2015 IEEE Conference on Collaboration and Internet Computing

Collaboration & Mobile Cloud-Computing

Using CoAP to enable resource-sharing between clouds of mobile devices

Nan Chen, Xiaodan Li
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
{nac856,xil623}@usask.ca

Ralph Deters
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
deters@cs.usask.ca

Abstract—**As the number of mobile devices per user increases, the need to connect/combine them grows. Current approaches focus on the use of cloud-hosted backend services which allow file and app-state synchronization but fail in providing true resource sharing among mobile devices. To enable true resource/service sharing, the mobile devices of a single user should be combined into a cloud of cooperating mobile devices. Instead of accessing the resources/services of an individual device, a user should be able to seamlessly access the combined resources/services of his/her device cloud. Enabling seamless access to the resources/services hosted on different mobile devices is therefore a key challenge. Exposing the resources/services of each mobile devices within the user's device cloud via RESTful micro-services, is one possible approach. This paper focusses on the use of the IoT protocol CoAP as an application layer protocol. To minimize the energy costs of communication, it was necessary to replace CoAP's standard transport protocol (UDP) with BLE 4.1. This paper presents the performance of the CoAP protocol using BLE 4.1 on Android Lollipop.**

CoAP, Mobile Cloud-Computing, Device Cloud, Bluetooth Low Energy.

I. Introduction

Baccue [1] introduced MCC in 2009 the idea of using cloud-hosted components as a means to overcome the resource-constraints of mobile devices. However, as users nowadays tend to own *multiple* mobile devices (device cloud) it becomes possible to compensate the resource constraints of one device by combining it with other devices in the user's device cloud. This in turn raises the question of how to enable apps to access/interact with the resources in the device cloud. One way of enabling seamless access to the combined resources of multiple mobile hosts is the use of web services (WS). If each mobile host exposes the resources/services as WS it becomes possible to access them in the mobile device cloud. Within the context of user-owned device clouds, two basic scenarios can be identified, single cloud and multi-cloud.

In the single-cloud scenario, one user engages a cloud of mobile devices that are owned/managed by her/him. Enabling apps to "horizontally scale" across multiple mobile devices by allowing them to access all available hardware and software resources within the cloud of mobile devices is the key concern. Please note that the issues and possible approaches for dealing with this scenario are discussed in [2].

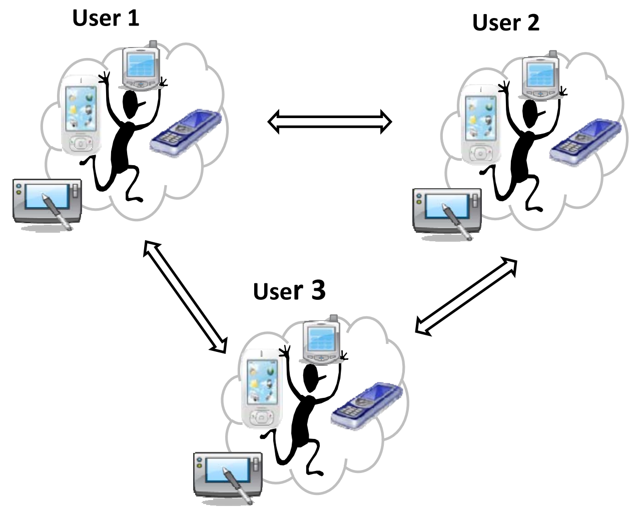

Figure 1. N clouds of mobile devices cooperating

The multi-cloud scenario focusses on people engaged in collaborative activities during work and/or social settings. Here the question is how to share resources/services across multiple clouds of mobile devices in a seamless and standards compliant way. Figure 1 shows a multi-cloud scenario in which N users with their clouds of mobile devices collaborate e.g. share their resources/services. In this scenario it is assumed that no backend connection to cloud services exists and that the users are within close proximity and have agreed to share.

