Combining Mobile & Fog Computing

Using CoAP to link mobile device clouds with fog computing

Heng Shi, Nan Chen
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
{heng.shi,nac856}@mail.usask.ca

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

***Abstract*—This paper focuses on the use of CoAP as a means of connecting clouds of sensors & smart devices via mobile devices with users. We present an alternative to the hierarchical view on fog-computing by enabling device clouds to interact in a P2P fashion with smart device/sensor clouds. Using the IoT protocol CoAP we expose smart device/sensors and resources in the mobile cloud in a uniform manner as RESful services.**

The paper presents an evaluation of an Erlang CoAP server and an evaluations of using BLE 4.1 for submitting CoAP messages that span between 1 – 7 BLE packets.

Keywords: 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. This is typically achieved by offloading part or parts of an application from the mobile device onto a resource-rich host. 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 [2], ISR [3], Horatio [4], MAUI [5] and Clone Cloud [6] are all based on the concept of offloading mobile devices by migrating part or parts of their virtual machines to a remote host. However as the available resources per mobile device and the number of devices per user increases it becomes possible to combine them into a single compute-platform (device cloud, see figure 1) e.g. via MapReduce [7,8] or linking/integrating mobile devices into existing distributed platforms [9, 10].

A particularly interesting aspect of device clouds is their potential role in the fog computing situated between the sensor clouds and the users (figure 2). Given the availability of wireless low energy protocols like BLE 4.1 in mobile devices and sensors it becomes possible to connect sensors with mobile devices. In case more high-end sensors like the TI CC2541 are used, direct connections between mobile devices and sensors are possible. However, sensors can also be connected with mobile devices via low-end compute nodes e.g. raspberry pi that are WIFI or BLE enabled.

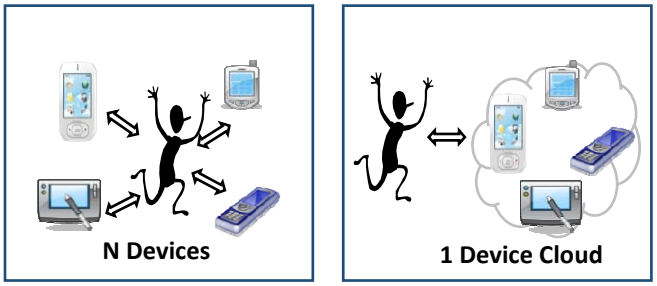

Figure 1. User with N devices versus User with decice cloud

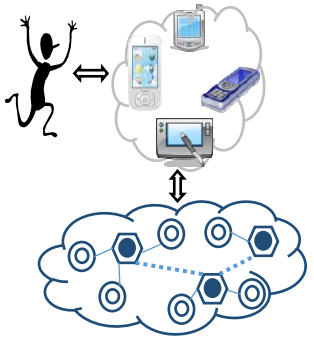

Figure 2. Connecting device cloud with sensors

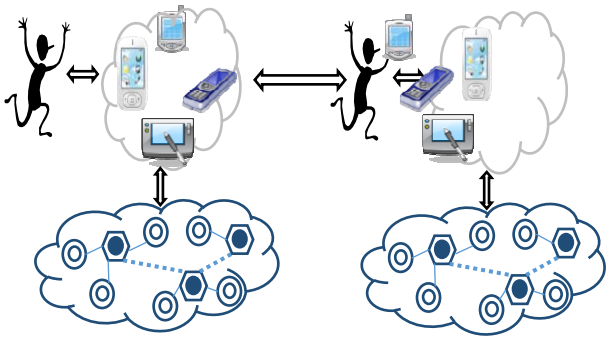

Figure 3. Connceting multiple sensor clouds via device clouds to different users

Connecting the device cloud of a user with a sensor network enables us of course to share data by simply enabling the device clouds to interact (figure 3).
This in turn offers an alternative to the existing fog-computing paradigm [11,12] that is based on a hierarchy of compute node layers with the fog as a layer between smart devices/ sensors and the cloud.

