Detecting Concurrent Distributed Anomalies in Multi-Domain SDN

Rohit Katiyar

A Thesis Submitted to

Indian Institute of Technology Hyderabad

In Partial Fulfillment of the Requirements for

The Degree of Master of Technology

Department of Computer Science & Engineering

June 2016

Declaration

Rohit Katiyar

(Signature)

(Rohit Katiyar)

CS13M1016

(Roll No.)

Approval Sheet

This Thesis entitled Detecting Concurrent Distributed Anomalies in Multi-Domain SDN by Rohit Katiyar is approved for the degree of Master of Technology from IIT Hyderabad

[illegible]

(T. Bheemarjuna Reddy) Internal Examiner
Dept. of Computer Science & Engineering
IITH

[illegible]

(Pravati Swain.) External Examiner

[illegible]

(Dr. Kotaro Kataoka) Adviser
Dept. of Computer Science & Engineering
IITH

[illegible]

(ANTONY FRANKLIN) Chairman
Dept. of Computer Science & Engineering
IITH

Acknowledgements

To start with, I would like to thank my adviser, Dr. Kotaro Kataoka for his valuable suggestion. His constant guidance, patience and immense knowledge were very helpful. Next, I would like to thank my NSA adviser Dr. Bheemarjuna Reddy Tamma and Practical Networking Group of IITH (PRANET) for the friendly advice and words of encouragement. I am also thankful to my seniors and friends for their motivation. Finally, I am grateful to my family for always believe in me.

Dedication

To my family & friends

Abstract

Anomaly Detection is essential to understand the difference between normal and anomalous traffic. An anomaly maybe something that disrupts the network or allows an attacker unwanted access to modify or steal data. An anomaly in a network may occur at various layers. There are application specific anomalies which attack a specific application or group of applications e.g., MAC layer anomalies mostly DOS/DDOS attacks and anomaly at transport layer is SYN-Flood Attack. It's a very widely studied concept in both traditional and Software-defined Networking (SDN).

The accuracy of anomaly detection is directly dependent on network flow measurement, which often introduces overhead in the network. This overhead is effected packet sampling rate that is essential to detect anomalies. Using centralized SDN architecture, the overhead on traffic measurement has been reduced to a certain extent. However, it still incurs a significant overhead and misses a special set of anomalies stated as Concurrent Distributed Anomalies (CDA). Hence, domain specific distributed SDN controllers have been proposed with additional benefits of being cost effective, scalable and reliable.

The main goal of this thesis is to define what can be considered an concurrent distributed anomaly and how these anomalies are undetectable in centralized SDN architecture. This is achieved by elaborating concurrent distributed anomalies scenarios and implementing detection technique using multi-domain SDN architecture. The proposed solution scaled the network into multi-domain SDN architecture. It comprises of two entities viz., a multi-domain SDN network and a CDA detector. The network is divided into multiple non-overlapping SDN domains and each domain consists of a SDN controller and a part of the network infrastructure. Detection of anomaly in one domain is independent of detecting anomalies in other domains. This reduces the computational load at centralized point of anomaly detection by distributing the traffic flow measurement among domains. Hence, this helps in detecting anomalies across the domains.

We evaluated the system in the terms of sensitivity, scalability and responsiveness of the system. The sensitivity, which came out to be 80% defines our design detects concurrent anomalies correctly. This enables the system to achieve better responsiveness to detect CDA with lesser delay. The solution was found to be scalable enough for deploying over large network. Our results shows that distributed architecture helps in detecting CDA as compared to centralized architecture without imposing additional overhead.

The proposed work can be used to detect coordinated attacks, which either caused the network damage or trying to find vulnerability in the system for the larger attack later in the future. Thus, this solution can be used for real-time early detection

Contents

Declaration ii
Approval Sheet iii
Acknowledgements iv
Abstract vi

Nomenclature **viii**

1 Introduction **1**
1.1 Anomaly Detection 1
1.2 Main Objective 2

2 Software Defined Networking **3**
2.1 Overview 3
2.2 OpenFlow Protocol 3
2.2.1 Message Types 4
2.2.2 Switch 5
2.2.3 Controller 5

3 Related Work **6**

4 Proposed Work **7**
4.1 Concurrent Distributed Anomalies (CDAs) 7
4.1.1 What is CDAs 7
4.1.2 Difficulty in Detecting CDAs 7
4.2 SDN Domain Functionality 9
4.2.1 Traffic Statistic Probe 9
4.2.2 Anomaly Detection 10
4.2.3 Anomaly Classifier 10
4.2.4 Messenger Client 11
4.3 CDA Detector Functionality 12
4.3.1 Messenger Server 12
4.3.2 Centralized Data Storage 12
4.3.3 CDA Probe 13
4.3.4 CDA Reporter 13
4.4 Implementation 13

5 Experiments and Results **15**
5.1 System Validation 15
5.2 Performance Evaluation 16
5.2.1 System Sensitivity 16
5.2.2 Scalability 17
5.2.3 Response Time 19
5.2.4 Performance-Gain 19
5.2.5 Complexity 20

6 Discussion **21**
6.1 System Improvement Heuristics 21
6.1.1 Interpret Planned Anomalies 21
6.1.2 SDN based network traffic knowledge 21
6.1.3 System Usability 21
6.2 Misc Topics in Security of SDN 22
6.2.1 Bootstrapping Flow Aggregation for Anomaly Detection in SDN 22
6.2.2 Addressing problem of collaboration of hacked switches in SDN 23
6.2.3 Detection of Cache Pollution in ICN using SDN 25

7 Conclusion and Future Work **26**

References **27**

Chapter 1

Introduction

1.1 Anomaly Detection

Anomaly detection in networks is stated as the prediction of anomalous traffic before it causes faults in the network. This feature is essential to provide a reliable, quality of service (QoS) and efficient load balancing scheme. The problem of network anomaly detection is of primary importance in network security due to various reasons such as network intrusion, denial-of-service attacks, buffer overflow attacks etc. Networks are targeted daily by attackers seeking to disrupt or disable them and the traffic they generate are the most common anomalies in the network. When the bandwidth of the network gets flooded by such anomalies, denial-of-service attacks may happen which eventually degrades the quality of the network. For large organizations, continuous denial-of-service attacks may result in the loss of several important functionality and thus lead to a decrease in the performance of the organization's services. Thus, it is very important to improve the accuracy of such anomaly detection mechanisms.

To maintain network availability, the monitoring system must detect and diagnose potential network problems and initiate appropriate recovery or mitigation actions. In traditional networks, anomaly detection has been done using proper placement of Intrusion Detection System (IDS) [1]. Drawbacks of IDS are: low detection accuracy, unbalanced detection rate for different types of attacks, high false positive and predefined policy based system with high computing power required for operation [2].

Most of the anomaly detection in traditional networks is done using machine learning techniques [3] which consist of following two steps:

- In the training phase, the behavior of the network is observed in an idle condition i.e., trusted network and machine learning techniques are used to train the system for such normal behavior
- In the detection phase, this training is compared against the current behavior of the system, and any deviations are triggered as potential attacks.

SDN [4] was proposed with a great functionality which changes the way that existing network architectures works in term of specializing device operation, intelligent packet forwarding and network deployment to reach an intelligent network. In SDN architecture [4], the actions in data plane are controlled by a programmable, logically centralized control plane, which can easily interact with other network management systems. Anomaly detection on SDN introduces some benefits in terms of easiness of deploying the monitoring points and flexible flow counting. This assists in acquiring packet sampling rate of the network in a better way.

Previously, anomaly detection has been done on centralized SDN architecture [5], wherein the network is programmed and controlled as a single entity. However, the centralized SDN controller faces the issues of scalability, reliability and single point of failure. Hence, a suggestion to build a logically centralized, but physically distributed control plane has been proposed [6]. This technique enjoys the advantages of the distributed architecture meanwhile keeping the simplicity of the centralized system. In anomaly detection, collecting statistics is one of the core functionalities of the controller. However, centralized statistic collection in SDN does not scale in some metrics, therefore the system may not be able to pick out a set of short-lived anomalies which can be Concurrent Distributed Anomalies (CDAs). CDA includes malicious traffic which is generated by a group of nodes simultaneously at multiple locations in the network. The collective impact of this event affects the core network and makes it congested. This increases the overhead on centralized anomaly detector which leads to failure in detecting CDA. The biggest difficulty of detecting CDA in centralized architecture is the latency in detection. As a result there is a delay between the cause and consequences in the network. Thus, we use distributed architecture to solve this problem without introducing additional load into the network.

