Bachelor Thesis

Datateknik
Computer Engineering

Comparison between SensibleThings and Kaa platform

Lu Cao
Comparison between SensibleThings and Kaa platform
Lu Cao

MID SWEDEN UNIVERSITY
Department of information and communication systems

Examiner: Ulf Jennehag, ulf.jennehag@miun.se
Supervisor: Forsström Stefan, stefan.forsstrom@miun.se
Author: Lu Cao, calu1502@student.miun.se
Degree program: Computer Engineering, 180 credits
Main field of study: Computer Engineering
Semester, year: VT, 2016
Abstract

With the Internet of Things becoming more and more popular, and a prediction that there will be more than 50 million devices connected to the Internet in 2020, the quantity of IoT platforms on the market is rapidly growing. Facing so many platforms to choose, the object of this thesis is to give some suggestions for reference by performing a quantitative comparison between two platforms: SensibleThings and Kaa. These two platforms have difference architectures so may suitable in different scenes. The comparison includes some measurement and evaluation under two designed scenarios and a general contrast in theory. Two scenarios cover cases of message delivery between two endpoints at different rates and multiple endpoints pushing log data continually. The result of measurement together with the theoretical analysis draw out the following conclusion. SensibleThings platform is more suitable for simple and small-scale message delivery between endpoints, like home environment with few devices. And Kaa platform is more suitable for large-scale and complicated application for data collection and processing, like meteorology field with huge amount of sensors and data.

Keywords: Internet of Things, IoT platforms, SensibleThings, Kaa, Comparison.
Table of Contents

Abstract ........................................................................................................................................ iii
Terminology ................................................................................................................................... v

1 Introduction ................................................................................................................................ 1
  1.1 Background and problem motivation ...................................................................................... 1
  1.2 Overall aim ............................................................................................................................... 1
  1.3 Concrete goals .......................................................................................................................... 2
  1.4 Scope ......................................................................................................................................... 2
  1.5 Outline ....................................................................................................................................... 2
  1.6 Contributions ............................................................................................................................ 3

2 Theory ........................................................................................................................................... 4
  2.1 The Internet of Things and IoT Platforms ............................................................................... 4
  2.2 SensibleThings Platform ........................................................................................................... 6
    2.2.1 Components ....................................................................................................................... 6
    2.2.2 Technical Principles .......................................................................................................... 6
    2.2.3 Applications ..................................................................................................................... 8
  2.3 Kaa Platform ............................................................................................................................ 8
    2.3.1 Components ....................................................................................................................... 9
    2.3.2 Technical principles .......................................................................................................... 9
    2.3.3 Applications ..................................................................................................................... 11

3 Methodology ............................................................................................................................... 12
  3.1 To achieve goal one ................................................................................................................... 12
  3.2 To achieve goal two ................................................................................................................... 12
  3.3 To achieve goal three ............................................................................................................... 12
  3.4 To achieve goal four ................................................................................................................ 13

4 Implementation ............................................................................................................................ 14
  4.1 Experiment environment .......................................................................................................... 14
  4.2 Scenario one ............................................................................................................................ 15
  4.3 Scenario two ............................................................................................................................ 17

5 Results ........................................................................................................................................... 19
  5.1 Scenario one ............................................................................................................................. 19
  5.2 Scenario two ............................................................................................................................ 22
  5.3 Theoretical comparison ............................................................................................................ 25

6 Conclusions .................................................................................................................................... 27
  6.1 Platform recommendation ........................................................................................................ 27
  6.2 Ethical consideration ................................................................................................................ 27
  6.3 Future work ............................................................................................................................... 28

References ....................................................................................................................................... 29

Appendix A: Code of own developed programs ............................................................................. 31
Comparison between SensibleThings and Kaa platform
Lu Cao

2016-06-08

Terminology

<table>
<thead>
<tr>
<th>Abbreviations</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoAP</td>
<td>Constrained Application Protocol</td>
</tr>
<tr>
<td>ms</td>
<td>milliseconds</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunication Union</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>RFID</td>
<td>Radio-frequency identification</td>
</tr>
<tr>
<td>RTT</td>
<td>Round-trip time</td>
</tr>
<tr>
<td>RUDP</td>
<td>Reliable User Datagram Protocol</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>UCI</td>
<td>Universal Context Identifier</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>ZB</td>
<td>Zettabyte</td>
</tr>
</tbody>
</table>
1 Introduction

This is a 15hp project to get a bachelor degree of Mid Sweden University for a Chinese international student. The idea of this thesis given by teacher is a popular topic, the Internet of Things (IoT) and IoT platform to study on. During the thesis, I will learn staffs about Internet of Things including its early history, development trend, and fields of applications. Then, I will learn about IoT platforms and study two specific ones. After that, have a quantitative measurement via some scenarios together with a general comparison in theory. Finally I will draw out the conclusions.

1.1 Background and problem motivation

Today, we can see an increasing trend in the Internet of Things all over the world. During the past eleven years since International Telecommunication Union (ITU) published a report[1], we can feel a great change and convenience that Internet of Things brings to our society and daily life. Nowadays, technologies of Internet of Things are wildly applied in different industries and many fields, such as government, transportation, agriculture, education, marketing, medical industry, public services, etc.

A forecast[2] from Gartner, Inc. shows that 6.4 billion connected things will be in use worldwide in 2016, up 30 percent from 2015, and will reach 20.8 billion by 2020. In 2016, 5.5 million new things will get connected every day, and the number will continue to increase. Another white-paper[3] from Internet Society shows that there may as many as 100 billion connected IoT devices generating a global economic impact of more than $11 trillion by 2025. The things indicate devices such as mobile phones, embedded devices, smart furniture, micro-controllers, and even sensors and actuators. All of them will connect to the cloud and each other in what is commonly called the Internet of Things.

