Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 1183–1189

DOI: 10.15439/2015F6
ACSIS, Vol. 5

Analysis of notification methods with respect to mobile system characteristics

Piotr Nawrocki *, Mikołaj Jakubowski † and Tomasz Godzik ‡
*AGH University of Science and Technology,
al. A. Mickiewicza 30, 30-059 Krakow, Poland
e-mail:piotr.nawrocki@agh.edu.pl
†e-mail:mkl.jakubowski@gmail.com
‡e-mail:tomek.godzik@gmail.com

***Abstract*—Recently, there has been an increasing need for secure, efficient and simple notification methods for mobile systems. Such systems are meant to provide users with precise tools best suited for work or leisure environments and a lot of effort has been put into creating a multitude of mobile applications. However, not much research has been put at the same time into determining which of the available protocols are best suited for individual tasks. Here a number of basic notification methods are presented and tests are performed for the most promising ones. An attempt is made to determine which methods have the best throughput, latency, security and other characteristics. A comprehensive comparison is provided, which can be used to select the right method for a specific project. Finally, conclusions are provided and the results of all the tests conducted are discussed.**

I. Introduction

THE PURPOSE of this paper is the analysis and tests of several selected notification methods for mobile platforms. The reason for this research is the need to determine the best way of sending simple as well as more advanced messages about the events involved in the operation of grid systems or telemetric networks. This makes it possible to use the optimal approach in numerous projects that need to inform users about their current status. This aspect is currently of utmost importance for the industry as such notification methods enable developers to engage users much more fully and keep them in constant contact with their leisure and work interests. These considerations have guided us throughout our research and affected all our decisions on the selection and ways of testing of the methods in question.

Several protocols and methods were considered based on their purposes and current industry standards. The main candidates were: CoAP (Constraint Application Protocol), XMPP (Extensible Messaging and Presence Protocol) and XMPP over SOAP (Simple Object Access Protocol), MQTT (Message Queuing Telemetry Transport), MQTT-SN (Message Queuing Telemetry Transport for Sensor Networks), AMQP (Advanced Message Queuing Protocol), Cloud notification systems (Google Cloud Messaging, Urban Airship), SMS (Short Message Service) and Restful HTTP (Hypertext Transfer Protocol).

Of course, these are not all the protocols that could be used for mobile notifications, but these listed appear to hold the most promise and therefore the purpose is to discern their usefulness in the best way possible.

In addition to the protocols and methods above, we investigated other solutions, such as the Apple push notification or Line application which, for various reasons, were not considered further. The Apple push notification technology is a good solution, but it is proprietary, i.e. limited to Apple devices and that is why we decided to test more universal solutions first. There are also solutions (applications) that use their own protocols. A good example is Line application, which uses a proprietary protocol. We considered testing this solution; however, there are significant difficulties with accessing the documentation for this protocol.

II. Related Work

Mobile systems are a relatively new field of study and picking a specific topic such as comparing available notification methods does not return many related work results. Some of the protocols have been covered in separate articles and while these took the sending of notifications into account, tests were not always conducted in mobile systems.

The one available article [1] that compared notification methods only covered cloud systems [2] and applications. It discussed the following methods: C2DM (Google Cloud to Device Messaging—the predecessor to GCM), Xtify, XMPP and Urban Airship. As during our research that article was relatively new, one might think that the information contained there would still be relevant, but it turned out to be already out of date. Google has meanwhile redesigned and rebranded its notification system and Xtify was purchased by IBM. Only Urban Airship is still available on the market in the same configuration as previously. The article is more a comparison of available commercial products than a real world testing suite. As expected, the conclusion was that the fastest protocol of the four tested was XMPP, but it had a characteristic slightly different from the others.

Another article [3] only tested the MQTT protocol. The author believed that it was the best possible choice and only aimed to describe its main features and capabilities. Just a single simple test and its averaged results together with the amount of data transferred and power consumption over a period of time were provided. In the conclusion, the author

978-83-60810-66-8/$25.00

described the MQTT protocol as being both lightweight and perfect for mobile platforms.

