ABSTRACT

NEURAL NETWORK-BASED CROSSFIRE ATTACK DETECTION IN SDN-ENABLED CELLULAR NETWORKS

by Nicholas Perry

In today's networked world, cybersecurity threats pose a significant challenge to the integrity and reliability of communication networks. One such threat is the crossfire attack, where adversaries exploit network vulnerabilities by injecting malicious packets into traffic flows. To address this, we present a novel crossfire detection scheme that solely inspects packet headers, reducing the computational overhead associated with packet inspection. Our proposed detection scheme includes both analysis of variance (ANOVA) and neural networks to identify anomalous packet behaviors indicative of crossfire attacks. To evaluate the effectiveness of our approach, we conducted experiments on a real ATT backbone topology, simulating a crossfire attack in the Mininet simulation environment. The results demonstrate that our detection scheme achieves an accuracy of 95.3% in detecting adversarial packets, effectively mitigating the crossfire threat. Furthermore, we introduce a traffic optimization model to adapt routing decisions in response to crossfire or link flooding attacks. Leveraging the detection scheme's real-time analysis, our optimization model dynamically alters routing paths to minimize the impact of attacks on network performance. Overall, our research presents an innovative and comprehensive framework that combines efficient crossfire detection using packet headers, high-accuracy detection using ANOVA and neural networks, and an adaptive traffic optimization model.

NEURAL NETWORK-BASED CROSSFIRE ATTACK DETECTION IN SDN-ENABLED CELLULAR NETWORKS

A Thesis

Submitted to the
Faculty of Miami University
in partial fulfillment of
the requirements for the degree of
Master of Science
by
Nicholas Perry
Miami University
Oxford, Ohio
2023

Advisor: Dr. Suman Bhunia
Reader: Dr. Vaskar Raychoudhary
Reader: Dr. Daniela Inclezan

This Thesis titled

NEURAL NETWORK-BASED CROSSFIRE ATTACK DETECTION IN SDN-ENABLED CELLULAR NETWORKS

by

Nicholas Perry

has been approved for publication by

The College of Engineering and Computing

and

The Department of Computer Science & Software Engineering

Dr. Suman Bhunia

Dr. Vaskar Raychoudhary

Dr. Daniela Inclezan

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to express my heartfelt gratitude to my esteemed committee members, Dr. Bhunia, Dr. Raychoudhary, and Dr. Inclezan, for their invaluable guidance and unwavering support throughout the course of my thesis project. Their expertise, insightful feedback, and encouragement have been instrumental in shaping the outcome of this research endeavor.

In particular, I would like to extend a special thanks to Dr. Bhunia for his exceptional role as my advisor. Despite the demands of teaching classes and assisting other advisees, Dr. Bhunia has been an indispensable part of this project. His dedication, patience, and willingness to share his knowledge have not only helped me navigate the complexities of research but also enabled me to develop crucial skills in the field. I am truly grateful for his mentorship and the valuable lessons he imparted.

I would also like to recognize Dr. Salman for his invaluable assistance as an external research advisor. His expertise in machine learning, traffic optimization, and Gurobi had a profound impact on the quality and outcomes of this thesis. His guidance and insights were invaluable in shaping the direction of my research and enhancing the overall depth of the project.

I am also grateful for my friends, classmates, and the entire academic community for providing me with an environment conducive to learning and growth. The resources and opportunities made available to me have been pivotal in the successful completion of this thesis.

Last but certainly not least, I want to extend my deepest gratitude to my parents, Jim and Lynn, and my siblings, Olivia and Lauren. Their unwavering support and boundless love have been an incredible source of strength and encouragement throughout this challenging journey. Their continued belief in my abilities has been a constant source of motivation, pushing me to overcome obstacles and strive for excellence.

To everyone who has played a part, big or small, in the realization of this thesis project, I extend my heartfelt thanks. Your contributions have been invaluable, and I am truly honored and grateful for your support.

Chapter 1

Introduction

As the internet continues to evolve and become more essential to daily lives than ever, there is a growing population looking to destroy it. Denial of service (DoS) attacks have been a threat to the internet for years and continue to cause issues even in today's networks. The continued improvement and introduction of internet-of-things (IoT) devices have pushed mobile carriers to update their existing infrastructure to support the increased number of devices. Due to the increased load on the network, a focus on Mobile Edge Computing (MEC) has increased. These MEC servers keep data from going across the center of the network and instead process the data near the edge of the network. A newer type of DoS attack, known as the crossfire attack has plagued the internet. This type of attack is extremely difficult to detect and can be lethal to networks if executed correctly. Software-defined networks (SDNs) have been discussed as a promising mitigation technology that could detect DoS attacks. While SDNs are helpful, they are not perfect and open up a different set of vulnerabilities to exploit. The end of DDoS attacks is not in sight, and therefore many different approaches must be taken to protect critical servers.

About 25 billion devices are currently interconnected and by 2025, 60 billion devices are expected to be connected [1]. As the Internet continues to develop, traditional devices are becoming "smart", meaning that they are connected to the Internet. The term "Internet of Things"(IoT) was coined in 1999 by Kevin Ashton which describes a global network of interconnected devices [1]. The motivation for IoT devices is to create large "smart" systems [2]. Technological advancements are the reason for the increased motivation to link devices together [1]. IoT devices take many forms and almost any traditional device can be converted to a smart device. Some examples of IoT devices include smart plugs, smart washing machines, smart lights, smart refrigerators, etc.

The crossfire attack is a type of DoS attack that is more difficult to detect. This attack uses many devices across large geographic regions to send low-intensity requests across the network to various servers on the other side of the network. This is especially problematic with the advent of IoT and IoMT, because even though these devices often have extremely limited processing power, these devices can be compromised and used since the attack only requires low-intensity attacks to be sent from any given device.

1.1 Motivation

Denial of Service attacks are a recurring issue in the world of networks. The first distributed denial of service attack took place in 1999, and over twenty years later, DoS and DDoS attacks are still a large threat. With an increase of internet-of-things devices, availability attacks using botnets have become common. An effective way to detect and mitigate such attacks is imperative as the internet becomes even more of a critical infrastructure than it already is. Many essential institutions rely on having available servers such as hospitals, banks, law enforcement, and government. If servers were compromised and made unavailable, the consequences could be severe. Being able to detect these attacks is the first step to combating them. Without a novel detection scheme, crossfire attacks and similar link-flooding attacks will continue to be lethal.

1.2 Proposed Crossfire Avoidance

Our proposed crossfire avoidance framework involves the process of discerning patterns within collected data to effectively differentiate between malicious attack traffic and legitimate network activity. Various techniques and statistical tests are employed to uncover unique characteristics associated with attack traffic, enabling the development of novel defense mechanisms. Furthermore, network optimization plays a crucial role in enhancing overall network performance and resilience. To achieve this, a deep learning optimization library is utilized to augment network bandwidth throughout the entire network infrastructure. By leveraging network topology, the optimization library generates diverse network scenarios, particularly in Software-Defined Networking (SDN) frameworks.