That is, Internet of Things is acting a more and more important role in the future. Therefore, IoT software platforms give an IoT solution to help manage huge amount of devices, fast develop IoT applications and meet more features. There are more than hundreds of IoT middle-ware platforms on the market, tens of which are developed well and popular. Different platforms have their features and different performance, so it is not easy to select suitable one for an IoT solution.

1.2 Overall aim

Because of the variety of IoT platforms, It can be helpful to select a suitable one that has a good performance and easy and user-friendly to use. So it may necessary to have a comparison between IoT platforms via some designed scenarios which are most commonly used features in an IoT application. This thesis is going to perform a comparison between two specific IoT platforms. The aim is to determine which IoT platform fits better for which kind of scenarios. In particular, the scenarios to be evaluated will include an embedded
device and a laptop exchanging messages through the platform. In detail the measurement should determine how the endpoints of platforms perform under stress and how it effects with different quantity of endpoints. The evaluation which mainly focuses on response time, together with a theoretical contrast will reach the conclusion.

Therefore, the problem I will solve in this thesis is to have a both theoretical and experimental contrast and decide which platform has a better performance for two scenarios. I will have a quantitative comparison between two IoT platforms by designing two specific scenarios, then measuring and evaluating the differences. Afterward, the theoretical contrast is as followed.

1.3 Concrete goals
To achieve the aim of the whole comparison between these two IoT platforms, I will split them into several steps to complete. The following are the concrete goals of the thesis to follow:

1. Understand the basis of the Internet of Things and IoT platforms. Then Decide two IoT platforms to study on and comprehend the overall and how each of them works by documents.

2. Know about both software and hardware requirements of the platforms and set up the environment to develop. Determine that how the topology of the experiment looks like.

3. Design two scenarios and write programs. Then measure and collect required data for two designed scenarios. Next, Use suitable data to make charts. Then have a theoretical contrast between platforms.

4. Evaluate the performance of each platform by analyzing the charts. Then make the comparison with explanations and a theoretical contrast. Finally draw out the conclusion.

1.4 Scope
Firstly, this thesis only focuses on two specific IoT platforms although there are many on the market. With a limit of time, scenarios will only cover two most commonly used features in an IoT application. Comparison for other more aspects will be left in the future. Then measurement of scenarios will mostly focus on response times of message delivery. An overall comparison will be talked in theory. Finally the evaluation is thus more focused on Which platform is more suitable and recommended for a scenario.

1.5 Outline
The remainder of this article is organized as follows: chapter 2 will have a brief introduction of Internet of Things as well as two specific platforms including how they work and everything necessary to understand the work afterward. chapter 3 shows the methodology used at each step in the project. And for detailed design of two scenarios, it will be displayed in chapter 4. Then the
result of measured data of each scenario is shown in chapter 5. There is also a theoretical comparison between two platforms. Chapter 6 will have a summary of analysis and the final conclusion. This part also includes an ethical consideration and the future work.

1.6 Contributions
Dr Tingting Zhang and Professor Forsström Stefan provided a lot of help and suggestions on the direction of the work. The community of Kaa gave some help in technology. All of programs and measurement in this thesis are finished by myself.
2 Theory

Since the late 1990s, when the concept of Internet of Things was raised, to nowadays many blueprint and dreams have been achieved. With the development of computer industry and the Internet, more and more relevant products, both hardware and software come out. And then IoT platforms are developed involuntary for control and management of IoT devices. Some platforms are open source and developing while some are proprietary and mature. This thesis want to compare two IoT platforms on the market and have an evaluation which may help those who has problems choosing or developing an IoT platform.

The following I will show all necessary theory to understand my work. The first part I will have a brief introduction of Internet of things and IoT platforms. Most of them are summed up from a series of papers and surveys. After that, there will be two sections to introduce the specific platforms I will study on. In each section, the components, architecture and work flow will be covered.

2.1 The Internet of Things and IoT Platforms

The Internet first began in the late 1960s as the link of between some university computer centers. In the 1970s and 1980s, the number of user was counted in thousand and computers are main component of Internet. In 1991 computer scientist Mark Weiser put forward a concept of ubiquitous computing which opened up huge opportunities for the Internet[4]. Before 21th century, users of Internet denominated in millions. In 1999, the term “Internet of Things” first became popular through the work[5] of Auto-ID Center at MIT which started to design Radio-frequency identification (RFID) infrastructure that become one of key technology supporting the Internet of Things. In 2005, the term “Internet of things” spread rapidly – The white paper[1] of “the Internet of Things” was published as the seventh in the series of “ITU Internet Reports”. After that in 2008 the first scientific conference[6] was held in this research area.

The Internet of Things had a very slow development at the beginning because some supporting technologies are not mature and expensive at that time. The white paper of ITU raised four important technological enablers of the Internet of Things: RFID, sensor technologies, smart technologies, and nanotechnology. (a) RFID refers to those technologies that use radio waves to automatically identify and track individual items. Nowadays it and similar technologies are very mature frequently used in logistics, library, Access Control System and other public utilities. (b) Sensor technology enables “things” to detect and measures physical stimuli – for instance, motion, heat, etc. It has become indispensable in a large of industries and even our daily life. (c) Smart technologies used on clothing, furniture, vehicles, and robotics become popular to assist people in their work and life. These smart things can connect to and become users of the Internet to make up the Internet of Things. (d) Nanotechnology is important but invisible to the naked eye. It involves medical
development, environmental benefits and information technologies. These four technologies are not independent and may cross together. With related technologies becoming mature and lower-costs, even a family is able to use them.