This paper focusses on how to enable clouds of mobile devices to share their resources/services in a seamless manner via the use of the REST design pattern and the IoT protocol CoAP [3]. The remainder of the paper is structured as follows. Section 2 discusses general approaches for resource-sharing within mobile computing. This is followed by a brief description of CoAP in section 3. Section 4 focusses on the evaluation of using BLE 4.1. in Android Lollipop under various settings. The paper concludes with a summary and outlook.

II. Resource Sharing in mobile Computing

Sharing resources has been studied extensively in the context of mobile computing and the following sections provides only a brief overview.

A. *Unified Platform*

One possible model for seamless resource/service access the use of a unifying platform. Combining mobile devices into a homogeneous compute platform e.g. via MapReduce [4,5] and/or linking/integrating mobile devices into existing distributed platforms [6, 7] has been suggested. However, the costs of providing/managing such platforms leads to significant computation and communication overhead resulting in increased power consumption. Consequently the uptake of these concepts has been limited.

B. *Offloading Mobile Devices*

An alternative to creating a unified platform is the *offloading* of VMs and/or components onto other devices. Offloading is focused on moving loads from a "weak" mobile device onto other "stronger" devices e.g. servers. Offloading is a well-studied approach within the context of apps on mobile devices. A common approach that has its roots in the mobile agent community is the sharing/moving of part or parts of the *VM*. Cloudlets [8], ISR [9], Horatio [10], MAUI [11] and Clone Cloud [12] are all based on the concept of offloading mobile devices by migrating part or parts of their virtual machines to a remote host.

C. *Services Ecology*

The Digital Ecosystem (DE) paradigm [13] is a holistic management/design/integration model. A DE consists of proactive self-managing autonomous entities that can interact among themselves and with the environment in which they are situated. Like its biological counterpart, a DE is highly decentralized and due to its self-management it is able to evolve [14] over time as its members adapt to changes.

A service ecology, is a DE that is composed of WS that are distributed over multiple hosts. Among the WS paradigms, the Representational State-Transfer (REST) approach [15,16], seems to gain most momentum due to its low footprint and close relation to web inspired protocols. In addition REST offers a very well-grounded alternative to SOA [17,18,19] WS. In contrast to the SOA/WS* model that is based on stateless interaction and stateless services, REST is based on the concept of stateless interaction and stateful resources.

Interaction in REST follows the HTTP request/response pattern in which each side assumes that all information is contained in the request and response. Robinson [16] identifies within his 4-level web maturity model two patterns that have become popular within REST, namely CRUD and Hypermedia. The most widely used REST pattern is CRUD [23,24] (Create Read Update Delete) approach that follows a basic data-centric style. CRUD has gained significant interest especially in the mobile and cloud-computing space, due to the easy mapping onto HTTP verbs. Create, read, update and delete can be achieved via POST, GET, PUT/PATCH and DELETE.

The second pattern that is less widespread is the use of hypermedia controls. Unlike the data-centric CRUD pattern, the hypermedia pattern focuses on embedding links into the responses of request. By offering links, the server provides the client with possible next steps and ways to obtain further information. Therefore the client drives and maintains the application state by selecting from the past and current choices presented in the server responses.

In a RESTful WS-ecology a service request is always the result of a previous communication context. Each digital entity starts interacting by engaging a root resource which will provide it with a response and links. This means that a RESTful WS-ecology deals with interaction state by providing the requester with all possible next actions. In REST the future state-transitions of a client are therefore driven by the sum of all server responses. Links provided in the response enable to search/browse within a context. Consequently there is no need for a centralized service/resource finder since REST allows for a decentralized approach. A particularly interesting aspect of this decentralized discovery it that it allows for a more agile and dynamic system, since each resource can be changed at any time (e.g. adding new functionality or changing existing functionality). In addition the links in the responses can change in each interaction. Given these advantages it comes as no surprise that REST is the standard in mobile cloud-computing and therefore an ideal platform for enabling seamless access to resources distributed over multiple hosts. Obviously the RESTful WS have to be hosted by servers. One possible approach is to use micro services [25]. A micro service (MS) in the context of mobile hosts can be defined as a server component of an app or one of its modules that exposes the app or module specific resources/services. The micro service (MS) model is based on the assumption that instead of a few "big" services a larger set of exchangeable services will provide better interoperability, manageability and performance.

