Optimizing Forensic Data Availability and Retention of SDN Forensic Logs by Using Bloom Filter

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computer Science and Engineering by Research

by

Varun Sharma
201150846
varun.sharma@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
April 2023

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Optimizing Forensic Data Availability and Retention of SDN Forensic Logs by Using Bloom Filter" by Varun Sharma, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Prof. Shatrunjay Rawat

To my parents and brother

Acknowledgments

I would like to express my deepest gratitude to my mentor, my guru, Prof. Shatrunjay Rawat. This project would have been impossible without his persistent support and guidance. His extraordinary attention to details and personal generosity have meant so much to me than is possible to express. He is a person of great thoughts and discipline and his meticulous manner of working is something to emulate. I shall forever remain indebted to him for his valuable insights and expertise. Apart from our several meetings and discussions, he has been kind enough to help me sail through the challenging times of pandemic.

My sincere thanks to the management of IIIT Hyderabad, the premier institute which is serving the society as a host of excellent knowledge source. I have always been an admirer of this wonderful institution which is dedicated towards meaningful research and academia.

I also wish to thank the almighty for providing me sound physical and mental health. Thank you, Saurabh for helping me through the technical discussions, resolving my queries and providing his valuable feedback. My sincere thanks to Kishore for his cheerful support which helped me to smoothly pursue my studies alongside my office work.

For my baa, my parents, Shri Ajay Shanker Sharma and Shrimati Hemlata Sharma, and my brother, Aditya Sharma. They helped me survive all the stress and not letting me give up during the adversities. I was able to wean off my bouts of procrastination only with their gentle persuasion and motivation throughout this journey. Thank you so much for always being my biggest support system.

Abstract

To perform network forensic analysis on an incident in software defined networking (SDN), logs are of the utmost importance. Without any logs, an investigator will not be able to complete and justify the forensic analysis. With the advancement of network and communication technologies, the volume of logs that needs to be collected and retained is growing exponentially big. The network administrators face the problems in maintaining such huge logs for longer period of time for forensic analysis. Network Providers rely on purging data based on a certain stipulated duration expected by the local rules for forensics evidence. SDN providers have limitations in managing such big data since the expense involved is commensurate with the duration of data retention. Here, in this work we propose a novel idea to reduce and summarize large forensic data sets for faster querying as well as reducing its space complexity by using bloom filters. Through this work we aim to propose a system which can deliver more optimized forensic data availability in SDN platform compared to existing systems.

Contents

List of Figures

List of Tables

Abbreviations

Following abbreviations are used in the thesis, including diagrams ::

IT – Information Technology

SDN – Software Defined Networking

IoT – Internet of Things

IP – Internet Protocol

ICMP – Internet Control Message Protocol

IDS – Intrusion Detection System

IPS – Intrusion Prevention System

BF – Bloom Filter

FPP – False Positive Probability

WLAN – Wireless Local Area Network

SAN – Storage Area Network

WAN – Wide Area Network

1 Introduction

1.1 Motivation

Network Forensics is a stream of forensic science which deals with forensic activity in the area of computer networks which mainly involves identifying sources and paths of cyber-attacks. This is the most important part of any cybercrime investigation and forensics. It primarily involves detecting all sources of attacks in the network, it can also be used as a proactive method to monitor and identify issues in the network infrastructure, overall performance, and bandwidth usage.

Security concerns in the network might start as something simple as traffic spikes or bottlenecks. These are, however, often ignored, especially in developing organizations, assumed to be caused by application growth or increase in the number of users. But neglecting these issues can result in data breaches, loss of customer data, device crashing, etc. With the exponential increase in network traffic every minute. There has been a proportional increase in the number of cyber-attacks or malfunctions across the globe. To deal with such attacks and malfunctions, we need to capture and store the data for forensic data analysis.

We have a wide variety of network systems like WLAN, SAN, WAN, etc. of which SDN is the upcoming future of network system. SDN is gaining momentum in the networking industry due to its multiple advantages such as centralized, flexible, and programmable network management. The SDN framework consists of diverse devices like IoT devices, data centers and internet service providers spread across vast geographical areas. Each of these devices produce logs, which need to be analyzed and monitored to provide effective solutions for network security and comply with the regulations.

Current information systems are crowded with log files, created in multiple places (e.g., network servers, user monitoring applications, system services and utilities) for multiple purposes like maintenance, security issues, traffic analysis, legal data, and forensic analysis. Log files in such complex systems can grow quickly to humongous proportions causing difficulty to retain, monitor, control, and secure them for forensic analysis. Often they are required for prolonged periods of time.

Due to the overhead in managing archived data and expenses, data older than a certain retention period is purged in current systems. While analyzing the critical forensic incident to trace back and identify the potential source of attack, if the forensic data required is slightly older than the certain retention period policy and is purged already, we might completely miss out on gathering potential evidences for analyzing the source of the cyber attack path. This area is crucially based on the exigency of the threat forensic experts deal with, and attracts our attention to provide a solution to this problem using the bloom filters.

The important factors that need to be considered in terms of forensic data collection in SDN platform necessary for forensic analysis are system security, information security, confidentiality, and expense. System security deals with hardening of systems which should be tolerant of any security breach. Information security deals with ensuring the safety of collected network data while, confidentiality deals with maintenance of sensitive or private information in the collected network data. In these scenarios, enormous quantities of information need to be gathered for analytical or archive purposes. This requires the streamlining of the quantity of data to minimize data transfer bandwidth utilization, maximize search throughput and reduce storage space requirements.

The expenses involved in network data storage is an important factor that should be considered in data availability in SDN platform for network forensics analysis. SDN providers cannot keep collecting all network data forever and since the expense is commensurate with the duration of data retention, a system should be well designed by paying attention to the overall expenses involved.

Most of the local government policies on network log retention are based on the feasibility of the ISP/network providers' capacity to hold logs for certain period of time. This capacity is based on multiple challenges involved in the process like network log data management, data security, hardware requirements, expenses incurred. With our system, duration to hold the forensic network logs can be enhanced to more than double the current capacity using almost similar hardware. This positively opens the scope for the local governments to consider this as an opportunity and may revise such retention policies in crucial systems which currently varies from 1 month to 3 months in moderately busy network systems. We discuss more on the challenges involved in the data retention in next chapters.

The previous research works conducted under the guidance of our esteemed Professor, which employed BF as a key methodology, have served as a significant motivating factor for us to extend the scope of our own research using BF, and after undergoing a series of iterations to identify the problems and derive their corresponding solutions, we have arrived at the decision to venture deeper into this area and explore its potential to a greater extent.

1.2 Research Contribution

Below are the Research Contribution:

- In this work, we introduce a framework which can hold additional SDN forensics data compared to traditional network data storage systems.
- We employ a hashed data approach provided by bloom filters.
- Network data logs integrated in bloom filter provide more forensics data storage which can be used as potential evidence to track path traversed by a packet for a longer duration of time compared to existing SDN systems.
- We simulate our framework on the network data using '*Mininet*' and evaluate the results.

1.3 Thesis Structure

This thesis is organized as follows: '**Chapter** 2' provides the background of Network Forensics, its categories, SDN forensics data, complexity to deal with forensic evidences, SDN logs sources, data collection, compression strategies, IT and system security, forensic data availability and acceptability, data retention policy and expenses concerns involved with data storage and management along with the fundamentals of data availability as a crucial parameter for cyber-crime evidences.

'**Chapter 3**' shows the related work done in solving some of the topics listed in '**Chapter 2**' like data security approaches. Next, we discuss effective data collection in fast growing SDN system. And the usage of blockchain to achieve data protection and how the system is designed for logs availability.

‘**Chapter 4**’ deals with the proposed system architecture along with the experiment which contain details of system like logs collection, integrating bloom filter and space & time complexities evaluation.

‘**Chapter 5**’ discusses the evaluations of experiment simulations, results and analysis.

‘**Chapter 6**’ deals with the Conclusion and future work which contains conclusion drawn from this thesis and the expected work in future.

2 Background

2.1 Network Forensics

Network Forensics is a branch of digital forensic science which is related to monitoring and analysis of computer network traffic. This is required to perform future analysis and to detect vulnerabilities exploited. Network forensics consist of steps like capturing, recording and analyzing network data for determining attack source. This information about the source of cyberattack is crucial and can be presented in the court of law as evidence.

We need to perform forensic analysis on the data flowing through the network with respect to occurrence of the attack. This activity could involve extracting data from saved packets data. While doing so, we are risking the privacy of internet users as a lot of confidential information flows through the network. So, there is a need to preserve data in a way that is precise and at the same time does not reveal the information of its contents.

[18] Network analysis is one of the good methods to detect the cybercrime that had taken place. Network forensics can also be used to secure and protect a system by finding about the uncommon or malicious activities in the network [14]. Unlike every other domains of digital forensics, this branch deals with the unstable and volatile information and thus it is a proactive investigation. In addition, various case studies have been added to make the concept easier and highlight the benefit of network forensics. Network Forensics can be defined as the procedure to capture, record, filter, analyse and extract network packets for finding out about the source of attack that has occurred in or against a particular network security framework. Network Forensics can also be used to detect the pattern and target of a particular attack [15]. It gathers information from various locales and distinctive network hardware, for example, Firewalls and IDS to dissect the network traffic information. Moreover, network forensics can be utilized to screen, forestall, and dissect likely assaults.

Network forensics can be used to collect various data present in network like packets, firewalls, log files, addresses etc. In recent years, there is a very huge change in the occurrence and types of Cyber-attack against or between network securities. So, network forensics is highly important now. The most important work of network administrators today is to analyse and observe traffic in the network by analyzing each and every packet transmitting in a network, if they are a danger to a secure system and what would be the instant response in case of breach in security.

There are various limiting factors that could be huge challenge for investigators [16]. Network analysis is sometimes challenging as there are many limiting factors such as access to IP addresses, data integration and privacy, storage in network device, lack of advance tools, management of high-speed data transmission. A person who works with a network can identify if there has been an attack, type of attack, damage it can cause and how to fix it as soon as possible. However, growth and advancement of network forensics is one of the reliable methodologies in the field of cyber-crime.

A general Network analysis used in forensic investigation includes steps such as:

- ***Identification*** where the investigator recognizes the network indicators.
- ***Preservation*** is done in the next step where the isolation and securing of physical or logical evidence are done to prevent it from altering.
- ***Collection:*** Investigating officers record the physical scene and duplicate evidence in case of the primary being corrupted.
- ***Examination:*** Search of the evidence relating to the attack in/on the network and includes detailed documentation also for further analysis. in-depth systematic search of evidence relating to the network attack.
- ***Analysis:*** Determine significance and reconstruct packets present in the network traffic data and lead to conclusions based on evidence analyzed.
- ***Presentation:*** Summarize and concludes results based on the evidence.
- ***Incident Response:*** The response to the intrusion detected is initiated based on the information gathered by the evidence [17].

Digital evidence that is presented in a court of law has to be well-organized, proving that the fact is un-tampered and data privacy has not been compromised.

Most of the earlier work to traceback the attacker was done majorly using '*IP Traceback*' methods. [2] Other methods may involve logging of entire network data and later data-mining techniques are applied to determine if the attack packet has passed through the router/switch. Few of the current network forensics techniques being used to traceback the attack source are as follows.

- ***IP Traceback:*** It is based on the volatile routing tables to traceback the attack path. Routing table is referred to back track the packet from the victim host to attackers' network. But in real time, this technique has its own limitations and cannot be used except for a very short time traceback because an attacker can potentially manipulate the path/ packet by introducing a rouge router in a weak network and the routing table also may change because of change in the network condition (topology, traffic, etc.)
- ***Input Debugging:*** In this method, when a victim recognizes that it is being attacked, the victim develops an attack signature that describes a common feature contained in all the attack packets. The victim communicates this attack signature to the upstream router that sends the packets. Based on this signature, the upstream router employs filters that prevent the attack packets from being forwarded through an egress port and determines which ingress port they arrived at. The process is then repeated recursively on the upstream routers, until the originating site is reached. This method can only be used in real-time when an attack is in progress with the Internet Service Provider (ISP) coordination.
- ***Controlled Flooding:*** The victim uses a pre-generated map of the Internet topology to iteratively select hosts that could be forced to flood each of the incoming links of the upstream router. Router has a common buffer that is shared by packets coming across all the incoming links. So, the attack packets have a higher probability of being dropped due to this flooding. By observing changes in the rate of packets received from the attacker, the victim infers the link through which the attack packet would have come. This basic testing procedure is then recursively applied on all the upstream routers until the source is reached. Here the approach itself will to a denial-of-service attack on the routers.

- ***Logging:*** In this approach, all the packets are logged at each router and data mining is used to determine if the packet has passed though the router. This will lead to privacy concerns because packets are logged at each router. Also it is not practically feasible because of the limited storage available at each router.
- ***ICMP Traceback:*** Every router will sample (or choose few of them) the packets it is forwarding, with a low probability (like 0.00005). Content of sampled packet is copied into an ICMP traceback message along with the information about adjacent routers and the message will be sent to the destination (or host). This information will be used by host to determine the attack path. This approach would be more useful for the flooding type of attack, so that the receiver gets enough messages about the routers in the path to reconstruct the attack path. This technique is more likely to be applicable for attacks that originate from a few sources and would involve overhead of ICMP messages sent to destination hosts.
- ***Basic Packet Marking:*** In this technique, each router marks its IP information into the packet before forwarding the packet, so that traceback can be completed by the victim. Since Packet Record Route can hold Maximum of 10 IP address this will enable to capture only few routers traversed in the path.
- ***Probabilistic Packet Marking:*** Upon receiving a packet, a router chooses to mark its own address on the packet with a probability p. Given that enough packets would be sent by the attacker and the route remains stable; the victim would receive at least one sample for every router to construct attack path.

We observed that the network data is of utmost importance for forensic analysis. This brings the known challenges associated with retaining this big data for a longer duration of time. Forensic analysis system requires optimum space and time complexity design to deal with such a huge amount of data and the processing of it. Below we briefly discuss challenges involved in retaining the network traffic logs for forensic analysis.

2.2 Categories of Network Forensics

There are majorly two types of network forensics, Investigation Mode and Data Processing Mode. Based on investigation mode, network forensics are further classified as, online and offline network forensics. This type of investigation depends on the time of the investigation.

- **Online / Live Network Forensics:**

  This type of network forensics is also known as dynamic forensics, where the investigation is conducted during its flow. Forensics of this type is generally suitable for large, distributed networks. Implementation of this strategy requires more computational resources and huge amount of storage.

- **Offline Network Forensics:**

  This type of network forensics is also known as static forensics. Static forensics is a type of network forensics in which network data is captured, recorded, and analyzed after an attack has occurred. As a result of storage space limitations, existing data might be overwritten even though each occurrence is correctly recorded from network logs and intruders' behavior is briefly and accurately monitored.

And majorly there are two categories of network forensics, based on the data processing mode. They are defined as proactive and reactive network forensics.

- **Proactive Network Forensics:**

  This is used in real time investigation of the incident. This is implemented by automating the devices while reducing user interaction. It provides more accurate and precise data. It also offers early detection of network attacks and reduces the possibility for intruders to delete the evidence after attack. However, in terms of attack patterns detection, it elevates processing overhead and storage.

- **Reactive Network Forensics:**

  This is a postmortem method which deals with investigating an attack after it has occurred. To ferret out the root cause of the attack, investigate the malicious incident with reduced processing, correlate the attacker to the attack, and mitigate the impact of the attack, it sifts the network vulnerabilities by storing, detecting, gathering and analyzing digital evidence collected from the network.

2.3 SDN Forensics Data

It is the data generated from SDN network system which can be used for network forensic analysis and can be produced in the court of law or investigating agencies as potential evidence in case of cybercrime. Currently with minimal number of contributions forensics is still at an early stage in SDN. As SDN allows the safe preservation of network activity traces, it provides support for digital forensics and root cause analysis. SDN controllers provides centralized control over the network and data storage which are generated from all data plane devices.

According to IT Act Rules, there are certain evidence acceptance criterion in the court of law, based on which it is considered as valid evidence, like if the raw data is hashed then both the data (raw and hashed) should yield the identical observations. One of the acceptable electronic types of evidence rule states:

"To derive or reconstruct the original electronic record from the hash result produced by the algorithm". We are focusing more on this criteria where the hash data and raw data should reconstruct the same forensic results upon analysis, as shown in

Proposed system with admissible evidence in court of law system flow is shown in 'Figure 3'.

2.3.1 Logs Data Sources

SDN logs are gathered from different network devices like network switches and routers generated in centralized network controller. The SDN controller is the 'brain' of the network. It is possible and very common these days to deploy multiple controllers in the network, each responsible for different tasks. In fact, to make it more complicated, some network operators decide to deploy multiple controllers, managed by a single master controller, giving us the challenge of running and managing many ‘brains’ in our network. Before we even tackle the orchestration and management issues, we need to discover if there are any issues in our network caused by either controller performance bottlenecks or networking equipment failure.

The control plane is composed of an SDN controller or a set of controllers. It is possible to understand the controller’s behavior and performance by looking at the logs produced by it. The SDN controller will contain most of the important information about the state of the network that can be used later in troubleshooting or debugging.

