Enabling Sensor Data Exchanges in Unstable Mobile Architectures

Steven Xue, Richard K. Lomotey and Ralph Deters
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada

steven.xue@usask.ca, richard.lomotey@usask.ca and deters@cs.usask.ca

***Abstract*—There is increasing interest from companies on the use of emerging mobile and cloud computing technologies to achieve enterprise specific tasks. Mobile devices such as smartphones, tablets, and notebooks can be used to access personalized and enterprise data. Moreover, recent launch of consumer devices such as wearable smart watches (e.g., Apple Watch) requires services and products re-alignment. More importantly, sensor capabilities of these mobile devices can be leveraged with desktops and cloud computing facilities to create next generation mobile applications that can be beneficial to enterprises such as health, agriculture, mining and so on. The challenge however is the limited and unstable wireless mediums through which these smart devices communicate. Due to this factor, data transfer can experience challenges such as high latency, data corruption, communication loss, and performance cost in terms of poor throughput. Hence, this work designed a prototypic system using the Constrained Application Protocol (CoAP) that seeks to provide speed-efficient data exchanges in a mobile-sensor-cloud ecosystem. Furthermore, a group of performance measurement experiments and conducted to understand the proposed system's performance for wireless networks. Preliminary results suggest CoAP can achieve quick I/O data response time.**

Keywords-CoAP; Mobile; Data Synchronization; Cloud Computing; Sensor, Wireless Network

I. INTRODUCTION

The use of mobile devices such as smartphones, tablets, and wearable devices is on the increase. This is because most of these devices can be employed to consume enterprise and personalized services and applications. As a result, mobile computing has become a reality as a merger of two technologies: the appearance of powerful portable computers and the development of fast reliable networks [22]. According to a recent report from Cisco [5] regarding 2013 mobile sales, the number of smartphones increased by 77%, average smartphone usage grew 50%, global mobile data traffic grew 81%, and, on an average, a smart device generated 29 times more traffic than a non-smart device. These staggering numbers will be shattered even in the coming years.

Another trend in recent years is cloud computing. Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services [3]. Cloud computing brings new opportunities to consume services in a reliable, flexible, energy-efficient and less expensive way. Since those services are provided over the Internet, the service is not necessarily only available in a local or a small geographical area. You can access application services in Google data center in US, hosting services on Amazon EC2 platform in Europe or storage service on Apple cloud in China from anywhere in the world. As we enjoy the entertainment, simplicity, flexibility and efficiency that cloud computing brings to us, we don't really care where those service are located any more. The computation resource used for providing such services are much wider distributed indeed [13].

As these emerging cloud and mobile computing services are commonly used in our daily lives and replace our traditional computation model, gradually the computation resources which used to be hosted locally have been well distributed over networks now. Although distributed resources and services create many problems such as challenges on resources management, data synchronization, collaboration, and security, it also brings new opportunities to create next generation applications. Mobile devices now have many sensors pre-installed on them from production. Examples of sensors include Accelerometer, Gravity, Magnetic Field, Pressure, and Rotation Vector. However those sensors are not available on our desktop or laptop computers. If we make the sensor data available on any other devices that do not have such data, it will create new possibilities and applications that make our life even convenient, efficient and economical. New applications such as location-aware application on a desktop can have GPS information from mobile a device, using mobile device as a remote keyboard/mice to control desktop or TV and screen sharing among family members or friends. Sensor generates data in very different characteristics. It can generate thousands of data in a second and the data generated can be very small in size.

What is the best way to transfer sensor data over networks becomes the next logical question. Sending the data as quickly as possible, for example at every single sensor data generated, will give us a good response time but it also kills our network throughput and transport protocol might not be able to efficiently handle that. If we lower sending rate with bigger packet, it will result in better network throughput but get worse response time which might not satisfy application and its usability requirements. Due to this nature of sensor data, it is very interesting to understand their network I/O performance and best practice.

978-1-4673-7284-8/15 $31.00 © 2015 IEEE
DOI 10.1109/MS.2015.60

To understand its performance can help us in the following ways: 1) identify challenges of such new applications, 2) design a better policy on data transferring over networks and better application architecture to satisfy throughput and response time requirements.

Thus the goal of this work is to explore the best approach that can aid a distributed mobile computing environment exchange sensor data reliably. The work proposes the adoption of the Constrained Application Protocol (CoAP) system ahead of the other exiting protocols in terms of how the sensors exchange data. Through a set of experiments designed for wireless network using CoAP, we realized that the system can achieve quick I/O data response time under 1.5ms regardless of the payload size. The maximum throughput is around 800KB/s at the maximum payload size of 1024 bytes. It also achieves at least 330 subscriptions per second at a maximum. 128 bytes payload size is a turning point on longer subscription response time and lower throughput.

The remaining sections of the paper are structured as follows. Section 2 describes CoAP (Constrained Application Protocol) which is the transfer protocol used in the system design. Section 3, describes in detail the system design and experiments are conducted in Section 4. The paper concludes in Section 5 with findings and future works.

