B.Sc. in Computer Science and Engineering Thesis

# A Reliable System To Detect Security Attacks In A Scalable SDN Architecture.


## Authors

Md. Sultanul Islam Ovi
160041036

Nafil Mahmud
160041031

Abida Taskin Oishee
160041035

## Supervisor

Dr. Muhammad Mahbub Alam
Professor

**Department of Computer Science and Engineering(CSE)**
**Islamic University of Technology(IUT)**
**Organization of the Islamic Cooperation (OIC)**

Gazipur, Bangladesh


March, 2021

# CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis, titled, **"A Reliable System To Detect Security Attacks In A Scalable SDN Architecture."**, is the outcome of the investigation and research carried out by under the supervision of Prof. Dr. Muhammad Mahbub Alam, Department of Computer Science and Engineering (CSE), Islamic University of Technology (IUT), Dhaka, Bangladesh.

It is also declared that neither of this thesis nor any part of this thesis has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

*Authors:*

---

Md. Sultanul Islam Ovi
160041036

---

Nafil Mahmud
160041031

---

Abida Taskin Oishee
160041035

# CERTIFICATION

This thesis titled, **"A Reliable System To Detect Security Attacks In A Scalable SDN Architecture."**, submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree B.Sc. in Computer Science and Engineering in March, 2021.

**Group Members:**

Md. Sultanul Islam Ovi

Nafil Mahmud

Abida Taskin Oishee

**Supervisor:**

---

Dr. Muhammad Mahbub Alam
Professor
Department of Computer Science and Engineering(CSE)
Islamic University of Technology(IUT)

# ACKNOWLEDGEMENT

At the outset, we express utmost gratitude to Almighty Allah for His blessings which allowed us to shape this research into reality and give it form.

We are very grateful to our supervisor Prof. Dr. Muhammad Mahbub Alam, Department of Computer Science and Engineering, Islamic University of Technology (IUT), for his supervision, knowledge and support, which has been invaluable for us.

Finally, we seize this opportunity to express our profound gratitude to our beloved parents for their love and continuous support both spiritually and mentally.

Gazipur,
Bangladesh
March, 2021

Md. Sultanul Islam Ovi

Nafil Mahmud

Abida Taskin Oishee

# Contents

# List of Figures

# ABSTRACT

Software-Defined Network (SDN) is a promising solution of network virtualization. But it is vulnerable to attacks by corrupted switches as existing detection mechanisms do not work in this environment. A corrupted switch can also compromise the SDN controller. In this paper, we propose a detection mechanism that can detect both compromised SDN switches and controllers. Our main idea is to cluster to the frequently used devices and then collect statistics of those switches to create the expected and actual path for a packet. Thus, we can identify specific compromised switches and also specify the attacks. The detection mechanism is not dependent on the controller performance as we collect statistics of the switches in real-time and in a periodic manner.

# Chapter 1

# Introduction

The primary innovation of SDN is the decoupling of the control and forwarding planes and the centralization of the network control function. It allows for network optimization, as well as flexible and scalable system management and creativity.

Nevertheless, this new architecture also brings potential security risks. SDN is based on an important assumption: all network devices will follow the network manager's commands. That is, all network switches should process packets according to programmed rules. When an SDN switch is exploited by an intruder, however, the presumption falls apart. Corrupted network entities may be used to steal sensitive data, conduct vulnerability exploits on other users, or maybe even bring the entire network down [1] [2].

To avoid these attacks, it is necessary to develop a compromised SDN switch detection mechanism to find out black sheep from all SDN switches [3] [4] [5]. Our research is based on identifying real-time security threats on network topology and data plane forwarding that exist within SDNs.

## 1.1 Software Defined Networking (SDN) Paradigm

SDN is the virtualization of a network by splitting the network's control layer from the data plane, which handles traffic. In a network infrastructure, there is a sophisticated device (a single or a group of controllers) that oversees all network traffic and a collection of dumb routers and switches that only transmit messages. The main goal of SDN is to be open and programmable. Network virtualization has many benefits, including the ability to dynamically spin up and down networks, fine-tune them for unique application use cases, and install security policies on each network.

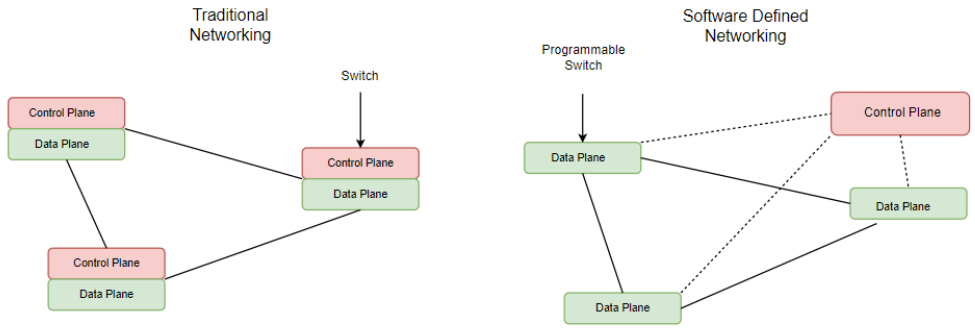


Figure 1.1: SDN Paradigm

SDN has become one of the most common methods for deploying applications in organizations. It's a design that decreases running costs and reduces the time it takes to make improvements or add new services. To improve security and application performance, SDN allows the network to connect directly to applications through APIs. SDN provides a versatile and adaptable network architecture that can adjust in response to evolving business requirements. This is achieved through the standardization and abstraction of network functions in a software-defined network (SDN). Through separating the system and control forwarding features, SDN allows network management to become customizable and the fundamental structure to be abstracted for applications and network operations. Through extrapolating network parameters, including control plane functionality, and deploying them on an SDN controller running SDN applications, IT teams can gain centralized control. Network teams may use the controller and application to interact with physical or virtual network elements using the OpenFlow networking protocol. Teams may use an Application Program Interface (API) to make improvements to the network that govern multiple devices without learning multiple proprietary commands or syntax from vendors. These networks are configurable and can be quickly updated depending on current market needs because they are centrally controlled and optimised using free software. Today's network environments are much more complex, manageable, cost-effective, and adaptable thanks to software-defined networking (SDN).

The three layers in an SDN are:

(i) Application layer

(ii) Control layer

(iii) Infrastructure layer

The Infrastructure Layer defines the real-world tools that the virtual SDN interacts with. The Application Layer is a representation of the network that has been developed. The Control Layer is in charge of managing the mapping between the actual and produced networks.

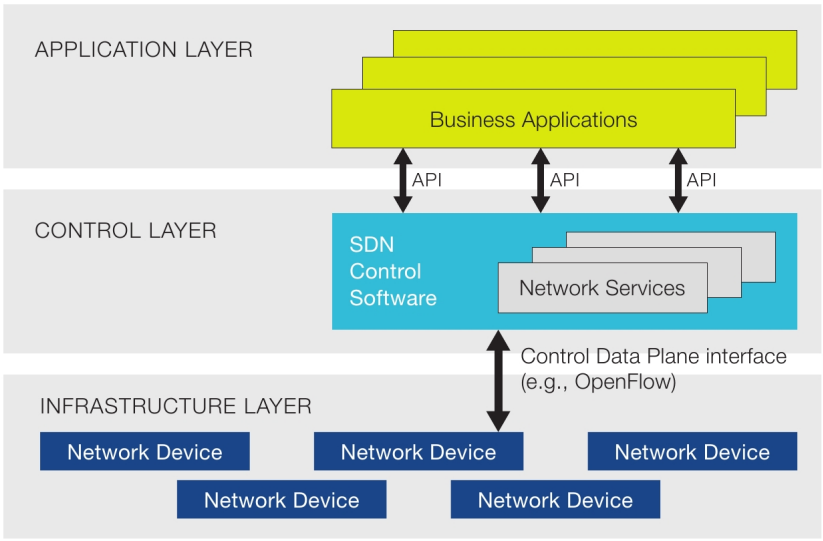


Figure 1.2: SDN Architecture

## 1.2 Main Reasons for SDN Vulnerability

Networks are vulnerable to attacks by compromised devices [6] [7]. In SDN, the networks are more vulnerable than traditional ones since, in SDN, it uses programmable software switches. [8]

(i) The first is that current solutions are incompatible with securing SDN. In reality, conventional network security mechanisms can no longer function as a result of the removal of intelligence from forwarding devices.