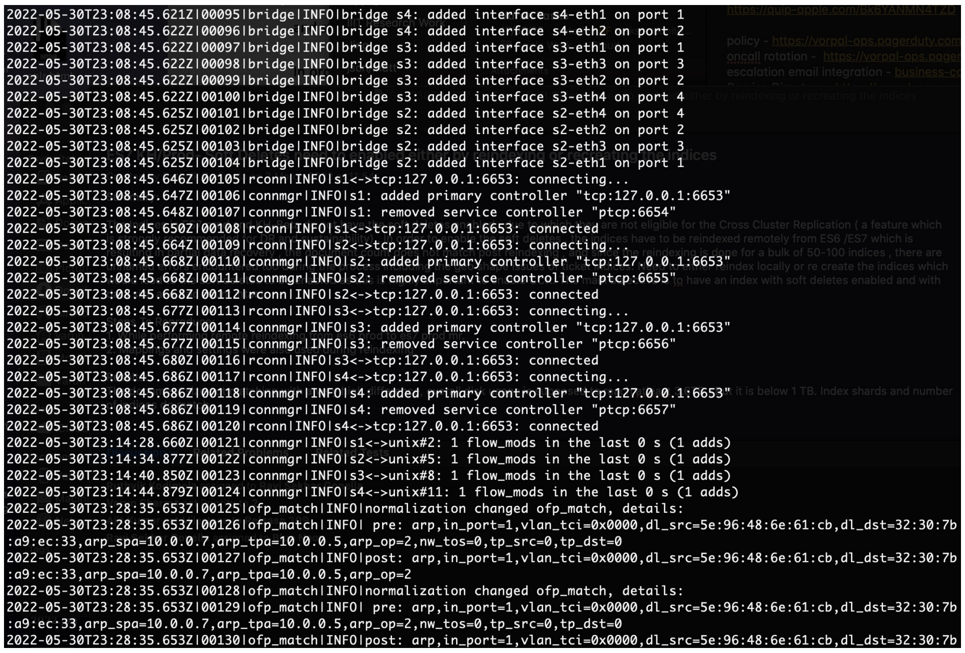
OvS is a production quality virtual switch, licensed under the open source Apache 2.0 license. It is designed to enable large scale network automation, while supporting standard management interfaces and protocols. By default, the ‘*OvS*’ sends log data to files in the server’s *“/var/log/openvswitch”* directory.

ovs-openvswitchd.log – ‘*switchd*’ is a daemon that manages and controls any number of ‘*Open vSwitch*’ devices on the local machine. This log holds information about the data paths set up, configuration updates, switch to controller connection status and per port information. Link failures are also logged to *openvswitchd.log* as part of ‘*Open vSwitch*’ link monitoring. These logs are collected in a centralized system and retained for offline forensic analysis. Sample logs are shown in ‘Figure 1’

2.3.2 Data Collection

Software Defined Networking (SDN) [1] is one of the most promising options for network management and the future of next-generation networks (Future Networks). SDN possesses an intelligent configuration, better flexibility to accommodate innovative networks, and high-performance architecture. The SDN mainly consists of 3 main components:

- **The Data Plane / Infrastructure layer** is the network architecture layer that physically handles the traffic based on the configurations supplied from the Control Plane.
- **The Control Plane / layer** refers to the network architecture component that defines the traffic routing and network topology.



```
2022-05-30T23:08:45.621Z|00095|bridge|INFO|bridge s4: added interface s4-eth1 on port 1
2022-05-30T23:08:45.622Z|00096|bridge|INFO|bridge s4: added interface s4-eth2 on port 2
2022-05-30T23:08:45.622Z|00097|bridge|INFO|bridge s3: added interface s3-eth1 on port 1
2022-05-30T23:08:45.622Z|00098|bridge|INFO|bridge s3: added interface s3-eth3 on port 3
2022-05-30T23:08:45.622Z|00099|bridge|INFO|bridge s3: added interface s3-eth2 on port 2
2022-05-30T23:08:45.622Z|00100|bridge|INFO|bridge s3: added interface s3-eth4 on port 4
2022-05-30T23:08:45.625Z|00101|bridge|INFO|bridge s2: added interface s2-eth4 on port 4
2022-05-30T23:08:45.625Z|00102|bridge|INFO|bridge s2: added interface s2-eth2 on port 2
2022-05-30T23:08:45.625Z|00103|bridge|INFO|bridge s2: added interface s2-eth3 on port 3
2022-05-30T23:08:45.625Z|00104|bridge|INFO|bridge s2: added interface s2-eth1 on port 1
2022-05-30T23:08:45.646Z|00105|rconn|INFO|s1<->tcp:127.0.0.1:6653: connecting...
2022-05-30T23:08:45.647Z|00106|connmgr|INFO|s1: added primary controller "tcp:127.0.0.1:6653"
2022-05-30T23:08:45.648Z|00107|connmgr|INFO|s1: removed service controller "ptcp:6654"
2022-05-30T23:08:45.650Z|00108|rconn|INFO|s1<->tcp:127.0.0.1:6653: connected
2022-05-30T23:08:45.664Z|00109|rconn|INFO|s2<->tcp:127.0.0.1:6653: connecting...
2022-05-30T23:08:45.668Z|00110|connmgr|INFO|s2: added primary controller "tcp:127.0.0.1:6653"
2022-05-30T23:08:45.668Z|00111|connmgr|INFO|s2: removed service controller "ptcp:6655"
2022-05-30T23:08:45.668Z|00112|rconn|INFO|s2<->tcp:127.0.0.1:6653: connected
2022-05-30T23:08:45.677Z|00113|rconn|INFO|s3<->tcp:127.0.0.1:6653: connecting...
2022-05-30T23:08:45.677Z|00114|connmgr|INFO|s3: added primary controller "tcp:127.0.0.1:6653"
2022-05-30T23:08:45.677Z|00115|connmgr|INFO|s3: removed service controller "ptcp:6656"
2022-05-30T23:08:45.680Z|00116|rconn|INFO|s3<->tcp:127.0.0.1:6653: connected
2022-05-30T23:08:45.686Z|00117|rconn|INFO|s4<->tcp:127.0.0.1:6653: connecting...
2022-05-30T23:08:45.686Z|00118|connmgr|INFO|s4: added primary controller "tcp:127.0.0.1:6653"
2022-05-30T23:08:45.686Z|00119|connmgr|INFO|s4: removed service controller "ptcp:6657"
2022-05-30T23:08:45.686Z|00120|rconn|INFO|s4<->tcp:127.0.0.1:6653: connected
2022-05-30T23:14:28.660Z|00121|connmgr|INFO|s1<->unix#2: 1 flow_mods in the last 0 s (1 adds)
2022-05-30T23:14:34.877Z|00122|connmgr|INFO|s2<->unix#5: 1 flow_mods in the last 0 s (1 adds)
2022-05-30T23:14:40.850Z|00123|connmgr|INFO|s3<->unix#8: 1 flow_mods in the last 0 s (1 adds)
2022-05-30T23:14:44.879Z|00124|connmgr|INFO|s4<->unix#11: 1 flow_mods in the last 0 s (1 adds)
2022-05-30T23:28:35.653Z|00125|ofp_match|INFO|normalization changed ofp_match, details:
2022-05-30T23:28:35.653Z|00126|ofp_match|INFO| pre: arp,in_port=1,vlan_tci=0x0000,dl_src=5e:96:48:6e:61:cb,dl_dst=32:30:7b
:a9:ec:33,arp_spa=10.0.0.7,arp_tpa=10.0.0.5,arp_op=2,nw_tos=0,tp_src=0,tp_dst=0
2022-05-30T23:28:35.653Z|00127|ofp_match|INFO|post: arp,in_port=1,vlan_tci=0x0000,dl_src=5e:96:48:6e:61:cb,dl_dst=32:30:7b
:a9:ec:33,arp_spa=10.0.0.7,arp_tpa=10.0.0.5,arp_op=2
2022-05-30T23:28:35.653Z|00128|ofp_match|INFO|normalization changed ofp_match, details:
2022-05-30T23:28:35.653Z|00129|ofp_match|INFO| pre: arp,in_port=1,vlan_tci=0x0000,dl_src=5e:96:48:6e:61:cb,dl_dst=32:30:7b
:a9:ec:33,arp_spa=10.0.0.7,arp_tpa=10.0.0.5,arp_op=2,nw_tos=0,tp_src=0,tp_dst=0
2022-05-30T23:28:35.653Z|00130|ofp_match|INFO|post: arp,in_port=1,vlan_tci=0x0000,dl_src=5e:96:48:6e:61:cb,dl_dst=32:30:7b
```

Figure 1: Sample Open vSwitch Logs

- **The Management Plane / Application layer** takes care of the wider network configuration, monitoring and management processes across all layers of the network stack.

'Figure 2' shows the flow of a network in a typical SDN system and forensics logs collection. Like most software systems, the SDN controller is typically equipped with logging features that can be used at run time to provide insight into the behavior of the software and the hardware that it is running on. The amount of logging depends on developer style, but typically it will log the occurrence of all major SDN functions such as flow table modification, network component connection status and topology characteristics. The SDN controller log files contain valuable raw information about the network events processed by the software modules.

Ideally, the collected data should be processed for analysis in real-time and in a non-invasive style, having negligible impact on the overall system performance. One of the solutions is to use the log analysis tools residing in the cloud to analyze logs from SDN controllers and switches. It requires extensive knowledge of logging, log analysis, network behavior and SDN domain-specific knowledge.

Current data storage system has data in text format / raw logs, compressed logs or archived and the data is retained for a certain stipulated period of time. With the acceleration of the 5G technologies and the promotion of the Internet of things (IoT) services, large scale and high-speed networks become the focus of current research and development. In order to collect and analyze network logs effectively, researchers and operators proposed many systems and applications as discussed in '**Chapter** 3'.

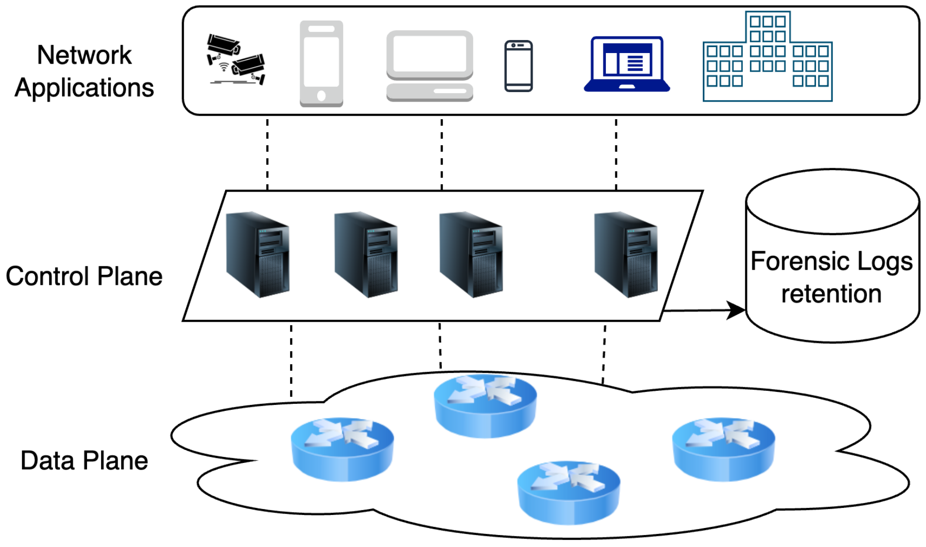


Figure 2: SDN Architecture with Logs Flow

2.3.3 Compression

Current systems store the data using the typical compression algorithms. All real time logs collection is done in the raw format and used for real time analysis in preventive threat detection. Old logs are retained in compressed format like zip, gzip, bz2, tar(+gz) algorithms, which can be restored to raw format for forensics analysis when needed.

2.3.4 SDN System Security

IT and system/network security are important aspects of securing data from vulnerabilities. Referring [1], To identify the hidden details of attacks and their causes, network forensics is a preferred technique for network security. Network forensic is a process of the identification, compilation, storage, examination, and reporting of network digital evidence. The technique adopted by various network administrators to investigate the source of attacks is called network forensics. A forensic investigation must be able to detect attack attempts regardless of whether they are successful, in order to ensure that networks (servers, end-hosts, and other related devices) are secure. Additionally, simple anomalous patterns must also be recognized for a solution to detect as many ways of attack and malicious activity as possible. In general, SDN forensic solutions provide a reasonable solution to network and cloud security.

The utilization of SDN in network forensics is worth studying for efficient networks. Majorly, there are two approaches defined for SDN security which are Content Inspection and Traffic Monitoring & Auditing

- **Content Inspection:** To inspect the contents of each packet of data on a network is known as content inspection. Using IDS, content inspection can be enhanced through flow level security and deep packet inspection. As SDN enables flow-level security for the network security systems, the flow of data is analyzed during the content inspection, and selected packets are then used for the content inspection. The role of an (IDS/IPS) is to stop or allow packets based on a thorough packet survey using pattern recognition, data mining or signature matching with an established threat inventory. In real-time, the SDN IDS can use a huge amount of flow-based knowledge.
- **Traffic Monitoring and Auditing:** Besides other fundamental network management tasks of SDN, network traffic monitoring promotes anomaly detection, network forensic analysis, and user application identification. Monitoring and auditing are very important instruments for certain security tests when we talk about forensics.

Besides the above-mentioned security approaches for SDN, there are other security approaches which are also used in various cases, which include

Flow Sampling, Access Control, Network Resilience, Security of Middle-Boxes, and Security-Defined Networking.

2.3.5 Forensic Data Availability

Data availability is a major concern in any forensic analysis system. All forensic analysis approaches become meaningful only when we have the data. Due to certain challenges in holding the data, it has become a costly affair as it requires a lot of hardware resources and operational working hours to manage the data. In SDN system, since it is centrally managed and controlled, we need to have a strong system to hold the data and then utilize it in forensic analysis.

2.3.6 Admissible Network Forensics Evidence

'*Network forensics [By Yong Guan, in Computer and Information Security Handbook (Third Edition), 2013]*' can be generally defined as a science of discovering and retrieving evidential information in a networked environment about a crime in such a way as to make it admissible in court. Different from intrusion detection, all the techniques used for the purpose of network forensics should satisfy both legal and technical requirements. For example, it is important to guarantee whether the developed network forensics solutions are practical and fast enough to be used in high-speed networks with heterogeneous network architecture and devices. More importantly, they need to satisfy general forensics principles such as the rules of evidence and the criteria for admissibility of novel scientific evidence. The five rules of evidence are as follows:

- **Admissible.** Must be able to be used in court or elsewhere.
- **Authentic.** Evidence relates to incident in a relevant way.
- **Complete.** No tunnel vision, exculpatory evidence for alternative suspects.
- **Reliable.** No question about authenticity and veracity.
- **Believable.** Clear, easy to understand, and believable by a jury.

The evidence and the investigative network forensics techniques should satisfy the criteria for admissibility of novel scientific evidence (Daubert v. Merrell):

- Whether the theory or technique has been reliably tested.
- Whether the theory or technique has been subject to peer review and publication.
- What is the known or potential rate of error of the method used?
- Whether the theory or method has been generally accepted by the scientific community.

The investigation of a cyber-crime often involves cases related to homeland security, corporate espionage, child pornography, traditional crime assisted by computer and network technology, employee monitoring, or medical records, where privacy plays an important role.

There are at least three distinct communities within digital forensics: law enforcement, military, and business & industry, each of which has its own objectives and priorities. For example, prosecution is the primary objective of the law enforcement agencies and their practitioners and is often done after the act. In military operations, primary objective is to guarantee the continuity of services, which often have strict real-time requirements. Business and industry's primary objectives vary significantly, many of which want to guarantee the availability of services and put prosecution as a secondary objective.

Usually there are three types of people who use digital evidence from network forensics investigations: police investigators, public investigators, and private investigators. The following are some examples:

- **Criminal prosecutors.** Incriminating documents related to homicide, financial fraud, drug-related records.
- **Insurance companies.** Records of bill, cost, services to prove fraud in medical bills and accidents.
- **xLaw enforcement officials.** Require assistance in search warrant preparation and in handling seized computer equipment.

- **Individuals.** To support a possible claim of wrongful termination, sexual harassment, or age discrimination.

The primary activities of network forensics are investigative in nature. The investigative process encompasses Identification, Preservation, Collection, Examination, Analysis, Presentation and last is Decision.

In 'Figure 3', the proposed system flow chart illustrates the admissible evidence that can be presented in a court of law for forensic analysis.

2.3.7 Forensic Data Retention Policy

All organizations, institutions, Internet Service Providers, etc. abide by a certain network log retention policy based on the local government guidelines. These retention policies are made based upon the challenges involved in holding the big data for forensic analysis. From the network provider's perspective, this stream is a complete investment, so we need to deal with the resources and budget limitations accordingly.

For an example, *In India*, Section 69 of the IT Act allows the interception, monitoring and decryption of information for a limited period of 2 months. Even in the US, there is no mandatory data retention law. Under the '*Electronics Communications Transactional Records Act*', the ISPs in the US are required to retain data for a maximum period of 90 days, and only upon the request of a government entity. As a result, there are no safeguards whatsoever for mass surveillance; there is no time limit on the period for which the surveillance can continue, no restriction on the type of data that can be collected and who it can be collected from.

2.3.8 Expense

The expense of network data collection [3] is also an important factor that should be considered in terms of data collection performance. Investigators and operators strive to develop an effective and low cost system for data collection. In a large-scale data collection process, each extra component in an individual thread could cause big redundancy. Typically, the expense of data collection consists of development expenses and maintenance cost. So, we need a data collection mechanism that is well designed by paying attention to the costs of development and maintenance. Deployment difficulty and application of expensive equipment in data collection can shoot up the expenses. A Trade-off should be designed and developed in order to solve this problem. Maintenance Expense (ME) heavily depends on update cycle and error probability. For example, to achieve high accuracy, we often use pattern matching method based on expert's knowledge where data is frequently updated and revised, which causes high maintenance expenses. However, the use of a precise and adaptive mechanism for data collection may reduce maintenance expenses due to its small maintenance requirement.

We have considered this as an important factor in dealing with forensic data storage and management as it attracts our attention to this area and also motivates our research. We analyzed the existing system end to end and designed a system which can solve the problem of handling data older than 3 months or more.

3 Related Work

3.1 Security Approaches

In [1], authors emphasize on the evolution and continuous adaptability of SDN in the current network systems. Authors have discussed the various advantages in SDN paradigm like ease of management, handling centralized connectivity across the nodes, cloud and network security concerns. This research work is aimed at providing the basics of SDN forensics and its advantages in cloud architecture.

Authors highlight challenges and Issues of SDN in Network Forensics. The centralized control of SDN draws attackers to exploit various network devices by taking illegal control of the controller by hijacking the controller itself. In the development years of SDN, the security initially was not considered as a key characteristic of SDN architecture, but with time and due to the centralized nature of the SDN, they are vulnerable to various attackers. Therefore, the security of SDN is given more priority. In the newly evolving SDN architecture, investigating attacks is a tiring and demanding task. Eventually, SDN seems to be the most intriguing development platform for future networks. SDN still faces many challenges and problems, despite its impressive advantages, particularly when it comes to a network security problem. The goal of SDN network measurement is to understand and quantify different aspects of network activity to promote network management, monitor the anomalies and the development of security mechanisms, and network troubleshooting.

Authors recommend designing the below-mentioned security-related primitives to be considered for a better and efficient network and cloud-based forensic.

- To prevent disruption and protection compromises, SDN security reference models and approaches based on protecting network entities should be introduced.
- Using the control channel, traffic tracking of the application-controller and identification of irregularities in particular avenues, such as cloud setups can be implemented.

