



**QUEEN'S
UNIVERSITY
BELFAST**

DIPA : A Distributed Intrusion Prevention Architecture against edge-based network attacks

MEng ELE-4001 Final Year Report

Name : Ben Kelly

Student Number : 40133635

Supervisor = Dr. Sandra Scott-Hayward

Moderator = Dr Kieran Mc Laughlin

April 29th , 2019

Abstract

With the rapid growth in technology, Computer networks are now becoming the backbone in a cyber-physical era. Industry has seen the introduction of interconnected IP enabled devices. This however has opened an attractive gateway for powerful distributed denial of service attacks. The power of these attacks has stemmed from the collusive behaviour and vast resources available. Current stand-alone intrusion detection and protection systems (IDPS) are ineffective with weak defence strategies to counteract such attacks.

This project leverages new emerging technologies such as SDN, NFV and Apache's pub sub; Pulsar, to enable collaborative security for IP-based IoT devices. This paper analyses current collaborative intrusion detection and prevention systems (CIDPS), researching strong methodologies and areas requiring improvement. A collaborative solution is designed and exposed to various testing conditions with all design decisions and development stages critically analysed. Performance, security features and limitations of the proposed solution are analysed providing recommendations for an edge based and system protection. Conclusions are drawn on the contribution to the Security field whilst recognising pulsar as a viable collaborative scheme to achieve global synthesis.

Specification

Edge-based Network Attack Protection using Apache Pulsar

Supervisor: Dr. Sandra Scott-Hayward

Moderator: Dr. Kieran McLaughlin

	Control		Embedded Systems		High Frequency Electronics		Microelectronics
	Electric Power	X	Software		Connected Health		MEMS
X	Cyber-Security		Wireless Communications		Signal/Image Processing		Intelligent Systems
	Digital Design		Sensor Networks		Data Analytics		Electronics

The increasing volume of deployed IoT devices and the potential for their misuse in network-based attacks has led to severe network interruptions in recent years (e.g. Dyn DDoS attack). The argument against these devices connecting directly to the Internet is clear. However, in current deployments, the majority of devices are IP enabled.

With emerging technologies such as Software Defined Networking and Network Functions Virtualization introducing logically centralized control, programmability etc., the opportunity arises to more efficiently detect and protect against network attacks. For example, service providers and network operators could contribute knowledge from their local networks for global synthesis across multiple autonomous systems. Network security related decisions could then be generated from this collaborative knowledge system and consumed locally for edge-based malicious traffic management.

The aim of this project is to explore the potential to deliver edge-based network attack detection and protection based on a distributed pub-sub messaging system. The proposed pub-sub messaging system is Apache Pulsar (<https://pulsar.incubator.apache.org/>).

Objectives

1. Study state-of-the-art in collaborative Intrusion Detection and Protection Systems (IDPS).
2. Become familiar with the Apache Pulsar distributed pub-sub messaging system.
3. Set up a testbed representing multiple SDN-controlled autonomous systems linked to the Pulsar system. The testbed can be a virtual implementation.
4. Analyse the security features of the Pulsar system and their suitability for secure information exchange between multiple vendors.
5. Develop and implement an edge-based network security application in the configured Pulsar testbed.
6. Evaluate the application performance and the security of the framework (e.g. vulnerability to authorized message senders that behave maliciously) and the suitability of the framework under load conditions (e.g. how does the pub/sub solution respond to the load surge in the case of a DoS attack?).
7. Provide recommendations for intelligent edge-based network protection.

MEng Extension

This is a MEng project and requires network security expertise, strong programming and networking skills.

Learning Outcomes

Upon completion of the project, you will expect to have:

1. A comprehensive understanding of SDN and distributed messaging systems.
2. A thorough knowledge of network security related threats and collaborative IDPS.
3. Developed practical skills in network testbed and SDN application development and testing.

Acknowledgments

I would like to acknowledge and express my gratitude to my supervisor Dr. Sandra Scott Hayward, for her continual support throughout the duration of this project. Project Objectives were both challenging and stimulating. Furthering my knowledge, in many new security applications and concepts. I feel this will be invaluable in aiding me in my future career as a security engineer. Continual assistance throughout the project has provided the necessary guidance to meet the project objectives and allow reflective learning to develop and enhance my skills and knowledge.

I am also greatly indebted to the Apache Pulsar's support team and community for their support, competence and experience. Particularly, developer Matteo Marli, Apache Pulsar's support developer. Matteo's suggestions and explanations throughout the process reinforced the foundations of my project. Within the community, I would like to thank Ali Ahmed for his contribution and experience in cloud deployment and system administration to help distribute the pulsar framework into multiple geo-replicated clusters.

Finally, I would like to thank the staff of Queens University Belfast. In particular, The School of Electronics, Electrical Engineering and Computer Science. This experience would not have been possible without the continual guidance and experience gathered over these five invaluable years.

Declaration of Academic Originality

I declare that I am the sole author of this written work, except where explicitly stated otherwise.

All resources used in this project are referenced in the bibliography

Signed:

Date:

Table of Contents

Abstract.....	i
Specification.....	ii
Objectives	iii
MEng Extension	iii
Learning Outcomes.....	iii
Acknowledgments.....	iv
Declaration of Academic Originality	v
Table of Contents.....	vi
1 Introduction	1
1.1 Relevance of the study.....	3
1.2 Research Question	3
1.3 Beneficiaries of this Research	4
1.4 Project Goals and Desired Outcomes	4
1.5 Project Scope	5
2 Background Research.....	6
2.1 Understanding the Mirai Botnet.....	6
2.2 Taxonomy of Collaborative Intrusion Detection and Prevention	7
2.2.1 Architectures.....	7
2.2.2 Requirements for CIPDS.....	9
2.3 Existing Solutions	9
2.3.1 Related work comparison	16
3 Technologies	17
3.1 Software Defined Networking	17
3.1.1 Traditional Networking vs the SDN paradigm.....	17
3.1.2 Ryu Controller	19
3.1.3 SDN and IDPS	20
3.1.4 NFV and IDPS.....	21
3.1.5 SDN Project Relevance.....	21
3.2 Apache Pulsar.....	22
3.2.1 Introduction to Apache Pulsar	22
3.2.2 Messaging Concepts	22
3.2.3 Pulsar architecture	24
3.2.4 Project Relevance.....	25

3.3 Topology Virtualization	26
3.3.1 Virtual Box.....	26
3.3.2 Mininet.....	26
4 System Architecture.....	27
4.1 Virtual Components	28
4.1.1 Control Plane Architecture	28
4.1.2 Virtual Machine Connection	29
4.1.3 Timing challenges.....	30
4.2 Pulsar architecture.....	31
4.2.1 Pulsar Architecture	31
4.2.2 Design Decisions and Challenges	32
5 System Realisation / Implementation.....	33
5.1 Logical View	33
5.2 Coarse Grain NIDS.....	34
5.2.1 Detection Functionality.....	34
5.2.2 Timing Algorithm.....	39
5.2 Pulsar Features.....	40
5.2.1 Data Privacy and Membership management	40
5.2.2 Geo-Replication.....	41
5.2.3 Async Communications.....	42
5.2.4 Message Deduplication.....	43
5.2.5 Message retention and topic compaction	44
6 Experimental Testing	45
6.1 Testbed Setup	45
6.1.1 Virtual Machine Testbed.....	45
6.1.2 Pulsar framework.....	46
6.1.3 Legitimate Benign Traffic	46
6.1.4 Attack Traffic + tools	47
6.1.5 Monitoring Tools.....	47
6.2 Functional Test Set.....	48
6.2.1 Reactive Collaborative protection	48
6.2.2 Network Intrusion Detection	49
6.2.4 Optimal Polling rate and Timeouts	51
6.2.5 Pulsar Security Analysis.....	53
6.3 Performance Test Set.....	56
6.3.1 Standalone IDPS protection time vs Pulsar CIDPS Protection time	57

6.3.2 Control + Data plane Vs Produce + Consume Time	60
6.3.3 System Accuracy	62
6.4 Results Analysis.....	63
6.4.1 Comparisons to Existing Solutions	64
6.4.2 Areas of Uncertainty	64
7 Discussion.....	65
7.1 System Limitation.....	65
7.2 System Exploits	67
7.3 Project Plannning	67
7.3.1 Risk mitigation.....	67
7.3.2 Project Approach and design stages	68
7.4 Future Work	69
7.4.1 Live Test Conditions	69
7.4.2 Pulsar IO Connectors.....	70
7.4.3 Pulsar statsRequest.....	71
7.5 Pulsar Cost and resource usage	71
7.6 Project Objectives Evaluation	71
7 Conclusions	73
8 References	74
Appendices.....	77i
Appendix 1 : Gantt Chart	77i
Annex 1.1 Risk Mitigation :	78
Appendix 2 : Github + User documentation	79
Appendix 3 : AWS Deployment.....	80
Annex 3.1 : Aws Billing Forecast	80
Annex 3.2 : AWS Deployment instances.....	80
Appendix 4 : Software Listings	81
Appendix 5: Mirai Botnet.....	81
Annex 5.1 Vulnerable Device list	81
Annex 5.2 Geo Locations of Mirai Botnet	82
Annex 5.3 Dictionary Content used by the Mirai malware.....	82
Annex 5.4 61 passwords that powered the mirai IoT botnet	83
Appendix 6 : Complete test set.....	83
Annex 6.1 Control Plan Time.....	83
Annex 6.2 Pulsar Publish Time	84
Annex 6.3 Vm protection time.....	86

Table of Figures

Figure 1 Power of DDoS attacks within the last decade	1
Figure 2 Number of malicious Devices found in 3 years.....	2
Figure 3 Overview of Mirai Botnet Malware spread [50]	7
Figure 4 Overview of CIPDS architecture ,Blue = Monitor unit ,Red = Correlation Unit, Orange =Monitor + Correlation Unit [9].....	8
Figure 5 SDN vs Traditional networking.....	17
Figure 6 SDN architecture with OpenFlow protocol.....	18
Figure 7 OpenFlow Table Entries [51].....	19
Figure 8 OSI layers , Stateful vs Stateless.....	21
Figure 9 Producer , Consumer and Topics (messaging concepts).....	23
Figure 10 Pulsar Cluster Architecture	24
Figure 11 Virtual Machines Vs Containers Overview.....	26
Figure 12 Overview of DIPA system architecture	27
Figure 13 Mininet Script : Create Link fat tree links	28
Figure 14 Routing table entry on the pulsar proxy	29
Figure 15 PTP Time Synchronization.....	30
Figure 16 DIPA's conceptual system model	33
Figure 17 Coarse grain NID data flow diagram	34
Figure 18 deploy_controller/DIPA-Controller.py : GetProtocol Function	35
Figure 19 deploy_controller/DIPA-Controller.py : Monitor Threads.....	35
Figure 20 deploy_controller/DIPA-Controller.py : Mirai_checker - botnet Classification.....	36
Figure 21 deploy_controller/DIPA-Controller.py : Monitor2 - Ingress rate limiting	37
Figure 22 deploy_controller/DIPA-Controller.py : getSwitch + link mapping.....	38
Figure 23 deployment_controller/DIPA-Controller : Add flow to open vSwitches	38
Figure 24 deployment_controller/DIPA-Controller : Drop all Telnet Flow Match	38
Figure 25 deployed_controller/DIPA-Controller.py : Control plan timer + production	39
Figure 26 global_view_and_timing/threaded_consumer.py : Time measurements	39
Figure 27 Deploy-Pulsar.yaml : Geo-replication on bookies.....	42
Figure 28 Geo-replication of example enterprise Data center	42
Figure 29 DIPA-Controller.py - Hard Timeout = send_timeout	43
Figure 30 templates/broker.conf - Deduplication flags.....	43
Figure 31 Deploy-pulsar.yaml : Message Retention and Compaction.....	44
Figure 32 Isolated Domains – traceroute	48
Figure 33 Reactive and Collaborative Flow rule	49
Figure 34 NID Algorithm Output	49
Figure 35 Functional Protection Flow rules	50
Figure 36 Bandwidth usage on switch ports.....	51
Figure 37 Ternary CAM OpenFlow operational lookup	52
Figure 38 Effects of polling rate on network Bandwidth	53
Figure 39 Pulsar to Broker encryption	56
Figure 40 Collaborative Protection Time with 2 Virtual Machines	59
Figure 41 Collaborative Protection Time with 3 Virtual Machines	59
Figure 42 Collaboration Time vs Background Traffic	61
Figure 43 Control Plane Time vs Background Traffic	61

Figure 44 Detection Accuracy vs Background Traffic.....	63
Figure 45 Most prominent Features in botnet Attacks [52]	66
Figure 46 Global WAN NIDS (Future Work)	70
Figure 47 Github User Documentation.....	79
Figure 48 AWS cost and resource usage per month.....	80
Figure 49 AWS Deployment setup.....	80
Figure 50 List of IoT devices , still vulnerable to Mirai [46]	81
Figure 51 Geo location heatmap from 2016 Dyn DDoS Attack [47]	82
Figure 52 Dictionary Content in main.go of mirai source code [48].....	82
Figure 53 Factory default password stored in Mirai SQL DB [49].....	83

Table of Tables

Table 1 CIDPS and the proposed Requirements.....	16
Table 2 Comparison of Opensource Controllers.....	20
Table 3 Pulsar feature critical analysis.....	22
Table 4 Effects of polling rate on network Bandwidth (Standard Deviation)	53
Table 5 Protection Time Standard Deviation (2VMs).....	60
Table 6 Protection Time Standard Deviation (3VMs).....	60
Table 7 Collaboration Time , Standard Deviation	62
Table 8 Control Plane Time , Standard Deviation	62
Table 9 Collaborative Detection Accuracy	63
Table 10 Risk mitigation overview	78
Table 11 Software Listing versions.....	81
Table 12 Control Plane : 1 Bot	83
Table 13 Control Plane : 10 Bots.....	83
Table 15 Control Plane : 50 Bots.....	84
Table 14 Control Plane : 100 Bots.....	84
Table 16 Control Plane : 500 Bots.....	84
Table 17 Control Plane : 250 Bots.....	84
Table 18 Pulsar Time : 1 Bot	84
Table 19 Pulsar Time : 10 Bots.....	85
Table 20 Pulsar Time : 50 Bots.....	85
Table 21 Pulsar Time : 100 Bots.....	85
Table 22 Pulsar Time : 250 Bots.....	85
Table 23 Pulsar Time : 500 Bots.....	85
Table 24 Vm Protection Time : 100 Flows	86
Table 25 Vm Protection Time : 1000 Flows	86
Table 26 Vm Protection Time : 500 Flows	86
Table 27 Vm Protection Time : 200 Flows	86
Table 28 Vm Protection Time : 5000 Flows	87
Table 29 Vm Protection Time : 7500 Flows	87
Table 30 Detection Accuracy : 1 Bot	87
Table 31 Detection Accuracy : 1 Bot	87
Table 32 Detection Accuracy : 100 Bots	88
Table 33 Detection Accuracy : 250 Bots	88
Table 34 Detection Accuracy : 500 Bots	88

1 Introduction

In March 2019 more than 56% of the world's population use the internet [1]. With this, industry has seen the emergence of new systems such as the internet of things (IoT). The IoT is a system of interrelated devices with the ability to exchange data over the network, without the need for human interaction. The volume of these devices in recent years has escalated, with an estimated 26 billion IOT devices by 2020, 60% of which are predicted to be exploitable poorly protected devices [2]. Whilst exponential growth in IoT presents advancements in technology, it also presents the opportunity for cybercriminals to exploit it. IoT has faced notable criticism in regards to data privacy, protection and security due to its static pervasive nature. IoT devices generally have low resources capabilities, unequipped to mitigate against edge based attacks.

On October 21st 2016 the world witnessed the largest distributed denial of service attacks with an estimated damages of \$110 million [3]. The attack was made possible by creating a veritable army of consumer IP enabled devices. Once infected, at a specific time these devices simultaneously generate large amounts of traffic towards critical infrastructures, exhausting their resources. The 2016 Dyn DDoS attack generated an unprecedented rate of 1.2Tbps, consequently immobilising company services for several hours. Recent Studies have shown that the power of these attacks continue to grow (Figure 1). Researchers reveal that next generation attack tools would be able to produce DDoS attacks that are thousands of times stronger than the 2016 Dyn DDoS attack [4]. Attackers can perform DDoS attacks using websites known as "Booters" which utilize lots of IoT devices, typically for a few pounds. [5]

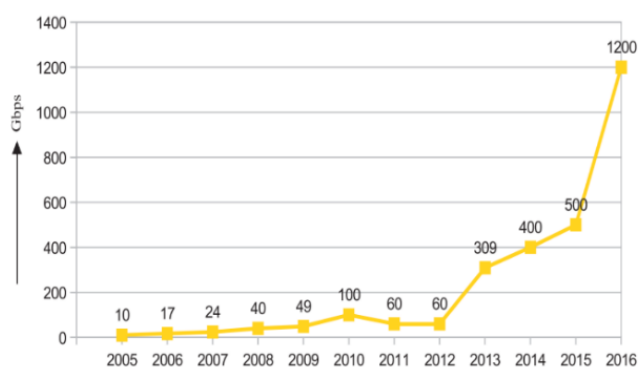


Figure 1 Power of DDoS attacks within the last decade

Researchers in turn set honeypots ; a security mechanism used to attract and record the botnet attack process. Attackers compromised these IoT devices using factory default authentication credentials to gather a mass of “zombies”, infecting weak neighbouring routers to spread malware. This malware worm is known as Mirai , which has brought a revolutionary change in cyber attacks [6] . Mirai utilizes telnet port 23 ; a communication orientated protocol that allows non-authorized remote access to these ill protected devices . Mirai still remains to be the most downloaded malware at 20.9% of all downloads. Kaspersky detail that 82.6% of attacks route back to Mirai malwares [7].

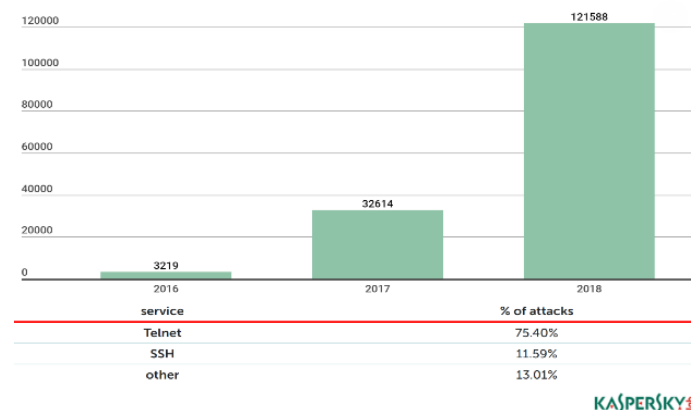


Figure 2 Number of malicious Devices found in 3 years

Research shows that infected devices were located at network domains geographically separated from each other. This however presents issues to popular detection and response mechanisms, deployed at the destination hosts. These intrusion detection and prevention systems (IDPS) work individually and independently to defend against the attack, making them ineffective against those attacks that are highly distributed. This calls for an efficient mitigation strategy to provide global synthesis to autonomous systems. Thus, industry have proposed collaborative architectures to detect and protect system wide attacks.

SDN is a promising paradigm to deal with such DDoS Attacks [8] . SDN offers network programmability of network operations through an abstraction layer. Separation of the data and control plan allow us to instruct the forwarding plane accordingly and drop malicious traffic . in this paper , a lightweight collaborative DDoS mitigation scheme is proposed. This project leverages SDN in conjunction with Apache’s publish subscribe messaging framework ‘Pulsar’ to protect neighbouring domains / controller on different autonomous systems (AS).

1.1 Relevance of the study

This research tries to understand the current state of collaborative intrusion detection and prevention technologies ; its performance but also its adaption to the advancement in distributed denial of service (DDoS) attacks in recent years. Power behind these attacks stem from the access to vast resources and weak IoT authentication. There is a substantial uncertainty around current protection mechanisms amongst IT organisations and security researchers. Causes of uncertainty include , CIDPS adaptability , network scalability but ultimately the immaturity of vendor solutions to offer full protection as DDoS attacks continue to scale . This research harnesses the attackers use of distribution and collaboration to offer global protection across Wide Area Networks (WAN)

This research provides a deep understanding of SDN as a new emerging technology in collaborative detection and protection of our network infrastructures. This research provides insight to the strengths and weaknesses of present solutions through background research and a literature review. A lightweight adaptable solution is developed to harness the collective knowledge from stand-alone network intrusion detection systems (NIDS). In turn, this offering global synthesis and protection. This project considers Apache's publish subscribe model 'Pulsar' as a viable collaborative protection framework, recommending future beneficiaries of intelligent edge based protection methodologies.

1.2 Research Question

This draws the ultimate research question :

Edge-based Network Attack Prevention using Apache Pulsar

In an attempt to answer the following research question a series of sub questions have been raised :

Q1 : What is the current state of Collaborative Protection ?

Q2 : How can we prevent further growth and harm from growing DDoS attacks ?

Q3 : Is Apache Pulsar a viable Solution for global synthesis

1.3 Beneficiaries of this Research

This research is designed to benefit security researchers, who wish to produce a complete end to end protection solution. As this study is based on extensive literature review and critical appraisal regarding the current state of CIDPS , this can benefit academic researchers in utilizing methodologies and concepts in this area. This research would also benefit organisations who aim to replace their legacy systems with new SDN , NFV technologies. Enterprises networks which employ a Software defined wide area network can integrate DIPA to offer global protection against insider attacks .

1.4 Project Goals and Desired Outcomes

With the drafted specification points, a series of project objectives have been constructed within the 30-week timeframe ,considering unseen challenges. These objectives have been ordered with priority and importance for research. The overarching goal of this project is to produce collaborative framework which will be tested with a lightweight NIDS. From the objectives the following metrics will be used to characterize the success of the project :

Reliability : The framework should accommodate for edge casing when exposing the system to various network conditions (eg. load testing) . Neighbouring Virtual Machines (VMs) should receive > 95% of malicious flows detected on the victim virtual machine.

Sustainability : Deduce the best topology setup. Where should pulsar sit ? , and propose additional security required to prevent system exploits . Consider Cost and resource usage of pulsar on daily usage.

Security : Produce a distributed architecture with no single points of Failure (SPoF) , providing recommendations for additional security for a complete end to end protected solution.

Suitability : Ultimately prove Apache Pulsar is a suitable framework for collaborative protection

Performance : Develop a solution that will collaboratively protect against a botnet attack across 3 isolated networks in <3s .

Scalability : Produce a scalable and manageable system that will compensate for different size local area networks (LAN) / interdomains. Pulsar framework should be able to handle domains ranging from size 1 – 50k hosts.

Interoperability : Develop a framework solution that will provide easy integration with other NIDS and different vendor solutions such as OpenDaylight (ODL) , snort.

The key and most paramount part of this project was to gain an understanding and deeper knowledge of the implementation of SDN and publish/ subscribe models and understand its importance within cybersecurity.

1.5 Project Scope

A series of assumptions have also been drafted to ensure clear focus and refined research to creating a collaborative IPS.

1. The collaborative flow rule protection is being tested, not a comprehensive detection algorithm to distinguish botnet attacks.
2. Coarse-Grain lightweight IDS to be used but only detect against edge-based attacks in scanning and loading phase . i.e not during port flooding (DOS)
3. Virtual implementation can be used rather than a physical topology to demonstrate a production environment on a smaller scale for a proof of concept.
4. Use Opensource software that is commonly used by many industries i.e , RYU, Mininet, Wireshark, AWS. Use existing applications don't reinvent.
5. Consider hardware limitations and limitations offered in terms of sourced software.
6. In production IoT devices don't reside on the same domain as switches and controller or have direct access control. This functionality is already extensively tested under SDN research so these can be virtualized under same domain (i.e Mininet).

2 Background Research

2.1 Understanding the Mirai Botnet

In September 2016, the Mirai botnet struck the security industry with three colossal distributed denial of service attacks. These attacks overwhelmed traditional standalone protection paradigms, introducing a new era of sophisticated cyber attacks tools. Mirai successfully generated 1.2Tbps of traffic, crippling several high profile services such as KrebsOnSecurity, OVH and Dyn [6].

Mirai harnesses the vast resources provided by thousands of small, innocuous internet-of-Things (IoT) devices such as security cameras, routers and even baby-monitoring systems. At its peak attack against Dyn, Mirai enslaved 600,000 vulnerable IoT devices. These interconnected computers can be controlled from an outside party through a remote procedure call (RPC). The Mirai malwares spread to vulnerable devices through continual scanning of the internet to find open, exploitable telnet ports. Mirai exploits IoT devices using factory default or hard-coded username and passwords.(Annex 4.4). With a vast army of bots enslaved, the attacker has access to a make-shift supercomputer to complete their nefarious attacks. Due to the distributed nature of the attack, this 'super-computer' can be hard to stop. IoT devices were also attractive to attackers as they often fall short in endpoint protection implementations.

Researchers set honeypots on IoT devices after the attacks for deep packet inspection to learn the attacks attack phases. Preliminary analysis showed that generated traffic used Generic Routing encapsulation (GRE) to restrict packet malicious visibility. Mirai remotely connects to targets using telnet access points, which are often left open by default. The malware uses a dictionary of commands to ensure an exclusive communication path between the bot and a control system known as the Command and Control (C&C). Annex 4.3. Once the commander has successfully enslaved a series of bots, the bots will join the port scanning phase to look for potential victims. Bots report the source IPs of potential victims on port 48101 to the Reporter Server. This will contact a loader to download malware to the targeted IoT device. Once the attacker has enslaved enough bots, a simultaneous SYN flood can be performed on the network infrastructure. Figure 3 summarizes Mirai's architecture.

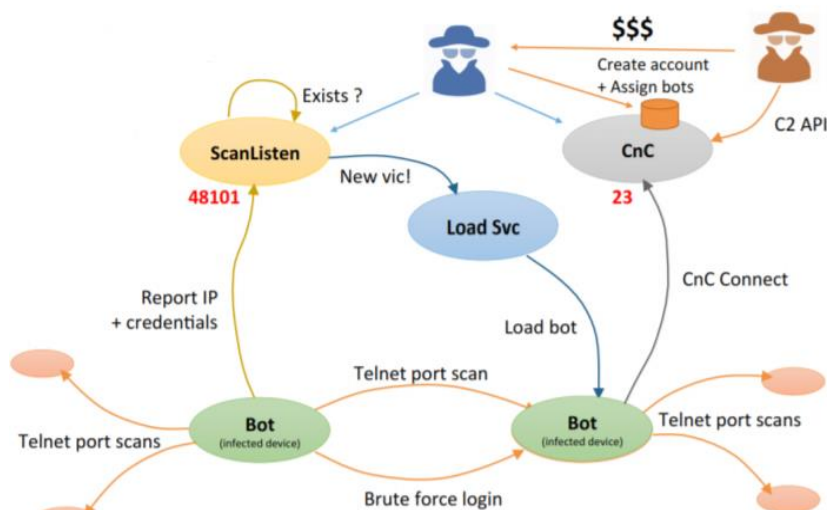


Figure 3 Overview of Mirai Botnet Malware spread [50]

The Mirai source code was released publicly in November 2016. The code contains a collection of go languages files. The main files listen for TCP connections continually on port 23 and 2323. The concern with Mirai is its ability to run on many architectures such as x86 , ARM , MIPS , m68k . After the malware is loaded , the bot prevents watchdog timers from rebooting the device. ,killing all SSH and telnet services on ports 80, 22 ,23 and 2323 . In this work , DIPA explores the scanning phases for early detection and mitigation approaches to prevent further spread of Mirai throughout the wide area Network. This phase is favoured in botnet mitigation schemes as the bots in this phase use a single telnet communication flow to random IP destination addresses.

2.2 Taxonomy of Collaborative Intrusion Detection and Prevention

2.2.1 Architectures

Our dependency on 24/7 availability of networked computers has become quite frightening. These system are beginning to be targeted by a number of sophisticated attacks. To counteract, industry have proposed collaborative prevention systems. These system consist of several monitoring devices which feed information to one or several correlation units to distribute the data. Collaborative intrusion detection and prevention systems (CIDPS) can be classified dependent on their communication architecture. As shown in Figure 4, the three architectures are centralized, decentralized and distributed.

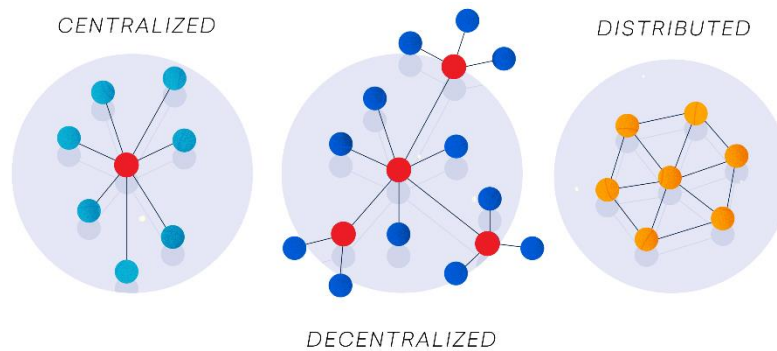


Figure 4 Overview of CIPDS architecture ,Blue = Monitor unit ,Red = Correlation Unit, Orange =Monitor + Correlation Unit [9]

Centralized – These system consist of one or several monitoring units connected to a single correlation unit. These monitors share data with the central correlation unit to distribute among each of the neighbouring domains. Centralized solutions commonly integrate detection algorithms at the central unit. This architecture is known to provide high accuracy rates at low network sizes due to their low architectural complexity. However these fail to maintain system availability as networks scale , due to the resource demand and allocation to a single processing unit. Furthermore these system all introduce a performance bottleneck and a single point of failure (SPoF) , leaving the entire system integritys vulnerable to a single point.