1.2 Main Objective

The main goal of this work is to propose an architecture which detects concurrent distributed anomalies. This architecture brings the advantage of fast computational at monitoring points that lower down the overhead on the controller. In this architecture the network is divided into multiple domains in which each controller detects anomalies generated in its vicinity(domain); which then helps in the detection of distributed anomalies. The main contributions of this thesis are as follows:

- To solve the problem of detecting concurrent and distributed anomalies in feasible multi-domain SDN architecture
- Evaluation of system in terms of efficiency, scalability and sensitivity with different distributed attacks comprising of numerous hosts

Chapter 2

Software Defined Networking

2.1 Overview

Software Defined Networking (SDN) is a network paradigm as shown in Figure 2.1 which separates the control plane from the data plane. This paradigm is different from traditional network in the term of packet processing. In traditional, network nodes have capability to decide how to process incoming packets. In SDN, those two tasks are decoupled. The switches forwarding the incoming packets to external entity called controller. This external entity is taking a packet forwarding decision. SDN provides a programmable network, in which the entire traffic can be adjusted dynamically or controlled by the centralized program.
Open Networking Foundation (ONF) was founded in March 2011 which state SDN architecture is dynamic, manageable, cost-effective and adaptable. The gole of this organization is the promotion and adoption of Software Defined Networking through open standards development. Later, ONF introduces OpenFlow protocol, which define the communication between switches and controllers

2.2 OpenFlow Protocol

The OpenFlow protocol cab be adopted as standard for SDN. This ensures communication using control plane and data plane between the controller and OpenFlow-enabled switches. Each switch maintain an internal flow table, which contains a set of rules called as Flow Rules. These rules is used by switch to process incoming packets on the data plane. The controller is managed the flow table, which takes the decision of adding, deleting or modify flows. To make this communication secure , the controller and the switch are connected via a secure channel, usually TLS/SSL.

The flow table entry consists of the following components

- **Header Field** – This field is matched against the arriving packets
- **Actions** – This is applied to the matching packets
- **Counters** – This increments every time if a packet matches

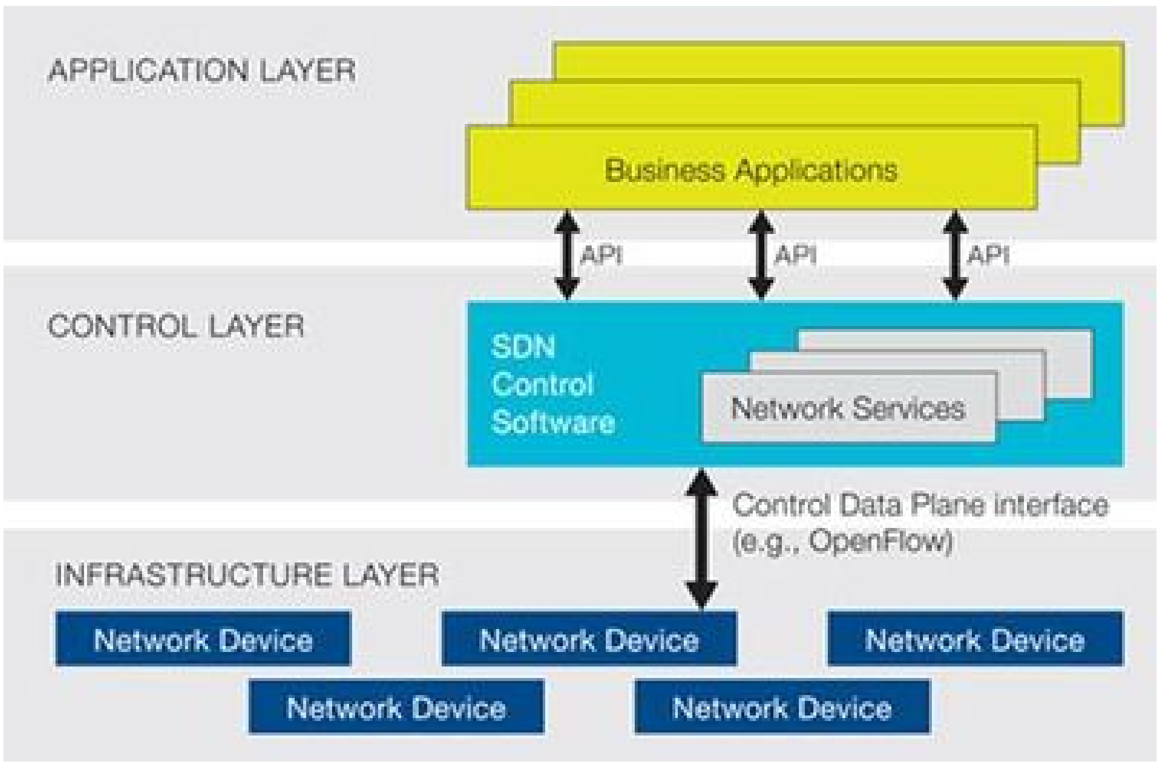

Figure 2.1: SDN Architecture (source: ONF homepage)

2.2.1 Message Types

The OpenFlow specification defines three classes of messages:

Asynchronous

- Packet_In: If an arriving packet does not match any flow table entry, a PACKET_IN message is sent to the controller
- Flow_removed: A flow can be removed for different reasons, i.e. a timeout (hard timeout or idle timeout)
- Port_status: The switch should send this messages on port configuration changes
- Error: The switch notifies the controller of errors

Controller-to-Switch

Controller-to-switch messages are initiated by the controller. These message are generally one way and switch may not give an answer. Six types are defined in the OpenFlow specification:

- Features: The controller sends a FEATURE_REQUEST message on connection and the switch answer with a FEATURE_REPLY specifying its capabilities
- Configuration: The controller can set and request configuration parameters
- Modify-State: Messages that are used by the controller to create, modify or delete flows

- Read-State: The controller can request statistics from flow tables, ports or flow table entries
- Send-Packet: the controller may send packets and asked the switch to output them to a specific port
- Barrier: Messages are used to request notifications for completed operations

Symmetric

Generally there are two types of symmetric messages are allowed in OpenFlow. HELLO messages are exchanged on connection start-up, ECHO_REQUEST and ECHO_REPLY messages are used to determine latency or bandwidth.

2.2.2 Switch

Open vSwitch is designed to perform one main task: in a hypervisor environment, it can be used as a bridge to connect virtual machines with external networks. Normally, this is done by the Linux bridge. However, Open vSwitch provides extended features. Open vSwitch may also be used as a mere virtual switch, e.g. for testing network applications. The software Mininet, for example, provides a local virtualized network on a single PC. Mininet uses Open vSwitch to simulate OpenFlow-enabled switches. Besides virtual switches, ONF members are producing OpenFlow-enabled hardware switches.

2.2.3 Controller

In SDN, the control plane is located in a special network component, called the controller. Since it is purely software-based, available with many development group. In addition to commercial solutions, many open source controllers are available. They differ mostly in the way the northbound API is implemented. While the southbound API is standardized to a great extent, the interfaces on top of the controllers are very different. The commonly used controllers are:

- Ryu: Ryu is a component-based software defined networking framework. Ryu provides software components with well defined API that make it easy for developers to create new network management and control applications
- Floodlight: The Floodlight Open SDN Controller is an enterprise-class, Apache-licensed, Java-based OpenFlow Controller. It is supported by a community of developers including a number of engineers from Big Switch Networks
- POX: Pox as a general SDN controller that supports OpenFlow. It has a high-level SDN API including a queriable topology graph and support for virtualization

Chapter 3

Related Work

Traditionally, anomaly detection was achieved by deploying additional computing devices with IDS capability in the backbone network. SDN infrastructure introduced faster mitigation by updating the anomaly detection techniques. This has increased the research in the area of anomaly detection in SDN infrastructure; to improve the efficiency and try to decrease the false alarm. The initial studies focused on testing the anomaly detection methods which have been successfully deployed in traditional networks and can now be deployed in SDN by using its advantages. Akbar Mehdi et al. [5] have done an extensive survey of the existing methods on anomaly detection on legacy networks and have extended them to SDN infrastructure. In this paper, many anomaly detection algorithms were used to validate that these methods were suitable for low network traffic rates. K. Gliotis et al. [7] proposed the solution by combining the functionality gained by sFlow [8] and Openflow [9] for anomaly detection and mitigation in SDN environments. Zhang Ying [10] formulate the problem in a different way and suggested a flow counting method with dynamic rule replacement algorithms for anomaly detection in SDNs. The work done by Himura et al. is a statistics based anomaly detection algorithm with automatic parameter tuning. Above mentioned researches feature regular network monitoring to detect anomalies based on flow rules.

However, previous research only considered anomaly detection on centralized SDN infrastructure. When a centralized system is further expanded to distributed domain, it gives many benefits such as less overhead, load balancing, scalibility, privacy and flexibility. The work done in DISCO [11], an extensible DIstributed SDN COntrol plane, manages its own network domain and communicates with other controllers to provide end-to-end network services. In another research in ElastiCon [12], an elastic distributed controller architecture contains a pool of controller which is dynamically grown or shrunk according to traffic conditions and the load is dynamically shifted across controllers.

Hence, proposed solution is a first attempt to solve a problem of detecting concurrent distributed anomalies. Since it uses distributed approach in solving these problems, its a novel method in itself.

Chapter 4

Proposed Work

4.1 Concurrent Distributed Anomalies (CDAs)

4.1.1 What is CDAs

CDAs are anomalies which are generated at logically or physically distributed locations at the same time or in a predefined amount of time. They can be short-lived which often occur over a large period of time. CDAs mainly consist of anomalies with ignorable impact which are undetectable. But when caused at multiple locations simultaneously, they could become problematic to the network. As an example, CDAs are activities which can be performed to search for vulnerability across the network. This kind of activity if done at multiple locations simultaneously in very short interval are considered as CDAs. A group of nodes, referred to as botnet, is remotely controlled by a botmaster and can be used for malicious activity. The purpose of this group can be to increase the number of bots in the group by trying to infect more number of users'. These kind of activities are important to detect in order to save a far worse attack in the network. So, CDAs detection helps in early detection of attacks.

