Designing an open source IoT Hub: bridging interoperability and security gaps with MQTT and your Android device

Melissa Adriana Simoes Saial Real, Hamed Haddadi
Queen Mary University of London

Abstract

With constantly evolving hardware and increased competitiveness from manufacturers in the construction of the IoT enabled home, the difficulty in managing and securing the multitude of internet enabled devices at any individual's disposal is ever greater, with competing applications tailored to manage Bluetooth devices, Wi-Fi Direct or NFC enabled "things". While the means of connectivity are ever increasing, the lack of a single standard of IoT connectivity as well as the lack of a single interoperability solution difficult consumer adoption of an internet enabled home.

The solution to these issues is here presented in the form of a single, simple, user-friendly interface that can be intuitively used by any consumer. Pairing this interface with an optimal communication protocol will assist in bridging the interoperability gap and provide the necessary abstraction layer to facilitate the interchange of data regardless of which device is being used. This paper proposes that the solution for both these issues lies with leveraging the capabilities of mobile devices, in this case particularly targeting Android, paired with an integration of the lightweight communication protocol MQTT.

Introduction

Designing an IoT hub is not a new concept: Lea and Blackstock [20] proposed it in 2014 as well and Amazon's Alexa facilitates the connections to different intelligent devices in the home, acting as a central hub for managing devices such as Philips Hue lightbulbs and TP-Link Smart Plugs. However, what is recent is the motivation to propose an open source solution.

With academic papers frequently discussing IoT magement solutions without open sourcing code, it was necessary to provide an accessible and testable solution that could be explored and tinkered with by other interested parties. We will showcase the different options provided and the only competing open source solution thus found – home assistant.

This paper will thus showcase the steps in building the ideal hub service, and how any individual can leverage the power of an open-source Java Android application aimed at facilitating the discovery of services provided by Internet enabled devices within a local area network. Afterwards we demonstrate how simple it is to establish a communication channel between an android device and a Mosquitto broker using the MQTT Protocol and an Android application provided by the Eclipse IoT project. This paper will further explore the current options provided for service discovery in the context of IoT, the most relevant protocols and architectures used in connecting these devices as well as the attempts that have been made at establishing a standard means of connectivity in the scope of the Internet of Things.

Defining IoT

The definition of IoT varies widely in academia. For the scope of this paper we begin by defining an IoT as an Internet enabled entity, following the concept given by de Melo et al in [1] of *"all application comprising objects or devices that can interact with other objects and*

applications over the internet". This interpretation understands IoT not to be necessarily physical, since IoT applications can be simulated using services such as AWS IoT or IBM's Bluemix. Core to this definition is the capacity of IoT to connect with different objects/applications and carry out functions, and so, an IoT device is here considered as a physical entity while IoT refers to a more general concept of an interconnected entity. Another valid interpretation that helps us define the architecture in which IoT is integrated is to propose not just "an IoT" but to perceive the Internet of Things as the system of interconnected "things". This means that rather than referring to IoT as individual entities, the word IoT is referring to a whole cognitive system, where "things" contribute data and intelligence [2]. In this approach, a "thing" is the IoT entity, and that thing can be a drone, a node, an intelligent toaster or any other deemed "intelligent" machine. The IoT network architecture in this sense describes the interaction of things within this system. A simplified version of this architecture is presented in [2] and states its four main components as the things (sensing devices), a communication network, a cloud and the back-end IoT applications. The devices use the IoT applications to make use of their services, and then reach the cloud via the communication network to make use of cloud services such as data storage. An illustration of this interaction is also provided by the authors.

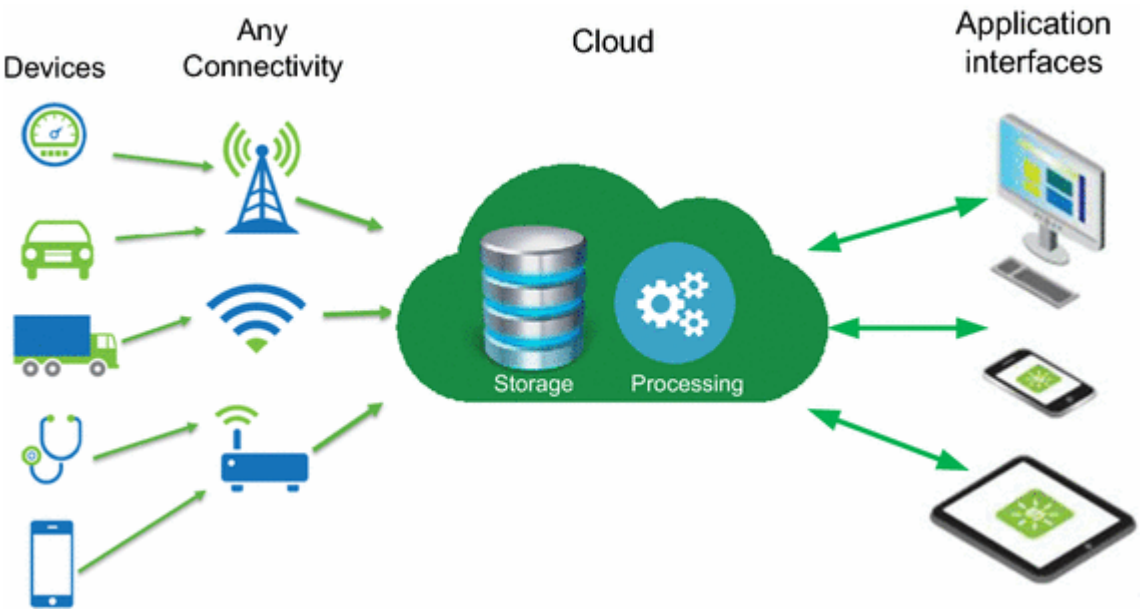


Figure 1. IoT network architecture [2]

Challenges to IoT connectivity

With over 8 billion connected devices at the time of writing [3], the number of connected devices may fail to meet the commonly quoted prediction of Cisco's 2011 report [4] of 50 billion devices for 2020, but it is undeniable that IoT is quickly becoming commonplace as the number of devices already surpasses the world population. Thus, IoT becomes a relevant subject to research, together with the topic of how we connect, interact and have data collected by these devices. The dialogue on these matters has been recently intensified by the security breaches that have seen millions of IoT devices hijacked for the purpose of DDoS attacks, which have exposed critical security flaws in the design and implementation of IoT communication protocols. One such example of these attacks is the "Mirai" virus

which hijacked over 900 thousand routers of Deutsche Telekom customers [5] by taking advantage of a telnet vulnerability.
These security flaws are a consequence of both the difficulty in implementing encryption on devices with constrained processing power [17], as well as a result of the rapid expansion of IoT technology without establishing a standard for a secure IoT architecture. Such a standard is still missing [6], not due to lack of attempts to establish one - as we will explore in section two - but mainly due to the rapid creation of different products prior to consideration for expansion, security and compatibility with different technologies. This, paired with a desire on the part of certain manufacturers[1] to not provide compatibility across different brands of devices, has caused a hindrance to the effort of standardisation. The need for a standard that enables best security practices to protect IoT consumers is thus the motivation for this paper and exploration of tools that allow the construction of an easy-to-use application which aggregates available IoT services in a single interface.

