Master's Thesis

Comparison of IoT frameworks for the smart home

Alexander Larsson
Erik Nimmermark

Department of Electrical and Information Technology,
Faculty of Engineering, LTH, Lund University, 2016.

Comparison of IoT frameworks for the smart home

Alexander Larsson dat11ala@student.lu.se
Erik Nimmermark fpr05eni@student.lu.se

Anima AB
Västra Varvsgatan 19, Malmö

Advisor: Maria Kihl, EIT

June 15, 2016

Printed in Sweden
E-huset, Lund, 2016

Abstract

The smart home landscape is rapidly evolving right now. There is yet to be an industry-wide common standard and several organizations are introducing their own proposed solutions. This has led to fragmentation and the field can appear confusing to interested parties trying to enter the market. There is a need to evaluate the different technologies to discern their distinguishing characteristics.

This thesis report provides a comparison between HomeKit and IoTivity based on our proposed evaluation model. The model identifies a set of relevant characteristics which are evaluated through operationalization of criteria into composite measurables. Our comparison shows that at this time there is no definite answer to what framework to choose. Developers will have to compromise and make a decision based on the type of product they want to make. The proposed model can readily be used as a tool to evaluate any Internet of Things framework and provide guidance when choosing what technology to use.

Populärvetenskaplig sammanfattning

Framtidens smarta hem

Idag finns det uppskattningsvis strax under 5 miljarder Internet of Things enheter i världen. De kommande fem åren beräknas den siffran växa till 15 miljarder ocn en stor del av dessa kommer finnas inom smarta hem. För att smarta hem ska nå sin fulla potential krävs att man enas om en gemensam standard.

Smarta hem idag

Drömmen om det smarta automatiserade hemmet där kaffebryggaren sätts på lagom till att alarmet ringer och garageporten öppnas av sig själv när man kör upp på uppfarten har funnits länge.

Allt detta kan idag lösas med hjälp av Internet of Things, problemet inom området är att man inte riktigt kan enas om **hur** det ska lösas. Det finns ett flertal olika tekniska lösningar på marknaden men för att det smarta hemmet ska fungera så måste alla uppkopplade prylar tala samma språk.

Alla de äldre etablerade lösningarna har olika brister. Vissa är specialiserade på ett specifikt område medan andra saknar radiotekniker för att kommunicera med vanliga uppkopplade enheter så som mobiltelefoner eller datorer. Dessa standarder är också helt inkompatibla med varandra, vilket gör det svårt för konsumenter och företag att veta vilken lösning de ska välja. På grund av de här bristerna har det smarta hemmet inte slagit igenom lika stort som man tidigare trodde i branschen.

Visionen

I visionen om det framtida smarta hemmet är allting uppkopplat. Termostaten sänker automatiskt värmen i huset när familjen är borta för att spara ström. Tvättmaskinen pratar med elnätet och tvättar då elpriset är som lägst. Kylskåpet beställer automatiskt hem basvarar så som mjölk och ägg i takt med att de tar slut.

I visionen ingår även att gemene mans hem är ett sådant smart hem. För att detta ska bli verklighet krävs det en gemensam standard. En sådan gemensam standard skulle göra det enkelt för konsumenter att köpa smarta hem produkter som fungerar med varandra. På så sätt blir det möjligt för konsumenterna att bygga ut sitt smarta hem med mer och mer prylar i en takt som passar dem.

Utvärdering

Under den senaste tiden har ett flertal förslag till sådana gemensamma standarder presenterats. Det är stora teknikföretag så som Google, Apple och Intel som ligger bakom dessa. I det här arbetet har vi undersökt några av de här teknologierna med avseende på de viktigaste egenskaperna för smarta hem produkter. Sådana egenskaper är till exempel vilka radioteknologier som går att använda och hur snabbt de reagerar på kommandon. Vår utvärdering visar att de olika lösningarna presterar ungefär lika bra rent tekniskt baserat på kriterier så som vilken svarstid de har och hur de kommunicerar med andra liknande enheter. De punkter där de skiljer sig åt är dels tillgängligheten i form av hur man kan styra sina enheter och vilken typ av produkter som utvecklarna av teknologin tillåter rent juridiskt. Det är i dagsläget svårt att säga vilken av lösningarna som kommer att vinna kriget om konsumenterna i slutändan. Några av de mer nischade ramverken kommer sannolikt att leva kvar en lång tid efter att en övergripande standard har etablerats.

Acknowledgements

We would like to thank Anima for giving us the chance to write this thesis. More specifically, we would like to thank all members of Anima Greenhouse for supporting and providing us with all needed resources. A special thanks goes out to Aleksandar Rodzevski, our supervisor at Anima, for his extensive support. Finally, we would also like to thank our supervisor at LTH, Maria Kihl.

Table of Contents

List of Figures

List of Tables

Acronyms

ACL Access Control List. 18

ADC Analog-to-digital converters. 30, 31

AEAD Authenticated Encryption with Associated Data. 26

AES Advanced Encryption Standard. 26

AJSCL AllJoyn Standard Core Library. 5

AJTCL AllJoyn Thin Core Library. 5

AOSP Android Open Source Project. 10

AP Access Point. 16

API Application Programming Interface. 9, 14, 19, 21–23, 28, 36, 38, 41, 49

ATT Attribute Protocol. 23

BDK Brillo Development Kit. 10

BLE Bluetooth Low Energy. 6, 9, 23, 25, 27–29, 31, 39

CBOR Concise Binary Object Representation. 4, 13, 18, 49, 50

CoAP Constrained Application Protocol. 4, 11–13, 15, 18, 30, 32, 41, 46, 49, 50

CRUDN create, read, update, delete and notify. 15

DTLS Datagram Transport Layer Security. 7, 13, 18, 40

GATT General Attribute Profile. 23, 25

GPIO General-purpose input/output. 27, 28, 30

GUI Graphical user interface. 29, 30, 33

HAP Homekit Accessory Protocol. 21, 23, 25, 26, 29, 45

HAT HomeKit Accessory Tester. 28, 41

HKDF HMAC-based Key Derivation Function. 26

HMAC Hashed Message Authentication Code. 26

HTTP Hypertext Transfer Protocol. 11–13, 18, 23, 45, 50

IoT Internet of Things. 3, 4, 10–13, 16, 18, 19, 29, 35–39, 45, 49–51, 53

IP Internet Protocol. 7, 23–25

JSON JavaScript Object Notation. 12–14, 23, 24, 50

KDF Key Derivation Function. 26

L2CAP Logical Link Control and Adaptation Protocol. 23

MAC Media Access Control. 7

MFi Made for iPhone/iPod/iPad. 26, 38

NFC Near Field Communications. 27

OCF Open Connectivity Foundation. 3, 4, 11, 14–19, 28, 32, 38–41, 49–51, 53

OIC Open Interconnect Consortium. 11

OS operating system. 10, 32, 38, 51

OTA Over-the-air programming. 37, 40, 50, 51

PLC Power-line communication. 5

RAML RESTful API Modeling Language. 14

REST Representational State Transfer. 9, 11, 12, 14, 15

SDK Software Development Kit. 28, 39

SHA Secure Hash Algorithm. 26

SiP System in package. 27

SoC System-on-a-Chip. 27, 40

SPA Single-page application. 30, 32

SRM Secure Resource Manager. 18

SRP Secure Remote Password. 26

SSID Service Set Identifier. 16, 31

TB Thin Block stack. 18, 19

TCP Transfer Control Protocol. 12, 13, 23, 25, 45, 50

TLS Transport Layer Security. 13

UB Unified Block stack. 18, 19

UDP User Datagram Protocol. 7, 12, 18, 45

URI Uniform Resource Identifier. 11, 12, 15

URL Uniform Resource Locator. 23

WLAN Wireless Local Area Network. 9, 27–30

WWDC Apple Worldwide Developers Conference. 21

XML Extensible Markup Language. 12

XMPP Extensible Messaging and Presence Protocol. 17, 39

Chapter 1

Introduction

1.1 Background

The Internet of Things (IoT) market is growing rapidly right now and most of that growth is focused on IoT solutions for the smart home. Many companies and industry groups are announcing their own IoT solutions for the smart home which has lead to major fragmentation and confusion regarding what solution to use. Because of this, there is a need to evaluate the different solutions available based on different performance characteristics.

This thesis was conducted at Anima Connected, a newly started company based in Malmö. They are operating in the IoT space and are interested in investigating different IoT technologies for use in future smart home products.

1.2 Project scope

The goal of this thesis project is to evaluate the IoT frameworks provided by Apple and the Open Connectivity Foundation (OCF). Apple's framework refers to HomeKit and OCF's framework refers to IoTivity. A comparison of the evaluated frameworks is performed to help hardware manufacturers when choosing between different frameworks for their smart home products.

A method for evaluating IoT frameworks is designed in order to be able to perform this comparison. Prototypes based on the different frameworks are implemented in order to get first-hand experience of the different frameworks. Performance measurements are performed on the prototypes in order to help the evaluation of the frameworks.

1.2.1 Limitations

The evaluation and comparison of the IoT frameworks are limited to a local connectivity setting. The power usage of the prototypes is not evaluated since the hardware platforms can not be considered equal in this respect. The power usage would have been tied to the hardware platform used rather than the IoT frameworks used.

1.3 Method

A literature study was performed in order to gain some initial knowledge of the smart home area. This study included both academic papers as well as online resources related to the smart home. Smart home solutions such as Alljoyn, Zigbee, IoTivity and HomeKit was studied. As well as other relevant technologies including Constrained Application Protocol (CoAP), Concise Binary Object Representation (CBOR) and Thread. Lastly, reference implementations of IoTivity and HomeKit was studied to gain an understanding on how to implement the prototypes.

A use-case relevant to Anima's current products was developed. Prototypes that realized this use-case was implemented in each framework and an evaluation model for comparing smart home IoT frameworks was designed. The evaluation model was used to perform a comparison of the two IoT frameworks.

1.4 Thesis outline

Chapter 2 presents relevant work done by the AllSeen Alliance, ZigBee Group, Thread Group and Google.

Chapter 3 introduces the OCF and their work developing an open specification for the IoT. Additionally, their open source implementation of this specification, IoTivity, is described.

Chapter 4 describes Apples smart home solution HomeKit.

Chapter 5 presents the prototypes that where developed as part of this project. The software tools and hardware platforms used are also described.

Chapter 6 describes the evaluation method used and presents the results of the evaluation.

Chapter 7 presents an analysis of the results of the evaluation. Additionally, the evaluated frameworks are discussed from a market perspective.

Chapter 8 provides a summary of this report and presents our conclusion.

Chapter 2

Related Work

One of the most exciting emerging technologies right now is the Internet of Things (IoT). The IoT consists of everyday objects that has been enhanced by connecting them to the Internet. This is made possible by equipping these devices with an embedded hardware platform. The IoT will enable companies to innovate and create better versions of existing product as well as make way for completely new product categories.

In 2021 there will be 28 billion connected devices out of which 15 billion will be IoT devices [1]. Today, there are a little under 5 billion IoT devices which means that the Internet of Things will grow with 10 billion devices in five years. A big part of this growth will come from the smart home market.

As a response to this projected growth, big industry players are introducing application frameworks, operating systems and low-powered network technologies for the smart home. These technologies are largely incompatible and none of them has emerged as a clear choice for developing IoT applications and devices.

2.1 AllJoyn

The AllJoyn protocol was originally developed by Qualcomm but was later signed over to the Linux Foundation. It is now a collaborative open source framework managed by the AllSeen Alliance which comprises more than 200 companies and the premier members include giants such as Microsoft and Electrolux.

An AllJoyn network consists of AllJoyn Apps and AllJoyn Routers, where every App is connected to a single Router. A physical device can host one or multiple Apps as well as one or several Routers. It is also possible to have a device host one or several Apps without having a Router, in this case a router on another physical device is used by the App(s). An example network is depicted in Figure 2.1. There are two versions of the AllJoyn framework, the AllJoyn Standard Core Library (AJSCL) and AllJoyn Thin Core Library (AJTCL), where AJTCL is intended to be used in resource-constrained embedded devices. The main drawback of the AJTCL is that it does not include a router daemon, so devices running the thin client has to utilize the router of an adjacent AJSCL device. The connection between the router hosting the bus segment and the device running AJTCL is made through TCP. The apps do not communicate directly with each other,

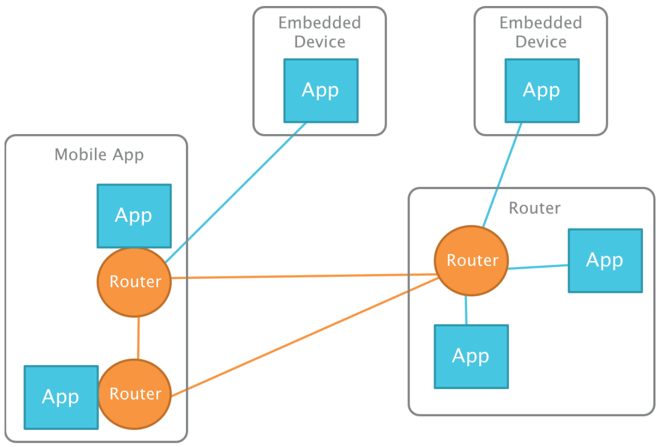


Figure 2.1: An example of an AllJoyn network.

instead, all communication is handled by the Routers which are used to form a virtual bus. This abstraction of the communication makes it appear to each app that it is communicating with a single entity when in fact it might be connected to a large Router network [2, 3].

The framework currently supports ethernet, serial, Power-line communication (PLC) and Wi-Fi. In other words, the framework does not have support for any of the low power wireless technologies such as Bluetooth Low Energy (BLE) or IEEE 802.15.4 although this might come in the future [4].