4.1.2 Difficulty in Detecting CDAs

Suppose, a particular anomaly is generated at multiple locations in the network at a certain time. Some of these anomalies are generated for a nominal amount of time and might get missed by the detector. On the other hand, some of these anomalies may get detected with a certain delay due to congestion in the network or overloading the controller. The detection of CDAs are difficult by using existing centralized SDN architecture. Various difficulties in detecting CDAs in centralized architecture are:

- Latency and sensitivity in detection leading to delay between the cause and observed behavior in the network. The main reason for latency is increased congestion in control plane due to short interval statistics in addition to normal traffic
- Degree of concurrency of the anomalies is increased beyond some threshold, which might lead to the system failure to perceive the difference between anomalies generated by coordination and other miscellaneous anomalies generated by attacks

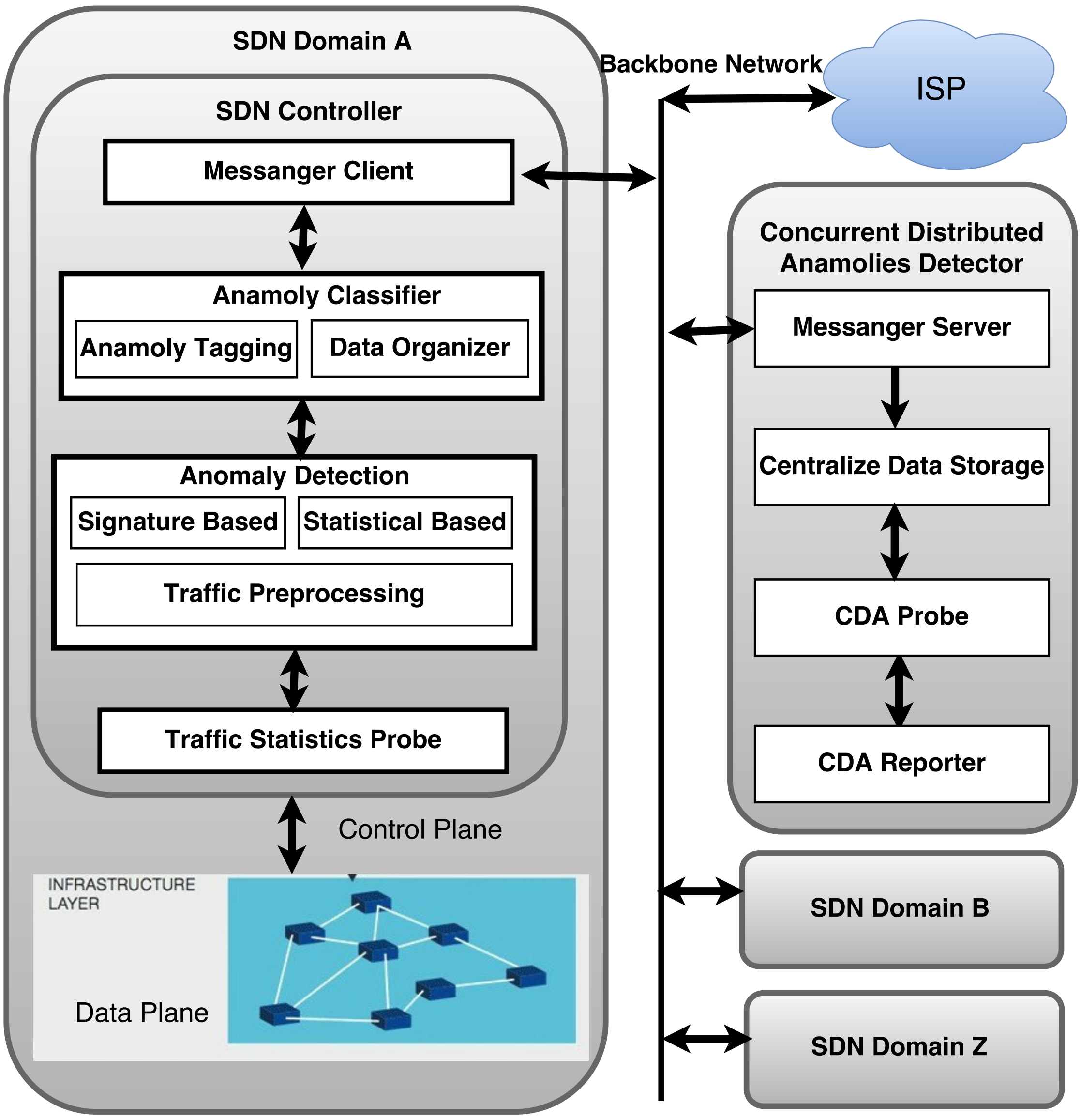

Figure 4.1: Multi Domain SDN Architecture with CDA Detector

- Minimum number of flow rules required to detect CDA. If the amount of flow rules used for monitoring is reduced, some concurrent anomalies could be missed

Technically, it is better if every flow rule is observed, but it is a very costly method. These issues can be handled by proper placement of multiple traffic capturing points in the form of multiple isolated domain-specific SDN controller architecture.

The basic architecture for CDA Detection is shown in Figure 4.1. It comprises of two entities viz., a multi-domain SDN network and a CDA detector. The network is divided into multiple non-overlapping SDN domains and each domain consists of a SDN controller and a part of the network infrastructure.

All these SDN domains report the detected anomalies in its infrastructure to the CDA detector. The CDA detector is a logically centralized system which communicates with all the domains in the network. It collects the information about the detected anomalies from different domains within the network.

4.2 SDN Domain Functionality

The network has multiple domains and each domain has the following stake holders:

4.2.1 Traffic Statistic Probe

The traffic statistic probe is designed on the simple match-and-count rules, installed and adjusted by the controller on the switch. The switches can send traffic counters to the controller periodically. If the statistics are collected in a large period, then the detection algorithms may miss frequent short-lived anomalies. Similarly, if it is done over a short period, it can generate a lot of traffic to the control plane, and overwhelm the controller. Our design provides flexibility to choose probing intervals to collect data and it can operate on flexible short and long period intervals as compared to centralized detection systems that work at only fixed intervals.

Detection Method	**Anomaly Type**	**Traffic Feature Parameter**	**Example of heuristics**
Signature Based	Anomaly Type-1	Packet Count, TCP Header Flag	If the ratio of SYN/ACK flagged packets is more than 20% then it is traffic generated by SYN Flooding
	Anomaly Type-2	Source IP, Destination IP	If the request is an ICMP packet and Destination IP as same as the Broadcast IP then it is traffic generated by Smurf attack
	Anomaly Type-3	Packet Size, Packet Count	If packet size is more than 100bytes and if count ratio more than 20% of total flow then it is traffic generated by Ping-of-Death attack
	Anomaly Type-4	Source IP, Destination IP, Destination Port, Flow Size and Packet Count	If a particular source IP is sending a request for a range of server port address on one host or multiple host then it is traffic generated by Port Scanner
	Anomaly Unknown-1	The Parameter is not classified into any above categories	If the host cannot be classified into any above categories, then the event is classified into Unknown category
Stastical Based	Anomaly Type-5	N number of packet in a window P$_i$ is the probability of each element in the window	$Entropy(H) = -\sum_{i=1} P_i \log P_i$
	Anomaly Type-6	Tranning data like Bandwidth	Classification and Clustering based approach is used to detects sudden changed in the network
	Anomaly Unknown-2	Automatic Parameter Tuning	Miscellaneous Anomalies

Table 4.1: Non-exhaustive list of anomalies used by the system based on sample heuristics

4.2.2 Anomaly Detection

Anomalies in the network are stated as the identification of sudden change in the network. This sudden change has happened because of traffic generated by some attacks. So in this research categorization of attacks has been done on the basis of existing detection techniques: signature based and statistical based on the underlying approach adopted by each technique:

- Signature Based – Signature based anomaly detection methods are designed to detect anomalies generated by known attacks. These attacks are recognized by the feature it affects in the network which are identified as its signatures. Such systems require predefined rules or knowledge of the anomalous traffic. In Table 4.1 the first four anomaly types belong to this category and require parameters like header flag, flow size, packet count, etc. to identify the signature of the malicious traffic whereas the last type is classified as unknown.
- Statistic Based – Network traffic varies with time or space and thus cannot be detected using rule based approach. So learning techniques have been used to detect these kind of anomalies.

The above discussed categories are a part of a non-exhaustive list of anomalies that our system can detect. In the Traffic Preprocessing functionality, we decide the input parameters to the anomaly detection module are given in *Traffic Feature Parameter* of Table 4.1.

4.2.3 Anomaly Classifier

This module first labels the anomalies based on traffic features and then organizes and stores the data in an efficient data structure.

- Anomaly Tagging – The nodes in the network face a plethora of malicious activities. So, there is a need to categorize similar activities into groups in the form of tagged `Anomaly Type` as shown in the Table 4.1.
- Data Organizer – The main function of the data organizer is to store the data in the best possible way in order to get faster search and access time. The detected anomaly is stored in a two-level data structure as shown in Figure 4.2.

  The first level hashes the input where the key is the anomaly type and the value is a pointer to the second level of sorted sets. The sorted set consists of `Anomaly Type` and the `TimeStamp` at which the anomaly occurred. Deploying a hashing mechanism allows for flexibility in adding more number of anomalies without changing the design of the system. Using sorted set helps in retrieving the most recently detected anomaly.

