Service Virtualisation of Internet-of-Things Devices: Techniques and Challenges

Zeinab Farahmandpour*, Steve Versteeg†, Jun Han*, Anand Kameswaran‡
*Swinburne University of Technology, Melbourne, Australia. Email: {zfarahmandpour,jhan}@swin.edu.au
†CA Technologies, Melbourne, Australia. Email: steve.versteeg@ca.com
‡CA Technologies, Plano, TX, USA. Email: anand.kameswaran@ca.com

***Abstract*—Service virtualization is an approach that uses virtualized environments to automatically test enterprise services in production-like conditions. Many techniques have been proposed to provide such a realistic environment for enterprise services. The Internet-of-Things (IoT) is an emerging field which connects a diverse set of devices over different transport layers, using a variety of protocols. Provisioning a virtual testbed of IoT devices can accelerate IoT application development by enabling automated testing without requiring a continuous connection to the physical devices. One solution is to expand existing enterprise service virtualization to IoT environments. There are various structural differences between the two environments that should be considered to implement appropriate service virtualization for IoT. This paper examines the structural differences between various IoT protocols and enterprise protocols and identifies key technical challenges that need to be addressed to implement service virtualization in IoT environments.**

***Keywords*-Service Virtualisation; Internet-of-Things; Continuous Delivery;**

I. Introduction

The Internet-of-Things (IoT) is an emerging field, which connects a diverse set of devices over different transport layers, using a variety of protocols. Gartner predicts that by 2020, IoT elements will be incorporated in more than half of major new business processes and systems [1]. And yet, there are many challenges to readily deliver IoT systems. As such, there is a pressing need to develop techniques to address these challenges.

As the IoT continues to emerge, there will be a growing number of software applications communicating with IoT devices. The IoT connected software components and applications can be categorised into tiers (as depicted in Figure 1):

- Device gateways (GW): responsible for interfacing directly with an IoT device and providing an API (such as REST) to other applications and services
- Monitors and data aggregators which collect data from IoT devices (edge nodes)
- Applications and services for managing IoT devices
- Analytics engines which data mine aggregated IoT data
- End user applications viewable on the web or mobile devices

Software developers writing IoT applications face challenges, which can delay the release of their application and affect software quality. In particular, to test their application requires interfacing with IoT devices. This may require the physical devices to be present every time the application is fully tested. Furthermore, IoT protocols are very diverse and fragmented, which makes developing and testing for this widespread set of protocols a challenge.

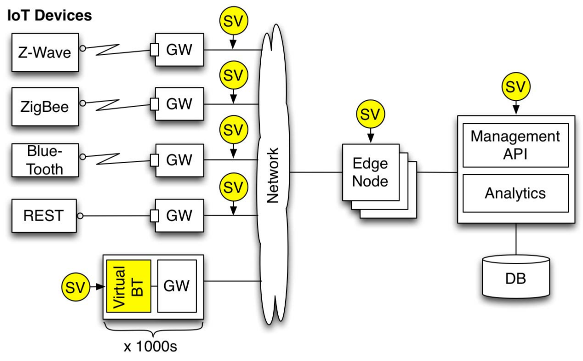

Fig. 1. Points where service virtualisation (SV) could be applied in IoT

Continuous Delivery (CD) [2] is the industry best practice for accelerating software delivery and increasing software quality. At its core, this includes automating each step of the development release cycle and bringing production-like conditions to every test phase. Due to the physical nature of IoT devices as well as their diversity, this poses a challenge to automation.

For enterprise software development, service virtualisation [3] has been applied as a means of emulating all the other services on which an application under test depends. The key idea of service virtualisation is to observe and log the network communication between an application under test and each other service that it interacts with in its production environment. These logged network traces can then be used to build an interactive model, called a virtual service, for each dependency service. The virtual service is then deployed in an emulation environment, allowing the application under test to send requests to and receive responses from the virtual service, as if it were communicating with the real service. This facilitates the automated testing of a software application in production-like conditions, as is required for continuous delivery.

This paper explores whether service virtualisation could be applied to IoT devices to support the realisation of CD for

IoT. IoT DevOps problems include heterogeneous hardware, multiple communication layers, lack of industry standards, and skill sets requiring both operations and development. IoT virtualization can remove constraints for IoT solutions development. Provisioning a virtual testbed of IoT devices can accelerate IoT application development by enabling automated testing without requiring a continuous connection to the physical devices. Figure 1 illustrates the points at which IoT applications could be virtualised. In this paper, we survey a sample of IoT protocols to examine their technical differences to enterprise protocols. On this basis, we examine how service virtualisation would need to be adapted to support IoT protocols.

