTP did, especially when considering goodput.

VI. Conclusion

Our goal was to investigate how the Android platform can function as an efficient client of Web services. By efficient, we mean maximizing battery life while minimizing network use. In this paper we explored and compared different ways to transport and compress SOAP in order to give recommendations regarding how to achieve this goal.

The library ksoap2 for Android proved to be a functional and lightweight SOAP library that can be altered to support different transport mechanisms and compression techniques. We extended the library with support for different SOAP transports and compression methods. This modified library along with our proxy server that

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

[6]Here, *goodput* means the application level throughput, that is the number of useful information bits delivered by the network to a certain destination per unit of time.

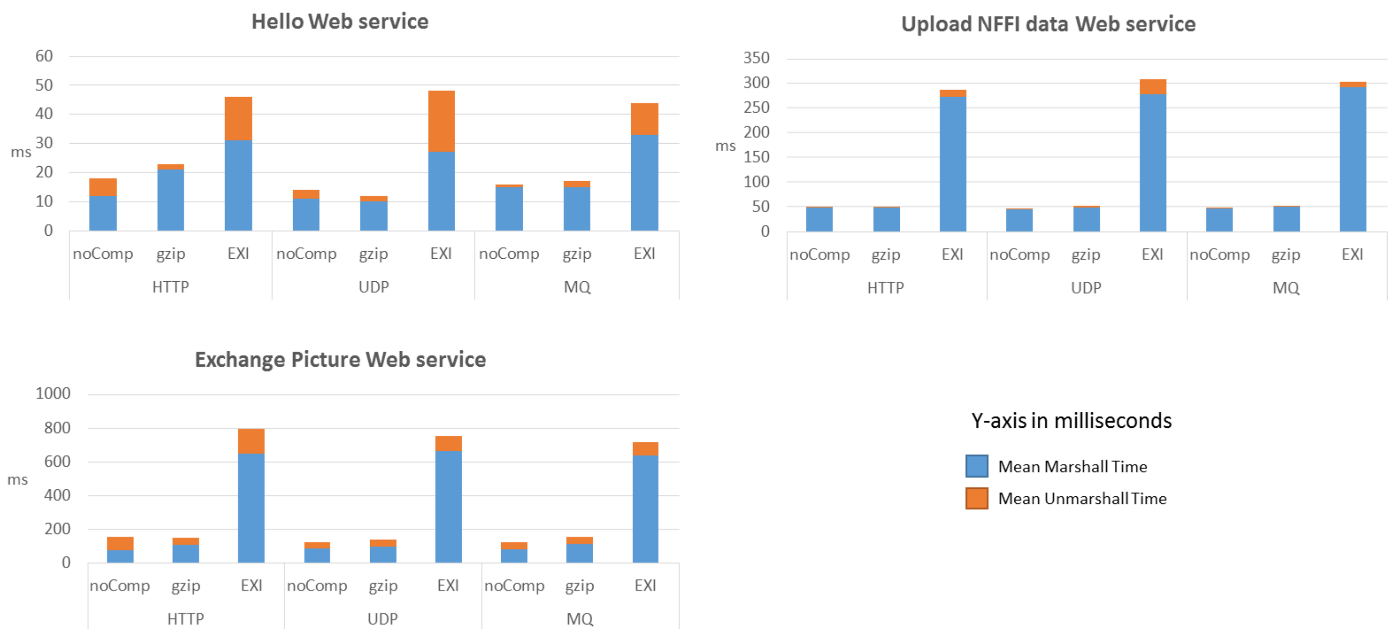

Fig. 6. CPU load by service type.