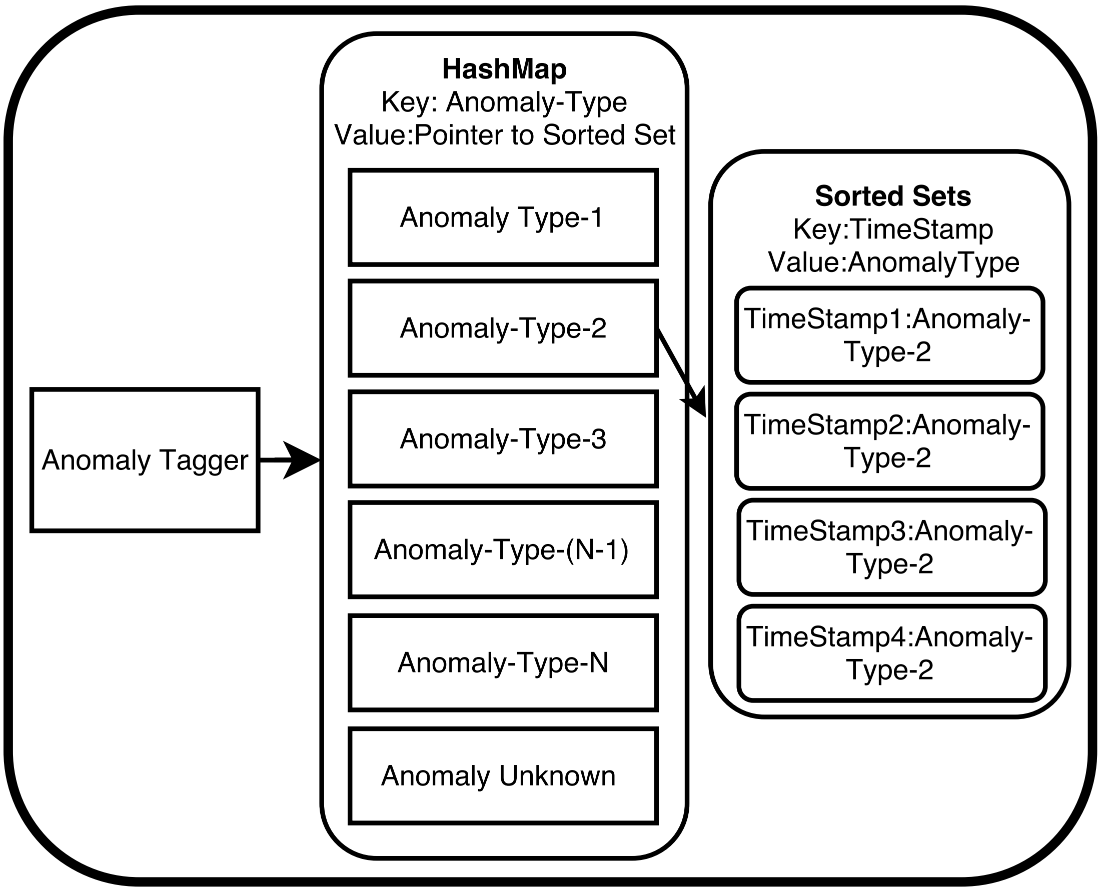

Figure 4.2: Data Structure used in Data Organizer

4.2.4 Messenger Client

The messenger client is used to send the most recently detected anomalies with timestamp to a messenger server running at the CDA detector. Message Passing Protocol with instant messaging functionality can be used for this purpose to obtain faster response time. The sample message type are given below

$< ControllerID = "MACAddress" >$

$< AnomalyType >$Type-1$< /AnomalyType >$

$< TimeStamp >$YYYY:DD:MM:HH:MM:SS$< /TimeStamp >$

$< AnomalyType >$Type-2$< /AnomalyType >$

$< TimeStamp >$YYYY:DD:MM:HH:MM:SS$< /TimeStamp >$

$< /ControllerID >$

4.3 CDA Detector Functionality

Logically, the detector is used to combine the information from multiple domains. The role of this detector is to report concurrent anomalies in the various network domains with the help of the following logical entities.

4.3.1 Messenger Server

Messenger server accepts packets from the messenger client on the SDN domain using an instantaneous message passing protocol and forwards the same to the centralized data storage module.

4.3.2 Centralized Data Storage

The centralized data storage works in a similar fashion to the data organizer in each domain of the network. In this two-level design, the first level hashing maintains the controller record where the key is a unique `controller_ID` and value is a pointer to second level of hashing as shown in Figure 4.3. On the second level, system stores `Anomaly Type` as the key and a pointer to sorted set as the value. The sorted set consists of `Anomaly Type` and the `TimeStamp` at which the anomaly occurred. Deploying a hashing mechanism allows for flexibility in adding any number of controllers, viz., domains to the network without changing the design of the system and the network.

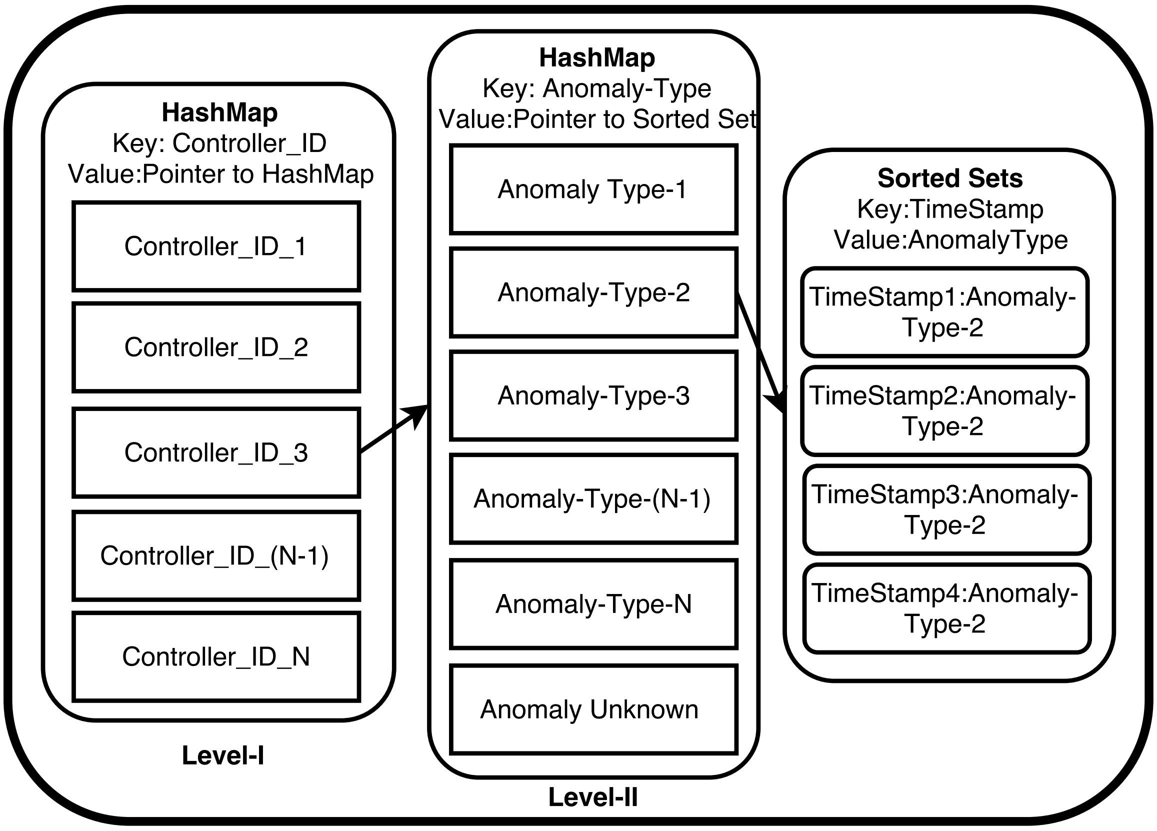

Figure 4.3: Centralized Data Storage in CDA Detector

4.3.3 CDA Probe

This module compares the timestamp of detected anomalies against the current timestamp of the system with a permissible difference between the two timestamps, while accounting for system and network overheads. The timestamp of the detected anomalies can be retrieved by visiting every sorted set from the last level of hashing present in the Centralized Data Storage as given in Algorithm 1.

4.3.4 CDA Reporter

The reporter informs the network administrator of the detected CDAs after getting the information from the CDA probe which is done by calling CDA Probe with given *Controller_ID* and polling time P_t. Concurrent distributed anomalies are detected by taking the intersection of the output of the CDA probe with the *Controller_ID* of each domain.

4.4 Implementation

Our implemented prototype design comprises of two main components: SDN Domain and CDA Detector.