II. IOT PROTOCOL SURVEY

To address the many challenges in IoT environments, different standards and communication protocols were introduced. There are a wide range of protocols used by IoT devices. In addition to standardised protocols, there are also many non-standard extensions as well as proprietary protocols. We examine five commonly used IoT protocols which give a spectrum of the potential challenges faced for virtualising an IoT environment. Table 1 summarises some key characteristics of the IoT protocols surveyed. As a comparison point, we also show the attributes of the LDAP protocol [5], as an example enterprise protocol.

A. MQTT

Message Queue Telemetry Transport (MQTT) [6] is a session layer publish-subscribe protocol that is used in applications like the Facebook mobile application. MQTT is an extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth on high-latency or unreliable networks. It is designed to provide embedded connectivity between applications and middleware on one side and networks and communications on the other side. The protocol's architecture consists of three main components: publishers, subscribers, and a broker. Publishers are lightweight sensors that connect to the broker to send their data, then go back to sleep whenever possible. Subscribers are applications that are interested in a certain topic, or a type of sensory data, so that they connect to the broker to be informed whenever new data is received. The broker classifies the sensory data into topics and sends it to interested subscribers.

B. CoAP

The Constrained Application Protocol (CoAP) [7] is another session layer protocol that provides a specialized web transfer protocol for use with resource-constrained devices. CoAP is based on the widely successful REST model. Servers make resources available under a URL, and clients access these resources using methods such as GET, PUT, POST, and DELETE. It is built over UDP and has a light-weight mechanism to provide reliability. CoAP contains four messaging modes: confirmable, non-confirmable, piggyback and separate, which support reliable and unreliable transmissions.

C. DDS

Data Distribution Service (DDS) [8] is a leading data-centric publish-subscribe communication standard. This model builds on the concept of a ”global data space” that is accessible to all interested applications. It is a stateless session layer protocol for real-time machine-to-machine communications, that supports both synchronous and asynchronous coordination.

D. ZigBee

ZigBee [9] is a very low-cost, very low-power consumption, two-way, wireless communications standard. Solutions adopting the ZigBee standard are embedded in consumer electronics, building automation, industrial controls, PC peripherals, medical sensor applications, toys, and games. The ZigBee network is comprised of a coordinator, routers and end devices. The coordinator is responsible for initializing, maintaining, and controlling the network. Routers form the network backbone to transfer end devices' packets.

E. Z-WAVE

The Z-Wave protocol [10] is a low bandwidth half-duplex session layer protocol designed for reliable wireless communication in a low cost control network. The protocol's main purpose is to communicate short control messages in a reliable manner from a control unit to one or more nodes in the network. It follows a master/slave architecture in which the master controls the slaves, sends them commands, and handles and schedules the whole network. It supports an asynchronous architecture communications and is used as a protocol to develop smart products and smart home systems.

III. IOT SERVICE VIRTUALISATION CHALLENGES

Based on the surveyed sample IoT protocols, which are listed in the Table I, we have identified three primary areas where IoT protocols differ from most enterprise protocols, which may pose challenges to implementing service virtualisation for IoT. These include communication challenges, message format challenges and modelling challenges.

A. COMMUNICATION SYNCRONISATION CHALLENGES

1) Pub/Sub protocols: IoT protocols such as MQTT and DDS support a Publish/Subscribe architecture. This requires an emulated service to handle situations where a response should be sent in the absence of a triggering request. While service virtualisation has been previously applied to enterprise protocols supporting Publish/Subscribe - it is more difficult to implement than the more widely used client-server protocols, and requires case-by-case implementation. In IoT, Publish/Subscribe architectures are even more prevalent, a generalised approach to emulating Publish/Subscribe therefore requires immediate attention.

TABLE I
IoT Protocol Characteristics

	Enterprise Protocol	IoT Protocols				
Taxonomy \ Protocols	**LDAP**	**MQTT**	**CoAP**	**DDS**	**ZigBee**	**Z-WAVE**
UDP/TCP	TCP	TCP	UDP	TCP/UDP	TCP/UDP	TCP/UDP
Architecture	Client-server	Pub-sub Client-server*	Client-server	Pub-sub Client-server*	Client-server	Client-server
State(ful/less)	Stateful	Stateful	Stateless	Stateless	Configurable	Configurable
Communication direction	Unidirectional**	Unidirectional Bidirectional*	Unidirectional Bidirectional*	Unidirectional Bidirectional*	Bidirectional	Bidirectional
Header Size	Not limited	2 max 5 bytes	4 bit fixed header + binary options	8 bytes	15 bytes	Not specified
coordination	Asynchronous Synchronous	Asynchronous	Asynchronous	Asynchronous Synchronous	Synchronous	Asynchronous
Network layer	Application	Session layer	Session layer	Session layer	Sub-application (application interface)	Sub-application
Real-time	Yes	No	No	Yes	Yes	No

* IoT protocols have standard and non-standard versions with different structures and properties to meet their environmental needs. Therefore, each protocol may cover different schemes for each property simultaneously. This highlights the demand for virtualization service environment for IoT.
** There is one exception for the LDAP server, which acts as an initiator and can be ignored when the LDAP server sends ”Notice of Disconnection” to advise the client that the server is going to terminate the LDAP session on its own initiative [4].

