2020

IoT-MQTT based denial of service attack modelling and detection

Naeem Firdous Syed
Edith Cowan University

Edith Cowan University

Copyright Warning

IoT-MQTT based denial of service attack modelling and detection

This thesis is presented for the degree of
Doctor of Philosophy

Naeem Firdous Syed

Edith Cowan University
School of Science
2020

Abstract

Internet of Things (IoT) is poised to transform the quality of life and provide new business opportunities with its wide range of applications. However, the benefits of this emerging paradigm are coupled with serious cyber security issues. The lack of strong cyber security measures in protecting IoT systems can result in cyber attacks targeting all the layers of IoT architecture which includes the IoT devices, the IoT communication protocols and the services accessing the IoT data. Various IoT malware such as Mirai, BASHLITE and BrickBot show an already rising IoT device based attacks as well as the usage of infected IoT devices to launch other cyber attacks. However, as sustained IoT deployment and functionality are heavily reliant on the use of effective data communication protocols, the attacks on other layers of IoT architecture are anticipated to increase. In the IoT landscape, the publish/-subscribe based Message Queuing Telemetry Transport (MQTT) protocol is widely popular. Hence, cyber security threats against the MQTT protocol are projected to rise at par with its increasing use by IoT manufacturers. In particular, the Internet exposed MQTT brokers are vulnerable to protocol-based Application Layer Denial of Service (DoS) attacks, which have been known to cause wide spread service disruptions in legacy systems. In this thesis, we propose Application Layer based DoS attacks that target the authentication and authorisation mechanism of the the MQTT protocol. In addition, we also propose an MQTT protocol attack detection framework based on machine learning. Through extensive experiments, we demonstrate the impact of authentication and authorisation DoS attacks on three open-source MQTT brokers. Based on the proposed DoS attack scenarios, an IoT-MQTT attack dataset was generated to evaluate the effectiveness of the proposed framework to detect these malicious attacks. The DoS attack evaluation results obtained indicate that such attacks can overwhelm the MQTT brokers resources even when legitimate access to it was denied and resources were restricted. The evaluations also indicate that the proposed DoS attack scenarios can significantly increase the MQTT message delay, especially in QoS2 messages causing heavy tail latencies. In addition, the proposed MQTT features showed high attack detection accuracy compared to simply using TCP based features to detect MQTT based attacks. It was also observed that the protocol field size and length based features drastically reduced the false positive rates and hence, are suitable for detecting IoT based attacks.

Copyright and access declaration

I certify that this thesis does not, to the best of my knowledge and belief:

(i) incorporate without acknowledgement any material previously submitted for a degree or diploma in any institution of higher education;

(ii) contain any material previously published or written by another person except where due reference is made in the text of this thesis;

(iii) contain any defamatory material;

.

Signed: 
Dated: 21-01-2020

Acknowledgements

Verily all praise is due to Allah (s.w.t). I praise Him; I seek His help and forgiveness. Peace and blessings be upon Prophet Muhammad, the last of Allah's Messengers and Prophets.

I would like to express my sincere gratitude to my thesis advisors Dr Ahmed Ibrahim, Dr Zubair Baig and Professor Craig Valli for their guidance, support and invaluable advice through-out this thesis. I thank you all for being my guides during this memorable research journey. I acknowledge the support and facilities provided by Edith Cowan University (ECU), especially the ECU Security Research Institute (SRI) for providing the opportunity to conduct this research.

I would like to express my deepest gratitude to my parents, my wife Samira, children (Amina and Hamza) and all my family members for their love, prayers, constant support and encouragement. I also thank all my fellow graduate students and all my friends who made this an enjoyable experience.

I dedicate this work to my father and mother who showered their love on me, supported and encouraged me always. To my wife who stood besides me and supported, encouraged me and was patient with me in this PhD journey and to my beloved children.

Publications arising from this research

The following are the publications arising as a result of the research conducted in this thesis.

- Firdous, S. N., Baig, Z., Valli, C., & Ibrahim, A. (2017, June). Modelling and evaluation of malicious attacks against the iot mqtt protocol. In 2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) (pp. 748-755). IEEE.

- Baig, Z. A., Sanguanpong, S., Firdous, S. N., Nguyen, T. G., & So-In, C. (2020). Averaged dependence estimators for DoS attack detection in IoT networks. Future Generation Computer Systems, 102, 198-209.

- Malik, M. I., McAteer, I. N., Hannay, P., Firdous, S. N., & Baig, Z. (2018). XMPP architecture and security challenges in an IoT ecosystem. In Proceedings of the 16th Australian Information Security Management Conference (p. 62).

Contents

List of Figures

List of Tables

Acronyms

IoT : Internet of Things

OASIS : Organisation for the Advancement of Structured Information Standards

IP : Internet Protocol

IPv4 : Internet Protocol version 4

IPv6 : Internet Protocol version 6

TCP : Transmission Control Protocol

UDP : User Datagram Protocol

MQTT : Message Queuing Telemetry Transport

HTTP : Hypertext Transport Protocol

FTP : File Transfer Protocol

SNMP : Simple Network Management Protocol

ARP : Address Resolution Protocol

ICMP : Internet Control Message Protocol

IGMP : Internet Group Management Protocol

GSM : Global System for Mobile Communications

DNS : Domain Name System

MTU : Maximum Transmission Unit

OS : Operating System

ML : Machine Learning

AI : Artificial Intelligence

AODE : Average One Dependence Estimator

C4.5 : Decision Tree based Classifier

MLP : Multi-Layer Perceptron

AI : Artificial Intelligence

DoS : Denial of Service

LAN : Local Area Network

CDF : Cumulative Distribution Function

CSV : Comma Separated Values

IEEE : Institute of Electrical and Electronics Engineers

RFID : Radio-frequency identification

RFC : Request for Proposal

LTE :Long-Term Evolution

CPU : Central Processing Unit

SCADA : Supervisory control and data acquisition

DNP3 : Distributed Network Protocol 3

SDN : Software Defined Networks

VPN : Virtual Private Network

Chapter 1

Introduction

1.1 Background

The evolution of multiple technologies comprising of sensor networks, cloud computing, wireless communication and low power electronics during the past decade, has made an Internet of connected devices a possibility. Approximately 75 billion devices are projected to be connected to the Internet by the year 2025 (Statista, 2016). Internet of Things (IoT) is changing the way in which computing devices and information are being used in the consumer space and the industry to foster the development of a smarter world. In a consumer environment, IoT devices typically comprise smart sensors, smart refrigerators, smart TVs, IP cameras, health trackers, smart home automation systems and others (Miller, 2015; Libelium, 2019). IoT is also playing a crucial role across the industries that include automotive, energy, healthcare, manufacturing and retail, to help improve the business processes by automating the various processes and systems (Gubbi, Buyya, Marusic, & Palaniswami, 2013; Libelium, 2019). The role of IoT in building smart future cities is indispensable and is already playing an important role in transforming the energy, transportation and communications sectors (Zanella, Bui, Castellani, Vangelista, & Zorzi, 2014). However, this has also led to the emergence of pathways for criminals to exploit these IoT systems. Furthermore, the increasing role of IoT devices in various domains is exposing their vulnerabilities to adversarial cyber threats (Bekara, 2014; Neshenko, Bou-Harb, Crichigno, Kaddoum, & Ghani, 2019).

Vulnerabilities of an IoT system emerge from its components including but not limited to sensor networks, wireless networks and the Internet (Andrea, Chrysostomou, & Hadjichristofi, 2015). Such vulnerabilities can

result in a range of cyber security threats from attacks to physical device, communication protocols and the services that access the IoT device data.

With IoT devices being the basic building block of an IoT system coupled with various device level vulnerabilities have made such devices an attractive target for adversaries. The most notable reasons for their security exploitation being (Heer et al., 2011; Li, Da Xu, & Zhao, 2015; Sadeghi, Wachsmann, & Waidner, 2015; Neshenko et al., 2019):

- their connectivity to the Internet,
- their capability of being remotely controlled,
- their lack extensive security mechanisms, and
- their ability to generate, exchange and consume potentially sensitive and safety-critical data.

Furthermore, various challenges exist in implementing security solutions in the IoT devices, such as resource constraints (limited CPU, Memory, Storage, Power), vendor competition in time-to-market IoT devices and solutions, lack of standard IoT security best practices, and heterogeneity of IoT devices and protocols (Heer et al., 2011; Al-Fuqaha, Guizani, Mohammadi, Aledhari, & Ayyash, 2015; Z. Zhang et al., 2014; Neshenko et al., 2019).

These cyber security challenges and vulnerabilities in the IoT devices has resulted in various cyber-attacks that have affected IoT operations in the recent past. In a research conducted by Flash Point Analysts in collaboration with L3 Threat research lab (FPAnalyst, 2016), it was found that malware targeting IoT devices like BASHLITE were compromising vulnerable IoT devices and using them for launching Distributed Denial of Service (DDoS) attacks. Some of these bots were reported to be comprising in excess of one million compromised devices; most of which being IoT devices. Mirai, a potent IoT device malware, infected millions of IoT devices by brute-forcing the Telnet ports to gain unauthorised device access (Kolias, Kambourakis, Stavrou, & Voas, 2017). Similarly, an IoT device-damaging attack, namely, 'BrickBot' attempted to help the adversary gain unauthorised access to vulnerable IoT devices and cause permanent damage to the compromised devices by corrupting the device storage (Olenick, 2017).

Attacks directly targeting the IoT devices can result in device damage, compromise of control or loss of information stored on them. The direct impact of such attacks on IoT devices is that, compromised devices can be

exploited to perpetrate other cyber-attacks such as DDoS attacks (Neshenko et al., 2019). Reports such as FPAnalyst (2016) and Krebs (2016a), indicate that infected IoT devices are being used for large-scale DDoS attacks generating around 650 Gbps of traffic. More recent DoS attacks, such as 1.35 Tbps DDoS attack on GitHub (Chadd, 2018), and Mirai botnet based DoS attack on Dyn DNS services (Mansfield-Devine, 2016), highlight the increasing issues associated with vulnerable IoT devices.

Many of the attacks discussed above are those that target vulnerable IoT devices exposed to the Internet. However in most IoT deployments, the IoT devices might not be directly exposed to the Internet, rather will have the ability to exchange messages through it (Waher, 2015). In such scenarios, the IoT protocols, gateways and the middle-ware platforms that facilitate message exchange will be prone to various cyber-attacks that target IoT deployments. Hence, the vulnerabilities in an IoT system can also be exploited through various IoT protocols and various other technologies used in their deployment. These vulnerabilities are exacerbated with the lack of built-in security measures in IoT Application Layer Protocols, which limits the implementation of end-to-end security in order to protect data transferred though the middle-ware or IoT gateways (Alliance, 2019; Al-Fuqaha et al., 2015). The dependence on middle-ware or gateways emerges from the device-to-device (D2D) and device-to-cloud (D2C) communication models (Meng, Wu, Muvianto, & Gray, 2017) widely adopted in IoT communications. Such middle-wares or gateways facilitate the inter-IoT device and device-to-server message exchange required for running the business logic and analytics.

A myriad of IoT protocols have been proposed to facilitate D2D and D2C communication. However, according to a recent developer survey reported in (eclipse.org, 2018), MQTT has been rated as the most popular IoT protocol for applications that range across various domains comprising industrial automation, smart cities and healthcare. MQTT has been adopted in various real-world critical applications; a few examples are:

- Deutsche Bahn AG (DB) German Railway network's use of MQTT for exchanging real-time information about location, delay, and diagnostic checks to notify delays, cancellations, or platform changes (eclipse.org, 2018).

- NEXCOMM's (Wu, 2017) use of the protocol in industry process automation, analytics and reporting.

- Eurotech's MQTT based IoT Edge Framework (Eurotech, 2019) which

is enabling the Industry 4.0 adoption across Europe.

These real-world examples show the growing reliance on the MQTT protocol in critical infrastructures and Industry 4.0. In addition, all the leading cloud solution providers such as Amazon AWS, Microsoft Azure and Google Cloud provide off-the-shelf MQTT support for IoT deployments. The increased adoption of the MQTT protocol is expected to attract adversaries to attack MQTT-enabled IoT systems, especially DoS attacks, that are expected to increase as adversaries gain access to millions of infected IoT devices (Peraković, Periša, & Cvitić, 2015; Chadd, 2018; Metongnon & Sadre, 2018).

Most of the work, as found in the IoT literature, either present the various DoS attack detection techniques (Kasinathan, Pastrone, Spirito, & Vinkovits, 2013; Alanazi et al., 2015) or merely refer to the DoS attack as one of the challenges to the IoT system (Roman, Zhou, & Lopez, 2013; Borgohain, Kumar, & Sanyal, 2015; Shaker & Zarrabi, 2017). Furthermore, the MQTT OASIS specification (global consortium that works on development and adoption of open standards for security, IoT and various other domains) (Cohn, Coppen, Banks, & Gupta, 2014) lists DoS attacks as one of the security threats to the MQTT protocol.

MQTT is a publish/subscribe based Application Layer protocol, where a central message broker routes the published messages to subscribers based on their topics of interest. The message broker plays a very important role in MQTT as it decouples the publishers and subscribers in both space and time. However, an Internet exposed message broker can provide an attack surface for the IoT system, as it can become the target of Application Layer DoS attacks (Firdous, Baig, Valli, & Ibrahim, 2017; Metongnon & Sadre, 2018). Especially, the rise of DoS attacks due to an increase in the number of comprised IoT devices poses as a major challenge to MQTT brokers, threatening the industrial or critical IoT applications relying on them.

The first step in protecting an IoT-MQTT system will be to detect such attacks. The existing detection techniques that focus on Network Layer features and metrics will be ineffective as Application Layer DoS attacks may contain legitimate requests and can therefore remain undetected (Praseed & Thilagam, 2018; Xiao, Yun, & Zhang, 2010). In addition, Application Layer DoS attack detection techniques require protocol specific or domain specific features to detect attacks, which cannot be extended to all Application Layer protocols. Hence, novel MQTT protocol-based features are essential to detect MQTT based Application Layer DoS attacks. The major hurdle in developing effective detection systems for IoT is the lack of real-world

datasets. Therefore, existing attack detection techniques use either simulated or non-IoT datasets to detect known attacks (Diro & Chilamkurti, 2018; Haddad Pajouh, Javidan, Khayami, Ali, & Choo, 2018; Moustafa, Turnbull, & Choo, 2019).

These challenges in protecting the IoT systems are the main motivations behind the choice of research topic for this thesis. The need to identify the vulnerabilities of IoT-MQTT protocol to the Application Layer DoS attacks and a machine learning based framework to detect such attacks are the main focus of this research.

1.2 Purpose of this Research

This research aims to contribute to the body of knowledge with regards to gaps found in detecting IoT-MQTT based DoS attacks as found in IoT deployments. The literature review presented in Chapter 2, identified the lack of MQTT based features to detect targeted Application Layer DoS attacks. The domain and protocol specific features presented by various studies cannot be extended to the MQTT protocol. Hence, new features that can effectively detect attacks against the MQTT protocol are required. In addition, DoS detection evaluation datasets such as KDD99 (UCI, 1999), DARPA99 (MIT, 1999), CAIDA DDoS (UCSD, 2007) are either old or lack IoT or MQTT related traffic and are no longer relevant to validate the techniques that are built to detect attacks in IoT traffic. Therefore, novel normal and attack traffic datasets were collected with relevant MQTT protocol specific attack traffic, so as to model an effective attack detection technique.

The significant contributions of this study are:

- Identifying threats to the IoT-MQTT protocol by presenting an MQTT threat model.
- Devising and evaluating authentication and authorisation based Application Layer DoS attacks against the MQTT protocol.
- Generating and collecting IoT-MQTT based attack dataset based on realistic IoT deployment.
- Proposing MQTT based time-window and flow-based features to effectively detect MQTT attacks.
- Evaluating the attack detection framework for its effectiveness in detecting MQTT based attacks.

1.3 Research Questions

The main research question to be addressed through this research is:

> ***RQ1****:"How can Application Layer based DoS attacks against the IoT-MQTT protocol be detected?"*. This can be answered by posing three sub-question:
>
> - **SQ1**: Is the MQTT protocol vulnerable to Application Layer DoS attacks?
> - **SQ2**: Are the MQTT protocol based features required to detect targeted DoS attacks against MQTT-IoT systems?
> - **SQ3**: How effective are the developed machine learning models in correlating between normal and attack traffic?

1.4 Thesis Structure

This thesis is organised into seven chapters, which present the background of this research, the methodology adopted, the attack models, detection framework, experimental results and discussion of the research conducted. Chapter 2 provides the background concepts, literature and the identified gaps relevant to the research undertaken. Subsequently, Chapter 3 examines the various research methodologies available in the body of literature and justifies the methodology chosen by the author in conducting this research. It also provides the research design and the processes followed during the course of this research.

Chapter 4 presents a detailed description of attack models and the proposed attack detection framework implemented in this research to conduct the experimental evaluations. This is followed by Chapter 5, which provides detailed empirical results obtained through the execution of numerous experiments for the collection of empirical evidences to answer the research questions.

Chapter 6 critically examines the observed empirical results and provides an in-depth reasoning and answers to the proposed research sub-questions. Finally, the thesis is concluded (Chapter 7) by presenting the answer to the major research question, the identified contributions of this research, limitations of this study and the possible future research directions.

Chapter 2

Literature Review

This chapter introduces the key concepts and terms referred to throughout this thesis. In addition, existing research work on the topic of IoT and security is reviewed to identify the gaps and the purview under which this research work is proposed. For the literature review, publications that are directly related to this research were investigated. Research publications were considered directly relevant to the study if they were related to the MQTT protocol, MQTT security and MQTT attack detection techniques. However, since this work falls under the broader research category of IoT, publications related to IoT and IoT security were studied and analysed to establish the background concepts.

The literature review is divided into four categories namely: IoT and IoT security, DoS attack techniques, DoS attack detection techniques, and MQTT protocol and security. Section 2.1 discusses the various definitions of IoT as found in the literature to establish the definition that was adopted in this research. Furthermore, the enabling technologies of IoT are discussed with specific emphasis on communication patterns. The issues with IoT security and the associated IoT protocols are also discussed in Sections 2.2 and 2.3. Section 2.4 reviews the literature related to various DoS attack models and scenarios against the Application Layer protocols of both legacy and IoT systems. Section 2.5 reviews the DoS detection techniques relevant to the proposed research. Finally in sections 2.7 and 2.8 the MQTT protocol standard, message formats, message exchange process, and the security issues and vulnerabilities in MQTT protocol are presented.

2.1 Internet of Things (IoT)

Approximately 75 billion devices are projected to be connected to the Internet by the year 2025 (Statista, 2016). Various definitions of IoT have been coined in the literature (Li et al., 2015) however, the exact definition of IoT is still a topic of debate. Hence, it is required to present the precise definition of IoT as implied in the context of this research. In this section, various important definitions used in the literature are first presented and then the definition adopted for this research is presented.

According to Perera, Zaslavsky, Christen, and Georgakopoulos (2014) IoT is a part of the evolution of Internet. The Internet evolved from enabling communication between computers (Olifer & Olifer, 2005), increasingly connecting more people to World Wide Web. Mobile phones and Social media networking connected people to the Internet, and currently devices surrounding us are being connected to the Internet forming the Internet of Things (Perera et al., 2014). Similarly, Serbanati, Medaglia, and Ceipidor (2011) identified two stand-out definitions for the term IoT. First, an IoT network is the extension of the existing Internet to new types of devices. Second, the IoT paradigm is the vision of connecting physical and digital worlds in an augmented space to facilitate users (humans or physical devices) to cooperate and satisfy their goals. Furthermore, Rose, Eldridge, and Chapin (2015) propose IoT to be related to extending the ability to connect and compute to non-computers, allowing these devices to generate, exchange and consume data with minimal human intervention.

However, the European research cluster on the Internet of things (IERC) considers these devices to be smart, self-configurable devices in their definition of IoT, which is as follows: *"A dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual 'things' have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network."* (Vermesan & Friess, 2014, p. 15)

In an effort to coin a reasonably encompassing definition of IoT, which includes various features of IoT, the IEEE IoT initiative presented by Minerva, Biru, and Rotondi (2015) collected various definitions of IoT from different sources and categorised the IoT definition into a small environment scenario and a large environment scenario. The definition for a small environment scenario is: *"An IoT is a network that connects uniquely identifiable 'Things' to the Internet. The 'Things' have sensing/actuation and potential programmability capabilities. Through the exploitation of unique identification*

and sensing, information about the 'Thing' can be collected and the state of the 'Thing' can be changed from anywhere, anytime, by anything." (Minerva et al., 2015, p. 73)

In addition, the definition for a large environment scenario is: *"Internet of Things envisions a self-configuring, adaptive, complex network that interconnects 'things' to the Internet through the use of standard communication protocols. The interconnected things have physical or virtual representation in the digital world, sensing/actuation capability, a programmability feature and are uniquely identifiable. The representation contains information including the thing's identity, status, location or any other business, social or privately relevant information. The things offer services, with or without human intervention, through the exploitation of unique identification, data capture and communication, and actuation capability. The service is exploited through the use of intelligent interfaces and is made available anywhere, anytime, and for anything taking security into consideration."* (Minerva et al., 2015, p. 74)

In order to simplify the definitions of IoT, Voas (2016) explained that there is no single definition that defines the IoT at a foundational level and states that the fundamentals of IoT are the ability to compute, communicate, sense and actuate. Similarly, Waher (2015) defined IoT in simple terms as: *"The IoT is what we get when we connect Things, which are not operated by humans, to Internet"* (Waher, 2015, p. 2)

As it is evident from the various definitions presented in this section, a single definition of IoT does not exist. However, the main characteristics of IoT can be extracted from these definitions and summarised as follows:

1. Ability to compute and communicate, generate data, consume data.
2. Ability to connect to the Internet.
3. Capability of sensing and actuation.
4. Programmable.
5. Unique identities and physical attributes.
6. Self-configurable.
7. Use intelligent interfaces.
8. Use standard communication protocols.
9. Work independently with or without human interventions.

10. Can be controlled from anywhere, anytime, and by anything.

From the above definitions, it is clear that IoT has two major parts: Internet and Things. Internet enables the communication between 'Things' and between humans and 'Things' using various communication protocols. On the other hand, 'Things' are uniquely identifiable, independent, programmable devices having sensors and actuators that can share data about their physical environment over the Internet with other devices and humans, who can take actions by controlling 'Things' anytime from anywhere.

2.1.1 IoT Enabling Technologies

The development of IoT has been made possible due to emergence and maturation of multiple Information and Communication Technologies (ICT) (Atzori, Iera, & Morabito, 2010).

The main technologies that contributed to the growth of IoT can be categorised into hardware, communication and application layers as highlighted in the Figure 2.1.

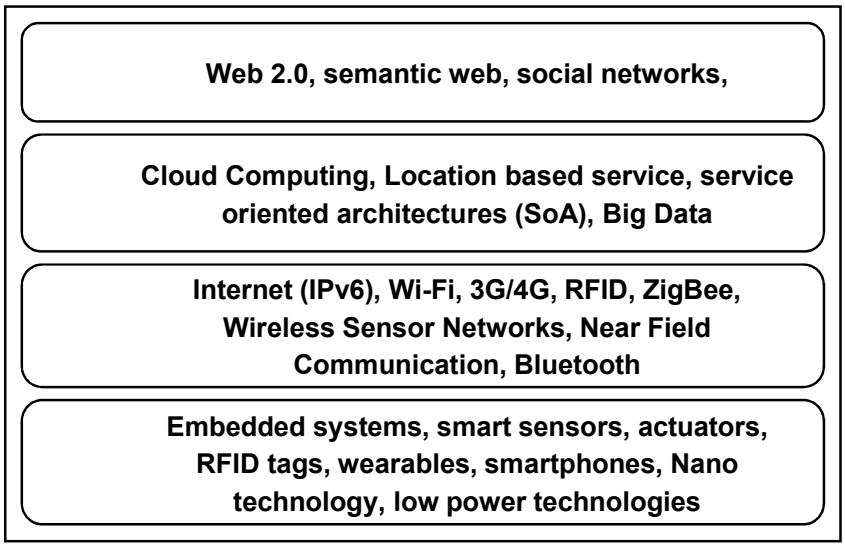


Figure 2.1: Enabling technologies that propelled the growth of IoT which range from hardware layer, communication layer, application layer, adopted from Li et al. (2015) and Al-Fuqaha et al. (2015)

At the Hardware Layer, the evolution of nano technology and low power technologies paved the way for the emergence of IoT. Furthermore, the

growth of wireless sensor networks and low energy wireless communications has extended the capabilities of sensor devices (Li et al., 2015).

Along with the growth of communication technologies, the projection of billions of IoT devices to be connected to the Internet requires a unique network addressing scheme to remotely control such devices. The development and evolution of IPv6 plays a crucial role in providing such a unique identification to IoT devices. In addition, the devices that do not support IP protocol stack can use Uniform Resource Name (URN) identification scheme and can be accessible using a communication gateway (Gubbi et al., 2013).

At the Application Layer, the emergence of Cloud computing and Big Data are key technology enablers for IoT (Al-Fuqaha et al., 2015). The increased amounts of data that will be generated by billions of connected IoT devices needs to be efficiently processed and stored to take full advantage of this highly connected world. Cloud computing along with Big Data technologies offers efficient storage and computation capabilities to extract useful information from unstructured data sources (Al-Fuqaha et al., 2015).

With a wide range of technologies contributing to the growth of IoT, the IoT ecosystem needs to deal with billions of heterogeneous devices communicating with each other. In such a heterogeneous environment, interoperability, scalability and security would be the key challenges that needs to be overcome for a successful IoT deployment. Standardisation efforts are required to reduce the challenges in interoperability. The lack of standardisation is evident from the various IoT architectures adopted based on the application domain (Al-Qaseemi, Almulhim, Almulhim, & Chaudhry, 2016; Al-Fuqaha et al., 2015; Ray, 2018; Shaikh, Zeadally, & Exposito, 2017). However, efforts are being made to propose generic IoT architectures based on the industry requirements (Al-Fuqaha et al., 2015). The most common architectures defined for IoT are the three-layer and the five-layer. The three-layer architecture consists of a Perception Layer, a Network Layer and an Application Layer. The Perception Layer constitutes the physical devices where sensing and actuation occurs. Physical devices or sensors sense the environment and send measurement data to the Network Layer. The Network Layer is responsible in connecting various endpoints and allows data data exchange between the Perception and Application Layers. Wired or Wireless network technologies are leveraged in this layer to support the data exchange between the endpoints. The Application Layer is responsible to process and store the collected data and provide useful functions to the end users.

In contrast, the five-layer architecture consists of addition two additional layers, namely, Access Gateway Layer and Middle-ware Layer (Al-Qaseemi

et al., 2016). The three layers of three-layer architecture server the same purpose as discussed earlier. In addition, an Access Gateway Layer is defined which manages communications between IoT devices by translating between communication protocols usually using a gateway devices. Furthermore, the Middle-War Layer provides additional processing and storage capabilities to the three-layer architecture enabling flexible communication between physical sensors and IoT applications (Al-Qaseemi et al., 2016).

The Perception Layer consists of the physical sensors and actuators that collect and process information. These sensors can collect information like temperature, location, weight, humidity, acceleration etc. The devices store the data locally or in the cloud for further processing. The Network Layer and Middle-ware Layer facilitates device-to-device or device-to-cloud communication (Rose et al., 2015). Finally, the Application Layer and the Business Layer allows for a secure and reliable delivery of services to other devices or to humans.

2.1.2 IoT Communication Models

Many IoT devices use a mix of different communication technologies to connect to other devices and to the Internet. The RFC 7452 defined by the Internet Architecture Board (IAB) (Tschofenig, Arkko, Thaler, & McPherson, 2015) describes the four communication models used by IoT devices as illustrated in Figure 2.2. These communication models are:

1. Device-to-Device Communication.
2. Device-to-Cloud Communication.
3. Device-to-Gateway Communication.
4. Backend Data Sharing pattern.

Device-to-Device (D2D) Communication:

D2D communication model (also referred to as Machine-to-Machine M2M) allows two or more devices to communicate and exchange data directly without any intermediary (Rose et al., 2015). Data exchange can occur on any type of communication network or any protocol. An example of this communication model is between a light bulb and smart switch over a wireless network like Bluetooth, Z-Wave or Zigbee, a common communication model used in home automation systems (Rose et al., 2015). Data Distribution Service is a D2D communication Application Layer protocol used in IoT (Saidu,

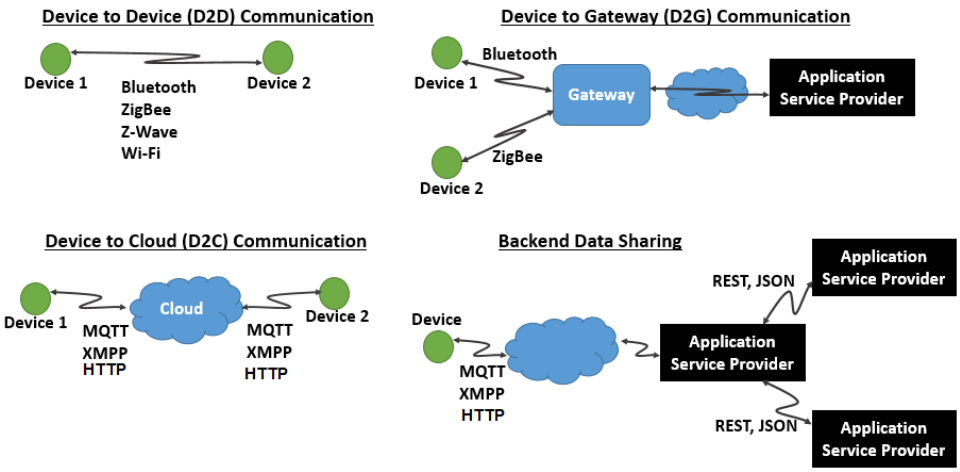


Figure 2.2: Four communication models used in IoT communications to exchange IoT data

Usman, & Ogedebe, 2015). As devices are expected to be the main participants in an IoT system, D2D will be the most preferred communication model for IoT (Bello & Zeadally, 2016).

Device-to-Cloud (D2C) Communication:

D2C communication model (also referred to as Device to Server D2S) allows the IoT device to directly connect to the application services hosted in the cloud (Rose et al., 2015). This model also allows the applications to remotely control the IoT devices. The devices can use traditional IP networks to connect to the cloud services. An example of this communication model is a smart thermostat transmitting data to the cloud where analytics is applied to analyse the energy consumption. The thermostat can also be accessed via a smartphone and remotely controlled by the users (Rose et al., 2015). Many IoT Application Layer protocols such as Message Queueing Telemetry Protocol (MQTT), Extensible Messaging and Presence Protocol (XMPP), Constrained Application Protocol (CoAP) support this communication model (Saidu et al., 2015).

Device-to-Gateway (D2G) Communication:

In this model the IoT devices access cloud services through a local network gateway. The network gateway will include application software that provides security and other functions like data or protocol translations. The devices will connect to the network gateway using protocols such as Zigbee, Bluetooth and Z-Wave. An example of this kind of communication model is

a fitness band connecting to the cloud via a smartphone application (Rose et al., 2015). Gateway devices also help achieve interoperability between devices using different protocols or integrate a smart device with a legacy device (Rose et al., 2015).

Backend Data sharing communication:

This communication model (also referred to as Server-to-Server S2S) allows IoT device data to be shared with authorised third parties (Rose et al., 2015). Many vendors collaborate with each other to build solutions or value added services for end users which requires data to be integrated from multiple devices. In such scenarios, back-end communication needs to be enabled using protocols such as Hypertext Transport Protocol (HTTP) RESTful API (Al-Fuqaha et al., 2015) or Advanced Message Queuing Protocol (AQMP) (Saidu et al., 2015). Web based Application Programming Interface (API) such as RESTful API allow different applications to interact with each other and exchange messages (Masse, 2011). Backend data sharing also allows users to move their device data when switching between different IoT services (Saidu et al., 2015).

The various technologies and communication models used in IoT devices indicate that numerous heterogeneous devices will interact with each other or with humans to automate their daily tasks. Many IoT protocols have emerged to facilitate communication between IoT devices, which are briefly discussed in the following section.

2.2 IoT Protocols

In order to facilitate various types of communication models in IoT, several IoT protocols have been developed or existing protocols have been adopted. Protocols and standards are necessary for message delivery, interoperability and application development. The different categories of IoT protocols (Al-Fuqaha et al., 2015) are illustrated in Figure 2.3.

Application delivery protocols are responsible for data exchange in IoT whereas service discover protocols are required for automatic service and resource discovery in large-scale IoT deployments. In contrast, the infrastructure protocols facilitate the deployment of various IoT topologies such as star, peer-to-peer and cluster-tree. In a myriad of IoT protocols, the choice of communication protocol for deploying an IoT system depends on the application requirements and the functionality offered by the IoT protocols (Al-Fuqaha et al., 2015). Table 2.1 briefly describes the various Application Layer IoT protocols that are used in IoT applications.

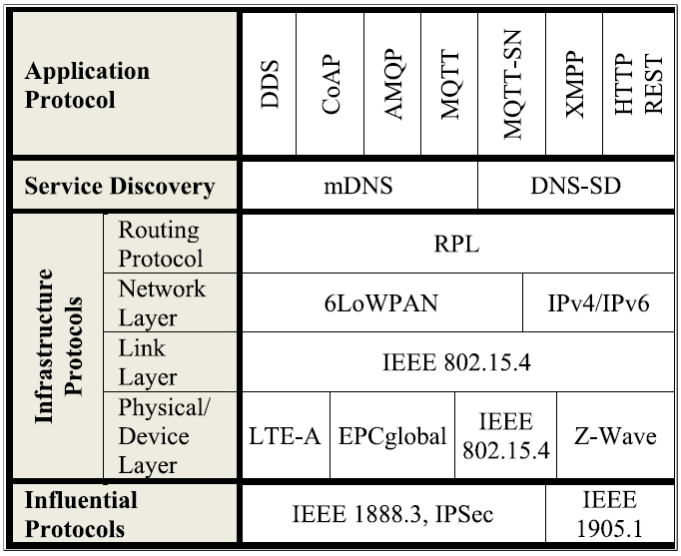


Figure 2.3: IoT protocols stack, categorising protocols based on the usage in IoT deployments. Adopted from Al-Fuqaha et al. (2015, p. 2353)

The various IoT Application Layer protocols adopt two main messaging patterns, namely: Request/Response and Publish/Subscribe (Al-Fuqaha et al., 2015). However, the publish/subscribe messaging pattern has more advantages than the request/response pattern for resource constrained IoT devices. Request/response protocols such as HTTP are typically resource demanding. In addition, the RESTful mode used in CoAP depends on regular updates or polling (Niruntasukrat et al., 2016) which can prove to be resource intensive. Universal Plug and Play (UPnP) is a discovery protocol for IoT devices to discover and connect with other compatible devices. The new version of UPnP protocol released by the UPnP group (UPnP, 2014) uses the XMPP protocol, to connect UPnP devices as XMPP clients. The XMPP protocol has been revised to support both request/response and publish/subscribe message patterns with greater support for IoT deployments. However, there exists security challenges with the use of XMPP protocol in the IoT deployments as highlighted by Malik, McAteer, Hannay, Firdous, and Baig (2018). MQTT a lightweight publish/subscribe protocol originally designed to work in unreliable satellite communications, has been adopted for IoT communication due to its simple messaging format with a small packet header.

According to a recent developer survey reported in (eclipse.org, 2018),

Table 2.1: Application Layer IoT protocols used for data exchange

Protocol	**Description**	**Type/Port Number**
MQTT	Message Queue Telemetry Protocol, used for connecting embedded devices with applications and middle-ware. It is a publish / subscribe protocol providing one-to-one, one-to-many and many-to-many message delivery. It is a lightweight, low bandwidth protocol ideal for resource constrained devices (Al-Fuqaha et al., 2015).	TCP/1883
XMPP	Extensible Messaging and Presence Protocol, defined by IETF for instant messaging, applicable in IoT environments due its simplicity and allows XMPP clients to connect to server using XML streams (Al-Fuqaha et al., 2015).	TCP/5222
DDS	Data Distribution Service is a publish/subscribe protocol developed by Object Management Group. (Salman, 2015). DDS uses a broker less architecture to provides a reliable multicast message delivery service and suitable real time in M2M communications (Al-Fuqaha et al., 2015).	TCP/UDP
AMQP	Advanced Message Queuing Protocol primarily designed for financial sector. In AMQP the broker has two components exchange and message queue (Al-Fuqaha et al., 2015; Salman & Jain, 2015).	TCP/ 5672
CoAP	Constrained Application protocol defined by IETF which provides a lightweight version of REST HTTP to low power devices. CoAP is a UDP based protocol and uses messages like GET, PUT, PUSH and DELETE (Al-Fuqaha et al., 2015; Salman & Jain, 2015).	UDP/ 5683
UPnP	Universal plug and play is a widely used protocol for device interconnections. Since it is a zero configuration networking protocol it is used in consumer devices (Ferreira, Canedo, & de Sousa, 2013).	UDP/1900
HTTP	HTTP is one the enabling technologies for IoT, many application communication is built upon the HTTP. Many consumer applications use HTTP management. REST API used in IoT for exchange of messages with cloud is built over HTTP protocol (Al-Fuqaha et al., 2015).	TCP/80

MQTT has been rated as the most popular IoT protocol for applications that range across various domains comprising industrial automation, smart cities

and healthcare. MQTT has been adopted in real-world critical applications such: as Deutsche Bahn AG (DB) German Railway networks use of MQTT for exchanging real-time information about location, delay, and diagnostic checks to notify delays, cancellations, or platform changes (eclipse.org, 2018), NEXCOMMs (Wu, 2017) use of the protocol for industry process automation, analytics and reporting, Eurotech's MQTT based IoT Edge Framework (Eurotech, 2019) that is enabling Industry 4.0. These examples show the growing reliance on MQTT protocol in critical infrastructures and Industry 4.0. In addition, all the leading cloud solution providers such as Amazon AWS, Azure and Google provide off-the-shelf MQTT support for IoT deployments.

In a ubiquitous and rapidly evolving technology such as IoT, cyber-security breaches can cause a widespread damage to infrastructure and humans utilising them. In the following section security issues that challenge the IoT are explained.

2.3 IoT Security

The emergence of IoT is transforming human lives and is reshaping how business function; providing effective and greater services to end users. However, the rapidly adoption of IoT technology is coupled with serious cyber-security issues due to various challenges and can attract cyber-attacks. The various challenges and threats to IoT and the impact of such vulnerabilities are discussed in this section.

2.3.1 IoT Threats

With the broader use of IoT, there is also a growing concern regarding the security of 'Things' and the IoT systems deployed through them. An IoT system inherits the vulnerabilities of its component parts including but not limited to sensor networks, wireless networks and the Internet (Andrea et al., 2015). Hence security threats to IoT systems can range from physical device based attacks to attacks on communication protocols and IoT tools. Threats to IoT also exist due to many challenges in implementing strong security solutions. Some of the challenges in implementing security solutions for IoT are listed below (Heer et al., 2011; Al-Fuqaha et al., 2015; Z. Zhang et al., 2014; Neshenko et al., 2019):

- Resource constraints (CPU, Memory, Storage, Power) of IoT devices ('Things').

- Vendor competition in quick time to market for devices and solutions.
- Lack of standard IoT security practices.
- Heterogeneous 'Things' and IoT protocols, challenge content protection.

Among the various challenges, resource constraint is one of the key challenge in building security solutions for IoT (Heer et al., 2011). IoT devices have tight resource constraints in terms of the CPU and memory, and often rely on unreliable communication channel, which makes the implementation of security measures difficult (Heer et al., 2011). These challenges can result in exposure of various vulnerabilities in IoT devices such as (Neshenko et al., 2019):

- weak physical device security which can lead to unauthorised access to the device and the data it contains,
- weak authentication mechanisms, resulting in bruteforce attacks,
- insufficient encryption, resulting in MitM attacks during data transmission,
- poor of lack of patch management leaving vulnerabilities unfixed, and
- poor programming practices and auditing causing in buffer-overflow attacks, data and device manipulation attacks.

In addition to device level threats, the vulnerabilities of an IoT system can also arise from the various underlying IoT protocols. Many of the IoT Application Layer protocols lack built-in security measures and depend on existing Transport Layer protocol (TCP and UDP) security mechanisms (Shang, Yu, Droms, & Zhang, 2016; Al-Fuqaha et al., 2015). For example, TCP based protocols run Transport Layer Security (TLS) encryption and some UDP based protocols use Datagram TLS (DTLS) for securing the communication channel between devices (Karagiannis, Chatzimisios, Vazquez-Gallego, & Alonso-Zarate, 2015). However, TLS and DTLS impose high resource utilisation in resource constrained IoT devices (Shang et al., 2016). These challenges in heterogeneous IoT architecture layers can make IoT systems vulnerable to various cyber attacks. According to Jeyanthi (2016), the attacks on the IoT systems can occur during various system operation phases, against dynamic architectures and heterogeneous components. Figure 2.4 summarises the various attacks on each IoT system phase.

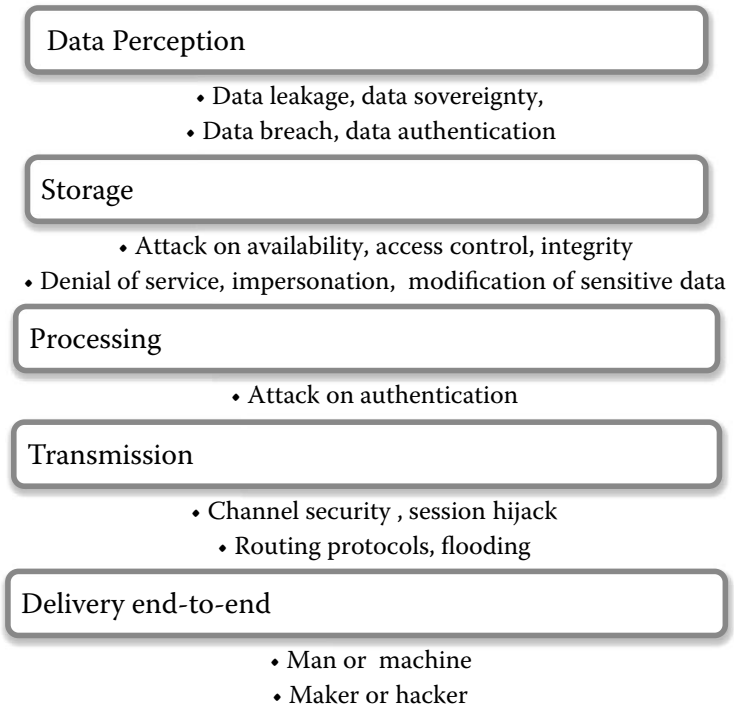


Figure 2.4: IoT attacks categorised based on the IoT system phases. Adopted from Jeyanthi (2016, p. 8)

The attacks on IoT can also be classified based on the various layers of IoT communication which include: Application Layer, Transport Layer, Network Layer and Perception Layer (Jeyanthi, 2016). According to Neshenko et al. (2019) and Hassija et al. (2019), vulnerabilities exist at all levels of IoT architecture such as device level, network levels and application level. Table 2.2 shows the various attacks that can target these IoT architectures.

Security issues in IoT also arise from its wide applications in industries to automate, increase productivity and reduce production costs (Panchal, Khadse, & Mahalle, 2018). Sadeghi et al. (2015) in their work discussed the security and privacy challenges in Industrial IoT (IIoT). According to the authors, IIoT are susceptible to multiple cyber-attacks as there are delays in implementing countermeasures. Figure 2.5 illustrates the attack surface provided by the IIoT systems.

Table 2.2: IoT attacks on various IoT communication architecture layers

IoT Layer	**IoT Attacks**
Sensing or Perception layer	Dictionary attacks, side-channel attacks, Firmware attacks, battery draining attacks, device capture
Network Layer	Routing attacks, impersonation attacks, IP spoofing, data transit attacks
Transport Layer	DoS, DDoS, Man-in-Middle, encryption attacks
Application Layer	Access control attacks, Application Layer DoS, DDoS, Code injection attacks, Data theft

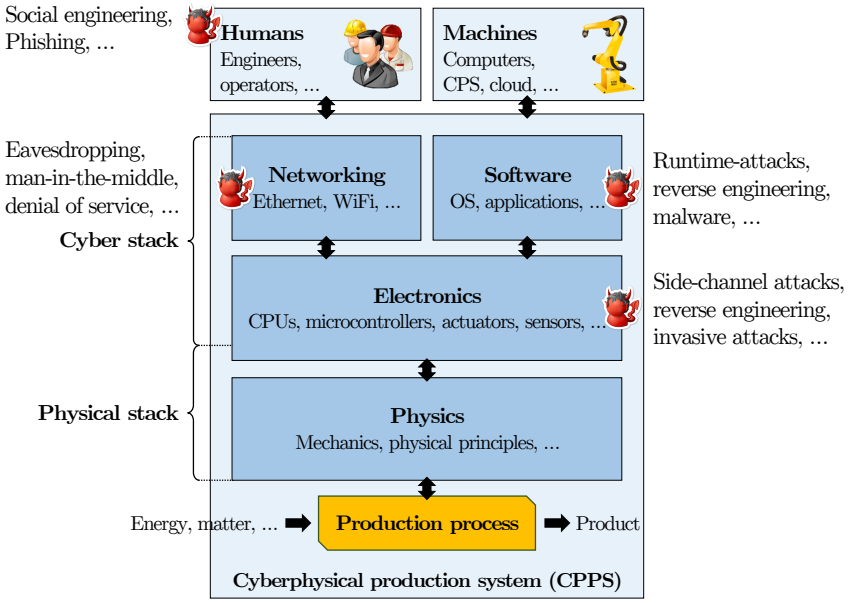


Figure 2.5: Categorising attacks on cyber-physical production systems based on the system architecture, Adopted from Sadeghi et al. (2015)

Another important cyber security threat factor in IIoT deployments is the convergence of Operational Technology (OT) and Information Technology (IT) (Murray, Johnstone, & Valli, 2017). In traditional industry architectures, OT handles all the hardware and software that controls industrial automation, whereas IT is associated with networking, data storage and data processing and analysis. However, with the advent of Industry 4.0, the two domains are increasingly being merged to effectively utilise the capabilities of IoT in industries (iebmedia, 2019). The IT is susceptible to attacks such as DoS attacks, authentication attacks, MitM attacks, phishing and application attacks. In contrast, the OT is prone to attacks such as data manipulation, malware attacks, reverse engineering and various other physical device attacks (Panchal et al., 2018). The cyber security threats of OT and IT as individual domains have been well researched, however, the security impact on industry 4.0 with the merging of IT and OT, warrants further investigation (Murray et al., 2017; Panchal et al., 2018).

2.3.2 IoT Threat Impact

The cyber-security threats to IoT can exist in the different layers of the IoT architecture as discussed in previous section. The attacks directly targeting the IoT devices can result in damage, including device compromise or loss of information stored on them. Compromised IoT devices can be used to perpetrate other cyber-attacks such as DDoS attacks (Neshenko et al., 2019). In addition, adversaries can attack the Network Layer which includes attacks on IoT protocols, communication gateways, middle-wares or against brokers. Attacks can also be independently launched against the Application Layer protocol, to target the various services and applications accessing the data from the IoT devices.

The recent real-world attacks directly targeting IoT devices show the impact of device level cyber-attacks. IoT malware targeting IoT devices have been reported such as Mirai (Kolias et al., 2017) and BASHLITE (also referred to as Lizkebab, Torlus, gafgyt) (Angrishi, 2017) and 'BrickBot' (Olenick, 2017). Its highlighted that some of the bots participating in such attacks have more than one million devices out of which a large majority of devices are Digital Video Recorders (DVR) (FPAnalyst, 2016; Krebs, 2016b).

In a survey on automotive attack surfaces, Charlie and Valasek (2014) identified the main stages of remote attack on vehicles as remote compromise, sending injected messages to cyber-physical components and instructing the Electronic Control Unit (ECU) to perform unsafe operations. In

their survey they presented the attack surface, internal network architecture and computer controlled features of various car models. The authors also exposed the vulnerability of a Jeep Cherokee car by remotely attacking the vehicle and performing unsafe actions on the car (Charlie & Valasek, 2014).

Such device level attacks have been used to launch other cyber-attacks as reported by Krebs (2016b) in his blog post about IoT devices being used as SOCKS proxies to hide the real identity of the attackers. Similarly, Carna botnet (Botnet, 2013) was successfully deployed to compromise many vulnerable IoT devices, in order to scan the entire IPv4 address space. Pa et al. (2015) evaluated the dataset from Japan's Darkent monitoring systems and concluded that the number of telnet attacks have increased drastically in the past couple of years. They deployed honeypots emulating the telnet service on IoT devices to study the IoT attacks. The experiments carried out by the authors show that most of the devices attacking their honeypot were IoT devices such as DVR, IP camera and Wireless routers (Pa et al., 2015). Reports such as FPAnalyst (2016); Krebs (2016a) have been published about IoT devices being used for launching large-scale Distributed DoS (DDoS) attacks. Krebs (2016a) reported that his blog site had fallen victim to a DDoS attack generating around 650 Gbps of traffic. This attack was mostly generated using a number of IoT devices like routers, IP cameras and DVR (Krebs, 2016a).

With access to millions of infected IoT devices, DoS attacks such as 1.35Tbps DDoS attack on GitHub (Chadd, 2018), and Mirai botnet based DoS attack on Dyn DNS services (Mansfield-Devine, 2016) are being perpetrated frequently. Hence, the Internet exposed components of IoT system will be prone to such DoS attacks. Peraković et al. (2015) conducted an analysis on the rise of DDoS attacks which indicates that the volume of DDoS attacks in 2013 rose by 475% and by 615% in 2014 compared to the attacks recorded in 2012. Especially, the Simple Service Discovery Protocol (SSDP) based attacks have sharply increased by 20% in 2015 compared to 2014. SSDP protocol is widely used in detecting UPnP enabled devices and the rise in attacks based on SSDP protocol has been attributed to the concurrent increase in the number of IoT devices. In a whitepaper presented by Moore (2013), the developer of Metasploit, stated that nearly 81 million unique IP addresses respond to UPnP discovery requests and out of these, 23 million are vulnerable to remote code execution. Many of the home devices have UPnP enabled and are vulnerable to attacks (Moore, 2013). The DoS attacks are also evolving and are increasingly being carried out as part of multi-vector attacks which include an initial DDoS attack followed by

malware, ransomware or viruses (Chadd, 2018). The DoS attacks are also projected to generate greater attack volume with the widespread adoption of IPv6 (Chadd, 2018). Figure 2.6 summarises the various IoT architecture layers, their role and the various attacks that target each layer.

Similar to IoT, the IIoT is also vulnerable to various attacks. Successful attacks against Industrial Control Systems (ICS) such as the Slammer worm (Poulsen, 2003) which attacked the critical monitoring systems of a nuclear power plant in the United States of America and Stuxnet (Langner, 2011), which caused the failure of centrifuges in a nuclear plant in Iran, and Triton malware which targeted industrial safety system (Giles, 2019) shows the potential impact of cyber threats in the IIoT.

Many of the attacks discussed above are those that target vulnerable IoT devices exposed to the Internet. However in most IoT deployments, the IoT devices might not be directly exposed to the Internet but will have the capabilities of exchanging messages through it (Waher, 2015). In such scenarios, the IoT protocols, gateways and the middle-ware platforms that facilitate message exchange will be prone to various cyber-attacks that target IoT deployments. Especially, public or cloud based message brokers or gateways will become the target of various cyber-attacks (Metongnon & Sadre, 2018). These central message brokers can be susceptible to DoS attacks and pose a serious threat to the numerous IoT devices that exchange messages through them. Hence it is necessary to identify DoS vulnerabilities of the IoT systems in order to detect and characterise such attacks, and to help build secure IoT systems. The following sections discuss the DoS attack techniques adopted by adversaries and the DoS detection techniques that are used to detect and prevent such attacks.

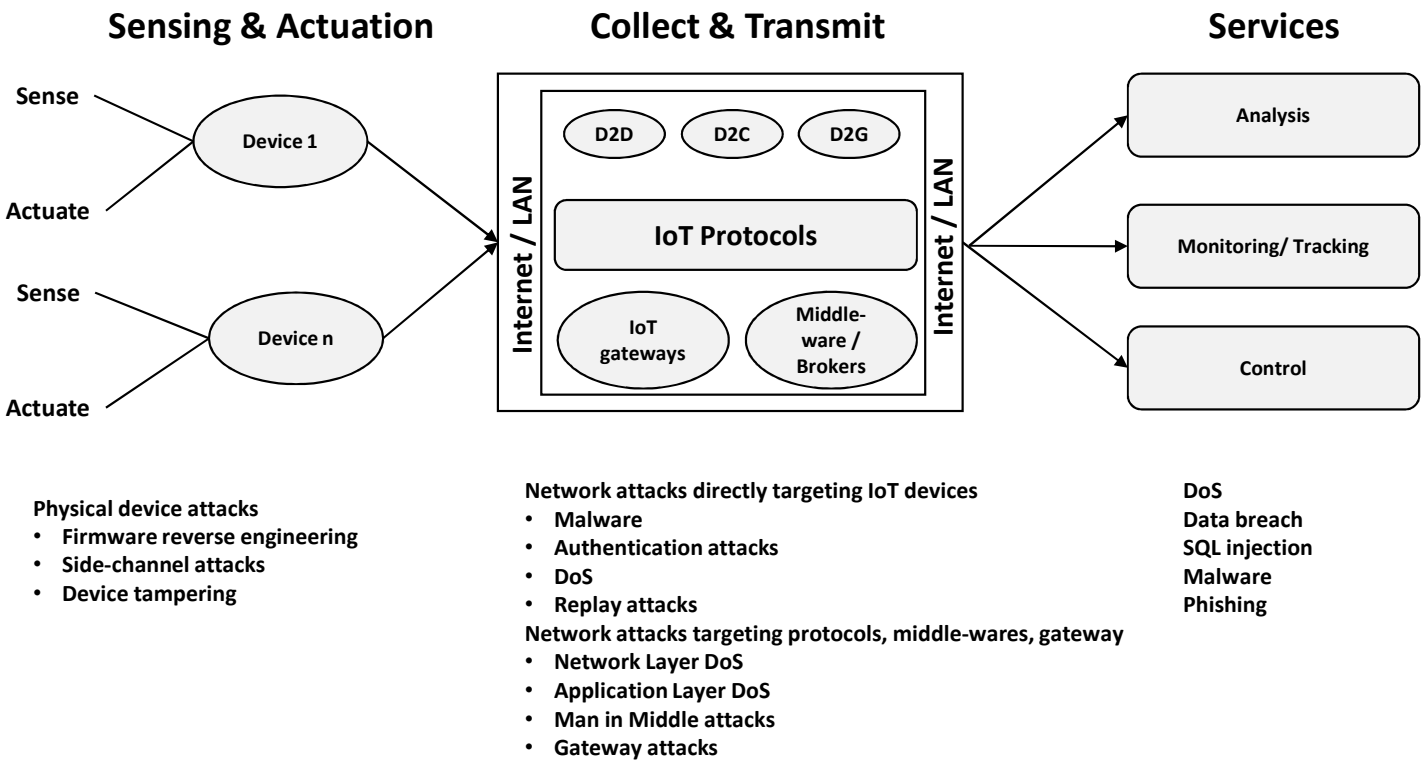


Figure 2.6: IoT attacks categorised by various architecture layers from perception layer (sensing and actuation) to application layer (services).

2.4 DoS Attack Techniques

A DoS attack is a type of cyber attack that aims to exhaust the resources of a target which leads to subsequent denial of resource access to legitimate users. The most common types of DoS attacks are those that aim to exhaust the network bandwidth, CPU cycles or memory on the target system to make services unavailable for legitimate users (Durcekova, Schwartz, & Shahmehri, 2012; Zlomislić, Fertalj, & Sruk, 2017).

The key network parameters that an adversary attempts to exploit in order to launch a DoS attack can be modelled using the Little's theorem (Little & Graves, 2008). According to Little's theorem, the number of items in the queuing system L can be written as:

$$L = \lambda * W \tag{2.1}$$

where λ is the arrival rate of items and W is the average processing time of items as highlighted in Figure 2.7.

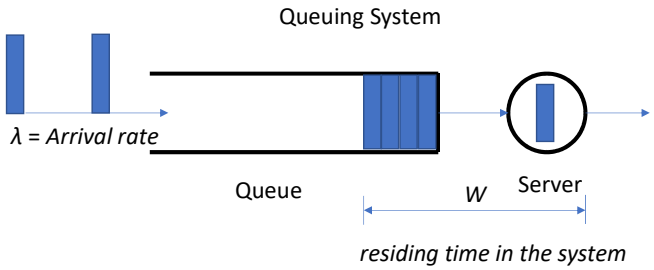
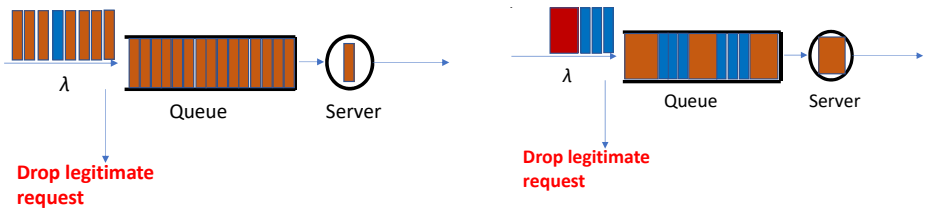


Figure 2.7: Representation of a typical queueing system characterised by packet arrival rate, queue length and processing time

Based on the queuing model, the DoS attack's aim is to increase the number of items in the system queue in such a manner that impacts the processing of current clients in the system or the availability of resources for new clients arriving into the system. Hence, DoS attack can either increase the arrival rate of packets at the victim machine (flooding attacks) so that it exceeds the processing capacity of the system ($\lambda > W$) or send requests that require higher processing time (semantic attacks) in the system (Khan & Traore, 2005) as shown in Figure 2.8. Thus the DoS attack control parameters are arrival rate to the system (λ) and complexity of the request.

DoS attacks could be launched from a single attack source or from multiple-attack sources (Ramanauskaite & Cenys, 2011). Attacks from sin-



(a) (b)

Figure 2.8: DoS Attack methodologies in exploiting the queue size by a) Flooding (increase arrival rate) and b) Semantic Attacks (increase processing time)

gle source usually target bugs or vulnerabilities in the target system or use powerful machines to achieve attack success. In contrast, those attacks that involve multiple attack sources rely on flooding the target system with requests emerging from distributed attack sources, to overwhelm the victim. Attacks launched from multiple sources are also referred to as Distributed DoS (DDoS) (Zlomislić et al., 2017). Since DDoS attacks are a subset of the DoS attacks, the usage of the term DoS will mean both DoS and DDoS attacks in the context of this thesis.

2.4.1 DoS Attack Classification

The Internet protocol suite is the most widely used suite of protocols in the data networks, that defines the data packet schemes, addressing schemes, packet routing, and transmission and receiving schemes. The Internet protocol suite is commonly referred to as the TCP/IP protocol stack. Based on the role of the protocol, the TCP/IP stack groups protocols into four layers of abstraction as listed in Table 2.3.

DoS in data networks targets various protocols used for end-to-end data communication between hosts, hence the attacks can be classified based on the protocol layer they target. Based on the communication protocol targeted, the DoS attacks can be classified in to Physical Layer attacks, Network Layer attacks and Application Layer attacks. In addition, the DoS attacks can also be classified based on the launch methods used in the attack; flooding or semantic attacks (Bhatia, Behal, & Ahmed, 2018). Since this work associated with higher layer DoS attacks, Physical Layer attacks are not discussed in the thesis. The attack launch type and the attack target protocol are further discussed in the following sections.

Table 2.3: TCP/IP protocol stack and associated TCP and UDP based protocols

TCP/IP Layers	**Role**	**TCP/UDP Protocols**
Application Layer	Application Data Access	HTTP, DNS, FTP, SMTP, SIP, MQTT, XMPP, AMQP, CoAP, SNMP ..
Transport Layer	End-to-End delivery	TCP, UDP
Network Layer	Packet Routing	IPv4 / IPv6, RLP, ARP, ICMP, IGMP
Network Access Layer	Physical Media Access	Ethernet, Wi-Fi, GSM, LTE, Z-WAVE, WirelessHART ..

2.4.2 Flooding DoS Attacks

Flooding DoS attacks involve transmission of higher volumes of communication requests than what the target system can handle in order to overwhelm the target's resources (Bhatia et al., 2018; Adi, Baig, Hingston, & Lam, 2016). To flood the victim machine with requests, the attacker can either use infected machines that are part of a botnet or use source IP address spoofing to amplify the malicious requests (also referred to as reflection attacks) sent to the target system. Figure 2.9 and Figure 2.10 illustrate the two techniques namely, DoS flooding and amplification.

2.4.3 Semantic DoS Attacks

Semantic DoS attacks operate in contrast to flooding attacks, targeting specific vulnerabilities of the system or utilising stealth attack techniques to disrupt services to legitimate users (Bhatia et al., 2018). These attacks require the adversaries to have in depth knowledge of the communication protocol, the application behaviour and their vulnerabilities being targeted. Such attacks will be difficult to detect and can be successfully launched from a system with few resources compared to the target (Bhatia et al., 2018). The stealth attack techniques exploit application or protocol logic to cause DoS and avoid detection by sending DoS traffic at a low rate (K. Singh, Singh, & Kumar, 2017; Adi et al., 2016). The flooding and semantic attacks both target the protocols of TCP/IP suite as further elaborated in the following sections.

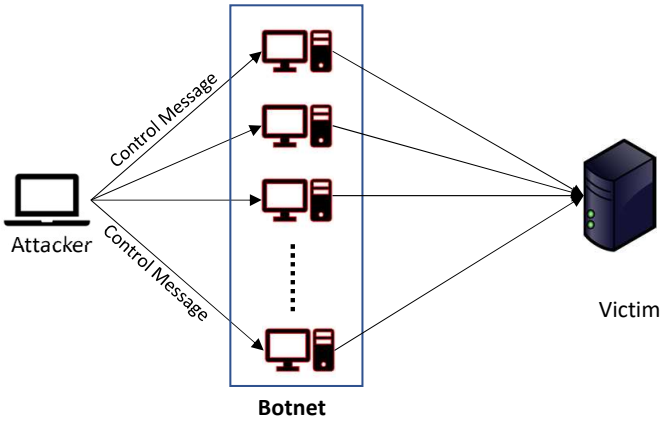


Figure 2.9: Flooding DoS attack using infected hosts

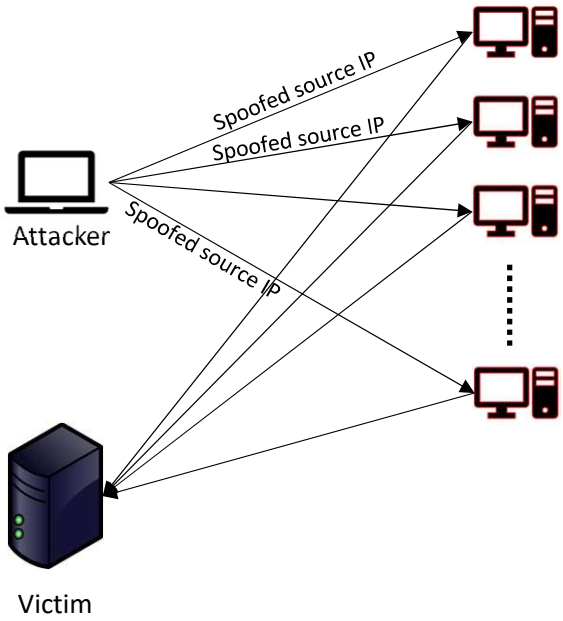


Figure 2.10: DoS amplification attack using spoofed source address

2.4.4 Network Layer Attacks

These attacks target the Transport and Network Layer protocols of the TCP/IP stack such as ICMP, TCP and UDP. The attacks could target a specific vulnerability of the protocol or send excessive requests to overwhelm the target. Examples of such attacks are: ICMP flood, Ping-of-Death, SYN-Flood, SYN-ACK flood and Teardrop attack.

