듀티사이클 환경의 무선센서네트워크에서 분산 브로드캐스트 스케줄링 기법

Thien-Binh Dang, Manh-Hung Tran, Duc-Tai Le, 염상길, 추현승
e-mail: {dtbinh, hungtm, ldtai, sanggil12, choo}@skku.edu

Performance Evaluation of an IoT Platform

Thien-Binh Dang, Manh-Hung Tran, Duc-Tai Le, Sanggil Yeom, Hyunseung Choo
The College of Software, Sungkyunkwan University

Abstract

Accompanying the Internet of Things (IoT) is a demand of advanced applications and services utilizing the potential of the IoT environment. Monitoring the environment for a provision of context-aware services to the human beings is one of the new trends in our future life. The IoTivity Cloud is one of the most notable open-source platform bringing an opportunity to collect, analyze, and interpret a huge amount of data available in the IoT environment. Based on the IoTivity Cloud, we aim to develop a novel platform for comprehensive monitoring of a future network, which facilitates on-demand data collection to enable the network behavior prediction and the quality of user experience maintenance. In consideration of performance evaluation of the monitoring platform, this paper presents results of a preliminary test on the data acquisition/supply process in the IoTivity Cloud.

1. Introduction

Several significant technology changes have come together to enable the rise of the Internet of Things (IoT) [1], [2], [3]. IoT is envisioned to contain billions of devices, including RFID devices, sensors, smartphones, cars and so on, in near future. To make variety of smart systems such as smart city, smart health care, smart transportation and smart manufacture feasible in the IoT environment, such devices are required to generate a large amount of data and communicate together [4]. There emerges the development of software frameworks to allow the heterogeneous IoT devices to communicate and leverage common software applications.

IoTivity open-source software framework has been developed to enable seamless device-to-device connectivity to address the emerging needs of the IoT [4]. The project, sponsored by the Open Connectivity Foundation (OCF) [6], aims to create a new standard and an open source implementation, which will help ensure interoperability among products and services regardless of maker and across multiple industries, including smart home, automotive, industrial automation, and healthcare. The goal of the framework is an extensible and robust architecture that works for smart and thin devices. As being sponsored by a group of industry leaders, it is expected to become a standard specification and certification program to ensure secure and reliable connections between IoT devices and the Internet.

Recently, the IoTivity Cloud open-source platform [7] has been developed based on the IoTivity framework. The platform brings an opportunity to collect, analyze, and interpret a huge amount of data available in the IoT environment. The processed data is then required to be available for a provision of smart services to human beings. All these data should be acquired in real-time, stored for a long period and analyzed in a proper way. To this end, we aim to develop a high scalability and low overhead monitoring platform. The monitoring platform will be designed in a way of collecting any type of data into a cloud database by utilizing IoTivity Cloud. This feature opens many opportunities for network operators to customize the monitoring system based on their demands in target deployment area.

In the scope of this paper, we study the stability and scalability of the IoTivity Cloud, which is the most essential component of our monitoring platform. Particularly, we measure throughputs of Cloud Interface server in the IoTivity Cloud under different network loads, in a simple testbed with single-machine-based deployment. The main goal of this work is to generate a baseline that we can use to re-evaluate the performance of the whole platform in a cloud-based deployment. For the rest of the paper we use the terms framework and platform to refer to the IoTivity framework and the IoTivity Cloud platform, respectively.

The rest of the paper is organized as follows. In Section 2, some popular IoT platforms are reviewed. Section 3 provides detail information of components in the IoTivity Cloud platform. Our testbed, testing scenarios, and performance evaluation results are presented in Section 4. Finally, we conclude our paper, and discuss our future work in Section 5.

2. Related Work

There exist many open-source platforms now a day, that can support entire development of IoT applications and systems [8]. In this section, we review some popular IoT platforms. ThingWrox [9] is one of the earliest software platforms designed to build and run the IoT applications. It focuses on rapid development of IoT applications such as Smart Home, Smart City, Smart Agriculture, Smart Grid, and Smart Water. ThingWrox is a complete development suite that enables application design, runtime, and intelligence environment. ThingWrox uses REST, MQTT and sockets for data communication. The strong features of ThingWrox include modern and complete platform provisioning, faster deployment and search-based intelligence.