Besides, mobile communication technology such as 4G, and development of wireless network support a high speed for information exchange between devices. Therefore, we can imagine that there will be more “things” easy to connect together in different fields to even grow up new ecosystems. Another white-paper[7] from Cisco predicts there will be 50 billion devices connected to the Internet by 2020 and these estimates do not take into account rapid advances in Internet or device technology. Oxford Economics study[8] found that till 2015 only 8% of businesses are actually using more than 25% of their IoT data. So the growth is stalling and Internet of Things has not yet filled the gap.

As the growth of the connected devices and the market of Internet of Things, data created by people, and things has reached 8.5 zettabytes(ZB) will attain 44 ZB by 2020. The huge data generated by the Internet of Things need a solid infrastructure to bring more business cases to life. From a bird’s eye view, four major technological building blocks of IoT are emerging[9]: (1) Hardware, where data is produced includes the physical devices with built-in microprocessors, sensors, actuators and communication hardware. (2) Communication, where data gets transported ensures the hardware is connected to the network. (3) Software back-end, manages all connected devices and networks and provides the necessary data integration as well as user interface. (4) Applications, which present IoT use cases to the user, run on smart phones, tablets, PCs or other devices/things.

With the demands on better services, IoT platforms comes out to integrate IoT devices, store and even analysis large amounts of generated data. Of the IoT platforms on the market, I will roughly divide them into two categories according to their architectures: centralized and distributed system:

(a) Most of IoT platforms being released today seem to be cloud-based. The cloud can be commercial server cluster provided by IT companies or an open source software deployed as a server in person. This kind of platforms are considered as centralized systems which means the things to connect relay on the cloud to delivery messages. Typical examples of these cloud-based architecture include: ThingWorx[10], Kaa[11], Nimbits[12], Azure IoT Suite[13], Yeelink[14], and many more.

(b) Few IoT platforms are distributed operating a peer-to-peer (P2P) manner, where things can exchange message directly. Moreover each entity need to store and administer the information locally in a P2P network. This kind of platforms do not contain single point of failure and thus more resilient. Typical examples of such systems are LinkSmart®[15], Nabto[16], and SensibleThings[17].
2.2 SensibleThings Platform

The SensibleThings is a platform for creating fast and efficient IoT applications. It is developed as a project of Mid Sweden University in collaboration with partners from both industry and academia. In detail, it is fully licensed under the GNU Lesser General Public License Version 3 (LGPLv3), which enables commercialization of third party products built on top of the platform. SensibleThings offers an open source framework for connecting sensors and actuators together, in order to enable real-time and scalable context-aware applications.

2.2.1 Components

The SensibleThings platform can be divided into different layers with ingoing components. As shown in figure 2.1 from paper [18], these layers include an interface layer, an add-in layer, a dissemination layer, a networking layer, and a sensor/actuator layer. The interface layer is the public interface through which applications interact with the SensibleThings platform. The add-in layer enables developers to add optional functionality and optimization algorithms to the platform. The dissemination layer enables dissemination of information between all entities that participate in the system and are connected to the platform. The networking layer enables connection of different entities over current IP based infrastructure. And finally, the sensor and actuator layer which enables different sensors and actuators to connect into the platform.

![Figure 2.1 The architecture of SensibleThings](image)

2.2.2 Technical Principles

To be specific, the SensibleThings platform is build with the following features: no central point of failure, scalable, fast, seamless, lightweight, and extensible. The platform is designed to use P2P technology to enable a more scalable system without any central points of failure. That is, it is a fully distributed system (but with a bootstrap node) where distributed hash tables (DHT) is used to look up identities. There is a choice between a Kelips and a Chord DHT. The platform use CoAP and RUDP protocol to transmit messages. So the payload data is sent in the first packet to keep the response time minimal. And it is
capable of signaling in real-time between endpoints. Furthermore, the platform is able to penetrate multiple NAT, using a proxy solution to make sure nodes behind NAT work well. All these are implemented by the open source LGPLv3 licensed Java code. So the nodes of platform can run on any device with a Java Virtual Machine like mobile phones or limited devices. The platform is also extensible to add new features without redistribution.

The SensibleThings platform enables two nodes in the system to send messages bidirectionally. I will make a simplified explanation with figure 2.2 below to show how the platform works.

Figure 2.2 shows that entities of SensibleThings platform are logically organized as a ring-like structure. Each entity running on a device represents a node in the ring. When a node wants to connect the platform, it needs to talk to a bootstrap node to join the ring as figure 2.2(2). Then nodes can register themselves by universal context identifiers (UCI) like figure 2.2(3). A UCI is a formatted string like "lu@miun.se/test", and should be unparalleled for each node. After that, the whole DHT knows which UCI belongs to which node. After registering, any other node can resolve the UCI to get the node address as figure 2.2(4). When a node gets other nodes' addresses, it can set up a peer-to-peer session. There are three primitives: GET, SET and NOTIFY. The first primitive can send a message without content while the latter two can send messages with a string.
2.2.3 Applications
The SensibleThings platform can be applied in a wide range of scenarios, e.g., health care, intelligent home, object tracking, and social applications, etc. Figure 2.3 from [19] shows some of the developed proof-of-concept applications that use the SensibleThings platform. For example simple sensor value readings (radon, CO2, temperature, humidity, etc.), intelligent home automation (to interact and monitoring power sockets), object tracking (for monitoring sensor enabled objects), surveillance (for remote monitoring), and showing historical data graphs (for displaying stored measurements).

Figure 2.3 Applications developed on SensibleThings

2.3 Kaa Platform
Kaa is a production-ready and open-source middleware platform supported by CyberVision Inc. for rapid development of the Internet of Things solutions, IoT applications, and smart products. It provides a feature-rich toolkit for the IoT product development and thus dramatically reduces associated cost, risks, and time-to-market. The platform is able to connect and manage any device via the cloud, have ready-to-go features with prototype, collect and analyze valuable data in real time, scale to support millions of endpoints and be deployed anywhere: on-premises or cloud. For development, experimentation, or small-scale uses, Kaa offers a Sandbox Image to install in a VirtualBox VM. It also provides packages and source code for Kaa cluster installations.
2.3.1 Components

Kaa enables data management for connected objects and a back-end infrastructure by providing the server and endpoint SDK components. The structure is shown in figure 2.4. The SDKs get embedded into connected device and implement real-time and bi-directional data exchange with the server. The Kaa platform is hardware-agnostic to support highly portable SDKs which are capable of being integrated with any type of connected device or microchip.

