A comparative study of open-source IoT middleware platforms.

En jämförande studie av open-source IoT middleware plattformar.

ROBERT SCOTT OCH DANIEL ÖSTBERG
A comparative study of open-source IoT middleware platforms.

En jämförande studie av open-source IoT middleware plattformar.

Robert Scott och Daniel Östberg

Examensarbete inom Datateknik, Grundnivå, 15 hp
Handledare på KTH: Johnny Panrike
Examinator: Ibrahim Orhan
TRITA-CBH-GRU-2018:7

KTH
Skolan för kemi, bioteknologi och hälsa
141 52 Huddinge, Sverige
Abstract

This is a comparative study of open-source IoT middleware platforms with the main focus on scalability and reliability. An initial evaluation of available open-source IoT platforms resulted in Kaa and Node-RED being the focus of this thesis. To further analyse the platforms, they were both subjected to testing with three real-world scenarios. The chosen scenarios were a remote-controlled LED, a chat application and a data transmitting sensor. Prototypes were developed for each scenario using a range of programming languages and devices like Raspberry Pi, Android and ESP8266.

According to the tests Node-RED has better performance on a single server. It also scales better with the possibility to communicate with external APIs directly unlike Kaa which would require a gateway. Despite these factors, Kaa proved to have better overall scalability and reliability with its built-in security and device discovery, it also supports clustering and should prove better in larger environments.

Keywords

IoT, Middleware, scalability and reliability
**Sammanfattning**


**Nyckelord**

IoT, Middleware, skalbarhet och tillförlitlighet
Preface

We want to thank *A Great Thing AB* for helping us to complete this thesis. We also want to thank William Dahlheim and Pelle Sjöqvist for supplying us with the most essential component for this work, coffee. We also want to thank them and Johnny Panrike for their support and counselling during this work.
List of acronyms and abbreviations

**IoT** - Internet of Things
**SDK** - Software Development Kit
**UI** - User Interface
**GUI** - Graphical User Interface
**CPU** - Central Process Unit
**SOA** - Service-oriented architecture
**HTTP** - Hypertext Transfer Protocol
**JSON** - JavaScript Object Notation
**REST** - Representational State Transfer
**LAN** - Local Area Network
**GPIO** - General-purpose input/output
**LED** - Light-emitting diode
**CoAP** – Constrained Application Protocol
**MQTT** – Message Queue Telemetry Transport
**AES** - Advanced Encryption Standard
Dictionary

**Device** - An endpoint object ranging from sensor, single processor cards, actuator and smart phones

**Gateway** - A device used to connect two different networks

**Application** - A type of software which allows a user to perform specific tasks
# Table of Contents

1 Introduction ............................................................................................................. 1  
  1.1 Problem statement ............................................................................................ 1 
  1.2 Goals .................................................................................................................. 1 
  1.3 Limitations .......................................................................................................... 2 
  1.4 Authors contribution ......................................................................................... 2 

2 Theory and background .......................................................................................... 3  
  2.1 Background ......................................................................................................... 3 
  2.2 Previous studies ................................................................................................... 3 
  2.3 Abstract architecture models ............................................................................... 4 
      2.3.1 Service-oriented architecture (SOA) ............................................................ 4 
      2.3.2 Actor-based architecture ............................................................................ 5 
  2.4 Other considered platforms ............................................................................... 6 
      2.4.1 Zetta.js ...................................................................................................... 6 
      2.4.2 Open connectivity Alliance (Alljoyn/Iotivity) .............................................. 7 
  2.5 Kaa .................................................................................................................... 8 
      2.5.1 Architecture .............................................................................................. 8 
      2.5.2 Components of Kaa-SDK .......................................................................... 9 
  2.6 Node-RED ........................................................................................................ 10 
      2.6.1 Architecture ............................................................................................ 10 
      2.6.2 Community .............................................................................................. 11 
  2.7 A comparative overview of Kaa and Node-RED attributes .............................. 11 
  2.8 Paho framework ............................................................................................... 12 

3 Methods and results .............................................................................................. 13  
  3.1 Prototypes and scenarios ................................................................................... 13 
  3.2 LED scenario ..................................................................................................... 13 
      3.2.1 Kaa prototype .......................................................................................... 14 
      3.2.2 Node-RED prototype .............................................................................. 14 
  3.3 Chat scenario .................................................................................................... 15 
      3.3.1 Kaa prototype ......................................................................................... 15 
      3.3.2 Node-RED prototype .............................................................................. 16 
  3.4 Sensor scenario .................................................................................................. 16 
      3.4.1 Kaa prototype ......................................................................................... 16
3.4.2 Node-RED prototype........................................................................................................ 17
3.5 Testing methods...................................................................................................................... 17
  3.5.1 Server specifications for all scenarios............................................................................. 17
  3.5.2 LED scenario.................................................................................................................... 17
  3.5.3 Chat scenario.................................................................................................................... 18
  3.5.4 Sensor scenario.................................................................................................................. 18
3.6 Results of testing.................................................................................................................... 18
  3.6.1 LED scenario results......................................................................................................... 18
  3.6.2 Chat scenario results......................................................................................................... 21
  3.6.3 Sensor scenario results..................................................................................................... 22
4 Analysis and discussion............................................................................................................. 23
  4.1 LED scenario....................................................................................................................... 23
  4.2 Chat scenario........................................................................................................................ 25
  4.3 Sensor scenario.................................................................................................................... 25
  4.4 Environmental and ethical considerations.......................................................................... 26
5 Conclusion.................................................................................................................................. 27
  5.1 Platform review..................................................................................................................... 27
  5.2 Overall conclusion................................................................................................................ 28
  5.4 Future works........................................................................................................................ 28
References...................................................................................................................................... 31
Attachments .................................................................................................................................. 33
1 Introduction

Today's society is progressively moving towards an inter-connected environment, where people and devices communicate through large networks of devices. These devices can often be controlled and managed through predefined interfaces and applications which are generally determined by the manufacturer and can vary considerably. How to get these devices and users to communicate with each other is the most crucial aspect for IoT platforms.