In [4], the authors investigated XMPP in the field of collaborative applications. Its main purpose was to assess the usefulness of XMPP in exchanging location data between mobile clients and web servers. No testing was conducted, but a thorough description of XMPP and the Android platform was provided while also taking into account the ways of integrating them. The article described XMPP as a general purpose messaging protocol that is easily extensible.

An important aspects in the context of notification methods are SLA parameters [5] and the power consumption of battery-powered devices. In [6], the authors discussed the problem of sending notification data using GPRS connectivity from remote telemetry stations [7]. They proposed the concept of adaptive message aggregation which extends the MQTT-SN protocol, adjusting its behaviour to the GPRS (General Packet Radio Service) connectivity profile in order to decrease the energy consumption related to data transmission.

III. Notification methods

The following section generally describes and analyses the possible notification methods for mobile devices mentioned in the introduction. As a result of this analysis, it was decided to select some of them in order to perform the thorough tests described later in this paper.

A. SMS

It is possible to use the Short Message Service as a notification mechanism. An application would have to intercept the SMS messages received by an Android phone and analyse them to check whether they contain notifications from the system. One could just use simple text messages without a dedicated client application, but this would severely limit the functionality available to users.

This approach has several major issues. First, the cost of sending multiple messages to numerous clients could be immense. Secondly, it is not guaranteed that the message will be delivered on time or (sometimes) even on the same day. What is more, all text messages have a maximum undelivered period (which cannot exceed 7 days), and this means that some notifications would not be delivered at all.

B. Google Cloud Messaging

In order to simplify the development of applications and to extend phone battery life, Google has created a simple built-in notification system for the Android platform, which only maintains a single connection at any time.

This approach has some obvious drawbacks. Firstly, the number of messages sent concurrently is limited to four per application and there is no guarantee that the message will be delivered, especially while the service is shared. Secondly, there is no specified maximum delay, which is not acceptable for most modern systems. Moreover, in posts like [8] it is claimed that the method is not all that well documented and it is not easy to make an application work reliably with Google Cloud Messaging (GCM). Another problem with GCM is that some people do not trust Google not to abuse its capabilities, citing privacy or security concerns. One must also keep in mind that GCM can be used by some malware applications as described in [9].

C. Restful HTTP

Another possible solution would be to use a RESTful HTTP service based on a pull queue model [10]. Such an implementation would have to pull notifications from the server at certain intervals or when the user turns on the application. Currently creating such a service is a very simple process and does not require additional knowledge from most developers, which is the main advantage of this approach.

However, using this method is very inefficient as it is not clear at what intervals requests should be made. Using too long an interval between requests may result in multiple notifications being sent all at once, making the older messages meaningless. Conversely, if the interval were too short, it would use too much device resources. Moreover, much of the workload is shifted to the mobile device and the amount of data sent between server and client is sometimes doubled.

Some ideas for REST notification systems are discussed in [11], however using a pure REST approach is highly discouraged. Using the AMQP/REST mixed approach seems much more plausible.

D. XMPP

XMPP is basically an open technology for real-time communication, using XML (Extensible Markup Language) as the base format for exchanging information. It was designed to be easily extensible and one of its main uses are publish-subscribe systems. It is the most mature protocol among all the solutions selected as it was already in use in 1999. Throughout its history it was used by companies such as Google in the Google Talk communicator, by Microsoft in Skype or by Facebook in WhatsApp Messenger.

The idea behind XMPP is similar to that of e-mail, with a distributed server network in which each and every server can create its own service. The XMPP standard enables message encryption and XML support allows for the use of such technologies as SOAP or EDI (Electronic Data Interchange).

A standard that is tightly coupled with XMPP is SOAP over XMPP, which can be tested using the same means, as sending a SOAP message is basically sending some content over XMPP, which provides effective and reliable messaging—both asynchronous and synchronous.

XMPP is a general purpose protocol that is easily extensible. It was only designed to meet mobile platform requirements and was not expected to outperform any other protocols. However, its flexibility makes it a choice worth considering. In [4], a few add-ons are mentioned like group chats or streaming services with a possibility to transfer files.

E. SOAP over HTTP

SOAP is a lightweight protocol for message exchanges that is independent from the system platform programming

language. Its specification does not define a specific transport layer protocol, but most implementations use HTTP. It is important to mention, however, that HTTP is of no use for asynchronous messaging and because of that SMTP is often used instead. The protocol makes it possible to send many short messages.