- Various methods and tools should be implemented to provide strong security in different forensic process stages.
- Different techniques should be used to provide strong security at different layers of SDN.
- It is possible to store and retrieve network/state data for post-event and forensic analysis for efficiency.
- Developing frameworks for the cloud forensic having ease to detect the attacks.
- Enhance the security, content inspection, traffic monitoring, auditing, and attack detection in cloud forensic.
- Creating enhanced Intrusion detection systems and improve their utilization in SDN.

Advantages:

- Focuses on an emerging technology (SDN) for network forensic analysis.
- Proposes a framework for conducting SDN forensic analysis.
- Provides a case study to evaluate the effectiveness of the framework.

Drawbacks:

- Limited discussion of the limitations and potential drawbacks of using SDN for forensic analysis.
- The proposed framework is not evaluated on a large-scale or in a real-world scenario.

Overall, the authors provide a useful overview of the potential benefits and challenges of using SDN for forensic analysis, as well as a framework for conducting such analysis. However, it could benefit from a more in-depth discussion of the limitations and potential drawbacks of using SDN for forensic analysis. The table above lists various techniques for SDN forensic analysis, along with their advantages and drawbacks, which could be useful for researchers and practitioners in the field.

Table 1: SDN Security Approaches

Technique	**Advantages**	**Drawbacks**
Network Traffic Analysis	Provides insight into network behavior and potential security threats	Requires a large amount of data processing and analysis
Flow-Based Analysis	Allows for the identification and analysis of specific network flows	Limited visibility into overall network behavior
Protocol Analysis	Provides insight into the behavior of specific network protocols	May not detect new or unknown protocols
Signature-Based Analysis	Can detect known network attacks and threats	May not detect new or unknown threats
Anomaly-Based Analysis	Can detect unknown or novel threats	May generate a high number of false positives

3.2 Effective Network Data Collection

In [3], The authors provide a comprehensive survey of the various techniques and tools used for network data collection. They discuss the advantages and drawbacks of each technique and provide an overview of the current state of the field.

As a fundamental procedure of network security measurement, network data collection executes real time network monitoring, supports network performance evaluation, assists network billing, and helps traffic testing and filtering. It plays a crucial and essential role while dealing with network intrusion detection and unwanted traffic control. But an adaptive and effective data collection mechanism that can be pervasively applied into heterogeneous networks is still lacking.

Authors introduce three basic types of network data collection methods , which are packet based data collection, flow based data collection, log based data collection. Based on the types of the data collection authors evaluated the system performance by instantaneity, effectiveness, scalability, expense, network performance, resource occupation, system security, adaptability and evaluated the information security by integrity, confidentiality, availability, traceability and authenticity. Furthermore, they figured out some open issues based on several investigations and did forecast the direction for future research.

In terms of techniques for network data collection, the authors discuss packet sniffing, flow-based data collection, and telemetry-based approaches. They also cover various tools for network data collection, such as Wireshark, tcpdump, ntop, and NetFlow.

One of the main advantages of packet sniffing is that it provides a complete view of all traffic on the network. However, it can be resource-intensive and may not scale well to large networks. Flow-based data collection, on the other hand, provides a more lightweight approach that can scale well but may not capture all traffic.

Telemetry-based approaches, which rely on network devices to provide data, offer a more efficient approach that can capture a wide range of data. However, this approach may require specialized hardware and may not capture all relevant data.

The authors also highlight the importance of considering privacy and security concerns when collecting network data. They discuss various techniques for anonymizing and securing data, as well as potential legal and ethical issues related to network data collection.

Overall, the authors provide a useful overview of the techniques and tools available for network data collection. They highlight the advantages and drawbacks of each approach and emphasize the importance of considering privacy and security concerns when collecting network data.

Below is a table summarizing the advantages and drawbacks of the different techniques discussed in the research work:

Table 2: Effective Network Data Collection

Technique	**Advantages**	**Drawbacks**
Packet Sniffing	Provides complete view of all traffic	Resource-intensive; may not scale well to large networks
Flow-Based Collection	Lightweight approach that can scale well	May not capture all traffic
Telemetry-based	Efficient approach that can capture a wide range of data	May require specialized hardware; may not capture all relevant data

3.3 Usage of Blockchain to Protect Data

In [12], The authors propose a system called SDNLog-Foren that uses blockchain technology to ensure the integrity and tamper resistance of log files used for SDN forensics. They argue that traditional logging mechanisms may be vulnerable to tampering or deletion, compromising the integrity of the log data. Authors emphasize securing the forensic logs and implementing blockchain based approach to improve the security of log management in SDN platform and named this approach SDNLog-Foren. Authors further compare this system with other prevailing systems to prove its worth and advantages over others. This work throws light on cyber security concerns while dealing with data. There are various attacks targeting a range of vulnerabilities on vital elements of this paradigm such as controller, Northbound and Southbound interfaces. In addition to solutions of security enhancement, it is important to build mechanisms for digital forensics in SDN which provide the ability to investigate and evaluate the security of the whole network system.

It provides features of identifying, collecting and analyzing log files and detailed information about network devices and their traffic. However, upon penetrating a machine or device, hackers can edit, even delete log files to remove the evidence about their presence and actions in the system. In this case, securing log files with fine-grained access control in proper storage without any modification plays a crucial role in digital forensics and cybersecurity. This model is also evaluated with different experiments to prove that it can help organizations keep sensitive log data of their network system in a secure way regardless of being compromised at some different components of SDN.

SDNLog-Foren works by storing log data in a blockchain, which is distributed across a network of nodes. The blockchain provides a tamper-resistant record of the log data, allowing for the verification of the integrity of the data at any point in time.

The authors highlight the advantages of using blockchain technology for SDN forensics, including the ability to detect tampering, the immutability of the log data, and the ability to provide a verifiable chain of custody for the log data.

One of the main advantages of SDNLog-Foren is that it provides a tamper-resistant and immutable record of log data, which is essential for forensic investigations. However, the system requires significant computational resources to maintain the blockchain, which may be a challenge for smaller organizations.

The authors also discuss the limitations of SDNLog-Foren, including the potential for delays in updating the blockchain due to the consensus mechanisms used by the blockchain. They also note that the system does not address all potential threats to the integrity of log data, such as attacks on individual loggers or sensors.

Overall, the authors propose an innovative approach to addressing the challenges associated with log data integrity in SDN forensics. They provide a useful overview of the advantages and drawbacks of using blockchain technology for this purpose.

3.4 Logs Availability for Analysis

In [13], authors have put the emphasis on how logs are important while dealing with run-time behavior of the software systems like debugging, system comprehensions and anomaly detection. To manage the unstructured nature and large size of logs, authors propose the solution with their system called LogAssist which assists administrators with log analysis. It significantly reduces the log size and improves the time required to analyze the logs. This research work opens up the channel to think in this direction and design more optimized solutions.

It is a novel approach for assisting practitioners with log analysis, which aims to address all the three above-mentioned challenges. First, it parses the raw logs into abstracted log events (i.e., addressing the challenge related to unstructured logs). Then, it untangles the raw logs into meaningful event sequences (i.e., workflows) using certain grouping IDs commonly available in logs, to address the challenge related to intermixed event sequences. Finally, it leverages n-gram models to identify common event sequences and further uses the identified sequences to compress the logs into a much more concise representation (i.e., addressing the challenge related to the large size of logs). In addition, it allows practitioners to expand and explore the compressed form on demand, providing practitioners the flexibility to access the complete information in the logs.

Their research results demonstrate that it can compress the raw logs into a much more concise representation, while allowing practitioners to access the complete information of logs only when necessary. It significantly simplifies log analysis tasks and improves practitioners' log analysis experiences. Authors document their experiences and lessons learned while developing and adopting their approach in practice, which can provide insights for researchers and practitioners who wish to develop similar tools to assist with log analysis tasks. It can be leveraged as a basis and starting point to further advance interactive log analysis techniques.

The conclusion of the research summarizes the key contributions of the paper and highlights the advantages of the LogAssist algorithm over other state-of-the-art log summarization algorithms. The authors also discuss the limitations of their work and suggest directions for future research.

Overall, authors provide a clear and well-written introduction to log analysis and summarization, proposes a novel algorithm for log summarization, and provides a thorough evaluation of the algorithm using two datasets. The paper could be improved by providing more details on the limitations of the LogAssist algorithm and potential areas for improvement. Here is a table listing various techniques, along with their advantages and drawbacks, used for log analysis and summarization:

Table 3: Logs Availability for Analysis

Technique	**Advantages**	**Drawbacks**
Log parsing	Accurate and detailed	Time-consuming and labor-intensive
Log clustering	Groups similar logs together	Requires prior knowledge of log structure
Log pattern mining	Identifies frequent and important log patterns	Limited to predefined log patterns
Log summarization	Reduces log volume and complexity	May lose important details
Log anomaly detection	Detects abnormal log behavior	Requires labeled data for training

4 Proposed System Architecture

Here we discuss our proposed system architecture in detail. We propose a system that leverages the strategy of integrating the data in BF. Our proposed system offers a great advantage in retaining the forensic logs for a much longer duration. ‘Figure 3 shows the proposed system flow chart.

Bloom filters are simple, space efficient data structures which work on the basis of a probabilistic algorithm to test membership in a huge set of hash functions. Bloom filters by design is a hashed data structure, hence we can rely on the system for reliability and safety of data. The system does not need any customised firewall systems or security systems to secure confidential forensic data.

BF do not store the items themselves hence they use less storage space than the lower theoretical limit required to store the data correctly. BF exhibit an error rate which is inversely proportional to its size. The limited space available for BF may cause high rate of false positives, but they do not have false negatives. The one-sidedness of this error turns out to be our benefit. When the BF reports the item as ‘*Found/Present*’, there is a small chance that it is not telling the truth, but when it reports the item as ‘*Not Found/Not Present*’, we know it’s telling the truth. Most of the time, in the context where the query answer is expected to be Not Present, Bloom filters offer great accuracy along with space-saving benefits.

Alongside being storage efficient, time efficient and secured data structure, it is admissible IT evidence as per the local rules, since it complies with the requirement of the hashed data should produce same results as the raw data, to achieve this we have optimized the data sets and size of BF which is explained in ‘Chapter 4.2.2’.

By analyzing the details as per the experimental values in ‘Table 4’ & ‘Figure 12’ we can say that the proposed system offers promising results. System integrated with Raw Data, zip compressed data and data integrated in BF enhances the availability of forensic logs with the similar hardware resources. It positively concludes that the introduced system can withhold comparatively more data as compared to the existing forensic data storage mechanisms. One method is to store the entire string to compare it with another string. This will be a O(m+n) comparison. Some of very common data structures that adapt to the string data type pretty easily are Tries(multiple

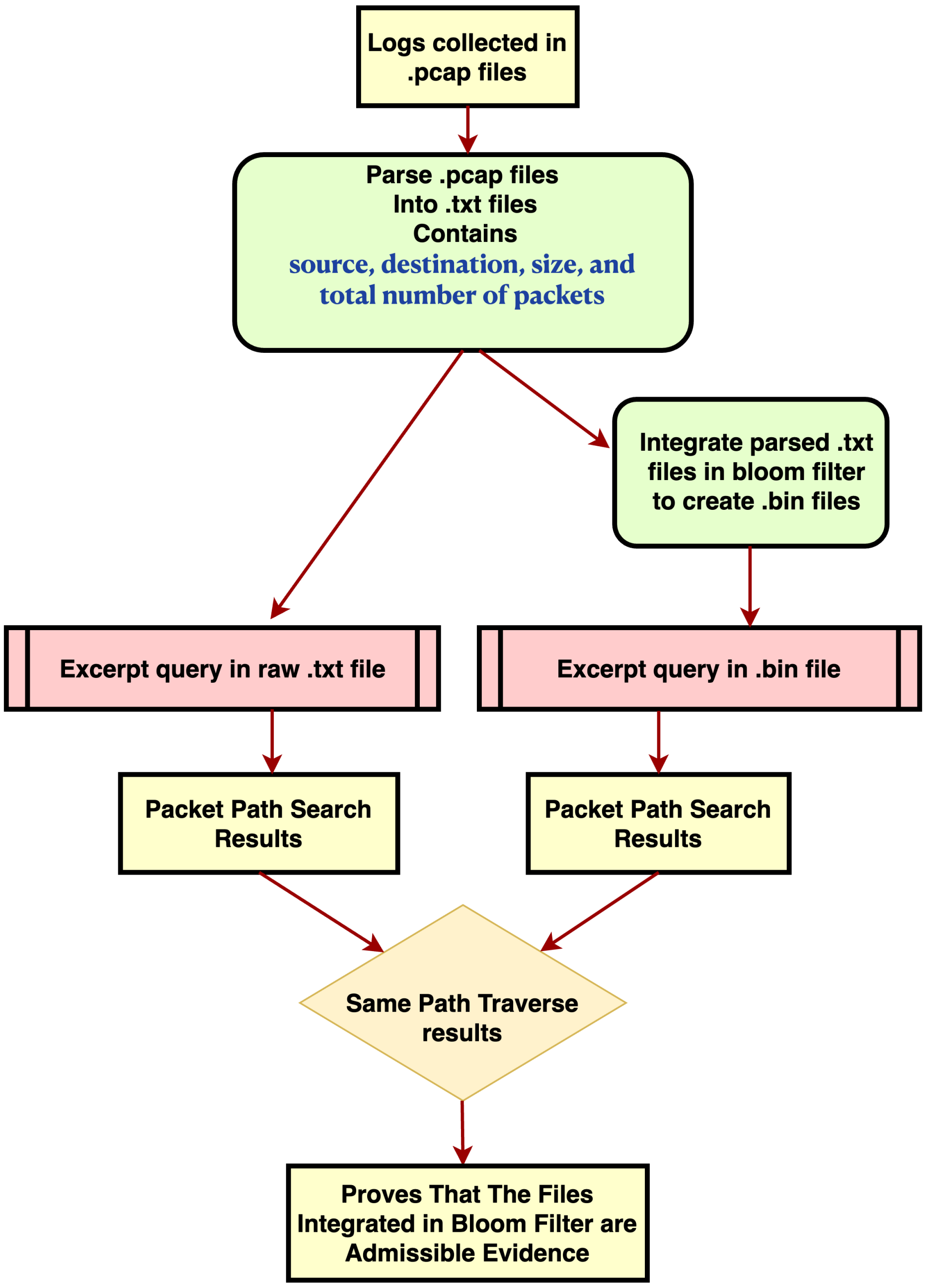

Logs collected in .pcap files
Parse .pcap files Into .txt files Contains source, destination, size, and total number of packets
Integrate parsed .txt files in bloom filter to create .bin files
Excerpt query in raw .txt file
Excerpt query in .bin file
Packet Path Search Results
Packet Path Search Results
Same Path Traverse results
Proves That The Files Integrated in Bloom Filter are Admissible Evidence

Figure 3: Proposed System Flow Chart - Admissible Evidence

try calls), Suffix trees, Binary Trees etc. These data structures have good complexity in theory, but it is not an easy task to optimise them for a given problem per se. The more efficient way is to store the hash of these strings. In practice, Hashing has one of the best complexities with a fantastic O(1) constant search time. We can take a quick look at disk size comparison done in 'Table 4' for the Data Size (GB). We may consider each file size of 200 MB in the directory as shown in 'Figure 6'.

Table 4: Data Storage Analysis

Data Size(GB)	***Zip (GB)***	***Bz2 (GB)***	***Bloom Filter (MB)***
128	1.94	1.33	12.7
512	7.75	5.33	25
1024	15.5	10.67	52

In current systems, network administrators rely on purging the data older than a certain stipulated time expected by governmental guidelines and organizational rules. In addition, considering the retention period of logs for forensic analysis as constant factor, the rate at which the log volume is growing, it becomes a crucial factor in the system design for forensic analysis. In existing systems, the data storage capacities in network devices are very less, which is not ideal to hold historical data for more than a couple of months.

After studying/ examining related research works and open problems, we propose a system which can accommodate more network forensics data in a similar set of resources.

Generally, we choose convenient network packet capturing tools for data collection and further analysis. '*WireShark and TCPdump*' are two classic applications of them. They rely on '*libpcap*' library and '*Berkeley Packet Filter (BPF)*'. Similarly, Network Intrusion Detection Systems (NIDS), such as '*Snort* and '*Bro*' collect data by packet capturing based on '*libpcap and BPF*' for detecting malicious traffic and further processing. '*WireShark and TCPdump*' tools are passive software-based packet capturing mechanisms.

The Proposed system is designed for SDN platform, where a controller is configured in such a way that each node/device in data plane keeps logging in a dedicated directory and keeps rolling these logs. The directories contain the data of each device in pcap (from tcpdump) format.

After the stipulated time in our system, instead of purging the data, we integrate it in BF. As per design, each node (device) generates its log and these log files are parsed and fetched with valuable information like *packet size, packets count, source and destination IPs and hexdump* of each packet considered as elements to be put in BF. This set of elements can be queried with sample packet data to verify if the file is present in any of the nodes'(devices) log directories (bloom filters). This results in either the elements definitely not present in the logs directory (set of elements) of the node(device) or maybe present. We can rely on the definitely not present response. This is accepted as valid evidence in terms of determining the packet traversal path. Though the scope of our proposed system is limited to demonstrate the optimized data availability and retention of SDN forensic logs, we try to touch the base of space and time complexity of data integrated with bloom filter.

With the reduced space and time complexity (refer 'Table 9'), inherited by design of bloom filter which states that a Bloom filter with 1% false positive rate requires only about 9.6 bits per element regardless of element size, the false positive rate can be reduced by a factor of ten each time 4.8 bits per element are added. By observing the less space utilization, its safe to claim that the expense involved in log collection and retention for network forensics logs are optimized by implementing bloom filter in current systems.

From our experimental results we have observed and verified to accommodate atleast 10 times more forensic data i.e., retention of data for longer duration, using the same resources and maintenance expenses as compared to existing systems.

Designed system stages are shown in 'Figure 4'.