ThingSpeak [10] is an IoT application platform for the development of IoT systems. With ThingSpeak, users can develop applications which can collect data from sensors, such as an application of location tracking, controlling, and monitoring home appliances. The key features of the platform include real time collection of data storage, data analytics and visualization using MATLAB, open API supporting and providing geolocation data. In addition, it enables an integration with Tweeter, i.e. users can get update status of their devices from tweets. The HTTP protocol is utilized to store or retrieve data from things over the Internet or via a Local Area Network.

Google Cloud platform [11] enables developers to code, test and deploy their IoT applications with highly scalable and reliable infrastructure. The developers now just focus on the programming work and Google handles issues regarding infrastructure, scalability, computing power and data storage. What make Google Cloud platform become one of the most popular IoT platforms are fast global network, higher performance, environment safe cloud, Google's big data tool, supporting of various available cloud services like BigQuery, PubSub, Connecting Arduino, RiptideIO and many more.

A common theme among the above platforms is that there is no performance evaluation report apart from general statements about their stability, scalability, and cloud deployment ability. Recently, Vandikas et.al. [12] have evaluated the performance of their platform, called IoT-Framework [13]. The work of these authors is our inspiration to evaluate the performance of the IoTivity Cloud platform.

3. IoT Cloud Platform

IoTivity is an open source framework implementing OCF standards for the IoT software developments. The framework operates as middleware across all operating systems and connectivity platforms. It consists of four key components including (1) device and resource discovery, (2) data transmission, (3) device management and (4) data management, as shown in Figure 1 [5].

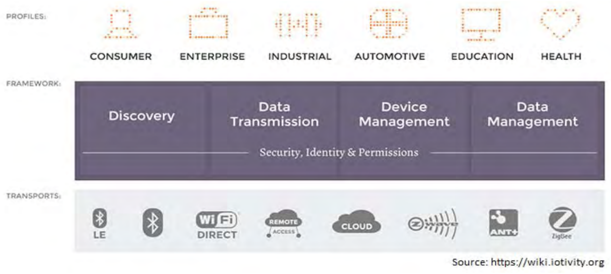


Figure 1. IoTivity common object model

The IoTivity discovery component supports multiple discovery mechanisms for devices and resources in proximately and remotely. IoTivity adopts the Constrained Application Protocol (CoAP) [14] defined by the Internet Engineering Task Force (IETF) [8] as a data transmission protocol. As CoAP is a lightweight alternative to Hypertext Transfer Protocol (HTTP), it can work with HTTP by using intermediaries to translate between two protocols. While the data management component supports the collection, storage and analytics of data from various devices, the device management component aims to provide a one-stop-shop that supports the configuration, provisioning and diagnostics of devices in an IoT network.

IoTivity Cloud is an open-source platform that aims to extend accessibility of IoTivity devices. The IoTivity Cloud supports techniques such as HTTP to CoAP proxy and OAuth2 [16] over CoAP to enable users to access their devices under their preference accounts over the cloud. Architecture of the IoTivity Cloud is depicted in Figure 2. The platform includes IoT controllers who own IoT devices. To be widely used over the cloud, both controllers and devices must be registered to the cloud first. The devices read sensory data of the physical world, and then send the data to IoTivity cloud servers. Once the data have been published, IoT controllers can access them even they are not co-located.

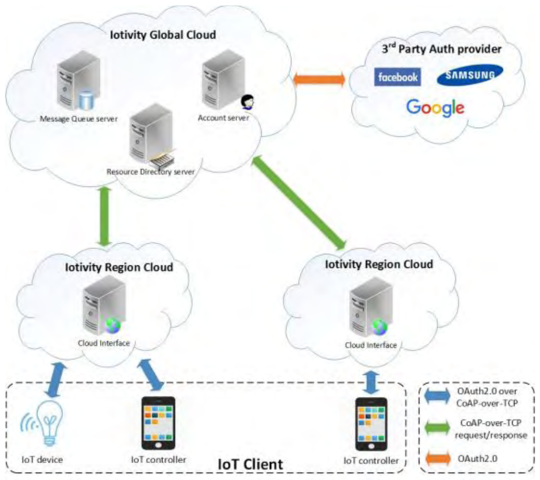


Figure 2. IoTivity Cloud architecture

Most of IoTivity framework core components are developed in C and C++, but the IoTivity Cloud is developed in Java. In the platform, servers are separated into two levels: IoTivity Region Cloud (IRC) and IoTivity Global Cloud (IGC). IRC includes regional Cloud Interface servers that accept connection from both IoT devices and IoT controllers, receive sensory data and sends RESTful messages through connected pipelines to IGC. The IGC is the global cloud that includes Message Queue, Resource Directory, and Account servers. The functionality of each component is explained as follows:

