A SDN-based Architecture for Horizontal Internet of Things Services

Yuhong Li
State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications
Beijing, China
Email: hoyli@bupt.edu.cn

Xiang Su, Jukka Riekki
Center for Ubiquitous Computing
University of Oulu
Oulu, Finland
Email: {xiangsu, jpr}@ee.oulu.fi

Theo Kanter, Rahim Rahmani
Department of Computer and Systems Sciences
Stockholm University
Stockholm, Sweden
Email: {kanter, rahim}@dsv.su.se

***Abstract*—The Internet of Things (IoT) architecture is expected to evolve into a horizontal model containing various open systems, integrated environments, and platforms. However, not much research effort has been devoted to developing architectures for horizontal IoT solutions so far. This paper presents an IoT architecture based on Software-Defined Networking (SDN). In this architecture, devices, gateways, and data are open and programmable to IoT application developers and service operators. Moreover, IoT data provision and interoperability are supported at different levels. We present an implementation of the proposed architecture. Our implementation shows that the proposed architecture enables rapid creation of IoT applications by reusing ready applications and data. The measurement and evaluation results demonstrate the feasibility of the proposed architecture.**

***Index Terms*—Internet of Things (IoT); Horizontal solution; Software-Defined Networking(SDN); Future network architecture**

I. Introduction

The development of communication technologies, nano electronics, embedded systems, smart phones, cloud-based networking, and physical objects enable connecting people, places, and things to the Internet anytime and anywhere. This is often referred to as the Internet of Things (IoT). Currently, many applications are being developed for a wide range of IoT scenarios, such as industrial, civilian, and military applications, including healthcare, home automation, earthquake warning, traffic control, and industrial process monitoring. IoT applications in different domains demand their own dedicated platforms and systems. Currently, most IoT services are provided by different platforms and systems, and each platform or system is associated with a certain application domains. This results in dedicated (i.e. vertical) platforms and systems, each containing a number of sensors and actuators, a set of gateways, a computing center, etc. For example, in a smart home, different platforms and applications are built by different providers for appliance systems, home security, and HVAC. Hence, there is a heavy redundancy when considering devices, data, operations, and management of vertical systems.

The benefit of developing dedicated IoT platforms is that it is relatively easy to operate and maintain IoT applications in one domain, which has a limited selection of different types of IoT devices and a computing center deployed for example on the cloud. End users need not to worry about the compatibility of the components. For service providers, it is easy to manage the devices and data of the domain specific applications.

The main challenge of vertical IoT platforms is lack of interoperability that leads to several limitations. First, it is hard to utilize hardware and software resources to their bests. These resources include devices, data, and operation and management functionalities. Users need to install different systems for different applications and services. Each system contains its own sensors, gateways and computing center, although many of the devices could potentially be utilized for other applications and services as well. In addition, IoT service providers need to design or reconfigure the devices and communication protocols for each application and even for each user. Finally, they also have to manage a large amount of redundant data.

Second, introducing new services and innovations is difficult and slow for both service operators and end users. Normally, only service providers can improve and upgrade the existing services. New services and innovations require deploying new sensors and computing centers. Also, connecting new types of physical things to ready systems can be challenging. In summary, vertical dedicated IoT platforms are comparable with proprietary enterprise networks; they both provide limited possibilities for information sharing and interoperability.

The above mentioned limitations motivate research on horizontal IoT platforms and architectures, which can provide a common framework with essential functions and interfaces for a wide range of applications in several domains. In the early research, the focus was mostly on extending the existing IoT platforms by developing separate endpoints, gateways and cloud platforms. The aim was that developers can build their services by selecting different components from different vendors. For instance, several companies have started to roll out cloud platforms and gateway hardware to support a larger amount of users. The Freescale One Box [1] is an example of such a gateway. Eurotech [2] also offers cloud services for developers. EnOcean [3] has recently released an accessory

that turns Raspberry Pi into a gateway for home automation devices. ST Microelectronics has created its SmartHome platform and gateway [4] for IoT developers to leverage their efforts.