Decentralized – This architecture organizes several monitoring units or multiple IDS deployments in a hierarchical structure. This architecture allows system control to be operated from multiple control points whilst rooted at a central analysis unit. This architecture can overcome bottleneck issues exhibited by centralized topologies however still exhibit vulnerability points which can challenge system availability on sections of the network. Furthermore ,with decentralized systems, information is often lost at each hierarchical level within the tree making it hard to effective distinguish and correlate against highly distributed attacks.

Distributed – A distributed Architecture will share each of its task among all the central units. Commonly in many existing solutions the correlation unit is also the analysis unit . These systems exhibit a peer to peer (P2P) design principle, scaling with any number of monitoring units within an arbitrary network size. No hierarchical structure accommodates for flexibility in interconnects, but with no central unit it is hard to distinguish a global view of the system.

2.2.2 Requirements for CIPDS

Industry have proposed a set of requirements in 2015 that a CIDPS must fulfil to provide complete network protection [10] . These requirements have been refactored below to accommodate recent development in DDoS since 2016.

Data privacy – Exchanged alerts may contain sensitive information regarding LAN topology. Sensitive information ideally should not be disclosed . If needed system must ensure full system coverage and not store all information of each LAN in a centralized exploitable point.

Scalability - Systems must contain no single points of failures ,or points for bottleneck which can be exploited. Performance of NIDS must grow linearly proportional to the size of the network

Interoperability – The system must accommodate for proliferation of incumbent system (system that already have a vast number of deployed IoT devices with legacy devices and no SDN). The system is recommended to used a standardized format such as the intrusion detection message exchange format (IDMEF).

Minimal Overhead – Techniques used for data dissemination must have low computational overhead. Overhead exploitation must be considered and analysed.

Resilience – The system should still maintain availability if under attack , whilst maintaining high accuracy levels. This should accommodate for SPoF and resilience to internal attacks.

Configuration – The system must be easily deployed and dynamically adjust to the given surroundings. If manual configuration is required, proper documentation must be provided.

2.3 Existing Solutions

This section looks at related work and existing collaborative solutions to combat these botnet attacks . A range of architectures and correlation units are analysed , to find the strengths of these protection systems, but also to find areas for improvement to provide complete end to end protection.

Paper 1 : A Collaborative IDPS in cloud Computing [11]

Hassani et al. propose a collaborative model ,comprised of hybrid detection algorithms and correlation units to prevent against highly distributed attacks. This system focuses on early detection such as port scanning , integrating the Signature Apriori Algorithm for generating new attack signatures. A decentralized architecture is proposed to collect and analyse local events to share these to destination hosts through cloud computing. The model focuses on the infrastructure layer to trigger event correlation.

Alert messages are structured via XML , to ensure interoperability between different systems . Data dissemination is proposed through a polled broadcasting system , this in turn introduces overhead if the broadcasting period is shortened . The system fails to offer reactive mitigation to the attacks when the attack is detected . Nevertheless resilience against insider attacks is achieved by the node architecture and flexibility. Hassen et al fail to implement the system , providing no factual evaluation behind their implementation . Further analysis is required to ensure the system can provide its claimed proposals.

Paper 2 : GrIDS Graph Based Intrusion Detection System [12]

GrIDS is a decentralized protection system , intended to protect large enterprise networks from propagating malware. GrIDS splits the network into several ‘departments’ ,organized in a hierarchical tree structure. Each department contains two special modules ; the software manager and a graph engine. The software manager is responsible for local domain hierarchy . The graph engine receives inputs from each of the monitors within a short timeframe to classify suspicious behaviour . Information is aggregated to the parental department to send out protection alerts.

As data is processed over a short time period, it suggest that the system is unable to process slow progressive attacks such as recent botnet attacks. GrIDS is a highly scalable framework due to its hierarchical department organization , this allow the system to protect networks of arbitrary size . The parental department however introduces a Single point of Failure (SPoF) and potential for bottleneck . As with most centralized and decentralized CIDPS , GrIDS is susceptible to insider and DDoS attacks , but can handle SYN flood attacks effectively. GrIDS’s best feature is it built in privacy protection mechanism ; this lets each department in the hierarchy only observe activity in a restricted configurable boundary.

Paper 3 – CoFence – A collaborative DDoS Defence using Network Function Virtualization [13]

CoFence present a collaborative DDoS defence system across Network Function Virtualization (NFV) based peer to peer domains. CoFence prioritises resource management and low computation power to reduce capital wastage of IDPS each year. CoFence state *“IDPS often present problematic costs versus functionality trade-offs”* [13]. With this CoFence use NFV to allow each domain network to decide the amount of resources to share with other peers through a ‘reciprocal-based’ function. Each NFV-enabled domain receive a virtual gateway and IDPS to detect and filter DDoS attacks.

With the flexibility of NFV , CoFence design a dynamically configurable IPS , allowing users to change domain permissions and its external traffic capacity. CoFence also consider system exploits by integrating a service agreement (SA) process for trusted virtual machines. CoFence address excessive traffic communication between neighbouring domains through a DDoS filter and resource allocation scheme . Each Domain decide how much spare resource it can offer to its requesting neighbours.

Although CoFence limit the malicious information exchanged over the network , the use of the centralized architecture and resource exchange impose system exploits. A single domain , if compromised can request resources from other domains , limiting performance of neighbouring NIDS. CoFence concentrate on resource allocation and IPS configurability .As a result, many details are missing to assess CoFence in terms of the CIPDS requirements set in section 2.2.2 Requirements for CIPDS

Paper 4 – Alert Correlation in a Co-operative Intrusion Detection framework (CRIM) [14]

CRIM utilize a centralized architecture that co-operatively obtains data from a series of isolated NIDS. CRIM focuses on alert analysis and the subsequent steps for mitigating the adversarial attacks. CRIM use a centralized relational database to store global alerts from each source . This is achieved by collecting a series of tuples which are subsequently feed into a correlation function. Adversarial attack information is displayed to the user with their next predicted steps.

CRIM is an early adaption of collaborative protection in which most recent systems use as a base reference point. CRIM provide a system with primary focus on attack correlation , this however is under developed to relate to recent sophisticated attacks . Emmanouil

Vasilonmanolakis [10] details that the system would be unable to protect against a slow distributed attack from different sources. This project aims to focus on the slow phase of the Mirai botnet attack, making this solution inadequate. This system uses the intrusion detection message exchange format (IDMEF) proposed in RFC 4765 [15]. This standard is used to address IDS interoperability, thus allowing CRIM to be applied to a variety of Network intrusion detection systems (NIDS). IDMEF is an experimental protocol using the extensible mark-up language; XML. However recent implementations have opted to adopt their own exchange format to create a more flexible framework.

Paper 5 – ECESID : Software Defined Edge Defence against IoT Based DDoS [16]

ECESID introduces an edge-oriented detection and migration scheme using SDN and Fog computing. Through the use of fog computing the system can offer cloud computing features at the edge of the network. ECESID adopts the SDN centralized paradigm in a distributed, localized and closer to the edge architecture. Mert et al. utilize credit based rate limiting (TRW-CB) at the end device of the network. Multiple correlation units / Fogs are located at the edge of each of the domains to provide cloud like services and data dissemination across each destination domain. The system is tested against a simulated Mirai botnet attack, as the prime focus of this project is to introduce an edge-oriented mitigation scheme. DIPA utilizes similar simulations to test Mirai mitigation.

ECESID can be viewed as a dynamic overlay in which local fog agents communicate with the global cloud hub. ECESID utilizes a collection of promising mechanisms towards distributed protection, in the form of distributed communications. This allows small cumulative attacks previously undetected to be seen from a global view. This system sends these agents to collect sensitive information from neighbouring fog agents which may conflict with the privacy requirements. ECESID puts primary focus on information exchange and detection, in turn introducing system exploits which could cause further harm. ECESID states they have integration issues approaching heterogeneous and loosely managed IoT networks that don't use legacy devices with no SDN. The future of IoT depends on an adaptable retrofittable system that will cover these issues.

Paper 6 – FireCol : A collaborative Protection Network for Detection of DDoS Attacks [17]

FireCol is a centralized CIDPS that detects DDoS flooding attacks within the internet service provider (ISP) layer. Each analysis unit/ IPS forms a virtual protection ring around each of the hosts. These units exchange specific traffic information for the early discovery of highly distributed botnet attacks. To achieve this, FireCol split their networks into a series of correlation and detection units. These configurable detection units pass aggregated traffic to a score manager who applies a score to each selected rule based on their frequencies , entropies and scores assigned by neighbouring IPSs.

FireCol utilize a scalable solution for early detection of flooding DDoS attacks. Effective use of belief score collaboration between each ‘virtual protection ring’ provides accurate coverage of the entire WAN. FireCol provide a lightweight robust system that is highly configurable , protecting users while saving network resources through positioning of the correlation unit at the ISP level. FireCol also address low setup time ,resource consumption and facilitated accounting , all as an incentive for mass deployment by ISPs. Detection and correlation are centralised at the same level , making the entire WAN susceptible to highly distributed attacks with little additional security strategies in place such as private keys.

Paper 7 – A collaborative Intrusion prevention architecture for programmable Network and SDN (CIPA) [18]

CIPA design a distributed CIDPS prototype that harnesses software-defined networks. An artificial neural network is applied over the substrate of a series of virtual networks. CIPA introduces back propagation neural networks which analyses a series of feature weights to classify the attack . These features are captured through a series of OpenFlow switches and a POX controller. Self-organized feature maps (SOFM) are used ,a form of unsupervised learning to produce low dimensional , discretized representation of the flow rule features.

After critical analysis CIPA were unable to uphold promises of their network capabilities. CIPA aimed to use 500 and 1000 node test cases , with aims to show that ANN is lightweight and effective even after network scaling. At a test case of 200 nodes , packet rate was capped at 8kbps. This is a results of the algorithms $O(n^2)$ performance complexity and the overhead produced the ANN. In CIPA’s self evaluation ,they address the system inability to function under a DDoS attack. The CIDS will produce a large amount of protection message , consuming the networks bandwidth if experiencing a flooding attack.

Paper 8 – INDRA : Intrusion detection and Rapid Action [19]

Janakiraman et al. have proposed a peer to peer (P2P) CIDPS to protect against generic malicious activity. INDRA apply a pastry distributed hash table (DHT) for efficient storage of attack related information. Information detected by the system is managed across analysis units using scribe [20], a large scale decentralized publish subscribe model. Each domain consists of four daemon sub components; watchers, access controllers, listeners and reporters. An attack category is created for each classification of attack , in which neighbouring nodes subscribe to . INDRA focuses on the administrators ability to create new attack plugins.

Although the proposed system it highly scalable and interoperable with different NIDS, this requires a lot of manual intervention. INDRA doesn't support an automatic way to integrate new attack classifications. INDRA main concerns is it lack of attention to system exploits and vulnerabilities. INDRA fail to consider a compromised monitor producing fake alerts, causing reluctance in industry to use INDRA as a viable solution. INDRA appends malicious IPs to a blacklist , blocking all communications. If exploited , the attacker can control legitimate traffic in the network.

Paper 9 – NetBiotic : Mimicry attacks on host-based intrusion detection systems [21]

NetBiotic adopts a distributed CIDPS architecture. This system uses the juxtapose (JXTA) open source peer to peer protocol to co-ordinate its protection mechanism. Similarly to DIPA this systems focal point isn't on attack classification but rather swift domain protection from correlated alert information exchange. NetBiotic's domains are both a monitor and analysis unit to detect against rapid propagation of malware throughout the network. NetBiotic consists of a notifier and handler component. The notifier will read log files , producing statistics of the attacks. The handler is responsible to triggering defence mecansims. NetBiotic takes actions when the statistics is higher than a preconfigured average i.e. 'pre-protecting' against an epidemic attack.

NetBiotic focuses on protection mechanisms , with attacks remaining undetected if the no significant overall difference between attacks. NetBiotic's architecture accommodates for network scaling . However the system requires a parser for each of the log files resulting in

interoperability and compatibility issues with any other NIDS. With zero consideration to insider attack ,use of security countermeasure such as encryption , key pairs or certificate authority (CA) , the proposed systems may be exploited .

Paper 10 : EMERALD : Event Monitoring Enabling Responses to Anomalous Disturbances

EMERALD [22] is decentralized CIDPS constructed to audit large SCADA networks , targeting illegitimate access of each sub domain's resources. This system employs three hierarchical layers ; service analysis , domain-wide analysis and enterprise-wide analysis. The service layer covers detection from a single domain. Domain-wide analysis covers multiple domain components. The enterprise layer aims to monitor traffic across a collection of domains before applying attack mitigation defences. EMERALD uses a range of signature and anomaly based detection engines at each of the layers. Data dissemination is achieved through Elvin ; a publish subscribe event delivery model [23]. Each domain synchronously subscribes to the model and receive alerts from neighbouring monitors.

Although EMERALD is an early approach to collaborative protection, the system adopts effective mechanisms for a complete end to end protected IDPS. The authors advise on pluggable security such as cryptography keys to protect each domain. EMERALD provides an API to allow the system to interconnect of different NIDS with no standardized data format. Data dissemination through a publish subscribe framework provides minimal overhead. The hybrid detection mechanism produce high accuracy however EMERALD has failed to adapt to recent intelligent attacks. Alfonso et al . Alert correlation environments such as [24] detail that the system provides zero detection or protection for insider attacks.

2.3.1 Related work comparison

Table 1 CIDPS and the proposed Requirements.

✓ = requirement fulfilment ✕ = requirement not fulfilled Φ = Partial Match ? = Unknown Cases

CIDPS Paper	Detection Approach	Architecture	Detection Accuracy	System Security	Botnet Resilience	System Scalability	Resource Consumption	Setup Time	Overhead	Data Privacy
1	Pattern Match	Decentralized	Φ	Φ	Φ	✓	✓	?	✓	✕
2	Hybrid Based	Decentralized	✕	?	✕	Φ	✓	?	✓	✓
3	Resource Alloc	Centralized	✕	✕	✕	✓	✓	?	✓	✕
4	Net-Based	Centralized	✕	?	✕	✕	✕	?	✓	?
5	Rate Limiting	Distributed	✓	✓	✓	?	Φ	✕	?	✕
6	Score-Collab	Centralized	Φ	✕	✕	✓	Φ	✕	✕	✓
7	ANN	Distributed	✓	?	✓	✓	✕	✕	✕	✕
8	Signature Pub Sub	Decentralized	Φ	✕	Φ	✓	✕	✓	Φ	✕
9	Host-Based	Distributed	✕	✕	✕	✓	✓	?	✓	✕
10	Hybrid PubSub	Decentralized	✕	✓	?	✓	✓	?	✕	?
DIPA	SDN , PubSub	Distributed	Φ	✓	Φ	✓	✓	✓	✓	✓

Table 1 details the requirements fulfilment of existing CIDPS ,presenting areas requiring improvement across multiple solutions. This is analysed in comparison to DIPA’s conceptual design. Many centralised and decentralized systems do not scale with the number of correlation units required , thus they cannot protect large networks. DIPA utilizes a distributed architecture following the P2P design principle. The lack of hierarchy restricts provides more freedom with monitor interconnections , which can benefit against sophisticated attacks. This paper proposes to provides a novel lightweight collaborative model through a collection of deployed clusters on AWS. This framework aims to propose pulsar as a viable solution to solve resource allocation , overhead and system security issues presented. Pulsar pub/sub model has been chosen as it currently serves 1.4million topics within production and provides an average publish latency of <5ms. DIPA attempts to provide an interoperable framework to be utilized with more comprehensive NIDS to improve detection accuracy and botnet resilience. A coarse grain solution is implemented to offer partial coverage.

3 Technologies

3.1 Software Defined Networking

Emerging technologies such as Software Defined Networking (SDN) and Network Function Virtualisation (NFV) have introduced an intelligent architecture for network programmability and logically centralized control. SDN is a promising paradigms that offers network operations to be programmed through an abstraction layer. Yan et al. have deduced that SDN is a efficient tools for DDoS detection and mitigation. Large companies such as Google have deployed SD-WAN system with great success. Yan et al. present three DDoS defence mechanisms harnessing SDN technologies ; source based, network based and destination based [25]. This research focuses on source based traffic attack mitigation.

3.1.1 Traditional Networking vs the SDN paradigm

SDN is a new concept in networking that simplifies network control through a programmable central point. This is achieved by decoupling the data and control plane to allow an administrator to control lower level protocols using high level programming. In SDN, the control plane is separated and controlled using another machine known as a controller. In traditional networking each switch has its own local brain, that makes local decisions when a packet is received. Each switch locally updates it MAC table using info it sees. It is unable to view flow statistics that have reached another switch. SDN technologies provide network architecture, programmability, cost-efficiency and vendor-agnostics. [26]. The abstraction difference is represented in Figure 5 .

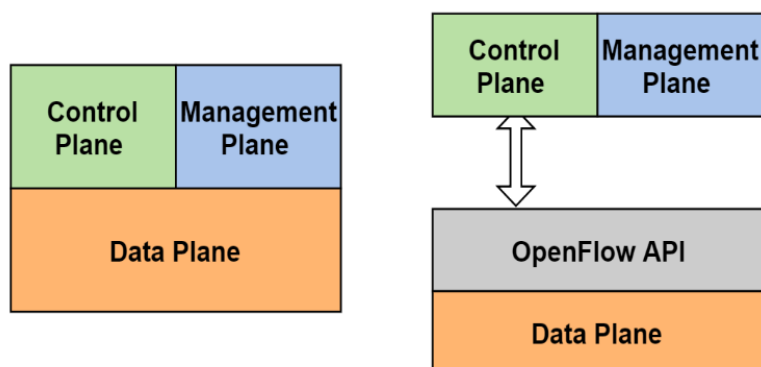


Figure 5 SDN vs Traditional networking

Within conventional networks the control and data plane reside on the same machine, however SDN technologies allow servers to be controlled remotely. This achieved using the north bound interface in the form of rest application programming interfaces (APIs). These APIs abstract network level services and allow manipulation of traffic and lower level protocols . Common examples of these rest APIs include Ryu.ofctl [27] and MD_SAL RESTCONF [28]. Northbound APIs are designed to support a variety of applications and standard assigned by the Open Networking Foundation (ONF). In Software defined wide area networks (SD-WAN) , most LANs typically do not integrate the same controller, architecture or hardware. The northbound allows the controller to control a series of applications of different types.

The southbound interface allows the controller to connect to communicate with lower-level components for network orchestration purposes. In SDN, the southbound interface is in the form of the OpenFlow protocol, a eminent interface and industry standard. OpenFlow enables communication between the controller and networks nodes (physical or virtual switches and routers). This interface is used for network discovery, flow rule definition but also to relay requests to the application layer through the northbound interface. Traffic routing is achieved in the network using OpenFlow flow tables set by the controller. Each switch has an ingress and an egress path that is constructed in each of the flow tables. OpenFlow allows the controller to access these tables, adding, deleting and modifying flows. Figure 6 demonstrates the architecture found in SDN.

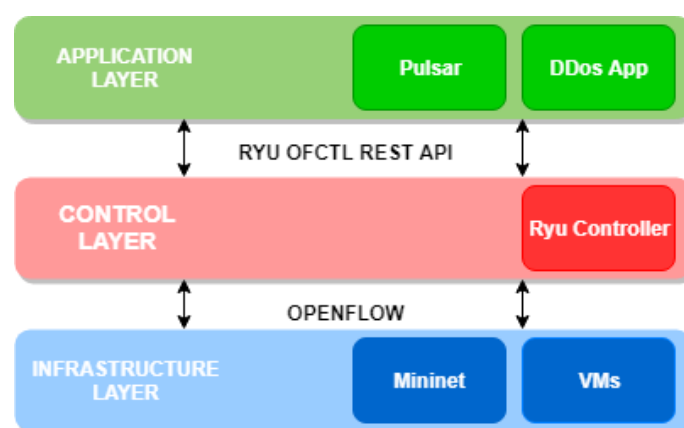


Figure 6 SDN architecture with OpenFlow protocol

3.1.2 Ryu Controller

Ryu is a lightweight, component-based software application designed to centralize the networking framework. Ryu supports a variety of southbound protocols such as Netconf, BGP and OpenFlow. This paper implements Ryu with OpenFlow1.3. Ryu is used to manage the infrastructure layer by constructing a series of reactive and proactive flow rules.

Reactive flow rules are activated when a OpenFlow switch (OVS) receives a packet which it currently has no flow entry for. The switch notifies the controller to build a flow entry with the given packet fields. These fields can be seen in Figure 7.

Through network programmability, these packets are parsed and analysed to determine an appropriate action for the given packet before installing the flow entry. Proactive flow rules pre-program these flow entries upon network setup stage. Rules can be analysed by the controller by a statistics request to each Datapath/switch. These statistics can be analysed to classify a DDoS attack, this provides the basis of the coarse grain detection algorithm.

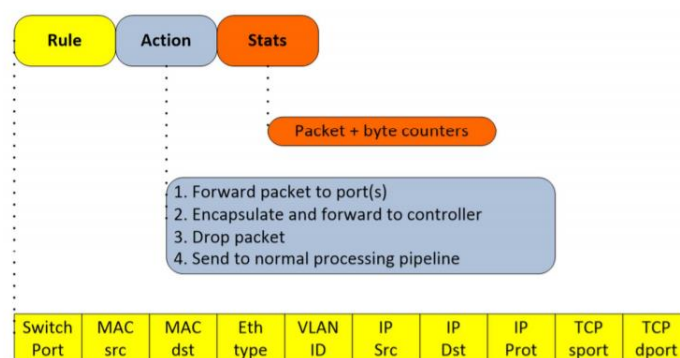


Figure 7 OpenFlow Table Entries [51]

A collection of opensource controllers have been analysed to support an early detection botnet application. This application will managing flow entries within each domain/LAN. As seen in Table 2 , each controller exhibits different strengths and weaknesses due to their application usage. Floodlight is written in Java and support openFlow1.0 . Floodlight is maintained by big switch networks, a large enterprise organisation that provides extensive documentation with a strong community. Floodlight also has good performance with a round trip time improvement of 14.4% over NoX Controllers. However, floodlight has a steep learning curve and is not suited for prototype designs.

OpenDaylight is a comprehensive controller, supporting stateful and stateless packet analysis. Stateful packet analysis compares the current packet with previously extracted information, where stateless retain no information of the state of communications. This project studies the art of stateful communications. OpenDaylight is soon to become an industry standard with many industries utilising this architecture, however it is not advised for prototyping. This controller requires a deep understanding of the application architecture. This controller has a smaller community base than Ryu, with less solutions and developed DDoS applications. This project aims to utilize a pre-built NIDS, thus solutions are required.

Table 2 Comparison of Opensource Controllers

Uses Cases\ controllers	Nox/Pox	OpenDayLight	Floodlight	Trema	RYU
Routing	✗	✓	✓	✓	✓
Lightweight	✗	✗	Φ	Φ	✓
Network monitoring	Φ	✓	✓	Φ	✓
NFV support	✓	✓	Φ	✓	✓
Traffic Engineering	Φ	✓	✓	Φ	✓
Load Balancing	✗	✓	✗	✗	✗
Legacy Network interoperability	✗	✓	Φ	✗	✓
Multi Layer Network Optimisations	✗	✓	✗	✗	✗
Service insertion and Chaining	✗	✓	✓	✗	✓

3.1.3 SDN and IDPS

Legacy systems that employ DDoS applications often require manual intervention, making it hard to effectively classify attacks across our Wide Area Networks (WAN). SDN replaces specialised hardware and software for networking monitoring with OpenFlow switches. This can reduce the cost and latency in the system whilst improving our detection and mitigation schemes. SDN has received a lot of attention from researchers in recent years, as it addresses the lack of network programmability and allowing the network to enable faster network intervention to DDoS attacks.

OpenFlow switches are used in replacement of legacy switches and firewalls, which are hard to manage, can stop legitimate traffic and are unable to adapt and evolve to recent evolution in highly distributed attacks such as Mirai. In the event of an attack, the controller can use stateful packet inspection to classify the packet and install a drop rule on each switch. In comparison to legacy system, traditional networks require firewalls at each of the boundaries. Utilization of SDN also eliminates the management servers / control plane from

being overwhelmed with harmful packets. Traditional networks filter the traffic on the switch even if the traffic is classified as harmful. In SDN these switches don't redirect the malicious packets to the control layer, leaving network orchestration intact.

3.1.4 NFV and IDPS

Network function virtualization (NFV), virtualizes layers 4-7 functions such as firewalls, load balancing and most importantly IDPS. NFV aims to segment layer functions on selected tunnels in the network to save time on network provisioning and training. NFV accommodates for network management by decoupling the IDPS to virtualize remote servers. This allows administrators to run controllers and the IDPS on high performance x86 platforms.

As shown in Figure 8, NFV abstracts layers 4-7, layers used for stateful packet inspection. The combination of SDN and NFV can remove dedicated appliances and expensive hardware with generic hardware and advanced software. The control plane is moved from the dedicated platform to an optimized data centre location which is less expensive. Abstracting the network layers allows for better network orchestrations, performance and manageability.

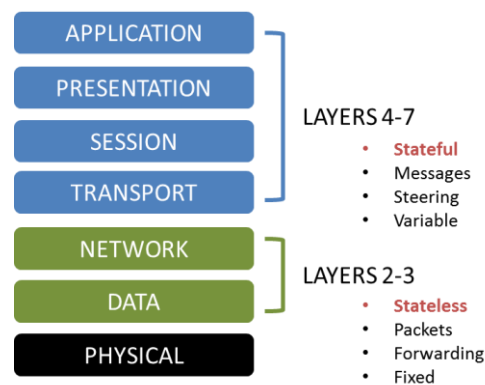


Figure 8 OSI layers, Stateful vs Stateless

3.1.5 SDN Project Relevance

SDN is a promising paradigm to mitigate the recent evolution in highly distributed attacks. Adopting the SDN architecture has a myriad of benefits including scalability, interoperability, flexibility, but most importantly protection. Utilizing SDN allows DIPA to monitor the network, exchanging malicious information between each controller. Alongside flexibility, administrators can simply add more virtual switches to scale the system. Removing the purchase of costly equipment and licensing faced within traditional networking, SDN can be

tested on these virtual systems before easily deploying into production . SDN is also portable when used with NFV , allowing for public or private cloud storage.

3.2 Apache Pulsar

3.2.1 Introduction to Apache Pulsar

Apache pulsar [29] is a multitenant publish subscribe model that supports a form of asynchronous service-to-service communication. The pub sub model makes use of the Data distribution service (DDS) middleware to share meta data via IP multicast and a series of brokers. Devices that aim to push messages to enable an event driven architecture are known as producers. Message are sent via a live data stream (topics) to update all subscribed devices , known as consumers. Unlike Polling architectures such as web sockets, producers immediately notify all subscribers within the channel through an intermediary called a message broker. This pattern provides loose coupling , simple communications and system reusability to facilitate peer to peer communications.

Table 3 Pulsar feature critical analysis

<u>Pulsar Pros</u>	<u>Pulsar Cons</u>
Low publish and end-to-end latency	Small community with little tutorial help
Simple APIs with seamless scalability	No message ordering
Lightweight compared to other pub sub	Cant retrieve last message easily
Loose Coupling i.e producers and consumer don't need to know about each other	Higher operational complexity compared to other messaging frameworks
Free to use and can test locally without cloud access	Device ID of subscriber can be spoofed without TLS
High throughput under load surge	Vulnerable to authorized message senders that behave maliciously
Tiered Storage through used of bookkeeper and zookeeper	

3.2.2 Messaging Concepts

Pulsar is designed to support multitenancy use cases in which a property/application represents a tenant in the system . This allows Pulsar to support a wide range of applications running in the same cluster. Each property contains a namespace, which is used for application use cases. Namespaces are the basic administrative unit within the pulsar cluster. At this level ,users can set features such as permissions, geo-replication , namespace retention and control message expiry . All topics within the same namespace inherit the same settings. There are two namespace types based on their level visibility. Local

namespace is only visible to the cluster it is defined in. Global clusters are visible across multiple tenants / properties. This is effective in data centres or systems that are geographically separated. With given namespaces the user can write to topics to exchange information. Topics that are undefined are created ad hoc.