ICMP Flood Attack In an ICMP Flood attack (also referred to as Smurf attack) the attacker overwhelms the victim machine by sending significantly higher volumes of ICMP ECHO Request messages (Lau, Rubin, Smith, & Trajkovic, 2000). Typically, an ICMP ECHO Request is responded with a ICMP ECHO Reply. Hence in this type of attack the attacker uses botnets to flood the victim machine with ICMP ECHO requests, the victim machine while trying to respond to individual requests with a ICMP ECHO Reply message, exhausts its resources.

Ping-of-Death Attack A more sophisticated type of ICMP attack is the Ping-of-Death attack that targets vulnerable OS implementations by sending IP datagrams which are larger than the allowed size of 65,535 bytes (Kenney, 1996). Figure 2.11 illustrates the Ping-of-Death attack process. The attack uses a carefully crafted ICMP packet to cause a buffer overflow at the server, resulting in a server crash or reboot. As IP packets get fragmented when packet size is greater than the typical Maximum Transmission Unit (MTU) of 1500 bytes, a carefully chosen valid offset and a suitable fragment size can cause a buffer overflow during fragment reassembly at the target computer. This will result in the victim machine crashing or rebooting due to the buffer overflow.

SYN-Flood SYN flood attacks target the TCP protocol's state retention mechanism (Eddy, 2007) and is a popular method for DoS (Mansfield-Devine, 2015). The attack exploits the OS's implementation of the TCP protocol which restricts the maximum number of simultaneous TCP connections in a SYN-RECEIVED state at the host. The attack tries to flood the target with multiple TCP SYN segments to exhaust the number of SYN-RECEIVED connections the target can handle at a given time, and to prevent subsequent processing of TCP SYN connection requests. With every SYN segment received, the victim machine allocates state, and all the state information is stored in a data-structure called Transmission Control Block (TCB). Memory is allocated to store the TCB which contains the

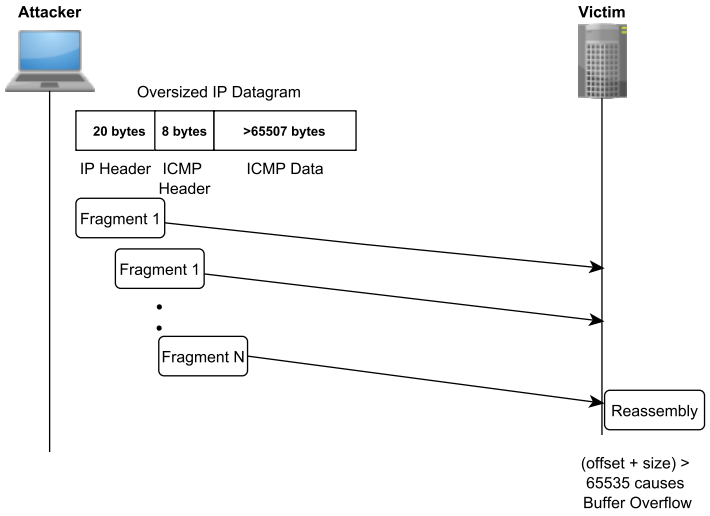
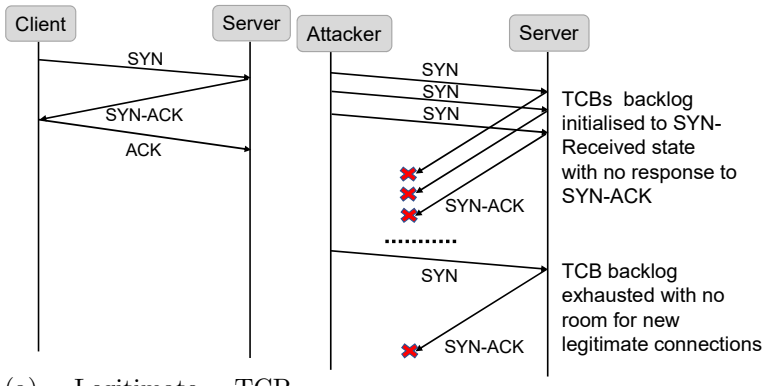


Figure 2.11: Ping-of-Death attack process

connection information from the SYN header. However, most OS kernels maintain limits to the number of TCB data structures that can reside in the memory at a given time, which are referred to as backlog. When the backlog limits are reached, the system discards the new SYN packets or replaces uncompleted sessions. Hence, the SYN-flood attack tries to exhaust the OS backlog limits by sending numerous SYN packets without completing the TCP three-way handshake, thereby filling up the backlog limits with half-open sessions. This causes legitimate connections to be dropped and denied access to resources, causing DoS. Figure 2.12 illustrates a legitimate TCP three-way handshake (a) and the malicious TCP handshake (b) to exhaust the backlog limits of the victim.

2.4.5 Application Layer Attacks

Similar to the attacks on Network and Transport Layer protocols, DoS attacks also target the Application Layer protocols of the TCP/IP protocol stack. Application Layer protocols are used by the applications to exchange user data between hosts by utilising the services of lower layer protocols. Since Application Layer protocols rely on functionality of the lower layer protocols, the DoS attacks targeting them use legitimate connection request mimicking the behaviour of legitimate users (K. Singh et al., 2017). Es-



(a) Legitimate TCP handshake process

(b) Malicious TCP handshake process

Figure 2.12: SYN-Flood Attack Process

pecially, Application Layer protocols relying on TCP protocol require the completion of the three-way handshake which necessitates the use of legitimate IP address and active hosts in DoS attacks. Hence, such attacks can pose challenges to the DoS detection system. Findings reported in the literature indicate Application Layer attacks are being increasingly perpetrated by adversaries (Brenner, 2010; Mantas, Stakhanova, Gonzalez, Jazi, & Ghorbani, 2015) to maximise the impact of DoS attacks by sending carefully crafted, legitimate requests towards the victim. Adversaries use either flooding or semantic technique to generate such attacks as described earlier. The most common DoS attacks against Application Layer protocols exploit vulnerabilities in the initial connection-establishment message exchanges. Table 2.4 lists the Application Layer protocols targeted, request types and the attack launching methods employed to cause DoS.

HTTP-Flood HTTP protocol is one the most widely deployed protocol on the Internet due to the dominant presence of World-Wide-Web (WWW). Due to its wide usage, HTTP protocol is also one of the most targeted Application Layer protocols (Network, 2015). HTTP is a Request/Response protocol and uses various request and response messages to exchange data between the clients and server. A HTTP DoS attack floods the webserver with request messages resulting in exhaustion of resources in responding to multiple requests. During flooding attacks the arrival rate of new requests is higher than the response rate of the webserver which quickly fills up the request queue thereby denying service to legitimate clients. The request

Table 2.4: DoS attack strategies adopting protocol-specific legitimate requests for various Application Layer protocols

Application Layer Protocol	**Request Type**	**Attach Launch Method**
HTTP (K. Singh et al., 2017; Adi et al., 2016; Ranjan, Swaminathan, Uysal, Nucci, & Knightly, 2009)	HTTP-GET, HTTP-POST	• Flooding (exponentially increasing attacks, Flash attacks, constant high-rate attacks) • Low Rate: Periodic (square wave DoS stream – low average packet rate (Shan, Wang, & Pu, 2017a)) , Slowloris
SIP (Rafique, Akbar, & Farooq, 2009; Luo, Peng, & Leckie, 2008)	SIP-INVITE, SIP-REGISTER	Flooding
DNS (Ballani & Francis, 2008)	Name resolution query	Flooding
SMTP (Bencsath & Ronai, 2007)	Email flooding	Flooding

messages used in HTTP flood attack are: HTTP-GET and HTTP-POST. In HTTP-GET attack multiple requests for resources such as images or files are sent to the webserver. In the HTTP-POST attack, multiple form submissions are performed on the website which in turn requires the webserver to push data to other layer of software architecture.

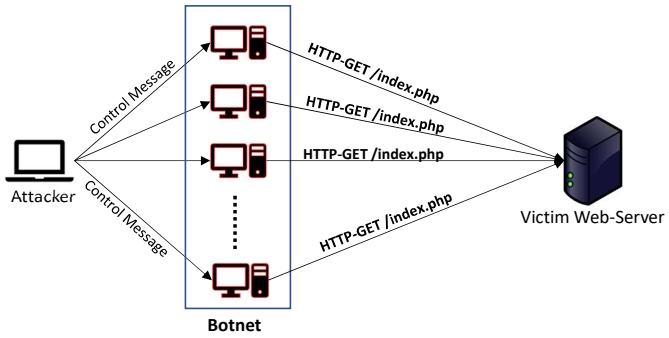


Figure 2.13: HTTP-GET DoS flooding attack

SIP-Flood Session Initiation Protocol (SIP) is a signalling protocol widely used to establish real time communication between participants, especially in IP telephony. It can be used for both voice and video communications which identifies the participants of the call and the methods to reach each other on the IP network. SIP servers are deployed to authenticate and setup calls between the participants. Similar to other Application Layer protocols, SIP uses signalling messages such as INVITE, REGISTER, ACK, BYE and CANCEL between the clients and server to establish calls. In SIP flooding attack the adversary can flood the SIP server with numerous INVITE or REGISTER messages to consume the resources of the server to cause DoS (Luo et al., 2008; Rafique et al., 2009).

Slowloris Slowloris attack is a semantic attack type which does not flood the victim with numerous requests instead it tries to exploit the application logic of webservers (K. Singh et al., 2017). In this attack, the adversary opens multiple connections with the webserver by sending partial HTTP-GET requests. A multi-threaded webserver opens a thread for each request and keeps the threads open until the connection is completed. If the connection remains idle for a long period the webserver will timeout the connection and frees the thread allocated to that connection. But in a Slowloris attack the adversary sends partial HTTP-GET request headers to keep the connection alive and prevents the webserver from timing out the connections. This results in webserver's threads locked in partial connections and once the thread limit is reached it stops responding to new requests causing DoS.

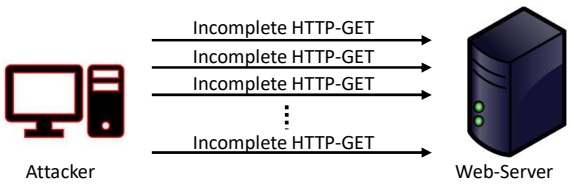


Figure 2.14: Slowloris DoS attack technique

The various DoS attack strategies and the data communication protocols targeted by the adversaries are summarised in Figure 2.15 as four-quadrants of attack classification.

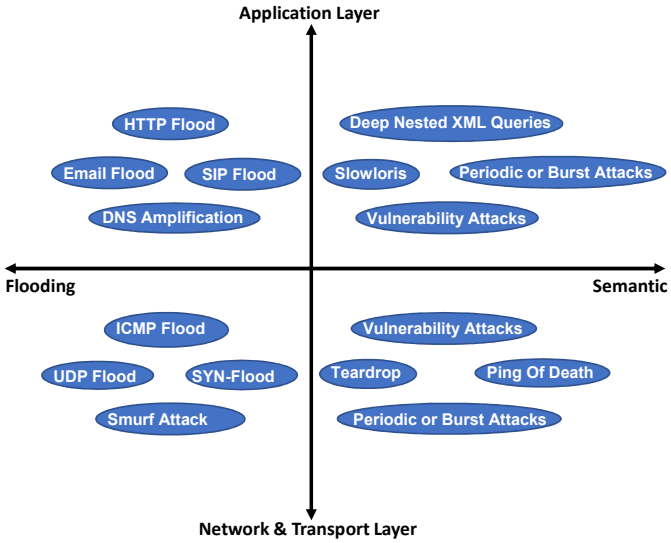


Figure 2.15: Four quadrant DoS attack classification by the protocol layer and dos attack type (flooding vs semantic)

2.5 DoS Attack Detection Techniques

Attack detection is one of the important steps in mitigating the malicious effects of DoS attacks. DoS attacks that target a specific vulnerability of the victim machine can be protected by patching the vulnerability (Carl, Brooks, & Rai, 2006; Bhuyan, Kashyap, Bhattacharyya, & Kalita, 2013). However, the flooding based and stealthy DoS attacks that exploit the various layers of the Internet Protocol suite pose a challenge to attack detection and countermeasure techniques. The detection techniques will also vary based on the location of deployment into (Bhuyan et al., 2013):

- Source-end,
- Victim-end, and
- Intermediate deployments.

Source-end deployments usually detect the attacks at the source of the attacks and prevent DoS traffic from traversing further. However, such methods would be ineffective in a mutli-source DDoS attack as source machines are distributed and the observed traffic does not deviate much from normal traffic (Bhuyan et al., 2013). In the victim-end detection scheme, either

online or offline detection techniques are employed at the target of the attack. In the intermediate detection scheme, intermediate devices through which the DoS traffic traverses are responsible to detect and rate limit the malicious traffic. In this work we only focus on the victim end detection scheme.

Various works have also suggested different classification methods for DoS attack detection. The earliest classification approaches considered DoS detection as a reactive mechanism to detect intrusions (Douligeris & Mitrokotsa, 2004; Mirkovic & Reiher, 2004; Peng, Leckie, & Ramamohanarao, 2007) and classified them into:

- Pattern or misuse detection
- Anomaly Detection

However, Carl et al. (2006) classified the detection approaches as statistical techniques which assess different statistical parameters of network traffic or monitoring statistical patterns for various traffic parameters (Asosheh & Ramezani, 2008). Other and more recent works have classified DoS attack detection techniques into (Bhuyan et al., 2013; Kaur, Kumar, & Bhandari, 2017; Hodo, Bellekens, Hamilton, Tachtatzis, & Atkinson, 2017; Tama & Rhee, 2015):

- Signature-based or knowledge-based
- Anomaly-based
  - Statistical
  - Data Mining
  - AI or ML techniques
- Hybrid

In a survey done by Zargar, Joshi, and Tipper (2013), the authors classified DoS detection techniques based on the protocol level at which DoS attack operates. The defence methods were classified as network level defence and application level defence. The application level defences are further classified into destination based and hybrid techniques. The destination based techniques detect and prevent attacks at the DoS attack target, typically a server.

Signature-based detection: use pre-defined signatures of attacks generated using attack data analysis. These are also referred to as misuse detection, rule-based or knowledge-based detection techniques. The behaviour

of existing attacks is identified using techniques such as expert systems, signature formulation, self organising maps, state transition analysis (Bhuyan et al., 2013). The main drawbacks of signature-based techniques are that: they are effective only in detecting known attacks; variations or new attacks can go undetected using such techniques, and therefore frequent learning and updating of new signatures is required.

Anomaly detection techniques: use various methods to distinguish between normal and abnormal patterns and behaviours in network traffic. They are also referred to as outlier or behaviour based detection techniques. This approach measures or identifies the normal behaviour of a system and any deviance from the normal or expected behaviour by a predefined threshold is flagged as an anomaly. This allows the anomaly detection method to detect new and unknown anomalies. However, it also generates a higher number of false alarms due to the presence of noise or changing network behaviour (Kaur et al., 2017).

Hybrid techniques: use combinations of signature-based and anomaly-based detection techniques to improve the overall detection accuracy of individual schemes.

A comprehensive DoS attack detection taxonomy presented in Kaur et al. (2017) is shown in Figure 2.16.

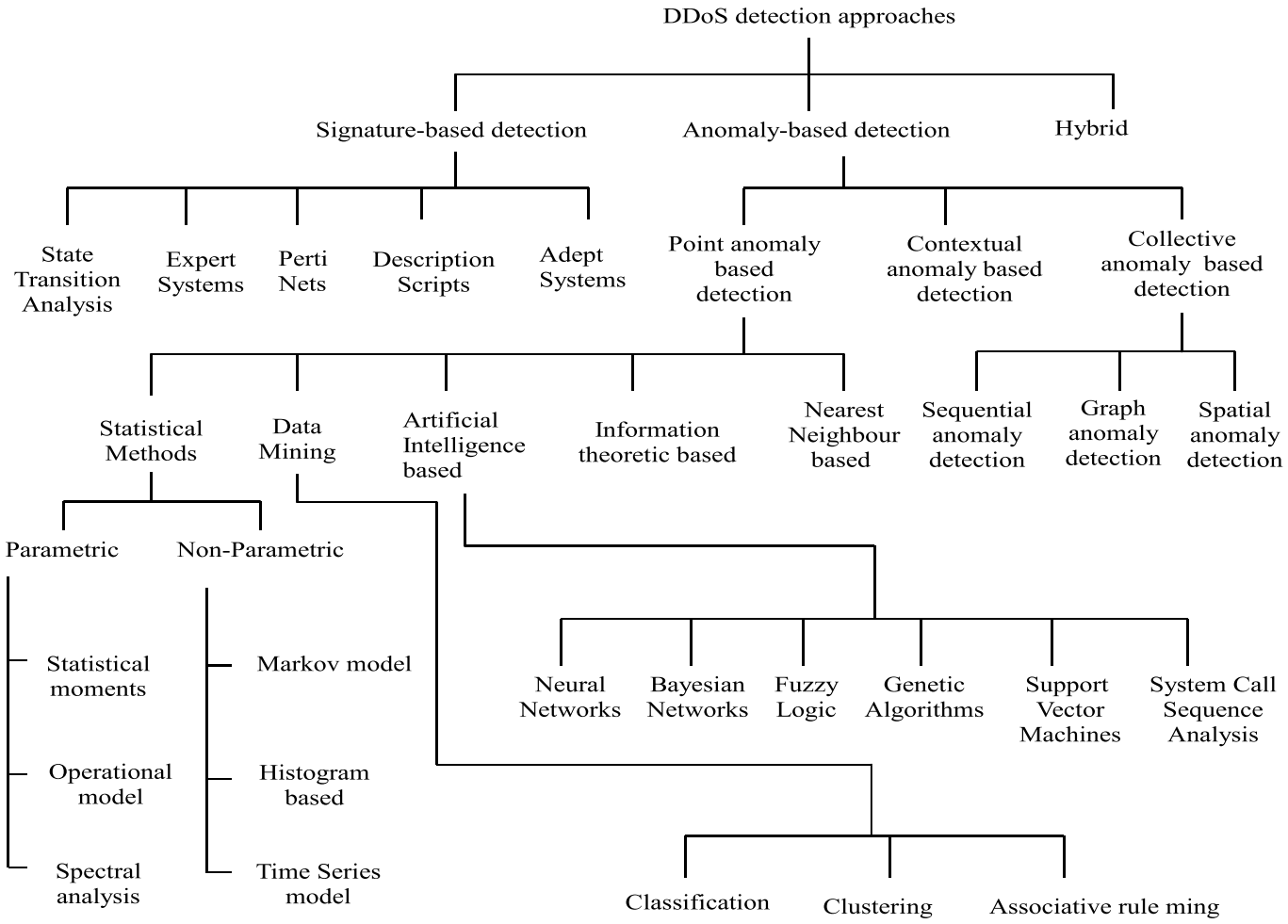


Figure 2.16: Taxonomy of DoS detection techniques classified by detection approach. Adopted from Kaur et al. (2017, p. 304)

2.5.1 Statistical Detection Techniques

Statistical detection techniques utilise the statistical properties of the normal and attack traffic to differentiate them and identify DoS attacks. These methods rely on developing a statistical model of the normal traffic and any traffic that deviates from the normal model by a threshold limit is flagged as attack traffic. The types of statistics measures applied are: change point detection using non-parametric Cumulative Sum (CUSUM) (Blazek, Kim, Rozovskii, & Tartakovsky, 2001; Wang, Zhang, & Shin, 2002), two-sample t-test (C. Chen, 2009), Auto-Regressive Integrated Moving Average (ARIMA) (G. Zhang, Jiang, Wei, & Guan, 2009), residue factor metric applied on session inter-arrival time (Ranjan, Swaminathan, Uysal, Nucci, & Knightly, 2008) or moving average (exponential and simple) on inter-arrival time (Bojović, Bašičević, Ocovaj, & Popović, 2019) or Kullback-Leibler Divergence (KLD) metric used to differentiate between normal and DoS attack probability distributions (Bouyeddou, Harrou, Sun, & Kadri, 2018). Other statistical techniques employed are based on Wavelet analysis (Dainotti, Pescapé, & Ventre, 2006; Lu & Ghorbani, 2009). The statistical techniques depend on models of normal and attack traffic measured based on features like inter-arrival time, IP addresses, packet sizes.

2.5.2 Targeted DoS Attack Detection

DoS attacks are becoming increasingly sophisticated and focused, as specific protocol layers or specific infrastructures are being targeted. As existing DoS detection techniques focus more on the packet arrival statistics to detect flooding or slow attacks, such schemes will fail to detect specially crafted Application Layer or domain specific DoS attacks (M. Singh, Rajan, Shivraj, & Balamuralidhar, 2015). Some of the challenges in detecting Application Layer DoS attacks are:

- Application Layer DoS attacks do not just rely only on vulnerability or gaps in the application or the protocol; but rather can also be launched using legitimate requests.
- Legitimate IP and Transport Layer parameters are used in Application Layer attacks, that can bypass lower layer detection mechanisms.
- Existing methods assume characteristics of DoS traffic differs from normal traffic, but such methods will fail when attack behaviour mimics the normal user behaviour.

- Presence of flash events (normal traffic but traffic resembles DoS traffic due to increased resource access) can cause high false alarms.
- Use of stealth or semantic attacks can increase the detection complexity.

This has resulted in researchers focusing on identifying protocol or infrastructure specific DoS vulnerabilities and detection techniques. Specific DoS attack classification or defence mechanisms proposed in the literature are listed in the Table 2.5. The current research work indicates that more and more targeted attacks are occurring and requires domain or application specific detection methods.

Table 2.5: Protocol or infrastructure specific DoS attacks and defences

Protocol / Infrastructure DoS Targets	**Specific Detection Classifications**
HTTP	HTTP-GET flood DoS attacks (M. Singh et al., 2015; Praseed & Thilagam, 2018)
SIP	SIP DoS attack detection survey (Ehlert, Geneiatakis, & Magedanz, 2010)
Web-Service	Web-service attacks survey (Jensen, Gruschka, & Herkenhöner, 2009)
Cloud	Intrusion detection techniques in cloud (Modi et al., 2013)
Publish/Subscribe	(Wun, Cheung, & Jacobsen, 2007)
Peer to peer networks	(Daswani & Garcia-Molina, 2002)
Smart-Grid /SCADA / DNP3	DNP3 attacks (East, Butts, Papa, & Shenoi, 2009), SCADA-specific intrusion detection survey (Zhu & Sastry, 2010), attack detection in Cyber-Physical Systems(CPS) (Mitchell & Chen, 2014)
SDN	survey of DoS attacks in SDN (Yan, Yu, Gong, & Li, 2015)

In order to detect such attacks, many domain or application specific features have been previously proposed. Some examples of domain specific features used for DoS attack detection are:

- HTTP browsing order of pages (Yatagai, Isohara, & Sasase, 2007):

The user behaviour of accessing web pages on a web server is used to differentiate between human and bot access behaviour.

- HTTP browsing time to information size correlation (Yatagai et al., 2007): The browsing time per page is used to distinguish between normal and DoS attack behaviour as the latter scenario would result in random and fast access to various web pages.
- SIP transaction features (E. Y. Chen, 2006): The call establishment process features are used to differentiate normal from malicious behaviour.
- Web server document popularity (Xie & Yu, 2009): The web server document popularity is used to distinguish between flash crowds and HTTP DoS attacks.

Based on the DoS attack classification presented in Figure 2.15 and the discussion on targeted DoS attack detection, a new DoS attack classification is presented in Figure 2.17. This classification scheme highlights that detection techniques employ different features based on the layer targeted by DoS attacks.

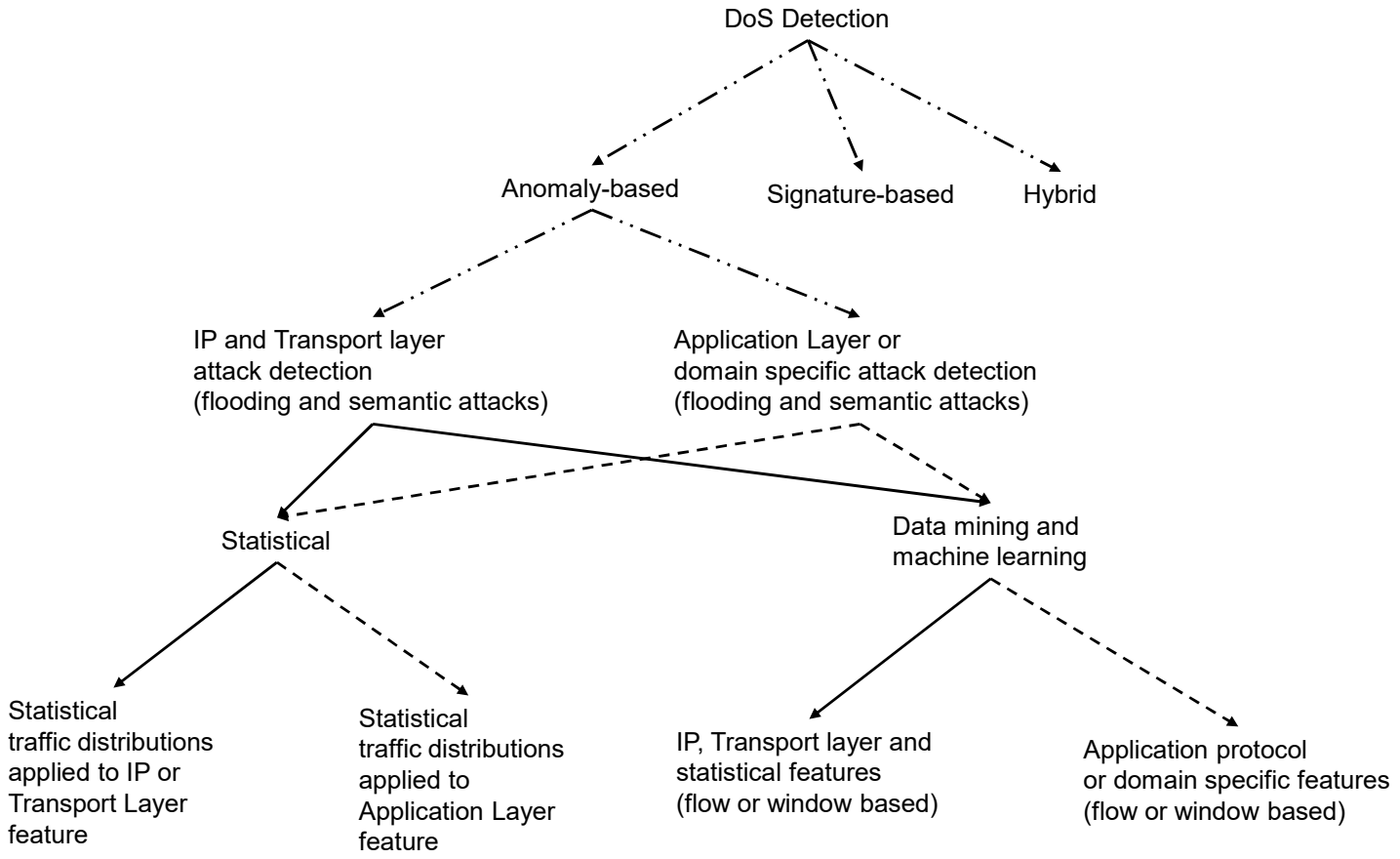


Figure 2.17: Proposed DoS detection techniques classification based on detection target layer and technique used

As discussed in earlier sections, IoT growth is powered by the complex amalgamation and advancements of various technologies and hence, are vulnerable to a wide range of targeted attacks. This necessitates building detection techniques relevant to IoT as the techniques discussed so far were designed and deployed to work for legacy networks. As discussed in sections 2.1 and 2.2, the communication paradigms and protocols used in IoT are different from traditional networks. In addition, Zarpelão, Miani, Kawakani, and de Alvarenga (2017) highlights that existing detection solutions are inadequate for IoT systems due to reasons such as: resource constraints of IoT devices, multi-hop communication strategies in IoT (use of gateways for resource constrained devices) and use of new generation protocols. The authors also highlight the lack of public IoT datasets for detection technique evaluation. The main challenges with DoS detection in IoT are:

- large number of endpoints,
- new communication paradigms,
- new communication protocols,
- more device generated traffic in IoT applications compared to human generated traffic in web applications, and
- increased DoS volume due to compromised IoT devices.

The recent trends in anomaly detection techniques indicate that AI based techniques are increasingly being adopted to detect DoS attack (Bhuyan et al., 2013; Kaur et al., 2017) due to various advantages such as:

- learning the behaviour of the system automatically,
- handling large volumes of data (in IoT age),
- using multiple features to create models of normal and attack traffic hence have more tolerance, and
- identifying complex relationships between features to detect novel attacks.

The following section discusses the various components of Machine Learning techniques and their phases.

2.5.3 Soft Computing, Data Mining and Machine Learning Based Detection

The other category of detection techniques employed in DoS attack detection are based on soft computing concepts such as Machine Learning (ML) algorithms. Data mining and machine learning are a subset of data science. Furthermore, machine learning forms a part of soft computing (Chaturvedi, 2008), hence it is reasonable to discuss these techniques together in the context of DoS attack detection techniques. Machine learning based detection techniques first identify suitable features from captured DoS attack data and then use unsupervised or supervised methods to classify network traffic as normal or DoS attack (Bhuyan et al., 2013; Tama & Rhee, 2015).

Machine Learning (ML) is an application of AI which deals with automatic recognition of useful patterns and intelligent decision making based on provided data (Shalev-Shwartz & Ben-David, 2014). The main goal of a ML is to learn patterns from the provided data samples and produce a generalised model in order to predict or classify unknown data instances. The learning step on sample data instances in ML is referred to as training phase and evaluation step on unknown data instances is referred to as testing phase.

Given a dataset $\mathcal{X}$ represented by a feature vector (F): $[f_1, f_2, ..., f_n]$, and an output class label $C = [C_1, .., C_l]$, provides a paired labelled training samples $\mathcal{S} = [(x_i, y_i), i = 1, ...m]$ where x_i represents the input data instance and y_i represents the corresponding expected output class label. So the task of an ML algorithm is to estimate a function $\mathcal{F}$ that best represents the relationship between input x_i and output y_i from $\mathcal{S}$ without the loss of generalisation and reducing the learning error between expected output class y_i and the predicted output class $\bar{y}_i$ (Dua & Du, 2016). The developed model's performance is evaluated by measuring the number of correctly classified instances and miss-classifications on a testing dataset **D**.

The typical work-flow of an ML project consists of collections from various sources, feature extraction from the raw data, model learning and model testing as depicted in Figure 2.18.

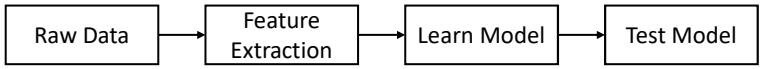


Figure 2.18: Typical ML work-flow used in classification or prediction

The feature extraction step and the model learning steps sit between the

raw data and the model testing phases. Hence choosing the right features and the classifier plays a crucial role in building a most effective model (Hodo et al., 2017; Zheng & Casari, 2018).

ML techniques have been widely used in cyber-security domain, especially in anomaly detection applied commonly in Intrusion Detection Systems (IDS) (Dua & Du, 2016; Hodo et al., 2017). Specifically, ML techniques have been employed in detecting DoS attacks (Tama & Rhee, 2015; Hodo et al., 2017). In order to build detection models for DoS attacks, datasets containing normal and attack traffic are required. Various methods or data sources have been used to collect the data required for model evaluation. The following section discusses the various data sources or methods used in the literature.

Data Sources

Data collection phase is used to collect raw data required to train and build a detection model. The effectiveness of the developed model will depend on the quality of data that is used to train the model. In DoS detection problem, the ML technique can be effective in detecting DoS attacks only if the training datasets include quality samples of benign and attack traffic. Some of the important characteristics of a good dataset as per Koroniotis, Moustafa, Sitnikova, and Turnbull (2019) are: the dataset should contain realistic traffic, the data should be labelled for supervised learning techniques, the dataset must contains diverse benign and attack scenarios, full packet captures are preferred so as to generate new features. Praseed and Thilagam (Praseed & Thilagam, 2018) highlight the limited availability of datasets containing Application Layer DoS traffic which has led many researchers to generate attack traces using existing DoS attack tools. In addition, the existing datasets are older than 2007 and are no longer relevant in the IoT era. Hence there is a dearth of IoT based public datasets to build effective detection techniques (Koroniotis et al., 2019). Some of the common methods that have been employed to obtain datasets required for DoS detection evaluation are (Bojović et al., 2019; Koroniotis et al., 2019; Hodo et al., 2017; Praseed & Thilagam, 2018):

- **Realistic Test-bed**: real or virtual endpoints are used which generate benign traffic and attacks are generated using existing tools or attack simulations.
- **Real network traffic**: dataset containing benign and attack traffic obtained from real production networks. This type of dataset is rare

as most organisations do not publicly share their network traffic for privacy and security concerns.

- **Public dataset**: publicly hosted dataset of benign and attack traffic or only benign traffic (attack traffic is synthetically injected to the dataset to evaluate the detection technique).

Table 2.6 lists the various public datasets used in DoS detection evaluation and their applicability in the IoT domain. Specifically these datasets are not suitable to detect MQTT protocol based attacks due to the lack of attack traffic related to the protocol.

Table 2.6: Comparison of public datasets used for anomaly detection

Dataset	**Year of collection**	**IoT Dataset**	**MQTT Attack Traces**
DARPA99 (MIT, 1999), KDD99 (UCI, 1999), CAIDA DDoS (UCSD, 2007)	Prior to 2007	No	No
UNSW-NB15 (Moustafa & Slay, 2015), CICIDS2017 (Sharafaldin, Lashkari, & Ghorbani, 2018)	After 2015	No	No
BoT-IoT (Koroniotis et al., 2019) dataset	2018	Yes	No

These limitations with regards to the availability of real production and public IoT datasets suitable to the DoS detection problem requires that a realistic test-bed that emulates a real IoT network should used to collect benign and attack datasets.

Feature Extraction

Features extraction plays an important role in anomaly detection. The features used for anomaly detection define the structure of the data, whereas ML algorithms are used to identify the patterns that exist in the data structure to differentiate between normal and attack traffic. Hence, the success of the ML algorithm depends on the most suitable features that can effectively

differentiate normal from attack traffic (Hodo et al., 2017). For network anomaly detection, direct or derived features are used to detect abnormal traffic patterns (Dua & Du, 2016; Hodo et al., 2017). Direct features can be based on the protocols used at the IP, Network or Application Layers. In contrast, derived features can be based on network access dynamics (generally applied to network flows or time-intervals) or based on domain specific behaviour (Hodo et al., 2017). Figure 2.19 shows various layers of packet data available for feature extraction in a IP packet.

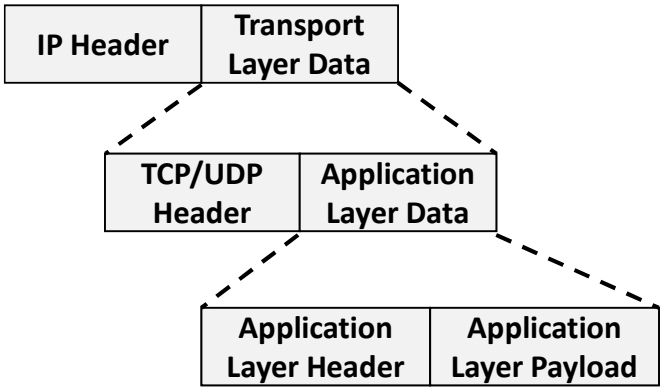


Figure 2.19: IP packet encapsulation that can be utilised to extract features for ML based detection

For Application Layer attacks detection, the network traffic can also be analysed at various levels to extract Application Layer information such as (Rieck, 2009):

- **Packet level**: The Application Layer data contained in packets can be easily accessed and can be directly used to extract application data. However, this requires quick decision making time to categorise packets as normal or malicious and can be bypassed by simple evasion techniques like segmentation (Rieck, 2009).
- **Request level**: The Application Layer data can be extracted by monitoring individual application requests which requires re-assembly of all the packets that belong to the request thereby solving the problem exhibited by simple evasion techniques. This allows the analysis of complete traffic payloads for pattern matching based intrusion detection (Sperotto et al., 2010).
- **Connection level**: At this level the full communication session is

monitored and all the packets that are transferred during the session are considered for marking a connection session as normal or malicious. This however, increases the decision time to due to variable length sessions.

- **Time-Windows/Multiple connections**: These techniques rely on change detection to detect intrusions in network traffic modelled as time-series (Krishnamurthy, Sen, Zhang, & Chen, 2003). They monitor multiple connections or packets contained either in a time-window or through a connection limit. These techniques can detect anomalies that span multiple connections and spread across various time-windows. This method might be limited in detecting malicious sessions from normal sessions as it depends on multiple sessions.

Table 2.7 lists the direct and derived features used in various works (M. Singh et al., 2015; Praseed & Thilagam, 2018; Hodo et al., 2017), for anomaly detection. Since HTTP protocol has attracted more attention from researchers, a wide range of domain specific features have been proposed in the literature (M. Singh et al., 2015; Praseed & Thilagam, 2018). The direct and derived features that are based on IP and Transport Layer protocols can be used across various Application Layer protocols for DoS detection as these protocols are encapsulated in the IP packet. However, domain specific features that are based on the internal structure and interactions of applications are not suitable to detecting attacks in other protocols or domains. For example, human user access behaviour and webpage popularity is used in HTTP protocol to detect attacks based on normal and attack traffic distributions, however, these features are not applicable to other applications. As domain specific or application protocol specific cannot be reused for other domains or protocols, features suitable to detect adversarial actions against new applications and protocols need to be identified by in depth analysis of the domain.

Model

ML employs various mathematical algorithms to find the relationships in the provided data. ML techniques can be classified into: supervised or un-supervised learning. Supervised ML techniques use a labelled dataset also referred to as ground truth to learn the relationship between the input and output. Hence, a prior knowledge exists about the output values of individual samples. In contrast, un-supervised learning uses a un-labelled dataset to identify the natural structure or patterns in the data. In supervised

Table 2.7: Direct and derived features used in various layers and protocols of TCP/IP stack

Protocol / Layer	**Types of Features used**	**Derived Features (flow based, or time-interval based)**	**Applicable to MQTT**
IP	Packet Size, IP address (source-destination)	Inter-arrival time, bytes from source to destination vice versa, count based features applied to flow/time-interval,	Yes
TCP	TCP Flags, source and destination ports, segment length	-	Yes
UDP	UDP Flags, source and destination ports, segment length	-	No
HTTP	HTTP protocol flags, response codes	User browsing behaviour, page/resource popularity, request dynamics, request workload behaviour, session inter-arrival time, backend request statistics	No
DNS	DNS Flags	Protocol based Request and responses	No
SIP	SIP flags,	Protocol based Request dynamics	No

ML, the algorithm takes samples of labelled data to build a mathematical model that formulates the relationship between input and output variables. The model building phase is also called training. The developed model's performance is assessed by data samples that were not previously used for building the model. The model evaluation phase is also referred to as testing. A number of ML algorithms exist, but the most popular methods include (Dua & Du, 2016; Buczak & Guven, 2015; Tama & Rhee, 2015): Artificial Neural Networks (ANN), Support Vector Machine (SVM), decision trees, Bayesian networks, Naive Bayes, k-Nearest Neighbour and Hidden Markov Model (HMM). A few methods are discussed in this section.

ANN: is a ML technique that is inspired from neural networks of biological brains that takes inputs X and transforms into outputs Y using non-linear functions over multiple hidden layers of artificial neurons. The connections between neurons are referred to as edges and each edge consists of weights W that are tuned during the training phase as shown in Figure 2.20. The main goal of an ANN model is to minimise the classification error by tuning the hidden layer weights. Given the ground truth Y and

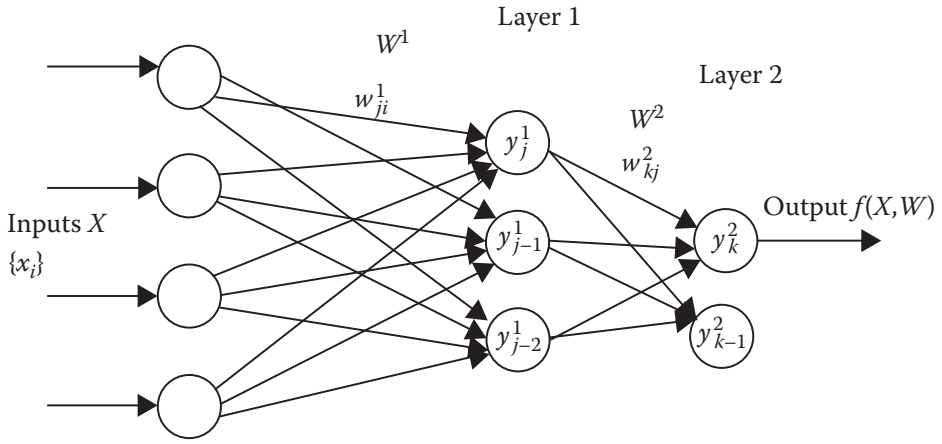


Figure 2.20: A two layer ANN feed-forward network consisting of one input layer, one hidden layer and an output layer. Adopted from Dua and Du (2016)

the expected output $f(X;W)$ or $\hat{Y}$, then ANN aims to reduce the error $E(X) = (\hat{Y} - Y)^2$. This is achieved by tuning the weights and choosing the correct activation functions of a neuron. Individual neurons have a summation function that combines all the inputs and the associated weights and is further transformed using a transfer function T_f (also referred to as action function) as shown in Figure 2.21. In certain networks a bias value b is also introduced to shift the output based on the learning requirement.

Hence, the output of individual ith neuron with input x_j and weights w_{ij} is represented as:

$$y_j = T_f(b + \sum_{j=1}^{n} x_j w_{ij}) \tag{2.2}$$

A wide range of functions are used as the transfer function. The most common types are step, threshold and sigmoid function (Dua & Du, 2016). In order to reduce the error $E(x)$, the learning task in an ANN involves calculating the best values of hidden layer weights and bias to reduce the error between predicted output and the ground truth. Various algorithms have been proposed which iteratively calculate the best weights and biases based on the training samples. The most popular learning algorithm used is back-propagation algorithm which uses gradient decent based approach to calculate the minimum of a function.

Decision Trees (DT) Decision trees uses tree-like structure to model

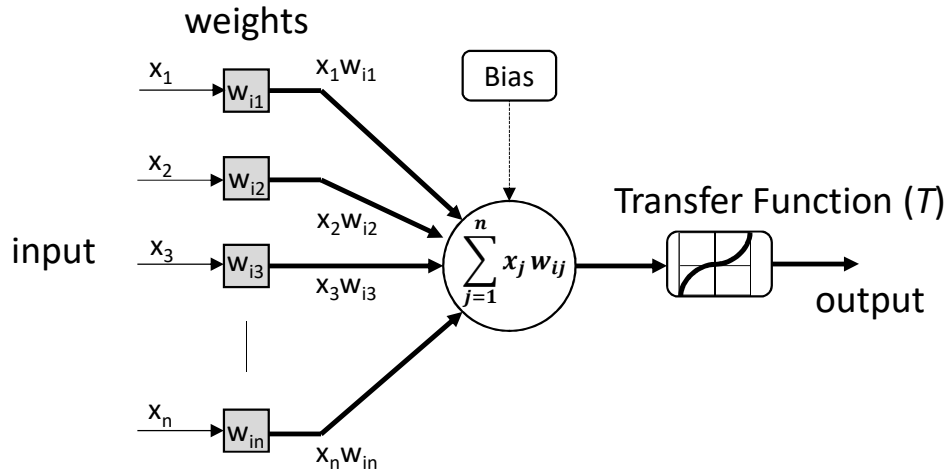


Figure 2.21: Structure of an ANN neuron consisting of aggregation function and a transfer function. Adopted from Dua and Du (2016)

the classification problem with leaves representing the decisions or class and branches representing the partitions based on the input features as shown in Figure 2.22. Given a training dataset $\mathbf{D} = \{x_i y_i\}_{i=1}^{n}$ containing n points in a d-dimensional data space $\mathcal{R}$, then y_i represents the output class for the corresponding input point x_i. A DT algorithm will recursively partition the input points x_i using axis-parallel hyper-plane to build the decision tree model that splits the data space $\mathcal{R}$ into half-spaces (Zaki, Meira Jr, & Meira, 2014). In terms of data space, the internal or intermediate nodes recursively divide the hyper-plane in half-spaces and the leaf nodes represent the final regions in the data space representing the majority class(Zaki et al., 2014). The task of the developed model will be to classify the *test* point by recursively evaluating the correct half-space it belongs to until the leaf node is reached in the DT.

A hyper-plane defines the decision boundary or split point as it divides the space into two sub spaces. DT uses the entropy measure as the split point criterion that provides the best separation between class labels. Hence a lower entropy is obtained if all the points in the partition belong to the same class and higher entropy reported if points in the partition belong to different class. A dataset containing k class labels where the output $y_i = \{c_1, c_2, ..., c_k\}$, then entropy of a partitions is given by:

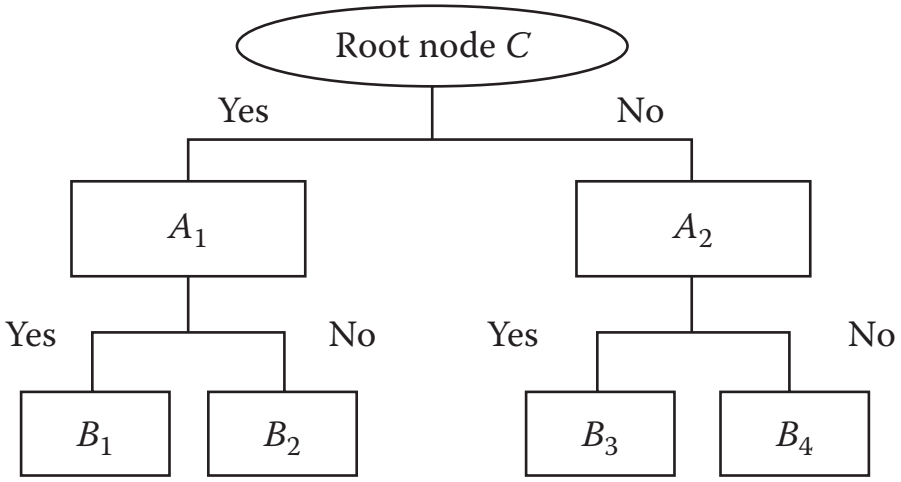


Figure 2.22: Sample structure of a DT consisting of leaf nodes and root node. Adopted from Dua and Du (2016)

$$H(\mathbf{D}) = -\sum_{i=1}^{k} P(c_i|\mathbf{D})\, log_2\, P(c_i|\mathbf{D}) \tag{2.3}$$

The split point is selected to partition a set of points $\mathbf{D}$ into sub partitions $\mathbf{D_Y}$ and $\mathbf{D_N}$ in such a way as to reduce the over all entropy which can be defined as the information gain given by:

$$Gain((D),(D_Y),(D_N)) = H(\mathbf{D}) - H(\mathbf{D_Y}, \mathbf{D_Y}) \tag{2.4}$$

The recursive partitioning techniques used in DT makes the model easy to interpret. The IF-ELSE type of rules make the decision rules easy to follow and can handle multi-type attributes.

Bayes Classifier Bayes classifiers are a type of probabilistic classifiers that use Bayes theorem to predict the output class of a given input. The main task is to maximise the posterior probability by estimating the joint probability density function of each class.

Given a dataset $\mathbf{D}$ containing n instances x_i and y_i is the output class with k labels such that $y_i = \{c_1, c_2, ..., c_k\}$ then according of Bayes rule, the posterior probability for each class c_i is given by:

$$P(c_i|\mathbf{x}) = \frac{P(\mathbf{x}|c_i)\,.\,P(c_i)}{P(\mathbf{x})} \tag{2.5}$$

where $P(c_i)$ is the prior probability and $P(\mathbf{x}|c_i)$ represents the conditional probability of x given the likelihood c_i. The probability of observing x from any class k is given by:

$$P(x) = \sum_{j=1}^{k} P(\mathbf{x}|c_j).P(c_j) \tag{2.6}$$

Naive Bayes classifier is one of the simple forms of Bayes classifier that assumes that the attributes are independent. Hence, the Naive Bayes classifier can be solved by maximum posterior probability map as :

$$arg\max_{c_j \in C} P(\mathbf{x}|c_j).P(c_j) \tag{2.7}$$

Bayes classifiers are efficient in inference learning tasks. Specifically, Naive Bayes classifier can be trained quickly as the attribute independence assumption reduces the complexity high-dimensional data (Zaki et al., 2014). In addition the technique produces good results even when assumption of attribute independence is violated (Dua & Du, 2016).

Support Vector Machines (SVM) SVM classifiers try to separate the data points belonging to classes using a hyperplane that maximises the distance between the closest data point in either class. The maximum margin or the hyperplane separation is obtained by solving a quadratic optimisation problem. SVMs have better generalisation ability and can obtain the classification boundary quickly and accurately even with a small and high dimensional training sample. However, SVMs are good in binary classification problems and require multiple binary class classifier combinations in a multi-class classification problem (Dua & Du, 2016).

The ML methods discussed in this section establishes that a right combination of good domain specific features and efficient ML algorithm trained and tested on real datasets is essential to build a robust attack detection system. However, the use of a particular ML algorithm for intrusion detection requires knowing the frequency of training the models and the availability of labelled datasets (Buczak & Guven, 2015). This because the intrusion detection systems require re-training when new intrusions occur and new attack patterns are discovered. Hence models that support incremental training are more suitable for intrusion detection. In addition, models that can be better interpreted are more suitable in building fine grained access policies

compared to black-box models which are difficult to interpret and extract policy rules (Buczak & Guven, 2015). Moreover, selecting the most suitable ML algorithm for the problem at hand is a challenging task and often dependent on the accuracy of the model in detecting complex attack data with minor variations, time it takes to classify a new instance and the scalability of the solution in working with large datasets(Buczak & Guven, 2015).

The following sections discuss about existing works done on attacks on publish/subscribe systems and their detection techniques.

2.6 Attacks on Publish/Subscribe Systems

Most of the attack detection techniques discussed thus far focus on the HTTP protocol which uses a request/response paradigm. The reasons for the HTTP protocol receiving a lot of focus from researchers is due its wide usage on the Internet and HTTP being one of the most targeted protocols during cyber attacks (M. Singh et al., 2015; Praseed & Thilagam, 2018; Gupta & Badve, 2017). However, as IoT deployments increase, IoT based protocols are likely to have a significant presence on the Internet, becoming new targets for potential cyber attacks (Metongnon & Sadre, 2018). Wun et al. (2007) presented a taxonomy of DoS attacks on Content-based Publish/Subscribe Systems (CPSS). According to Wun et al. (2007) both broker and clients can become targets of DoS attacks, which can be flooding based or amplification based attacks. The authors also experimentally evaluated the impact of DoS attacks on the system and identified that Application Layer DoS attacks in CPSS can be achieved using relatively low volumes of attack traffic. In another work related to publish/subscribe attacks, the authors Srivatsa and Liu (2005) proposed a signature based EventGaurd system to protect publish/subscribe overlay systems from various types of attacks including DoS attacks. One of the issues with such systems is the complexity introduced by cryptography techniques and their implementation in constrained IoT devices. Most recent work in detecting DoS attacks in publish/subscribe systems presented by Maresca (2017) uses threshold based detection methods to differentiate between normal and attack traffic. Simply relying on volume based thresholds and statistical techniques can increase false positive rates (Praseed & Thilagam, 2018; Kaur et al., 2017), hence a more robust technique that analyses various parameters of the Application Layer protocol are required. In addition, these techniques also do not identify the various protocol features that can be effectively utilised to detect DoS attacks and other attacks on publish/subscribe protocols.

2.7 MQTT Protocol

MQTT (formerly referred to as MQ Telemetry Transport (HiveMQ, 2015)) was developed by Andy Stanford-Clark of IBM and Arlen Nipper of Arcom in 1999 (Al-Fuqaha et al., 2015; Banks & Gupta, 2014). It was originally developed for communication between remote devices with unreliable communication links (Locke, 2010; Banks & Gupta, 2014), especially in monitoring oil pipelines via satellite communication. It was standardised by OASIS (Locke, 2010) in 2013. Even though MQTT was originally designed for remote site communication, it was adopted for IoT applications due to its simple model and low bandwidth usage (Niruntasukrat et al., 2016). Some of the important features of MQTT that makes it suitable for IoT devices are (Al-Fuqaha et al., 2015; Locke, 2010; Niruntasukrat et al., 2016; Waher, 2016):

1. Simple implementation.
2. One to Many communication model.
3. Consumes very low bandwidth (2kb header).
4. Three levels of QoS (Quality of Service).
5. Client/Server model between clients and the broker.

The following sections will discuss the messaging pattern, client connectivity, publish subscribe process and quality of services levels adopted in MQTT protocol.

2.7.1 Publish/Subscribe Pattern

Communication patterns define the mechanism by which messages are exchanged between the end-points of a system. Traditional request/response approach uses a synchronous communication model where the client or peers request information from a server, which then responds with the requested information. In contrast, an event-driven communication model uses an asynchronous communication model to send updates when events occur. Publish/subscribe is an asynchronous event-driven communication model that decouples the data producers and consumers to facilitate mass distribution of messages to data consumers. The MQTT protocol depends on the publish/subscribe communication pattern and uses a broker to facilitate message exchange between end-points. There are three main roles in

the MQTT architecture: publisher, subscriber and the broker as shown in Figure 2.23.

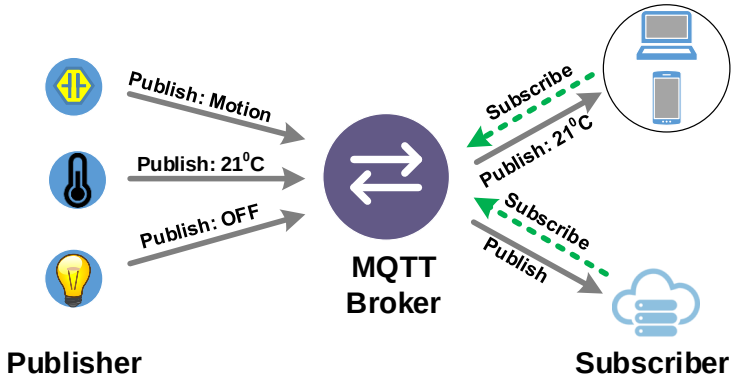


Figure 2.23: MQTT protocol architecture

The publishers generate data and publish their data to the message broker. The role of the message broker is to filter the published messages and disseminate the data correctly to subscribers (Al-Fuqaha et al., 2015; Locke, 2010). In an IoT scenario, the publishers are various low power sensors publishing data like temperature, pressure, humidity and subscribers could be smart devices like home automation controller, smart meter, smartphone, computer who can take actions based on the environment changes (Salman & Jain, 2015). The main advantages of the publish/subscribe communication pattern are (Eugster, Felber, Guerraoui, & Kermarrec, 2003):

- **Space decoupling**: the publishers and subscribers need not be aware of each other i.e., publishers do not keep references of the subscribers (IPaddress, port etc.). In addition, they can be located behind a firewall with only the MQTT broker publicly available (Waher, 2015),
- **Time decoupling**: publishers and subscribers need not be running at the same time
- **Synchronisation decoupling**: end-point operations are not blocked when publishing or receiving events.

In addition to decoupling the publishers and subscribers, MQTT also supports message level service agreements between the sender and receiver of the messages called the Quality of Service (QoS) level. The QoS agreement defines the guarantee of message delivery to the clients. MQTT supports

the following three types of quality of service for message delivery (Banks & Gupta, 2014):

1. At most once: message delivery uses best efforts of underlying TCP/IP protocol. Message loss and duplicate messages might occur.
2. At least once: message delivery is assured but might include duplicates
3. Exactly once: message delivery is assured exactly once. Duplicates and message loss is not acceptable.

The QoS feature in MQTT allows the client to set message delivery agreement according to its network reliability and importance of the message. These features facilitates scalable deployment architectures necessary for larger scale IoT deployments.

2.7.2 MQTT Connection Establishment

The MQTT protocol requires clients to first connect to the broker before they can publish or subscribe messages. The broker is responsible for authentication, authorisation, message filtering and message distribution. Clients include both publishers and subscribers that utilise MQTT libraries to exchange messages with the broker over the network. MQTT protocol defines a series of CONTROL packets that are used by the clients and broker to establish and exchange messages. The MQTT control packet consists of a fixed two byte header, a variable header and a payload in some control packets as show in Figure 2.24.

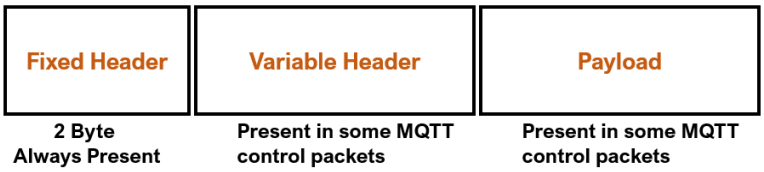


Figure 2.24: MQTT control packet structure

The fixed header identifies the control packet type and its related flag fields. Some of the important control packets are enumerated in Table 2.8. A complete list of the message types are listed in standards definition (Andrew Banks, 2014).

MQTT protocol runs over the TCP/IP stack and hence requires a complete TCP connection establishment process prior to message exchange as

Table 2.8: MQTT control packets

Control Packet	**Description**	value
CONNECT	First packet sent by the client to initiate a MQTT connection	1
CONNACK	Acknowledgement for a CONNECT packet sent by broker to the client	2
PUBLISH	Message sent by the publisher to publish a message to a Topic	3
SUBSCRIBE	Message sent by the subscriber to subscribe to a Topic	8
DISCONNECT	Message sent by the client to disconnect the MQTT connection	14

shown in Figure 2.25. Once the TCP session is established, the first control packet sent by the client is CONNECT packet which is acknowledged by the broker with a CONNACK packet. A client can only send one CONNECT packet in a single TCP session and sending multiple CONNECT packets causes the session to be terminated.

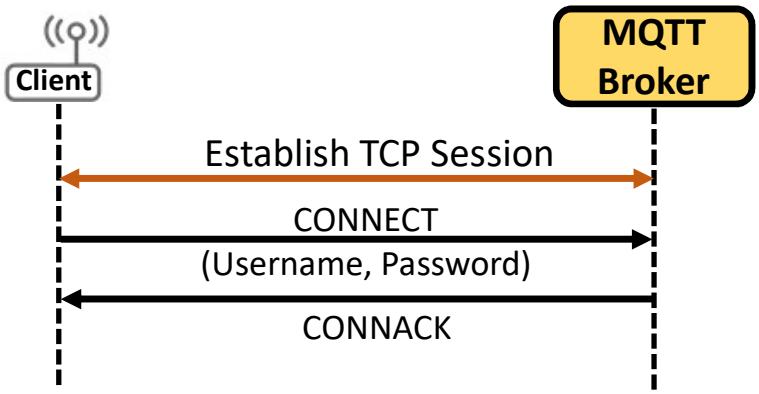


Figure 2.25: MQTT client connection process

The CONNECT control packet contains essential client information which is used by the broker to grant access to the client. The fixed header part of the control packet contains the flag bits and the related field contents are located in the payload. Table 2.9 lists the flag and payload fields of the

CONNECT packet.

Table 2.9: CONNECT packet fields

Client Information	**Description**	**Flag / Payload field**
clientID	Unique Client Identifier	Payload
CleanSession	Session state information	Flag
Username	Username used in client authentication	Flag + Payload
Password	Password used in client authentication	Flag + Payload
lastWill	Defines if last WILL message is enabled for the client in the event of abnormal network termination	Flag
lastWillTopic	Defines the last WILL topic to which last WILL messages are to be published	Payload
lastWillQoS	Last WILL message QoS level	Flag
lastWillMessage	Last WILL message	Payload
lastWillRetain	Last WILL message retention state	Flag
KeepAlive	Defines the maximum time interval in seconds permitted to elapse between successive control packets	Payload

clientID A client identifier is used by the broker to distinguish clients and maintain states for each client. MQTT only permits UTF-8 (unicode transformation format) encoded strings to be used as client identifier and it is the first field in the CONNECT packet payload.

CleanSession The clean session flag is used by the client to establish persistent connection with the broker. If the CleanSession flag is set to 0 the server needs to resume the client connection state from stored sessions identified by the clientID. If the CleanSession flag is set to 1 then server discards any previous sessions and a new session is started. The session information stored by the server includes: client subscriptions and messages published with higher quality of service.

Username/Password MQTT uses credentials to authenticate and authorise clients. The credentials are sent in plain text if the session is not encrypted. The use of username and password is indicated using the flag fields and the actual credentials are added to the payload by clients. The username and password fields can be 0 to 65,535 bytes long.

Last WILL MQTT protocol supports Last WILL and Testament (LWT) feature which allows a client to set a WILL message to notify other clients when it disconnects from broker ungracefully. The Last WILL message is set in the CONNECT control packet by setting the WILL flag and WILL QoS flag; while the actual WILL message and topic are added to the payload. The server is required to store the WILL message if it accepts the CONNECT request. The WILL message is sent by the broker on behalf of the client in situations such as (Banks & Gupta, 2014):

- I/O error or network failure detected by the server
- no communication detected from the client with in the KeepAlive interval
- connection termination by the client without sending the DISCONNECT control packet
- server terminating the connection due to protocol errors

KeepAlive It is a time interval in seconds that specifies the maximum permitted time interval between two successive control packets. The client sets the KeepAlive interval based on the application requirement using the KeepAlive bits. The protocol necessitates that the client ensures that the time interval between two successive control packets does not exceed the KeepAlive interval. If there are no control packets to send, the client needs to send a PINGREQ packet to keep the network connection active. The server terminates the network connection if no packets are received within one and half times the KeepAlive interval. The maximum allowed KeepAlive interval as per the MQTT 3.1.1 specification (Banks & Gupta, 2014) is 18 hours 12 minutes and 15 seconds

Broker Response

The broker responds to the connect request with acknowledgement packet known as CONNACK. The CONNACK control packet contains the return code indicating the outcome of the connection attempt and session present flag which indicates to the client of previously stored sessions. The various return codes sent by the broker are listed in Table 2.10.

Table 2.10: Response codes sent by broker in CONNACK packet

Return Code	Return Code Response
0	Connection accepted
1	Connection refused, unacceptable protocol version
2	Connection refused, identifier rejected
3	Connection refused, server unavailable
4	Connection refused, bad user name or password
5	Connection refused, not authorised

2.7.3 MQTT - Message Publish

After a successful connection the publishing MQTT clients can publish messages. The publish/subscribe messaging pattern employs two common forms of filtering: content-based and topic-based. In content-based filtering, the subscribers receive only those messages that contain or match the attributes defined by the subscribers, whereas for topic-based filtering, subscribers receive only a subset of published messages that match the message topics on logical channels subscribed by them. The MQTT protocol does topic-based filtering to route messages to interested subscribers. Hence every message published by clients must contain topic based on which the broker routes the messages to interested clients. PUBLISH control packet is either sent by a client to a broker or broker to a client as illustrated in Figure 2.26.

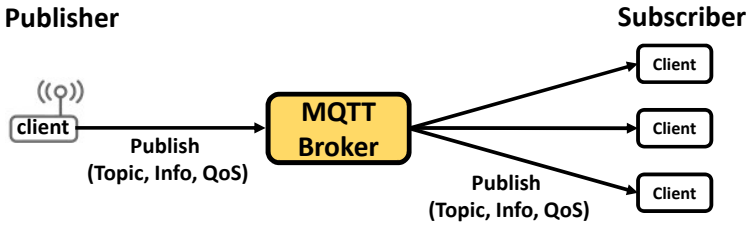


Figure 2.26: MQTT client publish process

The PUBLISH control packet contains a fixed header, variable header and a payload. The fixed header contains the following flag fields:

- DUP Flag: This flag indicates whether its is the first attempt or a re-delivery attempt to send the PUBLISH control packet. A value of

0 indicates that it is the first attempt by a client to send the control packet. In contrast, a value of 1 indicates client is attempting a re-delivery of a an earlier try.

- QoS Level: This field indicates the level guarantee required to send the application messages to the intended recipients. MQTT supports three levels of QoS: At most once (0), At least once (1) and Exactly once (2) delivery of messages.

- RETAIN: This flag field is used to indicate whether server should store the message and its associated QoS to be delivered to future subscribers of the topic.