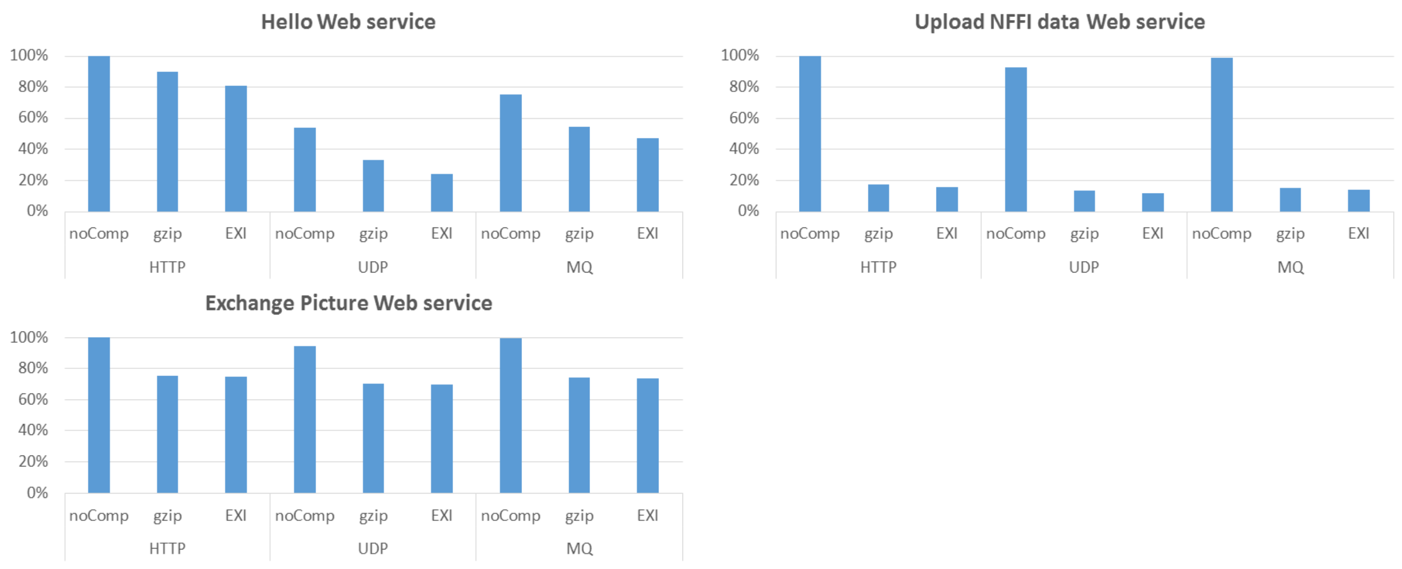

Fig. 7. Network load (data transmitted). The Y-axis shows percentage relative to the HTTP/No compression alternative.

can be used to interface the SOAP optimizations with a COTS application server have both been released as open source.

Our tests included using both the embedded wireless and the mobile network support, as well as executing tests involving exchanging SOAP messages with payloads consisting of text strings, JPEG-formatted images, and a standardized XML-based positioning format.

Based on our tests, the trend seems that using gzip together with AMQP (we used the RabbitMQ library — other implementations may behave differently) can be better than the other tested alternatives when it comes to reducing network overhead while simultaneously maximizing battery lifetime for a reliable connection. If, however, entirely reliable communication is not required, then using gzip together with UDP can be better than the alternatives when it comes to reducing network overhead while simultaneously maximizing battery lifetime for an unreliable connection. In general EXI (we used the