- **Cloud Interface (CI)**: a server acts as an interface of IGC. It is basically a proxy of Message Queue, Resource Directory, and Account servers. Additionally, CI handles the server side OAuth2.0 handshake protocol and the keep-alive messages from IoT devices to ensure the connectivity between the devices and the cloud. Last but not least, CI relays handler such that IoT devices and IoT controllers can communicate when they are connected to different regional clouds.
- **Resource Directory (RD)**: a server supports device registration, discovery, updating, or deleting to IoTivity Cloud. It deploys MongoDB to manage the database of IoT device information.
- **Account server**: a server supports third party OAuth2.0 enabled authentication providers like Google, Facebook, and Github extending their user's identity to IoTivity Cloud. After a user sign up to the Account server, his/her information and a corresponding access token will be stored into a database. Consequently, the user can register his/her devices to IoTivity Cloud for future use.

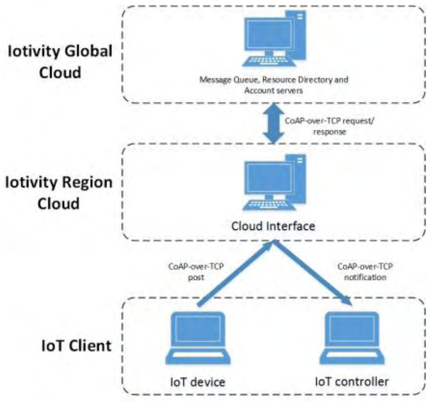
- **Message Queue (MQ)**: a broker exposes an interface for clients to initiate a publish/subscribe interaction. The server is built on the top of Apache Zookeeper [16] and Apache Kafka [17]. Apache Zookeeper is an open source providing high performance coordination service for distributed applications. It is mainly used to track status of nodes, content topics, and messages, stored in an Apache Kafka cluster. Apache Kafka is a distributed publish/subscribe messaging system. It is written in Scala programming language and designed for processing of real time activity stream data, e.g. logs and metrics collections.
- **IoTivity client**: IoTivity client consists of IoT devices and IoT controllers. An IoTivity client handles the client side OAuth2.0 handshake protocol and sends keep-alive messages to CI periodically. The client is also able to send resource registration/discovery requests to the cloud.

4. Performance Evaluation

In this section, we describe performance evaluation of the IoTivity Cloud. We start by depicting our testbed, and then describing the testing scenarios. Finally, we discuss on selected evaluation metrics and obtained evaluation results.

A. Testbed setting

Our testbed consists of four physical nodes as shown in Figure 3. IoT device node is a Mid-2010 MacBook Pro with 4GB RAM and an Intel Core 2 duo CPU 2.4 GHz. IoT controller node is hosted in a laptop computer with 8GB RAM and an Intel Core i5 duo CPU 2.3 GHz. The CI is hosted in a desktop computer with 3 GB RAM and an Intel Core i5-2500 CPU 3.3 GHz. The IGC with three components of MQ, Account, and RD servers is set up in a single desktop computer. The computer has 16 GB RAM and Intel Core i7 CPU 3.60 GHz. All the computers run the Ubuntu 16.04 LTS operating system, and support IoTivity framework 1.2.0. The IoTivity Cloud is deployed on the testbed similar to its architecture. We have added Java hooking scripts into source code of the CI and the MQ to measure the performance of these servers. The size of requests/responses are set in the simplest case where a message contains only one character. The number of devices are varied to identify maximum capacity of the system.


Iotivity Global Cloud
Message Queue, Resource Directory and Account servers
CoAP-over-TCP request/ response
Iotivity Region Cloud
Cloud Interface
CoAP-over-TCP post
CoAP-over-TCP notification
IoT Client
IoT device
IoT controller
Figure 3. Testbed model

B. Testing methodology

We are interested in how the IoTivity platform operates under stress testing with different number of IoT devices and IoT controllers. In our testing, we generate multiple threads on the IoT device node, each of them represents for a device which generates data. Similarly, multiple processes of controller, which receives data, run in the IoT controller node. Recall that we especially focus on the throughput of the CI. Even though the IoTivity Cloud is designed to deploy on a cluster of nodes, the test we have conducted is intentionally narrowed to a single node for the IRC and a single node for IGC.