(ii) The second aspect is the unproven and full dependence on switches by the control plane. An SDN controller's view of the network is focused on PACKET-IN messages, which are not safely authenticated or checked. A Denial of Service (DoS) attack can be executed using the same flaw. [8]

(iii) It's more difficult to protect configurable soft-switches like Open vSwitches than it is to secure hardware alternatives, which are harder for an intruder to physically manipulate. [8]

(iv) The fourth point is that the SDN protection framework is a moving target of continuously evolving principles and guidelines. [8]

(v) Fifth, Performance benefits such as smaller latency reply to network events and improved specification optimization for cryptography, MAC training, and codec control message (CCM) interactions have prompted plans to give the SDN data plane more capacity [8]. This makes the network more vulnerable to conventional attacks and broadens the types of attacks a malicious system might conduct against it.

In SDN, all the control functionalities from the switches are taken away. So they have become vulnerable to different kinds of attacks [9] [10] [11] [12]. And given time a compromised switch can also compromise the controller.

## 1.3 Types of Attacks

When an OpenFlow switch is compromised, the attacker may launch two kinds of attacks, passive attacks, and active attacks. [8] It is hard to detect a passive attack since the compromised switch acts as a normal switch except that the attacker can silently get all information of the switch. As for active attacks, the attacker will command the compromised switch to do abnormal actions instead of actions programmed by the controller [13]. In this case, though the OpenFlow controller can use an *ofp-flow-stats-request* to get all flow rules from the compromised switch, the switch can easily respond to the correct answer to the controller without following these rules. So *ofp-flow-stats-request* cannot be used as a detection approach. Here, we focus on active attacks. An intruder may use a corrupted forwarding device to perform the following malicious actions:

(i) **Packet Misrouting:** When a packet is deviated from its actual destination and does not arrive at its final destination, this happens. This may be used to initiate a network availability attack or as part of a more complex threat.

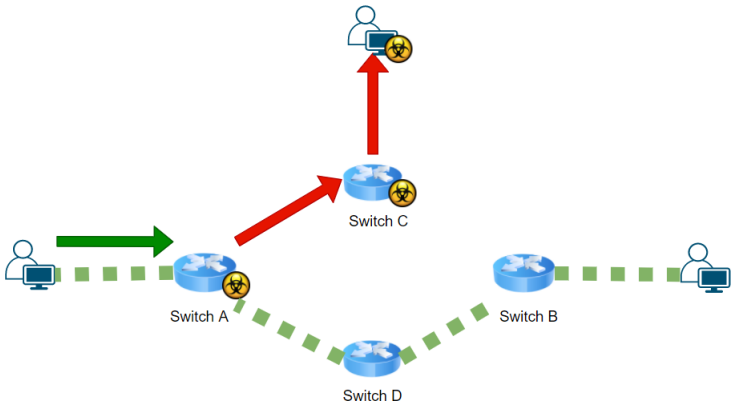


Figure 1.3: Packet Misrouting

(ii) **Packet Replay:** When a switch sends a copy of a packet to a third destination in addition to the actual destination, this happens. An intruder can use a switch that replays packets to carry out surveillance and authentication attacks.

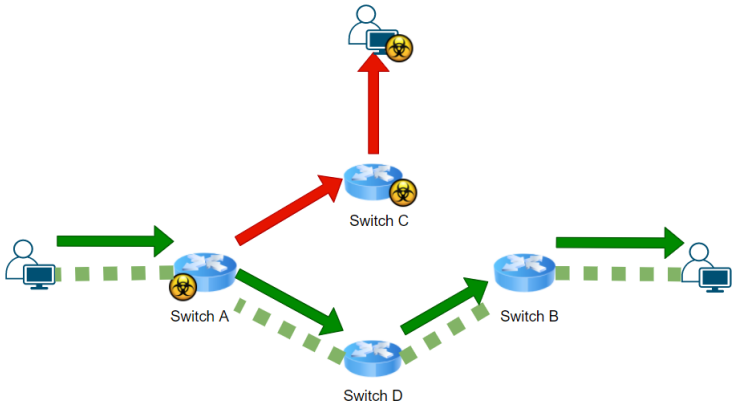


Figure 1.4: Packet Replay

(iii) **Packet Dropping:** A network black or grey gap is formed by a corrupted switch that dumps packets. In the past, it drops all packets; in the latter, it drops packets on a regular basis or re-transmits them, or it drops packets at random. Packet falling is used in attacks against network operators, such as Denial of Service (DoS).

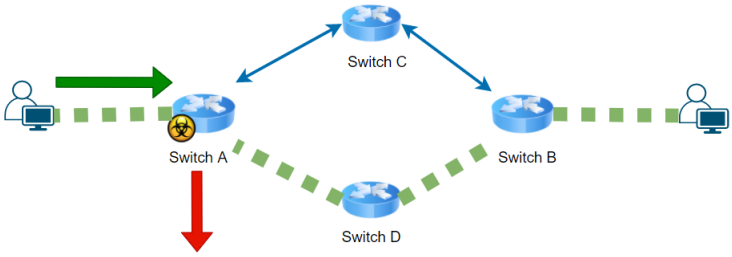


Figure 1.5: Packet Dropping

(iv) **Packet Delay:** A packet delay is a threat against time-sensitive traffic [14]. A delay in the TCP stream triggers false timeouts and unwanted re-transmission, placing the TCP throughput in jeopardy [15],

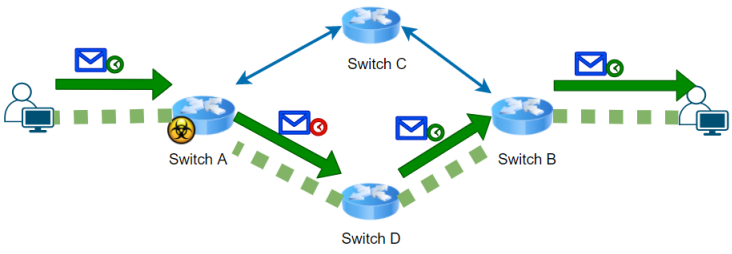


Figure 1.6: Packet Delay

(v) **Packet Manipulation:** The attacker may modify packets that pass the compromised switch. Though packets may be protected by encryption techniques, this attack can block all communications by decryption errors.

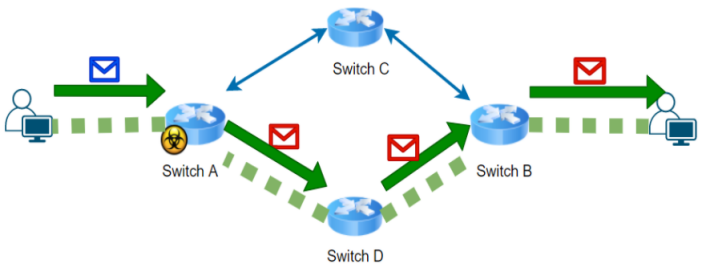


Figure 1.7: Packet Manipulation

All above attacks aim at flow actions since an OpenFlow switch processes packets according to programmed flow rules. Undoubtedly, an attacker may command the compromised switch to do more malicious behaviors other than the above attack models.

## 1.4 Security Threats From Compromised SDN Devices

The centralized control plane of SDN brings new security risks. Owing to the unavoidable software vulnerability, controllers have a potential risk of being compromised, and the whole network might be taken over by attackers even without being perceived by operators. The potential security threats brought by controller hijacking are mainly concluded as follows.

- **Maliciously Damaging:** The attacker may command a compromised controller to disable its switches by deleting all their flow table entries or to utilize its switches to launch DDoS attacks against servers, links, other controllers, switches, etc.
- **Resource Stealing:** The attacker may arbitrarily steal and utilize the network resources that belong to legitimate users by the compromised controller.
- **Monitoring and Manipulating:**The attacker may sneakily change the data flow between a pair of virtual machines through a certain switch for monitoring and manipulating.
- **Further Hijacking:** The attacker may continue to hijack other controllers for controlling more switches until achieves the attack objective.
- **Network View Tamper:** The attacker may tamper with the network view (topology) and flow table statistics (link utility), which will impact normal network Functions.
- **Log Tamper:** The attacker may tamper with the log of a compromised controller to prevent operators from detecting the hijacking attack or to accuse innocent SDN devices of misguiding operators.

Meanwhile, the switch (especially software switch) also has a risk of being compromised. Since the switch is the actual executant of controller instructions, compromised switches could bring

similar six kinds of security threats. In particular, switch hijacking is competent to enhance the impact of controller hijacking attacks.

## 1.5 Must-Have Characteristics of a Viable Solution

We propose that the 'should always' features of an effective approach against suspicious switches should include: [8] Considering significant variables and the shortcomings of current work such as SPHINX [16], we propose that the 'must-have' features of an effective approach toward malicious switches must include:

(i) To be able to identify intrusion attempts leveraging both hardware and/or software flaws of switches, there should be a reasonable dependency on predetermined rules and regulations for identification.

(ii) Ability to prioritize switches for examination in a structured and autonomous manner to enhance detection efficiency and rate of success.