Current standardisation efforts

Multiple open protocols appropriate for IoT have been recognised by standardisation bodies including the European Telecommunications Standard Institute(ETSI), The Organization of Structured Information Standards(OASIS), the Open Mobile Alliance(OMA) and the Internet Engineering Task Force (IETF). These include HTTP, CoAP and MQTT [7]. In terms of communication protocols, these have different components that can be integrated into applications to allow for device discovery and for the design of an IoT search system. However, these protocols do not define a single architectural model as they can be simply integrated into different IoT design infrastructures that support them.
Promoting the implementation of protocols, instead of hardware dependant communication components, signals a departure from how traditional communication has been established between IoT. For years, Bluetooth, ZigBee and other short-range transmission technologies have been used as gateways for accessing IoT devices and services [16] but the trend towards cloud computing and service virtualisation has required adjustments to be made on how we connect to IoT to allow for expansibility and IoT cloud access.
Flexible interoperability can be facilitated by the use of Wireless technologies being integrated into IoT devices to allow direct communication between things, without the need to be connected via a router. This is the standard promoted by Wi-Fi Direct, endorsed by the Wi-Fi Alliance, which allows for devices to talk to each other directly via a software access point [8]. Rather than using a router or even needing an internet connection, devices can connect to a group of multiple devices or make one-to-one connections. One device defines the software access point and becomes the owner of a given device group. This enables compatibility with older Wi-Fi devices that do not have Wi-Fi Direct, with the added benefit that Wi-Fi Direct already integrates security mechanisms within its specification and includes WPA2 protection [8].

---

[1] Apple's proprietary HomeKit software is an example, as users are required to purchase additional hardware in order to bridge the communication with devices from different manufacturers. One example of this are the Philips Hue IoT bulbs, which hardware is available from: https://www.apple.com/shop/product/HJE22VC/B/philips-hue-homekit-upgrade-bridge-for-current-hue-bridge-users

While Wi-Fi Direct is certainly an advantageous approach for systems that are enabled to support this technology, it is not without its faults - authors in [18] highlight a disadvantage of using wireless technologies, because these require an access point to be setup for Wi-Fi communications to be established between devices. The authors also reflect on other hardware, such as the need for devices to be running the same OS when it comes to Bluetooth communications.
As so, an ideal solution would not rely on a particular technology for interconnectivity but instead would allow for an abstraction layer from the underlining technology and permit interoperability between different hardware. Most protocols integrated into IoT only require the capacity of devices to communicate using HTTP/REST methods, which integration into IoT communication layers will be further explored in the next section.
A feature of protocols is requiring their respective implementations to provide for security, meaning security concerns are addressed differently with each implementation. Simply having a protocol to establish communication is not enough to program an IoT interoperability solution since an implementation of each protocol is required as part of a full application to be provided as a service the consumer can use. It is so important to understand how these protocols function as interoperability enablers and can be introduced into application design.

Communication Protocols for IoT connectivity
Machine-to-machine (M2M) communication is enabled by the web technologies of HTTP/REST that expose services using HTTP type requests and by using URIs as a means to identify resources [10]. Web transfer protocols such as MQTT and CoAP enable application transfer even when resources are limited [9], thus making these ideal to work with constrained devices - which are defined by being characteristically low powered and having only a few kb of RAM memory. Constrained devices do not have the capacity for peak operating systems such as Linux to be run in them.
When interacting with HTTP, CoAP provides a level of abstraction that avoids using specific application data to provide interaction, many times using an intermediary to translate the data transmitted in between both protocols [9]. The usage of URIs to identify resources by both protocols also facilitates the interchangeability of communications and traffic interceptions between them. As CoAP relies on REST connectivity it can be applied regardless of which hardware is being used to communicate, since clients can simply use GET requests to ask for resource updates from the server containing the resource [14].
While there are many different options for protocol implementations, the one chosen for this project and which is commonly used with constrained devices is the simple data exchange service MQTT. Created by IBM and Arcom to facilitate Machine to Machine data exchange, MQTT is a lightweight publish/subscribe protocol that has an accepted standardised specification by OASIS[15] and uses a client/broker system, which allows for two or more brokers to connect via a bridging system. Bridging allows for the connection of brokers by having one of the brokers initiate a bridge by defining a topic that all interested brokers/clients can subscribe to. This method also allows for clients to send messages to an MQTT broker, and those clients can also subscribe to the messages sent to the broker and receive updates notifying them of new messages.
A common broker implementation used with MQTT is Mosquitto[12], as it provides a solution that is also open source, lightweight and compatible across different platforms. It was the broker implementation of choice by the authors in [13] to create a message

transmission system that relies precisely on this MQTT and Mosquitto combination to reliably exchange messages in an IoT setting**.**
In order to use MQTT with a client implementation, a well-established option is provided by Eclipse Paho [38]. Paho has multiple open source client libraries such as Java, Python and JavaScript available, and Amazon also enables the usage of the Paho MQTT with WebSocket to connect to AWS IoT [11] - WebSocket is a communication protocol that opens communication channels over TCP connections. Within the next section we will explore how to use the MQTT Paho Java Android client to push notifications about devices to a Mosquitto broker. But firstly, we will explore different approaches that integrate the described protocols into the different layers of their respective IoT management solutions.

Former proposed solutions and implementations
The challenge posed by providing a communication structure between a service layer and different IoT architectures can be addressed by using a combination of protocols and client libraries and turning these into an API. One example is the solution proposed de Melo Silva et al in [1], in the form of an API based on UPnP standards, using REST and SOAP requests to retrieve data and create objects off that data, then passing the created object to a component handler module. While the API presents a useful solution to expose REST services it has not been incorporated in a complete application nor do the authors make it available for use via an open source means such as GitHub, providing a challenge in terms of testing the functionality of the described application.
Out of the different architecture designs described, the seemingly most convenient from a consumer facing perspective was the suggestion of a full solution as an application with an easy to use client interface, which describes a model for a "hub" management system. Authors in [19] propose the hub or gateway methods as those with most promise to deliver interoperability between devices, with an example of such as system described in [20]. The idea presents an open source hub that connects different IoT by allowing a gateway for interoperability amongst these.
The architecture of Lea and Blackstock's [20] described hub model contains a data aggregator that serves as an access point to services. In their model, the authors implemented a CKAN data harvester to aggregate data information about things and use the WotKit API[2] to find these things, or more specifically, to find the sensors that different IoT has for the purpose of locating physical or virtual IoT device capabilities. The information from both APIs is then sent in the form of a catalogue of resources, which is interpreted using the HyperCat specification, meaning IoT resources that had been exposed by both APIs are described as a catalogue of URI resources. Certain difficulties are pointed out by the authors regarding the use of this system: Mainly, when querying the HyperCat catalogue of items it is necessary to request specific metadata key value pairs, and since not all sensor data provided by the chosen WoTKit APIs exposed information in the form of metadata keys and values, it was not always possible to query available IoT resources off the IoT catalogue.
Moving on to work that has been recently published on the IoT arena, a practical example of a modern device discovery and management implementation is home assistant [22]. Home assistant is a python based open source IoT management application that has a simple command line installation. Once all the app dependencies are installed and running, home

[2] Further information about the WotKit API is available from https://wotkit.readthedocs.io/en/latest/user/quickstart.html#quickstart

assistant provides a straightforward interface to the user, displaying all the devices the system is able to find within the network range.