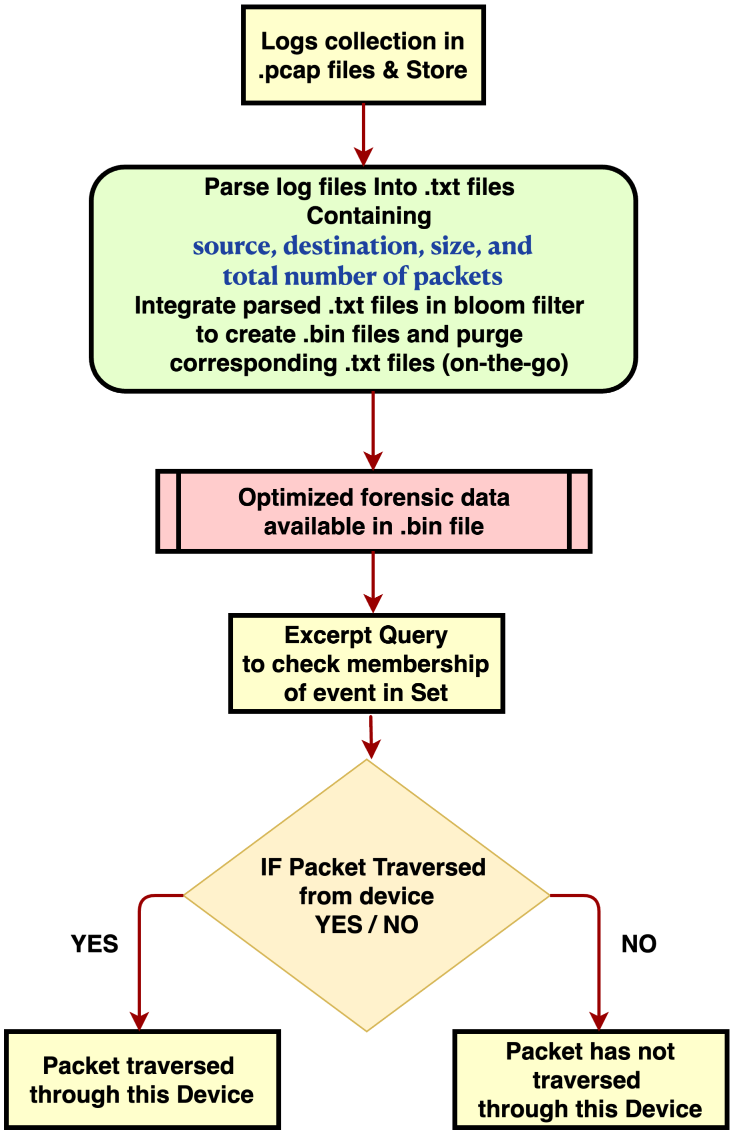

Logs collection in .pcap files & Store
Parse log files Into .txt files Containing
source, destination, size, and total number of packets
Integrate parsed .txt files in bloom filter to create .bin files and purge corresponding .txt files (on-the-go)
Optimized forensic data available in .bin file
Excerpt Query to check membership of event in Set
IF Packet Traversed from device YES / NO
YES
NO
Packet traversed through this Device
Packet has not traversed through this Device

Figure 4: Designed System

4.1 Logs Collection and Structure

In our proposed system, data processing is done on the collected and structured data in a specified structured format. The structured data is used to generate the expected results. To achieve this, we apply the log rolling and parsing the meaningful information from the logs (tcpdump packet capture data in pcap files).

4.1.1 Rolling Logs

The logs are collected and arranged in a format where each directory represents the logs collected from respective devices. In our controlled experiment, we specified that each file in the directory should not exceed 2 GB with a maximum limit (general industry standard log rolling fashion).

4.1.2 Detailed System Simulation

In our simulated system, the directory contains the log files of each device interface. In our experiment we use the *Mininet* to simulate a SDN network with 1 controller, 6 switches and 4 hosts as shown in 'Figure 5'. Controller is named as c0, switches are named as s1, s2, s3, s4, s5, s6 and hosts are named as h1, h2, h3, h4. We simulate a hybrid network topology by combining tree and mesh topology. All logs are centrally collected in a directory **experiment** with interfaces in their file names as shown in 'Figure 6'. Different files are created with names, '*s1-eth1.pcap, s1-eth2.pcap, etc.* which contain the packet captures of each interface of devices respectively. Simulated network dump and links are shown in 'Figure 7' and 'Figure 8' respectively.

'*The pox controller*' (SDN controller simulator used in our experiment) activation to handle hybrid network with mesh topology to avoid loops is shown in 'Figure 9'. SDN Controller is simulated to run at port *6633*.

Once we have the system set up and running as shown above, we now run a ping test from h1 to h4 to validate functional system simulation. Upon the successful ping test we can see the path traversed by the icmp packets:

s3 -> s1 -> s2 -> s6

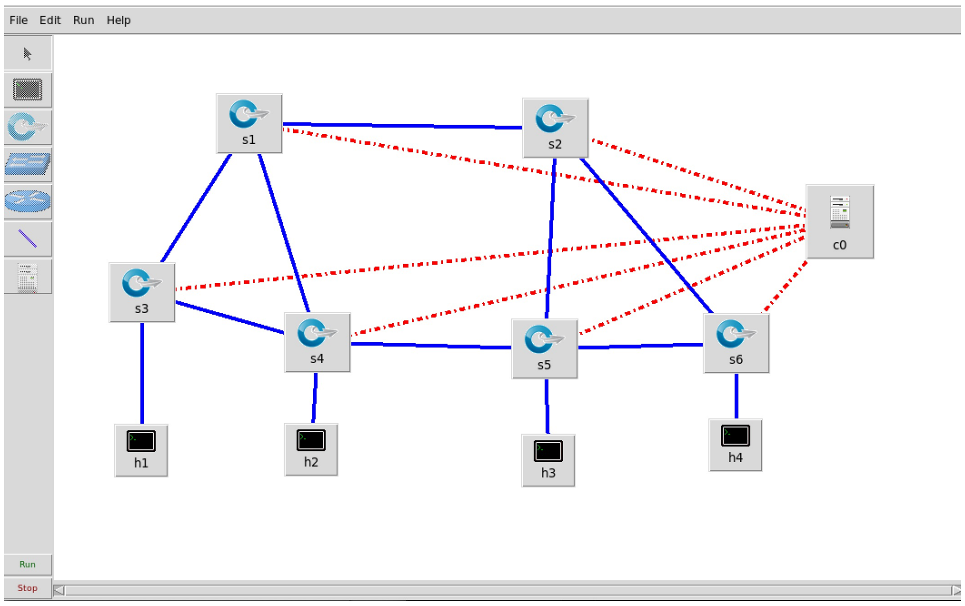
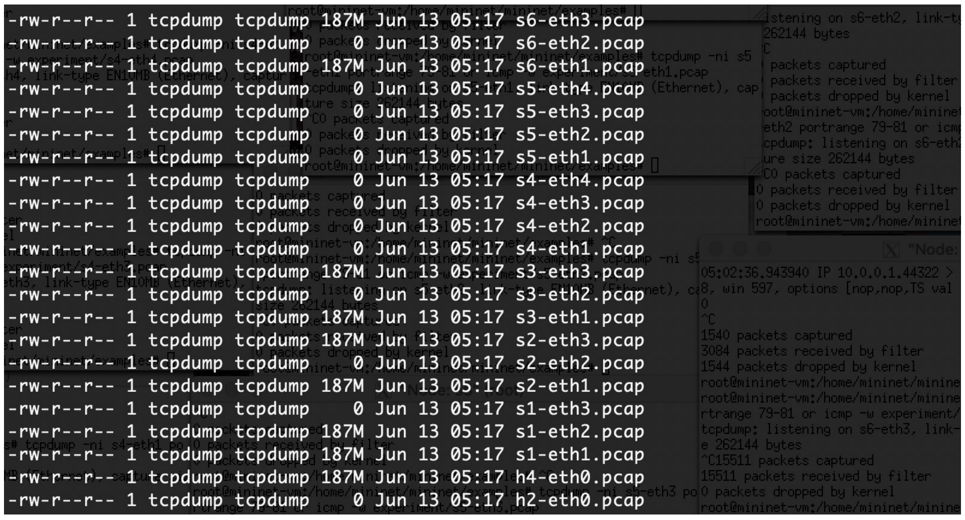


Figure 5: Simulation Experiment Topology

Complete ping test with the tcpdump captures enabled in all interfaces terminal can be seen in ‘Figure 10’. Packets detailed path with interfaces are recorded as **h1-eth0 -> s3-eth3 -> s3-eth1 -> s1-eth1 -> s1-eth2 -> s2-eth1 -> s2-eth3 -> s6-eth1 -> s6-eth3 -> h4-eth0** .

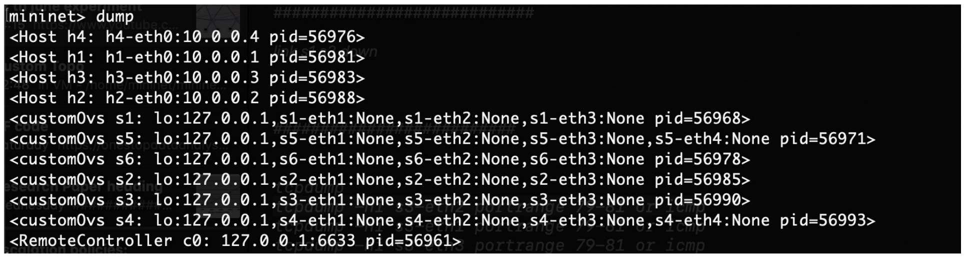
Upon successfully testing the network setup and smooth ping connectivity in all hosts we now extend the experiment with a web server serving on h4 and h1 being the client. We have setup h4 as a python http server which is serving the traffic at port 80 and h1 as client downloading a FILE of 95 MB from h4(web server) using ‘*wget*’ (wget http://10.0.0.4:80/FILE -o FILE). This experiment is run with active tcpdump captures at all interfaces of devices. After the FILE is downloaded at client h1, directory structure looks like as shown in ‘Figure 6’

Now logs are arranged in a well defined and meaningful format. We parse each tcpdump pcap file and fetch important information of each packet like source of the packet, destination of the packet, size of the packet and total number of packets as shown in ‘Figure 11’. Next we focus on how to integrate this data in BF, then design the query to find if the element exists in the BF or not and conclude the results.



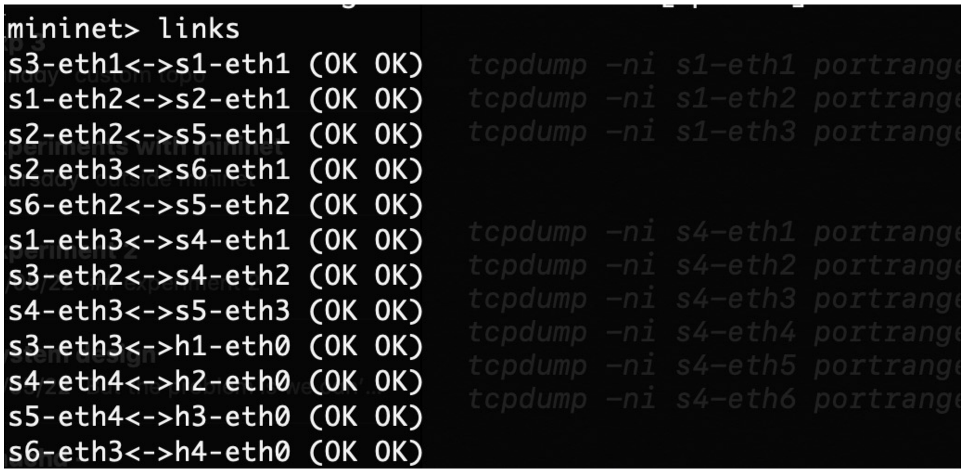
```
-rw-r--r-- 1 tcpdump tcpdump 187M Jun 13 05:17 s6-eth3.pcap
-rw-r--r-- 1 tcpdump tcpdump    0 Jun 13 05:17 s6-eth2.pcap
-rw-r--r-- 1 tcpdump tcpdump 187M Jun 13 05:17 s6-eth1.pcap
-rw-r--r-- 1 tcpdump tcpdump    0 Jun 13 05:17 s5-eth4.pcap
-rw-r--r-- 1 tcpdump tcpdump    0 Jun 13 05:17 s5-eth3.pcap
-rw-r--r-- 1 tcpdump tcpdump    0 Jun 13 05:17 s5-eth2.pcap
-rw-r--r-- 1 tcpdump tcpdump    0 Jun 13 05:17 s5-eth1.pcap
-rw-r--r-- 1 tcpdump tcpdump    0 Jun 13 05:17 s4-eth4.pcap
-rw-r--r-- 1 tcpdump tcpdump    0 Jun 13 05:17 s4-eth3.pcap
-rw-r--r-- 1 tcpdump tcpdump    0 Jun 13 05:17 s4-eth2.pcap
-rw-r--r-- 1 tcpdump tcpdump    0 Jun 13 05:17 s4-eth1.pcap
-rw-r--r-- 1 tcpdump tcpdump 187M Jun 13 05:17 s3-eth3.pcap
-rw-r--r-- 1 tcpdump tcpdump    0 Jun 13 05:17 s3-eth2.pcap
-rw-r--r-- 1 tcpdump tcpdump 187M Jun 13 05:17 s3-eth1.pcap
-rw-r--r-- 1 tcpdump tcpdump 187M Jun 13 05:17 s2-eth3.pcap
-rw-r--r-- 1 tcpdump tcpdump    0 Jun 13 05:17 s2-eth2.pcap
-rw-r--r-- 1 tcpdump tcpdump 187M Jun 13 05:17 s2-eth1.pcap
-rw-r--r-- 1 tcpdump tcpdump    0 Jun 13 05:17 s1-eth3.pcap
-rw-r--r-- 1 tcpdump tcpdump 187M Jun 13 05:17 s1-eth2.pcap
-rw-r--r-- 1 tcpdump tcpdump 187M Jun 13 05:17 s1-eth1.pcap
-rw-r--r-- 1 tcpdump tcpdump 187M Jun 13 05:17 h4-eth0.pcap
-rw-r--r-- 1 tcpdump tcpdump    0 Jun 13 05:17 h2-eth0.pcap
```

Figure 6: Logs Collection



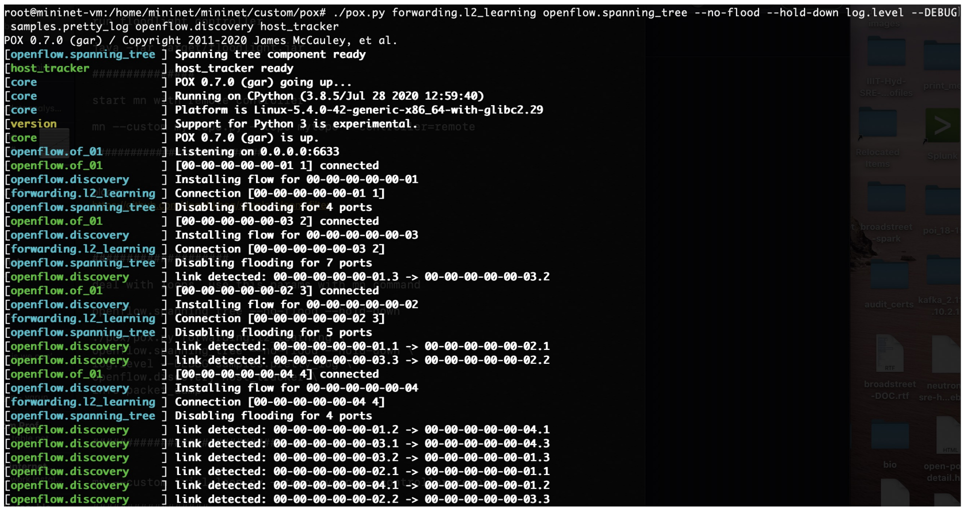
```
mininet> dump
<Host h4: h4-eth0:10.0.0.4 pid=56976>
<Host h1: h1-eth0:10.0.0.1 pid=56981>
<Host h3: h3-eth0:10.0.0.3 pid=56983>
<Host h2: h2-eth0:10.0.0.2 pid=56988>
<customOvs s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=56968>
<customOvs s5: lo:127.0.0.1,s5-eth1:None,s5-eth2:None,s5-eth3:None,s5-eth4:None pid=56971>
<customOvs s6: lo:127.0.0.1,s6-eth1:None,s6-eth2:None,s6-eth3:None pid=56978>
<customOvs s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=56985>
<customOvs s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=56990>
<customOvs s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None,s4-eth4:None pid=56993>
<RemoteController c0: 127.0.0.1:6633 pid=56961>
```

Figure 7: Network Dump



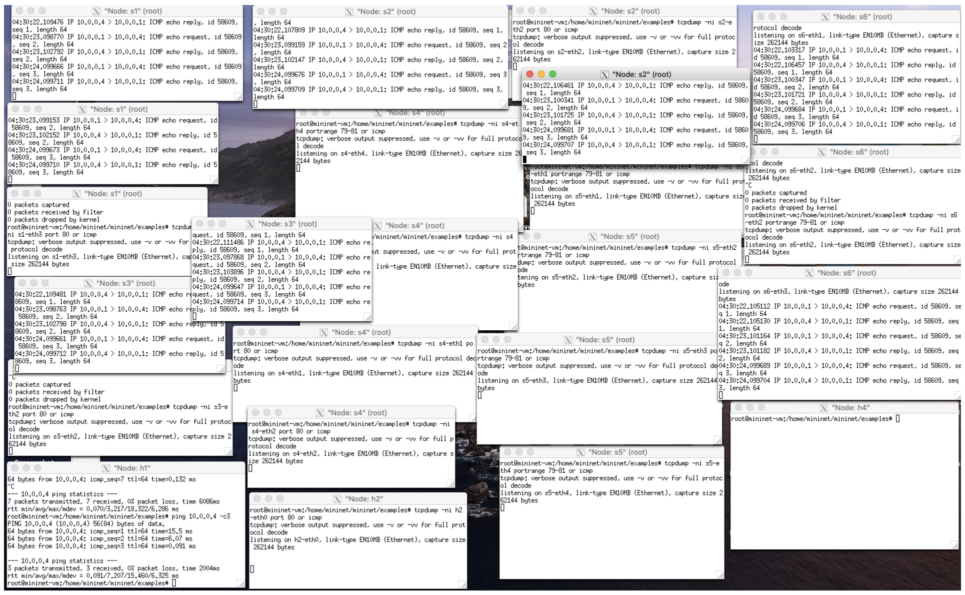
```
mininet> links
s3-eth1<->s1-eth1 (OK OK)
s1-eth2<->s2-eth1 (OK OK)
s2-eth2<->s5-eth1 (OK OK)
s2-eth3<->s6-eth1 (OK OK)
s6-eth2<->s5-eth2 (OK OK)
s1-eth3<->s4-eth1 (OK OK)
s3-eth2<->s4-eth2 (OK OK)
s4-eth3<->s5-eth3 (OK OK)
s3-eth3<->h1-eth0 (OK OK)
s4-eth4<->h2-eth0 (OK OK)
s5-eth4<->h3-eth0 (OK OK)
s6-eth3<->h4-eth0 (OK OK)
```