II. Literature Review

The Constrained Application Protocol (RFC7252) was initially proposed by ARM and Universitaet Bremen TZI in June 2014. It is a specialized transfer protocol for machine-to-machine communication and for use with constrained nodes and constrained networks. The nodes often have small microcontrollers with small amounts of ROM and RAM, while constrained networks often have low power and high packet error rates and a typical throughput of 10s of kbit/s. CoAP is based on UDP protocol and package size is varies from minimum 4 bytes to maximum 1024 bytes. It provides a 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. CoAP can also easily interface with HTTP for integration with the Web [25].

The Internet protocol has undergone decades of development which allows any communication object with embedding function connect together. While the Internet protocol connects devices, Web protocols like HTTP allow us exchange and visualize information provided by these devices. Unfortunately at the time that Web protocols and Internet protocols were invented they did not seriously take limitation of energy, memory and processing power into account. Nowadays a lot of devices like different kinds of sensors have network connections but with constrained resource, low frequency processor, limited RAM and limited power supply etc. Web and Internet protocol are too heavy for embedded system by adding extra overhead that the system and application do not care about. A relative simple but with sufficient functional application protocol is desired for such systems. CoAP as one of such application protocols has been proposed and continues to attract research attention [27, 28]. Not only is CoAP a complementary protocol for applications but also it is more suited to even higher protocols like REST compared to HTTP in such constrained network environment. Moreover, the recent surge in the use of sensors and the Internet of Things (IoT) means we can build an application level protocol that makes them work together in a huge or small inter-connected architecture. CoAP can be used for that purpose. Some researchers have discussed how CoAP adapts into the Internet of Things [18] and some researchers have proposed their architecture to move application logic from device specific domains to the cloud through the adoption of the CoAP [16].

There are many application protocols but two emerging protocols are designed for highly resource-constrained environments namely: the Message Queuing Telemetry Transport (MQTT), and the Constrained Application Protocol (CoAP). By applying these protocols on resource-constrained environments, we can improve device and application performance in terms of bandwidth consumption, battery lifetime and communication latency. Although they both can achieve such goal, they have different application scenarios to adapt into [7]. The major difference is their communication patterns. MQTT is a pub/sub protocol and CoAP is RESTful protocol with built-in pub/sub mechanism. Since a RESTful protocol uses HTTP verbs, CoAP can be very easily integrated with the most web application through a HTTP-CoAP proxy, which is very critical feature for distributed applications running in the Cloud. From transportation protocol perspective, they both provide basic congestion control by retransmitting messages that are not acknowledged. However CoAP is based on UDP and MQTT is based on TCP. TCP provides more reliable connection but with a lot extra overhead compared to UDP, so the same thing can be expected on MQTT and CoAP. The reliable connection provided by TCP is not necessarily an advantage in a sensor network. Sensor applications in general tolerate few missing data packages since a sensor can generate thousands of data packages in every second. Such big data generation rate can degrade application performance by TCP due to extra overhead in its protocol. Small extra overhead can be easily amplified at such high sending rate. CoAP protocol uses UDP so it has more advantage on this. Except the above difference, CoAP has more unique advantages that MQTT does not support. Since CoAP is a RESTful protocol and can be easily transformed into HTTP, CoAP application can obtain a better response performance by HTTP caching. CoAP also provides a mechanism called *Blockwise*, which is a solution for transferring large set of data in block fashion. One of the most important features for sensor network is resource publishing and subscribing. CoAP has built-in mechanism called Observe to discovery resource or services in network.

III. System Design

A. System Overview

A prototypic application is developed for sensor data collection as shown in Figure 1.

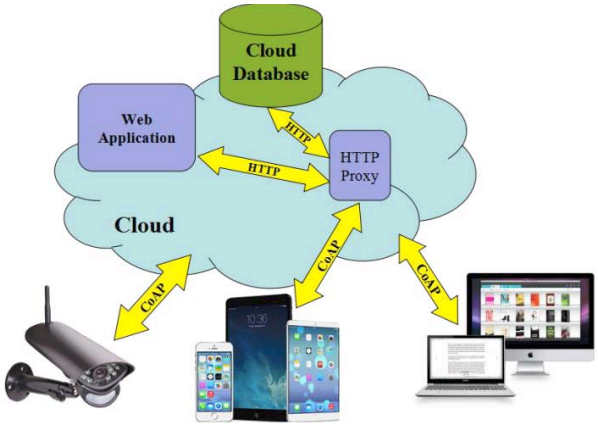

Figure 1. The proposed architectural design.

The designed system can be categorized into three: client, server, and network. The application collects selected sensor data from a mobile device and sends it to a remote server over a wireless network. The server does not necessary mean a powerful "monster" machine and it can be smartphone, tablet, laptop or desktop computer. Basically what the server does is receiving a resource observation request and returning a series of sensor data to the client until the client sends a RST request to cancel the subscription. Both the client and server run the same prototype application just that one is in client mode and another one is in server mode. So, they can communicate in the same environment. The sensor data is generated via a special hardware event. Once the application captures the event, it reads the raw sensor data, puts it into CoAP package and sends it out. Regarding different test scenarios, all communications between the client and server are accomplished in a dedicated wireless network and a public wireless network. The detail of hardware infrastructure is included in experiment setup section.