III. CoAP

An efficient and reliable communication protocol is important to micro services that expose RESTful WS. The CoAP (Constrained Application Protocol) [25,3] is a suitable protocol for micro services hosted on mobile devices since it was originally designed for IoT scenarios. CoAP (RFC7252) was initially proposed by ARM and the Universitaet Bremen TZI in June 2014. It is a specialized transfer protocol for machine-to-machine communication and optimized for use with constrained nodes and constrained networks. CoAP specifications use the UDP protocol as a transport protocol. While there are no real constraints on the size of CoAP messages/packets it is assumed that each CoAP message is contained in a single UDP packet (otherwise it has to be spread over N UDP packets). The CoAP package size varies from the minimum 4 bytes (simple GET requests) to a maximum of 1024 bytes.

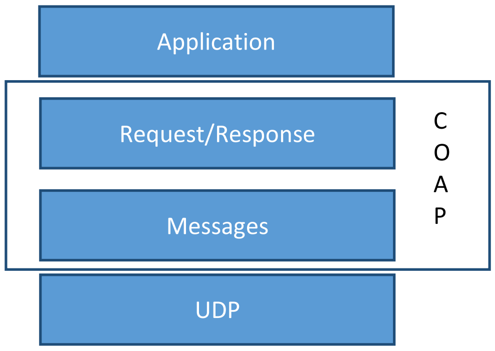

Figure 2. Abstract Layering of CoAP [3]

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Token (if any, TKL bytes) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3. CoAP Message Format [3]

CoAP follows the HTTP request/response interaction model between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types.

IV. COMMUNICATION WITHIN THE DEVICE-CLOUD

Given that CoAP is a good fit for a device-cloud consisting of cooperative self-managing micro-services the question arises of how to conduct the communication. Since the use of CoAP over WiFI has already been examined [2] we were interested in the use of low-range ad-hoc networks. We therefore decided to investigate the use of BLE 4.1 as a transport layer for CoAP. Nexus 9 tablets were used in these BLE 4.1 experiments. Since the Android implementation of BLE does not support IP we used the standard BLE protocol messages and embedded CoAP packets (please note that the default frame size 20 bytes is used) . Since we only send 20 byte BLE messages and needed 4 bytes for management/routing information of the CoAP packets we had to limit the max. size of CoAP packets to 16 bytes. If CoAP packets are larger than 16 bytes, we have to distribute them over multiple BLE packets.

A. Test Setup

In the following test two Nexus 9 devices are used. The devices that send the request is the peripheral device and the receiver is the master. Once the master receives a notification it pulls the data from the peripheral device.
The peripheral device runs a load generator app that consist of 2 threads. The app itself is written in C# and compiled into a native Android app using the cross-platform tool Xamarin. The producer thread creates CoAP messages and places them into a sending queue. The sender thread takes the message from the sending queue and calls the Android BLE sending routine. Two strategies for sending were implemented (sending-first & balanced). In the *sending-first* strategy the sender thread blocks the queue until all messages are sent. Obviously the producer thread is forced to wait since the queue is guarded by a lock.
In the *balanced* strategy the sender blocks the queue, dequeues a single message, releases the lock and sends the message. Obviously this approach is more balanced and blocks the producer thread less.
In each test 2 Nexus 9 tablets are used and each time 100 CoAP message are sent. Their arrival times are logged by the receiver device (master).