The Kaa server provides all the back-end functionality needed to operate even large-scale and mission-critical IoT solutions. It handles all the communication across connected objects, including data consistency and security, device interoperability, and failure-proof connectivity.

The Kaa server also features well-established interfaces for integration with data management and analytics systems. It acts as a foundation for the back-end system that are free to expand and customize to meet the specific requirements.

2.3.2 Technical principles

A Kaa deployment is a particular implementation of the Kaa platform and it consists of a Kaa cluster and endpoints. A Kaa cluster represents a number of interconnected Kaa server nodes. An endpoint is a specific Kaa client registered (or waiting to be registered) within a Kaa deployment.
For a high-level architecture as figure 2.5, Kaa cluster consists of Kaa nodes that use Apache ZooKeeper for services coordination and SQL and NoSQL database instances for information storage. The Kaa cluster node is comprised of the Control, Operations, and Bootstrap services.

A Kaa Control service is responsible for managing overall system data, processing API calls from the web UI and external integrated systems, and delivering notifications to Operations servers. In addition, Control service provides web UI, which is a standalone component that integrates with the Control server and allows users to create applications, register and configure endpoints, create endpoint groups, etc.

A Kaa Operations service is a “worker” service that is responsible for concurrently handling multiple requests from multiple clients. Most common Operations service tasks include endpoint registration, processing endpoint profile updates, configuration updates distribution, and notifications delivery. Multiple nodes with Operations service enabled may be set up in a Kaa cluster for the purpose of horizontal scaling.

A Kaa Bootstrap service is responsible for directing endpoints to Operations services. On their part, Kaa endpoints have a built-in list of Bootstrap services set up in the given Kaa deployment. The endpoints use this list to query the Bootstrap services and retrieve a list of currently available Operations services from them, as well as security credentials.

A Kaa endpoint is a particular application which uses the Kaa client SDK and resides on a particular connected device. The Kaa endpoint SDK provides functionality for communicating with the Kaa server, managing data locally in the client application, as well as provides integration APIs.

The Kaa Event subsystem enables generation of events on endpoints in near real-time fashion, handling those events on a Kaa server, and dispatching them to other endpoints that belong to the same user. Each event is based on a particular event class (EC) that is defined by the corresponding event class schema. An event class schema format is based on the Avro schema with the additional attribute classType that supports two values: event and object.

ECs are grouped into event class families (ECF) by subject areas. ECFs are registered within the Kaa tenant together with the corresponding event class family schemas. An ECF is uniquely identified to prevent naming collisions during the SDK generation. Once the application and ECF are created, the tenant administrator can create a mapping between these two entities by assigning a certain version of the ECF to the application. This mapping in Kaa is called event family mapping.

Events can be sent to a single endpoint or to all the event sink endpoints of the given user. In case of a multicast event, the Kaa server relays the event to all endpoints registered as the corresponding EC sinks during the ECF mapping. In case of a unicast event, the Kaa server delivers the event to the target endpoint only if the endpoint was registered as the corresponding EC sink during the
ECF mapping. Until being expired, event remains deliverable for the endpoints that were offline at the moment of the event generation.

The Kaa Logging subsystem is responsible for collecting records (logs) of pre-configured structure on the endpoints, periodically transferring these logs from endpoints to Operation servers, and, finally, either persisting them on the server for further processing or submitting them to immediate stream analysis. The Kaa logs structure is determined by the configurable log schema. The log schema is fully compatible with the Apache Avro schema. There is one log schema defined by default for each Kaa application. The application developer is responsible for designing the custom log schema and invoking the endpoint logging API from the client application.

2.3.3 Applications

While the Internet of Things has opened up a new technical innovations, which are equally valuable for a broad variety of industries. As figure 2.6 shows[20], Kaa platform provides many good IoT use case solutions on agriculture, automotive, consumer electronics, healthcare, industrial IoT, logistics, smart city, smart energy, smart retail, sport & fitness, wearables, etc.

Figure 2.6 some popular Kaa use cases
3 Methodology

Chapter 3 presents the methodology used in the whole project and start with the background of the work, followed by the approach for each concrete goal. This project studied in Mid Sweden University, begins with an individule study of “The Internet of Things” and IoT platforms. During the work, there are some irregular presentations and meetings with teachers to talk about milestones in different sessions. Then I will show the research methods for each goal.

3.1 To achieve goal one

Goal one is to study the theory of The Internet of Thing and IoT platforms. A fast way is to search some valuable papers first to understand some basic concepts. Then surveys or white papers published by international organizations are useful to know more details about Internet of Things. Google scholar search engine and databases of minu[21] are used to collect materials. Using “The Internet of Things”, “IoT platforms” as the key words, some papers which are quite repeatedly cited are downloaded and read. After awaring of knowledge of IoT, some newly paper and research also help to grasp its trend. After that, some popular IoT platforms are searched and two of them are selected in teacher's advice.

3.2 To achieve goal two

Goal two is to get familiar with two selected IoT platforms, set up development environment and then determine the topology of the experiments. Obviously, visiting the official website of the platforms is the first approach to know about the development environment according to their documents. To complete the experiment, an embedded device, Raspberry Pi is needed and should be well configured. A step-by-step instruction from its official website[22] can be easily found to follow. Meanwhile, some linux commands may need to learn. After that, decide the topology of experiments according to different characteristic of two platforms. After consideration, both of the platforms are set in a LAN to be relatively fairr and reduce complexity of the network.