Pulsar possesses three subscriptions models to receive messages published to topics; exclusive, shared and failover. This report studies and implements the shared implementation. With this each IDPS attaches their consumer to the same subscription, receiving a copy of unacknowledged messages that have been pushed. Exclusive and failover subscriptions only allow one consumer to receive and send messages at one given time. This is represented in Figure 9.

Unlike polling communication frameworks, publish-subscribe protocols usually specify requirements for protecting messages at rest. This forces Pulsar to utilise local cache encryption within its application layer. Pulsar utilises its API framework to authenticate and encrypt messages. Messages remain encrypted within the Pulsar bookie disks and can only be decrypted by an authenticated consumer. Encryption keys must be stored in a hardware trust module or in an air gapped software safe. This is described in further section 6.2.5.

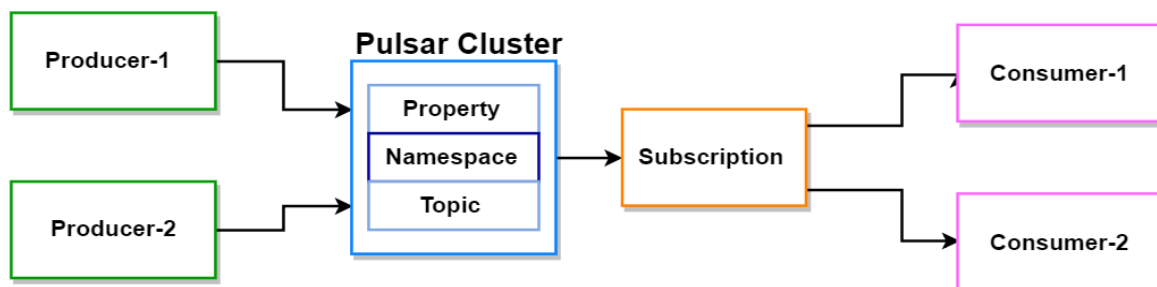


Figure 9 Producer, Consumer and Topics (messaging concepts)

Alternate Publish subscribe models use HTTP, MQTT and AMQP to communicate over websockets. Pulsar utilises a custom binary protocol to communicate between nodes. This protocol has more compact libraries, using less than 1MB for each device. This allows each system to consider security features to protect the LAN's integrity. Unlike other collaborative IDPS models, pub subs can act as an intermediary shock absorber between each NIDS. With the malicious production of sudden spikes in traffic to other networks, Pulsar can gather, buffer and deliver these telemetry messages in a safe manner.

3.2.3 Pulsar architecture

The Pulsar cluster consists of 3 main components; brokers, bookies and Zookeepers . Each piece is key for data partitioning, coordination and configuration management. The pulsar architecture commonly consists of a collection of brokers. Within distributed computing , a broker is middleware that receives, stores and delivers messages from a producer to each subscribed consumer whilst providing location transparency through remote procedure calls.

As packets are sent to the cluster ,the broker unpacks the protobuf message frame into two sections ; simple commands and payload information. Simple commands are passed to the in-built service discovery mechanism that alerts the broker of topic creation , topic updates , topic deletion or configuration changes .

Once the packet is received, pulsar guarantees packet retention in the event of power failures or errors. The brokers transmit topic data to the bookkeeper which holds multiple bookie nodes . These bookie nodes make a copy of the data into memory whilst simultaneously writing the data to the write-ahead logging (WAL) . This ‘stable storage’ ensures no data is lost in transmission and can be recovered in the event of a bookie node being compromised or hardware failure , a failover bookie can thus take over.

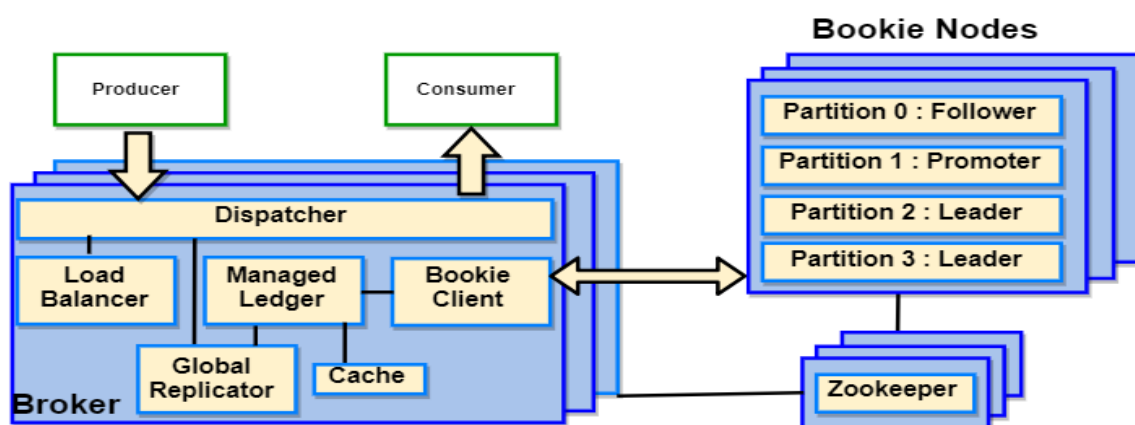


Figure 10 Pulsar Cluster Architecture

To ensure high throughput and data protection in the event of topic load surge, pulsar enables data partitioning across the bookie nodes . Pulsar shards the data into segments to store across multiple bookie nodes . The shards are represented by partitions in Figure 10. When a topic is created and data passed , Pulsar automatically distributes these data shards.

These shards are merged upon consumption to produce the whole message, whilst retaining a copy within the bookie nodes until messages have been successfully acknowledged. This message is also removed when the time to live (TTL) expires, or if namespace retention runs out of memory. Messages are dispatched to multiple subscribers through a managed ledger and cache. This is a TCP server which exposes a REST API to serve the custom binary protocol.

Pulsar's bookkeeper provides I/O isolation, this places read and writes on different paths to ensure of no topic lag upon lookup. This is achieved by Pulsar's publish/subscribe queue model. This allows applications to divide up processing among multiple subscribers for round-robin delivery. This is further explained in section 4.3.1.

The final component of Pulsar is ZooKeeper. An external system, responsible for global configuration of each producer, and task co-ordination. In a single Pulsar instance, Pulsar uses a local ZooKeeper that operates at the cluster level, and a configuration store which operates at the instance level. The local ZooKeeper quorum manages task co-ordination such as data partitioning in bookie nodes, topic creation and message lookups. The configuration store, stores information regarding the partitioned memory addresses, system latencies and authentication token IDs (not the tokens themselves). Decoupled producers and consumers (Thin nodes) authenticate through the broker which are fed token IDs by the ZooKeeper configuration store.

3.2.4 Project Relevance

Apache Pulsar is a promising open-source framework for Collaborative IDPS. As seen in section 3.2.1, existing CIDPS such as EMERALD and INDRA employ a publish/subscribe pattern to achieve global synthesis. EMERALD's pub/sub model 'Elvin' has been praised for its sophisticated content-based subscription system [10], but currently has a small community with limited development support. Similarly, INDRA's scribe pub/sub framework is an early adaptation of the publish/subscribe framework, with little consideration to authentication and authorization. Apache Pulsar provides a sophisticated model with high throughput, low latency, extensive security but also additional functionality such as Georeplication, cloud deployment flexibility and round-robin partitioning; features necessary to DIPA.

3.3 Topology Virtualization

3.3.1 Virtual Box

VirtualBox is a cross platform virtualization software that can be installed on existing intel and AMD computers. VirtualBox emulates a computer system by partitioning the hosts machines resources using an ISO image. Allowing the user to allocate hardware resources and a operating system (OS) which is monitored by VirtualBox's inbuilt hypervisor.

VirtualBox can host multiple machines within a single host machine. VirtualBox is utilized in this project to emulate a SD-WAN without using separate hardware servers, for cost efficiency and quick development. To control the development environment , each VM is cloned. Unlike container-based system such as Docker which share the same OS, VirtualBox ensures a fully isolated network between hosts at the cost of extra memory usage. Traffic shaping and iptables for firewall isolation is control from the OS layer. The difference in virtual machines and containerisation is shown in Figure 11.

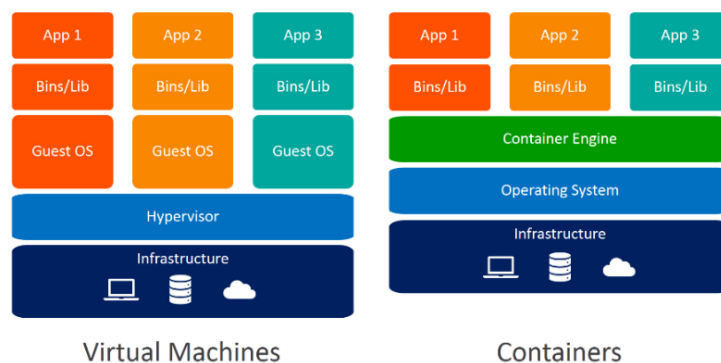


Figure 11 Virtual Machines Vs Containers Overview

3.3.2 Mininet

Mininet is a virtualization software that emulates enterprise topologies. Mininet Supports OpenFlow switches, controllers and virtual hosts to design highly flexible topologies with software defined networking. Mininet is a process-based virtualization application that can manage up to 4096 hosts with a single OS kernel . Mininet can be used in conjunction with the Ryu controller to emulate a three layer canonical tree commonly found in enterprise networks . This can be seen in Figure 12 . The developed controller/framework can be placed in a production network with little setup if tested using Mininet's virtualized topologies. To achieve this Mininet is passed a remote controller on a given port to integrate with Ryu. This is similar to DIPA deployment within live systems.

4 System Architecture

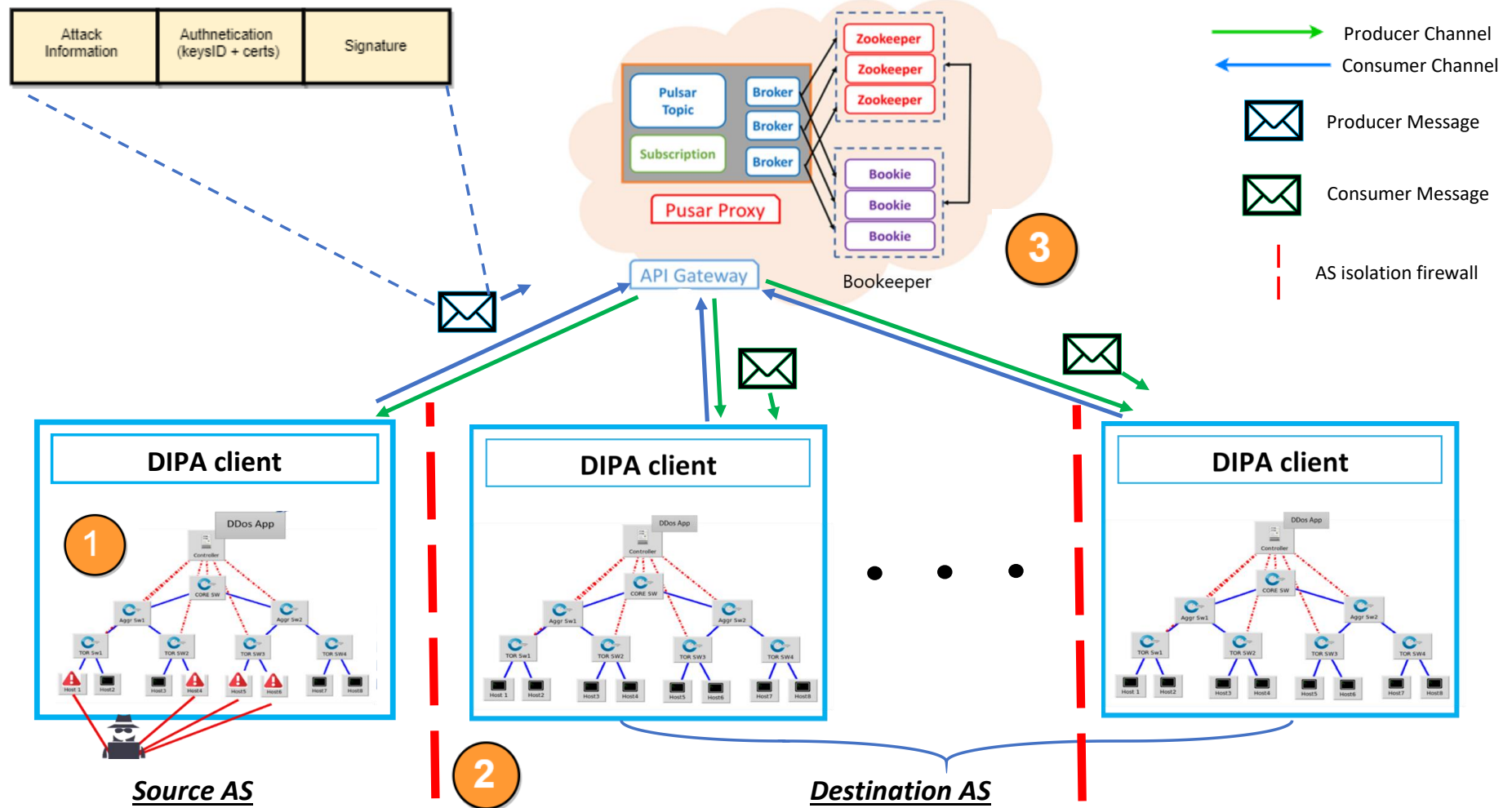


Figure 12 Overview of DIPA system architecture

4.1 Virtual Components

4.1.1 Control Plane Architecture

With the relevant research in place, a conceptual architecture has been designed and tested. The design provided in Figure 12 is divided into three main sections ; the virtual control/data plane , VM connections and isolation and the distributed pulsar architecture. Each piece is critically analysed, explaining decisions made through the process regarding the design. Each virtual machine within Figure 12 is both a source and destination domain. In which all domains effectively detect , produce and consume malicious alerts.

This section analyses point 1 within Figure 12 ; the control/data plane, crafted using Mininet. The proposed LAN architecture is based on the typical ‘fat tree’ / 3-layer canonical enterprise network topology. This topology utilises three layers of switches; core , aggregate and top of rack (TOR). This project aims to detect botnet attacks occurring at the edge of the network i.e. the TOR switches. Defence mechanisms must be integrated at these switches to prevent further propagation of malware to the upper switches . Enterprise SD-WANs employ a range of network topology formations and sizes. For this reason, DIPA installs the flow rule across all datapaths/switches. Further development is required to distinguish TOR switches to

The fat tree topology formation is constructed using the fattree.py script. This file builds each of these layers , linking adjacent switches using the code shown in Figure 13. Figure 13 links the core and aggregate switches with a bandwidth of 10Mbps , this is represented by bw_c2a . Further links between aggregate and TOR , and TOR and hosts are represented by bw_a2e and bw_h2a. This script bridges each of the OpenFlow switches to use OpenFlow v1.3 before connecting to a remote controller on 127.0.0.1:6633. Commented scripts can be accessed through the GitHub URL provided in Appendix 2 .

```

78 def createLink(self, bw_c2a=10, bw_a2e=5, bw_h2a=2.5):
79     end = self.pod/2
80     for x in xrange(0, self.iAggLayerSwitch, end):
81         for i in xrange(0, end):
82             for j in xrange(0, end):
83                 self.addLink(self.CoreSwitchList[i*end+j],
84                             self.AggSwitchList[x+i], bw=bw_c2a)
85 
```

Figure 13 Mininet Script : Create Link fat tree links

4.1.2 Virtual Machine Connection

The virtualised system in Figure 12 is a proof of concept of DIPAs proposed system model. This section analyses the domain isolation between source and destination domains represented at point 2 within Figure 12. Due to the geographically distributed nature of recent botnet attacks a collection of detached autonomous systems were implemented. The prototype uses three cloned virtual machines , running the same algorithm and local environments. Each domain is configured with two adapters to communicate with the pulsar proxy. Each VM use a configured Host-Only interface (virtualized network provided by the Hypervisor—VirtualBox) . The second interface is a Network address translation interface to add a gateway for Mininet hosts to access online resources.

To ensure exclusive full duplex communications between each domain and the pulsar proxy, routing was configured to pass data to its corresponding interface on the pulsar proxy cluster. The proxy uses is set to use an IP on the same subnet as each LAN. An example of one of the domains is shown in Figure 14.

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
192.168.42.0	192.168.42.1	255.255.255.0	U	0	0	0	eth2
192.168.43.0	192.168.43.1	255.255.255.0	U	0	0	0	eth3
192.168.44.0	192.168.44.1	255.255.255.0	U	0	0	0	eth4

Figure 14 Routing table entry on the pulsar proxy

‘Blackhole filtering’ was implemented using Iptables to block unwanted traffic between each interface. To further protect against VM communication a series of security groups were configured on the pulsar cluster to act as an access control list. These security groups were configured to block all traffic from external parties except on port 6651. Port 6651 is the broker service port used for TLS connections. For security purpose eth1 is not detailed to prevent malicious use of the cluster.

This ensured that the emulated LANs can only communicate through the pulsar proxy , exclusively using the pulsar port only . Connections from this port are authenticated within the brokers , this ensures attackers can send illegitimate commands over this port. This is achieved by setting the following commands in quaggas zebra.conf. Quagga is a networking routing suite that communicates routing updates using the zebra daemon. Iptables commands are used within the zebra.conf to prevent communication between interfaces.

```
iptables -A FORWARD -i eth2 -o eth3 -j DROP
```

4.1.3 Timing challenges

A series of challenges were encountered throughout the duration of this project. This in turn required continual assessment and management refactoring. The primary issue risen was clock drift between each virtual Machine. Even when clocks are initially set accurately to the same Network Time protocol (NTP) servers, the internal real clocks time count at different rates, diverging over time. Clock skew takes on more complexity when used in distributed systems that require several computers to utilize the same global reference time. Without time synchronization, there are credibility issues around the time results gathered in the performances test in section 6.3. These issues are common in distributed service-oriented architectures (SOA) such as DIPA.

To overcome clock drift between the pulsar proxy and SD-WAN, a combination of NTP and the Precision Time Protocol (PTP) were used. NTP is primarily designed to offer synchronization over the internet using unicast. NTP is described as a client-server model, in which each virtual machine will broadcast to specified NTP servers within a given region. As the AWS clusters are based in Oregon, (UTC-6) was used for each virtual machine. Four Oregon NTP servers were used with a maximum slew rate of 500 parts-per-million (PPM) by the UNIX kernel. This is edited within NTP's drift file to reinitialize the LANs time every 500ms.

As the Publish latency between a LAN and the pulsar proxy was >200ms there was discrepancies in results, producing negative latencies. To emend this issue a hierarchical master slave configuration known as PTP was used. A master clock, known as a transparent clock (TC) determines the clock drift offset between each LAN. This offset is added to the original time clock to synchronize each slave with the master reference time as shown in Figure 15. A Linux time-based job scheduler called cronjob was set up to periodically sync master and slaves clocks. This produced results of $\pm 50\text{ms}$ of the master reference point.

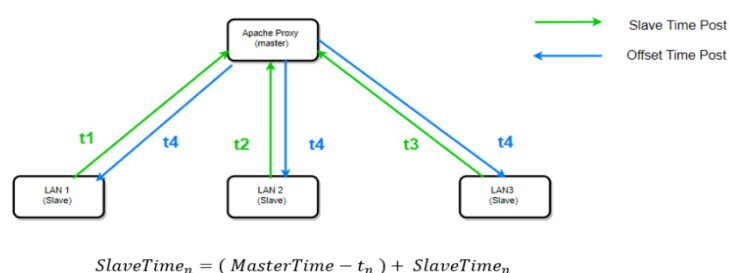


Figure 15 PTP Time Synchronization

4.2 Pulsar architecture

4.2.1 Pulsar Architecture

The prime focus of this research is the collaborative piece provided by Apache Pulsar's Publish subscribe framework . With the relevant research collected on the taxonomy of collaborative architecture , a distributed architecture was chosen. This is visualised in Figure 12 as point 3. The architecture was chosen for its promising abilities to detect and mitigate against new advancements in highly distributed attacks .To achieve this the pulsar frameworks was deployed on a collection of instances within the cloud.

Clusters consists of a broker, bookie and zookeeper node which are all connected to a proxy point . The distribution of the nodes prevents the system from performance bottlenecks and single points of failures. A single proxy point and an API gateway are used for developmental control to execute commands over each cluster , this proxy point can also be distributed in future development . Attacks of these points would compromise system availability. The attacker however would be unable to view or even eavesdrop messages travelling through these points. This is due to broker authentication explained in section 6.2.5.

The geo-replicated distributed architecture of the pulsar framework provides a 'high availability deployment architecture [30]. Assuming a cluster or a single node experience hardware failure , connection issues or is compromised by an outsider attack , the p remaining components will still communicate and function as normal. The attacker would have to attack the entire systems bookie, zookeeper or broker nodes to stunt the systems availability. DIPA utilizes elastic load balancing EC2 instances to distribute incoming communications and actions across multiple nodes. This elastic load balancer is placed in high availability mode, but this system can also be placed with automatic scaling to allow volume/storage growth as bookie nodes grow.

The following architecture is deployed using terraform . Terraform is an Infrastructure as code provisioning tool used to create the necessary resources . Configuration for each instance is setup in the terraform.tfvars. This details security groups (allowed connections), Virtual private cloud (VPC) resources ,instances size and CIDR subnetting , to allow communications to these nodes from the remote LAN testbeds deployed on the virtual machines. With the given resources available , pulsar is installed using ansible ; a server

automation tool . This sets each node up with their appropriate services in the correct locations . The `deploy-pulsar.yaml` ansible script automates pulsar features such as message deduplication on each node .

4.2.2 Design Decisions and Challenges

Designing a complete collaborative architecture required deep understanding of existing weaknesses and system exploits. In the process , a multitude of challenges arise .After extensive research , industry have suggested that distributed P2P architectures can greatly benefit prevention systems against the evolution of DDoS botnet attacks [10]. Decoupling of monitoring units prevent performance bottlenecks and SPoF however this makes the system harder to construct a global view of the protected network.

From this a hybrid unstructured CID can be integrated such as NetBiotic to introduce a distributed architecture with a Decentralized section to retain a global view against small cumulative attacks. As pulsar is the central focus of this project , a hybrid architecture can be integrated within future devlopment.

The AWS instance sizes were chosen to mimic the same test environment as Streamlio [31] . Streamlio are the original creators of the pulsar framework . This organisation has conducted a series of benchmark tests on pulsar to test its throughput under a series of test conditions . Using the same deployed environment , Streamlio recorded that the chosen hardware could manage ~220K messages per second before experiencing performance issues. Message guarantee was recorded to be within the 99.9th percentile under this load. This architecture was replicated to prevent repeated benchmark testing. This was also chosen as the current hardware was unable to produce more than ~1k messages per second; thus accurate benchmark performance could not be recorded. This is explained in further detail within section 6.

Under developmental conditions with low traffic production across the cluster this required the max heap memory size used by the bookie nodes to be lowered. The previously set heap size of 20G was too large for within development. For large heaps >5GB when the pulsar application is paused or stopped this cause the content management system (CMS) garbage collector to stall for several minutes during the initial remarking phase.

5 System Realisation / Implementation

5.1 Logical View

This section discusses the design of DIPA's flow-based proposal. This model detects at a source domain before alerting neighbouring destination domains to employ their defence mechanisms. This system is comprised of three critical phases: detection, communication and mitigation. The detection phase is responsible for classification of botnet attacks through Mirai feature analysis. This utilises custom information entropy algorithms to determine abnormal unidirectional telnet traffic and loading traffic. Upon receiving malicious information classification, the source domain protects itself before sending alerts to neighbouring domains through pulsar; this is the communication phase. The final phase , mitigation , installs a 'pre-protective' flow entry onto all of the OpenFlow v Switches. This flow entry contains specific flow details from the parsed pulsar alert.

Figure 16 visualizes the system's conceptual model. This prototype is partitioned into two main processes. Upon botnet classification, the information updates the topic stored in the pulsar bookies. In a separate thread, the consumer will asynchronously receive updates to this topic. DIPA employs round-robin partitioning to ensure the nearest neighbours are protected . This partitioning scheme also prevent the source domains from consuming its own produced message.

This section evaluates the most notable snippets of code to explain DIPAs functionality across each stage of botnet protection. Python was used to program and develop the ryu controller. The controller inherits class functions from the simpleSwitch13 example provided by ryu. This file processes basic traffic routing across a given network. Pulsar deployment files are constructed using YAML and terraform specific formats i.e tfvars and tfstate.

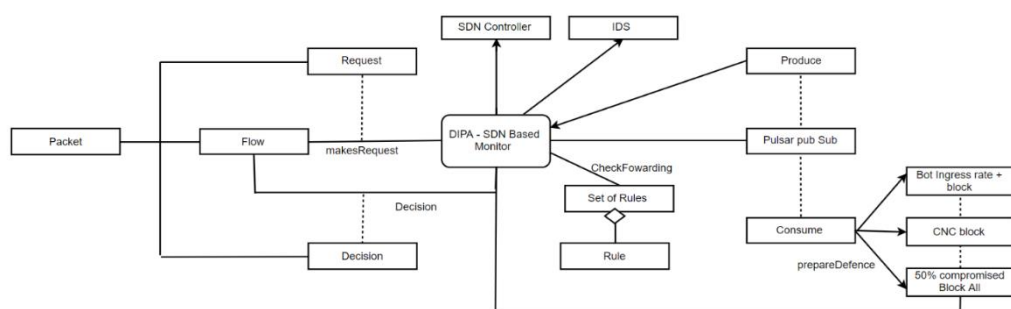


Figure 16 DIPA's conceptual system model

5.2 Coarse Grain NIDS

This section analyses design decisions made so construct DIPAs proof of concept. Botnet classification and timing functions are used to test system viability . The results are compared in relation to success metric and goals set in section 1.4. Figure 17 details an overview of the controller basic functionality. Upon receiving a new packet at the switch , the information writes a new rule if unknown or classifies the packet for alert correlation.

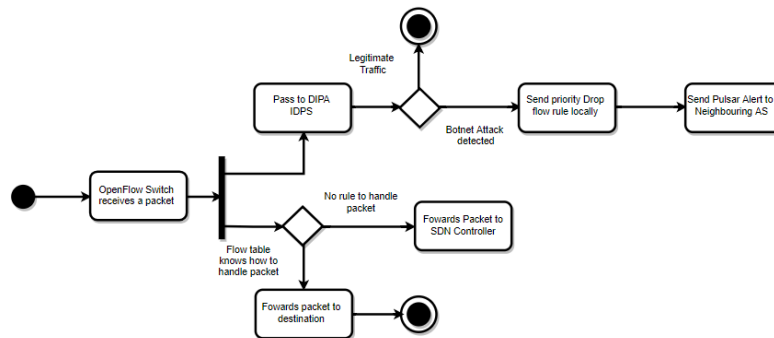


Figure 17 Coarse grain NID data flow diagram

5.2.1 Detection Functionality

In the event of a packet arriving at an OpenFlow switch that has no forwarding flow rule , the switch will forward the packet to the controller. To achieve this simpleSwitch13 uses proactive flow entries in the switch upon topology setup. Ryu subsequently parses the packet into a series of packet fields including protocol , ports , size . This is achieved using the getProtocol function shown in Figure 18. In the event of a packet or a malicious packet alert , this function is called to installed a flow Match.

OpenFlow uses the decimal notation to represent protocol types, set in RFC 793 [32]. This functions applies the appropriate packet fields to the flow match to determine the packet type and install a given flow rule. GetProtocol processes TCP,UDP and ICMP packets. Unknown packets such as ARP request are dropped at the controller as this is out of the project scope and unnecessary for botnet classification.

```

149 def getProtocol(self, pkt, parser, in_port, dst,src,
protoTrig,collabTrig):
163     if pkt_ipv4:
177     if protocol==6 or protoTrig=="6":
182         if port==23 or collabTrig=="Telnet":
183             return "Telnet",6, parser.OFPMatch(in_port=in_port,
                ipv4_dst=dst,ipv4_src=src, eth_type=ether.ETH_TYPE_IP,
                ip_proto=6, tcp_dst=23)
195     return "Unknown", 10, parser.OFPMatch(in_port=in_port)

```

Figure 18 deploy_controller/DIPA-Controller.py : GetProtocol Function

This function alerts DIPA's classifier of telnet traffic flows passing through the network to allow the system to effectively monitor these flows. protoTrig and CollabTrig are used on the occurrence of a malicious packet alert from a neighbouring VM. The pulsar message contains a protocol number within its message payload. This alerts the destination domains of the protocol type .

In the process of system design , a series of challenges emerged. When integrating the DIPA client into the NIDS , the algorithm experience data conflicts. The global interpreter lock in python is a mutex that prevents access to python objects. This prevents multiple python bytecodes from executing simultaneously. To resolve this issue , semaphores were used to share resources between the pulsar consumer and the OpenFlow statistics requester. The DIPA prototype executes both functions on the same process, for future development , abstraction of these processes is required.