(iii) To avoid executing overlapping and redundant procedures when reacting to attacks, Capability to identify malicious forwarding behavior and localize maliciousness.

(iv) Programmability for Threats: the capacity to modify actions when vulnerabilities are discovered.

(v) When identifying and reacting to attacks, the suggested solution should cause minimal system interruption, making it ideal for real-world network implementation.

In our study, we're attempting to come up with a solution that fits the following guidelines.

## 1.6 Problem Statement

In SDN, switches can be compromised by different kinds of attacks. Detection of compromised switches is a huge problem in the present time. We first need to identify if there is any compromised controller or not, if there is any compromised switch or not. If there is any, then we need to identify the compromised switches and also the specific attack for which the switch got compromised. And as the existing detection algorithm is dependent on the performance of the controller. We need a detection mechanism independent of the controller performance.

The problem can be described more specifically as:

*"Given a large-scale SDN network, how efficiently a system can detect a compromised device?"*

## 1.7 Overview of Our Solution Approach

The objective of our work is to develop a compromised SDN device detection algorithm with better complexity and which can be easily implemented in a real-world large-scale SDN architecture. We consider the scenario that both the controller and the switches can be compromised. The existing solutions to this problem are resource-heavy and time-consuming.

### 1.7.1 Clustering Technique

We will be using a different clustering Algorithm to cluster the switches into separate groups based on their frequency. It will assist us in finding greater complexity. We are currently focusing on the improved k-means Algorithm using Cluster Shifting and k++ initialization Algorithm for our research. We have implemented 1d K means algorithm for our experiments. It has reduced the time complexity needed for clustering of the forwarding devices.

### 1.7.2 Detection Mechanism

We want to improve the existing detection algorithm by adding a compromised switch identification mechanism. First, we localize the possible compromise switches. Then use probe packets to isolate the exact compromise switches. We collect statistics from switches for each packet to create expected path and actual path for each flow. Then we have implemented different kinds of active attacks then we detect those attacks.

### 1.7.3 Network Scalability

Network scalability is an issue in SDN architecture. The detection mechanisms usually depend on the statistics gathering by the controller performance. We want to use real-time statistics gathering independent of controller performance. We need to schedule the OpenFlow control messages at the switches. By using that we can get all those stat_reply messages at the same time.

## 1.8 Our Contribution

In summary, our contribution to this work is an improved detection system for compromised SDN devices.

- Detection algorithm will have better time complexity than the existing solutions.
- Detection algorithm can identify the specific compromised switches.
- Detection algorithm is independent of the performance of the controller so that it can be implemented in a scalable SDN network.
- Proposed system will be less vulnerable to an unwanted script running on the controller.
- Method is adaptable to multi-controller environments.

# Chapter 2

# Background Study

## 2.1 Wedge-Tail: An Intrusion Prevention System for the Data Plane of Software Defined Networks

Wedge-Tail [8] was built to keep the data plane of the SDN secure. This paper examines the issue of shielding SDNs from unauthorized switches by deciding whether the switch's traffic forwarding feature is stable. Wedge-Tail treats switches as nodes in a graph, and edges represent the paths packets follow when traversing the network. To save time, it uses an unsupervised trajectory-based sampling mechanism to prioritise switches until review. It is capable of automatically locating malicious forwarding devices and detecting the same malicious activity by observing the predicted and real trajectories of packets.

Researchers believe that a sophisticated enemy has taken complete possession of one or more of the switches. The attacker can perform different types of malicious activity. When a switch does not process packets according to the flows defined by the controller, it is referred to as "compromised." The data plane collects messages from the control plane and the specified policies in a stable and reliable manner. Except for their own name, the switches can lie about everything.

Wedge-Tail has two sections, Detection mechanism and Response program. The Detection program listens for OpenFlow messages between the controller and switches, creating a virtual simulation of the network that can be used to compute predicted packet path. The Response Machine, on the other hand, is installed as a program on top of the controller and assigns policies to the network OS, which decides how and when to implement them. For its inspection, Wedge-Tail emphasises forwarding instruments. The central principle is that the investigation should start with the switches that the bulk of packets meet as they travel across the network. When the Actual path [17] are not a part of the Expected path [18], then there maybe some corrupted switches in the path.

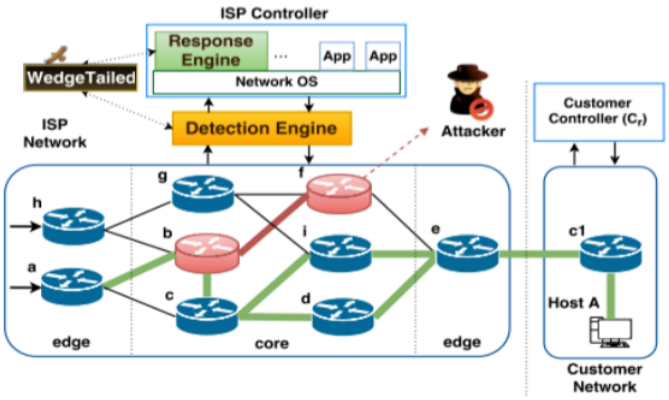


Figure 2.1: Wedgetail

### 2.1.1 Attack Detection

#### Packet Replay

When a switch sends a copy of a packet to a third destination in addition to the actual destination, this happens. An intruder can use a switch that replays packets to carry out surveillance and authentication attacks. This attack can be detected by checking if there is any unwanted switch in expected path or not, and if the packet reaches the destination or not. If the packet reaches the destination, the packet replay attack has occurred.

#### Packet Misrouting

When a packet is deviated from its actual destination and does not arrive at its final destination, this happens. This may be used to initiate a network availability attack or as part of a more complex threat. This attack can be detected by checking if there is any unwanted switch in expected path or not, and if the packet reaches the destination or not. If the packet does not reach the destination then packet misroute attack has occurred.

#### Packet Generation

The attacker may modify packets that pass the compromised switch. Though packets may be protected by encryption techniques, this attack can block all communications by decryption errors. Labeling is used to detect this attack. If labeling is changed then packet path becomes undefined

### Packet Delay

A packet delay is a threat against time-sensitive traffic [14]. A delay in the TCP stream triggers false timeouts and unwanted re-transmission, placing the TCP throughput in jeopardy [15]. This attack can be detected by checking the actual path time difference with the expected average path time.

### Packet Dropping:

A network black or grey gap is formed by a corrupted switch that dumps packets. In the past, it drops all packets; in the latter, it drops packets on a regular basis or re-transmits them, or it drops packets at random. Packet falling is used in attacks against network operators, such as Denial of Service (DoS). This attack can be detected by checking if the cardinality of the actual path is less than the expected path.

### Malicious Localization

A trajectory is regarded as a totally ordered set. By comparing Actual path and Expected path, it is possible to identify the corresponding forwarding system after one of the malicious actions has been identified.

## 2.2 SDN-RDCD: A Real-Time and Reliable Method for Detecting Compromised SDN Devices

SDN-RDCD [19],The aim of this paper is to solve the problem of identifying corrupted SDN devices where both the controller and the switch are untrustworthy. The basic concept is to use backup controllers to inspect the handling details of network update obtained from the primary controller and its switches, and to identify corrupted devices by identifying irregular or abnormal handling actions within the primary controller, backup controllers, and switches.

The controller and switch have a risk of being compromised.However, to keep our proposed method effective, we need to assume that at least one of the switches involved in a hijacking attack and at least one of the relevant auditor controllers are honest.

Master controller is designed to mirror every received legitimate network update request and the corresponding execution result to each of its auditor controllers. According to the inspection ID, the backup controller will create an audit record for each submitted valid network update request and re-execute every network update request. An auditor controller will add the received execution result of a certain network update request and its own execution result of this request to a matching audit Each audit record created by an auditor-controller will normally include five kinds of information:

1. the network update request.(Packet-In)
2. The execution result of the network update request in the master controller, denoted by $R_M$,
3. The execution result of the network update request in the auditor-controller, denoted by $R_A$,
4. The corresponding network update instructions received by relevant switches, denoted by $I_S$, and
5. the corresponding state updates in relevant switches, denoted by $R_S$.

If $R_M = I_S = R_S = R_A$, Then it means the corresponding four kinds of information are consistent and the network update request was correctly handled by the master controller and relevant switches.