3.3 To achieve goal three

Goal three is to design scenarios, write programs, then measure required data and make charts. These scenarios should be designed equally and run under fair conditions for the different platforms. At this step, there will be many attempts to design the scenarios. Ideas may come from teachers' suggestions, forums of IoT platforms and papers and finally to determine and program for them. For each scenario, there are many runs with different parameters. So it is a good way to use command line parameters so that the programs are compiled once. Shell scripts can also be used to improve efficiency. Results from the finalized scenarios include logs and needed data such as message, timestamp, etc. The data will be processed by Excel and converted to charts. After that, there will be a theoretical contrast to compare some characteristic between two platforms.
3.4 To achieve goal four

Goal four is to evaluate results and draw out the conclusion. Through the analysis of the given graphs and charts, together with the theoretical contrast between platforms, it can be concluded that which kind of scenario each platform is more suitable. In addition, give two possible suitable cases for reference according to the evaluation.
4 Implementation

After knowing about the features of two IoT platforms, I am going to design two basic scenarios to evaluate performance of each. Both of the scenarios should be simple and meaningful for an IoT application. The first one is designed to delivery messages between endpoints. This is pretty common because it is necessary that in an IoT application information need to be exchanged between endpoints to implement some use cases. The second one is collecting data from multiple endpoints. That is, several endpoints need to continually push logs to one endpoint for postprocessing. This is also a general function in an IoT application to get regular information from sensors, or track locations and status of endpoints, etc. Figure 4.1 below shows the overview structure of the implementation. The experiment environment is in a local area network (LAN), which will be mentioned in section 4.1. Under two designed scenarios, endpoints will delivery messages across the IoT platforms. The details will be mentioned in section 4.2 and section 4.3.

![Figure 4.1 Overview of implementation](image)

4.1 Experiment environment

Before all, the experiment environment where scenarios run need to be clarified. It is clear that the architecture of SensibleThings and Kaa platform are totally different. SensibleThings platform is a P2P network and realized by DHT while Kaa platform have a central server which each endpoint need to connect first. It is difficult to have a good evaluation with different and complicated Internet environment two platforms will face. So limiting platforms in the same LAN can be relatively fair and more convenient to reduce the interference of transmission delay on the Internet and other unknown factors.

The topology of the experiment is illustrated in figure 4.2 below. Need to notice, a local bootstrap node will run for SensibleThings and a Kaa sandbox is deployed as the server in VM instead of a cluster. All devices used in scenarios...
with private IP addresses in LAN are: (1) one raspberry Pi 1 running Linux with 512 MB memory (2) one laptop running Linux with 4 GB memory. (3) a VirtualBox VM (installed on a PC with 16 GB memory) running the Kaa Sandbox with 4 GB memory. Programs of nodes will be deployed on the raspberry pi and the laptop.

![Diagram](image.png)

**Figure 4.2 the topology of the experiment**

### 4.2 Scenario one

The core of the first scenario is message delivery between two endpoints which is a most basic function in an IoT application. To evaluate the message handling capability of endpoints under pressure, the message delivery should be tested with a range of rates. The specific designed is as followed.

In the scenario, there are two endpoints exchanging messages. After two endpoints are initialized, A is to send messages to B periodically. Then B is to send a message back when receiving one message. Finally A will receive the reply message and finish one round message exchange. At the side of A, it is possible to calculate the message exchange time by recording timestamps of sending and receiving. According to a series of the measured time, I can evaluate and analyze the performance of endpoints of each platform for this scenario. Figure 4.3 shows the work flow of the two endpoints.

For SensibleThings platform, besides a stable bootstrap node, there are two endpoints participating this scenario. All the three programs use a JAR packet which SensibleThings provides. The bootstrap node is set local so that endpoints make up a local ring in LAN. For the detailed code of the programs running on each endpoint, you can refer appendix A to get them.
Endpoint B at the Raspberry Pi first need to register itself in order that endpoint A at laptop can later resolve B. This is the initial process of two endpoints. After A resolves B's address, it sends messages to B at a constant delivery rate. The primitive used to send message is “NOTIFY”. Each message here is a string which contains the timestamps $T_1$ indicating the time for endpoint A to send message to B. When receiving the message, B immediately sends it back without modification. When receiving the reply, A records the current timestamps as $T_2$. Using $T_2$ minus $T_1$ from the message, one round message exchange time is got. Each run may contain several hundred rounds of messages. The delivery rate of each run will vary from 1ms to 1000ms.

![Figure 4.3 the work flow in scenario one](image)

For Kaa platform, there exists Event subsystem to handle messages delivery between endpoints. This can be seen a module that Kaa platform provide. As mentioned in theory part, event class and event class families need to be per-configured to define the message structure. The event contains a timestamps of Long type. Then the SDK can be generated from web UI of Kaa. The configuration of EC and ECF can be found in appendix A. The Kaa Sandbox runs as the central server on a virtual machine in the LAN.

When endpoint B runs at Raspberry Pi and endpoint A at laptop, they will first connect to the server and will only communicate with the server directly. Then endpoint A sends an event periodically to B. The event will be handled first by the server and then transferred to B. When receiving the event from A, B sends back the same event with a similar route through the server. When A receives the reply, the message exchange time can calculated according to the above method. Each run may contain several hundred rounds of messages and the delivery rate will vary from 1ms to 1000ms. The detailed code of Kaa platform in this scenario can be found in appendix A.
4.3 **Scenario two**

The second scenario is also one of the most common features in an IoT application. That is, logs from many endpoints need to be pushed periodically for storage or other postprocessing. So here I will let multiple endpoints push log messages to evaluate the performance of the platform in this kind of scenario.