B. Simple CoAP messages

The simplest CoAP message is the 4 byte CoAP GET message. To test the use of BLE 4.1 as a transport protocol for CoAP we conducted a series of experiments. In each experiment we send 100 CoAP messages of 4 bytes with a delay of 0, 50,100,150, 200, 250,300,350,400,450 ms.

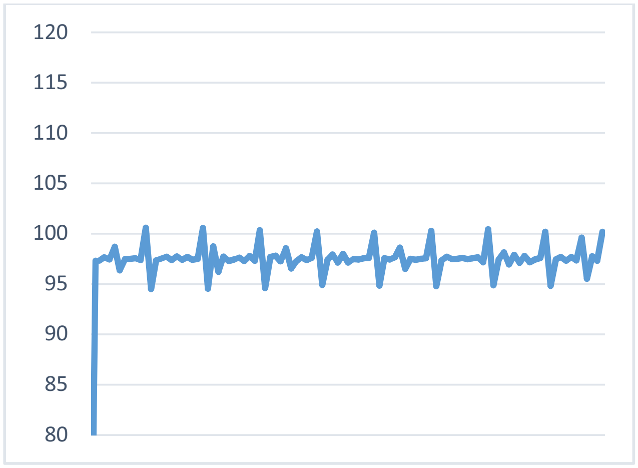

Figure 4. 4 byte CoAP message no delay

Figure 4 shows the results of sending 100 4 byte CoAP GET messages with no delay. The arrival rates at the receiving tablet fluctuate around 97 ms. However, it is interesting to note that there is also a distinctive heartbeat-like pattern visible.
The pattern continues to be visible when the delay is increased (figures 5 & 6). We also see a few major spikes that can be explained with thread context switches or the

know issues with the Android Lollipop BLE implementation. As figure 7 shows, the pattern is more obvious as we gradually increase the delays in steps from 150 to 450 in 50ms steps.

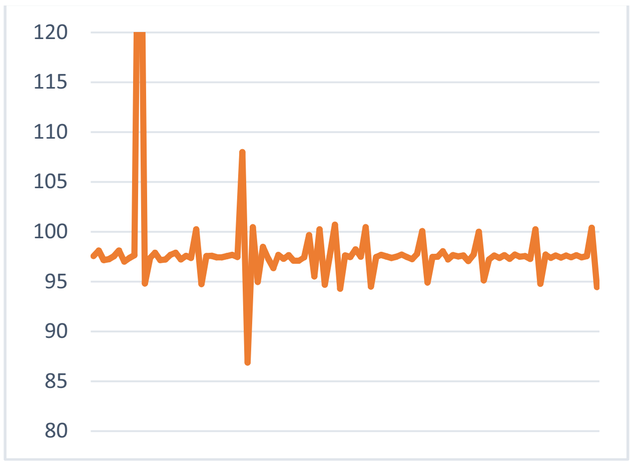

Figure 5. 4 byte CoAP message 50 ms delay

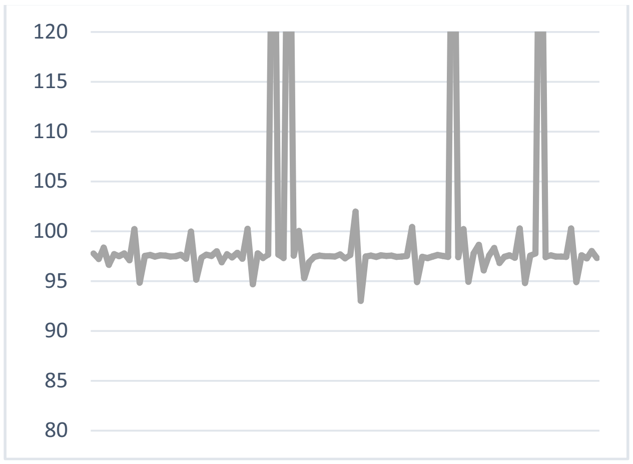

Figure 6. 4 byte CoAP message 100 ms delay

To investigate the behavior of the system further we focus on the size of the CoAP messages. As mentioned earlier, the 4 byte CoAP message is the most basic GET request. However, as headers and other information are added to the GET request the messages grows. This is also true if other HTTP verbs are used or a larger response is sent. In the following tests we used no delay and varied the messages of sizes {4,7,9,11,13,15 bytes}.