The variable header field of the PUBLISH control packet consists of the following fields:

- Topic Name: As MQTT protocol uses topic based message filtering, publish messages must contain the topic name based on which messages are forwarded to intended recipients. The topic name in the PUBLISH packet cannot contain wildcard characters and must be a UTF-8 encoded string. The topic name consists of one or more topic levels separated by forward slash as shown in Figure 2.27.

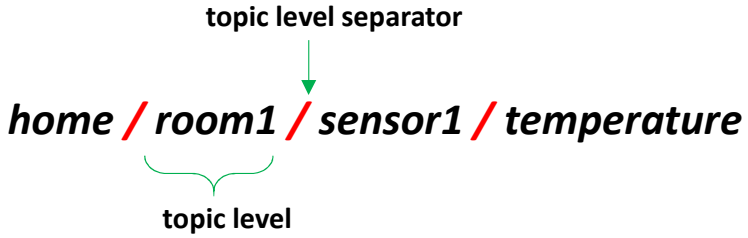


Figure 2.27: MQTT topic structure with topic levels separated by topic separator

- Packet Identifier: This field is used to identify packets when QoS1 or QoS2 is used to publish messages.

The MQTT PUBLISH packet payload contains the actual application message and can be used to send different types of data. A publish packet can contain zero length payload or a length that fits into a maximum packet size of 256 MB.

In MQTT, both the client and server can act as a sender or receiver of the publish request. The broker uses the topic name set in the publish packet to filter the subscribed clients in order to forward the messages. The two directions in which the message publishing occurs is:

- Messages published from a publishing client to the broker
- Messages published from broker to a subscribing client

Once the publish packet is received, the acknowledgement of the request depends on the QoS level set. The publishing client defines the QoS with which it publishes a message to a broker and a subscribing client defines the QoS (sQoS) with which it can receive a message from a broker. MQTT protocol supports three levels of QoS are described in 2.11.

Table 2.11: MQTT QoS types

QoS Level	**Message Guarantee**	**Behaviour**
QoS0	At most once	Messages sent to all subscribers once, no retries and no acknowledgement from receivers
QoS1	At least once	Messages sent to all subscribers at least once and is acknowledged by receivers
QoS2	Exactly once	Messages sent to all subscribers exactly once, no duplicates, extra acknowledgement messages to avoid duplicate messages and guaranteed message delivery

Based on the level of QoS various control packets are used in message exchange and are discussed in detail in the following sections.

QoS0 - At most once delivery In this QoS level the reliability of message delivery is only up to the level of underlying TCP/IP capabilities. It is also known as "Fire and Forget" scheme where the receiver does not acknowledge the receipt of PUBLISH packet and the sender does not store the message and attempt to retry message delivery. The publish process is illustrated in Figure 2.28. The receiver either receives the message once or doesn't receive at all.

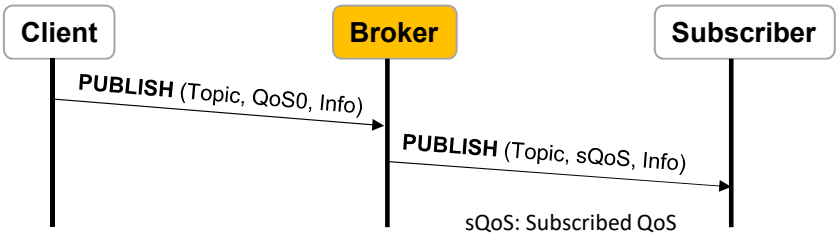


Figure 2.28: MQTT QoS0 publish process

QoS1 - At least once delivery In this QoS level the protocol ensures the messages are delivered at least once at the receiver. This is achieved by using a packet identifier and storing the message until the sender receives a PUBACK acknowledgement from the receiver. The sender deletes the message and releases the packet identifier once it receives a PUBACK packet for the QoS1 packet. The receiver can receive the message multiple times.

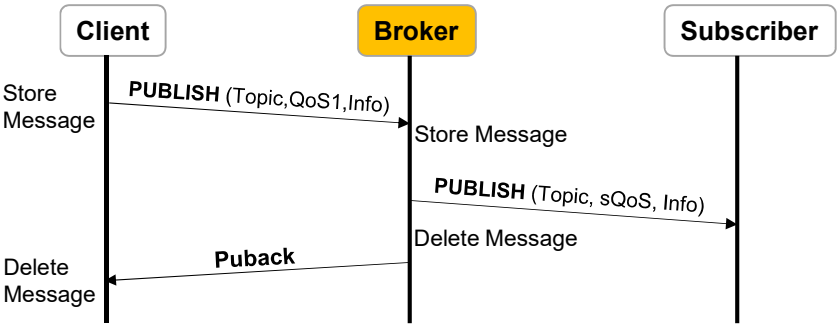


Figure 2.29: MQTT QoS1 publish process

QoS2-Exactly once delivery The highest QoS provided in MQTT protocol ensures an exactly once delivery of the message to the receiver. To ensure an exactly once delivery of messages, a two-step acknowledgement process is utilised by the receiver and sender. The client stores the message and sends the publish packet with a unique packet identifier which is acknowledged by the receiver with a PUBREC (Publish Received). The sender then discards the PUBLISH message and sends a PUBREL (Publish Release) packet. Once the PUBREL message is received the receiver now discards the message along with its state and sends a PUBCOMP (Publish Complete) message to sender. Upon receiving PUBCOMP the sender safely

discards the message and releases the packet identifier to be reused. The acknowledgement process is illustrated in Figure 2.30.

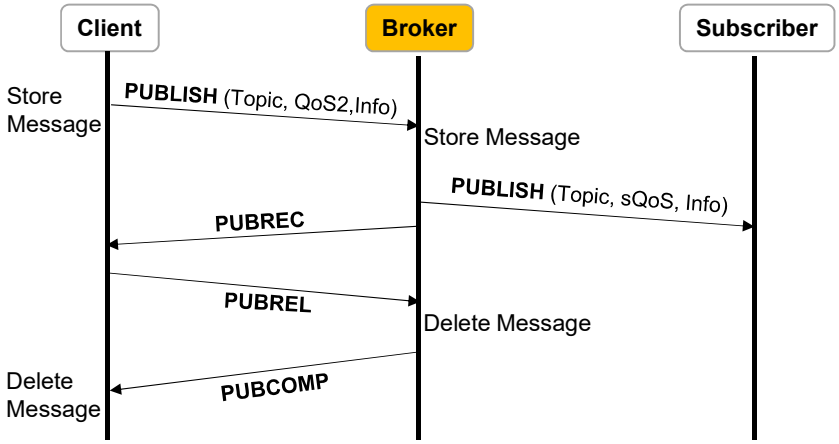


Figure 2.30: MQTT QoS2 publish process

2.7.4 MQTT - Message Subscribe

Message subscription feature allows the MQTT clients to indicate their interest in receiving messages published to topics. The indication of interest in a topic is achieved by sending a SUBSCRIBE control packet by a client to a broker. The broker then sends a PUBLISH packets to the subscribed clients to deliver the application messages published to the subscribed topics. A SUBSCRIBE control packet contains packet identifier, one or more topic filters and a QoS level with which the client intends to receive messages. Figure 2.31 shows the subscribe process in MQTT.

MQTT allows clients to subscribe to multiple topics and this is achieved by either using wildcard characters in subscription to topics or by adding multiple topic and QoS pairs in the SUBSCRIBE control packet. The two wildcard characters supported in the topic subscriptions are:

- Single-Level "+": this wildcard is used to subscribe to a single level topic hierarchy and is used between topic delimiters. An example of a single-level wildcards is as follows:
  - **home/room1/+/temperature** - this would match all the following topics:

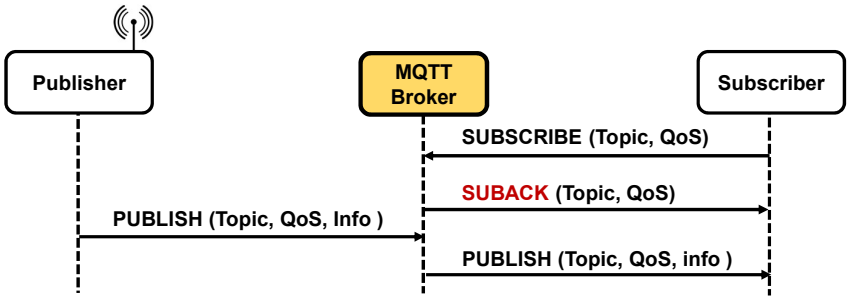


Figure 2.31: MQTT Client subscribe-publish process

1. home/room1/sensor1/temperature
2. home/room1/sensor2/temperature
3. home/room1/sensor3/temperature

- Multi-Level "#": this wildcard is used to subscribe to a all levels of the topic hierarchy and hence it must occur at the end of the topic string. An example of a multi-level wildcard is as follows:
  - **home/room1/#** - this would match all the following topics:
    1. home/room1/sensor1/temperature
    2. home/room1/sensor2/humidity
    3. home/room1/bulb/state

The second method to subscribe to multiple topics is by adding multiple topic and QoS pairs to the SUBSCRIBE control packet payload as shown in Figure 2.32.

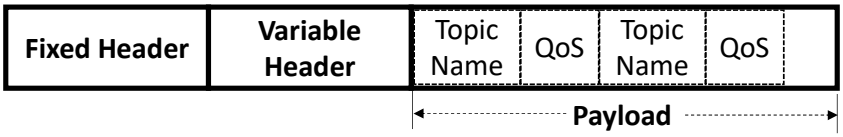


Figure 2.32: MQTT SUBSCRIBE control packet structure with multiple subscriptions

The subscription request is acknowledged by the broker by sending a SUBACK control packet to the client. The SUBACK packet contains the return code for each topic/QoS pair contained in the SUBSCRIBE control packet.

2.8 MQTT Security

MQTT standards specifies a list of threats that needs to be considered by solution providers. Some of the threats mentioned by MQTT Oasis standards (Banks & Gupta, 2014) are:

- Devices can be compromised,
- Unauthorised access of data in clients and servers,
- Attacks targeting the protocol behaviour,
- DoS attacks, and
- MitM attacks trying to intercept, alter or reroute communication and inject spoofed control packets.

According to Collina, Corazza, and Vanelli-Coralli (2012), security was not given important consideration when designing the MQTT protocol. Even though MQTT supports authentication it sends credentials in plain text and lacks dynamic ACLs updates to block malicious publishers and subscribers (Collina et al., 2012). In order to tackle these security issues the OASIS standard suggests the following security measures that can be employed by developers when building MQTT based applications:

1. Use of Secure MQTT (TLS version of MQTT),
2. Authentication of users and devices,
3. Client Authorisation,
4. Control packet and application data integrity check, and
5. Control packet and application data privacy check.

Even tough encrypted version of MQTT has been defined by the OASIS standard, the use of encryption in constrained IoT devices is a challenge (HiveMQ, 2015; M. Singh et al., 2015). Some of the additional security mechanisms that have been proposed to secure communication between clients and broker are (HiveMQ, 2015; M. Singh et al., 2015; Banks & Gupta, 2014): use of VPN to encrypt the sessions between client and server, use of client certificates for authentication, payload encryption, firewall configuration only to allow traffic on port 1883 (MQTT) and 8883 (MQTT-SSL), Load Balancer to distribute load to multiple MQTT brokers, use of DMZ to

prevent hackers from getting access to other enterprise systems, throttling MQTT clients and setting limits to MQTT message size. Even though applying such countermeasures is beneficial in reducing the impact of cyber attacks, the IoT devices using MQTT protocol to communicate and send control messages will be at risk due to the vulnerabilities that exist in the protocol. Hence it is necessary to study and understand the MQTT protocol behaviour and attack patterns targeting such application protocols. Most of the works related to MQTT protocol available in the literature focus on :

- Performance evaluation (Scalagent, 2015; Lee, Kim, k. Hong, & Ju, 2013; Thangavel, Ma, Valera, Tan, & Tan, 2014; Luzuriaga et al., 2015; Yokotani & Sasaki, 2016; Gündoğan et al., 2018; Fehrenbach, 2017),
- Proposing security enhancements to the existing protocol (M. Singh et al., 2015; Mektoubi, Hassani, Belhadaoui, Rifi, & Zakari, 2016; Shin, Kobara, Chuang, & Huang, 2016),
- Formal modelling (Houimli, Kahloul, & Benaoun, 2017; Aziz, 2016), and
- Security evaluation (Perrone, Vecchio, Pecori, & Giaffreda, 2017; Firdous et al., 2017; Andy, Rahardjo, & Hanindhito, 2017).

The security enhancement measures proposed in (M. Singh et al., 2015; Mektoubi et al., 2016; Shin et al., 2016), comprise of securing the data communication by using encryption techniques for constrained devices. In contrast, our work focuses on building an MQTT attack detection framework which can be used to develop an Intrusion Detection System (IDS) to detect attacks.

The various performance evaluation methods proposed in (Scalagent, 2015; Lee et al., 2013; Thangavel et al., 2014; Luzuriaga et al., 2015; Yokotani & Sasaki, 2016; Gündoğan et al., 2018; Fehrenbach, 2017) do not evaluate the broker performance during the DoS attacks. This aim of our work is to model DoS attacks scenarios that target the authentication and authorisation techniques of MQTT protocol and identify their impact on various MQTT brokers and deployment scenarios.

Other works on MQTT protocol attempts to present a formal model and security analysis of the protocol. Aziz (2016), presented a formal model of MQTT protocol based on message passing process algebra. The author analysed the various message QoS levels against the standards and identified that the QoS-2 message model was prone to errors. Perrone et al. (2017)

presented a security analysis of the MQTT protocol and described the various security requirements for IoT deployments. In another work, Andy et al. (2017) presented some attack scenarios as well as a security analysis of the MQTT protocol. In their work, the authors highlighted the security issues of the MQTT protocol and discussed attack scenarios against brokers with open authentication. Feasibility of such attacks is questionable as most MQTT broker deployments in industrial environment disable open authentication feature as it poses security risk of unauthorised access. In order to understand the security issues in the MQTT protocol a threat model and the impact of SYN-Flood DoS attack on message brokers was presented in (Firdous et al., 2017).

Santiago Hernández Ramos and Lacuesta (2018) proposed a fuzzing approach to test vulnerabilities of an MQTT based application. The proposed approach tested the behaviour of the MQTT based application when fuzzed data was inserted between clients and the broker. The authors used a proxy fuzzing technique along with a non-normative packet variable header data template to asses the behaviour of both broker and clients when presented with unexpected data. Failures were detected in certain versions of the broker software and in client applications. A similar MQTT fuzzer tool known as F-secure MQTT-FUZZ was developed by (Vähä-Sipilä, 2015) which uses sniffed raw MQTT control packet payload to launch fuzzed MQTT packets against the broker.

2.8.1 MQTT Attack Detection

The main requirements for building a ML based MQTT attack detection system is the availability of normal and attack traffic datasets of the protocol and identifying features that can distinguish between the two traffic classes. A work that attempts to identify IoT based attacks using MQTT transaction based features was proposed by Moustafa et al. (2019). However, these features are based on the TCP protocol analysis, which do not provide sufficient information on the MQTT protocol parameters. Hence these features will not be effective in differentiating between the types of MQTT attacks evaluated in our work. In addition, the main drawback of (Moustafa et al., 2019) is that the performance of their attack detection scheme was not presented for MQTT attacks. The primary reason behind this was that no real MQTT attack datasets existed to evaluate the detection technique. This necessitates that new datasets be generated which contain the normal and attack MQTT traffic and new features be defined based on protocol analysis to effectively detect attacks against protocol based applications.

Table 2.12: Comparison of existing MQTT attack model and attack detection features with that proposed in this thesis

	Existing	**Proposed**
Attack Modelling		
Attack Technique		
Control Packet Flooding	✓	✓
Application Layer Control Packets		
MQTT-CONNECT Flooding	–	✓
Malformed MQTT Packet Flooding	–	✓
MQTT-WILL Payload Flooding	–	✓
MQTT-SUBSCRIBE Flooding	–	✓
Features for Attack Detection		
Statistical Features		
Flow, size, duration, volume	✓	✓
Network-Layer Features		
IP (Source, Destination)	✓	✓
TCP Ports	✓	✓
TCP Flags	✓	–
Application-Layer Features		
MQTT packet Length	✓	✓
MQTT Field length (clientID, username, password, Topic, Last WILL Payload length)	–	✓
MQTT Flags (KeepAlive, CleanSession)	–	✓
Control Packet Volume (CONNECT, PUBLISH, SUBSCRIBE, DISCONNECT, PING)	–	✓
MQTT QoS Fields	–	✓

2.9 Summary

IoT is a new paradigm of the technology world and is proving to be a game changer in improving the lives of people. However, every technology has its pros and cons which are critical factors to its survival and growth. The proliferation of smart devices in everyday lives of people has attracted attention from adversaries. Due to this reason there is an urgent need to identify the adversaries and their behaviour to build protection tools to safeguard the IoT. As adoption rate for MQTT protocol in building IoT based applications increases, the attacks targeting the IoT Application Layer protocol are predicted to increase. Hence such targeted attacks need to be detected and

solutions need to be built to prevent such attacks, especially DoS attacks.

In this chapter the various enabling technologies of IoT were presented along with the IoT security threats. A detailed discussion on MQTT protocol and its security were presented to identify potential threats from adversaries, specially related to DoS attacks. This discussion was followed by classifying the various DoS attack techniques that have been identified in the literature which indicate that sophisticated and targeted Application Layer DoS attacks are on the rise. Such attacks can cause serious impact to the sensitive applications relying on the MQTT protocol and hence, its vulnerabilities need to be identified and detection systems need to be developed. The various DoS detection techniques that have been discussed in the literature indicate that ML techniques are being extensively used for their wide range of benefits over statistical and signature based detection techniques. However, ML techniques rely on features that can be effective in classifying normal and attack traffic. The domain and protocol specific features presented in the literature cannot be extended to other protocols and domains. Hence, new features that can effectively detect attacks in protocols such as MQTT are required. In addition, DoS detection evaluation datasets are either old or lack IoT or MQTT related traffic and are no longer relevant to validate the techniques that are built to detect attacks in IoT traffic. So, new normal and attack traffic datasets need to be collected with relevant MQTT protocol specific attack traffic to model effective attack detection techniques. Table 2.13 summarises the literature with regards to the proposed work in this research.

Table 2.13: Comparison of research contributions with existing and proposed study

Research Contribution	**Existing and Proposed Studies**					
Threat modelling	IoT: (Atamli & Martin, 2014)	IoT-MQTT: **This work**				
Application Layer DoS attack modelling	HTTP: (K. Singh et al., 2017), (Adi et al., 2016), (Ranjan et al., 2009), Shan, Wang, and Pu (2017b)	SIP: (Rafique et al., 2009), (Luo et al., 2008)	SMTP: (Bencsath & Ronai, 2007)	DNS: (Ballani & Francis, 2008)	MQTT: **This work**	
IoT attack dataset	IoT-Botnet: (Koroniotis et al., 2019)	UNSW-NB15: (Moustafa & Slay, 2015)	MQTT IoT Dataset: **This work**			
MQTT Security	Performance evaluation: (Scalagent, 2015), (Lee et al., 2013), (Thangavel et al., 2014), (Luzuriaga et al., 2015), (Yokotani & Sasaki, 2016), (Gündoğan et al., 2018), (Fehrenbach, 2017)	Proposing security enhancements to the existing protocol: (M. Singh et al., 2015), (Mektoubi et al., 2016), (Shin et al., 2016)	Formal modelling: (Houimli et al., 2017; Aziz, 2016)	Security evaluation: (Perrone et al., 2017), (Firdous et al., 2017), (Andy et al., 2017), (Santiago Hernández Ramos & Lacuesta, 2018)	DoS attacks on publish\subscribe (Wun et al., 2007)	MQTT DoS Attacks: **This work**
MQTT attack detection features	MQTT Transaction features: Moustafa et al. (2019)	MQTT features and attack detection **This work**				

Hence, this research is focused on the following gaps that were not addressed thus far in the literature:

1. Identifying MQTT protocol vulnerabilities and designing DoS attack scenarios (Section 4.2)
2. Evaluate the DoS attack impact (Section 5.1)
3. Build realistic IoT test-bed to generate IoT traffic (Section 4.4)
4. Collect normal traffic and MQTT DoS scenarios to collect attack traffic (Section 4.4.1)
5. Extract novel MQTT features (Section 4.4.2)
6. Evaluate ML classifiers using proposed features for their attack detection performance (Section 5.3)

In the following chapter, the research methodology is presented along with the steps followed in this research and variables identified to evaluate and test the proposed scheme. Chapter 4 discusses the MQTT threat model, MQTT DoS attack methodology, IoT experimental test-bed and MQTT attack detection framework. Chapter 5 discuses the results obtained for DoS attack impact measurements and evaluations of MQTT attack detection framework. Chapter 6 presents the discussion of the results and Chapter 7 presents the conclusion and future work.

Chapter 3

Research Methodology and Design

This chapter describes the research approach, research design and research methods used in this work to answer the proposed research question. Furthermore, the research variables, experimental procedures, data analysis methods, threats to validity and the instruments used in this work are also elaborated.

3.1 Research Methodology

According to Creswell and Creswell (2017), identifying the philosophical ideas held by the researcher is the first step in the research process. He states that philosophical ideas influence the research practice and provides the context as to why a specific research approach was chosen. Creswell uses the term "worldview" to define the philosophical views or beliefs held by the researcher that controls the action or approach in conducting the research (Creswell & Creswell, 2017). The term paradigm is also used in the research domain to describe the worldviews held by the researcher (Mackenzie & Knipe, 2006). Paradigms define the beliefs that the researcher espouses and how such beliefs shape the research methods used to collect and interpret the data (Guba & Lincoln, 1985). According to Guba and Lincoln (1985), the basic set of beliefs, assumptions and values regarding a particular paradigm can be described using the positions held by the researcher with regards to ontology, epistemology, methodology and axiology about the knowledge being researched.

Ontology is concerned with the assumptions that one has regarding the

nature of existence or reality. The underlying beliefs held by the researcher can be answered by posing the questions such as: *What is the nature of reality?* or *When do we consider something to be real?* Epistemology focuses on the nature of knowledge, how it can be collected and how it can be communicated (Kivunja & Kuyini, 2017). It can be answered by posing questions such as: *what is the relationship between researcher and what is being researched?* Methodology broadly refers to the logical process followed in conducting a research to answer the research problem. Axiology refers to the values that the researcher brings to the research project and identifies the stance that needs to be adopted by the researcher regarding fairness, moral issues, ethical issues and rights of the participants. In the literature a large number of paradigms have been proposed, however they can be broadly classified into positivism, Interpretivism / Constructivism and Pragmatism (Creswell & Creswell, 2017; Williamson, 2018; Kivunja & Kuyini, 2017).

Creswell and Creswell (2017), states that the paradigm or the worldview chosen by the researcher influences the research approach adopted in conducting the research. The three research approaches defined by Creswell and Creswell (2017) are: quantitative, qualitative and mixed methods. A quantitative research approach tries to validate objective theories by identifying the relationships between variables. These variables can be measured and quantified, and can be further analysed using statistical techniques. During the quantitative research process, the researcher usually tries to deductively verify generalised theory by breaking it down into specific hypothesis which can be tested in an unbiased setting. This is also referred to as top-down approach (Creswell & Clark, 2017). In contrast, the qualitative research aims to explore and subjectively explain the social problems by collecting data on human experiences with their surroundings. Such a research approach tends to use an inductive logic (bottom-up) to observe specific patterns to produce generalisations. The mixed method approach uses both quantitative and qualitative methods to refine and produce a complete or deeper understanding of a research problem compared to that achieved by any single approach.

Based on the research approach and the paradigm used by the researcher, the research methods also vary (Williamson, 2018). Table 3.1 provides a comparison of various paradigms used in conducting research. This research assumes a post-positivist quantitative approach where results are probabilistic since real world datasets for detecting MQTT attack do not exist and the conclusions made in this research are based only on the data produced and collected in a controlled environment. The results might vary in various circumstances which cannot be completely replicated.

Table 3.1: Description of various research paradigms in terms of ontology, epistemology, axiology, methodology, and mode of inquiry. Adopted from Creswell and Creswell (2017), Creswell and Clark (2017), Williamson (2018) and Kivunja and Kuyini (2017)

Paradigms	**Positivist**	**Post-Positivist**	**Interpretivist / Constructivist**	**Pragmatic**
Ontology	Single reality exists that can be discovered	Single reality exists but cannot be easily discovered	Multiple realities exist and varies with the experiences of people	Socially dependent multiple realities arising out of multiple actions, situations and consequences
Epistemology	Knowledge can be observed and experienced. It can be derived using repeatable methods. Researcher is independent of the knowledge being researched	Knowledge cannot completely be based on assessments, rather its based-on probable human conclusions from incomplete information. The probable knowledge can be challenged with further investigations	Knowledge can be subjectively deduced or interpreted based on the research participant's interactions with the surrounding world. Researcher is dependent on the knowledge being researched.	Uses both positivist and interpretivist approach to the nature of knowledge and relationship between the researcher and the knowledge being researched.
Axiology	Value-free, researcher uses an unbiased stance and hence objective	Bias is unavoidable, but it is undesired, hence corrective measures need to be taken to prevent it.	Value-bound, researcher cannot be separated from what is being researched and influence each other, hence subjective	Values influence the interpretation of results. Uses both objective and subjective approach
Methodology	Quantitative	Quantitative as well as Qualitative	Qualitative	Mixed Methods
Mode Of inquiry	• Close-ended questions • Experiments • Quasi-Experiments • Tests • Questionnaires	• Quasi-Experiments • Questionnaires	• Open-Ended Questions • Interviews • Observations • Ethnographic case studies • Surveys • Document Reviews	• Both quantitative and qualitative tools are used

3.2 Research Design

The positivist paradigm provides various methods to conduct the research and gather data to answer the research problem. The most common methods of the positivist paradigm are (Kivunja & Kuyini, 2017):

- Experimental
  - True-Experimental
  - Quasi-Experimental
  - Natural-Experiment
- Non-Experimental
  - Correlational
  - Casual-Comparative
  - Randomised control trials
  - Survey research

Experimental research examines the influence a specific variation has on the outcome. In other words the researcher changes the conditions or values of a particular variable referred to as independent variable (IV) in order to examine the impact on an observed variable known as dependent variable (DV). In addition, the researcher applies the variation only on a specific group and withholds it from another group of participants. These are referred to as experimental group and control group respectively. Experiments can be of two types based on the assignment of participants to groups (Cohen, Manion, & Morrison, 2007): true experiments and quasi-experiments. In true experiments the participants are randomly assigned to experimental and control groups, however in quasi-experiments the participants are not randomly assigned to a specific group due to the independent variable being an inherent characteristic of the participants. Experiments can also be classified based on the control settings used in collecting the data, into laboratory and field experiments. Laboratory experiments are also true experiments where the data is collected in a laboratory conditions. In contrast, field experiments are quasi-experiments where natural settings are used to collect the data and the researcher has limited or no control on the settings (Asgari & Nunes, 2011). Non-experimental research such as surveys try to quantitatively measure the trends, attitudes and opinions of the participants which represent only a sample of larger population.

A quantitative research approach with a post-positivist paradigm was chosen for this research as statistical analysis is required to establish the cause and effect relationship between the treatments applied and the observed outcomes. In this work, an experimental research design was selected to identify the relationship between research variables through controlled laboratory experiments. This is because firstly, to determine the impact of DoS attack effect on the MQTT protocol, an experimental setup is required on which various DoS attacks can be evaluated by carefully controlling the variables. Secondly, a constrained environment is most suitable to evaluate the effectiveness of the proposed MQTT feature vector in detecting attacks. The factors that limit the DoS impact assessments to be conducted in a field environment are the legal policies that prohibit launching DoS attacks in live networks and also unexpected consequences that could be damaging to live systems. In addition, to the best of researcher's knowledge, no publicly available data sources of MQTT attack traffic exists. Hence, the limitation of this study lies in its inability to completely identify the relationship between real world variables as highlighted by Galliers (1991).

The experiments were conducted in a controlled laboratory settings however, random assignment was not used to allocate participants to the experimental and control groups, hence making this experimental research a quasi-experimental study. This design is further demonstrated in the DoS attack impact analysis where the broker software was not randomly selected for the analysis. Similarly, when evaluating the attack detection effectiveness the datasets were not randomly chosen for classification and feature vector generation, rather procedures were applied to all the samples. Some of the common design types of quasi experimental research designs are; Pretest-Posttest non-equivalent control group design, and interrupted time series design or regression-discontinuity designs (Williamson, 2018). In the Pretest-Posttest non-equivalent control group design the researcher considers the participants to be non-equivalent, hence cannot be randomly assigned to treatments. In addition, a pretest baseline and a posttest comparison is performed before and after applying the treatment. In the interrupted time series design, a series of observations or measurements are conducted before and after the treatment over an extended period of time. This method tries to eliminate the maturation and testing factors that pose a threat to the internal validity. Regression-discontinuity designs aim to estimate the mean effect of treatment in non-random participants by selecting subjects in experimental and control group based on threshold score.

After examining the various quasi-experimental designs, a pretest-posttest non-equivalent control group design was selected for this study. The observa-

tions were conducted after the treatment was applied, comparing the results within and between groups. No pretests were conducted and only posttest results were measured because a no-load condition of the broker was considered and DoS impact was measured only after applying the DoS attack. In conclusion, the philosophical assumptions held by the researcher to conduct this research can be summarised into:

- Paradigm: Post-positivist
- Methodology: Quantitative
- Mode of Enquiry: Experimental Research
  - Sub-category: Quasi-Experimental with pretest-posttest non-equivalent control group

3.3 Research Procedure

In this section the procedures adopted in conducting this research are elaborated. The research process consists of five stages which includes problem exploration (RP-1), experimental design (RP-2), conducting experiments (RP-3), observations (RP-4), and analysis (RP-5) as shown in Figure 3.1. Due to the unavailability of MQTT attack datasets, a two phase approach in experiment design and conducting experiments was chosen. Phase-1 of the experimental stage focuses on designing and evaluating the DoS attack scenarios on the MQTT protocol. In Phase-2, the MQTT DoS attack scenarios were used for dataset generation. In addition, Phase-2 also involved feature extraction and evaluation for testing MQTT attack detection accuracy. The main steps of experiment design and conducting experiment stages with respect to Phase-1 and Phase-2 are represented in Figure 3.2. Finally the observations from of both phases were collected and an analysis was conducted.

3.3.1 Experimental Phase-1

The experimental Phase-1 was utilised to answer SQ1 which necessitated the identification of the MQTT protocol vulnerabilities and the modelling of DoS attacks to exploit them. This step was also essential to generate the MQTT datasets as no public datasets were available to evaluate the proposed attack detection method. The important steps of Phase-1 are the following:

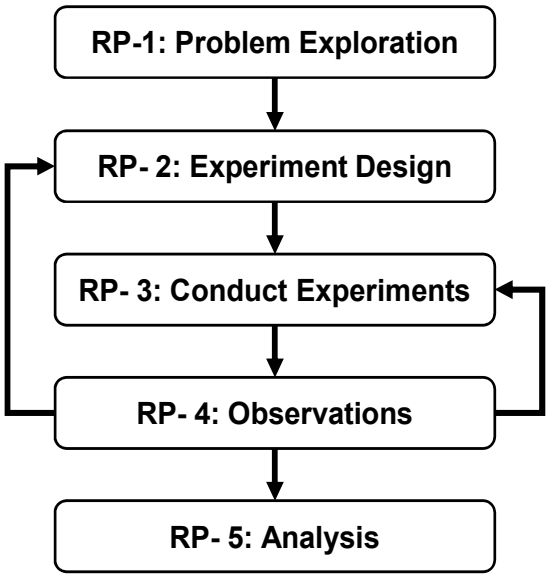


Figure 3.1: Research process used in this research

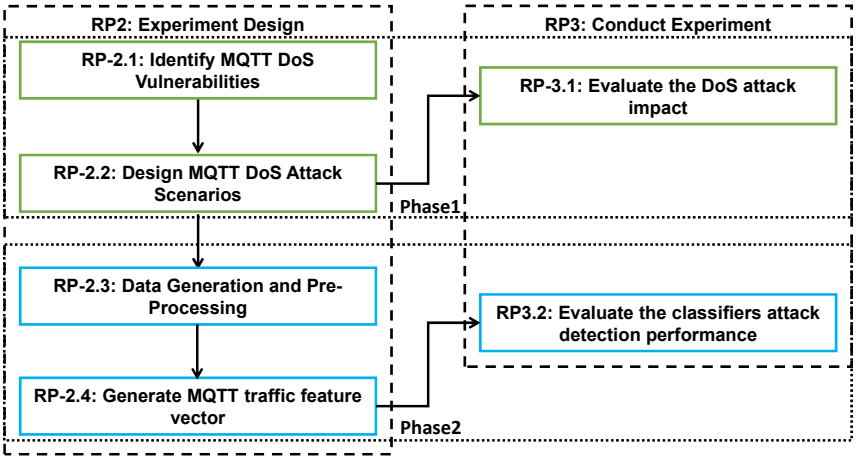


Figure 3.2: Two phases utilised in this research

RP-2.1: Identifying DoS Vulnerabilities The first step in modelling attacks on the MQTT protocol was to identify the vulnerabilities of the protocol. This involved referring to the protocol standard documents and available literature on MQTT vulnerabilities (discussed in Chapter 2) and presenting a MQTT threat model (discussed in Chapter 4).

RP-2.2: Designing MQTT DoS Attack Scenarios Based on the vulnerabilities of the MQTT protocol and DoS attack models of other Application Layer protocols available in the literature, DoS attack scenarios for the MQTT protocol were proposed. The DoS attack scenarios were categorised into authentication and authorisation based attacks. The authentication based attacks targeted the authentication vulnerabilities of the MQTT protocol whereas authorisation attacks targeted the weakness in the authorisation mechanism in the protocol. The four DoS attack scenarios proposed in this study are further discussed in Chapter 4.

RP-3.1: Evaluate DoS Attack Impact In order to evaluate the impact of the attack scenarios designed in the previous step, an experimental testbed using a virtual environment was setup. To avoid any bias on the DoS attack observations, three different open-source MQTT brokers were used. In addition, the deployment scenarios of three brokers were varied to assess the performance of DoS attacks under various server configurations. The choice of MQTT brokers was restricted to open-source software as acquiring and testing commercial software was out of the scope for this research. Another factor in the choice of broker software was based on its wide usage and the scalability features. All the experiments were repeated three times to ensure the validity of the observations and its repeatability. The evaluation results are presented in Chapter 5.

3.3.2 Experimental Phase-2

In order to answer SQ2 and SQ3, a physical IoT testbed was setup to generate normal and attack MQTT data, which was then used to perform attack detection. The main steps in the experimental Phase-2 were as follows:

RP2.3: Data Generation and Pre-Processing A physical IoT testbed was setup with IoT sensors and an open-source MQTT broker. IoT testbed configuration settings were based on real-world implementations to produce realistic MQTT data. Normal traffic was captured under routine operations of the IoT devices. The attack scenarios evaluated in Phase-1 were used to generate MQTT attacks on the broker. The normal and attack traffic were both captured in the form of raw packet captures using packet capture tools. A packet capture tool was configured to store packets in a fixed size files to avoid memory issues during the data pre-processing step. The data pre-processing step also involved extracting the relevant packet information from the raw packet capture files into a Comma Separated Values (CSV)

format. Further details on the IoT testbed design, components used and the tools used to store and process packet captures are discussed in Chapter 4.

RP2.4: MQTT FV Generation With ever increasing transmission speeds and with exponential growth of devices connected to the Internet, inspecting each and every packet payload for anomaly detection is challenging. This leads to Network Intrusion Detection System (NIDS) research into detecting attacks at higher aggregation levels such as flow based (Sperotto et al., 2010) and time-window based detection. Hence in this work flow based and time-window based packet aggregation levels were considered.

The time-window based detection technique models packet arrival as a time-series (Garcia-Teodoro, Diaz-Verdejo, Maciá-Fernández, & Vázquez, 2009) and aggregates packets into multiple time windows. Statistical aggregate features that capture the stochastic behaviour of network activity over fixed length time-windows are considered, to distinguish between normal and anomalous time windows. Since certain anomalies can span multiple time-windows, an overlapping sliding window approach was used in this work. The MQTT based time-window statistical features proposed in this work are based on metrics such as number of specific control packet requests per window, average MQTT field sizes in the observed window and arrival rate of packets and unique flows per window. A two-second time-window was considered in this work as DoS attack traffic delivers a very large volume of packets, which could be challenging to iterate with higher window sizes. The process of time-window based feature extraction is further elaborated in Section 4.4.2

As for the MQTT flow based detection technique, a statistical flow feature vector was extracted by grouping the packets that belong to a network flow. A network flow is identified by five-tuples, namely, source and destination IP address, source and destination port numbers and protocol. All the packets that match with the five-tuples are considered to be part of the same flow. Since certain flows can be long, the flow duration was limited to length of the captured file. Various statistical features based on the flow were generated and stored in CSV format. The datasets were further labelled and merged before the evaluation of machine learning algorithms.

For both time-window and flow-based detection, major and sub-class labelling was used. In major class labelling, the DoS attack scenarios presented in this work were labelled with a single label as MQTT-DOS. Whereas in sub-class labelling, the four DoS attack scenarios were labelled as a separate class. This was done to identify the effectiveness of the proposed features in

detecting attacks that had higher level of similarity. Since real world attacks use stealthy means by resembling normal traffic to avoid detection, attack detection techniques should be capable in differentiating between stealth attack and normal traffic. The MQTT traffic feature vector generation is further explained in Chapter 4.

RP3.2: Evaluate classifier attack detection performance The last step in experimental Phase-2 was the evaluation of various Machine Learning (ML) algorithms. In this step three fundamentally different ML algorithms were applied to labelled MQTT datasets. The evaluation of algorithms was conducted by building models using different feature groups and by measuring the model performance in accurately detecting the attacks. In order to prevent over-fitting and misleading classifier performance results, a 10-fold cross-validation and class balancing methods were applied. A 10-fold cross-validation method was employed in evaluating the datasets to create several groups of training and testing records with each instance occurring in both training and testing dataset at-least once. The class balancing prevents misleading classifier accuracy if records are dominated by a single class producing high detection accuracy. In addition, two MQTT attacks available in the literature were introduced in the dataset to evaluate the performance of the classifier models in detecting attacks, which were not presented in this research. This was also done to avoid any model bias in detecting only attacks presented in this research. The results of MQTT attack detection evaluation is presented in Chapter 5.

3.4 Research Variables

In a quantitative research setting, the researcher manipulates or applies treatment on a set of variables and measures the outcome of another set of variables to observe the cause and effect relationship. The set of variables that receive the research treatment or manipulation are referred to as independent variables and set of variables that are measured for the outcome of the manipulation or influence are referred to as dependent variables. In addition, research involves control and confounding variables that have a potential influence on the dependent variables (Creswell & Creswell, 2017). As discussed in the previous section, two experimental phases were tested in this work which necessitated separate research variables for each phase. The following sections present the various research variables used in the two phases.

3.4.1 Research Variables for Phase-1

Independent Variables

Various categories of DoS attacks were designed to identify the impact of specific vulnerabilities in the MQTT protocol. Based on the vulnerabilities in authentication and authorisation of MQTT protocol, four attack scenarios were designed. The independent variables used in research Phase-1 were the DoS attack types proposed in this research:

1. IV1 - Basic CONNECT flooding attack (BF1): The treatment applied to IV1 was the sleep-interval and the attack threads that controlled the number of attack packets sent to the victim machine.

    - Sleep-Interval (seconds): The interval between two subsequent control packets sent to the victim machine. As discussed in Section 2.4, arrival rate (λ) of packets is one of the parameters used to control the DoS attacks. Hence the sleep-interval controls the inter-arrival time between the requests sent to the victim machine. Lower the sleep-interval, higher the arrival rate. The lowest value this variable can take is zero seconds.
    - Attack-Threads: The number of attack program threads used to launch the attacks. To asses the impact of multiple attack sources, a multi-threaded approach was adopted. The impact of increasing the number of attack threads on the number of attack packets delivered and CPU utilisation, was measured. The attack threads were only incremented in steps of one due to the limitation in using large number of program threads as this would increase the resource contention among threads, thus reducing the effectiveness of the DoS attack.

2. IV2- Delayed CONNECT Flooding attack (BF2): As discussed in Section 2.7.2, MQTT is a TCP based protocol and requires to establish a TCP session before sending a CONNECT request. Similarly one of the parameters used to control DoS attacks is the complexity of requests to increases its processing time, as discussed in Section 2.4. Hence, to assess the impact of delaying the CONNECT request and increasing the time spent by the request in the system, the treatment applied to IV2 was the delay between TCP session establishment and the CONNECT request.

- Delay (seconds): The time delay introduced between the establishment of the TCP session and sending of the CONNECT request.

3. IV3 - CONNECT flooding attack with WILL payload (BF3): As discussed in Section 2.4, one of the aims of DoS attacks is to exhaust the bandwidth resources of the victim machine. Hence, to assess the impact of sending larger CONNECT requests using a WILL payload on the victim bandwidth, the treatment applied to IV3 was the payload size used in the WILL message and the number of attack threads. This was done to measure the impact on bandwidth utilisation on the victim machine.

    - Payload-Size(bytes): The WILL payload size used in the CONNECT control packet.
    - Attack-Threads: The number of attack program threads used to launch the attacks

4. IV4: Invalid Subscription Flooding (IAUTHS). As discussed in Section 2.7.4, MQTT supports multiple subscription topic filters in a single MQTT session. In order to assess the impact on MQTT authorisation mechanism, the treatment applied to IV4 was the number of subscription control packets with invalid authorisation sent to the broker after the MQTT session is established.

    - Number of Subscriptions Loops: Number of subscriptions loops used to send control packets in a single session.

The treatments applied to the IVs and the method of launching various DoS attack scenarios is further elaborated in Chapter 4.

Dependent Variables

The dependent variables measure the impact or influence of the treatment applied to the independent variables. The dependent variables that were observed or measured during the experimental phase were as follows:

- DV1 - Broker Performance:

    1. CPU Utilisation (idle % - percentage of time the CPU was idle during the attack. A lower idle percentage indicates higher CPU utilisation and vice versa)

2. Memory (Percentage of Random Access Memory(RAM) consumed during the attack)
3. Bandwidth Utilisation

- DV2 - Messaging Performances: In order to measure the impact of DoS attack scenarios the attack parameters were kept constant and an MQTT message traffic of 500 messages per second was introduced on a single CPU configuration. The traffic load was introduced to measure more realistic delays in the presence of multiple devices exchanging through the broker. Subsequently the impact of DoS attacks on messages exchanged through three MQTT brokers were observed.

  1. Message Delay (Message delay measured in milliseconds between the sender and the receiver). The average, 50^{th} percentile, 75^{th} percentile and 95^{th} percentile delays were measured for each QoS level under normal and attack scenarios.
  2. Message Publish Rate (Number of published messages received by the subscriber per second)

Control Variables

Control variables are those that have a direct influence on the dependent variable and are controlled to measure its true impact. In this study the validation of the DoS attack impact was done by evaluating the performances of various broker software during the attack. Furthermore, the evaluations were also conducted on two single server configurations and on a load-balanced broker cluster setup. Hence the control variables used in this study are as follows:

- CV1 - Software: Message brokers are one of the most critical components of the MQTT environment as they are used to decouple the publishers and subscribers. In order to ensure the validity of the DoS attack, experiments were evaluated on three broker software namely:

  1. Mosquitto
  2. VerneMQ
  3. EMQ

  The broker software were individually deployed on three separate servers running the same Operating System (OS), MQTT protocol version and

OS settings replicated across the brokers. No updates to either the broker software, OS or default application configuration were performed during the entire course of experimental phase.

- CV2 - Type of Deployment: The number of Central Processing Units (CPUs) available to the MQTT broker software during the DoS attack evaluation were controlled and fixed to:

  1. Single-CPU
  2. Six-CPU

  The single-CPU configuration was selected to ensure a fair comparison of various broker performance was obtained during DoS attack, as MQTT brokers differed in terms of their utilisation of CPU cores for application processing. The six-node deployment was utilised to asses the impact of DoS attack on brokers with higher computing resources. In addition to the two CPU configurations, a load-balanced scalable deployment of a six-node EMQ cluster was evaluated for DoS attack performance. Only EMQ broker software was tested in this deployment type as it supported both load-balancing and cluster configurations by default. All the other hardware and software parameters of the broker servers were not altered during the experimental phase.

- CV3 - Attack tool, libraries : Once the correct setting for the attack tool were identified before the experimental phase, no further changes were made to the attack tool.

- CV4 - QoS levels: The message delay of each of three QoS levels supported by MQTT was measured independently for observing the impact on DV2.

Confounding Variables

Confounding variables are those that the researcher does not intend to measure but influence the relationship between the independent and dependent variables.

- Broker application design: Since this study was focused on studying the impact of MQTT protocol vulnerabilities, open-source broker software implementing the protocol were acquired. Even though, the broker software were based on the same MQTT version, but they differed

in terms of the application configurations used, connection handling mechanisms and system resource utilisation design. Hence, the design and internal settings of the broker application were outside the control of the researcher.

3.4.2 Research Variables for Phase-2

Independent Variables

Independent variables used in the research Phase-2 were as follows:

- IV5 - Classifier Algorithm: Three fundamentally different classifier algorithms were chosen to assess the attack detection performance of the proposed MQTT features. The classifiers used in this study were:

  1. AODE (Average One Dependence Estimator based on Navie Bayes)
  2. C4.5 (Decision Tree based algorithm)
  3. MLP (Multi-layer Perceptron based on Artificial Neural Networks (ANN))

  Two different class labelling were used in the evaluation of the three classifier performances. The class labelling used was:

  1. Major-Class (Normal, MQTT-DOS, MQTT-FUZZ, TCP-DOS)
  2. Sub-Class (Normal, MQTT-DOS-BF1, MQTT-DOS-BF2, MQTT-DOS-BF3, MQTT-DOS-IAUTHS, MQTT-FUZZ, TCP-DOS)

  The class labelling schemes were used in this research to identify the detection performance of the proposed features. The major class represented all major the attack types used in this research to generate the datasets and were considered as a single class (MQTT-DOS). Two additional attack types (MQTT-FUZZ, TCP-DOS) from literature were introduced to evaluate the performance of attack detection.

- IV6 - Feature Vector Type (FV): As discussed in Section 2.8.1, TCP based features and MQTT features based on TCP protocol analysis have been used to detect Application Layer attacks on IoT. Hence to compare the proposed MQTT features in this research with existing studies as well as to identify the features contributing to the detection performance, various feature groups were assessed. In addition, time-window and flow-based aggregation levels were used to create two separate datasets. The feature groups used to build the FV were:

1. FULL features (FULL-FV)
2. TCP-based Features (TCP-FV)
3. Count-based flow features (COUNT-FV)
4. Packet and field length/size based Features (SIZE-FV)

The treatment applied to different independent variables was measured and analysed using the dependent variable.

Dependent Variables

- DV3 - Classifier Performance: The influence of choosing various classifiers and the FV groups was measured using the detection performance of the classifier. The various metrics used in experimental Phase-2 to measure the classifier performance are listed below and further elaborated in Section 3.6.2.

  1. Accuracy
  2. Error
  3. True positive Rate (TPR)
  4. False Positive Rate (FPR)
  5. Training Time

To compare the performances of models developed after training, true positive rates of individual classes were used instead of the accuracy. The true positive rates of individual classes show the performance of the built model in accurately detecting the attack class and misclassification can be identified.

Control / Confounding Variables

- CV5 - Computing Platform (Software and Hardware): The hardware and software of the computing platform used to perform the experiments for Phase-2 were not altered and no updates were carried out during the experimental phase.

- CV6 - Classifier algorithm implementations: the algorithm implementations or the parameters were not modified while comparing the performance of various classifiers and FV.

- CV7 - MQTT Dataset: Since the values of various MQTT fields had a direct impact on the classifier performance, random values matching the normal traffic characteristics were chosen for the MQTT field values to reduce bias in classifier detection performance.

In experiments with more than one independent variable, the interactions of various combinations of independent variables with the dependent variables need to be observed, which allows to strengthen the relationship between the two variables (Asgari & Nunes, 2011). This is achieved by using a factorial design to generate all the possible combinations of independent variables for experimentation. Since all the treatment group were subjected to the variations in the independent variables, a quasi-experimental approach was used to conduct the experiments.

Table 3.2: Factorial Design of 3x3 showing the combinations of various treatment groups of independent variables

		Feature Vector (IV6)			
		FULL-FV(FV1)	**TCP-FV(FV2)**	**COUNT-FV(FV4)**	**SIZE-FV(FV4)**
Classifier (IV5)	**AODE (C1)**	FV1-C1	FV2-C1	FV3-C1	FV4-C1
	C4.5 (C2)	FV1-C1	FV2-C2	FV3-C2	FV4-C2
	MLP (C3)	FV1-C3	FV2-C3	FV3-C3	FV4-C3

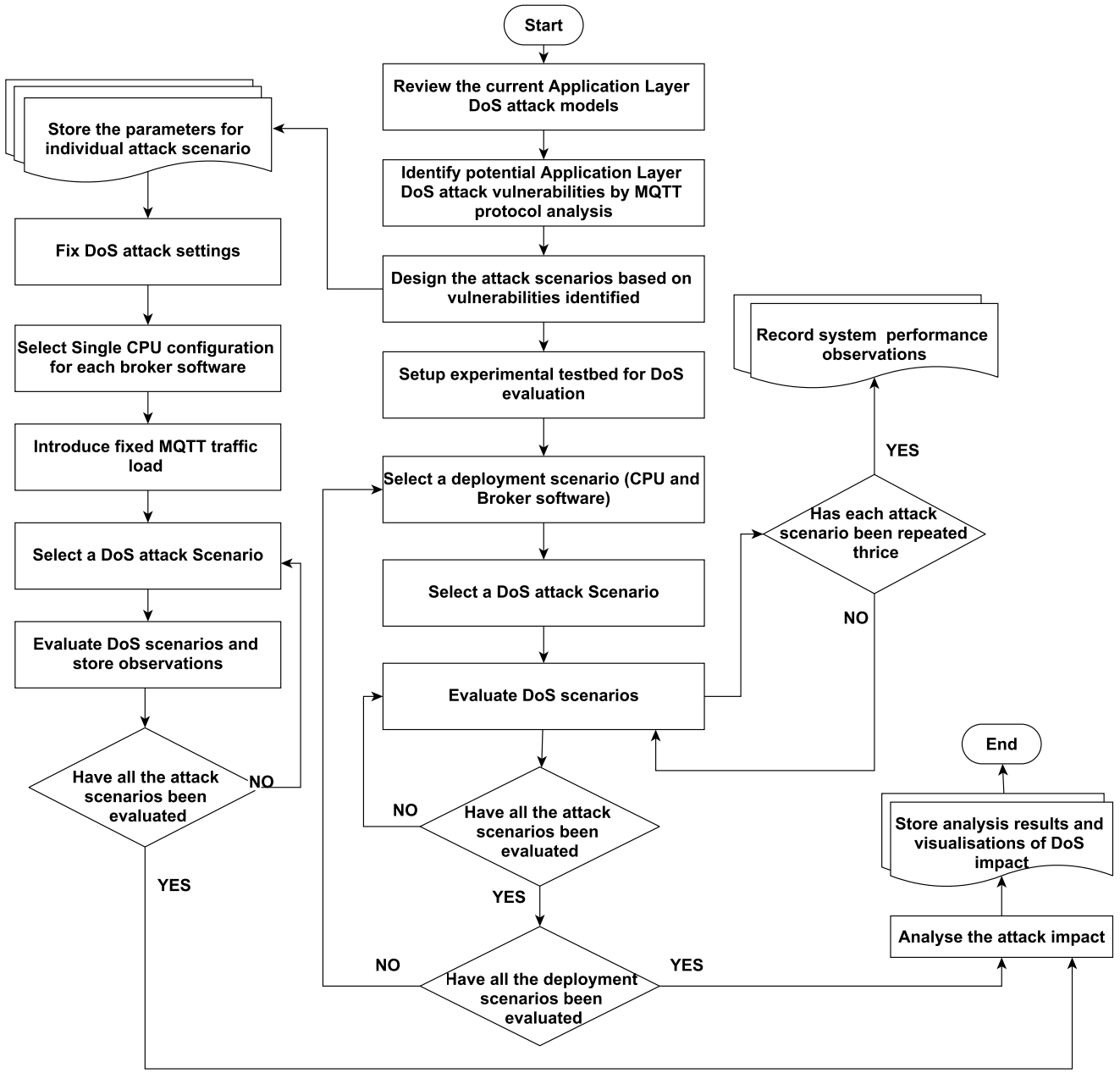
3.5 Experimental Procedures

The various experiments conducted to collect empirical observations to answer SQ1, SQ2 and SQ3 are presented in Table 3.3.

Table 3.3: Treatments applied and parameters observed for various experiments conducted in this research

Experiment	**Treatments Applied**	**Parameters Observed**	**Research Question Answered**
CONNECT Flood	Sleep interval, Attack Threads	CPU idle, Packets per second	SQ1
Delayed CONNECT Flood	CONNECT Delay	Packets per second, CPU idle, Half-open sessions	SQ1
CONNECT Flood with WILL message	WILL Message Payload Size, Attack Threads	Bandwidth utilisation, CPU Idle	SQ1
Invalid Subscription Flood	Subscription Loops	CPU idle	SQ1
Attack Detection Accuracy (Classifier)	Classifier Algorithms	Classifier Performance (Accuracy, TPR,FPR)	SQ2 and SQ3
Attack Detection Accuracy (Feature-fv)	Feature Vectors groups	Classifier Performance (Accuracy, TPR,FPR)	SQ2 and SQ3

The main steps carried out in conducting experiments of Phase1 are highlighted in Figure 3.3. The steps included identifying the existing DoS attack models, identifying the protocol vulnerabilities, designing the attack scenarios, experimental evaluation testbed and evaluation and analysis of the observed results. Artefacts such as attack scenario parameters, DoS attack system performance records, messaging performance and analysis of the DoS attacks were captured during the experimental procedure.


Start
Review the current Application Layer DoS attack models
Identify potential Application Layer DoS attack vulnerabilities by MQTT protocol analysis
Design the attack scenarios based on vulnerabilities identified
Setup experimental testbed for DoS evaluation
Select a deployment scenario (CPU and Broker software)
Select a DoS attack Scenario
Evaluate DoS scenarios
Have all the attack scenarios been evaluated
NO
YES
Has each attack scenario been repeated thrice
NO
YES
Record system performance observations
Have all the deployment scenarios been evaluated
NO
YES
Store the parameters for individual attack scenario
Fix DoS attack settings
Select Single CPU configuration for each broker software
Introduce fixed MQTT traffic load
Select a DoS attack Scenario
Evaluate DoS scenarios and store observations
Have all the attack scenarios been evaluated
NO
YES
Analyse the attack impact
Store analysis results and visualisations of DoS impact
End

In Phase-2 the MQTT attack detection was evaluated by first generating the datasets and then building various ML models based on feature groups and classifier algorithms as shown in Figure 3.4 and Figure 3.5. Both figures are part of the same flow chart and connect each other through the connector 'A'. The research procedures that were used in this step included: deploying a real IoT testbed, capturing benign traffic during normal operations, capturing the attack network traffic, pre-processing raw packet capture files to generate packet information in CSV format, statistical time-window and flow feature extraction, dataset labelling and integrating labelled instances of normal and various MQTT attack scenarios. Several artefacts were generated in this step which consisted of raw packet capture files, packet information in CSV format, FV records, and labelled datasets. The four datasets generated using aggregation levels and class labels are presented in Table 3.4.

Table 3.4: Datasets used in this research to compare models built using FV groups and classifier algorithms

Aggregation Level	**Class Labels**	**Dataset Name**
Time-window	Major Class	TW-Major-DS
Time-window	Sub Class	TW-Sub-DS
Flow-Based	Major Class	FL-Major-DS
Flow-Based	Sub Class	FL-Sub-DS

Subsequently, experiments were conducted to evaluate the attack detection performance of classifiers on the labelled datasets. The various steps included: balancing the classes in the dataset, forming feature groups for classifier evaluation, classifier evaluation using 10-fold cross-validation step, and analysis of the obtained results. The artefacts collected during this step were: classifier performance metrics and visualisations.

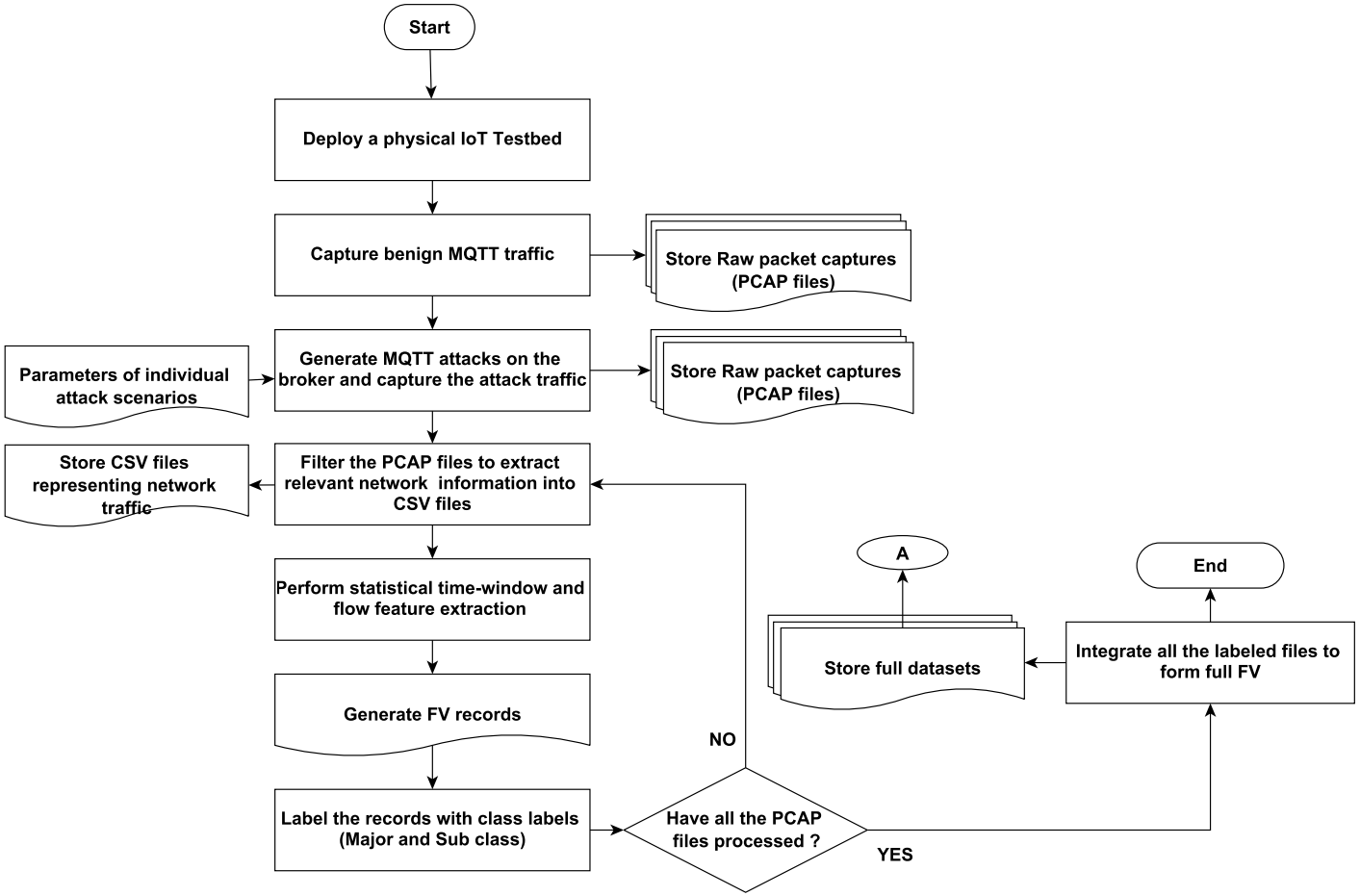


Figure 3.4: Experimental Phase-2 (RP-2.3,2.4)

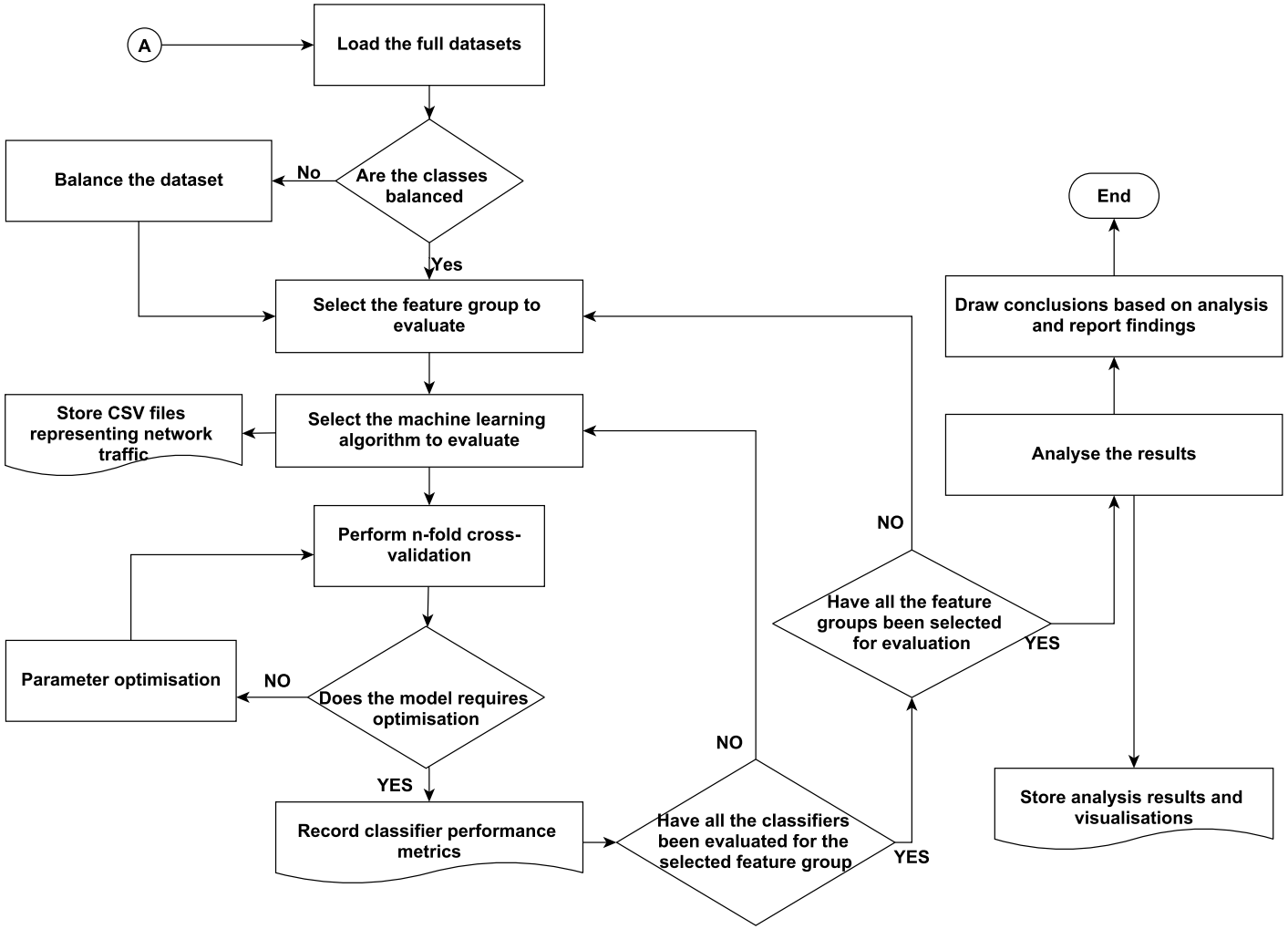


Figure 3.5: Experimental Phase-2 (RP-3.2)

3.6 Data Analysis

The data analysis was carried out to assess the impact of DoS attacks and to evaluate the performance of classifiers selected in the detection framework. The metrics used in this study are discussed in this section.

