Indian Journal of Science and Technology, Vol 9(24), DOI: 10.17485/ijst/2016/v9i24/96155, June 2016
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

A Method for Ensuring Stable IoT Services based on SDN

Joonseok Park[1], Mikyeong Moon[2*] and Keunhyuk Yeom[3]

[1]Research Institute of Logistics Innovation and Networking, Pusan National University, Busan, 46241, Korea; pjs50@pusan.ac.kr
[2]Division of Computer Engineering, Dongseo University, Busan, 47011, Korea; mkmoon@dongseo.ac.kr
[3]Computer Science and Engineering, Pusan National University, Busan, 46241, Korea; yeom@pusan.ac.kr

Abstract

Background/Objectives: When an IoT service failure occurs, it is difficult to identify the source of the problem. In order to ensure stale IoT services, error causes need to be identified and the flow of interconnection services controlled. **Methods/Statistical Analysis**: Two types of requirements, namely, platform-device direct connections, and platform-device network connections, are specified for stable IoT services. A SDN based architecture using REST APIs that combines the analysis results in accordance with the requirements of both connection types is represented. **Findings:** In this paper, a method for providing stable IoT services was presented. The efficacy of the proposed method was verified via simulation using a floodlight SDN controller to the SDN adaptor of proposed architecture in a case study. Network errors and the causes of IOT service faults can be easily identified using our proposed architecture and API. **Application/Improvements:** The proposed architecture can be applied to manage errors in IoT services. Further, it can be used with a reference architecture to instantiate systems that provide stable IoT services.

Keywords: Internet of Things (IoT), IoT Management, IoT Service, Reliable IoT Service, Software Defined Network (SDN)

1. Introduction

The IoT (Internet of Things) paradigm[1–3] consists of three hierarchical layers, i.e., IoT services, IoT platform, and IoT devices, as shown in Figure 1. These layers communicate via the Internet. The top layer, IoT services, realizes and facilitates the provision of business values to external objects. The middle layer, IoT platform, supports execution of IoT services and intermediates the processing and communications between IoT services and devices. The bottom layer consists of IoT devices, which are physical objects that acquire real-world data.

In the IoT paradigms, IoT services utilize various combined IoT devices. This means that various IoT services can be implemented using the information acquired by numerous IoT devices. However, IoT service defects can occur based on problems in the interconnected environments and IoT devices. Consequently, a monitoring and control mechanism that facilitates the provision and reception of stable IoT services is essential.

In IoT environment, developers and users cannot quickly detect problems that occur with IoT services. These problems include unexpected errors and faults, such as device failures and network problems. However, when they occur, it is difficult to identify their source. Further, there is currently much debate about how to ensure QoS (Quality of Service) in IoT, how to detect IoT service faults, and how to monitor the status of IoT services.

The SDN (Software Defined Network) concept[4–6], in which network flow can be controlled and information on the status of devices acquired via software, has recently been introduced. The key characteristic of SDN is facilitation of programmable networks. SDN facilitates network control operations via the northbound and southbound APIs of SDN controllers. Further, it facilitates

*Author for correspondence

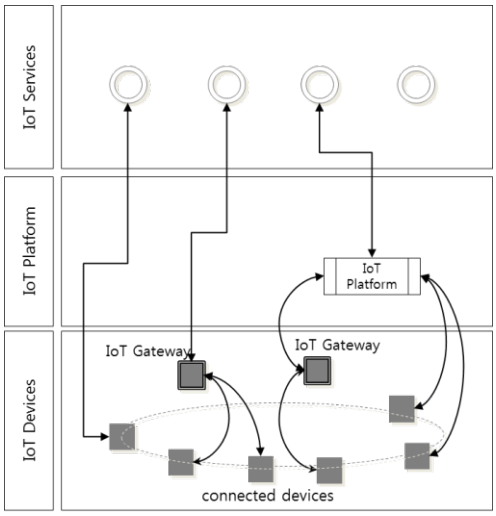

Figure 1. IoT layers.

the collection of statistics about network status by using LLDP (Link Layer Discovery Protocol)[7]. Thus, it can help to detect network topology and the status of each network device such as network port and destination. In addition, an SDN controller can be used to define network control rules such as change of destination when a network problem occurs.