Many IoT platforms have also been proposed [5]. We focus on the platforms utilizing Software-Defined Networking (SDN). The early efforts for employing SDN to support policies to manage wireless sensor networks include FlowSensor [6], Sensor OpenFlow [7], and SDWN [8]. Moreover, Yiakoumis et al. [9] reported a home network slicing mechanism, which allows multiple service providers to share a common infrastructure in a smart home. However, many of these efforts are isolated to specific application domains.

Floeck et al. propose a generic Machine-To-Machine (M2M) architecture [10]. They advocate standard based horizontal service platforms in the domain of M2M industry to enable different types of devices to communicate with each other. However, the challenge of interoperability among different data models still remains. Yue et al. suggested the concept of IoT community [11]. Users with the same interests construct a community to facilitate data maintenance, retrieval, and distribution. However, the suggested architecture is neither open to users nor convenient for introducing new IoT applications and services. One recent effort is to design a software-defined approach for IoT environments to dynamically achieve differentiated quality levels to different IoT tasks [12], but here the focus was on developing layered SDN controller in IoT setting and flow QoS performance. The 6TiSCH WG at the IETF has proposed an IPv6 enabled architecture for Industrial IoT applications [13], which is based on SDN. However, the work focuses on supporting the best effort traffic on deterministic TSCH-based networks, which relies on centralized scheduling facilitated by SDN. The developers of oneM2M [14] made efforts on horizontal IoT through standardizing IoT protocols and data models. It addresses the need for a common M2M service layer that can be readily embedded within various hardware and software. We share the same idea with oneM2M by using a common service layer. However, we go one step further by embodying the service interface through the SDN NBI (North-bound Interface).

Some research [15] has advocated that with the development and maturity of distributed intelligent information transmission and processing technologies IoT systems will make intelligent sensing widely available through information sharing and collaboration. These efforts also push the development of a horizontal IoT system.

SDN technologies facilitate building network services [16] [17]. The logically centralized controller and the standardized south bound interfaces provide a basic but powerful architecture to support common functions and interfaces on which a wide range of user applications can be developed rapidly. Moreover, interoperability of data in different formats and models can also be achieved easily and in a dynamically controlled way. Therefore, applications in different domains can be supported easily and efficiently by the same system.

In this paper, we propose an IoT architecture based on SDN technologies for the horizontal IoT services. This architecture enables sharing various resources, including devices, data, and software, among various applications at different levels. Data interoperability can be supported interiorly in the network, and new services and applications in different domains can be supported rapidly and efficiently. We also present a prototype of the proposed architecture using OpenFlow and Open vSwitch. Moreover, we describe measurement results and analyses of running IoT services on the implemented prototype. Finally, we explain how new IoT applications can be created based on the existing ones.

The rest of the paper is organized as follows. In Section II, we elaborate the proposed architecture, including its main components and interfaces. In Section III, we describe the prototype. We present the measurement results and analysis in Section IV, and conclude the paper in Section V.

II. An Architecture for Horizontal IoT

A. Design Principles

Our basic motivation for developing the IoT architecture is to promote the reuse of various resources and to allow the rapid introduction and deployment of new IoT services and applications. Thus, we specify our design principles as follows:

1) Layered architecture. Various types of resources are needed for providing IoT services, such as devices for collecting data, networking resources for transferring the data, and computing resources for processing the data according to user needs. In order to organize and use the resources as a whole, the network architecture should be organized into a layered structure, each layer with clear responsibilities.

2) Openness and programmability. This is the basis for the rapid introduction of new applications and services and the rapid proliferation of new businesses. Thus, the architecture should provide open programmable interfaces, functionalities, and services. In addition, data and methods for processing and obtaining the data should be open as well. In this way, services can be developed rapidly by reusing the same raw data and by annotating different context information, which may lead to more commercial services.

3) Data provision and sharing at different levels. Providing data (environment, people, and things) for services is one of the major requirements set to IoT architecture. Horizontal IoT infrastructures should provide data for a wide range of services in different domains, which demands data at different levels. For example, some users may need only raw data for further analysis and use their own knowledge bases of specific domains, while others may need meaningful information that has been processed utilizing knowledge in the network, for instance, information about weather in a city and conditions of its roads.