From Figure 1, it is important to point out that the sensor device and the mobile can share information via CoAP. However, when the sensor data is required to be stored into an external storage, say cloud database, then a different communication protocol is required. In this work, we proposed an HTTP proxy which can act as a message router between the CoAP supported devices and the cloud services. The services running in the cloud consist of a cloud database and Web applications.

B. Design

To get wireless network I/O performance data, the experimental aspects are identified at first. They help us obtain and understand consistent performance results. Here is a list of the aspects identified:

- Resource discovery and I/O subscription
- CoAP payload size
- Packet sending Interval
- Type of wireless network
- Network protocol
- Data source
- Repeat packet sending times
- OS environment

All above aspects are divided into two major categories. The first five aspects are considered for how we better understand wireless network I/O performance. The last three aspects are considered for producing consistent experimental results. All the aspects are briefly introduced here. Resource discovery is an important mechanism in a sensor network. CoAP has the built-in mechanism called Observe. The client sends out a GET request to a special URI (./well-known/core) on a server. The URI is part of the CoAP standard. The server receives such request and it knows this is a request to obtain all resources available on this device. The server then returns a resource list to the client. Once the client makes decision on which sensor (I/O resource) to subscribe, it sends out an Observe request with chosen sensor id to the server for receiving the sensor's update. The server adds the client into a subscription list on the chosen resource and once the sensor data is changed all subscribers on the list are updated.

The update does not require any further request from the client. The server keeps sending sensor data to the client instead of the client pulling the data from the server. Unless the client is no longer interested in the sensor data, it sends out a RST request to the server to cancel the sensor I/O subscription. In that case, the server will no longer send any information to the client. CoAP payload size is used to investigate the relationship between the package size and performance. The maximum payload size in CoAP is 1024 bytes, multiple payload size will be chosen in my experiments. Sending interval is used for measuring sensor I/O throughput. After a fixed interval time or any time sensor produces a new data server will send sensor data to notify the client.

Since these experiments are designed for performance evaluation, we need find out the upper and lower boundary of the throughput. The lower boundary obviously is 0 and the upper boundary depends on not only network delay but also how fast the server can send out the data. So sending intervals in all tests are 0, which means once a sensor data is sent out from the server, the next sensor data is ready for sending. To understand how different types of wireless networks impact the I/O performance, both dedicated network and public network are tested.

A dedicated network as its name indicated is no other network traffics except our experiment traffic. A public network will be used to simulate a real world environment. Since the experiments are based on CoAP, UDP protocol is used underneath. As mentioned above, the last three aspects are considered for getting a consistent result. For data source, we need to choose a consistent and stable data generator for every experiment; otherwise the result will not be consistent.

To evaluate the performance result, some measurement metrics are identified too and they are:

- Subscription Throughput - maximum number I/O subscription per second

- Subscription Response time - I/O subscription response time in millisecond
- Sensor Data Throughput - maximum sensor I/O data per second in KB
- Sensor Data Response time - I/O data transfer response time in millisecond
- Subscription service time at cold server - time spend on server to handle I/O subscription at very first time
- Subscription service time at warm server - time spend on server to handle I/O subscription, not at the first time
- Variance - between repeat each I/O data transfer and I/O subscription

Throughput is measured in how many bytes or number of subscription that can be sent or made in every second based on different payload size. Response time is measured in milliseconds. It is interesting to see if there is a big variance between repeated experiments as well.

C. Data Collection

In the experimental setup, we have two Android devices. One device plays client role and another one runs as a server. Samsung GT-P7510 is the client sending all requests and all raw data are collected on it too. Before running into measurement phase in a test, it connects to remote server which is Nexus 7 and discovers all of its available sensors. Single sensor is chosen and used in all experiments. In order to remove bias on individual sampling, the I/O subscription and I/O transfers are repeated multiple times. The subscription repeats 30 times and the I/O transfer repeats 100 times for each I/O subscription.

In each experiment, all collected data are recorded in a log file. When an experiment is finished, the mean response time is calculated from individual subscriptions or data transfer response time and its throughput is calculated from the sum of the sent I/O data size divided by its total elapsed transfer time.

IV. Performance Evaluation

Following the experimental design, tests are conducted in different networks and all performance data needed for the analysis are collected. The following performance graphics are produced based on the raw data. For all graphics except particular indication, the horizontal axis represents payload size from 16 up to 1024 bytes and the vertical axis represents KB/s in throughput graphics and millisecond in response time graphics.

A. Dedicated Network

From Figure 2, we can see there is no big difference in data transfer response time. They all fall into a small range, 1.08 to 1.45 ms. Since all response time are very fast, any network interference or OS activity can easily cause the response time to fluctuate. Although at 64 and 512 bytes payload the response time drops a bit, the general trend of response time increases slowly when the payload size becomes bigger. The data transfer rate grows along as the payload size grows and the throughput ends up with 791 KB/s on payload size of 1024 bytes as indicated in Figure 3. This is expected that a bigger payload size has higher I/O throughput. Closely looking into the data, we can see the I/O throughput approximately doubles as the payload size is doubled. For the subscription response time, until 128 bytes payload size the response time keeps low and less than 100ms as seen in Figure 4.