1.1 Problem statement

IoT platforms have recently received more attention fuelled by cheaper technologies and the ambition for leveraging different devices. This has also led to an increase in available platforms. The company A Great Thing AB has a vision to incorporate one of these platforms into its product but there seemed to be a lack of studies conducted on different open-source solutions. This work addresses this void by investigating and identifying available platforms. The most promising platforms will then be compared and evaluated with focus on reliability and scalability. This evaluation will then be used to determine which platform perform best during selected scenarios.

1.2 Goals

The goal of this work was to investigate and evaluate different open-source IoT platforms. The two most promising platforms would then be compared using real world scenarios with reliability and scalability as key aspects. The scenarios would then be tested with the appropriate constructed prototypes. In these tests a prototype is considered to be an endpoint solution such as a host or client device that use the relevant service provided by the platform. The prototypes were designed to accomplish the following tasks:

- Perform stress/load tests
- Determine round-trip time

The stress and load testing were performed by sending messages with a set time interval to the platform which then returned the message, the client could then compare the timestamps and determine the RTT. The stress test would then decrease the time interval to send more messages during the same period of time. These values can then be used to show trends and the capacity of the platforms.
In consideration of a fair result, the following two requirements were placed on the implementation of the prototypes. Firstly, only the default libraries provided by the IoT platforms were to be used where feasibly possible. Secondly, the prototypes were to be developed using as similar implementations as possible with methods and classes in mind.

The results were then analysed and used to determine which platform that was best suited for the different scenarios. Based on this evaluation and its results, a suggestion of a fitting platform was presented to *A Great Thing AB*.

### 1.3 Limitations

This work is delimited to IoT platforms that are open-source, not 100% cloud-based and cost-free. The tests conducted on the prototypes did not include evaluations of security and visual interfaces. The time allotment for the project was 10 weeks of fulltime work after which the results were to be presented as a report to KTH and *A Great Thing AB*.

### 1.4 Authors contribution

This bachelor thesis was carried out as a collaboration between Robert Scott and Daniel Östberg.
2 Theory and background

This chapter explores and presents background information concerning IoT, studies relating to this work, IoT architectures, other considered IoT platforms, followed by a more comprehensive examination of the selected platforms and additional frameworks.

2.1 Background

The Internet of Things (IoT) sits on the intersection of a wide range of technologies, business models, design patterns, communities, ideas and concepts. These factors often make it hard to elaborate and define a specific definition for IoT, despite this the European Union has defined IoT as “the next step towards the digitisation of our society and economy, where objects and people are interconnected through communication networks and report about their status and/or the surrounding environment”[1].

Although a clear vision for the future of IoT has been present since 2005[3], it has not received much attention until recently. IoT shows great potential to impact multiple industries such as automotive, agriculture and manufacturing. According to Ericsson's forecast, over 18 billion IoT related devices will be used worldwide by 2022. The large increase of devices is fuelled by three factors according to Ericsson, the emergence of applications for IoT, new business models and falling device costs[2].

2.2 Previous studies

A variety of studies within the field of IoT platforms for testing and standardization of architectures have been conducted, especially within academia.

A study by Cao[4] completed in 2016, involved two different test scenarios using the Kaa and SensibleThings platforms. SensibleThings is a peer-to-peer based IoT architecture developed by Mid Sweden University and is an open-source framework for connecting sensors and actuators together for real-time and scalable context-aware applications. The first test scenario was conducted using a Raspberry Pi and a Java client over an Ethernet network and revealed steady performance of both platforms at 100 milliseconds for simple message delivery. The second test scenario used 18 endpoints and sent log messages to each server with Kaa being judged as having guaranteed
delivery and high latency which means that connections can suffer from long delays.

Another study from 2015 presented a holistic view of the challenges for testing open-source IoT software[5]. The challenges evaluated and discussed are IoT virtualization, IoT testbeds, and interoperability testing. Virtualization such as emulators and simulators are useful for basic testing but can be limited by availability of types and can be considered insufficient and must therefore be complemented by tests on target platforms and in testbeds. IoT testbeds are often a network made of real IoT devices used for hardware-based testing. Testbeds offer a few drawbacks such as that they are very expensive to setup and maintain, often difficult to use and often outdated due to the high turnover of new products within the IoT development. Interoperability testing is important as protocol specifications can often contain subtle ambiguities depending on the device or platform which can result in partly or completely incompatible implementations. Another downside to interoperability testing usually requires all possible software and hardware combinations which could result in a high number of combinations to be tested. In order to balance out the shortcomings of each testing method, a limited combination of each should be used to complete a general testing overview of the platform.

There are a limited number of relevant works within the field of testing for scalability and performance concerning IoT that have been produced recently. However, a general summary written in 2015 by IEEE offers a great and valuable insight into IoT concepts and definitions[6].

2.3 Abstract architecture models

Although IoT and its architecture is not defined by a conventional set of standards, several attempts have been made by both academia and standards organizations. In this work the definition of IoT and its architectures are based on the concepts from Guth[7] and Ngu[8]. In unison, they provide a suitable generic overview of different IoT architectures and concepts, which are presented and evaluated in the following sections.