In this study, we apply the SDN concept to control and monitor the status of IoT services. Further, to ensure stable IoT services, an architecture that classifies two types of connections for stable IoT services and helps in the management of IoT service errors is proposed.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 introduces the identified requirements and the proposed architecture. Section 4 presents a case study in which the proposed architecture was applied. Section 5 concludes this paper and outlines future work.

2. Related Work

Much research is being conducted on IoT QoS (Quality of Service) and to date researchers have made related proposals such as a three-layer QoS model[8], QoS-aware computational method[9], and a context-based network infrastructure[10]. QoS refer to the provision of preferential delivery service for applications that need it by ensuring sufficient bandwidth, controlling latency and jitter, and reducing data loss.

The proposed three-layer QoS model utilizes a mechanism that schedules the QoS attribute at the application, network, and sensing layers. The application layer schedules the services based on the QoS and transfers the resource request to the QoS model in the network layer. The network layer allocates network resources to the selected services. The sensing layer selects the applicable sensing devices. This approach is focused on the optimization of attributes of QoS in IoT. It is a top-down method that starts from the QoS concerns of service and divides the IoT layers. Our approach also utilizes the notion of layers. However, it is bottom-up and focuses on identification and management of service failures.

The proposed QoS-aware computational method comprises four basic modes: Serial mode, parallel mode, branch mode, and circulation mode. The method can be used to calculate IoT QoS. In serial mode, the services perform sequentially based on a certain order. In parallel mode, tasks are executed simultaneously. In branch mode, the system can complete the task following the completion of successful service detection. In circulation mode, the services execute in series and each is performed based on a given cycle time. This approach decomposes the total QoS metrics of composite IoT services into their simple constituent services and calculates the corresponding range of indicators. From an IoT services aspects, this approach focuses on quality analysis of composite IoT service. Whereas our approach focuses on requirement analysis and actual fault detection in IoT services.

The proposed context-based network infrastructure is composed of five components: Context manager, virtualization manager, SDN-based network manager, multi-network access mobility manager, and trust manager. The context manager extracts context (e.g., device location, user's activity) and delivers it to the virtualization manger and the SDN-based network manager. The virtualization manger creates a virtualized network instance. The SDN-based network manager performs traffic engineering, load balancing, and QoS administration. The multi-network access mobility manager performs context-based connection for the self-organizing group network. This research is focused on context-awareness in order to support user-centric IoT services. However, it is still ongoing and provides conceptual elements for network infrastructure. Our approach applies the concept of SDN to detect and solve network connection problems. Further, it defines primary requirements, APIs, and explicit architecture elements to ensure the provision of stable services.

3. Our Approach

Two types of connection are specified for stable IoT services: platform-device direct connections and platform-device network connections. A single architecture that combines the analysis results in accordance with the requirements of both connection types is proposed.

3.1 Requirements

IoT services run on a platform by using IoT device information. Two types of monitoring requirements are specified based on the relationships among the IoT layers shown in Figure 1. The focus is on the connection between the platform and each device. The first type of requirement is called platform-device direct connections requirement. This requirement identifies whether a device is stable or not. It is divided into four categories: communication, connectivity, data flow stability, and data reliability, as shown in Table 1.

- Communication indicates the monitoring behavior of the device protocol. The communication coverage, data rate, and power consumptions of devices differ based on the protocol employed.
- Connectivity refers to the activation status of the current device. When a device is not connected, IoT services generate inappropriate values or do not operate.
- Data flow stability indicates whether the device transmits information reliably. Data flow stability can be tested by changing the device data transmission period.
- Data reliability indicates whether the transmitted information has errors. Each device has a defined range of values. By checking the data range, the current error status can be determined.

Table 1. Platform-device direct connections requirement categories

Type	Description
Communication	Monitors device protocol, such as ZigBee, Zwave, and IP-based.
Connectivity	Indicates whether the current device is activated.
Data Flow stability	Indicates whether the device provides stable information, such as setting device period.
Data Reliabiity	Indicates whether the device produces a range or reasonable values.