In the discussion on the use of SOAP in notification systems, the following solutions should be considered: polling, both endpoints having their SOA interfaces, using WS-notification and using the message queueing solution encapsulated in HTTP.

All the solutions above have been analysed and none of them are easy to adapt to the needs of mobile notification systems. The first solution requires the client to make requests at certain points in time, which generates a lot of unnecessary traffic and is quite resource-heavy on small devices. The second idea is better, but would not work for most mobile devices as not all requests would pass from the server to the device since such HTTP requests are often blocked. A good solution is to use WS-notification, but the problem with making requests from the server is still present. What is more, it is not a standard supported by all web servers. The final solution uses queueing, but it involves a lot of unnecessary technology, especially given that there are ready-made queueing mechanisms that do not have to be encapsulated in HTTP requests.

F. MQTT/MQTT-SN

MQTT is a publish-subscribe lightweight messaging protocol based on TCP/IP. It was designed to be open, simple, lightweight and easy to implement, since it was intended to be used in constrained environments with limitations such as: expensive, low bandwidth, unreliable network, limited processor or memory resources.

The entire protocol is based upon a central message broker, which distributes messages published on a topic to all the interested consumers. The "MQ" part of the name comes from "Message Queueing", however this protocol does not support queuing by default. It has three types of quality of service for message delivery, which are "At most once", "At least once" and "Exactly once". It also has a mechanism that can be used to inform interested parties about an abnormal disconnection using the "Testament" and "Last Will" features.

What is interesting is the fact that MQTT has already been used in numerous applications. The first implementation of GCM(C2D)[1] used exactly this protocol. DeltaRail's latest version of their IECC (Integrated Electronic Control Centre) also uses MQTT for communications within their signalling system, which is covered in [12].

This standard was created by IBM and because of that fact the IBM MQTT client Java library was used for testing and the Mosquitto open source message broker for distributing messages. Mosquitto's simple construction allowed us to create a bash script sending a set number of messages. The Android client connects to the broker using the IBM library and is fed the messages sent by the script.

MQTT-SN is a variation of MQTT designed to be used in sensor networks. In particular, it is supposed to be lightweight and easily implementable on small devices (e.g. in non-TCP/IP[2] networks).

G. CoAP

CoAP[3] is a specialised web transfer protocol for use with constrained nodes and networks based on UDP (User Datagram Protocol). Its main task is to allow for communication between small devices such as sensors, switches, etc. It was designed on the basis of HTTP in order to simplify its architecture and allow for multicast. It also provides a simple mapping between CoAP and HTTP, which can be used to create RESTful services. The messages are sent in a binary format and their size is limited by the maximum size of a datagram. Messages can be sent with acknowledgements or without them depending on the designer's needs. Although it is a relatively new standard, it already has some additional features proposed like "Observable", which makes it possible to notify all the clients subscribed about changes to the resource.

H. AMQP

AMQP is an open standard application layer protocol for message-oriented middleware that uses a binary format to send its messages. It was designed to solve the problem of interoperability between heterogeneous systems and message brokers. It was first used in 2006 by JP Morgan. It offers both point-to-point and publish-subscribe messaging types.

The most important advantage of AMQP is the fact that it is independent of programming languages and platforms unlike most messaging standards, for example JMS (Java Message Service) [13]. Moreover, it offers several types of quality of service in terms of delivery guarantees; these types are at-most-once, at-least-once or exactly-once guarantees. It also allows to encrypt messages, which is important especially in the case of valuable scientific data. Currently it is a widely used standard and has a large number of implementing libraries like Apache Qpid, RabbitMQ [14] or StormMQ.

IV. TESTS

There are currently three main mobile operating systems (Android, iOS and Windows Phone) available on the market and numerous devices that support them. As it would be neither possible nor sensible to test each and every one of them, only one testing platform and device was chosen.

Google's Android system was selected as the mobile platform for testing purposes because of the considerable availability and open nature of the solution. All major protocols and methods selected have working implementations for this system.