However starting from 256 bytes payload, the response time is increased rapidly and ended up with 515ms at 1024 bytes payload. The response time is subjected to payload size because it measures the elapsed time including the time that the first sensor data is received. Before 128 bytes payload, although the size is doubled every time they are still considered relatively small. After 128 bytes payload, the doubled payload size introduces some extra overhead to finish the request. These extra overheads could come from memory allocation, network transfer or their combination. If the application is sensitive to subscription request time, we need to choose the payload size less than or equal to 128 bytes.

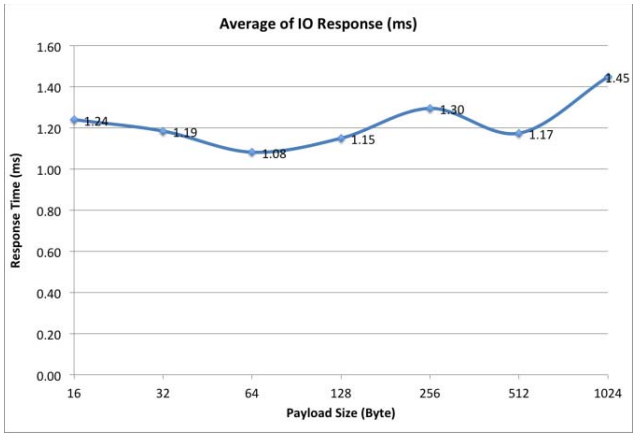

Figure 2. Dedicated Network - Average IO Data Transfer Response Time

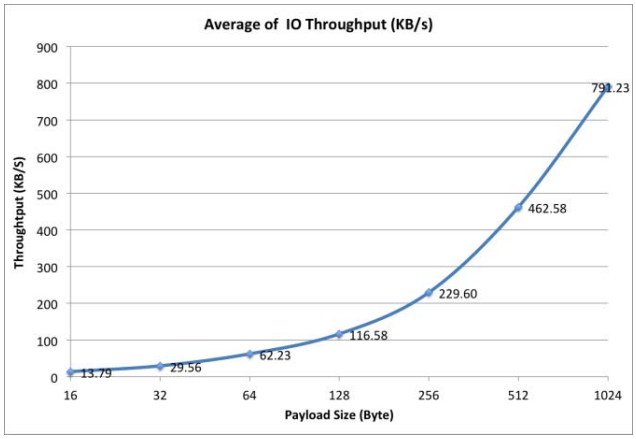

Figure 3. Dedicated Network - Average IO Data Transfer Throughput

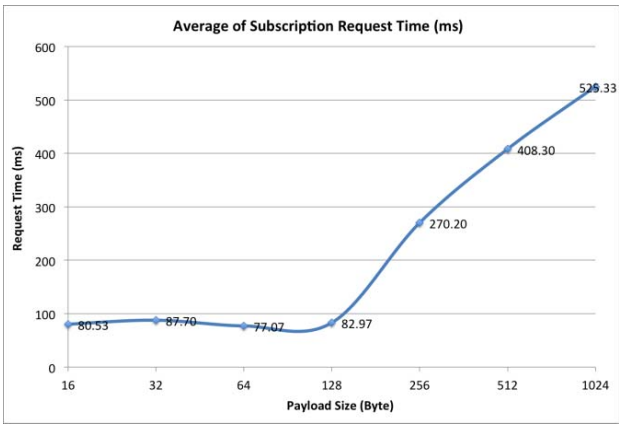

Figure 4. Dedicated Network - Average Subscription Response Time

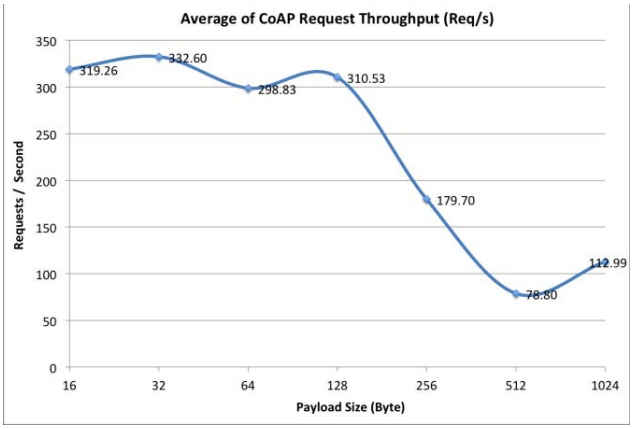

Figure 5. Dedicated Network - Average Subscription Throughput

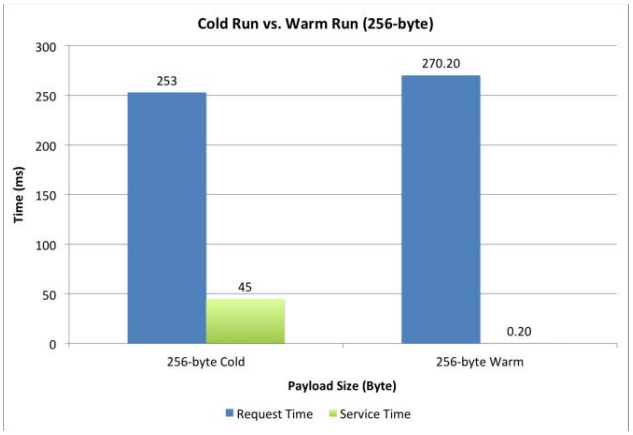

Figure 6. Cold Subscription versus Warm Subscription on Response Time

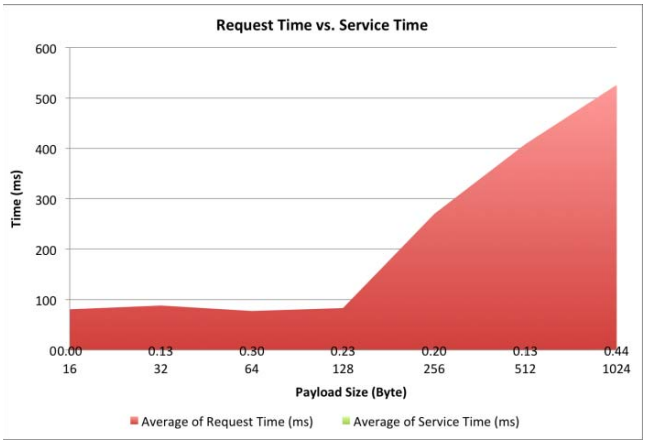

Figure 7: Subscription Response Time versus Service Time

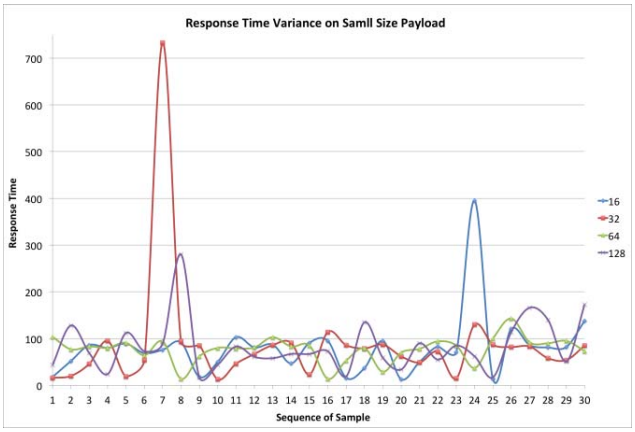

Figure 8: Response Time Variance on Small Size Payload

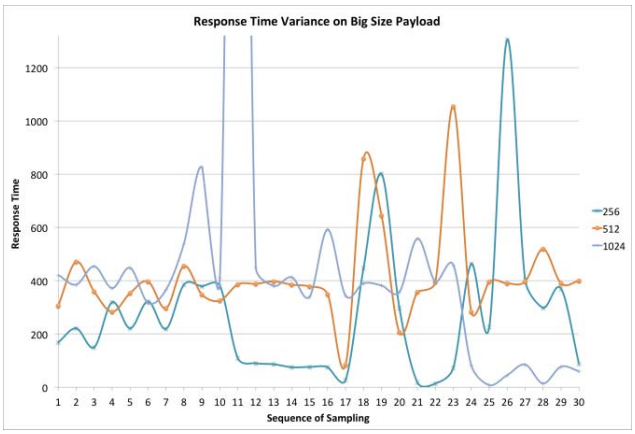

Figure 9: Response Time Variance on Big Size Payload

In Figure 5, the phenomenon is basically the same as the subscription response time although there is bigger fluctuation. The turning point is still 128 bytes payload and the throughput goes down. Is there any difference between the first time subscription (cold subscription) and later subscription (warm subscription)? In the experiment, it shows there is a big difference in services time as seen in Figure 6. The service time for the first subscription takes 45ms but for later subscription service time is reduced to less than 0.5ms. The interesting information reading from the figure is the request time. What we expect is that the request time is reduced when the service time is reduced. This can be

explained by individual sampling. Since the cold subscription is the very first subscription and it is one time event, the response time can be inaccurate and it is lucky enough to run at a lower network delay. However the service time is accurate on both measurements because it does not depend on any network activity and only measure the elapsed time on the server.

Figure 7 shows how much time is spent on network delay and how much time is spent on the server for a I/O subscription. The green line which is service time can barely be seen because they are very short and all are less than 0.5ms. This figure also tells us the major overhead for subscription response time on big payload size >128 bytes (Figure 4) comes from network delay. The last thing learned from the experiments is the variability for the response time.