### 2.2.1 Attack Detection

- $R_M = I_S = R_S \neq R_A$. This means the execution results of the network update request in the master and auditor controllers are inconsistent and the master controller might be

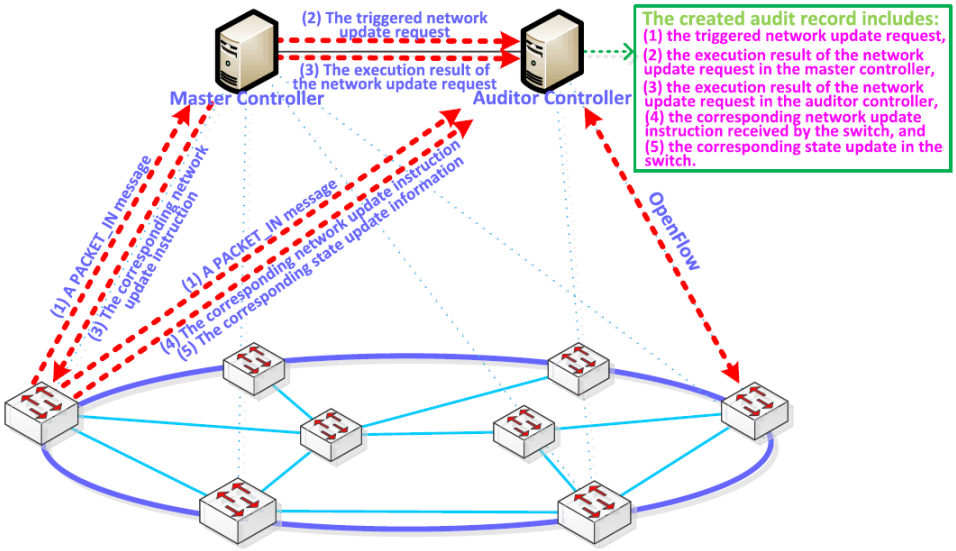


Figure 2.2: SDN-RDCD

compromised.

- $R_M = I_S \neq R_S$. This means some network update instructions were incorrectly performed in some switches and those switches might be compromised.
- $R_M \neq I_S$. This means some network update instructions were possibly manipulated during the transmission from the master controller to some switches and man-in-the middle.
- Unmatched switch state update. This means a state update in some switch fails to match any audit record and the switch is thus suspected to surreptitiously perform a state update without the network update instruction from the master controller.

## 2.3 FOCES: Detecting Forwarding Anomalies in Software Defined Networks

FOCES [20] endeavors to address stream counter condition frameworks, which require registering grid reversals. This can be expensive when there are countless streams and rules, To make FOCES adaptable, we propose a technique by cutting the first huge FCM into numerous more modest sub-FCMs, whose network reversal can be registered a lot quicker.

- FOCES [20] proposes, a network-wide sending inconsistency discovery technique in SDN, which can check whether all streams in a network are sent accurately at the same time, without introducing additional guidelines.

- FOCES [20] hypothetically investigate the condition for effective discovery utilizing FOCES and lessen the condition to the issue of finding a circle in a bipartite diagram.
- FOCES [20] plans an edge-based location calculation to dispose of bogus positives brought about by counter commotions, and a cutting-based technique to make FOCES adaptable for bigger networks.
- FOCES [20] utilizes tests to show FOCES can precisely identify sending inconsistencies in four network geographies, with insignificant calculation overhead.

The foe intends to change the ways that packets ought to be sent, subsequently causing what we call sending inconsistency.

- Way Deviation: Packets take an unexpected way in comparison to what is normal by the regulator.
- Switch Bypass: Packets are gotten by the objective switch, yet at least one switches are skipped.
- Way Detour: Packets go amiss starting with one switch Si then onto the next switch other than the expected next-bounce Si+1, and return to Si later.
- Early Drop: Packets are dropped prior to arriving at the objective switch.

It is certainly expected the last-bounce switch isn't undermined, as it can drop packets imagining that packets are gotten by the end has.

In light of the above thought, FOCES [20] first concentrates the imperatives that all counters in the network ought to fulfill, from the network arrangements (e.g., stream tables) in the control plane. At that point, FOCES [20] gathers counters from the information plane and checks whether these counters fulfill the requirements. On the off chance that the requirements are abused, there should be some sending oddities in the network.

## 2.4 WhiteRabbit: Scalable Software-Defined Network Data-Plane Verification Method Through Time Scheduling

This paper,WhiteRabbit [21], makes the accompanying commitments. It presents the arrangement of the measurements gathering with time planning using Scheduled Bundle in WhiteRabbit [21]. By social affair the exchange measurements usingScheduled Bundle all the while, WhiteRabbit [21] can accumulate the insights without relying upon controller execution. This paper present the verication calculation for utilizing the insights assembled in planned time dependent on the byte consistency check in SPHINX [22]. It analyzes the passed time for sending messages to all switches and the booking mistake. it additionally reports the planning mistake is lower than the time needed in sending the messages to all switches. It assesses WhiteRabbit [21] to show that it can accomplish lower bogus positives than SPHINX, and it doesn't influence packet moving.

WhiteRabbit [21] can be used not exclusively to recognize traded off switch practices yet additionally to find network deserts. Concerning suspicion of SPHINX, we accept that the controller applications are trusted and most of the switches are real. Consequently, the messages from the controller are dependable, though those messages from any switches might be produced by undermined switches. To center the investigation on just OpenFlow control messages, we consider that the verier knows the solid actual geography data and accept a shut SDN framework. Furthermore, we accept that the hours, everything being equal, also, controllers are synchronized precisely through the time synchronization convention, like PTP. Despite the fact that undermined switches synchronized time like real switches and masked the byte move insights, numerous other switches report real insights. The distinction in byte move measurements is, therefore, higher than that with the traded off and many authentic switches. In this way, WhiteRabbi [21]t can recognize assaults by undermined switches

- The approval sequence to a specic traffic flow utilizing WhiteRabbit [21] is as following:

  1) Calculate the way that the controller assumes, utilizing actual geography data and FLOWMOD messages sent from the controller application.

  2) Obtain the real exchange measurements of all switches utilizing Booked Bundle all the while.

  3) Validate the exchange state consistency utilizing the expected way of the controller and the distinction of measurements among the adjoining switches.

WhiteRabbit [21] requires an flow graph, which is a diagram hypothetical viewpoint of the network expected by a confided in controller, to get a current way like that of SPHINX. The flow graph is developed just utilizing the FLOWMOD messages given by the confided in controller.

It incorporates coordinate eld and guidance, including a src/dst MAC address, src/dst IP address, and in/out port data of the switches. The flow diagram doesn't experience the ill effects of untrusted switches on the grounds that the untrusted STATSREPLY messages are not utilized in constructing this diagram.

The current way expected by the believed controller can be get by joining data of FLOW-MOD messages also, actual geography. The current way is utilized for recognizing the switches through which a specic trafc passes when during the verication execution.

WhiteRabbit [21] intermittently assembles move measurements in genuine time from the switches. When gathering measurements, a controller for the most part sends STATSREQUEST messages all the while, and the time needed to send these messages relies upon the controller execution. Specifically, WhiteRabbit [21] utilizes STATSREQUEST wrapped with Scheduled Group since it accumulates insights progressively between switches. Moreover, it can accumulate insights without depending on the controller execution. The insights distinction between the switches may happen attributable to the circumstance of FLOWMOD message sent by means of the controller application relying upon the system of the directing control. Moreover, the insights of constant social affair depend on the presentation of the switches, for example, Flow table size also, plan execution exactness. Accordingly, WhiteRabbit employments moving midpoints of the distinction of the last four measurements report (i.e., use bytecnt contrast) all the while between switches, alluded to as ByteDiff . Since this span is sufficient to kill the impacts of scheduling mistakes and traffic blasts, our component can maintain a strategic distance from bogus cautions. Dissimilar to with of SPHINX [22], given that scheduling instrument assembles the measurements progressively, ByteDiff of WhiteRabbit doesn't rely upon the controller execution.

### 2.4.1 Attack Detection

To begin with, this calculation approves the insights of the switches over a current way of the traffic stream F from the closest change from a source have. Thus, the calculation needs ought to think about the distinction of the measurements esteems that happen attributable to spread deferral, scheduling mistake, and the distinction of the stream table sizes kept up by the switches. Second, the calculation confirms whether the insights of the switches that are excluded from the current way related with the traffic stream F are zero. Accordingly, it can affirm that no traffic has been infused and dropped by the switches that are out of the current way.

The calculation requires the limit ( ) as an info, which is utilized as the edge of the insights esteem comparability. Taking into account that ByteDiff fluctuates with correspondence circumstance, the calculation ascertains the most extreme/least ByteDiff by increasing PrevByte

with the limit. Furthermore, in light of the fact that the presentation of insights gathering relies upon the switch execution, which happens from the timetable execution precision, stream table size [22], and switch usage , should be controlled by considering the switch execution. On the off chance that the estimation of in this calculation is altogether huge, at that point bogus negatives may happen and a certifiable caution may not be yielded. Conversely, if the estimation of is minuscule, at that point the calculation may result in bogus positives. Subsequently, the director ought to altogether decide the estimation of .