3.6.1 DoS Attack Evaluation

As DoS attacks intended to exhaust the victim's resources, their impact can be measured using metrics such as CPU time, CPU utilisation of MQTT process, bandwidth, memory and the number of packets per second. The CPU time was measured using *mpstat* Linux command line tool, which provides a break-up of CPU time usage by various system tasks. A custom BASH script was used to measure memory and bandwidth utilisation at every one second interval. Table 3.5 lists the break-up CPU time measured by *mpstat* and the custom BASH script.

3.6.2 Machine Learning Performance Evaluation

The ML algorithms used in the detection framework were evaluated to assess their attack detection performance. The most common ML performance metrics used are : accuracy, error rate, True Positive Rate (TPR), False Positive Rate (FPR) (Dua & Du, 2016; Zaki et al., 2014). Accuracy and error rate of the classifier can be defined as the probabilities of correct and incorrect classifications respectively (Zaki et al., 2014). For a given testing data $\mathbf{D}$ consisting of n records in a d dimensional space with k class labels, the true output class of an input $x_i \in \mathbf{D}$ is y_i and the output class predicted by the classifier is $\bar{y_i}$. Then the general accuracy of the classifier can be represented as:

$$Accuracy = \frac{1}{n}\sum_{i=1}^{n} I(y_i = \bar{y_i}) \tag{3.1}$$

where $I = 1$ if $y_i == \bar{y_i}$ and $I = 0$ if $y_i \neq \bar{y_i}$. and the error rate can be expressed as:

$$Error\ Rate = 1 - Accuracy \tag{3.2}$$

Hence accuracy indicates the number of correct output class classifications and error rate represents the number of misclassification over the entire testing dataset. A more accurate representation of the classifier performance is achieved by identifying the class specific true and false classifications. This

Table 3.5: Attack metrics used to measure the DoS Impact

Parameter	**Description**
%usr	Percentage time spent by CPU executing application related tasks
%sys	Percentage time spent by CPU executing kernel level tasks
%iowait	Percentage time spent by CPU executing disk I/O requests
%soft	Percentage time spent by CPU servicing software interrupts
%idle	Percentage time spent by CPU not executing any tasks and no pending I/O requests. A high idle% indicates the CPU is least utilised and a low idle % indicates high CPU utilisation.
Process CPU (pCPU)	Measured using BASH script fetching the CPU utilisation associated with broker process ID using top Linux command
Bandwidth	Total bandwidth consumed during the attack (kbytes)
Memory	Percentage Memory consumed during the attack
Number half-open TCP sessions	Average Number of TCP sessions in ESTABLISHED state recorded during the attack. Only used to measure the impact of delay CONNECT flood attack

can be represented in a $k \times k$ contingency table also known as *confusion matrix*. A confusion matrix for two-class classification is represented in Table 3.6

True Positives (TP) are the number of correctly detected anomalous instances in the dataset. True Negatives (TN) are the number of correctly detected legitimate instances. False Positives (FP) are the number of normal records classified as anomalous while the False Negatives (FN) are the number of anomalous instances classified as legitimate.

Table 3.6: Confusion matrix for two classes

	True Class	
Predicted Class	Positive	Negative
Positive	True Positive (TP)	False Positive (FP)
Negative	False Negative (FN)	True Negative (TN)

Based on the confusion matrix, the Accuracy can also be represented as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.3}$$

The accuracy metric is highly sensitive to the dataset changes and do not provide useful performance measures of the classifier when the dataset has imbalanced classes (Dua & Du, 2016). Metrics such as precision, recall F-score and receiver operating characteristics (ROC) have been adopted to highlight the classification performance with imbalanced classes. The class specific performance can be obtained by TPR and FPR. TPR measures the percentage of instances correctly classified as anomalous and also referred to as *sensitivity* which can be calculated by:

$$TPR = \frac{TP}{TP + FN} \tag{3.4}$$

FPR is the percentage of instances incorrectly classified as anomalous and is calculated by:

$$FPR = \frac{FP}{FP + TN} \tag{3.5}$$

As both TPR and FPR are insensitive to class sizes or distributions (Zaki et al., 2014), these metrics are used in this study to compare the performance of the classifiers. In addition, the results of classifier's performance with individual datasets presented in this study are weighted average of all classes which take into account the number instances in each class. The TPR rates of the individual classes is also adopted in this study to discuss the classifier's performance of the datasets used.

3.7 Threats to Validity

In experimental research, threats to validity can raise questions about the correctness of the results and their generalisability to other scenarios and

circumstances (Campbell & Stanley, 2015). Two types of threats to validity exist: internal validity and external validity. Experimental research should identify the potential threats to validity and take measures to avoid or minimise their effects (Creswell & Creswell, 2017). Internal validity relates to the methods, procedures, treatments and analysis of the study. In contrast, external validity refers to incorrect inferences made from the results to generalise their applicability on other settings or situations (Creswell & Creswell, 2017). This can occur due to uniqueness of the experimental settings and selected data sample. In order to prevent the internal and external threats to validity several measures were employed in this study such as:

- **Broker Software**: More than one type of broker software having different designs was used to evaluate the DoS attack impact, to increase the generalisability of the results.
- **DoS treatments**: These were based on the vulnerabilities of the MQTT protocol instead of the target broker software used in evaluation, hence these can be reproduced on other broker software.
- **Repetition**: The DoS evaluation experiments were repeated three times with gaps between each repetition to allow the broker to restore to its normal conditions. Repeating the experiments independently several times with similar outcome serves as an evidence that the results can be reproduced (Vaux, Fidler, & Cumming, 2012). Although there are no standard number of repetitions and may vary based on the field of study (Vaux et al., 2012), a three time repetition is adopted in this study to verify that the reported results are not random.
- **External MQTT attack tools**: Attack tools used in other works which significantly differ from the proposed attack scenarios were also used to generate the attack datasets. This was done to prevent the bias of model's performance in detecting only attacks studied in this work.
- **Cross-Validation**: To prevent classifier over-fitting, cross validation was performed using the k-fold method which validates the classifier performance by creating k equal sized datasets (resulting in k models) with each instance used at-least once for testing. The final performance result is the average performance of k models from k equal datasets. In this study a 10-fold cross validation was used.

Table 3.7: Hardware equipment used in the experimental setup of this study

Name	Specification	Role
Virtualisation host	Processor: Intel Core i7-5820K CPU capacity: 6 Core, 12 Threads, 3.30GHz Memory: 64GB RAM	Hosts the broker software virtual machines, conduct machine learning experiments
DoS Attack Machine	Processor: AMD FX(tm)-8120 CPU capacity: 8-Core, 8 Threads, 3.1 GHz Memory: 16GB OS: Ubuntu server 18.04	Generates the DoS attacks
Raspberry Pi3	Processor: ARMv7 Processor rev 4 (v7l), CPU capacity: 4 Core, 1 Thread, 1.2 GHz Memory: 1GB OS: Raspbian GNU/Linux 9, Ubuntu 16.04.2 LTS	For deployment of IoT testbed with hardware sensors
ESP8266-WemosD1 Mini	MCU: Tensilica Xtensa lx106 Capacity: 160 MHz Flash: 16M bytes Firmware: custom	For deployment of IoT testbed with hardware sensors
WiFi Router 2	Model: NetComm Wireless, NF10WV Wi-Fi spec: IEEE 802.11n, 2.4GHz WiFi Ports: 4 x RJ45 10/100Mbps LAN ports	For connecting Raspberry Pi devices to the IoT testbed
WiFi Router 1	Model Name: TP-LINK ARCHER C7, WiFi spec: 802.11ac/Dual Band Ports: 4 x 1 Gbps LAN	For connecting Raspberry Pi devices to the IoT testbed

3.8 Instrumentation

The resources used in this study to conduct the experiments are presented in this section. Various hardware and software tools were employed during the experimental phase and the data analysis phase. The hardware components used for the evaluating the DoS experiments, IoT testbed and ML evaluations are listed in the Table 3.7.

In addition physical sensors were connected to the Raspberry Pi and ESP8266 in the IoT testbed. Table 3.8 lists the various sensors and devices used to build IoT devices.

A number of software resources were used for the IoT testbed deployment, broker software, MQTT attack generation, performance measurements, data processing and machine learning analysis. Table 3.9 lists the various software resources used in this work.

Table 3.8: Sensors and devices used in the physical setup

Name	**Model**	**Quantity**
Temperature Sensor	DS18B20	7
PIR Motion Sensor	PIR - HC-SR501	5
RobotDyn - Real Time Clock	DS1307	1
Optical Fingerprint Reader	Mega2560	2
RobotDyn - USB to TTL UART	CH340G	10
Adafruit MCP3008 8-Channel 10-Bit ADC	MCP3008	5
Adafruit CCS811 Air Quality Sensor	CCS811	2
MQ-2 Smoke Sensor	MH-MQ-2	5

Table 3.9: Software resources used in the experimental and analysis phases of this study

Name	**Version**	**Role**
Oracle Virtual Box	5.2.22	Virtualisation software for deployment of MQTT broker virtual machines
Eclipse Mosquitto	1.4.12	Single-threaded MQTT broker for evaluating the DoS attack impact
VerneMQ	1.6.2	Multi-threaded and scalable MQTT broker built using Erlang OTP used to evaluate DoS attack impact
EMQ	3.0	Multi-threaded and scalable MQTT broker built using Erlang OTP used to evaluate DoS attack impact
HA-Proxy	1.8.8	Software load-balancer to deploy six-node EMQ cluster to evaluate DoS attacks
Python 3	3.6.5	Interpreted, high-level programming language to develop attack scenarios, feature-extraction and classifier tuning.
Weka	3.8.3	Java based ML workbench used for attack detection evaluations
Tshark	3.0	Network protocol analyser tool used for raw data processing
sysstat (mpstat)	11.6.1	system CPU statistics collection tool used to measure the DoS impact
F-secure/ Mqtt_Fuzz	N.A	Protocol fuzzer tool to generate fuzzed MQTT packets
Hping3	3.0	TCP/IP packet assembler used to generate SYN_Flood DoS attack

3.9 Ethics

The research conducted in this study was completely based upon machines and no humans or animal subjects were involved. At all stages of this study only computers were used and configured to conduct experiments and collect the necessary data. In addition, to comply with the PhD candidature procedures of Edith Cowan University an Ethics declaration was submitted to the Ethics committee and approval was obtained.

3.10 Summary

In this chapter, the research methodology adopted in conducting a systematic research was presented. A quantitative methodology was chosen with a post-positivist worldview. An quasi-experimental approach was adopted to gather data to answer the research questions posed in this work.

The experiments were conducted in two phases with separate set of research variables identified for each phase. The independent, dependent, control and confounding variables identified for designing the experiments were elaborated in this Chapter. The first phase of the experiments were focused on evaluating the impact of various MQTT attacks scenarios defined using the protocol vulnerabilities. Based on the attack scenarios and other attack tools, MQTT attack datasets were generated to evaluate detection performance of the proposed attack detection framework.

In addition, a detailed explanation of the research procedures utilised in this study were presented in this chapter. Flowcharts explaining the individual steps and workflow adopted in this research study were illustrated. The data analysis tools and methods deployed to analyse the observations recorded during the experiments were discussed. Finally, the measures adopted to minimise the impact of internal and external threats to validity of the obtained results were presented along with the equipment and resources utilised in conducting the research.

Chapter 4

MQTT Attack Model and Detection Framework

This chapter explains in detail the MQTT DoS attack model and the attack detection framework. First, threats to the MQTT protocol are hypothesised and enumerated using the MQTT threat model. Among the various cyber security threats, DoS attack is further elaborated and the attack model followed to generate the DoS traffic is explained. Secondly, the MQTT attack detection framework for DoS attack detection is presented, which comprises of the steps to generate normal and attack traffic, data collection, features extraction from MQTT traffic, classifiers and the feature selection techniques deployed in this work.

4.1 Threat Model for MQTT Based IoT System

Identifying the potential threats to the IoT protocols and the possible mitigation steps is an important factor in building a secure IoT system. The process of identifying, ascertaining and analysing potential threats and mitigation steps is achieved by developing a threat model. Threat modelling helps in enumerating the attack vectors that arise due to vulnerabilities, and to identify the threat agents who perpetrate attacks. Threat modelling methods like Microsoft's Security Development Life-Cycle (SDL) (Howard & Lipner, 2006), and the Open Web Application Security Project (OWASP) (OWASP, 2018), Process for Attack Simulation and Threat Modelling (PASTA), Operationally Critical Threat, Asset and Vulnerability Evaluation (OCTAVE) and TRIKE (Saitta, Larcom, & Eddington, 2005) have been proposed in literature.

The SDL modelling technique focuses on identifying the assets, application uses cases and subsequently identifies the threats. The most common models of SDL technique are STRIDE and DREAD, with the later being more applicable for ranking the threats based on their risks. STRIDE focuses on identifying technical details of threats to the system by analysing the assets, threat actors and vulnerabilities of the system which result in a lightweight threat modelling process. In addition to the modelling methods, the categories of risk and their ordering are presented in methods such as Microsoft's STRIDE and DREAD models (Hussain, Kamal, Ahmad, Rasool, & Iqbal, 2014). The STRIDE model comprises of six categories namely: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privileges. The DREAD model presents five categories of risk namely: Damage, Reproducibility, Exploitability, Affect Users, and Discoverability. In contrast, the PASTA model is a seven-layer model for extensive modelling which translates the business objectives to identifying the impact or risks of threats. The many layers of modelling required by this approach is suitable for large systems with complex interactions (Potteiger, Martins, & Koutsoukos, 2016). Similarly OCTAVE is a complex threat modelling approach which requires several levels of modelling and is suitable in identifying organisation level threats (Klingel, Khondoker, Marx, & Bayarou, 2014). Since the aim of this work is to identify the threats to a generic MQTT based IoT system without assuming specific business requirements, we adopt the STRIDE model to identify the assets, threat actors and the possible threats to the IoT-MQTT system.

One of the contributions of this work is to present a MQTT threat model which includes identifying components of the system, access points and threat agents which exploit the vulnerabilities of the system. Subsequently the threats to the system are enumerated by hypothesising the breaches of security goals of an IoT system such as confidentiality, integrity, and availability. Finally a mitigation plan is presented with possible countermeasures for the threats. In order to enumerate threats to an MQTT based IoT system, the STRIDE model was adopted. The six threat classes for to describe threats to MQTT protocol are:

- Spoofing: is an attempt to gain unauthorised access to the system using false or fake identity.
- Tampering: is unauthorised modification of any part of the system or data in the system.
- Repudiation: is the ability of users to deny their actions due to failure

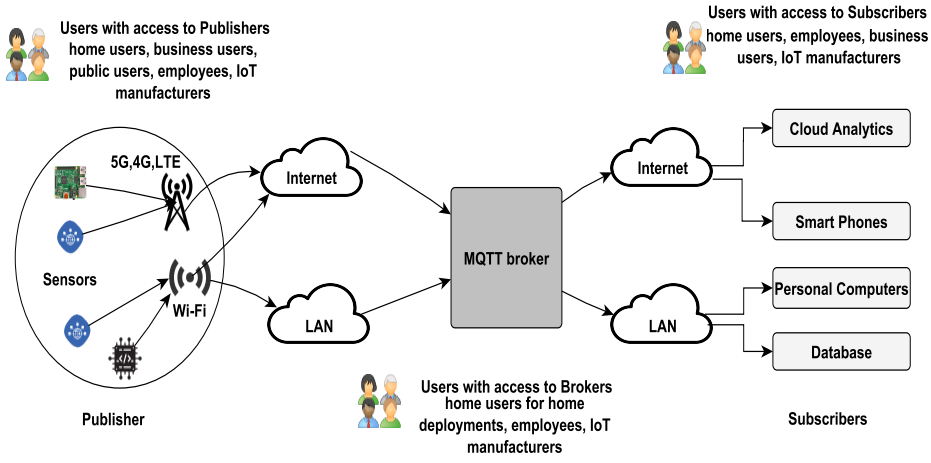


Figure 4.1: IoT based MQTT Publish/Subscribe architecture showing components, users having access to various parts of the system and the connectivity of various parts

to track or log activities

- Information Disclosure: exposing sensitive information to unintended users
- Denial of Service: is making the system or application unavailable to legitimate users.
- Elevation of Privileges: happens when users with limited access tricks the system to gain administrative rights to access privileged resources.

Figure 4.1 shows a typical MQTT based IoT system which consists of IoT clients, an MQTT broker and non IoT endpoints, which comprise smartphone, personal computer (PC) or server. IoT devices could be deployed in smart homes, remote sites, and public places in the case of smart cities, industries, vehicles, etc. Due to the ubiquitous applications of IoT, various types of users can have access and control of these devices. The data publishers can be located in the same LAN as the MQTT broker or located in remote locations with Internet access, hence the publishers can connect to a broker with in the LAN or over the Internet. Subscribers to the published

data can vary from Analytics platforms to smart-phone Applications, Personal computers, databases to application servers; and can be located in the same LAN or on the Internet. To perform threat modelling, we first identify the assets of a typical MQTT based IoT system followed by identification of threat agents and finally enumerating the threats to the system. Atamli and Martin (2014) presented a threat model for security and privacy risks in IoT, which categorised the threat sources as malicious users, bad manufacturers, and external adversary. In addition, the threats to IoT system were classified into device tampering, information disclosure, privacy breach, DoS, spoofing, elevation of privileges, signal injection and side-channel attacks. However, the article does not cover the application protocol specific threats. In another article published by Security Compass (SecurityCompass, 2016), the author described the threat model for publish/subscribe communication, but the threats in MQTT protocol are not specified. In this work, we cover the gap in knowledge by presenting the threat model of IoT based MQTT system, which is divided into asset identification, threat agents, attack categories, threats and a study of its impact.

4.1.1 Asset Identification

Assets in an MQTT based IoT system are defined as components that need to be protected from adversaries. The assets range from physical devices to data exchanged in an IoT-MQTT system. Table 4.1 lists the various assets available in a typical MQTT based IoT deployment.

4.1.2 Threat Agents

Identifying the potential adversaries of a system is an essential part of threat modelling. Hence the threat agent or a threat actor can be defined as the potential adversary who can harm the system by exploiting its vulnerabilities. In MQTT based IoT system, threats agents can be either internal or external agents. Internal agents are those that have authorised access to the components of the system unlike external agents who do not have authorised access. The threat agents of an IoT system (Atamli & Martin, 2014) are reliant on the protocol deployed on it. The threat agents presented in this work are:

- Malicious internal user,
- Bad manufacturer,

Table 4.1: Assets in a typical MQTT based IoT deployment and their descriptions

S/N	Asset	Description
	Publisher	
1	Physical IoT devices	All the physical sensor devices deployed to measure and observe the surroundings using sensors
2	IoT credentials	Credentials of the IoT devices to access the firmware or Operating System (OS)
3	MQTT Client-side applications	MQTT client software used to connect to broker and publish data
4	MQTT client credentials	MQTT credentials used by the client software to authenticate and authorise the publishing client
5	IoT Data	Data collected by IoT devices
6	Communication Devices	Communication equipment used by IoT devices such as wired and wireless devices
	Broker	
1	Broker Server	MQTT broker used to route authenticate, authorise and route IoT messages to subscribed clients
2	Broker Credentials	Credentials to access broker software and OS
3	All Client (publisher and subscriber) credentials	All the Credentials stored in broker or authentication servers used by broker to authenticate clients
4	All Client authorisations rules	All the authorisation rules stored in server to authorise publishing and subscribing clients to access MQTT topics
5	All Client data	Data exchanged between publishers and subscribers via the broker
	Subscriber	
1	Subscriber Devices (Smart-phone, PCs etc.)	End point devices subscribed to MQTT topics and receiving updates when new data sent by the publishing clients
2	MQTT Subscriber application	Application software used by subscribing clients to connect and exchange messages with the brokers
3	Subscriber credentials	Credentials used by subscribing clients to connect to brokers
4	Publisher Data	Data sent to subscribers from the data publishers
5	Subscriber Communication devices	Wired and wireless communication devices used by subscribing clients to connect to the broker

- External attackers, and
- Curious user.

Malicious internal user: This is a user owning the device or having legal access to the device with malicious intentions. This could also be an

internal employees of an IoT application provider who might be involved in the deployment and operations of the devices. This kind of user tries to manipulate the IoT device software to launch attacks or retrieves sensitive information such as MQTT credentials or certificates. A malicious internal user having access to the MQTT broker services can also launch attacks on other IoT devices.

Bad manufacturer: This is a manufacturer who deliberately puts backdoors in the IoT device software to gain information about the users or for remote access to the device. This also includes adversaries embedding malicious code in MQTT client software as well as the MQTT message broker software to collect sensitive user information.

External attackers: These are cyber criminals, script kiddies or expert hackers trying to gain unauthorised access to the devices. These attackers can steal sensitive information, bring down services or hold devices for ransom, for financial or political gains.

Curious user: This is a researcher or a curious user who finds gaps in the technology and pursues further to test vulnerabilities. These could be either an internal or an external user.

Threat agents can have various levels of access to an IoT system such as having physical access to devices, access to the information of the system or access to the communication channels between the devices. The internal users with more privileged access pose a greater risk to the IoT system compared to the external users with limited access. For example, a system administrator managing the MQTT based IoT application will have access to the broker and data exchanged between devices, whereas an owner of the IoT device can have physical access to the device.

4.1.3 Attack Categories, Threats and Impact

Attack categorisation is an important step in understanding the underlying threats to the MQTT IoT system. Understanding different threats to the system and its impact will help in prioritising the mitigation steps according to the threat level. As mentioned earlier, the Microsoft's STRIDE model was adopted to categorise the threats to the MQTT based IoT system into six classes namely: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privileges.

Identity Spoofing

The IoT clients and other endpoints need to use client identifiers while connecting to the MQTT broker. Additional security measures such as username, password or client certificates are used to authenticate the users. However, the client information such as client identifiers, username, password and client certificates can be harvested or credentials can be guessed to impersonate a legitimate MQTT client. Malicious clients gaining access to the MQTT service using spoofed identities can publish unauthorised messages and also subscribe to unauthorised messages. This can cause leakage of confidential information, and sending unauthorised control commands to IoT devices, which can harm the infrastructure or people using these devices. In addition, illegitimate brokers can spoof the identities of legitimate brokers and intercept messages between publishers and subscribers.

Attack scenario 1: Internal malicious user or curious user owning an IoT device or malicious internal employee, obtains MQTT client information such as client identifier, credentials or certificates and connects to the broker using his PC/Laptop, impersonating the IoT device. This user then publishes malicious messages or subscribes to unauthorised topics. For example, a malicious user or curious user purchases a baby monitoring device which comes with access to the vendor's MQTT cloud deployment to remotely monitor and control the device using the vendor‘s MQTT client software. The malicious user can use firmware extraction techniques to access the MQTT information stored on the baby monitor and use this information to connect to the brokers and perform unauthorised publish and subscriptions.

Attack Scenario 2: External attacker gains access to client information by either attacking the IoT device by physical security breach, side channel attacks or brutefore attacks to fetch the client information. The attacker then uses the fetched information to impersonate the MQTT client and performs unauthorised activities.

Attack Scenario 3: External attacker or internal malicious user spoofs the DNS service and redirects the IoT devices to an illegitimate broker, which then captures all the messages exchanged between the clients. The MQTT clients using Fully Qualified Domain Names (FQDN) of the brokers will send DNS queries to obtains the IP address of the broker to connect. Modifying the DNS response with illegitimate broker IP address will result in clients connecting to unauthorised brokers and exchanging messages through it.

Tampering Data

Altering MQTT messages can occur when published messages are altered and delivered to subscribers putting the message integrity at risk. Since many IoT devices take decisions based on updates from the surrounding environment, the message integrity is crucial. Modified messages can cause IoT devices to take wrong decisions which can adversely impact the end users. Tampering data also includes modifying or deleting client data, session information and logs stored in the broker.

Attack Scenario 1: Internal malicious user hijacks the communication channel to modify the usage data being sent by smart meter to the utility company. The malicious user can intercept messages exchanged between smart meters and the utility company using Man in the Middle attacks (MitM) and modify the meter readings sent to the utility company.

Attack Scenario 2: External attacker modifying data stored in MQTT brokers. An external attacker can compromise vulnerable broker servers and modify the data stored to cause disruption of service.

Repudiation

Repudiation attacks can occur in MQTT systems when messages sent through the broker or actions performed on it can be denied by the threat agent. This attack type can use data tampering attacks or exploit the vulnerabilities of system configured with weak logging. In the MQTT protocol, multiple clients can share the credentials with unique client identifiers. This can be exploited to cause repudiation attacks unless unique parameters of clients are logged in the system for identification purposes.

Attack Scenario 1: Internal malicious user logs into the MQTT broker and disables activity logging and deletes user activities and performs illegal activity.

Attack Scenario 2: External malicious user impersonates a legitimate device or uses a compromised device to launch other attacks in the system, which will make it challenging to associate the attack to the external malicious user.

Information Disclosure

Unauthorised access of data stored in the IoT devices or MQTT broker and revealing to unauthorised end users causes information disclosure. MQTT brokers contain client credentials, client authorisation details, messages with QoS2 and messages with retain flag set to true. It is necessary to protect this

information from disclosure to unauthorised entities. With many MQTT client software available on the World Wide Web and Google Play, MQTT clients with embedded malicious code can capture messages being exchanged between client and broker and disclose to adversaries.

Attack Scenario 1: Internal malicious user accesses the broker and reveals confidential information such as MQTT connection credentials, broker configurations, information from different topics, sensitive information exchanged between endpoints.

Attack Scenario 2: External attackers compromises an insecure MQTT broker and accesses information stored in the broker and releases to the public. A broker server, which exposes other ports such as SSH or Telnet, can be vulnerable to compromise and attackers can reveal sensitive MQTT information from the compromised broker servers.

Denial of Service (DoS)

One of the important role of a broker server is to route messages between publishers and subscribers. Disrupting broker services can cause DoS in an IoT environments that exchanges messages via the message broker. Exhausting broker resources can result in messages being delayed or dropped or denied access for legitimate client. For example, MQTT supports a maximum payload size of 256 MB, and IoT client resources can be exhausted if messages with larger payload are sent to them. An attacker can exploit this and cause DOS on both client and broker. Furthermore, MQTT is a TCP based protocol hence, brokers will be vulnerable to TCP based DoS attacks targeting bandwidth or system resources. In addition to TCP attacks, the adversaries can also exploit the QoS levels provided by the MQTT protocol to cause DoS. The messages published with QoS2 require more broker resources compared to the QoS1 and QoS0 messages, as listed in Table 2.11. The broker also retains messages sent with QoS1 and QoS2 until messages are delivered to the subscribed clients. Adversaries can send large number of messages with QoS2 and exhaust the broker resources resulting in DoS.

Attack Scenario 1: External attacker/disgruntled internal user crafting TCP based (e.g. SYN Flooding) DDoS attack to exhaust broker resources. Since TCP sessions require acknowledgements, sending multiple SYN messages will create multiple half-opened TCP sessions which can exhaust message broker resources.

Attack Scenario 2: External attacker/disgruntled internal user flooding the MQTT broker with CONNECT packets. The CONNECT packet is used by MQTT client to initiate an MQTT session with the broker and

contains client ID and other optional parameters. When the broker server receives a CONNECT packet, it verifies the client identifier and other optional parameters to either allow or deny connection to the client. Malicious users can send multiple CONNECT packets with different client identifiers to exhaust server resources.

Attack Scenario 3: External attacker/disgruntled internal user sending packets with a larger payload to exhaust the broker and subscriber resources in order to deny service to clients. Malicious users can also use higher levels of QoS with larger payload messages and consume broker resources, as the broker is required to store the messages until they are delivered to all the subscribed clients for QoS1 and QoS2.

Elevation of Privileges

Gaining access to restricted privileges to execute restricted commands is defined as elevation of privileges. In an MQTT environment elevation of privileges occurs with unauthorised publishing and unauthorised subscription. MQTT permits wildcard subscription which allows clients to receive messages from all the topics that match the wildcard. An unauthorised user can gain access to wildcard topics or to "#" which gives the attacker access to all the messages sent by clients.

Attack Scenario 1: External attacker/internal malicious user/curious user subscribing to restricted topics to eavesdrop on messages exchanged between endpoints. Malicious user connects to broker and then subscribes to wild card topics or "#" topic and receives all the messages exchanged between clients.

Attack Scenario 2: External attacker/internal malicious user publishing privileged messages. In MQTT operations, the clients can be configured to listen to messages and take action based on the message content. As these devices can be remotely controlled, a malicious user can send messages with commands such as "OFF" to disable some IoT devices or send commands such as "FORMAT", "RESTORE FACTORY SETTINGS" to damage the IoT devices.

The various threats to MQTT protocol are summarised in the Figure 4.2, which illustrates the threat model and the threats to a MQTT based IoT system. Among the various threats discussed in the threat model, DoS attack has been elaborated in this work as DoS attacks are a common and very harmful threats to Application Layer protocols as discussed in Section 2.4.5. Such targeted DoS attacks come as both flooding or semantic attacks as shown in Figure 2.15 and needs to be modelled to cause maximum impact.

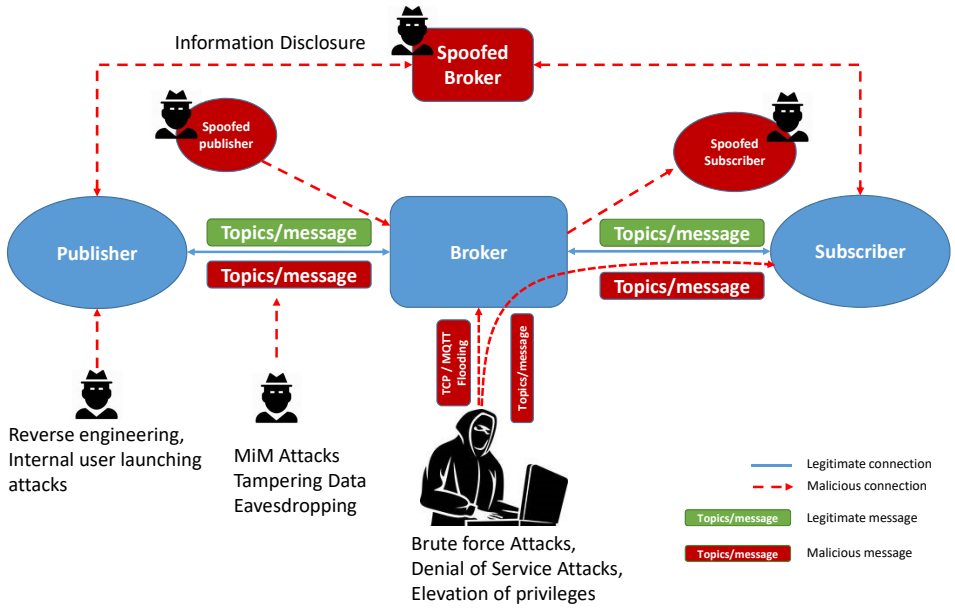


Figure 4.2: MQTT threat model illustrating the various threat agents and the threats to a MQTT based IoT system

As no prior work exists on modelling and enumerating the various MQTT protocol DoS attacks, this work aims to model and analyse the impact of MQTT based DoS attacks on broker performance and message delay. The following section discusses the MQTT DoS attack model and the various attack scenarios designed to cause DoS .

4.2 DoS Attack Model for MQTT/IoT

Based on the DoS attack options the MQTT protocol can be targeted with flooding attacks that send excessive packets to the broker or send complex packets that either stay longer in the system or utilise more CPU cycles to process. As presented in Table 2.8, MQTT supports various control packets for message exchange and control packet flooding is a common technique used in Application Layer DoS as discussed in Section 2.4.5. The CONNECT packet is used for MQTT session establishment and PUBLISH/SUBSCRIBE control packets are used for publishing and subscribing messages. In most industry deployed MQTT system, the clients must first connect to the system. Authenticated clients are then authorised to either send or receive

messages on selected topics. Hence the key mechanisms that allow clients to access MQTT system are authentication and authorisation. Access levels available for MQTT client are as below:

- Valid Credentials to connect to MQTT broker
- Valid Authorisation to Publish/Subscribe to topics

Assuming that the attackers have either no or limited access to the MQTT broker various attack scenarios can be designed for the MQTT protocol. Firstly with invalid credentials, the attacker can only vary the parameters of the CONNECT packet. Secondly, after a successful connection using valid credentials but without valid authorisation to publish and subscribe to topics, the attacker can vary PUBLISH or SUBSCRIBE control packet parameters. Based on the access levels and authorisation levels four attack scenarios are defined:

- Basic CONNECT Flooding (BF1),
- Delayed CONNECT Flooding (BF2),
- CONNECT flooding with WILL payload (BF3), and
- Invalid Subscription Flooding (IAUTHS).

These attack scenarios are discussed in detail in the following sections.

4.2.1 Basic CONNECT Flooding (BF1)

The attacker only sends a large volume of CONNECT requests to the target server to overwhelm the server with processing of requests. Based on the complexity of the authentication mechanism, broker resources can be overloaded at various degrees. Some of the common authentication mechanisms are shown in Figure 4.3:

In this case the external DB and Authentication server based client authentication deployments will have higher attack impact of CONNECT flooding. With the connect flooding attack, multiple CONNECT requests are sent to the broker by opening multiple TCP sessions, as shown in Figure 4.4. This will require the broker to process individual requests and acknowledge these with a CONNACK packet. The attack works by increasing the new connection request arrival rate (λ) at a faster rate than the average processing rate (W) of the broker, thus exhausting the CPU capacity.

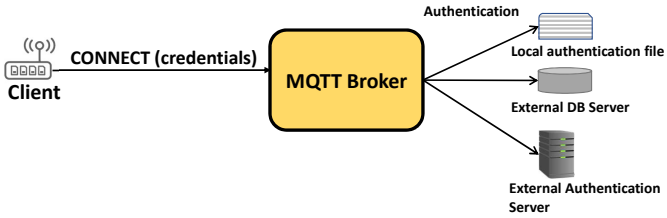


Figure 4.3: Client authentication mechanisms used in MQTT broker, a more complex authentication mechanism will have higher impact with authentication flooding attacks

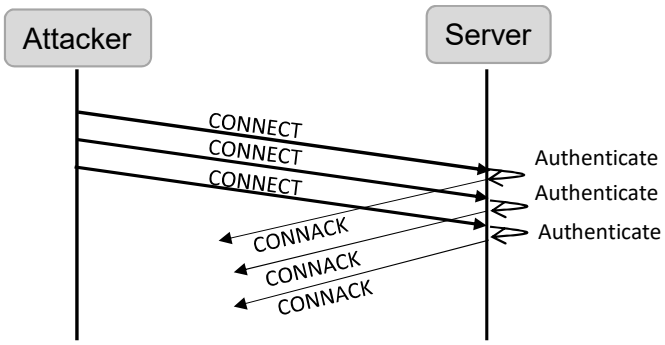


Figure 4.4: Basic CONNECT flooding scenario showing the attack processes of sending multiple connection requests

Algorithm 1 describes steps involved in generating the attack packets for this scenario. The MQTT parameters such as client identifier, username and password are randomly assigned. The attack machine first establishes the TCP session and sends the CONNECT packet. The rate at which the connect packets are sent is controlled by Sleep_Interval parameter. A multi-threaded function then spawns multiple threads to maximise the attack rate.

4.2.2 Delayed CONNECT Flooding (BF2)

The aim of the attack scenario is to increase the time spent by the attack request in the broker processing queue, so as to exhaust the queue. This is achieved by exploiting the TCP time-out setting used in IoT deployments. The time-out of an established TCP session depends on the application and generally in IoT deployments higher time-out settings are required as most of these devices operate through unreliable network links. Attacker

1: **Inputs:**
Number of Threads(nT), BrokerIP (B), Port(P)
2: **procedure** ATTACK(B, P)
while *True* **do**
$ClientID \leftarrow random()$;
$Username \leftarrow random()$;
$Password \leftarrow random()$;
$TCP\_Connection \leftarrow Establish()$;
Send CONNECT Packet ;
$Sleep \leftarrow Sleep\_Interval$
end
3: **end procedure**
4: **procedure** MULTI-THREAD()
for $i \leftarrow 1, nT$ **do**
Spawn Attack() Threads;
end
5: **end procedure**

Algorithm 1: Basic CONNECT flood generation algorithm

can abuse these settings to send delayed CONNECT packets after TCP connection establishment. This will result in a large number of half-open TCP sessions at the broker, as it is waiting for the CONNECT request to complete. It also causes the broker to process these connection requests with invalid credentials, thus leading to an increase in CPU utilisation. This is a semantic attack scenario as it abuses the application sensitivity to delays and setting lower TCP time-outs can have negative impacts on IoT devices, which can result in frequent session time-outs and reduced battery life for battery powered devices. Figure 4.5 shows the delayed CONNECT packet flooding attack.

Algorithm 2 describes the delayed CONNECT flood generation steps for generating the attack packets. Similar to basic CONNECT flood, the client identifier, username and password are initiated to random values. A delay is introduced between the establishment of the TCP connection and the sending of the CONNECT request. The MQTT Paho client library was modified to introduce delay in sending the CONNECT request. A multi-threaded function was introduced to launch multiple connection requests to the broker.

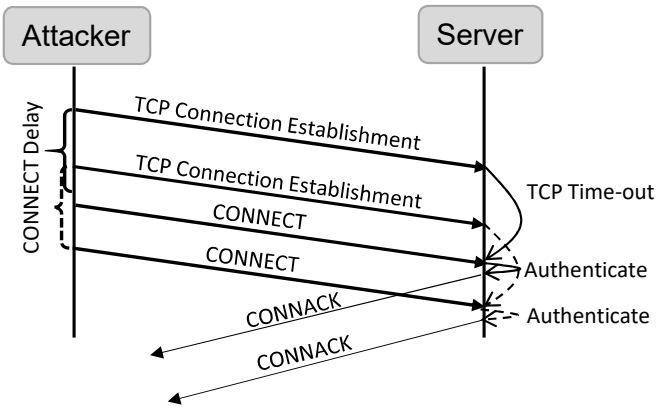


Figure 4.5: Delayed CONNECT flooding scenario attack which introduces a delay between the TCP connection establishment and sending the CONNECT control packet

1: **Inputs:**
Number of Threads(nT), BrokerIP (B), Port(P), Delay(D)
2: **procedure** ATTACK(B, P, D)
while *True* **do**
$ClientID \leftarrow random()$;
$Username \leftarrow random()$;
$Password \leftarrow random()$;
$TCP - Connection \leftarrow Establish()$;
$Sleep \leftarrow D$;
Send CONNECT Packet
end
3: **end procedure**
4: **procedure** MULTI-THREAD()
for $i \leftarrow 1, nT$ **do**
Spawn Attack() Threads;
end
5: **end procedure**

Algorithm 2: Delayed CONNECT flood generation algorithm

4.2.3 CONNECT flooding with WILL payload (BF3)

The CONNECT packet size is increased by the attacker through piggybacking a WILL Payload on a CONNECT packet. As discussed in Section 2.7.2, a WILL payload can be added to the CONNECT control message and this feature can be exploited by sending a large sized WILL payload in the CONNECT control packet. This can lead to consumption of both the bandwidth resources at the victim server as well as CPU resources in processing connections with invalid credentials, preventing it from processing new connections. Figure 4.6 shows the attack technique against the broker server. The difference between this attack scenario and CONNECT flood-

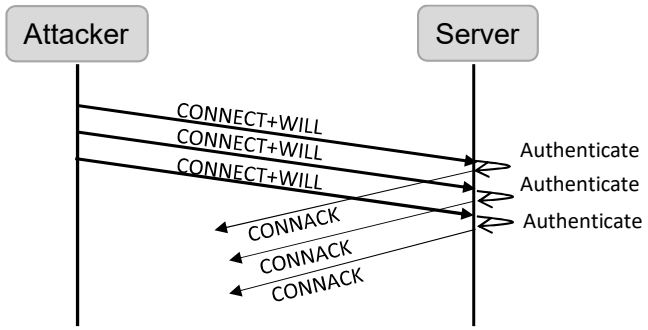


Figure 4.6: CONNECT flooding with WILL payload attack scenario showing the piggybacking of WILL payload with the CONNECT packet

ing scenarios is that, this attack carries a larger payload which can induce fragmentation of packets as well as slow down the number of control packets sent to the broker. Algorithm 3, describes the steps undertaken in generating CONNECT flood attack packets with WILL message. Similar to the basic CONNECT flood, the variables are initialised to random values and the WILL payload, WILL Topic and WILL QoS are added to the payload. A multi-threaded function allows the control of number of attack requests sent to the broker.

4.2.4 Invalid Subscription Flooding (IAUTHS)

With valid credentials but no authorisation to access various topics, an attacker can flood the broker with invalid subscriptions or publish requests to the subscriber. As discussed in Section 2.7.4, the client can send multiple SUBSCRIBE requests in the same message and this can misused by adver-

1: **Inputs:**
Number of Threads(nT), BrokerIP (B), Port(P),
WILL_payload(W)
2: **procedure** ATTACK(B, P, W)
while *True* **do**
$ClientID \leftarrow random()$;
$Username \leftarrow random()$;
$Password \leftarrow random()$;
$TCP\_Connection \leftarrow Establish()$;
$WILL\_Topic \leftarrow random()$;
$WILL\_QoS \leftarrow random(0, 2)$;
$WILL\_Msg \leftarrow W$;
Send CONNECT Packet ;
end
3: **end procedure**
4: **procedure** MULTI-THREAD()
for $i \leftarrow 1, nT$ **do**
Spawn Attack() Threads;
end
5: **end procedure**

Algorithm 3: CONNECT + WILL Payload flood generation algorithm

saries to send multiple invalid subscriptions. This will result in consumption of broker CPU resources in verifying the authorisation levels of individual request.

In SUBSCRIBE flooding, valid credentials are used to establish MQTT session with the broker and subsequently multiple subscription requests are sent to the broker as shown in the Algorithm 4. The multi-threading function is used to increase the attack connections sent to the broker.

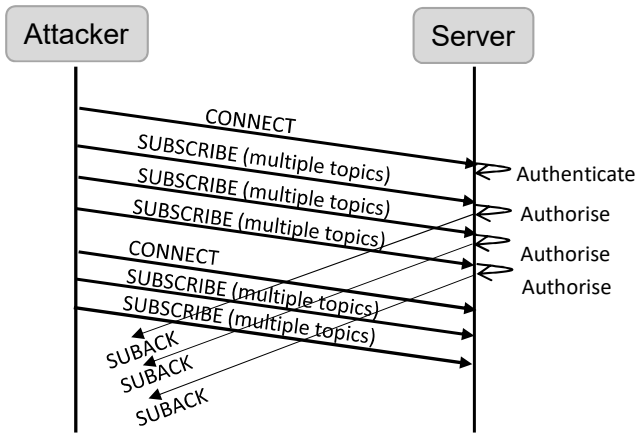


Figure 4.7: SUBSCRIBE Flooding flood attack scenario illustrating the process of launching the attack

1: **Inputs:**
Number of Threads(nT), BrokerIP (B), Port(P),
Number of Subscriptions (S)
2: **procedure** ATTACK(B, P, S)
while *True* **do**
$ClientID \leftarrow random()$;
$Username \leftarrow correct\_Username$;
$Password \leftarrow correct\_Password$;
$TCP - Connection \leftarrow Establish()$;
Send CONNECT Packet ;
for $j \leftarrow 1, S$ **do**
$SUBSCRIBE - Topics \leftarrow random()$;
Send SUBCRIBE ;
end
end
3: **end procedure**
4: **procedure** MULTI-THREAD()
for $i \leftarrow 1, nT$ **do**
Spawn Attack() Threads;
end
5: **end procedure**

Algorithm 4: SUBSCRIBE flood generation algorithm

4.3 MQTT DoS Evaluation Test-bed

The four DoS attack scenarios were evaluated on the MQTT protocol version 3.1, deployed through three open-source broker software tools, on standalone and load-balanced server configurations as described in Table 4.2.

Table 4.2: Hardware Deployment to Test DoS attack Scenarios

Deployment Type	**Number of CPUs Per Server**	RAM
Standalone Server	1 CPU	8GB
Standalone Server	6 CPU	8GB
Load-Balanced with 6-Node Server cluster	2 CPU/Node and 1 CPU for HAProxy	4GB RAM/Node

The standalone and load-balanced MQTT broker servers were hosted as virtual machines on a Windows 10 machine (64GB RAM, Intel Core i7-5820K, 6 physical CPUs, 12vCPU, 3.30GHz) using Oracle Virtual Box (Oracle, 2018). In the load-balanced setup, HA-Proxy (HAproxy, 2018) software load-balancer deployed on a Ubuntu Server 18.04 virtual machine was deployed to balance traffic across the six-node cluster in a Round-Robin scheme.

The three open-source MQTT broker implementations deployed in this study were namely: Eclipse Mosquitto (1.4.12) (Mosquitto, 2017), VerneMQ (1.6.2) (VerneMQ, 2018) and EMQ (3.0) (EMQ, 2018). Mosquitto is a light-weight, portable and single-threaded MQTT broker. In contrast, VerneMQ and EMQ brokers are multi-threaded and scalable MQTT brokers designed using Erlang/Open Telecom Platform (OTP) to achieve high scalability. These brokers were deployed on single-CPU and six-CPU configurations running the Ubuntu Server 17.10. However, only the EMQ broker was deployed for the load-balanced setup owing to ease of configuration. A separate physical machine running Ubuntu server 18.04, connected to the broker network using a router, was configured to launch the DoS attacks as shown in Figure 4.8.

DoS attack traffic was generated using a custom built MQTT attack tool based on the Ecliplse-Paho library (Eclipse, 2018). Each attack was based on specific MQTT protocol settings as discussed in Section 4.2. The CONNECT flooding attacks were configured with a random length character comprising a ClientID, username and password, to emulate a real client. The BF3 attack was launched with varied sizes of payloads containing random characters including non-ASCII characters.

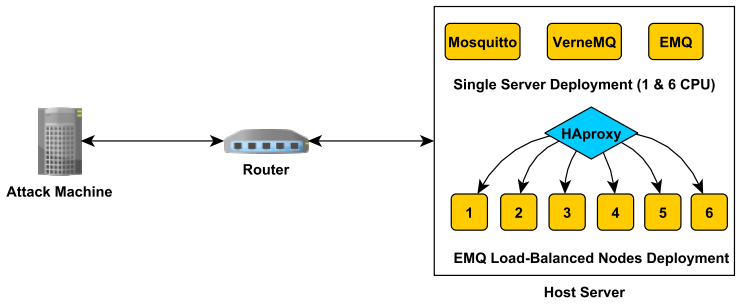


Figure 4.8: MQTT broker deployment (standalone and load-balanced) on virtual machines to assess the four DoS Attacks scenarios

Since these MQTT attacks were generated from a single attack source, a multi-threaded approach was adopted in the attack tool to maximise the impact on a victim machine's available resources. The impact of increasing the number of attack threads on Single-CPU victim machine was measured using the number of packets per second received as well as the CPU utilisation on the victim machine. For BF1 and BF3 attack scenarios, the experimental evaluation consisted for five iterations with single thread increments in each iteration. The reason for incrementing the threads only by one, was that the attack machine had limited resources and resource contention among threads increases with the spawning of new threads. However, for delayed CONNECT flooding attack (BF2), 250 threads were launched with various delay intervals for both single, six and LB deployments to evaluate the broker performance. The IAUTHS attack impact on the victim machine's performance was measured by using various subscribe loop settings and a single attack thread. In contrast, the impact on Six-CPU and LB setup for the attack scenarios BF1, BF3 and IAUTHS flooding attacks were evaluated using three attack threads, which will be further discussed in Chapter 5.

The broker messaging performances were evaluated using a constant attack and load settings as no-load conditions do not reflect real-world delay scenarios. The attack and load settings along with the delay measurement method are further elaborated in Section 5.2. The following section discuses the normal and attack traffic generation methods used in this study.

4.4 DoS Attack Detection Framework

In this section, a DoS attack detection framework is proposed. The framework comprises the following components; MQTT traffic generator, feature extraction engine and a machine learning-based DoS attack traffic classifier. The MQTT traffic generation step involves generating both normal and attack traffic.

4.4.1 MQTT Traffic Generation

An IoT-MQTT network typically comprises a set of IoT sensors that observe environmental phenomena and constantly communicate with each other or with monitoring or control devices through a centralised message broker. The normal traffic was modelled by identifying the normal states of the MOTT protocol, as depicted in Figure 4.9. For example, a publisher only connects to a broker for publishing the sensing data when it is available. In contrast, a subscriber is always connected to the broker to receive updates related to the subscribed topics. A normal publish event will consist of a

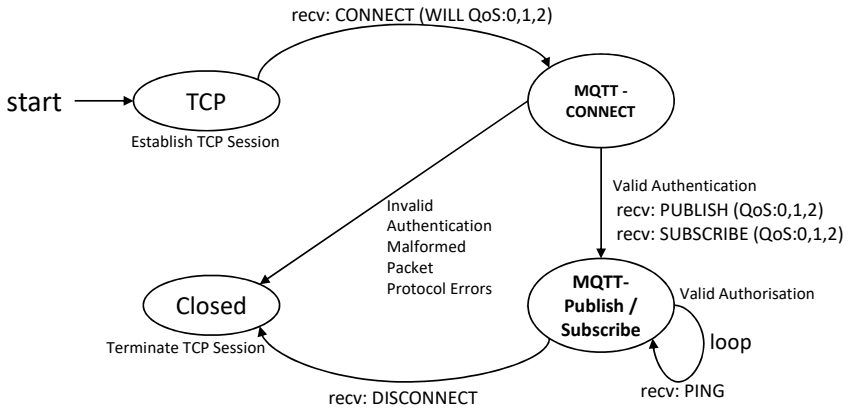


Figure 4.9: MQTT states used for normal traffic modelling

CONNECT request, authentication and PUBLISH request, authorisation and DISCONNECT request. The time interval between successive publish requests is application dependent. Similarly, in a normal subscribe request, the subscriber first establishes a connection by sending a CONNECT request and in order to subscribe to topics it sends a SUBSCRIBE request to various topics. Once subscription is granted, the subscriber stays connected to the broker to receive future updates to the topics. The subscriber sends

PING control packets to keep the connection alive when no other control packets are available to be sent to the broker. The proposed traffic generation component of the detection framework includes two physical servers, 30 Raspberry Pi devices and four ESP8266 micro-controller. One of the servers hosted VerneMQ MQTT broker and the second served as the attack machine. 30 Raspberry Pi devices and four WEMOS ESP8266 devices were connected to two wireless routers; equally distributed. Twenty PIs and four ESP8266s each interfaced with a physical sensors were configured to publish sensor data periodically. Figure 4.10 to Figure 4.12 show the physical deployment of various devices connected to the test-bed which were deployed for data generation. The test-bed included various sensors such as: PIR motion sensor, CCS811-Air Quality Sensor, DS130 RTC clock, DS18B20 temperature sensor and MH-MQ Gas sensor. The remaining 14 Raspberry Pis were configured to periodically send MQTT messages to the broker.

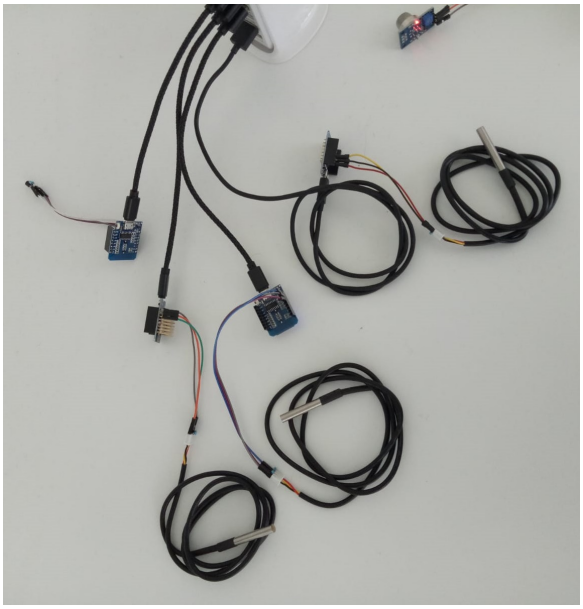


Figure 4.10: Lab deployment of ESP8266 devices

Real-life IoT test-beds were studied to set a realistic publish intervals as described in Table 4.3, to emulate an actual IoT network.

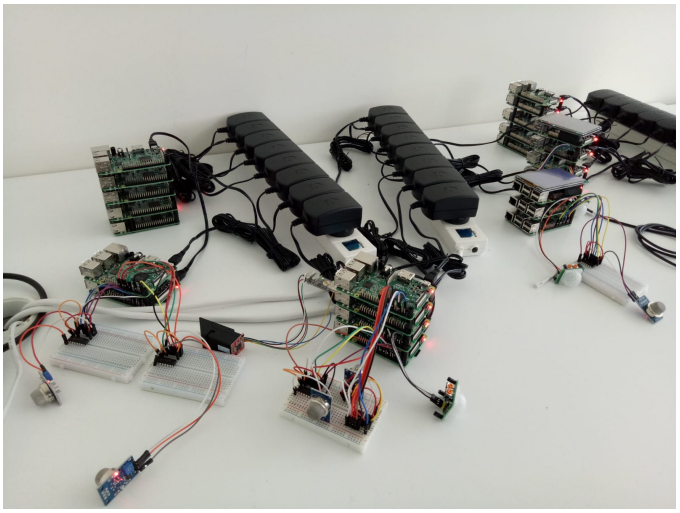


Figure 4.11: Lab deployment of 30 Raspberry Pi devices - part 1

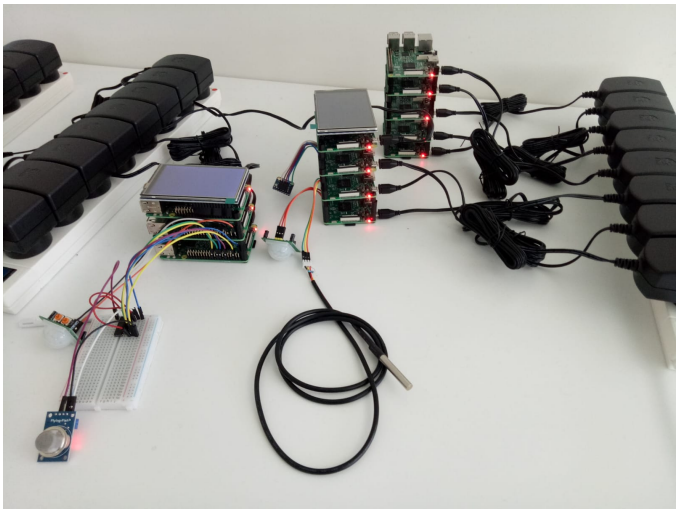


Figure 4.12: Lab deployment of 30 Raspberry Pi devices - part 2

Table 4.3: Real-world MQTT deployments used in a public transport CI

Organisation	**MQTT-IOT use**	**Information Transmitted**	**MQTT Tools Used**	**Sensors**	**Message sending/receiving rate**
Deutsche Bahn AG (DB), Germany's DB Railway System (eclipse.org, 2018)	Long Distance Trains	Real-time information about location, delay, and diagnostic checks	IBM Message Sight running at control centre, Eclipse-PAHO client library for sending messages	600 trains with gateways	Sending location data every 10 seconds, 3000 messages per day per vehicle
	Dynamic Text Displays	Scheduled updates: delays, cancellations, or platform changes	Custom Eclipse-Paho Library	Low-power 6500 edge LED devices	Receiving 25 message per second
	Escalators and Elevators	State information, working condition, power consumption	Ecliplse-Paho MQTT library	3000 edge devices	Send 10 messages per second
	InterCity Express trains	Various vehicle information	Mosquitto broker for inter train communication and train to control centre communication	Deployed on 256 trains	3000 messages per day per vehicle
NEXCOM (Wu, 2017)	Industry process automation, analytics and reporting	Telemetry information, device updates	Azure IoT Hub		Messages sent every 30 seconds

These studies indicate that the MQTT brokers will have to handle high volume traffic emerging from a large number of sensors. The impact of DoS attacks on such infrastructures can have a detrimental effect on IoT systems. Utilising the study on real deployments the sensors were configured to send updates to the broker with a varying periodicity. Figure 4.13 shows the cumulative distribution plot (CDF) of sleep intervals of various sensors of the test-bed recorded in a one-hour duration, which shows that most devices had a sleep interval between 4-8 seconds. The reason to keep the sleep intervals below ten seconds was to achieve a realistic message publish rate, in alignment with the standard practice (Sivanathan et al., 2018).

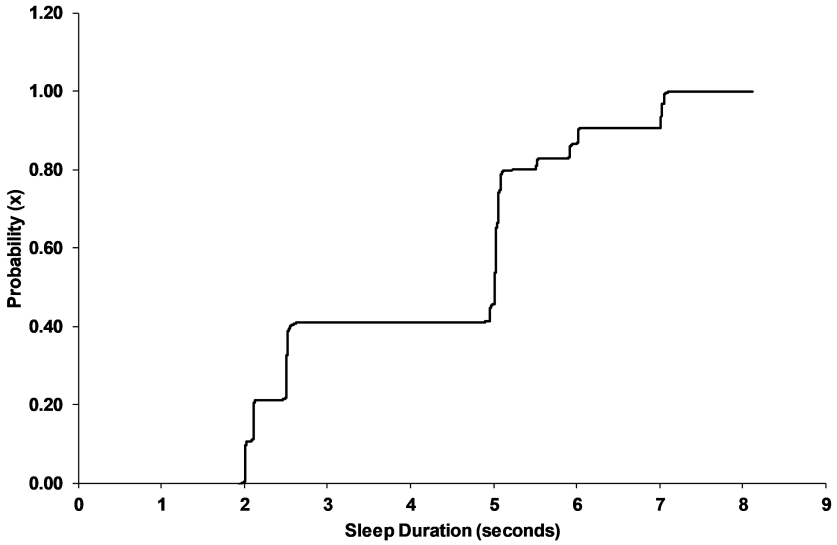


Figure 4.13: CDF plot showing the sleep-interval of IoT devices (Raspberry PIs and ESP8266) used in the test-bed observed in one-hour duration

In addition to publish intervals, sensors were also configured with varying length LAST WILL messages, where a LAST WILL message is transmitted to update the subscribed clients if the publishing client disconnects abruptly. The broker was configured with 1000 username/password combinations to authenticate the clients, and 1000 MQTT Access Control list (ACL) to authorise devices to publish/subscribe to various topics. This was done to add realistic loads on the broker CPU during authentication and authorisation. Anonymous login was disabled to allow only authenticated access to publish and subscribe topics. Algorithm 5 lists the steps implemented on IoT devices to send sensor updates through the MQTT protocol. The algorithm

implements the various normal MQTT states as shown in Figure 4.9, which includes initiating a TCP connections, setting valid credentials, sending a valid CONNECT and PUBLISH request and finally disconnecting. The physical sensor devices were interfaced to Raspberry Pi using the General Purpose Input Output (GPIO) pins. Once the variables are initialised the client code reads the data updated by the sensor and publishes data to a preconfigured topic. The $Sleep\_Time$ defines when to pause execution of the client code in order to introduce periodicity.

1: **Inputs:**
Sensor-Digital_Output,Broker_IP, Port,Sleep_Time
2: **Outputs:**
Sensor-Updates sent in MQTT messages
$Initialisations$: ;
$Pins \leftarrow GPIO\_Pins$;
$B \leftarrow Broker\_IP$;
$P \leftarrow Port$;
$U \leftarrow Username$;
$Pass \leftarrow Password$;
$WILL\_Msg \leftarrow Msg\_String$;
$WILL\_Topic \leftarrow Topic$;
$Topic \leftarrow topic\_string + sensorID$;
while *True* **do**
$Data \leftarrow Read\_Data(Pins)$;
$CONNECT(B, P, U, Pass, WILL\_Msg, WILL\_Topic)$;
$PUBLISH(Topic, Data)$;
$Disconnect()$;
$time.sleep(Sleep\_Time)$;
end

Algorithm 5: Algorithm for sending sensor data using MQTT protocol

Attack traffic was generated from a separate physical server connected to the network based on the various attack scenarios described previously. MQTT traffic was captured on the victim machine using the TCPDUMP tool (TCPDUMP, 2019) in pcap format, separately for normal and individual attacks to ease the labelling process for supervised classification. The TCPDUMP tool was configured to save the captured packets in 30MB chunks so as to reduce the processing load associated with feature extraction. Tshark tool (Wireshark, 2019) was deployed to extract specific packet parameters of the TCP and MQTT protocols and other packet parameters.

For generating time-window features, only incoming packets received by the broker were considered as time-window based detection (further elaborated in next section) is utilised to detect change in traffic patterns and the DoS attack emphasis is on the target machine as illustrated in Figure 4.14. In contrast, the features used to detect abnormal flows (further elaborated in next section) was based on bi-directional exchange of messages between the client and the broker. The Tshark command used for extracting flow parameters from pcap files is shown in Figure 4.15.

```
for /r %i in (*.pcap)
        do
           tshark -r %~nxi
                -T fields
                    -e frame.time_epoch
                    -e ip.src -e frame.len
                    -e tcp.stream
                    -e tcp.analysis.initial_rtt
                    -e tcp.time_delta
                    -e tcp.len
                    -e mqtt.msgtype
                    -e mqtt.len
                    -e mqtt.conflag.cleansess
                    -e mqtt.kalive
                    -e mqtt.clientid_len
                    -e mqtt.username_len
                    -e mqtt.passwd_len
                    -e mqtt.qos
                    -e mqtt.retain
                    -e mqtt.topic_len
                    -e mqtt.sub.qos
                    -e mqtt.willtopic_len
                    -e mqtt.willmsg_len
                    -e mqtt.conflag.qos
                -E header=y
                -E separator=,
                -E quote=d
                -E occurrence=a >  %~ni.csv
```

Figure 4.14: Tshark command used to extract specific packet features from pcap files to extract time-window based features

The various components of the MQTT traffic generator test-bed deployed in this study are illustrated in Figure 4.16.

4.4.2 MQTT Feature Extraction

Feature extraction is the most important component in the attack detection phases. As discussed in Section 2.5.2, targeted attacks require application or domain specific features to accurately detect them. In this section, the various features extracted from the MQTT protocol are discussed in detail. Feature extraction involves mapping (ϕ) of application payloads (input space (X)) to a real number vector space $\mathbb{R}^N$ with dimensions N as highlighted in Equation 4.1.

```
for /r %i in (*.pcap)
        do
           tshark -r %~nxi
                -Y "(ip.dst==192.168.20.16 && tcp.dstport==1883)"
                -T fields
                    -e frame.time_epoch
                    -e ip.src -e frame.len
                    -e tcp.stream
                    -e tcp.analysis.initial_rtt
                    -e tcp.time_delta
                    -e tcp.len
                    -e mqtt.msgtype
                    -e mqtt.len
                    -e mqtt.conflag.cleansess
                    -e mqtt.kalive
                    -e mqtt.clientid_len
                    -e mqtt.username_len
                    -e mqtt.passwd_len
                    -e mqtt.qos
                    -e mqtt.retain
                    -e mqtt.topic_len
                    -e mqtt.sub.qos
                    -e mqtt.willtopic_len
                    -e mqtt.willmsg_len
                    -e mqtt.conflag.qos
                -E header=y
                -E separator=,
                -E quote=d
                -E occurrence=a >  %~ni.csv
```

Figure 4.15: Tshark command used to extract specific packet features from pcap files to extract flow-based features

$$X \mapsto \phi(X) = (\phi_1(X), \phi_2(X), ..., \phi_N(X)) \qquad with \quad 1 \leq N \leq \infty \qquad (4.1)$$

As discussed in Section 2.5.3, for Application Layer DoS attack detection, features extracted at packet level and request level can have low detection capability since such DoS attacks use legitimate connection requests. However, monitoring a DoS attack using flow level and window based features can provide better detection capabilities, as DoS traffic can be distinguished from normal traffic by analysing multiple states in a connection or traffic characteristics over a period of time.

In this work, the MQTT protocol fields were extracted using Tshark tool. These fields were mapped to a vector space of MQTT statistical features aggregated at flow level and time-window. A custom-built feature generator module based on the PANDAS data analysis library (Pandas, 2018), was deployed to aggregate data based on flow-based and time-window based statistical data. Flow-based aggregation was achieved by using packet features observed in a network flow defined by five-tuple: Source IP, Destination IP, Protocol, Source Port and Destination Port. The time-window based aggregation was based on packet features observed in the specified time-window. The two feature groups are further explained in the following sections.