2.2 ZigBee

Zigbee is a wireless network standard with a focus on low cost and energy consumption. It is developed by the ZigBee Alliance and having been around since 1998 it is one of the most widespread IoT solutions with over 1,000 certified products [5].

The ZigBee protocol stack is built upon IEEE 802.15.4, which means that all devices running ZigBee needs to include a 802.15.4 chip. As 802.15.4 chips are usually not available in mobile phones or home networks a ZigBee gateway is needed in order to connect the ZigBee network to the cloud or a mobile device.

ZigBee networks are composed of three different types of devices: ZigBee Coordinators, ZigBee Routers and ZigBee End Devices which are connected in a mesh topology. The Coordinators control the security and formation of the network, the Routers extend the networks range and End Devices performs control functions and/or collects sensor data. A device often has multiple functions, a Router for example can be contained in a thermostat or switch, and a Coordinator in a set-top box [6].

ZigBee provides three different network specifications, being ZigBee PRO, ZigBee IP and ZigBee RF4CE. ZigBee PRO is what most ZigBee devices are based upon and is considered "standard" ZigBee. ZigBee IP is as one might suspect

IP-based and uses IPv6 addressing. This comes with the benefits of an almost infinite number of addresses and easier integration with other IP-based networks. ZigBee RF4CE is intended to be used in remote controls and similar devices. It uses a star topology with two different kind of devices, controllers and targets. While ZigBee PRO and ZigBee IP are not interoperable, RF4CE and PRO uses the same communication foundation and RF4CE devices are easily bridged to work with other ZigBee devices [7, 6].

2.3 Thread

Thread is a low-powered mesh network protocol developed by the Thread Group. It is founded by Alphabet owned company Nest together with other big companies such as ARM, Qualcomm and Samsung. In total, they have over 200 member companies to date [8].

From a technical point of view, Thread is mostly a combination of existing standards combined into a network stack that can be seen in Figure 2.2. It uses IEEE 802.15.4 at the physical and Media Access Control (MAC) layer. Thread is IPv6 based and uses 6LowPAN to enable Internet Protocol (IP) routing over 802.15.4. It uses User Datagram Protocol (UDP) for messaging and security is provided by Datagram Transport Layer Security (DTLS) together with 802.15.4 MAC security.

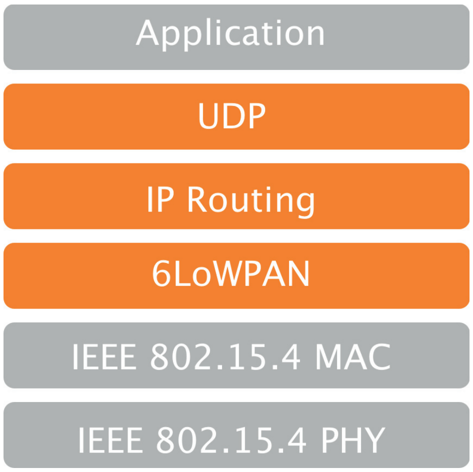


Figure 2.2: An overview of the Thread stack

Conceptually, a Thread network is made up of devices with different roles.

Routers	Route messages and provide security services for devices that try to join the network.
REEDs	Router-eligible End Devices, a device that can act as a router. The network collectively decides what REEDs should be acting as routers at any given time.

Border Router	A special kind of router that has the capability to connect the Thread 802.15.4 network with other kinds of networks such as Wi-Fi and Ethernet.
Sleepy End devices	Host devices that communicate with the rest of the network through its host router.

The network is designed in such a way that there is no single point of failure. If a host device's router goes down the host will select a new router. If there are any REEDs available in the network, one of them might be promoted to acting as a router. In this way, the network can make its own decisions and self heal without user action. There is however some scenarios when the network wont be able to do this. One such scenario is when a network consists of a single router and it goes down [9].

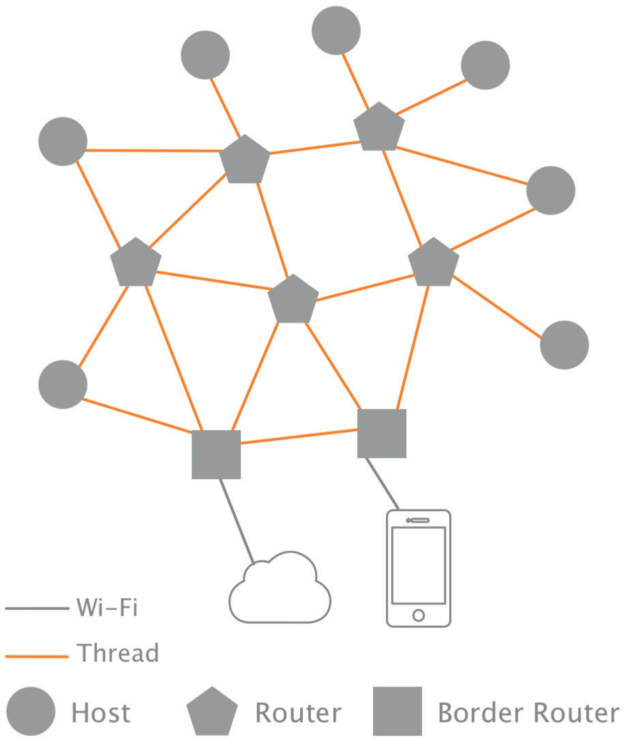


Figure 2.3: An example of a possible Thread mesh network. The different roles of routers, border routers and hosts are shown.

2.4 Brillo and Weave

Weave and Brillo are two technologies that are currently being developed by Google. Both of these technologies are currently under development, with an early access program available for invited developers.

2.4.1 Weave

Weave is an application layer protocol that can be used to interact with devices. It currently supports Wireless Local Area Network (WLAN) as its only transport with support for BLE under development. The protocol is part of a larger framework that can be divided into three main components. These are listed below and can also be seen in Figure 2.4.

Cloud	The Weave cloud service enables support for things like remote access and Representational State Transfer (REST) Application Programming Interfaces (APIs).
Device	An open source C++ library called "libeweave" that aims to make it easy for manufacturers to build devices.
Client	Libraries are available for mobile devices (iOS and Android) and the web. These enable the clients to do things such as controlling devices and reading a device's state either locally or remotely.

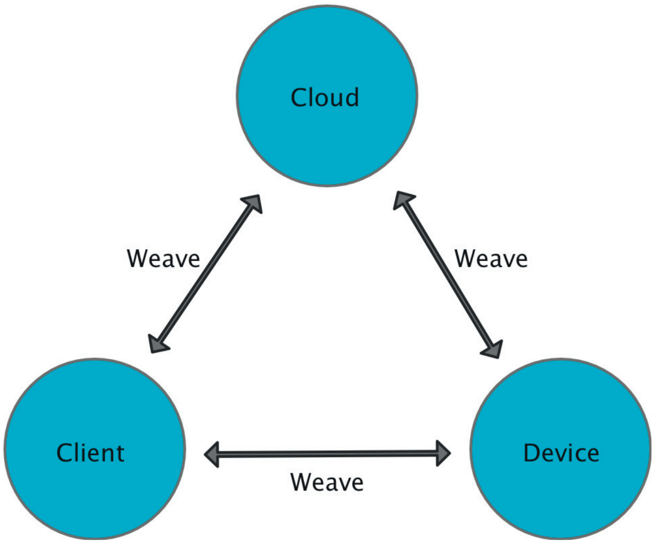


Figure 2.4: The three main components of the Weave framework. Communication between these components is done using the Weave protocol.

Weave is designed with the end users in mind and this shows in a number of ways. First, it supports device discovery and provisioning that is designed to make it simple for the end user to set up a new device in their homes. If the device is a WLAN device for example, the user can add it to their network without typing any passwords. It also supports sharing of devices with family members and friends and the ability to define different levels of access for different people. Lastly, a user that has access to a device can define what apps and services will get access to these devices. This will work similarly to how users grant applications access to functionality in today's smart phones e.g. the camera.

Interoperability is provided by using schemas. These schemas define common classes of devices in a consistent way. This makes it so that two different devices based on the same schema can be interacted with in exactly the same way. In order to enable innovation of new kinds of products, these schemas are also extensible to enable new functionality to be defined [10].

2.4.2 Brillo

Brillo is an operating system (OS) for the IoT with built in support for Weave. It is based on Android but has been stripped down to only contain the most essential parts needed for connected embedded devices. Since it is part of the Android Open Source Project (AOSP), it is available under an open-source license.

The OS is designed to make it very easy for manufacturers to make Weave enabled devices. To attain this goal, it provides a number of benefits for developers [10].

Security	The OS is secured by various methods such as a verified boot architecture and software fault isolation. In addition to this, Google will also provide security fixes for the OS.
BDK	The Brillo Development Kit (BDK) provides an embedded development environment using adb and fastboot.
Updates	Built-in support for sending updates over the internet to devices that are already in use by users.
Metrics	Manufacturers can get information on how their devices are performing in the field. They also get access to crash reporting data that enables them to fix bugs quickly.

Chapter 3

OCF and IoTivity

The Open Connectivity Foundation (OCF), formerly known as Open Interconnect Consortium (OIC), is an industry group created in 2014 whose goal is to create a specification for the IoT enabling communication between devices of different brands and regardless of what physical transport is used. The organization has over 160 member companies including Microsoft, Intel and Samsung. Qualcomm, the company that founded the AllSeen Alliance is also in the list of Diamond members [11, 12].

IoTivity is an open source project building a framework implementing the specification defined by the Open Connectivity Foundation (OCF). Therefore, the term "OCF framework" refers to the specification and "IoTivity framework" refers to the open source implementation.

3.1 Technologies used

There are a number of technologies used by OCF and IoTivity that are quite new and therefore needs an introduction. Some background on these technologies are presented below.

3.1.1 CoAP

The Constrained Application Protocol (CoAP) is a lightweight alternative to Hypertext Transfer Protocol (HTTP) designed for use in embedded devices such as those found in the Internet of Things (IoT). In order to understand CoAP, some knowledge of HTTP and REST is required [13].

REST

The architecture of the web is frequently described as REST and is traditionally supported by the use of HTTP. REST provides a loosely coupled application layer architecture. A key concept in REST is resources, these are hosted on a server and identified using a Uniform Resource Identifier (URI). A client can then interact with these using the different request methods defined in HTTP such as GET, PUT, POST and DELETE. An example use case might be that a client can get a list of all items on a menu by sending a GET request to the URI `http://examplerestaurant.com/api/menu`. The server would then send a response message containing the menu in some format, for example JavaScript Object Notation (JSON) or Extensible Markup Language (XML) [13, 14].

Problems with HTTP

When it comes to the IoT, HTTP is not an ideal application protocol. Firstly, it uses a relatively large amount of network bandwidth which will increasingly become a problem with the growth of the IoT. Secondly, the size of a software implementation of HTTP takes up a lot of code space, which makes it hard to use in embedded devices with limited memory. Both of these problems can partly be attributed to the fact that HTTP is designed for the web and has evolved with it. During the years, functionality has been added that is great for the web but unnecessary for the IoT [13].

CoAP design

CoAP was designed to provide the basic functionality of REST using very limited resources. It does this by utilizing a much simpler design than HTTP.

One of the things that contribute to the simplicity of CoAP is that it uses UDP instead of the Transfer Control Protocol (TCP). Since UDP does not provide the reliability of TCP, CoAP defines its own functionality for resending undelivered packages.

Another thing that contributes to the reduced complexity is the simple packet header design that can be seen in Figure 3.1. A CoAP packet is made up of a four byte binary header followed by an optional token and a sequence of options as can be seen in Figure 3.1. This compact design make it so that a typical CoAP header is between 10 to 20 bytes [13, 15].

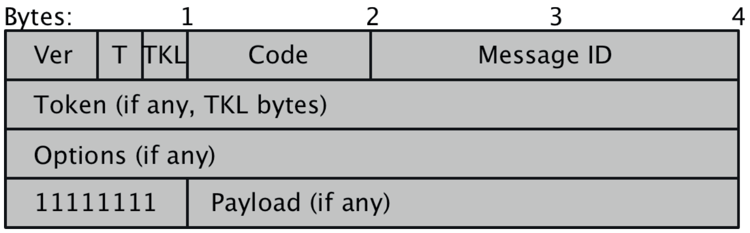


Figure 3.1: A CoAP packet.

CoAP HTTP compatibility

Even with this simple design, CoAP supports the four request methods of HTTP (GET, PUT, POST and DELETE). In addition to this, it also supports response codes that are very similar to those that can be found in HTTP. This enables CoAP to work together with HTTP by using intermediaries that translate between the two protocols.

In many cases there are equivalent methods, response codes and options available in both protocols. When this is the case, it is possible for the intermediary to perform a straightforward and stateless translation. In an IoT setting, this enables web clients to access resources on an embedded CoAP enabled device using a general purpose intermediary [13].

CoAP observe

If an HTTP client wants to receive continuous updates from a resource, it has to refer to polling the resource. This consumes an unnecessary amount of energy and network resources and is not suitable in the context of the IoT. CoAP solves this issue by making it possible to observe a resource. A client can observe a resource by sending a GET request with a special "Observe" option. If accepted by the server, the client will receive a notification message every time the resource is changed [13].

3.1.2 DTLS

Transport Layer Security (TLS) is the most widely used security protocol but it can not be used to secure datagram based protocols. There are two main reasons for this:

1. It is not possible to decrypt individual packets using TLS since they have to be decrypted in the right order.

2. The TLS handshake will fail if packets are delivered in the wrong order since it relies on the reliability of TCP.

The most desirable way to secure datagram traffic would be to use TLS but, as stated above, this is not possible. DTLS was created to solve this problem while staying as similar to TLS as possible. It minimizes differences compared to TLS and makes changes only where it is necessary. Most of these changes are done because of the two problems with TLS that are listed above [16].