Since crossfire attacks are so lethal, it is important to detect when they are occurring as soon as possible. Therefore, using a software-defined network (SDN) is ideal so that a holistic view of the network can be obtained. The use of SDN is proposed as an ideal approach to obtain a holistic view of the network. In an SDN, the control plane is decoupled from the data plane, allowing centralized control and management of the network. The SDN architecture consists of OpenFlow switches that forward network traffic based on instructions received from the SDN controller. Each OpenFlow switch in the network reports the packet headers of incoming packets to the SDN controller. Packet headers contain important information such as source and destination IP addresses, transport protocol, port numbers, etc. By inspecting these headers, the SDN controller can gain visibility into the network and analyze the characteristics of the packets flowing through it. To defend against crossfire attacks, a traffic classification model is proposed to determine whether a packet is adversarial (part of the attack) or benign. This model is implemented on the SDN controller. It leverages machine learning or rule-based techniques to analyze the packet headers and make an informed decision about the nature of the packet. The SDN controller sends the packet headers to a cloud-based IDS for further analysis. The IDS hosts the proposed traffic classification model, which evaluates the received packet headers and determines if they correspond to an adversarial or benign packet. The IDS is equipped with computational resources and advanced analysis techniques to perform this task effectively. Once the IDS determines a packet is adversarial, the SDN controller instructs the respective OpenFlow switch to drop the offending packet(s) from the processing pipeline. By discarding the malicious packets, congestion on the network can be reduced, preventing the crossfire attack from spreading further. In summary, this defense mechanism combines the capabilities of SDN, packet header inspection, a traffic classification model, and a cloud-based IDS to detect and mitigate crossfire attacks. By inspecting packet headers, identifying adversarial packets, and dropping them in real time, the mechanism helps protect the network from the detrimental effects of crossfire attacks, minimizing potential damage and maintaining network performance. Figure 1.1

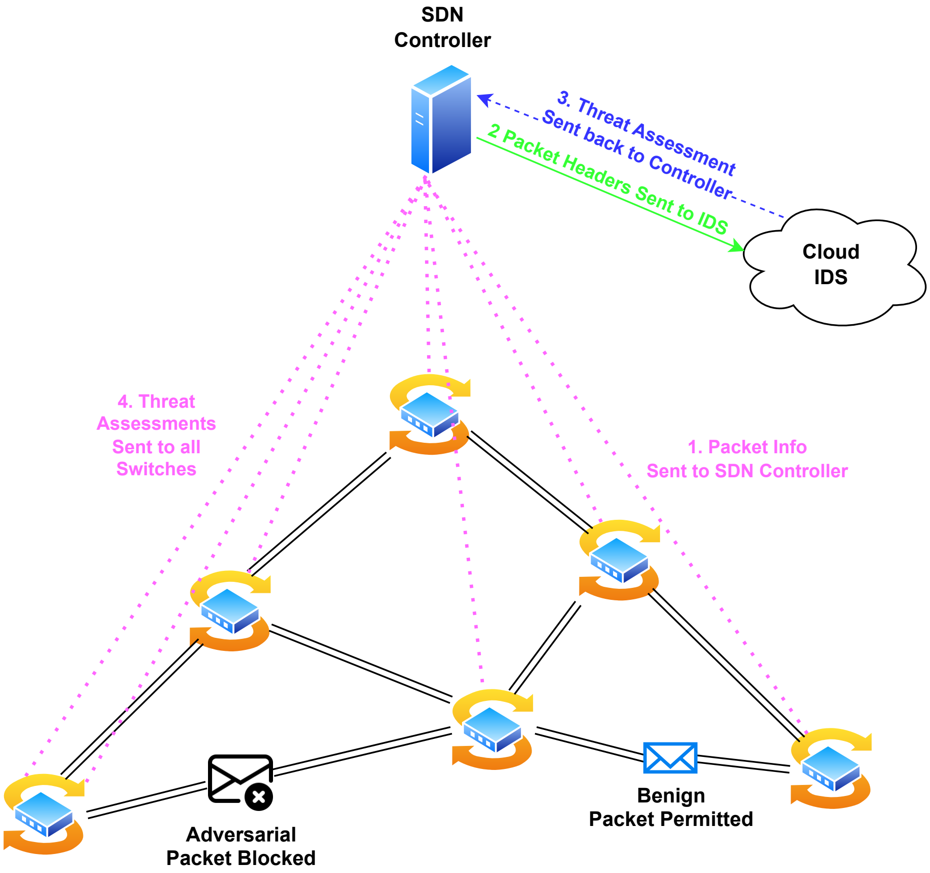


Figure 1.1: Proposed Crossfire Defense Mechanism

details the mechanism.Packets are sent between the switches. When these packets are sent, info is sent to the SDN controller, which forwards the headers to a cloud-based intrusion detection system (IDS) that has implemented our proposed model. The IDS then sends back a threat assessment of the specific packet. If the assessment returns an adversarial result, the resulting packet is blocked. Otherwise, the packet is permitted to continue propagating.

1.3 Summary of Novelty

The summary of contributions are:

- The thesis proposes a statistical detection model for crossfire attacks using Analysis of Variance (ANOVA) and neural networks.
- The proposed mechanism only uses packet headers and not packet content to determine if a packet is adversarial that achieves an accuracy of 95.3% in detecting these packets

- We evaluated the proposed defense mechanism on a real-world network topology from the ATT North America backbone topology using the Mininet simulation environment
- We design a traffic optimization using Binary Linear Programming for SDN.

Chapter 2

Background & Related Work

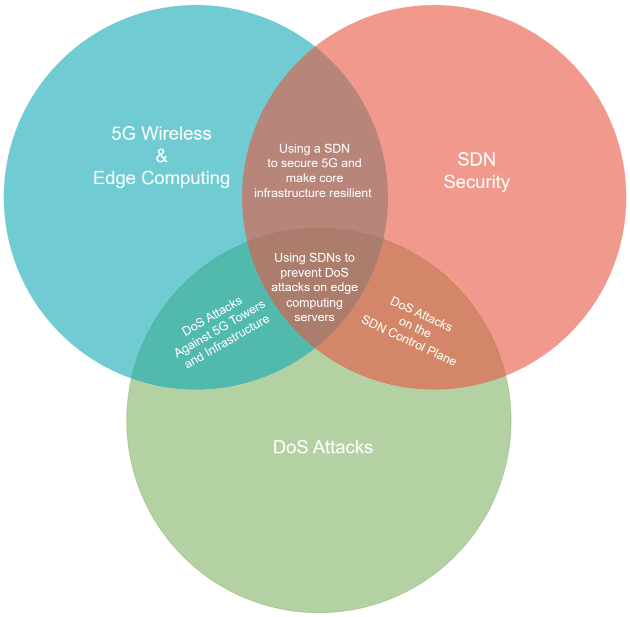


Figure 2.1: Research Area

2.1 5G/6G Architecture

Until recently, mobile communication was handled by fourth-generation (4G) and Long Term Evolution (LTE) systems. Recently with the rise of Internet of Things (IoT) devices and a larger focus on edge computing, a new standard had to be created in order to support the rapidly growing Internet. Fifth-generation technology (5G) is the next standard of mobile communication that will be able to support such a wide variety of devices simultaneously. 5G services are attempting to meet 3 main constraints as it develops: ubiquitous connectivity, zero latency, and high-speed gigabit connection [3].