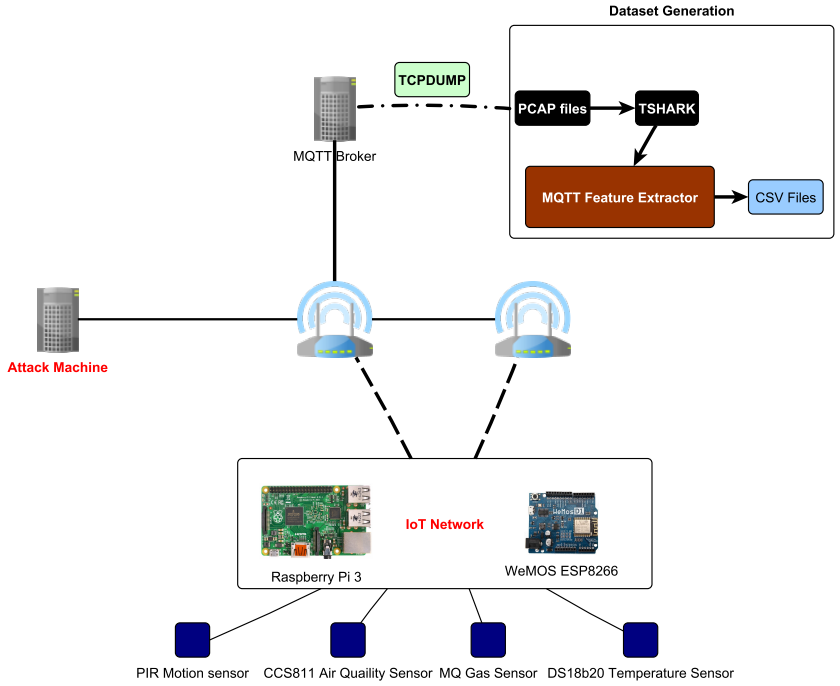


Figure 4.16: MQTT Dataset generation test-bed and the associated tools for feature generation

Flow-Based Features

In a packet switching network, network flow is defined by five-tuples listed earlier. No two active network flows will share the same five-tuple values, however packets belonging to the same flow will share the same five-tuple values. Flows provide information related to the interactions between the source and destination such as numbers of packets and bytes exchanged and can be useful in detecting attacks (Sperotto et al., 2010). To aggregate network traffic at flow-level all packets that belong to a flow are identified and aggregate features are extracted. In MQTT traffic, a flow can consist of TCP handshakes followed by the exchange of control packets. In this work, individual MQTT sessions are identified and aggregate features were extracted. The resulting dataset dataset consisted of flow instances identified by the feature vector $f_v = [f_1, f_2, f_3, , , f_n]$, where each feature vector represented a MQTT session parameter. Specifically, these aggregate/statistical flow features were calculated based on the patterns of MQTT sessions, such

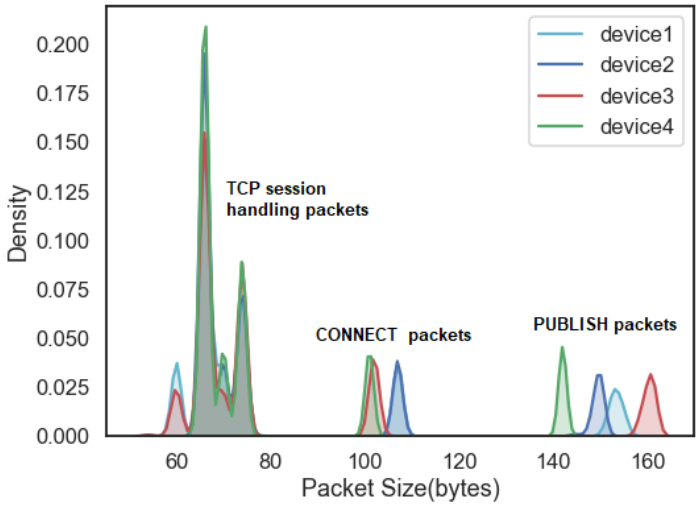
as count, size and field lengths.

The intuition behind using count based features is to detect changes in states of normal and attack traffic. For example, a normal MQTT traffic will contain a single CONNECT request, a PUBLISH request and a single DISCONNECT request as shown in Figure 4.9. An abnormal request might have multiple control packet requests in a single session as demonstrated in Section 4.2. To increase the detection capabilities of count based features, parameters such as flow duration, total number of packets in flow and inter-arrival time between packets in a flow were added, which can help detect abnormal sessions such as slow attacks with long flows, fragmentation attacks and deliberate delay of packets in a flow. By counting the number of control packets in a session, normal and abnormal sessions can be distinguished.

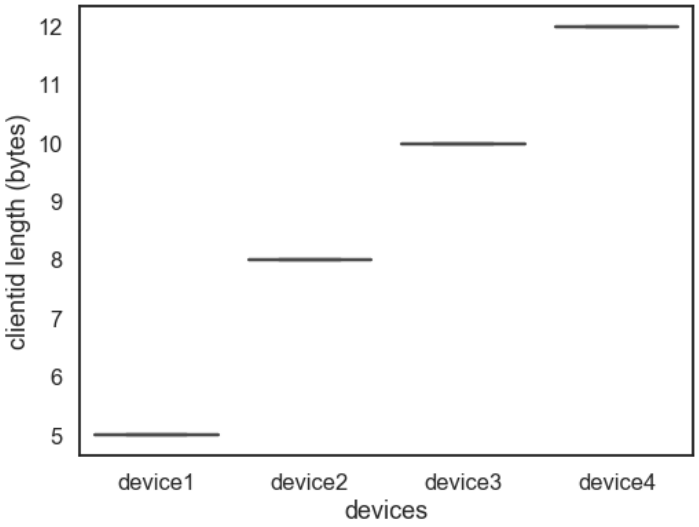
However, not all flows with multiple PUBLISH or SUBSCRIBE packets are malicious, as the protocol permits sending multiple PUBLISH and SUBSCRIBE packets in the same MQTT session. In order to distinguish between legitimate and abnormal connections, we explore size based features as MQTT packets contain several case-sensitive custom fields which are configured to predefined values. Furthermore, IoT devices are programmed to send specific messages which do not change frequently, unlike human users who have random behaviour in exchanging messages or accessing content. This would result in a distribution of MQTT packet and field sizes from the same IoT device to have a distinguishing distribution compared to other IoT devices in the network. Figure 4.17a shows the distribution of TCP, CONNECT and PUBLISH packets sizes in MQTT sessions observed from four IoT devices of the test-bed. The PUBLISH control packet distribution for each device shows clear dissimilarity between devices sending those messages. In addition, Figure 4.17b shows distributions of clientID length which can effectively help distinguish between clients. Using the assumption that attacks launched by external adversary will be unable to replicate the exact field lengths and MQTT payloads, aggregating the packet sizes and field lengths for individual flows can therefore be effectively leveraged to distinguish normal from attack traffic.

In addition, adversaries with limited access to MQTT network and using bruteforce techniques are likely to use random values or values obtained from other deployments for various protocol fields. Hence, size based features can provide useful distinction between normal and abnormal user behaviour.

Algorithm 6 shows the steps followed in generating a feature vector from various CSV files extracted from pcap files. The algorithm loops over the captured file to extract all the flows K in the file and number of packets in



(a)



(b)

Figure 4.17: (a) Density plot showing the distribution of TCP, CONNECT and PUBLISH packet sizes (bytes) in an MQTT flow observed on four IoT devices used in test-bed (b) Boxplot of client identifier lengths (bytes) observed on four IoT devices used in test-bed

each flow M. The counts and average of count-based and size-based features are calculated using the formulas 4.2 and 4.3. The process is repeated for all the flows and all the CSV files; finally returning a flow-based dataset represented using the feature vector.

1: **Inputs:**
CSV containing individual packet records extracted from pcap file using Tshark
2: **Outputs:**
feature vector,f_{counts}, f_{size}
while *ALL CSV files are read* **do**
while *ALL packets in CSV are read* **do**
$flowid \leftarrow tcp.stream, udp.stream$;
$K \leftarrow total\_number\_of\_flows$;
$M \leftarrow total\_number\_of\_packets\_in\_flow$;
end
for $i \leftarrow 1, K$ **do**
for $j \leftarrow 1, M$ **do**
if *feature[i][j] == feature[i][j+1]* **then**
$f_{count} = f_{count} + 1$;
$f_{total} = f_{total} + f_{value}$;
end
end
$f_{counts}[i] = f_{count}$;
$f_{size}[i] = f_{total}/f_{count}$;
end
$f_{counts} \leftarrow merge(f_{counts}[i])$;
$f_{size} \leftarrow merge(f_{size}[i])$;
end
$Return \leftarrow f_{counts}, f_{size}$;

Algorithm 6: Algorithm for generating flow-based feature vector

The count based features were calculated using the Formula 4.2.

$$f_{count} = \sum_{n=1}^{M} f_n \tag{4.2}$$

The size and field length were based on average bytes of the features, as calculated through the Formula 4.3.

$$f_{size} = \frac{1}{M} \sum_{n=1}^{M} f_n \tag{4.3}$$

Where, M is the total number of packets in the flow per captured pcap file. An example of the count features is presented using a sample packet data extracted using Tshark presented in Table 4.4.

Table 4.4: Sample packet data extracted using Tshark from MQTT traffic

frame.time_epoch	ip.src	ip.dst	ip.proto	frame.len	tcp.stream	tcp.srcport	tcp.dstport	mqtt.msgtype
1554601023.4332	192.168.20.19	192.168.20.16	6	74	250	35505	1883	
1554601023.4332	192.168.20.16	192.168.20.19	6	74	250	1883	35505	
1554601023.4340	192.168.20.19	192.168.20.16	6	66	250	35505	1883	1
1554601023.4350	192.168.20.19	192.168.20.16	6	101	250	35505	1883	
1554601023.4350	192.168.20.16	192.168.20.19	6	66	250	1883	35505	3
1554601023.4361	192.168.20.19	192.168.20.16	6	131	250	35505	1883	
1554601023.4361	192.168.20.16	192.168.20.19	6	66	250	1883	35505	2
1554601023.4371	192.168.20.16	192.168.20.19	6	70	250	1883	35505	
1554601023.4376	192.168.20.19	192.168.20.16	6	66	250	35505	1883	14
1554601025.4395	192.168.20.19	192.168.20.16	6	68	250	35505	1883	
1554601025.4399	192.168.20.19	192.168.20.16	6	66	250	35505	1883	

The flow duration of the flow 250 is calculated as $frame.time\_epoch =$ 1,554,601,025.4399 (seconds elapsed since the epoch), $frame.time\_epoch =$ 1,554,601,023.4332 and $flow\_duration = 2.0067$ seconds. Similarly $number\_of\_packets$ in the flow 250 are 11. The control packets are counted based on the *mqtt.msgtype* field, which indicates the type of control packet.

The average frame length of flow 250 can be calculated as $avg\, frame\_len$ = 74+74+66+101+66+131+66+70+66+68+66/11 = 77.09 and, $avg\, mqtt\_len$ = 33+63+2/11 = 8.902. A sample of resultant dataset is presented in Table 4.5.

Table 4.5: Sample dataset containing n flows represented by k features

	feature_1	feature_2	....	feature_k
flow_1	value(1,1)	value(1,2)	...	value(1,k)
flow_2	value(2,1)	value(2,2)	...	value(2,k)
flow_3	value(3,1)	value(3,2)	...	value(3,k)
..	..	..	..	..
flow_n	value(n,1)	value(n,2)	...	value(n,k)

The count based features were based on the MQTT session such as counts of number of packets, number of control packets and number of QoS packets

that belonged to the same flow. Packet size and field length features were based on the captured IP packets and the various MQTT field lengths, as illustrated in Table 4.6. Only packet meta-data was utilised to generate the features, instead of deep inspection of the payload. Hence, this feature extraction method can also be utilised on flows with encrypted MQTT payloads. One of the challenges in extracting data for subscription flooding attacks was that MQTT allows multiple subscription topics as well as multiple subscription payloads to be piggybacked in the same MQTT packet, as shown in Figure 4.18. In order to measure the accurate number of subscription requests per flow, the feature generation module counts the individual subscription requests and the number of topics in the request as separate request.

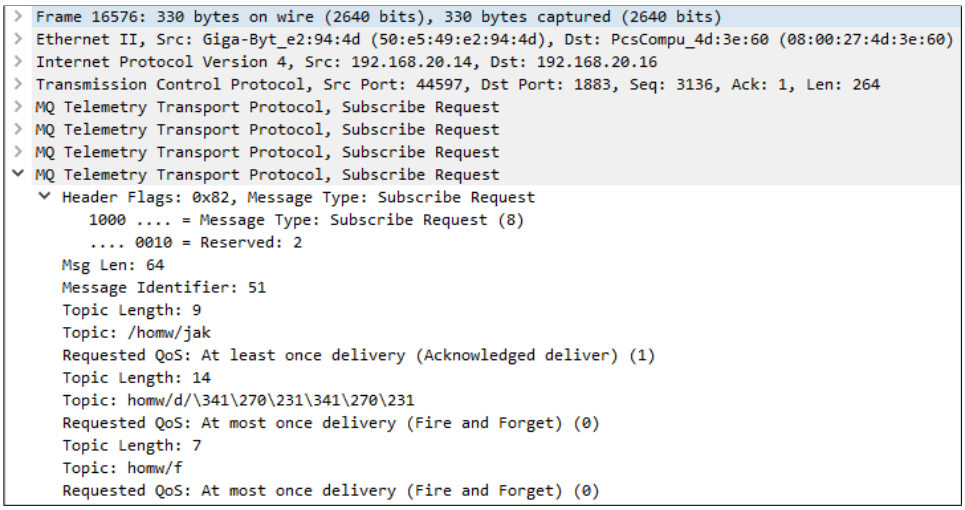


Figure 4.18: Subscribe flooding attack packet displayed in Wireshark showing multiple SUBSCRIBE requests in the same packet

Time-Window Based Features

In addition to the flow-based features, sliding window based features were also analysed for MQTT attack detection. A network traffic can be modelled as a time series and can be used to detect anomalies that cause variation in the time-series parameters. Sliding windows method that was used to analyse network traffic has advantages over fixed landmark points, to extract statistical network summaries (Golab, DeHaan, Demaine, Lopez-Ortiz, & Munro, 2003). Commonly used measures to define the sliding window size

Table 4.6: Proposed MQTT DoS detection features (feature type N: Numeric, B: Binary)

S.No	Feature	Description	Type
		MQTT Session Statistical Features	
1	flow_duration	Duration of Flow	N
2	pkt_in_flow	No. of Packets in a Flow	N
3	Connect_Command	No. of CONNECT packets in Flow	N
4	Publish_Message	No. of PUBLISH packets in the Flow	N
5	Subscribe_Request	No. of SUBSCRIBE packets in the Flow	N
6	Disconnect_Req	No. of DISCONNECT packets in the Flow	N
7	Ping_Request	No. of PING packets in the Flow	N
8	Subs_Qos0	No. of SUBSCRIBE packets with QoS 0 in the Flow	N
9	Subs_Qos1	No. of SUBSCRIBE packets with QoS 1 in the Flow	N
10	Subs_Qos2	No. of SUBSCRIBE packets with QoS 2 in the Flow	N
11	Pub_Qos0	No. of PUBLISH packets with QoS 0 in the Flow	N
12	Pub_Qos1	No. of PUBLISH packets with QoS 1 in the Flow	N
13	Pubs_Qos2	No. of PUBLISH packets with QoS 2 in the Flow	N
14	Will_Qos0	No. of WILL messages with QoS 0 in the Flow	N
15	Will_Qos1	No. of WILL messages with QoS 1 in the Flow	N
16	Will_Qos2	No. of WILL messages with QoS 2 in the Flow	N
17	tcp.time_delta	Time between packets in the flow	N
		MQTT Packet and Field Length Features	
18	frame.len	Avg. Frame Length in the Flow	N
19	tcp.len	Avg. TCP length in the Flow	N
20	mqtt.clientid_len	ClientID length in the Flow	N
21	mqtt.username_len	username length in the Flow	N
22	mqtt.passwd_len	password length in the Flow	N
23	mqtt.willtopic_len	WILL Topic length in the Flow	N
24	mqtt.willmsg_len	WILL Message length in the Flow	N
25	mqtt.len	Avg. MQTT packet length	N
26	mqtt.topic_len	Avg MQTT Topic Length	N
27	mqtt.kalive	MQTT Keep Alive interval	N
28	mqtt.conflag.cleansess	CleanSession Flag Set/Unset	B

are count-based and time-based (Golab et al., 2003), where count-based windows observe the last N packets; and time-based windows observe all the packets arrived in the last t time units. The number of packets in a time-based window vary based on the arrival of packets in the time interval. Hence, statistical features based on it allows detecting anomalies that cause variations in the observed time series (Linda, Vollmer, & Manic, 2009). For an efficient detection of anomalies in the time series, a sliding window approach with a time-based window was adopted to extract features from the network traffic, similar to what is proposed in (Linda et al., 2009). Both count based and size based features were extracted from network traffic time-series. The count based features were calculated by frequencies of features observed in all the packets in the window. Size based features are based on average value of the feature observed in the window, and is calculated as:

$$f_{size} = \frac{1}{M} \sum_{n=1}^{n=M} V_i \tag{4.4}$$

where M is number of packets in the window and V is the value of the feature in the packet.

Only packets flowing in one direction in MQTT traffic flow are considered because the DoS attack emphasis is on flooding the broker with excessive traffic and depleting the server resources. In the dataset preparation step the PANDAS rolling function was used to calculate the statistics of various features in the window. The features that were extracted are presented in Table 4.7. The sliding-window algorithm loops over the captured file to extract all the overlapping windows K in the file and the number of packets in each time-window M. The counts and average of count-based and size-based features are calculated and the process is repeated for all the time windows in all the CSV files; finally returning a time window-based dataset represented based on the feature vector. Algorithm 7 describes the steps followed in generating a time-window based datasets. A sample of resultant dataset is presented in Table 4.8.

4.4.3 Detecting Framework Classification Techniques

The attack detection module of the framework is a machine learning (ML) based detection system. Statistical flow features and time-window based features extracted from MQTT network traffic serve as input to the classification system and help differentiate normal from attack traffic, as well as inter-attack classification. The task of differentiating between the var-

1: **Inputs:**
CSV containing individual packet records extracted from pcap file using Tshark
2: **Outputs:**
feature vector,f_{counts}, f_{size}
while *ALL CSV files are read* **do**
while *ALL packets in CSV are read* **do**
$time \leftarrow frame.time\_epoch$;
$w \leftarrow time - window(w)$;
$K \leftarrow total\_number\_of\_overlapping\_windows$;
$M \leftarrow total\_number\_of\_packets\_in\_time - window$;
end
for $i \leftarrow 1, K$ **do**
for $j \leftarrow 1, M$ **do**
if *feature[i][j] == feature[i][j+1]* **then**
$f_{count} = f_{count} + 1$;
$f_{total} = f_{total} + f_{value}$;
end
end
$f_{counts}[i] = f_{count}$;
$f_{size}[i] = f_{total}/f_{count}$;
end
$f_{counts} \leftarrow merge(f_{counts}[i])$;
$f_{size} \leftarrow merge(f_{size}[i])$;
end
$Return \leftarrow f_{counts}, f_{size}$;

Algorithm 7: Algorithm for generating time-window-based feature vector

Table 4.7: Time-Window based features

S.No	Feature	Description
		MQTT Count Based Features
1	Pktcount	Number of packets in 2s window
2	pkt_delta	Avg Inter-arrival Time window
3	Unique_SRC	Number of unique source Ips in window
4	Unique_streams	Number of unique flows in windows
5	CleanSessionSet	Number of packets with clean session set in window
6	CleanSessionNSet	Number of packets with clean session unset in window
7	RetainSet	Number of packets with Retain flag set in window
8	RetainNSet	Number of packets with Retain flag unset in window
9	Connect_Command	Number of Connect Control packets in window
10	Publish_Message	Number of Publish Control packets in window
11	Subscribe_Request	Number of Subscribe Control packets in window
12	Disconnect_Req	Number of Disconnect Control packets in window
13	Ping_Request	Number of Ping Control packets in window
14	Subs_qos0	Number of QoS0 Subscriptions in window
15	Subs_qos1	Number of QoS1 Subscriptions in window
16	Subs_qos2	Number of QoS2 Subscriptions in window
17	Pub_qos0	Number of QoS0 Publish messages in window
18	Pub_qos1	Number of QoS1 Publish messages in window
19	Pub_qos2	Number of QoS2 Publish messages in window
20	will_qos0	Number of QoS0 WILL messages in window
21	will_qos1	Number of QoS1 WILL messages in window
22	will_qos2	Number of QoS2 WILL messages in window
		MQTT Size Based Features
23	AVG_FrameLen	Avg Frame length in window
24	AVG_tcpdelta	Avg tcpdelta in a flow per window
25	AVG_tcpseglen	Avg tcp segment length in window
26	AVG_msglen	Avg MQTT message length in window
27	AVG_keepalive	Avg keepalive interval in window
28	AVG_Clientidlen	Avg Clientid length in window
29	AVG_Usrnamelen	Avg User name length in window
30	AVG_PasswordLen	Avg password length in window
31	AVG_WillTopicLen	Avg WILL topic length in window
32	AVG_WillMsgLen	Avg WILL message length in window
33	AVG_TopicLen	Avg topic length in window

ious flooding attacks will enable effective counter measures to be applied to thwart such attacks. The use of legitimate requests in Application layer

Table 4.8: Sample dataset containing n windows represented by k features

	feature_1	feature_2	....	feature_k
window_1	value(1,1)	value(1,2)	...	value(1,k)
window_2	value(2,1)	value(2,2)	...	value(2,k)
window_3	value(3,1)	value(3,2)	...	value(3,k)
..	..	..	..	..
window_n	value(n,1)	value(n,2)	...	value(n,k)

DoS attacks can pose a significant challenge to the detection framework in differentiating between normal and attack network flows. Furthermore, broken sessions due to loss of network connectivity can potentially increase the challenge to detect attacks. Hence, the ML algorithms selected in the framework should be sensitive to small variations in feature vector values, to accurately classify network traffic. For example, a normal flow can have the following communication sequence in the IoT-MQTT message exchange: *TCP handshake + Connect + Publish/Subscribe + Disconnect*, whereas, an attack communication sequence can be: *TCP handshake + Connect* or *TCP handshake + Connect + multiple Publish/Subscribe + Disconnect.*

In this study, three fundamentally different machine learning approaches namely, average one-dependence estimator (AODE), C4.5 decision trees and artificial neural network (ANN) were integrated into the detection framework. The AODE classifier is based on the Naive Bayes algorithm, which adopts a probabilistic approach in estimating the distribution of network traffic classes. This allows the AODE classifier to converge quickly and requires fewer training samples. C4.5 decision tree (DT) classifier recursively divides the feature space to learn the decision boundary that separates the classes. One of the advantages of the DT classifiers is that they detect feature interactions effectively and the model is easy to interpret. MLP belongs to the ANN family of classifiers that try to imitate the biological brain by creating a network of neurons grouped into input, hidden and output layers. The MLP classifier uses a deterministic model to iteratively learn the explicit decision boundary by adjusting the weights of the neural network based on the training samples. Even though MLP classifiers are memory and CPU-intensive, they are effective on large datasets with complex structures. The selected classification algorithms were adopted in detecting MQTT based DoS attacks because of the following reasons:

- They are capable in detecting small variations expected in Application layer DoS attacks by using effective techniques in learning the decision boundaries between classes,
- They can identify complex interactions between the MQTT features to detect anomalies,
- As the IoT network is expected to scale rapidly producing large volumes of data, the selected classifiers can also be scaled up to handle large datasets.

The steps followed in the detection framework to classify MQTT traffic are illustrated in the Figure 4.19 and the three classifiers adopted in MQTT attack detection are discussed below:

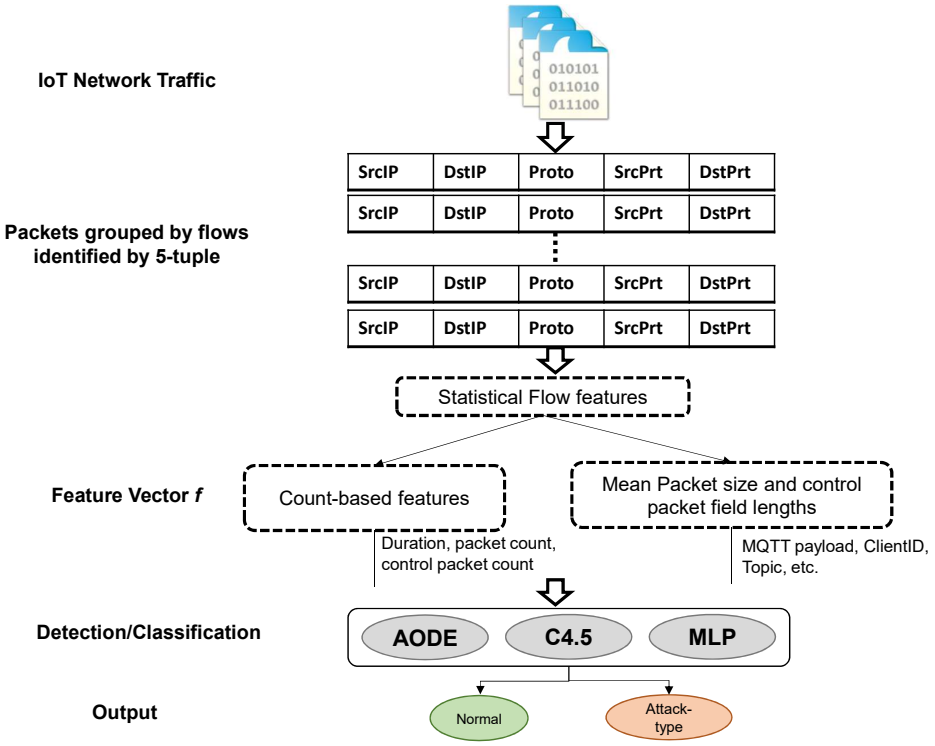


Figure 4.19: Detection framework work-flow in detecting MQTT attacks using flow-based features

AODE Classifier (Webb, Boughton, & Wang, 2005): The AODE classifier is a variant of the Naïve Bayes classifier that estimates the probability

of the class of each output variable Y given a set of input features x_1,...x_n. It is based on a simple Naïve Bayes classifier which relies on the assumption of independence of attributes. Assuming that all attributes are independent given the class, then Naïve Bayes can be defined as:

$$\arg\max_{c \in C} P(C) \prod_{i=1}^{n} P(a_i|C) \tag{4.5}$$

Where C is class label and a is the attribute. In this scenario, the computation cost is reduced, however the performance of Naïve bayes decreases if the dependency between the attributes is high (Koc & Carswell, 2015). To counter this effect the AODE classification technique uses a weaker independence assumption to achieve a higher accuracy rate compared to Naïve bayes. AODE classifiers are simple to implement and provide high accuracy in classifying cyber-attacks specifically DoS attacks as highlighted by (Baig et al., 2020) .

Decision Trees (DT): The decision tree-based algorithms build training datasets into the tree structures, applying the information entropy principle. Each branch of the tree represents an association between the feature vector and the class label. C4.5 is one of widely used DT method which recursively partitions the training dataset by choosing the most effective features to differentiate between the classes. In the first step, C4.5 identifies the best feature that can divide the data instances. In further steps, child nodes are created to divide the instances into subclasses. The attributes selected in each division point in the tree is based on the largest information gain using the best attribute. Entropy is used as measure of information gain (Hssina, Merbouha, Ezzikouri, & Erritali, 2014), calculated as follows:

$$H(X) = -\sum_{i=i}^{n} p_i log_2(p_i) \tag{4.6}$$

Multi-Layer Perceptron (MLP): MLP is a type of feed-forward artificial neural network (ANN) that comprises multiple nodes known as artificial neurons, emulating the biological neurons of brain. The nodes in the MLP are grouped as input layers representing the input features, hidden layer and an output layer. In MLP the nodes of a given layer use activation functions to control the node's output, as well as to serve as an input for the next node. The nodes in MLP are connected by weights which are tuned by using back-propagation algorithms, that adjust the weights to reduce the error between outputs and expected results where, the error is calculated as

follows (Dua & Du, 2016):

$$e_i(n) = t_i(n) - y_i(n) \tag{4.7}$$

Where $t_i(n)$ is the expected output and $y_i(n)$ produced output value of the instance n and output node i.

4.5 Summary

In this chapter the MQTT Attack model and detection model were presented. The various threats to MQTT protocol with an emphasis to DoS attacks were described. The MQTT protocol features that can be exploited to cause DoS attack scenario were subsequently discussed. The test-bed used to evaluate the DoS attack scenarios along with the tools employed in this study were elaborated. Finally the DoS attack detection framework used in this study was presented with steps explaining the traffic generation, feature extraction, ML algorithms used in the framework. The following chapters will present the results and discussion of results for the experiments conducted.

Chapter 5

Results

This research conducted an analysis of the impact of Application Layer DoS attacks on MQTT protocol based IoT deployments and the effectiveness of MQTT based statistical features in detecting such attacks. As per the research procedure described in Section 3.3, this chapter will present the results for the research processes **RP-3.1** and **RP-3.2**. The results will provide quantitative measures of the dependent variables (DV) observed due to the application of treatment on the various independent variables (IV) as described in Section 3.4. The results are divided into DoS impact assessment and detection framework performance. The DoS impact assessment is further divided into system performance and delay measurements.

5.1 DoS Impact Assessment

In order to show the impact of the proposed MQTT based Application Layer DoS attack scenarios presented in Section 4.2, the DoS impact on Single-CPU, Six-CPU, and Load Balanced (LB) deployment with three different brokers has been presented. The DoS impact on the broker system performance has been presented first followed by the impact on message delay and publish rate. The impact on the system performance was measured with no-load conditions to highlight the impact on the broker CPU, memory and bandwidth utilisation. The message delay was measured with 1,000 clients publishing at an average rate of 500 messages per second together. Subsequently the delay due to DoS attacks was measured for the three QoS supported by the MQTT protocol. The following section presents the broker system performance measured with Single-CPU, Six-CPU and LB configurations. Table 5.1 lists the various IV treatments and the observations on

the DV presented in this section.

Table 5.1: IV treatments and observations on DV in DoS impact assessment

DoS Scenario	**Treatment**	**Deployment scenarios/ configurations**	**Broker**	**Observations**
BF1 (IV1)	Sleep-Interval, Attack Threads	Single-CPU, Six-CPU, LB	Mosquitto, VerneMQ, EMQ	CPU idle (%), Packets per Second (PPS) (DV1)
BF2 (IV2)	CONNECT Delay			CPU idle (%), PPS, Half-Open TCP sessions (DV1)
BF3 (IV3)	WILL Payload Size, Attack Thread			Bandwidth Utilisation (MB/s), CPU Idle (%) (DV1)
IAUTHS (IV4)	Subscription Loop			CPU Idle (%) (DV1)

5.1.1 Single-CPU deployment

The Single-CPU deployment configuration was utilised to compare the performance of the three brokers with a single CPU core as described in Section 4.3.

Impact of Basic CONNECT Flooding (BF1) Attack

The basic CONNECT flooding attack (IV1) attempts to send CONNECT control packets with invalid credentials at a rapid rate to consume broker resources in authenticating individual requests. A separate attack machine was deployed to launch the attack and the impact was measured on the three brokers.

The treatment applied to IV1 was the manipulation of sleep-interval time and the number of attack threads. The experiment consisted of five iterations. In each iteration, the number of attack threads were kept fixed and the sleep-interval was decreased gradually from 0.5 to 0 seconds after sending a pre-determined number of attack requests with each sleep-interval as described in Table 5.2. Higher number of attack packets were sent with lower sleep-intervals to achieve a similar measurement durations otherwise a higher sleep-interval would take longer to complete compared to lower sleep-interval setting. Five independent iterations were performed by incrementing the attack threads by one. The attack threads were only incremented in

steps of one due to the limitation in using large number of program threads as this would increase the resource contention among threads, thus reducing the effectiveness of the DoS attack. It was also observed that the attack packets generated by the DoS attack initially increased with the increase of threads, but reduced after reaching three attack threads. Hence only five increments of attack threads were tested in this research. The impact of applying the two treatments to IV1 was measured in terms of CPU idle percentage (DV1), which indicates the CPU utilisation of the broker server due to the DoS attack. A lower CPU idle percentage indicates a high CPU utilisation and a high CPU idle percentage indicates a low CPU utilisation. The impact was individually measured on each of the three MQTT brokers. The attack volume measured in terms of the number of attacks packets per second (PPS) received by the broker was also measured to provide a supporting evidence to explain the behaviour of DoS attack on the broker CPU utilisation.

Table 5.2: Sleep Intervals and the number of attack requests sent in each step

Sleep-Interval (seconds)	**Number of attack requests sent**
0.5	100
0.1	1000
0.01	5000
0.005	10000
0	200000

Figure 5.1 shows the impact of reducing the sleep-interval (increasing the arrival rate) and increasing the number of attack threads on the CPU utilisation of the Mosquitto broker. The results indicate that, reducing the sleep-interval and increasing the attack threads caused an increase in the CPU utilisation. Especially with the sleep-intervals of 0.01 and 0.005 seconds, increasing the attack threads caused a higher CPU utilisation, which is indicated by the wide separation in the CPU idle percentage. With the sleep-interval of zero seconds and single attack thread, the CPU utilisation fluctuated when compared to using higher attack threads as indicated by the peaks with sleep-interval of zero in Figure 5.1.

The CPU utilisation results can be further explained by the attack volume measured during the experiment. Figure 5.2 shows the average number of attack packets and the maximum number of attack packets that were received by the Mosquitto broker during the BF1 attack. The broker received

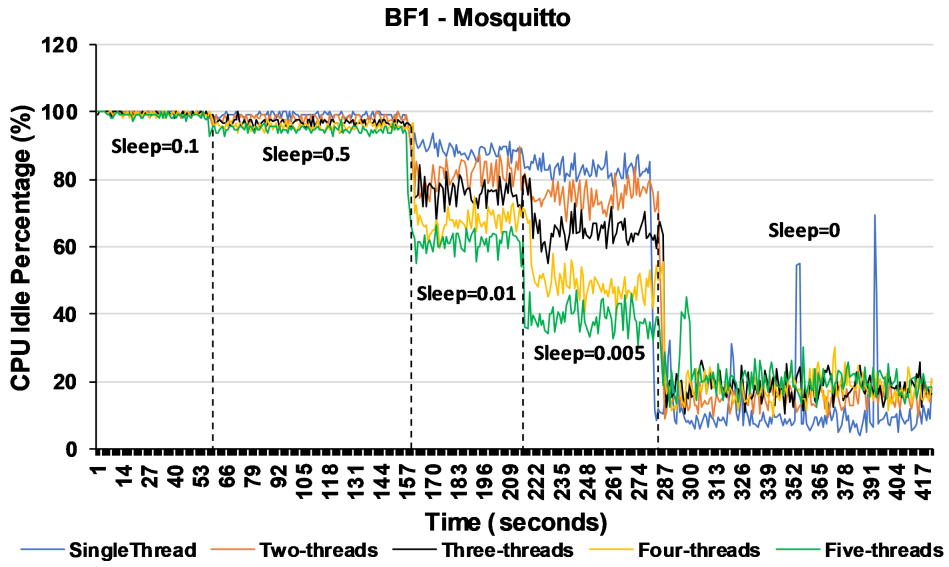
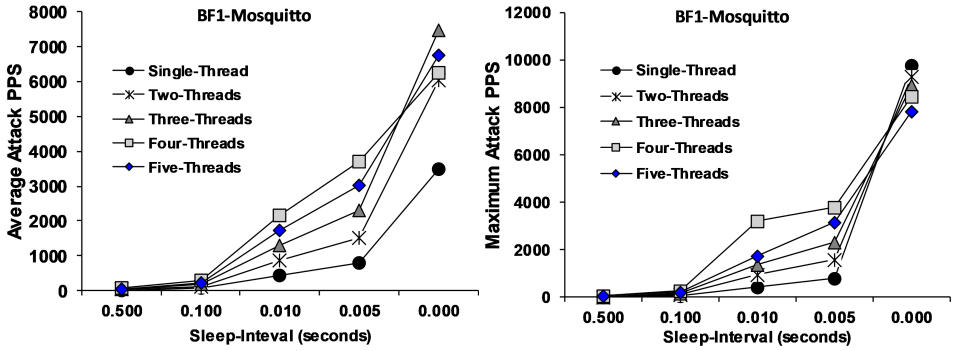


Figure 5.1: Impact of BF1 attack on Mosquitto Broker CPU utilisation while reducing sleep-interval and increasing the number of threads



(a) (b)

Figure 5.2: Attack packets received by the Mosquitto broker with increasing attack threads and decreasing sleep intervals (seconds) for BF1 attack category. Sub-Figure (a) shows the average attack packets received and Sub-Figure (b) shows the maximum attack packets received

higher average packets per second (PPS) during the attack when sleep-intervals were reduced and the number of attack threads were increased,

thus resulting in a higher CPU utilisation. With the sleep-interval of zero, the three attack threads produced an average 7,500 PPS which was the highest among the five thread settings. However, with a single attack thread and a zero sleep-interval, the Mosquitto broker received a lower average attack PPS, but contained a higher maximum attack PPS (9,700 PPS) compared to other attack thread settings resulting in a higher CPU (>90%) utilisation with fluctuations which were observed in Figure 5.1. In contrast, the other thread settings with the sleep-interval of zero generated similar average and maximum PPS resulting in similar CPU utilisation performance during the BF1 attack.

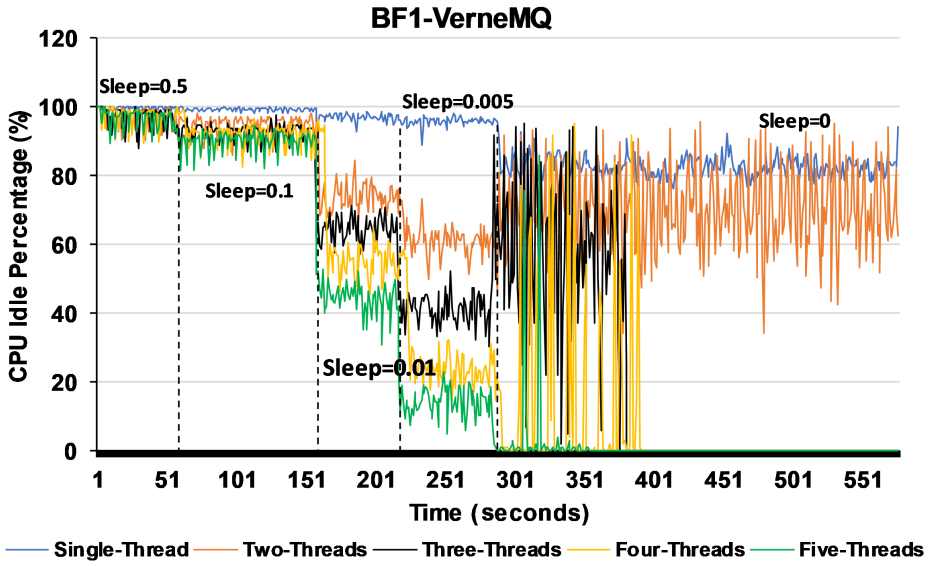
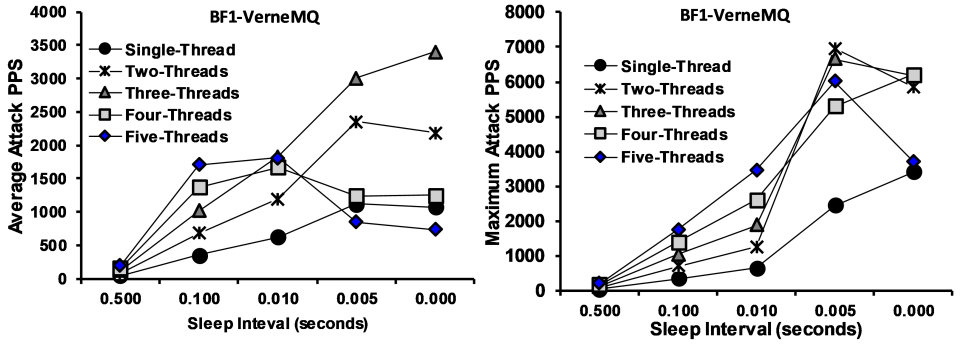


Figure 5.3: Impact of BF1 attack on VerneMQ Broker CPU utilisation while reducing sleep-interval and increasing the number of threads

The overall impact on VerneMQ broker had similarities to impact observed on the Mosquitto broker, as reducing the sleep-interval and increasing the attack threads caused higher CPU utilisation. However, with single and two thread settings the impact was negligible as shown by the drop in the CPU idle percentage in Figure 5.3. With three, four and five attack threads, the impact on VerneMQ broker CPU utilisation was higher than with single and two attack threads. Specifically, with the sleep-intervals of 0.01 and 0.005 seconds when compared to using single and two attack threads. In addition, with a zero second sleep-interval and three, four and five attack

threads, the CPU idle percentage reached zero (100% CPU utilisation) after an initial fluctuation of CPU utilisation.



(a) (b)

Figure 5.4: Attack packets received by the VerneMQ broker with increasing attack threads and decreasing sleep intervals (seconds) for BF1 attack category. Sub-Figure (a) shows the average attack packets received and the Sub-Figure (b) shows the maximum attack packets received

The VerneMQ broker received higher average attack packets with the increase of attack threads until three threads and reduces with four and five attack threads as shown in Figure 5.4. Similarly, the maximum attack packets received by the broker also increased with the increase of attack threads until four threads. The maximum packets reduced for the five attack threads.

This indicates that the CPU utilisation of the VerneMQ broker was affected more by the increase of the number of attack threads compared to the increase of the number of attack packets received by the broker, as high CPU utilisation was observed with the five attack threads compared to all the other attack thread settings, even though it produced lower attack volume compared to other thread settings.

In addition to the CPU exhaustion on VerneMQ broker due to CONNECT packet flooding, the use of non-ASCII characters in client identifier to generate a malformed MQTT CONNECT packets for BF1 attack category, caused considerable increase in the broker memory, especially with four and five threads as shown in Figure 5.5. This attack has the potential to cause memory exhaustion on the broker.

Compared to all the brokers, EMQ broker's CPU utilisation recorded the maximum impact during BF1 attack as the CPU idle percentage reached

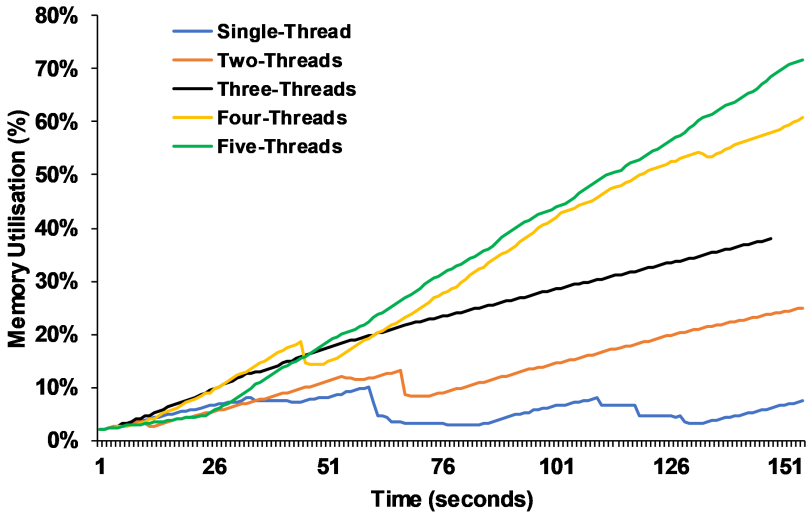


Figure 5.5: Impact of BF1 attack using malformed (non-ASCII characters in client identifier) packets on memory utilisation of the VerneMQ broker with the increase of attack threads

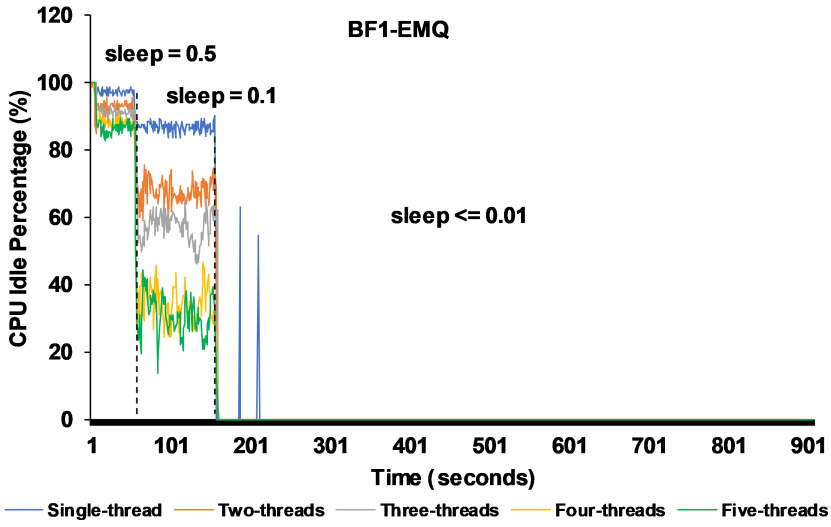
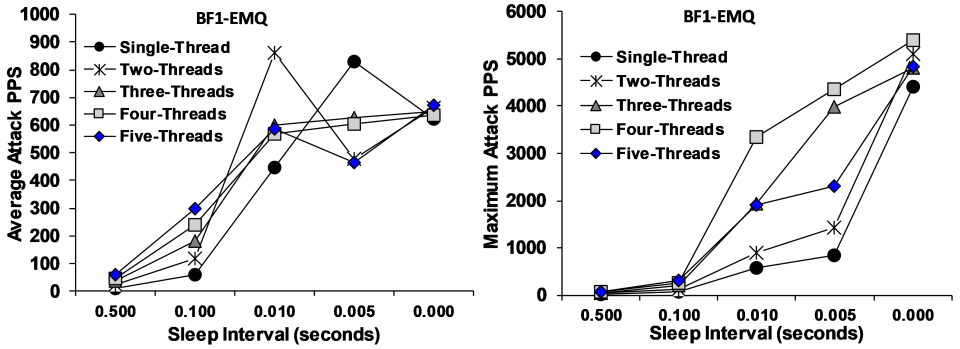


Figure 5.6: CPU Idle percentage during the BF1 attack with various sleep intervals and number of threads on the EMQ broker

zero percentage with every choice of attack thread and a sleep-interval lower than 0.01 seconds as shown in Figure 5.6.

The EMQ broker also received lower average attack PPS when compared all the other brokers as none of the thread settings achieved more than 1,000 average PPS. However the broker received a maximum of more than 4000 attack PPS with zero sleep-interval and with all the attack thread settings as shown in Figure 5.7.



(a) (b)

Figure 5.7: Attack packets received by EMQ broker with increasing attack threads and decreasing sleep intervals (seconds). Sub-Figure (a) shows the average attack packets received and the Sub-Figure (b) shows the maximum attack packets received

Impact of Delayed CONNECT Flooding (BF2) Attack

The delayed CONNECT flood (IV2) attack aims to consume available TCP connections by opening multiple TCP sessions and delaying the CONNECT control packet. In addition, the CPU resources are consumed by sending a CONNECT request with invalid credentials. The treatment applied to IV2 was CONNECT delay (seconds), which causes the execution of the attack thread to be paused for the specified period. The MQTT PAHO client library was modified to introduce delays in sending the CONNECT request. Four iterations were performed with delays set to 0.01, 0.1, 0.5 and 1 seconds respectively. As described in Algorithm 2, multiple threads are launched, with each thread configured to delay the sending of the CONNECT request. Since the focus of the attack was to observe the impact of launching multiple connections requests with delayed CONNECT on the broker CPU utilisation and the half-open TCP sessions, a maximum of 250 attack were only

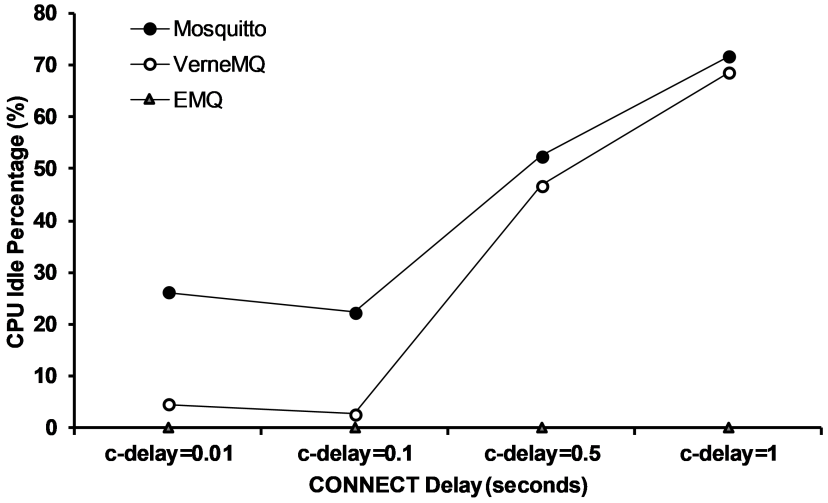


Figure 5.8: Average CPU Idle percentage of three brokers with various delay settings used in BF2 attack using 250 attack threads

launched. The maximum attack threads were fixed to 250 as the attack was launched from a single attack machine with limited resources in launching multiple threads. The delay settings that resulted in the highest (maximum 250) half-open TCP sessions was assessed in this experiment. The impact of applying the CONNECT delay treatment on the IV2 was measured in terms of the CPU utilisation and the number of available TCP sessions. The CPU utilisation and the half-open sessions (number of established TCP connections) on the broker was recorded for a five minute period during the attack and the average CPU utilisation and the average half-open sessions for the chosen delay setting is reported in this section.

The measurement of CPU utilisation during the attack, showed that a smaller delay caused a higher CPU utilisation in all three brokers indicating CPU exhaustion especially in VerneMQ and EMQ brokers as shown in Figure 5.8. However, a higher delay had less impact on Mosquitto and VerneMQ brokers when compared to EMQ broker which had zero CPU idle percentage for all the delay settings.

Even though the impact on CPU utilisation was less with a higher delay, the number of half-open TCP sessions increased considerably with the increase in delay, as shown in Figure 5.9. With 250 attack threads, all the three brokers recorded higher half-open TCP sessions for a one second CONNECT delay, compared to the sattack launched using 0.01 second delay.

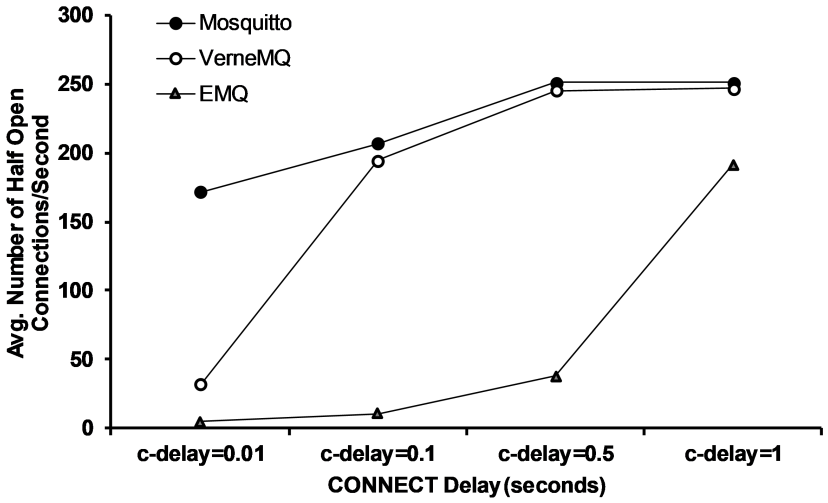


Figure 5.9: Average number of half-open MQTT sessions per second observed on the three brokers with various delay settings used in BF2 attack using 250 attack threads

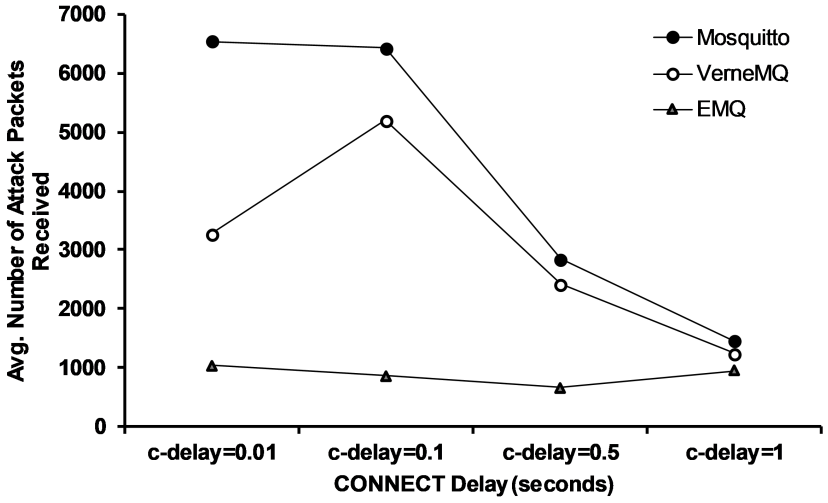


Figure 5.10: Average number of attack packets received by three brokers with various delay settings used in the BF2 attack using 250 attack threads

In addition, the average attack PPS received by the brokers also explains the reason for low CPU utilisation with higher CONNECT delay. Figure

5.10 shows the average number of attack PPS received by the three brokers for five minute period with the various delay settings. As the delay is increased, the Mosquitto and VerneMQ brokers received fewer attack packets compared to the EMQ broker, which showed marginal variation with the increase in delay.

These results also indicate that choosing a smaller delay makes the BF2 attack behaviour identical to BF1 attack and by increasing the delay a more stealthier attack can be launched for exhausting the TCP resources. In addition, the TCP connection limits on individual broker can be affected using fewer attack packets resulting DoS for new connections.

Impact of CONNECT and WILL Message Flooding (BF3) Attack

The BF3 attack is a modified version of basic flooding attack BF1, with a piggybacked WILL message. The attack attempts to consume CPU and bandwidth resources of the broker by adding a payload to the CONNECT packet. The treatments applied to BF3 attack (IV3) was the manipulation of WILL payload size and the number of attack threads. The experiment consisted of five iterations. In each iteration, the number of attack threads were fixed and the payload size was increased gradually from 50 to 43,000 bytes. Five independent iterations were performed by incrementing the attack threads by one. The impact of applying the two treatments to BF3 attack was measured in terms of bandwidth utilisation (MB/s) and CPU idle percentage (CPU utilisation - DV1).

Figure 5.11 shows the impact of increasing the number of threads and WILL payload size on the attack bandwidth measured on the Mosquitto broker. The results indicate that varying the number of threads did not impact the amount of bandwidth consumed on the broker. However, BF3 attack launched using a single thread had greater impact on CPU utilisation when compared to two and more threads as shown in Figure 5.12. In addition, the CPU utilisation was higher with lower payload size compared to the higher payload size.

Similar to Mosquitto broker, varying the number of threads did not impact the amount of bandwidth utilised during the attack on the VerneMQ broker as shown in Figure 5.13. The bandwidth utilisation increased with the increase of the payload size but only slightly increased with the increase of attack threads. However, the CPU idle percentage dropped close to zero percentage with four and five attack threads indicating CPU exhaustion, as shown in Figure 5.14.

The EMQ broker recorded lower bandwidth consumption with maxi-

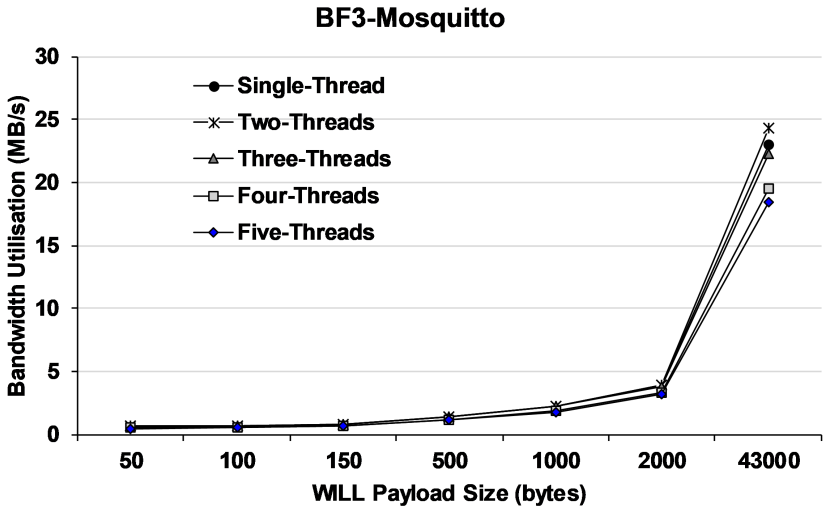


Figure 5.11: Impact of number of increasing attack threads and WILL message payload size on Mosquitto Broker bandwidth utilisation during the BF3 attack

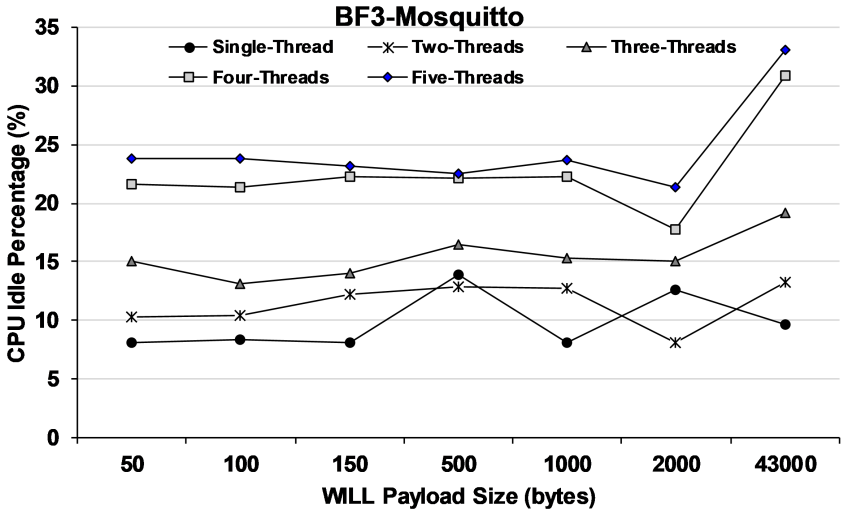


Figure 5.12: Impact of number of increasing attack threads and WILL message payload size on Mosquitto Broker CPU idle percentage during the BF3 attack

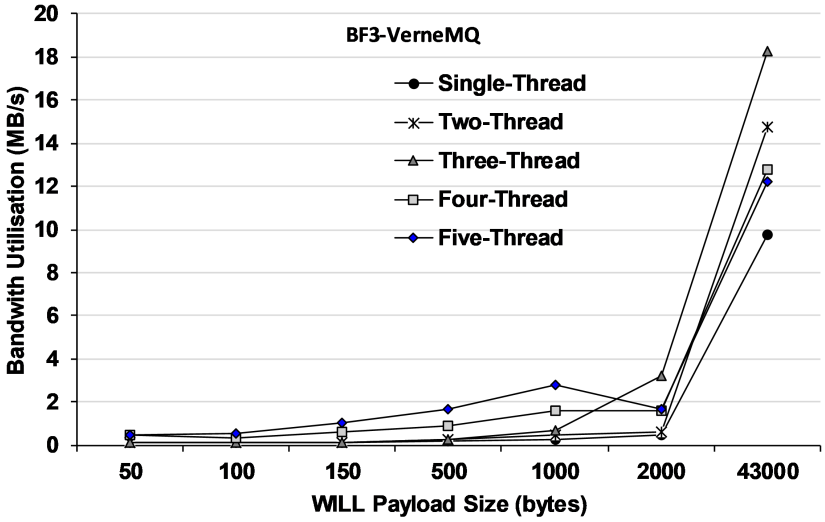


Figure 5.13: Impact of number of increasing attack threads and WILL message payload size on VerneMQ Broker bandwidth utilisation during the BF3 attack

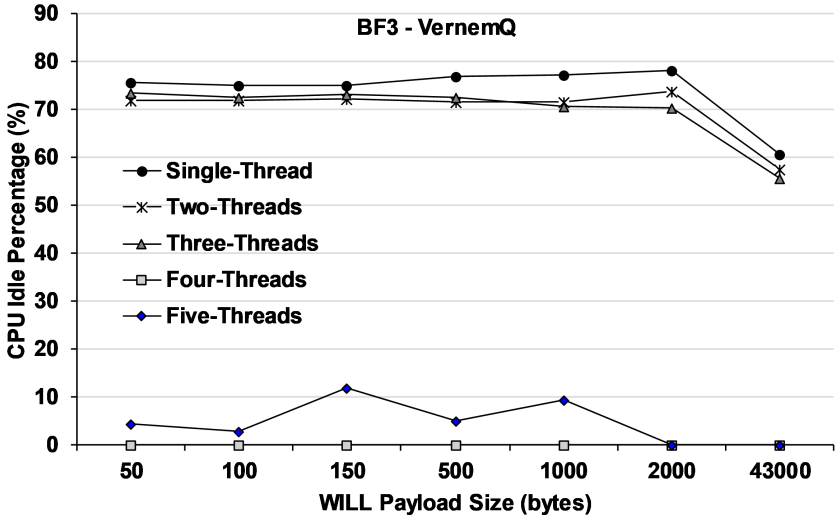


Figure 5.14: Impact of number of increasing attack threads and WILL message payload size on VerneMQ Broker CPU idle percentage during the BF3 attack

mum average bandwidth utilisation observed as low as 0.35 MB per second as shown in Figure 5.15. However, the CPU utilisation of EMQ broker was reduced to zero percentage for all the payload sizes among the different number of attack threads, as shown in Figure 5.16 indicating CPU exhaustion.

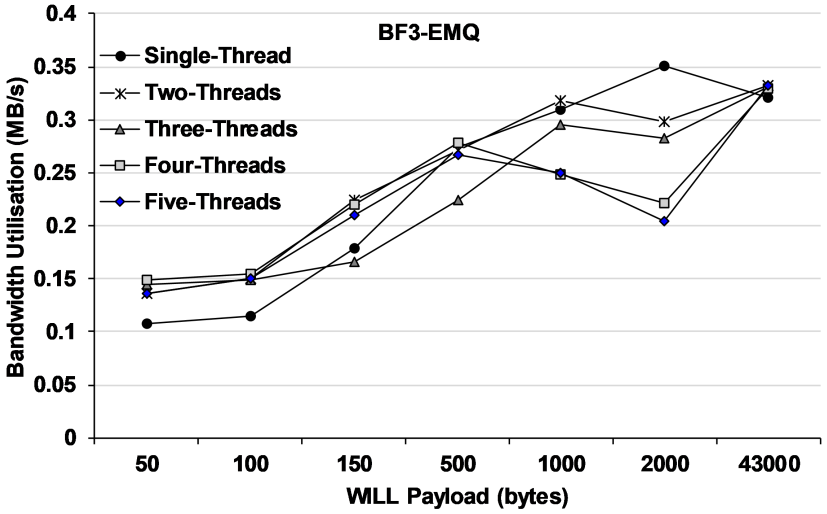


Figure 5.15: Impact of number of increasing attack threads and WILL message payload size on the EMQ Broker bandwidth utilisation during the BF3 attack

In addition to CPU and bandwidth utilisation due to the CONNECT flood attacks using a WILL message, the 43,000 bytes WILL payload contained malformed characters, which impacted the memory utilisation of the EMQ broker. Figure 5.17 shows the memory utilisation of EMQ broker with increasing number of attack threads for a 43,000 byte WILL payload. With the three, four and five attack threads the broker MQTT service crashed indicating a vulnerability in EMQ brokers to malformed packets.