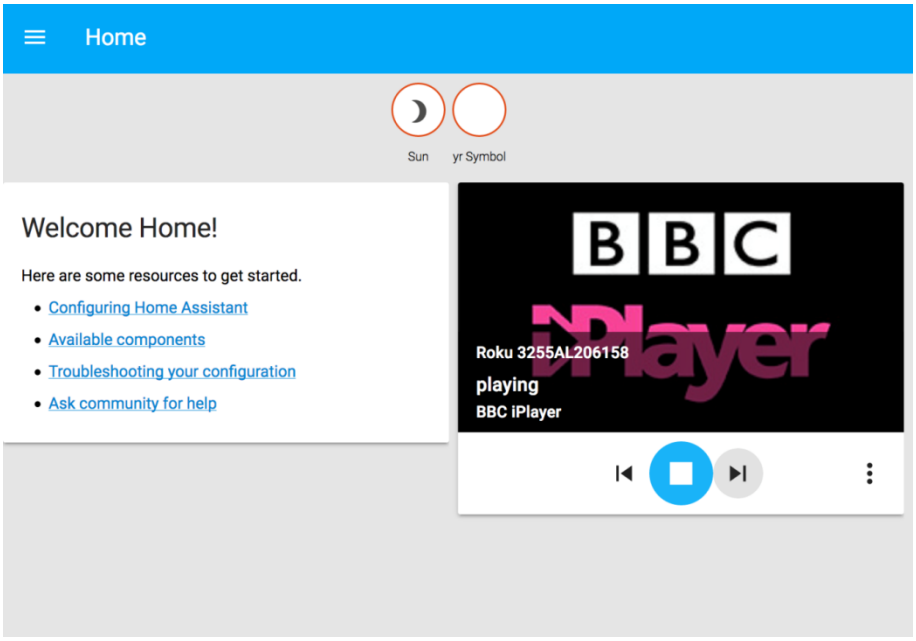


Figure 2. Home-assistant interface displayed after the system located one of the IoT devices present in the network, a Roku NowTV box

The service discovery methods implemented by home assistant use a combination of libraries that search for components and adjust to the need of each individual component. Consequently, there are separate code libraries depending on what resource is to be found, with specific code library extensions and instructions for adding devices such as an Amazon Fire TV Stick[3].
Out of the academic solutions researched, none provided access to the code implementations of their respective implementations. The only complete solution found during the research for this project that offers an open source repository, which code could be investigated and tested, was the python based home-assistant [22]. There is thus the need for an open source solution that facilitates communication and transparency between IoT devices, thus better bridging both the theory that supports the application design as well as clear explanation of the components that enable the discovery and communication of devices within a given network.

Discovering services on a LAN: The Port Authority Application

To assess the efficiency of the Port Authority app, we can compare the found devices against those connected to our home router, and so the least we'll expect from an efficient discovery service is six connected devices, the same ones displayed connected to our sky home router:

[3] Instructions on setting up the Fire TV extension are available from https://home-assistant.io/components/media_player.firetv/

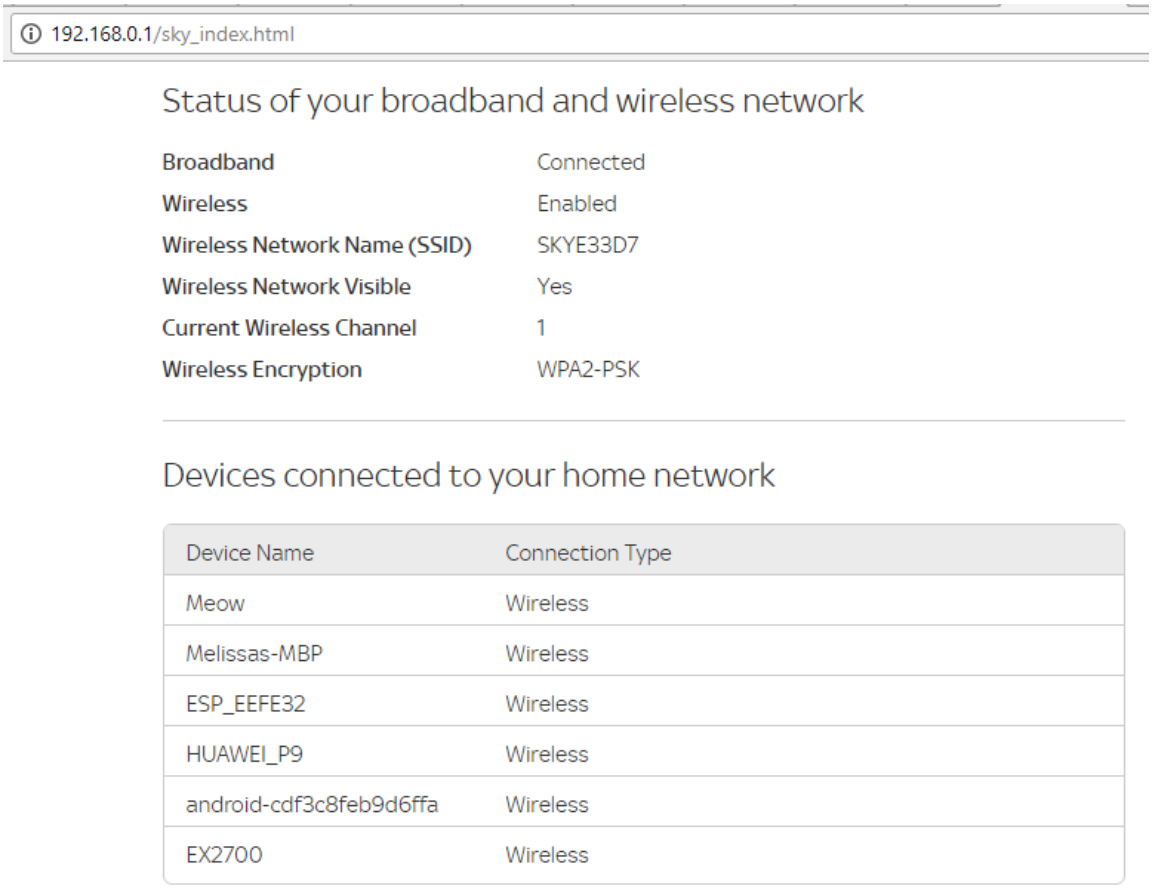
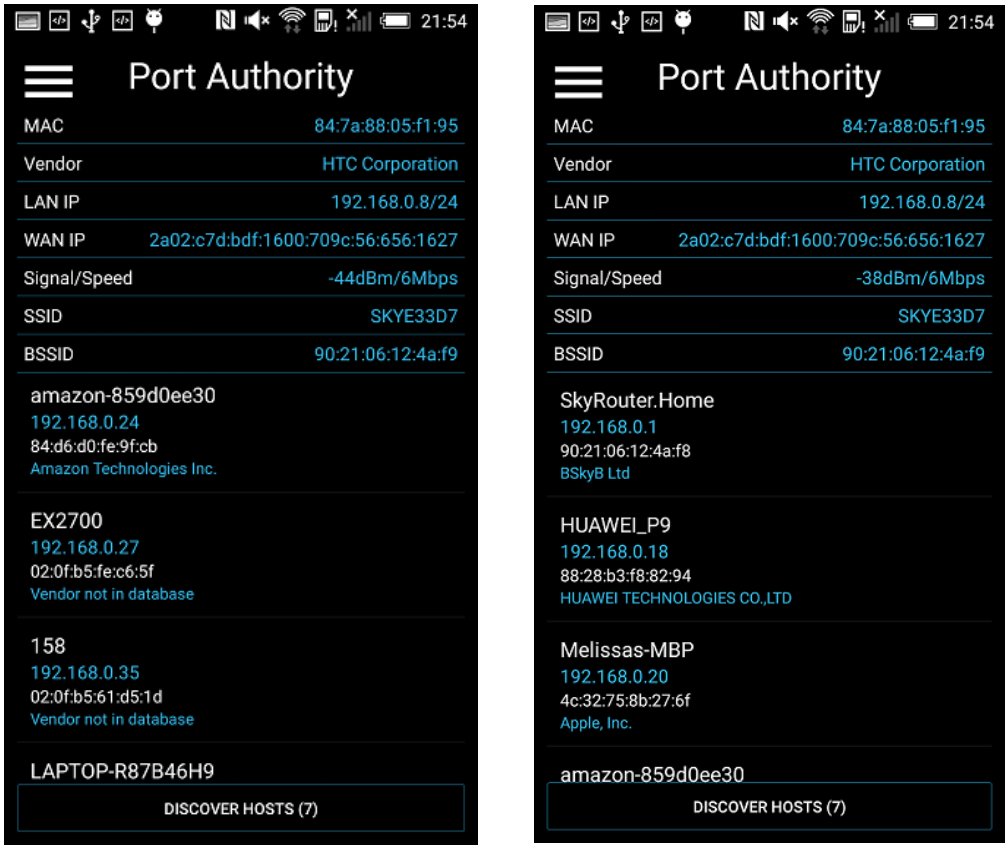