2.1.1 Network Architecture

As the number of mobile devices exponentially increases, there is a need for an architecture redesign from the previous generation. Differences in the waves used for 5G that permit increased speed require careful consideration due to differences in propagation.

5G cellular network architecture is distinct from previous generations but retains many features of those generations. A renewed focus on interior architecture is necessary. With 4G, an outdoor base station had the ability to allow both inside and outside users to communicate. Due to the constraints and changes in architecture, the shorter waves of 5G cannot penetrate walls as easily [4]. Therefore, 5G architecture must consider distinct interior architecture to overcome the issue of penetration-loss [5]. The use of multiple-input, multiple-output (MIMO) technology can help reduce the burden of penetration loss.

Multiple-input, multiple-output (MIMO) is a technology that allows a wireless network to transmit and receive signals simultaneously over shared radio channels using many antennas [6]. Massive MIMO scales up this idea to larger proportions, though no concrete definition of what "massive" entails exists. Massive MIMO includes antennas both located on-site and distributed [5]. By including interior antennas, inside users would only have to communicate with an interior access point which reduces penetration loss [4]. This technology allows 5G to be more energy efficient, allowing for the capacity to increase 10 times and increasing the efficiency of the energy radiated by 100 times. Using massive MIMO also allows lower power components to be used which are often cheaper to purchase [6]. Finally, using MIMO allows for the reduction of latency which helps 5G achieve a core goal of "zero latency".

Table 2.1: OSI Layers

Layer #	**OSI Layer**
7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Data Link
1	Physical

As with any other network, 5G is comprised of all 7 OSI layers. Table 2.1 shows the 7 OSI layers in order from top to bottom. Layer 1, known as the physical layer describes the physical medium in which packets transmit. They can take the forms of wires, fiber, and radio waves. The data link layer is the second link in the OSI model. This layer provides access rules and protocols for sharing the physical media. This layer also contains the media access control (MAC) address which is unique for every physical device. The third layer, the network layer, provides a means for open systems to communicate by establishing, maintaining, and terminating connections. The network layer contains the Internet Protocol (IP) and establishes IP addresses. The transport layer is the fourth layer in the OSI layer taxonomy. The transmission control protocol (TCP) is contained within this layer. The transport layer is responsible for ensuring the integrity and reliability of the data transmitted across the network. Layer 5, the session layer, allows for 2 communicating presentation entities to communicate with one another. The presentation layer is the sixth layer. This layer is responsible for packing and unpacking application data to get it ready for transport.

Encryption, decryption, and graphic expansion occur in this layer. The final layer of the OSI model is the application layer. End-user software that communicates on the network is present on this layer. This layer is the highest abstraction of the network model and each application has a specific, distinct purpose [7].

2.1.2 Internet of Things

About 25 billion devices are currently interconnected and by 2025, 60 billion devices are expected to be connected [1]. As the Internet continues to develop, traditional devices are becoming "smart", meaning that they are connected to the Internet. The term "Internet of Things" was coined in 1999 by Kevin Ashton that describes a global network of interconnected devices [1]. The motivation for IoT devices is to create large "smart" systems [2]. Technological advancements are the reason for the increased motivation to link devices together [1]. IoT devices take many forms and almost any traditional device can be converted to a smart device. Some examples of IoT devices include smart plugs, smart washing machines, smart lights, smart refrigerators, etc.