In Figure 8 and Figure 9, both graphs show such variability on a small and a big payload size respectively. As we can see, a bigger payload size has more variance compared to small payload size. All 256, 512 and 1024 bytes payload have big variation during their repeated runs. In contrast, the small payload size (<256 bytes) keeps variance within 100 to 200ms. Only very few cases have big increased response time. Since the bigger payload size has much high variance, it might explain why the payload bigger than 128 bytes has noticeable increase in the response time. They can respond in faster time but with high variance, the average response time is increased indeed.

B. Public Network

From Figure 10, we can see there is no big difference in data transfer response time. They all respond very fast and their range is from 1.15 to 1.3 ms. The maximum response time 1.3ms still happens on maximum payload size of 1024 bytes. The data transfer rate grows along as the payload size grows and the throughput ends up with 865.11 KB/s with payload size of 1024 bytes (Figure 11), which is a little bit faster than the maximum throughput on the dedicated network.

Again we can see the I/O throughput approximately doubled as the payload size double. For the subscription response time, 128 bytes payload size still is the turning point where the response time with the bigger payload size gets the longer time (Figure 12). The response time with >128 bytes payload is increased rapidly and ended up with 307ms at 1024 bytes payload. The maximum response time is much faster than what we see in the experiment over the dedicated network but the trend is same except the line shifts lower.

In Figure 13, it basically tells the same story as we see in the subscription response time. The throughput increased a bit and this is expected since the response time is improved from the experiment on the dedicated network. In Figure 14, the cold and warm subscription comparison, we see the same outcome but also different perspective. The service times are the same. The service time in cold subscription is 36ms and warm subscription is 0.33ms. The big service time gap is still there. The difference however is the subscription response time. This time we get 389ms for cold and 159ms for warm subscription. Actually this is the result that we expect since the service time gets much shorter. However in the previous comparison (Figure 6) on the dedicated network we saw a reverse result. As explained above, the comparison is based on one time event and the previous one is just not luck getting faster network delay. In comparison with Figure 15, the request time is still far more than the service time. They are same as the previous comparison but with a faster request time. The maximum response time at 1024 bytes payload is around 300ms compared to more than 500ms before (Figure 7).

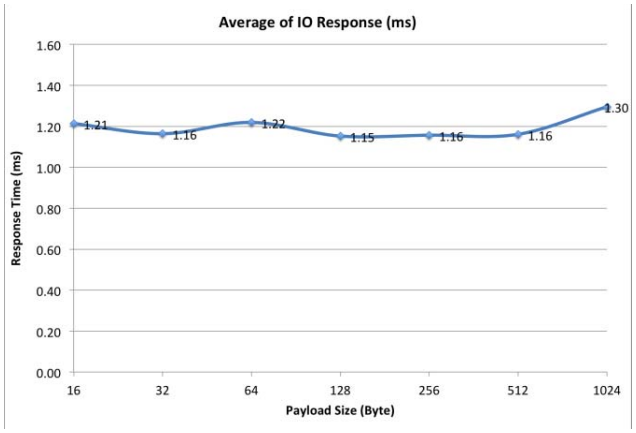

Figure 10. Public Network - Average IO Data Transfer Response Time

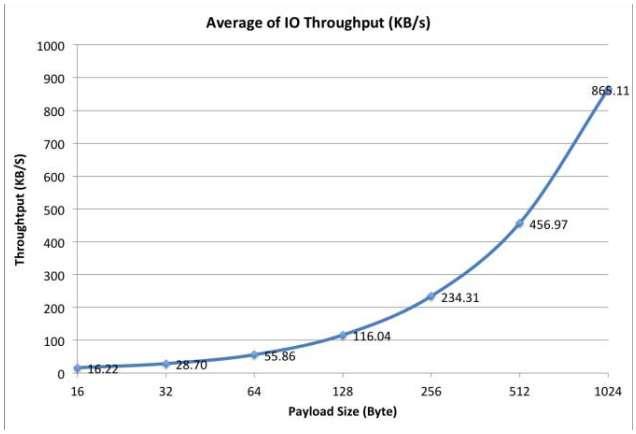

Figure 11. Public Network - Average IO Data Transfer Throughput

C. Dedicated Network vs. Public Network

In all the exact same set of experiments conducted within the dedicated and public network, we can see there is no significant different behavior between the two types of networks. For the I/O performance, they both can respond in less than 1.5ms and the maximum throughput reaches around 800KB/s at maximum 1024 bytes payload size.

On subscription perspective, they all show 128 bytes payload size is a turning point. The bigger payload size gets much longer response time and lower throughput. Regardless of the network, we are able to get minimum subscription request around 100 subscription request per second and

maximum at more than 330 request per second. The only big performance difference between the two networks is in the public network where we get a bit higher performance in terms of response time and throughput. The major factor contributing to this difference can be the router. The router used in the dedicated network is a regular and home router. Although the exact model used in the public network is unknown, the expectation is it's a high-end router since it serves much more people. The high-end router always has more powerful processor, more interval memory and more efficient scheduling algorithm to give a quicker response time and serve more online users. This might be reason why we see a better result in the public network experiments.

However the good thing is we do not see a big difference between the two, so this means the performance is more sensitive to the device and its OS rather than the network infrastructure. This is why the paper did not emphasis any maximum or boundary value. Instead, the focus is placed on the level they are able to reach.