Figure 3. Home router listing of connected devices

Port Authority is an open-source tool with an up-to-date repository provided by [23]. The Port Authority Android application runs network scans using a combination of a native service discovery API as well as a few native Android methods – Android has libraries for Network Service Discovery which can support discovering HTTP services when its serviceType parameters are set to “http._tcp” or discovering printer types with serviceType set to “_ipp._tcp” [24]. The application uses asynchronous threading techniques to make the discovery processes run faster on the background.

The results exposed all the devices connected to the LAN, while further providing ports information on each of the exposed addresses. It discovers seven hosts, since it lists not only the devices connected on the network but also our sky hub router. The information and implementation provided were the most detailed and quickest to run out of different options tested, reason why the Port Authority notoriously stood out as an optimal discovery service to support the structure of an IoT home hub.



Figures 6 and 7. Port Authority discovery results

The next steps were to trial our chosen communication protocol before testing how to push messages via the MQTT broker.

Setting up MQTT as a message broker for the IoT discovery service

To create an IoT hub application, it is necessary to integrate a messaging protocol that can search and push notifications to devices within a given LAN or a virtualised environment. Current solutions that are compatible with Internet of Things devices need to be lightweight due to the simplicity of hardware, storage and memory limitations of constrained devices. The authors in [21] highlight four messaging protocols as most suitable for IoT compatibility, namely AMQP, MQTT, ZeroMQ and ZMPP. Their paper compares the performance of these messaging protocols in dealing with publishing and subscription of IoT in a virtualised environment. The authors in this comparison paper denoted the ability of MQTT to perform above average for transporting multiple sensor loads, and so the choice MQTT application derived from a combination of factors: MQTT offers a well-documented, up-to-date and simple to configure protocol, easily adjusted to fit the needs of any IoT application. Further, MQTT has been adopted by companies currently leading the IoT space, namely the online retailer Amazon, which has implemented the MQTT protocol in its IoT solutions albeit with a few modifications [11], as well as social media giant Facebook currently using MQTT for its Messenger App [25].

MQTT works on a client/broker model. Consequently, the first step to use this communication protocol was to set up an online broker that would allow for multiple

devices to communicate over the internet. Firstly, to test the protocol capabilities we set up an open-source source broker, Mosquitto. After setting up the broker, a client implementation is necessary, and the one used was the open source Paho Java client [38].

Installing and launching the Mosquitto broker was a straightforward process, Figure 8 demonstrates its first usage and activation.