Figure 8: Network Links



```
root@mininet-vm:/home/mininet/mininet/custom/pox# ./pox.py forwarding.l2_learning openflow.spanning_tree --no-flood --hold-down log.level --DEBUG
 samples.pretty_log openflow.discovery host_tracker
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
[openflow.spanning_tree ] Spanning tree component ready
[host_tracker           ] host_tracker ready
[core                   ] POX 0.7.0 (gar) going up...
[core                   ] Running on CPython (3.8.5/Jul 28 2020 12:59:40)
[core                   ] Platform is Linux-5.4.0-42-generic-x86_64-with-glibc2.29
[version                ] Support for Python 3 is experimental.
[core                   ] POX 0.7.0 (gar) is up.
[openflow.of_01         ] Listening on 0.0.0.0:6633
[openflow.of_01         ] [00-00-00-00-00-01 1] connected
[openflow.discovery     ] Installing flow for 00-00-00-00-00-01
[forwarding.l2_learning ] Connection [00-00-00-00-00-01 1]
[openflow.spanning_tree ] Disabling flooding for 4 ports
[openflow.of_01         ] [00-00-00-00-00-03 2] connected
[openflow.discovery     ] Installing flow for 00-00-00-00-00-03
[forwarding.l2_learning ] Connection [00-00-00-00-00-03 2]
[openflow.spanning_tree ] Disabling flooding for 7 ports
[openflow.discovery     ] link detected: 00-00-00-00-00-01.3 -> 00-00-00-00-00-03.2
[openflow.of_01         ] [00-00-00-00-00-02 3] connected
[openflow.discovery     ] Installing flow for 00-00-00-00-00-02
[forwarding.l2_learning ] Connection [00-00-00-00-00-02 3]
[openflow.spanning_tree ] Disabling flooding for 5 ports
[openflow.discovery     ] link detected: 00-00-00-00-00-01.1 -> 00-00-00-00-00-02.1
[openflow.discovery     ] link detected: 00-00-00-00-00-03.3 -> 00-00-00-00-00-02.2
[openflow.of_01         ] [00-00-00-00-00-04 4] connected
[openflow.discovery     ] Installing flow for 00-00-00-00-00-04
[forwarding.l2_learning ] Connection [00-00-00-00-00-04 4]
[openflow.spanning_tree ] Disabling flooding for 4 ports
[openflow.discovery     ] link detected: 00-00-00-00-00-01.2 -> 00-00-00-00-00-04.1
[openflow.discovery     ] link detected: 00-00-00-00-00-03.1 -> 00-00-00-00-00-04.3
[openflow.discovery     ] link detected: 00-00-00-00-00-03.2 -> 00-00-00-00-00-01.3
[openflow.discovery     ] link detected: 00-00-00-00-00-02.1 -> 00-00-00-00-00-01.1
[openflow.discovery     ] link detected: 00-00-00-00-00-04.1 -> 00-00-00-00-00-01.2
[openflow.discovery     ] link detected: 00-00-00-00-00-02.2 -> 00-00-00-00-00-03.3
```

Figure 9: SDN Controller Activation



```
"Node: s1" (root)
04:30:22.109476 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id 58609,
seq 1, length 64
04:30:23.098770 IP 10.0.0.1 > 10.0.0.4: ICMP echo request, id 58609
, seq 2, length 64
04:30:23.102792 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id 58609,
seq 2, length 64
04:30:24.099666 IP 10.0.0.1 > 10.0.0.4: ICMP echo request, id 58609
, seq 3, length 64
04:30:24.099711 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id 58609,
seq 3, length 64
```

```
"Node: s2" (root)
, length 64
04:30:22.107809 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id 58609, seq 1,
length 64
04:30:23.099159 IP 10.0.0.1 > 10.0.0.4: ICMP echo request, id 58609, seq 2
, length 64
04:30:23.102147 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id 58609, seq 2,
length 64
04:30:24.099676 IP 10.0.0.1 > 10.0.0.4: ICMP echo request, id 58609, seq 3
, length 64
04:30:24.099709 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id 58609, seq 3,
length 64
```

```
"Node: s2" (root)
root@mininet-vm:/home/mininet/mininet/examples# tcpdump -ni s2-e
th2 port 80 or icmp
tcpdump: verbose output suppressed, use -v or -vv for full protoc
ol decode
listening on s2-eth2, link-type EN10MB (Ethernet), capture size 2
62144 bytes
```

```
"Node: s6" (root)
rotocol decode
listening on s6-eth1, link-type EN10MB (Ethernet), capture s
ize 262144 bytes
04:30:22.103317 IP 10.0.0.1 > 10.0.0.4: ICMP echo request, i
d 58609, seq 1, length 64
04:30:22.106457 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id
58609, seq 1, length 64
04:30:23.100347 IP 10.0.0.1 > 10.0.0.4: ICMP echo request, i
d 58609, seq 2, length 64
04:30:23.101721 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id
58609, seq 2, length 64
04:30:24.099684 IP 10.0.0.1 > 10.0.0.4: ICMP echo request, i
d 58609, seq 3, length 64
04:30:24.099706 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id
58609, seq 3, length 64
```

```
"Node: s1" (root)
04:30:23.099153 IP 10.0.0.1 > 10.0.0.4: ICMP echo request, id
 58609, seq 2, length 64
04:30:23.102152 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id 5
8609, seq 2, length 64
04:30:24.099673 IP 10.0.0.1 > 10.0.0.4: ICMP echo request, id
 58609, seq 3, length 64
04:30:24.099710 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id 5
8609, seq 3, length 64
```

```
"Node: s4" (root)
root@mininet-vm:/home/mininet/mininet/examples# tcpdump -ni s4-et
h4 portrange 79-81 or icmp
tcpdump: verbose output suppressed, use -v or -vv for full protoco
l decode
listening on s4-eth4, link-type EN10MB (Ethernet), capture size 26
2144 bytes
```

```
"Node: s2" (root)
04:30:22.106461 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id 58609,
 seq 1, length 64
04:30:23.100341 IP 10.0.0.1 > 10.0.0.4: ICMP echo request, id 5860
9, seq 2, length 64
04:30:23.101725 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id 58609,
 seq 2, length 64
04:30:24.099681 IP 10.0.0.1 > 10.0.0.4: ICMP echo request, id 5860
9, seq 3, length 64
04:30:24.099707 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id 58609,
 seq 3, length 64
```

```
root@mininet-vm:/home/mininet/mininet/examples# tcpdump -ni s5
-eth1 portrange 79-81 or icmp
tcpdump: verbose output suppressed, use -v or -vv for full prot
ocol decode
listening on s5-eth1, link-type EN10MB (Ethernet), capture size
 262144 bytes
```

```
"Node: s6" (root)
ocol decode
listening on s6-eth2, link-type EN10MB (Ethernet), capture size
 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@mininet-vm:/home/mininet/mininet/examples# tcpdump -ni s6
-eth2 portrange 79-81 or icmp
tcpdump: verbose output suppressed, use -v or -vv for full prot
ocol decode
listening on s6-eth2, link-type EN10MB (Ethernet), capture size
 262144 bytes
```

```
"Node: s1" (root)
0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@mininet-vm:/home/mininet/mininet/examples# tcpdum
ni s1-eth3 port 80 or icmp
tcpdump: verbose output suppressed, use -v or -vv for
 protocol decode
listening on s1-eth3, link-type EN10MB (Ethernet), cap
 size 262144 bytes
```

```
"Node: s3" (root)
quest, id 58609, seq 1, length 64
04:30:22.111486 IP 10.0.0.4 > 10.0.0.1: ICMP echo re
ply, id 58609, seq 1, length 64
04:30:23.097868 IP 10.0.0.1 > 10.0.0.4: ICMP echo re
quest, id 58609, seq 2, length 64
04:30:23.103896 IP 10.0.0.4 > 10.0.0.1: ICMP echo re
ply, id 58609, seq 2, length 64
04:30:24.099647 IP 10.0.0.1 > 10.0.0.4: ICMP echo re
quest, id 58609, seq 3, length 64
04:30:24.099714 IP 10.0.0.4 > 10.0.0.1: ICMP echo re
ply, id 58609, seq 3, length 64
```

```
"Node: s4" (root)
/mininet/mininet/examples# tcpdump -ni s4

ut suppressed, use -v or -vv for full prot

 link-type EN10MB (Ethernet), capture size
```

```
"Node: s5" (root)
t@mininet-vm:/home/mininet/mininet/examples# tcpdump -ni s5-eth2
rtrange 79-81 or icmp
dump: verbose output suppressed, use -v or -vv for full protocol
ode
tening on s5-eth2, link-type EN10MB (Ethernet), capture siz
bytes
```

```
"Node: s6" (root)
ode
listening on s6-eth3, link-type EN10MB (Ethernet), capture size 262144
bytes
04:30:22.105112 IP 10.0.0.1 > 10.0.0.4: ICMP echo request, id 58609, se
q 1, length 64
04:30:22.105130 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id 58609, seq
1, length 64
04:30:23.101164 IP 10.0.0.1 > 10.0.0.4: ICMP echo request, id 58609, se
q 2, length 64
04:30:23.101182 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id 58609, seq
2, length 64
04:30:24.099689 IP 10.0.0.1 > 10.0.0.4: ICMP echo request, id 58609, se
q 3, length 64
04:30:24.099704 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id 58609, seq
3, length 64
```

```
"Node: s3" (root)
04:30:22.109481 IP 10.0.0.4 > 10.0.0.1: ICMP echo r
8609, seq 1, length 64
04:30:23.098763 IP 10.0.0.1 > 10.0.0.4: ICMP echo r
 58609, seq 2, length 64
04:30:23.102798 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id 5
8609, seq 2, length 64
04:30:24.099661 IP 10.0.0.1 > 10.0.0.4: ICMP echo request, id
 58609, seq 3, length 64
04:30:24.099712 IP 10.0.0.4 > 10.0.0.1: ICMP echo reply, id 5
8609, seq 3, length 64
```

```
"Node: s4" (root)
root@mininet-vm:/home/mininet/mininet/examples# tcpdump -ni s4-eth1 po
rt 80 or icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol dec
ode
listening on s4-eth1, link-type EN10MB (Ethernet), capture size 262144
bytes
```

```
"Node: s5" (root)
root@mininet-vm:/home/mininet/mininet/examples# tcpdump -ni s5-eth3 po
rtrange 79-81 or icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol de
ode
listening on s5-eth3, link-type EN10MB (Ethernet), capture size 262144
bytes
```

```
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@mininet-vm:/home/mininet/mininet/examples# tcpdump -ni s3-e
th2 port 80 or icmp
tcpdump: verbose output suppressed, use -v or -vv for full protoc
ol decode
listening on s3-eth2, link-type EN10MB (Ethernet), capture size 2
62144 bytes
```

```
"Node: s4" (root)
root@mininet-vm:/home/mininet/mininet/examples# tcpdump -ni
 s4-eth2 port 80 or icmp
tcpdump: verbose output suppressed, use -v or -vv for full p
rotocol decode
listening on s4-eth2, link-type EN10MB (Ethernet), capture s
ize 262144 bytes
```

```
"Node: h4"
root@mininet-vm:/home/mininet/mininet/examples#
```

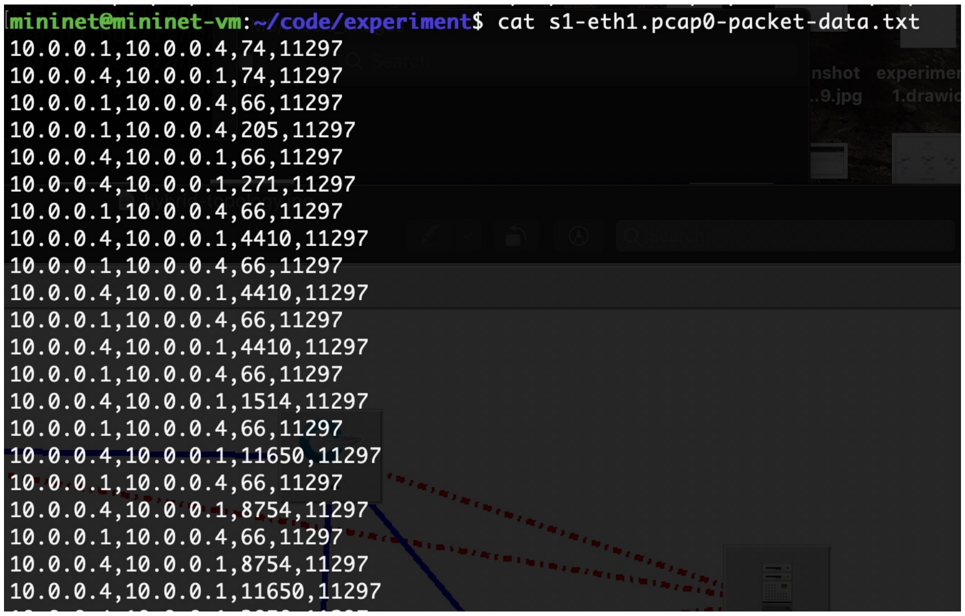
```
"Node: s5" (root)
root@mininet-vm:/home/mininet/mininet/examples# tcpdump -ni s5-e
th4 portrange 79-81 or icmp
tcpdump: verbose output suppressed, use -v or -vv for full protoc
ol decode
listening on s5-eth4, link-type EN10MB (Ethernet), capture size 2
62144 bytes
```

```
"Node: h1"
64 bytes from 10.0.0.4: icmp_seq=7 ttl=64 time=0.132 ms
^C
--- 10.0.0.4 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6086ms
rtt min/avg/max/mdev = 0.070/3.217/18.322/6.286 ms
root@mininet-vm:/home/mininet/mininet/examples# ping 10.0.0.4 -c3
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=15.5 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=6.07 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.091 ms

--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 0.091/7.207/15.460/6.325 ms
root@mininet-vm:/home/mininet/mininet/examples#
```

```
"Node: h2"
root@mininet-vm:/home/mininet/mininet/examples# tcpdump -ni h2
-eth0 port 80 or icmp
tcpdump: verbose output suppressed, use -v or -vv for full prot
ocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size
 262144 bytes
```

Figure 10: *Ping Test on Simulated Network:* h1->s3->s1->s2->s6->h4



```
mininet@mininet-vm:~/code/experiment$ cat s1-eth1.pcap0-packet-data.txt
10.0.0.1,10.0.0.4,74,11297
10.0.0.4,10.0.0.1,74,11297
10.0.0.1,10.0.0.4,66,11297
10.0.0.1,10.0.0.4,205,11297
10.0.0.4,10.0.0.1,66,11297
10.0.0.4,10.0.0.1,271,11297
10.0.0.1,10.0.0.4,66,11297
10.0.0.4,10.0.0.1,4410,11297
10.0.0.1,10.0.0.4,66,11297
10.0.0.4,10.0.0.1,4410,11297
10.0.0.1,10.0.0.4,66,11297
10.0.0.4,10.0.0.1,4410,11297
10.0.0.1,10.0.0.4,66,11297
10.0.0.4,10.0.0.1,1514,11297
10.0.0.1,10.0.0.4,66,11297
10.0.0.4,10.0.0.1,11650,11297
10.0.0.1,10.0.0.4,66,11297
10.0.0.4,10.0.0.1,8754,11297
10.0.0.1,10.0.0.4,66,11297
10.0.0.4,10.0.0.1,8754,11297
10.0.0.4,10.0.0.1,11650,11297
```

Figure 11: Parsed Data

4.2 Bloom Filter Integration

Now we integrate the parsed data as shown in 'Figure 11' in bloom filter.

4.2.1 Bloom Filter Implementation

As explained above in 'Chapter 4.1.1' and 'Chapter 4.1.2', we add the collected data into bloom filters. We put the values fetched from each packet like packet size, packets count, source and destination IP of each packet into the bloom filters and then we query to check if that specific packet exists in any of the device pcap files. In 'Chapter 4.2.2' we analyse and create the bloom filters that can avoid the saturation and deliver the optimised results. Upon reaching the threshold to hit the bloom filter saturation, the system creates another .bin (bloom filter) file and so on in rolling fashion.

4.2.2 Analyze Optimum Hash Functions

We have analyzed the optimum hash functions in the experiment and for the scaled use case. As we increase the number of hash functions, the false positive probability goes low as shown in the space efficiency of a bloom filter:

$$FP = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k} \approx (1 - e^{kn/m})^{k}) \tag{1}$$

The false positive probability 'p' is a function of number of elements 'n' in the filter and the filter size 'm'. An optimal number of hash functions has been assumed. E.g., please refer 'Table 5', 'Table 6', 'Table 7' and 'Table 8' for selecting the number of hash functions to be implemented in bloom filter for the respective false positive probability. In the experiments we have implemented bloom filters with 10 hash functions and a false positive probability of 0.001.

$$k = (m/n) \ln 2 \tag{2}$$

Table 5: Number of hash functions used for various false positive probabilities

Suggested FPP of Bloom Filter (p)	***Hash Functions (k)***
3%	5
1%	7
0.1%*	10*
0.01%	13
0.001%	17

Table 6: Probability of false positives with hash functions

Items in BF(n)	***hash function(k)***	***bits in filter(m)***	***FPP (p)***
1 million	1	100 MB	1 in 801
1 million	3	100 MB	1 in 19 million
1 million*	5*	100 MB	1 in 106 billion*

‘Table 7’ illustrates the behavior of FPP with the number of items in bloom filter. In this experiment we observed that the data with 1 million items in bloom filter, 5 hashes and 100 MB bits in filter (size) is actually much more fault tolerant which is 1 false positive in 106 billion queries.

Table 7: Probability of false positives with number of items in Bloomfilter

Items in BF(n)	***hash function(k)***	***bits in filter(m)***	***FPP (p)***
1 million*	5*	100 MB	1 in 106 billion*
2 million	5	100 MB	1 in 3 billion
3 million	5	100 MB	1 in 452 million
4 million	5	100 MB	1 in 108 million
10 million	5	100 MB	1 in 1 million

‘Table 8’ illustrates the behavior of FPP with the bits in filter (size). During the experiments we analyzed that we can create an optimized data structure using the bit size of 2000MB with 1 million items and 10 hashes.