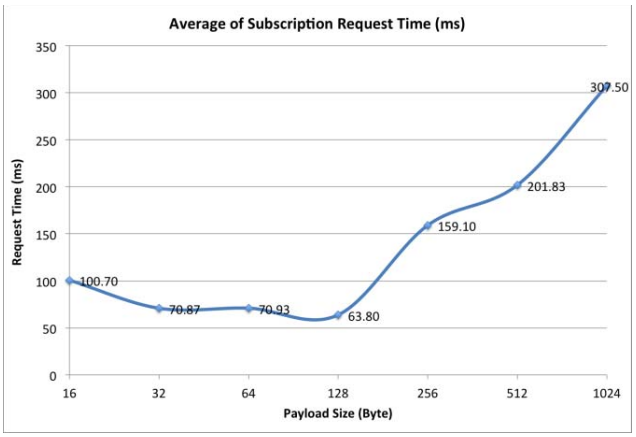

Figure 12. Public Network - Average Subscription Response Time

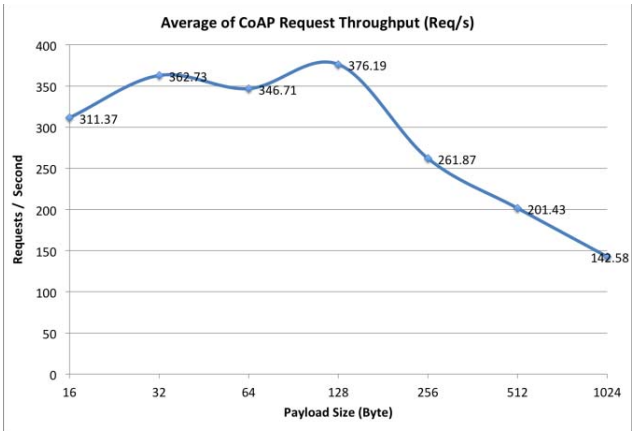

Figure 13: Public Network - Average Subscription Throughput

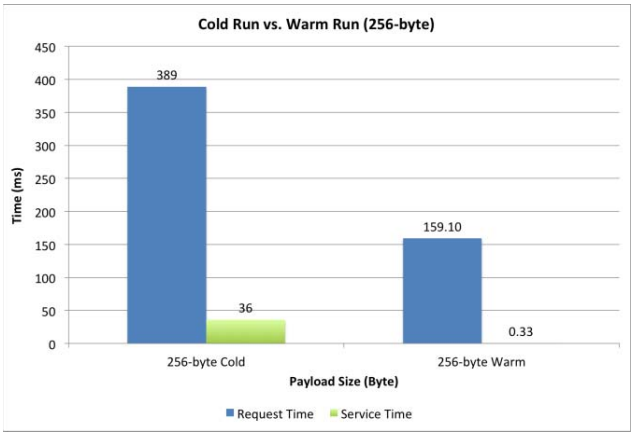

Figure 14: Cold Subscription versus Warm Subscription on Response Time

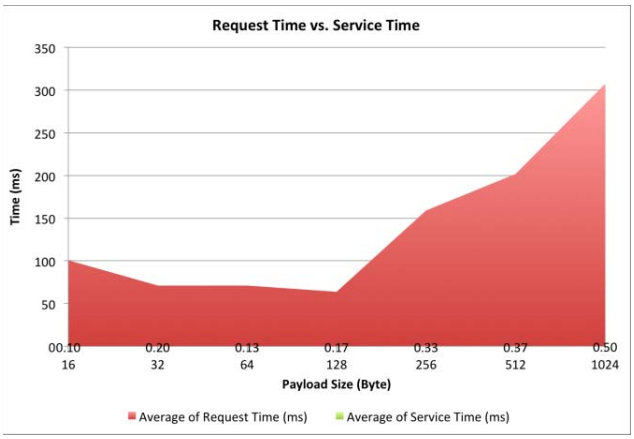

Figure 15: Subscription Response Time versus Service Time

V. CONCLUSIONS AND FURTHER OUTLOOK

As mobile and cloud computing powered technologies are commonly used in our daily lives and the interest continues to grow, the computation resources have been well distributed over networks. Such networks bring new opportunities to create next generation applications that were almost impossible before. Utilization of mobile device's sensor data for remote computers is one example of such new services and applications delivery.

The Constrained Application Protocol (CoAP) as a new application protocol suites the communication requirement in a constrained environment. To understand I/O performance over CoAP in wireless networks can help us identify challenges of such new applications, design a better policy on data exchanges over a network and better application architecture to satisfy both throughput and response time requirements.

This work is therefore dedicated to the performance analysis of CoAP in a sensor-mobile-cloud ecosystem. Through a set of experiments designed for wireless network I/O in this paper, it is observed that CoAP can achieve quick I/O data response time under 1.5ms regardless of the payload size. The maximum throughput is around 800KB/s at the

maximum payload size of 1024 bytes. It also achieves at least 330 subscriptions per second at maximum. 128 bytes payload size is a turning point on longer subscription response time and lower throughput.