2.3.1 Service-oriented architecture (SOA)

Service-oriented architecture (SOA) in IoT are built on a three or four-layer architecture. According to Ngu (see Figure 2.1) the architecture consists of an
application layer, a virtual layer and a physical layer. The physical layer contains endpoints such as sensors and embedded devices. The virtual layer can be a server or cloud infrastructure which delivers a combination of the following services for example, access control, storage, event management and web service interfaces. The SOA architecture proposed by Guth adds another virtual layer called the gateway layer. The gateway layer's purpose is to add the possibility of forwarding data in a different protocol to the servers. The final layer is the application layer where applications such as browsers, smartphones and devices consume services. According to Ngu, SOA based architecture are high-performing heavyweight middleware which are deployed on multiple nodes in the virtual layer. [8]

2.3.2 Actor-based architecture

Actor-based architecture is designed to be lightweight with the purpose of having a low impact on the network and server performance while providing interactivity between devices and services (see Figure 2.2). According to Ngu[8] the architecture consists of three concentric circles. The outer circle contains sensors, devices and actuators while the middle circle is used as mobile access or gateways. The innermost circle is cloud services with the host controller or middleware placed on the edge of the two other circles. The platforms computations are distributed in the network using the endpoints processing power which gives the platform lightweight properties.
In this section, other considered platforms which meet the limitations of this thesis are presented. These platforms were investigated and lightly tested but rejected due to four key factors: the difficulty of implementation, inadequacy of documentation, overall lack of a community and immaturity of the platform.

2.4.1 Zetta.js

Zetta[9] is a Node.js platform which is built on the concept of API first and takes advantage of reactive programming. Zetta uses event-driven and non-blocking I/O modules inherited from Node.js and is written in JavaScript. Zetta uses both a three and four-layer SOA based architecture and extends services with a hub acting as a gateway for this purpose.

The key concept of Zetta is the embedding of SDKs on hubs such as the BeagleBone Black, Intel Edison and the Raspberry Pi which exposes API’s for interaction with other Zetta servers or devices. Zetta servers based on this hub uses a scout service to interact with protocol specific devices which are mediating via HTTP.

Zetta offers real-time interaction with devices using WebSockets, simple cloud-based integration, registering of hub-less devices and querying for devices on a set server. Zetta also allows for an establishing of a secure
tunnelling link between two servers although there is no other form of encryption provided.

When the platform was examined, a few shortcomings of using Zetta were presented, the limited documentation and additionally, the lack of maturity in the form of working practical examples and tutorials. This might be the result of Zetta being community built and that it therefore has no official corporate backer that would ensure a minimal quality level of software.

### 2.4.2 Open connectivity Alliance (Alljoyn/Iotivity)

The Alljoyn platform was created as a result of the Allseen Alliance which was a collaboration between several companies within the field of software and electronics[10,11]. In 2016, the Allseen Alliance merged with Iotivity and their sponsors under a new name, the Open connectivity Alliance. The Open connectivity Alliance has promised that the new and unified solution will be backwards compatible and capable of supporting both the Alljoyn and Iotivity platform.

The Alljoyn platform seemed to be widely used and backed by several large corporations and was therefore considered an interesting candidate to investigate. The Alljoyn platforms architecture is based on both peer-to-peer and a three-layer SOA architecture. The peer-to-peer architecture is used for services over LAN while the SOA architecture is utilized via a Gateway Node for communication over the internet.

The key concept of Alljoyn is the implementation of devices using a distribution bus which is used as a medium to communicate between published API’s. The distribution bus technology is built on ad hoc processes where devices can leave and enter routing node points. Routing node points are built on a mesh structure where devices are linked to nodes which in turn is linked to other nodes, this allows point to point communication between devices. Devices and node points are categorized as consumers, providers and services. Alljoyn supports device discovery between devices and peer sessions.

Alljoyn offers a unique device discovery method using APIs together with sessionless signals and functions which advertises the available functionality of each node. Alljoyn implement SASL security for ad hoc communication between devices.
The Alljoyn platform was investigated and an implementation was attempted, but due to the complexity and lack of available tutorials the platform was abandoned. The decision to abandon the platform was based on the fact that the time was limited during this work and it was believed that the information had not yet had the chance to mature after the recent merger between Alljoyn and Iotivity.

2.5 Kaa

Kaa is an open-source platform designed for building end-to-end IoT solutions and is administered by KaaIoT Technologies and Cybervision Inc[12]. It is an Apache based platform which uses a web page GUI for data delivery schema creation, endpoint SDK generation and provides support for multi-tenancy on servers. The architecture used in Kaa is based on both the three and four-layer SOA model presented in section 2.3.1. This means that it is both possible to interact with endpoints via gateway-devices as well as directly address them through the Kaa server.

Kaa supports endpoint SDK implementation for development languages such as C++, C and Java. Kaa is transport-agonistic and offers support of a wide-range of network protocols for instance HTTP, CoAP, WebSockets, TCP and IBM’s asynchronous publish/subscribe protocol called Message Queue Telemetry Transport (MQTT)[13]. Kaa uses RSA and AES encryption methods for relevant network protocols.

2.5.1 Architecture

A server running the Kaa platform is called a Kaa Node which consists of three services: Control, Operations and Bootstrap services (see Figure 2.3). Kaa Nodes can also be combined to form a Kaa cluster using Apache ZooKeeper and a database [14].

The Control service is responsible for managing the overall system data, the web user interface and sending notifications to the operation service. The control service is also responsible for communication with ZooKeeper. ZooKeeper provides the control service with information about available operation services. When Kaa Nodes are running in a cluster, one node’s control service is set as active and the others are set in standby mode. If the active service become unresponsive, ZooKeeper appoints and notifies another available Nodes control service to become the new active service.
The operation service is responsible for concurrently communicating with multiple endpoints. Its primary task is to handle and process data from endpoint devices, such as registration requests and taking care of notification delivery.

The Bootstrap service is used to direct endpoints to the relevant operation service as well as providing information about connection parameters.

2.5.2 Components of Kaa-SDK

The key concept of Kaa is the endpoint SDK which is a library that provides communication, authentication, data marshalling and encryption services. The endpoint SDK is generated on the server and then embedded into an application or a device. The endpoint SDK is generated together with a user defined or a Common Type (CT) schema for data delivery. Schemas used in Kaa are based on the Apache Avro format and identified by using a uniquely Fully Qualified Name (FQN). Each schema is a representation of expected data types based upon Avro primitive types, version type and FQN. Schemas and CT are stored in a server-based repository which is called Common Type Library (CTL) that allows for reusability and version control.