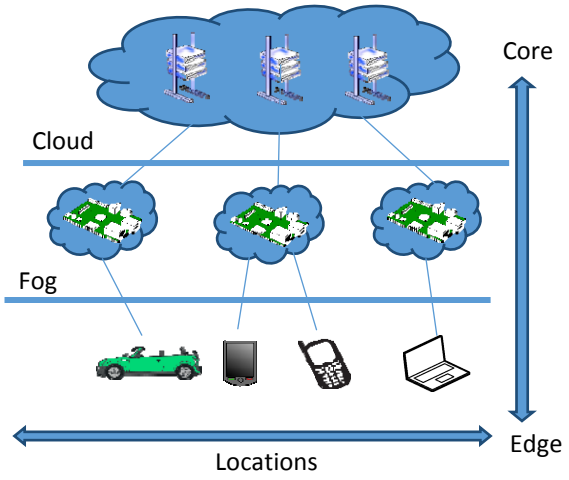

Figure 4. The fog as a layer between smart devices/sensor and the cloud [12]

Instead of this hierarchical view, we propose that the sensor /smart device cloud can be connected with mobile devices, which in turn can consume and offer resources / services. This transforms the classical view on fog computing (edge computing) into a less hierarchical P2P style approach.

This paper focusses on how to enable device clouds and sensor networks to share their resources/services in a seamless manner via the use of the REST design pattern and the IoT protocol CoAP [13]. The remainder of the paper is structured as follows. Section 2 discusses potential IoT protocols. Section 3 discusses the use of CoAP and section 4 the design and evaluation of an Erlang based CoAP server. Section 5 presents our experiments in WIFI and BLE environments. The paper concludes with a summary in section 6.

II. IoT Protocols

A key issue within the IoT space is communication between the participants e.g. sensors, effectors, compute nodes, backend services, etc. The current IoT communication protocols can be divided into 3 categories.

- Data-oriented
- Message-oriented
- Resource-oriented

A. Data Oriented IoT Communciataion

OMG's Data Distribution Service (DDS) is the most widely used example of a data-oriented IoT communication protocol. The DDS specifications states that the "the efficient and robust delivery of the right information to the right place at the right time" is its primary goal. It implements a data-centric publish-subscribe system that is designed to ensure fast and reliable delivery of data from data-sources to data-sinks. In DDS, similar to other publish-subscribe systems, the participants are decoupled in regards to time space and synchronization.

- Time:
  Participants do not need to know each other.
- Space:
  Participants do not need to be active at the same time.
- Synchronization:
  Publishers are not blocked when sending and subscribers are asynchronously notified.

The key features of DDS are:

- Discovery of participants at runtime (e.g. SDP)
- Recovery for subscribers in terms of retransmission of missed data
- Support for different QoS settings
- "Real-Time" publish subscribe
- True peer-to-peer communication of participants – no brokers
- Interoperability of DDS systems when RTPS is used as the wire protocol.

It is important to note that OMG's DDS specification is silent on the wire-protocol. This in turn has led to the problem that DDS products from different vendors face interoperability issues. To overcome these interoperability issues OMG promotes the use of RTPS [14] as a wire-protocol for DDS. RTPS itself relies on the use of UDP and multicast to deliver the data from publishers to subscribers. Depending on the network-topology and routers used, RTPS can deliver impressive throughput. However, there are known scalability [15] and performance [16] issues associated with RTPS.
While DDS inspired/compatible protocols have been proposed for wireless and sensor scenarios, there seems to be a lack of successful deployments. While the overhead costs for DDS maybe part of the problem, we believe that DDS's Global Data Space (GDS) concept is of limited use when faced with unreliable, low bandwidth and high latency networks.

B. *Message-Oriented IoT Communciataion*

Message-oriented communication protocols focus on the delivery of messages from producers to consumers. Within IoT MQTT [17] is the most popular and widely used one. MQTT follows the publish-subscribe architecture in which routers are responsible for connecting producers of messages with consumers. Like DDS, MQTT offers decoupling with respect to time, space and synchronization.
The classical MQTT deployment follows a hub-spoke model in which compute nodes are linked directly to sensors and effectors. The compute nodes are connected via brokers to the backend-services (figure 5) [18].