![Schematic diagram for scenario two](image)

**Figure 4.4 Schematic diagram for scenario two**

Figure 4.4 shows the schematic diagram of two platforms to handle the scenario. Multiple endpoints will delivery messages simultaneously at a constant rate. But it is clear that two platforms have totally different architecture to achieve it. Of SensibleThings, an endpoint have to act as a receiver to communicate with many other endpoints. But Kaa has a powerful server and the logging subsystem to handle it. Though not fair enough, it depends on the features of each platform.

For SensibleThings, It is not easy to implement this scenario. Because each node in SensibleThings platform is treated equally. However, to implement this scenario I have to treat one node as the sink node to receive logs from other endpoints which act as source nodes. The log messages contain the endpoint ID and transmission timestamps which are spliced into a string.

First, source nodes will be run, to join the ring and register their UCI. Then the sink node will resolve these endpoints, after which it uses the primitive “GET” to let source nodes begin to send logs. The logs includes the ID of the source nodes and transmission timestamps. When receiving a log, the sink node prints it on the screen and then replies the log message back. At the source nodes' side, the round-trip time can be calculated according to the method of scenario one. The number of source nodes is in the range from 4 to 20. The programs of source nodes are the same, which use command line arguments to complete different runs to avoid repeating the compilation. Then I have the round-trip time of each source nodes when different quantity of source nodes send logs simultaneously.
For Kaa platform, it is very convenient to use log subsystem to continuously push data to the server. Log subsystem allow endpoints to push custom messages to the server. And the Kaa server can store the received messages in the database, then reply a confirm acknowledgement back to the endpoint. As similar as the Event subsystem, the format of the log massages is custom and need to per-configured in log schema. You can find the log schema in appendix A.

Different from SensibleThings, I don't need to write programs for the server in this scenario. Using the generated SDK, the programs at endpoints need to first connect and then send log messages to the server. The messages contain the ID of nodes and messages and timestamps. When the server receives a log message, it stores the log into databases and then replies an ack to the endpoint. At the endpoints' side, the SDK provides a method to get the round-trip time after it receives the ack. Then I can also get a series of round-trip time when different number of endpoints work. The command line arguments and bash scripts are used to let endpoints send logs almost at the same time. You can refer appendix A to find the codes of endpoints of Kaa in scenario two.

As fair as possible for the measurement of scenario, the source nodes are placed on the laptop. For Kaa platform, the Kaa Sandbox runs on a virtual machine installed in a PC. So for SensibleThings the sink node runs on the same virtual machine. The delivery rate of scenario two is set as 100 ms.
5 Results

Here I will show the results of above two scenarios. Measured data from programs will be first processed by Excel and then used to create different charts to analyze their meanings. The following two sections will show tables and charts of each scenario.

5.1 Scenario one

For SensibleThings platform, I set seven sample groups where one endpoint sends messages to another periodically. The transmission intervals can be 1, 5, 10, 50, 100, 500 and 1000 milliseconds(ms). And for each sample group, the number of messages two endpoints exchange will up to ten thousand. After getting the message exchange time, calculating the average and standard deviation, and charting, the following table and graphs come out.

<table>
<thead>
<tr>
<th>Interval (ms)</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>50</th>
<th>100</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average (ms)</td>
<td>15332</td>
<td>24263</td>
<td>2291.5</td>
<td>15.054</td>
<td>14.846</td>
<td>14.351</td>
<td>14.428</td>
</tr>
<tr>
<td>STDEV</td>
<td>4332.4</td>
<td>11366</td>
<td>1306.1</td>
<td>14.134</td>
<td>15.187</td>
<td>15.508</td>
<td>13.881</td>
</tr>
<tr>
<td>Packet loss</td>
<td>64.06%</td>
<td>18.01%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 5.1 Average message exchange time of seven delivery intervals

Table 5.1 displays the average message exchange time, standard deviation and packet loss rate of different delivery intervals. Figure 5.1 is a scatter diagram showing the relationship between message sequence and average message exchange time.

As can be seen, for the data set whose delivery intervals are 1, 5, and 10 ms, the general trend is increase. And both of the average message exchange time and
standard deviation are very high. When the interval is larger from 1 to 10 ms, the average message exchange time, standard deviation will be smaller. With a large packet loss rate, the average and standard deviation when the interval is 1 ms is lower than those when interval is 5 ms.

That is, the endpoint of SensibleThings is not able to handle that high delivery rate. Especially when the delivery interval is smaller than 10 ms, loss of messages will happen. The smaller the delivery interval is, the more message it will lost.

Table 5.1 shows that when the delivery interval is greater or equal to 50 ms the average message exchange time is around 15 ms. It is shown in Figure 5.2 that most points concentrated at the bottom and 99.99 percent of the points are under 200 ms when some of the points rise high. Thus, the standard deviation of these four intervals is low. And longer the delivery interval is, more steady the node of SensibleThings performs.

According to the above result, there may be less significant to set delivery interval as 1 and 5 ms. So for Kaa platform, I set 5 sample groups where the delivery interval are 10, 50, 100, 500, and 1000 ms. For each sample group, the number of messages two endpoints exchange is up to one hundred.

Table 5.2 below shows the average and standard deviation of message exchange time of Kaa platform. We can see when the interval is shorter than 50 ms, the average message exchange time can be thousands. When the interval is longer than 100 ms, the average message exchange time is around one hundred.

<table>
<thead>
<tr>
<th>Interval (ms)</th>
<th>10</th>
<th>50</th>
<th>100</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average time (ms)</td>
<td>4785.13</td>
<td>2433.94</td>
<td>127.68</td>
<td>110.22</td>
<td>150.63</td>
</tr>
<tr>
<td>STDEV</td>
<td>908.19</td>
<td>1496.73</td>
<td>86.25</td>
<td>58.99</td>
<td>132.2</td>
</tr>
</tbody>
</table>
Figure 5.3 shows when the delivery interval is 10 ms, the average message exchange time at first is very high, then goes up smoothly, and flat out gradually. When the delivery interval is 50 ms, the average message exchange time raises up from a small value, and then has a slowly downward trend. Both of them have relatively high standard deviation. Therefore it shows that the endpoint of Kaa platform may not be so stable to handle these delivery rate.