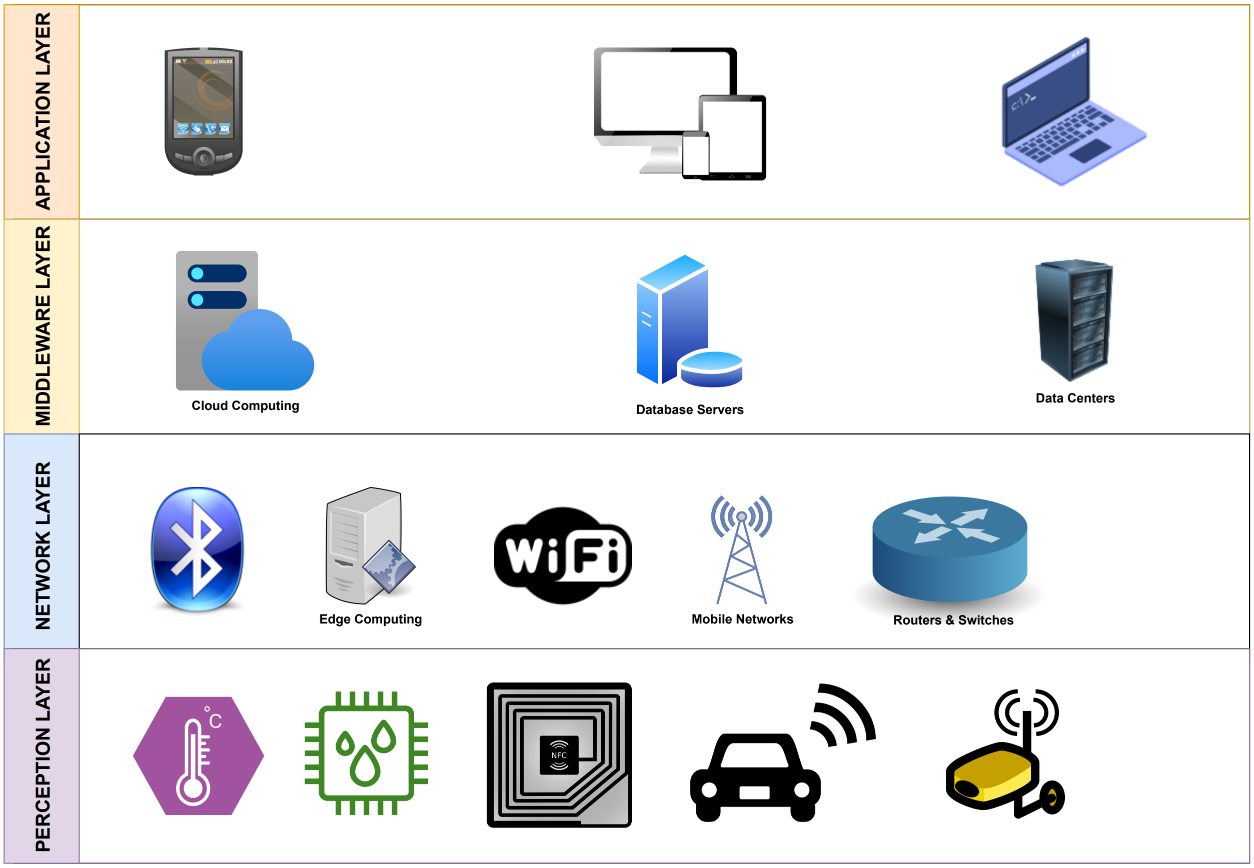


Figure 2.2: The 4 Layer IoT Architecture

The Internet of Things has its own distinct architecture that works with the Internet. Typically IoT is categorized into 4 layers. Figure 2.2 details the types of devices that are present on each layer. The perception layer is the lowest level of the IoT architecture. The perception layer contains the sensor-enabled physical objects which act as endpoints to the IoT ecosystem. The next layer, the network layer, consists of various communication protocols, edge computing, and network connectivity. This layer transfers information securely from IoT endpoints to a processing device. The middleware layer receives data from the network layer and stores it in a database. Cloud computing servers and database management systems are typically middleware devices that allow applications and sensors to connect. Finally, the top layer of the four-layer IoT architecture is the application layer. This layer IoT exists as a result of many technologies. These technologies work together to create a holistic system that is able to communicate across the internet. Radio frequency identification (RFID) was a large precursor to IoT as it allowed machines to record metadata, recognize objects, and control devices through radio waves [2]

IoT Devices may provide useful services, however, they currently present a large security problem in the world of networking. The first concern that arises with the introduction of IoT is that creating additional devices that are addressable can allow attackers to intrude [2]. Security measures are only as good as the weakest link, and the introduction of new devices opens the door to additional vulnerabilities that could be exploited by an adversary. Due to the low cost of IoT devices, corners may be cut in terms of manufacturing. Oftentimes, IoT devices may use default or anonymous logins which an adversary can use to intrude on a network [8].

2.1.3 Security Concerns in 5G/6G

5G is helping to connect the world by creating an ever-growing network. With the creation of new software and architecture, both new and old vulnerabilities are concerning. With the increased computational power of mobile devices, launching complicated attacks on a mobile network using these devices is possible. These vulnerabilities extend across all 7 OSI Layers [9].

The application layer is vulnerable to attacks as applications have their own concerns due to the way they were constructed. The presentation layer holds vulnerabilities such as hiding data by using multimedia components. The session layer is vulnerable to man-in-the-middle attacks where an adversary intercepts communication between 2 points and either sniffs or manipulates it before passing it on to the intended party. In the transport layer, transport layer security is a channel to execute DDoS attacks. While IPSec is unfeasible to break, the network layer is still vulnerable to layer-agnostic attacks such as DoS and man-in-the-middle. The data link layer handles protocols such as WiFi and Bluetooth. These protocols are vulnerable to password cracking and other types of protocol attacks such as bluejacking. Finally, in the physical layer, vulnerabilities exist in wireless communication, as anyone with equipment can eavesdrop on the radio waves that are being transmitted through the air. Each layer has its own vulnerability, and while it is virtually impossible to patch every vulnerability, 5G will have to work to solve some of these issues to remain secure enough for daily use [9].

2.1.4 Mobile Edge Computing

Mobile edge computing (MEC) is an architecture where cloud computing services are placed on the edge of the network using mobile base stations [10]. With the ever-increasing need for cloud computing services while using mobile devices, placing computing servers within the radio access network (RAN) and in close proximity to these devices allows mobile traffic to connect to the nearest cloud service edge network. By placing MEC services within the RAN, bottlenecks associated with traveling through the core of the internet can be reduced [10]. The European Telecommunications Standards Institute characterizes MEC by the following criteria: [11]

1. On-Premises - The edge services should be located at the edge of the network, meaning they should be able to run isolated from the core of the network
2. Proximity - By being close to the source of the data/information, MEC is useful for analytics and data collection
3. Lower Latency - By being closer to the edge devices, latency is considerably reduced. This can be used to reduce latency or improve user experience.
4. Location Awareness - When connected to WiFi or cellular, services can use low-level signaling to determine the location of connected devices.
5. Network Context Information - Real-time network statistics can be used by applications to provide context-specific services that can be monetized and change the experience of mobile communication.

Mobile edge computing can be used in many sectors to offload core services. Augmented reality (AR) systems typically require high computational power. Many users use (AR) on their mobile devices, so computations have to be offloaded to servers. Edge computing would allow these high-demand, low-latency tasks to remain at the edge of the network [10]. Edge computing also will play a key role with respect to web performance and caching HTML content. By deploying content delivery servers at the edge, HTTP requests would travel through these servers that would handle many of these requests, reducing traffic across the core network [10]. MEC services allow 5G to continue to work towards the core goal of "zero-latency" as reducing congestion in the core allows more traffic to be routed. This in turn improves the experience for users of 5G technology.

2.2 Software Defined Networks

Traditional networks were built to enforce specific policies and implementations that are no longer relevant due to the advanced nature of today's devices [12]. Software-defined networks (SDNs) are a newer type of network that separates the data plane from the control plane [13]. With the control plane separated, network control and routing decisions can be centralized allowing them to be more flexible and dynamic [14]. Software-defined networks are sometimes differently defined. For this document, an SDN can be defined as a network architecture that has the following four pillars: [13]

1. The control and data planes are separated. Control functionality is removed from network hardware.

2. Forwarding decisions are made with respect to flow instead of the destination.
3. Control logic is delegated to an external entity known as an SDN controller or NOS.
4. Network is programmable via software that runs on top of the NOS which interacts with the data plane devices.

2.2.1 SDN Architecture

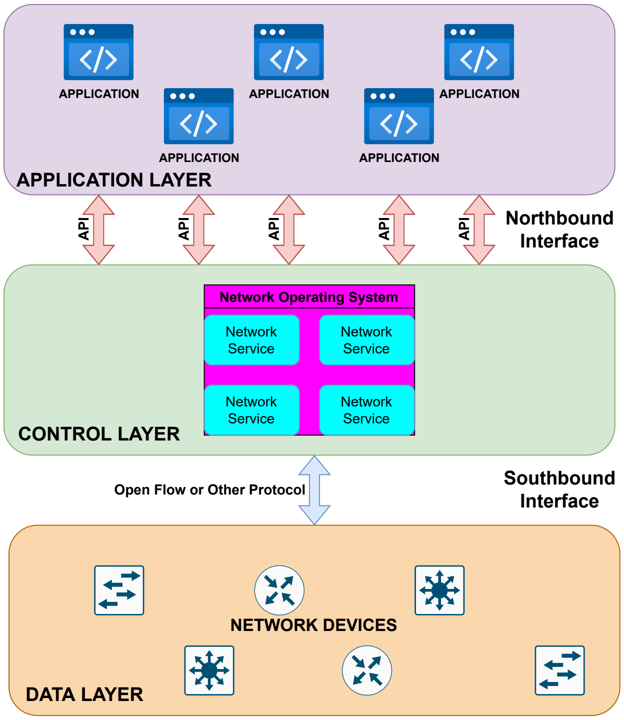


Figure 2.3: SDN Architecture. [15][1]

The architecture of software-defined networks can be broken into three layers: application layer, control layer, and data layer. Figure 2.3 shows these layers, what types of devices reside on each layer, and how each layer communicates with the others. The three layers of the SDN architecture are independent but work together to route packets correctly.