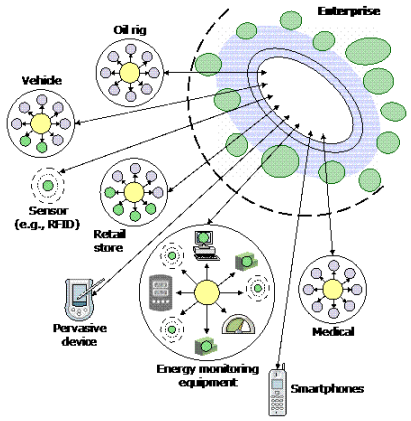

Figure 5. MQTT machine to machine example
(picture copied from IBM online article on MQTT [22])

MQTT uses TCP for communicating with the message brokers. This in turn can lead to challenges with respect to communication costs. Consequently a UDP based MQTT for sensors (MQTT-S) [19] was developed.
The main drawback from our point of view is the level of abstraction. MQTT is a message-oriented protocol which means that it is content agnostic and only focusses on the delivery of messages.

C. *Resource-Oriented IoT Communciataion*

Since sensors and effectors tend to be resource-constrained devices with limited processing power, they are often connected to compute nodes that do basic preprocessing of data. Such a compute node is also an excellent host for a server that exposes sensors and preprocessed data in a RESTful manner [20,21]. 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 [22,23] (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.
Obviously the RESTful WS have to be hosted by servers. One possible approach is to use micro services [24]. 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. Micro services can be used on the compute nodes and on the mobile devices.

A particularly interesting aspect of the resource-oriented communication is the natural emergence of edge/fog-computing. Since the resources need to process the requests, they must have some basic processing capabilities. This in turn allow us to distribute the processing load and to reduce the load on backend-services. In addition, since the requests have a clear read-write semantic, it becomes possible to add infrastructure support in form of caching and reverse proxies thus allowing for more distribution of load and network traffic. Consequently we feel that the resource-oriented approach is best suited for linking clouds of mobile devices with clouds of sensors.

III. IMPLEMENTING MICROSERVICES WITH COAP

An efficient and reliable communication protocol is important to micro services (MS) that expose RESTful WS. The CoAP (Constrained Application Protocol) [25,26] 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 UDP 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.
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.

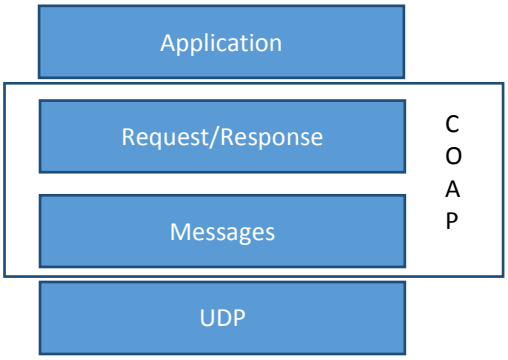

Figure 6. Abstract Layering of CoAP [13]

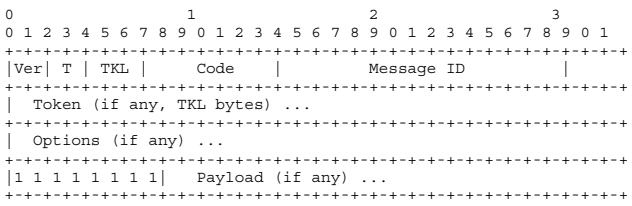

Figure 7. CoAP Message Format [13]

IV. CoAP Server

Since compute nodes in a sensor network should be low-cost nodes, we decided to explore the use of single board computers like Raspberry Pi. The Raspberry Pi can be easily configured to connect to various sensors using a variety of short-range & low-energy communication protocols. In addition the Raspberry Pi is powerful enough to perform basic pre-processing of sensor data and to host a server (ideal edge computing device). Since we were not satisfied with the performance and fault-tolerance of existing CoAP servers in C, Python and Java for the Raspberry Pi we developed one based on Erlang. Erlang was chosen for its fault-tolerance, soft-real-time and high throughput.

In our implementation (figure 8), a root supervisor process is responsible for supervising the application. PID (Process Identifier) <0.35.0> and <0.36.0> are application master processes. *coap_srv_sup* is the application root supervisor process. *coap_gen_server* is the process responsible for receiving new CoAP messages (as UDP messages). *coap_worker_sup* is the supervisor process of all worker processes. PID <0.45.0> to <0.54.0> are worker processes responsible for handling requests and sending back replies (10 workers in the figure). The CoAP server uses a pool of worker processes to handle requests in order to achieve concurrency. When the server boots up, a predefined number of workers are started in advance.