Table 2. Platform-device network connections requirement categories

Type	Description
Topology	Indicates the device's connection topology.
Throughput	Indicates bandwidth and packet transmission.
Routing	Indicates port, and start and end points of each device.
Control	Checks current status of the device and hosts and defines rules.

The second requirement type is called the platform-device network connection requirement. This requirement is used to ascertain the problem in the network status of a device. It is divided into four categories: Topology, throughput, routing, and control, as shown in Table 2.

- Topology indicates how the device is connected. The path through which the device information flows can be identified by checking the network topology and the device connection.
- Throughput shows how many packets are transmitted from each device and indicates the status of the switch bandwidth. This information can be used to alert or calculate the delay or bottleneck in device traffic.
- Routing indicates the destination of each device. When an IoT service error occurs, the routing information of each device can be traced and problem at the network level detected.
- Control checks and acquires the real-time traffic information of each host and switch. The defined rule can be changed when suspect data flows are identified.

3.2 Architecture

We also propose an architecture that supports the identified requirements. Figure 2 shows the proposed architecture and its constituent elements. The architectural elements are as follows:

- IoT network access interface: Shows access points composed of monitoring functions.
- Device profile manager: Creates, reads, updates, and deletes device information, including supported protocol.
- Device connection manager: Checks device connection status and sets device information transmission cycle.

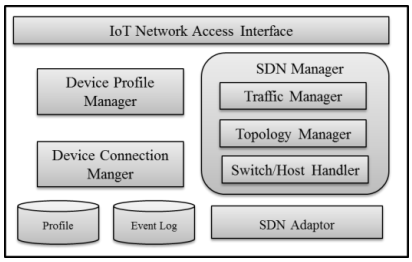

Figure 2. Proposed architecture.

- Traffic manager: Checks throughput and routing, and calculates traffic information, such as delay, bottleneck.
- Topology manager: Views and lists the topology of all devices.
- Switch/Host handler: Retrieves real-time information for switch and host, and changes defined rules.
- SDN adapter: Presents the binding points where various SDN controllers can be adapted.
- Profile: States the database where device profile information is stored.
- Event Log: Denotes the database where all network-related events, such as transmission rate and bandwidth, are stored.

Table 3 shows the defined REST (Representational State Transfer)[11] APIs provided to identify error causes and control interconnection flow. In Table 3, the API list column gives the ID and a brief description of each API's role. The API description column consists of method type, API, and related requirement type sub-columns. The method type column shows the REST API type, such as get, put, post, delete. The API column denotes the API call structure. The result is obtained by assigning a value for uid in the <uid> section. The related requirement type column lists the requirement supported by the API.

For example, to view a device's port information by using the API denoted ID-8, the API can be called using /api/view_statistic_info/port/json. As shown in Figure 3, the result is then displayed in JSON (JavaScript Object Notation)[12] format. The result shows the device's port status detail, such as port number, number of received packets, and number of transmitted packets. Using this information, we can identify the network problem by comparing the assigned value with the defined metric assigned minimum and maximum values.

In the architecture, IoT access interfaces are realized using the APIs defined in Table 3. Table 4 shows the identified architecture elements and related requirements type. As shown in the table, each architectural element is defined to support the identified requirements.

4. Case study and Evaluation

We applied our approach to develop IoT services to measure air pollution. Figure 4 shows the structure of the system used in our case study.