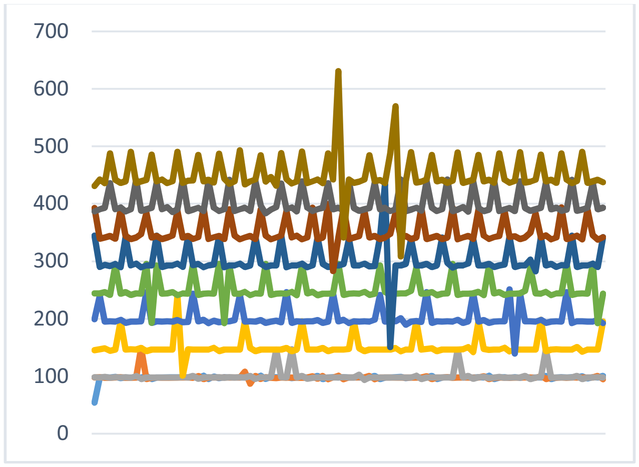

Figure 7. 4 byte CoAP mesaage with 0-450 ms

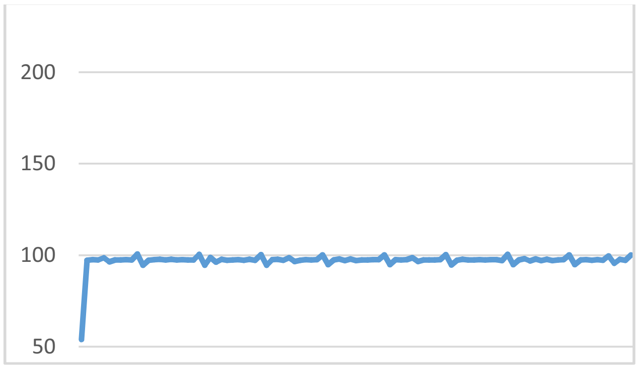

Figure 8. 4 byte CoAP mesaage with 0 ms delay

As can be seen in the figures 8-14 the increased size doesn't change the heartbeat like pattern. However as the payload increases more and/or larger chaotic spikes appear.

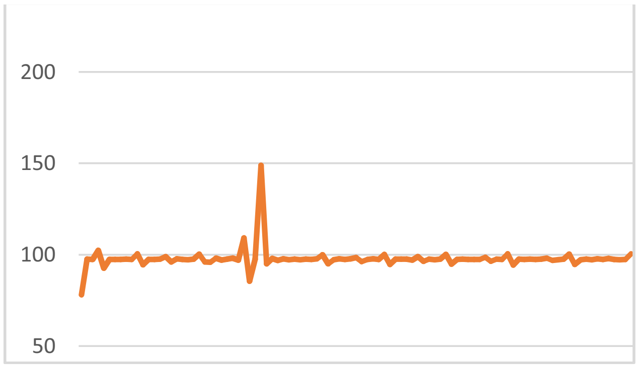

Figure 9. 7 byte CoAP mesaage with 0 ms delay

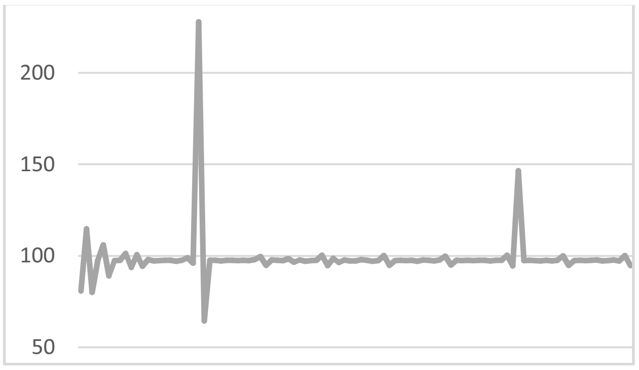

Figure 10. 9 byte CoAP mesaage with 0 ms delay

To rule out thread blocking and thread context switching we conducted tests with the balanced strategy. As can be seen in figure 15 the patterns look very similar. Repeating the test always leads to the emergence of the characteristic pattern. While there are many reports about chaotic behaviors with the BLE layer of Android Lollipop the pattern must have a different reason.