Kaa Events subsystem based in the endpoint SDK is used to generate real-time events on the endpoint such as sending or receiving. Each event is based on a particular event class (EC) which use a defined CT for data delivery. EC are grouped into event class families (ECF) and registered within the Kaa tenant. The Kaa Events functionality can be accessed using the ECF factory together with specific ECF objects and relevant listeners.
2.6 Node-RED

Node-RED[15] was originally developed by IBM but was later signed over to the JS Foundation and is available under Apache 2.0 license. Node-RED is an Actor based architecture that uses Flow-based programming to wire together hardware devices, API’s and online services. Node-RED offers rapid prototyping, unique integration of social network injections, event-driven templates and real-time interaction with physical devices.

Node-RED uses common network protocols such as WebSockets, TCP and MQTT. Node-RED offers support for implementation of Bcryptjs and AES encryption methods for communication.

2.6.1 Architecture

Node-RED (see Figure 2.4) is built on top of Node.js and Google's V-8 JavaScript engine. Node-RED has the following features such as being event-driven, having a non-blocking model, single-threaded event queues and NPM. NPM is a JavaScript package manager used for downloading modules and allows for integration of other modules in the browser UI.

The browser UI is a visual representation of a flow where drag and drop node templates can be wired together. Each node template is a pre-built set of JavaScript code which is designed to perform specific functions according to user defined values or device functionality. The node templates are stored in the node palette which is divided into defined groups such as input, output or function. Input and output nodes can vary from injections to connection types such as TCP, WebSockets or MQTT. Function nodes allow for user defined functions, switches, delays and triggers for interaction between output and input nodes. The browser UI also acts as execution and
committing point for flows to the runtime.

A practical example of a design flow using Node-RED would involve an input node listening for a connection such as TCP or a Websocket on port from an android device or a browser for example. The input node would pass information to function nodes which uses the payload to extracted data and passes the information or settings to the relevant connected output nodes such as a sensor, android devices or Raspberry Pi.

2.6.2 Community

Node-RED development is community driven through GitHub. The palette of nodes in the browser UI can easily be extended from the Node-RED library while flows can be shared and imported as JSON files.

2.7 A comparative overview of Kaa and Node-RED attributes

Kaa and Node-Red use fundamentally different architecture types which results in different solutions for device discovery methods. These discovery methods are a key factor to limitations on development options such as programming languages, supported communication protocols, profiling options and levels of encryption. A general overview of the fundamental differences and attributes of each platform is exhibited in Table 2.1.

<table>
<thead>
<tr>
<th></th>
<th>Kaa</th>
<th>Node-RED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SOA</td>
<td>Actor</td>
</tr>
<tr>
<td>GUI/Dashboard</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Discovery</td>
<td>Embedding Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>Development Language</td>
<td>C++/C/Java</td>
<td>JavaScript</td>
</tr>
<tr>
<td>Encryption</td>
<td>RSA, AES</td>
<td>Bcryptjs &amp; AES*</td>
</tr>
<tr>
<td>Communication protocols (Default)</td>
<td>HTTP, CoAP, MQTT, WebSockets, TCP</td>
<td>MQTT, WebSockets, TCP</td>
</tr>
<tr>
<td>Profile and Grouping</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

* Not default user implementation
2.8 Paho framework

Eclipse Paho project provides open-source client implementation with the MQTT protocol. Paho is geared towards machine-to-machine messaging and IoT applications with the inherent need of physical and cost constraints on the used devices[16].
3 Methods and results

This chapter presents the methods and implementations used to achieve the set goals of this thesis. In order to accomplish this, an initial literature study was performed to survey different IoT architectures, protocols, standards and relevant platforms. This study became the foundation and the information gathered was then used to model the scenarios to be tested, a remote-controlled LED, a chat application and a data transmitting sensor. To test the models, prototypes were developed for each platform. These prototypes sent messages to the appropriate platform to measure the round-trip time. The resulting data is presented at the end of the chapter.

3.1 Prototypes and scenarios

The prototypes used to test the different scenarios were implemented following the documentation of the chosen platforms, Kaa and Node-RED. Although the implementation process varied for the different platforms an effort was made to make them as similar as possible.

In order to evaluate the performance and scalability of the selected platforms a series of scenarios were devised. The first scenario was a basic implementation using Raspberry Pi, where the user could turn an LED on and off remotely. The second scenario was a chat simulation that was implemented using multiple virtual android devices. The last scenario consisted of a small simulated sensor connected to a ESP8266 microcontroller.

3.2 LED scenario

The core idea of this scenario was to test the response time and performance of the platforms with an implementation of a remote-controlled LED (see Figure 3.1). This was accomplished by creating an Android application that functioned as a remote-controller. The application then sent commands such as on, off and flash via the server to the Raspberry Pi with the attached LED. For testing purposes, the prototypes were then modified to perform round-trip time message tests. Therefore, each test was designed to send a thousand single messages to the Raspberry Pi client with a fixed time interval of 5, 7, 10, 50 or 100 milliseconds intervals between each message. The Raspberry Pi client then returned the timestamped message to the remote-controller which could then calculate the RTT. These results were then stored in a
SQLite database on the remote-controller device. This was done to test the response time and capability of the platforms during varying degrees of pressure.