Table 3. API list example

API List	API Description		
	Method Type	API	Related Requirement Type
ID-1: Retrive device list	Get	/api/get_device_list/<uid>/json	Communication
ID-2: Retrive device protocol	Get	/api/get_device_protolcol/<uid>/json	Communication
ID-3: Check device status	Get	/api/get_device_status/<uid>/json	Connectivity
ID-4: Update data transmission period	Put	/api/set_device_ period/<uid>/<period>/json	Connectivity
ID-5: Retrive device value	Get	/api/get_value/<uid>/<date>/json	Data Flow Stability
ID-6: Retrive device sensor value	Get	/api/get_value/<uid>/<sid>/json	Data Reliability
ID-7: View topology	Get	/api/view_topology/json	Topology
ID-8: Get device statistical data	Get	/api/view_statistic_info/<statType>/ json	Throughput
ID-9: Retreive device network information	Get	/api/view_network_info/<uid>/json	Throughput
ID-10: Retreive device route information	Get	/api/view_network_info/<src-id>/<src-port>/<des-id>/<des-port>	Routing
ID-11: Delete device flow information	Delete	/api/del_network_flow/<uid>/json	Control

```
{
  "portNumber": "1",
  "receivePackets": "17",
  "transmitPackets": "1639",
  "receiveBytes": "1230",
  "transmitBytes": "270828",
  "receiveDropped": "0",
  "transmitDropped": "1",
  "receiveErrors": "0",
  "transmitErrors": "0",
  "receiveFrameErrors": "0",
  "receiveOverrunErrors": "0",
  "receiveCRCErrors": "0",
  "collisions": "0"
}
```


Figure 3. Exmple results of an API call.

Table 4. Requiremnt type and architecure elements

Requirements	Architecture Elements
Communicatioin	Device Profile Manager, Profile
Connectivity	Device Connection Manager, Event Log
Data Flow Statbility	Device Connection Manager, Event Log
Data Reliabity	Device Profile Manager, Device Connection Manager, Profile
Topology	SDN Adaptor, Topology Manager
Throughput	SDN Adaptor, Switch/Host Handler
Routing	SDN Adaptor, Switch/Host Handler, Traffic Manger
Control	SDN Adaptor, Switch/Host Handler, Traffic Manger

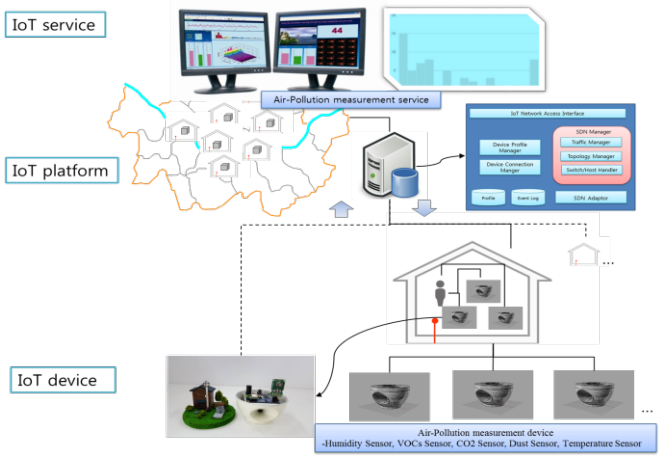

Figure 4. Structure of the system used in our case study.

Table 5. Sensor specification

Sensor	Measurement Specification
Humidity	0–100% RH
VOCs	Up to 100 ppm
CO_2	400–5,000 ppm
Dust	0–0.5 mg/m^3
Temperature	-40–120 °C

As shown in Figure 4, the air-pollution measurement services provide information on air-quality data such as temperature, humidity, carbon dioxide, volatile organic compounds, and dust. IoT devices were developed to acquire these air-quality data.

Table 5 shows the specifications for the measurement sensors. As shown in Figure 4, air-pollution measurement devices can be placed wherever a user located. They can measure the air quality in real time. The measured air-quality data are transmitted to the IoT platform. IoT services that provide region-, zone- and time-specific air-pollution analysis and report are operated using the air-quality data passed from the IoT platform. To apply our approach for the provision of stable IoT services, we implemented a platform dashboard prototype in accordance with the proposed architecture and APIs. Figure 5 shows the interface supporting the platform-device direct connection requirement.

In Figure 5, the graph in the upper left, for example, shows the data flow stability requirement type. The ID-5 API shown in Table 3 is used. The bottom left of the figure shows the connectivity and data reliability obtained using the ID-4 and ID-6 APIs in Table 3. In addition, it shows the accumulation of quality measurement data for each region (upper right) and air-quality monitoring data for a specific region (bottom right) by using the ID-6 API.

Figure 6 and 7 show the interface for the platform-device network connection requirement. In this case study, we bound the floodlight[13] SDN controller to the SDN adaptor of proposed architecture and tested our network environment by using the mininet[14] network simulator. Figure 6 shows the topology status obtained using the ID-7 API.