3.1.3 CBOR

Concise Binary Object Representation (CBOR) is a binary data format based on an extended version of the JSON data model. It supports all JSON data types to ensure compatibility with all JSON documents. Some of the major design goals of Concise Binary Object Representation (CBOR) is to enable minimal implementation code size and schema-less decoding. Other design goals include reasonably compact code size, conservative CPU-usage and format extensibility.

An example of how a simple JSON document looks like encoded into CBOR can be seen in Listing 1 and 2 [17].

```
{
"string": "a string",
"number": 13,
"boolean": true
}
```

Listing 1: A JSON example.

```
a3                    # map(3)
   66                 # text(6)
      737472696e67    # "string"
   68                 # text(8)
      6120737472696e67 # "a string"
   66                 # text(6)
      6e756d626572    # "number"
   0d                 # unsigned(13)
   67                 # text(7)
      626f6f6c65616e  # "boolean"
   f5                 # primitive(21)
```

Listing 2: The CBOR representation of the JSON example above with describing comments.

3.2 OCF standardization

The vision of the OCF is a world where billions of connected devices communicate with each other. These devices come from a broad range of products such as appliances, phones, computers and industrial equipment. Furthermore, different devices will be able to communicate using different operating systems, chipsets and transports. Lastly, anyone from the global technology company to the lone garage maker will be able to create such connected devices [18].

In order to enable this vision, OCF works as a standardizing organisation by creating a specification that will enable interoperability between compatible devices. They also sponsor the IoTivity open source project, to enable anyone to make devices that are compatible with the specification [18].

There are two parts to the standardization work performed by the OCF. The first consists of creating data models using JSON and the RESTful API Modeling Language (RAML). The JSON model defines the devices and the RAML model describes the REST API that should be used when communicating with the device. These models are available at `http://oneiota.org/`, a site that also enables crowd-sourcing of data models for new types of devices [19]. The second is the creation of the specification that is described in the upcoming sections.

3.3 Architectural overview

The OCF specification defines devices, resources and operations. A device is defined as a logical entity that implements the OCF specification. There can be more than one device on a single hardware platform and each device can assume more than one OCF role (client and server). A resource is defined as a device component that can be discovered and controlled by another OCF enabled device. Operations are defined as actions that can be performed on a resource. These operations are defined as generic create, read, update, delete and notify (CRUDN) operations that uses the REST paradigm to model interactions on resources. Example: A light device consists of a single resource at `/a/light`. A device can send a CoAP GET request to this resource to get the current light state (if the light is on or off) [20, 21].

3.4 OCF resource model

The OCF resource model describes how resources are represented in the OCF framework. The specification lists several concepts and mechanisms used for this purpose. To be able to comprehend how OCF works it is necessary to get an understanding of how these mechanisms are built and why they are needed.

OCF Resource	In the OCF framework, anything that needs to be interacted with is represented as an OCF resource. For example, a connected ceiling fan with a light would provide separate resources for controlling the light and the fan. In most cases, a resource can be assigned any URI and does not enforce or provide any information about the characteristics of the resource. However, some core resources have a fixed URI whose characteristics are defined by the specification.
OCF Interface	An OCF interface is used to determine what kind of requests are permitted on a specific OCF resource. The same requests will produce different kinds of responses depending on the interface of the OCF resource. A simple example of this would be to make a POST request to a resource that only accepts GET requests which would result in an error message.
OCF Resource Property	The resource properties is what describes the features of an OCF resource. In the ceiling fan example above the power state and rotation speed, as well as the energy consumption of the fan could be the *properties* of the ceiling fan resource. Other meta data such as resource type or name is also defined as resource properties.

3.5 On-boarding and discovery

The process of joining new devices to a network as well as finding other devices on that network is fundamental for interoperability. Because this is such a central part of the IoT, the OCF specification defines some mechanics that *must* be implemented for a device to be considered an OCF device.

3.5.1 Detecting new devices

On-boarding is the process of exchanging information, such as Service Set Identifier (SSID) and log-in credentials for a Wi-Fi connection, to allow a new OCF device to join an existing OCF network. This can be done in a multitude of different ways, for example, a device with a simple user interface from which to take some kind of input may use a different method than a simple sensor device with no way for the user to interact with it. It will also be different depending on what physical transport the device is using for communication. For this reason it is not specified how exactly this should be done and as a result it is up to the developers to engineer their own solutions. One way to do on-boarding for Wi-Fi devices is that the device sets up a temporary Access Point (AP). The user then connects to that AP with another device, for example a mobile phone, and enters the information that is needed to establish a connection to the local Wi-Fi network.

3.5.2 Resource discovery

To be able to access and communicate with other devices an OCF client must find information about other OCF peers. The process of finding this information is referred to as resource discovery. The OCF specification defines two main discovery mechanisms.

Direct discovery To enable direct discovery a device must publish the information available for discovery, i.e. the resource properties and interfaces, along with the local resource. To discover this resource a client must issue a retrieve request directly to the resource either as a unicast or a multicast. The server with the discovered resource then responds with the resource information directly to the client issuing the request. All OCF devices must support direct discovery.

Indirect discovery In indirect discovery the resource and the information about the resource to be discovered is not hosted on the same device. To enable indirect discovery the device hosting the resource must publish the resource information on a separate device. The device holding the resource information then acts on behalf of the device holding the resource when it comes to discovery.

3.6 User experience

The interface the user will interact with when controlling an OCF device is largely up to the developers of the product, and as such will vary from brand to brand. However, there are some features defined in the specification to enable a developer to make life easier for the end users.

3.6.1 Remote Access

There are two models of accomplishing remote access to devices. In the first case the device itself possess the resources to be able to connect to the Internet, but this might not be possible on some constrained devices. In this case the external connection is handled by a proxy which relays the information bidirectionally to and from a remote host. The remote access feature is optional to implement according to the OCF standard, so it is up to the product developers whether it will be available on their product or not. Remote access is realized by letting devices connect to a Extensible Messaging and Presence Protocol (XMPP) server. All devices connected to the same XMPP account can then communicate with each other.

3.6.2 Scenes, Rules and Scripts

To enable users to automate some operations the specification defines *Scenes, Rules* and *Scripts*. A scene stores a set of resource values for a collection of resources which enables a client to recall a specific setup. An example of this would be if that the user could define a "goodnight" setting, which would turn off all the lights and turn down the thermostat with a single button press. The specification also defines *rule members*. Rule members are resources holding the values of a resource property that are set when an associated rule condition evaluates true.

A *rule* works like an if-then statement. It has a condition that when fulfilled, executes the *Rule Member* (script). A condition is evaluated every time one of the observed resources change value. An example of a rule could be that when the temperature in the room exceeds a certain point, the ceiling fan starts spinning.

Scripts in this context are the set of *Rule Members* that are executed when a *rule* condition holds true. Scripts can be used to incorporate conditionals, delays, loops or to read and set scenes.

Rules and scenes are represented as a resource on an OCF server. A client wanting to create a scene or a rule must first verify that the server supports the respective feature. This is done by sending a GET request to the server, requesting the rule list or scene list respectively. The client can then create/update/delete scenes and rules by communicating this to the server. Because scenes and rules are stored as resources on a server it enables the end user to access them from any OCF client after the initial setup.

3.7 Security

Any device connected to the Internet will always be exposed to different kinds of attacks, and the IoT brings new challenges when it comes to defending against these attacks. One such challenge is that IoT devices might have performance restrictions which makes heavy encryption or other security measures impossible to implement. Another factor that must be considered is the emergence of new types of attacks aimed specifically at constrained IoT devices.

The OCF security specification addresses security at two layers, transport and application. At the transport layer security is provided by the use of DTLS to enable packet encryption. At the application layer the security is maintained by an Access Control List (ACL) for each resource that controls which clients have access to it [22].

3.7.1 Access policies and ACLs

OCF does enforce access policies, the server holds and controls the resources and provides eligible clients with access. An example of a client-server interaction with end-to-end encryption: The client first establishes a network connection to the server and they set up a secure end-to-end channel. To be able to access any resources the client has to be authenticated to the server. The server consults the ACL pertaining that resource and looks for an entry matching the client requesting access to the resource. The server then applies the ACL permission to the resource and the server's Secure Resource Manager (SRM) enforces the decision to allow or deny access.

3.8 The IoTivity open source implementation

IoTivity is an open source implementation of the OCF specification and as of this writing the current version is 1.1.0. It supports several platforms and operating systems such as Linux, Android, Tizen and Arduino. IoTivity also has special support for the Yocto project, a customizable embedded Linux distribution [23].

3.8.1 Software Stack

The IoTivity framework provides two different software stacks presented in Figure 3.2. The Thin Block stack (TB) intended for devices with limited resources and the Unified Block stack (UB) for devices without this restriction [23].

The Thin Block stack uses CoAP to communicate over UDP and security is provided by the DTLS protocol. The Unified Block stack also supports CoAP and UDP but extends it with support for additional protocols. There are no alternative protocols available yet but more will come in the future, starting with HTTP over TCP/IP. Both of these stacks make use of CBOR for the serialization and deserialization of resources [23, 20].

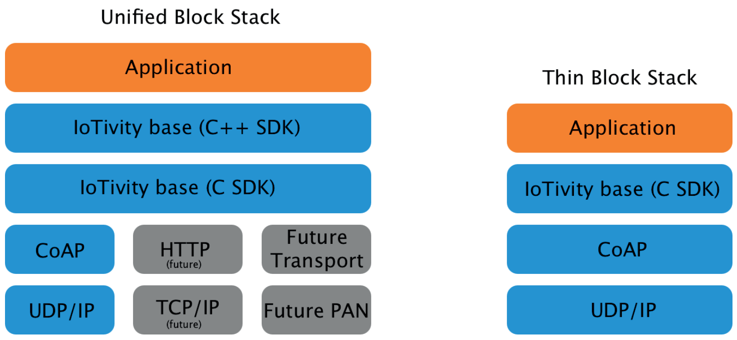


Figure 3.2: An overview of the two different stacks available for IoTivity.

3.8.2 APIs

The TB provides a core C API while the UB extends this with a high level C++ API. In addition to this, there are also API bindings based on the C++ API available for Java (Android) and JavaScript (node.js). The API functionality is divided into two different roles, server and client. It is, however, also possible to make a device that functions as both a server and a client. Some of the functionality provided for the server and client role respectively is described below [20].

Server role

Resource registration	By registering a resource with a device, it is made available to be controlled by other OCF compatible devices [24].
Defining behaviour	The API makes it possible to write custom code that defines the behaviour of the server when properties are modified. Example: The state of a light resource is set to off, custom code is defined that will actually turn off the light.
Resource tree	Enables resources to have resources of its own in a tree structure [20].

Client role

Resource discovery	The ability to discover resources on the local network. This is done by sending a multicast message to all devices [25].
Device discovery	The ability to discover devices, also done using multicast.
Querying Resource State	The ability to fetch the current property values from a resource. Example use: Check if the garage door is open [26].
Setting a resource state	The ability to modify the property values on a resource. This is what enables remote control of IoT devices. Example use case: A light can be turned on by changing the value of a resource property to on [27].

Observing resource state	Registers the client as an observer of a resource. The client will then receive notifications whenever any of the resource's properties changes. Example use case: A fan can observe a temperature sensor to automatically turn on when it is too hot [28].

Chapter 4
HomeKit

HomeKit was introduced at the 2014 edition of the Apple Worldwide Developers Conference (WWDC). Its main goal is to provide a consistent experience for users when using home automation products with Apple iOS devices. By doing this, Apple hopes to make home automation accessible for a wide audience of users [29].

4.1 Accessories

In the world of HomeKit, a smart device such as a HomeKit enabled light bulb or garage door is known as an "accessory". In other words, this is the name Apple use to refer to a "thing" in the Internet of Things [29].

4.2 Conceptual overview

HomeKit is made up of two parts, the first of these is a set of iOS APIs that is used to develop home automation apps. The second part is the Homekit Accessory Protocol (HAP) which is the application protocol that iOS devices use when communicating with HomeKit enabled accessories.

The HomeKit iOS APIs and HAP together, form a common interface between accessories and apps. Developers can create apps that can make use of any HomeKit enabled accessories that are connected to the device. Hardware manufacturers on the other hand, simply has to implement HAP on their accessory to make it talk with HomeKit. In other words, HomeKit acts as a man in the middle between the apps and the accessories [29].

4.2.1 Common database

In order to provide consistency between all these different apps and accessories, HomeKit uses a common database, which is stored on the iOS device. This database contains all the information about the user's home or homes. This includes all configured accessories as well as other data, such as in which room each accessory is located. This database is used by all HomeKit apps via the HomeKit iOS APIs to set up and control HomeKit enabled accessories [29].

4.3 User experience

The common database enables the consistent user experience that Apple wants to deliver. All HomeKit enabled apps, whether it is a manufacturer bundled app or a third-party app, has access to the same database via the HomeKit API. This means that whatever app the user uses to interact with their home, they see the same accessories. The database also enables controlling all configured accessories using iOS's smart assistant Siri.

HomeKit supports home sharing and remote access. Home sharing makes it possible for the owner of the home to give other users control over the accessories in the home. These other users can then control the accessories in the home but they cannot make any changes to the configuration of the home.

Remote access enables a user to control their home from anywhere. This is automatically enabled for all accessories in the home if there is an Apple TV 3rd generation in the home logged in on the same Apple ID as on the iOS device [30].

Scenes

A scene is a way for the user to control their home with a single command. To enable this, the user creates a new scene and gives it a name. He or she then adds a set of actions that are to be executed at the same time. This is done by configuring a set of target states for the relevant accessories, for example:

- Door $\rightarrow$ Locked
- Lights $\rightarrow$ Off

These scenes can then be executed using an app or by speaking their name to Siri [30].

Triggers

There are two kinds of triggers in HomeKit, timer triggers and event triggers. A timer trigger activates one or more scenes at a specific time of the day. An event trigger activates one or more scenes as a response to an event. There are currently two types of events.