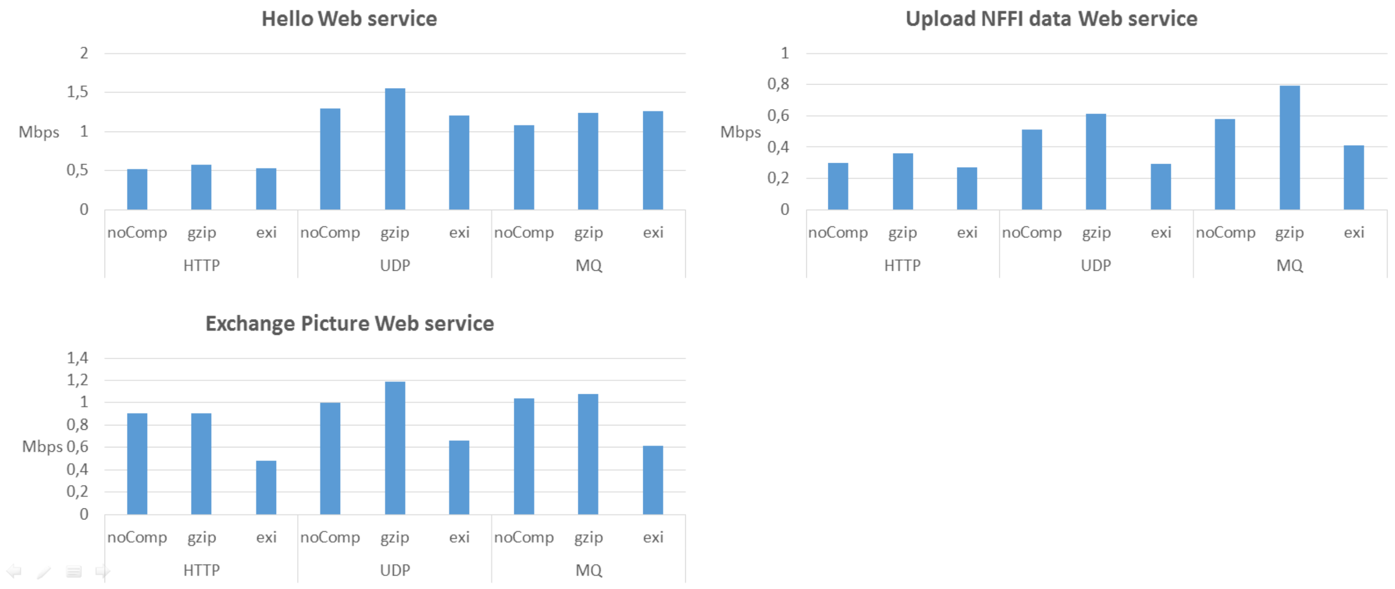

Fig. 8. Goodput using mobile broadband on the Android device.

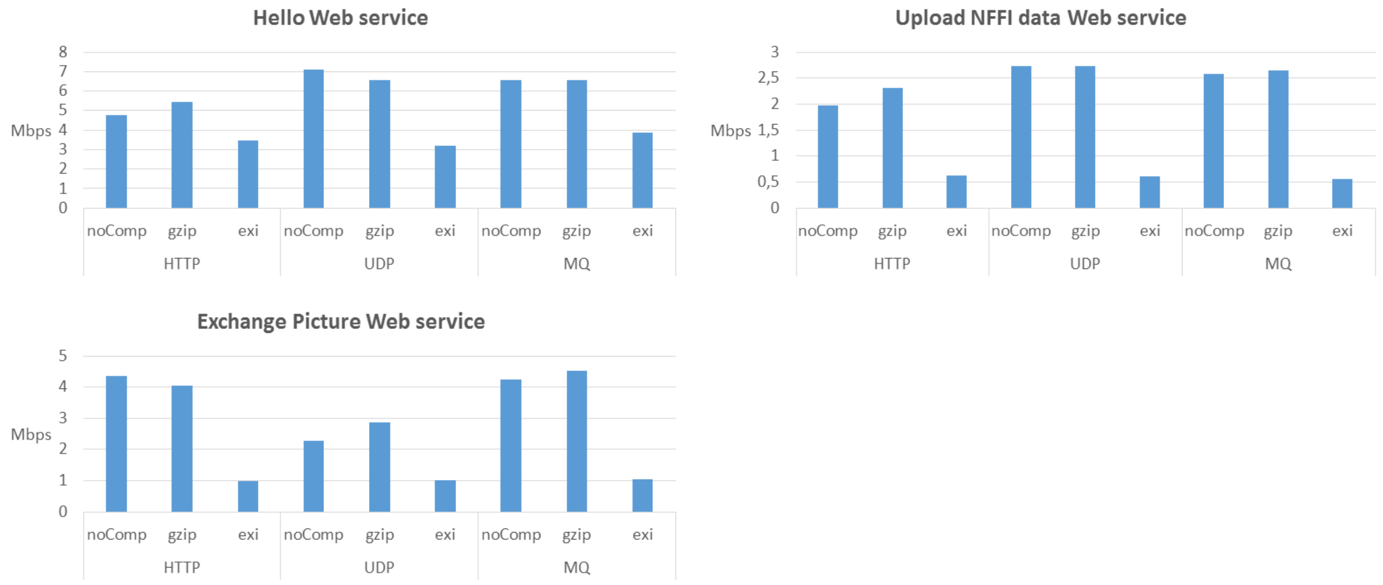

Fig. 9. Goodput using wireless network on the Android device

EXIficient library — other implementations may behave differently) performed so poorly with respect to CPU load and battery lifetime that we advice against using it on Android.

Acknowledgment

The authors would like to acknowledge professor Knut Øvsthus at Bergen University College and professor Josef Noll at UNIK. Their efforts and insights have contributed to the work discussed in this paper. Last but not least, thanks to Johnny Johnsen for proofreading the paper.

REFERENCES

[1] J. R. Agre, K. D. Gordon, and M. S. Vassiliou. Practical considerations for use of mobile apps at the tactical edge. 19th International Command and Control Research and Technology Symposium (ICCRTS), Alexandria, VA, USA, June 16-19, 2014.