## 2.5 SPHINX: Detecting Security Attacks in Software-Defined Networks

SPHINX [22] analyses specific OpenFlow control messages to discover new system behavior and information for both geometrical and forwarding states, and creates flow graphs for each traffic stream seen in the network. It constantly refreshes what's more, screens these stream diagrams for allowable changes, and raises cautions on the off chance that it recognizes degenerate conduct. SPHINX influences custom calculations that gradually interaction network refreshes to decide in realtime if the updates causing deviant conduct ought to be permitted or not. SPHINX [22] additionally gives a light-weight strategy motor that empowers chairmen to determine expressive arrangements over network assets and identify security infringement. Not at all like the present controllers where every module actualizes its own checks making strategy implementation cart, SPHINX gives an essential issue to implementing complex approaches.

This paper makes the accompanying commitments:

- The paper analyze four famous SDN controllers and illustrate that they are defenseless against a different cluster of assaults on network geography and information plane forwarding.
- Sphinx present gradual stream charts as a novel deliberation for realtime recognition of security dangers.
- The paper present the plan and execution of SPHINX and its arrangement motor, which permits network executives to indicate fine-grained security arrangements, and empowers simple activity attribution.
- It evaluates the detector to see if it is functional and has adequate overheads. We moreover report on encounters acquired utilizing SPHINX in four extraordinary contextual analyses .

Attackers frequently break into the network to use inside vantage focuses, and hence dispatch

assaults on the inward network. Since the objective is to (I) check beginning of assaults on network geography and information plane forwarding, and (ii) recognize infringement of approaches inside SDNs, our danger model centers solely around situations where the enemy starts assaults from inside the SDN. Consequently, we model SDNs as a shut framework. Eliminating requirements on the obscure outside correspondence helps center our investigation just around OpenFlow control messages inner to the SDN

### 2.5.1 Attack Detection

Flow graphs are used by SPHINX to model network topology and data plane forwarding in SDNs. It extracts flow metadata from OpenFlow control messages and constructs flow graphs incrementally to closely resemble current network activities, allowing real-time validation of all network changes and restrictions on any flow graph in the network. As a result, flow graphs offer a simple method for detecting various constraint breaches in SDNs, including network topology and data plane forwarding. Since FLOWMOD messages are sent by the trustworthy controller, only FLOWMOD messages are used to create flow routes. Untrusted STATSREPLY messages from each switch only change the corresponding switch's flow statistics and have no effect on the flow graph structure. As a result, even in the case of untrusted switches and hosts, the flow-specific network topology and data plane forwarding condition as embodied in the flow graph remains unaffected. Furthermore, the presence of an honest plurality of switches along the flow path allows SPHINX to specifically identify any deceptive changes to flow statistics at any switch along the flow path.

To detect intrusions occurring within the SDN, flow graphs use the predictive power and structure of both topological and data plane forwarding observed from control messages. Although flow graphs are a useful instrument for verifying natural and consistent network operations, the existence of messages sent over the control plane and the dynamism of the topology limit their capabilities. If a large number of messages have been tampered with or are untrustworthy, flow graphs will interpret inaccurate messages as common actions and will not raise any alarms. Furthermore, if the topology of the network varies repeatedly, some of the taught invariants can be broken, leading to attacks.

A Flow is a coordinated traffic design saw between two endpoints with unmistakable MAC addresses over determined ports. A flow diagram is a conceptual illustration of a traffic flow in which the edges represent the flow information and the nodes represent the switches. SPHINX utilizes these Flow charts to demonstrate both network geography and information plane forwarding in SDNs.It gathers Flow metadata from OpenFlow control messages and gradually assembles the Flow charts to intently rough the real network tasks, accordingly empowering approval of all network updates and limitations on each Flow diagram in the network in real-time. Hence, Flow diagrams give a clean system that guides discovery of different imperative

infringement for both network geography and information plane forwarding in SDNs.

Flow ways are developed just utilizing FLOWMOD messages since they are given by the confided in controller. Untrusted STATSREPLY messages from each switch just update Flow measurements of the comparing switch, and don't influence the Flow chart structure. Consequently, the Flow explicit network geography and information plane forwarding state as typified in the Flow chart stays uncorrupted even within the sight of untrusted switches,the presence of a genuine dominant part of switches along the Flow way empowers SPHINX to exactly identify any malignant updates to Flow insights at any switch in the Flow way. Flow charts misuse the consistency and example in both topological and information plane forwarding derived from control messages to distinguish assaults beginning inside the SDN. While Flow diagrams are a compelling apparatus to confirm typical furthermore, unsurprising network activities, they are restricted in their capacities by the idea of messages sent over the control plane and the dynamism in the geography. On the off chance that there is a dominant part of altered or untrusted messages, at that point Flow diagrams will see off base messages as ordinary conduct and not raise any cautions. Further, if the network geography changes very habitually, at that point a few of the learned invariants might be disregarded, bringing about cautions.

## 2.6 HSA

Header Space Analysis(HSA) [18] is a strategy for troubleshooting network setup. HSA treats a L-bit packet header as L-dimensional space, and models all cycles of switches and center boxes as move capacities, which change subspaces of the L-dimensional space to different subspaces. Along with esteemed lines, by breaking down the sending rules of the network, HSA can figure the way a packet traversing the network on a specific port will take.

## 2.7 NetSight

NetSight [17] is a network investigating arrangement that permits the SDN application to recover the packet history. netshark is an illustration of apparatuses worked over this stage, which empowers clients to characterize and execute channels on the whole history of packets. With this apparatus, a network administrator can likewise see the total rundown of packet properties at each jump, like an info port, a yield port, and packet header esteems.

# Chapter 3

# Proposed Method

The objective of our work is to develop a compromised SDN device detection algorithm with better complexity and which can be easily implemented in a real-world large-scale SDN architecture. We consider the scenario that both the controller and the switches can be compromised. The existing solutions to this problem are resource-heavy and time-consuming. There are three sections to our proposed system.

## 3.1 Clustering Technique

Previous detection methods use unsupervised trajectory sampling Algorithms to switches into separate groups. But the problem with this method is that it is a time-consuming method and it performs poorly when we consider the whole network. We will be using a different clustering Algorithm [23] to cluster the switches into separate groups based on their frequency. It will be faster and more efficient considering we take the whole network at a time. We are currently focusing on the different modified versions of the k-means algorithm that can be used for our purpose.

### 3.1.1 K means Clustering

A cluster is a set of data points that have been clustered together due to similarity. the K-means algorithm identifies the k number of centroids, and then allocates every data point to the nearest cluster while keeping the centroids as small as possible. One of the most common and frequently used unsupervised machine learning algorithms is K-means clustering.

### 3.1.2 K-means++ Initialization

The K-means algorithm has the downside of being susceptible to the initialization of the centroids or mean points. As a consequence, if a centroid is set to be a "much further" point, it might have no points associated with it. Similarly, if more than one centroids are allocated into the same cluster, the clustering would be weak. We use K-means++ [24] to solve these pitfalls. This algorithm ensures that the centroids are properly configured and that the clustering efficiency is increased. The majority of the algorithm is similar to the regular K-means algorithm, with the exception of initialization. That is, K-means++ is the traditional K-means algorithm with better centroids initialization. While K-means++'s initialization is more computationally costly than the regular K-means algorithm, K-means++'s runtimes for convergence to the optimum is reduced significantly. This new initialization method makes k-means more accurate and fast.

### 3.1.3 K-means using Cluster Shifting

The quadratic time complexity of the k-means algorithm is well known. As a consequence, the algorithm cannot be used successfully in large - scale applications. The classical method can be used to construct a linear time complexity k-means algorithm. To achieve this linear time complexity for the k-means algorithm [25], a technique of gradual shifting of intermediate clusters, at consecutive iterations has been used.

### 3.1.4 1D K-means Clustering

**kmeans1d** is a Python library that implements k-means clustering on 1D data, relying on the methodology provided in section 2.2 of (Gronlund et al., 2017) [26] by (Xiaolin 1991) [27]. The Lloyd's algorithm is a well-known method for determining a locally efficient solution. There are polynomial-time algorithms for 1-dimensional input. This algorithm has a complexity of O(kn + n logn). C++ and Python are used to write the code.

For our implementation, we have used the 1D K-means clustering [26] as it is more efficient and suited for our purpose. It also provides a better time complexity.

## 3.2 Detection Mechanism

We want to improve the existing detection algorithm by adding a compromised switch identification mechanism. First, we localize the possible compromise switches. Then use probe packets to isolate the exact compromise switches.

### 3.2.1 Custom Topology

We have created custom SDN topology using python script in mininet emulator. We have assign mac address, ip address, port address to each hosts so that they can be easily identified. We will use this topology for our detection implementation.