We consider two testing scenarios. The first scenario uses various number of IoT devices on producing side and no IoT controllers. In this scenario, the CI is tasked to only process a large number of post requests from IoT clients without disseminating these post requests to IoT controllers. The second testing scenario is similar to the first one but adding various number of IoT controllers. In both testing scenarios, the IoT devices generate post requests in a Poisson process [19]. In the scope of this paper, we just experiment the first scenario, the second scenario will be carried out in future work. The number of IoT devices and the arrival rate λ (msg/sec) determine the level of stress testing that the higher value of λ or the greater number of IoT devices means more post requests arrive at the CI in a period of time. The value of λ is set equally to all IoT devices for an individual test. Donald Knuth describes a method to generate random timings for a Poisson process in [20]. In this method, the exact amount of time until the next event is determined by the formula:

$$nextTime = \frac{-\ln p}{\lambda},$$

where p is a random value between 0 and 1.

The algorithms to generate random timings for a Poisson process are demonstrated in the Algorithm 1 and 2. All source code for the simulation, customized CI and customized MQ are available on the Github in order to make the simulation and evaluation are accessible and reproducible to wider audience.

Algorithm 1 Calculate the time for next event

```
1: procedure NEXTTIME($\lambda$)
2:     p = random()
3:     Return -ln(p)/$\lambda$
4: end procedure
```

Algorithm 2 Checking timer to generate events

```
1: procedure CHECK($\lambda$,$timer$,$deltaSecond$,$endInterval$)
2:     $timer = timer + deltaSecond$
3:     if $timer > endInterval$ then
4:         timer = 0
5:         endInterval = nextTime($\lambda$)
6:         doEvent()
7:     end if
8: end procedure
```

C. Testing scenarios and results

We use a C++ program which generates a maximum number of 25000 CoAP-over-TCP post requests from IoT client towards the CI. These requests are sent by IoT devices which are represented by C++ threads. The number of IoT devices is set to 1, 10, 100, 500 and 1000. The size of one CoAP_over_TCP post request is 0.109 KB and the size of one CoAP_over_TCP response is 0.087 KB. Each post request contains sensory data in Cbor [18] format which is the data read from IoT devices. λ is set to 15000 (req/sec). When the CI received a CoAP_over_TCP post request:

(1) It forwards the request to the MQ

(2) The MQ extracts sensory data from the payload of the request

(3) The MQ pushes sensory data into corresponding queue in Kakfa publish/subscribe messaging system in order to make it available to subscribers who have subscribed to the same queue

An optimization has been made on producing side is using a single sign-in connection for all IoT devices. Since this optimization does not set at the CI, the evaluation results will not be impacted.

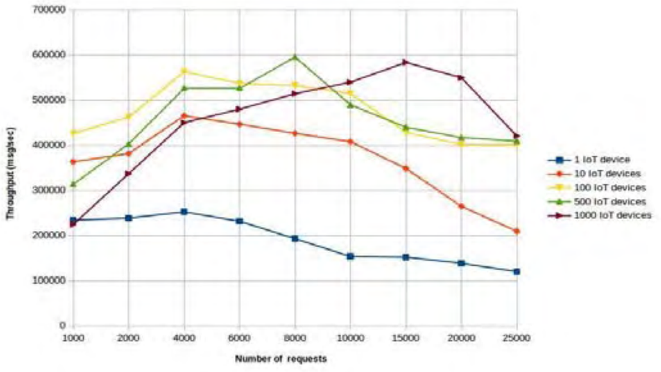


Figure 4. Throughput of CI server

Figure 4 illustrates similar trends of throughputs with different numbers of IoT devices. It gradually increases and reaches the maximum value before starting decreasing as the number of requests continues to increase. The increase of throughput can be justified by the fact that a greater number of requests is introduced into the system and the system uses more CPU utilization for handle those requests. On the other hand, the throughput decreases when the number of requests is large enough and impacts the processing capacity of the system. The throughput increases from 252269.2 (msg/sec) for one IoT device setup to 465519.2 (msg/sec) for 10 IoT devices setup, to 562967.7 (msg/sec) for 100 IoT devices setup and to 595406.2 (msg/sec) for 500 IoT devices setup. In other words, the throughput increases at 343137 (msg/sec) which is almost a 57.63% increase. However, the maximum throughput lightly decreases to 583455.1 (msg/sec) with the 1000 IoT devices setup. The maximum throughput of the CI reaches 595406.2 (msg/sec) for 500 IoT devices setup in the scope of our testbed configuration.