4) Interoperability. Heterogeneous devices may produce data in various formats and models. In order to realize and support services in different domains, which may use different data models, patterns and communication protocols, interoperabil-

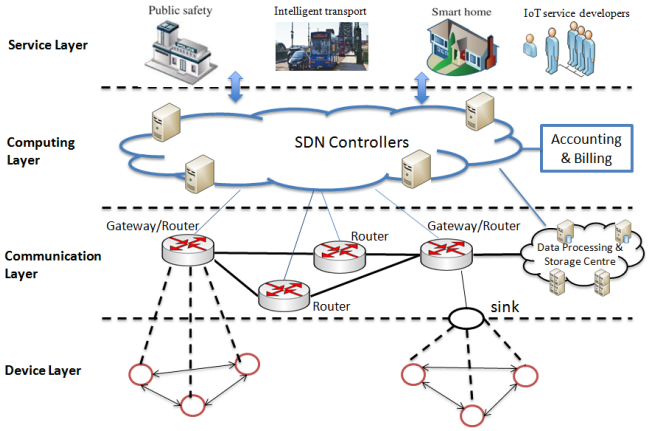

Fig. 1. A layered IoT architecture based on SDN.

ity should be supported in the network from the point of view of devices, data, and communication protocols.

B. A SDN-based IoT Architecture

According to these design principles, we propose an IoT architecture as shown in Fig. 1. This architecture consists of four layers. The lowest layer is the device layer, which contains sensors collecting a large amount of data in different formats and potentially for different IoT application domains. Some devices can also act as actuators receiving commands from the network and performing tasks. The communication layer is comprised of SDN gateways and routers, which can forward data under the control of the SDN controller. The computing layer contains SDN controllers and the accounting and billing mechanisms. They control the forwarding of data according to the requirements of applications. The service developers and operators build IoT services at the service layer by programming the SDN controllers.

According to the network scale and existing deployments, there can be one central SDN controller or several controllers implementing the control functions cooperatively. We focus on the horizontal IoT architecture and inter-working among distributed SDN controllers is out of the scope of this paper. In this paper, we follow the philosophy of SDN, in other words, the controllers can be deployed in a physically distributed fashion but they are logically centralized.

The functions of the major components in the architecture are the following:

SDN controllers. The controllers control not only data forwarding, but also processing of data. As shown in Fig. 2, SDN controllers have the following functions:

- Equipment management, such as configuring gateways/routers, virtual network resources, policies and rules for processing data in the corresponding devices etc.
- IoT service management, such as adding, modifying and deleting services supported by gateways, storing and caching policies, and updating rules and algorithms for Data Processing & Storage center.
- Topology management, such as routing calculation and updating topology.
- Operation and maintenance (O&M), such as maintaining operating logs, monitoring user interfaces, alarming, and managing functional modules.
- Security management, such as detecting conflicts and authenticating service access.
- Implementing SDN south bound (SB) and north bound (NB) interfaces.

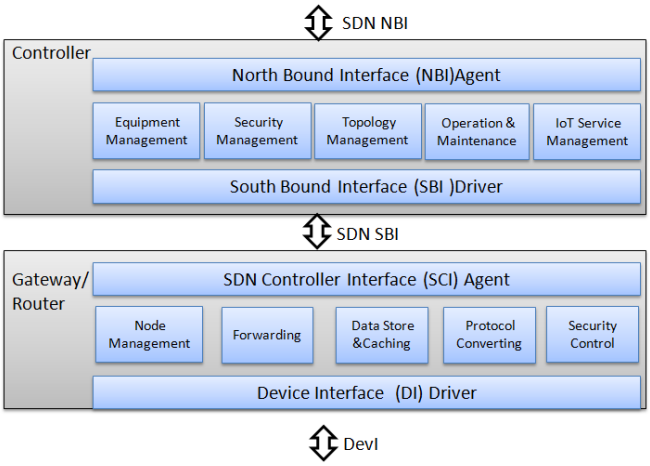

Fig. 2. Functional modules of controller and gateways.

Gateways/Routers. As shown in Fig. 2, gateways/routers are responsible for data forwarding in the networks. Besides forwarding data, gateways can store or cache local data, or process data under the instruction of SDN controllers. Other functions for gateways/routers include node management, protocol converting, and security control. In addition, simple data processing methods or rules can also be downloaded to the gateways. Data can be processed according to the selected method and cached locally.