Table 8: Probability of false positives with number of bits in filter

Items in BF(n)	***hash functions(k)***	***bits in filter(m)***	***FPP (p)***
1 million	5	1000 MB	1 in 10 Quadri
1 million	5	2000 MB*	1 in 335 Quadri
1 million	5	3000 MB	1 in 2 Quinti
1 million	5	10000 MB	1 in 1 Sexti

4.3 Complexity Analysis

4.3.1 Space complexity Analysis

By analyzing the details as per the experimental values in ‘Table 5’ & ‘Figure 12’ the proposed system excludes the bz2 compression strategy due to the costlier time complexity algorithm. System integrated with raw data, zip compressed data and data added in bloom filter enhances the availability of forensic logs with similar hardware resources. This derives the conclusion that the introduced system can hold substantially more forensic data as compared to existing forensic data storage mechanisms.

4.3.2 Time complexity Analysis

In practice Hashing has one of the best complexities with a fantastic O(1) constant search time. 'Table 9' illustrates the time complexity of the queries on the organized data in bloom filter. We observed that 10,000 items in bloom filter produces considerably optimized search results with around 5 seconds for the query.

Table 9: Time Complexity

Number of Items in BF (n)	***Time Taken to search (ms)***
1000	1000
5000	2500
10000*	5100*
20000	11500
30000	17300

5 Evaluation

5.1 System Execution and Results

We successfully simulated a '*hybrid network topology*' network as shown in 'Figure 5'. We have performed experiments to compare the proposed system with existing systems. Refer to 'Table 5' for disk space utilization ratio for storing the logs with different compression algorithms and for the comparative analysis.

'Figure 12' compares the compression algorithms applied to 1024 GB of sample log directories of 50 MB files each. Important thing to note here is that the disk size utilized by bloom filter does not depend on log file size, instead each element in bloom filter occupies around 5 Bytes of disk space for storing a file of size 200 MB assuming the number of hash function as 7 or 10. Our experiments are done on diverse directory sizes of 128 GB, 512 GB and 1024 GB with rolling file size of 50 MB / 200 MB / 1GB / 2 GB.

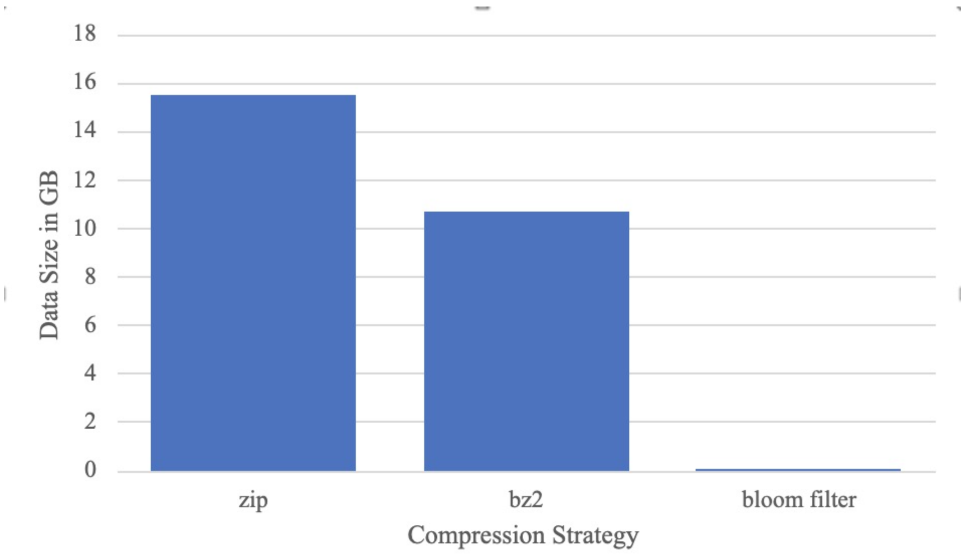
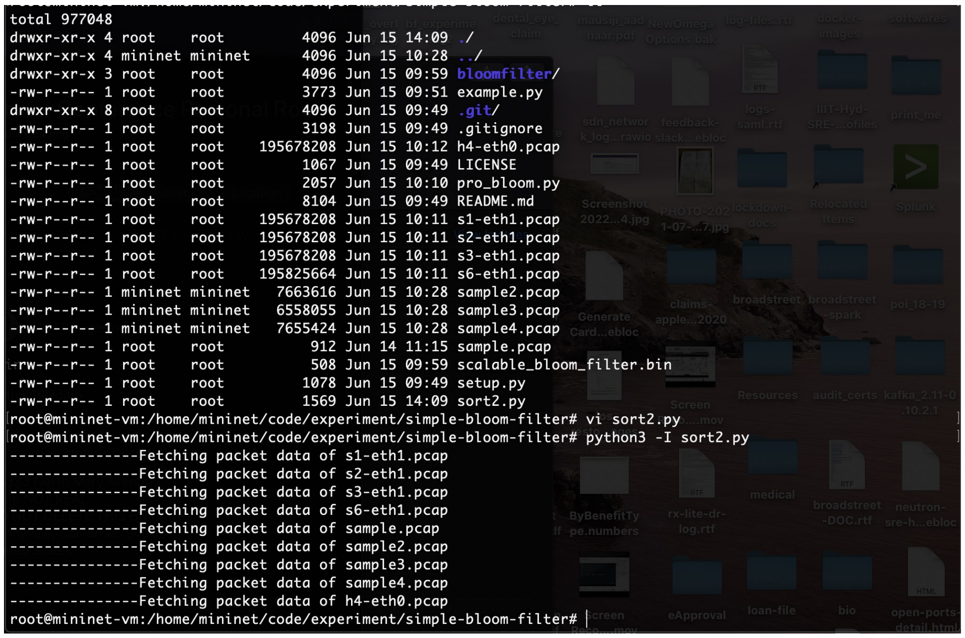


Figure 12: Compression Analysis of 1024 GB sample data

The experiment execution is demonstrated in the figures below with step-by-step explanation.

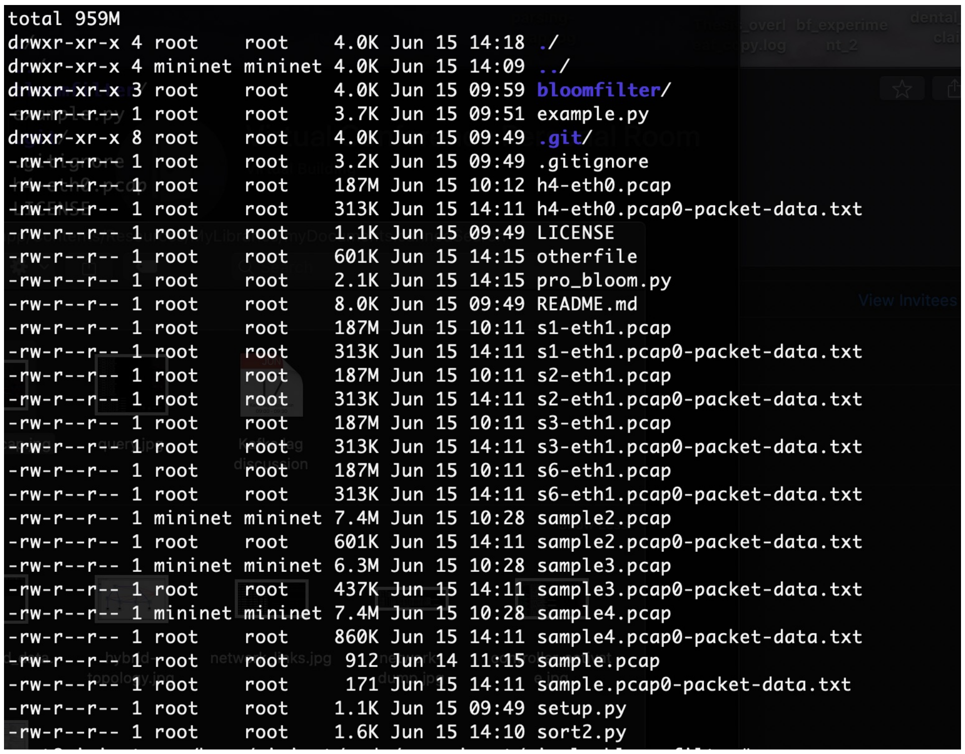


```
total 977048
drwxr-xr-x 4 root    root         4096 Jun 15 14:09 ./
drwxr-xr-x 4 mininet mininet      4096 Jun 15 10:28 ../
drwxr-xr-x 3 root    root         4096 Jun 15 09:59 bloomfilter/
-rw-r--r-- 1 root    root         3773 Jun 15 09:51 example.py
drwxr-xr-x 8 root    root         4096 Jun 15 09:49 .git/
-rw-r--r-- 1 root    root         3198 Jun 15 09:49 .gitignore
-rw-r--r-- 1 root    root    195678208 Jun 15 10:12 h4-eth0.pcap
-rw-r--r-- 1 root    root         1067 Jun 15 09:49 LICENSE
-rw-r--r-- 1 root    root         2057 Jun 15 10:10 pro_bloom.py
-rw-r--r-- 1 root    root         8104 Jun 15 09:49 README.md
-rw-r--r-- 1 root    root    195678208 Jun 15 10:11 s1-eth1.pcap
-rw-r--r-- 1 root    root    195678208 Jun 15 10:11 s2-eth1.pcap
-rw-r--r-- 1 root    root    195678208 Jun 15 10:11 s3-eth1.pcap
-rw-r--r-- 1 root    root    195825664 Jun 15 10:11 s6-eth1.pcap
-rw-r--r-- 1 mininet mininet   7663616 Jun 15 10:28 sample2.pcap
-rw-r--r-- 1 mininet mininet   6558055 Jun 15 10:28 sample3.pcap
-rw-r--r-- 1 mininet mininet   7655424 Jun 15 10:28 sample4.pcap
-rw-r--r-- 1 root    root          912 Jun 14 11:15 sample.pcap
-rw-r--r-- 1 root    root          508 Jun 15 09:59 scalable_bloom_filter.bin
-rw-r--r-- 1 root    root         1078 Jun 15 09:49 setup.py
-rw-r--r-- 1 root    root         1569 Jun 15 14:09 sort2.py
root@mininet-vm:/home/mininet/code/experiment/simple-bloom-filter# vi sort2.py
root@mininet-vm:/home/mininet/code/experiment/simple-bloom-filter# python3 -I sort2.py
---------------Fetching packet data of s1-eth1.pcap
---------------Fetching packet data of s2-eth1.pcap
---------------Fetching packet data of s3-eth1.pcap
---------------Fetching packet data of s6-eth1.pcap
---------------Fetching packet data of sample.pcap
---------------Fetching packet data of sample2.pcap
---------------Fetching packet data of sample3.pcap
---------------Fetching packet data of sample4.pcap
---------------Fetching packet data of h4-eth0.pcap
root@mininet-vm:/home/mininet/code/experiment/simple-bloom-filter# |
```

Figure 13: Parsing pcap Files

‘Figure 13’ shows the code execution to parse the *pcap* files into text files with all necessary elements.

‘Figure 14’ shows how the directory look post processing the raw pcap files into txt files.

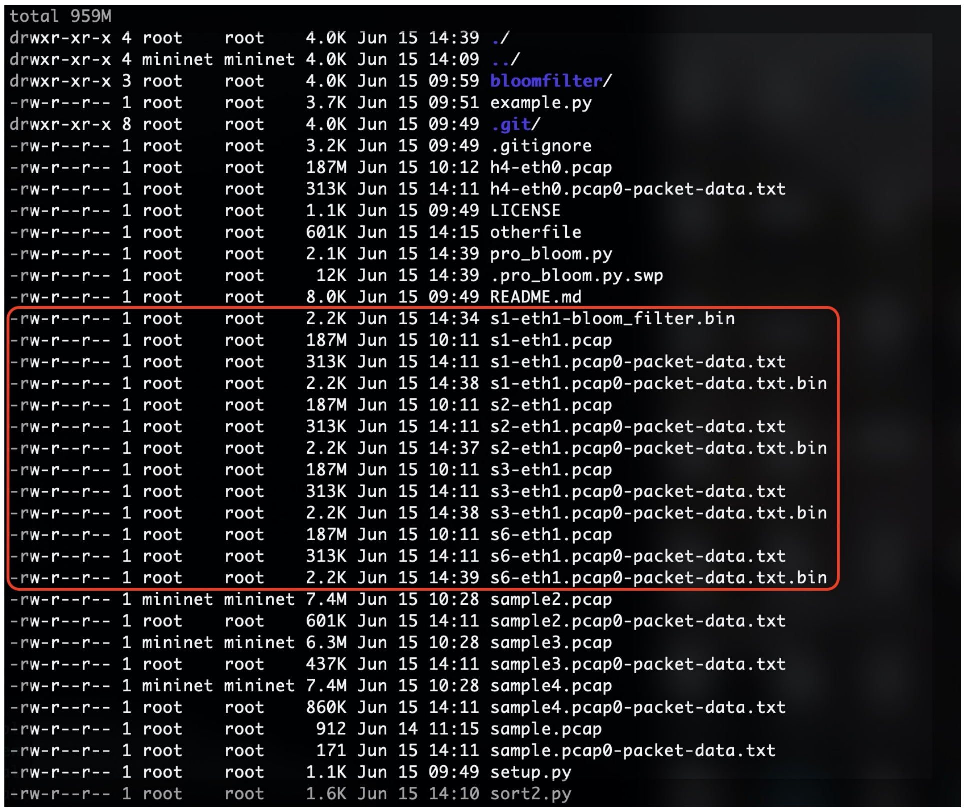


```
total 959M
drwxr-xr-x 4 root    root    4.0K Jun 15 14:18 ./
drwxr-xr-x 4 mininet mininet 4.0K Jun 15 14:09 ../
drwxr-xr-x 3 root    root    4.0K Jun 15 09:59 bloomfilter/
-rw-r--r-- 1 root    root    3.7K Jun 15 09:51 example.py
drwxr-xr-x 8 root    root    4.0K Jun 15 09:49 .git/
-rw-r--r-- 1 root    root    3.2K Jun 15 09:49 .gitignore
-rw-r--r-- 1 root    root    187M Jun 15 10:12 h4-eth0.pcap
-rw-r--r-- 1 root    root    313K Jun 15 14:11 h4-eth0.pcap0-packet-data.txt
-rw-r--r-- 1 root    root    1.1K Jun 15 09:49 LICENSE
-rw-r--r-- 1 root    root    601K Jun 15 14:15 otherfile
-rw-r--r-- 1 root    root    2.1K Jun 15 14:15 pro_bloom.py
-rw-r--r-- 1 root    root    8.0K Jun 15 09:49 README.md
-rw-r--r-- 1 root    root    187M Jun 15 10:11 s1-eth1.pcap
-rw-r--r-- 1 root    root    313K Jun 15 14:11 s1-eth1.pcap0-packet-data.txt
-rw-r--r-- 1 root    root    187M Jun 15 10:11 s2-eth1.pcap
-rw-r--r-- 1 root    root    313K Jun 15 14:11 s2-eth1.pcap0-packet-data.txt
-rw-r--r-- 1 root    root    187M Jun 15 10:11 s3-eth1.pcap
-rw-r--r-- 1 root    root    313K Jun 15 14:11 s3-eth1.pcap0-packet-data.txt
-rw-r--r-- 1 root    root    187M Jun 15 10:11 s6-eth1.pcap
-rw-r--r-- 1 root    root    313K Jun 15 14:11 s6-eth1.pcap0-packet-data.txt
-rw-r--r-- 1 mininet mininet 7.4M Jun 15 10:28 sample2.pcap
-rw-r--r-- 1 root    root    601K Jun 15 14:11 sample2.pcap0-packet-data.txt
-rw-r--r-- 1 mininet mininet 6.3M Jun 15 10:28 sample3.pcap
-rw-r--r-- 1 root    root    437K Jun 15 14:11 sample3.pcap0-packet-data.txt
-rw-r--r-- 1 mininet mininet 7.4M Jun 15 10:28 sample4.pcap
-rw-r--r-- 1 root    root    860K Jun 15 14:11 sample4.pcap0-packet-data.txt
-rw-r--r-- 1 root    root     912 Jun 14 11:15 sample.pcap
-rw-r--r-- 1 root    root     171 Jun 15 14:11 sample.pcap0-packet-data.txt
-rw-r--r-- 1 root    root    1.1K Jun 15 09:49 setup.py
-rw-r--r-- 1 root    root    1.6K Jun 15 14:10 sort2.py
```

Figure 14: Files After Parsing

'Figure 15' shows the processing of data to add elements from txt files into BF. We can observe the space utilization of pcap files vs bin files.