SDN Domain topologies have been built by using Mininet [13] to emulate Open VSwitch [14] with virtual hosts and on the top we have written applications using python based Ryu [15], an open source OpenFlow controller. The modules involved in these domains are: 1) Traffic Statistic Probe, 2) Traffic Preprocessing, 3) Anomaly Detection, 4) Anomaly Classifier and 5) Messenger Client as shown in Figure 4.1. The Messenger has two instances running on client side and server side respectively. This messenger uses XMPP to ensure instant communication. The client side instance which is running on all SDN domains network is encapsulating the data (anomalies with its type and timestamp at which it occurs) and at server side it decapsulates the data. Messenger is fast as required since it uses XML format while encapsulation. E.g.: suppose a specific SDN network found multiple anomalies with respect to time so the same information is encapsulated as

$< ControllerID = "MACAddress - 1" >$

$< AnomalyType >$Type-1$< /AnomalyType >$

$< TimeStamp >$YYYY:DD:MM:HH:MM:SS$< /TimeStamp >$

$< AnomalyType >$Type-2$< /AnomalyType >$

$< TimeStamp >$YYYY:DD:MM:HH:MM:SS$< /TimeStamp >$

$< /ControllerID >$

$< ControllerID = "MACAddress - 2" >$

$< AnomalyType >$Type-1$< /AnomalyType >$

$< TimeStamp >$YYYY:DD:MM:HH:MM:SS$< /TimeStamp >$

$< AnomalyType >$Type-2$< /AnomalyType >$

$< TimeStamp >$YYYY:DD:MM:HH:MM:SS$< /TimeStamp >$

$< /ControllerID >$

in an XML tagged packet. Now if the first tag reaches the messenger server it will not have to wait for the entire packet to be received and processed. Instead it will start processing the packet with the help of tags. In this way, having an XML based packet is fast and instant. Data organizer and centralized data storage have been built using Redis server [16].

Algorithm 1: Concurrent Distributed Anomaly Probe

```
Input: Polling Time $P_t$, Controller_ID Id
Output: List of Distributed Anomalies L
  procedure CDA_PROBE ()
      Address to HashTable-II $A_1$ = HashTable-I(Id)
                                                    ▷ HashTable-I Controller_ID
      Address to Sset $S_1$ = HashTable-II($A_1$)
                                                    ▷ HashTable-II Anomaly Type
      for each entry in S ϵ Sset($S_1$) do:

      if $P_t$ < CurrentTime-S.TimeStamp then
          L_Add(S.Anomaly_Type)
      else
          break
  end procedure
```

Chapter 5

Experiments and Results

In this section, we evaluate the performance of the system in two steps. First, checking the correctness of the system prototype using an emulated SDN based multi-domain testbed i.e., system validation. Second, reporting the performance of the system with the help of various metrics like sensitivity, response time, efficiency, scalability, etc.

5.1 System Validation

Our experimental testbed is built on top of Mininet, which emulates a network of Open VSwitches and it consists of two SDN domain specific controllers that are connected to system integrator. For validating the correctness of the system we require some malicious activities running on the system. There are existing open source tools available which generate anomalous traffic. But here we describe specific SDN based attacks on control plane by running malicious applications on the controller. These applications control the way in which packets would be forwarded to the hosts in the network. In normal function, a switch sends a PACKET_IN message to the controller because no flow rule matches for the destination host. But the attacker modifies the flow rules of user's trying to use the FTP service by monitoring the request and response of PACKET_IN and PACKET_OUT messages generated by the switches. The attack is carried out by ordering the switch to forward the packet to the controller for every packet with destination port number 20 and not installing the flow rules on the switch for the same.

The traffic generated by this attack can be detected as *Anomaly_A* by the domain-specific controller. If the same anomaly happens simultaneously on many domains in the network, the system integrator detects it as planned anomalies thereby validating the system. Since the attacker aims to congest the network with dubious flow rules, the anomaly detection algorithm identifies the amount of traffic generated by the controller to be different from the normal behavior of the controller. This is detected as anomalous behavior of the system thereby validating it.

5.2 Performance Evaluation

We consider real traffic data taken from the MAWI traffic repository [17] for evaluation purposes and a detection algorithm which is available using Weka [18], a tool with a collection of supervised and unsupervised machine learning algorithms for data mining tasks. We use NetMate for the flow statistics for network traffic which help in calculating the sensitivity of the data. In some cases random anomalies have been generated on the prototype architecture which serves as a proof of concept. Performance of the proposed system has been evaluated based on the following parameters

5.2.1 System Sensitivity

The sensitivity of the system can be defined as the ratio of True Positive (TP) and sum of TP and False Negative (FN) where TP is defined as the number of planned anomalies detected correctly within polling time and FN is defined as the number of planned anomalies that went undetected within polling time. So, on average the sensitivity rate of our system is given as

$$Sensitivity = \frac{TP}{TP + FN}$$

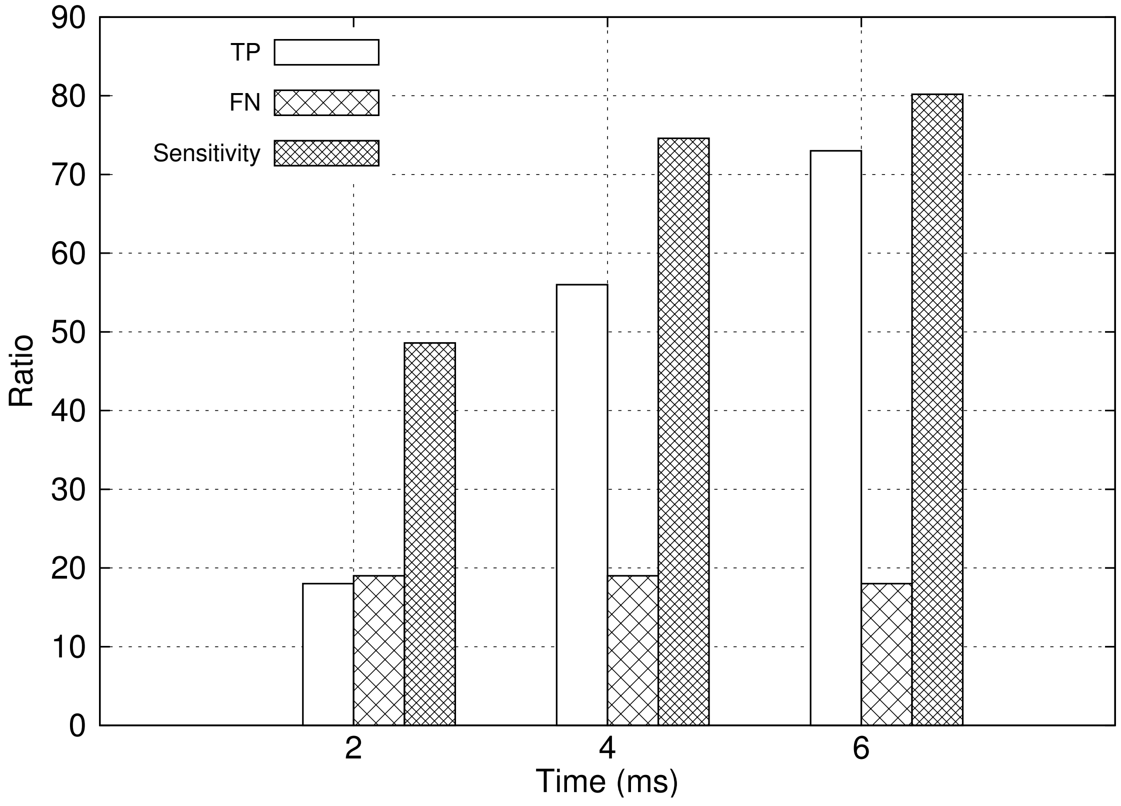

Figure 5.1: Sensitivity of CDA Detector

For example, Domain A, Domain B and Domain C detected Anomaly Type-T1 at time t and reported the same to the system integrator. Now polling has been done on integrator and found Anomaly Type-T1 at Domains A and B but not on Domain C because of polling time being less than desired. This leads to system integrator not being able to detect anomalies on all three domains. This case can be dubbed as false negative case. Now if we increase the polling time, we are able to detect it at all three domains leading to the true positive case.

As shown in the Figure 5.1, we test our system on random generated data with five anomalies at three different polling times by keeping 20 controllers. With increasing polling time, the TP and the sensitivity of the system increase and FN decreases. Finally we achieved almost 80% sensitivity by keeping a moderate polling time.

5.2.2 Scalability

Scalability in distributed systems is an important point for evaluating the system. Thus, we evaluate this measure by expanding the number of domains in our system. In our experiment, we have two variables viz., anomaly types and number of domains.