In general, the following functions for processing data are needed:

- Functions for interacting with local IoT devices (to obtain data) and for forwarding the data to remote gateways or data processing centers for further analysis or permanent storage.
- Functions for local analysis and processing of the data acquired by IoT devices.
- Functions for interacting with remote IoT devices.
- Functions for application specific data analysis and processing.

Data Processing & Storage Center. The data obtained from IoT devices and sinks in the networks can be selectively stored in this module as instructed by the controller. Moreover, mechanisms and algorithms for processing data such as data format converting, data mining and reasoning can be performed in this component through the controller. Since the task of this component is to provide the desired data in the desired format for users, just like the task of the gateways, this component belongs to the communication layer.

Sinks. They are responsible for aggregating and caching the data obtained from IoT devices. Unlike gateways, they cannot be programmed by SDN controllers. However, depending on their capabilities, they can perform simple processing, such as eliminating some redundant data obtained from the sensors.

Accounting and Billing Center. Different from the traditional networking services, which consume mainly the network bandwidth, IoT services consume also computing and storage resources. In addition, new services can also be provided on demand, which may require support from service developers. Thus, new mechanisms for accounting and billing should be considered. In general, accounting and billing can be based on consumed time, data amount and services used by applications. However, the prerequisite is that gateways/routers and controllers can provide accurate mechanisms for measuring the use of different types of resources. Billing policies are out of the scope of this paper. Nevertheless, we do consider mechanisms for measuring and recording the consumed resources.

Information for instructing how controllers should control gateways/routers is passed through the SDN North Bound Interface (SDN NBI). Generally, the following information should be carried through SDN NBI:

- IoT service logic and its operations, such as programs, algorithms and rules for new services and data processing; modifying, deleting and querying service operation, etc.
- Mechanisms and policies for data storage and caching. For instance, where and what kind of data should be cached or stored; what data should be stored in the Data Processing & Storage Center, etc.
- Policies related with interoperability, security, accounting, and billing, etc.
- Information related with the operation and maintenance of the networking equipments, including controllers, such as logs, alarms and other functions defined by ONF [18].

The south bound interface mainly realizes the dynamic request and response paths between the controller and the routers/gateways. Moreover, it is used to configure the routers. For these purposes, standard protocols, such as OpenFlow can be used. However, since some of the devices store and cache data, and may need to support different data formats for interoperability, the OpenFlow protocol needs to be extended.

III. Implementation

To validate the proposed IoT architecture, we implemented the main modules of the architecture and analysed their performance. The modules have been installed in both real devices and virtual machines. A test network has been set up and several test services have run in the test network. Performance evaluations have also been done on the virtual machines based on the implementation.

The controller is implemented based on POX, an open platform for the rapid development and prototyping of network control software and a framework for interacting with OpenFlow switches [19]. Besides using the functions provided by POX, such as network topology maintenance, routing path calculation, interacting with switches through OpenFlow protocol, we implemented the forwarding function on POX, which can configure the flow tables in Routers/Gateways according to the instructions from the IoT control plane applications and the data storing and caching in the Gateways and Data Processing & Storage Center.

The Gateways/Routers are implemented based on Open vSwitch [20], a virtual switch supporting OpenFlow among other features. We implemented the Data Store & Caching and Node Management module. Collected data can be stored in the gateways and be processed according to the algorithms written by the IoT developers and distributed to the gateways on demand. In our implementation and tests, we use Open vSwitch 2.3.0. The gateways and routers are configured with the same functions, in other words, they all can route data and can store sensed data.

The SDN north bound data exchange is implemented using JSON. The IoT service logics are encapsulated using JSON. Fig. 3 shows the data fields currently implemented in our test network. Among them, the field Duration means how long the service will last. Effective Time determines when a service will be in operation, for example, immediately or after a specified time.

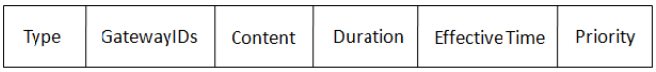

Fig. 3. Data fields of JSON through SDN north bound interface.