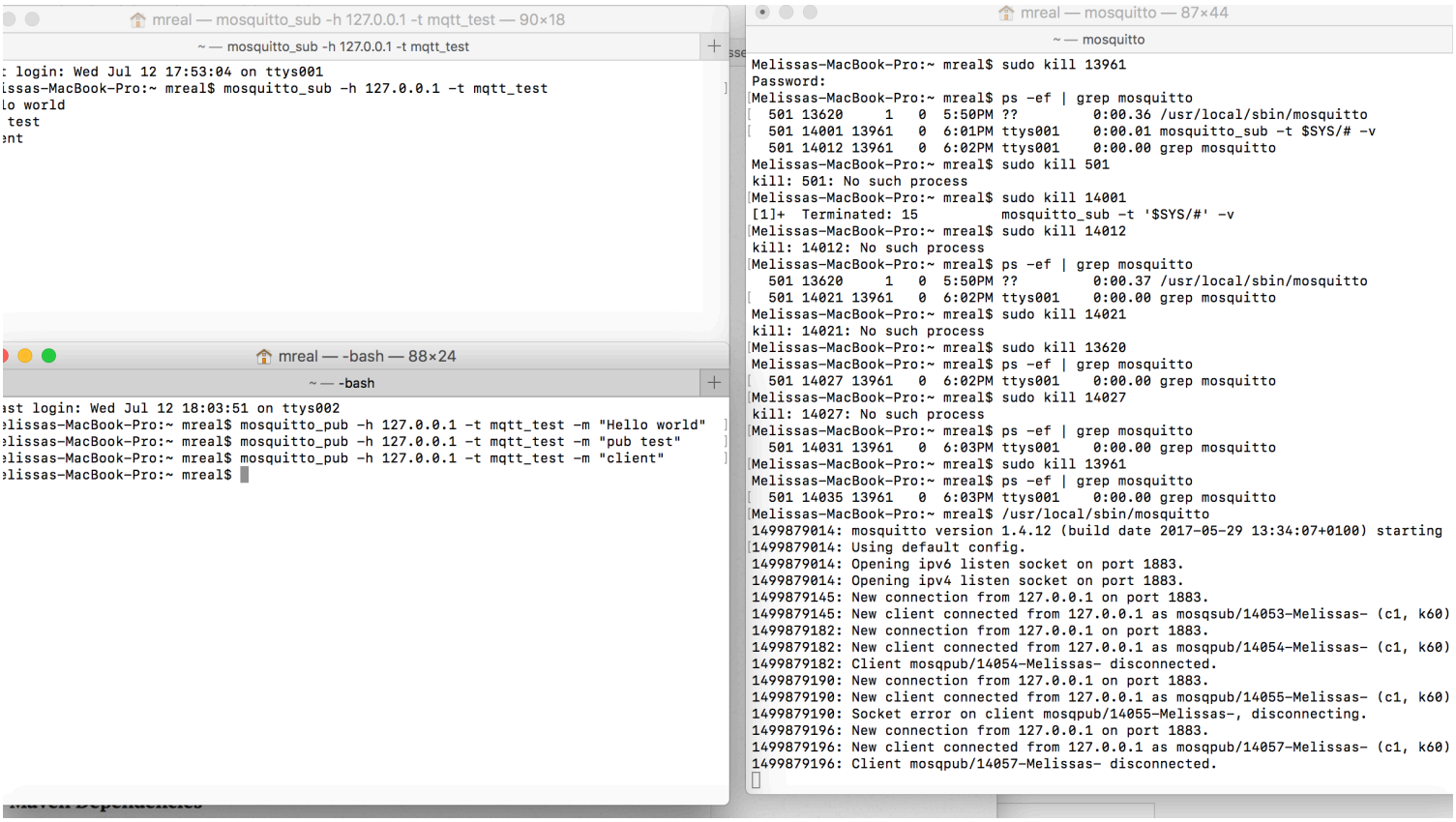


Figure 8. Running the Mosquitto broker and publishing/subscribing to topics

This use case exemplifies the properties of the Mosquitto broker, where we set Mosquitto to run in one Terminal, and simulate a client by creating a topic in another terminal window with mosquitto_pub, and yet another client on a third terminal window subscribing to the same topic and receiving updates published to the broker. We can also see on the right hand side how Mosquitto only opens the connection for establishing communication and closes it straight after the message is pushed. This demonstrates why MQTT with Mosquitto is useful for constrained devices, as the connection is only established for as long as necessary to transmit the message, thus occupying minimum bandwidth. A Raspberry Pi was also used to keep running the Mosquitto broker for application testing, this way allowing for different devices on the network to connect to its server via SSH. We also tested using Mosquitto on the Pi 3 (Figure 9.).

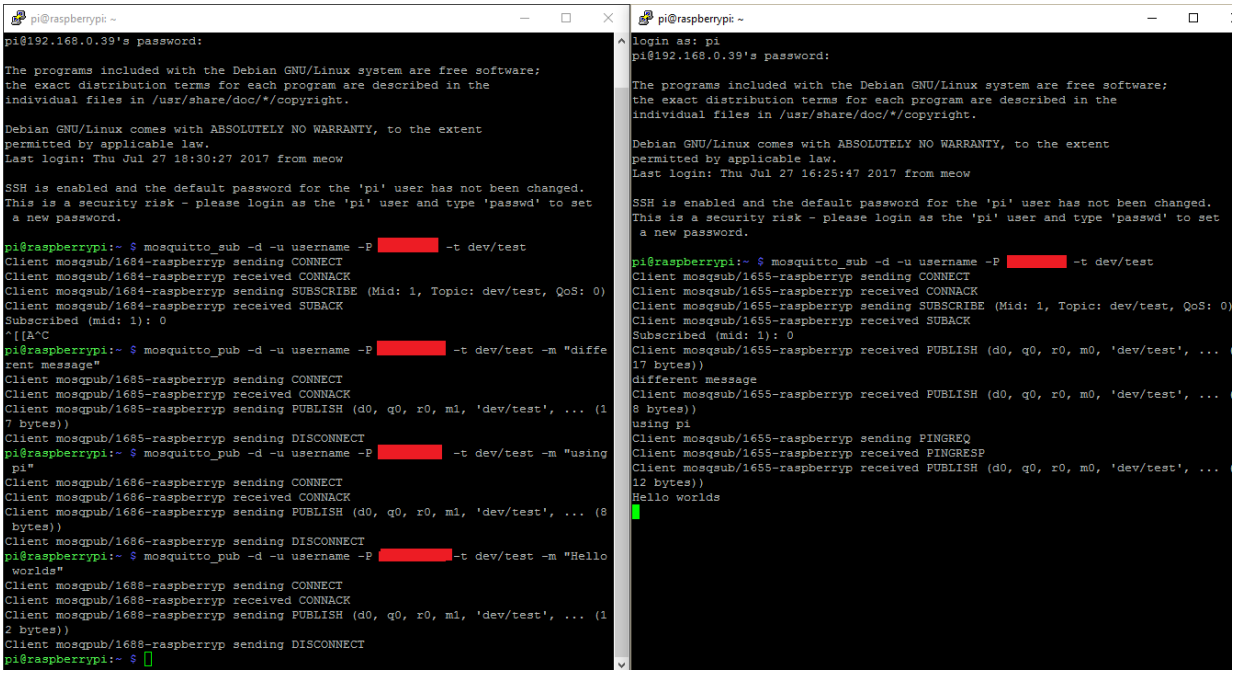


Figure 9. Pi 3 Mosquitto broker

Sitting on top of the TCP/IP stack means that MQTT relies on a client opening a TCP connection to establish an MQTT connection and send messages across the client and broker [38]. Running MQTT allows for communication to happen between devices that are running on the same LAN via network sockets, but communication is also possible from outside the LAN by using web sockets, which establish endpoints for internet connections to occur between services and devices. By adding a web socket listener to the MQTT configuration it is possible to transmit messages beyond the local area connection [38], but for the scope of this project we will limit the messaging to be transmitted to devices in our LAN.

The purpose of this paper was not only to facilitate the creation of an IoT hub, but also to gather a means to better explore the services being offered by IoTs on a given LAN, and so perceive whether our devices are exposed to security risks by having certain ports open and accessible via the internet. The next steps in improving the application require comprehending how best security practices are put in place, and understanding which security measures are currently possible in the scope of the IoT and MQTT in particular. The next section introduces the current state of security threats for these devices as well as proposed methods to tackle IoT malware threats.

Security concerns handling IoT connections

The susceptibility of IoT devices to hacking became far more prominent in the last quarter of 2016, when the Mirai virus harvested millions of connected devices to form a botnet that would bring down the DNS provider Dyn [26], thus causing the unavailability of services from websites including Twitter, Netflix and Spotify. This Distributed Denial of Service (DDoS) attack overwhelmed the capacity of the service providers that used Dyn, which were flooded with data sent and requested by the hijacked IoT devices, so taking advantage of both the vulnerability of IoT devices and the vulnerability of DNS.

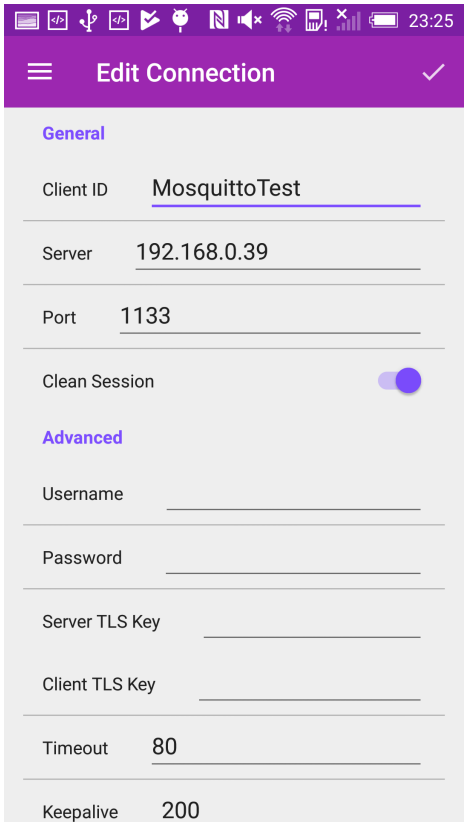


Figure 20. Configuration of the raspberry pi IP to act as a local server for the MQTT message exchange