Figure 5.3 Average message exchange time of five delivery intervals

Figure 5.4 shows the graph where the delivery interval is longer or equal to 100 ms. We can see almost all points are under 400 ms and most of them concentrate between 0 and 200 ms. This indicates the endpoint of Kaa platform can be a bit more stable when the delivery rate is slower.

Figure 5.4 Average message exchange time of three delivery intervals
Figure 5.5 show the comparison of message exchange time between two platforms with different delivery intervals. This column chart displays the average time together with the standard deviation. Only from the viewpoint of the time cost and stability, SensibleThings is better than Kaa platform. The endpoint in SensibleThings may handle a bit quicker delivery rate than those in Kaa platform. When the delivery rate is not so high, the endpoint of SensibleThings performs quicker and more stable. So in this scenario of messages exchange between two endpoints at different rates, the node of SensibleThings is better.

![Figure 5.5 Comparison between two platforms in scenario one](image)

### 5.2 Scenario two

In this scenario for SensibleThings platform, I plan to set nine groups and use up to twenty endpoints pushing logs to one periodically. Table 5.3 below shows the round-trip time and standard deviation of different number of endpoints from 4 to 18.

<table>
<thead>
<tr>
<th>Number of Endpoints</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average RTT (ms)</td>
<td>13.72</td>
<td>11.81</td>
<td>12.21</td>
<td>15.89</td>
<td>18.97</td>
<td>26.99</td>
<td>12.78</td>
<td>39.17</td>
</tr>
<tr>
<td>STDEV</td>
<td>35.81</td>
<td>28.64</td>
<td>33.82</td>
<td>52.38</td>
<td>97.04</td>
<td>104.8</td>
<td>34.00</td>
<td>292.1</td>
</tr>
</tbody>
</table>

We can see that the average round-trip time and standard deviation have a slow upward trend with the increase of endpoints as marked with yellow. But the data is a bit abnormal when the number of endpoints is sixteen. Furthermore, to be aware that packet loss happens when number of endpoints is eighteen. It can be calculated that a node of SensibleThings can handle about 18*10=180 messages per second at max. It meets the result of packet loss rate in scenario one.
Quantitative Comparison of SensibleThings and Kaa
Lu Cao

Figure 5.6 shows eight charts corresponding to eight groups. X-axis represents the message sequence and Y-axis shows the round-trip time of each message. The data of each chart comes from one endpoint of each group. From top-left to bottom-right, the quantity of endpoints is 4, 6, 8, 10, 12, 14, 16, 18. The scale of Y-axis is set to the same (2000 ms) to observe the distribution of the points. It is not clear but seems that in the first six charts, with the increase of endpoints, points in the chart have a bit wider distribution which is consistent with the trend of standard deviation.

![Figure 5.6 eight charts of round-trip time with different number of endpoints](image)

**Table 5.4 the data of different quantity of endpoints in Kaa platform**

<table>
<thead>
<tr>
<th>Number of Endpoints</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>13</th>
<th>14</th>
<th>16</th>
<th>18</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average RTT (ms)</td>
<td>58.16</td>
<td>148.00</td>
<td>132.64</td>
<td>226.28</td>
<td>225.17</td>
<td>539.45</td>
<td>307.31</td>
<td>4272.5</td>
<td>900.9</td>
</tr>
<tr>
<td>STDEV</td>
<td>190.02</td>
<td>529.64</td>
<td>449.17</td>
<td>606.63</td>
<td>643.56</td>
<td>1135.5</td>
<td>760.75</td>
<td>7947.8</td>
<td>2180.0</td>
</tr>
</tbody>
</table>

Table 5.4 shows the result of Kaa platform in scenario two. The number of endpoints varies from 4 to 20. The average of round-trip time have an upward trend as marked with yellow but there are outliers when number of endpoints is...
16 and 18. And sometimes when number of endpoints is from 16 to 20, timeout delivery happens which means the latency is larger than 60 seconds (but the endpoint can still get ACK from Kaa server finally). As for the standard deviation, generally it becomes greater when then number of endpoints increases.

Figure 5.7 nine charts of round-trip time with different number of endpoints

Figure 5.7 displays the measured data of nine groups. Each chart shows the data of one endpoint from each group. X-axis displays the message sequence and Y-axis shows the round-trip time of each message. From top-left to bottom-right, the quantity of endpoints is 4, 6, 8, 10, 12, 14, 16, 18, 20. It can be easily seen that there are many breakouts in each chart. One breakout shows that in a range of pushed message, the round-trip time gets very high and then recover to the original. The scale of Y-axis is set to the same (6000 ms) to observe the distribution of the points. It seems that the breakouts become more and reach higher with the increase of quantity of endpoints. It has a great effect on the average round-trip time and standard deviation.

Figure 5.8 Comparison between two platforms in scenario two

Figure 5.8 compares the performance of platforms in scenario two. The column chart shows the average round-trip time together with the standard deviation. The left chart is for SensibleThings while the right one shows the data of Kaa platform. Generally speaking, for SensibleThings, the round-trip time is much lower and have a more stable upward trend than Kaa with the increase of
endpoints. But when the number of endpoints up to eighteen, packet loss happens. It is calculated that the node of SensibleThings can handle about 180 message per second at most. Kaa platform can guarantee delivery of each message without packet loss. For both platforms, the round-trip time has a great fluctuation with larger standard deviation with the increase of endpoints. The lines of standard deviation are even over the charts. When the number of endpoints is around ten, SensibleThings can be considered. If the number of endpoints is larger than twenty, Kaa platform may be better to ensure the message delivery.