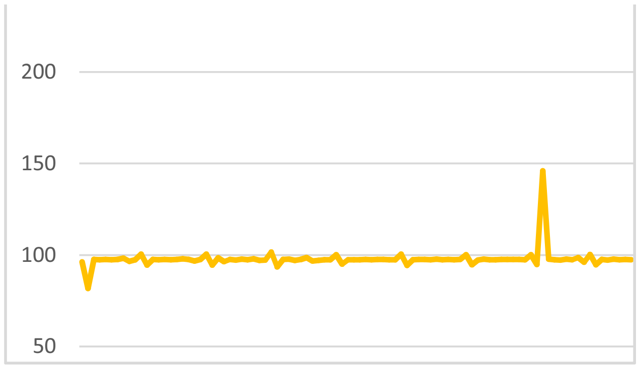

Figure 11. 11 byte CoAP message with 0 ms delay

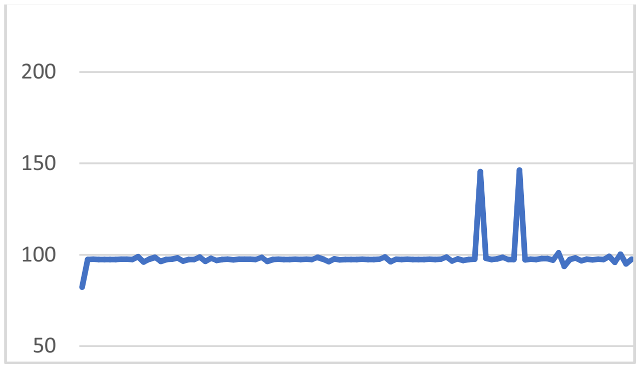

Figure 12. 13 byte CoAP message with 0 ms delay

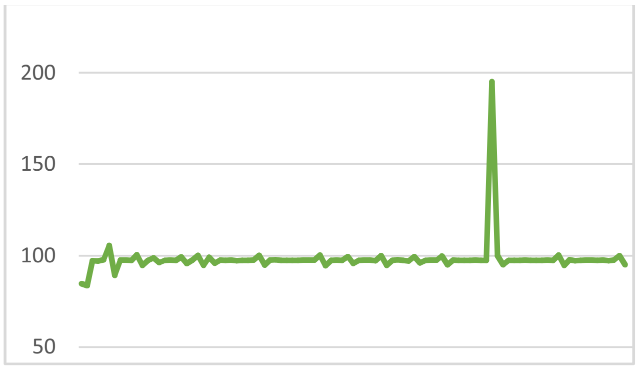

Figure 13. 15 byte CoAP mesaage with 0 ms delay

Examining the pattern in different message delay settings shows that the peaks get closer as the delay is increased e.g. ca. 4 messages apart at 450 ms delay and 11 messages apart at 0 - 100 ms delay. However the spikes from 0-100 ms. delays and 150-450 ms. delays are different in terms of their height. Spikes at delays from 0-100 ms are only 3-4 ms above the normal arrival rate of the corresponding series. Spikes between 150-450 ms all tend to be ca. 40 ms above the normal arrival rate.