The application layer contains the applications that make decisions about traffic. This layer is the highest level of abstraction and the easiest to program on. The next layer is the control layer which houses SDN controllers that contain the control logic about the network. The controllers often run a network operating system (NOS) which contains various network services that make decisions based on information received by the other two layers. The lowest layer, the data layer contains the physical network devices such as routers and switches that are responsible for forwarding the packets. Since SDN abstracts away the decision-making, these devices no longer

[1] Reproduced with permission from Springer Nature

make decisions about where to forward packets and instead rely on the control layer to instruct them [15].

The layers of SDN architecture communicate with each other using interfaces, known as the northbound and southbound interfaces. The application layer communicates with the control layer via the use of application programming interface (API) calls. These API calls can take many forms including REST, JSON, and XML. This connection using APIs is called the "northbound interface" since it connects the two northernmost layers. The control layer and data layer communicate with each other through the use of the southbound interface. The southbound interface is a protocol that shares routing information between the two layers. The most popular southbound interface protocol is OpenFlow [15].

2.2.2 Technologies

Network function virtualization (NFV) was originally proposed by over twenty of the largest telecommunication operators. The term describes a network architecture that transforms how one builds and operates a network by leveraging existing virtualization technology as well as consolidating proprietary functions into standard devices [16]. The concept of NFV has created many new terms to describe functions that are introduced. NFV is still very new however the virtualization technology that backs it has been around for many years. NFV is important as it allows SDN devices to communicate with virtualized devices, such as virtualized load balancers, firewalls, and network address translators [17].

OpenFlow is a protocol that was proposed to standardize the way that switches and controllers interact with each other in an SDN [18]. As mentioned previously this protocol has become the most popular southbound interface protocol [15]. The OpenFlow architecture only requires three main components: an OpenFlow-enabled switch, an external controller, and a secure channel for both devices to communicate. A controller is a piece of software responsible for monitoring and changing the flow tables of a switch using OpenFlow protocols. An OpenFlow-compliant switch is a network switch that is capable of forwarding packets based on rules defined in a dynamic flow table. Several OpenFlow-compliant switches exist and can be purchased with support for this protocol out of the box [18]. OpenFlow switches have one or more tables of routing rules. These flow tables match a subset of the traffic exchanged and perform actions on that traffic. Therefore, an Open Flow switch can act like many devices, such as a router, a switch, a firewall, or a load balancer [13]. Networks that are OpenFlow enabled had many additional capabilities. They are able to control many switches from a single controller, they can analyze traffic information across many nodes, and they can dynamically update forwarding information based on many factors [18]. OpenFlow allows for many types of applications to be created in order to maintain a network. Some applications can make configuring network devices easier while others can manage the flows of the network. Programs that provide security to the network can also take advantage of the protocol. Programs that can detect, prevent, and mitigate DDoS attacks have been proposed due to the additional control these programs have over the entire network [18]. This protocol has become so versatile and associated with SDNs that the term "OpenFlow" and "SDN" are often interchanged. OpenFlow has changed the network paradigm and enabled many features not seen before.

MiniNet is a network simulator capable of simulation software-defined, traditional, and hybrid

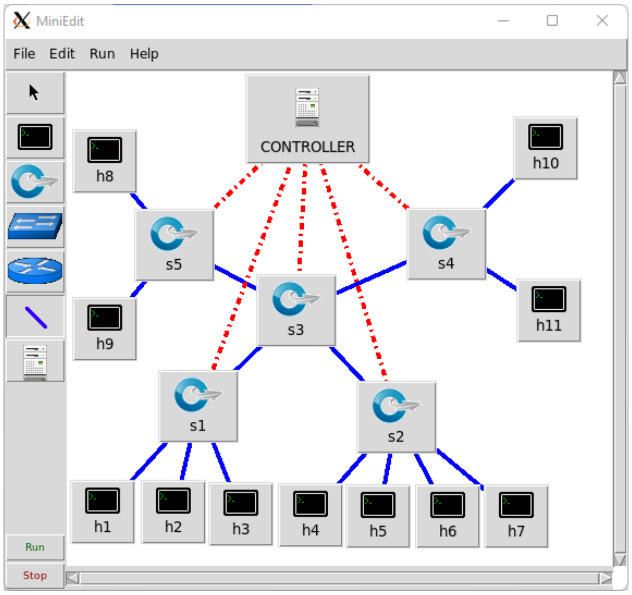


Figure 2.4: MiniEdit, the Graphical User Interface for MiniNet

networks. MiniNet typically runs on a virtual machine and modifies the network adapters of the virtual machine to mimic a network. Mininet supports the use of hosts, OpenFlow switches, legacy switches, legacy routers, and controllers. Since OpenFlow is so dynamic, OpenFlow switches can take the place of many network devices such as routers, switches, firewalls, and intrusion detection systems. MiniNet uses OpenFlow as its southbound interface for SDN switches. MiniNet allows the user to use terminal commands as a specific host, so commands like "iperf" can be used to send packets from one host to another. Mininet also allows for simulation of real-world conditions. Links can be forced down to simulate a crash. Additionally, links can be programmed to lose packets a certain percentage of the time, and delay can be programmed in too. By allowing for real-world conditions, a researcher is able to observe how the given network topology responds to unexpected outages and events. MiniEdit, the graphical user interface for MiniNet allows users to construct complex networks graphically instead of through the command line interface. Figure 2.4 shows the MiniEdit graphical user interface.

2.2.3 Benefits

SDNs allow for a higher level of abstraction, meaning that the complexities of managing individual network devices are abstracted away [14]. SDNs are considered viable for both data centers and backbone networks [14]. Higher level programming languages are made available for SDNs which allows implementation of abstraction for important properties and functions of SDN such as structures, distributed updates, and modular composition [13]. By decoupling the control and data layers, higher flexibility allows for new features. Allowing a centralized control plane to handle routing decisions improves routing as the controller has a better view of the entire network. In traditional networks, each network device tries to create its own 2-D view of the topology and tries to route to the best of its ability [19]. Immediate advantages also exist for network operators by adopting SDN. IPv6, network virtualization, load balancing, and others exist once SDN is implemented [14]. Network virtualization is very important as it provides the capability to create many logical networks on top of the already established physical infrastructure [15].