[1]Android Developer Central - GCM Advanced Topic - http://developer.android.com/google/gcm/adv.html

[2]Transmission Control Protocol/Internet Protocol

[3]CoAP RFC 7252 - http://tools.ietf.org/html/rfc7252

As a mobile device the Nexus 5 (LG D821) was used with Android version 4.4.3 using the standard Dalvik engine. At this point Android Runtime was already available but it seemed not yet ready for serious testing. All data (from a mobile device) was transmitted using an HSPA technology (operator: T-Mobile).

In order to conduct all tests, a server platform was also needed, which consisted of an Asus laptop with the Intel i5-3320M processor and 8GB of RAM with Ubuntu 12.04LTS and Oracle Java 1.7.0_60 installed. All data (from a server platform) was transmitted over the Wi-Fi network using the 802.11g standard (54 Mbps).

For time-related test cases ClockSync[4] was used to synchronise with time servers on the mobile device. All applications launched their message connectors in separate threads. Services were not used so all memory usage diagrams show the combined values of the connector and activity screen.

All useful notification methods should meet most of the specifications listed below:

- the financial cost should be low—mostly for open source and university projects;
- it should be possible to transmit more than just simple text—to enable interaction between the application and the main system;
- energy and memory usage should be minimal—the solution is to be used on mobile devices;
- message contents should be secure—confidential information could be transmitted;
- minimal message loss—important data could be transmitted;
- minimal delay—fast interaction is sometimes needed.

As a result of analysing the notification methods available (see Section III) and taking the above assumptions into account, it was decided to test the following protocols: XMPP, SOAP, MQTT, CoAP and AMQP.

In order to conduct testing for each protocol, the following solutions were used:

- XMPP—in order to prepare the XMPP test, the Smack library was used to implement the mobile client. It has been ported to Android in a version called Asmack. ejabber was used as a message broker. The second client, which was sending messages to the mobile client, was implemented in Python using the SleekXMPP library. The mobile solution was plain and simple with its task limited to keeping an open connection to the broker.
- SOAP—an attempt was made to test a basic polling mechanism using a simple Python SOAP server[5] and a basic Android client[6]. It involved making a number of requests for stress testing and a single request to measure single message performance. After obtaining initial results this method was discarded as it was more than 10 times slower than any other and used a lot of resources for polling, which is unacceptable for mobile devices. It was decided to concentrate efforts on other solutions specifically intended for such devices. The results collected are shown together with the other protocols tested, but are not included in graph comparisons (in Figure I) as they were much worse than for any other test conducted.
- MQTT/MQTT-SN—a broker that works with the MQTT/MQTT-SN protocol is the RSMB (Really Small Message Broker) from IBM and while it is quite easy to find, locating an appropriate client library (especially for MQTT-SN) is much more difficult. The one that is available for MQTT-SN[7] is written in the C programming language so it was of no use whatsoever for Android devices. The project also appeared to have been abandoned (no recent contributions). A further search led to a library written in Python[8], which was then used to implement a client that would be run using the QPython interpreter. After a certain amount of research and testing a stage was reached where messages were delivered from the broker to the device but never reliably. Attempts to change QoS settings failed as it appeared that the client library implementation was not complete yet. For these reasons MQTT-SN was excluded from testing and only MQTT was tested. It appears that the protocol is not mature enough yet to be used on a larger scale and that it has no reliable or finished implementations.
- CoAP—there is a limited number of implementations. The main Java libraries are jCoAP and nCoAP. jCoAP is not up to date with the RFC (Requests for Comments) 7252 and therefore nCoAP was used which additionally implements the "Observable" feature. To test the protocol, a simple server was created with a time service and a mobile client that was sending GET requests to the server. At first it was intended to use the "Observable" feature, but during stress testing it turned out that messages cannot be sent too often using this implementation because of errors. As a consequence, although "Observable" can be quite useful, especially in mobile systems, it was decided to have each notification sent as an answer to a separate GET request.
- AMQP—RabbitMQ was selected, which is one of the best documented and popular libraries. For testing purposes a simple Python script was developed that can send a set number of simple messages containing timestamps and an Android client application. The client connects to the RabbitMQ message broker and then the Python script is used to send messages.