As seen in Figure 19 this monitor will try acquire the lock every 0.5 seconds , provided the pulsar thread is not in use. A semaphore check of 0.5s was chosen as it took the system ~300ms to process and write 500 pulsar alerts. The algorithm will request port flow statistics on each Datapath/switch before calling the mirai_checker function to classify the attack. In the event of malicious traffic , mirai_checker will send a series of alerts in respect to its suspected device. Information exchange is evaluated in further detail in section 5.2.1. Each monitor/thread will release the semaphore to allow the other function be processed.

```

308 def _monitor(self):
309     while True:
310         while not sem.acquire(blocking=False):
311             time.sleep(0.5)
312         else:
313             for dp in self.datapaths.values():
314                 self._request_stats(dp)
315                 self.mirai_checker(self)
316             sem.release()
317

```

Figure 19 deploy_controller/DIPA-Controller.py : Monitor Threads

To classify a botnet attack a selection of features have been analysed from the CTU13 dataset provided by stratosphere [33]. Common Mirai attributes are as follows:

- 1) Loading Traffic on port 48101
- 2) Biased Telnet Traffic from a CNC
- 3) Port Scan on open telnet ports

As this conceptual model focuses on the early phases of the botnet attack , these main botnet features have been analysed and integrated into the coarse grain NIDS. DIPA achieves this through a suspicions list and a suspected blacklist. This algorithm checks for traffic using port 48101 , the original port used by AnnaSenpai [34] . Successive botnets such as HoHo or Turbo , still use this same port number. Upon detecting 48101 traffic the system places the source IP under surveillance by appending it to a suspicions dict.

To measure biased telnet traffic , a dictionary of all telnet flows is captured for each source IP. The traffic dictionary uses key pairs to match the source IP with all telnet packet flows matching its IPV4 address. The CNC device is determined by the IPV4 address with the max number of telnet traffic . This function is only called if there are suspected bots within the system and if the given flow from the *statRequest* uses the telnet protocol. Suspicious bots are processed to suspected bots by calculating the telnet weighting between the bot and the suspected CNC IPV4 address. In the event that 80% of the telnet traffic between these devices originates from the CNC device, the suspicious bot is flagged and blacklisted. As shown on line 497 of Figure 20.

```

469 if(stat.match['udp_dst'] == 48101 and stat.match['udp_src'] == 48101):
470     if stat.match['ipv4_src'] not in bot_dict:
471         bot_dict[stat.match['ipv4_src']] = 0.0
472     if(stat.match['tcp_dst'] == 23):
473         traf_dict[stat.match['ipv4_src']] = {stat.match['ipv4_dst'] :
                                                stat.packet_count}
474     for keygo , valuego in traf_dict.items():
475         mean_dict[keygo] = sum(traf_dict[keygo].values())
476         self._cnc_ip = max(mean_dict.items(), key=operator.itemgetter(1))[0]
477         if stat.match['ipv4_src'] != CNC_IP:
478             bot_dict[stat.match['ipv4_src']] = (1 - (stat.packet_count /
                                                         (stat.packet_count + mean_dict[self._cnc_ip])))
497 self._suspected_bots = set((k for k,v in bot_dict.items() if v >= 0.8))

```

Figure 20 deploy_controller/DIPA-Controller.py : Mirai_checker - botnet Classification

Upon receiving a alert from a neighbouring domain , the consumer thread will asynchronous receive and decode the message. The message is split into multiple fields including ; device , protocol and source IPV4 address . Dependent on the device type , a different defence mechanism is employed. As seen in Figure 21 , if the alerts consumed relates to a suspected bot , the algorithm will apply quality of service (QoS) rate limiting.

Unlike policing that simply drops packets in excess of the given threshold , rate limiting shapes the traffic. This is used to deter the attacker and slow down the loading process. Limiting the ingress port bandwidth of the bots parental TOR switch. Mininet currently has two ports for each switch grouping ingress and outgress traffic . Rate limiting is placed on the ingress switch , thus limiting legitimate traffic also. In live enterprise systems , these OpenFlow hardware switches can be configured to apply QoS on telnet traffic port 23. [35]

```

357 def _monitor2(self):
367     msg = consumer.receive(timeout_millis=100)
368     data = msg.data().decode('utf-8')
377     if device == "BOT":
378         attackerSwitch, attackerPort = self.getSwitch(ip_add)
380         ingressPolicingBurst = "ingress_policing_burst=1"
381         ingressPolicingRate = "ingress_policing_rate=0"
382         subprocess.call(["sudo", "ovs-vsctl", "set", "interface",
                           attackerSwitch + "-eth" + attackerPort,
                           ingressPolicingBurst])
383         subprocess.call(["sudo", "ovs-vsctl", "set", "interface",
                           attackerSwitch + "-eth" + attackerPort,
                           ingressPolicingRate])

```

Figure 21 deploy_controller/DIPA-Controller.py : Monitor2 - Ingress rate limiting

DIPA restricts this link between this the edge device and the TOR switch to 1kbps with zero tolerance over this limit. These edge switches are found using a predefined link matrix. This is statically set as a global variable at the beginning of the file. Each node is represented with its attached nodes through key-value pairs.

The TOR switches are found by passing the bots IPV4 address to the the getSwitch Function. This function traverses through each of the values stored in the dictionary to find the respective key matching the given IPV4 address. Once found this switch is returned to apply rate limiting. Functionality can be seen in Figure 22

```

138 self.portMaps = {"s1": ["s2", "s5"],
139                  "s2": ["s3", "s4"],
140                  "s3": ["10.0.0.1", "10.0.0.2"],
141                  "s4": ["10.0.0.3", "10.0.0.4"],
142                  "s5": ["s6", "s7"],
143                  "s6": ["10.0.0.5", "10.0.0.6"],
144                  "s7": ["10.0.0.7", "10.0.0.8"]}

441 def getSwitch(self, node):
442     for switch in self.portMaps:
443         if node in self.portMaps[switch]:
444             return switch, str(self.portMaps[switch].index(node) + 1)

```

Figure 22 *deploy_controller/DIPA-Controller.py : getSwitch + link mapping*

DIPA also supports the appropriate defence mitigation against the suspected command and control center. Upon packet parsing, a priority flow drop rule is applied against the given IPV4 address. This adds a drop flow rule to all of the switches in the neighbouring domains, preventing further telnet communications. This prevents the suspected CNC device from scanning and accessing new devices.

To apply this flow entry to all switches, the `add_flow` function is used as shown in Figure 23. The pulsar consumer threads pass an `OFPPFlowMatch` object with an empty actions list and a priority of 100. Packets hitting these switches will match against these higher priority rules at the start of the lookup process. This prevents performance bottlenecks with TCAM lookups. This is explained in further detail in section 6.2.4.

```

213 def add_flow(self, datapath, priority, match, actions, buffer_id=None):
220     mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                             priority=priority, match=match, instructions=inst,
                             hard_timeout=100, idle_timeout=60)

```

Figure 23 *deployment_controller/DIPA-Controller : Add flow to open vSwitches*

The final alert detection mechanism is DIPA's fallback scheme. This alert is received if a neighbouring domain is suspected to have more than 50% of its IP-enabled device enslaved by a CNC. If this occurs, this will stop all telnet traffic travelling across the network. Although this restricts legitimate telnet within the system, it prevents further propagation of the malware across the network. Unlike previous flow rules, this flow entry doesn't match against specific IPV4 address. In turn, dropping all packets using the telnet port. DIPA installs flow entries for port 2323 also. An alternate port used for telnet communications. Figure 24

```

410 mat1 = parser.OFPMatch(eth_type=ether.ETH_TYPE_IP, ip_proto=6,
                        tcp_dst=23)

```

Figure 24 *deployment_controller/DIPA-Controller : Drop all Telnet Flow Match*

5.1.3 Timing Algorithm

DIPA's protection time is measured by summing each link latency . The full formula to calculate this can be found in section 6.3.1 . To measure the SDN control plane and NIDS processing time , a perf counter is used. A perf counter measures relative time with no defined relationship to real-world time . This counter measures the round trip time of the request stats function , ending the counter after all information has been processed by the NIDS's classifier. The control plane latency is represented by *timeRequest* and *timeReply* in Figure 25. To mark measure the link between the source domain and the pulsar proxy a timestamp is used before sending an update to the pulsar *time* topic.

```

499 self.timeReply = time.perf_counter()
500 nowTime = datetime.now()
501 producerTime.send("{}@{}".format(self.timeReply - self.timeRequest,
                                     nowTime, event_timestamp=True)
                                     ).encode('utf-8'))

```

Figure 25 *deployed_controller/DIPA-Controller.py : Control plan timer + production*

DIPA measures the control plane time using python datetime module , this calculate POSIX time to a precision of 1 microsecond. To measure the producer and consumer latency , pulsar provide these through event and publish timestamps . These functions measure time within a millisecond accuracy . Protection time accuracy was thus restricted to a precision of 1 millisecond .

To calculate the time, a centralized instance (proxy instance) was used to collect all the data from each domain. For each domain , the *threaded_consumer.py* file receives incoming updates to the time topic .As seen in Figure 26, *nowtime* is used to timestamp the time this message is received. This is used to measure the link latency between the source domain and the pulsar proxy. Time measurement equations can be viewed in section 6.3.1. Event and publish time are gathered using Prometheus ; a monitoring metric tool . Further explanation noted within section 6.

```

47 def receiver(self,vm_index,num_subnets):
51 msg.receive_async(timeout_millis=10000)
56 nowTime = datetime.now()
57 latency = nowTime - datetime.strptime(msg_arr[1],
                                     "%Y-%m-%d %H:%M:%S.%f")
58 self.time_details[vm_index, 0] = msg_arr[0]
59 self.time_details[vm_index, 1] = latency.total_seconds()

```

Figure 26 *global_view_and_timing/threaded_consumer.py : Time measurements*

5.2 Pulsar Features

An efficient and holistic collaborative protection system that fulfils the requirements specified in section 2.2.2 must be designed carefully. This section suggests a series of pulsar feature implementations to improve the prototype design . To structure the solution space , these features are divided their respective sections , discussing decisions and challenge encountered.

5.2.1 Data Privacy and Membership management

Membership management is responsible for ensuring connectivity between each monitor and the LANs that make the requests. Dependent on the user's preference ,the pulsar architecture can be configured with overlays that will allow for dynamic inclusion or exclusion of marked domains. This is done through namespace permissions and token based access. Pulsar can be configured to restrict read/write (consume/produce) access to specified domains at the topic level. Each Domain is subsequently placed onto a different topic within the namespace to allow each topic within the domain to have different configurable permissions. Domains are configured to subscribe to a namespace. This receives all topic updates under that given namespace.

As an alert is pushed to the pulsar cluster , it is stored in a persistent disk in the bookies. Subscribed domains to successfully receive the message and acknowledgement the message will remove the outstanding alerts from the disks. This introduces a data race between each of the domains. The variance in consumption can differ greatly , thus utilizing the same topic for each domains results in several domains not receiving an alert. To solve this the *availablePermits* flag within pulsar can be set to the number of domains used. This however requires manual configuration changes as new networks are added. This is further reinforcement to use different topics for each domain , subscribing to the entire namespace to prevent the data race.

Privacy protection is paramount within collaborative architecture . As LANs employ different security levels , the system must accommodate for system wide failure; (i.e if an attacker compromises an entire LAN). To ensure data privacy is upheld ,a collection of additional security mechanism detailed in section 6 were implemented . As the early phase of the Mirai botnet use a single telnet communication flow to random IPs , sensitive information is

not required to be transmitted between neighbouring domains. DIPA only requires the domain to alert the neighbouring cluster of the suspected malicious IP , protocol used , port used (all sorted in keyss) and the predicted device (i.e Bot or CNC).

```
333 producer.send_async(("{}@{}".format(keyss, "BOT",
334                               replication_clusters='us-west-2a',
335                               partiton_key=RoundRobinPartition,
336                               event_timestamp=True)).encode('utf-8'))
```

If a neighbouring system has been compromised , the attacker will only retrieve information regarding its own attack . DIPA sends no topology information of the source domain which the attacker could utilise.

5.2.2 Geo-Replication

Unlike other messaging frameworks , Pulsar offers geo-replication. Geo-replication is a common feature in data centers to distribute data across geographically distributed data networks. This is intended to improve the response time between applications, in this case LAN publish and event times captured. Large organisations often provision networks in multiple data centers worldwide.

Enabling replication of message data between Pulsar clusters allow enterprise networks to access distributed insider attack information within their SD-WAN. In the event of a network failure as shown in Figure 28 , the data is replicated and stored in neighbouring clusters upholding the system availability. Geo-replication was integrated to counteract Mirai geographically pervasive nature visualized in Annex 5.2 ; integrating well with data center organisations.

Asynchronous Geo-replication has been implemented in case of system/broker failure within a given cluster level. Pulsar data segmentation , splits the message into partitions and stores data across each of the brokers. With geo-replication , these segments are replicated across each cluster in the event of a compromised broker. This is used as a disaster recovery technique in multi-faceted enterprise software to provide additional redundancy in the case of a bookie or zookeeper failure. As messages are produced to a pulsar topic , this malicious information persists on the local cluster before replicating on remote clusters. Permissions can be enabled per-property (tenant level) using selective replication. To enable geo-replication the following commands are used. (full script of all pulsar features can be found

within the ansible script 'deploy-pulsar.yaml') . geo-Replication introduces a security–resource trade-off.

```

121     shell: |
122         bin/pulsar-admin namespaces set-clusters public/1 \
123             --cluster {{ cluster_name }} \
124             --zookeeper localhost:2181 \
125             --configuration-store localhost:2181 \
126             --web-service-url {{ http_url }} \
127             --broker-service-url {{ service_url }}
128     - name: Geo-Replication clusters allowed
129     shell: |
130         bin/pulsar-admin tenants create Domain1 \
131             --admin-roles admin
132             --allowed-cluster 1 , 2 , 3

```

Figure 27 Deploy-Pulsar.yaml : Geo-replication on bookies

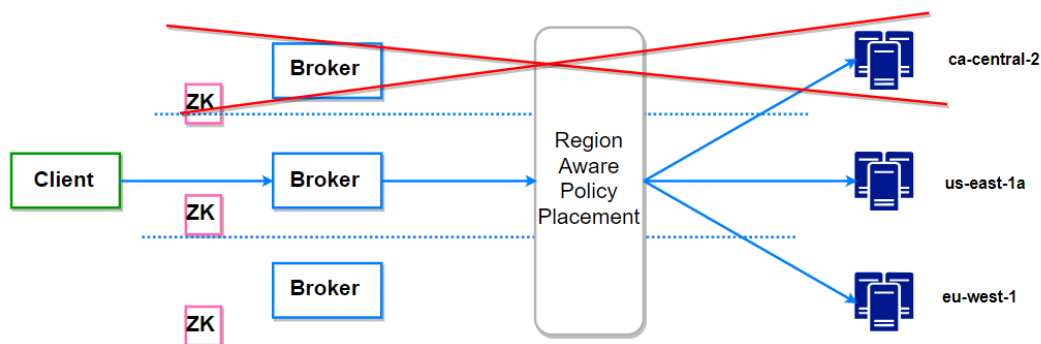


Figure 28 Geo-replication of example enterprise Data center

5.2.3 Async Communications

As attacks are classified using the coarse grain NID detailed in section 5.1.2 , the aggregated data needs to be efficiently distributed. Data is disseminated using asynchronous communications. DIPA uses multi-threading to concurrently requests stats from each OpenFlow vSwitch whilst receiving alerts from the neighbouring domains .. Switching between these threads is achieved using semaphores. Semaphores lock the pulsar consumer thread whilst requesting for new switch statistics.

Synchronous messaging paradigms introduces explicit barriers within the algorithm , this in turn cause system deadlock. DIPA utilises asynchronous communication , to accommodate for scenarios in which a LAN is unable to send to or receive from the pulsar framework . Send and receive communications are processed in the background whilst the algorithm processes the flow statistics received. If malicious information is detected the DIPA client will produce the message . Neighbouring domains will finish processing their own flow statistics before passing the semaphore lock to the pulsar consumer to check for alerts. This design decision was used to prevent neighbouring domains missing local insider attacks. This

introduces a slight processing latency to protection algorithm however is a necessary decision to cover this system exploit.

5.2.4 Message Deduplication

The proposed conceptual model considers system exploits to insider attacks in the event of a compromised system . To mitigate, DIPA employs Apache Pulsar's message deduplication . This ensures an 'effectively-once' messaging guarantee to prevent the attacker from flooding the pulsar framework. Unlike other messaging frameworks that employ a 'at-most-once' paradigm , pulsar ensure messages are processed and retained in the disk once for a statically set time. This is also known as producer idempotency.

Message retransmission and message call-backs have been removed from the DIPA framework. This is a design decision made to prevent a LAN from consuming pulsar processing power from continual retransmissions. In the event of an attack a LAN must send an alert to the pulsar framework and shouldn't continually try or wait for the message guarantee acknowledgment (ACK).

Alerts to other networks are secondary protection and each LAN should protect itself first. This unburdens LANs of responsibility .The pulsar proxy is then responsible for message delivery. Message deduplication can be enabled at both the topic and namespace level. DIPA has set deduplication at the namespace level for each domain. Deduplication uses the same idle_timeout and hard_timeout times as the OpenFlow vSwitches to ensure these ipv4 source remain only in the pulsar disks for the same time they remain in the flow tables.

Configuration to integrate this is shown below (full configuration on Github):

```
89 brokerDeduplicationEnabled=True
99 brokerDeduplicationEntriesInterval=1      # Producer Idempotency
103 brokerDeduplicationProducerInactivityTimeoutMinutes=1 # same as idle timeout
```

Figure 30 templates/broker.conf - Deduplication flags

```
producer = client.create_producer('persistent://public/standalone/1/mirai',
                                  send_timeout_millis=0,
                                  compression_type=pulsar.CompressionType.ZLib,
                                  producer_name='VM-1 Prod',
                                  max_pending_messages_across_partitions=500000,
                                  block_if_queue_full=False,
                                  message_routing_mode=PartitionsRoutingMode \
                                      .RoundRobinDistribution)
```

Figure 29 DIPA-Controller.py - Hard Timeout = send_timeout

5.2.5 Message retention and topic compaction

In the event of a botnet attack , the source domains will produce alerts to each of the neighbouring domains/namespaces. In the event of a domain failure or compromised LAN , these unacknowledged messages will remain unacknowledged in the disk. Attackers can exploits this, exhausting bookie memory with unused malicious message permutations. This will in turn affect performance and protection speed. DIPA utilises pulsar's message retention and topic compaction features to mitigate this.

DIPA retains all unacknowledged and acknowledged messages on the disk for 100s to prevent deduplication. Unacknowledged messages that exceed the 10G store space set on the disk are passed to a backlog (backlog -quota) . This appends a time to live (TTL) to each of these messages before discarding the message from the system . Passing unacknowledged messages to the backlogs prevent unacknowledged messages from timing out due to the 'idle_timeout' feature provided by the message deduplication .

When a LAN subscribes to the Mirai topic ,it is only interested in the latest information alerts send . This LAN doesn't want to waste resources and time reading through the whole message backlog for this information. DIPA utilizes pulsars topic compaction , which prunes older topics in the backlog , allowing for faster reads in the topic history. This refers to messages in the backlog that have already been acknowledged but still have their TTL. This prioritises packets that haven't been acknowledged in the backlog. This has been set to 10 gigabyte in DIPA. Message deduplication and topic compaction are currently only supported in the Java language , thus the following command-line API commands were executed on the pulsar proxy : These commands can be found in the deploy-pulsar.yaml

```

133 - name: Namespace Retention
134   shell: |
135     bin/pulsar-admin namespaces set-retention public/standalone/1 \
136       --size 10G
137 - name: MSG TTL
138   shell: |
139     bin/pulsar-admin namespaces set-message-ttl public/standalone/1 \
140       --messageTTL 200
141 - name: Backlog
142   shell: |
143     bin/pulsar-admin namespaces set-backlog-quota public/standalone/1
144       --policy producer_request_hold
145 - name: Compaction
146   shell: |
147     bin/pulsar-admin namespaces set-compaction-threshold
148     public/standalone/1 --threshold 1G

```

Figure 31 Deploy-pulsar.yaml : Message Retention and Compaction

6 Experimental Testing

This section evaluates and presents comprehensive analysis of empirical data gathered via a series of functional and performance test. Results captured are used to justify pulsar as a viable solution in CIDPS. The following functional test set is used to determine pulsars limitations and suitability in IDPS. A coarse Grain DDoS app constructed via Ryu is used to control legitimate and malicious traffic flow within the network. An IDS is required to test the collaborative protection system response mechanism. However the primary focus of the following results are to test the defence mechanisms and strategies. All traffic and attack files are also supplied on the GitHub URL in Appendix 2.

6.1 Testbed Setup

The following section details the test conditions of the testbeds including ; testbed environments, attack configurations and topologies. Each testbeds uses a three layer canonical fat-tree explained in further detail in section 4.2.1. Test Topologies consist of 8 virtual hosts ; 1 CNC Master , 1 legitimate Client , 1 legitimate server and 5 victim IoT devices. The testbeds are split into two different networks i.e *Source* , *Destination* networks for testing, however each domain consumes and produces alerts.

The Source networks are VMs that produce benign traffic and attack traffic . For this legitimate traffic is passed between the legitimate client and server. Background traffic is represented by random flows in each switch.

The Destination networks are autonomous systems that use the same topology configuration but use only legitimate traffic. The destination host fulfils requests sent from the source networks subsequently blocking malicious traffic. Attack traffic on destination domains is generated five seconds after the original attack on the source network.

6.1.1 Virtual Machine Testbed

The test cases employ three identical virtual machines to simulate isolated autonomous systems. Each server VMs use an Ubuntu 18.4 ISO image consisting of 16GB memory and 12GB of storage. Each domain shares an Intel Core i7-7500U 2.70GHz with 4 logical cores. Each VM is installed with Mininet 2.2.1 and Ryu 4.30 as the controller. Finally each VM is installed with Apache Pulsar client 2.3.0 to link each testbed to the Pulsar brokers. Further

Software Listing and parameters can be viewed in Appendix 5. The following Testbeds are isolated into separate domains using quagga and iptables routing. This ensures domains can only communicate through pulsar message alerts.

6.1.2 Pulsar framework

A distributed architecture described in section 4.3.3 is used for the pulsar pub sub framework. Apache pulsar resides on a series of AWS clusters containing nine EC2 instances. Each Cluster is region-specific to Oregon (us-west-2a). Oregon was chosen due to its instance and Amazon machine interface (AMI) availability. To accommodate each VM use UTC-6 (Oregon Mountain daylight time) to ensure accurate timing. Deployment options are described in section 4.3.1. Each VM connect to pulsar through a service URL. The AMI used is ami-9fa343e7 running Red Hat Enterprise Linux (RHEL) 7.4 . Further Resources used by this configuration are :

- 1) 3 small VMS for Zookeeper using t2.small instances
- 2) 3 larger VMS for Bookkeeper bookies using i3.xlarge instances
- 3) 2 larger VMs for the Pulsar Brokers using c5.2xlarge instances
- 4) 1 larger VM for the Pulsar proxy using a c5.2xlarge instance
- 5) EC2 Security Group and Virtual private cloud (VPC)
- 6) API Gateway (Allow external connections)

6.1.3 Legitimate Benign Traffic

A series of tools are used to generate legitimate traffic on each testbed. A client and server were used for legitimate traffic. These scripts send legitimate telnet traffic to test false positives . Scapy was used to build a series of pcaps for each host. Scapy is python based interactive packet manipulation tool for crafting different types of packets. Each pcap was replayed on each host after Mininet has remotely connected to each controller. Each replayed pcap was only able to hit the host TOR switch. This is sufficient ,as background traffic is equivalent to number of flows received by Ryu's *statsRequest* (its doesn't matter of flow placement). This project only requires TOR / edge switches to be protected thus topology size, shape etc is proportional to the sum number of flows the controller receives.

6.1.4 Attack Traffic + tools

Due to the Mirai's harmful nature and difficulty producing an air-gapped network, the Mirai botnet scanning phase is emulated. This was decided as accurate Mirai classification is not the focal point of this project. Thus a variety of attack tools were used to emulate the early phase of a Mirai botnet attack. Hping3 was used within the initial functional test set to ensure malicious flow information was captured on port 23 and 2323 (telnet). An example of the hping3 commands used are as follows:

Hping3 -S -i u100000 -c 10000 -p 23 10.0.0.<1-7>

-S = Syn Packet

-C = Max number of packets

-I = Packet_rate

-p = Destination port

To further test system reliability the command-and-control.pcap file provided by Ixia ATI [36]. The pcap presents simple botnet CNC traffic between a CNC device and one host. TcpRewrite was used on the supplied pcap to rewrite destination and source IPV4 addresses before using TcpReplay two attacking hosts.

For performance test a highly configurable Botnet simulator known as BoNeSi was used [37]. BoNeSi is a network traffic generator that includes a simple TCP-stack to handle TCP connections in promiscuous mode. For the performance test set, BoNeSi was configured to use port number 23 with a send rate of 500 packets per second (pps). The source IP addresses bot lists are stored in a text file. Performance test Cases used a range of bots/ source-ip address ranging from 1-500 bots. The following command was used on the CNC communicating with a target server on a neighbouring host:

bonesi -ips <filename i.e 500bots.txt> -p 23 -send_rate 500 -max_bots=500 -url=http://10.0.0.<1-7> -d eth1

6.1.5 Monitoring Tools

A variety of test methods were used to ensure examine the results for both functional and performance tests. Prometheus was used to monitor and snapshot metrics of the pulsar architecture. Prometheus was used to capture event and publish time. These two features are used to calculate the produce and consume time. Pulsar measures event and publish latency to millisecond accuracy. To ensure system wide timing accuracy this project uses Python3's datetime module to calculate control plan time with microsecond precision. The

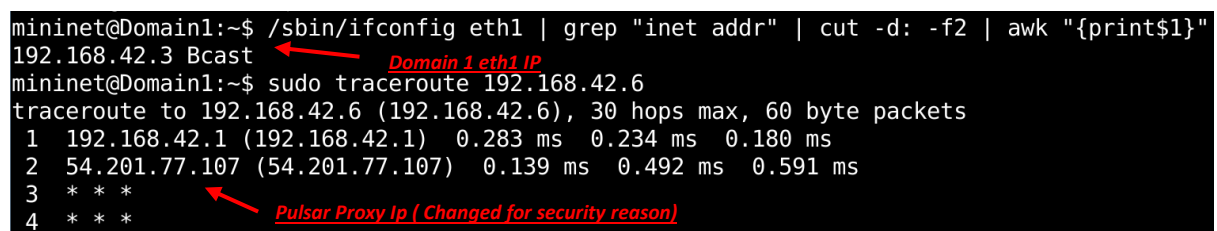
Timing Function functionality mentioned in section 5.1 was used calculate protection time from the source hosts read time to the destination hosts write protection time. Each section below explains methodologies used to capture results.

6.2 Functional Test Set

The Following section explores different functionalities of the system . This is used to ensure system correctness but also to detail the limitations of the system under different edge casing scenarios. These test scenarios are based off complications the system may exhibit in a production environment. This section is used to find the optimal parameters using the given hardware.

6.2.1 Reactive Collaborative protection

Due to the geographically distributed nature of recent botnet attacks a collection of detached autonomous systems were implemented using the system architecture detailed in section 4.2.2. To test this a the traceroute command was used to show each hop on the network. Figure 32 details the communication hops between source (192.168.42.3) and destination (192.168.42.6) domains. The Source VM is able to communicate with the pulsar proxy on AWS , however the iptables stops any communication between interface eth2 and eth3.



```
mininet@Domain1:~$ /sbin/ifconfig eth1 | grep "inet addr" | cut -d: -f2 | awk "{print$1}"
192.168.42.3 Bcast
mininet@Domain1:~$ sudo traceroute 192.168.42.6
traceroute to 192.168.42.6 (192.168.42.6), 30 hops max, 60 byte packets
 1  192.168.42.1 (192.168.42.1)  0.283 ms  0.234 ms  0.180 ms
 2  54.201.77.107 (54.201.77.107)  0.139 ms  0.492 ms  0.591 ms
 3  * * *
 4  * * *
```

Figure 32 Isolated Domains – traceroute

With domain segregation implemented and tested the basic functionalities of pulsar producer and consume mechanisms were tested. This test case adopted Ryu’s Simple Switch 13 to send all flow entries captured by Domain 1 onto it’s a neighbouring destination VM . As shown in

Figure 33 through reactive flow rule programming , the controller mediates all new traffic parsing packet fields via OFCTLs API Function within the event Listener *StatsRequest*. Upon building a flow rule ,Pulsar send these fields to a listening neighbouring VM, cloning all flow

rules. This methodology provides the foundation for collaborative protection , allowing the controller to refine traffic exchange to malicious information. This can be seen in section 6.2.2. The replicated flow rule in Domain 3 currently has 0 n_packets. This occurs as no packets have hit this flow rule on the switch. This operates as our pre-emptive defence mechanism.