3.2.1 Kaa prototype

The Kaa Java SDK was generated and implemented on both the Android and Raspberry Pi devices. The Kaa EventFamily factory was used in conjunction with the generated classes in the SDK to send commands in String format from the Android device. The commands were then handled in an onEvent listener declared in the Java program running on the Raspberry Pi. The listener could then access the General-Purpose Input/Output (GPIO) via the runtime class in the Java 1.8 Library (see sample Code 3.1). In the modified version of the LED scenario an echo function was added that responded over TCP.

\[ \text{Runtime.getRuntime.exec("gpio write 4 1")} \] (3.1)

3.2.2 Node-RED prototype

For this implementation a simple solution was used, an Android device was connected to Node-REDS flow through WebSockets. Commands were then sent from the Android device to the Node-RED server which in turn forwarded the commands to the Python client on the Raspberry Pi. The Python client then used the RPI.GPIO library to access and set the value of the GPIO pin (see sample Code 3.2). For testing purposes, the prototype was modified to a Java echo client using the Neovisionaries[18] WebSocket library for communication.

\[ \text{GPIO output(4,True)} \] (3.2)
3.3 Chat scenario

The central idea of this scenario was to create a chat simulation using multiple virtual android devices to test for scalability with varying amounts of interacting devices (see Figure 3.2). The scenario was set up with two Windows machines and an Android device. The first Windows machine was hosting a virtual Linux machine with an instance of the IoT platform, the second machine hosted multiple instances of Android devices using Nox Player[17] and the separate Android device was used as an endpoint controller and data collector.

The scenario runs using the following steps, the endpoint controller (Android Host) sends 10 instant messages to all client android devices with a timestamp. Each client device then sends a reply to each other as well as returns the original timestamp messages to the controller. The endpoint controller device proceeds by saving the client device name and the roundtrip time to its SQLite Database.

3.3.1 Kaa prototype

For this implementation, Kaa Android SDK was generated and the EventFamily factory with the associated classes was used for encrypted TCP communication between all android devices. These classes are based on the message schema which contained two string values one for chatroom and the other for chat messages. The message schema is handled in the onEvent listener and messages are then redirected according to the device usage. The client host devices redirected messages to all other clients including the host, using the sendEventToAll function in the EventFamily class (see Code 3.3). The host android device stored all the returned message results in a local instance of the SQLite Database.

```java
mChatEventFamily.sendEventToAll(New Message("Chat", "Message"))
```
3.3.2 Node-RED prototype

All devices in this prototype communicated over a WebSockets protocol and used an AES encryption with a common key value on each endpoint. AES encryption was added to this prototype to allow for a fair comparison against Kaa standard encryption.

In order for Java or Android devices to send messages using WebSockets, all messages must be wrapped as JSON objects. In this prototype, the JSON object class was used to package two string values, the first value was the source name and the other a message. The entire JSON object was then encrypted alternatively decrypted using an AES cipher from Javax Crypto class on each endpoint.

![Figure 3.3 Flow wiring schema for chat simulation](image)

The Node-RED server is based on the wiring schema presented in Figure 3.3. Each message is passed to a simple function which disables the private session which enables the server to send the message to all listening clients including the original sender (see Code 3.4).

\[
\text{delete msg.} - \text{session}; \quad (3.4)
\]

3.4 Sensor scenario

This scenario was performed with a TMP36 temperature sensor and an Arduino wireless ESP8266-01 module with 1MB flash memory. The ESP8266 is a small microcontroller with a built in Wi-Fi chip. The idea behind the scenario was to see how the platforms would handle small devices that would be ideal for hooking up to sensors and perform small tasks like measuring temperature and illumination. The implementation for both platforms used MQTT over TCP.

3.4.1 Kaa prototype

Implementing a simple client on the ESP8266 proved challenging since Kaa requires all endpoints to incorporate the generated SDK. This required the
code to be cross-compiled from the C-SDK. The guide on the website was out of date and some steps were missing which resulted in hours of unnecessary troubleshooting. It was discovered that the example code for this device was disabled in the latest release which meant that the code had to be rewritten to work. There was also a problem with memory since the Kaa-SDK was a bit too large, the memory linker-files had to be adjusted for the binary to fit on the ESP8266 module.

3.4.2 Node-RED prototype

The process of implementation with Node-RED was simple. The different parts of the scenario were represented by nodes that were connected via a schema that had to be drawn in the Node-RED UI as shown in Figure 2.4. These nodes represent different operations to be performed that can be bound together with Node-RED as seen in Figure 3.3. The essential node used in this prototype was the MQTT node which represented the MQTT server on the Node-RED server. In order to communicate with the server, the ESP8266 was programmed with the Arduino interface and since Node-RED is using MQTT an external library had to be added. The external library used was the Paho project library, this library was both needed in the ESP8266 and the Android prototype.

3.5 Testing methods

All the scenarios were tested on a small Wi-Fi network in order to reduce faulty measurements and limit external interference, all connections were also properly established and stabilized before tests were performed.

3.5.1 Server specifications for all scenarios

The platforms used were Node-RED version 0.16.2 and Kaa version 0.9 installed on a virtual machine using 14.04 LTS Ubuntu 64 with 4GB RAM. The virtual machine was running on a system with Windows 10 with an Intel core I7 and 6Gb RAM.

3.5.2 LED scenario

To achieve the goal set for this scenario, the following items were used, a Raspberry Pi 3 using Raspbian GNU/ Linux, a 220Ω resistor, a LED, two male to female jumpers, breadboard and an Android Moto 5G+ device.
3.5.3 Chat scenario

This scenario uses Intel Pentium N4770 with Windows 10 and 8 Gb RAM to host the simulated Android devices with Nox Player and a single Moto 5G+ Android devices.

3.5.4 Sensor scenario

Arduino, a TMP36 temperature sensor and an external wireless ESP8266-01 module were used in order to complete the goals set for this scenario.

3.6 Results of testing

3.6.1 LED scenario results

In this scenario, a thousand single messages were sent using defined time intervals to a Raspberry Pi client. This test was done to determine the response time and capability of the platforms during varying degrees of constant pressure. Each time interval was tested ten times in order to verify the results. Table 3.1 and 3.2 represent the overall average round-trip time with defined intervals for Kaa and Node-RED. The data which these tables are based upon can be found in Attachment 1.