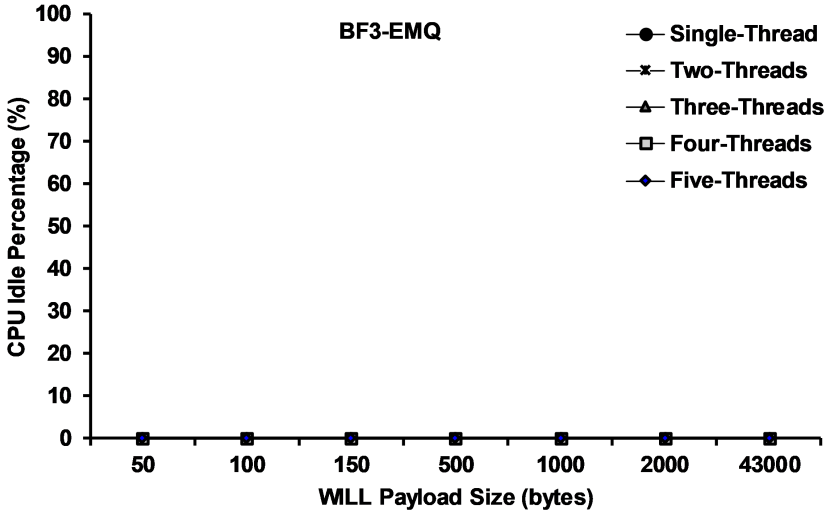


Figure 5.16: Impact of number of increasing attack threads and WILL message payload size on EMQ Broker CPU idle percentage during the BF3 attack

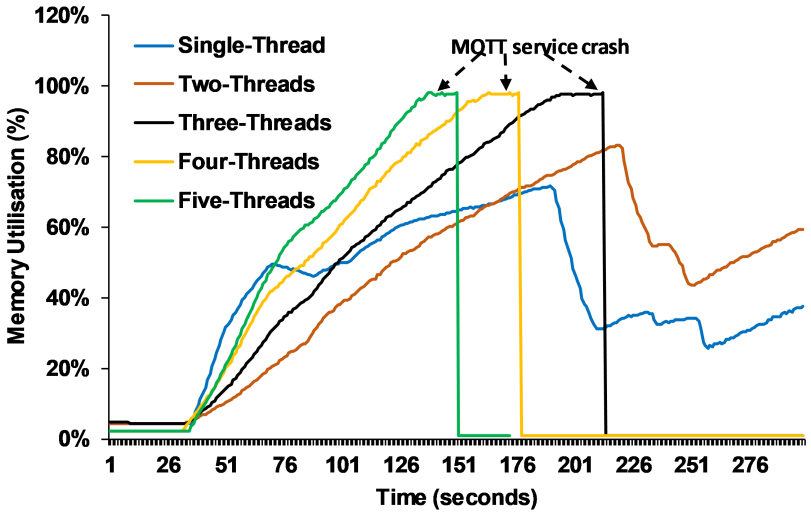


Figure 5.17: Impact of BF3 attack on average Memory utilisation of EMQ broker with increasing attack threads

Impact of Subscribe Flooding (IAUTHS) Attack

The subscription flooding attack (IV4) aims to consume CPU resources of the target broker server by sending multiple subscription requests in a single MQTT sessions. The treatment applied to IV4 was the number of subscription requests sent after establishing the MQTT session.

The impact of this attack was measured in terms of the CPU utilisation. The number of subscription requests were varied between 10 and 250 requests per session and the impact on CPU was measured independently. Figure 5.18 shows the impact of increasing the number of subscriptions on the three brokers. The results indicate that the CPU idle percentage of the Mosquitto broker dropped below five percentage with the increase of number of subscription requests, specifically with 50 and above subscription requests. However, on VerneMQ and EMQ brokers, the CPU idle percentage was zero for all the values of subscriptions per session, indicating a CPU exhaustion.

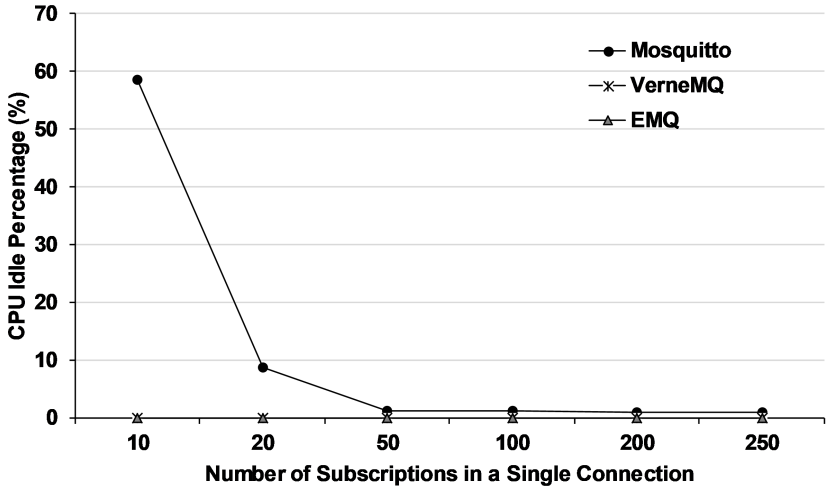


Figure 5.18: Impact of number of subscriptions in a single MQTT session on the CPU utilisation (Idle Percentage) on three MQTT brokers

In order to measure the impact of DoS attacks on the brokers configured with higher CPU resources and high-availability settings, a Six-CPU settings and a load-balanced configuration was evaluated; which is discussed in the following sections.

5.1.2 Six-CPU Deployment

The Six-CPU deployment was evaluated on the three brokers configured with six CPU cores to increase the number of CPU cores available for processing MQTT requests. The parameters of various DoS attack scenarios such as number of attack threads, sleep-interval, CONNECT delay and number of subscriptions in a single session identified using Single-CPU setting was used in the evaluation of the impact on the Six-CPU setting. The Six-CPU deployment experimental results show that increasing the number of CPUs available to process the MQTT requests reduced the impact of DoS attacks on the Mosquitto and the VerneMQ brokers for CONNECT flooding scenarios (BF1 to BF3), but the EMQ broker's CPU utilisation was impacted and idle percentage reduced below 60% for all the attack scenarios. However, all the brokers showed lower CPU idle percentage for the IAUTHS attack indicating significant effects of IAUTHS attacks on the MQTT brokers. The Six-CPU utilisation for all the three brokers is shown in Figure 5.19. Since the Mosquitto broker is a single-threaded application and only uses a single CPU, the actual utilisation is shown in terms of a single CPU. The Mosquitto broker showed an average idle percentage of 20%, as it is a single threaded application and does not use multi-core CPUs effectively. The results also show that VerneMQ had lower CPU idle percentage only for IAUTHS attack. In contrast, the EMQ broker had less than 20% CPU idle time for the two attack categories. Table 5.4 presents the values of various attack metrics measured to highlight the attack impact on DV1 with the treatments applied to IV1 to IV4.

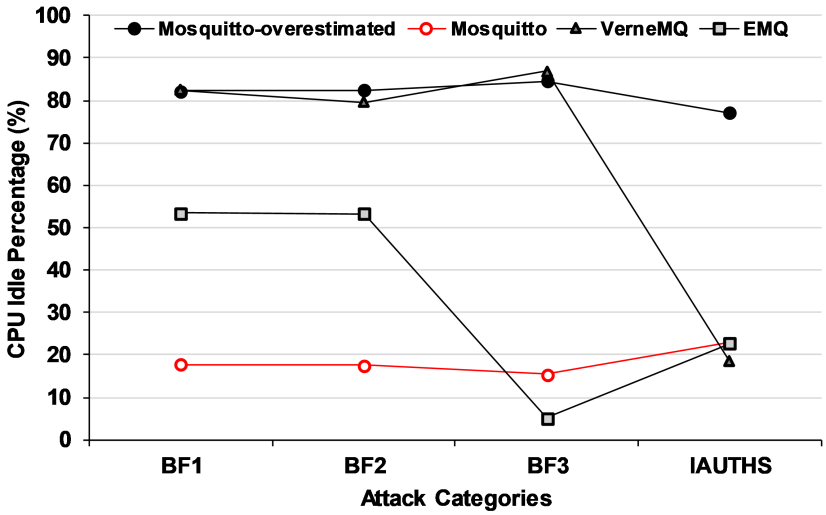


Figure 5.19: Comparison of CPU Idle Percentage measured during various attack scenarios with Six-CPU configuration

Table 5.3: Comparison of CPU utilisation breakup measured for various attack scenarios with Six-CPU configuration

Attack Type	Broker Type	%usr	%sys	%iowait	%soft	%idle
BF1	VerneMQ	52.22	22.43	0.01	7.22	18.09
	EMQ	14.68	10.24	17.36	4.22	53.50
	Mosquitto	1.69	4.04	0.02	12.07	82.19
BF2	VerneMQ	43.61	19.51	0.01	7.12	29.73
	EMQ	15.92	11.27	16.55	3.00	53.26
	Mosquitto	2.31	5.05	0.01	10.25	82.39
BF3	VerneMQ	7.95	3.24	13.29	1.54	73.98
	EMQ	72.28	7.70	13.50	1.31	5.20
	Mosquitto	1.79	3.96	0.01	9.63	84.61
IAUTHS	VerneMQ	71.31	4.21	0.00	8.68	15.76
	EMQ	70.45	3.50	0.00	3.37	22.68
	Mosquitto	18.34	2.86	0.01	1.68	77.11

Table 5.4: Bandwidth, Memory and Process CPU utilisation during various attack scenarios in the Six-CPU configuration

Attack Type	**Broker**	**Bandwidth (kbytes/s)**	**Memory%**	**pCPU(%)**	**Packets/Sec**
BF1	VerneMQ	481.51	0.09	402.88	6738.09
	EMQ	141.78	0.16	92.65	2019.92
	Mosquitto	542.82	0.03	64.52	7854.89
BF2	VerneMQ	341.44	0.17	406.80	4788.19
	EMQ	92.96	0.16	58.36	1338.36
	Mosquitto	455.97	0.03	60.36	6554.38
BF3	VerneMQ	18473.87	0.06	83.22	14552.83
	EMQ	4539.02	0.98	546.18	3691.76
	Mosquitto	18761.25	0.03	42.64	16394.63
IAUTHS	VerneMQ	1421.28	0.09	454.39	4938.29
	EMQ	758.95	0.06	432.44	1400.99
	Mosquitto	721.61	0.04	99.06	1701.62

5.1.3 Load-Balanced Deployment

The load-balanced deployment was tested with a six-node EMQ cluster configured with HA-Proxy software load-balancer. The experimental results are presented in Table 5.5, which shows the CPU, memory and bandwidth consumed (DV1) during the four attack categories. Figure 5.20 shows the break-down of CPU utilisation with the application of four IVs. The results indicate that BF3 and IAUTHS attacks had the higher impact on the six-node cluster compared to BF1 and BF2 flood attacks, as all the nodes in cluster had less than 5% CPU idle time during BF3 attack and less than 25% during IAUTHS attack.

Table 5.5: Bandwidth, Memory and Process CPU utilisation during four attack scenarios on load-balanced deployment

	Node	**Bandwidth Kbytes/s**	**Memory (%)**	**pCPU(%)**	**Packets/sec received**
BF1	1	91.70	0.07	27.69	762.97
	2	158.05	0.10	30.31	1323.96
	3	169.15	0.11	29.88	1413.89
	4	112.93	0.07	31.17	947.62
	5	83.42	0.08	26.42	700.59
	6	132.93	0.08	29.82	1106.18
BF2	1	83.82	0.08	31.22	688.70
	2	151.70	0.09	39.38	1297.76
	3	163.63	0.09	41.41	1405.91
	4	109.75	0.07	31.45	918.08
	5	76.88	0.08	25.90	629.67
	6	130.21	0.07	36.60	1102.24
BF3	1	1292.21	0.35	149.58	1098.50
	2	1138.16	0.24	168.21	1289.79
	3	1185.69	0.19	169.12	1357.84
	4	1289.77	0.19	166.43	1243.17
	5	1328.40	0.34	157.62	1111.30
	6	1253.56	0.19	169.31	1265.88
IAUTHS	1	103.29	0.09	74.23	686.77
	2	120.86	0.07	99.43	836.05
	3	157.89	0.09	146.92	1155.22
	4	135.60	0.08	122.30	967.00
	5	124.33	0.08	110.75	870.86
	6	133.42	0.08	112.02	942.05

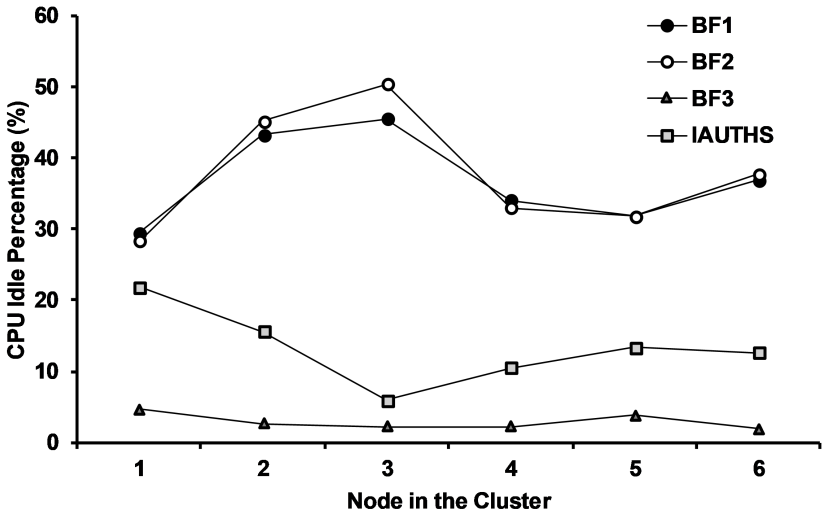


Figure 5.20: Comparison of CPU idle percentage variation measured for the four attack scenarios on load-balanced deployment

Various attack scenarios when launched against the three brokers indicated that all the authentication and authorisation attacks resulted in high CPU utilisation, which can have a negative impact on the messages being exchanged through them. The impact of authentication attacks can be considerably reduced with the increase of processing capabilities, however, the impact of authorisation attacks was not reduced. In order to measure the impact on the messages exchanged through the broker, the delay and message publish rates were measured during various attack scenarios, which are further discussed in the following section.

5.2 Delay and Message Publish Rate Measurements

The primary aim of publish/subscribe systems is to ensure delivery of messages published by clients to subscribers with acceptable service delays. Excessive delays can be detrimental to systems exchanging messages via the MQTT brokers. In addition to broker system performance, the message exchange performance (DV2) were also measured during the DoS attack scenarios (IV1 to IV4). The impact of individual DoS attack scenario on DV2 was measured in terms of message delay (milliseconds) with the three QoS levels and the average publish rate. Three Raspberry Pi client devices were configured to send and receive messages through the victim broker and

the message delay for normal and individual attack scenarios was measured respectively. The delay was measured as the time taken to receive a published message with three different QoS settings. Let T_p be the time at which the message is published to a topic A by the client, to which it also subscribes. The client receives the message for topic A at time T_q. The Round Trip Delay (RTD) for the message is then calculated as:

$$Round\,Trip\,Delay\,(RTD) = T_q - T_p \tag{5.1}$$

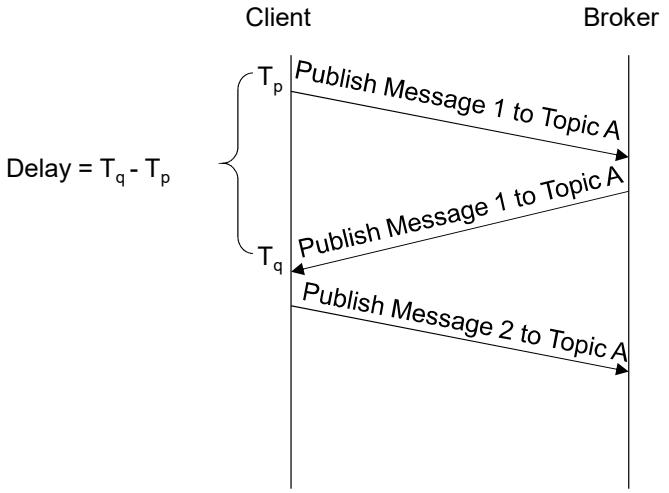


Figure 5.21: MQTT message delay calculation procedure with a single client publishing and subscribing to the same topic

The delay measurement was observed on the same client which published the message to avoid the problems occurring with synchronising clocks between separate clients as discussed in Ferrari et al. (2018). However, the delay measurement was repeated on three separate clients and an average delay was calculated to compare the impact of various attack scenarios. In order to get a realistic delay measurements, 1,000 client connections, publishing a message every two seconds with QoS1 was introduced. This provided an average publish rate of 500 messages per second and a subscriber client was used to measure the message publish rates during the various attack scenarios. The reason for choosing QoS1 for the load was because QoS1 provides reliable message delivery when compared to QoS0 and less overhead as compared to QoS2. The normal delay prior to introducing the load was also measured to compare the impact of increasing load on delay performance. The number of threads used to launch the attack on the

three brokers were fixed to one for BF1, BF3 and IAUTHS attacks and 250 threads for BF2 attack, as listed in the Table 5.6.

Table 5.6: Attack parameter settings for delay measurements

DoS Scenario	**Fixed Treatment**	**Broker**	**Load Settings**	**QoS Levels**	**Observations**
BF1 (IV1)	Sleep Interval = 0, Attack Threads = 1	Mosquitto, VerneMQ, EMQ	QoS =1 , 500 Messages per second	QoS0, QoS1, QoS2	Message Delay (ms), Publish Rate (Messages Published per second) (DV2)
BF2 (IV2)	CONNECT Delay = 0.5, Attack Threads = 250				
BF3 (IV3)	WILL message payload size = 43,000 bytes, Attack threads = 1				
IAUTHS (IV4)	Subscription loops = 50, Attack threads = 1				

Figure 5.22 shows the impact of DoS scenarios on MQTT message delay for the Mosquitto broker. The results indicate that the average message delay did not vary drastically for all the scenarios when compared to the normal delay. However the 75^{th} and 95^{th} percentile values presented in Table 5.7 reveal that the multiple messages experienced a higher delay than the average noted delay. Especially the QoS2 messages experienced a higher delay when compared to other QoS levels.

The impact of DoS attack on average message delay measured using VerneMQ broker was marginal compared to the normal delay as shown in Figure 5.23. However, the 75^{th} and 95^{th} percentile delay presented in Table 5.8 shows that a higher delay was experienced by more than 25% and 5% percentage of messages. Especially, QoS2 messages experienced higher 95^{th} percentile delay for BF1, BF3 and IAUTHS DoS scenarios.

The EMQ broker experienced a higher delay impact when compared to Mosquitto and VerneMQ broker. Especially, the QoS2 messages with IAUTHS attack, resulted in a average delay of above 400 milliseconds as compared to 68 milliseconds during normal operations, which is highlighted in Figure 5.24. Analysing the 75^{th} and 95^{th} percentile delays presented in Table 5.9 indicates that more than 25% and 5% percent of messages suffered a higher delays with QoS2 messages with IAUTHS attack resulting in a 2,300 millisecond delay.

The delay analysis results indicate that DoS attack scenarios had marginal

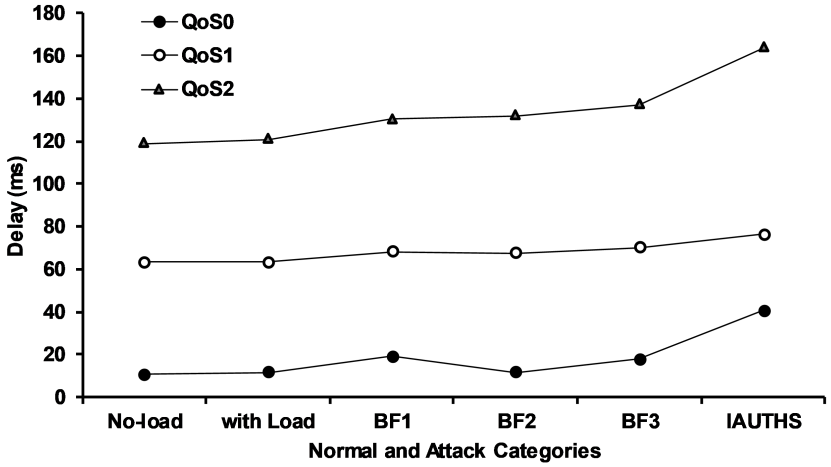


Figure 5.22: Message delay measured on Raspberry Pi clients while exchanging MQTT messages through the Mosquitto broker during various attack scenarios and on three QoS levels

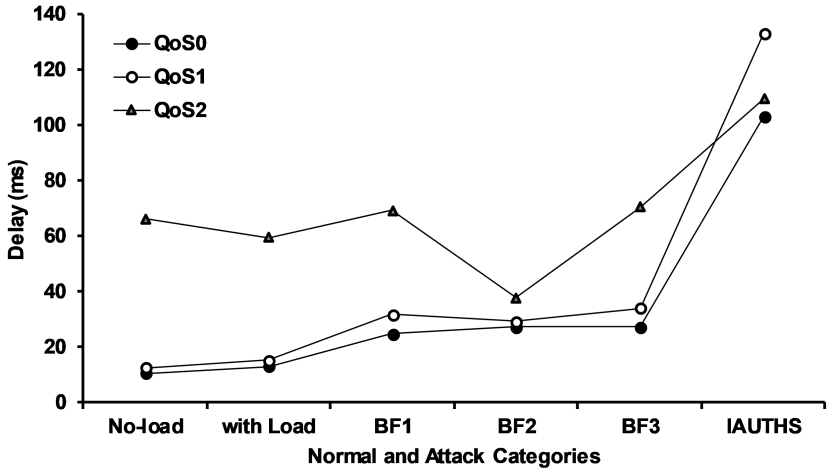


Figure 5.23: Message delay measured on Raspberry Pi clients exchanging MQTT messages through the VerneMQ broker during various attack scenarios and on three QoS levels

impact on the average delay recorded for all the three brokers, however the 75^{th} and 95^{th} percentile delays show that DoS attacks caused a considerable

Table 5.7: Comparison of average, 50^{th}, 75^{th} and 95^{th} percentile RTD (ms) for various attack scenarios for three supported QoS levels on Mosquitto broker

	Avg Delay(ms)	**50th-%-ile**	**75-%-ile**	**95-%-ile**
Normal with Load				
QoS0	11.67	8.83	11.38	25.33
QoS1	63.40	58.40	63.16	79.40
QoS2	120.77	114.46	122.51	148.89
BF1				
QoS0	17.75	13.69	20.66	42.49
QoS1	70.13	65.23	69.92	103.32
QoS2	137.15	126.94	140.45	190.33
BF2				
QoS0	19.21	16.35	22.18	44.23
QoS1	68.21	59.98	70.03	91.51
QoS2	130.43	122.00	133.96	183.31
BF3				
QoS0	11.53	9.02	11.26	21.68
QoS1	67.71	61.13	68.32	96.10
QoS2	131.86	120.52	132.96	182.54
IAUTHS				
QoS0	40.74	35.44	57.16	98.57
QoS1	76.53	70.36	81.83	117.32
QoS2	163.84	154.90	184.22	240.19

performance degradation for QoS2 messages which could be detrimental to the timely delivery of critical messages. In addition to the RTD of MQTT messages, the average publish rate with the introduced load was also analysed to verify the impact of the DoS attacks. The message publish rate was measured by using a subscriber client which was subscribed to receive the messages sent by the 1,000 client threads deployed to introduce load; receiving an average of 500 messages per second. The message publish rate for each attack scenario was measured while obtaining the delay measurements for individual QoS levels, and an average publish rate was obtained. Since the average publish rates are calculated for the period of measurement, a reduction of publish rate below the normal average indicates either a loss of published messages or heavily delayed messages.

Figure 5.25 shows the message publish rate achieved by the Mosquitto broker during the various attack scenarios. The results show that the Mosquitto broker's publish rate reduced by 50% for BF1, BF3 and IAUTHS attacks indicating successful DoS.

The impact on the VerneMQ broker is highlighted in Figure 5.26. The

Table 5.8: Comparison of average, 50^{th}, 75^{th} and 95^{th} percentile RTD (ms) for various attack scenarios for three supported QoS levels on VerneMQ broker

	Avg Delay(ms)	**50-%-ile**	**75-%-ile**	**95-%-ile**
Normal with Load				
QoS0	13.01	9.81	13.43	29.12
QoS1	15.28	10.77	16.43	37.50
QoS2	59.29	60.28	67.56	90.44
BF1				
QoS0	24.80	11.44	28.77	89.26
QoS1	31.69	13.23	36.28	128.84
QoS2	69.27	61.22	77.27	153.17
BF2				
QoS0	27.30	17.43	27.80	92.26
QoS1	29.30	20.13	30.55	63.19
QoS2	37.73	26.94	44.88	86.90
BF3				
QoS0	27.09	14.59	29.47	101.07
QoS1	34.01	15.25	43.39	122.41
QoS2	70.40	62.48	79.45	147.32
IAUTHS				
QoS0	102.95	66.08	157.74	328.56
QoS1	132.89	80.15	204.03	380.78
QoS2	109.23	70.11	151.26	334.18

results show that the message publish rate was relatively stable compared to the Mosquitto broker, however, a publish rate drop was observed in BF2 and IAUTHS attacks. The message publish rate for BF1 and BF3 was marginally higher than the normal average. This indicates a possible retransmissions of QoS1 messages configured for the two publishing clients.

The EMQ broker had a stable publish rate for BF1, BF2 and BF3 attacks compared to the other two brokers, however, the publish rates reduced drastically for IAUTHS attack as shown in Figure 5.27.

Combining the publish rate results with the message delay results highlight that the VerneMQ and EMQ broker had less impact on the publish rates with the three attack scenarios while the message delay increased when compared to the Mosquitto broker. This indicates a best effort performance of the EMQ broker to deliver the messages even with a higher delay compared to a degraded performance of the Mosquitto broker which yielded a lower delay albeit incurring a higher publish rate drop.

The results also indicate that the IAUTHS attack had the maximum impact on all the three brokers with degraded performances on the message

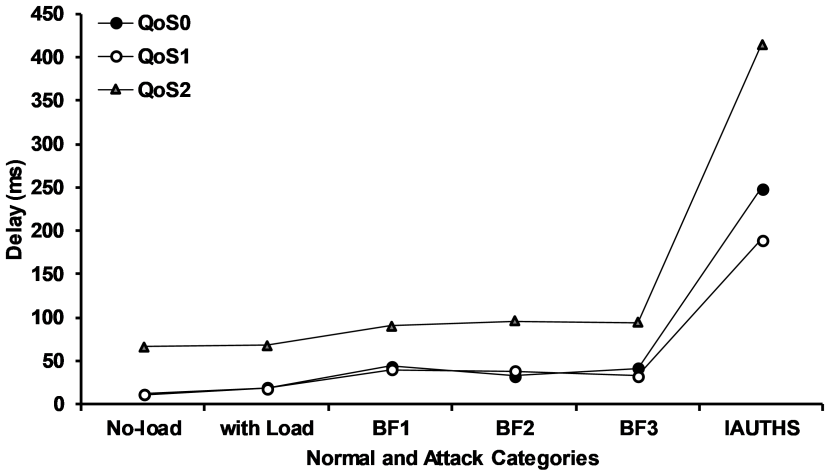


Figure 5.24: Message delay (ms) of measured on Raspberry Pi clients exchanging MQTT messages through the EMQ broker during various attack scenarios and on three QoS levels

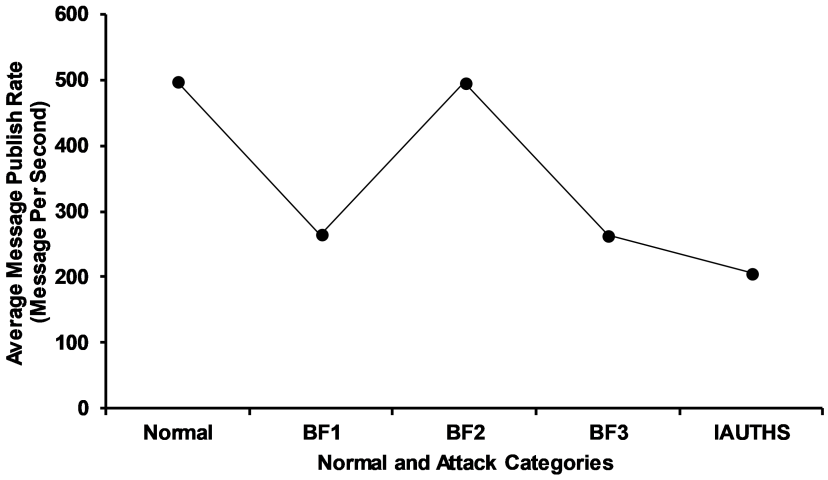


Figure 5.25: MQTT message publish rate observed through the Mosquitto broker during various attack scenarios

delay and message publish rates. The choice of number of subscription loops also was analysed, which is shown in Figure 5.28. Results indicate that the performance of VerneMQ and EMQ brokers degraded with the increase of

Table 5.9: Comparison of average, 50^{th}, 75^{th} and 95^{th} percentile RTD (ms) for various attack scenarios for three supported QoS levels on EMQ broker

	Avg Delay(ms)	**50-%-ile**	**75-%-ile**	**95-%-ile**
Normal with Load				
QoS0	19.34	12.90	22.24	57.98
QoS1	18.45	12.72	20.50	45.00
QoS2	68.33	64.92	75.33	112.90
BF1				
QoS0	43.83	18.01	36.48	205.01
QoS1	40.31	19.05	41.57	166.59
QoS2	90.04	70.61	91.50	259.36
BF2				
QoS0	32.40	17.49	32.49	124.51
QoS1	38.49	18.49	32.63	165.58
QoS2	96.27	73.17	99.71	287.37
BF3				
QoS0	41.17	15.43	34.85	198.80
QoS1	32.99	15.43	29.37	150.13
QoS2	93.93	71.44	95.36	269.80
IAUTHS				
QoS0	248.76	125.53	367.62	886.10
QoS1	189.84	54.86	223.17	867.02
QoS2	414.31	130.51	393.08	2390.35

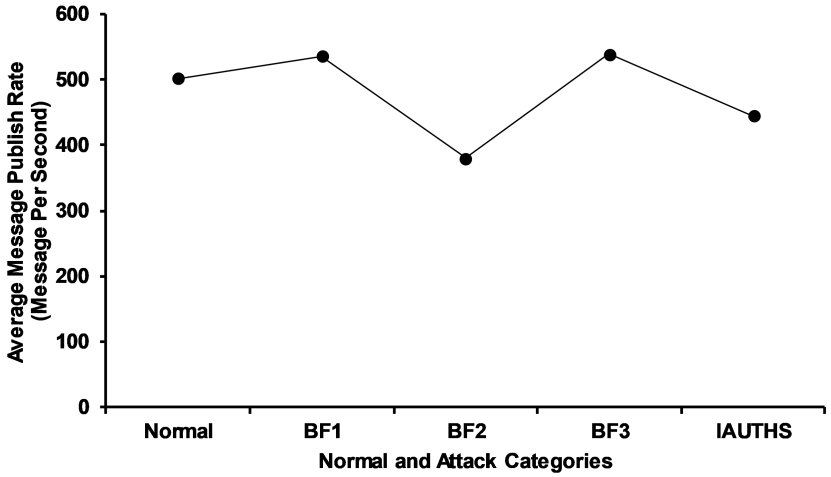


Figure 5.26: MQTT message publish rate observed through the VerneMQ broker during various attack scenarios

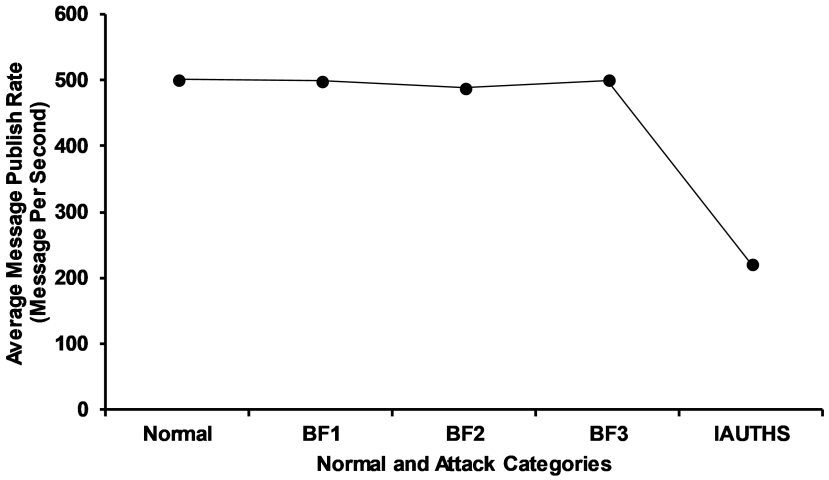


Figure 5.27: MQTT message publish rate observed through the EMQ broker during various attack scenarios

number of subscriptions sent per session when compared to the Mosquitto broker, which yielded a lower publish rate with 50 subscriptions per session.

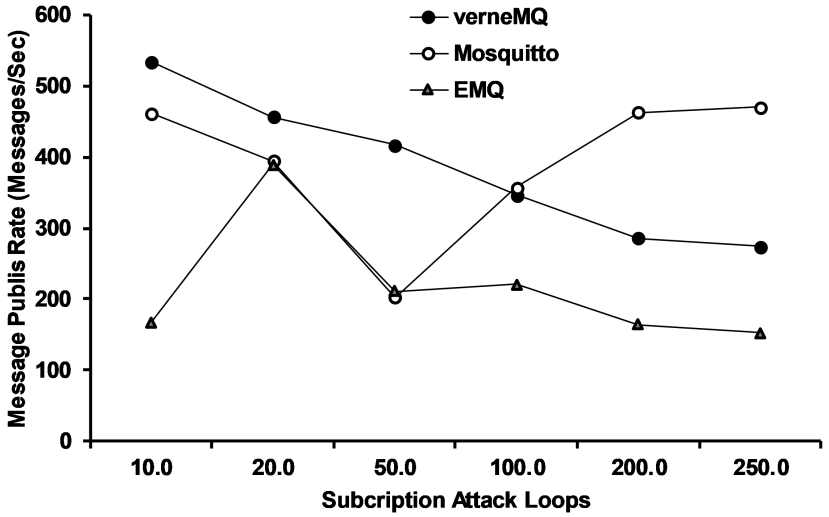


Figure 5.28: MQTT message publish rate observed through the three brokers during various subscribe flooding loops in subscription flooding attack

In this section the impact of DoS attacks (IV1 to IV4) on the system performance (DV1) of the three MQTT brokers was presented along with the impact on message exchange performance (DV2). The results indicate that the proposed MQTT based authentication and authorisation DoS attack scenarios caused degradation of performance, delay and caused publish rate drops. These results also indicate that such DoS attacks are potential threats to the MQTT brokers and communication messages facilitated by them; consequently detecting and protecting MQTT systems from such malicious attacks are essential. Hence, in the next section we present the attack detection results of the MQTT attack detection framework discussed in Section 4.4.

5.3 Detection Framework Performance

The DoS attack detection framework proposed in this research was evaluated by applying the treatment to two IVs (classifier algorithm-IV5 and feature-vector-IV6) and measuring the impact on DV3 (classifier performance). The treatments applied to IV5 were the three classifier algorithms evaluated in this research namely: AODE, C4.5 and MLP classifiers. The treatments applied to IV6 were: FULL features, TCP based features, Count based features and Size based features. The impact of the treatments applied to IV5 and IV6 were measured in terms of the classifier accuracy, error, true positive rate (TPR) and false positive rate (FPR). Furthermore, the number of labelled classes were varied to test the effectiveness of the proposed features in classifying them. The following section discusses the datasets used to evaluate the detection framework. The performance of classifiers with time-window and flow-based MQTT features have been discussed in Sections 5.3.3 and 5.3.3 respectively.

5.3.1 Description of Datasets

Normal and attack packets were captured separately and pre-processed to generate the dataset for training and testing the detection framework. The feature vectors for time-window based and flow-based aggregation were generated separately and the corresponding classifier models were evaluated independently.MQTT attacks presented in this research and two attacks external to this research were included to test the effectiveness of the detection framework.

Three different MQTT attacks: MQTT-DoS (based on the attack scenarios described in this work), MQTT-FUZZER (using a MQTT Fuzzing

tool (Vähä-Sipilä, 2015)) and TCP-DOS (using hping3 SYN-Flood tool (Sanfilippo, 2006)) were generated against the broker. The MQTT fuzzing attack was launched using an MQTT fuzzing tool which was configured to send fuzzed packets sniffed from the deployed IoT network. The attack traffic was subsequently captured for dataset generation. Based on the aggregation level and the attack classes used, four datasets were generated as presented in Table 3.4.

The time-window datasets were generated with only two types of MQTT attacks namely: MQTT-DoS and MQTT-FUZZER. Since TCP-DoS attack generates a large amount of attack traffic in a short period of time, this attack was only included in the flow based dataset and excluded from the time-window dataset. Based on the attack types used, two datasets were generated for time-window aggregation, namely, TW-Major-DS and TW-Sub-DS. The TW-Major-DS consisted of normal traffic as well as MQTT-DoS attacks and MQTT-Fuzz attacks. In contrast, the TW-Sub-DS dataset consisted of sub-classes of MQTT-DoS attacks, demonstrated in this work. This was done to compare the detection performance of the detection framework in differentiating between various classes of MQTT-DoS attacks. The flow based detection analysis was performed by including all the three MQTT attacks to generate attack traffic. The flow-based dataset was subsequently generated from MQTT flow features as discussed in Chapter 4. This resulted two datasets FL-Major-DS and FL-Sub-DS dataset, which were evaluated for the flow based MQTT features. The class labels used in the four datasets are listed in Table 5.10.

The total number of flows collected during the normal and attack instances were 1,042,500 for both time-window and flow-based datasets. The number of time-window records generated from captured network packets was 58,600 and the number of flows records were 1,012,052. The distribution of instances based on the classes of attack are presented in tables 5.11 and 5.12. The flow dataset was further balanced using re-sampling to balance the classes. Re-sampling technique was applied to balance the classes to avoid bias in classifier accuracy. The data was under-sampled to produce random sub-samples of the original dataset with following setting in Weka: *biasToUniformClass=1.0, noReplacement=True, sampleSizePercent=40.0.* The biasToUniformClass attribute ensures that re-sampled data contains uniform class distribution. The noReplacement attribute ensures that the re-sampled data does not contain duplicate copies of instances for a class that contains fewer instances. The sampleSizePercent attribute was used to produce a re-sampled dataset that contains approximately 40% of the original dataset which was selected based on the percentage difference

Table 5.10: Datasets used in this research to compare models built using FV groups and classifier algorithms

Aggregation Level	**Class Labels**	**Dataset Name**
Time-window	Normal, MQTT-DOS, MQTT-FUZZ	TW-Major-DS
Time-window	Normal, MQTT-DOS-BF1, MQTT-DOS-BF2, MQTT-DOS-BF3, MQTT-DOS-IAUTHS, MQTT-FUZZ	TW-Sub-DS
Flow-Based	Normal, MQTT-DOS, MQTT-FUZZ, TCP-DOS	FL-Major-DS
Flow-Based	Normal, MQTT-DOS-BF1, MQTT-DOS-BF2, MQTT-DOS-BF3, MQTT-DOS-IAUTHS, MQTT-FUZZ, TCP-DOS	FL-Sub-DS

between largest and the smallest classes in the original dataset.

Table 5.11: Data distribution of various classes in the time-window based datasets

Dataset Name	**Classes**	**Number of Instances**
TW-Major-DS	Normal	46,464
	MQTT-DoS	11,181
	MQTT-FUZZ	955
TW-Sub-DS	Normal	46,464
	MQTT-DOS-BF1	2,756
	MQTT-DOS-BF2	2,865
	MQTT-DOS-BF3	3,508
	MQTT-DOS-IAUTHS	2,052
	MQTT-FUZZ	955

Table 5.12: Data distribution in various classes in the flow-based datasets

Dataset Name	**Classes**	**Number of Instances**	
		Unbalanced	Balanced
FL-Major-DS	Normal	71,217	63,376
	MQTT-DoS	623,246	63,376
	MQTT-FUZZ	49,916	49,916
	TCP-DOS	267,673	63,376
FL-Sub-DS	Normal	73,982	59,571
	MQTT-DOS-BF1	240,349	59,571
	MQTT-DOS-BF2	90,550	59,571
	MQTT-DOS-BF3	277,454	59,571
	MQTT-DOS-IAUTHS	42,576	42,576
	MQTT-FUZZ	49,916	49,916
	TCP-DOS	267,673	59,571

5.3.2 Training, Testing and Feature Groups

Several experiments were conducted to measure the performance of AODE, DT and MLP classifiers used in the detection framework. A 10-fold cross-validation method was enforced, which trained and tested the machine learning models on complementary subsets of training data, to prevent bias and over-fitting issues. The Weka ML workbench uses a stratified cross-validation approach which ensures the data selected in each fold is a fair representation of the entire dataset in terms of the proportions of various classes. In each fold, 90% of data is used as training data and 10% as testing data which provides a fair estimate of the performance of the developed model on unseen data. This processes is repeated 10 times, estimating the individual models performance on new set of test data and the average accuracy of 10 folds is reported. In addition, a separate test data was not used as the network traffic was captured in a controlled IoT testbed deployed for this work and may not contain varied instances of normal and attack traffic like a real network traffic data.

Default Weka settings were used for cross-validating the three ML classifiers on the four datasets used in this work. In order to show the effectiveness of the proposed MQTT features compared to only using Transport Layer

features for attack detection, two sets of feature groups were evaluated, namely, FULL-FV and TCP-FV. The FULL feature vector included both TCP based and the proposed MQTT based features, whereas the TCP-FV consisted of only TCP based features. Transport Layer features such as IP addresses, port numbers were not included as the attacks were launched from a single attack source and only traffic directed to MQTT port 1883 was captured for attack detection.

In addition to the analysis of TCP based and full features, the effectiveness of count based (COUNT-FV) and size based features was compared (SIZE-FV). This was performed to identify the most prominent group of features that contributed the detection performance. Furthermore, comparing the various combinations of features provides better insights into the features that provide better classification accuracy. The count based features included features that captured the frequency of occurrence of the attribute in the dataset. Whereas the size based features included features that measured the average length or size of various MQTT fields.

Traditional statistical feature selection methods were not applied as many MQTT features are dependent on each other and can produce high correlations, causing it to be eliminated by the feature selection techniques. However, some of the dependent features are binary in nature and are either enabled or disabled which can indicate different behaviour of the connection request. For example, every CONNECT request must have the cleanSessionFlag set or unset and could indicate a persistent or non-persistent connection having different connection behaviours. Similarly, the PUBLISH and SUBSCRIBE requests contain fields that indicate the QoS levels set for the request and causes different message exchange behaviour between the broker and the client. Some of these features could introduce redundant information but would be essential in distinguishing different connection requests in the MQTT protocol. Due to these reasons the groups of features that contributed more to the model performance were analysed instead. The datasets used, the two treatments applied to IV5 (classifiers) and IV6 (feature groups), and the observations performed in attack detection evaluation are listed in Table 5.13.

5.3.3 Detection Results

The evaluation results of the classifiers used in the MQTT attack detection framework have been presented in terms of the accuracy (%), error (%), TPR and FPR, and time to build the model (DV3). The following sections present the performance of time-window and flow-based features to classify

Table 5.13: Datasets used, treatment applied to IVs and the observations made for attack detection evaluation

Dataset Name	Classifer (IV5)	Feature-Vector (IV6)	Observations
TW-Major-DS	AODE, C4.5, MLP	FULL-FV, TCP-FV, COUNT-FV, SIZE-FV	Accuracy, Error, TPR, FPR
TW-Sub-DS			
FL-Major-DS			
FL-Sub-DS			

normal and attack instances in the datasets.

Time-Window Detection

The time-window detection results of the three classifiers utilising various combination of feature vectors on TW-Major-DS dataset are illustrated in Figure 5.29. The results indicate that all the three classifiers had higher classification accuracy with full, count and size based features when compared to only using TCP based features. The performance of classifiers reduced with the use of only TCP based features (TCP-FV: AODE yielded 98% and MLP yielded 97% accuracy) and improved with the addition of MQTT based features (FULL-FV: all classifiers yielded greater than 99.5% accuracy), highlighting the importance of the features proposed in this work.

Table 5.14 lists the various performance metrics measured for the three classifiers of the detection framework on the TW-Major-DS dataset. The results indicate that AODE classifier had the best performance among all the classifiers especially with full feature set (greater than 99%). The AODE classifier also required the least training time (1.49 seconds) compared to C4.5 (4.6 seconds) and MLP (259 seconds) classifiers.

Similar to the performance with TW-Major-DS, the classifiers performed better with full (all classifiers yielded greater than 99% accuracy) and size based features (AODE:99.7%, C4.5 99.7% and MLP 98.18%) in the TW-Sub-DS dataset as shown in Figure 5.30. The performance of classifiers decreased (AODE 98%, C4.5 98%) compared to all the other combinations when TCP based based features were used for classification, especially the MLP classifier only yielded a mere 94% accuracy.

Table 5.15 lists the performance of various classifiers and feature groups with the TW-Sub-DS datasets which included the sub-classes of MQTT-DoS attacks. The results indicate that the proposed features provided good detection accuracy (greater than 99%) when the full feature set was utilised

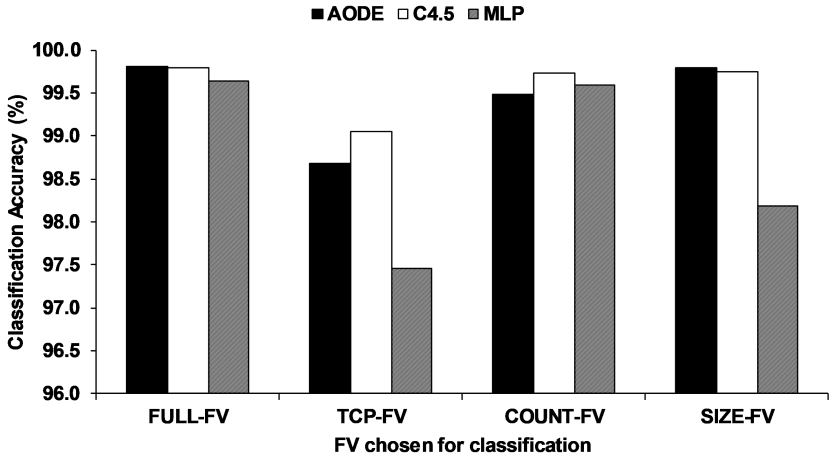


Figure 5.29: Performance comparison of classifiers with various combinations of time-window feature vector (FV) chosen for classification on the TW-Major-DS dataset

Table 5.14: Performance comparison of three classifiers using the time-window features used in the MQTT attack detection framework applied on TW-Major-DS datasets

	ACC%	**Error%**	**TPR**	**FPR**	**Training Time (seconds)**
FULL Features					
AODE	99.80	0.20	1.00	0.01	1.49
C4.5	99.79	0.21	1.00	0.01	4.64
MLP	99.65	0.35	1.00	0.01	259.01
TCP Features					
AODE	98.68	1.32	0.99	0.02	0.30
C4.5	99.05	0.95	0.99	0.02	1.06
MLP	97.45	2.55	0.98	0.06	40.15
COUNT Features					
AODE	99.48	0.52	1.00	0.01	0.58
C4.5	99.74	0.26	1.00	0.01	1.92
MLP	99.59	0.41	1.00	0.01	139.46
SIZE Features					
AODE	99.79	0.21	1.00	0.01	0.58
C4.5	99.74	0.26	1.00	0.01	1.39
MLP	98.18	1.82	0.98	0.05	63.17

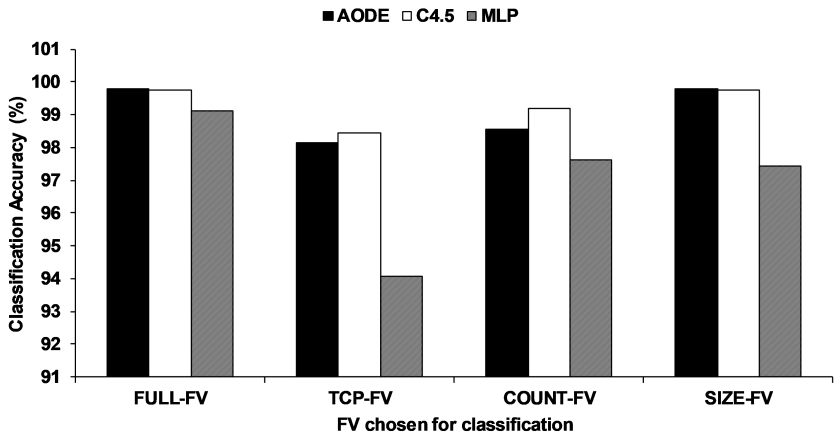


Figure 5.30: Performance comparison of classifiers with various combinations of time-window feature vector (FV) chosen for classification on the on TW-Sub-DS dataset

when compared to TCP based features. These results also indicate that the proposed features could effectively differentiate between the various MQTT-DoS attack classes when all the features were utilised for classification.

Tables 5.16, 5.17 and 5.18 show the confusion matrices of the three classifiers on the TW-Sub-DS dataset with full feature (FULL-FV) set. The confusion matrices indicate that BF2 attack was the most misclassified attack by all the three classifiers especially, MLP had the most misclassifications among all the classifiers for most of the classes.

The time-window based detection results indicate that the proposed MQTT based time window features are suitable to detect MQTT based attacks presented in this work.

Flow-Based Detection

Figure 5.31 shows that all three classifiers achieved a high classification accuracy (AODE:99.9%, C4.5: 99.9% and MLP:99.1%) in detecting the attack traffic in FL-Major-DS. The results also show that using the TCP (TCP-FV) and COUNT (COUNT-FV) based features reduced the classification accuracy and increased the false positive rate of all the classifiers compared to full (FULL-FV) and size (SIZE-FV) features, thus indicating that full MQTT based features provided high detection accuracy for detecting malicious flows.

Table 5.15: Performance comparison of three classifiers using the time-window features used in the MQTT attack detection framework applied on TW-Sub-DS datasets

	ACC%	**Error%**	**TPR**	**FPR**	**Training Time (seconds)**
FULL Features					
AODE	99.75	0.23	1.00	0.01	1.52
C4.5	99.73	0.27	1.00	0.01	4.25
MLP	99.11	0.89	0.99	0.01	331.44
TCP Features					
AODE	98.16	1.84	0.98	0.02	0.32
C4.5	98.45	1.55	0.98	0.02	1.04
MLP	94.07	5.93	0.94	0.06	62.42
COUNT Features					
AODE	98.54	1.46	0.99	0.01	0.68
C4.5	99.21	0.79	0.99	0.01	2.87
MLP	97.63	2.37	0.98	0.01	178.33
SIZE Features					
AODE	99.80	0.20	1.00	0.01	0.52
C4.5	99.73	0.27	1.00	0.01	1.54
MLP	97.43	2.57	0.97	0.04	90.50

Table 5.16: Confusion matrix with AODE classifier on TW-Sub-DS dataset with full features

Actual	**Predicted**					
	Normal	BF1	BF2	BF3	IAUTHS	FUZZ
Normal	46,423	6	26	8	0	1
BF1	16	2730	10	0	0	0
BF2	43	0	3,462	3	0	0
BF3	15	1	3	2,846	0	0
IAUTHS	4	0	0	0	2,048	0
FUZZ	10	1	0	0	0	944

However, the accuracy of MLP classifier reduced (75%) with the FL-Sub-DS datasets as shown in Figure 5.32 compared to AODE (99.8%) and C4.5 (99.8%) classifiers.

The performance metrics of the three classifiers on the FL-Major-DS and FL-Sub-DS datasets respectively, have been presented in tables 5.19 and 5.20. The results indicate that the detection accuracy of the classifiers increased when packet size and field length based features were considered,

Table 5.17: Confusion matrix with C4.5 classifier on TW-Sub-DS dataset with full features

Actual	**Predicted**					
	Normal	BF1	BF2	BF3	IAUTHS	FUZZ
Normal	46,434	2	19	2	3	4
BF1	16	2,731	9	0	0	0
BF2	58	2	3,446	2	0	0
BF3	16	0	3	2,846	0	0
IAUTHS	2	0	2	0	2,047	1
FUZZ	8	0	2	0	0	945

Table 5.18: Confusion matrix with MLP classifier on TW-Sub-DS dataset with full features

Actual	**Predicted**					
	Normal	BF1	BF2	BF3	IAUTHS	FUZZ
Normal	46,457	5	0	2	0	0
BF1	17	2,620	42	74	0	3
BF2	145	11	3,241	111	0	0
BF3	32	48	15	2,770	0	0
IAUTHS	4	0	0	0	2,048	0
FUZZ	14	1	0	2	2	936

when compared to only utilising count-based features (all classifiers had less than 87% accuracy). This indicates that size based features had better capability in separating the normal flows from malicious flows. The AODE classifier yielded the lowest training time (10 seconds for FULL-FV) and MLP classifier yielded the highest training (1598 seconds for FULL-FV) times among the selected classifiers. These results indicate that the proposed MQTT features provided good separation between normal and attack records, resulting in high detection rates and low false positives. However, the MLP classifier only achieved a classification accuracy of 75% for the FL-Sub-DS dataset, when all the features were used. Hence, the MLP classifier was further evaluated with modification to various optimisation parameters to identify the most optimal settings to increase its detection performance. The optimisation parameters considered in this study for improving MLP classifier were: activation and solver functions.

Tables 5.21, 5.22 and 5.23 show the confusion matrices of the three classifiers with FL-Sub-DS dataset. The confusion matrices indicate that AODE

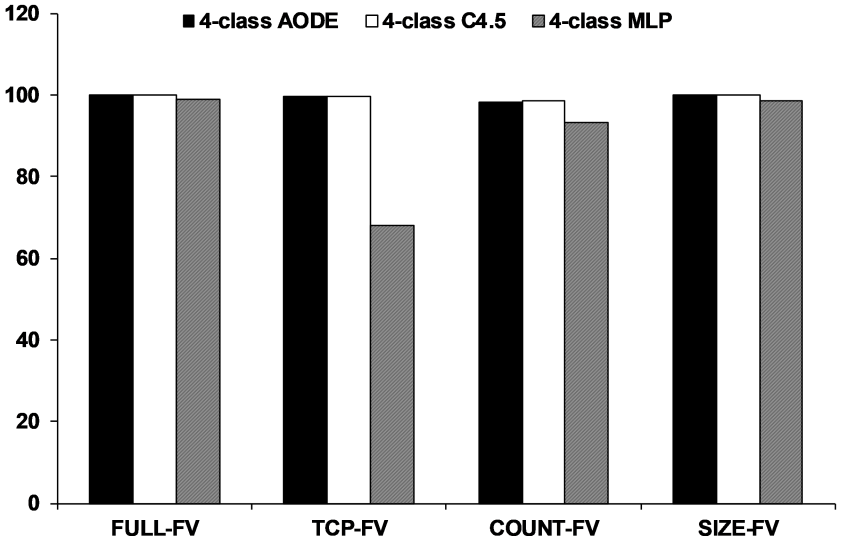


Figure 5.31: Performance comparison of classifiers with various combination of flow-based features on FL-Major-DS dataset

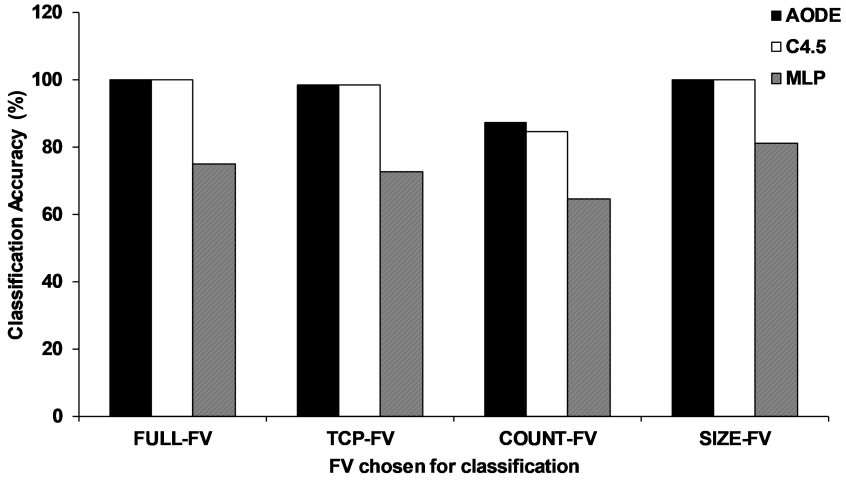


Figure 5.32: Performance comparison of classifiers with various combination of flow-based features on FL-Sub-DS dataset

and C4.5 classified few instances of normal connections as IAUTHS or vice versa. In addition, the MLP classifier misclassified many instances of BF1,

Table 5.19: Performance comparison of three classifiers using the flow-based features used in the MQTT attack detection framework applied on four class datasets

	ACC%	**Error%**	**TPR**	**FPR**	**Training Time (seconds)**
FULL Features					
AODE	99.97	0.03	1.00	0.00	5.70
C4.5	99.95	0.05	1.00	0.00	21.72
MLP	99.16	0.84	0.99	0.00	858.35
TCP Features					
AODE	99.67	0.33	1.00	0.00	1.78
C4.5	99.60	0.40	1.00	0.00	5.09
MLP	68.06	31.94	0.68	0.11	135.55
COUNT Features					
AODE	98.45	1.55	0.99	0.01	2.72
C4.5	98.46	1.54	0.99	0.01	17.57
MLP	93.25	6.75	0.93	0.02	417.32
SIZE Features					
AODE	99.93	0.07	1.00	0.00	2.53
C4.5	99.92	0.08	1.00	0.00	8.36
MLP	98.78	1.22	0.99	0.00	263.59

BF2 and BF3, which resulted in a lower overall detection accuracy. In order to improve the MLP classifier performance, the hyper-parameters used by Weka were further investigated and it was identified that the Weka software does not allow modifying the activation functions and the solver-functions. Hence an external ML workbench was utilised to identify the optimal hyper-parameters that could improve the classifier performance as discussed in the following section.

Table 5.20: Performance comparison of three classifiers using the flow-based features used in the MQTT attack detection framework applied on seven class datasets

	ACC%	**Error%**	**TPR**	**FPR**	**Training Time (seconds)**
FULL Features					
AODE	99.85	0.15	1.00	0.00	10.95
C4.5	99.83	0.17	1.00	0.00	37.26
MLP	75.07	24.93	0.75	0.04	1598.56
TCP Features					
AODE	98.47	1.53	0.99	0.00	3.09
C4.5	98.47	1.53	0.99	0.00	7.14
MLP	72.62	27.38	0.73	0.05	414.18
COUNT Features					
AODE	87.29	12.71	0.87	0.02	4.68
C4.5	84.67	15.33	0.85	0.03	26.99
MLP	64.38	35.62	0.64	0.06	952.42
SIZE Features					
AODE	99.83	0.17	1.00	0.00	4.53
C4.5	99.83	0.17	1.00	0.00	9.80
MLP	80.89	19.11	0.81	0.03	641.41

Table 5.21: Confusion matrix with AODE classifier on FL-Sub-DS dataset with full features

Normal	BF1	BF2	BF3	IAUTHS	FUZZ	TCP_DOS	
Normal	59,039	0	0	2	518	10	2
BF1	1	59,567	2	1	0	0	0
BF2	4	0	59,567	0	0	0	0
BF3	0	1	2	59,567	0	0	1
IAUTHS	6	1	0	0	42,567	0	2
FUZZ	6	8	2	0	1	49,898	1
TCP_DOS	14	0	0	0	2	0	59,555

Table 5.22: Confusion matrix with C4.5 classifier on FL-Sub-DS dataset with full features

Actual	**Predicted**						
	Normal	BF1	BF2	BF3	IAUTHS	FUZZ	TCP_DOS
Normal	59,283	0	2	1	264	17	4
BF1	0	59,568	2	1	0	0	0
BF2	4	3	59,563	1	0	0	0
BF3	1	0	0	59,570	0	0	0
IAUTHS	285	1	0	0	42,287	0	3
FUZZ	25	12	1	5	0	49,873	0
TCP_DOS	16	0	0	0	0	1	59,554

Table 5.23: Confusion matrix with MLP classifier on FL-Sub-DS dataset with full features

Actual	**Predicted**						
	Normal	BF1	BF2	BF3	IAUTHS	FUZZ	TCP_DOS
Normal	57,490	61	52	4	0	1,939	25
BF1	1	46,001	12,287	1,279	0	3	0
BF2	67	43,858	12,769	2,327	0	550	0
BF3	2,756	6,554	19,879	26,602	0	3,780	0
IAUTHS	65	0	1	1	41,888	620	1
FUZZ	491	196	415	52	13	48,746	3
TCP_DOS	6	0	0	0	0	34	59,531

MLP Optimisation

Due to the poor performances recorded by the MLP classifier for flow-based features, the MLP classifier parameters were further analysed and tuned to improve the classifier accuracy. In an ANN, an activation function of a neuron maps the input signal to an output signal. Choosing the correct activation function supports the MLP classifier in generating more accurate and complex non-linear mappings between the inputs and outputs, hence improving the classifier accuracy (Karlik & Olgac, 2011). The solver functions refer to algorithms that try to estimate the optimal weights for the hidden and output layers in order to reduce the training errors. These are classified into first and second order methods and vary in computation complexity when they are minimising or maximising the loss function. Since Weka does not have options to vary the activation and solver functions, the MLP optimisation parameters were tested using Python scikit-learn ML platform (Pedregosa et al., 2011). On this platform, the performance was evaluated with three activation functions: Relu, logistic-sigmoid and tanh. In addition, two solver algorithms: Stochastic Gradient Descent (SGD) and limited-Memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) were compared for their optimisation performance in tuning the ANN weights for the FL-Sub-DS dataset. The SGD algorithm uses learning-rate and momentum to optimise the model by iteratively estimating the training loss using samples from the training dataset. The optimal momentum and learning-rate were identified by iteratively varying the two variables and the settings, selecting the least training loss for evaluating the activation functions. Figure 5.33 and Figure 5.34 show the observed training loss of the MLP classifier for various values of momentum and learning rates, respectively. These results show that a momentum of 0.9 and a learning-rate of 0.001 yielded the least training loss.

Furthermore, Figure 5.35 shows the performance of MLP classifier indicating a higher detection accuracy with the relu activation function and applied for both SGD and L-BFGS optimisation algorithms.

Table 5.24 shows the confusion matrix for MLP classifier with optimal hyper-parameters and using Relu activation function. The results indicate that a major performance improvement was achieved especially in BF1, BF2 and BF3 attack classification. However, some of the BF3 attack instances were classified as normal.