[2] Annunziata, F. et al. IST-090 SOA Challenges for Disadvantaged Grids. 15th International Command and Control Research and Technology Symposium (ICCRTS), Santa Monica, California, USA, June 22-24, 2010.

[3] Augeri, C.J., et al. An Analysis of XML Compression Efficiency. Workshop on Experimental Computer Science (ExpCS). New York, NY, USA., 2007.

[4] P. Bartolomasi, T. Buckman, A. Campbell, J. Grainger, J. Mahaffey, R. Marchand, O. Kruidhof, C. Shawcross, and K. Veum. NATO network enabled capability feasibility study. Version 2.0, October 2005.

[5] Consultation, Command and Control Board (C3B). Core enterprise services standards recommendations: The soa baseline profile version 1.7. Enclosure 1 to AC/322-N(2011)0205, NATO Unclassified releasable to EAPC/PFP, 11 November 2011.

[6] A. Gbor. AndroidSOAP. http://wiki.javaforum.hu/display/ANDROIDSOAP/Home, Accessed 29 August 2014.

[7] International Organization for Standardization (ISO). Fast infoset. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=41327, 2007.

[8] G. Jaiswal and M. Mishra. Why use Efficient XML Interchange instead of Fast Infoset. in proceedings of IEEE Advance Computing Conference (IACC), pp. 925 – 930, Ghaziabad, India., 22–23 February 2013.

[9] F. T. Johnsen, T. H. Bloebaum, M. Avlesen, S. Spjelkavik, and B. Vik. Evaluation of transport protocols for web services. Military Communications and Information Systems Conference (MCC 2013), 7-8 October, Saint-Malo, France, 2013.

[10] F. T. Johnsen, T. H. Bloebaum, P.-P. Meiler, I. Owens, C. Barz, and N. Jansen. IST-118 — SOA recommendations for Disadvantaged Grids in the Tactical Domain. 18th International Command and Control Research and Technology Symposium (ICCRTS), Alexandria, VA, USA, June 19-21, 2013.

[11] F. T. Johnsen, T. H. Bloebaum, L. Schenkels, R. Fiske, M. van Zelm, V. de Sortis, A. van der Zanden, J. Sliwa, , and P. Caban. Soa over disadvantaged grids experiment and demonstrator. *Military Communications and Information Systems Conference (MCC 2012), 8-9 October, Gdansk, Poland*, 2012.

[12] L. Johnsrud, D. Hadzic, T. Hafsøe, F. T. Johnsen, and K. Lund. Efficient web services in mobile networks. *The 6th IEEE European Conference on Web Services (ECOWS), Dublin, Ireland*, November 2008.

[13] NeuroSpeech Inc. Wsclient++. http://wsclient.neurospeech.com/wsclient/java-android-blackberry/, Accessed 29 August 2014.

[14] K. A. Phan, Z. Tari, and P. Bartok. A benchmark on soap's transport protocols performance for mobile applications. *ACM SAC '06, 23-27 April, Dijon, France*, 2006.

[15] B. K. Reitan, M. Fidjeland, H. Hafnor, and R. Darisiro. Approaching the mobile complex — in search of new ways of doing things. 17th International Command and Control Research and Technology Symposium (ICCRTS), Fairfax, VA, USA, June 19-21, 2012.

[16] Shen, Z., et al. A light mobile web service framework based on axis2. in Future Information Communication Technology and Applications, ISBN: 978-94-007-6515-3, 2013.

[17] Tari, Z., et al. Benchmarking SOAP Binding. in On the Performance of Web Services, pp. 35–58, ISBN: 978-1-4614-1930-3, 2011.

[18] Teixeira, M.A., et al. New Approaches for XML Data Compression. in proceedings of International Conference on Web Information Systems and Technologies (WEBIST 2012), pp. 233–237., 2012.

[19] The ksoap2-android project. ksoap2-android. https://code.google.com/p/ksoap2-android/, Accessed 29 August 2014.

[20] W3C. Efficient XML Interchange (EXI) Format 1.0 (Second Edition). W3C Recommendation, http://www.w3.org/TR/2014/REC-exi-20140211/, 11 February 2014.

[21] W3C Working Group Note. Web services architecture. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, 11 February 2004.

[22] C. Werner, C. Buschmann, T. Jäcker, and S. Fischer. Bandwidth and latency considerations for efficient SOAP messaging. *International Journal of Web Services Research, Vol. 3, Issue 1*, 2005.