### 3.2.2 Attack Implementation

We have implemented different kinds of attacks using rest API applications. We have added, modified, deleted flows for creating different kinds of active attacks. Attacks include Packet Replay, Packet Misroute, Packet Drop, Packet generation, Packet modification.

### 3.2.3 Expected Path Calculation

We calculate expected path for every packet by collecting statistics from every switch. The flows are filtered according to our needs. We can collect packet count, byte count etc based on each flow. We can use them to generate expected path for each packet. For our experiments we have used k-means algorithm here.

### 3.2.4 Actual Path Calculation

We can calculate the actual path similar in the similar manner we calculated the expected path. But Generating actual path is very tricky as the a compromise switch may lie about anything except it's identity.

### 3.2.5 Attack Detection

We can detect any compromise switch by checking the difference between expected path and actual path. Then we can isolate the compromised switches and specify the type of attacks.

## 3.3 Network Scalability

In SDN architecture, network scalability is a concern. The detection mechanisms usually depend on the statistics gathering [28] by the controller performance. We want to use real-time statistics gathering independent of controller performance. We need to schedule the OpenFlow control messages at the switches.

# Chapter 4

# Experimental Setup

Traditional networks' static architecture is decentralized and complex, while today's networks need more simplicity and ease of troubleshooting. SDN was developed to fix this issue. SDN recommends centralizing network information in a single network component ie. control plane by decoupling network packet forwarding (data plane) from the routing process.The control plane consists of some controllers who serve as the network's brain. Control plane is centralized and runs on a computer. Two planes ie. control plane and data plane are separated from each-other. SDN network is programmable and application and network services are ensured by the abstraction of underlying architecture. SDN implementation is highly depended on OpenFlow protocol.

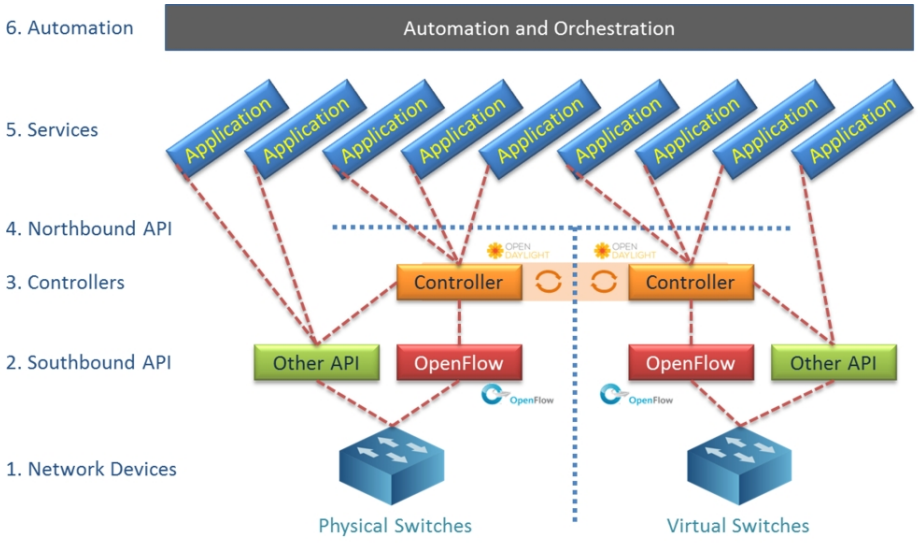


Figure 4.1: SDN Bigger Picture

SDN architectures mostly made of three components or functionality:

- **SDN Networking Devices / Network Infrastructure:** Network devices mainly focus on the forwarding of packets and data processing. Data path identification is also a major

task of network devices. Example: Openvswitch (OpenFlow protocol)

- Southbound interface makes the connection between physical networking hardware and the controller.Using this interface controller communicates with the lower layers of the architecture.

- **SDN Controller:** SDN controller is the middle man between application layer and networking devices. It receives instructions from the application layer and pass them to the intended devices of the network. Controller also does the job of collecting statistics from the network devices and send necessary information to the Application layer. Controller contains abstract view of the whole network. Example: RYU, OpenDayLight [29], ONOS [30], FloodLight etc

- Northbound interface is the communication medium between the controller and upper layer applications.

- **SDN Applications:** SDN Application uses an application programming interface(APIs) through which it provides the resources and instructions needed by the controller. These applications indirectly manage the whole network and process large data and give analytical decisions. This is the area of creativity, inventions, research, etc.

## 4.1 SDN Controllers Overview

The Controller uses OpenFlow protocol to configure the network devices of the Software-defined network. SDN Controller uses southbound API to instruct the switches of the actions that need to be performed. Packets in the switches are forwarded according to the flow table. Northbound interface is used for providing information to the application layer. Controller is positioned in the middle of applications and devices. Openflow is widely used as the communication protocol between controllers and network devices.

The controller is in charge of the manipulation of Flow Tables in a switch. When a packet comes to a switch there are two scenarios that can happen. One is Packet matches an existing flow entry of the switch and switch takes necessary forwarding actions according to the flow entry. Another scenario is when a packet comes and doesn't match with existing flow entries. In this case, switch sends a Packet_In message to the controller for further instructions or drop the packet.

- **POX:** POX is an open-source controller that provides an efficient way to use and implement Openflow Protocol. Different applications are supported in POX controllers like switch, hub, load-balancer, etc.

- **RYU:** RYU is an open-source SDN controller developed in Python language. It is modular and provides a network operating system. New control application can be constructed using RYU. It supports all the versions of OpenFlow protocol(v1.0,v1.2, v1.3, v1.4, v1.5). RYU also supports Ofconfid and Netconf.
- **OpenDayLight:** OpenDayLight is an open-source SDN controller whose aim is to create robust code that can be used on the devices of SDN architecture. It is mainly developed for commercial products and wants acceptance among the users.
- **ONOS:** ONOS is developed in JAVA language and mainly used in Telecom used areas. It comes with operator support for the users.
- **Floodlight:** Floodlight is an open-source JAVA based SDN controller. It is considered an enterprise-class and has Apache-license. It comes with several modules. Modules use API to give services to applications as well as other modules.

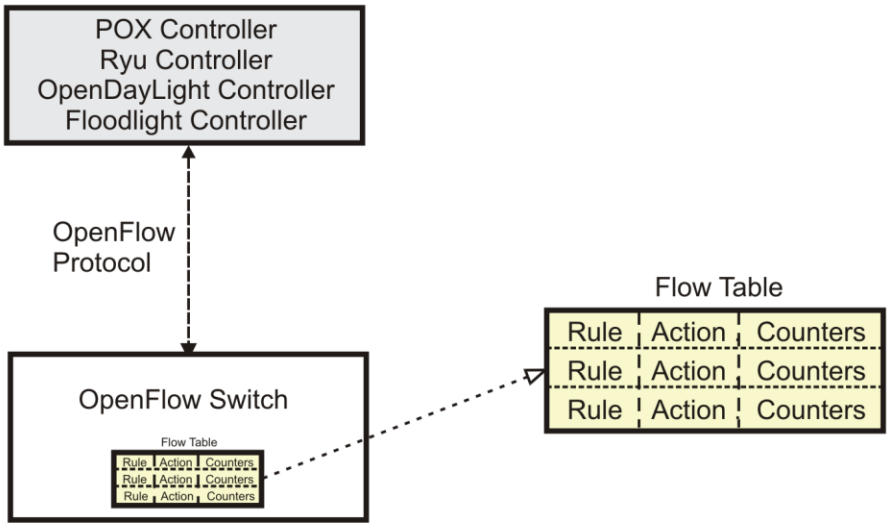


Figure 4.2: OpenFlow

## 4.2 OpenFlow

OpenFlow [31] is an SDN standard that has two main entities, OpenFlow Switch and OpenFlow Controller.OpenFlow Switch is the forwarding plane of the network. On the other hand, Control plane is implemented by controller. These two entities communicate with each other with the help of Openflow protocol. Openflow switches have software installed in them to communicate with the controller. Switch has flow tables and forwards the packet. Quality of service is ensured by a Meter table.OpenFlow standard describes the network devices and functions of switches

and OpenFlow switch protocol is used to manipulate the switches from a Controller. Openflow Version Details:

Most widely supported: Openflow 1.3

### 4.2.1 Switch Components

OpenFlow [31] Logical Switch contains flow tables as well as group tables for the purpose of packet forwarding and packet lookup. The switch has OpenFlow channels to a controller.

The switch communicates with the controller using the OpenFlow switch protocol, and the controller controls the switch. The controller can add, refresh, and uninstall flow entries in flow tables both reactively (in response to packets) and proactively (in advance) using the OpenFlow switch protocol. Each flow table in the switch has a collection of flow entries, which are made up of match fields, counters, and directions to apply to matching packets.