5.3 Theoretical comparison

Besides quantitative experiments with two scenarios, there will be a general theoretical comparison in this part between two IoT platforms from a user's point of view. To make it clear, there is a brief summary as table 5.4 shows below.

<table>
<thead>
<tr>
<th></th>
<th>SensibleThings</th>
<th>Kaa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Fully distributed</td>
<td>Centralized</td>
</tr>
<tr>
<td>Supported device/Language</td>
<td>Java, any device with JVM</td>
<td>More</td>
</tr>
<tr>
<td>Protocol</td>
<td>CoAP, RUDP</td>
<td>HTTP, TCP</td>
</tr>
<tr>
<td>Round-trip time</td>
<td>Less</td>
<td>More</td>
</tr>
<tr>
<td>Features</td>
<td>Very few</td>
<td>Rich</td>
</tr>
<tr>
<td>Security</td>
<td>Potential</td>
<td>Better</td>
</tr>
<tr>
<td>Community</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 5.4 theoretical contrast between SensibleThings and Kaa

SensibleThings and Kaa platform can be two representatives of distributed and cloud-based IoT platforms. This means, many difference in the comparison is caused by the architecture.

Talking about a fully distributed system as SensibleThings, it is designed for embedded devices and each node works equally. The program of node is developed by Java, so the platform support any device with a JVM. While Kaa platform is cloud-based and has a powerful server as back-end. Endpoints with portable SDK available in four languages support a wider range of hardware, starting from those powered by fully-functional operating systems down to resource-constrained micro-controllers. Kaa platform use HTTP to support REST API and TCP to communicate between server and endpoints.

SensibleThings CoAP and RUDP which are more suitable for the Internet of Things. Thus, communication between nodes of SensibleThings has a much lower response time. SensibleThings only support three primitives API for developers to use. As for Kaa platform, It is feature-rich to support custom message structure while messages in SensibleThings can only be string. Kaa server of the platform also provides Admin UI from the web, REST API, databases, and visualization tools, all of which SensibleThings don't have. Because embedded devices are poor to support them.
Furthermore, SensibleThings may have some potential security problems like spoofing because of the protocol. Kaa platform performs much better with message verification, authentication, and endpoints grouping. Each endpoint of Kaa will use ssh key to encrypt the messages. And Finally Kaa has a better community which continually developing the platform, while SensibleThings don't have the advantage.
6 Conclusions

In Chapter 5, the thesis has given tables and charts to display the performance of each platform in each scenario as well as a theoretical analysis. In scenario one, we can see the endpoint of SensibleThings can handle quicker delivery rate, and has a much lower and more stable latency than the one of Kaa platform. But packets loss may happen in SensibleThings when delivery rate is much higher. When delivery interval is larger than 100ms, both of two platforms have steady performance. Event subsystem of Kaa need the per-configuration while SensibleThings is more flexible and have a lower latency. So for some oversimplified message delivery like scenario one, SensibleThings is better to choose.

In scenario two, with the increase of endpoints, latency of both platforms goes up and have a large standard deviation. Although latency of SensibleThings is relatively lower, packet loss happen when 18 endpoints work at the same time. That is, one node of SensibleThings can handle belike 180 messages per second. SensibleThings is not good at scenario two because of its architecture. Kaa platform has high and latency, but it guarantee the delivery. Besides, a powerful server is able to handle huge amount of messages and many data processing system to support at back-end. So Kaa is better at this kind of scenario.

6.1 Platform recommendation

SensibleThings and Kaa platform with different architectures are needed in different deployment environments. For example, if you want a lightweight IoT solution with few devices for a home-like environment, SensibleThings is more suitable. And Kaa platform is more suitable for large-scale and complicated application for data collection and processing, like meteorology field with huge amount of sensors and data. While the number of connected device reach hundred, and the application need more back-end supports, Kaa platform is perfect to choose.

6.2 Ethical consideration

This thesis talks about the Internet of Things and IoT platforms. Since the concept of the Internet of Things raised, almost 20 years past. It has been giving so much benefits to industries and our daily life. “Things” is becoming a member of the Internet. The IoT applications provide a convenient to obtain the information. Simultaneously, there also exist some privacy problems where some personal information and privacy could be stolen with a weak security protection.

For the two IoT platforms I studied on, the measurement and evaluation may not cause any ethical problems. Because all the experiments are done in LAN. But talking about a cloud-based IoT platform, there may some hidden troubles to consider. For the clouds provided by some large IT companies like Microsoft,
the data of applications may not be totally private. For some clouds deployed by personal, they may be in danger of attacks from the Internet.

6.3 Future work

The measurement and evaluation via two designed scenarios are rough. Some outliers are strange to happen. If I have more time, a rigorous experiment should be made. Another future work to do is to test the scalability of two platforms. Because of the difference of architecture, the way to measure scalability of each platform is totally different. For SensibleThings, consider to add more nodes into the ring, and let them send messages to each other. Here is a relative paper[23] for reference. For Kaa platform, consider to use a cluster to deploy, evaluate whether the response time will reduce and more endpoints can be hold.
References

[1] International Telecommunication Union (ITU) Internet Reports, 2005: The Internet of Things


Retrieved 2016-03

Retrieved 2016-03

Retrieved 2016-03

https://www.linksmart.eu/redmine
Retrieved 2016-03

Retrieved 2016-03

Retrieved 2016-02


Retrieved 2016-02

Retrieved 2016-03

Retrieved 2016-02

Retrieved 2016-03

Appendix A: Code of own developed programs

All the source code used in this thesis can be found by the link: https://github.com/Cathon/BA_Thesis_IoT_Miun.git