We evaluated the performance of the CoAP server in terms of throughput and latency. The experimental environment consists of one client machine and a single server. The server machine is a Raspberry Pi 2 Model B with a 900MHz quad-core ARM Cortex-A7 CPU and 1GB LPDDR2 SDRAM. Raspbian OS with Linux kernel version 3.18.7-v7+ and Erlang 17.3 is used as the running environment of the Erlang CoAP server. The server is connected to the client machine over a dedicated 10/100 desktop Ethernet switch (figure 9).

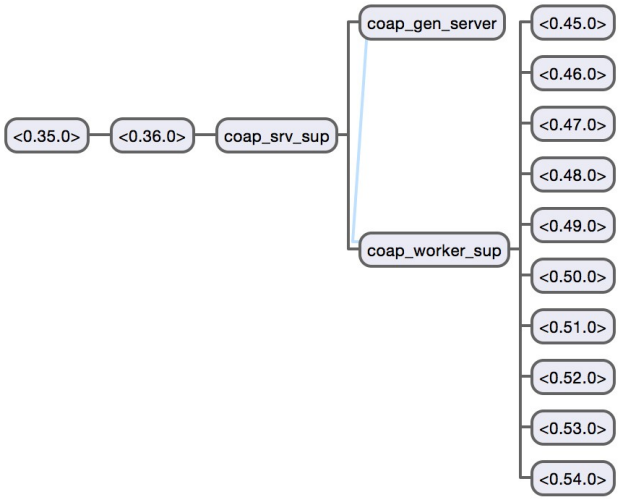

Figure 8. Supervison Tree in CoAP server

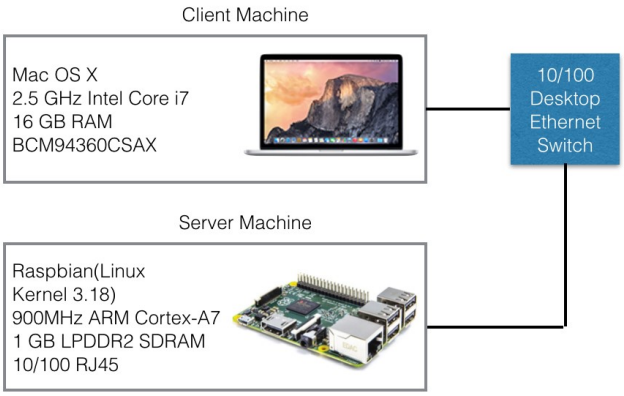

Figure 9. Evalauation of CoAP server

To test the server throughput, we used virtual clients, each sending requests to the server concurrently. Each virtual

client opens a separate port and sends requests for 30 seconds and records the responses. The sum of responses of all virtual clients divided by the time period of 30 seconds to determine the average throughput that the server is able to achieve.
UDP packets could get lost and a virtual client does not receive a response. In this case after 10 seconds with no response, the virtual client considers the request as failed and sends a new one.

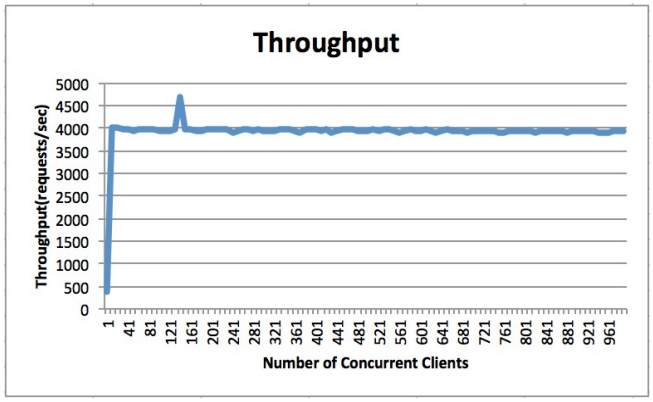

Figure 10. Throughput of CoAP server

All GET requests are confirmable and the responses are piggy-backed ACKs. A worker pool with 10 worker processes is used on the server side. The server is evaluated with different numbers of virtual clients. The test starts with 1 concurrent virtual client and increases up to 1000. Figure 10 shows the average number of requests that the CoAP server is able to process per second. When we reach 10 concurrent virtual clients, the performance curve of the server stabilizes at round 4000 requests per second. And the number of requests which eventually time out was below a hundredth of the amount of successful exchanges. The peak that appears is most likely caused by other Linux tasks running and thus disturbing the Erlang VM. The throughput curve remains flat under a large number of concurrent clients because Erlang's scheduler minimizes non voluntary context switches. Please note that the excellent performance of the server is a direct result of using Erlang which in turn indicates its usefulness within IoT settings.