2) Asynchronous messaging: As the IoT nodes aim to conserve battery they minimize energy consumption by utilising sleep mode. For example in Z-Wave, at the time when the control node sends a command to a slave node, the slave node may be in sleep mode. Asynchronous communication is therefore adopted to allow messages to be sent at an arbitrary time. For service virtualisation two challenges arise:

- How to correlate requests and responses?
- How to time when to send responses: this requires keeping track of timestamps.

3) Bi-directional communication: In IoT protocols, the initiation of communication can be unidirectional, bidirectional or a combination of both. For example, in the MQTT protocol, the connection between the publisher and broker as an intervening entity is unidirectional but the connection between the subscriber and broker is bidirectional. Sometimes nodes act as a sender and send their information based on their internal events. For example for a push button, if there is a button press event the node will start sending data without receiving any request. In other situations they respond to ”get information” requests from the server, to send their information. The dominant pattern in enterprise protocol service virtualisation, is for service nodes to act as responders, rather than initiators. For the IoT context, the predominant pattern is for bi-directional nodes, capable of acting as both initiators and responders. An IoT service virtualisation solution therefore requires generalised support for bi-directional emulated nodes.

B. MESSAGE FORMAT CHALLENGES

1) Different messaging modes: There are different modes of messaging in IoT protocols. For example, in the CoAP protocol there are four different messaging modes which can be used based on the requirements, i.e., confirmable, non-confirmable, piggyback and separate. For each mode the structure of response packets is different.

2) Chained commands: A Z-Wave message can contain multiple commands in one message. This is in contrast to most enterprise protocols which have one operation per request. For service virtualisation, this increases the complexity of message format identification, as chained commands would first need to be separated before they can be processed.

3) Fields with less than one byte long: Since IoT deals with resource constrained devices, protocols try to use shorter packets for their communication. It is more common to have bit fields in IoT compared to in enterprise protocols. This increases the challenge of format identification since fields are not limited by byte boundaries.

C. MODELLING CHALLENGES

1) Encapsulated sensory data: Sensory data is encapsulated in multiple protocol layers. For example, ZigBee acts as a transport protocol. Application protocols, layered above it, contain the actual sensory information. The sensory data constitutes the key payload information which would need to be captured by any useful virtual service model. However, extracting the sensory data fields from the multiple protocol layers poses a challenge.

2) Correlation of data models: An IoT service virtualization approach needs to provide an accurate simulation of sensory values. It is important to develop and test an appropriate control system to deal with real devices. Therefore service virtualization needs to derive ”good” emulation of data coming from sensors in an IoT environment, i.e., the generated data from emulated nodes or sensors should be realistic to help testing the controller. A key question is how close the emulated data should be to a real data stream. Not only do the sensory values need to be accurate, but also sensory values need to be responsive to commands of a controller. For example, for an air-conditioner use case, if we want to generate a model of the temperature sensor, we cannot consider it as an isolated node, because its value is dependent on the commands that the controller sends to the air conditioner. If the temperature that is captured by the sensor is above a threshold, the controller sends a command to the air-conditioner to increase the power of the air conditioner. In response, the temperature is expected to decrease. Therefore for the purpose of service virtualisation, correlation between different elements of the network should be considered in extracting and generating a data model. While this issue also exists in enterprise systems, it is even more paramount in IoT.

IV. IOT OPAQUE SERVICE VIRTUALISATION

From our analysis it is clear that there are differences between IoT protocols and enterprise protocols which require adaptations to service virtualisation for it to be applied successfully to IoT devices. As identified the key challenges are the large diversity of protocols, communication challenges, message format challenges and data modelling.

Most service virtualisation approaches decode incoming requests into tokens in order to extract fields and values. A set of defined rules based on these fields and values will then be applied to construct a response to send back to the system under test. A limitation of this approach is that it requires a decoder and protocol handler for every protocol. Due to the large diversity and heterogeneity of IoT protocols it is unrealistic to develop and support a protocol handler for every IoT protocol and their variations. This leads us to exploit methods, which use nor or less prior knowledge and try to extract the model of the services automatically.