5. Conclusion

This work result can be used to re-evaluate the performance of the platform in a clustered cloud based development. Our test results indicate that the IoTivity Cloud is a stable platform since no drops of messages have been found under the stress testing. Moreover, the testing results show that the CI can handle a load of 1000 IoT devices simultaneously and the maximum throughput reaches 595406.2 (msg/sec).

Our next step is to continue this evaluation work on testing scenario second as aforementioned. Moreover, we will consider the performance evaluation regrading memory consumption of both the CI and the MQ. In addition, we plan to use a number of computers to generate heavy load instead of using a single computer as this testbed did and then compare the results to the results of this work. Finally, we will develop our IoT monitoring platform based on the IoTivity Cloud and re-use this work for performance evaluation.

ACKNOWLEDGEMENT

본 논문은 기초연구사업(NRF-2010-0020210)과 Grand ICT 연구센터지원사업 (IITP-2015-R6812-15-0001)의 연구결과로 수행되었음.

References

[1] K. Ashton, “That ‘Internet of Things’ thing,” *RFID Journal*, June 2009.

[2] L. Atzori, A. Iera, and G. Morabito, “The internet of things: A survey,” *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2010.05.010

[3] C. Alcaraz, P. Najera, J. Lopez, and R. Roman, “Wireless sensor networks and the Internet of Things: Do we need a complete integration?” in *Proceedings of the 1st International Workshop on the Security of the Internet of Things (SecIoT’10), IEEE.* Tokyo (Japan): IEEE, December 2010, [Online]. Available: https://www.nics.uma.es/pub/papers/calcaraz10.pdf

[4] D. Miorandi, S. Sicari, F. D. Pellegrini, and I. Chlamtac, “Internet of Things: Vision, applications and research challenges,” *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497 – 1516, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870512000674

[5] IoTivity software framework. [Online]. Available: https://www.iotivity.org

[6] Open Connectivity Foundation. [Online]. Available: https://openconnec tivity.org

[7] IoTivity Cloud. [Online]. Available: https://wiki.iotivity.org/iotivity_cloud_programming_guide

[8] Bhumi Nakhuva, Tushar Champaneria, “Study of various Internet of Things platforms,” *International Journal of Computer Science & Engineering Survey (IJCSES)*, vol.6, no.6, pp. 61-74, December 2015. doi:10.5121/ijcses.2015.6605.

[9] ThingWorx: Enterprise IoT Solutions and Platform Technology. [Online]. Available: https://www.thingworx.com

[10] ThingSpeak: IoT Analytics. [Online]. Available: https://thingspeak.com

[11] Google Cloud Platform: Google Cloud Computing, Hosting Services & APIs. [Online]. Available: https://cloud.google.com

[12] Vandikas, Konstantinos, and Vlasios Tsiatsis. “Performance Evaluation of an IoT Platform.” In *Proceeding of 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*. IEEE, 2014.

[13] IoT-Framework. [Online]. Available: https://github.com/projectcs13

[14] RFC 7252 Constrained Application Protocol. [Online]. Available: http://coap.technology

[15] The Internet Engineering Task Force (IETF®). [Online]. Available: https://www.ietf.org

[16] OAuth 2.0. [Online]. Available: https://oauth.net/2Zookeeper. [Online]. Available: http://zookeeper.apache.org

[17] Kafka. [Online]. Available: http://kafka.apache.org

[18] Cbor – Concise Binary Object Representation. [Online]. Available: http://cbor.io

[19] Poisson point process. [Online]. Available: https://en.wikipedia.org/wiki /Poisson_point_process

[20] D. E. Knuth. *The Art of Computer Programming Volume 1*. Addison-Weseley Publ. Co., 196