The image shows two Oracle VM VirtualBox windows. The top window is titled 'Domain-1 [Running] - Oracle VM VirtualBox' and contains a terminal window with the command 'sudo ovs-ofctl -O OpenFlow13 dump-flows s1'. The output shows three flow rules with various statistics. The bottom window is titled 'Domain-3 (Snapshot 1) [Running] - Oracle VM VirtualBox' and contains a terminal window with the command 'sudo ovs-ofctl -O OpenFlow13 dump-flows s1'. The output shows three flow rules, all with n_packets=0.

```
mininet@Domain1:~/ryu/ryu/app$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
  cookie=0x0, duration=187.308s, table=0, n_packets=3, n_bytes=238, priority=10,in_port=2,dl_dst=1e:11:8d:e5:0b:90 actions=output:1
  cookie=0x0, duration=187.306s, table=0, n_packets=2, n_bytes=140, priority=10,in_port=1,dl_dst=8a:18:ab:99:59:aa actions=output:2
  cookie=0x0, duration=194.133s, table=0, n_packets=3, n_bytes=182, priority=0 actions=CONTROLLER:65535
mininet@Domain1:~/ryu/ryu/app$

mininet@Domain2:~/ryu/ryu/app$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
  cookie=0x0, duration=30.984s, table=0, n_packets=0, n_bytes=0, priority=100,in_port=2,dl_dst=1e:11:8d:e5:0b:90 actions=output:1
  cookie=0x0, duration=30.984s, table=0, n_packets=0, n_bytes=0, priority=100,in_port=1,dl_dst=8a:18:ab:99:59:aa actions=output:2
  cookie=0x0, duration=170.523s, table=0, n_packets=0, n_bytes=0, priority=0, actions=CONTROLLER:65535
```

Figure 33 Reactive and Collaborative Flow rule

6.2.2 Network Intrusion Detection

In order to evaluate DIPA's defence mechanism ,a coarse grained detection NID was constructed .The detection engine can reside on both source and destination networks to allow bidirectional protection. The following algorithm is not restricted to virtualized testbeds and can be used hardware systems. Collaborative networks face ethical scrutiny over data privacy and protection . For this case pulsar will not exchange sensitive information such as hardware addresses. Test Case 6.2.1 was reimplemented to exchange only source IP addresses , and destination ports. If the neighbouring domain is compromised the attacker is only able to collect malicious ipv4 source addresses and a series of ports.

```
applying ingress for bot : 10.0.0.5
applying ingress for bot : 10.0.0.6
Flow Rule Write for suspected CNC : 10.0.0.4 on Switch 1
Flow Rule Write for suspected CNC : 10.0.0.4 on Switch 2
Flow Rule Write for suspected CNC : 10.0.0.4 on Switch 3
Flow Rule Write for suspected CNC : 10.0.0.4 on Switch 4
Flow Rule Write for suspected CNC : 10.0.0.4 on Switch 5
Flow Rule Write for suspected CNC : 10.0.0.4 on Switch 6
Flow Rule Write for suspected CNC : 10.0.0.4 on Switch 7
The CNC has enslaved 50.0 of the Network, ceasing all telnet on Switch 1
The CNC has enslaved 50.0 of the Network, ceasing all telnet on Switch 2
The CNC has enslaved 50.0 of the Network, ceasing all telnet on Switch 3
The CNC has enslaved 50.0 of the Network, ceasing all telnet on Switch 4
The CNC has enslaved 50.0 of the Network, ceasing all telnet on Switch 5
The CNC has enslaved 50.0 of the Network, ceasing all telnet on Switch 6
The CNC has enslaved 50.0 of the Network, ceasing all telnet on Switch 7
```

Figure 34 NID Algorithm Output

This section tests cases' functionality and classification decision are detailed in section 5.1.2. The following test consisted of emulating the early , loading and scanning phase of the Mirai botnet attack . The coarse grained NID has of four primary functions :

- 1) block 48101 on bots and CNC from malware loading
- 2) block unidirectional telnet from CNC on port 23 and 2323
- 3) If > 50% of the network is compromised cease all telnet comms
- 4) Applying ingress Policy Rate to slow traffic from bots to CNC

As presented in Figure 34 the algorithm successfully captured each point, however a series of functional tests were performed to ensure functionality of each defence mechanism.

The image shows two terminal windows from Oracle VM VirtualBox. The top window is titled 'Domain-2 (Snapshot 1) [Running] - Oracle VM VirtualBox' and shows the output of 'sudo ovs-ofctl -O OpenFlow13 dump-commands s1 | grep drop'. It lists several flow rules with actions like 'drop' for specific traffic. The bottom window is titled 'Domain-1 (Running) - Oracle VM VirtualBox' and shows the output of 'sudo ovs-ofctl -O OpenFlow13 dump-commands s1 | grep drop'. It also lists several flow rules with actions like 'drop' for specific traffic.

Figure 35 Functional Protection Flow rules

When the system detects anomalous behaviour it begins the mitigation phase. As seen in Figure 35 both source and destination domains installed the respective flow rules. Upon classification of botnet attack the algorithm will deduce the CNC through feature extraction of biased telnet connections (i.e most one-sided telnet communications in network). Figure 35 displays high priority rules on both source and destination to drop 48101 traffic , CNC and bot telnet traffic .To test packet drop , a series of hping3 commands were executed against these ports. All packets were successfully dropped at the switch. The initial two rules displayed at the head of the flow table of both local and neighbouring domain switches show the drop off all telnet traffic , as there is no specified ipv4 source.

To deter the attacker and bots from continual scanning , Open vSwitches Quality of Service (QoS) rate limiting was used. This reduces the bandwidth on the ingress ports between the CNC and each of the bots. This limits the 'commands' send by the CNC but also limits the

bandwidth from bots to other vulnerable IoT devices which are in the loading phase. This will slow the loading phase of botnet , reducing the malware spread.

As seen in Figure 36, at t1 the controller begins mitigation phase , rate limiting telnet and port 48101 traffic on a suspected bot/CNC . As network enslaved continues to grow , all telnet communications are block(t2). For visual representation this host only used telnet traffic , explaining why zero traffic is seen after t2. The defence mechanism proved ineffective against simultaneous attacks on neighbouring VMS. This current system implementation is effective for pre-emptive protection. Bandwidth was measured using speedometer. This displays ingress (top of Figure 36) and outgress (bottom of Figure 36) of data passing through the switch.

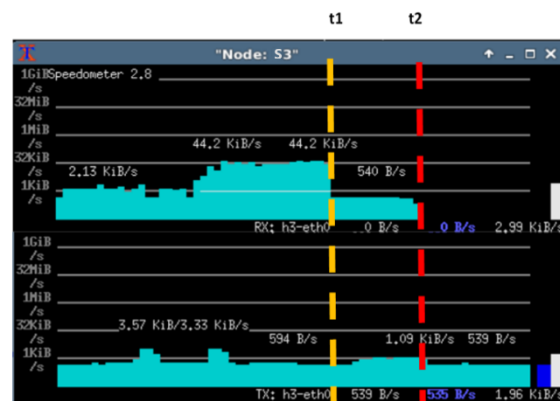


Figure 36 Bandwidth usage on switch ports

6.2.3 Optimal Polling rate and Timeouts

The performance of the proposed system depends on the payload size of the attack definition . This affects detection polling and processing time. Unlike Routers , the Ternary content-addressable memory (TCAM) used by OpenFlow switches is expensive ,and power hungry . TCAM is unique memory used to do lookups in a single clock cycle in a parallel fashion across multiple fields. TCAM lookups match explicit fields including 'don't care' bits, which acts as a wildcard bit when searching for flow rules within the switches. As show in

Figure 37 this can be used to quickly search through flow rules on each switch to pick out flows containing a specific field i.e tcp_dst=23 (telnet traffic).

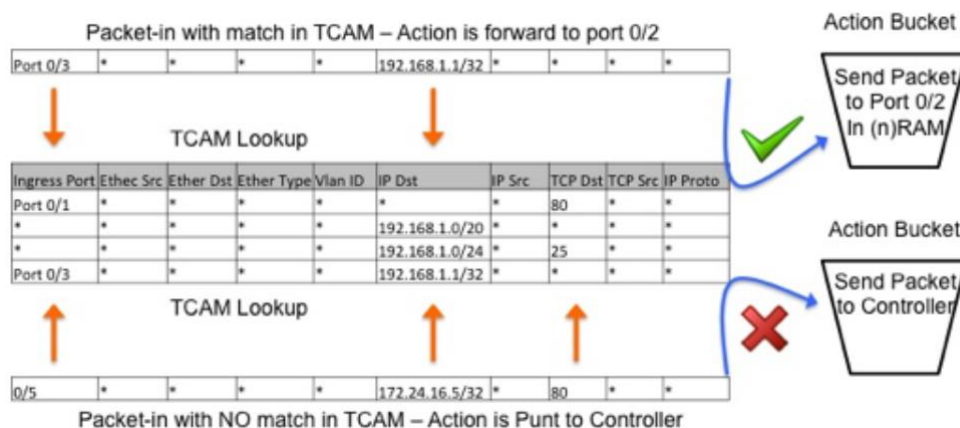


Figure 37 Ternary CAM OpenFlow operational lookup

In order to control detection granularity and efficiency, a hard and idle timeout mechanism is implemented. This is used to phase out 'spoofed' or unused flow table entries. This frees space for faster lookups in the detection phase reinforcing system resilience and detection availability.

Timeouts are explicitly added to all flows to prevent those devices whose sole purpose is to overwhelm the capacity of the OpenFlow switches. To ensure this doesn't hinder legitimate communication an idle timeout of 60s and hard timeout of 100s was chosen. This figure will scale dependent on hardware's silicon space but also traffic /domain size. The size of the domains is considered equivalent to the number of flows captured on each switch.

The detection performance also strongly depends on the bandwidth consumption between controller and switches. The time it takes to look up each matching flow and run botnet classification compared to the polling time of each *StatsRequest* is considered (i.e polling time > process time). If the *StatsRequest* polling time is less than the processing time, not all flows will be classified before receiving a new batch to check.

A series of stress tests were compiled to analyse the effects of polling rate on bandwidth. As shown in Figure 38 as the payload of the *StatsRequest* packets increase the amount of bandwidth consumed on the network exponentially rises. Notable change can be seen in lower polling rates (Table 4). For the final solution a polling rate of two seconds was chosen as it exhibited low bandwidth usage, similar to that of 3 seconds. Bwm-ng tool was used snapshot bandwidth usage. Full results can be viewed in Appendix 6.

Table 4 Effects of polling rate on network Bandwidth (Standard Deviation)

	0.5s Polling	1s Polling	2s Polling	3s Polling
100	29.56349	7.334848	1.923538	1.095445
200	29.36495	3.781534	1.732051	1.516575
500	58.11024	23.67066	6.534524	4.301163
1000	97.18899	32.01094	17.89693	9.859006
2000	83.44579	43.36243	9.833616	14.19507
3000	162.466	40.46974	36.94997	36.97296
5000	227.0998	108.735	47.25992	17.79607
7500	271.3619	145.1665	117.2327	107.5346

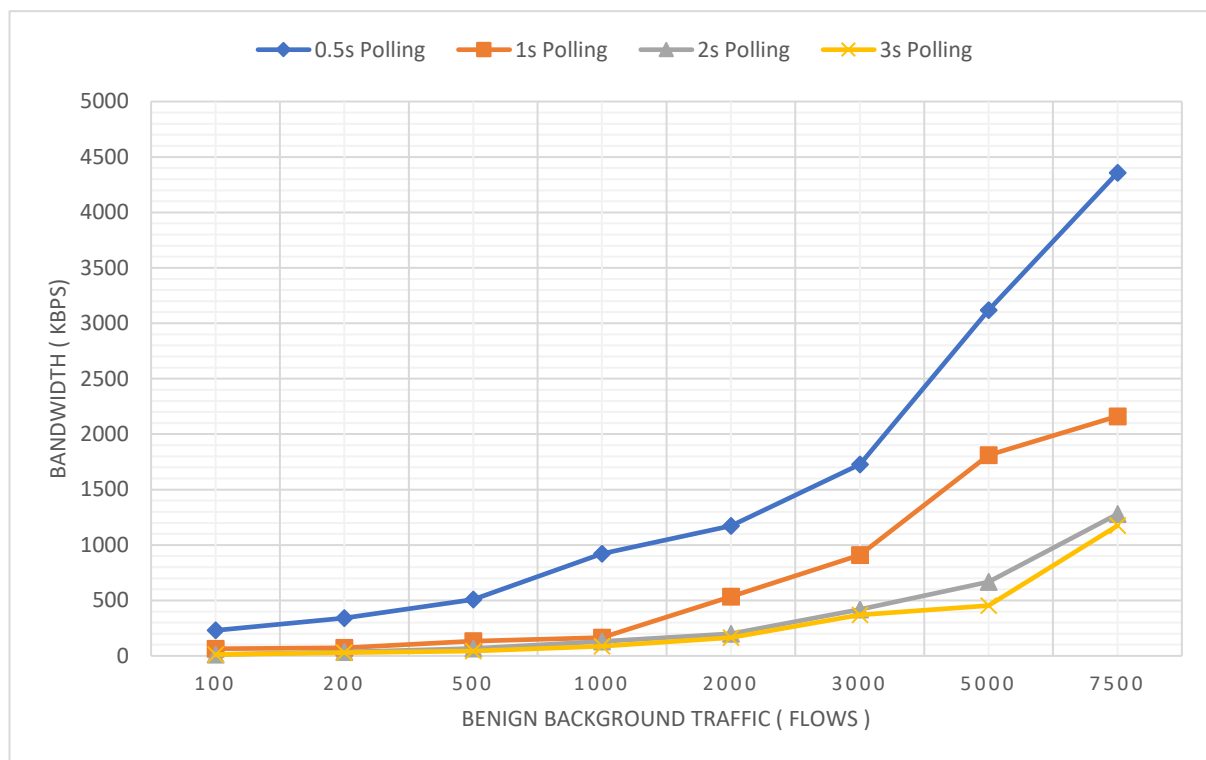


Figure 38 Effects of polling rate on network Bandwidth

6.2.4 Pulsar Security Analysis

There is substantial uncertainty around security in publish subscribe system [38]. Broker-based pub subs must send messages to subscribers with in-band signalling. This is especially true for pub sub systems that use multicast messaging. In short if the pulsar framework is compromised the attacker can exploit this system to drop legitimate traffic , dominating controller flows. As Apache pulsar is the central platform to exchange malicious info , enabling security features it critical. Pulsar's supported security features are analysed to determine Pulsar as a viable messaging exchange model platform for IDPSs.

Out-of Box configuration

There is no encryption, authentication and authorization with the default configuration of the pulsar framework. Without complete TLS configured , man-in-the-middle attackers can eavesdrop the network , finding pulsar's service URL. Access to the pulsar clusters via the plain text service URLs must be restricted to trusted clients only. Without this the clusters are accessible to outsiders and easily exploited. Pulsar strongly recommends that deployed clusters make use of recommended secure service components. Apache pulsar provides well documented security features and tutorials.

Network Segmentation / Distribution

System integrity and availability rely on its ability protect entry points exploited by attackers. Weak points within the architecture can compromise the entire SD_WANs availability and integrity. To prevent single point of failure, industry have proposed the concept of network segmentation. Apache Pulsar supports platform distribution to restrict access to trusted IPS in such cases.

Segmenting the network with geo-replicated clusters will uphold system availability if a single cluster is compromised. This means that brokers, bookies and zookeeper act independently and unsupervised against outsider attacks. The machine hosting the pulsar proxy will produce a public ,IP making is susceptible to all kinds of attacks. If this instance is compromised the user can remove the protection system thus this should also be distributed as show in [30].

If the attacker has knowledge of the athenz/ACL topic principal name , the attacker can control the proxy to produce and consume from any topic . PIP-9 [39] have added authentication to authenticate the client at the proxy , this will pass the role token to a broker who will authenticate and authorize both proxy and user. Without this pluggable security feature deploying a pulsar proxy introduces a SPoF.

TLS

Pulsar propose the integration of transport layer security (TLS), which can be configured with encryption and authentication . Pulsar supports pluggable security features and multiple authentication sources. Pulsar's TLS if in the form of public key cryptography. Pulsar support the use of 3 key pairs. The first key pair is the server key pair , this encrypts

message between the proxy the brokers, this is then decrypted by the bookies with a private key. The second key is the certificate authority, in which the admin generates a certificate. Transport encryption uses the trust cert to verify that the client or server they are communicating with is signed by the certificate authority. The third key pair is client key pairs are used for authentication, explained under the heading Athenz. Each of these keys such be stored in a very secure location such as an air gapped computer / Demilitarized Zone (DMZ).

Pulsar have strongly considered previous platform exploits. A common vulnerability is attackers requesting downgraded TLS protocol versions. Modern platforms utilize TLSv1.2 however attacker force the proxy to abandon its encrypted connection to favour exploitable clear text communication. Pulsar require specified protocol versions and ciphers preventing the attacker from gaining access to the proxy cluster.

Pulsar also support hostname verification, this considers the chance the attacker has compromised Pulsar. This will allow each client to refuse connection in an expected man n the middle attack. This is common in clusters with multiple proxy nodes hidden behind a virtual IP address (VIP). This would require the attacker to have complete access over an entire cluster and have access to the certificate authority (CA).

Authentication / Athenz

The first step to collaboration and multi-tenancy is to prevent attackers within the system accessing or producing to topics. Authentication and access control list (ACL) is used to prevent a compromised system damaging other LANs. This is achieved through role tokens. These role tokens control permission for clients to produce and consume messages from specified topics. In DIPA, each local domain use their own specified namespace and topics for configurable permissions. These security tokens are based on JSON web Tokens proposed in RFC(7519) [40].

Pulsar utilises Athenz [41], a decentralized token management system created by Yahoo. This token-based authentication system has comprehensive support and a large community involvement. Encryption is strongly recommended to protect these tokens. Tokens allow the system to grant permission levels to each domain, i.e the system can be setup to offer

support to an vulnerable LAN but restrict producing permissions if the system is easily exploited. Effectively Pulsar authentication isolates tenants from performing action they have no permission for , or locking out compromised networks from spreading.

End to end encryption

Encryption is used to ensure adequate system protection . Pulsar supports both asymmetric and symmetric encryption . As DIPA does not publish messages containing sensitive information , encryption isn't required however the future of IDPS collaboration relies in exchanging more information across the WAN.

If the bookies are compromised the attacker will be able to read all messages stored on the persistent disk. Currently pulsar uses AES data key with the application of ECDSA/RSA key pair. Messages remain encrypted on the persisted disk within the bookie and can only be decrypted by a consumer with a valid key. Messages will be lost and unrecoverable if these key are lost. In the event of an attacker obtaining a data key , pulsar utilizes key rotation to generate a key every 4 hours or the keys can be rotated after a certain number of messages. With this the keys can be rotated if the attacker attempts to DDoS the system . Figure 39 details the encryption operation.

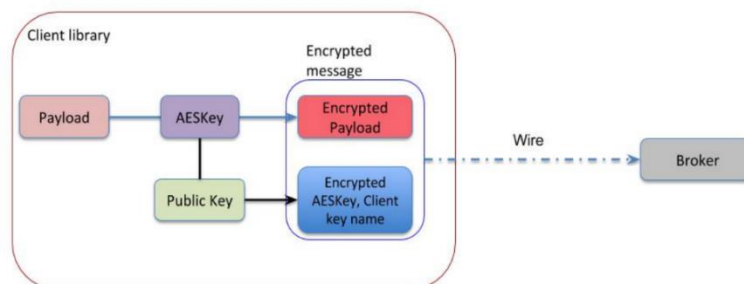


Figure 39 Pulsar to Broker encryption

6.3 Performance Test Set

To ensure pulsar's viability as a collaborative message exchange system , a series of performance test were constructed . Each test was compared with a stand alone IDPS. As detection is not within the project scoping , the protection piece is the central focus point. The coarse grain NIDS is used as a reference point for a standalone IPS. Within each of the below scenarios it is assumed that the attack should be mitigated at it source and destination domains. These tests are used to evaluate system functionality and pulsar's

protection piece, thus it is not the intention to perform tests with thousands of nodes as a virtual testbed for this would not be the best solution. For each scenario the number of flows is equivalent to the number of hosts using the network. Control plane time is used as a reference point and not the intention to improve this piece.

6.3.1 Standalone IDPS protection time vs Pulsar CIDPS Protection time

In this experiment the source network uses a range of background traffic to test the system under a variety of loads. This experiment measures the time to detect the CNC at the source destination, protecting against it on a series of neighbouring virtual Machines. The hping3 tool and BoNesi are used to send customized telnet packets to a series of hosts. This will emulate the unidirectional telnet traffic inherent from the Mirai botnet attack. This attack will be replicated on two and three VMs five seconds later to emulate Mirai scanning/loading other parts of the network.

Standalone IDSs force each domain to detect the same attack multiple times. However collaborative protection pre-emptively alerts the neighbouring domains of an attacker, protecting the system before the CNC device has a chance to open communication with a vulnerable device.

As described in section 6.2.3 the system is not as effective protecting simultaneous attacks from the same source. If both domains exhibit the same control plane time, it will take both systems the same time to detect, resulting in protection overhead. In reality, production LANs are of various sizes with different amounts of traffic. In such cases, faster processing systems can protect neighbouring destinations domains that are being attacked simultaneously by the same source.

DIPA can be used to protect SDN systems that have no detection mechanisms installed. Comprehensive NIDS can be costly across large system domains, thus organization may wish to have no detection mechanisms but still ensure system availability. Integrating DIPA allows malicious traffic detected on a neighbouring LAN can be shared with this domain. Collaborative protection can also be integrated with legacy systems and share malicious information to other protection tools such as snort.

As seen in Figure 40, as the attack is executed on both virtual machines, it takes the standalone protection algorithm almost 7 seconds to detect and apply mitigation schemes

against this malicious flows. However when the same attack is executed with collaborative protection the source domain alerts the neighbouring destination domain of a malicious CNC, protecting the system from the same attack without the need for detecting again.

The effects of this pre-emptive protection are notable with the introduction of a 2nd destination domain. Figure 41. With Standalone NIDS the system is forced to detect the same attack multiple times , in turn wasting time and resources. It takes the domains almost 14 seconds to protect against the same attack with 7500 flows of benign traffic. With Collaboration integrated the system 'pre-protects' each destination domain in approximately 4s under heavy load conditions.

As both source and destination domains all have the same network conditions , the control plane time is negligible . System protection time for the following test case scenario is represented below. The following formulas assume the same test conditions. The below formulas are approximations used to estimate the time taken to protect the system .

In live systems there is a lot more variables such as network size , number of hosts , varying packet sizes , background processes. Thus this formula cant be applied to any network topology . However the following formulas provide relative timing equations for the given setup ,which has been designed to emulate a production environment. It is predicted that similar time trends will be exhibited with further testing.

$$\sum_{i=1}^n (Poll_t + Latency_{sr} + DDoS_p + Latency_{sw}) \quad (1)$$

n = Number of Destination Domains

$Poll_t$ = DDoS detection Poll Time

$Latency_{sr}$ = Latency of stats Request

$DDoS_p$ = DDoS process time

$Latency_{sw}$ = Latency Flow rule write

$$Poll_t + Latency_{sr} + DDoS_p + pulsar_p + pulsar_b + \max(pulsar_c + Latency_{sw}) \quad (2)$$

$Pulsar_p$ = Pulsar Process time

$pulsar_c$ = Pulsar Consume time

$Pulsar_b$ = Pulsar Broker Lookup time

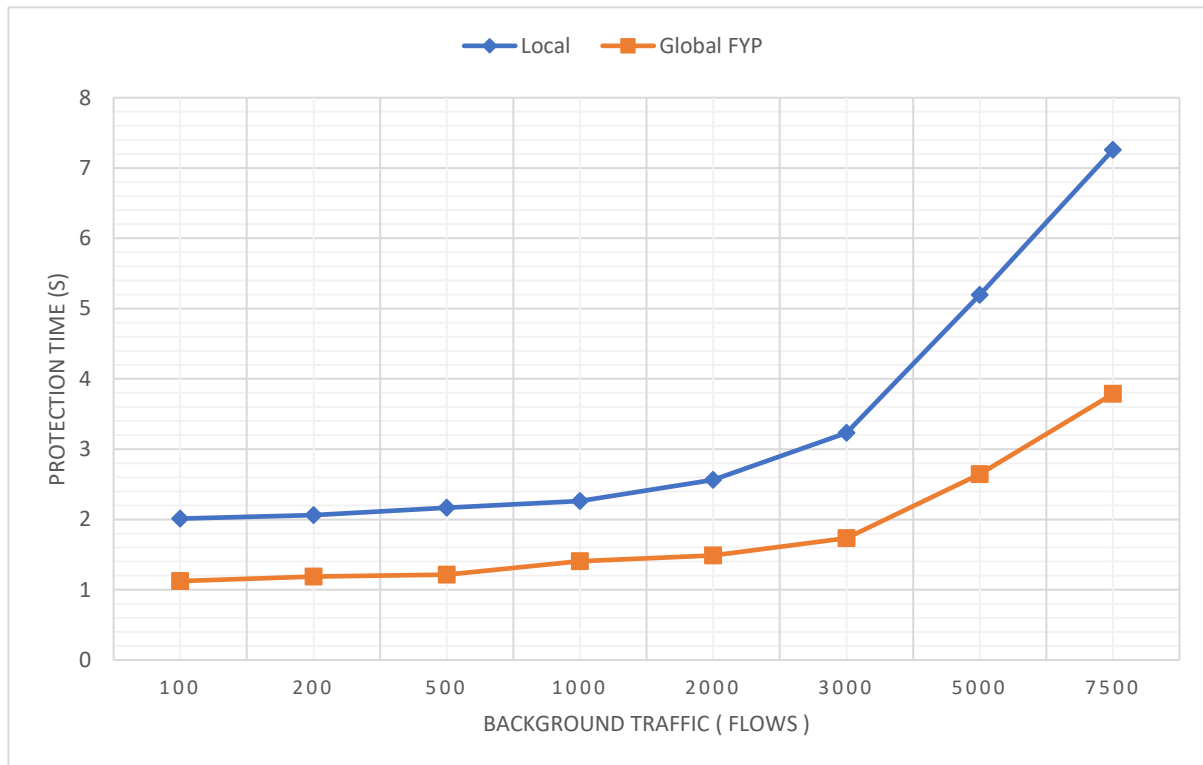


Figure 40 Collaborative Protection Time with 2 Virtual Machines

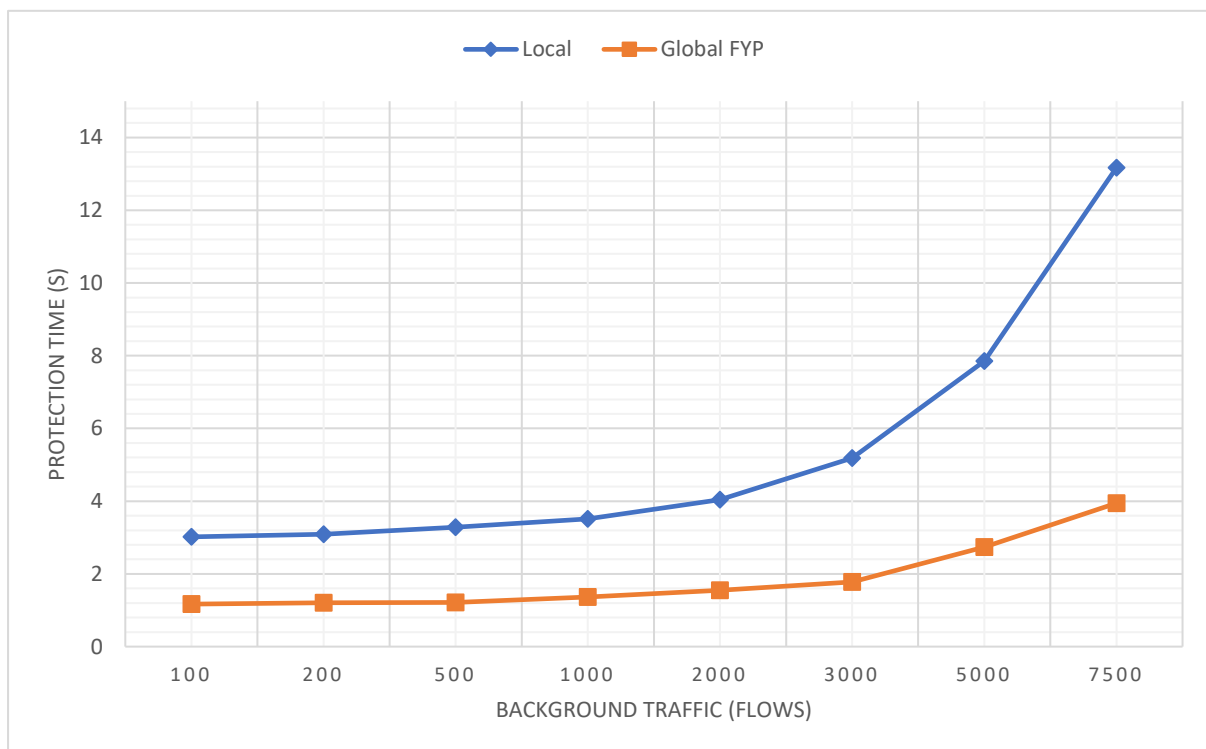


Figure 41 Collaborative Protection Time with 3 Virtual Machines

Table 5 Protection Time Standard Deviation (2VMs)

	Local	Global
100	0.001868	0.04836
200	0.003673	0.053881
500	0.022795	0.014375
1000	0.040095	0.019083
2000	0.044653	0.014774
3000	0.136356	0.03612
5000	0.11569	0.080657
7500	0.090411	0.038345

Table 6 Protection Time Standard Deviation (3VMs)

	Local	Global
100	0.002292214	0.016277
200	0.013646203	0.018308
500	0.051847062	0.011552
1000	0.020104982	0.034775
2000	0.105509883	0.026703
3000	0.234518679	0.017223
5000	0.295850121	0.079801
7500	1.114606982	0.055189

6.3.2 Control + Data plane Vs Produce + Consume Time

The following test case adopt the same test scenario as above. This experiment extension is used to identify the prominent time factor within the proposed system . The following test cases analyse the NIDS' process time in comparison to Pulsar's time to produce , process and consume by neighbouring domains. As shown in Figure 43 , as the system is subjected to an increase in background traffic , the time to classify the attack exponentially rises due to the number of lookups in each OpenFlow packet.