Software-defined networks can also benefit data centers that handle big data applications. Within the data center, big data applications rely on internal networks to transfer large amounts of data between server nodes. The data that travels within the data center is often greater in volume than the data that travels from the user to the data center; therefore these intranet links are critical to processing big data. Once SDN controllers and switches are set up, these controllers can make better decisions about routing through a holistic perspective. These controllers can identify large flows and prioritize them. Additionally, these controllers can respond to dynamic traffic patterns such as seasonal demand. Information also flows between data centers so implementing an SDN may benefit the communication between these links. The same policies that work inside of a data center can be applied to many different data centers across geographical regions by using a multi-tier controller to connect these data centers [19].

2.2.4 Concerns and Drawbacks

Table 2.2: Industry Concerns with SDNs [20]

Industry Sector	**Concern**	**Percentage of Respondents**
Finance	Cost	41%
	Integration with existing systems	38%
	Security	36%
Government	Cost	51%
	Integration with existing systems	34%
	Lack of skills possessed by employees	32%
Education	Cost	53%
	Integration with existing systems	35%
	Security	35%

While SDNs are often considered beneficial to networks, there are some concerns that occur as a result of adopting the new network paradigm. SDNs are not a completely secure solution. Table 2.2 discusses some of the industry concerns associated with SDNs in 2014. While this data is not recent and many improvements with SDNs have taken place, many of these concerns are still valid. For example, the cost of infrastructure replacement will always be present, as there are associated costs with removing and replacing devices with new ones. Since the industry is constantly evolving and no solution is perfect, there will always be concerns surrounding the use of SDNs. As time progresses, ideally the concerns should decrease in severity to allow for the widespread adoption of this network technology.

Additionally, many security threats exist as a result of the new paradigm. The first type of threat exists on the application plane of SDNs. Table 2.3 summarizes the types of vulnerabilities

Table 2.3: Security Vulnerabilities and Threats in Software Defined Networks

Attack Plane	Security Vulnerability / Threat
Application Plane	Unauthenticated and unauthorized applications, bugs in applications
Northbound Interface	Insertion of fraudulent rules
Control Plane	Unauthorized access of controllers, DoS attacks
Southbound Interface	Man-In-The-Middle Attacks
Data Plane	Unauthorized joining and access of network devices.

and threats that are present when using SDNs. SDN controllers allow applications to manage the network. These applications have read and write access to the controllers, which allows for application vulnerabilities to open the network during the attack. Application developers must ensure that all requests and applications are both authenticated and authorized. Bugs in development can be used to exploit the entire network. The northbound interface is also vulnerable to attack. Even if applications are both authorized and authenticated, fraudulent rules can be inserted via the interface which would in turn change the operations of the network in the adversary's favor [21].

The control plane is a centralized plane in which instructions are communicated to the data plane. This centralized point-of-failure can become the subject of Denial of Service (DoS) attacks. Additionally, unauthorized access to these controllers would allow for full network control, creating a situation in which an attacker would be able to modify the network in any way that they please. The southbound interface communicated across the clearnet. This means that if a secure means of communication between network devices and controllers are established, the communication link would be susceptible to man-in-the-middle attacks. These attacks would allow for the acquisition and modification of data sent to and from each device. Finally, the data plane is vulnerable to threats if proper authentication and authorization is not implemented. Proper authorization and authentication must be implemented to ensure the management of data plane nodes is restricted to those permitted. Without this proper authorization, enforced policies could be modified or removed, or additionally, devices could join the network fabric. When implementing SDNs, it is imperative that security measures are taken to ensure the integrity of the network remains [21].

2.3 Denial of Service Attacks

Denial of service (DoS) attacks are a lethal type of attack that threatens the internet every day. These attacks are extremely easy to execute and are very difficult to detect and counter. As the internet evolves, different forms of DoS attacks are introduced. Distributed denial of service, link flooding, and crossfire attacks are some of the newer attacks that have evolved from the traditional DoS attack.

2.3.1 Traditional Denial of Service Attacks

The traditional denial of service attack works by overwhelming a victim server with requests to consume all of the available bandwidth and processing power thus making the machine unavailable to legitimate users. There are different ways to execute such an attack. One way to execute a DoS attack is to flood the victim server with ping (ICMP) requests that have a spoofed origin IP address to saturate the node. This attack is known as a SMURF attack. The Fraggle attack is similar to the SMURF attack however the Fraggle attack uses UDP packets instead of ICMP requests [22]. Another variation of the DoS attack hijacks the TCP handshake protocol by sending TCP SYN requests to a destination server often spoofing their IP to an unreachable address so that the destination server waits for an ACK message from a server that does not exist. The server keeps the connection open for a specified amount of time. The attacker will do this many times to fill the server memory with illegitimate SYN messages to prevent legitimate requests from being processed or queued [23].

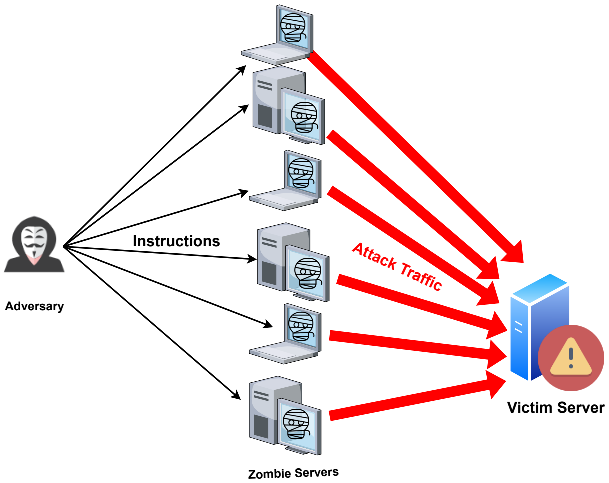


Figure 2.5: Execution of a DDoS Attack

Denial of service attacks are often classified by the number of sources that send illegitimate traffic. Single-source attacks are executed using one machine. Multi-source, or distributed denial of service (DDoS) attacks work by using many machines to attack a single destination. Many times, these machines are compromised by adversaries so that their computing powers can be harnessed for free. The rightful owner of the hijacked machines is often aware that their machine is being used to execute attacks. The groups of these machines are known as botnets [22]. Figure 2.5 shows the execution methodology of a DDoS attack.

The first DDoS attack occurred in 1999 when the network of the University of Minnesota was disabled for two days [24]. After this attack, DDoS attacks have continued to become more common and larger in size. In 2010, the largest attack was 100Gbps whereas in 2016 the largest attack was 1Tbps [24]. A ten-fold increase in the size of the largest attack that occurred over the short period of six years shows how evolved DDoS attacks have become.