Figure 7 shows a network problem detected at the platform-device network connection level. The upper left part of the figure visualizes the current device's network status by displaying the throughput among switches by using throughput requirement type and the ID-8 and ID-9 APIs. The network status is also checked based on the defined interval. When an error occurs, the device's IP address, error type, and occurrence time are displayed, as shown in upper right corner, using the ID-8 API. The link topology is also checked and displayed, as shown in the bottom left part, using the ID-7 API that is, topology requirements

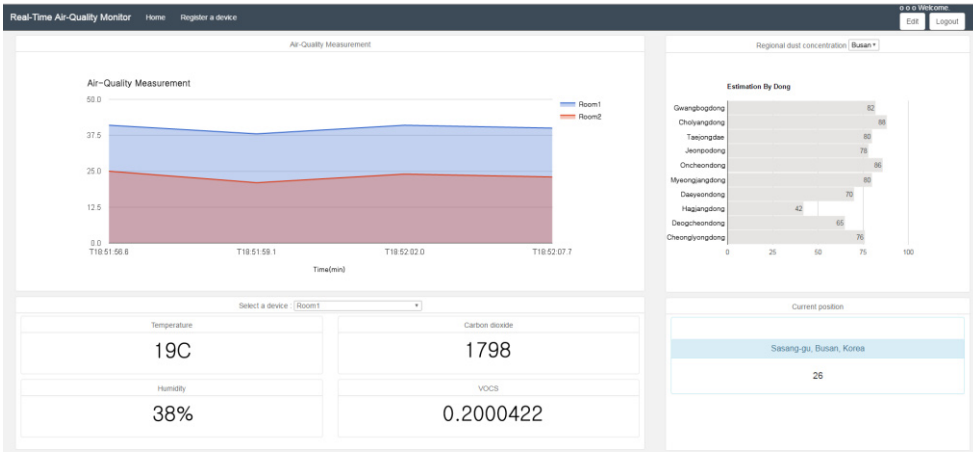

Figure 5. Platform-device direct connection requirement interface.

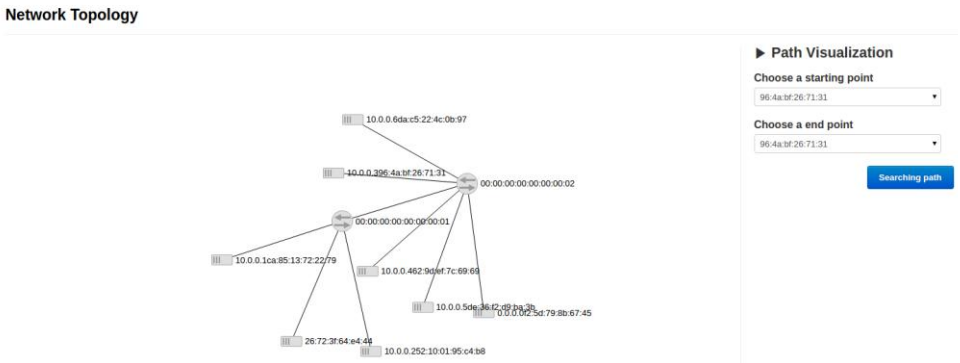

Figure 6. Toplogy interface for platform-device network connection requirement.