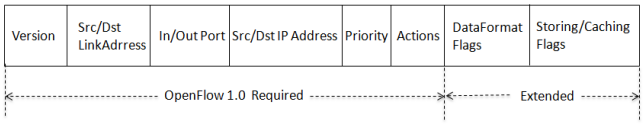

Fig. 4. Data fields of OpenFlow through SDN south bound interface.

In our prototype, we use OpenFlow 1.0 to realize the SDN south bound interface. Fig. 4 illustrates the fields of the extended OpenFlow protocol. Here, besides the standard OpenFlow fields, two fields are added to instruct what type of data format the gateway should use to transfer the data, and if the data should be stored or cached in the corresponding gateway.

IV. Tests and Evaluations

A. Test Scenarios and Results

To illustrate the IoT services provided by our IoT architecture, we setup a test network in our laboratory and run several IoT applications in different scenarios.

Fig. 5 presents the configuration of the test network. Raspberry Pis are sinks (see Fig. 1) and they are configured through the connecting Gateways. The beacons and students' phones are the devices in the network. The beacons 1, 2, and 3 are deployed near a door and inside a laboratory. The beacons 4, 5, and 6 are deployed near a door and inside a meeting room. The beacons are initialized by the corresponding Raspberry

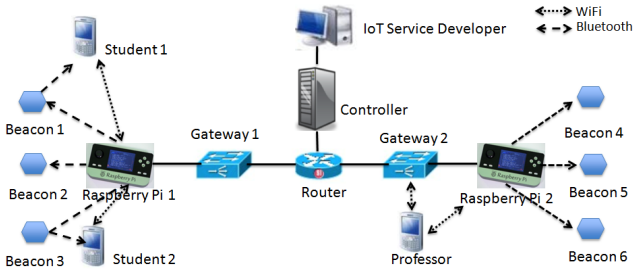

Fig. 5. The configuration of the test network.

Pis. Beacons are connected to the corresponding Raspberry Pis and smart mobile phones through Bluetooth. Smart mobile phones can also connect to Raspberry Pis through WiFi.

In the first scenario, a "lab check-in" service can be provided by the test network. This service has been implemented as follows. When a student approaches the laboratory, a map will be popped on his mobile phone prompting him to finish the checking in to the lab. Several alternative beacons will be illustrated that the student can use to connect to the network, including not only those that he can connect to (the pink one in Fig. 6 (a)), but also those that are detected but with no connection currently available (the green one in Fig. 6 (a)). Next, he can select one beacon with connections, input his name and ID, and finish his checking in, as shown in Fig. 6 (b). All the data are stored in Raspberry Pi 1.

A professor can check at any time which students are in the laboratory by using his mobile phone. The mobile phone can connect to the network through, e.g., Raspberry Pi 2. In this situation, a message RequestLab (check_in) is sent to the Raspberry Pi 2. The path Raspberry Pi 2 $\Rightarrow$ Gateway 2 $\Rightarrow$ Router already exists in Raspberry Pi 2 and Gateway 2, but the router does not know how to reach Raspberry Pi 1. Therefore, the message is forwarded to the controller, and the path Router $\Rightarrow$ Gateway 1 $\Rightarrow$ Raspberry Pi 1 is found. Finally, the check-in list is sent back to the professor, as shown in Fig. 6 (c). For the purpose of demonstration, only a name list is returned.

In the second scenario, a "meeting room booking" service can be provided by the test network. The service has been implemented as follows: When someone enters the meeting room, an interface will be popped on his mobile phone requesting him to book the meeting room. Then, the booking information, including his name and the period of the meeting, is stored locally in the Raspberry Pi 2.

In the third scenario, a professor thinks he may have a meeting with students when he has time and more than ten students are in the laboratory. What he can do now is to use the "lab check-in" service to count how many students are in the lab according to the check-in list, and to use the "meeting room book" service to find if a meeting room is available. But he thinks it is too complex to regularly check the list of students in the laboratory, and also whether somebody is in the meeting room and when the meeting will be finished. Therefore, he discusses with the IoT service developer about such a service. The service developer finds that there is no need to deploy any new beacon, since from the "check-in list" in the Raspberry Pi 1 the number of the students in the lab can be calculated and obtained, and from the "meeting room booking" states, if the available meeting room status can be calculated.