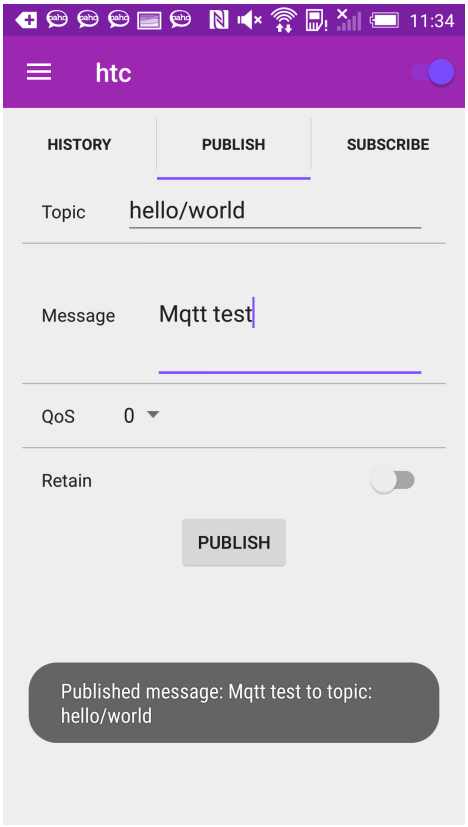


Figure 21. Paho MQTT Client on an HTC One Android Device, sending messages to the raspberry pi mosquitto broker

Mirai provides an interesting example when exposing vulnerabilities in the scope of IoT. At its core, the virus has a simple structure: it scans for random IP addresses of internet connected devices that have weak password policies, and uses a lookup table of common username/password combinations to connect to these devices via brute-force, that is, by attempting all possible password combinations for these [30][34]. The malware starts by exploiting password vulnerabilities, and then inserts the Mirai bot into the vulnerable device. The bot stays latent until a request to attack a given server is emitted by the command and control (C&C) server. Because rebooting deletes the virus from the device, Mirai also provides a "keep alive" method to prevent this – although re-infection, when a vulnerable device is connected to the internet, takes only an estimated 5 minutes based on the experience conducted by [31]. The McAfee security report from April 2017 provides an illustration of the Mirai architecture [34].

Another Mirai particularity is that it also provides for means to expel other viruses from the same device. It searches for common malware executables, deletes them from the system and closes vulnerable ports to prevent further infections and gain complete control: it can terminate applications bound to SSH or Telnet ports [34, p.18], and after infection it closes port 22 for SSH, 80 for HTTP and 23 for Telnet.

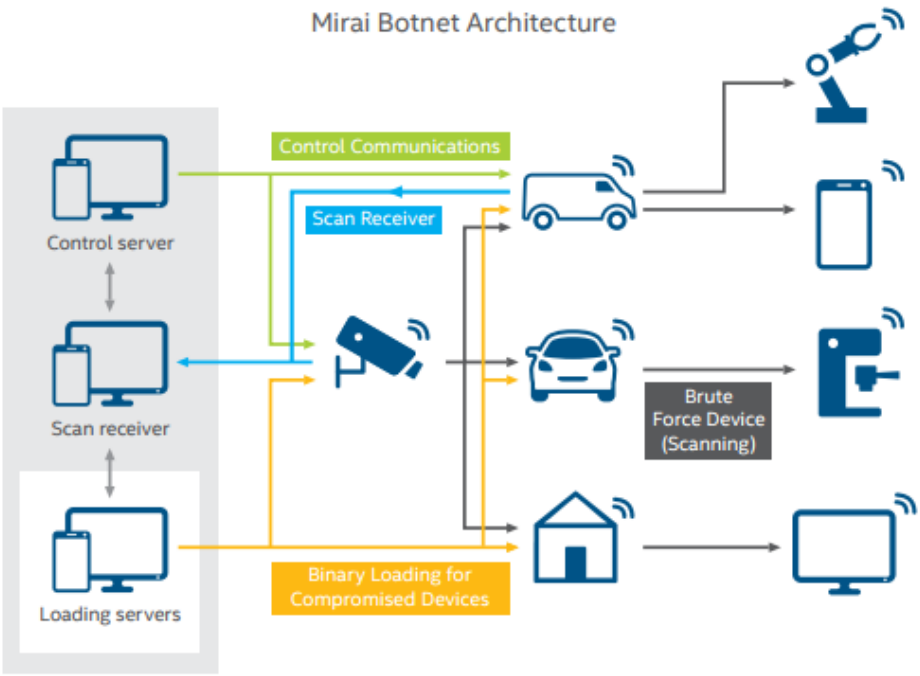


Figure 24. Mirai architecture from McAfee's threat report [34]

This particular feature of Mirai, closing ports 22, 80 and 23, emphasises why IoT devices are especially vulnerable – IoT that keeps specific ports open for accessibility and remote management without requiring any form of authentication facilitates hijacking of its functions. This vulnerability is what prompted the previously mentioned attack that took over 900 thousand routers – Deutsche Telekom had kept TCP port 7547 open to allow for remote management of the routers, without limiting internet connections from accessing this port.

```
busybox iptables -A INPUT -p tcp --destination-port 7547 -j DROP
busybox killall -9 telnetd
```

Figure 25. Mirai command that initiates malware attack [35]

The fault on port 7547 was patched by an update provided by the company, which could only be delivered after customers rebooted their respective devices to clear the virus and receive the update [36]. However, while Deutsche Telekom customers may no longer need to worry about this particular vulnerability, combatting new malware strands such as Mirai derivatives remains a challenge since many of the devices it hijacks, especially older IoT models, contain old hardware that cannot be updated, or which default passwords cannot be changed [32]. Websites such as insecam (http://www.insecam.org/en/) expose examples of these insecurities, where streams are provided from online insecure cameras, where users can watch live camera feeds from different parts of the world.

Tackling the IoT security threat

The author in [26] highlights the lack of a standard for IoT to have security integrated as a compulsory part of the system as one of the reasons why manufacturers do not invest in this feature when producing smart devices. Suggested alternatives to the current system include the use of encryption, with one example being to replace the widely used Telnet with SSH, which, despite also having security vulnerabilities[4] [27], provides a secure communication session channel that includes supporting RSA authentication and encryption of authentication. Protocols such as Telnet and FTP on the other hand, openly transmit passwords without encrypting these over the network, instead displaying them as cleartext to anyone listening on the network.

To tackle the Mirai threat specifically, Cao et al in [32] describe a method to use Mirai as a virus expeller, by changing the code structure of the virus, eliminating attack functions and implementing a heart-beat module which alerts a server as to whether the virus expeller is live. This version of the malware, proposed as a "white Mirai", depends on a time frame agreed with the device user to disconnect the device and have the virus expeller be installed as soon as the device is back on. However, this still depends on the user actively assisting in removing the threat from the infected device, rather than having a more "passive" approach for users that are unaware their respective IoT may have been infected.

The alternative to the approach proposed by Cao et al. was to push the modified malware without user consent, which results in a violation of user privacy. This premise puts forward an entirely different debate on the difficulty of patching devices that have been hijacked by Mirai-type viruses – if users are unaware of their devices being infected in the first place, and will likely remain unaware if a fix is deployed to the device, the ethical concerns of an external party accessing a private device become a point of debate, even if the outcome of the security patch is aimed at protecting the users privacy while infringing on user privacy. With this consideration in mind, its best to design IoT systems that comply to best security practices and require authentication mechanisms with the original implementation, rather than prioritising ease of connectivity while sacrificing security, which is the trade made by UPnP devices to date.