- Accessory state: The event is generated when the state of an accessory changes to a specified state.
- Geofence: The event is generated when the user enters or leaves a geographical area.

Timer triggers always activate the configured scenes at the specified time but event triggers can be configured to only do so if additional conditions are fulfilled at the time when the event is generated. Currently, there are three types of conditions. These are: time, accessory state and significant events. There are currently two significant events: sunrise and sunset.

Example

Event triggers and conditions together with scenes enables the user to easily set up advanced home automation scenarios. As an example the user can set up a trigger on their smart door lock that is activated when the door is unlocked. The user can then configure this trigger with a condition that says to only activate the scene before midnight. Lastly the user configures the trigger to activate a scene that turns on all lights in the home. Using these simple steps, the user has set up the home to always turn on the lights when he or she arrives home but not in the middle of the night when people might be sleeping.

4.4 HomeKit Accessory Protocol

The HomeKit Accessory Protocol is the application protocol that accessories use to connect to HomeKit on an iOS device. It supports two transports, IP and BLE. When using IP, a connection can be made between the iOS device and the accessory when they are on the same subnetwork. This would typically mean that the iOS device and the accessory is connected to the same home network using Wi-Fi. In the case of BLE, a direct connection between the accessory and the iOS device is used [31].

4.4.1 The HomeKit protocol stack

HomeKit and HAP are implemented on top of a protocol stack that can be seen in Figure 4.1. There are currently two available transports for HAP; BLE and IP.

Starting from the top, General Attribute Profile (GATT) and JSON are used to serialize the services and characteristics that are defined by HAP. This serialized data is then packaged using Attribute Protocol (ATT) on the BLE side and HTTP on the IP side. The Logical Link Control and Adaptation Protocol (L2CAP) and TCP are used to transmit the packages [31].

On the IP side of the stack, there are some additional details that are of interest. Firstly, Apples Bonjour is used for accessory discovery. Bonjour is Apple's implementation of zero-configuration networking [32]. Secondly, only IP accessories can act as bridges. A bridge is a type accessory that can provide compatibility with home automation products that are implemented using some other application protocol than HAP. Lastly, HAP is implemented as a RESTful API on the IP side. This means that the Uniform Resource Locator (URL) can be used to specify what accessory, service and characteristic that a specific request refers to [31].

4.4.2 Common functionality definitions

On top of the protocol stack, HAP defines a common language that is to be used by all HomeKit compatible products. This language contains definitions of functionality that are commonly provided by home automation accessories. In order to model these definitions, a model using services and characteristics is used [31].

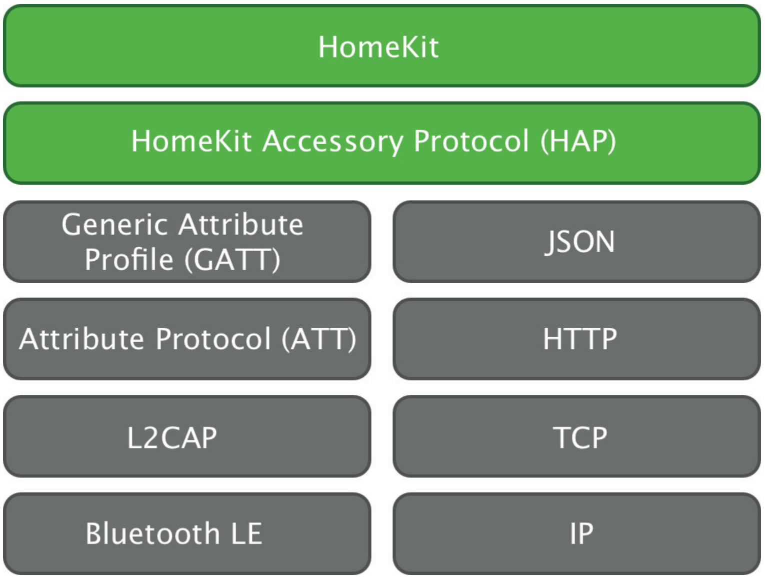

HomeKit
HomeKit Accessory Protocol (HAP)
Generic Attribute Profile (GATT)
JSON
Attribute Protocol (ATT)
HTTP
L2CAP
TCP
Bluetooth LE
IP

Figure 4.1: The protocol stacks that are used in HomeKit for IP and BLE transports respectively.

Services and Characteristics

A characteristic is the smallest piece of functionality that a user can interact with. These characteristics are then divided in groups of related functionality called services. An example of an how an IP based accessory can be configured using JSON is showed in Listing 3. In this example a light bulb is represented by "service2". It then presents two related pieces of functionality for this light bulb, two characteristics. These are the current state (on/off) and the brightness of the light bulb [31].

```
accessory : {
  service1 : "public.hap.accessory-information" {
    characteristic : "serial-number",
    characteristic : "identify"
  },
  service2 : "public.hap.lightbulb" {
    characteristic : "on",
    characteristic : "brightness"
  }
}
```

Listing 3: An example of an IP connected light bulb defined using HAT and serialized using JSON.

Protocol extensibility

Apple has predefined a lot of commonly used services and characteristics, like the ones in Listing 3. In addition to these, manufacturers can define their own services and characteristics for products that are completely new or not very common. It is also possible for manufacturers to use Apple's predefined ones together with their own in any way they see fit. An accessory can contain Apple defined services together with custom services. Custom services can contain Apple defined characteristics and custom characteristics. Apple defined services can also contain both custom- and Apple defined characteristics.

When using Apple defined services and characteristics to define functionality, that functionality will automatically be controllable using Siri on iOS devices [31].

4.4.3 Security

Apple has designed their own security solution for HAP by combining a number of existing technologies. The resulting security solution provides the following security features:

Bi-directional authentication	The iOS device and the accessory both authenticate each other to make sure that the other end is trusted.
Per-session encryption	A new key is used every time a connection is established between an iOS device and an accessory.
Perfect forward secrecy	The encryption key used in a particular session cannot be used to decrypt messages from any previous- or future session.
End-to-end encryption	Only the iOS device and the accessory have access to the encryption key so it is impossible for any third party to decrypt the messages that are being sent [31].

Setup code

A setup code is used when setting up an accessory for the first time. This code is provided by the accessory, either printed on the packaging or displayed on a screen. An iOS device is paired with the accessory using this code, which is then used as a basis for their future cryptographic relationship [31].

Cryptographic boundary

There is a a difference in where encryption takes place in the two different stacks. In the case of IP, encryption is applied on the transport layer, meaning that it is the TCP packets that are being encrypted. When it comes to BLE, Apple has made the choice not to use BLE pairing. Instead they apply encryption on the application layer. This means that it is the sent values that are encrypted before being serialized using GATT [31].

Cryptographic algorithms

The security features above are achieved using standard cryptographic algorithms. A description of each of them and an explanation of how they are being used in HAP can be seen below.

Secure Remote Password (SRP)	A secure password-based authentication and key-exchange protocol. It makes it possible for a client to authenticate with a server in the case that the client has access to a secret and the server has a verifier for each client. This verifier allows the server to authenticate the client but at the same time does not allow a potential attacker to impersonate the client in the case that the attacker is able to obtain the verifier. In HAP, SRP is used to encrypt and authenticate the initial pairing key exchange [31, 33].
Ed25519	A public-key signature system that is designed to be very fast while at the same time not compromise security. Ed25519 is based on elliptic curves, more specifically the twisted Edwards curve. In HAP, Ed25519 is used for the long-term pairing and authentication keys [31, 34].
Curve25519	An elliptic curve Diffie-Hellmann function that is optimised for speed while maintaining high-security. HAP uses Curve25519 to encrypt the initial authentication for each session [31, 35].
HKDF-SHA-512	SHA-512 is a Secure Hash Algorithm (SHA) adopted by the US government. HMAC-based Key Derivation Function (HKDF) is a Key Derivation Function (KDF) based on Hashed Message Authentication Code (HMAC). These are both used together in HAP to derive per-session ephemeral encryption keys [31, 36].
ChaCha20-Poly1305	ChaCha20 is a stream cipher that runs significantly faster than Advanced Encryption Standard (AES) in software implementations. Poly1305 is a high speed message authentication code. ChaCha20 and Poly1305 are combined using an Authenticated Encryption with Associated Data (AEAD) construction. HAP uses this combination to encrypt and authenticate data [31, 37].

4.5 MFi program

Apple requires all manufacturers of HomeKit products to be certified through the Made for iPhone/iPod/iPad (MFi) program. To become certified, manufacturers must install an authentication chip in the accessory and pass extensive tests conducted by Apple. To pass these tests, manufacturers must send prototypes to Apple's headquarters where the products can be tested by Apple personnel [38].

Chapter 5

Prototype

5.1 Environment

The implementation of prototypes for IoTivity and HomeKit were performed on different hardware platforms since there was no single platform that supported both frameworks. The programming languages used to build the prototypes were C, C++ and JavaScript. Git was used to provide version control of source code.

5.1.1 HomeKit

Several prototypes implementing HomeKit were developed for development kits from both Broadcom and Nordic Semiconductor. The details of the hardware platforms as well as the software used to develop and test the prototypes are presented in this section.

Hardware

The first hardware platform is the `BCM943341WCD1_EVB` which is a development kit from Broadcom. The development kit consists of a `BCM943341WCD1` System in package (SiP) module, which is mounted on a development board that provides USB connectivity. This USB connection is used to provide a serial connection and for flashing device firmware during development. The `BCM943341WCD1` is powered by the STM32F407 32-bit ARM microcontroller and provides connectivity in the form of WLAN, Bluetooth, BLE, and Near Field Communications (NFC) [39].

The second platform used was the nRF51 DK, which is a development kit from Nordic Semiconductor based on the nRF51 series System-on-a-Chip (SoC). The nRF51 DK provides connectivity for BLE, ANT+ and 2.4GHz proprietary applications. The nRF51 DK is based on an ARM Cortex M0 microprocessor and features a General-purpose input/output (GPIO) pin layout that is compatible with the Arduino Uno Revision 3 standard [40, 41].

The `BCM943341WCD1_EVB` was used for developing a HomeKit prototype utilising WLAN connectivity. In the case of BLE, the Nordic nRF51 DK was used instead.

Software

All development for the `BCM943341WCD1_EVB` was performed using the Broadcom WICED MFI/HomeKit Software Development Kit (SDK). The nRF5 SDK for HomeKit was used instead in the case of the Nordic nRF51 DK. All code for both these platforms was written in the C programming language.

In addition to these SDKs, other software was used in the form of an iPhone app called "MyHome" and HomeKit Accessory Tester (HAT). "MyHome" is a free app that can be used to control HomeKit accessories. The app was downloaded from the App Store. HAT is Apple's testing tool that allows hardware developers to test their accessories using a Mac.

5.1.2 IoTivity

All of the IoTivity prototypes were run on Linux platforms which made the development process easier since it was not necessary to flash the hardware between every test run.

Hardware

The hardware platforms used for the IoTivity prototype were Intel Edison kits with support for Arduino shields. Intel Edison is a very powerful platform with a dual-core Intel Atom CPU, 1 GB of RAM and 4 GB flash storage. The development kit that was used provides flashing via USB, several GPIOs, analog I/Os, as well as connectivity via WLAN, Bluetooth, and BLE.

Software

The devices used a custom built Yocto operating system with support for the IoTivity framework as well as the GPIOs of the Intel Edison board. The OCF devices were implemented in C and C++ using the respective IoTivity APIs available in IoTivity version 1.0.1.

To be able to control the devices independently of each other a web interface was implemented in JavaScript that could observe and change the current state of the resources.

The Yocto project

The Yocto project is an open source collaboration to provide a custom built Linux environment for embedded devices regardless of the hardware architecture [42]. A Yocto image is built from a recipe where it is specified what modules and frameworks should be built into the image. In this project, a Yocto recipe for Intel Edison was modified to build an image with support for the IoTivity C and C++ API.

5.2 Scope

Anima, the company at which this thesis was conducted, has an interest in controlling IoT devices with their smartwatch product. This product is communicating using BLE, which means that they also have an interest in communicating with IoT devices using this technology. Additionally, they also have an interest in the features and limitations of different IoT frameworks.

The scope of the prototypes are based on these interests. It is feasible to evaluate the limitations and features of a framework with almost any prototype to varying degrees. However, facilitating the interest in controlling IoT devices requires two prototypes that communicate with each other in some way. Therefore, a use-case where two prototype devices have to communicate with each other was designed. The use of a Graphical user interface (GUI) to control the IoT devices were also added to the use-case since it is the most common way of controlling IoT devices in the smart home. Lastly, it would be desirable to investigate BLE in an IoT setting since Anima uses BLE in their product. A set of prototypes was developed for each IoT framework that facilitates the requirements of this use-case.

5.3 Prototype design

The use-case was developed to include three main components, two IoT devices and a GUI. Additionally, a minimal BLE device prototype was developed in the case of HomeKit. These components are described below:

Power strip device	An IoT device that has the ability to toggle a power strip on and off. The power strip is exposed as an abstract network resource and can be controlled by other IoT devices.
Sensor device	An IoT device with an attached piezo element. The piezo element is used as a vibration sensor. The vibration sensor is exposed as an abstract network resource. This enables other IoT devices to read the current sensor data.
GUI	A GUI that has the ability to control the power strip device and monitor the state of the sensor device.
Minimal BLE device	A minimal implementation of a BLE device that was implemented when the IoT framework provided support for BLE. It was only implemented for HomeKit since IoTivity does not support BLE yet.