Therefore, the IoT service developer develops applications of counting the student number from the "check-in list" and checking availability of the meeting room from the meeting room booking, and loading them to Raspberry Pi 1 and 2, respectively. Another service MeetingPossibility (StudentNum, MeetingRoom), which makes the decision whether the meeting can be organized according to the number of the students and the availability of the meeting room is developed and loaded to the Router through the Controller. Now, everybody can use this new IoT service without deploying new sensors.

Now, a new service can be provided by the network. When a request for checking meeting possibility is sent to the router, the router will first fetch the number of the students and meeting room availability from Raspberry Pi 1 and 2, respectively, and check if the meeting is possible to organize and notify the results directly to the professor.

We have measured three latencies with real devices during the experiment in the above scenarios. First, the time for loading a new service from the Controller to the Router (i.e., the Meeting Possibility) is 425ms (average value of ten experiments). This loading time includes the time used by the Controller for resolving JSON, interpreting and sending it to the Router, and the time used by the Router to execute it to create the corresponding configuration files. Second, the time for the Router to request a new path from the Controller and get a response is 57ms (average value of 10 experiments). This includes the time for the Router to send the request to the Controller, the time used for the Controller to calculate the path and then send it to the Router, and the time used for the Router to install it in the Flow Table and use the Flow Table for the first time. Third, the time for transmitting data request from Raspberry Pi 2 to Raspberry Pi 1, then transmitting the data back to Raspberry Pi 2 is 0.81ms (average value of 10 experiments).

From the test scenarios and measurements, we notice that by using the proposed architecture, new services can be introduced and deployed rapidly through reusing the existing services and data. In addition, it is feasible to use SDN technique for the IoT services.

B. Performance Evaluation

We evaluate the proposed IoT architecture in terms of Round-Trip Time (RTT) and packet loss rate when a IoT services exists and the RTT when a new IoT service is introduced.

In our experiments, one controller, eleven gateways/ routers and nine hosts (PC) are installed as virtual machines in a server (ubuntu 14.04.2 LTS). The network topology is shown in Fig. 7. RTT is measured for packet sending from a sending host (any of h1 to h9) to a receiving host (any of h1 to h9 except

(a) (b) (c)

Fig. 6. Example of laboratory check-in service. (a) Information prompting a student to check-in when near the door. (b) Information on the student's mobile phone after check-in. (c) Check-in list visuable for a professor.

the sending host) and back to the sending host when, a) there is already an existing path between the two hosts, and b) there is no existing path, in order to see the performance of the SDN controller.

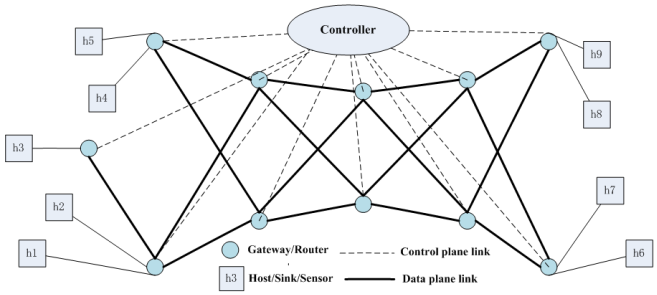

Fig. 7. Network topology for evaluation.

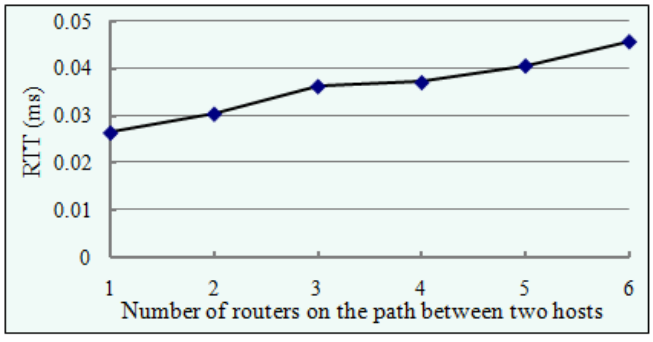

Fig. 8. RTT when there exist paths between two hosts.