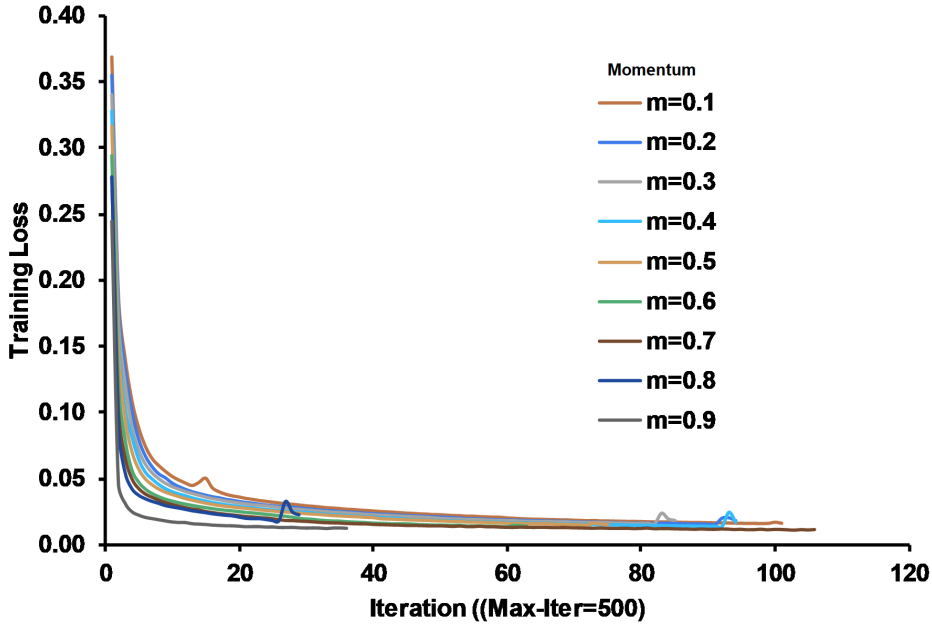


Figure 5.33: Training loss observed for the MLP classifier for various values of momentum and numbers of iterations on FL-Sub-DS dataset

Table 5.24: Confusion matrix with MLP classifier on FL-Sub-DS dataset with full features after choosing optimal Hyper-parameters

Actual	**Predicted**						
	Normal	BF1	BF2	BF3	IAUTHS	FUZZ	TCP-DOS
Normal	59,510	0	1	59	0	1	0
BF1	0	59,561	2	7	0	1	0
BF2	1	1	58,697	287	82	493	10
BF3	5,921	1	211	53,293	44	101	0
IAUTHS	0	3	526	5	41,959	80	3
FUZZ	130	30	532	67	34	49,121	2
TCP-DOS	0	0	23	3	3	7	59,535

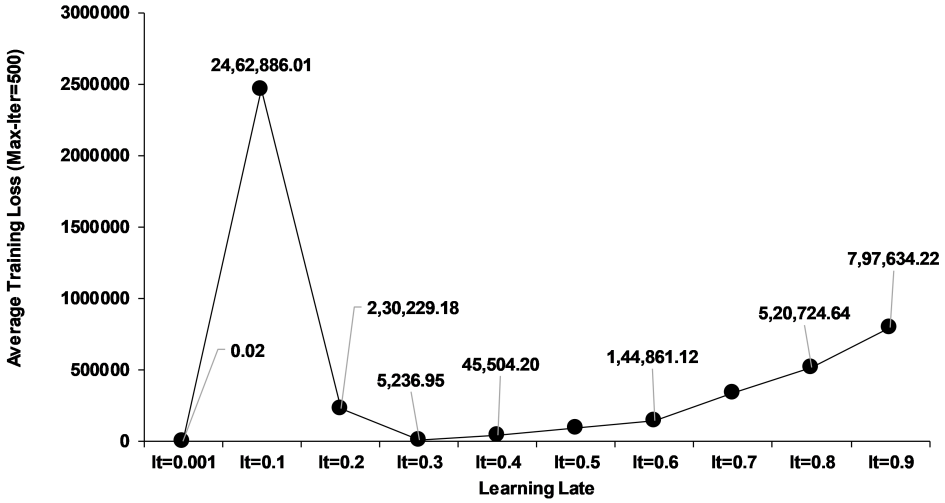


Figure 5.34: Average training loss for the MLP classifier calculated for various values of learning rate with a maximum of 500 iterations on FL-Sub-DS dataset

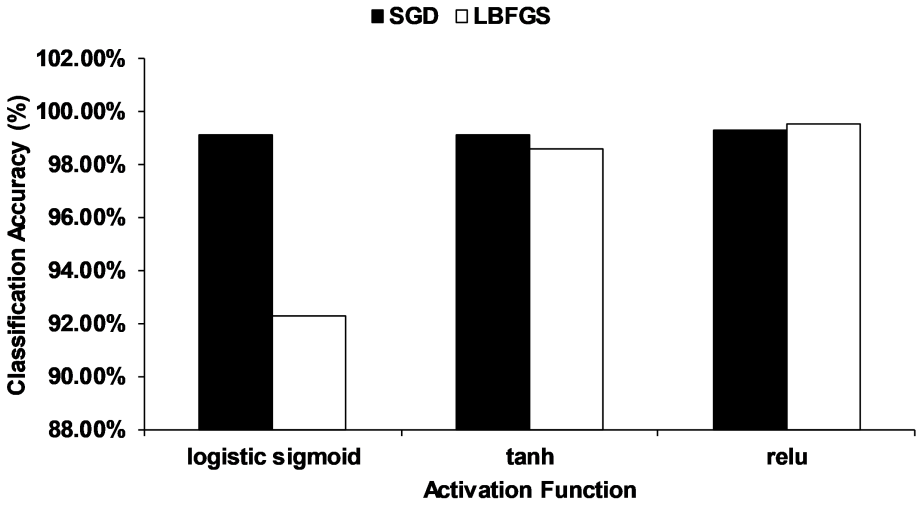


Figure 5.35: Performance evaluation of the activation function and MLP optimisation solver functions on FL-Sub-DS dataset

5.4 Summary

This chapter presented all the empirical observations of the impact of DoS attack scenarios on three MQTT brokers and the performance of the intelligent detection framework in detecting MQTT based DoS attacks. Several experiments were carried out to gather evidences which helped answer the questions posed in Chapter 1. The results were grouped into:

- Impact of four DoS attack scenarios on MQTT broker system performance
- Impact of four DoS attack scenarios on MQTT message delay and message publish rate
- DoS detection performance using Time-Window based features
- DoS detection performance using Flow based features

All the relevant findings were presented in this chapter and additional results are appended to the Appendix Additional Results. The system performance measurements were collected by controlling the parameters such as inter-arrival time of MQTT requests, WILL message payload size, CONNECT-Delay, number of subscription requests in a single session and number of threads used to launch the attacks. The impact of the attack scenarios was measured through CPU idle percentage representing the CPU utilisation, attack packets per second received, half-open sessions, and bandwidth utilisation.

Subsequently, the impact on MQTT message delay was measured by fixing the attack parameters. The delay was measured under normal no load conditions, with load condition (500 messages/second) and also for individual attack scenarios. The measurements were collected for the three QoS levels supported by MQTT. In addition to message delay, the impact on message publish rate was also measured with QoS1 messages, which share certain features with both QoS0 and QoS2 levels.

After the DoS attacks were evaluated, a detection framework was developed. Two network traffic aggregation levels were considered in this work namely: time-window aggregation and flow-level aggregation. A realistic IoT test-bed deployed using MQTT protocol was setup to collect normal and attack traffic and all the attack scenarios developed in previous phases were practically used to launch attacks on MQTT broker. Based on the aggregation levels, two different MQTT features sets were generated, which were further used to generate time-window based and flow-based datasets.

In order to test the proposed features, MQTT fuzzing and TCP-SYN flood attack were introduced into the dataset. However, TCP-SYN flood attack was only added to the flow aggregated dataset as this particular attack produced large variations in distributions of time-window aggregation making it a easy candidate for detection, but individual distinguishing individual TCP-SYN flows from other relevant flows will still be challenging.

For each aggregation level, two evaluation datasets were generated namely: major attack class dataset and attack sub-class dataset. Using the evaluation datasets, the classification accuracy of the classifiers chosen in the detection framework were assessed.

Chapter 6

Discussion

This chapter identifies the relationship between the proposed research questions posed in Chapter 1 with the empirical results presented in Chapter 5. This chapter also discusses the method used in conducting the experiments and also provides reasoning for the obtained results.

6.1 Purpose of the Study

The principal research question and its sub-questions posed in this work were:

- RQ1: How can Application Layer DoS attacks against the IoT-MQTT protocol be detected.
  - ✓ SQ1: Is the MQTT protocol vulnerable to Application Layer DoS attacks?
  - ✓ SQ2: Are the MQTT protocol based features required to detect targeted DoS attacks against MQTT-IoT system
  - ✓ SQ3: How effective are the developed ML models in correlating between normal and attack traffic

In order to answer these questions the following research steps were conducted:

- Threat modelling was performed on IoT-MQTT based system and the emphasis of DoS attack was presented (Section 4.1),
- The MQTT DoS attack model was proposed by identifying four attack methods which target the authentication and authorisation mechanisms of the MQTT protocol (Section 4.2),

- An MQTT DoS attack analysis test-bed (Section 4.3) was deployed to asses the impact of proposed DoS attacks on system resources, message delay and publish rates (Section 5.1),
- A realistic physical IoT-MQTT test-bed was deployed to generate normal and attack traffic (Section 4.4.1),
- A detection framework with three different ML algorithms was developed which extracted novel features from MQTT traffic. This resulted in two types of datasets based on two packet aggregation levels (time-window and flow-based) (Section 4.4),
- The effectiveness of proposed MQTT features were evaluated by comparing the classification accuracy of ML models with different subset of feature groups (Section 5.3), and
- The effectiveness of ML models were evaluated using labelled datasets containing major and sub-class of attacks (Section 5.3).

The MQTT DoS attack evaluations and DoS detection were performed in two different experimental phases as presented in Chapter 3. Splitting the design and experimental evaluation phases into two phases allowed experiments to be conducted independently. Experimental Phase-1 (RP2.1, RP2.2 and RP 3.1) was completed through rigorous evaluation and iterations allowing improvements to be made to the DoS attack scenarios. The DoS attack model of Phase1 was incorporated to generate MQTT attack datasets required for conducting the experiments in Phase-2 (RP2.3, RP2.4 and RP3.2). The following sections will present the inferences derived from the various empirical evidences obtained in Phase-1 and Phase-2.

6.2 MQTT DoS Attack Modelling

MQTT protocol is a publish/subscribe protocol and uses a broker to facilitate message exchange between publishers and subscribers. Since message brokers are pivotal to message exchange, they can become target to various cyber-attacks. The STIRDE threat model presented in Section 4.1 identified the DoS attack as a potential threat to IoT-MQTT system. In order to further verify the vulnerability of MQTT protocol to Application Layer DoS attacks, MQTT DoS attack modelling was performed to identify the DoS vulnerabilities. Based on the Little's queueing theorem (Eq. 2.1) two types of attacks can be modelled: flooding and semantic attacks. The flooding

attacks aim to increase the rate of arrival of connections into the system in order to consume the system queue and deny access to new legitimate connections. In contrast, semantic attacks aim to increase time spent by a connection in the system which will eventually fill the system queue denying access to new connections. Hence the two main variables that can be varied are the arrival rate (controlled using inter-arrival time) or the complexity of the request.

Various MQTT parameters were assessed for their DoS vulnerability however, only the basic CONNECT and SUBSCRIBE requests were considered as most commercial MQTT brokers enforce authentication and authorisation of users before allowing publishers and subscribers to exchange messages. Hence, the Application Layer DoS attacks were modelled to target the authentication and authorisation mechanisms of the MQTT protocol. Based on the arrival rate and complexity of the request, four novel MQTT DoS attack scenarios were proposed as discussed in Section 4.2.

Further experiments were conducted using the MQTT DoS analysis testbed consisting of three MQTT brokers deployed using virtual machines. The impact of increasing the request arrivals rate and and time spent by attack requests in the system were evaluated. Various methods were analysed to launch the flooding attacks and maximise the attack packets sent to the MQTT broker. Methods such as multi-processing and multi-threading were considered for launching the DoS attacks. However, the multi-threaded approach to launch DoS attack was chosen as it has various advantages compared to multi-processing system. The most important factor being that it occupies less memory and allows fast task switching compared to switching between multiple processes. Experiments were conducted to asses the number of attack threads that generated maximum attack packets which could result in higher CPU utilisation on the victim host as presented in Section 5.1.1. In addition to attack threads, the sleep-interval between requests was controlled to measure its impact on packet rate and CPU utilisation on victim host. The parameters of CONNECT request such as the WILL payload size (Section 5.1.1) and delay between the TCP session establishment and the first CONNECT request (Section 5.1.1) were controlled to assess their respective impact on the broker performance.

For the authorisation attacks, subscription flooding was chosen as the MQTT protocol allows multi topic subscription in a single request and in a single session. The PUBLISH request was not considered for DoS attacks as the MQTT protocol does not allow multi topic publishing. In order to asses the impact of sending multiple SUBSCRIPTION requests with invalid authorisations, a number of subscription requests in a single session were

controlled using a subscription loop (Section 5.1.1). The various treatments and the observation on IVs and DVs respectively in measuring the DoS attack impact are presented in Table 5.1.

The results of varying the attack threads and sleep intervals presented in Section 5.1.1 show that Mosquitto and VerneMQ brokers received higher average packets per second with the increase of number of attack threads. This was especially observed when using three attack threads for BF1 and BF2, along with a connect delay of 0.1 seconds, payload size of 43Kb and 50 or more subscribe loops. However, the VerneMQ broker received lower average packet per second with four and five threads. The behaviour observed on the VerneMQ broker can be explained by the number of TCP sessions in CLOSE_WAIT state. CLOSE_WAIT state indicates the termination of TCP connection by the remote host. With single, two and three threads, the number of TCP sessions in CLOSE_WAIT state recorded on the broker spiked irregularly whereas with four and five threads, there were a constant number of connections in CLOSE_WAIT state. This indicates that with four and five threads the attack machine's connection limit exhausted rapidly and fewer new attack requests were sent resulting in lower average PPS on the broker.

Comparing the three MQTT brokers, the EMQ broker received less packets with all choices of attack threads. Further investigating the EMQ broker shows that, EMQ broker moves the terminated TCP connections to CLOSE_WAIT state and the attacker host's TCP connections move to a FIN_WAIT2 and finally to TIME_WAIT state. This TCP state behaviour does not affect the number of new connections the broker can accept as the TCP session on the broker are configured to reuse port addresses. However, the attacker machine might have fewer available ports as outbound ports cannot reuse port addresses waiting in TIME_WAIT state to launch new connections, rapidly reducing the overall attack packets sent to the broker.

The various attack scenarios tested with the MQTT broker configurations show that flooding based DoS attacks had major impact on the CPU utilisation, memory and bandwidth consumption on the broker. The experimental results show that all the attack scenarios were successful in causing high CPU utilisation on the MQTT brokers. Figure 6.1 shows summary of CPU impact and the percentage of time the CPU was idle reduced below 25% for all attack scenarios with the following attack settings: three attack threads for BF1, mBF1 (malformed) and BF2, 0.1 connect delay for BF3 and single attack thread with 200 subscription loops for IAUTHS attack. The VerneMQ and EMQ brokers had the maximum impact as the CPU idle percentage reached zero for more than one attack scenario. However, the

idle percentage for Mosquitto broker was close to 20% and reached zero% for IAUTHS attack. The results also show that invalid subscription flooding attack caused the maximum impact on the CPU utilisation as all the brokers had CPU idle % below five percentage.

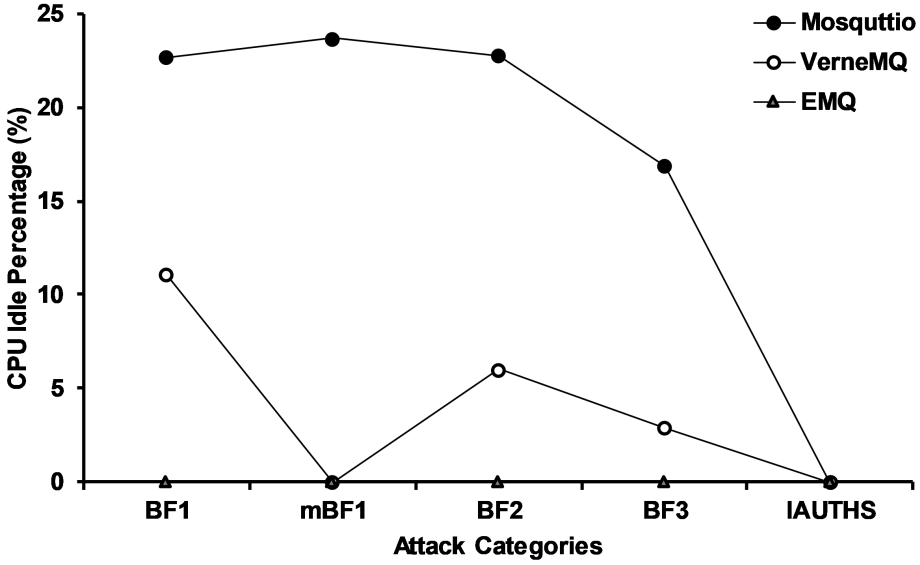


Figure 6.1: CPU Idle percentage of three MQTT brokers during various attack scenarios

The CPU utilisation breakup in Table 6.1 shows that the during the various attack scenarios the EMQ broker spent more time in IOWait and application related processing. Especially, the EMQ broker CPU was in IOWait state with basic CONNECT flood (BF1) and CONNECT with WILL message flooding (BF3). Similarly VerneMQ broker CPU was in IOWait state for malformed CONNECT flood attack. High IOWait state also indicates that the CPU is available for other tasks unless it is waiting for an asynchronous I/O operation which blocks the execution of other operations. Compared to the other brokers, the Mosquitto broker spent more time in kernel functions and software interrupts. This shows that various MQTT brokers employ different techniques to handle connection requests as mentioned in Karagiannis et al. (2015).

A summary of memory utilisations of the three brokers during attack scenarios shown in Figure 6.2 indicates that VerneMQ and EMQ brokers are vulnerable to malformed packets when non-ASCII characters are added

Table 6.1: Summary of impact on CPU utilisation breakup of three MQTT brokers during various attack scenarios

Attack Type	Broker	%usr	%sys	%iowait	%soft	%idle
BF1	Mosquitto	7.08	28.42	0.02	41.77	22.69
	VerneMQ	31.95	26.69	0.00	30.25	11.08
	EMQ	20.92	10.12	61.78	7.16	0.00
mBF1 (malformed clientid)	Mosquitto	6.84	26.79	0.01	42.66	23.69
	VerneMQ	36.03	13.56	31.86	18.55	0.00
	EMQ	36.39	7.78	44.10	11.73	0.00
BF2	Mosquitto	7.17	30.88	0.01	39.19	22.75
	VerneMQ	32.79	30.04	0.00	31.13	6.03
	EMQ	21.27	11.81	59.74	7.17	0.00
BF3	Mosquitto	4.63	20.75	0.01	57.75	16.86
	VerneMQ	27.41	24.88	0.01	44.81	2.90
	EMQ	92.78	4.05	2.21	0.95	0.00
	Mosquitto	84.00	0.50	0.00	15.50	0.00
IAUTHS	VerneMQ	82.47	1.33	0.00	16.21	0.00
	EMQ	90.67	0.40	0.00	8.93	0.00

to the MQTT fields. Both the brokers suffered high memory utilisation with the use of non-ASCII characters in the MQTT fields. This can be potentially exploited to cause memory exhaustion attacks to completely crash the broker. The EMQ broker had high memory utilisation during the CONNECT flooding attack with WILL payload (BF3).

Furthermore, the effect of using multi-core CPU and a load-balanced setup was analysed and presented in sections 5.1.2 and 5.1.3. The results indicate that increasing the number of CPU cores available for processing the attack requests reduced the impact of BF1, BF2 and BF3 attack scenarios. However, IAUTHS attack had similar impact on six CPU configuration as the single CPU configuration. Similarly, the load-balanced setup for EMQ broker reduced the overall impact of attacks but certain nodes of the cluster had higher impact compared to others. It is also important to note that the attack launched on Six-CPU configuration had the same hardware settings as Single-CPU attacks, hence using a higher configuration attack machine or using a distributed attack sources can have larger impact on the brokers.

In addition to system performance assessment with various attack sce-

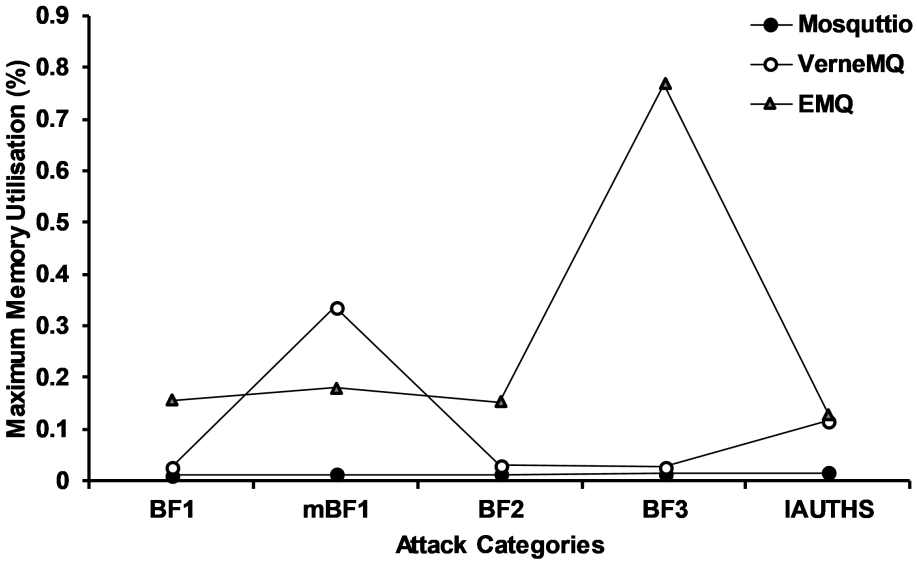


Figure 6.2: Memory Utilisation of three brokers during various attack scenarios

narios, the impact on the MQTT message delay and publish rate was performed in Section 5.2. Figure 6.3 shows the summary of percentage increase in average delay measurements obtained using the three MQTT brokers and the three QoS levels compared to the normal with load settings. The delay results indicate that the IAUTHS DoS attack scenario caused highest percentage increase in delay compared to all the scenarios. In BF3 attack scenario, a lower delay was observed in VerneMQ broker for QoS2. The reason for this could be that server can either delay the acknowledgements for each packet or reduce the number of acknowledgements, which could reduce time taken by QoS2 messages to complete compared to normal operations where server acknowledges all the packets. The average delay increase shows the delay for BF1, BF2 and BF3 attack scenarios did not drastically increase compared to normal delay however, the 95^{th} percentile delay shows a higher increase in all the attack scenarios especially in QoS2 messages. The increase in delay for QoS2 messages can be detrimental to the critical applications as QoS2 messages are essential for exchanging critical data in which the protocol guarantees once delivery.

The message publish rates observed during the attack scenarios was measured and a summary of percentage drop of QoS1 messages with respect to

normal rate is shown in Figure 6.4. The results indicate that the Mosquitto broker suffered publish rate drop for three attack scenarios and IAUTHS was successful on all the brokers causing drop in publish rates. In addition, VerneMQ broker showed a marginal increase in publish rate which could be due to duplicate QoS1 messages which is permitted in the MQTT protocol. The EMQ broker had more stable message publish rates for three CONNECT flooding attacks and lower rate for IAUTHS attack. This indicates that even though EMQ broker CPU utilisation was in the IOWait state during the attack, there was enough CPU time available for processing the MQTT messages. However, this came at the cost of higher message delay. Overall, VerneMQ broker had superior performance compared to all the three brokers in terms of system resource utilisation and its impact on message delay and publish rates.

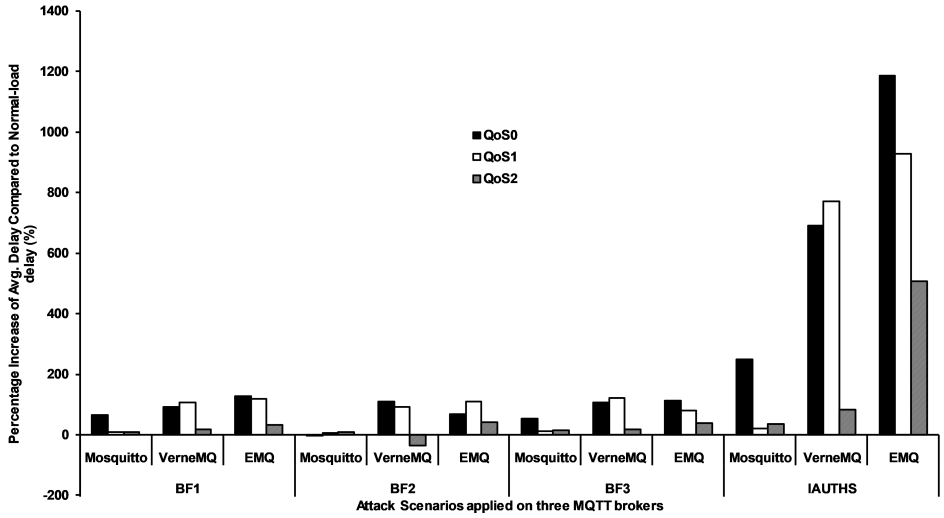


Figure 6.3: Summary of percentage increase of average MQTT message delay (ms) compared to normal delay observed on three Raspberry Pi clients using three brokers while exchanging QoS0, QoS1 and QoS2 messages

The attack scenarios and the results obtained in this work were based on a single attack source and with only MQTT traffic on the test network. Hence, the real attack scenarios with real load conditions and launched from multiple attack sources can have higher impact on the broker. This would require a true experimental research design compared to the quasi-experimental research design chosen in this thesis.

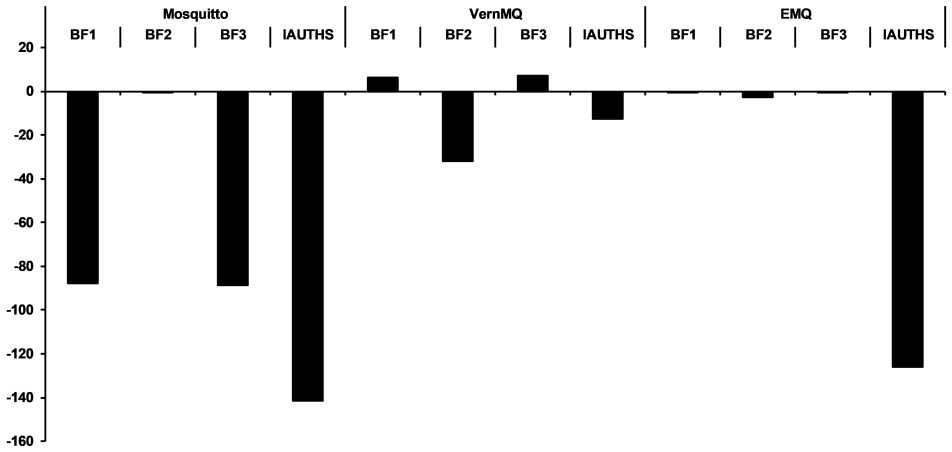


Figure 6.4: Summary of percentage drop of message publish rates observed on Raspberry Pi client measuring the message publish rate of QoS1 messages published by 1000 concurrent connections publishing at 500 message per second though three MQTT brokers

6.3 MQTT DoS Attack Detection

The MQTT attack modelling phase validated the vulnerability of the MQTT protocol to Application Layer DoS attacks, targeting authentication and authorisation mechanism. The experimental results indicate that MQTT DoS attacks can be launched by flooding the broker with control packets to cause CPU and memory exhaustion, which can adversely impact the message delay and publish rates, especially the critical messages sent with the highest quality of service. Hence, detecting MQTT DoS attacks is essential to protect the IoT-MQTT systems from its adverse impacts. The phase-2 of this work focused on detecting MQTT based DoS attacks by developing a detection framework. Novel MQTT based features were identified and their effectiveness in detecting DoS attacks was verified and presented in Section 5.3. Based on the network packet aggregation levels presented in Section 2.5.3, two detection approaches were analysed: Time-Window based detection and Flow-based detection. Time-Window based detection aims to detect anomalous time-windows which correspond to DoS attacks. Since DoS attacks cause changes to network traffic distributions, a Time-Window based detection allows developing attack mitigation techniques based on the attack traffic distributions to protect the MQTT brokers. However, stealth or semantic attacks can be configured to avoid causing major network traf-

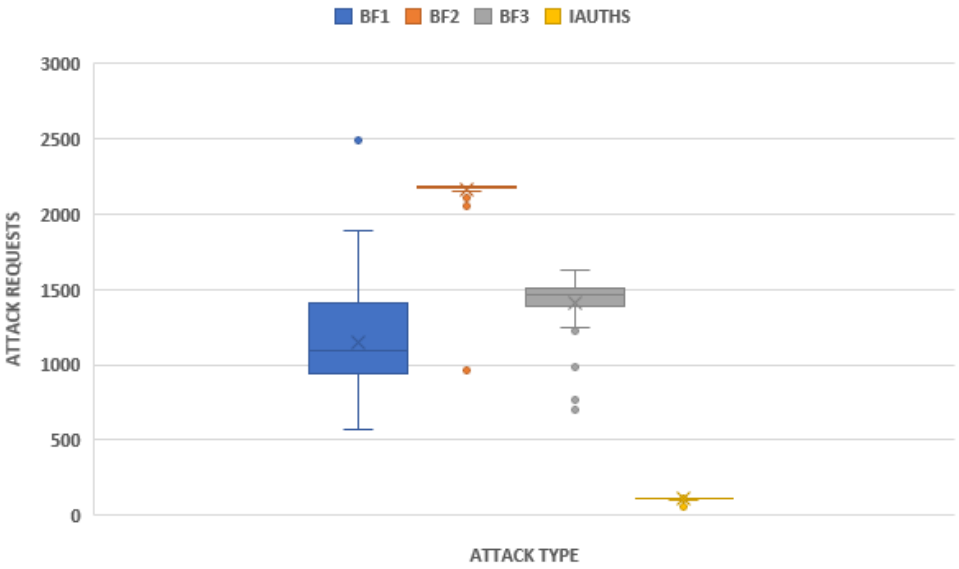


Figure 6.5: Variation in number of CONNECT packets in two second time window

fic distributions, thereby bypassing anomaly detection systems. As shown in Figure 6.5, each attack produces different volume of CONNECT request in a short time window, which could be challenging to detect using fixed thresholds.

In addition to detecting malicious time-windows, it would be beneficial to differentiate between normal and malicious flows in order to effectively terminate malicious flows before they cause any impact on the broker. Furthermore, as network devices already store flow related information for packet processing, using a flow based detection will result in less overhead compared to aggregating at other levels.

Hence, a flow-based MQTT attack detection using MQTT flow based features was proposed in addition to time-window based detection. The empirical results of time-window based detection were presented in Section 5.3.3 and the flow-based detection results were presented in Section 5.3.3. The detection framework was evaluated using two labelled datasets for each aggregation level (TW-Major-DS, TW-Sub-DS, FW-Major-DS and FW-Sub-DS). In order to assess the capabilities of the proposed features, the performance of ML models built using four feature vector groups (FULL-FV, TCP-FV, COUNT-FV and Size-FV) were compared. The feature vector

groups selected for evaluation in this study were chosen to highlight the detection capability of existing features proposed in the literature (TCP-FV) in isolation versus using the existing features along with the MQTT protocol features proposed in this work (FULL-FV). In addition, the proposed features were also divided into COUNT based features (COUNT-FV) and SIZE based features (SIZE-FV) to show the significance of the two classes of proposed features in detecting the attacks presented in this work. Figure 6.6 and Figure 6.7 show the summary of detection performance of chosen feature vector groups and the ML classifiers selected for classification in time-window dataset.

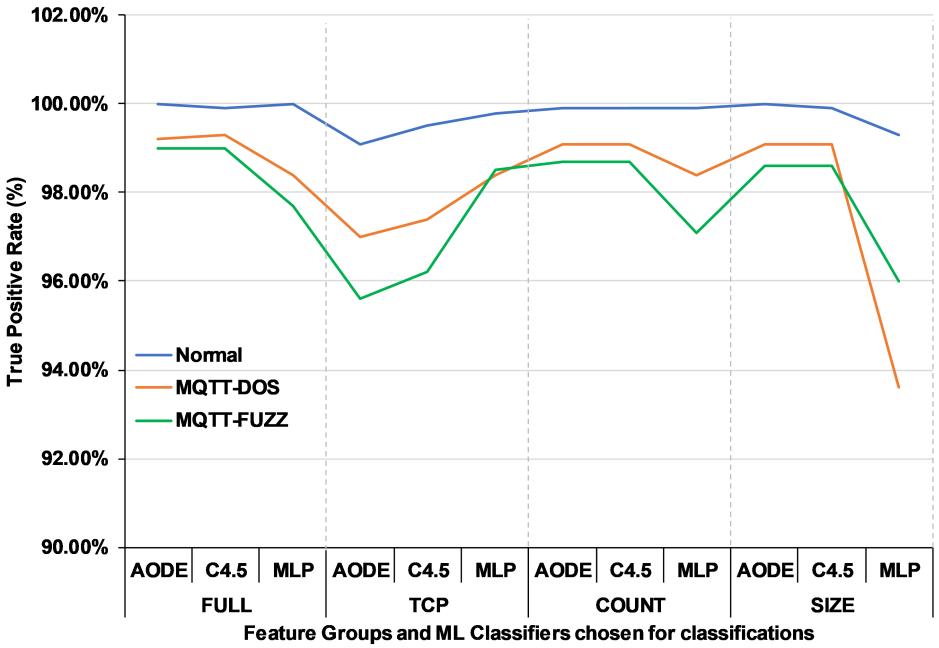


Figure 6.6: TPR of individual classes for ML models and feature vector groups chosen for TW-Major-DS time-window based detection

For the time-window datasets, the results indicate that using the full feature (FULL-FV) set (TCP + MQTT features) provided higher true positive rates (TPR) compared to only using the TCP based features. The results also indicate that MLP classifier had lower accuracy in detecting attacks compared to AODE and C4.5 classifiers. In TW-Major-DS, the MQTT Fuzzing attack class showed lower TPR rates with all the classifiers compared normal and MQTT DoS class. However in TW-Sub-DS, the BF1, BF2

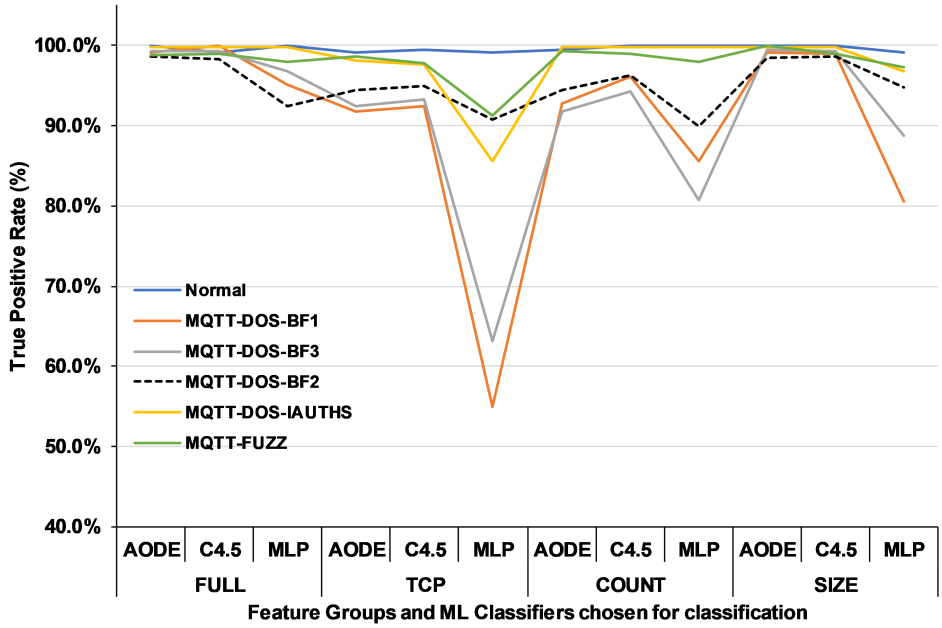


Figure 6.7: TPR of individual classes for ML models and feature vector groups chosen for TW-Sub-DS time-window based detection

and BF3 attacks showed lower TPR rates compared to other classes. The reason for this could be that BF1, BF2 and BF3 attacks have similar attack characteristics as well as have similarities with normal traffic which resulted in mis-classifications. The size based features provided higher separation between the attack classes for AODE and C4.5 classifiers. The superior detection accuracy with full feature set indicates that the proposed feature set had better detection accuracy compared to only using TCP based features proposed in existing literature.

A similar performance was observed with flow-based features as shown in figures 6.8 and 6.9 which illustrate the summary of detection performance of chosen feature flow groups and ML classifiers in flow-based dataset. The results show that the classifiers performed better in detecting attack classes in FW-Major-DS compared to attack classes in FW-Sub-DS dataset. However, the TPR rates for individual classes indicate that just using TCP based features reduced the classifier accuracy, especially the MLP classifier.

This indicates that the performance of all the three classifier techniques was affected with the introduction of sub-attack classes. Specifically, the

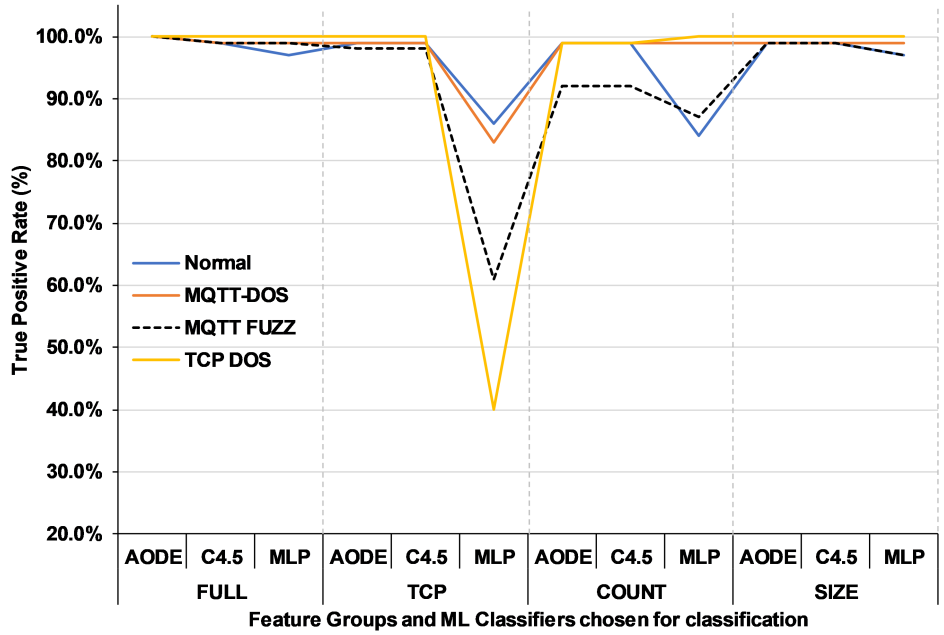


Figure 6.8: TPR of individual classes for ML models and feature vector groups chosen for FL-Major-DS flow based detection

MLP classifier's performance with full features (FULL-FV) degraded drastically and required hyper-parameter tuning to identify optimal setting for performance improvement. In contrast, the AODE and C4.5 classifier techniques recorded superior attack type detection performance with full features without requiring addition tuning.

The performance of the three classifier techniques was also affected with the use of TCP (TCP-FV), Count (COUNT-FV) and Size (SIZE-FV) based features vector groups individually. Among the TCP, Count and Size based feature vector groups, the detection accuracy reduced considerably using the TCP and Count based features. On the contrary, the Size based features showed better detection in correctly classifying attack classes with AODE and C4.5 classifiers. This means that Size based features provided better separation between flow instances compared to TCP and Count features groups. These results also highlight the challenges in detecting malicious flows using TCP connection settings or the count based thresholds which do not vary much between flows. Hence, the MQTT protocol features that capture the distribution of protocol fields are essential for effective malicious

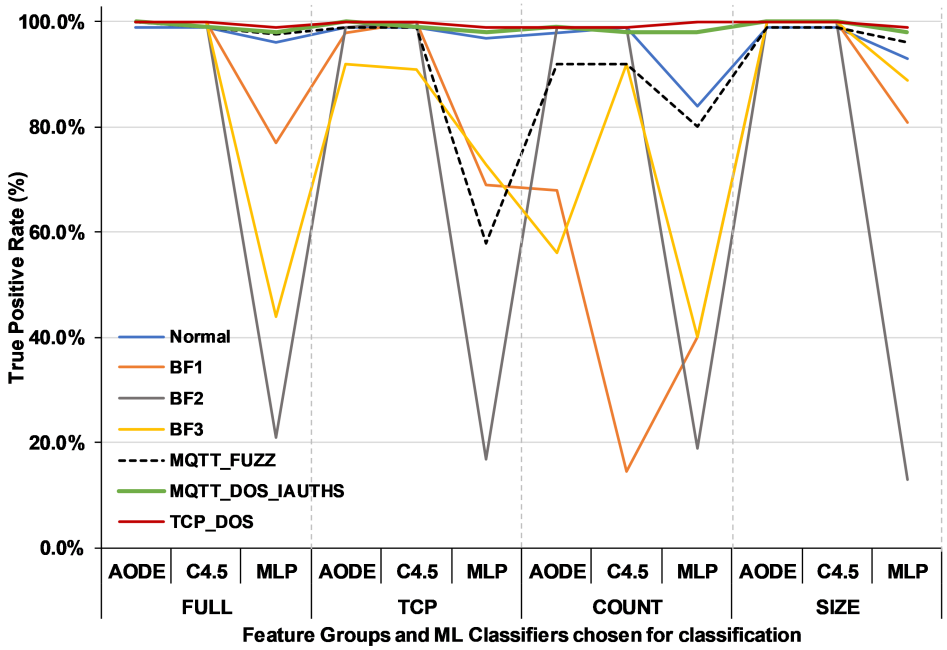


Figure 6.9: TPR of individual classes for ML models and feature vector groups chosen for FL-Major-DS flow based detection

flow detection.

The results obtained with full flow based features indicate that the full features provided a superior information gain to select split points between classes for C4.5 technique and provided a superior posterior probability threshold for AODE classifier technique to separate between classes. For the MLP classifier, tuning the hyper-parameters was required to improve the classifier performance. The parameters tuned for MLP classifier were learning rate, momentum and activation function. The results presented in Section 5.3.3 show that using a Relu activation function and a learning rate of 0.001 increased the classifier performance significantly. The poor results obtained with Sigmoid activation function can be attributed to the short range of values available to the derivative, resulting in information loss. In contrast, Relu activation function has higher range with output equal to the input for input values greater than zero.

Further analysis of various proposed features was conducted to identify features with greater contribution in classifying instances into multiple classes. The empirical results of MQTT attack detection to detect anoma-

lous time-windows and flows indicate that the proposed MQTT features increased the detection performance compared to using TCP features in isolation. The reason for this is that proposed features were capable of separating the normal and anomalous data points in the both time-window and flow datasets. Figure 6.10 shows the distribution of various time-window based features assessed in classifying anomalous time-windows in TW-Major-DS dataset into various attack categories. The distribution of MQTT features in Figure 6.10c, Figure 6.10d and Figure 6.10f demonstrate clear dissimilarity between the classes.

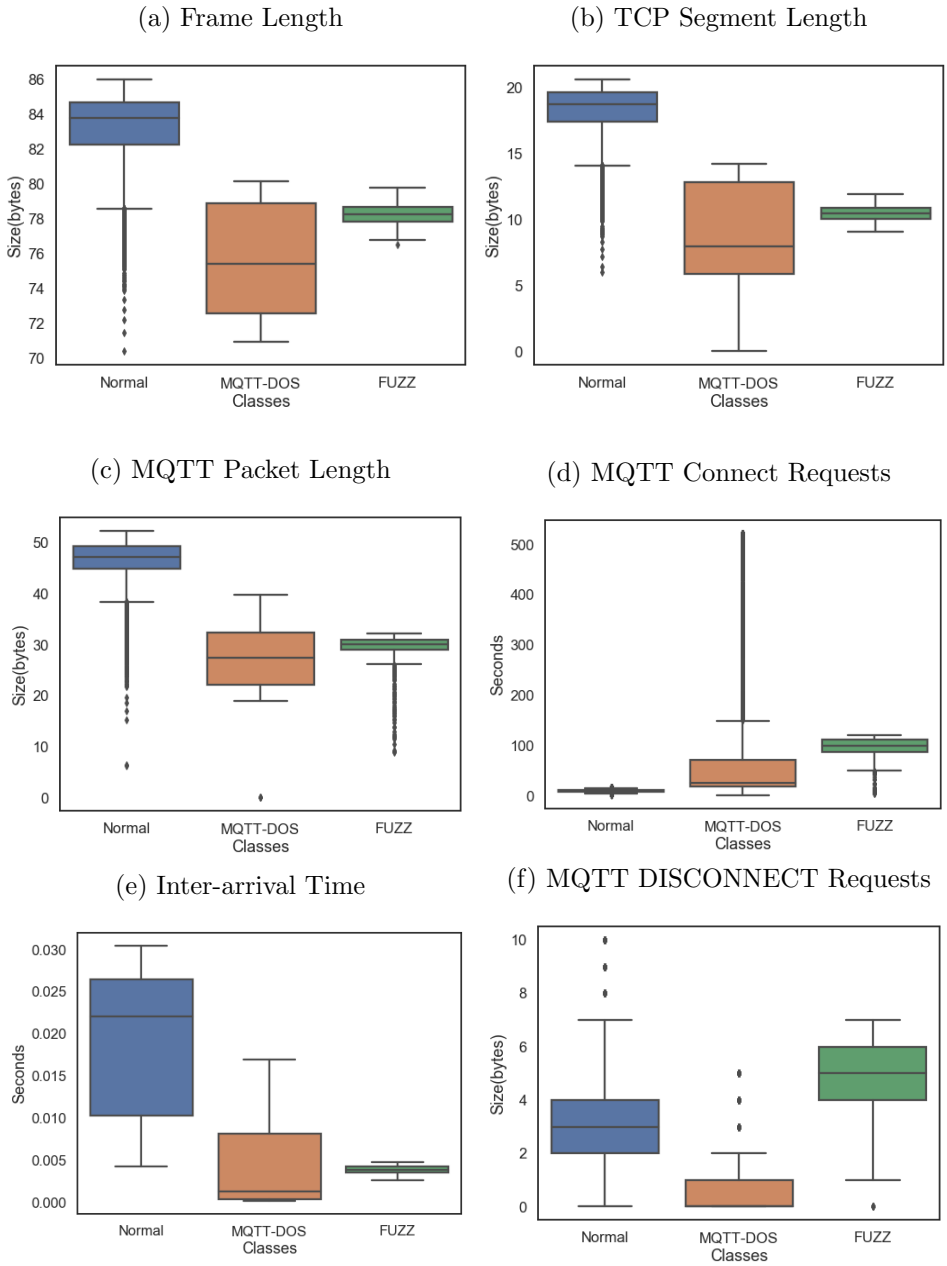


Figure 6.10: Boxplots showing the distribution of normal and attack classes for various features in TW-Major-DS dataset, (a) Frame Length (b) TCP Segment Length (c) MQTT Packet Length (d) MQTT CONNECT requests per window (e) Packet Inter-arrival Time (f) Disconnect Request per window. Both count based and size based features were required to accurately detect anomalous time windows

Figure 6.11 shows the distribution of various sub-classes for various proposed features assessed in identifying malicious time-windows. The distributions shows an overlaps between MQTT DoS attack classes and features such as average message length (Figure 6.11c), packet inter arrival time (Figure 6.11e) and MQTT WILL message length (Figure 6.11f) provide clear separation between the attack classes. The BF3 attack class shows a drop in average will message length as smaller WILL payload was configured for easier management of captured packets and feature extraction.

In time-window detection the combination of both size based and count based features showed good detection accuracy as both sets of features were required to differentiate between normal and attack instances. However in detecting malicious flows in flow-based dataset, size based features showed greater dissimilarity between normal and attack classes as highlighted by the distribution of classes for various features in FL-Major-DS dataset shown in boxplot presented in Figure 6.12. Specifically, MQTT message length (6.12c) and MQTT WILL message length (6.12f) provided greater separation between classes. Size based features such as Frame length, TCP segment length, MQTT packet length and WILL message length showed better dissimilarity between classes compared to count features such as TCP flow duration and SUBSCRIBE packet count.

With sub-class detection, the size based features showed clear separation between normal, invalid SUBSCRIBE flooding, MQTT fuzzing attack and TCP DoS attacks as highlighted in the distributions of classes for various features in FL-Sub-DS flow dataset shown in Figure 6.13. However, for distinguishing between three CONNECT flooding attacks, features such as flow-duration (Figure 6.13d) and WILL message (Figure 6.13f) were required as they provided clear dissimilarity between the MQTT DoS classes. This is because the attacks BF1 and BF2 use the same property of CONNECT flooding and only difference between them is the delayed sending of the CONNECT packet which is captured using the flow duration feature. Similarly, the difference between the BF3 and other CONNECT flooding is the use of WILL message which captures the separation between the MQTT DoS classes.

In addition to the distribution analysis of various features in separating normal and attack traffic, the correlations between all the features utilised for classification were also analysed. Features with high correlation provide redundant information and thus do not contribute towards anomaly detection. In contrast, features with low correlation score provide non-redundant information thereby contributing to the detection. The correlation plot of time-window based features shown in Figure 6.14 shows that the selected fea-

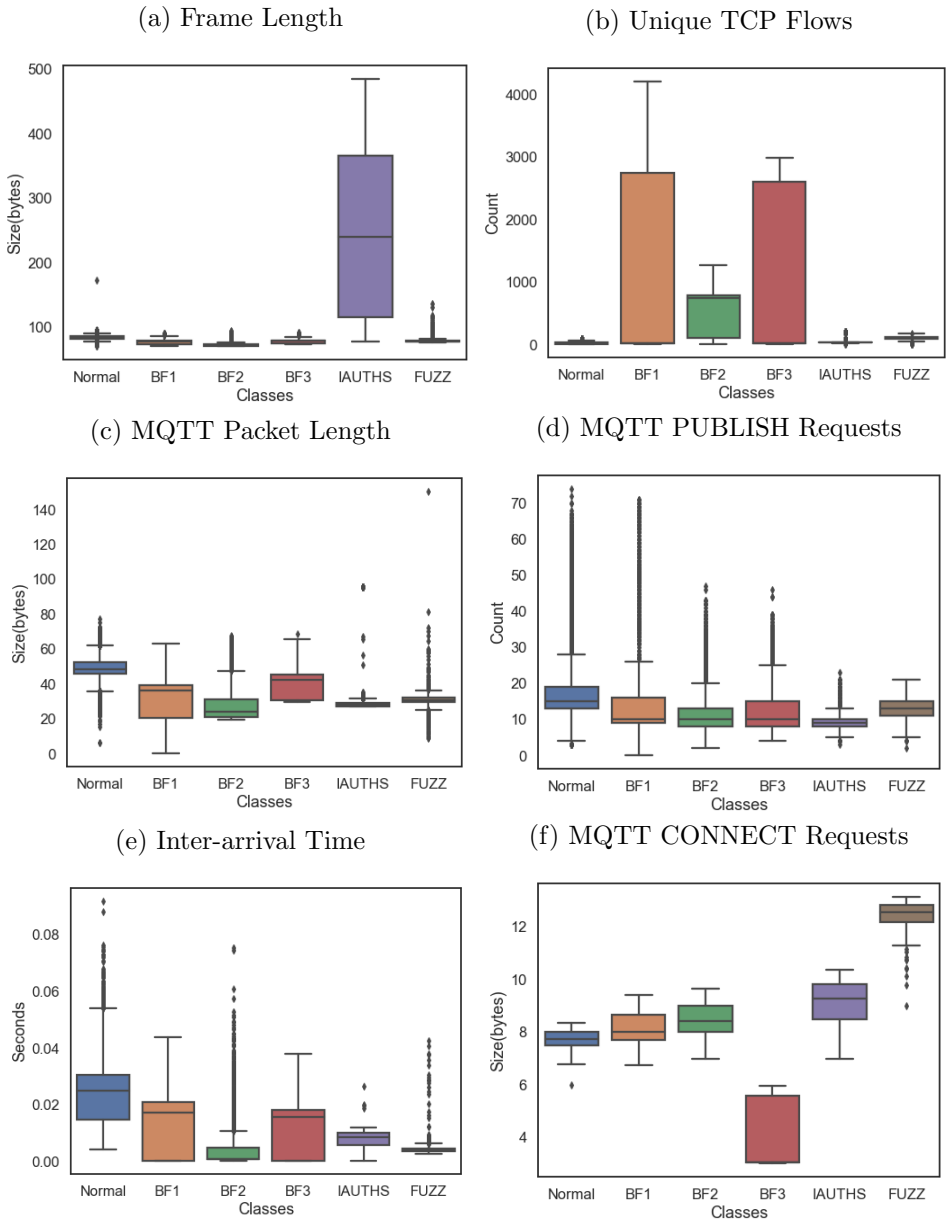


Figure 6.11: Boxplots showing the distribution of normal and attack classes for various features in TW-Sub-DS dataset, (a) Frame Length (b) number of unique flows in window (c) Avg. MQTT Packet Length (d) MQTT PUBLISH requests per window (e) Packet Inter-arrival Time (f) MQTT CONNECT requests per window. Both count based and size based features were required to accurately detect anomalous time windows

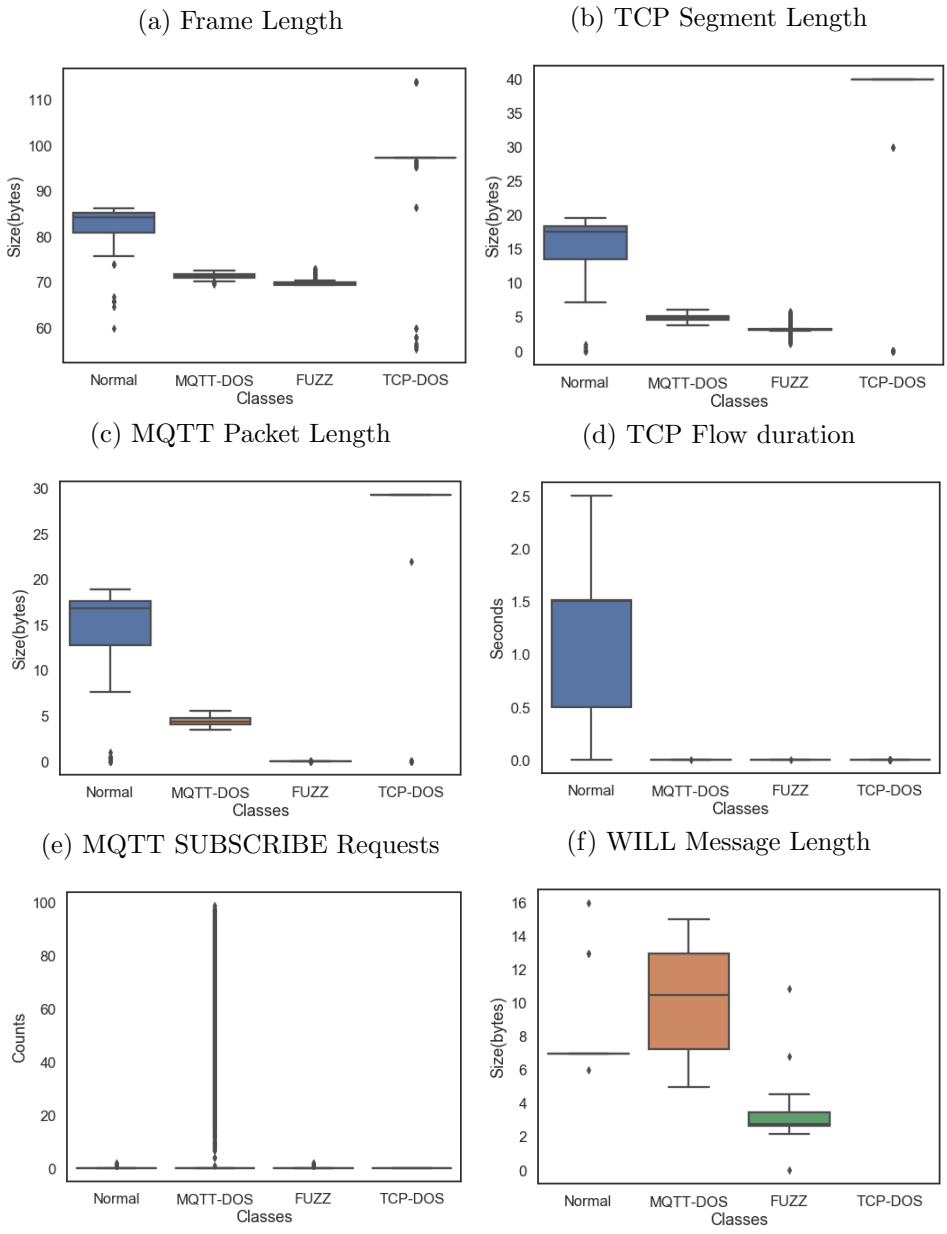
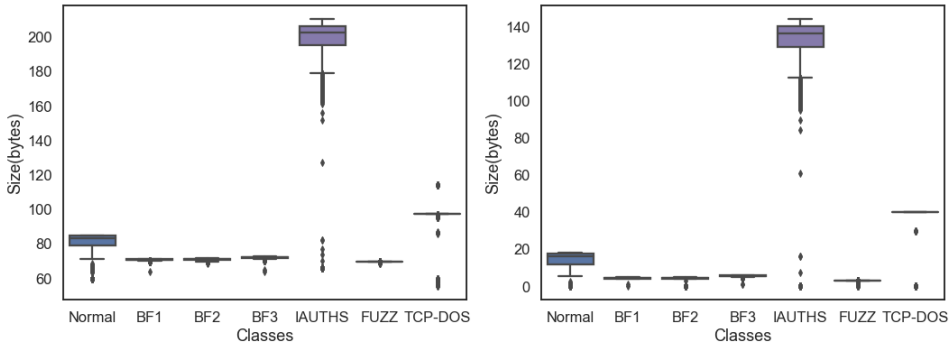


Figure 6.12: Boxplots showing the distribution of normal and attack classes for various features in FL-Major-DS dataset, (a) Avg. Frame Length (b) Avg. TCP Segment Length (c) Avg. MQTT Packet Length (d) TCP Flow duration (e) SUBSCRIBE packet count (f) Avg. WIll Message Length. Size based features showed better separation between classes compared to count based features.

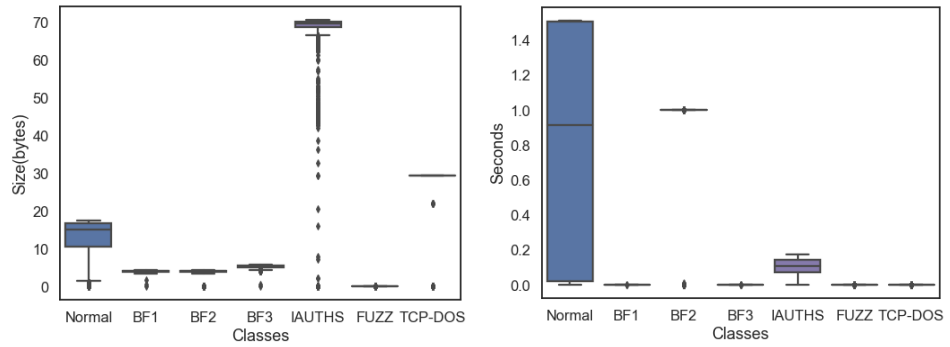
(a) Frame Length

(b) TCP Segment Length



(c) MQTT Packet Length

(d) TCP Flow Duration



(e) Packets Per Flow

(f) WILL Message Length

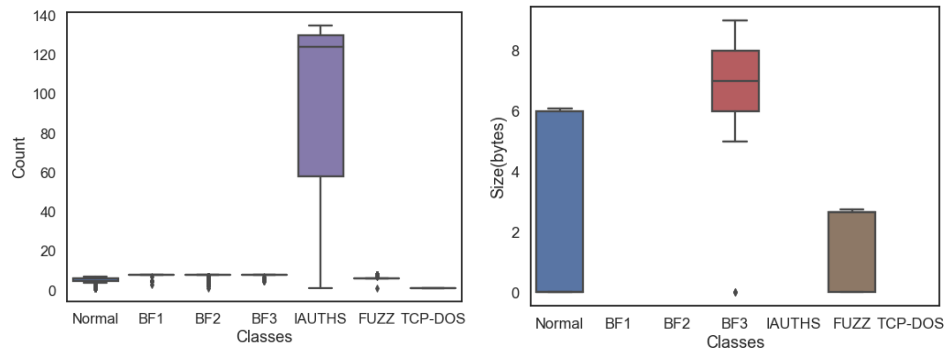


Figure 6.13: Boxplots showing the distribution of normal and attack classes for various features in FL-Sub-DS dataset, (a) Avg. Frame Length (b) Avg. TCP Segment Length (c) Avg. MQTT Packet Length (d) TCP Flow duration (e) Packets per flow (f) Avg. WIll Message Length. Size based features showed better separation between classes compared to count based features.

tures had weak relationships as most correlation scores was between -0.4 and 0.4. This shows the suitability of the selected features in detecting MQTT based attacks. A few features showed strong correlation such as cleanSessionSet and Connect_Command as CONNECT request either has the clean session flag enabled or disabled. In the selected dataset, most CONNECT sessions had cleanSession set, but this might differ in real settings and can be applied to detect attacks that exploit the session persistence functionality of the MQTT protocol. Compared to time-window features the flow based features had low correlation scores as shown in the correlation plot for flow based features in Figure 6.15. This highlights that most features had correlation score between -0.4 and 0.4 indicating weak associations between each other making them suitable for detecting malicious flows.

The distribution of normal and attack classes and correlations between various features show that overall the proposed features showed clear distinction between normal and attack instances, providing high true positive rates and low false positive rates. Hence these features are suitable to detect MQTT based DoS attacks. One of the key findings of this work is that the packet size and MQTT field length distributions provide high differentiation between normal and attack traffic when compared to count-based statistical features. The reason for this is that the MQTT protocol has various control packets with many custom fields to exchange small and pre-configured messages between individual devices, and deviations from the normal size/length distributions can effectively differentiate anomalies. Hence, packet size distribution and various packet header field lengths are suitable for anomaly detection in IoT traffic. In addition, most IoT devices are pre-configured to send specific content such as temperature, pressure, humidity or binary values along with specific message identification fields and are not randomly chosen by the device during IoT operations, hence these parameters can effectively characterise IoT traffic. In this work, various size/length based features such as MQTT payload size, topic length, WILL payload, WILL message length were considered in the detection framework, to differentiate between normal and attack traffic.