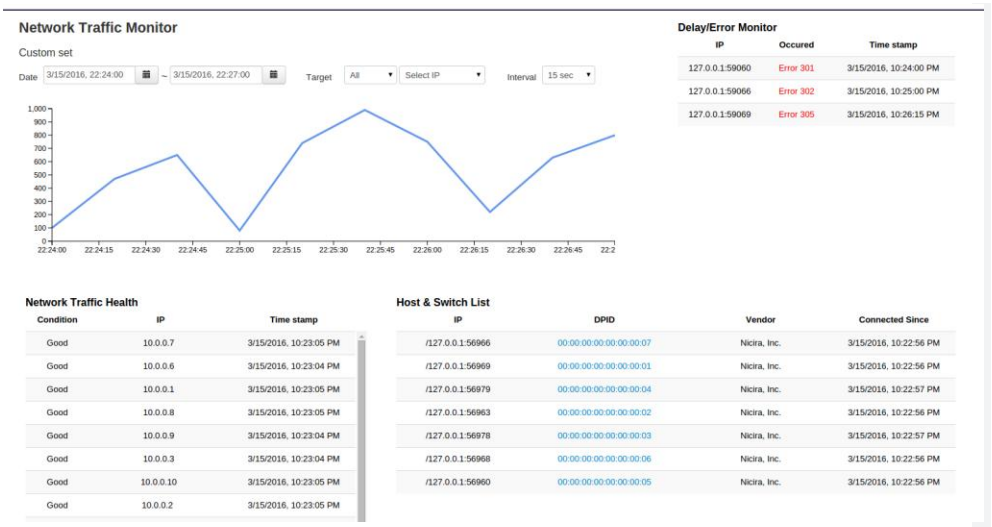

Figure 7. Interface for network-connection aspects.

type in Table 3. The display shows whether the device has failed or not. As shown in the bottom right corner, host and switch information where the traffic is concentrated excessively are also displayed using the ID-9 API that is, the routing requirement type. In this manner, network errors and the causes of IoT service faults can be easily identified using our proposed architecture and API.

When an IoT service failure occurs, it is difficult to identify the causes. To overcome this problem, we explicitly identify the types of errors. Using our implemented dashboard, we can easily identify the cause of any IoT service problem.

5. Conclusion

In this paper, a method for providing stable IoT services was presented. Two types of requirements, namely, platform-device direct connections, and platform-device network connections, were proposed. In addition, an architecture that supports these requirements by using REST APIs was also proposed. The efficacy of the proposed method was verified via simulation by using a floodlight SDN controller in a case study. The proposed architecture can be applied to manage errors in IoT services. Further, it can be used with a reference architecture to instantiate systems that provide stable IoT services. In future work, we plan to extend our research to integrated IoT platform developments.

6. Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2014R1A1A2007061, NRF- 2014R1A1A2055924).

7. References

1. Gubbi J, Buyya R, Marusic S, Palaniswami M. Internet of Thing (IoT): A vision, architectual elements, and future directions. future Generation Computer Systems. 2013 Sep; 29(7):1645–60.
2. Dijkman R, Sprenkels B, Peeters T, Janssen A. Business models for the Internet of Things. International Journal of Information Management. 2015 Dec; 35(6):672–8.
3. Jayavel K, Nagaraja V. Survey of migration, integration and interconnection techniques of data centric network to intenet-towards Intenet of Things (IoT). Indian Journal of Science and Technology. 2016 Mar; 9(11):1–8.
4. Jarraya Y, Madi T, Debbabi M. A survey and a layered taxonomy of software-defined networking. IEEE Communication Surveys and Tutorials. 2014 Apr; 16(4):1955–80
5. Farhady H, Lee H, Nakao A. Software-defined networking: A survey. Computer Networks. 2015 Apr; 81(1):79–95.
6. Agnise XK, Balagopal D. Leveraging the power of software defined paradigm to control communication in a network. Indian Journal of Science and Technology. 2016 Apr; 9(14):1–5.
7. IEEE Standards Association. Available from: http://standards.ieee.org/getieee802/download/802.1AB-2009.pdf
8. Li L, Li S, Zhao S. QoS-aware scheduling of services-orinted internet of things. IEEE Transactions on Industrial Informatics. 2014 May; 10(2):1497–505.
9. Ming Z, Yan M. QoS-aware computational method for IoT composite service. The Journal of China Univerities of Posts and Telecommuncations. 2013 Aug; 20(1):35–9.
10. Chin W, Kim H, Heo Y, Jang J. A context-based future networks and communications. Procedia Computer Science. 2015 Aug; 56(1):266–70.
11. Architectural styles and the design of network-based software architectures chapter 5 representaional state transfer. Available from: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
12. JSON. Available from: http://json.org/
13. Floodlight. Available from: http://www.projectfloodlight.org/floodlight/
14. Mininet. Available from: http://mininet.org/