Fig. 8 shows the average RTT for already existing paths between two hosts as a function of the number of routers on the path. Fig. 9 shows the average RTT when there are no existing path between two hosts as a function of the number of routers on the path. In this case, the first gateway (i.e., the gateway closest to the sending host) sends the packet forwarding request to the controller. The controller calculates the corresponding path and sends back an OpenFlow message to configure the Flow Table. Then, the packet is forwarded to the next router based on the Flow Table. Note that the controller sends an OpenFlow message to each node along the calculated path from the source to the destination. Therefore, when the packet arrives at the second router, there is already an entry in the Flow Table for it, thus the packet can be forwarded immediately. If the Flow Table has not been configured due to some delays in the network, a forwarding request will be sent to the controller in the second router. However, the controller does not need to calculate the path this time, since it already maintains this information due to the request from the first gateway. Thus, the result is sent back to the second gateway immediately. Since all the routers along the path are configured proactively, the increasing of the routers' number will not cause notable difference in RTT, as shown in Fig. 9.

Fig. 10 shows the packet loss rate when an IoT service sends packets at different rates and there are already paths between the sender and the receiver. In this experiment, the network bandwidth is fixed at 300M bit per second, and the application sends data at different rates for ten seconds. We have repeated the tests when no path between the two hosts exists when the service began to run. In other words, the controller is involved in finding the path. However, the results are similar, as shown in Fig. 10. This is because the controller did not cause much packet loss.

V. Conclusions and Future Work

In this paper, we proposed an IoT architecture based on SDN for horizontal IoT services. This architecture achieved

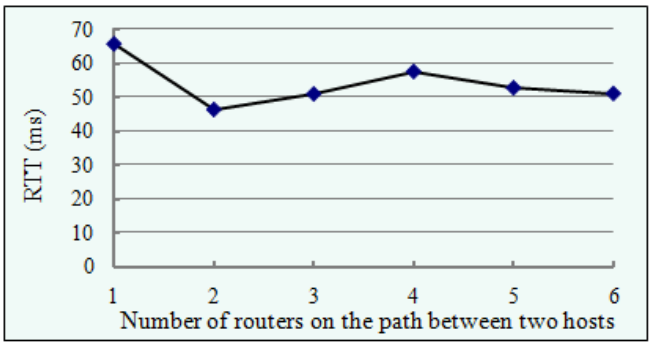

Fig. 9. RTT when no paths exist between two hosts.

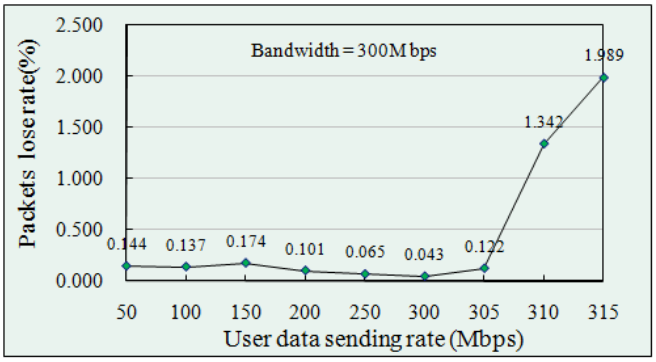

Fig. 10. Packet loss rate.

our design goals, i.e. developing a layered architecture with open and programmable devices and data at different levels. We reported the design principles and architecture in detail, presented our implementation, three test scenarios, and performance evaluations.

The horizontal IoT solution based on the proposed architecture aims at supporting multiple services with different scenarios and in different domains. The proposed architecture enables new IoT services to be provided rapidly. In addition, both IoT devices and IoT data can be reused among multiple services. The implementation and evaluation demonstrate the feasibility of our approach.

Our proposed architecture does not provide any extra security mechanism. However, security is one of the foundation issues when SDN is designed. Security mechanisms could be applied to protect the SDN controller, establish trust among entities, and create a robust policy mechanism. Moreover, security as a service could be delivered to protect the availability, integrity, and privacy of all connected resources and information in the IoT architecture.