5.3.1 HomeKit

The HomeKit use-case implementation consists of two accessories that are connected to an iPhone. Both of these accessories are implemented on `BCM943341WCD1_EVB` boards and communicate using HAP over WLAN. The first accessory is the *power* *strip device* and exposes a service that enables control of a power strip. It is realized by connecting the power strip to the board using an electrical circuit. The circuit enables the power strip to be toggled on and off using a GPIO on the board. The other accessory is the *sensor device*. It exposes a vibration sensor service that consists of a piezo element connected to an Analog-to-digital converters (ADC) on the board. For the GUI component, an iPhone app called "MyHome" was used.

An iPhone with a HomeKit enabled app such as "MyHome" can be used to control and monitor the states of these accessories. However, due to restrictions in the HomeKit framework, the accessories cannot directly communicate with each other. In order to control the state of the first accessory with sensor readings of the second, the iPhone has to be configured to do so. This configuration consists of a trigger that fires when the iPhone receives an event from the sensor device. When the trigger fires, a request is sent to change the state of the controllable device.

5.3.2 IoTivity

The IoTivity implementation of the use-case consists of two devices and a custom made GUI in the form of a Single-page application (SPA). The two IoTivity devices are hosted on a Intel Edison hardware platform and are communicating using CoAP over WLAN.

The first IoTivity prototype is the *power strip device*. It is implemented as a client-server, meaning it can act as both a client and a server. The prototype hosts a resource that enables control of a power strip. It is realized by connecting the board to the same circuit that is used for the HomeKit prototype. The second device is the *sensor device*. It is implemented as a server and hosts a vibration sensor resource. The vibration sensor consists of the same piezo element and electrical circuit that was used for the HomeKit prototype, connected to an ADC on the Edison board.

The device-to-device use case was implemented by letting the *power strip device* observe the resource state of the sensor resource on the *sensor device*. Whenever the sensor changes state, the *sensor device* sends a response to the *power strip device*. The *power strip device* will then turn on or off appropriately.

Since there are no standard user interfaces available for IoTivity, a web interface was implemented. The web interface enables a user to control and observe IoTivity devices using a GUI.

5.4 Prototype implementation

This section describes how the different prototypes are implemented and highlights some of the differences between the use-case implementations.

5.4.1 HomeKit

Three HomeKit prototypes were implemented, in addition to the *power strip device* and the *sensor device* a *minimal BLE device* was developed to investigate BLE connectivity with HomeKit. The main use-case with the *power strip device* and *sensor device* is depicted in Figure 5.1.

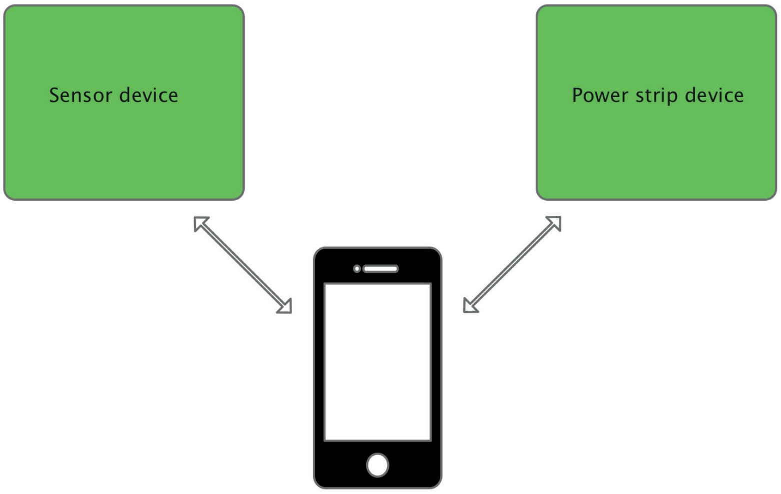


Figure 5.1: The setup for the HomeKit use-case. Both devices communicate only with the iPhone.

Power strip device

The HomeKit *power strip device* software is implemented in C and after initializing the platform the program connects to the Wi-Fi network and waits for an iPhone to request a pairing. In this implementation, the Wi-Fi SSID and credentials are hard-coded. Apple supplies an on-boarding process to connect new devices to the network, but that process requires specialized Apple hardware that was not available for this project. In the pairing process the iPhone user is asked to supply a password generated by the accessory. The password can be presented in different ways and in this implementation it is printed in a terminal via a serial connection. After the pairing is complete the accessory listens for updates to its characteristics that can be sent from the paired iPhone.

Sensor device

The HomeKit *sensor device* software implementation is very similar to that of the *power strip device*. This prototype is also implemented in C and the connection and pairing process is the same. Instead of waiting for updates from the iPhone however, the prototype reads from the ADC connected to the piezo element. If the HomeKit *sensor device* detects a value above the pre-set threshold it sends an update to the iPhone notifying it about the event.

Minimal BLE device

A *minimal BLE device* prototype was also implemented to investigate BLE connectivity with HomeKit. The *minimal BLE device* prototype consists of a single accessory that is paired and controlled with an iPhone over BLE. The hardware platform used for this prototype was the nRF51 DK. The nRF51 DK is a platform that Anima holds an interest in, which made it a natural choice for testing out the BLE connectivity. This implementation is also done in C and the functionality is similar to the lightbulb service implementation.

5.4.2 IoTivity

The setup of the IoTivity use-case is shown in Figure 5.2. The IoTivity prototype implementations of the *power strip-* and *sensor device* are similar to their HomeKit counterparts. The key differences between the HomeKit and IoTivity implementations are described in to following sections.

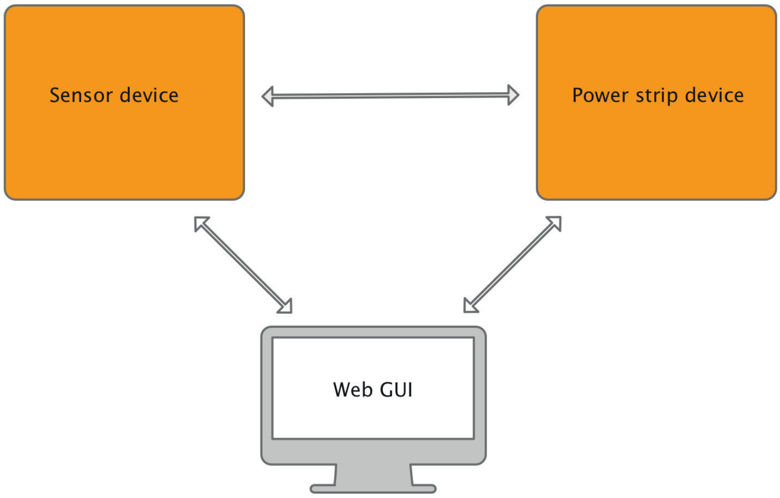


Figure 5.2: The setup for the IoTivity use-case. The devices can communicate directly with each other. The web GUI can be used to check and change the states of the devices.

Power strip device

The IoTivity *power strip device* prototype differs from the HomeKit counterpart in that it has two operational modes, manual and auto. In manual mode, the prototype acts as a server and waits for requests from a client. In auto mode the IoTivity *power strip device* also acts as a client in that the prototype observes the *sensor device* and updates its state based on the notifications it gets. Since the IoTivity use case prototypes run on a Yocto OS, on-boarding and network configuration do not need to be handled in the prototype software.

Sensor device

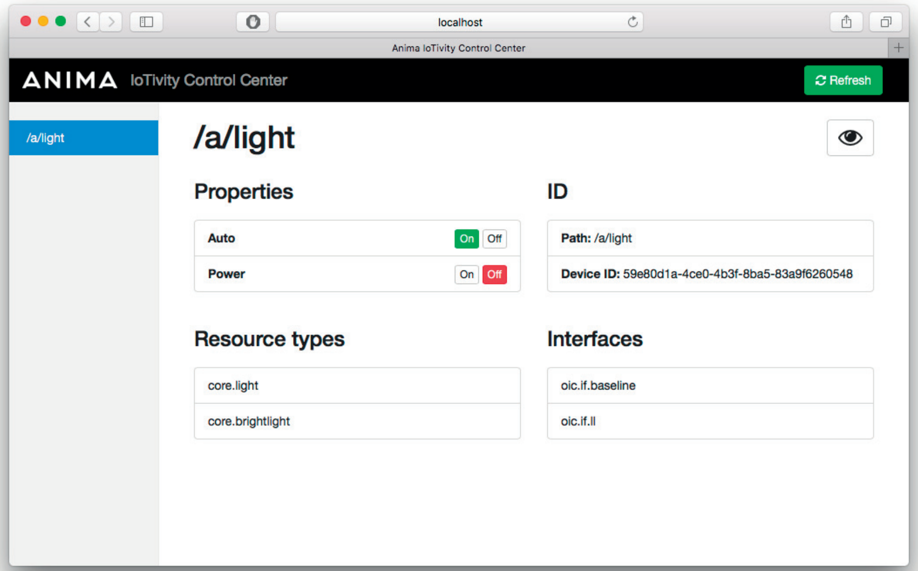
The IoTivity *sensor device* prototype is an OCF server implementation. The prototype holds a vibration sensor resource that is marked as observable. This reveals to all clients discovering the *sensor device* that it supports and handles observe requests.

The IoTivity *sensor device* is implemented so that the device sleeps until the device receives an observe request. After receiving an observe request the device starts reading the sensor values. If the device detects a reading above the threshold, the device sends a notification to all observing clients that it detected a vibration. An observer can at any time send an unregister request, in which case the *sensor device* removes the client from the device's list of observer. If the list of observers is empty at any point, the *sensor device* goes back into sleep mode.

GUI

The web GUI was implemented in JavaScript, CSS and HTML and can be seen in Figure 5.3. The GUI prototype uses the JavaScript bindings for IoTivity that is provided by the `iotivity-node` node.js module to communicate with IoTivity devices. Since these bindings only work for the server side, communication between a server and client was needed.

The client is a SPA implemented using Angular.js and Bootstrap. All communication between the server and client is performed using WebSockets. The server acts as an intermediary. It relays messages received from IoTivity devices as CoAP packets to the SPA using the WebSocket connection.



localhost

Anima IoTivity Control Center

ANIMA IoTivity Control Center

Refresh

/a/light

/a/light

Properties

Auto	On Off
Power	On Off

ID

Path: /a/light

Device ID: 59e80d1a-4ce0-4b3f-8ba5-83a9f6260548

Resource types

core.light

core.brightlight

Interfaces

oic.if.baseline

oic.if.ll

Figure 5.3: A screenshot of the web interface that was developed to serve as a GUI for IoTivity devices.

Chapter 6

Experiments and Evaluation

To be able to compare the frameworks an evaluation model was designed where important characteristics were identified. The prototypes were used to provide measurable data in the cases where data generation were necessary. All of the results are summarized in Table 6.3 on page 52 for easy reference.

6.1 Methodology

The evaluation of the IoT frameworks was conducted using a method based on the criteria based evaluation model. These kind of models are described in a paper by Gediga et al. [43]. This paper describes the general steps involved in criteria based evaluation. Our evaluation model is loosely based on these general steps but has been adapted to fit our case of evaluating IoT frameworks for the smart home.

The general steps used in our evaluation are as follows:

1. Determine what characteristics should be evaluated.
2. Define observable measurables based on the characteristics.
3. Perform measurements on the measurables.
4. Perform the evaluation and comparison based on the measurements.

The biggest adaption made is that we do not derive requirements on the characteristics. The method described in the paper by Gediga et al. is designed to evaluate a single software product. Since we are comparing different IoT frameworks, we instead chose to compare the measurables of the different IoT frameworks.

6.1.1 Measurement types

Many of the characteristics have binary measurables not suited for experimental measuring, such as the presence of a specific feature. Because of this, the evaluation model uses both experimental and non-experimental ways of evaluating the frameworks. The different methods are defined as follows:

Experimental	The results have been produced by making measurements on the prototypes.
Specification	The results are derived from the framework specification and/or APIs.
Prototype	The results are supported by a prototype. (No results are based solely on prototypes, but they are used as consolidation of a result.)
Manufacturer Data	The results have been obtained from the relevant hardware manufacturers.

6.2 Characteristics

The process of identifying the important characteristics of a smart home product were performed as a combination of three different factors. A study of relevant literature, the experiences gained by implementing the use-case as well as discussions with Aleksandar Rodzevski, a senior software architect at Anima. The literature studied was mainly academic papers but also various online resources with articles relating to the IoT and smart homes. Based on these experiences, a list of five characteristics that were judged to be of core importance to smart home products were produced.

6.2.1 Interoperability

Interoperability in this context is defined as the ability for smart home devices to communicate and work together with disparate devices and interfaces. This is important because a product that is interoperable will allow users who already owns products from a different vendor to incorporate the product into their smart home. Other things to consider that also fall under interoperability are the feasibility of bridging other frameworks and what platforms are supported for building a user interface. Is it feasible to control the devices using an Android smart phone for example, or does the IoT framework only support iPhone or vice versa?

6.2.2 Connectivity

Connectivity in the scope of this report is concerning the connection capabilities of the evaluated frameworks. Different kinds of smart home devices have different requirements when it comes to connectivity. Connectivity is considered a relevant characteristic because it contributes to the versatility of the framework. It is also closely related to network scalability and performance.

6.2.3 Scalability

The scalability of the framework refers to the memory and computation speed requirements of running the framework. Low system requirements enable the use of more constrained hardware platforms. Having a smaller platform is advantageous for three reasons, lower hardware cost and power consumption as well as a smaller physical size. Having a lower production cost is obviously desirable and lower power consumption opens up the option of having a battery powered device. This in combination with a smaller physical size opens up more product possibilities.

6.2.4 Security

Since many smart home devices will have a direct or indirect connection to the Internet, it should be self-explanatory why security for smart home devices is required. In addition to already known threats, new types of attacks might emerge that are aimed specifically at smart home devices. In order to combat these new threats and fix any overlooked security flaws, Over-the-air programming (OTA) updates are required. Another important security factor is device authentication to make sure that no unauthorized actors can gain access. Since software security is such an extensive subject this report will not investigate *how* secure the frameworks are, but rather if they retain certain security features.