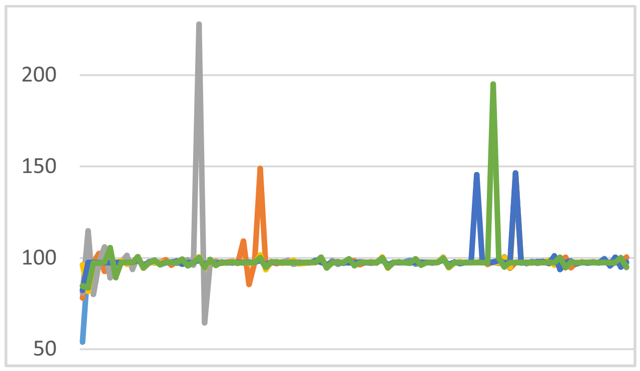

Figure 14. {4,7,9,11,13,15} byte CoAP mesages with 0 ms delay

Repeating the test always leads to the emergence of the characteristic pattern. While there are many reports about chaotic behaviors with the BLE layer of Android Lollipop the pattern must have a different reason. Examining the pattern of different message delay settings shows that the peaks get closer as the delay is increased e.g. ca. 4 messages apart at 450 ms. delay and 11 messages apart at 0 - 100 ms delay. However the spikes from 0-100 ms. delays and 150-450 ms. delays are different in terms of their height. Spikes at delays from 0-100 ms. are only 3-4 ms. above the normal arrival rate of the corresponding series. Spikes between 150-450 ms. all tend to be ca. 40 ms. above the normal arrival rate.

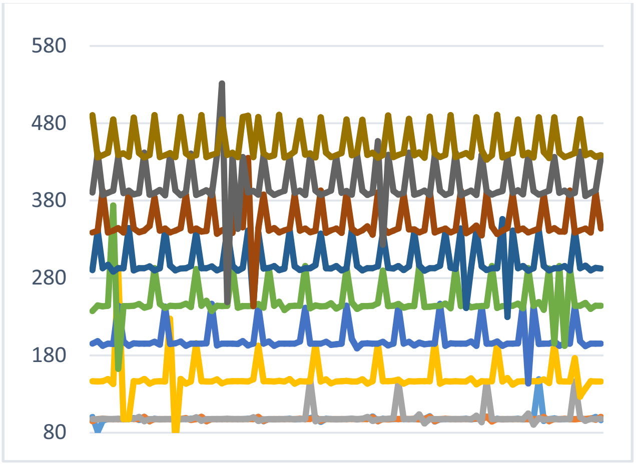

Figure 15. 4 byte CoAP mesaage with 0-450 ms (balanced strategy)

We explain the small spikes of the 0-100 ms. delay series with the connection interval of the BLE layer. The Android version/device specific BLE connection interval can be anywhere between 7.5 ms. and 4 sec. and marks the cycle in which a test for available data and the transmission is performed. If data arrives after the check-phase in a connection interval it must wait for a new cycle. However the more or less constant 40ms spikes must have a different reason. Studies on BLE's energy saving behavior [26] show that sleep or lower energy cycles are a standard approach for BLE to preserve energy. Since the 40 ms. spikes tend to appear only in series with delays longer than 100 ms. we assume that the BLE layer detects connection interval cycles with no data transmission. To save energy it begins to power down ca. every 1.5 secs. if such cycles are detected. Once messages need to be sent it takes ca. 40 ms. to reestablish transmission.

V. Conclusions & Future Work

This paper focusses on enabling seamless access to resources that are hosted on mobile host using the IoT protocol CoAP. We present the use of BLE 4.1 as a transport layer protocol for CoAP's use on mobile devise and evaluate its performance. The test system indicates that it takes an average 97 ms. to transmit 4-15 byte CoAP messages. While this is significantly slower than comparable tests in WiFi settings [2] it is still acceptable for many scenarios.

Our future work will focus on improving the performance and on the evaluation of communication with more devices in different settings. In addition we plan to explore the impact of encryption in the communication.

References

[1] M. Beccue, ABIresearch Report RR-MCC: "Mobile Cloud Computing", 64 pages, 2009.