To assess the efficiency and usefulness of the notification methods selected, several test cases were created and run for each protocol. It is not claimed that it is a complete test suite,

[4]ClockSync - http://amip.tools-for.net/wiki/android/clocksync

[5]Python simple and lightweight SOAP Library (a.k.a. soap2py) - https://code.google.com/p/pysimplesoap/wiki/SoapServer

[6]A simple SOAP client for Android - https://code.google.com/p/droidsoapclient

[7]MQTT-SN client in C - https://github.com/njh/mqtt-sn-tools

[8]MQTT-SN client in Python - http://git.eclipse.org/c/mosquitto/org.eclipse.mosquitto.rsmb.git/tree/rsmb/src/MQTTSClient/Python

but rather preliminary testing. More work remains to be done in this field.

A. Time per message

Each of the four applications designed to check how much time it takes to process a single message while stress testing was used with different numbers of concurrent messages sent. Set sizes of 10, 50, 100, 200, 300, 400 and 500 messages were chosen. Each message contained its timestamp in order to enable the calculation of exact delay. Figures 1 (nCoAP), 2 (MQTT), 3 (RabbitMQ), 4 (XMPP), 5 (SOAP over HTTP) show how much time it took to process a single message for different stress test set sizes.

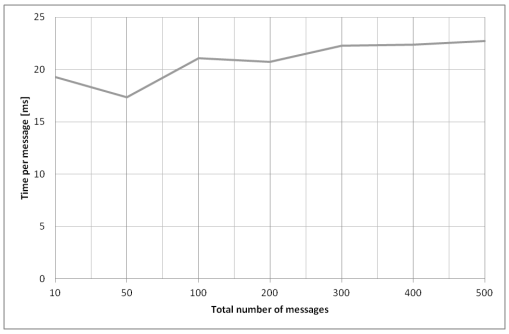

Fig. 1. nCoAP

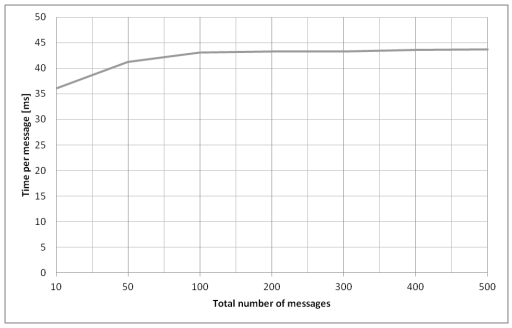

Fig. 2. MQTT

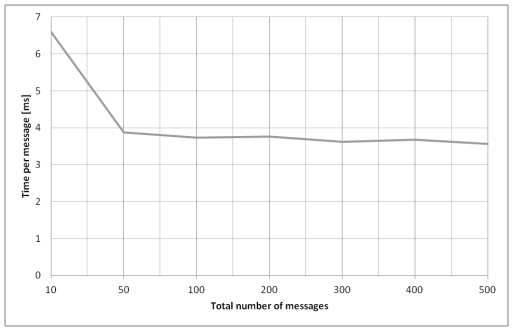

Fig. 3. RabbitMQ

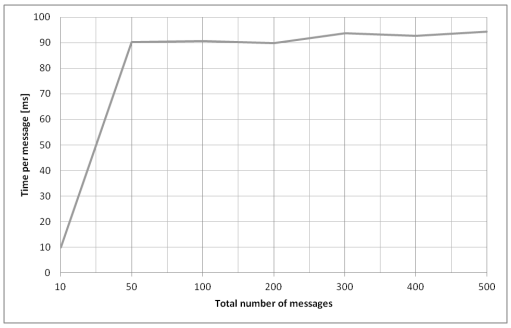

Fig. 4. XMPP

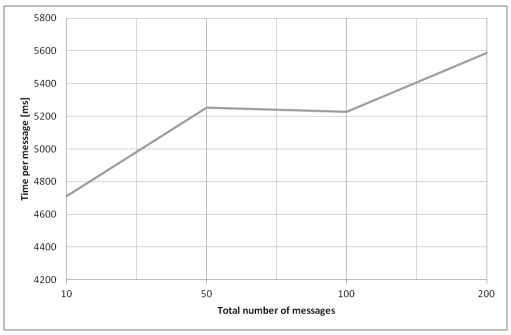

Fig. 5. SOAP over HTTP

Based on the graphs generated it can be stated that RabbitMQ is the fastest in terms of performance and its performance actually improves as more messages are sent concurrently. The nCoAP was also quite effective, but it should be kept in mind that each message was sent in response to a GET request, so it could be faster yet. Both MQTT and XMPP exhibit quite long message sending times. However SOAP over HTTP being definitely the slowest solution. In this test, RabbitMQ was the clear winner.