'Figure 16' shows the query results. Query data is any line of txt file parsed for interfaces '*(s1-eth1.pcap0-packet-data.txt / s6-eth1.pcap0-packet-data.txt / s3-eth1.pcap0-packet-data.txt / s2-eth1.pcap0-packet-data.txt)* which is queried upon bloom filter bin files created for each interface to check if the packet passed through. We also parsed a couple of random sample.pcap files *(downloaded from https://www.wireshark.org/download/automated/captures/)* and elements are put in txt files. The data from *(sample.pcap0-packet-data.txt / sample2.pcap0-packet-data.txt / sample3.pcap0-packet-data.txt / sample4.pcap0-packet-data.txt)* files is then queried against any of generated bin file from our experiment topology interfaces and the results are shown in 'Figure 16'.



```
total 959M
drwxr-xr-x 4 root    root    4.0K Jun 15 14:39 ./
drwxr-xr-x 4 mininet mininet 4.0K Jun 15 14:09 ../
drwxr-xr-x 3 root    root    4.0K Jun 15 09:59 bloomfilter/
-rw-r--r-- 1 root    root    3.7K Jun 15 09:51 example.py
drwxr-xr-x 8 root    root    4.0K Jun 15 09:49 .git/
-rw-r--r-- 1 root    root    3.2K Jun 15 09:49 .gitignore
-rw-r--r-- 1 root    root    187M Jun 15 10:12 h4-eth0.pcap
-rw-r--r-- 1 root    root    313K Jun 15 14:11 h4-eth0.pcap0-packet-data.txt
-rw-r--r-- 1 root    root    1.1K Jun 15 09:49 LICENSE
-rw-r--r-- 1 root    root    601K Jun 15 14:15 otherfile
-rw-r--r-- 1 root    root    2.1K Jun 15 14:39 pro_bloom.py
-rw-r--r-- 1 root    root     12K Jun 15 14:39 .pro_bloom.py.swp
-rw-r--r-- 1 root    root    8.0K Jun 15 09:49 README.md
-rw-r--r-- 1 root    root    2.2K Jun 15 14:34 s1-eth1-bloom_filter.bin
-rw-r--r-- 1 root    root    187M Jun 15 10:11 s1-eth1.pcap
-rw-r--r-- 1 root    root    313K Jun 15 14:11 s1-eth1.pcap0-packet-data.txt
-rw-r--r-- 1 root    root    2.2K Jun 15 14:38 s1-eth1.pcap0-packet-data.txt.bin
-rw-r--r-- 1 root    root    187M Jun 15 10:11 s2-eth1.pcap
-rw-r--r-- 1 root    root    313K Jun 15 14:11 s2-eth1.pcap0-packet-data.txt
-rw-r--r-- 1 root    root    2.2K Jun 15 14:37 s2-eth1.pcap0-packet-data.txt.bin
-rw-r--r-- 1 root    root    187M Jun 15 10:11 s3-eth1.pcap
-rw-r--r-- 1 root    root    313K Jun 15 14:11 s3-eth1.pcap0-packet-data.txt
-rw-r--r-- 1 root    root    2.2K Jun 15 14:38 s3-eth1.pcap0-packet-data.txt.bin
-rw-r--r-- 1 root    root    187M Jun 15 10:11 s6-eth1.pcap
-rw-r--r-- 1 root    root    313K Jun 15 14:11 s6-eth1.pcap0-packet-data.txt
-rw-r--r-- 1 root    root    2.2K Jun 15 14:39 s6-eth1.pcap0-packet-data.txt.bin
-rw-r--r-- 1 mininet mininet 7.4M Jun 15 10:28 sample2.pcap
-rw-r--r-- 1 root    root    601K Jun 15 14:11 sample2.pcap0-packet-data.txt
-rw-r--r-- 1 mininet mininet 6.3M Jun 15 10:28 sample3.pcap
-rw-r--r-- 1 root    root    437K Jun 15 14:11 sample3.pcap0-packet-data.txt
-rw-r--r-- 1 mininet mininet 7.4M Jun 15 10:28 sample4.pcap
-rw-r--r-- 1 root    root    860K Jun 15 14:11 sample4.pcap0-packet-data.txt
-rw-r--r-- 1 root    root     912 Jun 14 11:15 sample.pcap
-rw-r--r-- 1 root    root     171 Jun 15 14:11 sample.pcap0-packet-data.txt
-rw-r--r-- 1 root    root    1.1K Jun 15 09:49 setup.py
-rw-r--r-- 1 root    root    1.6K Jun 15 14:10 sort2.py
```

Figure 15: .pcap Files and Corresponding .bin Files

Code snippet for parsing and sorting the pcap files.

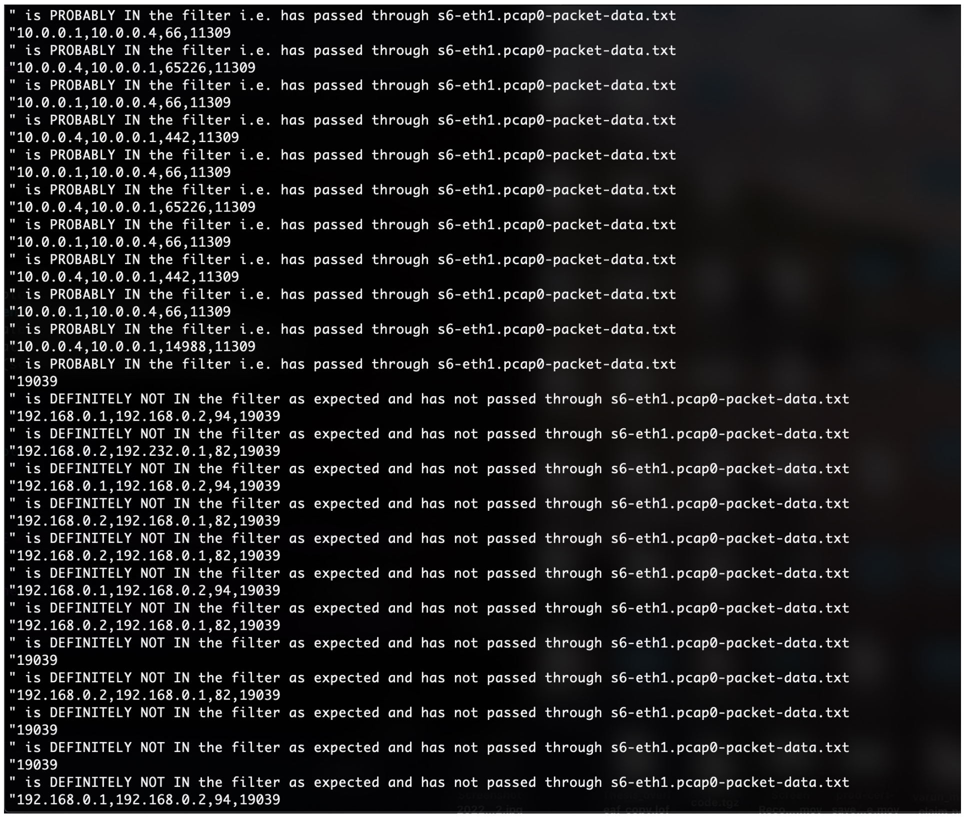
```python
import argparse
import os
import sys
from scapy.utils import RawPcapReader
from scapy.all import *

def get_pack_data(kk):
    count = 0
    fileName = os.path.basename(kk) + str(count)
    with open('{}-packet-data.txt'.format(fileName), 'w') as
   file1:  # write mode
        for p in pcap:
            count += 1
            if p.haslayer(IP) == 1:
                s1 = p[IP].fields['src']
                d1 = p[IP].fields['dst']
                sd = (format(s1)+','+ format(d1))
                psize = (format(len(p)))

                file1.write(sd +"," + psize +",")
            file1.write(format(len(pcap))+"\n")
if __name__ == "__main__":

    pcap_path = ["s1-eth1.pcap", "s2-eth1.pcap", "s3-eth1.pcap", "s6-eth1.pcap", "sample.pcap", "sample2.pcap", "sample3.pcap", "sample4.pcap", "h4-eth0.pcap"]

    for k in pcap_path:
        pcap = rdpcap(k)
        print('---------------Fetching packet data of ' + k)
        get_pack_data(k)
```

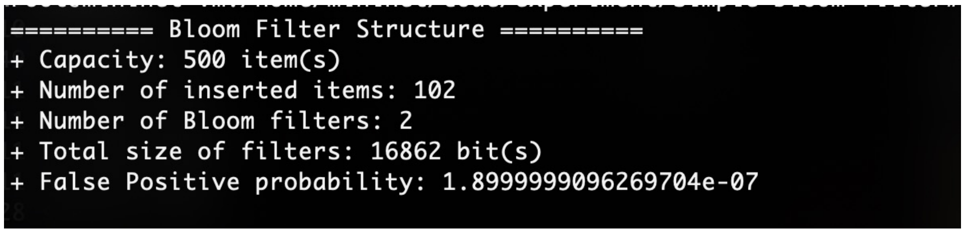


```
" is PROBABLY IN the filter i.e. has passed through s6-eth1.pcap0-packet-data.txt
"10.0.0.1,10.0.0.4,66,11309
" is PROBABLY IN the filter i.e. has passed through s6-eth1.pcap0-packet-data.txt
"10.0.0.4,10.0.0.1,65226,11309
" is PROBABLY IN the filter i.e. has passed through s6-eth1.pcap0-packet-data.txt
"10.0.0.1,10.0.0.4,66,11309
" is PROBABLY IN the filter i.e. has passed through s6-eth1.pcap0-packet-data.txt
"10.0.0.4,10.0.0.1,442,11309
" is PROBABLY IN the filter i.e. has passed through s6-eth1.pcap0-packet-data.txt
"10.0.0.1,10.0.0.4,66,11309
" is PROBABLY IN the filter i.e. has passed through s6-eth1.pcap0-packet-data.txt
"10.0.0.4,10.0.0.1,65226,11309
" is PROBABLY IN the filter i.e. has passed through s6-eth1.pcap0-packet-data.txt
"10.0.0.1,10.0.0.4,66,11309
" is PROBABLY IN the filter i.e. has passed through s6-eth1.pcap0-packet-data.txt
"10.0.0.4,10.0.0.1,442,11309
" is PROBABLY IN the filter i.e. has passed through s6-eth1.pcap0-packet-data.txt
"10.0.0.1,10.0.0.4,66,11309
" is PROBABLY IN the filter i.e. has passed through s6-eth1.pcap0-packet-data.txt
"10.0.0.4,10.0.0.1,14988,11309
" is PROBABLY IN the filter i.e. has passed through s6-eth1.pcap0-packet-data.txt
"19039
" is DEFINITELY NOT IN the filter as expected and has not passed through s6-eth1.pcap0-packet-data.txt
"192.168.0.1,192.168.0.2,94,19039
" is DEFINITELY NOT IN the filter as expected and has not passed through s6-eth1.pcap0-packet-data.txt
"192.168.0.2,192.232.0.1,82,19039
" is DEFINITELY NOT IN the filter as expected and has not passed through s6-eth1.pcap0-packet-data.txt
"192.168.0.1,192.168.0.2,94,19039
" is DEFINITELY NOT IN the filter as expected and has not passed through s6-eth1.pcap0-packet-data.txt
"192.168.0.2,192.168.0.1,82,19039
" is DEFINITELY NOT IN the filter as expected and has not passed through s6-eth1.pcap0-packet-data.txt
"192.168.0.2,192.168.0.1,82,19039
" is DEFINITELY NOT IN the filter as expected and has not passed through s6-eth1.pcap0-packet-data.txt
"192.168.0.1,192.168.0.2,94,19039
" is DEFINITELY NOT IN the filter as expected and has not passed through s6-eth1.pcap0-packet-data.txt
"192.168.0.2,192.168.0.1,82,19039
" is DEFINITELY NOT IN the filter as expected and has not passed through s6-eth1.pcap0-packet-data.txt
"19039
" is DEFINITELY NOT IN the filter as expected and has not passed through s6-eth1.pcap0-packet-data.txt
"192.168.0.2,192.168.0.1,82,19039
" is DEFINITELY NOT IN the filter as expected and has not passed through s6-eth1.pcap0-packet-data.txt
"19039
" is DEFINITELY NOT IN the filter as expected and has not passed through s6-eth1.pcap0-packet-data.txt
"19039
" is DEFINITELY NOT IN the filter as expected and has not passed through s6-eth1.pcap0-packet-data.txt
"192.168.0.1,192.168.0.2,94,19039
```

Figure 16: Querying the Elements

and ‘Figure 17’ shows the bloom filter structure designed for our experiment.

This successfully demonstrates that we can optimize forensic data availability and retention of SDN forensic logs by using bloom filter in the proposed system.



```
========== Bloom Filter Structure ==========
+ Capacity: 500 item(s)
+ Number of inserted items: 102
+ Number of Bloom filters: 2
+ Total size of filters: 16862 bit(s)
+ False Positive probability: 1.8999999096269704e-07
```

Figure 17: Bloom Filter Sample Structure of Experiment

Code Snippet of Bloom Filter integration & excerpt query and excerpt query in text raw parsed data. Both code snippets are mentioned below.

Code Snippet of string search in text format raw parsed data.

```python
from bloomfilter import BloomFilter
import os
import time
import sys

# Define the directory containing the text files
directory = './final_dir/'

# Define the string to search for
arg1 = sys.argv[1]

search_string = arg1

# Create an empty list to store the file names
file_list = []

# Loop through the files in the directory
for filename in os.listdir(directory):
    # Check if the file is a text file
    if filename.endswith('.txt'):
        # Append the file name to the list
        file_list.append(os.path.join(directory, filename))

# Create a variable to store the total time taken
```

```python
total_time = 0

# Loop through the files in the list
for file_name in file_list:
    # Open the file
    with open(file_name, 'r') as file:
        # Read the contents of the file
        contents = file.read()

        # Start the timer
        start_time = time.time()

        # Search for the string in the file
        if search_string in contents:
            print(f'{search_string} found in {file_name}')
        else:
            print(f'{search_string} not found in {file_name}')

        # Stop the timer and calculate the elapsed time
        elapsed_time = time.time() - start_time
       # end_time = time.time()
        # Add the elapsed time to the total time
        total_time += elapsed_time
        #total_time = end_time - start_time

# Print the total time taken
print(f'Total time taken: {total_time:.10f} seconds')
```

Code snippet of BF integration and excerpt query in *.bin* files

```python
import os
import hashlib
import argparse
import time

class BloomFilter:
    def __init__(self, size, hash_count):
        self.size = size
        self.hash_count = hash_count
        self.bit_array = [0] * size

    def add(self, string):
        for seed in range(self.hash_count):
            result = hash(string + str(seed)) % self.size
            self.bit_array[result] = 1

    def __contains__(self, string):
        for seed in range(self.hash_count):
            result = hash(string + str(seed)) % self.size
            if self.bit_array[result] == 0:
                return False
        return True

def create_bloom_filter(file_path):
    # Read the contents of the text file
    with open(file_path, "r") as file:
        lines = file.readlines()

    # Create a new bloom filter with a size proportional to the number of lines in the file
    bloom_filter = BloomFilter(len(lines) * 10, 5)

    # Add each line of the file to the bloom filter
    for line in lines:
        bloom_filter.add(line.strip())

    # Write the bloom filter to a binary file
    bloom_filter_file = os.path.splitext(file_path)[0] + ".bin"
    with open(bloom_filter_file, "wb") as file:
        for bit in bloom_filter.bit_array:
            file.write(bit.to_bytes(1, byteorder="big"))

    print(f"Bloom filter created for {file_path} -> {bloom_filter_file}")

if __name__ == "__main__":
    # Create an argument parser to get the search string and directory path from the command line
    parser = argparse.ArgumentParser(description="Create Bloom filter binary files and search for a string")
    parser.add_argument("search_string", help="the string to search for")
    parser.add_argument("directory_path", help="the path to the directory containing the text files")
    args = parser.parse_args()

    # Create a bloom filter binary file for each text file in the directory
    for filename in os.listdir(args.directory_path):
        if filename.endswith(".txt"):
            file_path = os.path.join(args.directory_path, filename)
            create_bloom_filter(file_path)

    start_time = time.time()
#     print(start_time)
    # Search for the string in each bloom filter binary file in the directory
    for filename in os.listdir(args.directory_path):
        if filename.endswith(".bin"):
            file_path = os.path.join(args.directory_path, filename)

            # Read the contents of the bloom filter binary file
            with open(file_path, "rb") as file:
                bit_array = []
                byte = file.read(1)
                while byte:
                    bit_array.append(int.from_bytes(byte, byteorder="big"))
                    byte = file.read(1)

            # Initialize the bloom filter with the same size and hash count as the original bloom filter
            bloom_filter = BloomFilter(len(bit_array), 5)
            bloom_filter.bit_array = bit_array

            # Check if the search string is in the bloom filter
            if args.search_string in bloom_filter:
             #   end_time = time.time()
            #    time_taken = end_time - start_time
                print(f"'{args.search_string}' may be present in {file_path}") # (time taken: {time_taken:.10f} seconds)")
            else:
             #   end_time = time.time()
             #   time_taken = end_time - start_time
                print(f"'{args.search_string}' is not present in {file_path}") # (time taken: {time_taken:.10f} seconds)")
    end_time = time.time()
#     print(end_time)
    time_taken = end_time - start_time
    print(f"Total time taken: {time_taken:.10f} seconds")
```

Result of experiment while search in raw text

```
Varuns-M1pro:experiments_final varun$ python 2_final_bloom_raw_search.py 10.0.0.1,10.0.0.4,74,11309
10.0.0.1,10.0.0.4,74,11309 not found in ./final_dir/test1.txt
10.0.0.1,10.0.0.4,74,11309 not found in ./final_dir/sample3.pcap0-packet-data.txt
10.0.0.1,10.0.0.4,74,11309 not found in ./final_dir/s1-eth1.pcap0-packet-data.txt
10.0.0.1,10.0.0.4,74,11309 found in ./final_dir/s6-eth1.pcap0-packet-data.txt
```

```
10.0.0.1,10.0.0.4,74,11309 not found in ./final_dir/s3-eth1.
    pcap0-packet-data.txt
10.0.0.1,10.0.0.4,74,11309 not found in ./final_dir/sample.
    pcap0-packet-data.txt
10.0.0.1,10.0.0.4,74,11309 not found in ./final_dir/sample2.
    pcap0-packet-data.txt
10.0.0.1,10.0.0.4,74,11309 not found in ./final_dir/sample4.
    pcap0-packet-data.txt
10.0.0.1,10.0.0.4,74,11309 not found in ./final_dir/s2-eth1.
    pcap0-packet-data.txt
10.0.0.1,10.0.0.4,74,11309 not found in ./final_dir/h4-eth0.
    pcap0-packet-data.txt
```

Result of experiment while search in binary files (using BF)

```

Varuns-M1pro:experiments_final varun$ python 1
    _final_bloom_search_time.py 10.0.0.1,10.0.0.4,74,11309
    final_dir
Bloom filter created for final_dir/test1.txt -> final_dir/test1
    .bin
Bloom filter created for final_dir/sample3.pcap0-packet-data.
    txt -> final_dir/sample3.pcap0-packet-data.bin
Bloom filter created for final_dir/s1-eth1.pcap0-packet-data.
    txt -> final_dir/s1-eth1.pcap0-packet-data.bin
Bloom filter created for final_dir/s6-eth1.pcap0-packet-data.
    txt -> final_dir/s6-eth1.pcap0-packet-data.bin
Bloom filter created for final_dir/s3-eth1.pcap0-packet-data.
    txt -> final_dir/s3-eth1.pcap0-packet-data.bin
Bloom filter created for final_dir/sample.pcap0-packet-data.txt
     -> final_dir/sample.pcap0-packet-data.bin
Bloom filter created for final_dir/sample2.pcap0-packet-data.
    txt -> final_dir/sample2.pcap0-packet-data.bin
Bloom filter created for final_dir/sample4.pcap0-packet-data.
    txt -> final_dir/sample4.pcap0-packet-data.bin
Bloom filter created for final_dir/s2-eth1.pcap0-packet-data.
    txt -> final_dir/s2-eth1.pcap0-packet-data.bin
Bloom filter created for final_dir/h4-eth0.pcap0-packet-data.
    txt -> final_dir/h4-eth0.pcap0-packet-data.bin
'10.0.0.1,10.0.0.4,74,11309' is not present in final_dir/test1.
    bin
'10.0.0.1,10.0.0.4,74,11309' is not present in final_dir/
    sample3.pcap0-packet-data.bin
'10.0.0.1,10.0.0.4,74,11309' is not present in final_dir/s1-
    eth1.pcap0-packet-data.bin
'10.0.0.1,10.0.0.4,74,11309' may be present in final_dir/s6-
    eth1.pcap0-packet-data.bin
'10.0.0.1,10.0.0.4,74,11309' is not present in final_dir/s3-
    eth1.pcap0-packet-data.bin
```

```
'10.0.0.1,10.0.0.4,74,11309' is not present in final_dir/sample
    .pcap0-packet-data.bin
'10.0.0.1,10.0.0.4,74,11309' is not present in final_dir/
    sample2.pcap0-packet-data.bin
'10.0.0.1,10.0.0.4,74,11309' is not present in final_dir/
    sample4.pcap0-packet-data.bin
'10.0.0.1,10.0.0.4,74,11309' is not present in final_dir/s2-
    eth1.pcap0-packet-data.bin
'10.0.0.1,10.0.0.4,74,11309' is not present in final_dir/h4-
    eth0.pcap0-packet-data.bin
```

- Our designed system optimize large SDN forensic data sets by reducing its space complexity as well as faster querying by using bloom filters.
- In addition to the current logging strategies, network packet data (parsed) older than system retention period is integrated in bloom filter (one bloom filter is made for each device's data) and used as forensic data.
- Query the bloom filter later to search whether a particular data was seen before or not.
- It offers a great advantage in retaining the forensic logs for a much longer duration (at least 5 times more forensic data).
- By suitably modifying the value of disk size and bloom filter we may alter the duration of forensic data retention as per the operational need.
- In our designed system, we traced the path of the network packet while downloading a sample file from a server into the client in a simulated SDN with hybrid network with mesh topology using mininet.
- Parse each captured tcpdump (pcap) file and fetch important information of each packet like source of the packet, destination of the packet, size of the packet and total number of packets.
- Integrate packets data in bloom filter creating "bin" files for respective devices, then query to find if the element(packet data) exists in the bloom filter(device) or not to conclude the results.
- We repeated the experiment to trace the path of a packet using the raw log files and then searching (excerpt query) with the data integrated in bloom filter. Both result the same path. Experiments outputs are shown above.