[2] S. Xue, R. Deters: "Resource sharing in mobile cloud-computing with CoAP", 6th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2015), 8 pages.

[3] CoAP https://tools.ietf.org/html/rfc7252

[4] Marinelli, Eugene E. *Hyrax: cloud computing on mobile devices using MapReduce*. No. CMU-CS-09-164. Carnegie Mellon School of Computer Science, 2009.

[5] Dou, Adam, Vana Kalogeraki, Dimitrios Gunopulos, Taneli Mielikainen, and Ville H. Tuulos. "Misco: a MapReduce framework for mobile systems." In *Proceedings of the 3rd international conference on pervasive technologies related to assistive environments*, p. 32. ACM, 2010.

[6] Chu, David C., and Marty Humphrey. "Mobile ogsi. net: Grid computing on mobile devices." In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pp. 182-191. IEEE, 2004.

[7] Kotilainen, Niko, Matthieu Weber, Mikko Vapa, and Juori Vuori. "Mobile Chedar-a peer-to-peer middleware for mobile devices." In *Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops. Third IEEE International Conference on*, pp. 86-90. IEEE, 2005.

[8] Satyanarayanan, Mahadev, Paramvir Bahl, Ramón Caceres, and Nigel Davies. "The case for vm-based cloudlets in mobile computing." *Pervasive Computing, IEEE* 8, no. 4 (2009): 14-23.

[9] M. Satyanarayanan, M. A. Kozuch, C. J. Helfrich, and D. R. O. Hallaron, "Towards Seamless Mobility on Pervasive Hardware," Pervasive and Mobile Computing, vol. 1, no. 2, pp. 157–189, Jul. 2005.

[10] S. Smaldone, B. Gilbert, N. Bila, L. Iftode, E. Lara, and Mahadev Satyanarayanan, "Leveraging Smart Phones to Reduce Mobility Footprints", MobiSys '09, Kraków, Poland, June, 2009

[11] Cuervo, Eduardo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. "MAUI: making smartphones last longer with code offload." In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 49-62. ACM, 2010.

[12] Chun, Byung-Gon, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. "Clonecloud: elastic execution between mobile device and cloud." In *Proceedings of the sixth conference on Computer systems*, pp. 301-314. ACM, 2011.

[13] Ferronato, P.: Architecture for Digital Ecosystems, beyond Service Oriented Architecture, IEEE-DEST 2007.

[14] Uden, L., Damiani, E.: Activity Theory for OSS Ecosystems, IEEE-DEST 2007

[15] Fielding R.: "Architectural Styles and the Design of Network-based Software Architectures", Dissertation University of Irvine, 2000

[16] Robinson, L.: "Richardson Maturity Model"

[17] http://martinfowler.com/articles/richardsonMaturityModel.hml

[18] "Four Tenets of Service Orientation"

[19] msdn.microsoft.com/msdnmag/issues/04/01/Indigo/default.asx

[20] Chatarji, J. "Introduction to Service Oriented Architecture (SOA)".

[21] www.devshed.com/c/a/Web-Services/Introduction-to-Service-Oriented-Architecture-SOA, 5 pages. 2004.

[22] Roy W. Schulte and Yefim V. Natis Service Oriented Architecture, Gartner, 12 April 1996.

[23] CRUD: "Create Read, Update and Delete", http://en.wikipedia.org/wiki/Create,_read,_update_and_delete

[24] Namiot, Dmitry, and Manfred Sneps-Sneppe. "On micro-services architecture." *International Journal of Open Information Technologies* 2.9 (2014): 24-27.

[25] Z. Shelby, K. Hartke, and C. Bormann. The constrained application protocol (CoAP), 2014.

[26] Dementyev, Artem, et al. "Power consumption analysis of Bluetooth Low Energy, ZigBee and ANT sensor nodes in a cyclic sleep scenario." *Wireless Symposium (IWS), 2013 IEEE International*. IEEE, 2013