This is shown in comparison to the time to collaborative exchange this malicious info with a destination host. Figure 42 presents that precautionary collaborative action to prevent malware spread is inexpensive in comparison to redetection. Pulsar is unaffected by the control plane time of each domain or the amount of bots published over the framework. As detailed in section 2.5 , with the given cloud hardware , pulsar can handle up to ~250, 000 messages per second before experiencing process fluctuation. This is reinforced by the uniform standard deviation in the collaborative plane show in Table 7.

Slight variance in results is notable in the collaborative plane . This is a results of input/output operations per second restriction on the zookeeper nodes. T2.small instances with HDD volumes were used which limits IOPS to ~ 200 configuration operations per seconds [42]. Pulsar will most likely to run into variance with new topics (i.e new namespaces / domains). This claim was reinforced by communicating with pulsar community via slack . To verify the following commands were executed to examine CPU usages on each of these nodes :

```
watch 'cat /proc/meminfo'
```

```
watch free -m
```

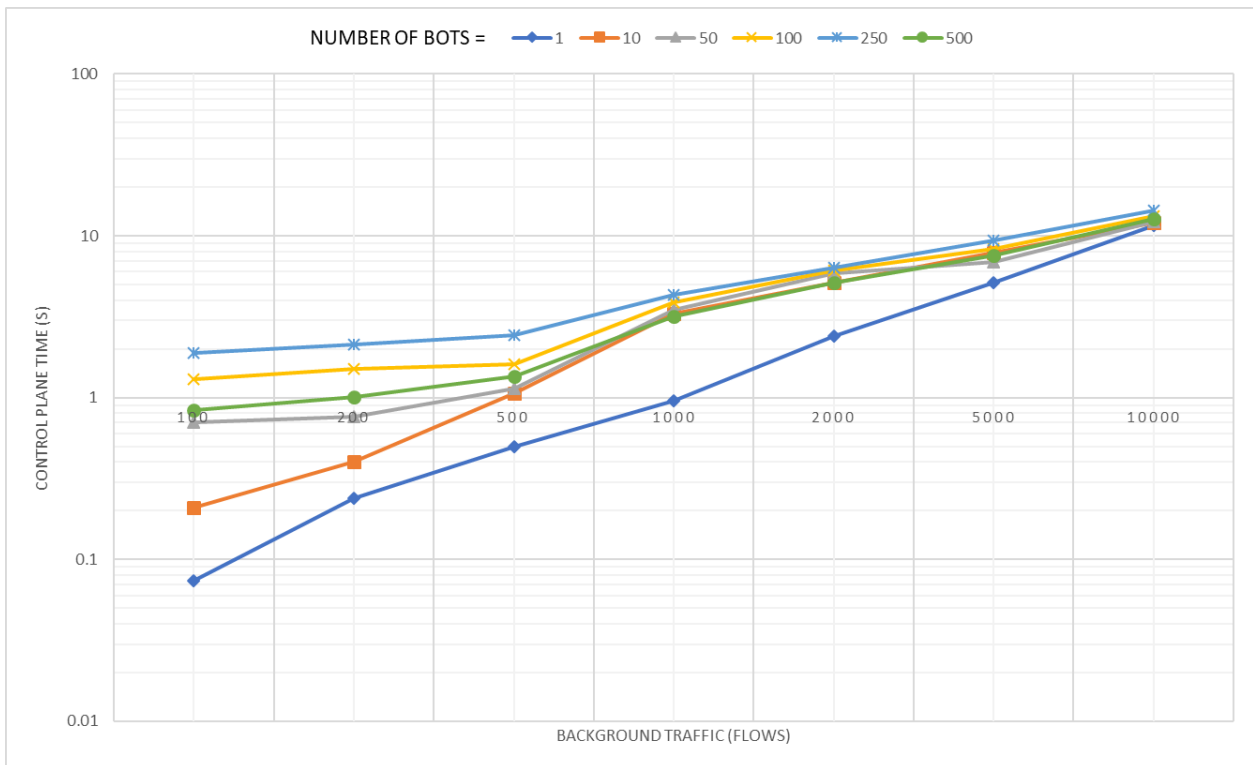



Figure 43 Control Plane Time vs Background Traffic

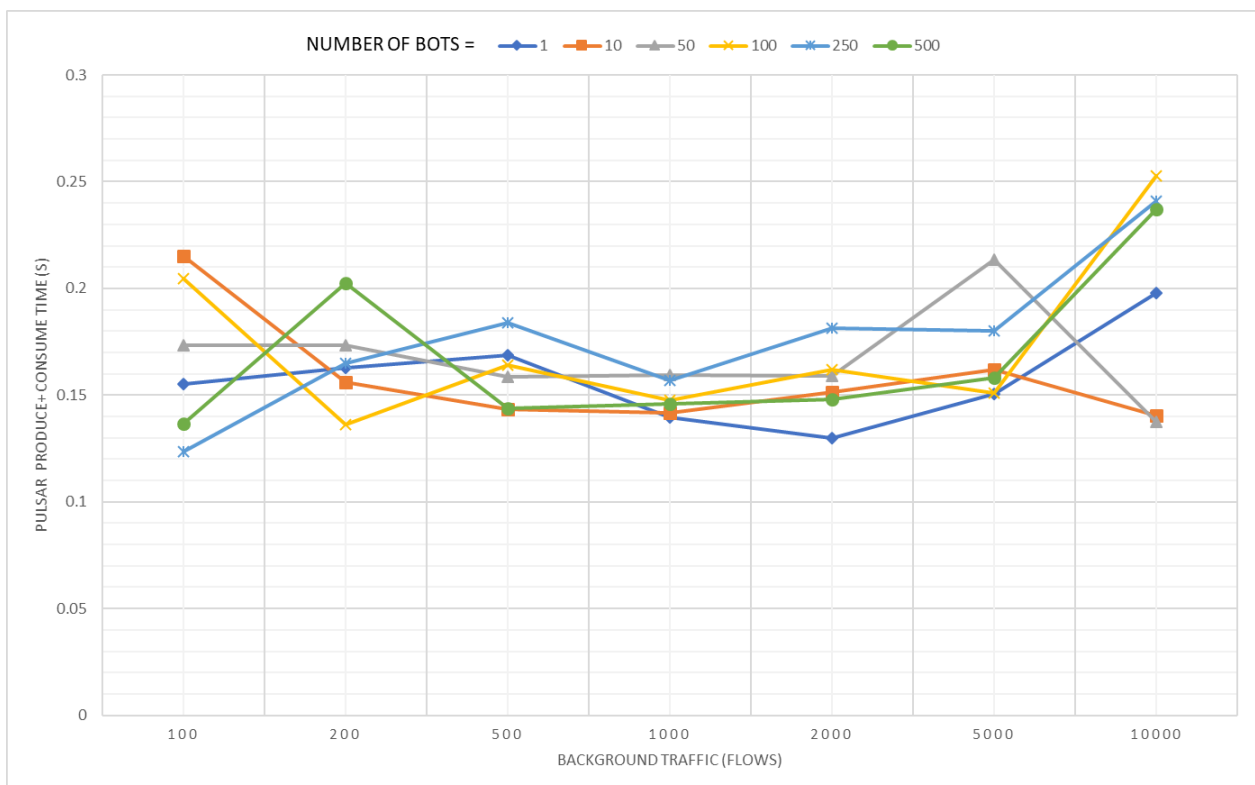


Figure 42 Collaboration Time vs Background Traffic

Table 8 Control Plane Time , Standard Deviation

	100	200	500	1000	2000	5000	7500
1	0.006197039	0.010663481	0.028779479	0.131915122	0.379412547	0.519171593	0.866436527
10	0.027311369	0.022237577	0.175310493	0.37447959	0.144941995	0.207756562	0.973555236
50	0.046269531	0.034033864	0.157726969	0.446619146	0.694078818	0.174889443	0.165733458
100	0.031962176	0.034382107	0.067569502	0.269507932	0.162925719	0.327586347	0.507718827
250	0.115536228	0.279949216	0.458473878	0.322892538	0.293718689	0.925636618	1.900386191
500	0.141104124	0.28025984	0.569261375	0.753291158	0.701457566	1.020841002	1.017393308

Table 7 Collaboration Time , Standard Deviation

	100	200	500	1000	2000	5000	7500
1	0.025307978	0.020528589	0.023156806	0.021397139	0.03995852	0.030615942	0.04905391
10	0.101738829	0.006148359	0.035745007	0.021594373	0.032958018	0.033652991	0.0376632
50	0.079002292	0.020199688	0.049426966	0.020378971	0.054788044	0.017285039	0.02759091
100	0.090425859	0.035449989	0.039397201	0.030655233	0.03133934	0.038169479	0.12237994
250	0.023884247	0.015968886	0.055449296	0.033966146	0.050565285	0.104066422	0.10234435
500	0.025153168	0.063537211	0.041672764	0.036434887	0.029825662	0.032503643	0.05721719

6.3.3 System Accuracy

This section analyses the NIDS accuracy against edge based attacks as a reference point . A comprehensive detection piece is required to produce an end-to-end solution however it isn't within the project scope. Nevertheless the coarse grain NID is used to emulate a stand-alone IDPS to compare with the proposed collaborative model. As shown in Figure 44 the detection algorithm's accuracy dramatically drops as the background traffic reaches ~7500 . With the given hardware the controller is able to support ~8000 flows in the OpenFlow Switches before the legitimate traffic begins to flood the controller. In this case accuracy is greatly reduced as the detection classifier is unable to process all of the data within the flow tables in the given time frame before the next *statsRequest* is received.

A series of test scenarios were constructed to test global flow accuracy from source to destination domain .Pulsar was able to produce and consume all malicious traffic detected by the source host as shown in Table 9. Regardless of control plan time or effects on the controller. Within initial stages of the software realisation, the NIDS and the protection system were abstracted to let both functions execute on separate threads. As a result , problems encountered by the controller do not have adverse effects on the pulsar client. The following command was executed on the destination domains to ensure that the number of flows matched the number of malicious bots generated by BoNesi.

Watch 'Sudo ovs-ofctl -O OpenFlow13 dump-flows s<switch_num> | grep drop'

Table 9 Collaborative Detection Accuracy

Test Scenarios	Num Benign Flows	Num Bots	Num Source Flows	Accuracy	Num Destination Flows	Collab Accuracy (source/dest)
1	500	1	1	1	1	1
2	5000	10	7	0.7	7	1
3	100	50	47	0.94	47	1
4	2000	100	59	0.59	59	1
5	200	200	176	0.88	176	1
6	7500	500	45	0.09	45	1

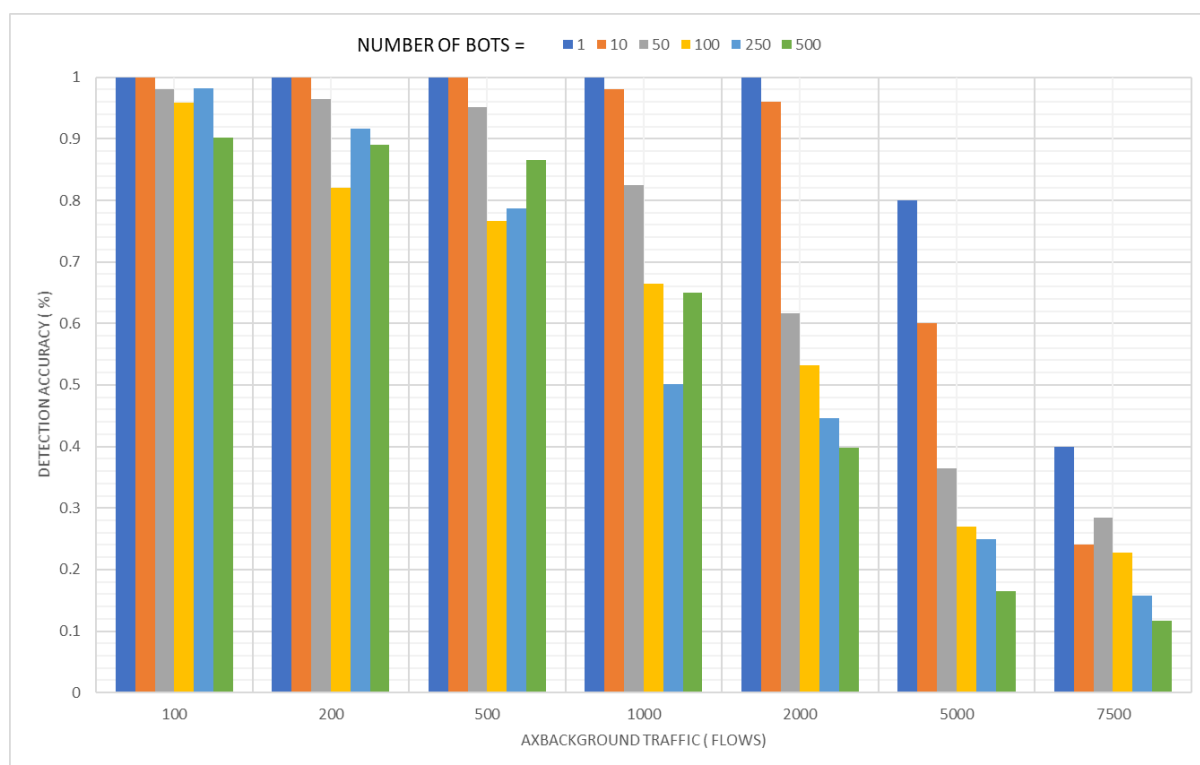


Figure 44 Detection Accuracy vs Background Traffic

6.4 Results Analysis

This section evaluates the test results received, presenting both strength and weakness in test conditions. Chosen test scenarios were constructed from existing solutions for comparable measurements for pulsar viability in collaborative protection. In relation to objective points, a test set was used to test system functionalities. Pulsar distribution allows the users to configure each LAN; enforce access levels to each Domain, set Domain priority. With the given test scenarios, destination domains consumed and installed 100% of the malicious flow rules detection by the source address. Pulsars high throughput makes it a viable solution for network scaling and passing more information over the WAN.

However due to large number of variables within networking, further testing is proposed to satisfy uncertainties regarding system performance.

6.4.1 Comparisons to Existing Solutions

Hameed and Khan [43] propose a custom collaborative protocol to exchange information, this proposed system is able to detect and protect against a simulated attack within 2.14s on an eight hop linear deployment (5000 benign flows). Data dissemination using this protocols estimates a latency of 21s to protect 5000 autonomous systems. DIPA is able to protect 2 domains in ~2.9s, to get accurate comparison these system would need to use the same hardware setup . It is predicted that data dissemination using Apache pulsar however would be a lot lower for 5000 subscribers from Streamlio's benchmark test. [29]. This requires further testing.

ECESID claims to detect and offer protection in an average time of 6.02s. Large latencies are introduced within this system with the overhead of deploying fog monitor units in the cloud. Without this overhead it is able to detect within 3.6 at its fastest recording. ECESID employ a more comprehensive botnet classifier to DIPA, thus taking a longer time to detect.

Centralized architecture such as FireCol [17] and CoFence [13] fail to address single point of failure issues. This in turn allows the attacker to affect system availability from a single compromised LAN. DIPA's distributed architecture addresses these issues through pulsar's cloud deployment support and its ability to geo-replicate data partitions across multiple bookie nodes.

Pulsar offers system flexibility and scalability. Existing solution such as CRIM [14] and [22] fail to adapt to different environments including adding new destination hosts. These systems assume the same hardware is used for each LAN and don't account for variance in LAN performance.

6.4.2 Areas of Uncertainty

Although section 6 presents promising results for pulsar within CIDPS , there is some areas of uncertainty found under system testing. This section looks at areas of uncertainty in testing results and scenarios. A variety of test conditions are proposed to cover a range of network variables such as domains size and domain amount .

Pulsar Load Testing – Currently Bonesi is used to generate up to unique 500 bots , however uncertainty still lies around Pulsar’s capabilities. With the given hardware , the limitation of Ryu are presented with the amount of traffic the controller can govern. However ambiguity remains to pulsars flow rate limitations. Due to this limitations , the system was designed to use the same configuration and instance size as Streamlio [29]. Streamlio use apache bench to benchmark pulsar’s capabilities. This is used as a estimated reference point however further research is required in a production environment. The tested proof of concepts meets the set objectives and goals desired however further load testing is need to emulate production SD-WAN.

Number of Consumers – Given the set hardware , it is undetermined how many LANs/Domains can be used. This test scenario utilizes up to 3 virtual test beds to validate pulsar functionality. It is also used to examine the standalone vs collaboration , showing the disparity in results with more destination hosts. A series of virtual testbed is not suitable for the current test cases due to hardware restriction . It is proposed that docker is implemented to containerize each domain. Kubernetes should be integrated to separate containers into pods for Autonomous system isolation.

Detection Algorithm – The current test results are compared with the coarse grain NID. This reference point is used to justify the time to collaborate is less than the time to detect the same attack . With improved hardware , the TCAM lookup and process time may be lower than this collaboration time. A range of NIDS should be tested with the proposed pulsar framework for further validation. This system aims to integrate a more comprehensive NID for a complete end to end solution.

7 Discussion

7.1 System Limitation

After deep revision of the system capabilities a collection of system limitations and exploits were discovered . The following system limitations are out of the scope of this current project , however provide additional focal points for further development.

The main limitations of the proposed solution is it operational ability under different phases within the Mirai attack . The scope of this project is protection of the early phases of Mirai

(i.e scanning and loading phases). However the current system isn't advised to be deployed in systems that are already under attack, in which the attacker has enslaved a vast portion of the network. Deploying DIPA in an already compromised environment may give attackers access to sensitive pulsar information upon setup. The current system is also not designed to prevent DDoS flows. This system exclusively monitors high telnet traffic and malware loading traffic from bots and CNC on port 48101. Latter phases of the Mirai Botnet attack involve attacks such as TCP Syn Floods against a critical system such as DNS. Additional system protection is required for these latter phases.

The current implementation is tailored for the given project specification , in turn resulting in a limited detection algorithm .Existing comprehensive NIDS commonly use deep learning on datasets ,unable to work on live traffic .This however is not suitable for the given project objectives. The protection mechanisms are hindered by the coarse grains abilities.

The proposed solution is also limited in its ability to share sensitive information over the wire. Hardware addresses and sensitive topology information cannot be transmitted over the link. This design decision accommodates for other destination domains being compromised , thus the attacker is unable to learn any harmful information regarding neighbouring domains. This solution aims to build with a more comprehensive NIDS which will require more knowledge across each network to classify the attack. Figure 45 details the most prominent features that are used to classify a botnet attack. All these features except HW_ADDR can be shared over the wire without the attacker gaining sensitive information .

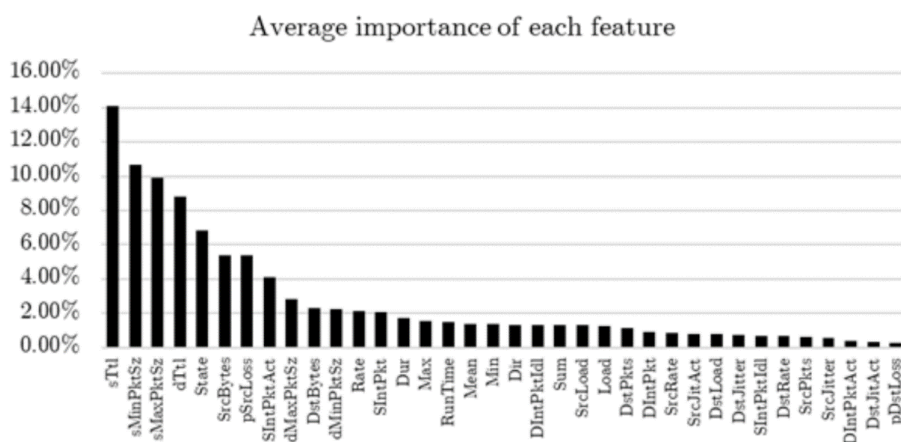


Figure 45 Most prominent Features in botnet Attacks [52]

7.2 System Exploits

Revision of the proposed solution presented system vulnerabilities that the user can exploit to bypass the defence mechanisms. Currently the solution is unable to compensate Arp Spoof. With this attackers can spoof their source address to stay hidden under the detection algorithm . With this the protection mechanism will never be triggered. Current ARP spoof mitigations include introduction of a virtual private Network (VPN) , forcing devices to attach to a virtual IP. This will allow devices to be restricted to a single IP. However not all networks use a VPN, thus ARP spoofing software should be used to prevent ARP cache poisoning. Ryu support a collection of ARP spoofing migraters which can be added in future development.

Currently the implemented solution applies dropped flow rules to all the switches in the destination domains. However as this project protects against edge based attacks , these flows rule should only be placed at the TOR switches (switches that the edge of the network connect to). These unnecessary and unused flows induce unwanted bandwidth consumption for each *statsRequest*. With this the attacker can fill these tables with unused flows , consume the control plane bandwidth , in turn stunting the detection algorithms efficiency and process time.

7.3 Project Plannning

7.3.1 Risk mitigation

Throughout the 30 week assessment process , progression and continuity of the project remained constant. With the objectives and goals drafted, a time management structure was outlined to overview milestone and objectives to be achieved. These goal are represented in the Gantt chart in Appendix 1. This chart provided realistic accomplishments within the given timeframe , compensating for unexpected challenges encountered during the process. First semester was primarily used to research purposes ; understanding software defined networking and existing mitigation defence system used to combat recent intelligent attacks. Revision of these concepts was required throughout the given timeframe to ensure the project goals and objectives were achieved whilst remaining within the project scope.

A variety of problems were encountered throughout the process. To ensure project continuity and fruition ,a series of outstanding objectives were outlined each week .

Objectives were ranked in order of priority . using a scale to rank the severity of the problem. To ensure smoother transitions and to ensure all base objectives were achieved a variation of methodologies were listed. The reduced list can be viewed in Annex 1.1 and weekly minutes. To compensate for project time constraints ,realistic milestones were set with 20% tolerance for each major task to allow for unseen challenges such as other module time pressures but mostly to compensate for problems with system design and setup. The Gantt chart, priority list and solution list were a valuable asset to the completion of this project. Basis functionality was explored from each risk factor before expanding solutions in further depth.

7.3.2 Project Approach and design stages

First semester was mainly used for research purposes. With no prior knowledge in software defined networking , this was a steep learning curve to acquire a deep understanding of these new emerging technologies. Revision of background research topics were required throughout the process for functionality clarifications. Weekly meetings with recorded minutes helped retain focus with project scope , preventing deviation of the set project objectives. These minutes can be viewed on the GitHub URL provided in Appendix 2

Extensive research into the taxonomy of collaborative protection systems and recent new on the evolution of Mirai , helped to gain an understanding and need in industry for an end to end solution. This research provided vital information for design decisions made to the final design. Research into existing collaborative schemes proposed a variety of detection and mitigations schemes. Recreation of these strategies proved time consuming to produce both a comprehensive detection and protection piece. However trial and error of these strategies was necessary to gain more understanding.

To prevent deviation and keep focus on the protection side of this piece a series of design stages were outlined at the end of the first semester. Design stages prioritized the basic functionality of pulsar before integrating the Ryu controller . The pulsar framework was tested and analysed, testing simple functionality and the systems limitations before designing the DIPA architecture. System architecture was an integral part of this project. This required extensive planning and configuration to ensure the architecture emulated a series of isolated autonomous systems. This became the primary focus of the conceptual system model, to replicate an enterprise network in a test environment.

After basic functionality had been implemented and tested , second semester was used for critical analysis , test cases , report writing and optimisations. Critical analysis of the design after the preliminary report helped re-evaluate project objectives and scope. This helped gain an understanding of system weaknesses and areas to focus on. Multiple design phases and system reviews were required at each stage.. Planning sufficient time for report writing helped adhere to the project objectives and stay within the project scope .

7.4 Future Work

Each of the goals set at the beginning of the project have been satisfied, showing areas of improvement compared to related work. Nevertheless critical analysis and revision of the results introduced areas of uncertainty within the system. Critical analysis of the system detailed future tests to justify assumptions and estimations calculated. It also exposed system limitation and exploits , which with more time and professional field knowledge can be analysed to produce a full end to end solution.

7.4.1 Live Test Conditions

As discussed in section 6.4 , there is uncertainty around the effectiveness of the solution in a live environment. The current implementation can be placed within a production environment and collaboratively protect without much modification of protection mechanisms. By following setup steps in the GitHub, this will enable protection for OpenFlow SDN environments. However the proposed solution should be augmented with a smarter NIDS to produce a complete end to end system . Currently the coarse grain NIDS will provide little support in production . To enhance this system , further research can be used to insert a DMZ testbed to install the Mirai source code and test it effectiveness against a physical attack. .

After the Dyn DDoS attack , organizations were forced to recall a list of IoT devices and enforcing them to meet a specified standard [44].These devices still remain unprotected and sold worldwide. A list of vulnerable devices detailed in Appendix 6 still exist . Future development would test the system using these vulnerable IoT devices .

7.4.2 Pulsar IO Connectors

Apache pulsar is a lightweight messaging system that although still in early development includes a lot of powerful features. One of these features is IO connectors, a specialized pulsar function with the purpose to build off external system integration. With this Pulsar can utilize the data stored on persistent disk, storing messages on external databases. This will increase storage space, increasing read and write speed of partition lookups. Apache Cassandra [45] is one of these systems that can manage data scalability with compromising availability and system performance.

As an extension of the implemented protection system, a global protection tool was designed with a simple proof of concept. Code can be found in the Github. This protection mechanism received updates of any telnet flows from the domains, storing all values in a python dictionary. The focus of this defence mechanism was to understand the domains from a global level and find low cumulative attacks passing under the radar. An IO connector will be used to integrate a NIDS in the pulsar framework to classify data across the WAN from a global perspective. This source detected a CNC by analysing telnet traffic over each LAN, producing a drop signal if a reviewed IP was over a specified threshold.

With further implementation and introduction of IO connector this global protection system can be used to find these small cumulative attacks providing global synthesis. Proof of Concept functionality can be seen below in Figure 46.

Figure 46 Global WAN NIDS (Future Work)

```

apache@apache-VirtualBox:~/clones/ELF4001-FinalYear-Project/TestCase_6$ python threaded_consumer.py
2019-04-14 22:44:44.035 INFO ConnectionPool:63 | Created connection for pulsar://192.168.42.4:6650
2019-04-14 22:44:44.037 INFO ClientConnection:285 | [192.168.42.4:33380 -> 192.168.42.4:6650] Connected to broker
2019-04-14 22:44:44.041 INFO HandlerBase:53 | [non-persistent://sample/standalone/ns/my-topic1, ] Getting connection fr
om pool
2019-04-14 22:44:44.045 INFO ConnectionPool:63 | Created connection for pulsar://apache-VirtualBox:6650
2019-04-14 22:44:44.048 INFO ClientConnection:287 | [192.168.42.4:33382 -> 192.168.42.4:6650] Connected to broker throu
gh proxy. Logical broker: pulsar://apache-VirtualBox:6650
2019-04-14 22:44:44.085 INFO ProducerImpl:155 | [non-persistent://sample/standalone/ns/my-topic1, ] Created producer on
broker [192.168.42.4:33382 -> 192.168.42.4:6650]
2019-04-14 22:44:44.087 INFO HandlerBase:53 | [non-persistent://sample/standalone/ns/my-topic2, ] Getting connection fr
om pool
2019-04-14 22:44:44.133 INFO ProducerImpl:155 | [non-persistent://sample/standalone/ns/my-topic2, ] Created producer on
broker [192.168.42.4:33382 -> 192.168.42.4:6650]
How Many Subnets are being tested ?1
2019-04-14 22:45:01.997 INFO HandlerBase:53 | [non-persistent://sample/standalone/update/update0, updates0, 0] Getting
connection from pool
2019-04-14 22:45:02.000 INFO ConsumerImpl:168 | [non-persistent://sample/standalone/update/update0, updates0, 0] Create
d consumer on broker [192.168.42.4:33382 -> 192.168.42.4:6650]
Mean : 14.3333333333, Std :0.0
Mean : -14.3333333333, Std :0.0
('CNC Bot detected at : ', '10.0.0.2')
All subnets havents produced a results yet
Mean : -14.3333333333, Std :0.0
('CNC Bot detected at : ', '10.0.0.2')
Mean : 14.3333333333, Std :0.0
Mean : 14.3333333333, Std :0.0
Mean : -14.3333333333, Std :0.0
('CNC Bot detected at : ', '10.0.0.2')
Mean : -14.3333333333, Std :0.0
('CNC Bot detected at : ', '10.0.0.2')
Mean : 14.3333333333, Std :0.0
Mean : 14.4, Std :0.0
('CNC Bot detected at : ', '10.0.0.1')
Mean : -14.2666666667, Std :0.0

```

7.4.3 Pulsar statsRequest

Critical review of the DIPA architecture , showed that data collaboration performance is hindered by the time to receive switch statistics. Ryu implement a legacy polling model to receive switch statistics in a pre-configured timeframe. Ryu can harness a Pulsar's message exchange framework to control data stored on the switch. Multi-tenancy can be used to concurrently store and serve data to the controller with high throughput and low latency. Removing the polling models will reduce can effectively reduce the time to receive and classify packets from ~10s to ~400ms , if using the same hardware and traffic size as DIPA.