To apply better security practices, we need to consider how MQTT communicates over the network. MQTT was designed without integrated authentication mechanisms since the implementation of these would not allow for the protocol to be as lightweight as intended. Authentication mechanisms are supported but these need to be implemented on top of the protocol [28]. As for our broker, in Mosquitto the default configuration does not use any form of authentication, keeping the port 1883 open and listening for connections [27], but there are multiple options available to add an extra layer of security: It is possible to encrypt the connection between the MQTT broker, in this case Mosquitto, and the MQTT client. For this, one option is to setup a trusted server certificate on the Broker and use a service such as Certbot (https://certbot.eff.org/about/) or OpenSSL to generate the necessary security certificates.

[4] G.Schultz highlights that particularly OpenSSH has a vast amount of vulnerabilities.

After securing the connection with a trusted certificate we need to setup a password by editing the configuration file inside /etc/mosquitto/conf.d/default.conf to disable connections that have not been authenticated (anonymous connections) and require a password file:

```
allow_anonymous false
password_file /etc/mosquitto/passwd
```

The final step to securing the broker is to configure the SSL certificates and change the default port that MQTT is listening. An example of a configuration for this is suggested in Figure 13, retrieved from [37].

/etc/mosquitto/conf.d/default.conf

```
. . .
listener 1883 localhost

listener 8883
certfile /etc/letsencrypt/live/mqtt.example.com/cert.pem
cafile /etc/letsencrypt/live/mqtt.example.com/chain.pem
keyfile /etc/letsencrypt/live/mqtt.example.com/privkey.pem
```

Figure 26. Configuration of mosquitto broker to point to certificates and encrypt connection

Port 1883 localhost is replaced with a listener on port 8883. The displayed config also displays where the certificates to encrypt connections will be found.

It is thus straightforward to encrypt MQTT connections and modify the native passwords used by the broker. As for the raspberry pi altering the default password as soon as the connection is established for the first time was the best approach to safeguard from Mirai infections on connection.

Finally, considering related work on this area and efforts towards a more secure IoT, a security framework aimed specifically at MQTT is proposed in [29] where the authors implement a mechanism that uses asymmetric key encryption algorithms to sign root certificates with a public key and decrypt these with a private key, to allow for clients to subscribe to a given topic only if these possess the correct private key necessary to decrypt the messages sent to that topic. The authors test both RSA and Elliptic Curve algorithms with feasible results. Amazon IoT also contains implementations that communicate over MQTT and it provides the option to encrypt traffic using Transport Layer Security.

Conclusion

This work was motivated by the recent propagation of IoT dedicated viruses, and for the need of further transparency in the mapping and management of intelligent devices. We have thus discussed services that can act as a starting point for a full IoT hub, by allowing for the exposure of different types of devices, whether these are connected via IP or Bluetooth, by allowing the user to discover these via a simple, user-friendly interface.

Ideally future work would involve adapting the MQTT service to push topics to devices that can support a version of the MQTT client. However, this work also realised that due to the variations in the software of each thing, it is necessary to build specific communication modules that can initiate connections to each “thing”. MQTT however, can provide an abstraction layer on top of which functionality can be further developed to indeed establish connections to each IoT on a given LAN.

During the project development, we further explored different approaches to interconnectivity and security, as well as the intrinsic difficulties in securing the hardware of constrained devices. Conflicts in the establishment of a single standard that can gain the consensus of IoT device manufacturers, along with design flaws such as not allowing users to modify default passwords on older devices or designs that overlook interconnectivity security by keeping ports open to incoming unauthenticated connections, these have all contributed towards turning internet-enabled devices into malware-enabled devices.

On the future of IoT security, the subject is ever more pertinent with the growth of IoT device sales and the success of commercial giants such as Amazon in accelerating the deployment of new IoT solutions to the market. It is possible that the commercial success of Amazon will facilitate its role in advocating for best security practices when establishing IoT connections, and so its standards may resonate more quickly with the industry than the standards advocated by the IEEE or the Wi-Fi Alliance – albeit such standard may never truly materialise. As so, rather than waiting for a standard or a single authority to lead security efforts, the best approach to enable security is centralising the management of our IoT devices through an IoT HuB: using an easy interface and an application that can provide for an abstraction layer to all devices and encrypt connections to IoT. This would be the solution to avoid interference and malware attacks from the outside world, thus striving for a better, more secure future for the Internet of Things.

References

1. de Melo Silva, C.C., Ferreira, H.G.C., de Sousa Júnior, R.T. et al. Wireless Pers Commun [internet] 2016. 91: 1711. doi:10.1007/s11277-015-3168-6. Available from http://link.springer.com.ezproxy.library.qmul.ac.uk/article/10.1007%2Fs11277-015-3168-6 [Accessed 4th May 2017]
2. S.Tomovic , K. Yoshigoe, I. Maljevic, et al. Software-Defined Fog Network Architecture for IoT. Wireless Pers Commun [internet] [2017] 92: 181. doi:10.1007/s11277-016-3845-0 Available from https://link-springer-com.ezproxy.library.qmul.ac.uk/article/10.1007%2Fs11277-016-3845-0 [Accessed 14th July 2017]
3. R. Meulen. Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016 [internet] [2017]. Available from http://www.gartner.com/newsroom/id/3598917 [Accessed 4th May 2017]
4. Cisco [2011] [internet] [cited 2017 April 8th] Available from http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf [Accessed 4th May 2017]
5. M. Reynolds. TalkTalk and Post Office customers hit by Mirai worm attack 29 Nov 2016. Available from http://www.wired.co.uk/article/deutsche-telekom-cyber-attack-mirai