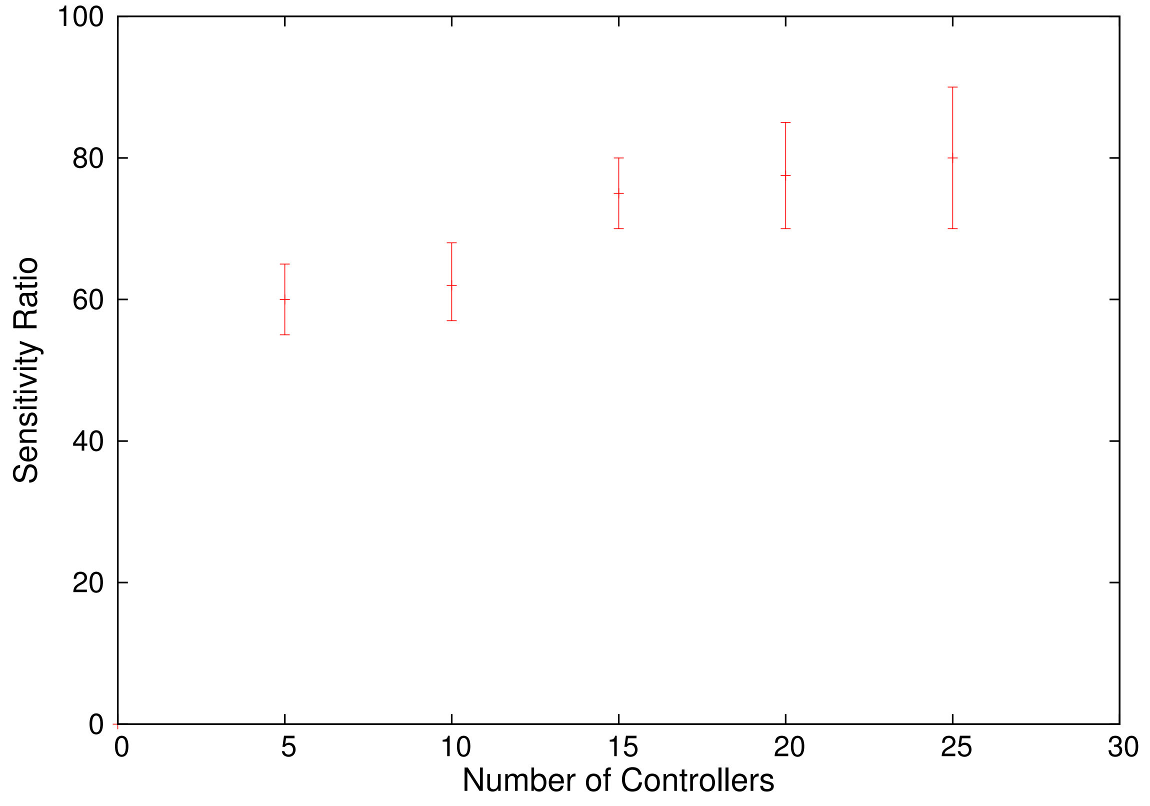

Figure 5.2: Scalability with constant anomalies

We perform trials to test the scalability by varying the number of participating domains. The number of domains range from 5 to 20 and the number of anomalies are kept constant at 5. These experiments are performed by generating all the five types of anomalies randomly with equal probability in each domain. Using this technique we are ensuring that the same type of anomalies are generated at multiple domains simultaneously. As shown in the Figure 5.2, the scalability varies between 60-80% by increasing the number of controllers. The reason behind this is that with less number of domains, the possibility of generating same type of anomalies is low. So we get limited scalability. On the other hand, we achieve more accuracy by deploying more number of controllers, with the amount of anomalies as compared to the number of controllers.

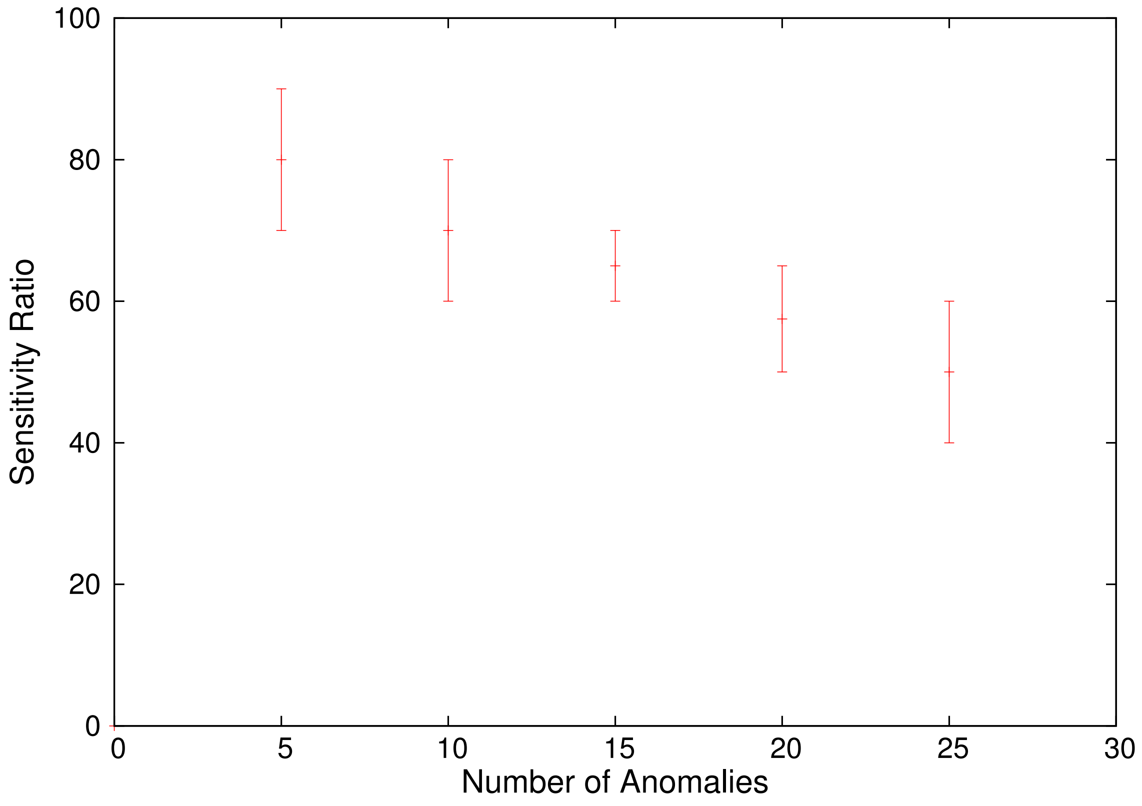

Figure 5.3: Scalability with constant controller

In another scenario we performed an experiment by keeping the participating domains constant at 20 and varying the type of anomalies from 5 to 20. We observed that the efficiency of the system reduces with increasing number of anomalies with respect to a constant number of controllers as shown in in Figure 5.3. Based on the above discussion, we provide a bound on our system scalability by considering the real-time behavior of the network.

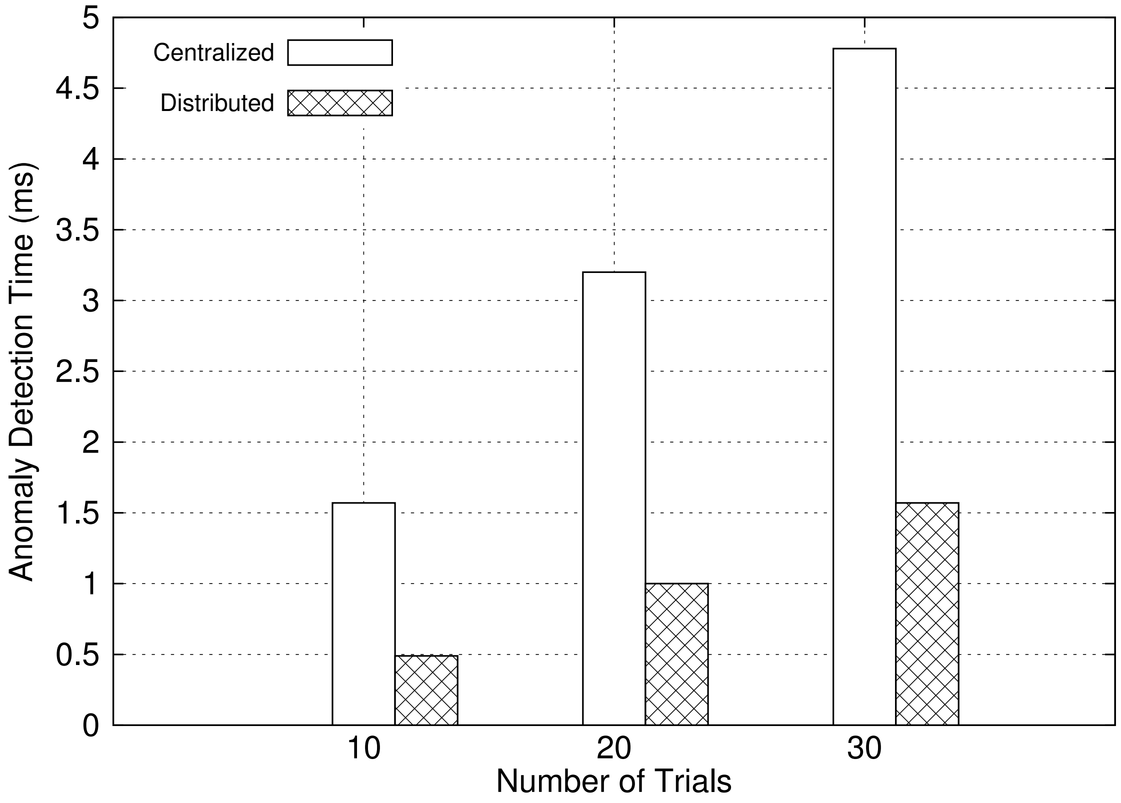

Figure 5.4: Response Time of anomaly detection in Centralized Architecture and Distributed Architecture.

5.2.3 Response Time

The response time of the system is measured by detecting concurrent distributed anomalies on centralized architecture and distributed architecture. The graph in Fig. 5.4 and 5.5 shows the response time of a centralized system with respect to a distributed system for anomaly detection in domain and CDA detection respectively. The result have shown designed architecture detect CDAs in lesser time.