B. Resource usage

The second most important criterion after performance was resource usage. It is crucial to use as little device resources as possible on a mobile platform in order to consume less power and allow for greater efficiency. In this section, peak memory usage (shown in Table I - “RAM (Random Access Memory) usage peak”) and CPU (Central Processing Unit) power consumption (by using PowerTutor tool [15]) were measured as presented in Figures 6 (nCoAP), 7 (MQTT), 8 (RabbitMQ), 9 (XMPP), 10 (SOAP over HTTP), while sending 1000 messages concurrently to be processed by each of the mobile clients developed.

It is clearly visible that almost all protocols used similar amounts of memory, with the only outlier being MQTT with 10 MB less RAM usage than others. A much larger difference can be seen in power consumption levels. These seem to be strongly correlated with each individual protocol’s processing time. nCoAP and RabbitMQ consumed the least power. About two times more power was consumed by XMPP and SOAP

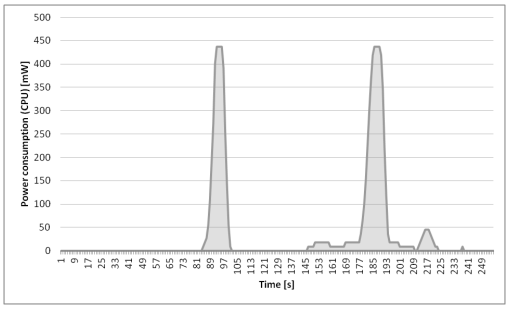

Fig. 6. nCoAP (2 consecutive runs shown)

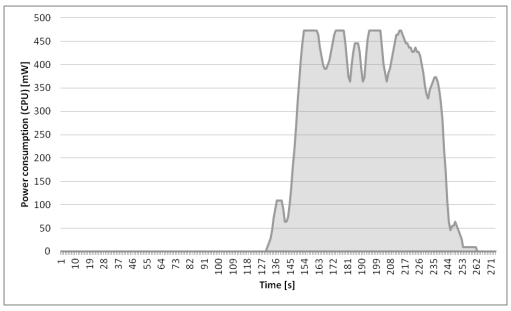

Fig. 7. MQTT

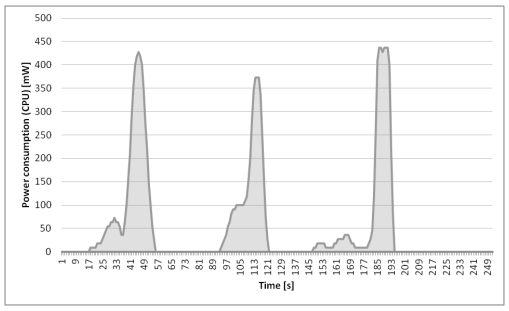

Fig. 8. RabbitMQ (3 consecutive runs shown)

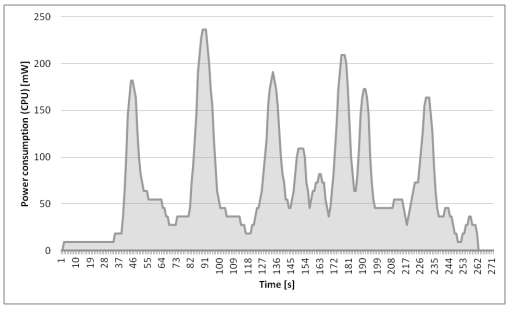

Fig. 9. XMPP

over HTTP. The worst result was achieved by the MQTT protocol.