These different DoS methodologies all attempt to achieve the same goal: to disrupt, degrade or disable network traffic between legitimate users and services. DoS attacks may be executed for a variety of reasons. The motives of DDoS attacks can be categorized into the following [23]

1. Economic Considerations

2. Political Considerations
3. Ideological Considerations
4. Extortion
5. Personal Feuds
6. Naive Enthusiasts
7. Cyber Warefare

Regardless of the reason, DoS and DDoS attacks are extremely lethal and difficult to detect. The distributed nature of DDoS attacks furthers the difficulties in detecting the attack [24].

2.3.2 Link Flooding Attacks

Table 2.4: Types of LFA Attacks

Attack	Characteristics
Coremelt	Low persistence Compromised machines used Traffic sent to/from the attacker's machines Traffic indistinguishable from legitimate traffic.
Spamhaus	Low persistence Specialized bots used
Crossfire	Independent distribution of bots Traffic indistinguishable from legitimate traffic. Specialized Bots Traffic sent from attacking servers to public decoy servers Scalable

Link flooding attacks (LFAs) are a type of DoS attack that target critical network links instead of single destination servers. These attacks are extremely dangerous as they can isolate a victim's entire network [25]. Link flooding attacks are also very difficult to detect, as network traffic is rarely if ever, sent to the victim server(s). Instead, the adversaries send packets to decoy servers that occupy a common link with the victim, congesting the link and degrading or disabling the link. LFAs are lethal to both traditional networks and SDNs. SDN infrastructure can be the target of LFAs, which could disrupt traffic on a much larger scale than seen on traditional networks. As mentioned before, SDNs often have a single point of failure, the controller, that can be the target of both DoS and LFA attacks [21].

LFA attacks can be classified into three categories: Coremelt, Spamhaus, and Crossfire. Table 2.4 shows the different types of LFA attacks as well as the specific characteristics of each attack.

The Coremelt attack utilizes compromised systems that send packets to and from each other to congest a shared link. The adversary can circumvent detection and defense measures by using protocol-conforming traffic. This attack can allow an attacker to take down both local links and backbone links so long as compromised hosts are sufficiently distributed across many networks [25].

One major LFA attack was the Spamhaus attack. This attack targeted Spamhaus, an organization that provides email spam filtering services. The attack targeted critical links to Internet exchange points (IXPs) in Europe and Asia to deny service. The attack started first by attacking servers but then evolved into flooding network links that connected Spamhaus to IXPs. This attack was so severe that some consider the attack in its category as a discreet type of LFA attack [25].

2.3.3 Crossfire

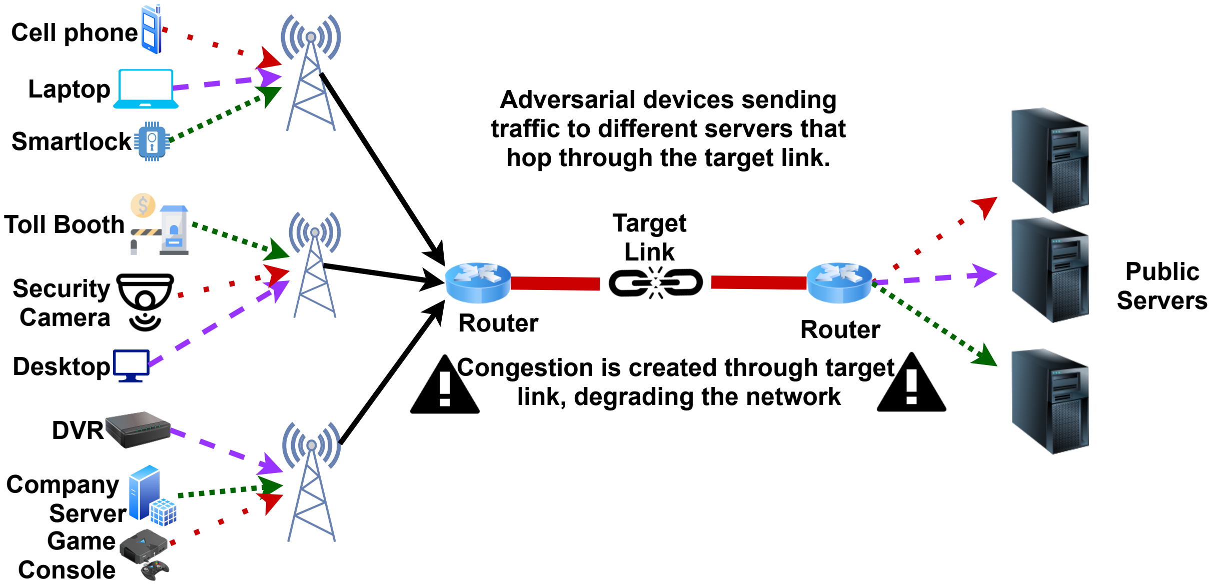

Cell phone
Laptop
Smartlock
Toll Booth
Security Camera
Desktop
DVR
Company Server
Game Console
Adversarial devices sending traffic to different servers that hop through the target link.
Target Link
Router
Router
Congestion is created through target link, degrading the network
Public Servers

Figure 2.6: Execution of the Crossfire Attack

The crossfire attack is a type of LFA attack that attempts to degrade or disable connections to a specific geographical region of the internet. This attack is perpetuated by directing low-intensity requests to various public servers that share a common link to flood that shared link. The attack uses multiple attacking nodes, each sending low-intensity traffic to different destination nodes. This type of attack does not affect one specific destination, instead, it targets a large geographical region, served by the target link node. This type of attack is devastating to a specific geographical region, as both upstream and downstream traffic is affected [26]. Figure 2.6 show the way a crossfire attack can be executed. The crossfire attack is more difficult to detect, as crossfire is an *indirect* attack, contrary to most other attacks. Since the attack spreads low-intensity traffic to various destinations, this allows the attack traffic to blend in with legitimate traffic and is virtually undetectable in standard DoS detection and mitigation protocols, at least until after substantial damage has been done [26].

To execute a crossfire attack, first, a potential adversary would select a list of public servers within the targeted areas and a set of decoy servers away from the target area. Since these servers are publicly accessible, they can be easily found. Next, the adversary would have to perform reconnaissance and generates a link map. This link map would be a map of layer 3 links that connect their decoy servers to public servers. The map enables the adversary to select a set of "target servers" that when flooded, would be able to cut off the target area from the open internet. The adversary then coordinates the decoy servers to flood the target link, effectively blocking most flows to the target area. Each server sends low-intensity traffic so as to not arouse suspicion of anomaly-based detection systems. The individual low-intensity flows are indistinguishable from legitimate traffic. Finally, the adversary begins to flood each target link one at a time to not trigger automatic packet route mutation [26].