<table>
<thead>
<tr>
<th>Time Interval (ms)</th>
<th>100</th>
<th>50</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Round-trip time (ms)</td>
<td>33.59</td>
<td>71.62</td>
<td>125.30</td>
</tr>
<tr>
<td>STDdev</td>
<td>3.47</td>
<td>4.09</td>
<td>17.51</td>
</tr>
</tbody>
</table>

Table 3.2: Average RTT (milliseconds) for Node-RED at different sending time intervals

<table>
<thead>
<tr>
<th>Time Interval (ms)</th>
<th>100</th>
<th>50</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Round-trip time (ms)</td>
<td>31.56</td>
<td>56.57</td>
<td>114.04</td>
</tr>
<tr>
<td>STDdev</td>
<td>8.38</td>
<td>3.32</td>
<td>5.38</td>
</tr>
</tbody>
</table>

Figure 3.4 and 3.6 show the average round-trip time for 5 and 7 millisecond sending intervals over sequential order of the packets for both Kaa and Node-RED. Figure 3.5 and 3.7 presents a single sample 1000 messages with a 10/50/100 millisecond interval over sequential order of the packets for both Kaa and Node-RED.
Figure 3.4. Average RTT for Kaa when 1000 messages are sent with a 5 & 7 millisecond interval

Figure 3.5. RTT for Kaa when 1000 messages are sent with a 10/50/100 millisecond interval
Figure 3.6. Average RTT for Node-RED when 1000 messages are sent with a 5 & 7 millisecond interval

Figure 3.7. RTT for Node-RED when 1000 messages are sent with a 10/50/100 millisecond interval
3.6.2 Chat scenario results

In this scenario, 10 instant messages with timestamps were sent to a set number of virtual android clients. The messages were then returned to the original host device as a response. Each test was performed ten times with a different number of devices after which the average round-trip time was documented. This information was then used to compile the average round-trip time and standard deviation for the different tests, see Table 3.3. The data from the performed tests can be found in Attachment 2.

Table 3.3 and 3.4 present the final results of round-trip and standard deviation for Kaa and Node-RED prototypes respectively. Figure 3.8 is consolidated of those results in table 3.3 and 3.4 using average round-trip per device.

<table>
<thead>
<tr>
<th>No of Devices</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Round-Trip Time (ms)</td>
<td>279.40</td>
<td>361.00</td>
<td>510.10</td>
<td>770.10</td>
<td>964.60</td>
</tr>
<tr>
<td>STDdev</td>
<td>41.57</td>
<td>58.09</td>
<td>88.25</td>
<td>113.24</td>
<td>124.92</td>
</tr>
</tbody>
</table>

Table 3.4: Node-REDs results from the 10 message bursts with two to six devices

<table>
<thead>
<tr>
<th>No of Devices</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Round-Trip Time (ms)</td>
<td>146.20</td>
<td>193.50</td>
<td>241.50</td>
<td>285.20</td>
<td>344.30</td>
</tr>
<tr>
<td>STDdev</td>
<td>41.65</td>
<td>46.99</td>
<td>50.28</td>
<td>53.49</td>
<td>61.51</td>
</tr>
</tbody>
</table>

Figure 3.6. Average RTT for two to six devices with Kaa and Node-RED
3.6.3 Sensor scenario results

This scenario was intended to be tested with an ESP8266 microcontroller. The microcontroller was coupled with a temperature sensor, the readings would then be forwarded to the IoT platform. The reading was then echoed back in order to determine the round-trip time.

The scenario was easily implemented in Node-RED but proved to be extremely difficult to implement in Kaa. This difficulty with implementing Kaa on the microcontroller eventually left it unsuccessfully completed. This undesired outcome could thus not be used to compare the performance, although this result exposed some shortcomings with Kaa. The lack of community, tutorials and information on how to implement Kaa on the microcontroller was the largest contributing factor to this difficulty. The guide that was provided on Kaa’s website[21] is missing some key information such as libraries that need to be installed as well as having invalid links to the example code for the microcontroller. The guide also omitted to mention that the built-in encryption also had to be disabled for the ESP8266 to work. The implementation was eventually uncompleted when the microcontroller started throwing fatal exceptions when attempting to send messages.

The complexity of Kaa’s procedure is the result of the need to incorporate the SDK on the device. In order to achieve this the C-SDK had to be cross-compiled to the microcontroller. Node-RED can use the Arduino standard IDE to program the microcontroller with the drawback that many features have to be implemented by the developer on the device such as device discovery and encryption.
4 Analysis and discussion

This chapter presents an analysis and a general discussion of the result from the previous chapter. Each scenario is evaluated and differences between the platforms are discussed. The chapter concludes with environmental and ethical concerns.

4.1 LED scenario

The purpose of this scenario was to evaluate message delivery capabilities of both platforms under constant pressure. The results demonstrated a high latency for Kaa at 5 milliseconds sending interval with one test sample showing a 32 seconds round-trip time on its final message. Node-RED revealed a proportional increasing at 5 and 7 milliseconds sending intervals which in turn would lead to a system slow down over time. Both platforms performed at relative similar levels and stable results at 10 milliseconds intervals with Node-RED average round-trip at about 114 milliseconds and Kaa at about 125 milliseconds intervals.

A sample result for Kaa at the 50 millisecond interval and one for Node-RED at the 100 millisecond interval presents large time spikes around 300 and 600 messages respectively (see marked areas in Figure 4.1 and 4.2). These spikes may be caused by factors such as extra CPU usage on either endpoint, although this shows us something much more important, the platforms ability to recover. This is a key factor when dealing with unpredictable real world network factors such as geographical location, time of day or day of week[19].
Using the above results, one could theoretically predict that both Kaa and Node-RED would be able to easily maintain and run 10 devices at 100 milliseconds intervals.

Another key result is the small difference in round-trip time between Kaa and Node-RED at 10, 50 and 100 milliseconds. An interesting result as Node-RED has no form of encryption or security for this prototype while Kaa
offers full end-to-end security. This factor can be considered negligible due to the small packet size but would be a relevant aspect with large data types.