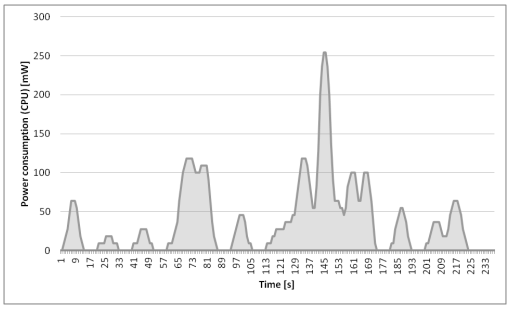

Fig. 10. SOAP over HTTP

C. Reliability

Message sending reliability was also tested as it is one of the crucial issues to be tackled in mobile notification applications. Especially important is the issue of what happens to messages in a queue when the connection to the client is lost. This was simulated by reconnecting to a Wi-Fi network while sending a set of 1000 messages. All protocols were tested using default settings. It turned out that only nCoAP managed to deliver all messages, while the other protocols lost some or most of the messages sent. Developers must take care to use the correct settings for each protocol as QoS is not usually switched on by default.

D. Ordering

This test case was meant to show whether protocols deliver messages in the same order in which they were sent. Similarly to the first test, a set of messages was sent containing timestamps and the comparison of arrival times of successive messages made it possible to determine whether they were correctly ordered. Only nCoAP changed the order of messages, which is most probably caused by using UDP. All other protocols delivered messages in the correct order even during high load.

E. Average delay

The final test case was used to calculate the average delay when sending a single isolated message using each of the five protocols (Table I - "Average delay"). It turns out that nCoAP is the fastest when it comes to sending individual messages and SOAP over HTTP is the slowest one among all the protocols tested.

V. Conclusions

The results as shown in Figure 11 clearly demonstrate that in terms of the maximum number of messages delivered per second RabbitMQ is the leader; however, when it comes to minimal delay, nCoAP tends to be able to deliver single messages much quicker. This means that if large numbers of notifications are to be sent, RabbitMQ could be used, while in a sparse notification system CoAP should perhaps be recommended. In the power consumption test the best results were also achieved by RabbitMQ and nCoAP.

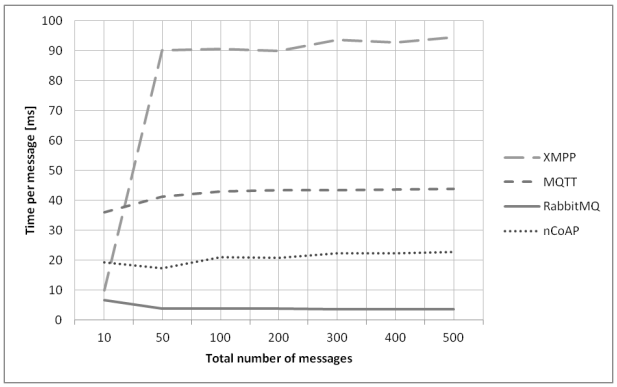

Fig. 11. Comparison of protocols

It should also be noted that not all protocols are able to easily pass through firewalls and NATs like XMPP, which is the most mature of all the protocols tested. When it comes to RAM usage, MQTT turned out to require the least megabytes, which can be of great importance on mobile devices. A summary of test results is shown in Table I.

In conclusion, the most promising solution seems to be RabbitMQ but none of the protocols proposed outperformed all the others in all test cases. This test suite only demonstrates the performance of some implementations currently available and the results might change for future releases or different platforms. Before any protocol is selected, it is important to specify the needs of the project in question and then compare the protocols to determine which one best suits these needs.

TABLE I
OVERVIEW OF PROTOCOL PROPERTIES

	nCoAP	RabbitMQ	MQTT	XMPP	SOAP
RAM usage peak [MB]	47	44	35	46	38
Average delay [ms]	91	185.5	339.6	192.3	972.2
Ordered	no	yes	yes	yes	yes
Lost messages	no	yes	yes	yes	yes
Content	binary	binary	binary	text	text
Based on	UDP	TCP	TCP	TCP	TCP
SSL support	no	yes	yes	soon	yes

ACKNOWLEDGMENT

The research presented in this paper was partially supported by the National Centre for Research and Development (NCBiR) under Grant No. PBS1/B9/18/2013 and by the Polish Ministry of Science and Higher Education under AGH University of Science and Technology Grant 11.11.230.124 (statutory project).