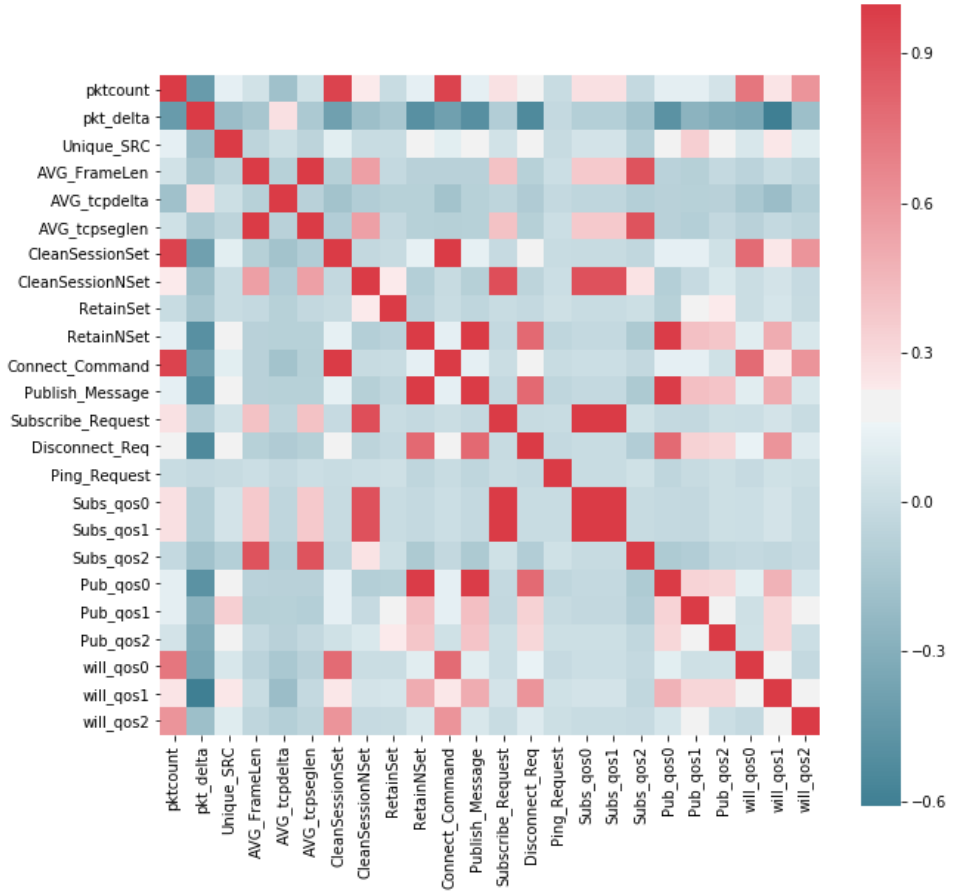


Figure 6.14: Correlation Plot of proposed time-window based features used in malicious time-window detection approach

6.4 Research Question Outcomes

6.4.1 SQ1: Is MQTT protocol Vulnerable to Application Layer DoS attacks?

In order to answer this question, the MQTT protocol specifications were assessed to identify potential DoS attack scenarios. The authentication and authorisation based DoS attacks targeting the MQTT broker based on control packet flooding was evaluated. The impact of DoS attack on MQTT broker system performance and overall impact on message delay and publish rate was assessed. Possible DoS scenarios using MQTT CONNECT (authen-

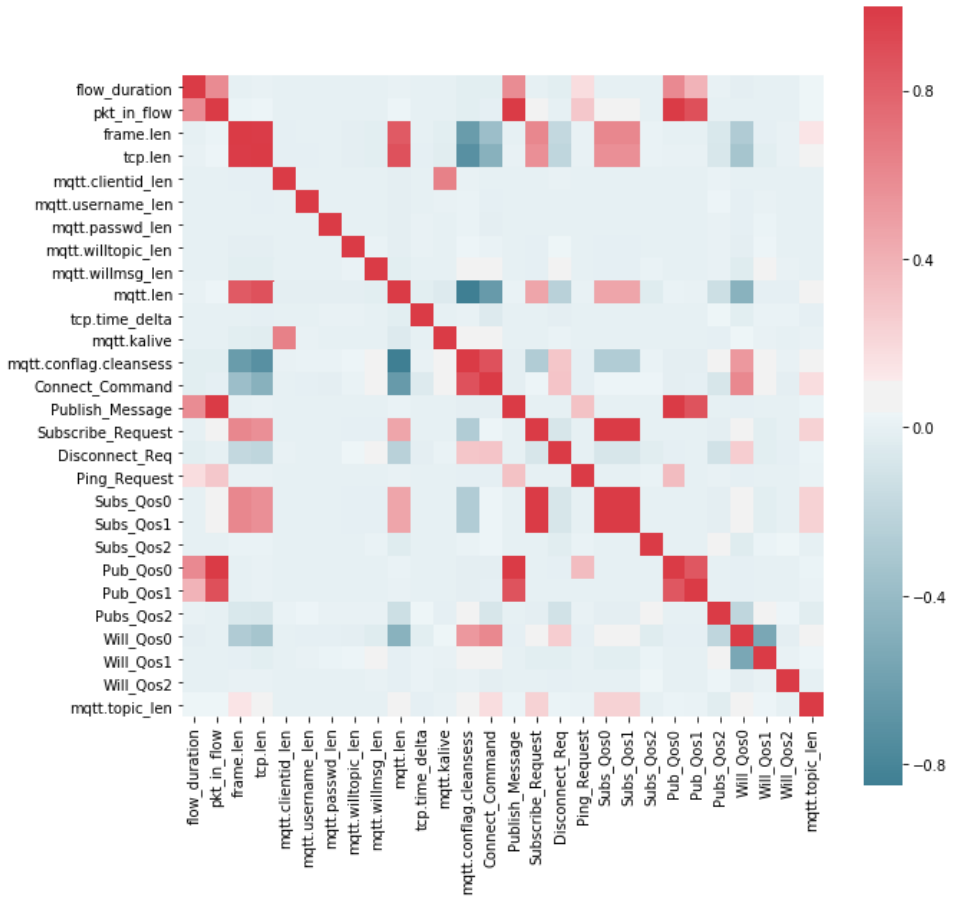


Figure 6.15: Correlation Plot of proposed flow-based features used in malicious flow detection approach

tication attacks) and SUBSCRIBE control packets (authorisation attacks) was modelled and was presented in Section 4.2. A DoS evaluation testbed was setup and four DoS attack scenarios which attack the authentication and authorisation mechanism of the MQTT protocol. In order to validate the impact of DoS attacks the evaluation was assessed on three different brokers. In Section 5.1, it was identified that authentication and authorisation attacks had a considerable impact on CPU, memory and bandwidth utilisation. In addition, certain MQTT brokers were identified to be vulnerable to non-ASCII characters added in MQTT fields. It was also identified that authorisation based attacks had higher impact than the authentication

attacks.

Furthermore, the impact of DoS attacks on delay and publish rate was evaluated and presented in Section 5.2. It was identified that DoS attacks considerably increased the average message delay with QoS0 having the highest percentage increase in delay compared to QoS1 and QoS2. However, 95^{th} percentile results showed that DoS attacks caused a heavy-tail latencies in MQTT messages especially in QoS2 with delays increasing more that 2000 milliseconds in EMQ broker using SUBSCRIBE flooding attack. The publish rates measured during the DoS attack scenarios indicate that MQTT brokers suffered degraded publish rates indicating either message loss or heavily delayed messages.

The DoS attack impact evaluation show that the Application Layer DoS attack scenarios using MQTT control packets can be modelled using limited access privileges to the MQTT broker which can cause considerable impact on MQTT broker performance. These attacks can have adverse impact on messages exchanged through the brokers especially critical QoS2 messages. Such delays in critical messages can be detrimental to delay sensitive critical applications which require near real-time message delivery (Sun & Wang, 2012).

These empirical results establish that both flooding and semantic DoS attacks can be launched against the MQTT protocol to cause significant impact on broker performances and messages exchanged through them. Especially, the authentication and authorisation mechanisms of the protocol can be exploited to cause DoS attacks. Such attacks can also be detrimental to IoT devices operating in resource-constrained conditions as these message delays or failures can result in high resource consumption and battery usage.

6.4.2 SQ2: Are MQTT protocol based features required to detect targeted DoS attacks against MQTT-IoT systems?

Statistical and ML approaches have previously been employed in detecting DoS attacks as discussed in Section 2.5. However, targeted DoS attacks involving Application Layer protocols or application domains requires identifying new features based on protocol or domain to detect them as elaborated in Section 2.5.2. Hence to answer this question, both normal and attack MQTT traffic datasets were generated to derive features based on it. A physical MQTT based IoT system was deployed using physical IoT devices based on configurations adopted from real-world deployments. The normal states of MQTT protocol were studied and replicated in the phys-

ical IoT testbed to generate the normal MQTT traffic. Based on the DoS attack models identified in the previous phase, DoS attack traffic was generated. Furthermore, the analysis of both normal and attack traffic was conducted to identify the MQTT traffic features suitable for DoS attack detection. Time-window based and flow level aggregations were analysed to generate respective features sets which can detect anomalous time-windows and network flows. The proposed feature sets consisted of both generic TCP based features and MQTT protocol features which can effectively answer the sub-question (SQ2). Distinct ML algorithms were evaluated and model performances based on TCP and MQTT features were compared.

Findings presented in Section 5.3 show that combining MQTT based features with TCP features increased the detection accuracy of algorithms compared to only using TCP features. Hence MQTT based features are required for detecting targeted MQTT Application Layer DoS attacks. In addition to comparing the performance of MQTT and TCP features, the performance of COUNT based and size based features was evaluated. The COUNT based features measured the frequency of occurrence of MQTT control packets or specific parameters of the protocol. In contrast the SIZE based features captured the distributions of frame, segment, MQTT packet and field sizes. The comparison results presented in Section 6.3 indicated that size based features improved the detection accuracy of the models.

6.4.3 SQ3: How effective are the developed ML models in correlating between normal and attack traffic?

To answer this question, three fundamentally different ML classifiers were deployed along with major and sub-class labelled datasets to evaluate the effectiveness of the models in detecting normal and MQTT based attacks. A 10-fold cross-validation approach was selected to split the training and test dataset to prevent over-fitting of the model where all the instances are chosen in a test set at-least once. In addition, MQTT protocol based FUZZING attacks which differ from DoS attacks were added to evaluate the detection performance of the models built using proposed MQTT features. Since the testbed only consisted of single attack source, all TCP based features based on source and destination IP addressed were not used to prevent detection bias.

The findings in Section 5.3 highlight that the three classifiers had greater than 99% detection accuracy in classifying between normal and attack time-window classes in major (TW-Major-DS) and sub-class (TW-Sub-DS) labelled time-window based dataset. The AODE (based on Naive Bayes),

C4.5 (based on Decision Trees), MLP (based on ANN) classifier had similar detection performance. The models performed extremely well in detecting MQTT FUZZ attacks as shown in figures 6.6 and 6.7 indicating that a high TPR was achieved in detecting FUZZ attacks which also makes the proposed features suitable in detecting non-DoS attacks.

Similarly, in the flow feature based dataset the AODE and C4.5 classifiers showed superior detection accuracy of greater than 99% in detecting malicious flows belonging to both major (FW-Major-DS) and sub (FW-Sub-DS) attack classes. This also indicates that the models performed well even when the complexity of detection task was increased with sub-class detection without tuning any classifier parameters. However, MLP classifier required hyper-parameter tuning to increase the detection performance. The MLP classifier had high false positives in detecting delayed CONNECT flooding attack which had similarities with normal traffic. All the models performed well in detecting MQTT FUZZ attacks achieving greater than 98% TPR as shown in Figure 6.8 and 6.9.

6.5 Research Implications

6.5.1 Threat Modelling

The threat model presented for the MQTT based IoT system contributes to the threat models presented for IoT systems such as Atamli and Martin (2014). The threat models for IoT systems capture only the generic threats to an IoT system but do not capture all the threats to the MQTT based system. The reason for this is that the components employed by the protocol introduces new threats to the IoT system. Hence this research focused on identifying IoT-MQTT system specific threats using the STRIDE threat model.

6.5.2 MQTT DoS Attack Modelling

The DoS attack models presented in this work identify DoS vulnerabilities of the MQTT protocol. The significant findings of this work on DoS vulnerabilities in MQTT protocol are as follows:

- MQTT brokers are vulnerable to Application Layer DoS attacks as both flooding based and semantic attacks can be modelled against the MQTT protocol.

- Various MQTT fields can be exploited to send non-ASCII characters or malformed packets and broker vulnerabilities to such attacks were identified which resulted in CPU and memory exhaustion.
- Delay sensitivities of IoT clients operating in unreliable network connections can be exploited to launch stealth attacks which were highlighted using a delayed CONNECT attack.
- Adversaries with basic authentication access can cause an adverse impact to MQTT system even without, sufficient authorisations. This was demonstrated using the SUBSCRIBE flooding attack with invalid subscription requests.
- Rate limiting approaches will fail if the MQTT DoS attacks are launched from distributed attack sources as lower attack rates per source can be successfully adopted as demonstrated in this work with a multi-threaded attack launching approach.

The findings of this work indicate that stringent validations of MQTT brokers must be done prior deployment in industrial IoT systems, especially in CIs. In addition, having valid credentials or open authentication settings as in the case of public MQTT brokers that allow anonymous logins to connect to brokers, can be exploited by adversaries to cause a significant impact using DoS attacks. Therefore, it is undesirable to have simple authentication or no authentication in MQTT deployments. Similarly, robust authentication and authorisation techniques need to be enforced in critical production deployment scenarios such as CIs. The attack modelling showed that MQTT based DoS attack can have a significant impact on MQTT message delays, especially on tail latencies. This can have a significant impact in CIs which are sensitive to message delays and such DoS attacks can have a cascading impact on the services dependent on them.

6.5.3 MQTT Attack Detection

Many attack detection techniques have been proposed in the literature as discussed in Section 2.5. However, as identified in Section 2.5.2 attacks that target Application Layer protocols and domain specific attacks require identifying protocol or domain specific features for effective detection of attacks because Application Layer DoS attacks utilise legitimate connections and are capable of bypassing lower layer attack detection techniques. Hence in this work features specific to the MQTT protocol were identified that can

detect attacks against the MQTT based IoT system. This work specifically, contributes to an IoT attack detection scheme proposed by Moustafa et al. (2019) in which MQTT transaction based features were proposed. In addition, the work presented by Moustafa et al. (2019) did not contain any real MQTT attack traffic and the proposed features were based only on the analysis of TCP payloads. Hence, the results showing the effectiveness of the proposed features in detecting MQTT attacks were not presented. In contrast, the work presented in this thesis identifies the various MQTT protocol based features suitable for MQTT based attack detection and their effectiveness were analysed using the MQTT datasets collected on realistic IoT-MQTT testbed. Hence, a significant contribution of this work was to generate MQTT attack datasets which contain DoS attacks proposed in this work as well as MQTT Fuzzing attacks proposed by Vähä-Sipilä (2015).

ML techniques have been effectively utilised in anomaly detection in the existing literature as discussed in Section 2.5. Based on this approach a detection framework which consists of a feature extraction module configured with statistical MQTT protocol based features and a detection module consisting of three fundamentally different machine learning algorithms was proposed. Two levels of feature aggregation: time-window and flow-based were used to evaluate the effectiveness of the proposed features in detecting anomalous time-windows and flows which contain attacks.

The proposed features showed significant detection results in distinguishing normal and attack instances compared to only using features based on TCP layer. Specifically features that were based on packet lengths and MQTT field lengths drastically improved the accuracy of detection. Such features can be further extended to detecting IoT specific attacks as IoT devices are configured to exchange predefined messages which have a distinct length distributions.

Chapter 7

Conclusion

This chapter presents the conclusions of this research work; which was focused on detecting the Application Layer DoS attacks against the IoT-MQTT protocol. To answer the primary research question, conclusions are drawn from the discussion of various findings presented in Chapter 6. The main contributions of this research to the body of knowledge are then presented. The chapter also presents the limitation of this research and provides future research directions.

7.1 How can Application Layer based DoS attacks against the IoT-MQTT protocol be detected?

In order to answer the primary research question, three sub-questions were posed and the answers to SQ1, SQ2 and SQ3 were presented in Section 6.4. The answer to SQ1 was presented in Section 4.1 by presenting an MQTT based threat model, which highlights that the protocol is susceptible to Application Layer DoS attacks. It was identified in Section 4.2, that Application Layer DoS attacks can be designed to target authentication and authorisation mechanism of the protocol, to effect the attack. The empirical results of the four DoS scenarios presented in Section 5.1 show that such an Application Layer DoS can have a significant impact on the broker system resources.

It was also identified that certain types of MQTT broker software can be vulnerable to malformed packets, to causing DoS. In addition, the delay and publish rate evaluations presented in Section 5.2 indicate that such DoS attacks can significantly impact the quality and delivery of messages exchanged through MQTT brokers, also introducing a heavy-tailed delay

especially in QoS2 messages. Such delays and publish rate drops can impact critical IoT messages exchanged through these brokers.

Even tough the impact of such DoS attacks can be reduced by increasing the processing capabilities of MQTT brokers, such attacks ought to be detected in a timely manner to prevent their harmful impact on critical MQTT-based IoT systems. This took us to pose question SQ2 and SQ3; to investigate the potential capabilities of MQTT protocol-based features in detecting the DoS attacks presented in Section 4.2.

The investigation was carried out by first designing an MQTT attack detection framework to capture normal and malicious MQTT traffic generated using the attack scenarios evaluated as part of SQ1. Furthermore, the detection framework comprised of statistical MQTT features, which were based on the two packet aggregation levels namely: time-window and flow levels. The effectiveness of the detection framework was evaluated using three fundamentally different machine learning algorithms, namely, AODE based on Naïve Bayes, C4.5 based on Decision Tress and MLP based on Artificial Neural Network. In order to compare the detection performance of the proposed features, classifier models built using four feature vector groups, namely, FULL-FV, TCP-FV, COUNT-FV and SIZE-FV were compared.

In order to reduce the bias exposed by these models in detecting attacks presented in this work, two new attack scenarios were introduced. The evaluations performed using 10-fold cross-validation and presented in Section 5.3 indicate that, AODE classifier had the most superior performance among the three classifiers evaluated. In addition, classifier models that used the FULL-FV feature set which included TCP-based, Count-based and Size-based features showed promising detection results. These results highlight that by simply relying on TCP based features and counter or frequency based features can result in DoS attacks remaining undetected in an IoT based detection system. Size based features that capture the distributions of the MQTT packets as well as various MQTT protocol fields provide an effective separation between normal and attack traffic. In conclusion, the empirical evidences collected in this research strongly suggest that the proposed MQTT based features can be used effectively to detect Application Layer DoS attacks against the IoT-MQTT protocol.

7.2 Significant Contributions of this Research

The significant contributions of this research are:

1. **Identification of threats to the IoT-MQTT protocol through**

the design of an MQTT threat model: this research comprehensively identified the various MQTT threats that can lead to potential cyber-security attacks against the IoT systems deployed, as presented in (Firdous et al., 2017),

2. **Devised and implemented four Application Layer DoS attacks against MQTT protocol**: this research presented four novel MQTT based DoS attack scenarios that can potentially cause DoS in IoT systems deployed upon this protocol. Such DoS attacks can significantly affect the broker system resources leading to degraded message exchange performance especially for critical QoS2 messages,

3. **Vulnerabilities identified for three open-source MQTT brokers**: the investigations conducted in this thesis for DoS attack evaluations identified potential vulnerabilities in three open-source MQTT brokers. Both VerneMQ and EMQ brokers were vulnerable to non-ASCII characters in packet fields and Mosquitto broker suffered severe publish message loss with flooding attacks,

4. **Generated and collected IoT-MQTT based attack data**: the normal MQTT protocol states were used to generate normal traffic and the DoS attack scenarios presented in this research were used to generate MQTT attack packets resulting in MQTT datasets containing normal and attack traffic,

5. **Identified MQTT based time-window and flow-based features to effectively detect MQTT attacks**: the research presented MQTT features for attack detection based on two aggregation levels namely: time-window and flow level. These features can be used to effectively detect MQTT based attacks.

6. **Size based features for effective attack detection in IoT were identified**: findings included that a superior IoT based detection system would require size based features as they yielded greater separation between normal and attack traffic distributions. This is because IoT devices are pre-configured to send specific type of messages with predictable size (pre-defined field length, message etc.) and differ greatly from human initiated traffic which could contain random size messages. Hence potential IoT detection systems can leverage this unique IoT feature in distinguishing between normal and attack traffic.

7.3 Limitations of this Study

Although the DoS scenarios and the detection mechanism presented in this research showed promising results, there are various limitation of this research which need to be highlighted.

One of the drawbacks of the four DoS attack scenarios and its evaluation presented in sections 4.3 and 5.1 is that the proposed DoS attacks were evaluated in a laboratory environment with brokers deployed on virtual machines. However, real networks would vary greatly from a laboratory environment in terms of control settings, background traffic, heterogeneity of IoT devices, and computing resources available for brokers, which could not be fully replicated in this study.

Another drawback of this research was that the DoS attacks were launched from a single attack machine based on a multi-threaded attack generator architecture, whereas real world DoS attacks are distributed in nature, involving a number of compromised devices. Hence the full potential of MQTT based DoS attacks could not be measured with a single attack source.

With regards to the MQTT based DoS attack detection, the dataset was generated for an IoT test-bed with limited number of IoT devices configured with single message exchange model and no background traffic. This also potentially explains the reasons for the models achieving high accuracy rates as the variations in normal and attack traffic were based only on limited number of IoT devices and a single attack source. However, in real world scenarios, thousands of heterogeneous IoT devices would operate along with other communication traffic, which could not be replicated in this research. Hence the models were evaluated only using test-bed generated traffic. With lack of real world IoT-MQTT attack datasets, and other studies that contain similar platforms, a direct comparison of results could not be made and all the findings reported in this thesis were based on the DoS simulations and detection results obtained through the test-bed evaluations. In addition, since the attack detection method relied on the packet and field size distributions between normal and attack traffic, the proposed features may fail if replay attacks are launched with previously captured legitimate network packets.

Furthermore, there were software based limitations such as the attack scenarios were evaluated only on open-source brokers and not on commercial brokers. Hence, the impact on commercial production deployments was not fully ascertained. Similarly, a third-party MQTT client library was used to develop the DoS attack scripts, which contained several client side validation measures that prevented maximising the impact of a DoS attack.

The variations in MQTT broker software design also impacted the number of attack packets that could be generated which resulted in variation on attack packets received by the brokers.

The limitation mentioned above regarding lack of similarity to real-world implementation or environments were ultimately intended as referred to the research design, i.e. laboratory experimentation. Furthermore, they were also unavoidable for practical reasons. However, the researcher is hopeful that further research will be able to overcome these limitations.

7.4 Future Work

The impact of DoS attacks on MQTT brokers were evaluated in this study, however, the impact on IoT devices in terms of battery usage due to re-transmissions, message delays and message loss is of interest. This research can be extended in identifying improvements to MQTT protocol in terms of authentication and authorisation mechanisms. More complex attack scenarios need to be studied for MQTT protocol to identify vulnerabilities and for generating attack datasets, in order to improve detection techniques. The DoS attack evaluation techniques can also be extended to other IoT Application Protocols in order to evaluate their vulnerabilities.

It was identified that the size based features provided better detection accuracy in attack detection. This can be extended in general to other IoT protocols to detect IoT based attacks. The effectiveness of the proposed features in detecting non-DoS attacks such as fuzzing attacks, data injection attacks, and other malicious attacks needs to be further evaluated. Dataset generation can also be greatly improved by introducing various IoT traffic models, including diverse MQTT attacks and through using a more realistic IoT test-bed. More focus should be given to IIoT threats and their detection system design, as IoT usage in the Industry increases.

References

Adi, E., Baig, Z. A., Hingston, P., & Lam, C.-P. (2016, Mar 01). Distributed denial-of-service attacks against http/2 services. *Cluster Computing*, *19*(1), 79–86. doi: 10.1007/s10586-015-0528-7

Alanazi, S., Al-Muhtadi, J., Derhab, A., Saleem, K., AlRomi, A. N., Alholaibah, H. S., & Rodrigues, J. J. (2015). On resilience of wireless mesh routing protocol against dos attacks in iot-based ambient assisted living applications. In *E-health networking, application & services (healthcom), 2015 17th international conference on* (pp. 205–210). doi: 10.1109/HealthCom.2015.7454499

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, *17*(4), 2347–2376. doi: 10.1109/COMST.2015.2444095

Alliance, O. M. (2019). *An application-layer approach to end-to-end security for the internet of things.* Retrieved from `https://www.omaspecworks.org/wp-content/uploads/2019/07/OMA-WP-e2e_Sec_IoT-20190606-C.pdf`

Al-Qaseemi, S. A., Almulhim, H. A., Almulhim, M. F., & Chaudhry, S. R. (2016). Iot architecture challenges and issues: Lack of standardization. In *2016 future technologies conference (ftc)* (pp. 731–738). doi: 10.1109/FTC.2016.7821686

Andrea, I., Chrysostomou, C., & Hadjichristofi, G. (2015). Internet of things: Security vulnerabilities and challenges. In *Computers and communication (iscc), 2015 ieee symposium on* (pp. 180–187). doi: 10.1109/ISCC.2015.7405513

Andrew Banks, R. G. (2014). *Mqtt version 3.1.1.* Retrieved from http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718028

Andy, S., Rahardjo, B., & Hanindhito, B. (2017). Attack scenarios and security analysis of mqtt communication protocol in iot system. In *2017 4th international conference on electrical engineering, computer science and informatics (eecsi)* (p. 1-6). doi: 10.1109/EECSI.2017.8239179

Angrishi, K. (2017). Turning internet of things (iot) into internet of vulnerabilities (iov): Iot botnets. *arXiv preprint arXiv:1702.03681*.

Asgari, S., & Nunes, J. B. (2011, July). Experimental and quasi-experimental research in information systems. In *Iadis international workshop information systems research trends: approaches and methodologies (isrtam 2011).* Retrieved from http://eprints.whiterose.ac.uk/75405/ (International association for development of the information society (iadis))

Asosheh, A., & Ramezani, N. (2008). A comprehensive taxonomy of ddos attacks and defense mechanism applying in a smart classification. *WSEAS Transactions on Computers*, *7*(4), 281–290.

Atamli, A. W., & Martin, A. (2014, Sept). Threat-based security analysis for the internet of things. In *2014 international workshop on secure internet of things* (p. 35-43). doi: 10.1109/SIoT.2014.10

Atzori, L., Iera, A., & Morabito, G. (2010). The internet of things: A survey. *Computer networks*, *54*(15), 2787–2805. doi: https://doi.org/10.1016/j.comnet.2010.05.010

Aziz, B. (2016). A formal model and analysis of an iot protocol. *Ad Hoc Networks*, *36*, 49 - 57. doi: https://doi.org/10.1016/j.adhoc.2015.05.013

Baig, Z. A., Sanguanpong, S., Firdous, S. N., Vo, V. N., Nguyen, T. G., & So-In, C. (2020). Averaged dependence estimators for dos attack detection in iot networks. *Future Generation Computer Systems*, *102*, 198 - 209. Retrieved from http://www.sciencedirect.com/science/article/pii/S0167739X19300834 doi: https://doi.org/10.1016/j.future.2019.08.007

Ballani, H., & Francis, P. (2008). Mitigating dns dos attacks. In *Proceedings of the 15th acm conference on computer and communications security* (pp.

189–198). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1455770.1455796 doi: 10.1145/1455770.1455796

Banks, A., & Gupta, R. (2014). Mqtt version 3.1. 1. *OASIS standard*. Retrieved from http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

Bekara, C. (2014). Security issues and challenges for the iot-based smart grid. *Procedia Computer Science*, *34*, 532 - 537. Retrieved from http://www.sciencedirect.com/science/article/pii/S1877050914009193 (The 9th International Conference on Future Networks and Communications (FNC'14)The 11th International Conference on Mobile Systems and Pervasive Computing (MobiSPC'14)Affiliated Workshops) doi: https://doi.org/10.1016/j.procs.2014.07.064

Bello, O., & Zeadally, S. (2016, Sep.). Intelligent device-to-device communication in the internet of things. *IEEE Systems Journal*, *10*(3), 1172-1182. doi: 10.1109/JSYST.2014.2298837

Bencsath, B., & Ronai, M. A. (2007). Empirical analysis of denial of service attack against smtp servers. In *2007 international symposium on collaborative technologies and systems* (p. 72-79). doi: 10.1109/CTS.2007.4621740

Bhatia, S., Behal, S., & Ahmed, I. (2018). Distributed denial of service attacks and defense mechanisms: Current landscape and future directions. In *Versatile cybersecurity* (pp. 55–97). Springer. doi: 10.1007/978-3-319-97643-3_3

Bhuyan, M. H., Kashyap, H. J., Bhattacharyya, D. K., & Kalita, J. K. (2013). Detecting distributed denial of service attacks: methods, tools and future directions. *The Computer Journal*, *57*(4), 537–556. doi: https://doi.org/10.1093/comjnl/bxt031

Blazek, R. B., Kim, H., Rozovskii, B., & Tartakovsky, A. (2001). A novel approach to detection of denial-of-service attacks via adaptive sequential and batch-sequential change-point detection methods. In *Proceedings of ieee systems, man and cybernetics information assurance workshop* (pp. 220–226). doi: 10.1109/TSP.2006.879308

Bojović, P., Bašičević, I., Ocovaj, S., & Popović, M. (2019). A practical approach to detection of distributed denial-of-service attacks using a hybrid detection method. *Computers and Electrical Engineering*, *73*, 84 -

96. Retrieved from http://www.sciencedirect.com/science/article/pii/S0045790617327490 doi: https://doi.org/10.1016/j.compeleceng.2018.11.004

Borgohain, T., Kumar, U., & Sanyal, S. (2015). Survey of security and privacy issues of internet of things. *CoRR*, *abs/1501.02211*. Retrieved from http://arxiv.org/abs/1501.02211

Botnet, C. (2013). Internet census 2012: Port scanning/0 using insecure embedded devices. *URL http://internetcensus2012. bitbucket. org/paper. html*.

Bouyeddou, B., Harrou, F., Sun, Y., & Kadri, B. (2018, May). Detection of smurf flooding attacks using kullback-leibler-based scheme. In *2018 4th international conference on computer and technology applications (iccta)* (p. 11-15). doi: 10.1109/CATA.2018.8398647

Brenner, B. (2010). *Layer 7 increasingly under ddos gun.* Retrieved from https://www.csoonline.com/article/2124801/report--layer-7-increasingly-under-ddos-gun.html

Buczak, A. L., & Guven, E. (2015). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials*, *18*(2), 1153–1176.

Campbell, D. T., & Stanley, J. C. (2015). *Experimental and quasi-experimental designs for research.* Ravenio Books.

Carl, G., Brooks, R. R., & Rai, S. (2006). Wavelet based denial-of-service detection. *Computers & Security*, *25*(8), 600 - 615. Retrieved from http://www.sciencedirect.com/science/article/pii/S0167404806001210 doi: https://doi.org/10.1016/j.cose.2006.08.017

Chadd, A. (2018). Ddos attacks: past, present and future. *Network Security*, *2018*(7), 13 - 15. Retrieved from http://www.sciencedirect.com/science/article/pii/S1353485818300692 doi: https://doi.org/10.1016/S1353-4858(18)30069-2

Charlie, M., & Valasek, C. (2014). *A survey of remote automotive attack surfaces.* Retrieved from http://www.ioactive.com/pdfs/IOActive_Remote_Attack_Surfaces.pdf

Chaturvedi, D. K. (2008). Introduction to soft computing. In *Soft computing: Techniques and its applications in electrical engineering* (pp. 1–10). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from https://doi.org/10.1007/978-3-540-77481-5_1 doi: 10.1007/978-3-540-77481-5_1

Chen, C. (2009). A new detection method for distributed denial-of-service attack traffic based on statistical test. *J. UCS*, *15*(2), 488–504. Retrieved from https://doi.org/10.3217/jucs-015-02-0488 doi: 10.3217/jucs-015-02-0488

Chen, E. Y. (2006). Detecting dos attacks on sip systems. In *1st ieee workshop on voip management and security, 2006.* (pp. 53–58). doi: 10.1109/VOIPMS.2006.1638123

Cohen, L., Manion, L., & Morrison, K. (2007). *Research methods in education*. routledge. doi: https://doi.org/10.4324/9780203224342

Cohn, R. J., Coppen, I. R. J., Banks, A., & Gupta, R. (2014). Mqtt version 3.1.

Collina, M., Corazza, G. E., & Vanelli-Coralli, A. (2012). Introducing the qest broker: Scaling the iot by bridging mqtt and rest. In *Personal indoor and mobile radio communications (pimrc), 2012 ieee 23rd international symposium on* (pp. 36–41). doi: 10.1109/PIMRC.2012.6362813

Creswell, J. W., & Clark, V. L. P. (2017). *Designing and conducting mixed methods research*. Sage publications.

Creswell, J. W., & Creswell, J. D. (2017). *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications.

Dainotti, A., Pescapé, A., & Ventre, G. (2006). Nis04-1: Wavelet-based detection of dos attacks. In *Ieee globecom 2006* (pp. 1–6). doi: 10.1109/GLOCOM.2006.279

Daswani, N., & Garcia-Molina, H. (2002). Query-flood dos attacks in gnutella. In *Proceedings of the 9th acm conference on computer and communications security* (pp. 181–192).

Diro, A. A., & Chilamkurti, N. (2018). Distributed attack detection scheme using deep learning approach for internet of things. *Future Generation Computer Systems*, *82*, 761 - 768. doi: https://doi.org/10.1016/j.future.2017.08.043

Douligeris, C., & Mitrokotsa, A. (2004). Ddos attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks*, *44*(5), 643–666. doi: https://doi.org/10.1016/j.comnet.2003.10.003

Dua, S., & Du, X. (2016). *Data mining and machine learning in cybersecurity*. Auerbach Publications.

Durcekova, V., Schwartz, L., & Shahmehri, N. (2012, May). Sophisticated denial of service attacks aimed at application layer. In *2012 elektro* (p. 55-60).

East, S., Butts, J., Papa, M., & Shenoi, S. (2009). A taxonomy of attacks on the dnp3 protocol. In *International conference on critical infrastructure protection* (pp. 67–81). doi: https://doi.org/10.1007/978-3-642-04798-5_5

Eclipse. (2018). *Eclipse paho.* `https://www.eclipse.org/paho/`.

eclipse.org. (2018). *Deploying the internet of things on germany's db railway system.* `https://iot.eclipse.org/resources/case-studies/Eclipse%20IoT%20Success%20Story%20-%20DB.pdf`.

Eddy, W. (2007). *Tcp syn flooding attacks and common mitigations* (Tech. Rep.). Retrieved from `https://www.rfc-editor.org/pdfrfc/rfc4987.txt.pdf`

Ehlert, S., Geneiatakis, D., & Magedanz, T. (2010). Survey of network security systems to counter sip-based denial-of-service attacks. *Computers & Security*, *29*(2), 225–243. doi: https://doi.org/10.1016/j.cose.2009.09.004

EMQ. (2018). *Emqx.* `https://www.emqx.io/`.

Eugster, P. T., Felber, P. A., Guerraoui, R., & Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM computing surveys (CSUR)*, *35*(2), 114–131. doi: 10.1145/857076.857078

Eurotech. (2019). *Iot edge framework.* `https://www.eurotech.com/en/products/iot/iot-edge-framework/everyware-software-framework`.

Fehrenbach, P. (2017). *Messaging queues in the iot under pressure.* Retrieved from `https://blog.it-securityguard.com/wp-content/uploads/2017/10/IOT_Mosquitto_Pfehrenbach.pdf`

Ferrari, P., Flammini, A., Sisinni, E., Rinaldi, S., Brandão, D., & Rocha, M. S. (2018). Delay estimation of industrial iot applications based on messaging protocols. *IEEE Transactions on Instrumentation and Measurement*, *67*(9), 2188–2199. doi: 10.1109/TIM.2018.2813798

Ferreira, H. G. C., Canedo, E. D., & de Sousa, R. T. (2013). Iot architecture to enable intercommunication through rest api and upnp using ip, zigbee and arduino. In *2013 ieee 9th international conference on wireless and mobile computing, networking and communications (wimob)* (pp. 53–60). doi: 10.1109/WiMOB.2013.6673340

Firdous, S. N., Baig, Z., Valli, C., & Ibrahim, A. (2017). Modelling and evaluation of malicious attacks against the iot mqtt protocol. In *2017 ieee international conference on internet of things (ithings) and ieee green computing and communications (greencom) and ieee cyber, physical and social computing (cpscom) and ieee smart data (smartdata)* (p. 748-755). doi: 10.1109/iThings-GreenCom-CPSCom-SmartData.2017.115

FPAnalyst. (2016). *Attack of things!* Retrieved from `https://www.flashpoint-intel.com/attack-of-things/`

Galliers, R. (1991, 01). Choosing appropriate information systems research approaches: A revised taxonomy. *The Information Research Arena of the 90s*, 155–173.

Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, *28*(1-2), 18–28. doi: https://doi.org/10.1016/j.cose.2008.08.003

Giles, M. (2019). *Triton is the world's most murderous malware, and it's spreading.* `https://www.technologyreview.com/s/613054/cybersecurity-critical-infrastructure-triton-malware/`.

Golab, L., DeHaan, D., Demaine, E. D., Lopez-Ortiz, A., & Munro, J. I. (2003). Identifying frequent items in sliding windows over on-line packet streams. In *Proceedings of the 3rd acm sigcomm conference on internet measurement* (pp. 173–178). doi: 10.1145/948205.948227

Guba, E. G., & Lincoln, Y. S. (1985). *Naturalistic inquiry.* Newbury Park, CA: Sage.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, *29*(7), 1645–1660. doi: https://doi.org/10.1016/j.future.2013.01.010

Gündoğan, C., Kietzmann, P., Schmidt, T. C., Lenders, M., Petersen, H., & Wählisch, M. (2018). Ndn, coap, and mqtt: A comparative measurement study in the iot. *arXiv preprint arXiv:1806.01444*.

Gupta, B., & Badve, O. P. (2017). Taxonomy of dos and ddos attacks and desirable defense mechanism in a cloud computing environment. *Neural Computing and Applications*, *28*(12), 3655–3682. doi: https://doi.org/10.1007/s00521-016-2317-5

Haddad Pajouh, H., Javidan, R., Khayami, R., Ali, D., & Choo, K. R. (2018). A two-layer dimension reduction and two-tier classification model for anomaly-based intrusion detection in iot backbone networks. *IEEE Transactions on Emerging Topics in Computing*, 1-1. doi: 10.1109/TETC.2016.2633228

HAproxy. (2018). *Haproxy.* `http://www.haproxy.org/`.

Hassija, V., Chamola, V., Saxena, V., Jain, D., Goyal, P., & Sikdar, B. (2019). A survey on iot security: Application areas, security threats, and solution architectures. *IEEE Access*, *7*, 82721–82743. doi: 10.1109/ACCESS.2019.2924045

Heer, T., Garcia-Morchon, O., Hummen, R., Keoh, S. L., Kumar, S. S., & Wehrle, K. (2011, Dec 01). Security challenges in the ip-based internet of things. *Wireless Personal Communications*, *61*(3), 527–542. doi: 10.1007/s11277-011-0385-5

HiveMQ. (2015). *Mqtt essentials: Part 1 – introducing mqtt.* Retrieved from `https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/`

Hodo, E., Bellekens, X., Hamilton, A., Tachtatzis, C., & Atkinson, R. (2017). Shallow and deep networks intrusion detection system: A taxonomy and survey. *arXiv preprint arXiv:1701.02145*.

Houimli, M., Kahloul, L., & Benaoun, S. (2017). Formal specification, verification and evaluation of the mqtt protocol in the internet of things. In *2017 international conference on mathematics and information technology (icmit)* (pp. 214–221).

Howard, M., & Lipner, S. (2006). *The security development lifecycle* (Vol. 8). Microsoft Press Redmond.

Hssina, B., Merbouha, A., Ezzikouri, H., & Erritali, M. (2014). A comparative study of decision tree id3 and c4. 5. *International Journal of Advanced Computer Science and Applications*, *4*(2), 13–19.

Hussain, S., Kamal, A., Ahmad, S., Rasool, G., & Iqbal, S. (2014). Threat modelling methodologies: a survey. *Sci. Int.(Lahore)*, *26*(4), 1607–1609.

iebmedia. (2019). *Iiot: Combining the best of ot and it.* Retrieved from `https://iebmedia.com/index.php?id=11673&parentid=63&themeid=255&hft=95&showdetail=true&bb=1`

Jensen, M., Gruschka, N., & Herkenhöner, R. (2009). A survey of attacks on web services. *Computer Science-Research and Development*, *24*(4), 185. doi: https://doi.org/10.1007/s00450-009-0092-6

Jeyanthi, N. (2016). Internet of things (iot) as interconnection of threats (iot). In *Security and privacy in internet of things (iots)* (pp. 3–21). CRC Press. Retrieved from `http://dx.doi.org/10.1201/b19516-3` doi: doi: 10.1201/b19516-3

Karagiannis, V., Chatzimisios, P., Vazquez-Gallego, F., & Alonso-Zarate, J. (2015). A survey on application layer protocols for the internet of things..

Karlik, B., & Olgac, A. V. (2011). Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, *1*(4), 111–122.

Kasinathan, P., Pastrone, C., Spirito, M. A., & Vinkovits, M. (2013, Oct). Denial-of-service detection in 6lowpan based internet of things. In *2013 ieee 9th international conference on wireless and mobile computing, networking and communications (wimob)* (p. 600-607). doi: 10.1109/WiMOB.2013.6673419

Kaur, P., Kumar, M., & Bhandari, A. (2017). A review of detection approaches for distributed denial of service attacks. *Systems Science & Control Engineering*, *5*(1), 301-320. Retrieved from `https://doi.org/10.1080/21642583.2017.1331768` doi: 10.1080/21642583.2017.1331768

Kenney, M. (1996). *Ping of death.* Retrieved from `https://insecure.org/sploits/ping-o-death.html`

Khan, S., & Traore, I. (2005). Queue-based analysis of dos attacks. In *Proceedings from the sixth annual ieee smc information assurance workshop* (pp. 266–273). doi: 10.1109/IAW.2005.1495962

Kivunja, C., & Kuyini, A. B. (2017). Understanding and applying research paradigms in educational contexts. *International Journal of Higher Education*, *6*(5), 26–41.

Klingel, D., Khondoker, R., Marx, R., & Bayarou, K. (2014). Security analysis of software defined networking architectures: Pce, 4d and sane. In *Proceedings of the aintec 2014 on asian internet engineering conference* (pp. 15–22).

Koc, L., & Carswell, A. D. (2015). Application of an aode based classifier to detect dos attacks. *International Journal of Computer Science and Network Security (IJCSNS)*, *15*(2), 24.

Kolias, C., Kambourakis, G., Stavrou, A., & Voas, J. (2017). Ddos in the iot: Mirai and other botnets. *Computer*, *50*(7), 80–84. doi: 10.1109/MC.2017.201

Koroniotis, N., Moustafa, N., Sitnikova, E., & Turnbull, B. (2019). Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems*, *100*, 779–796.

Krebs, B. (2016a). *Iot devices as proxies for cybercrime.* Retrieved from `https://krebsonsecurity.com/2016/10/iot-devices-as-proxies-for-cybercrime/`

Krebs, B. (2016b). *Krebsonsecurity hit with record ddos.* Retrieved from `https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/`

Krishnamurthy, B., Sen, S., Zhang, Y., & Chen, Y. (2003). Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd acm sigcomm conference on internet measurement* (pp. 234–247). doi: https://doi.org/10.1145/948205.948236

Langner, R. (2011). Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy*, *9*(3), 49–51. doi: 10.1109/MSP.2011.67

Lau, F., Rubin, S. H., Smith, M. H., & Trajkovic, L. (2000). Distributed denial of service attacks. In *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0* (Vol. 3, pp. 2275–2280).

Lee, S., Kim, H., k. Hong, D., & Ju, H. (2013, Jan). Correlation analysis of mqtt loss and delay according to qos level. In *The international conference on information networking 2013 (icoin)* (p. 714-717). doi: 10.1109/ICOIN.2013.6496715

Li, S., Da Xu, L., & Zhao, S. (2015). The internet of things: a survey. *Information Systems Frontiers*, *17*(2), 243–259.

Libelium. (2019). *50 sensor applications for a smarter world.* libelium. Retrieved from `http://www.libelium.com/resources/top_50_iot_sensor_applications_ranking/`

Linda, O., Vollmer, T., & Manic, M. (2009). Neural network based intrusion detection system for critical infrastructures. In *2009 international joint conference on neural networks* (pp. 1827–1834). doi: 10.1109/IJCNN.2009.5178592

Little, J. D., & Graves, S. C. (2008). Little's law. In *Building intuition* (pp. 81–100). Springer.

Locke, D. (2010). Mq telemetry transport (mqtt) v3. 1 protocol specification. *IBM developerWorks Technical Library*, 15.

Lu, W., & Ghorbani, A. A. (2009). Network anomaly detection based on wavelet analysis. *EURASIP Journal on Advances in Signal Processing*, *2009*, 4. doi: https://doi.org/10.1155/2009/837601

Luo, M., Peng, T., & Leckie, C. (2008). Cpu-based dos attacks against sip servers. In *Noms 2008 - 2008 ieee network operations and management symposium* (p. 41-48). doi: 10.1109/NOMS.2008.4575115

Luzuriaga, J. E., Perez, M., Boronat, P., Cano, J. C., Calafate, C., & Manzoni, P. (2015). A comparative evaluation of amqp and mqtt protocols over unstable and mobile networks. In *2015 12th annual ieee consumer communications and networking conference (ccnc)* (pp. 931–936). doi: 10.1109/CCNC.2015.7158101

Mackenzie, N., & Knipe, S. (2006). Research dilemmas: Paradigms, methods and methodology. *Issues in educational research*, *16*(2), 193–205.

Malik, M. I., McAteer, I. N., Hannay, P., Firdous, S. N., & Baig, Z. (2018). Xmpp architecture and security challenges in an iot ecosystem. In *Proceedings of the 16th australian information security management conference* (p. 62). doi: 10.25958/5c52735166690

Mansfield-Devine, S. (2015). The growth and evolution of ddos. *Network Security*, *2015*(10), 13 - 20. Retrieved from `http://www.sciencedirect.com/science/article/pii/S1353485815300921` doi: https://doi.org/10.1016/S1353-4858(15)30092-1

Mansfield-Devine, S. (2016). Ddos goes mainstream: how headline-grabbing attacks could make this threat an organisation's biggest nightmare. *Network Security*, *2016*(11), 7 - 13. Retrieved from `http://www.sciencedirect.com/science/article/pii/S1353485816301040` doi: https://doi.org/10.1016/S1353-4858(16)30104-0

Mantas, G., Stakhanova, N., Gonzalez, H., Jazi, H. H., & Ghorbani, A. A. (2015). Application-layer denial of service attacks: taxonomy and survey. *International Journal of Information and Computer Security*, *7*(2-4), 216–239.

Maresca, P. (2017, July 6). *Detection, prevention, and/or mitigation of dos attacks in publish/subscribe infrastructure.* Google Patents. (US Patent App. 14/984,016)

Masse, M. (2011). *Rest api design rulebook: Designing consistent restful web service interfaces.* " O'Reilly Media, Inc.".

Mektoubi, A., Hassani, H. L., Belhadaoui, H., Rifi, M., & Zakari, A. (2016). New approach for securing communication over mqtt protocol a comparaison between rsa and elliptic curve. In *2016 third international conference on systems of collaboration (sysco)* (pp. 1–6). doi: 10.1109/SYSCO.2016.7831326

Meng, Z., Wu, Z., Muvianto, C., & Gray, J. (2017, Feb). A data-oriented m2m messaging mechanism for industrial iot applications. *IEEE Internet of Things Journal*, *4*(1), 236-246. doi: 10.1109/JIOT.2016.2646375

Metongnon, L., & Sadre, R. (2018). Beyond telnet: Prevalence of iot protocols in telescope and honeypot measurements. In *Proceedings of the*

2018 workshop on traffic measurements for cybersecurity (pp. 21–26). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/3229598.3229604 doi: 10.1145/3229598.3229604

Miller, M. (2015). *The internet of things: How smart tvs, smart cars, smart homes, and smart cities are changing the world*. Pearson Education.

Minerva, R., Biru, A., & Rotondi, D. (2015). Towards a definition of the internet of things (iot). *IEEE Internet Initiative*(1).

Mirkovic, J., & Reiher, P. (2004). A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, *34*(2), 39–53.

MIT, L. L. (1999). *1999 darpa intrusion detection evaluation dataset.* https://www.ll.mit.edu/r-d/datasets/1999-darpa-intrusion-detection-evaluation-dataset.

Mitchell, R., & Chen, I.-R. (2014). A survey of intrusion detection techniques for cyber-physical systems. *ACM Computing Surveys (CSUR)*, *46*(4), 55.

Modi, C., Patel, D., Borisaniya, B., Patel, H., Patel, A., & Rajarajan, M. (2013). A survey of intrusion detection techniques in cloud. *Journal of network and computer applications*, *36*(1), 42–57.

Moore, H. (2013). Security flaws in universal plug and play: Unplug. don't play. *Rapid7, Ltd*, *8*. Retrieved from https://community.rapid7.com/docs/DOC-2150

Mosquitto. (2017). *Eclipse mosquitto.* Retrieved from https://mosquitto.org/

Moustafa, N., & Slay, J. (2015, Nov). Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (milcis)* (p. 1-6). doi: 10.1109/MilCIS.2015.7348942

Moustafa, N., Turnbull, B., & Choo, K. R. (2019). An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things. *IEEE Internet of Things Journal*, 1-1. doi: 10.1109/JIOT.2018.2871719

Murray, G., Johnstone, M. N., & Valli, C. (2017). The convergence of it and ot in critical infrastructure.

Neshenko, N., Bou-Harb, E., Crichigno, J., Kaddoum, G., & Ghani, N. (2019). Demystifying iot security: an exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations. *IEEE Communications Surveys & Tutorials*.

Network, A. (2015). *Worldwide infrastructure security report.*

Niruntasukrat, A., Issariyapat, C., Pongpaibool, P., Meesublak, K., Aiumsupucgul, P., & Panya, A. (2016). Authorization mechanism for mqtt-based internet of things. In *Communications workshops (icc), 2016 ieee international conference on* (pp. 290–295). doi: 10.1109/ICCW.2016.7503802

Olenick, D. (2017). *Brickerbot malware attacks and destroys unsecure iot devices.* `https://www.scmagazine.com/home/security-news/iot/brickerbot-malware-attacks-and-destroys-unsecure-iot-devices/`.

Olifer, N., & Olifer, V. (2005). *Computer networks: Principles, technologies and protocols for network design.* Wiley Publishing.

Oracle. (2018). *Oracle virtual box.* `https://www.virtualbox.org/`.

OWASP. (2018). Application threat modeling. Retrieved from `https://www.owasp.org/index.php/Application_Threat_Modeling`

Pa, Y. M. P., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T., & Rossow, C. (2015). Iotpot: analysing the rise of iot compromises. *EMU*, *9*, 1.

Panchal, A. C., Khadse, V. M., & Mahalle, P. N. (2018). Security issues in iiot: A comprehensive survey of attacks on iiot and its countermeasures. In *2018 ieee global conference on wireless computing and networking (gcwcn)* (pp. 124–130).

Pandas. (2018). *Pandas.* `https://pandas.pydata.org/`.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Peng, T., Leckie, C., & Ramamohanarao, K. (2007, April). Survey of network-based defense mechanisms countering the dos and ddos problems. *ACM Comput. Surv.*, *39*(1). Retrieved from `http://doi.acm.org/10.1145/1216370.1216373` doi: 10.1145/1216370.1216373

Peraković, D., Periša, M., & Cvitić, I. (2015). Analysis of the iot impact on volume of ddos attacks. In *33rd symposium on new technologies in postal and telecommunication traffic (postel 2015)* (pp. 295–304).

Perera, C., Zaslavsky, A., Christen, P., & Georgakopoulos, D. (2014). Context aware computing for the internet of things: A survey. *IEEE Communications Surveys & Tutorials*, *16*(1), 414–454.

Perrone, G., Vecchio, M., Pecori, R., & Giaffreda, R. (2017). The day after mirai: A survey on mqtt security solutions after the largest cyber-attack carried out through an army of iot devices. In *Iotbds* (pp. 246–253).

Potteiger, B., Martins, G., & Koutsoukos, X. (2016). Software and attack centric integrated threat modeling for quantitative risk assessment. In *Proceedings of the symposium and bootcamp on the science of security* (pp. 99–108).

Poulsen, K. (2003). Slammer worm crashed ohio nuke plant network. *Security Focus*, *19*.

Praseed, A., & Thilagam, P. S. (2018). Ddos attacks at the application layer: Challenges and research perspectives for safeguarding web applications. *IEEE Communications Surveys & Tutorials*, *21*(1), 661–685. doi: 10.1109/COMST.2018.2870658

Rafique, M. Z., Akbar, M. A., & Farooq, M. (2009). Evaluating dos attacks against sip-based voip systems. In *Globecom 2009 - 2009 ieee global telecommunications conference* (p. 1-6). doi: 10.1109/GLOCOM.2009.5426247

Ramanauskaite, S., & Cenys, A. (2011). Taxonomy of dos attacks and their countermeasures. *Open Computer Science*, *1*(3), 355–366.

Ranjan, S., Swaminathan, R., Uysal, M., Nucci, A., & Knightly, E. (2008). Ddos-shield: Ddos-resilient scheduling to counter application layer attacks. *IEEE/ACM Transactions on networking*, *17*(1), 26–39. doi: 10.1109/TNET.2008.926503

Ranjan, S., Swaminathan, R., Uysal, M., Nucci, A., & Knightly, E. (2009, Feb). Ddos-shield: Ddos-resilient scheduling to counter application layer attacks. *IEEE/ACM Transactions on Networking*, *17*(1), 26-39. doi: 10 .1109/TNET.2008.926503

Ray, P. (2018). A survey on internet of things architectures. *Journal of King Saud University - Computer and Information Sciences*, *30*(3), 291 - 319. Retrieved from `http://www.sciencedirect.com/science/article/pii/S1319157816300799` doi: https://doi.org/10.1016/j.jksuci.2016.10 .003

Rieck, K. (2009). Machine learning for application-layer intrusion detection.

doi: http://dx.doi.org/10.14279/depositonce-2199

Roman, R., Zhou, J., & Lopez, J. (2013). On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, *57*(10), 2266–2279.

Rose, K., Eldridge, S., & Chapin, L. (2015). The internet of things: An overview. *The Internet Society (ISOC)*, 1–50.

Sadeghi, A.-R., Wachsmann, C., & Waidner, M. (2015). Security and privacy challenges in industrial internet of things. In *Design automation conference (dac), 2015 52nd acm/edac/ieee* (pp. 1–6). doi: 10.1145/ 2744769.2747942

Saidu, C. I., Usman, A. S., & Ogedebe, P. (2015). Internet of things: Impact on economy. *British Journal of Mathematics & Computer Science*, *7*(4), 241. doi: 10.9734/BJMCS/2015/14742

Saitta, P., Larcom, B., & Eddington, M. (2005). Trike v. 1 methodology document [draft]. *URL: http://dymaxion. org/trike/Trike v1 Methodology Documentdraft. pdf*.

Salman, T., & Jain, R. (2015). Networking protocols and standards for internet of things. *Internet of Things and Data Analytics Handbook (2015)*, *7*.

Sanfilippo, S. (2006). *hping.* `http://www.hping.org/`.

Santiago Hernández Ramos, M. T. V., & Lacuesta, R. (2018). Mqtt security: A novel fuzzing approach. *Wireless Communications and Mobile Computing*, *2018*. doi: https://doi.org/10.1155/2018/8261746

Scalagent. (2015). *Benchmark of mqtt servers.* Retrieved from http://www.scalagent.com/IMG/pdf/Benchmark_MQTT_servers-v1-1.pdf

SecurityCompass. (2016). *Publish-subscribe threat modeling.* Retrieved from https://blog.securitycompass.com/publish-subscribe-threat-modeling-11add54f1d07#.w5x9zfbr7

Serbanati, A., Medaglia, C. M., & Ceipidor, U. B. (2011). *Building blocks of the internet of things: State of the art and beyond.* INTECH Open Access Publisher. doi: 10.5772/19997

Shaikh, F. K., Zeadally, S., & Exposito, E. (2017, June). Enabling technologies for green internet of things. *IEEE Systems Journal*, *11*(2), 983-994. doi: 10.1109/JSYST.2015.2415194

Shaker, V., & Zarrabi, H. (2017). Dos attacks in internet of things. *International Journal of Advanced Research in Computer Science*, *8*(5).

Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms.* Cambridge university press.

Shan, H., Wang, Q., & Pu, C. (2017a). Tail attacks on web applications. In *Proceedings of the 2017 acm sigsac conference on computer and communications security* (pp. 1725–1739). ACM. Retrieved from http://doi.acm.org/10.1145/3133956.3133968 doi: 10.1145/3133956.3133968

Shan, H., Wang, Q., & Pu, C. (2017b). Tail attacks on web applications. In *Proceedings of the 2017 acm sigsac conference on computer and communications security* (pp. 1725–1739).

Shang, W., Yu, Y., Droms, R., & Zhang, L. (2016). Challenges in iot networking via tcp/ip architecture. *Technical Report NDN-0038. NDN Project*.

Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Icissp* (pp. 108–116).

Shin, S., Kobara, K., Chuang, C.-C., & Huang, W. (2016). A security framework for mqtt. In *2016 ieee conference on communications and network security (cns)* (pp. 432–436).

Singh, K., Singh, P., & Kumar, K. (2017). Application layer http-get flood ddos attacks: Research landscape and challenges. *Computers and Security*, *65*, 344 - 372. doi: https://doi.org/10.1016/j.cose.2016.10.005

Singh, M., Rajan, M., Shivraj, V., & Balamuralidhar, P. (2015). Secure mqtt for internet of things (iot). In *2015 fifth international conference on communication systems and network technologies* (pp. 746–751).

Sivanathan, A., Habibi Gharakheili, H., Loi, F., Radford, A., Wijenayake, C., Vishwanath, A., & Sivaraman, V. (2018). Classifying iot devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing*, 1-1. doi: 10.1109/TMC.2018.2866249

Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., & Stiller, B. (2010, Third). An overview of ip flow-based intrusion detection. *IEEE Communications Surveys Tutorials*, *12*(3), 343-356. doi: 10.1109/SURV.2010.032210.00054

Srivatsa, M., & Liu, L. (2005). Securing publish-subscribe overlay services with eventguard. In *Proceedings of the 12th acm conference on computer and communications security* (pp. 289–298).

Statista. (2016). *Internet of things (iot) connected devices installed base worldwide from 2015 to 2025 (in billions).* Retrieved from `https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/`

Sun, L., & Wang, W. (2012). On latency distribution and scaling: From finite to large cognitive radio networks under general mobility. In *2012 proceedings ieee infocom* (pp. 1287–1295).

Tama, B. A., & Rhee, K.-H. (2015). Data mining techniques in dos/ddos attack detection: A literature review. *Information (Japan)*, *18*(8), 3739.

TCPDUMP. (2019). *Tcpdump.* `https://www.tcpdump.org/`.

Thangavel, D., Ma, X., Valera, A., Tan, H.-X., & Tan, C. K.-Y. (2014). Performance evaluation of mqtt and coap via a common middleware. In *2014 ieee ninth international conference on intelligent sensors, sensor networks and information processing (issnip)* (pp. 1–6). doi: 10.1109/ISSNIP.2014.6827678

Tschofenig, H., Arkko, J., Thaler, D., & McPherson, D. (2015). *Architectural considerations in smart object networking, rfc 7452* (Tech. Rep.).

UCI. (1999). *Kdd cup 1999 data.* `https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`.

UCSD, C. (2007). *The caida ucsd "ddos attack 2007" dataset.* https://www.caida.org/data/passive/ddos-20070804_dataset.xml.

UPnP. (2014). *Upnp iot overview.* Retrieved from http://upnp.org/resources/documents/UPnP_IoT_Overview_Dec2014.pdf

Vaux, D. L., Fidler, F., & Cumming, G. (2012). Replicates and repeats—what is the difference and is it significant? *EMBO reports*, *13*(4), 291–296.

Vermesan, O., & Friess, P. (2014). *Internet of things-from research and innovation to market deployment.* River Publishers Aalborg.

VerneMQ. (2018). *Vernemq.* https://vernemq.com/.

Vähä-Sipilä, A. (2015). *mqtt_fuzz.* https://github.com/F-Secure/mqtt_fuzz.

Voas, J. (2016, June). Demystifying the internet of things. *Computer*, *49*(6), 80-83. doi: 10.1109/MC.2016.162

Waher, P. (2015). *Learning internet of things.* Packt Publishing Ltd.

Waher, P. (2016). *Communication patterns for the internet of things.* Retrieved from https://software.intel.com/en-us/articles/communication-patterns-for-the-internet-of-things

Wang, H., Zhang, D., & Shin, K. G. (2002). Detecting syn flooding attacks. In *Proceedings. twenty-first annual joint conference of the ieee computer and communications societies* (Vol. 3, pp. 1530–1539). doi: 10.1109/INFCOM.2002.1019404

Webb, G. I., Boughton, J. R., & Wang, Z. (2005, Jan 01). Not so naive bayes: Aggregating one-dependence estimators. *Machine Learning*, *58*(1), 5–24. doi: 10.1007/s10994-005-4258-6

Williamson, K. (2018). Chapter 1 - research concepts. In K. Williamson & G. Johanson (Eds.), *Research methods (second edition)* (Second Edition ed., p. 3 - 25). Chandos Publishing. doi: https://doi.org/10.1016/B978-0-08-102220-7.00001-7

Wireshark. (2019). *Tshark.* https://www.wireshark.org/docs/man-pages/tshark.html.

Wu, H. (2017). *With industry 4.0 in mind, nexcom implements a smart gateway iot solution.* `https://microsoft.github.io/techcasestudies/iot/2016/12/06/nexcom.html`.

Wun, A., Cheung, A., & Jacobsen, H.-A. (2007). A taxonomy for denial of service attacks in content-based publish/subscribe systems. In *Proceedings of the 2007 inaugural international conference on distributed event-based systems* (pp. 116–127). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1266894.1266917` doi: 10.1145/1266894.1266917

Xiao, J., Yun, X.-C., & Zhang, Y.-Z. (2010). Defend against application-layer distributed denial-of-service attacks based on session suspicion probability model. *Jisuanji Xuebao(Chinese Journal of Computers)*, *33*(9), 1713–1724.

Xie, Y., & Yu, S.-Z. (2009). Monitoring the application-layer ddos attacks for popular websites. *IEEE/ACM Transactions on Networking (TON)*, *17*(1), 15–25.

Yan, Q., Yu, F. R., Gong, Q., & Li, J. (2015). Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges. *IEEE communications surveys & tutorials*, *18*(1), 602–622.

Yatagai, T., Isohara, T., & Sasase, I. (2007). Detection of http-get flood attack based on analysis of page access behavior. In *2007 ieee pacific rim conference on communications, computers and signal processing* (pp. 232–235).

Yokotani, T., & Sasaki, Y. (2016, Dec). Transfer protocols of tiny data blocks in iot and their performance evaluation. In *2016 ieee 3rd world forum on internet of things (wf-iot)* (p. 54-57). doi: 10.1109/WF-IoT.2016.7845442

Zaki, M. J., Meira Jr, W., & Meira, W. (2014). *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press.

Zanella, A., Bui, N., Castellani, A., Vangelista, L., & Zorzi, M. (2014). Internet of things for smart cities. *IEEE Internet of Things journal*, *1*(1), 22–32. doi: 10.1109/JIOT.2014.2306328

Zargar, S. T., Joshi, J., & Tipper, D. (2013). A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE communications surveys & tutorials*, *15*(4), 2046–2069.

Zarpelão, B. B., Miani, R. S., Kawakani, C. T., & de Alvarenga, S. C. (2017). A survey of intrusion detection in internet of things. *Journal of Network and Computer Applications*, *84*, 25 - 37. Retrieved from `http://www.sciencedirect.com/science/article/pii/S1084804517300802` doi: https://doi.org/10.1016/j.jnca.2017.02.009

Zhang, G., Jiang, S., Wei, G., & Guan, Q. (2009). A prediction-based detection algorithm against distributed denial-of-service attacks. In *Proceedings of the 2009 international conference on wireless communications and mobile computing: Connecting the world wirelessly* (pp. 106–110). doi: https://doi.org/10.1145/1582379.1582403

Zhang, Z., Cho, M. C. Y., Wang, C.-W., Hsu, C.-W., Chen, C.-K., & Shieh, S. (2014). Iot security: ongoing challenges and research opportunities. In *2014 ieee 7th international conference on service-oriented computing and applications* (pp. 230–234).

Zheng, A., & Casari, A. (2018). *Feature engineering for machine learning: Principles and techniques for data scientists.* " O'Reilly Media, Inc.".

Zhu, B., & Sastry, S. (2010). Scada-specific intrusion detection/prevention systems: a survey and taxonomy. In *Proceedings of the 1st workshop on secure control systems (scs)* (Vol. 11, p. 7).

Zlomislić, V., Fertalj, K., & Sruk, V. (2017, Mar 01). Denial of service attacks, defences and research challenges. *Cluster Computing*, *20*(1), 661–671. Retrieved from `https://doi.org/10.1007/s10586-017-0730-x` doi: 10.1007/s10586-017-0730-x

Appendix A

Additional Results

A.1 Additional Results for BF1 Attack Impact on Three Brokers

Table A.1: Average CPU idle percentage for BF1 attack on the Mosquitto Broker

Sleep Interval (seconds)	Single Thread	Two threads	Three threads	Four threads	Five threads
0.1	99.58	99.36	99.20	99.02	98.37
0.5	98.96	98.02	96.99	95.94	94.80
0.01	88.09	80.86	75.88	67.17	58.54
0.005	62.02	56.81	50.66	43.69	38.96
0	18.25	19.84	20.58	20.58	21.48

Table A.2: Average CPU idle percentage for BF1 attack on the VerneMQ Broker

Sleep-Interval (seconds)	Single Thread	Two Threads	Three Threads	Four Threads	Five Threads
0.1	99.65	97.64	97.21	97.29	96.82
0.5	99.24	94.66	92.32	92.33	89.92
0.01	96.91	73.99	63.34	58.76	43.01
0.005	95.81	61.05	42.08	25.64	13.79
0	82.74	69.61	18.75	5.11	0.73

Table A.3: Average CPU idle percentage for BF1 attack on the EMQ Broker

Sleep-Interval (seconds)	Single Thread	Two Threads	Three Threads	Four Threads	Five Threads
0.1	97.63	93.09	92.50	89.73	88.16
0.5	87.26	68.77	59.00	37.42	33.57
0.01	1.90	1.20	1.52	0.59	0.54
0.005	0.00	0.00	0.00	0.00	0.00
0	0.00	0.00	0.00	0.00	0.00