The applications that are sensitive to response time need to choose the payload size that is less than 256 bytes. Bigger payload size (greater or equal to 256) has dramatically high variance on response time so a longer response time is expected sometime.

In future studies, the scalability of CoAP inspired networks with multiple actors can be investigated.

REFERENCES

[1] Jamal N Al-Karaki and Ahmed E Kamal. Routing techniques in wireless sensor networks: a survey. Wireless communications, IEEE, 11(6):6–28, 2004.

[2] William Allcock, John Bresnahan, Rajkumar Kettimuthu, and Joseph Link. The globus extensible input/output system (xio): A protocol independent io system for the grid. In Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International, pages 8–pp. IEEE, 2005.

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. Communications of the ACM, 53(4):50–58, 2010.

[4] Soma Bandyopadhyay and Abhijan Bhattacharyya. Lightweight internet protocols for web enablement of sensors using constrained gateway devices. In Computing, Networking and Communications (ICNC), 2013 International Conference on, pages 334–340. IEEE, 2013.

[5] Cisco. Cisco visual networking index: Global mobile data traffic forecast update, 2013-2018, feb 2014.

[6] Walter Colitti, Kris Steenhaut, Niccol`o De Caro, Bogdan Buta, and Virgil Dobrota. Evaluation of constrained application protocol for wireless sensor networks. In Local & Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on, pages 1–6. IEEE, 2011.

[7] Niccol`o De Caro, Walter Colitti, Kris Steenhaut, Giuseppe Mangino, and Gianluca Reali. Comparison of two lightweight protocols for smartphone-based sensing. In Communications and Vehicular Technology in the Benelux (SCVT), 2013 IEEE 20th Symposium on, pages 1–6. IEEE, 2013.

[8] Dawson R Engler, M Frans Kaashoek, et al. Exokernel: An operating system architecture for application-level resource management, volume 29. ACM, 1995.

[9] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. CISCO white paper, 1, 2011.

[10] Ian Foster, David Kohr Jr, Rakesh Krishnaiyer, and Jace Mogill. Remote i/o: Fast access to distant storage. In Proceedings of the fifth workshop on I/O in parallel and distributed systems, pages 14–25. ACM, 1997.

[11] Ajay Gulati, Anne Holler, Minwen Ji, Ganesha Shanmuganathan, Carl Waldspurger, and Xiaoyun Zhu. Vmware distributed resource management: Design, implementation, and lessons learned. VMware Technical Journal, 1(1):45–64, 2012.

[12] Ajay Gulati, Ganesha Shanmuganathan, Anne Holler, and Irfan Ahmad. Cloud-scale resource management: challenges and techniques. In Proceedings of the 3rd USENIX conference on Hot topics in cloud computing, pages 3–3. USENIX Association, 2011.

[13] IDC. Idc forecasts worldwide public it cloud services spending to reach nearly $108 billion by 2017 as focus shifts from savings to innovation, sep 2013.

[14] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Mobile computing, pages 153–181. Springer, 1996.

[15] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code o✏oading. In INFOCOM, 2012 Proceedings IEEE, pages 945–953. IEEE, 2012.

[16] Matthias Kovatsch, Simon Mayer, and Benedikt Ostermaier. Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things. In Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on, pages 751–756. IEEE, 2012.

[17] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. Software: Practice and Experience, 32(2):135–164, 2002.

[18] Christian Lerche, Klaus Hartke, and Matthias Kovatsch. Industry adoption of the internet of things: a constrained application protocol survey. In Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on, pages 1–6. IEEE, 2012.

[19] Vivek S Pai, Peter Druschel, and Willy Zwaenepoel. Io-lite: a unified i/o bu↵ering and caching system. ACM Transactions on Computer Systems (TOCS), 18(1):37–66, 2000.

[20] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In Middleware 2001, pages 329–350. Springer, 2001.

[21] Elizabeth M Royer and Chai-Keong Toh. A review of current routing protocols for ad hoc mobile wireless networks. Personal Communications, IEEE, 6(2):46–55, 1999.

[22] Mahadev Satyanarayanan. Fundamental challenges in mobile computing. In Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing, pages 1–7. ACM, 1996.

[23] Mahadev Satyanarayanan. Mobile computing: the next decade. ACM SIGMOBILE Mobile Computing and Communications Review, 15(2):2–10, 2011.

[24] Mahadev Satyanarayanan, Paramvir Bahl, Ram´on Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. Pervasive Computing, IEEE, 8(4):14–23, 2009.

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

[26] Zach Shelby. Embedded web services. IEEE Wireless Communications, 17(6):52, 2010.

[27] Berta Carballido Villaverde, Dirk Pesch, Rodolfo De Paz Alberola, Szymon Fedor, and Menouer Boubekeur. Constrained application protocol for low power embedded networks: A survey. In Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on, pages 702–707. IEEE, 2012.

[28] Carl Waldspurger and Mendel Rosenblum. I/o virtualization. Communications of the ACM, 55(1):66–73, 2012.