To test the round trip time, asynchronous and synchronous virtual clients are used. With synchronous virtual clients, each client opens a separate port, sends a request and waits for the reply, and then sends another request and waits. With asynchronous virtual clients, each client opens a separate port, sends a request, does not wait and sends next request. In order to give enough buffer space for UDP, asynchronous virtual clients are set to sending one request every 3 milliseconds. If a virtual client does not receive response in 10 seconds, it marks the request as failed and does not count the round trip time. Virtual clients send the same GET request as in the server throughput test.

In the synchronous tests, the test starts with 0 workload clients (only one recording virtual client sending requests) and increases it up to 200, while in asynchronous test, the test starts with 0 workload clients and increases it up to 100, as showed in figure 11 and 12.

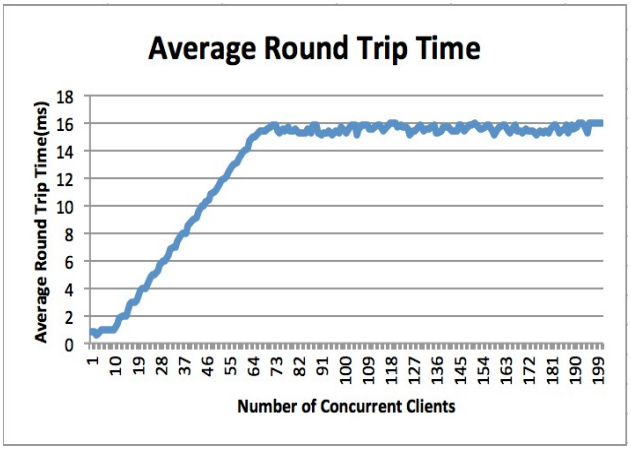

Figure 11. Average Round-Trip-Time with synchronous virtual clients

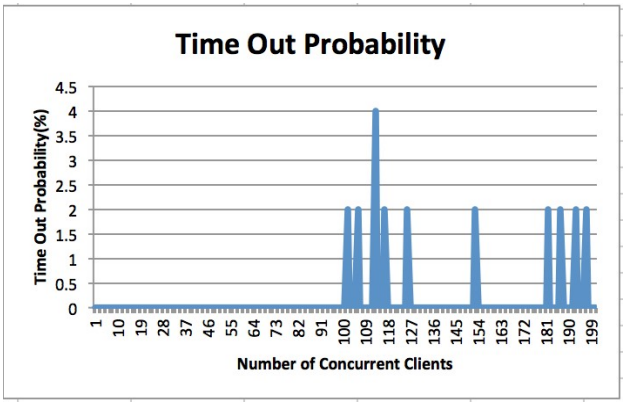

Figure 12. Time-Out Probabbility with synchronous virtual clients

In the synchronous test, average round trip time stabilizes at around 16 milliseconds after 73 concurrent virtual clients were added. After 100 clients the server was still achieving a time out probability < 4%. In the asynchronous test, the average round trip stabilizes at around 12 to 16 milliseconds, but time out probability increases rapidly after 18 concurrent virtual clients were added. This is because asynchronous clients send requests faster than synchronous clients; the server faces a higher load. When 96 virtual

clients were added, time out probability approaches nearly 50%.