5.2.4 Performance-Gain

The performance of the distributed system can be defined in terms of gain in performance while extending from centralized to distributed design. The time elapsed between the beginning and end of execution on a centralized system is given as T$_c$. In the same way T$_d$ is defined for distributed system. So, the ratio of time taken to solve a problem on a centralized system to the time required to solve the same problem on a distributed system is given by

$$Performance - Gain = \frac{T_c}{T_d}$$

The bar chart shown in Figure 5.4 and 5.5 points out the experimental scenario where five anomalies are simultaneously generated on both the architectures. The gain in performance of the architecture is found by using summation of response times of centralized system and distributed

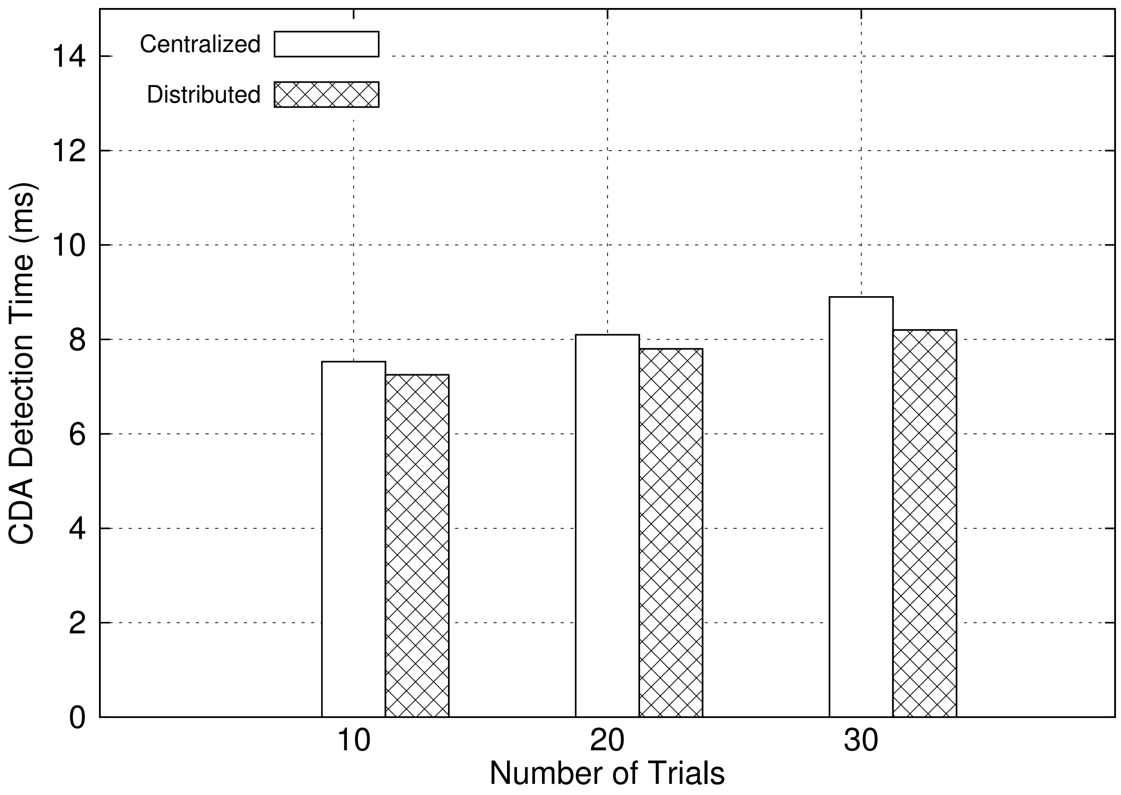

Figure 5.5: Response Time of CDA Detection Algorithm in Centralized Architecture and Distributed Architecture.

system calculated using the above formula. It was found to be 1.16, 1.26 and 1.37 respectively for given number of trials.

5.2.5 Complexity

The time complexity of CDA detector is $\mathcal{O}(logn)$ as it uses the two stage hashing data structure followed by a sorted set in which first level hashing provides variability to the number of domains. In the same way, the second level hashing provides change in the type of anomalies with respect to a particular domain.

Chapter 6

Discussion

6.1 System Improvement Heuristics

6.1.1 Interpret Planned Anomalies

In our scenario we defined planned anomalies as those that occur at the same time across all the domains. We can also define planned anomalies in a different way like each domain may be generating different anomalies at the same time as a result of different attacks.

6.1.2 SDN based network traffic knowledge

We would be focusing on improving the system by gaining the domain knowledge of various attacks that are specifically carried out in the SDN network. This knowledge helps us to understand the behavior of anomalous traffic and makes it easier to select the features for the prediction algorithms. The goal of feature selection is to filter the dataset to retain relevant attributes.

Sometimes, network faced unexpected behavior. For example, suppose in an NFV environment some service suddenly goes down. As a result NFV launches a virtual service on a different location in the network. This leads to a deviation from normal traffic pattern in the network and it is classified as anomalous traffic by the existing detector. Thus, having detailed knowledge of SDN network increases the efficiency and improves the system.

6.1.3 System Usability

This system can be used in detecting concurrent malicious activities generated by the software simultaneously. Others problem such as blocking malicious hosts, saving bandwidth at core and concurrent anomalies.

6.2 Misc Topics in Security of SDN

6.2.1 Bootstrapping Flow Aggregation for Anomaly Detection in SDN

Detecting non-application layer anomalies involves counting flow statistics by intermediate devices. In large networks this may overload the centralized management system. Thus, most common approaches use sampling and aggregation of flow rules to reduce the amount of load. But the extent to which flows must be sampled is a crucial point. Poor aggregation methods are very likely to miss important anomalies and a large sampling frequency may overload the system. Thus, a fine balance is required between the overhead and the accuracy of the system so as to minimize aggregation/sampling footprint. Approaches have been suggested that use Software Defined Networking (SDN) and take advantage of its flexibility to dynamically update the rules of aggregating flows to reduce the loss of missing anomalies. But most of the existing methods perform aggregation on a single parameter and thus are biased to high false positive results that impose extra load to counteract them or possibly higher false negative results that miss out important anomalies in the system.

Our key motive is to validate a model that improves the accuracy of an anomaly detection system. Towards this, we bootstrap the flow aggregates from the network based on multiple parameters. For each of the parameters a linear prediction is applied to calculate the set of aggregates for anomaly detection. Based on these predictions, an appropriate rule replacement method is used to get the final set of flow aggregates for each parameter. We apply a greedy approach to predict anomalies by choosing the parameter that replaces highest number of rules that detect anomalies. This stacking of dynamic rule replacements with the bootstrapped multi-parameter aggregates help in improving the accuracy of a real-time anomaly detection system dependent on the attributes of the network.

Aforementioned task can be done easily on Software-Defined Networking (SDNs) based on OpenFlow (OF) protocol exports control programmability of switched substrates. As a result, rich functionality in traffic management, load balancing, routing, firewall configuration etc., that may pertain to specific flows they control can be easily developed. Implementing this method on current legacy networks is not possible since changing configurations for updating rules requires manual configuration of routers. Hence, the SDN paradigm is used so that its programmability inside controller makes it possible to dynamically update rules. It allows us to provide specification for multi-parameter bootstrapping on network flow measurements.
Network anomaly detection is of utmost importance especially when it concerns critical high security system architectures which requires constant monitoring. To have a accurate tool to measure anomaly is hence very important. Hence new approaches and efforts are made to find more accurate results for anomaly detection. Our approach considers multiple parameters for detecting anomaly so that accuracy will increase significantly while introducing a very slight extra load on system.

6.2.2 Addressing problem of collaboration of hacked switches in SDN

The Software-defined Network (SDN) architecture relies on the assumption that all the SDN switches in the network obey the commands of the controller. However, if someone compromises an SDN switch using physical access or harmful patches, it can cause malicious activities in the network and can even bring the entire network down. Therefore, detection of compromised SDN switches is a major concern. In this paper, we propose a solution for detecting a compromised switch with the collaboration of other trusted switches.
This enables a network manager to discover the compromised switch present between the trusted switches and find a possible solution for mitigation. Our implemented prototype validation system works on the network simulation using only the required features of the OpenFlow 1.0 specification enabled on Mininet and Ryu controller. Its effectiveness can be verified by varying the number of switches in the network.

A traditional computer network is a hierarchical one which is suited to client-server computing. But recent advancements in computing such as virtualization and cloud computing need a robust network which can sustain dynamically changing traffic patterns. This gave to a new network architecture called Software Defined Networking or SDN. The idea here is to make the network programmable. As already pointed out, the data plane and control plane are separated and the control plane is centralized. Which implies that the network admin has full control over the aspects of the network. So, any new functionality can be added over existing network easily. This can increase the robustness of system considered as a whole in scenarios like load distribution on various servers. Thus, SDN takes a major leap in network architecture design.

Despite being a promising architecture for future, there are various concerns about security in SDN. Some of these either did not exist before or were hard to exploit in traditional architecture. Consider the Denial of Service attack. Distribution of logic gave an inbuilt defense mechanism to network against Dos attacks. An individual server on a network could still be brought down but whole network got prevented from being affected . However, centralization of control has turned the tables. A powerful DoS attack on SDN controller would handicap the network [**?**]. Even if the traffic is not directed on the controller, unnecessary traffic could slow down the functioning of the network.