6. K. Batool and M. A. Niazi, Modeling the internet of things: a hybrid modeling approach using complex networks and agent-based models, Complex Adaptive Systems Modeling [2017] 5:4 DOI: 10.1186/s40294-017-0043-1 Published: 24 March 2017 [Internet] Available from https://casmodeling.springeropen.com/articles/10.1186/s40294-017-0043-1
7. L. F. Rahman, T. Ozcelebi , J. J. Lukkien. Choosing Your IoT Programming Framework: Architectural Aspects. Future Internet of Things and Cloud (FiCloud), 2016 IEEE 4th International Conference [2016] DOI: 10.1109/FiCloud.2016.49 [internet] Available from http://ieeexplore.ieee.org.ezproxy.library.qmul.ac.uk/document/7575877/
8. Wi-Fi Alliance. Discover Wi-Fi. Wi-Fi Direct.[2017] Available from http://www.wi-fi.org/discover-wi-fi/wi-fi-direct
9. C. Bormann, A. P. Castellani and Z. Shelby, "CoAP: An Application Protocol for Billions of Tiny Internet Nodes," in IEEE Internet Computing, vol. 16, no. 2, pp. 62-67, March-April 2012. doi: 10.1109/MIC.2012.29. Available from: http://ieeexplore.ieee.org.ezproxy.library.qmul.ac.uk/stamp/stamp.jsp?tp=&arnumber=6159216&isnumber=6159208
10. Goland et al. Simple Service Discovery Protocol/1.0 Operating without an Arbiter <draft-cai-ssdp-v1-03.txt>. Internet Engineering Task Force [internet] Available from https://tools.ietf.org/html/draft-cai-ssdp-v1-03
11. AWS IoT Developer Guide [online] Available from http://docs.aws.amazon.com/iot/latest/developerguide/protocols.html
12. R. A. Light. Mosquitto: server and client implementation of the MQTT protocol. The Journal of Open Source Software, vol. 2, no. 13, May 2017, DOI: 10.21105/joss.00265 Available from http://dx.doi.org/10.21105/joss.00265
13. H.C. Hwang, J. Park, & J.G. Shon. Design and Implementation of a Reliable Message Transmission System Based on MQTT Protocol in IoT. Wireless Pers Commun (2016) 91: 1765. doi:10.1007/s11277-016-3398-2[Internet]Available from https://link-springer-com.ezproxy.library.qmul.ac.uk/article/10.1007%2Fs11277-016-3398-2
14. CoAP overview. Available from http://coap.technology/
15. MQTT OASIS specification, Available from http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html [Accessed 4th May 2017]
16. M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi, Long-Range Communications in Unlicensed Bands: the Rising Stars in the IoT and Smart City Scenarios. IEEE Wireless Communications, Vol. 23, Oct. [2016] [Internet] Available from https://arxiv.org/pdf/1510.00620.pdf [Accessed 4th May 2017]
17. Ş. Arseni, M. Miţoi, A. Vulpe. Pass-IoT: A platform for studying security, privacy and trust in IoT [2016] International Conference on Communications (COMM), DOI:10.1109/ICComm.2016.7528258 Available from http://ieeexplore.ieee.org.ezproxy.library.qmul.ac.uk/document/7528258/ [Accessed 4th May 2017]
18. M. B. Chung, H. Choo. Near wireless-control technology between smart devices using inaudible high-frequencies. H. Multimed Tools Appl [2015] 74: 5955. doi:10.1007/s11042-014-1901-x Available from https://link-springer-com.ezproxy.library.qmul.ac.uk/article/10.1007%2Fs11042-014-1901-x [Accessed 4th May 2017]
19. S. M.A. Oteafy , H. S. Hassanein Resilient IoT Architectures Over Dynamic Sensor Networks With Adaptive Components DOI: 10.1109/JIOT.2016.2621998Publisher: IEEE

[201] Available from http://ieeexplore.ieee.org.ezproxy.library.qmul.ac.uk/document/7707340/?reload=true

20. R.Lea, M. Blackstock. Smart Citites: an IoT-centric Approach Proceeding IWWISS '14 Proceedings of the 2014 International Workshop on Web Intelligence and Smart Sensing [2014] Available from http://dl.acm.org.ezproxy.library.qmul.ac.uk/citation.cfm?id=2637096 [Accessed 4th May 2017]
21. D.Happ, N.Karowski, T.Menzel, V.Handziski, A. Wolisz. Meeting IoT platform requirements with open pub/sub solutions. Ann. Telecommun. [2017] 72: 41. doi:10.1007/s12243-016-0537-4. Available from https://link-springer-com.ezproxy.library.qmul.ac.uk/article/10.1007%2Fs12243-016-0537-4 [Accessed 4th May 2017]
22. Home assistant. Version tested: 0.48.0. Released: July 02, 2017. Available from https://home-assistant.io/ [Accessed 7th July 2017]
23. Port Discovery Android Application, open source repository. Available from https://github.com/aaronjwood/PortAuthority/ [Accessed 16th June 2017]
24. Android developer NSD Manager reference documentation. Available from https://developer.android.com/reference/android/net/nsd/NsdManager.html#discoverServices(java.lang.String, int, android.net.nsd.NsdManager.DiscoveryListener) [Accessed 16th June 2017]
25. C. Karasiewicz *Why Facebook is using MQTT on mobile* IBM blog. Available from https://www.ibm.com/developerworks/community/blogs/mobileblog/entry/why_facebook_is_using_mqtt_on_mobile?lang=en [Accessed 8th July 2017]
26. HiveMQ. *MQTT Essentials* Available from http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment; HiveMQ *MQTT over Websockets* Available from http://www.hivemq.com/blog/mqtt-over-websockets-with-hivemq [Accessed 12th July 2017]
27. M. Murphy. The Internet of Things and the threat it poses to DNS. Network Security Volume 2017, Issue 7, July 2017, Pages 17–19 [online] 19 July 2017. Available from https://doi.org/10.1016/S1353-4858(17)30072-7 [Accessed 30th July 2017]
28. G. Schultz. Using ssh: Do security risks outweigh the benefits? Network Security Volume Issue 10, October [2004], Pages 7-10. https://doi.org/10.1016/S1353-4858(04)00143-6 Available from http://www.sciencedirect.com.ezproxy.library.qmul.ac.uk/science/article/pii/S1353485804001436?_rdoc=1&_fmt=high&_origin=gateway&_docanchor=&md5=b8429449ccfc9c30159a5f9aeaa92ffb [Accessed 27th July 2017]
29. C. Lesjak et al. Securing smart maintenance services: Hardware-security and TLS for MQTT Conference Proceedings: Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on Securing smart maintenance services: Hardware-security and TLS for MQTT [Accessed 28th July 2017]
30. A. Mektoubi et al. New approach for securing communication over MQTT protocol A comparison between RSA and Elliptic Curve. Published in: Systems of Collaboration (SysCo), International Conference on [2016]. DOI: 10.1109/SYSCO.2016.7831326 . Publisher: IEEE Available from http://ieeexplore.ieee.org.ezproxy.library.qmul.ac.uk/document/7831326/ [Accessed 7th July 2017]

31. R. Graham. *Robert Graham from Errata Security conducts Camera Experience and details the infection process on twitter*. Available from https://twitter.com/ErrataRob/status/799556482719162368 [Accessed 28th July 2017]
32. (used to be 47) Cao et. Al. Hey, you, keep away from my device: remotely implanting a virus expeller to defeat Mirai on IoT devices. Report Number PSU-S2-TR-2017-04001, arXiv:1706.05779 [cs.CR] June 2017[online] [2017] Available from https://arxiv.org/pdf/1706.05779.pdf [Accessed 26th July 2017]
33. 48 Y. M. Pa Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, C. Rossow. IoTPOT: Analysing the Rise of IoT Compromises. 9th USENIX Workshop on Offensive Technologies. USENIX Association, 2015 [online] Available from https://www.usenix.org/system/files/conference/woot15/woot15-paper-pa.pdf [Accessed 14th May 2017]
34. McAfee Labs Threats Report April 2017 [2017] [online] Available from https://www.mcafee.com/us/resources/reports/rp-quarterly-threats-mar-2017.pdf [Accessed 12th May 2017]
35. D. Goodin. *Newly discovered router flaw being hammered by in-the-wild attacks Researchers detect barrage of exploits targeting potentially millions of devices.* Published in Ars Technica 28 November 2016 [online] Available from https://arstechnica.com/information-technology/2016/11/notorious-iot-botnets-weaponize-new-flaw-found-in-millions-of-home-routers/ [Accessed 23rd June 2017]
36. European Union Agency for Network and Information Security. *"Mirai" malware, attacks Home Routers*. Published December 14, 2016. Available from https://www.enisa.europa.eu/publications/info-notes/mirai-malware-attacks-home-routers[Accessed 25th June 2017]
37. B. Boucheron. How to Install and Secure the Mosquitto MQTT Messaging Broker on Ubuntu 16.04. December 9, 2016[online]. Available from https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-the-mosquitto-mqtt-messaging-broker-on-ubuntu-16-04 [Accessed 25th June 2017]
38. Eclipse Paho Libraries. Available from: https://eclipse.org/paho/clients/java/ [Accessed 25th June 2017]