7.5 Pulsar Cost and resource usage

This sections analyses the cost evaluation of Pulsar and its viability for SD-WAN integration. Pulsar is currently an Opensource application like many other Apaches' products. This application requires no licensing unless a large organisation wish to make a feature contribution. With the current deployment of DIPA , 2 weeks usages with 1.8 million elastic search requests cost approximately \$973 as shown in Appendix 3.1. This is forecast to be approximately \$2000 per month . This can be greatly reduced with instance reconfiguration dependent on the size of the SD-WAN. Legacy firewall systems such as Cloudflare or Cloudbric currently provide stand alone security scheme for ~\$150 per month on 100GB of traffic .

The cost of collaborative protection introduces a trade-off of capital wastage vs system security each year. When implementing the DIPA architecture , organisations need to consider the assets loss vs the CIDPS expenditure each year.

7.6 Project Objectives Evaluation

Reliability : To ensure a reliable system , a series of test cases were crafted to test both DIPA's functionality but also its performance under a series of test conditions. The system was exposed to a variety of conditions ranging from low cumulative attacks, to classifying large amounts of bots large amount of benign traffic . The system detection dropped dramatically when exposed to heavy load. Nevertheless the system was able to producer

100% of the attacks classified on the source domains. This objective aimed to receive a minimum of 95% of the alerts. This shows that the DIPA architecture is hindered by its detection rather than its protection scheme.

Sustainability : After numerous design stages and extensive background research into existing solutions , industry have deduced that distributed architecture are the most promising at providing mitigation against these recent evolutions . Pulsar is a promising framework at offering high throughput and low latency. It is suggested that pulsar is used in a distributed architecture with geo replication to prevent system exploitation .

Security : DIPA integrates a variety of additional security features offered by pulsar to ensure DIPA;s CIA triad is upheld. As a proof of concept this piece has a few single point of failures through the API gateway and a single pulsar proxy. It is proposed that these points are distributed in future development.

Suitability : After extensive research and testing of apache pulsar in collaborative IDPS , pulsar is a promising model at providing collaborative protection and information exchange within an SD-WAN. Pulsar provides numerous features such as message deduplication , retention and geo replication to offer a comprehensive protection piece.

Performance : The DIPA model was unable to meet this requirement, collaboratively protecting isolated autonomous systems in under 3 seconds. With a background traffic load of 7500 flows , the system was able to protection 2 and 3 VMs in ~3.8s. As seen in section 6.3.2 , this is a lot lower than standalone IDPS.

Scalability : This was tested by varying the benign traffic in the system . Traffic load is proportional to the size of the network. However using a distributed pub sub framework , shared instances resources across each domains , regardless of the LAN size.

Interoperability : Pulsar is a viable solution to integrate in OpenDaylight projects. Pulsar predominantly support java which is the same language used to construct OpenDaylight controllers. DIPA currently requires manual integration of pulsar consumer and producer commands if used with a separate NIDS.

7 Conclusions

This paper takes an important step to produce a lightweight, efficient and easy to deploy collaborative edge-based mitigation scheme. The proposed scheme, implements SDN controllers within an isolated autonomous system to communicate with adjacent networks via the apache pulsar publish subscribe framework. Pulsar's high throughput and low latency helps efficiently disseminate attack definitions across an enterprise SD-WAN. DIPA introduces a distributed architecture to solve performance bottlenecks and SPoF with centralized and decentralized systems.

DIPA's conceptual models was designed considering scalability, adaptability but importantly Pulsar's viability a collaborative solution. Experiments with the DIPA prototype successfully transfers malicious data from source to multiple destination domains. It took the system approximately 3.7s with 7500 benign flows to mitigate three emulated topologies (i.e 8 hosts). This project aimed to protect and mitigate in under 3 seconds, however critical analysis of each time plane showed that this systems performance is limited by its hardware and its lack of detection complexity.

It takes the system between 100-300 ms to process and forward attack definitions to adjacent networks though the pulsar networks. Malicious information detected on a given source domain was delivered and received by two destination domains 100% of the time. Deployment of Pulsar in the cloud introduced a link latency – resource trade-off , however this was a necessary step to produce a resilient system.

Writing this report exemplifies the hard work and significant process made towards collaborative protection with edged based attacks and achieving the set objectives . An extensive survey of existing strategies helped outline an efficient conceptual architecture design. This required a deep understanding of the Mirai strategy , publish-subscribe system limitations but also features with existing architecture which have failed to evolve with recent DDoS attacks. DIPA produces an intelligent edge based mitigation scheme , employing geo-replication , message deduplication, retention to mitigate system exploit notable with related work. This report finally reviews the system architecture , proposing intelligent solutions to stop the ever-growing threats of the Mirai Botnet.

8 References

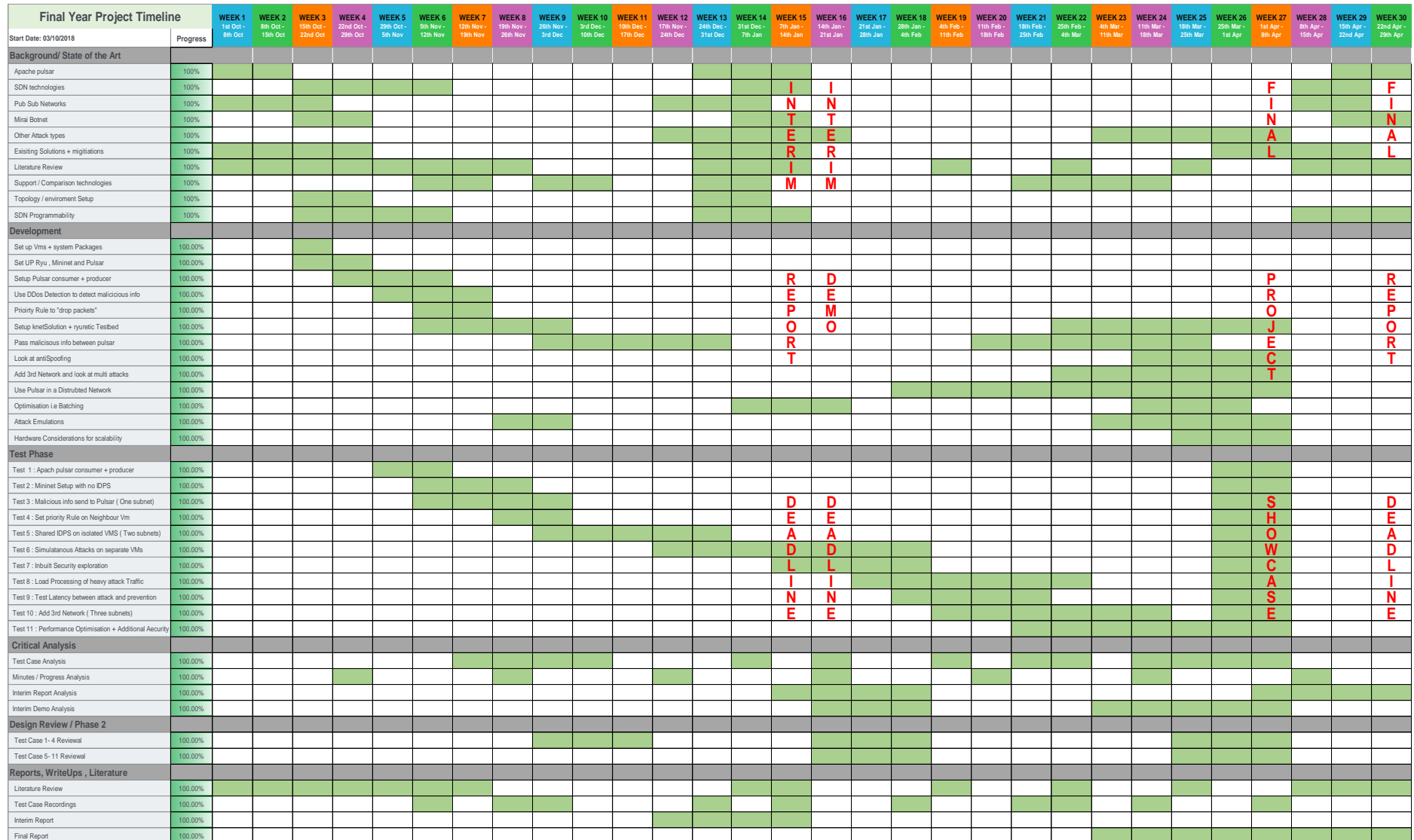
- [1] M. M. Group, "Internet World Stats by region," internetWorldStats, 31 March 2019. [Online]. Available: <https://www.internetworldstats.com/stats.htm>. [Accessed 1 April 2019].
- [2] H. Kristi Rawlinson, "HP Study Reveals 70 Percent of Internet of Things Device Vulnerable to Attack," HP, 29 July 2014. [Online]. Available: <https://www8.hp.com/uk/en/hp-news/press-release.html?id=1744676>. [Accessed 2 April 2019].
- [3] Burke, Samuel, "Massive cyberattack turned ordinary devices into weapons," CNNTech, 22 October 2016. [Online]. Available: <https://money.cnn.com/2016/10/22/technology/cyberattack-dyn-ddos/index.html>. [Accessed 2 April 2019].
- [4] Aiko Pras, Cardoso de Santanna, Jose Jair, Jessica Steinberger and Anna Sperotto, "'Measurement, Modelling and Evaluation of Computing Systems' and 'Dependability and Fault-Tolerance,'" MMB & DFT 2016," in *Proceedings of the 18th International GI/ITG Conference*, Berlin, 2016.
- [5] Santanna, Jose Jair and van Rijswijk-Deij, Roland and Hofstede, Rick and Sperotto, Anna and Wierbosch, Mark and Granville, Lisandro Zambenedetti and Pras, Aiko, "Booters—An analysis of DDoS-as-a-service attacks," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015.
- [6] Antonakakis, Manos and April, Tim and Bailey, Michael and Bernhard, Matt and Bursztein, Elie and Cochran, Jaime and Durumeric, Zakir and Halderman, J Alex and Invernizzi, Luca and Kallitsis, Michalis and others, "Understanding the mirai botnet," in *26th USENIX Security Symposium* *USENIX Security* 17, 2017.
- [7] Mikhail Kuzin, Yaroslav Shmelev, Vladimir Kuskov, "New trends in the world of IoT Threats," Kaspersky, 18 September 2018. [Online]. Available: <https://securelist.com/new-trends-in-the-world-of-iot-threats/87991/>. [Accessed 3 April 2019].
- [8] Yan, Qiao and Yu, F Richard, "Distributed denial of service attacks in software-defined networking with cloud computing," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 52--59, 2015.
- [9] Rickens, Erich, "What is a distributed ledger?," Blockport, January 2019. [Online]. Available: <https://blog.blockport.io/what-is-a-distributed-ledger/>. [Accessed 14 April 2019].
- [10] Vasilomanolakis, Emmanouil and Karuppayah, Shankar and Mulhauser, Max and Fischer, Mathias, "Taxonomy and survey of collaborative intrusion detection," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, p. 55, 2015.
- [11] Mohamed, Hassani and Adil, Lebbat and Said, Tallal and Hicham, Medromi, "A collaborative intrusion detection and prevention system in cloud computing," in *2013 Africon*, 2013.
- [12] Staniford-Chen, Stuart and Cheung, Steven and Crawford, Richard and Dilger, Mark and Frank, Jeremy and Hoagland, James and Levitt, Karl and Wee, Christopher and Yip, Raymond and Zerkle, Dan, "GrIDS-a graph based intrusion detection system for large networks," in *Proceedings of the 19th national information systems security conference*, 1996.
- [13] Rashidi, Bahman and Fung, Carol, "CoFence: A collaborative DDoS defence using network function virtualization," in *2016 12th International Conference on Network and Service Management (CNSM)*, 2016.
- [14] Cuppens, Frederic and Mieke, Alexandre, "Alert correlation in a cooperative intrusion detection framework," in *Proceedings 2002 IEEE symposium on security and privacy*, 2002.
- [15] Debar, H and Curry, D and Feinstein, B, "The Intrusion Detection Message Exchange Format (IDMEF): RFC 4765," *Network Working Group*, 2007.
- [16] Ozccelik, Mert and Chalabianloo, Niaz and Gur, Gurkan, "Software-defined edge defense against IoT-based DDoS," in *2017 IEEE International Conference on Computer and Information Technology (CIT)*, 2017.
- [17] Francois, Jerome and Aib, Issam and Boutaba, Raouf, "FireCol: a collaborative protection network for the detection of flooding DDoS attacks," *IEEE/ACM Transactions on Networking (TON)*, vol. 20, no. 6, pp. 1828-1841, 2012.

- [18] Chen, Xiao-Fan and Yu, Shun-Zheng, "CIPA: A collaborative intrusion prevention architecture for programmable network and SDN," *Computers & Security*, vol. 58, pp. 1--19, 2016.
- [19] Janakiraman, Ramaprabhu and Waldvogel, Marcel and Zhang, Qi, "Indra: A peer-to-peer approach to network intrusion detection and prevention," in *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003*, 2003.
- [20] Rowstron, Antony and Kermarrec, Anne-Marie and Castro, Miguel and Druschel, Peter, "SCRIBE: The design of a large-scale event notification infrastructure," in *International workshop on networked group communication*, 2001.
- [21] Duma, Claudiu and Karresand, Martin and Shahmehri, Nahid and Caronni, Germano, "A trust-aware, p2p-based overlay for intrusion detection," in *17th International Workshop on Database and Expert Systems Applications (DEXA'06)*, 2006.
- [22] Cheung, Steven and Dutertre, Bruno and Fong, Martin and Lindqvist, Ulf and Skinner, Keith and Valdes, Alfonso, "Using model-based intrusion detection for SCADA networks," *Proceedings of the SCADA security scientific symposium*, vol. 46, pp. 1--12, 2007.
- [23] Segall, Bill and Arnold, David, "Elvin has left the building: A publish/subscribe notification service with quenching," *Proceedings of the 1997 Australian UNLX Users Group (A UUG'1997)*, pp. 243--255, 1997.
- [24] Porras, Phillip A and Fong, Martin W and Valdes, Alfonso, "A mission-impact-based approach to INFOSEC alarm correlation," in *International Workshop on Recent Advances in Intrusion Detection*, 2002.
- [25] Kreutz, Diego and Ramos, Fernando MV and Verissimo, Paulo and Rothenberg, Christian Esteve and Azodolmolky, Siamak and Uhlig, Steve, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, pp. 14--76, 2015.
- [26] Azodolmolky, Siamak, *Software defined networking with OpenFlow*, Packt Publishing Ltd, 2013.
- [27] Ryu, "ryu.app.ofctl," Ryu, 2014. [Online]. Available: https://ryu.readthedocs.io/en/latest/app/ofctl_rest.html. [Accessed 13 April 2019].
- [28] OpenDaylight, "OpenDaylight Controller:RESTCONF Northbound APIs," OpenDaylight, [Online]. Available: https://wiki.opendaylight.org/view/OpenDaylight_Controller:RESTCONF_Northbound_APis. [Accessed 13 April 2019].
- [29] Streamlio , Pulsar, "Apache pulsar ; an open-source distributed pub-sub messaging system," Apache, 2019. [Online]. Available: <https://pulsar.apache.org/>. [Accessed 15 April 2019].
- [30] RightScale, "Designing and Deploying High-Availability Websites," Right Scale, 2019. [Online]. Available: https://docs.rightscale.com/cm/designers_guide/cm-designing-and-deploying-high-availability-websites.html. [Accessed 13 April 2019].
- [31] M. Merli, "High performance messaging with Apache Pulsar," Streamlio, 4 October 2018. [Online]. Available: <https://www.slideshare.net/merlimat/high-performance-messaging-with-apache-pulsar>. [Accessed 15 April 2019].
- [32] J. Postel, "Transmission control protocol," RFC, 1981.
- [33] Malowidzki, Marek and Berezinski, P and Mazur, Michal, "Network intrusion detection: Half a kingdom for a good dataset," in *Proceedings of NATO STO SAS-139 Workshop, Portugal*, 2016.
- [34] Allison Nixon , Ammar Zuberi, "Who is Anna-Senpai, the Mirai Worm Author?," KrebsonSecurity, 18 January 2019. [Online]. Available: <https://krebsonsecurity.com/2017/01/who-is-anna-senpai-the-mirai-worm-author/>. [Accessed 20 April 2019].
- [35] rsissons, "QoS of Telnet Traffic," Cisco, 10 October 2002. [Online]. Available: <https://community.cisco.com/t5/other-network-architecture/qos-of-telnet-traffic/td-p/50543>. [Accessed 20 April 2019].
- [36] I. A. Team, "ATI - Open Source Pcaps," Ixia, 11 December 2016. [Online]. Available: <https://github.com/ixiacom/ATI>. [Accessed 27 March 2019].
- [37] M. Goldstein, "BoNeSi - the DDoS Botnet Simulator," Deutsches Forschungszentrum fuer Kuenstliche Intelligenz, 1 December 2018. [Online]. Available: <https://github.com/markus-go/bonesi>. [Accessed 30 March 2019].

- [38] Onica, Emanuel and Felber, Pascal and Mercier, Hugues and Riviere, Etienne, "Confidentiality-preserving publish/subscribe: A survey," *ACM computing surveys (CSUR)*, vol. 49, no. 2, p. 27, 2016.
- [39] J. Asher, "PIP-9 Adding more Security checks to Pulsar Proxy," Github , 26 July 2018. [Online]. Available: <https://github.com/apache/pulsar/wiki/PIP-9:-Adding-more-Security-checks-to-Pulsar-Proxy>. [Accessed 4 April 2019].
- [40] Bradley, John and Sakimura, Nat and Jones, Michael B, "JSON web token (JWT)," *RFC*, 2015.
- [41] Yahoo, "Athenz," Yahoo, 2016. [Online]. Available: <https://github.com/yahoo/athenz>. [Accessed 12 April 2019].
- [42] aws, "I/O Characteristics and Monitoring," Amazon, 2019. [Online]. Available: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-io-characteristics.html>. [Accessed 16 April 2019].
- [43] Hameed, Sufian and Ahmed Khan, Hassan, "SDN based collaborative scheme for mitigation of DDoS attacks," *Future Internet*, vol. 10, no. 3, p. 23, 2018.
- [44] M. J. Schwartz, "Mirai Aftermath: China's Xiongmair Details Webcam Recall," InfoSec, 26 October 2016. [Online]. Available: <https://www.bankinfosecurity.com/mirai-aftermath-chinas-xiongmair-details-webcam-recall-a-9484>. [Accessed 23 April 2019].
- [45] A. Cassandra, "Manage massive amounts of data, fast, without losing sleep," Apache, 2019. [Online]. Available: <http://cassandra.apache.org/>. [Accessed 20 April 2019].
- [46] A. Fitzpatrick, "These Are Some of the Devices Vulnerable to Mirai," Time, 27 October 2016. [Online]. Available: <https://time.com/4543132/these-are-some-of-the-devices-vulnerable-to-mirai/>. [Accessed 10 April 2019].
- [47] Ben Herzberg, Igal Zeifman, Dima Bekerman, "Breaking Down Mirai: An IoT DDoS Botnet Analysis," imperva, 26 October 2016. [Online]. Available: https://www.imperva.com/blog/malware-analysis-mirai-ddos-botnet/?utm_campaign=Incapsula-moved. [Accessed 11 April 2019].
- [48] Ben-Shimol, Snir, "Let's discuss facts: An insight into Mirai's source-code," Radware, 3 November 2016. [Online]. Available: <https://blog.radware.com/security/2016/11/insight-into-mirais-source-code/>. [Accessed 5 April 2019].
- [49] S. Ragan, "Here are the 61 passwords that powered the Mirai IoT botnet," CSO, 3 October 2016. [Online]. Available: <https://www.csoonline.com/article/3126924/here-are-the-61-passwords-that-powered-the-mirai-iot-botnet.html>. [Accessed 10 April 2019].
- [50] Kumar, Ayush and Lim, Teng Joon, "Early Detection of Mirai-Like IoT Bots in Large-Scale Networks through Sub-sampled Packet Traffic Analysis," in *Future of Information and Communication Conference*, 2019.
- [51] Hakim, Achmad Khalif and Abdurrohman, Maman and Yulianto, Fazmah Arif, "Improving DDoS Detection Accuracy Using Six-Sigma in SDN Environment," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 8, pp. 365--370, 2018.
- [52] Stephanie Ding, Julian Bunn, Sherry Wang, "Machine Learning for Cybersecurity: Network-based Botnet Detection Using Time-Limited Flows," CURJ - Caltech Undergraduate Research , 11 July 2018. [Online]. Available: <https://curj.caltech.edu/2018/07/11/machine-learning-for-cybersecurity-network-based-botnet-detection-using-time-limited-flows/>. [Accessed 20 April 2019].

Appendices

Appendix 1 : Gantt Chart



Annex 1.1 Risk Mitigation :

Table 10 Risk mitigation overview

Risk Factor	Consequence Assessment	Risk Mitigation Planning	Severity
VM isolation	Unable to emulate nature of WAN systems Results in uncertainty around system functionality and pulsar collaboration	Solution 1 : Use Quagga to setup router config and set firewalls in config Solution 2: Manually install iptables drop rules	High
Domain Setup	Need virtual environment to test IDPS, without it will require live solution which will be harder	Solution 1 : Use docker containers , managed by Kubernetes. (must have native Linux) Solution 2 : Set up a series of VMs and set up route connections manually Solution 3 : Setup a series of raspberry pis to run virtual domains on Solution 2 : Set up a series of VMs and set up route connections manually Solution 3 : Setup a series of raspberry pis to run virtual domains on	Critical
Use Distributed CIDPS Architecture	Centralized system just on a single vm will run into scalability problem like existing solutions Centralized and decentralized have SPoF problems	Solution 1 : Use AWS and deploy instances on the cloud using terraform Solution 2 : Containerize solution using docker and Kubernetes Solution 3 : Use multiple VMs to run a pulsar standalone cluster across multiple nodes	High
Make/Use A NIDS	Need a NID to be used as a reference point of standalone IDPS vs collaboration and No system viability or comparison	Solution 1 : Use off the shelf NIDS that detects mirai botnet attacks Solution 2: Source comprehensive NID solution Solution 3 : Create a coarse grain NIDS manually	Critical
Use the Mirai Source Code	Uncertainty around results and if the system will work in a production environment against these attacks	Solution 1 : Use the Mira Source code Solution 2: TcpReplay a Mira dataset Solution 3 : emulate the attack	High
Cant exchange collaborative messages	No objectives can be achieved , no credibility and o results.	Solution 1: reconsider packets to send over network Solution 2 : Reconsider a different pub sub framework Solution 3: look at Apache Kafka	critical
Exploitable pulsar security	Pulsar would in turn not be a viable solution, project viability tarnished	Solution 1: consider pluggable security features Solution 2 : Put pulsar in an air-gapped network Solution 3 : leave pulsar unprotected but share as little information as possible	medium

Appendix 2 : Github + User documentation

The DIPA GitHub repository contains user documentation and files of how to deploy the proposed architecture. This URL provides the user with all configuration files used in the aws deployment with user guides of how to set up the proposed test network. Attack tools and traffic generators are all stored to allow users to replicate the same network conditions. This project analyses the deployed_controller/DIPA-Controller however this project also includes a local deployment controller used for the project showcase. The GitHub also contains minutes throughout the 30 week process. Feel free to access and test the proposed system. To access this use the public URL below :

<https://github.com/benkasaurus/DIPA-Collaborative-Pulsar-IDPS>

The following files are a proof of concept and must not be deployed into production without further development and considering system exploits and limitations deduced from this report. Development design cases are also included to step through each design phase.

benkasaurus / DIPA-Collaborative-Pulsar-IDPS

Watch 0 Star 0 Fork 0

Code Issues Pull requests Projects Wiki Insights Settings

Collaborative intrusion detection and prevention system using apache pulsar (pub sub) framework . Distributed architecture used in the cloud (AWS)

Manage topics

25 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find File Clone or download

40133635 Added Timing Function Latest commit ed29d4f 20 hours ago

attackEmulation	Added attack tool readMe	2 days ago
mininet	Migration from private developmental repo	2 days ago
minutes	Migration from private developmental repo	2 days ago
pulsar	Fixed Deployment Yaml	2 days ago
report	Migration from private developmental repo	2 days ago
ryu	Added Timing Function	20 hours ago
README.md	Added important reminder	2 days ago
requirements.txt	Added LAN requirements	2 days ago
requirements_pulsar.txt	Added pulsar requirements	2 days ago

README.md

DIPA-Collaborative-Pulsar-IDPS

Collaborative intrusion detection and prevention system using apache pulsar (pub sub) framework . Distributed architecture used in the cloud (AWS). This project investigates recent evolution in the mirai botnet attacks . The proposed architecture utilizes a deployed AWS distributed series of clusters with apache pulsar installed . Classified attacks are pushed to the pub sub framework , this results in neighbouring domains consuming alerts.