### 4.2 Chat scenario

The purpose of this scenario was to simulate a chatroom with various number of Android devices. The tests revealed that a single burst of 10 messages across a simple Wi-Fi network resulted in the round-trip time moving towards 1 second with 6 Devices for Kaa. While Node-RED results showed a low but constant increase in round-trip time as the number of devices increased. A clear and developing curve can be created using trend lines (see Figure 4.3). According to J.Nielsen[20] a response time of 1 second gives the user the impression of a small delay and can therefore be considered a relevant benchmark for a chat simulation. Using the 1 second mark as a determinant value, one could theoretically plot scalability for Kaa at 6 or 7 devices while Node-RED could possibly have 12 or more devices.

![Figure 4.3. Round-trip time from chat-scenario with trendlines](image)

### 4.3 Sensor scenario

The sensor scenario was unsuccessful and could therefore not be used for comparing the performance between the platforms, but it provided valuable
insight about the difference in complexity between the platforms. The reason the scenario was left uncompleted was the inability to determine why the microcontroller started throwing fatal exceptions when calling Kaa’s send function. A theory was that the power consumption exceeded the power supply’s output but that was proven not to be the case when an alternate power-converter was constructed. It could also have been that the RAM buffer was insufficient, but this could not be determined within the time frame of this work. To solve this, further investigation and understanding of the microcontroller and Kaa was needed.

4.4 Environmental and ethical considerations

A requirement from KTH was to investigate the potential ethical, social, economic and environmental impact associated with the results from this thesis.

The ethical concern in this thesis has been identified to be the subject of privacy and security of data. Although this subject has not been thoroughly investigated in this work, minor suggestions are presented in the conclusion in chapter 5.

The potential social impact of this work is considered to be rather small, although it might help future users to identify the appropriate platform for their situation. Beyond this, the social impact of IoT in general is considered to be huge, affecting everyday life with seamless integration of devices.

The economic consequences are not significant since the platforms used are all open-source and cost free. Although a small economic gain could be achieved by choosing a less resource intensive platform as investigated in this work.

The environmental aspect of this work is limited to power usage which is not investigated in this work and therefore is inconsequential.
5 Conclusion

In this work scalability and reliability was evaluated for the two platforms Kaa and Node-RED through three different scenarios. The results from the tests revealed several distinct differences in their performance and in their potential for scalability. The results from the LED scenario revealed that both platforms had the same performance when messages were sent at 10 milliseconds intervals, however it is worth mentioning that the communication was encrypted with Kaa but not with Node-RED. A realistic interval such as 100 milliseconds for both platforms could be used when dealing with external interference and other clients. When examining the results from the chat scenario, a significant difference between Kaa and Node-RED could be observed. Kaa struggled to handle 6 devices while Node-RED performed considerably better. Through analysing trendlines, it could be theorized that Kaa could handle approximately 6 or 7 devices when Node-RED would manage 12 or more without exceeding a one second delay. The sensor scenario, although uncompleted, revealed the complexity of Kaa. This complexity combined with other factors such as the shortcomings in community, tutorials and general information could discourage using the platform.

5.1 Platform review

The platforms provided by Kaa and Node-RED offers two different architectural solutions, although both have their own strengths and weaknesses.

Node-RED offers a simple, easy, lightweight IoT solution and a user-friendly UI as well as code generation through its wiring schemas and flows. These aspects make Node-RED perfect for home environments and hobby developers. The preferred method of communication with Node-RED is WebSockets which utilizes a small packet size over TCP, although the lack of implemented security as a default is a concern it is not the main focus of this thesis. Node-RED has crowd-sourcing which give the developers a wide range of unique solutions with a quick turnaround which is key to its current popularity.

Kaa's key feature is a complete end-to-end implementation, although this comes with varying costs in the performance to the endpoints. Kaa offers better scalability as it can run in cluster format but with a big footprint at the
network's edge. Kaa is perfect for large corporate solutions where performance, reliability and security are important factors.

Another important feature worth comparing is the possibility and simplicity of communicating with external APIs. The Kaa platform seems unable to directly communicate with external APIs and it is expected to be handled by the endpoints instead, by using a gateway. This process is significantly easier in Node-RED where nodes in the flow can be used to send Representational State Transfer (REST) messages to external APIs directly.

5.2 Overall conclusion

With the results from the scenario-based testing and the review of the platforms an overall conclusion can be formed. Node-RED has a much less complex process of implementation compared to Kaa which became especially clear during the implementation of the sensor-scenario. However, the complexity of Kaa is due to the array of features provided such as device discovery, encryption, clustering and marshalling of data. As a result, Kaa is required to perform more work and this was assumed to be the reason that Node-RED performed better in the LED and chat scenario. This suggests that Node-RED is both faster and easier to use than Kaa and would therefore be the preferred platform unless certain specifications are required.

The fact that Kaa supports clustering and Node-RED does not, indicates that even though Kaa proved to be slower than Node-RED during heavier traffic, the Kaa platform should be chosen for large installations while Node-RED should perform better in smaller environments such as home-networks or small business-networks.

5.4 Future works

Although, three different scenarios were examined in this thesis there are still many aspects concerning scalability and reliability that could be researched. One possibility for future works would be to examine Kaa's clustering ability. Clustering increases the traffic capacity and could potentially increase the performance of Kaa substantially. Another possible area to investigate is the security of these platforms. For example, Bcryptjs could be used to implement secure WebSockets in Node-RED which would then be compared with the security provided by Kaa. Furthermore, other platforms could also be investigated and evaluated, for example the platform
provided by the Open Connectivity Foundation that were previously mentioned but not used in this work.
References


Attachments

Attachment 1: The raw average results from the tests conducted on Node-RED and Kaa using 1000 messages sent at different sending intervals. The results values are presented in milliseconds and based on the total average round-trip time. The table is also showing the mean and standard deviation.