Another major concern is that the switches in SDN are dumb entities which in case of any unknown packet take information from the controller about the action to be taken, yet they are important enough that someone who has access to it acquires the ability to break havoc on the entire network. Not much work has been done in the area of security in SDN. Researchers have talked about authentication and secure channel but the assumption of one master controller and slave switches bowing to it has always been relied upon. Owing to which the security in SDN can get compromised in many situations.

Suppose one has physical access to the switch. In this case, a backdoor can be implanted to the network using this switch which will lead to undesirable consequences. The cracker can remotely control the entire activity of the switch which was ideally supposed to be done according to the rules set out by network admin using the controller. Switch can be accessed in a different way also. Consider the case of virtualised networks. Here, switches run over actual end hosts. These hosts can belong to a traditional IP network. If these systems are targeted by the bad guys, then the SDN switch which is just an application running on it is no longer safe. These may look like traditional problems but corresponding traditional methods which have been extensively used in past to deal with them cannot be applied directly to SDN because they assume switches to be intelligent which is not true in case of SDN.

After getting the control over the switch, though the hacker virtually control of every aspect of network related to that switch. The unwanted activities can take certain forms[**?**]. It might be forwarding the packet to the incorrect port. Say, the controller wanted the packet with a certain match to be forwarded to port 1 but rather than it, the hacker might drive the packet toward port 2. Or perhaps even worse case might be to duplicate the packet on all of the ports bringing in unwanted network traffic. Also, the content of packet can be changed.

Similar attack models were considered in previous works on this area but they have not been much effective in handling the cases where multiple compromised switches are there. The situation gets worse where these cracked switches are in collaboration with each other. The problem in this is unreliability on the results obtained by the test. So, the initial solutions to compromised switch detection consider a special scenario by not considering all the cases and thus making a questionable assumption. In this paper, we consider this exact problem of detection of compromised switches in case there exists a collaboration among them. We offer a tactful method in which rather than adding all of the previously unaccounted cases at once, which might turn the problem into an unsolvable one, we consider a bigger subset of cases at once and thus relatively generalizing the scenario. The case which would not be taken here is not left ignored forever and a probabilistic mechanism has been described to account for the same.

We use OpenFlow based SDN. OpenFlow is a standard which allows experimental protocols to be run on a network. It is the first standard communication interface designed between control and forwarding layers of an SDN architecture. It allows direct access to and manipulation of the forwarding plane of network devices. Ryu controller has been used. Ryu is a component based software based software defined networking framework. It provides well defined API for developers to build network management and control applications. It supports many communication protocols including OpenFlow. The effectiveness and efficiency of the approach was tested for different network graphs. The simulation was carried out on Mininet. We use the word good switch or a healthy switch for a normal functioning switch and a bad switch for a compromised one. Also, we assume that bad switches can have some intelligence which they owe to malware or any other way which helped crack that switch. Using this intelligence, they can carry out intelligent bad activities such as making their collaboration.

6.2.3 Detection of Cache Pollution in ICN using SDN

Information Centric Networking (ICN) is categorized as a content-driven architecture. In this paradigm, each node has a unique DNS style naming convention and hosts contents rather than having a dedicated server. ICN differs from host-centric architecture in terms of data independence from location, storage, application and transportation thereby enabling in-network caching and replication. This brings benefits like better scalability with respect to information demand and improved efficiency. Some of the widely used types of ICN architectures are Named Data Networking (NDN) [19], Content Centric Networking (CCN) [20], Data Oriented Network Architecture (DONA) [21], Network of Information (NetInf) [22] among the few.

ICN achieves universal caching with the help of following motives: uniform, i.e., applied to all content delivered by any protocol; democratic, i.e., published by any content providers; and pervasive, i.e., available to all network nodes [23]. Normally, the new cache decision policies store the most recent requested content in their storage. Sometimes this data might be an unpopular content which can degrade its performance if held for a long period of time. This can lead to compromise in the performance of ICN.

Security is an inbuilt characteristic of ICN architecture. Still they are prone to attacks. Attacks belong to various categories. Some of these include but are not restricted to packet sniffing, cache pollution, breaching privacy by breaking keys and routing attacks. Classifying the attacks is necessary because every attack requires a separate detection and mitigation mechanism and categorizing the attacks provides with a way to assess the impact of the same.

Caching attack in ICN can be of the following types: (1) Bogus announcements (2) Random or unavailable content requests and (3) Cache pollution. In (1), the attacker sends false updates regarding the content on the network which leads to the router being overwhelmed and not able to send the latest updated content to the requesting node. With the attack (2), attacker sends request for unavailable content at a rate greater than the converging rate of the router leading to denial of service. Finally, attack (3) can be described as polluting the cache with unpopular content. An unpopular content refers to a content that is not frequently requested. This attack may require prior knowledge of popularity of the content in the network.

Since most of the existing solutions of caching related attacks are designed for a dedicated cache server and may not work in ICN because all nodes can enable caching we need a dedicated solution for caching related attacks. So in this paper we are discussing caching attacks in ICN in which the attacker replaces the content in the cache server with unpopular content. ICN with SDN brings the added advantage of SDN as it helps in controlling the network characteristics with the help of programs.

The motivation of this research is to use SDN control plane to obtain a deciding parameter to distinguish between popular and unpopular content in ICN. This prevents the problem of cache poisoning in ICN by using advantage of SDN. Obtaining these parameters in ICN was a challenging task, thus, by using SDN these metrics can be calculated in an efficient manner. This research can be contribution by providing following functionalities:

- To provide a new solution for cache decision policy of replacing popular content with random content and
- To solve the problem of cache poisoning in ICN over SDN.

Chapter 7

Conclusion and Future Work

We propose a Concurrent Distributed Anomaly Detection mechanism. It relies on a CDA detector which comprises of multi-domain isolated SDN networks. This detector system gathers up the logical data i.e., without any user traffic flow, for detecting anomalies from isolated domain controllers of SDN. We demonstrated how it can efficiently detect the anomalies in isolated environments and later instantly detects concurrent anomalies in distributed environments. In future we planned to focus on designing of algorithms with more complex concurrent distributed anomalies definition.

References

[1] K. Leung and C. Leckie. Unsupervised Anomaly Detection in Network Intrusion Detection Using Clusters. In Proceedings of the Twenty-eighth Australasian Conference on Computer Science - Volume 38, ACSC '05. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 2005 333–342.

[2] T. Lappas and K. Pelechrinis. Data mining techniques for (network) intrusion detection systems. *Department of Computer Science and Engineering UC Riverside, Riverside CA* 92521.

[3] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based Anomaly Detection: A New Approach for Detecting Network Intrusions. In Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS '02. ACM, New York, NY, USA, 2002 265–274.

[4] D. Kreutz, F. M. V. Ramos, P. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-Defined Networking: A Comprehensive Survey. *CoRR* abs/1406.0440.

[5] S. A. Mehdi, J. Khalid, and S. A. Khayam. Revisiting traffic anomaly detection using software defined networking. In Recent Advances in Intrusion Detection. Springer, 2011 161–180.

[6] A. Tootoonchian and Y. Ganjali. HyperFlow: A distributed control plane for OpenFlow. In Proceedings of the 2010 internet network management conference on Research on enterprise networking. 2010 3–3.

[7] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris. Combining OpenFlow and sFlow for an Effective and Scalable Anomaly Detection and Mitigation Mechanism on SDN Environments. *Comput. Netw.* 62, (2014) 122–136.

[8] P. Phaal, S. Panchen, and N. McKee. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks .

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38, (2008) 69–74.

[10] Y. Zhang. An Adaptive Flow Counting Method for Anomaly Detection in SDN. In Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, CoNEXT '13. ACM, New York, NY, USA, 2013 25–30.

[11] K. Phemius, M. Bouet, and J. Leguay. Disco: Distributed multi-domain sdn controllers. In Network Operations and Management Symposium (NOMS), 2014 IEEE. IEEE, 2014 1–4.

[12] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella. Towards an elastic distributed SDN controller. *ACM SIGCOMM Computer Communication Review* 43, (2013) 7–12.

[13] B. Lantz, B. Heller, and N. McKeown. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Hotnets-IX. ACM, New York, NY, USA, 2010 19:1–19:6.

[14] O. vSwitch. Production Quality, Multilayer Open Virtual Switch. `http://openvswitch.org/` 2016.

[15] Ryu. SDN Framework. `http://www.osrg.ryu.com/` 2016.

[16] R. Labs. Redis. `http://redis.io/` 2015.

[17] K. Cho, K. Mitsuya, and A. Kato. TRAFFIC DATA REPOSITORY AT THEWIDEPROJECT .

[18] M. L. G. at the University of Waikato. WEKA. `http://www.cs.waikato.ac.nz/ml/weka/` 2016.

[19] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang. Named Data Networking. *SIGCOMM Comput. Commun. Rev.* 44, (2014) 66–73.

[20] V. Jacobson, M. Mosko, D. Smetters, and J. Garcia-Luna-Aceves. Content-centric networking. *Whitepaper, Palo Alto Research Center* 2–4.

[21] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. In ACM SIGCOMM Computer Communication Review, volume 37. ACM, 2007 181–192.

[22] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl. Network of Information (NetInf)–An information-centric networking architecture. *Computer Communications* 36, (2013) 721–735.

[23] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox. Information-centric Networking: Seeing the Forest for the Trees. In Proceedings of the 10th ACM Workshop on Hot Topics in Networks, HotNets-X. ACM, New York, NY, USA, 2011 1:1–1:6.