NOTE

This project is part of ELE4001 Final year project This project s experimental and a proof of concept to detail Pulsar's Viability in collaborative IDPS

Figure 47 Github User Documentation

Appendix 3 : AWS Deployment

Annex 3.1 : Aws Billing Forecast

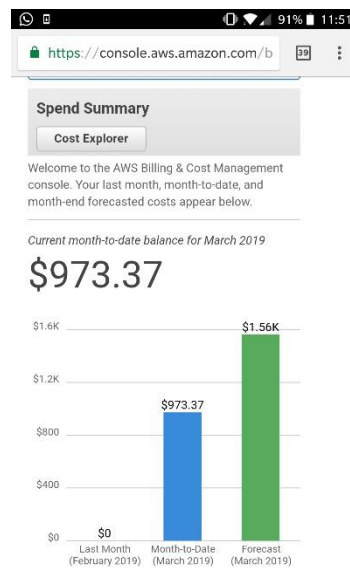


Figure 48 AWS cost and resource usage per month

Annex 3.2 : AWS Deployment instances

```

aws_instance.bookie.0: Still modifying... (ID: i-03bd4062da6cd1745, 20s elapsed)
aws_elb.default: Still creating... (10s elapsed)
aws_instance.bookie.2: Still modifying... (ID: i-078c8342ad1796392, 30s elapsed)
aws_instance.bookie.0: Still modifying... (ID: i-03bd4062da6cd1745, 30s elapsed)
aws_instance.bookie.1: Still creating... (30s elapsed)
aws_instance.bookie[1]: Creation complete after 30s (ID: i-097627f3d210ac2bb)
aws_elb.default: Creation complete after 12s (ID: pulsar-elb)
aws_instance.bookie.2: Still modifying... (ID: i-078c8342ad1796392, 40s elapsed)
aws_instance.bookie.0: Still modifying... (ID: i-03bd4062da6cd1745, 40s elapsed)
aws_instance.bookie[2]: Modifications complete after 41s (ID: i-078c8342ad1796392)
aws_instance.bookie[0]: Modifications complete after 44s (ID: i-03bd4062da6cd1745)

Apply complete! Resources: 5 added, 2 changed, 0 destroyed.

Outputs:
dns_name = pulsar-elb-...us-west-2.elb.amazonaws.com
pulsar_service_url = pulsar://pulsar-elb-...us-west-2.elb.amazonaws.com:6650
pulsar_ssh_host = ...
pulsar_web_url = http://pulsar-elb-...us-west-2.elb.amazonaws.com:8080
apache@apache-VirtualBox:~/incubator-pulsar/deployment/terraform-ansible/aws$
  
```

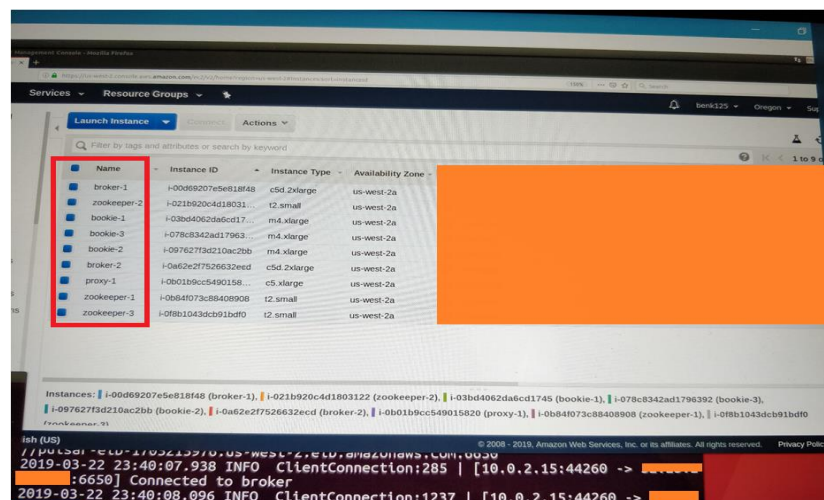


Figure 49 AWS Deployment setup

Appendix 4 : Software Listings

Table 11 Software Listing versions

Paramter	Value
OS	Ubuntu 18.4
Ryu	4.30
Aws OS	Red Hat 7.4
Pulsar	2.3.0
OpenFlow	1.3
Mininet	2.2.1
Pulsar client	2.3.0
Python	3.0
Ansible	2.4.1.0
Aws-cli	1.16.116
terraform	0.11.13
Terraform-inventory	0.8
BoneSi	1.15

Appendix 5: Mirai Botnet

Annex 5.1 Vulnerable Device list

Username/Password	Manufacturer	Link to supporting evidence
admin/123456	ACTi IP Camera	https://ipvm.com/reports/ip-cameras-default-passwords-directory
root/anko	ANKO Products DVR	http://www.cctvforum.com/viewtopic.php?f=3&t=44250
root/pass	Axis IP Camera, et. al	http://www.cleancss.com/router-default/Axis/0543-001
root/vizxv	Dahua Camera	http://www.cam-it.org/index.php?topic=5192.0
root/888888	Dahua DVR	http://www.cam-it.org/index.php?topic=5035.0
root/666666	Dahua DVR	http://www.cam-it.org/index.php?topic=5035.0
root/7ujMko0vixv	Dahua IP Camera	http://www.cam-it.org/index.php?topic=9396.0
root/7ujMko0admin	Dahua IP Camera	http://www.cam-it.org/index.php?topic=9396.0
666666/666666	Dahua IP Camera	http://www.cleancss.com/router-default/Dahua/DH-IPC-HDW4300C
root/dreambox	Dreambox TV receiver	https://www.satellites.co.uk/forums/threads/reset-root-password-plugin.101146/
root/zlxx	EV ZLX Two-way Speaker?	?
root/juante dh	Guangzhou Juan Optical	https://news.ycombinator.com/item?id=11114012
root/x3511	H.264 - Chinese DVR	http://www.cctvforum.com/viewtopic.php?f=56&t=34930&start=15
root/h3518	HiSilicon IP Camera	https://acassis.wordpress.com/2014/08/10/i-got-a-new-hi3518-ip-camera-modules/
root/klv123	HiSilicon IP Camera	https://gist.github.com/gabonator/74cdd6ab4f733ff047356198c781f27d
root/klv1234	HiSilicon IP Camera	https://gist.github.com/gabonator/74cdd6ab4f733ff047356198c781f27d
root/jvbdz	HiSilicon IP Camera	https://gist.github.com/gabonator/74cdd6ab4f733ff047356198c781f27d
root/admin	IPX-DDK Network Camera	http://www.ipxinc.com/products/cameras-and-video-servers/network-cameras/
root/system	IQin Vision Cameras, et. al	https://ipvm.com/reports/ip-cameras-default-passwords-directory
admin/meinsm	Mobotix Network Camera	http://www.forum-use-ip.co.uk/threads/mobotix-default-password.76/
root/54321	Packet8 VOIP Phone, et. al	http://webcache.googleusercontent.com/search?q=cache:W1phozQZURUJ:community.freepbx.org/t/packet8-atas-phones/415
root/00000000	Panasonic Printer	https://www.experts-exchange.com/questions/26194395/Default-User-Password-for-Panasonic-DP-C405-Web-Interface.html
root/realtek	RealTek Routers	
admin/1111111	Samsung IP Camera	https://ipvm.com/reports/ip-cameras-default-passwords-directory
root/xmhdipc	Shenzhen Anran Security Camera	https://www.amazon.com/MegaPixel-Wireless-Network-Surveillance-Camera/product-reviews/B00EB6FNDI
admin/smcadmin	SMC Routers	http://www.cleancss.com/router-default/SMC/ROUTER
root/ikwb	Toshiba Network Camera	http://faq.surveillixdvrssupport.com/index.php?action=artikel&cat=4&id=8&artlang=en
ubnt/ubnt	Ubiquiti AirOS Router	http://setuprouter.com/router/ubiquiti/airos-airgrid-m5hp/login.htm
supervisor/supervisor	VideoIQ	https://ipvm.com/reports/ip-cameras-default-passwords-directory
root/<none>	Vivitek IP Camera	https://ipvm.com/reports/ip-cameras-default-passwords-directory
admin/1111	Xerox printers, et. al	https://atyourservice.blogs.xerox.com/2012/08/28/logging-in-as-system-administrator-on-your-xerox-printer/
root/Zte521	ZTE Router	http://www.ironbugs.com/2016/02/hack-and-patch-your-zte-f660-routers.html

Figure 50 List of IoT devices , still vulnerable to Mirai [46]

Annex 5.2 Geo Locations of Mirai Botnet

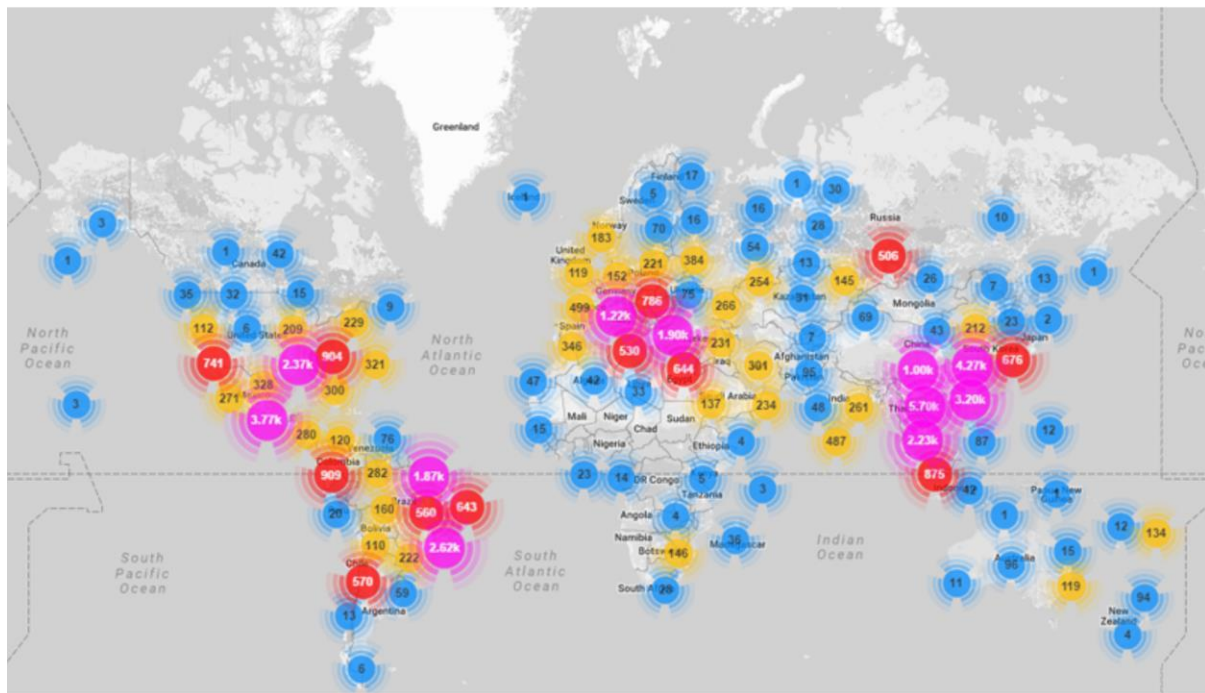


Figure 51 Geo location heatmap from 2016 Dyn DDoS Attack [47]

Annex 5.3 Dictionary Content used by the Mirai malware

```
#define ATK_VEC_UDP      0 /* Straight up UDP flood */
#define ATK_VEC_VSE      1 /* Valve Source Engine query flood */
#define ATK_VEC_DNS      2 /* DNS water torture */
#define ATK_VEC_SYN      3 /* SYN flood with options */
#define ATK_VEC_ACK      4 /* ACK flood */
#define ATK_VEC_STOMP     5 /* ACK flood to bypass mitigation devices */
#define ATK_VEC_GREIP     6 /* GRE IP flood */
#define ATK_VEC_GREETH    7 /* GRE Ethernet flood */
// #define ATK_VEC_PROXY   8 /* Proxy knockback connection */
#define ATK_VEC_UDP_PLAIN 9 /* Plain UDP flood optimized for speed */
#define ATK_VEC_HTTP     10 /* HTTP layer 7 flood */
```

Figure 52 Dictionary Content in main.go of mirai source code [48]

Annex 5.4 61 passwords that powered the mirai IoT botnet

USER:	PASS:	USER:	PASS:
-----	-----	-----	-----
root	xc3511	admin1	password
root	vizxv	administrator	1234
root	admin	666666	666666
admin	admin	888888	888888
root	888888	ubnt	ubnt
root	xmhdipc	root	klv1234
root	default	root	Zte521
root	juantech	root	hi3518
root	123456	root	jvbsd
root	54321	root	anko
support	support	root	zlxx.
root	(none)	root	7ujMko0vizxv
admin	password	root	7ujMko0admin
root	root	root	system
root	12345	root	ikwb
user	user	root	dreambox
admin	(none)	root	user
root	pass	root	realtek
admin	admin1234	root	00000000
root	1111	admin	1111111
admin	smcadmin	admin	1234
admin	1111	admin	12345
root	666666	admin	54321
root	password	admin	123456
root	1234	admin	7ujMko0admin
root	klv123	admin	1234
Administrator	admin	admin	pass
service	service	admin	meinsm
supervisor	supervisor	tech	tech
guest	guest	mother	fucker
guest	12345		
guest	12345		

Figure 53 Factory default password stored in Mirai SQL DB [49]

Appendix 6 : Complete test set

Annex 6.1 Control Plan Time

Table 12 Control Plane : 1 Bot

Bots = 1	100	200	500	1000	2000	5000	7500
Run 1:	0.071381923	0.241937846	0.538829618	1.056098384	2.128204199	4.625638738	11.9235827
Run 2:	0.068231192	0.224017294	0.510731839	0.837104129	2.851294913	5.92314113	11.35203441
Run 3:	0.084297236	0.253860063	0.462910304	0.912732405	2.410297239	5.312063912	10.43419952
Run 4:	0.072803219	0.240203818	0.492112035	0.862013915	2.631204122	4.923995766	11.44129479
Run 5:	0.071407644	0.23791478	0.483204751	1.142910332	1.904183942	4.782910023	12.80530417
Mean	0.073624243	0.23958676	0.49755771	0.962171833	2.385036883	5.113549914	11.59128312
Standard Deviation	0.006197039	0.010663481	0.028779479	0.131915122	0.379412547	0.519171593	0.866436527

Table 13 Control Plane : 10 Bots

Bots = 10	100	200	500	1000	2000	5000	7500
Run 1:	0.200782102	0.37317812	1.296368	3.38291862	5.17422953	7.754738579	12.941292
Run 2:	0.219475913	0.401519522	1.08741048	2.849623048	5.136630797	7.844492184	11.17056561
Run 3:	0.168401032	0.401328577	0.987692301	3.113948513	5.037737232	8.017619431	11.38198506
Run 4:	0.218403058	0.394759201	1.123957382	3.240763123	4.906917349	8.056418539	11.29701879
Run 5:	0.241750237	0.435129524	0.821148751	3.860050156	5.290931342	7.544970586	13.1582915
Mean	0.209762468	0.401182989	1.063315383	3.289460692	5.10928925	7.843647864	11.98983059
Standard Deviation	0.027311369	0.022237577	0.175310493	0.37447959	0.144941995	0.207756562	0.973555236

Table 15 Control Plane : 50 Bots

Bots = 50	100	200	500	1000	2000	5000	7500
Run 1:	0.721192	0.7391022	1.16033814	2.88855963	5.73236797	6.98203448	12.428194
Run 2:	0.746850407	0.810860211	1.02161899	3.951155413	5.342786321	7.049324254	12.15743985
Run 3:	0.653873238	0.724871904	0.929256105	3.265285031	6.672799185	6.798137133	12.03335465
Run 4:	0.650068913	0.747918297	1.257953416	3.509304593	6.515882225	6.60610617	12.27584063
Run 5:	0.736222887	0.775186678	1.304016212	3.906674871	5.115936555	6.923368772	12.04795136
Mean	0.701641489	0.759587858	1.134636572	3.504195908	5.875954451	6.871794162	12.1885561
Standard Deviation	0.046269531	0.034033864	0.157726969	0.446619146	0.694078818	0.174889443	0.165733458

Table 14 Control Plane : 100 Bots

Bots = 100	100	200	500	1000	2000	5000	7500
Run 1:	1.32522012	1.4872993	1.6830134	3.53658832	5.8758231	7.78339412	13.108554
Run 2:	1.243884266	1.48411118	1.643870566	4.237398034	6.174006365	8.434519116	13.10500319
Run 3:	1.305569966	1.508994425	1.54046125	4.029254134	6.084099902	8.383078482	13.05524468
Run 4:	1.296774038	1.458551267	1.657941675	3.768537446	6.29042056	8.419135894	14.21389647
Run 5:	1.316936462	1.550384347	1.541399535	3.776525481	5.97341136	8.660104973	13.05187614
Mean	1.29767697	1.497868104	1.613337285	3.869660683	6.079552257	8.336046517	13.3069149
Standard Deviation	0.031962176	0.034382107	0.067569502	0.269507932	0.162925719	0.327586347	0.507718827

Table 17 Control Plane : 250 Bots

Bots = 250	100	200	500	1000	2000	5000	7500
Run 1:	1.7329907	1.7739223	1.8495732	3.89539119	5.9833422	8.90394105	16.002379
Run 2:	1.883967539	2.322788619	2.58673475	4.674150227	6.656666402	10.77272706	13.85962421
Run 3:	1.849147333	1.970877627	2.041204437	4.161238417	6.615435051	9.630443857	11.70609573
Run 4:	2.055274379	2.482756745	2.817044338	4.320000337	6.162238514	9.316521103	13.88506573
Run 5:	1.869477271	2.136850777	2.852337252	4.614926378	6.471512763	8.287222032	16.41676777
Mean	1.878171444	2.137439214	2.429378795	4.33314131	6.377838986	9.38217102	14.37398649
Standard Deviation	0.115536228	0.279949216	0.458473878	0.322892538	0.293718689	0.925636618	1.900386191

Table 16 Control Plane : 500 Bots

Bots = 500	100	200	500	1000	2000	5000	7500
Run 1:	1.7829427	1.8219244	2.06107897	4.05837662	6.87451124	9.12063119	14.401293
Run 2:	1.962528348	1.842691573	3.196191635	5.702380458	6.572772853	11.13861297	16.14764118
Run 3:	1.996800617	2.149004018	2.739592981	5.504998738	6.871107181	8.573592812	14.35282293
Run 4:	1.966454216	2.284221185	3.368478921	4.495773852	8.260415254	10.22230887	14.16225212
Run 5:	2.180158981	2.467651554	3.425800335	4.237934315	7.701920932	9.218505507	16.16385165
Mean	1.977776972	2.113098546	2.958228568	4.799892797	7.256145492	9.654730269	15.04557218
Standard Deviation	0.141104124	0.28025984	0.569261375	0.753291158	0.701457566	1.020841002	1.017393308

Annex 6.2 Pulsar Publish Time

Table 18 Pulsar Time : 1 Bot

Bots = 1	100	200	500	1000	2000	5000	7500
Run 1:	0.1642008	0.153113	0.149617	0.1353396	0.153113	0.142239	0.152851
Run 2:	0.174299726	0.193594952	0.16099753	0.149350806	0.189233304	0.170157008	0.20234188
Run 3:	0.137212691	0.172262207	0.146177026	0.106223197	0.096287665	0.114881789	0.14612743
Run 4:	0.179337233	0.140878525	0.189844126	0.143816228	0.101180861	0.192032412	0.2626954
Run 5:	0.120413951	0.154230187	0.196397416	0.163815553	0.110585935	0.133147662	0.22430902
Mean	0.15509288	0.162815774	0.16860662	0.139709077	0.130080153	0.150491574	0.19766495
Standard Deviation	0.025307978	0.020528589	0.023156806	0.021397139	0.03995852	0.030615942	0.04905391

Table 19 Pulsar Time : 10 Bots

Bots = 10	100	200	500	1000	2000	5000	7500
Run 1:	0.1289212	0.1512292	0.124122	0.139981	0.13302	0.1772119	0.141294
Run 2:	0.141889272	0.164224032	0.16864381	0.173845304	0.104876634	0.186062081	0.19123439
Run 3:	0.203552414	0.149262266	0.11354227	0.141210502	0.187929598	0.192288863	0.15962299
Run 4:	0.217716725	0.159934517	0.193260179	0.140405058	0.172530949	0.140274613	0.09444617
Run 5:	0.38384953	0.156392103	0.116481795	0.112941026	0.159262835	0.113726405	0.11574483
Mean	0.215185828	0.156208424	0.143210011	0.141676578	0.151524003	0.161912773	0.14046848
Standard Devia	0.101738829	0.006148359	0.035745007	0.021594373	0.032958018	0.033652991	0.0376632

Table 20 Pulsar Time : 50 Bots

Bots = 50	100	200	500	1000	2000	5000	7500
Run 1:	0.094321	0.201549	0.0801701	0.1503874	0.0903552	0.229402	0.179367
Run 2:	0.092211913	0.157041571	0.148713714	0.130837008	0.138492303	0.185929024	0.14504371
Run 3:	0.181060171	0.162223791	0.197070714	0.164802323	0.188983529	0.216621922	0.104781
Run 4:	0.264609682	0.188148058	0.163408213	0.164390062	0.234607338	0.225886344	0.13362783
Run 5:	0.234576631	0.158242798	0.203544398	0.186047886	0.142913119	0.209147678	0.12525988
Mean	0.17335588	0.173441044	0.158581428	0.159292936	0.159070298	0.213397393	0.13761589
Standard Devia	0.079002292	0.020199688	0.049426966	0.020378971	0.054788044	0.017285039	0.02759091

Table 21 Pulsar Time : 100 Bots

Bots = 100	100	200	500	1000	2000	5000	7500
Run 1:	0.347071	0.139044	0.182914	0.172642	0.113804	0.094023	0.33852
Run 2:	0.17349679	0.1137516	0.11600398	0.12680767	0.15801713	0.15038407	0.1276275
Run 3:	0.23642986	0.1038572	0.19567685	0.17905485	0.15897632	0.13952246	0.3759396
Run 4:	0.12457327	0.1947164	0.19844302	0.1065805	0.18664879	0.18108647	0.1155231
Run 5:	0.1411607	0.1295238	0.12715622	0.15284832	0.19319738	0.19029336	0.3064939
Mean	0.20454632	0.1361786	0.16403881	0.14758667	0.16212872	0.15106187	0.2528208
Standard Devia	0.09042586	0.03545	0.0393972	0.03065523	0.03133934	0.03816948	0.1223799

Table 22 Pulsar Time : 250 Bots

Bots = 250	100	200	500	1000	2000	5000	7500
Run 1:	0.16233	0.148202	0.109611	0.182048	0.20152	0.094021	0.404924
Run 2:	0.13083714	0.147738	0.13994268	0.10662623	0.12821837	0.11833055	0.1762928
Run 3:	0.10850532	0.1763666	0.21240772	0.18796027	0.25897593	0.28997906	0.2007375
Run 4:	0.10847857	0.1703604	0.2234979	0.16939558	0.16186369	0.1003876	0.2718301
Run 5:	0.10707571	0.1817928	0.23356646	0.13862305	0.15701415	0.29700537	0.1500711
Mean	0.12344535	0.164892	0.18380515	0.15693062	0.18151843	0.17994472	0.2407711
Standard Devia	0.02388425	0.0159689	0.0554493	0.03396615	0.05056528	0.10406642	0.1023444

Table 23 Pulsar Time : 500 Bots

Bots = 500	100	200	500	1000	2000	5000	7500
Run 1:	0.167402	0.323954	0.189003	0.108421	0.1842193	0.192783	0.331965
Run 2:	0.12556685	0.2445004	0.18545807	0.17307709	0.16515503	0.10995993	0.2137916
Run 3:	0.1037924	0.2831746	0.1056365	0.17191685	0.15302139	0.18313869	0.1781554
Run 4:	0.15542745	0.1767667	0.13617303	0.10362817	0.12786072	0.14977906	0.2309045
Run 5:	0.1311733	0.1830518	0.10314067	0.17241321	0.10899417	0.15503337	0.2307974
Mean	0.1366724	0.2422895	0.14388225	0.14589126	0.14785012	0.15813881	0.2371228
Standard Devia	0.02515317	0.0635372	0.04167276	0.03643489	0.02982566	0.03250364	0.0572172

Annex 6.3 Vm protection time

Table 24 Vm Protection Time : 100 Flows

100 Flows	Local 2Vms	Local 3Vms	Global (FYP) 2 VMs	Global (FYP) 3VMS
Run 1	2.012497356	3.022038434	1.143333922	1.176225436
Run 2	2.011501022	3.016042187	1.089659215	1.150634291
Run 3	2.013513994	3.017469733	1.192224237	1.15317436
Run 4	2.009272876	3.019874965	1.125112667	1.185212396
Run 5	2.013959943	3.018736291	1.067905501	1.181553491
Mean	2.012149038	3.018832322	1.123647108	1.169359995
Standard Deviation	0.001867889	0.002292214	0.048360223	0.016276832

Table 27 Vm Protection Time : 200 Flows

200 Flows	Local 2Vms	Local 3Vms	Global (FYP) 2 VMs	Global (FYP) 3VMS
Run 1	2.067549742	3.108342384	1.170916785	1.221027916
Run 2	2.061889364	3.080636482	1.279755066	1.210499458
Run 3	2.059071741	3.097681182	1.138874323	1.175931781
Run 4	2.064057325	3.07751738	1.187239618	1.214863441
Run 5	2.058730048	3.079302167	1.165878554	1.217450431
Mean	2.062259644	3.088695919	1.188532869	1.207954605
Standard Deviation	0.003673373	0.013646203	0.053881067	0.018308138

Table 26 Vm Protection Time : 500 Flows

500 Flows	Local 2Vms	Local 3Vms	Global (FYP) 2 VMs	Global (FYP) 3VMS
Run 1	2.17813196	3.31625867	1.227159697	1.225854099
Run 2	2.177613031	3.350370336	1.203781194	1.223290626
Run 3	2.189062309	3.243139691	1.196454855	1.205557148
Run 4	2.141057485	3.224282932	1.208666191	1.226577642
Run 5	2.140603613	3.274020543	1.228766172	1.203097882
Mean	2.165293679	3.281614434	1.212965622	1.216875479
Standard Deviation	0.022795439	0.051847062	0.014375095	0.011552365

Table 25 Vm Protection Time : 1000 Flows

1000 Flows	Local 2Vms	Local 3Vms	Global (FYP) 2 VMs	Global (FYP) 3VMS
Run 1	2.296219908	3.539117507	1.414726338	1.334550953
Run 2	2.267938917	3.491089842	1.389838742	1.342796715
Run 3	2.307757457	3.499503149	1.423708441	1.377119926
Run 4	2.221604271	3.520264927	1.38133517	1.340529124
Run 5	2.223173701	3.495882772	1.420048239	1.417011349
Mean	2.263338851	3.509171639	1.405931386	1.362401613
Standard Deviation	0.040094736	0.020104982	0.019082764	0.034775239

Table 28 Vm Protection Time : 5000 Flows

5000 Flows	Local 2Vms	Local 3Vms	Global (FYP) 2 VMs	Global (FYP) 3VMS
Run 1	5.107968472	8.115332921	2.691594574	2.807541541
Run 2	5.145785306	7.454461738	2.731229895	2.661979698
Run 3	5.397073018	7.889550945	2.677567382	2.770349213
Run 4	5.185756106	7.665808418	2.563129687	2.80107608
Run 5	5.143746902	8.146410683	2.551884334	2.638732807
Mean	5.196065961	7.854312941	2.643081174	2.735935868
Standard Deviation	0.115690023	0.295850121	0.080656911	0.079801078

Table 29 Vm Protection Time : 7500 Flows

7500 Flows	Local 2 Vms	Local 3 Vms	Global (FYP) 2 Vms	Global (FYP) 3Vms
Run 1	7.275989636	13.77170924	3.775345756	3.863305896
Run 2	7.152584429	11.84953503	3.763400504	3.989775261
Run 3	7.332148158	13.87647508	3.824113503	3.976586041
Run 4	7.1745973	12.09428562	3.827582924	3.982824787
Run 5	7.351851967	14.26884861	3.740443853	3.912194992
Mean	7.257434298	13.17217071	3.786177308	3.944937395
Standard Deviation	0.090411035	1.114606982	0.038344603	0.055188976

Annex 6.4 Detection Accuracy

Table 30 Detection Accuracy : 1 Bot

Bots = 1	100	200	500	1000	2000	5000	7500
Run 1:	1	1	1	1	1	1	0
Run 2:	1	1	1	1	1	0	1
Run 3:	1	1	1	1	1	1	1
Run 4:	1	1	1	1	1	1	0
Run 5:	1	1	1	1	1	1	0
Mean	1	1	1	1	1	0.8	0.4
Standard Devia	0	0	0	0	0	0.447214	0.547723

Table 31 Detection Accuracy : 1 Bot

Bots = 10	100	200	500	1000	2000	5000	7500
Run 1:	1	1	1	1	1	0.6	0.3
Run 2:	1	1	1	1	0.9	0.7	0
Run 3:	1	1	1	1	0.9	0.5	0.5
Run 4:	1	1	1	1	1	0.6	0.3
Run 5:	1	1	1	0.9	1	0.6	0.1
Mean	1	1	1	0.98	0.96	0.6	0.24
Standard Devia	0	0	0	0.044721	0.054772	0.070711	0.194936

Table 32 Detection Accuracy : 100 Bots

Bots = 100	100	200	500	1000	2000	5000	7500
Run 1:	0.96	0.8	0.82	0.65	0.55	0.23	0.32
Run 2:	1	0.82	0.81	0.61	0.67	0.37	0.24
Run 3:	1	0.76	0.67	0.59	0.45	0.34	0.19
Run 4:	0.95	0.8	0.73	0.78	0.46	0.21	0.18
Run 5:	0.88	0.92	0.8	0.69	0.53	0.2	0.21
Mean	0.958	0.82	0.766	0.664	0.532	0.27	0.228
Standard Deviation	0.049193	0.06	0.064265	0.075366	0.088431	0.079057	0.056303

Table 33 Detection Accuracy : 250 Bots

Bots = 250	100	200	500	1000	2000	5000	7500
Run 1:	1	0.948	0.768	0.592	0.376	0.188	0.186
Run 2:	1	0.872	0.788	0.532	0.448	0.236	0.2
Run 3:	0.996	0.948	0.816	0.452	0.404	0.204	0.192
Run 4:	0.968	0.964	0.836	0.464	0.516	0.268	0.096
Run 5:	0.948	0.848	0.726	0.468	0.488	0.348	0.116
Mean	0.9824	0.916	0.7868	0.5016	0.4464	0.2488	0.158
Standard Deviation	0.023426	0.05223	0.042793	0.05937	0.05773	0.063397	0.048249

Table 34 Detection Accuracy : 500 Bots

Bots = 500	100	200	500	1000	2000	5000	7500
Run 1:	0.846	0.912	0.878	0.598	0.402	0.062	0.202
Run 2:	0.902	0.956	0.932	0.81	0.294	0.208	0.092
Run 3:	0.956	0.824	0.822	0.584	0.246	0.224	0.122
Run 4:	0.93	0.862	0.86	0.636	0.602	0.158	0.086
Run 5:	0.878	0.894	0.838	0.624	0.444	0.176	0.08
Mean	0.9024	0.8896	0.866	0.6504	0.3976	0.1656	0.1164
Standard Deviation	0.043044	0.049988	0.042591	0.091558	0.139344	0.063457	0.050505