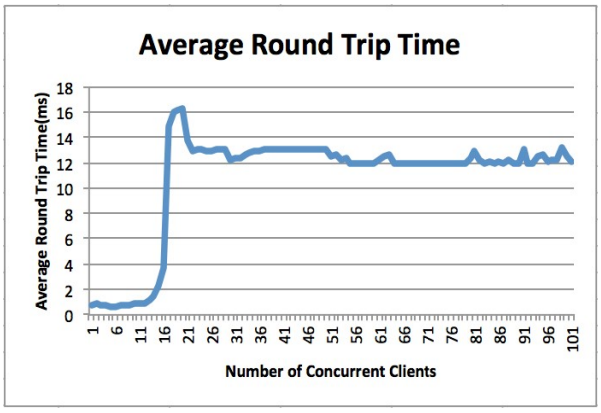

Figure 13. Average Round-Trip-Time with assynchronous virtual clients

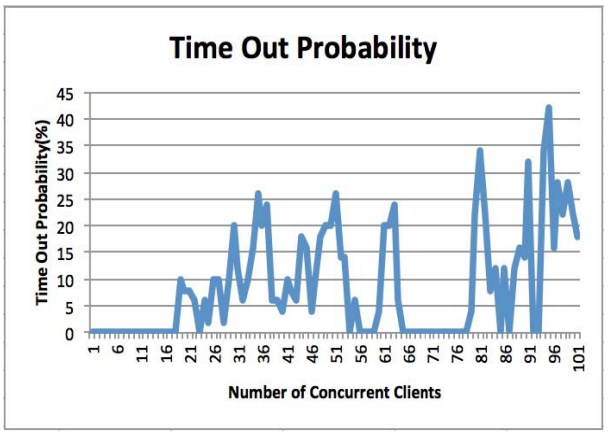

Figure 14. Time-Out Probability with assynchronous virtual clients

V. COMMUNICATION OF WIFI & BLE

Given that CoAP is a good fit for cooperative self-managing micro-services the question arises of how to conduct the communication. Our first tests [27] centered on the use of CoAP in the context of WIFI. We used a dedicated network with a TP-Link TL-WR841N wireless router (802.11b/g/n) as the access point and a public Starbucks wireless network. Figure 15 shows that there are only minor differences in the data transfer response time with respect to payload size between the two network scenarios.

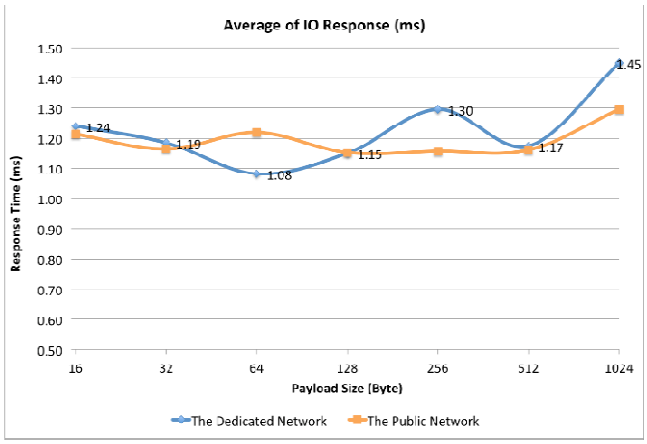

Figure 15. Average IO Data Transfer Response Time [27]

To examine the use of low-energy protocols as an alternative for constrained devices, we decided to evaluate BLE 4.1. Our tests [28] were conducted in a variety of settings with respect to payloads, and arrival rates. Since BLE 4.1 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.

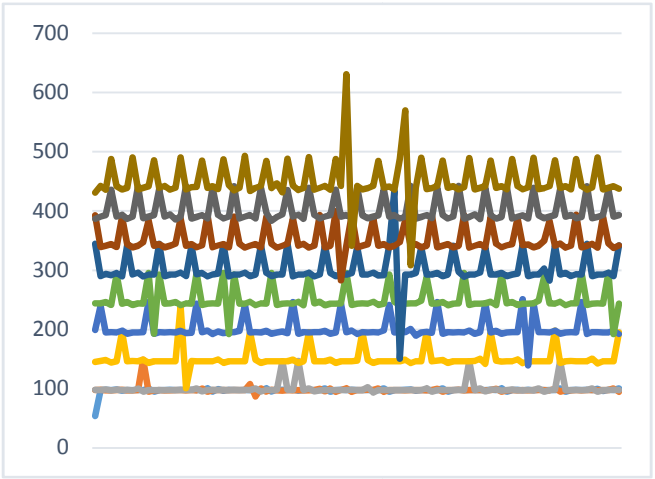

Figure 16. 4 byte CoAP mesaage with 0-450 ms delays [28]