6.2.5 Network related properties

There are a number of network related properties that are of interest when evaluating and comparing IoT frameworks.

Response time

The response time of an IoT product is connected to the usability of the product. As such, it is of importance when it is being controlled by a person. The general advice regarding response times for usability has been the same since 1968 [44].

0.1 second	When a user performs an action, it feels like it happens instantaneously. This gives the impression of directly manipulating the system being controlled.
1 second	Gives the impression of a delay, but the users train of thought is uninterrupted.
10 seconds	The users might get distracted by other things. Some kind of visual feedback is needed to reassure the user that the computer is working.

Judging from these guidelines, it is important to keep the response time under one second. It would be suboptimal if the user lost their train of though waiting for an action to complete. A response time under 0.1 second is desired since it gives a feeling of directly controlling the smart home devices.

Network traffic used

The number of IoT devices in a smart home should be able to scale to billions in the world and hundreds in a home. Therefore, it is desirable to keep the amount of data needed to perform an action (e.g. turn on a light) to a minimum. This also gives the extra benefit of reduced power usage, which is important in battery powered constrained devices.

6.3 Interoperabilty

The following questions were formulated to evaluate the interoperability:

Vendor interoperability	How does the IoT framework ensure interoperability between products from different vendors?
Bridging	Does the IoT framework support bridging devices implementing other frameworks?
User interfaces	Is there support for Android/iOS/web as a user interface?

6.3.1 HomeKit

To ensure interoperability, the MFi program requires all products to go through a certification process that makes certain that the product is fully compatible with HomeKit [45].

Bridging of HomeKit accessories is supported with some restrictions. The restrictions include accessories supporting Wi-Fi as well as accessories enabling access to the home. A connected door lock, for example, may not be bridged [46]. HomeKit is also restrictive when it comes to building interfaces, as iOS is the only OS available for controlling HomeKit accessories [45].

6.3.2 IoTivity

The OCF certification program is still under development. Once it is finished, the applicant vendor can send their product to a test laboratory where they will test the device. If the device passes the device is OCF certified and guaranteed to be fully compatible [47].

The OCF specification does not impose any restrictions for bridging devices to other frameworks [21]. It is also possible to build user interfaces for any platform though APIs are only available for Android and web interfaces at this time [48].

6.4 Connectivity

The following points concerning connectivity are addressed:

Physical transports	What are the supported network stacks (Wi-Fi, BLE, 802.15.4)?
Device-to-device	Does the IoT framework support device-to-device communication?
Remote	Is there support for remote accessibility?

6.4.1 HomeKit

HomeKit supports both Wi-Fi and BLE physical transports but all traffic over these transports has to be sent between an iOS device and an accessory. This means that direct communication between accessories is not supported by HomeKit. When it comes to remote access, it is possible through Apple's cloud service but requires an Apple TV (3rd generation or later) to function. The Apple TV is used to relay packages between the iOS device and the accessory [49].

6.4.2 IoTivity

IoTivity is built so that it can run on top of any wireless transport, i.e. Bluetooth, BLE, Wi-Fi and 802.15.4. In contrast to HomeKit, there are no restrictions concerning device-to-device communication [21]. Furthermore, remote access is supported through the use of XMPP. Remote access requires that the devices connect to an OCF server that supports XMPP and is accessible via the Internet [50].

6.5 Scalability

The scalability of the frameworks is evaluated based on the most constrained platforms available for each IoT framework. This evaluation was performed this way because of two main reasons: Firstly, the hardware requirements are not specified within the respective framework specifications. Secondly, because Apple does not supply a reference implementation of the HomeKit specification.

6.5.1 HomeKit

The hardware platforms supported by the HomeKit SDKs of Broadcom, Marvell Technology Group and Nordic Semiconductor were studied. The platforms are all similar in performance, with the nRF52832 somewhat more constrained than the others, as can be seen in Table 6.1.

Table 6.1: HomeKit - hardware platforms

Hardware platform	CPU	RAM	Flash memory
nRF52832	Cortex-M4F 64 MHz	64 kB	512 kB
BCM943341WCD1	Cortex-M4 168 MHz	192 kB	512 kB
88MC200	Cortex-M3 200 MHz	512 kB	1 MB

6.5.2 IoTivity

The IoTivity website lists several platforms that are confirmed to run the IoTivity framework. In addition to the platforms listed in Table 6.2, IoTivity can run on any SoC that supports Linux, Android or Tizen. However, all of these operating systems have higher system requirements than the ones listed in Table 6.2 [51, 52, 53]. The table presents Arduino Mega2560 as the most constrained of the IoTivity devices [48].

Table 6.2: IoTivity - hardware platforms

Hardware platform	**CPU**	**RAM**	**Flash memory**
Arduino Mega2560	ATMega2560 16 MHz	8 kB	256 kB
Arduino Due	Cortex-M3 84 MHz	96 kB	512 kB

6.6 Security

The availability of the following security features is evaluated:

Authentication	Does the specification define bi-directional authentication?
Encryption	Is there session encryption? If so, what protocols are used?
OTA	Is it possible to do OTA software updates?

6.6.1 HomeKit

HomeKit enforces Bi-directional authentication and provides encryption using Apple's proprietary encryption suite. Additionally, Over-the-air firmware updates are supported by HomeKit accessories [31].

6.6.2 IoTivity

The OCF specification defines bi-directional authentication and it is required for restricted resources. The specification also supports session encryption and encourages the use of DTLS [54]. OTA updates are not defined in the specification but it is feasible for device developers to implement it themselves.

6.7 Network related metrics

The network measurements that were performed covers response time and network traffic usage. This section describes how the measurements were conducted and presents the possible error sources of the measured data.

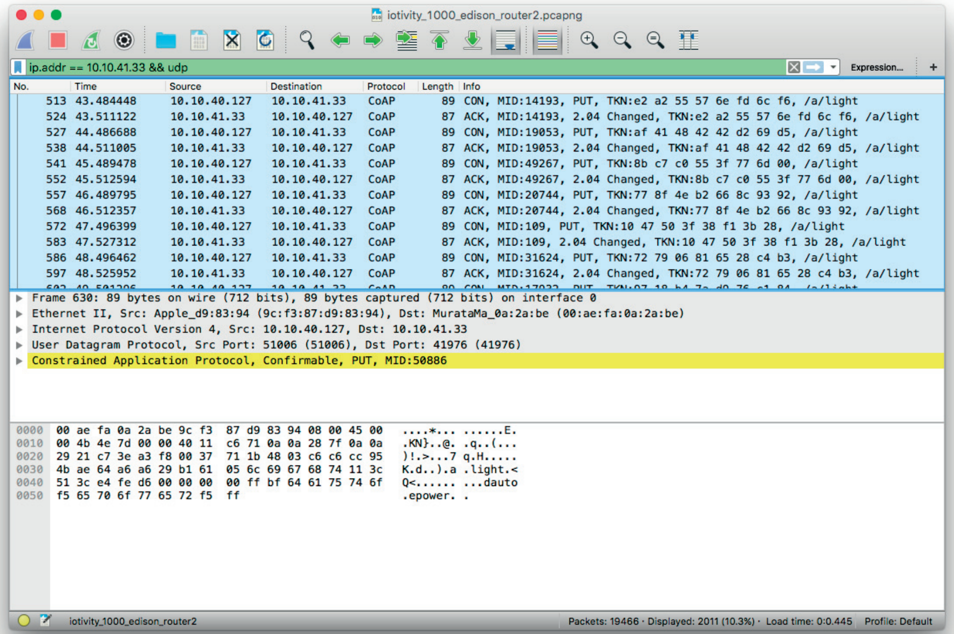
6.7.1 Response time

To not create additional error sources, the response time data was generated and collected with the help of scripts and tools.

Methodology and Tools

The response times were measured using Wireshark, as shown in Figure 6.1, in the case of both IoTivity and HomeKit. For HomeKit, the response time is defined as the "RTT to ACK" as measured by Wireshark. In the case of IoTivity, Wireshark does not provide response time measurements for CoAP so the response time was calculated as the difference between the timestamps of the PUT request and the response.

The measured HomeKit data was generated using HAT since Wireshark can only measure on data that is sent or received by the computer it is running on. The accessory used for these measurements was the *HomeKit power strip device*. A command to toggle the power strip on or off was sent with one second intervals.


iotivity_1000_edison_router2.pcapng
Expression...
iotivity_1000_edison_router2
Packets: 19466 · Displayed: 2011 (10.3%) · Load time: 0:0.445
Profile: Default

```
ip.addr == 10.10.41.33 && udp
```

```
No.  Time       Source        Destination   Protocol Length Info
 513 43.484448  10.10.40.127  10.10.41.33   CoAP     89 CON, MID:14193, PUT, TKN:e2 a2 55 57 6e fd 6c f6, /a/light
 524 43.511122  10.10.41.33   10.10.40.127  CoAP     87 ACK, MID:14193, 2.04 Changed, TKN:e2 a2 55 57 6e fd 6c f6, /a/light
 527 44.486688  10.10.40.127  10.10.41.33   CoAP     89 CON, MID:19053, PUT, TKN:af 41 48 42 42 d2 69 d5, /a/light
 538 44.511005  10.10.41.33   10.10.40.127  CoAP     87 ACK, MID:19053, 2.04 Changed, TKN:af 41 48 42 42 d2 69 d5, /a/light
 541 45.489478  10.10.40.127  10.10.41.33   CoAP     89 CON, MID:49267, PUT, TKN:8b c7 c0 55 3f 77 6d 00, /a/light
 552 45.512594  10.10.41.33   10.10.40.127  CoAP     87 ACK, MID:49267, 2.04 Changed, TKN:8b c7 c0 55 3f 77 6d 00, /a/light
 557 46.489795  10.10.40.127  10.10.41.33   CoAP     89 CON, MID:20744, PUT, TKN:77 8f 4e b2 66 8c 93 92, /a/light
 568 46.512357  10.10.41.33   10.10.40.127  CoAP     87 ACK, MID:20744, 2.04 Changed, TKN:77 8f 4e b2 66 8c 93 92, /a/light
 572 47.496399  10.10.40.127  10.10.41.33   CoAP     89 CON, MID:109, PUT, TKN:10 47 50 3f 38 f1 3b 28, /a/light
 583 47.527312  10.10.41.33   10.10.40.127  CoAP     87 ACK, MID:109, 2.04 Changed, TKN:10 47 50 3f 38 f1 3b 28, /a/light
 586 48.496462  10.10.40.127  10.10.41.33   CoAP     89 CON, MID:31624, PUT, TKN:72 79 06 81 65 28 c4 b3, /a/light
 597 48.525952  10.10.41.33   10.10.40.127  CoAP     87 ACK, MID:31624, 2.04 Changed, TKN:72 79 06 81 65 28 c4 b3, /a/light
```

```
▶ Frame 630: 89 bytes on wire (712 bits), 89 bytes captured (712 bits) on interface 0
▶ Ethernet II, Src: Apple_d9:83:94 (9c:f3:87:d9:83:94), Dst: MurataMa_0a:2a:be (00:ae:fa:0a:2a:be)
▶ Internet Protocol Version 4, Src: 10.10.40.127, Dst: 10.10.41.33
▶ User Datagram Protocol, Src Port: 51006 (51006), Dst Port: 41976 (41976)
▶ Constrained Application Protocol, Confirmable, PUT, MID:50886
```

```
0000  00 ae fa 0a 2a be 9c f3  87 d9 83 94 08 00 45 00   ....*... ......E.
0010  00 4b 4e 7d 00 00 40 11  c6 71 0a 0a 28 7f 0a 0a   .KN}..@. .q..(...
0020  29 21 c7 3e a3 f8 00 37  71 1b 48 03 c6 c6 cc 95   )!.>...7 q.H.....
0030  4b ae 64 a6 a6 29 b1 61  05 6c 69 67 68 74 11 3c   K.d..).a .light.<
0040  51 3c e4 fe d6 00 00 00  00 ff bf 64 61 75 74 6f   Q<...... ...dauto
0050  f5 65 70 6f 77 65 72 f5  ff                        .epower. .
```

Figure 6.1: Thek network measurements was performed using Wireshark. This figure shows an example of captured IoTivity traffic.

In the case of IoTivity, a script was written in JavaScript that generates traffic using the API-bindings provided by the `iotivity_node` package. This script discovers the OCF device on the network and proceeds to toggle the power strip on and off with one second intervals. The prototype used for these measurements was the *IoTivity power strip device*

The raw data provided by Wireshark was exported as a .csv file. This data was then parsed and analysed using Python in conjunction with scientific libraries such as NumPy, SciPy and matplotlib.

Error sources

There are some sources of error relevant to the measured data. Firstly, the hardware platforms have different performance characteristics. The Intel Edison has a dual-core 500 MHz Intel Atom processor and 1 GB of RAM. The Broadcom `BCM943341WCD1` board has a 168 MHz ARM M4 processor with 192 kB of RAM. Secondly, there is software running the devices that can have an impact on the results, such as the operating system. Another aspect of software differences is the implementation of the protocols. IoTivity is the only implementation of the OCF protocol but HomeKit has several implementations from different manufacturers. It is possible that some of these implementations perform differently than the Broadcom implementation that we have used for our HomeKit prototypes. Lastly, the router load at the time of testing could have had an impact on the measured values.

IoTivity

A histogram showing all measured data can be seen in Figure 6.2 and the range where most data is located can be seen in Figure 6.3. In these figures, each sample represents the response time of a command.

As can be seen in these figures, most of the samples are located in the range 7 to 50 ms with a peak between 20 and 30 ms. However, there are also quite a few samples with relatively high response time of more than 150 ms.

The mean response time was calculated to 29.6 ms with a standard derivation of 13.8. The probability distribution of response times for IoTivity can be seen in Figure 6.4. It can be seen in this figure that the probability of the response time exceeding 75 ms is very low.