The matching process begins with the first flow table and may extend to all flow tables in the pipeline. Flow entries are used to align packets in order of precedence, with the first corresponding entry in each table being used. The instructions associated with the unique flow entry are followed if a similar entry is detected. If no match is made in a flow table, the result is determined by the table's configuration. If the packet fails to enter the flow table, it may be forwarded to the controllers over the OpenFlow channel, dropped, or continued to the next flow table. Packet routing, packet adjustment, and group table processing are all described in the instructions. Flow entries may be routed to a specific port. This is normally a physical port, but it may also be a switch-defined logical port (such as link aggregation classes, tunnels, or loopback interfaces) or a reserved port defined by this specification.

### 4.2.2 Openvswitch

Open vSwitch [32] is a high-performance multilayer virtual switch licensed under the Apache 2.0 open-source license. It's designed to support common management interfaces and protocols while allowing for vast network automation by programmatic expansion (e.g. NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag).

### 4.2.3 Openflow channel

The OpenFlow [31] channel serves as the connection between each OpenFlow Logical Switch and an OpenFlow controller. The controller uses this interface to set up and monitors the transfer, as well as accept events from it and send packets out of it. A single OpenFlow channel

with a single controller may be supported by the switch's Control Channel or several OpenFlow channels with multiple controllers may be supported by the switch's Control Channel. The OpenFlow channel is usually encrypted with TLS, but it can also be run over TCP. 6653 is the default port code.

### 4.2.4 SDN Switch(Openflow Switch)

SDN Controller IP and Openflow protocol version are installed on the switch. The contact between the switch and the SDN controller is created. The default OpenFlow rule (TABLE MISS ENTRY) is installed by SDN Controller in the turn flow table. MISSED TABLE ENTRY The CONTROLLER receives all packets that match the OpenFlow law. In the table, the lowest priority comes first (0) The Host Data Packet will be paired with TABLE MISS ENTRY as it arrives in the Switch, and the packet will be sent to the Controller ( PACKET IN Message) The packets are received by the controller, and the packets are used to construct the Switch Logic. The OpenFlow flows are added to the switch by the controller. The controller adds the OpenFlow flows to the switch. Now, the Switch data path is built with flows. So next time, when the Packet arrives it will be matched with the Flow table and forward the packet to the respective port.

### 4.2.5 Counters

Counters are used to maintain and collect statistics about the network from OpenFlow Switches. Counters are implements on a per-flow entry basis, per-flow table basis, per switch port basis, and many more.

### 4.2.6 OpenFlow Flow Table

A flow table entry is defined by its match fields and priority: when the match fields and priority are combined, a single flow entry in a given flow table is identified. Flow entries make up a flow chart.

The table-miss flow entry is a flow entry that has precedence equal to 0 and wildcards all match fields (all fields omitted). At the very least, the table-miss flow entry must allow packets to be sent to the controller through the CONTROLLER reserved port.

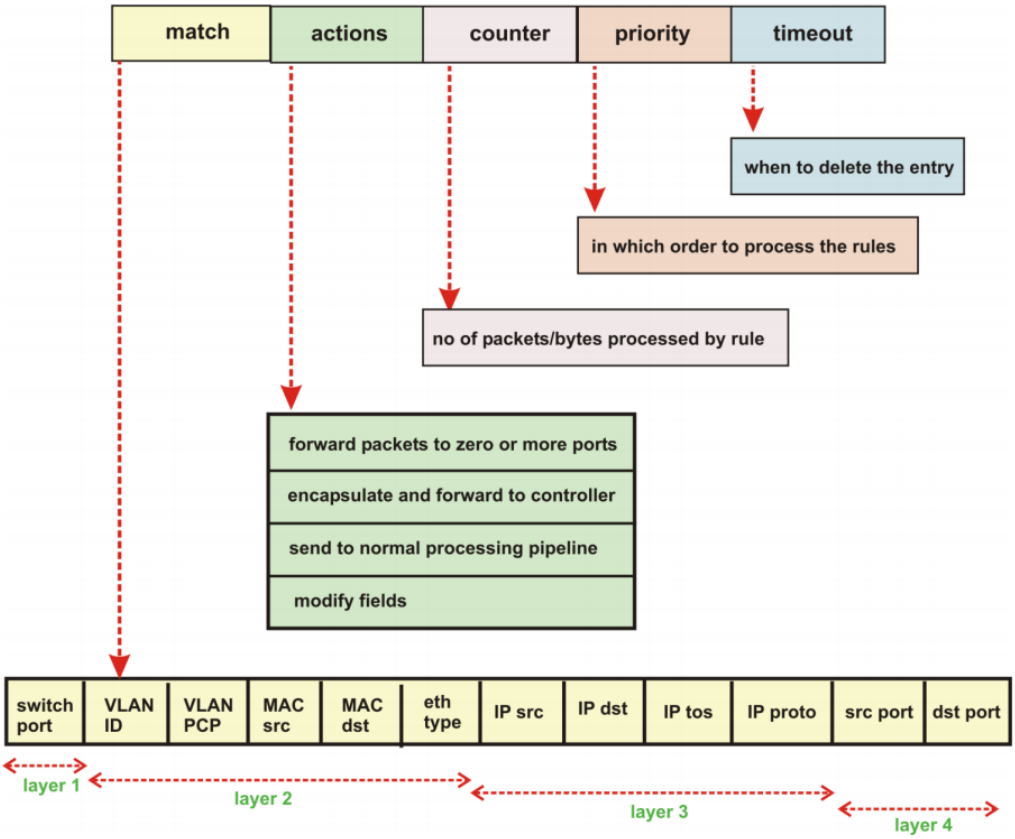


Figure 4.3: OpenFlow Flow Table

### 4.2.7 OpenFlow Matching

Depending on pipeline processing, the switch can perform table lookups in other flow tables after performing a table lookup in the first flow table. There are two kinds of match fields: header match fields and pipeline match fields. Match fields derived from packet headers are known as header match fields. The majority of header match fields correspond to a field in the packet header specified by a data path protocol. Sizes, prerequisites, and masking capabilities vary across header match fields. Pipeline match fields, such as METADATA and TUNNEL ID, are match fields that match values added to the packet for pipeline processing but are not correlated with packet headers.

## 4.3 Tools used for our experiment

**OS:** Ubuntu 20.04 Desktop Edition

**Test Bed:** Mininet Emulator

**Controller:** RYU Controller

**Switch:** Openvswitch

**Packet Capture:** Wireshark Packet Analyzer

**Traffic Generator:** IPerf

## 4.4 Mininet

Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking.

Mininet provides an easy way to get correct system behavior and to experiment with topologies. Mininet networks run real code including standard Unix/Linux network applications as well as the real Linux kernel and network stack including any kernel extensions which you may have available, as long as they are compatible with network namespaces. Because of this, the code you develop and test on Mininet, for an OpenFlow controller, modified switch, or host, can move to a real system with minimal changes, for real-world testing, performance evaluation, and deployment. Importantly this means that a design that works in Mininet can usually move directly to hardware switches for line-rate packet forwarding.

## 4.5 WireShark

Wireshark [33] intercepts traffic and converts that binary traffic into human-readable format. This makes it simple to see what traffic is passing through the network, how much of it is passing through, how often it is passing through, how much latency there is between those hops, and so on. Although Wireshark supports over 2,000 network protocols, many of which are obscure, rare, or obsolete, the current security specialist will find that analyzing IP packets is the most helpful. TCP, UDP, and ICMP are expected to make up the bulk of packets on your network. Given the high amount of traffic that passes through a standard business network, Wireshark's methods for filtering the traffic are particularly useful. Capture filters can only collect the kinds of traffic that you're interested in, while view filters will let you zoom in on the traffic you're looking at. The network protocol analyzer includes search features such as regular expressions and colored coloring to help users find what they're searching for.