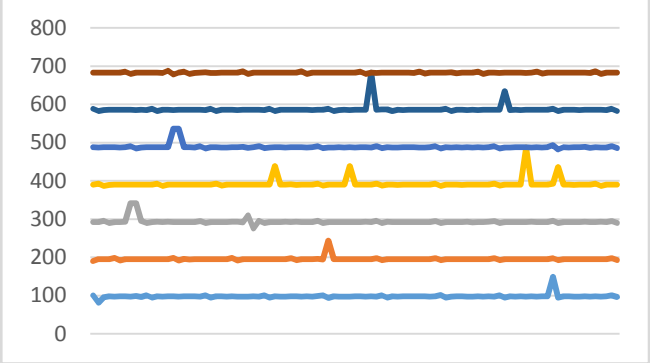

Figure 17. CoAP messages that span muliple BLE packets

As can be seen in figure 16, adding delays between the emissions of CoAP messages doesn't result in major changes. The only visible sign of increased delays (reduced arrival rates) is the appearance of periodic spikes. The small spikes of the 0-100 ms. delay can be explained 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 [29] 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.

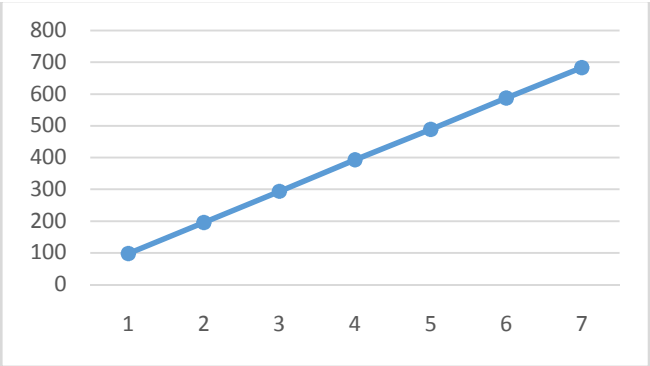

Figure 18. Relationship of packets & time

Since our previous experiments [23] focus only on CoAP messages that fit within a single BLE packet we tested the impact of sending larger CoAP messages that have to be spread over multiple packets. Figure 17 shows the results of sending CoAP messages that span 1,2,3,4,5,6,7 packets with no delay. As expected there is a linear relationship (figure 18) between the number of packets that have to be sent and the time it takes e.g. sending a CoAP messages that needs 2 BLE packets takes twice the time.

VI. Conclusions & Future Work

This paper focusses on the use of device clouds as a means to access smart devices/sensors. Unlike classical fog computing approaches that assume a hierarchy of compute nodes with a strong emphasis on edge computing, we present a non-hierarchical, P2P inspired view on fog computing. Using mobile device clouds, it becomes possible to support compute nodes and smart devices/sensors with computational resources and to distribute computation and share resources among mobile devices. In addition it allows for a decentral dissemination of data between users. While we believe that this fog computing approach is particularly well suited for smart cities in which citizens access sensors and resources in their environment and contribute by sharing computational costs, it is by no means limited to that scenario. We present an approach for enabling seamless access between smart devices and sensors and mobile device resources using the IoT protocol CoAP. We present the use of BLE 4.1 as a transport layer protocol for CoAP's use in mobile device clouds and report on the performance of an Erlang-based CoAP server.

References

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

[2] 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.

[3] 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.

[4] 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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing* (MCC '12). ACM, New York, NY, USA, 13-16.

[12] Stojmenovic, Ivan, and Sheng Wen. "The Fog computing paradigm: Scenarios and security issues." *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*. IEEE, 2014.

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

[14] http://www.omg.org/spec/DDSI-RTPS/2.2/PDF/

[15] Esposito, Christian. "Data distribution service (DDS) limitations for data dissemination wrt large-scale complex critical infrastructures (LCCI)." *MobiLab, Università degli Studi di Napoli Federico II, Napoli, Italy, Tech. Rep* (2011).

[16] Sanchez-Monedero, Javier, et al. "Bloom filter-based discovery protocol for DDS middleware." *Journal of Parallel and Distributed Computing* 71.10 (2011): 1305-1317.

[17] http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf

[18] https://www.ibm.com/developerworks/community/blogs/aimsupport/entry/what_is_mqtt_and_how_does_it_work_with_websphere_mq

[19] http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf

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

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

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

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

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

[27] 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.

[28] N. Chen, H. Li, R. Deters, "Collaboration & Mobile Cloud Computing", IEEE CIC 2015, 6 pages.

[29] 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