HomeKit

All measured HomeKit data can be seen in the histograms presented in Figure 6.5 and 6.6. Compared to IoTivity, the samples are much more compressed to a small area around 20 ms but at the same time there are also outlier samples all the way up to 200 ms.

The mean value was calculated to 22.2 ms with a standard derivation of 22.3. The probability distribution of response times for HomeKit can be seen in Figure 6.7. Compared to IoTivity, the distribution is quite a bit wider but it is still very unlikely that the response time exceeds 100 ms.

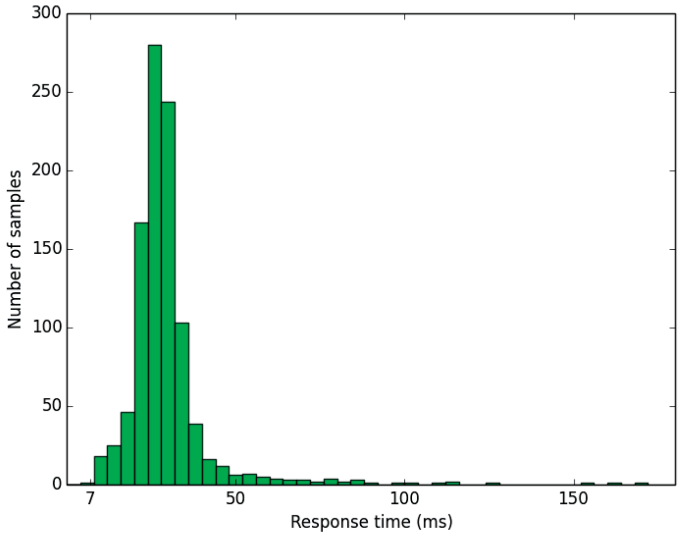


Figure 6.2: A histogram showing all measured response times for IoTivity.

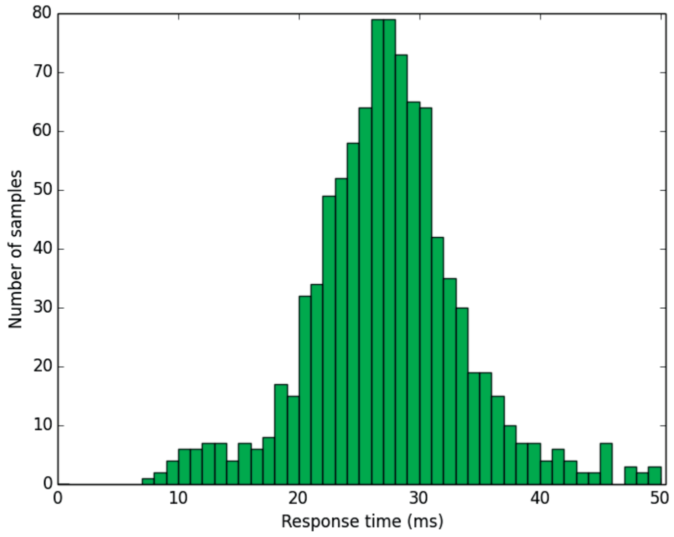


Figure 6.3: A histogram showing the measured response times for IoTivity in the range 0-50 ms.

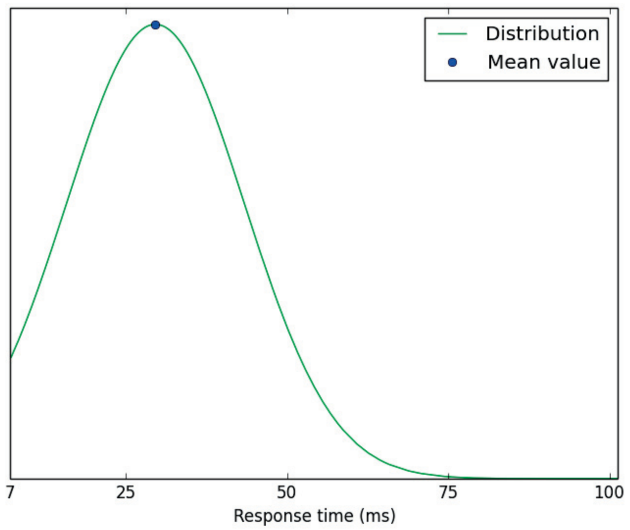


Figure 6.4: The distribution of response times measured on IoTivity.

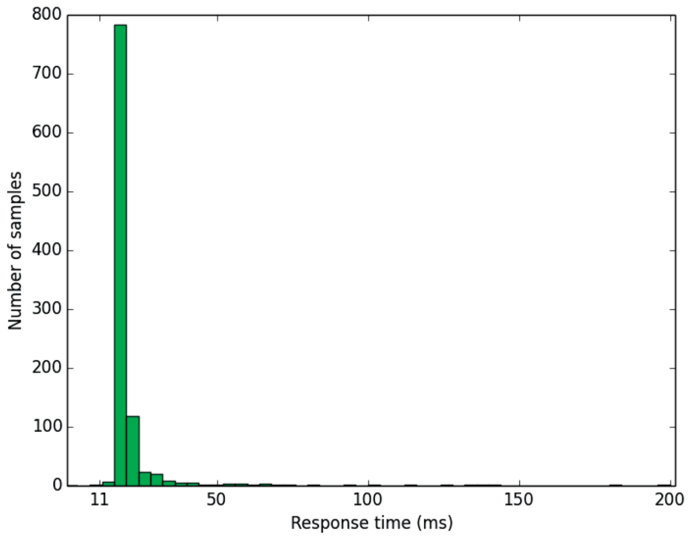


Figure 6.5: A histogram showing all measured response times for HomeKit.

6.7.2 Network traffic used

To evaluate the network traffic usage, the same data was used as in the response time evaluation.

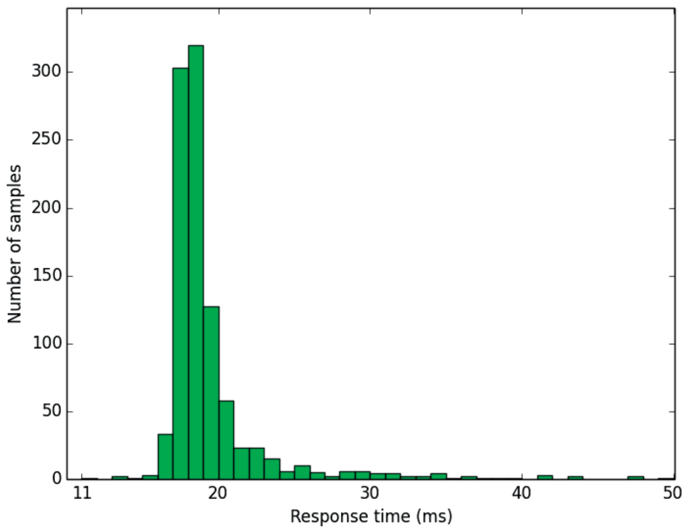


Figure 6.6: A histogram showing the measured response times for HomeKit in the range 0-50 ms.

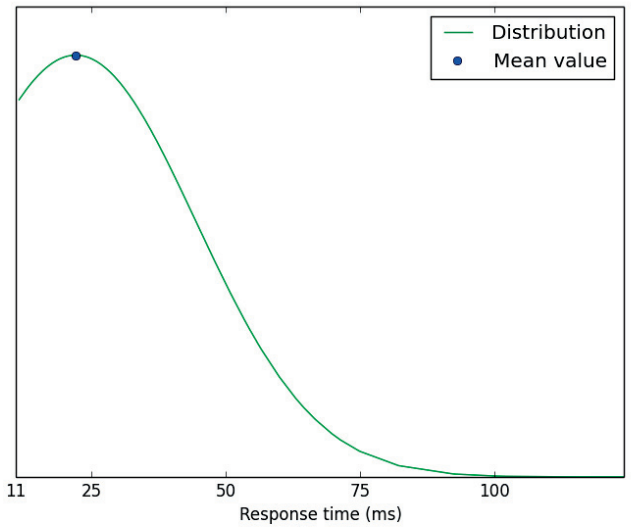


Figure 6.7: The distribution of response times measured on HomeKit.

Methodology and Tools

The network traffic used is defined as the total amount of data that is needed to fulfill a command such as turning on or off a light. To make this comparable the same type of command was used as an example, namely toggling a *Power strip device* on or off. The same data was used as in the response time evaluation. As such, there were 1000 samples for each framework collected using Wireshark.

In this data, each identical command uses the exact same amount of network resources. An identical command results in exactly the same number of requests and each request has the same size. This is true for both HomeKit and IoTivity, with some exceptions. In the case of HomeKit, there might be some TCP handshake data that has to be sent before the command depending on if there already is an established TCP session. This handshake data is something one should be aware of but it is not included in the comparison provided in this thesis. Additionally, a very small amount of packages (less than 1 %) have to be retransmitted in the case of both IoT frameworks. This retransmission of data is also not considered here as we are only discussing the typical scenario of a command that succeeds without any problems. We only consider this typical scenario since it makes it easier to get a clear comparison.

Results

In the case of HomeKit, each command produces three TCP packets that are sent over the network.

1. A 276 byte HTTP request sent from the HAP software to the device.
2. A 99 byte HTTP response sent from the device to the HAP software.
3. A 54 byte TCP ACK sent from the HAP software to the device.

This makes for a total of 429 bytes sent over the network to fulfill this one command.

If we instead consider the equivalent command in IoTivity, only two UDP packets are sent over the network.

1. An 89 byte CoAP request sent from the JavaScript testing script to the device.
2. An 87 byte CoAP response sent from the device to the JavaScript testing script.

This makes for a considerably smaller amount of data needed to fulfill the command. At a total of 176 bytes, IoTivity uses 41% of HomeKit's network resource useage. A bar chart visualizing this difference can be seen in Figure 6.8.

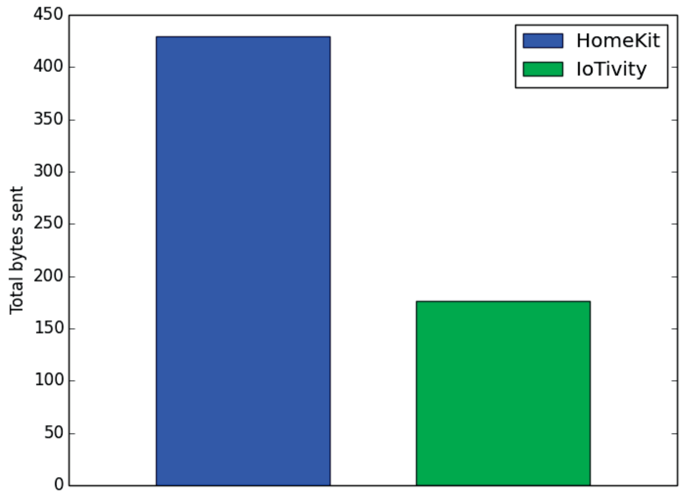


Figure 6.8: A comparison between the total number of bytes needed to perform a command in HomeKit and IoTivity.

Table 6.3: A table summarizing the evaluation that is presented in this chapter.

Measurable or Feature	HomeKit	IoTivity	Evaluation method
Interoperability			
Vendor Interoperability	Yes	No[1]	Specification
Bridging	Yes[2]	Yes	Specification
User interfaces	iOS only	Android, Web, Tizen, Other[3]	Specification, Prototype
Connectivity			
Physical Transports	Wi-Fi, BLE	Any[4]	Specification, Prototype
Device-to-device	No[5]	Yes	Specification, Prototype
Remote	Yes[6]	Yes	Specification
Scalability			
CPU	64 MHz	16 MHz	Manufacturer Data
Flash Memory	512 kB	256 kB	Manufacturer Data
RAM	64 kB	8 kB	Manufacturer Data
Network related metrics			
Response time	22ms	30ms	Experimental
Network traffic generated for one command	429 bytes	176 bytes	Experimental
Security			
Encryption	Yes	Yes	Specification
OTA	Yes	No[7]	Specification
Authentication	Yes	Yes	Specification

[1] Certification program will guarantee vendor interoperability once complete.
[2] Restrictions apply to what types of devices can be bridged.
[3] In theory, an OCF device can be controlled from any platform, the listed platforms already have IoTivity APIs.
[4] IoTivity is not dependant on a specific physical transport and can utilize any transport capable of IP.
[5] Any communication between devices must be routed through an Apple iOS device.
[6] Remote access requires an Apple TV 3rd generation or later.
[7] No support from specification as of version 1.0.0, up to device developers to implement.

Chapter 7
Discussion

The IoT and smart home landscape is evolving rapidly right now. New technologies are emerging and the existing solutions are still being developed to provide more functionality. It will probably take some time before the dust settles and a dominant standard is established. This thesis has evaluated some of the more promising alternatives on the market today to help provide guidance when choosing what smart home solution to implement in a product.

7.1 Results

Once the OCF certification process is complete, interoperability between devices from different vendors implementing the same protocols will likely not be an issue for any of the frameworks. The more interesting differences are found in two main areas. The approach to bridging and what platforms that can be used as "clients" controlling the devices. Apple's restrictions to bridging is most likely in place to minimize the risk of getting dragged into legal issues that might arise with devices that have not gone through their certification process. The inability to control HomeKit accessories from Android devices or through a web interface is also a large drawback. While IoTivity does not provide an API for iOS, all that is required to be able to control OCF devices is that someone makes an own implementation for iOS.

Considering that HomeKit is so closely bound to iOS, it is no surprise that the supported wireless technologies are the same as what is available on the iPhone. This tie is what distinguishes HomeKit from other available technologies. It is probably the most significant drawback of HomeKit, as the absence of device-to-device communication severely limits the possible applications for HomeKit devices. The OCF and IoTivity has a more open approach to connectivity which makes it much more appealing option in many application areas.