<table>
<thead>
<tr>
<th>Node-RED</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval</td>
<td>100</td>
<td>50</td>
<td>10</td>
</tr>
<tr>
<td>Mean</td>
<td>31.56</td>
<td>56.57</td>
<td>114.04</td>
</tr>
<tr>
<td>Stdev</td>
<td>8.38</td>
<td>3.32</td>
<td>5.38</td>
</tr>
<tr>
<td>1</td>
<td>27.24</td>
<td>62.92</td>
<td>107.73</td>
</tr>
<tr>
<td>2</td>
<td>31.74</td>
<td>53.66</td>
<td>121.74</td>
</tr>
<tr>
<td>3</td>
<td>26.98</td>
<td>54.03</td>
<td>109.89</td>
</tr>
<tr>
<td>4</td>
<td>24.63</td>
<td>55.53</td>
<td>115.12</td>
</tr>
<tr>
<td>5</td>
<td>23.08</td>
<td>54.09</td>
<td>117.93</td>
</tr>
<tr>
<td>6</td>
<td>23.81</td>
<td>60.78</td>
<td>108.91</td>
</tr>
<tr>
<td>7</td>
<td>36.46</td>
<td>59.86</td>
<td>109.34</td>
</tr>
<tr>
<td>8</td>
<td>45.86</td>
<td>55.06</td>
<td>117.29</td>
</tr>
<tr>
<td>9</td>
<td>30.71</td>
<td>54.66</td>
<td>110.73</td>
</tr>
<tr>
<td>10</td>
<td>45.11</td>
<td>55.11</td>
<td>121.67</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kaa</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval</td>
<td>100</td>
<td>50</td>
<td>10</td>
</tr>
<tr>
<td>Mean</td>
<td>33.59</td>
<td>71.62</td>
<td>125.30</td>
</tr>
<tr>
<td>Stdev</td>
<td>3.47</td>
<td>4.09</td>
<td>17.51</td>
</tr>
<tr>
<td>1</td>
<td>31.94</td>
<td>71.44</td>
<td>121.31</td>
</tr>
<tr>
<td>2</td>
<td>31.48</td>
<td>67.52</td>
<td>120.74</td>
</tr>
<tr>
<td>3</td>
<td>32.14</td>
<td>76.8</td>
<td>143.63</td>
</tr>
<tr>
<td>4</td>
<td>31.08</td>
<td>66.33</td>
<td>115.12</td>
</tr>
<tr>
<td>5</td>
<td>29.53</td>
<td>69.11</td>
<td>143.6</td>
</tr>
<tr>
<td>6</td>
<td>35.67</td>
<td>72.67</td>
<td>107.61</td>
</tr>
<tr>
<td>7</td>
<td>41.67</td>
<td>70.81</td>
<td>109.34</td>
</tr>
<tr>
<td>8</td>
<td>32.13</td>
<td>73.31</td>
<td>159.29</td>
</tr>
<tr>
<td>9</td>
<td>34.93</td>
<td>79.38</td>
<td>110.73</td>
</tr>
<tr>
<td>10</td>
<td>35.33</td>
<td>68.79</td>
<td>121.67</td>
</tr>
</tbody>
</table>
Attachment 2: Raw results from the tests conducted on Node-RED and Kaa using 10 message bursts with two to six devices. The resulting RTT-values are presented in milliseconds. The table is also showing the mean and standard deviation.

<table>
<thead>
<tr>
<th>Node-RED</th>
<th>Devices</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>146.20</td>
<td>193.50</td>
<td>241.50</td>
<td>285.20</td>
<td>344.30</td>
<td></td>
</tr>
<tr>
<td>Stdev</td>
<td>41.65</td>
<td>46.99</td>
<td>50.28</td>
<td>53.49</td>
<td>61.51</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>165</td>
<td>119</td>
<td>283</td>
<td>329</td>
<td>269</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>126</td>
<td>171</td>
<td>187</td>
<td>258</td>
<td>309</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>138</td>
<td>240</td>
<td>179</td>
<td>229</td>
<td>271</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>122</td>
<td>148</td>
<td>172</td>
<td>244</td>
<td>337</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>168</td>
<td>165</td>
<td>186</td>
<td>210</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>190</td>
<td>159</td>
<td>186</td>
<td>273</td>
<td>380</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>82</td>
<td>230</td>
<td>200</td>
<td>343</td>
<td>312</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>81</td>
<td>147</td>
<td>243</td>
<td>263</td>
<td>413</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>110</td>
<td>236</td>
<td>243</td>
<td>258</td>
<td>294</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>172</td>
<td>164</td>
<td>264</td>
<td>223</td>
<td>305</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>108</td>
<td>156</td>
<td>272</td>
<td>222</td>
<td>303</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kaa</th>
<th>Devices</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>279.40</td>
<td>361.00</td>
<td>510.10</td>
<td>770.10</td>
<td>964.60</td>
<td></td>
</tr>
<tr>
<td>Stdev</td>
<td>41.57</td>
<td>58.09</td>
<td>88.25</td>
<td>113.24</td>
<td>124.92</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>223</td>
<td>323</td>
<td>449</td>
<td>748</td>
<td>878</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>220</td>
<td>349</td>
<td>332</td>
<td>749</td>
<td>863</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>267</td>
<td>390</td>
<td>450</td>
<td>626</td>
<td>714</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>294</td>
<td>316</td>
<td>469</td>
<td>640</td>
<td>884</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>231</td>
<td>395</td>
<td>477</td>
<td>784</td>
<td>863</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>280</td>
<td>391</td>
<td>455</td>
<td>718</td>
<td>1028</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>266</td>
<td>308</td>
<td>505</td>
<td>769</td>
<td>872</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>285</td>
<td>337</td>
<td>556</td>
<td>818</td>
<td>920</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>226</td>
<td>240</td>
<td>555</td>
<td>633</td>
<td>831</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>278</td>
<td>308</td>
<td>367</td>
<td>597</td>
<td>871</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>224</td>
<td>343</td>
<td>486</td>
<td>619</td>
<td>922</td>
<td></td>
</tr>
</tbody>
</table>