A recently proposed approach is opaque service virtualization [11]. Opaque service virtualisation utilises sequence alignment and data mining methods to analyse samples of recorded messages. Rules for constructing responses are automatically derived. Opaque service virtualisation can be applied to a wide variety of protocols without requiring an individual protocol handler for each protocol. An adaptation of opaque service virtualisation therefore seems well suited to handling the heterogeneity challenge of IoT protocols.

Extensions to opaque service virtualisation are required to handle the communication challenges and the message format challenges. The key consideration is the data modelling challenge. For many enterprise use cases, the data is defined by a schema and is discrete. Protocol operations allow records to be created, read, updated or deleted (CRUD). From a service modelling point of view this is relatively straight forward. Either the record is there or it is not, many of the precise values of the record do not matter for the purpose of the testing scenario. In contrast, for IoT scenarios the continuous nature of the data is integral to the testing scenario (such as a for a controller). For example, as discussed in section C, a temperature sensor has a continuous range of values which is a function of the controller settings and the environment. For a realistic IoT virtual service, opaque service virtualisation needs to be extended to include an explicit data modelling step. Data mining methods could be employed to automatically derive correlations between controller settings and sensory fields.

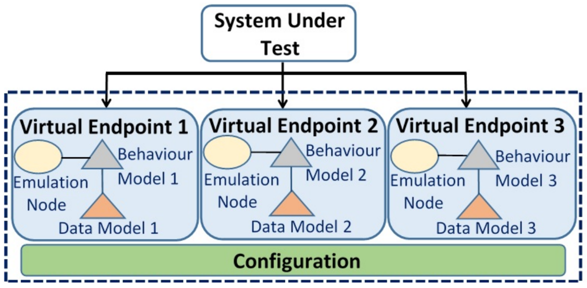

Fig. 2. Virtual Testing Environment (VTE)

Figure 2 illustrates the conceptual virtual service models with data models included.

V. CONCLUSION AND FUTURE WORK

In this paper, we have compared different IoT protocols and enterprise protocols with the focus on expanding service virtualization to the IoT environment. Some key challenges identified for virtualising IoT environments include: heterogeneity, communication synchronisation, formatting complexity and data of continuous nature. To address heterogeneity we plan to implement an extension of Opaque Service Virtualisation. In particular, an explicit data modelling phase needs to be included in the approach. This will allow the automatic virtualisation of IoT environments without requiring prior knowledge of the IoT protocols. We believe this will greatly support IoT developers in enabling them to continuously test their IoT applications in an automated fashion without requiring access to the physical devices.

REFERENCES

[1] Gartner. (2016) By 2020, more than half of major new business processes and systems will incorporate some element of the internet of things. [Online]. Available: http://www.gartner.com/newsroom/id/3185623
[2] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education, 2010.
[3] J. Michelsen and J. English, “What is service virtualization?” in *Service Virtualization*. Springer, 2012, pp. 27–35.
[4] J. Sermersheim. (2006) Lightweight Directory Access Protocol (LDAP): The protocol. [Online]. Available: https://tools.ietf.org/html/rfc4511.html
[5] J. Hodges and R. B. Morgan. (2002) Lightweight Directory Access Protocol (V3): Technical specification, RFC3377. [Online]. Available: https://tools.ietf.org/html/rfc3377.html
[6] A. Stanford-Clark and H. L. Truong. (2013) MQTT for sensor networks (MQTT-SN) protocol specification (V1.2). [Online]. Available: http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf
[7] Z. Shelby, K. Hartke, and C. Bormann. (2014) The Constrained Application Protocol (CoAP), RFC7252. [Online]. Available: https://tools.ietf.org/html/rfc7252
[8] Object Management Group. (2015) Data Distribution Service (DDS) (V1.4). [Online]. Available: http://www.omg.org/spec/DDS/1.4/PDF/
[9] S. Safaric and K. Malaric, “ZigBee wireless standard,” in *Multimedia Signal Processing and Communications, 48th International Symposium ELMAR-2006 focused on*. IEEE, 2006, pp. 259–262.
[10] Zensys A/S. (2006) Z-Wave protocol overview. [Online]. Available: https://wiki.ase.tut.fi/courseWiki/images/9/94/SDS10243_2_Z_Wave_Protocol_Overview.pdf
[11] S. Versteeg, M. Du, J.-G. Schneider, J. Grundy, J. Han, and M. Goyal, “Opaque service virtualisation: a practical tool for emulating endpoint systems,” in *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 202–211.