- Data integrated with Bloom filter requires significantly less space, and the search time complexity is also lower, with a complexity of O(1) compared to O(m+n).

According to the design, once data is integrated into a bloom filter, it becomes irreversible and unrecoverable, but it has the potential to prove that an element does not exist in a set. Our approach utilizes the architecture of bloom filter to provide acceptable evidence with significantly less data storage requirements. While it may not offer exact log details in the event of an incident, it can offer potential evidence in identifying the attack path if older forensic data is integrated into our proposed design instead of being purged.

Assuming an SDN system generates 100 GB of logs per month and the available disk space in a traditional system is 500 GB, we cannot store any data older than five months. Our proposed system, with a disk size of 510 GB (10 GB more), can store five months of raw data in the traditional format and around five months of forensic hashed data that is added in bloom filter. By adjusting the disk size and bloom filter values, we can modify the duration of forensic data retention to meet operational needs.

5.2 Cybersecurity Analysis of Designed System

- Bloom filters are data structures that are commonly used in computer science to efficiently test the membership of an element in a set. Bloom filters are often implemented as binary files, which are susceptible to cybersecurity threats.
- One of the main cybersecurity risks associated with bloom filter binary files is the possibility of a malicious actor tampering with the contents of the file. This could tamper the search results in bloom filter integrated files, which can mislead the packet traverse path.
- To mitigate these risks, it is important to implement strong security measures when handling bloom filter binary files. This may include access controls, and monitoring for any unauthorized access or modifications to the file. Additionally, it is important to ensure that the bloom filter is designed to resist common attacks, such as hash collision attacks, which could be used to manipulate the contents of the filter.

- Overall, cybersecurity is an important consideration when working with bloom filter binary files, and appropriate security measures should be taken to protect against potential threats.

Referring to the flow chart shown in 'Figure 4' of our designed system, To begin with, data collection and storage is the initial step. Following that, we parse the gathered data (*.pcap* files) to create *.txt* files and these files are integrated into the BF to create binary (*.bin*) files, the *.txt* files are purged at the same time and only the binary (*.bin*) files are retained as optimized forensic data in the system, as the second step. Third step is optimized forensic data storage. Finally, we perform the excerpt query to verify the existence of a subset in the binary files generated to track the path traversed by the packet.

Our designed system processes and transforms data older than the stipulated retention period into optimized forensic data before purging the original data from traditional storage systems. Below, we will delve into the various cybersecurity threats that pose a risk to this process.

Logs Collection: The logs are collected centrally into traditional storage systems. This could involve configuring the SDN controllers to forward logs to a central log server or using a SIEM (Security Information and Event Management) tool to aggregate and analyze the logs. It is a software solution that combines security information management (SIM) and security event management (SEM) capabilities to provide real-time analysis of security alerts generated by network hardware and applications. SIEM tools are used to collect and analyze security-related data from multiple sources within an organization's IT infrastructure. SIEM solutions typically use a combination of rule-based and statistical analysis to detect and respond to security events.

Logs / data storage is exposed to the potential attacks such as unauthorized access attempts or changes to device configurations. Based on the analysis of the SDN logs, we may need to implement additional security controls to prevent future incidents. This could include deploying intrusion detection systems, tightening access controls, or implementing network segmentation to limit the impact of an incident. It is important to continuously monitor and evaluate the SDN logs to identify new security threats and assess the effectiveness of existing security controls. This could involve regularly reviewing log files, conducting vulnerability scans, and performing penetration testing to identify potential weaknesses in the network. It is an ongoing process that requires continuous monitoring and evaluation to effectively mitigate potential security threats.

Some of the security risks that may arise during data transmission from various devices (routers or switches) in SDN system to storage servers include interception, eavesdropping, and data manipulation. Attackers may intercept data packets as they travel over the network, allowing them to gain unauthorized access to sensitive information. Eavesdropping involves monitoring network traffic to collect information about the communication between two devices. Data manipulation occurs when an attacker alters the data being transmitted to gain unauthorized access or cause harm. In our system, since the data is hashed the information is secured and not exposed to attacker to alter.

To mitigate these risks, designed system is required to ensure the confidentiality, integrity, and availability of the data, hence various security measures can be implemented, such as encryption, firewalls, and intrusion detection systems. Encryption involves encoding the data in such a way that only authorized parties can access and read it. Firewalls are used to prevent unauthorized access to the network, while intrusion detection systems monitor network traffic for signs of suspicious activity and alert security personnel when necessary. Data transmission in the designed system is done using scp (Secured Copy : SCP-294 version)

Data Storage System: Network logs may contain sensitive information, such as user credentials, IP addresses, and device configurations. To comply with integrity in our system, logs are encrypted (integrated in BF) during storage and transmission, and access should be restricted to authorized personnel only. To maintain integrity, logs should be protected from unauthorized modification or deletion. The minimum possible session timeout should be adhered to by users when accessing the system. Network logs may be subject to legal or regulatory requirements for retention, and may need to be kept for extended periods of time. To maintain compliance, logs are stored in a secure and tamper-proof storage solution that supports retention policies and secure deletion after the data is encrypted using BF. It is necessary to ensure Full Disk Encryption (FDE) is applied when storing binary files.

Backup and recovery procedures should be established to ensure that data can be restored in case of data loss or a security incident. The network infrastructure that connects the storage servers should be secured with firewalls, intrusion detection systems, and other network security measures.

Excerpt Query:

Excerpt BF query is a technique used to search for specific data subsets in large data sets using a compressed data structure called a Bloom filter. Excerpt BF query can be vulnerable to several cybersecurity exposures, which can potentially compromise the confidentiality, integrity, and availability of sensitive data subsets. Some of the potential security exposure are:

Unauthorized Access - Unauthorized access to the proposed system can potentially compromise the integrity of the data sets stored in it. We ensure unauthorized access is barred.

Integrity - The integrity of the data sets stored in the BF should be maintained to prevent unauthorized modifications or tampering.

Network Security - The network infrastructure that connects to the optimized data storage should be secured with firewalls, intrusion detection systems, and other network security measures.

Physical Security - Physical security measures are implemented to protect the optimized data from physical threats, such as theft, vandalism, or environmental damage.

Packet Traversed Path:

Achieving this result is critical for the designed system, and to do so, it is essential to address all the cybersecurity vulnerabilities mentioned above. Any compromise, manipulation or alteration of the data can lead to inaccurate or invalid results that cannot be used as evidence.

6 Conclusions

6.1 Thesis Summary and Future Work

After analyzing our experimental results, we can confidently state that the proposed system has the capability to store at least ten times more forensic data compared to existing systems. By implementing our system, we have found that we no longer need to delete all data from storage systems that are older than a specific retention period. In fact, we have noticed an improvement in forensic data availability, which could potentially serve as evidence in a court of law to analyze the path of the packet.

- Post evaluation of our experimental results, it is safe to conclude that the proposed system can withhold significantly more forensic data as compared to existing systems.
- We have observed that utilizing the proposed system eliminates the need to delete all data from storage systems that are past their retention period. Rather, this approach improves forensic data availability, which could be admissible as potential evidence in a court of law for analyzing the path taken by the packet.
- Cybersecurity: The use of hashing is deliberate, as it helps to ensure that data stored on servers and cloud storage systems cannot be read by hackers. Therefore, we can infer that our optimized data is secure / unreadable unless the binary files are tampered.
- Our research work is published in International Conference ICOIN 2023.
- `https://www.computer.org/csdl/proceedings-article/icoin/2023/10048908/1KYsMyFrrFK`

As part of future work, we can experiment with optimizing the bloom filter application by utilizing enhanced multiple layered hashing, counting bloom filter, and cuckoo filter. This could lead to the design of a more agile and robust system capable of automatically handling logs and rolling files in an organized manner for various types of SDN traffic.

The proposed system can be improved and adapted to other network systems that generate large volumes of logs. This expands the potential for optimizing the availability of forensic data in various existing systems. Further experiments could be conducted to demonstrate and compare time complexity across different systems.

References

[1] Q. Waseem, S. S. Alshamrani, K. Nisar, W. I. S. Wan Din, and A. S. Alghamdi, "Future Technology: Software-Defined Network (SDN) Forensic," *Symmetry*, vol. 13, no. 5, p. 767, Apr. 2021, doi: 10.3390/sym13050767. [Online]. Available: http://dx.doi.org/10.3390/sym13050767

[2] S. S. B. Renukuntla and S. Rawat, "Optimization of excerpt query process for Packet Attribution System," *2014 10th International Conference on Information Assurance and Security*, 2014, pp. 41-46, doi: 10.1109/ISIAS.2014.7064618.

[3] Donghao Zhou, Zheng Yan, Yulong Fu, Zhen Yao, "A survey on network data collection," *Journal of Network and Computer Applications*, Volume 116, 2018, Pages 9-23, ISSN 1084-8045, https://doi.org/10.1016/j.jnca.2018.05.004.

[4] B. Celesova, J. Val'ko, R. Grezo and P. Helebrandt, "Enhancing security of SDN focusing on control plane and data plane," *2019 7th International Symposium on Digital Forensics and Security (ISDFS)*, 2019, pp. 1-6, doi: 10.1109/ISDFS.2019.8757542.

[5] V. Varadharajan, K. Karmakar, U. Tupakula and M. Hitchens, "A Policy-Based Security Architecture for Software-Defined Networks," *in IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 897-912, April 2019, doi: 10.1109/TIFS.2018.2868220.

[6] I. Ahmad, S. Namal, M. Ylianttila and A. Gurtov, "Security in Software Defined Networks: A Survey," *in IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2317-2346, Fourthquarter 2015, doi: 10.1109/COMST.2015.2474118.

[7] D. Kumar and A. Girdhar, "Network monitoring and analysis along with comparative study of honeypots," *2017 International Conference on Intelligent Sustainable Systems (ICISS)*, 2017, pp. 736-739, doi: 10.1109/ISS1.2017.8389270.

[8] J. S. Marean, M. Losavio and I. Imam, "A Research Configuration for a Digital Network Forensic Lab," *2008 Third International Workshop on Systematic Approaches to Digital Forensic Engineering*, 2008, pp. 141-142, doi: 10.1109/SADFE.2008.23.

[9] R. Lu and L. Li, "Research on Forensic Model of Online Social Network," *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, 2019, pp. 116-119, doi: 10.1109/ICCCBDA.2019.8725746.

[10] B. Siniarski, C. Olariu, P. Perry, T. Parsons and J. Murphy, "Real-time monitoring of SDN networks using non-invasive cloud-based logging platforms," *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2016, pp. 1-6, doi: 10.1109/PIMRC.2016.7794973.

[11] A. Asaduzzaman, A. Almohaimeed and K. K. Chidella, "Shared Entry Logger to Eliminate Duplicate Requests to SDN Controller," *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, 2019, pp. 0579-0584, doi: 10.1109/CCWC.2019.8666595.

[12] P. T. Duy, H. Do Hoang, D. T. Thu Hien, N. Ba Khanh, V. Pham, "SDNLog-Foren: Ensuring the Integrity and Tamper Resistance of Log Files for SDN Forensics using Blockchain," *2019 6th NAFOSTED Conference on Information and Computer Science (NICS)*, 2019, pp. 416-421, doi: 10.1109/NICS48868.2019.9023852.

[13] S. Locke, H. Li, T. -H. P. Chen, W. Shang, W. Liu, "LogAssist: Assisting Log Analysis Through Log Summarization," *in IEEE Transactions on Software Engineering*, doi: 10.1109/TSE.2021.3083715.

[14] Patil, Rachana Y., and Satish R. Devane, "Network Forensic Investigation Protocol to Identify True Origin of Cyber Crime," *Journal of King Saud University-Computer and Information Sciences (2019).*

[15] Kotsiuba, Igor, at el., "Basic Forensic Procedures for Cyber Crime Investigation in Smart Grid Networks," *In 2019 IEEE International Conference on Big Data (Big Data)*, pp. 4255-4264. IEEE, 2019.

[16] G. Palmer, "A Road Map for Digital Forensic Research," *Technical Report (DTR-T001-01) for Digital Forensic Research Workshop (DFRWS)*, New York, 2001.

[17] Montasari, Reza, Pekka Peltola, and David Evans, "Integrated computer forensics investigation process model (ICFIPM) for computer crime investigations." *In International Conference on Global Security, Safety, and Sustainability*, pp. 83-95. Springer, Cham, 2015.

[18] A. Mishra, C. Singh, A. Dwivedi, D. Singh and A. K. Biswal, "Network Forensics: An approach towards detecting Cyber Crime," *2021 International Conference in Advances in Power, Signal, and Information Technology (APSIT)*, 2021, pp. 1-6, doi: 10.1109/APSIT52773.2021.9641399.

[19] Wikipedia contributors. (2021, August 28). *Bloom filter. In Wikipedia, The Free Encyclopedia.* Retrieved 19:23, September 24, 2021, from `https://en.wikipedia.org/w/index.php?title=Bloom_filter&oldid=1041007243`

[20] Abomhara, M., and Koien, G. M., "(2015) Forensic investigation of cyber-attacks: A systematic review," *Journal of Network and Computer Applications, 50, 45-65.*

[21] Alazab, M., Venkatraman, S., and Slay, J., "(2017) An overview of cybercrime investigation: A systematic literature review," *Journal of Network and Computer Applications, 82, 30-47.*

[22] Baggili, I., Breitinger, F., and Marrington, A., "(2012) Investigating network forensic automatic event reconstruction through correlation analysis," *Digital Investigation, 9(1), 30-41.*

[23] Bhatt, M., and Kaur, P., "(2018). Network forensics: A comprehensive review," *Computers & Security, 79, 98-120.*

[24] Rathi, M., and Kaur, P., "(2018). Network forensics: A systematic review," *Journal of Ambient Intelligence and Humanized Computing, 9(2), 461-476.*

[25] Richard, J., and Le-Khac, N. A., "(2019). Network forensics: An overview," *Journal of Information Security and Applications, 47, 77-87.*

[26] Singh, D., Singh, N., and Verma, A. K., "(2018). A comprehensive review of network forensics techniques and tools," *Journal of Ambient Intelligence and Humanized Computing, 9(2), 389-403.*

[27] Firdausi, I. K., and Kartit, S., "(2017). A survey on packet attribution techniques for network forensics," *International Journal of Computer Applications, 160(9), 35-42.*

[28] He, X., Zhu, S., and Zhang, Y., "(2019). Lightweight packet attribution based on deep learning for encrypted network traffic," *IEEE Access, 7, 148836-148848.*

[29] Huang, T., and Lee, K., "(2019). An efficient query processing approach for forensic analysis of big network data," *Journal of Ambient Intelligence and Humanized Computing, 10(2), 757-767.*

[30] Javadi, H. H., and Rahmani, A. M., "(2016). A survey on network forensic techniques based on packet sniffing," *International Journal of Computer Science and Mobile Computing, 5(2), 184-189.*

[31] Shen, Y., and Wu, S., "(2016). A query processing approach for packet attribution in network forensics," *IEEE Transactions on Information Forensics and Security, 11(11), 2453-2466.*

[32] Wang, J., Zhang, J., and Li, X., "(2018). An efficient packet attribution scheme based on multi-level Bloom filter," *International Journal of Communication Systems, 31(11), e3631.*

[33] Al-azzawi, A., and Abdullah, A., "(2019). A review of network data collection and monitoring techniques," *IEEE Access, 7, 41612-41625.*

[34] Alharbi, F., and Fortino, G., "(2021). A survey on data collection techniques and protocols in wireless sensor networks," *Journal of Network and Computer Applications, 172, 102913.*

[35] Bhattacharyya, D., and De, S., "(2016). A review on data collection schemes in wireless sensor networks," *International Journal of Computer Applications, 145(1), 1-5.*

[36] Ding, L., Luo, X., Mao, Y., and Zhao, L., "(2020). A survey on data collection techniques in Internet of Things," *IEEE Access, 8, 164663-164676.*

[37] Wang, X., Li, X., Zhu, S., and Zhang, X., "(2019). A survey on data collection and analysis for mobile crowdsensing," *IEEE Access, 7, 72038-72060.*

[38] Yang, Z., He, J., Li, C., and Gao, F., "(2019). A survey of big data collection, storage, and analytics in smart grids," *Journal of Network and Computer Applications, 131, 1-16.*