Another factor affecting the application area of the frameworks is that IoTivity can be deployed on much smaller platforms than HomeKit. The underlying reason for that being the difference in focus of the different frameworks. IoTivity is aimed at a broad spectrum of products, trying to be the ultimate solution to all kinds of IoT products. They chose to utilize CoAP and CBOR which are specifically tailored for the IoT to reduce the hardware load. HomeKit is more security focused and enforces heavier encryption which is a contributing factor to the higher system requirements of HomeKit devices compared to IoTivity.

A comparison of the security features discussed in this report shows that the two frameworks can be considered roughly equal. The biggest difference being the support for OTA updates that HomeKit has. OTA updates are essential to be able to continuously address new security threats that are discovered. At this point it is unclear if OTA firmware updates are going to be a requirement for certified OCF devices. It is feasible for a manufacturer to create an equally secure device by implementing this functionality on their own. As have been mentioned earlier this report evaluates security based on features provided and does not investigate the quality of the security.

7.1.1 Response time

The distribution of response times was measured in section 6.7.1. In the case of IoTivity, there is a very small probability that the response time exceeds 75 ms. The same figure for HomeKit is a little higher at 100 ms but it still does not exceed 0.1 second. This is important to note since this means that a user will get the impression of instantaneous execution of commands when interacting with a device based on either framework. The frameworks can therefore be considered to perform equally based on a usability perspective on response time. It should be noted however, that all network related measurements in this report were performed on a local network. How these IoT frameworks compare in a cloud connected setting is not measured or discussed.

7.1.2 Network traffic used

The network resource usage for IoTivity was measured to be considerably smaller than that of HomeKit in section 6.7.2. This is an expected result since IoTivity uses CoAP and CBOR while HomeKit is using the standard web protocols HTTP and JSON. Both of the technologies used by IoTivity are designed with the IoT in mind and are therefore optimized to use less network resources.

The two factors that make up this difference are packet size and number of packets. CoAP have a smaller header than HTTP and using CBOR reduces the size of a JSON document, which makes the individual packets smaller. When it comes to the number of packets, HomeKit sends three packets instead of the two that IoTivity sends. Both of the protocols send a request and receive a response in return for each command issued and the extra packet sent by HomeKit is a TCP ACK for the response message. This extra packet is unnecessary and sent because HomeKit is utilizing technologies that never was intended for use in IoT devices.

7.2 Market perspective

There are a number of things to consider when choosing what technology to use when building a smart home product. The most obvious being the technical limitations of the frameworks covered in this evaluation, such as scalability and connectivity options. The other factors include things such as the existing product base, legal aspects such as licenses, and brand reputation.

The Apple brand is probably the most prominent of these other factors. Apple already has a loyal consumer base and the reputation of building solid products that work well with other Apple products. They also have a large service back-end with features such as already built-in voice control via Siri and an existing cloud service. IoTivity lacks all of these but instead have the advantage of being open-source under an Apache 2.0 license and being more lenient on how some features can be implemented.

It is also worth noting that HomeKit is considered production ready with several products already released on the market. It is easy to work with and the documentation is well written. The OCF is still working on their certification process and the method of discovering and setting up new devices in a network is not yet fully specified.

7.3 Other competitors

There are other competitors already on the market such as ZigBee and AllJoyn. Google's Weave in combination with their IoT tailored OS Brillo also lurks on the horizon. When comparing the approaches that the different organizations has taken to the IoT area, IoTivity and Weave seem to be the closest in what they are trying to achieve. Google has the advantage over the OCF that they have control of a majority of the mobile market with their Android OS. Google also promise a solid cloud back-end with tools for analysis of user-data and OTA updates. However, the cloud-based solution also gives rise to privacy concerns because it gives Google access to the data of all users utilizing Weave products.

Weave is not evaluated in this report because it is still not released to the public. Google has not published any information of when it is expected to get to market.

Chapter 8

Conclusion

HomeKit and IoTivity are not in direct competition with each other and both the technologies can exist side by side. HomeKit is aimed mainly at Apple's already existing customer base with people who already own and use many of Apple's products. IoTivity's goal is to create an open-source market standard that can be used by everyone in any type of product. Both frameworks are close in terms of performance, the key differences lies in design choices and the scope of application areas.

A manufacturer wanting to make a HomeKit product will have to make trade-offs. On the one hand they will get the Apple brand at their back. But on the other hand they will have to accept that there is no device-to-device communication, that they have to install Apple's authentication chip on all their hardware and HomeKit requires the user to have an iPhone. In addition HomeKit also has the advantage that it works out of the box with Siri, which adds an extra dimension of controlling the devices.

IoTivity and the OCF do not have the same brand recognition among consumers, but instead do not limit the users or manufacturers by requiring special hardware. The OCF and IoTivity is also much more scalable and does not present any hindrances for device-to-device communication. This allows for a broader range of products enabling a network setup consisting of several devices communicating with each other to create a better user experience.

The OCF still has a lot of work ahead if they want to set the standard for building IoT devices. IoTivity shows great potential for the future and as long as they can continue developing the standard and manage to engage device manufacturers they have a good chance of becoming for IoT what HTML is for the Internet.

HomeKit is likely to be the first choice for Apple enthusiasts, with their superior integration with other Apple products and services. The restriction to only allow iOS devices for controlling HomeKit products effectively locks out the rest of the consumer market from using HomeKit. Because of the iOS lock, it is unlikely that HomeKit will become the market leading standard for smart home products.

References

[1] "Ericsson mobility report." http://www.ericsson.com/res/docs/2015/mobility-report/ericsson-mobility-report-nov-2015.pdf, nov 2015. p 10.

[2] "Allseen alliance documentation." https://allseenalliance.org/framework/documentation/learn
Accessed 15/2 2016.

[3] Y. Wang, L. Wei, Q. Jin, and J. Ma, "Alljoyn based direct proximity service development: Overview and prototype," in *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on*, pp. 634–641, IEEE, 2014.

[4] "Allseen alliance faq." https://allseenalliance.org/alliance/faq. Accessed 15/2 2016.

[5] "Zigbee press release: Zigbee certified products surpass 1,000." http://www.zigbee.org/zigbee-press-release-zigbee-certified-products-surpass-1000/
Accessed 16/2 2016.

[6] "Zigbee alliance." http://http://www.zigbee.org/
Accessed 17/2 2016.

[7] D. Gislason, *ZigBee Wireless Networking*. Newnes, 2008.

[8] "Thread group." http://threadgroup.org/
Accessed 16/2 2016.

[9] "Thread overview whitepaper." http://threadgroup.org/Portals/0/documents/whitepapers/Thread%20Stack%20Fundamentals_v2_public.pdf, jul 2015. Accessed 16/2 2016.

[10] "Getting started with brillo & weave - device, mobile, cloud (ubiquity dev summit 2016) (video)." https://www.youtube.com/watch?v=thUJARumXWE, jan 2016. Accessed 22/5 2016.

[11] "Oic members." http://openconnectivity.org/about/membership-list
Accessed 15/2 2016.

[12] "Open connectivity foundation faq." http://openconnectivity.org/about/faq
Accessed 15/2 2016.

[13] C. Bormann, A. P. Castellani, and Z. Shelby, "Coap: An application protocol for billions of tiny internet nodes," *IEEE Internet Computing*, vol. 16, no. 2, p. 62, 2012.

[14] R. Fielding and J. Reschke, "Rfc 7230: Hypertext transfer protocol (http/1.1): Message syntax and routing." https://tools.ietf.org/html/rfc7230, 2014.

[15] Z. Shelby, K. Hartke, and C. Bormann, "Rfc 7252: The constrained application protocol (coap)." https://tools.ietf.org/html/rfc7252, 2014.

[16] E. Rescorla and N. Modadugu, "Rfc 6347: Datagram transport layer security version 1.2." https://tools.ietf.org/html/rfc6347, 2012.

[17] C. Bormann and P. Hoffman, "Rfc 7049: Concise binary object representation (cbor)." https://tools.ietf.org/html/rfc7049, 2013.

[18] "Oic - open connectivity foundation brings massive scale to iot ecosystem." http://openconnectivity.org/news/open-connectivity-foundation-brings-massive-scale-to-iot-ecosystem, feb 2016. Accessed 28/4 2016.

[19] "Oic - oneiota data model tool." http://openconnectivity.org/resources/oneiota-data-model-tool. Accessed 28/4 2016.

[20] "Iotivity programmers guide." https://www.iotivity.org/documentation/linux/programmers-guide
Accessed 29/4 2016.

[21] "Oic core specification v1.0.0." http://openconnectivity.org/resources/specifications, 2015. Accessed 19/4 2016.

[22] H. Virji, "The layered architecture of iotivity - samsung open source group blog." https://blogs.s-osg.org/layered-architecture-iotivity/, nov 2015. Accessed 2/5 2016.

[23] "Iotivity features." https://www.iotivity.org/documentation/features
Accessed 29/4 2016.

[24] "Registering a resource | iotivity." https://www.iotivity.org/documentation/linux/programmers-guide/registering-resource.
Accessed 29/4 2016.

[25] "Finding a resource | iotivity." https://www.iotivity.org/documentation/linux/programmers-guide/finding-resource. Accessed 29/4 2016.

[26] "Querying resource state [get] | iotivity." https://www.iotivity.org/documentation/linux/programmers-guide/querying-resource-state-get. Accessed 29/4 2016.

[27] “Setting a resource state [put] | iotivity.” https://www.iotivity.org/documentation/linux/programmers-guide/setting-resource-state-put. Accessed 29/4 2016.

[28] “Observing resource state [observe] | iotivity.” https://www.iotivity.org/documentation/linux/programmers-guide/observing-resource-state-observe. Accessed 29/4 2016.

[29] “Introducing homekit.” https://developer.apple.com/videos/play/wwdc2014-213/, 2014. Accessed 22/2 2016.

[30] “What’s new in homekit - wwdc 2015 - videos - apple developer.” https://developer.apple.com/videos/play/wwdc2015/210/. Accessed 3/3 2016.

[31] “Designing accessories for ios and os x.” https://developer.apple.com/videos/play/wwdc2014-701/, 2014. Accessed 22/2 2016.

[32] “Bonjour concepts.” https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/NetServices/Articles/about.html. Accessed 29/2 2016.

[33] “Srp: What is it?.” http://srp.stanford.edu/whatisit.html. Accessed on 03/01/2016.

[34] B. et al., “High-speed high-security signatures.” http://ed25519.cr.yp.to/ed25519-20110926.pdf, sep 2011. Accessed 1/3 2016.

[35] D. J. Bernstein, “Curve25519: new diffie-hellman speed records.” https://cr.yp.to/ecdh/curve25519-20060209.pdf, feb 2006. Accessed 1/3 2016.

[36] T. Hansen, “Rfc 6234 - us secure hash algorithms (sha and sha-based hmac and hkdf).” https://tools.ietf.org/html/rfc6234, may 2011. Accessed 1/3 2016.

[37] Y. Nir and A. Langley, “Rfc 7539 - chacha20 and poly1305 for ietf protocols.” https://tools.ietf.org/html/rfc7539, may 2015. Accessed 1/3 2016.

[38] “Apple’s homekit is proving to be too demanding for bluetooth smart home devices - forbes.” http://www.forbes.com/sites/aarontilley/2015/07/21/whats-the-hold-up-for-apples-homekit/#656731a2322b. Accessed 8/3 2016.

[39] “Wiced hot sheet.” http://www.mouser.com/ds/2/678/MMPWICED-HS102-R-773833.pdf. Accessed 2/5 2016.

[40] “nrf51 dk / products / home - ultra low power wireless solutions from nordic semiconductor.” https://www.nordicsemi.com/eng/Products/nRF51-DK. Accessed 2/5 2016.

[41] “nrf51 series soc / products / home - ultra low power wireless solutions from nordic semiconductor.” https://www.nordicsemi.com/eng/Products/nRF51-Series-SoC. Accessed 2/5 2016.

[42] “The yocto project.” https://www.yoctoproject.org/about. Accessed 2/5 2016.

[43] G. Gediga, K.-C. Hamborg, and I. Düntsch, "Evaluation of software systems," *Encyclopedia of computer science and technology*, vol. 45, no. supplement 30, pp. 127–53, 2002.

[44] J. Nielsen, *Usability engineering*. Elsevier, 1994.

[45] "Mfi program." `https://developer.apple.com/programs/mfi/`. Accessed 18/5 2016.

[46] "Apple homekit release date rumours: First homekit enabled devices on sale next month." `http://www.macworld.co.uk/feature/apple/apple-homekit-release-date-rumours-3585269/`. Accessed 18/5 2016.

[47] "Ocf certification." `http://openconnectivity.org/certification`. Accessed 18/5 2016.

[48] "Iotivity documentation." `https://www.iotivity.org/documentation` Accessed 29/4 2016.

[49] "Use homekit-enabled accessories." `https://support.apple.com/en-us/HT204893`. Accessed 19/5 2016.

[50] "Oic remote access specification v1.0.0." `http://openconnectivity.org/resources/specifications`, 2015. Accessed 19/4 2016.

[51] "Tizen micro, small footprint tizen for headless iot devices." `https://download.tizen.org/misc/media/tds2014/slides/Tizen-Micro-Bingwei%20-%20Liu_en.pdf`. Accessed 20/5 2016.

[52] "Google android system requirements." `http://www.mycomputeraid.com/computers/google-android-system-requirements/`. Accessed 20/5 2016.

[53] "Yocto faq." `https://www.yoctoproject.org/question/what-smallest-footprint-yocto-project-can-support-effectively`. Accessed 20/5 2016.

[54] "Oic security specification v1.0.0." `http://openconnectivity.org/resources/specifications`, 2015. Accessed 19/4 2016.

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2016-520
http://www.eit.lth.se