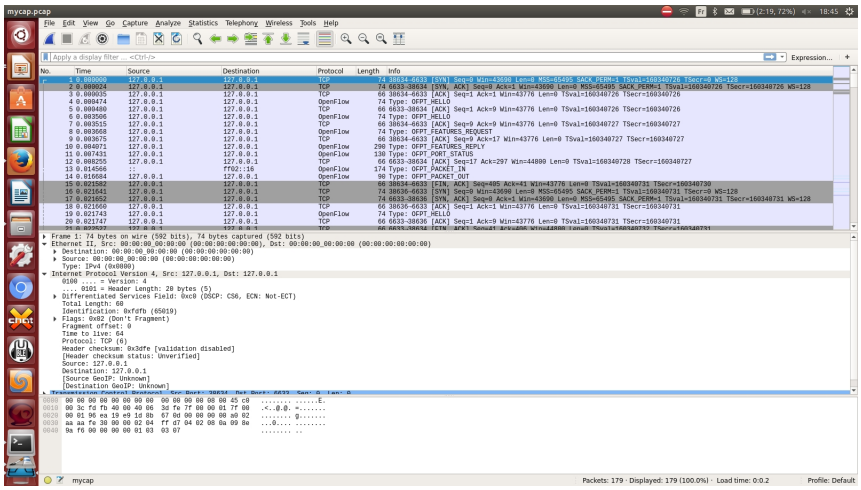

Figure 4.4: Wireshark

## 4.6 Time4

TIME4 [28], an approach that uses accurate time to coordinate network updates. TIME4 [28] is a powerful tool in softwarized environments, that can be used for various network update scenarios, including in heavily utilized networks. Specifically, we characterize a set of update scenarios called flow swaps, for which TIME4 [28] is the optimal update approach, yielding less packet loss than existing update approaches without requiring spare capacity, and without temporarily reducing the network's bandwidth.The prototype is publicly available as open source. TIME4 [28] requires the switches and controller to maintain a local clock, enabling time-triggered events. Hence, the local clocks should be synchronized. The OpenFlow time extension defined does not mandate a specific synchronization method. Various mechanisms may be used, e.g., the Network Time Protocol (NTP) [34], the Precision Time Protocol (PTP) [35],or GPS-based synchronization. The prototype we designed and implemented uses REVERSEPTP [36], a variant of PTP that is customized for SDN.

## 4.7 Flow-Manager

The Flow-Manager is an RYU controller that allows users to manually manipulate the flow tables in an SDN network. Directly from the program, someone can build, alter, and uninstall flows. we will also display statistics and watch the OpenFlow switches. The Flow-Manager is suitable for studying SDN in a lab setting or for tweaking the actions of network flows in a production setting when used in combination with other applications.

# Chapter 5

# Conclusion

## 5.1 Summary

For traditional networks, identifying corrupted network devices has long been a concern. Misbehavior identification for conventional network equipment, such as routers, switches, and hosts, is the focus of relevant study. Our system for detecting corrupted SDN devices, on the other hand, is expected to yield much better performance. We have modeled the SDN network with a mininet emulator. The OpenFlow messages are analyzed using the WireShark [33] tool. The RYU controller, which is based on Python, is used. Switch clustering was applied to locate the most commonly used switches. For coordination of control messages, Time4 technology would be introduced as an API on top of the Controllers. However, several simulations are to be used to test our verifier's compliance with distributed controller environments.

One of our challenges was switch clustering. It takes a long time to perform a detection mechanism through an entire network. We used a number of clustering methods in our solution for this study. We have implemented switch clustering with the help of the 1d k-means algorithm. It provides better complexity than previous methods. Another difficulty was determining the individual switches that had been exploited as well as the method of attack so that adequate responses could be made. We need to improve the existing algorithm to identify the individual corrupted device. And the last challenge was to make the verifier controller independent. We used a schedule bundle to collect statistics of switches in real-time to get better accuracy. Our entire setup of the experiment is based on ideal situation assumptions and theoretical calculations. We hardly tested the setup in a practical situation with widespread unpredictability. We're creating a plan to deploy our setup in a real-world scenario.

## 5.2 Future Work

Our tests were carried out using a single controller ( RYU controller). We must verify our verifier's compliance with distributed controller environments such as ONOS and OpenDayLight controllers. The real-world environment varies from the emulated environment in a variety of aspects. Extensive research in real-world conditions will be needed in the future. This algorithm is incapable of identifying corrupted Edge switches. It is important to identify corrupted edge switches in order to provide End-to-End verification.

# References


[1] R. Klöti, V. Kotronis, and P. Smith, "Openflow: A security analysis," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, pp. 1–6, IEEE, 2013.

[2] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[3] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[4] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623–654, 2015.

[5] S. T. Ali, V. Sivaraman, A. Radford, and S. Jha, "A survey of securing networks using software defined networking," *IEEE transactions on reliability*, vol. 64, no. 3, pp. 1086–1097, 2015.

[6] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 55–60, 2013.

[7] A. Feldmann, P. Heyder, M. Kreutzer, S. Schmid, J.-P. Seifert, H. Shulman, K. Thimmaraju, M. Waidner, and J. Sieberg, "Netco: Reliable routing with unreliable routers," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, pp. 128–135, IEEE, 2016.

[8] A. Shaghaghi, M. A. Kaafar, and S. Jha, "Wedgetail: An intrusion prevention system for the data plane of software defined networks," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pp. 849–861, 2017.

[9] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage, "Detecting and isolating malicious routers," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 3, pp. 230–244, 2006.

[10] A. T. Mzrak, S. Savage, and K. Marzullo, "Detecting malicious packet losses," *IEEE Transactions on Parallel and distributed systems*, vol. 20, no. 2, pp. 191–206, 2008.

[11] T. H.-J. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig, "Lightweight source authentication and path validation," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, pp. 271–282, 2014.

[12] S. Lee, T. Wong, and H. S. Kim, "Secure split assignment trajectory sampling: A malicious router detection system," in *International Conference on Dependable Systems and Networks (DSN'06)*, pp. 333–342, IEEE, 2006.

[13] P.-W. Chi, C.-T. Kuo, J.-W. Guo, and C.-L. Lei, "How to detect a compromised sdn switch," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–6, IEEE, 2015.

[14] R. Ghannam and A. Chung, "Handling malicious switches in software defined networks," in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 1245–1248, IEEE, 2016.

[15] Y. J. Zhu and L. Jacob, "On making tcp robust against spurious retransmissions," *Computer communications*, vol. 28, no. 1, pp. 25–36, 2005.

[16] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "Sphinx: detecting security attacks in software-defined networks.," in *Ndss*, vol. 15, pp. 8–11, 2015.

[17] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown, "I know what your packet did last hop: Using packet histories to troubleshoot networks," in *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, pp. 71–85, 2014.

[18] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, pp. 113–126, 2012.

[19] H. Zhou, C. Wu, C. Yang, P. Wang, Q. Yang, Z. Lu, and Q. Cheng, "Sdn-rdcd: A real-time and reliable method for detecting compromised sdn devices," *IEEE/ACM Transactions on Networking*, vol. 26, no. 5, pp. 2048–2061, 2018.

[20] P. Zhang, S. Xu, Z. Yang, H. Li, Q. Li, H. Wang, and C. Hu, "Foces: Detecting forwarding anomalies in software defined networks," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 830–840, IEEE, 2018.

[21] T. Shimizu, N. Kitagawa, K. Ohshima, and N. Yamai, "Whiterabbit: Scalable software-defined network data-plane verification method through time scheduling," *IEEE Access*, vol. 7, pp. 97296–97306, 2019.

[22] K.-F. Lee, H.-W. Hon, and R. Reddy, "An overview of the sphinx speech recognition system," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 1, pp. 35–45, 1990.

[23] N. Pelekis, I. Kopanakis, C. Panagiotakis, and Y. Theodoridis, "Unsupervised trajectory sampling," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 17–33, Springer, 2010.

[24] S. Vassilvitskii and D. Arthur, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027–1035, 2006.

[25] M. K. Pakhira, "A linear time-complexity k-means algorithm using cluster shifting," in *2014 International Conference on Computational Intelligence and Communication Networks*, pp. 1047–1051, IEEE, 2014.

[26] A. Grønlund, K. G. Larsen, A. Mathiasen, J. S. Nielsen, S. Schneider, and M. Song, "Fast exact k-means, k-medians and bregman divergence clustering in 1d," *arXiv preprint arXiv:1701.07204*, 2017.

[27] X. Wu, "Optimal quantization by matrix searching," *Journal of algorithms*, vol. 12, no. 4, pp. 663–673, 1991.

[28] T. Mizrahi and Y. Moses, "Time4: Time for sdn," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 433–446, 2016.

[29] S. Badotra and J. Singh, "Open daylight as a controller for software defined networking.," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 5, 2017.

[30] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 1–6, 2014.

[31] 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*, vol. 38, no. 2, pp. 69–74, 2008.

[32] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, *et al.*, "The design and implementation of open vswitch," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pp. 117–130, 2015.

[33] U. Lamping and E. Warnicke, "Wireshark user's guide," *Interface*, vol. 4, no. 6, p. 1, 2004.

[34] D. Mills, *RFC1305: Network Time Protocol (Version 3) Specification, Implementation*. RFC Editor, 1992.

[35] K. Correll, N. Barendt, and M. Branicky, "Design considerations for software only implementations of the ieee 1588 precision time protocol," in *Conference on IEEE*, vol. 1588, pp. 11–15, Citeseer, 2005.

[36] T. Mizrahi and Y. Moses, "Reverseptp: A software defined networking approach to clock synchronization," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 203–204, 2014.