As future work, our plan is to implement more functions and algorithms on SDN controllers and gateways, for example, calculating routing paths by considering the caching in the gateways and some security strategies. In addition, we will implement more complex IoT scenarios (e.g. related to smart cities) with the integration of multiple information sources and perform evaluation to validate our architecture. We will also study horizontally interworking between different domains by enabling nodes to be context-aware.

ACKNOWLEDGMENT

This work was partly supported by TEKES as part of the Internet of Things program of DIGILE (Finnish Strategic Center for Science, Technology and Innovation in the field of ICT and digital business). Special thanks to Congliang Zhao, Ming Yao, Yuanyuan Huang, Wenchao Liu, Han Zheng, Xiaoyu Hao, master students at the State Key Laboratory of Networking and Switching Technologies, Beijing University of Posts and Telecommunications, who have implemented the prototype and testbed. Xiang Su thanks the funding from Tekniikan edistämissäätiö, Tauno Tönningin säätiö, and Walter Ahlströmin säätiö.

REFERENCES

[1] Onebox, 2016. [Online]. Available: www.onebox.com/
[2] Eutech, 2016. [Online]. Available: http://www.eurotech.com/en/
[3] EnOceanOrg, 2016. [Online]. Available: http://www.enocean.org/
[4] ST microelectronics, 2016. [Online]. Available: http://www.st.com/web/en/home.html
[5] J. Mineraud, O. Mazhelis, X. Su and S. Tarkoma, "A gap analysis of Internet-of-Things platforms," Comput. Commun., in press.
[6] A. Mahmud and R. Rahmani, "Exploitation of openflow in wireless sensor networks," in *Proc. of the 2011 International Conference on Computer Science and Network Technology*. IEEE, 2011, pp. 594-600.
[7] T. Luo, H. Tan, and T. Q. S. Quek, "Sensor openflow: Enabling software-defined wireless sensor networks," IEEE Commun. Lett., vol. 16, no. 11, pp. 1896–1899, 2012.
[8] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software defined wireless networks: Unbridling SDNs," in *Proc. 2012 European Workshop on Software Defined Networking.* IEEE, 2012, pp. 1–6.
[9] Y. Yiakoumis, K. Yap, S. Katti, G. Parulkar, and N. McKeown, "Slicing home networks," in *Proc. of the 2nd ACM SIGCOMM workshop on Home networks,* ACM, 2011, pp. 1–6.
[10] M. Floeck, A. Papageorgious, and A. Schuelke, "Horizontal M2M platforms boost vertical industry: effectiveness study for building energy management systems," in *Proc. of 2014 IEEE World Forum on Internet of Things.* IEEE, 2014, pp. 15–20.
[11] H. Yue, L. Guo, R. Li, H. Asaeda, and Y. Fang, "DataClouds: Enabling Community-Based Data-Centric Services Over the Internet of Things," IEEE Internet of Things Journal, vol. 1, No. 5, pp. 472–482, 2014.
[12] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, "A Software Defined Networking Architecture for the Internet-of-Things." in *Proc. 2014 IEEE/IFIP Network Operations and Management Symposium,* IEEE, 2014, pp. 1–9.
[13] P. Thubert, M.R. Palattella, and T. Engel, "6TiSCH Centralized Scheduling: when SDN Meet IoT," in Proc. of *IEEE Conf. on Standards for Communications and Networking, Tokyo,* IEEE, 2015, pp. 42–47.
[14] oneM2M - Standards for M2M and the Internet of Things, 2016. [Online]. Available: http://www.onem2m.org/
[15] O. Vermesan and P. Friess (Eds), "Internet of Things From Research and Innovation to Market Deployment," River Publishers Series in Communication, 2014.
[16] N. McKeown, T. Anderson, H. Balakrshman, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Tuner, "OpenFlow: enabling innovation in campus networks," Sigcomm Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, 2008.
[17] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," IEEE Commun. Surveys Tuts.,vol.16, no. 3, pp. 1617–1634, 2014.
[18] Open networking foundation, 2016. [Online]. Available: https://www.opennetworking.org/
[19] POX, 2016. [Online]. Available: http://www.noxrepo.org/pox/about-pox/
[20] Open vSwitch, 2016. [Online]. Available: http://openvswitch.org/