REFERENCES

[1] J. Hansen, T.-M. Grønli, and G. Ghinea, "Towards cloud to device push messaging on android: Technologies, possibilities and challenges," *International Journal of Communications, Network and System Sciences*, vol. 5, no. 12, pp. 839–849, 2012. doi: 10.4236/ijcns.2012.512089

[2] P. Nawrocki and M. Soboń, "Public cloud computing for software as a service platforms," *Computer Science*, vol. 15, no. 1, 2014. doi: 10.7494/csci.2014.15.1.89. [Online]. Available: http://journals.agh.edu.pl/csci/article/view/519

[3] K. Tang, Y. Wang, H. Liu, Y. Sheng, X. Wang, and Z. Wei, "Design and implementation of push notification system based on the MQTT protocol," in *2013 International Conference on Information Science and Computer Applications (ISCA 2013)*. Atlantis Press, 2013. doi: 10.2991/isca-13.2013.20

[4] D. Schuster, I. Koren, T. Springer, D. Hering, B. Söllner, M. Endler, and A. Schill, *Creating Applications for Real-Time Collaboration with XMPP and Android on Mobile Devices*. IGI Global: Handbook of Research on Mobile Software Engineering: Design, Implementation and Emergent Applications, 2012.

[5] J. Kosinski, P. Nawrocki, D. Radziszowski, K. Zielinski, S. Zielinski, G. Przybylski, and P. Wnek, "SLA monitoring and management framework for telecommunication services," in *Networking and Services, 2008. ICNS 2008. Fourth International Conference on*, March 2008. doi: 10.1109/ICNS.2008.31 pp. 170–175.

[6] T. Szydlo, P. Nawrocki, R. Brzoza-Woch, and K. Zielinski, "Power aware MOM for telemetry-oriented applications using GPRS-enabled embedded devices—levee monitoring use case," in *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, ser. Annals of Computer Science and Information Systems, M. P. M. Ganzha, L. Maciaszek, Ed., vol. 2. IEEE, 2014. doi: 10.15439/2014F252 pp. pages 1059–1064. [Online]. Available: http://dx.doi.org/10.15439/2014F252

[7] R. Brzoza-Woch, M. Konieczny, B. Kwolek, P. Nawrocki, T. Szydło, and K. Zieliński, "Holistic approach to urgent computing for flood decision support," *Procedia Computer Science*, vol. 51, no. 0, pp. 2387 – 2396, 2015. doi: 10.1016/j.procs.2015.05.414 International Conference On Computational Science, ICCS 2015 Computational Science at the Gates of Nature. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915012223

[8] R. Oldenburg, "Keeping google cloud messaging for android working reliably [technical post]," http://blog.pushbullet.com/2014/02/12/keeping-google-cloud-messaging-for-android-working-reliably-techincal-post, February 2014.

[9] C. Duckett, "Android malware utilising google cloud messaging service," http://www.zdnet.com/android-malware-utilising-google-cloud-messaging-service-7000019427/, August 2013.

[10] G. Ghinamo, F. Vadala, C. Corbi, P. Bettassa, F. Risso, and R. Sisto, "Vehicle navigation service based on real-time traffic information: A RESTful netAPI solution with long polling notification," in *Ubiquitous Positioning, Indoor Navigation, and Location Based Service (UPINLBS), 2012*, Oct 2012. doi: 10.1109/UPINLBS.2012.6409749 pp. 1–8.

[11] K. Wylie, "REST requires asynchronous notification," http://kirkwylie.blogspot.com/2008/12/rest-requires-asynchronous-notification.html, December 2008.

[12] D. Wood and D. Robson, "Message broker technology for flexible signalling control," in *Proc. ASPECT 2012 Conference*, 2012.

[13] M. Richards, "Understanding the difference between AMQP and JMS," *NFJS Magazine*, May 2011.

[14] M. Rostański, K. Grochla, and A. Seman, "Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ," in *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, ser. Annals of Computer Science and Information Systems, M. P. M. Ganzha, L. Maciaszek, Ed., vol. 2. IEEE, 2014. doi: 10.15439/2014F48 pp. pages 879–884. [Online]. Available: http://dx.doi.org/10.15439/2014F48

[15] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES/ISSS '10. New York, NY, USA: ACM, 2010. doi: 10.1145/1878961.1878982. ISBN 978-1-60558-905-3 pp. 105–114. [Online]. Available: http://doi.acm.org/10.1145/1878961.1878982