2.4 Related Work

2.4.1 MTD

One proposed crossfire defense solution is the moving target defense (MTD). Traditional networks are static in nature, allowing attacks to spend as much time as needed to gather information and find vulnerabilities [27]. MTD has historically been a warfare strategy but recently has been adopted into the IT world. A moving target defense can first delay network mapping and reconnaissance. ICMP and UDP scans can be disrupted by imposing fake hosts and services on random ports that do not exist. The fake listeners can increase the time and workload an attacker would need in order to launch an attack [28]. Additionally, a mutable network can be created that changes the IP address and ports of network hosts dynamically. This means that machines will not be able to be located at the same address at any given time [27]. Oftentimes, MTD takes the form of random route and address mutation. Randomization has been a common strategy in moving target defenses. This strategy is based on the idea that if addresses of targets within the network constantly change or access policies between the attacker and target change, then the attack success rate will drastically reduce. An attacker's ability to effectively do reconnaissance is sharply diminished as well due to the ever-changing network landscape [28]. Obfuscation of links via SDN has also been proposed to confuse and thwart attackers [29]. By obfuscating links between the attacker and target, an adversary would not be able to identify common links, a key step in performing a crossfire attack.

2.4.2 Rerouting-Based Defenses

Moving target defense can also be used to create virtual "routable IPs". While the real IPs of hosts remain unchanged and static, the virtual IPs assigned by the network consistently changed frequently and synchronously. However, the higher the rate of mutation, the more overhead is required to run the network [27]. This type of approach is often used by load-balancers, and

Another proposed way to mitigate large-scale DDoS attacks is by using a routing around congestion (RAC) defense. The RAC defense works by routing traffic between a service deployer and a critical autonomous system around degraded links. RAC defense asserts that attack traffic is irrelevant and does not need to be filtered when using this defense [30]. The RAC defense offers path isolation by dynamically creating detour routes for critical flow [31].

2.4.3 Infeasibility of Current Mitigation Techniques

While the proposed defense solutions may be promising in theory, their feasibility in practical application is questionable. One example of an infeasible defense is the use of rerouting-based solutions such as the Routing Around Congestion (RAC) technique. RAC aims to avoid specific autonomous systems by employing border gateway protocol (BGP) poisoning. However, implementing RAC in production servers presents numerous challenges that make it unviable. Firstly, the current border gateway protocol infrastructure may not be compatible with the modifications required to support RAC. Updating the protocol in a way that enables this defense may prove to be technically unachievable or require significant overhauls that are impractical to implement in existing systems. Furthermore, even if the necessary changes were made to enable RAC, there is a concern about potential misuse for malicious purposes. Allowing the manipulation of BGP to avoid certain autonomous systems could be exploited by threat actors with nefarious intent. They could utilize this defense to launch attacks against specific autonomous systems, disrupting their operations or compromising their security. Moreover, rerouting-based defenses like RAC are susceptible to hijacking. Adversaries could take advantage of the rerouting mechanism and continuously force reroutes, leading to packet drops or delays. This constant instability in network routing can significantly impact the performance and reliability of the network, rendering the defense ineffective and causing disruptions for legitimate traffic. Lastly, the implementation of a moving target defense, which involves constantly changing network configurations and system parameters, imposes significant overhead. This overhead may not be practical to handle on large-scale networks, where the sheer size and complexity of the infrastructure make frequent changes challenging to manage. The cost, resources, and potential disruptions caused by implementing a moving target defense can outweigh the benefits it offers, making it unfeasible for widespread adoption. In summary, while rerouting-based defenses like RAC may hold promise in theory, their feasibility in real-world scenarios is questionable. Technical compatibility issues, potential misuse, vulnerability to hijacking, and the practical challenges of implementing a moving target defense all contribute to their infeasibility in nature. Alternative approaches that address these limitations may need to be explored to ensure effective and practical defense solutions for large-scale networks [31].

2.4.4 Current Detection Efforts

Current defense mechanisms treat both legitimate and attack traffic the same, degrading the performance of legitimate users. Current attack traffic detection methods point to detecting DoS and DDoS attacks, not link-flooding attacks. These detection efforts often rely on traffic being directed to a singular endpoint. Therefore, models that detect standard DoS and DDoS attacks may not be able to accurately detect crossfire attack traffic.

One study, Narayanadoss et al. [32] proposes a deep-learning model to detect crossfire attacks in intelligent transport systems. This study provides a machine-learning-based model to detect vehicles in a network that are involved in the attack. The created models reflected a detection rate of 80% in a network of 35 nodes [32]. This model includes data irrelevant to traditional networks (vehicle speed) that may impact the model's accuracy in networks that are not intelligent transport systems. Additionally, as the network grew, detection accuracy decreased. Only a maximum of 35 nodes were implemented. As noted by the author, as the number of nodes increased, "[m]any legitimate flows could be detected as part of attacking traffic as they may have a temporal correlation with other attacking flows" [32]. This model may prove infeasible in larger networks, such as networks in large cities. Beyond this singular model, significant work has not been done to detect crossfire attacks and classify traffic based on characteristics.

Chapter 3

Crossfire Avoidance Framework

As mentioned before, the crossfire attack is a devastating form of LFA (Link Flooding Attack) that aims to disrupt connectivity to a specific geographical region of the internet by flooding a shared link. It employs multiple attacking nodes, each sending low-intensity traffic to different destination nodes, making it difficult to detect and differentiate from legitimate traffic. The attacker strategically selects public servers within the targeted area and decoy servers elsewhere, creating a link map to identify target servers for flooding. By coordinating the decoy servers to flood the target link, the attack cuts off the target area from the internet, affecting both upstream and downstream traffic [26].

This thesis aims to determine an effective way to detect and mitigate crossfire attacks. Since crossfire attacks are currently difficult to detect and virtually indistinguishable from legitimate flash mobs, current crossfire defense mechanisms treat all traffic the same. By finding discriminating characteristics between attack traffic and legitimate traffic, defense mechanisms may be developed that allow legitimate traffic to pass through the critical link while blocking attack traffic entirely.

3.1 Execution of a Crossfire Attack on an SDN

Execution of a crossfire attack may seem relatively simple in theory, but its practical implementation poses challenges due to the dynamic nature of network routing. The fundamental concept behind a crossfire attack is to flood a specific link by saturating it with a large number of low-intensity requests [26]. However, determining the potential routes that will utilize the targeted link can be a complex task. In a network, the routing paths taken by packets are influenced by a variety of factors, such as routing protocols, network congestion, load-balancing mechanisms, and dynamic network conditions. As a result, accurately predicting the routes taken by packets in real-time becomes a non-trivial problem. Network paths are subject to frequent changes, and the actual routes traversed by packets may not always be readily available or predictable.

To effectively execute a crossfire attack, one must first identify potential routes that are likely to utilize the target link. This involves analyzing the network topology, routing tables, and potentially even monitoring network traffic to gather information about the routing decisions made by the network devices. By understanding the network's routing protocols and configurations, an attacker can gain insights into the potential paths packets might take. Once a sufficient number of potential routes have been identified, the attacker can proceed to generate low-intensity requests across all of these routes simultaneously. The intention is to distribute the attack traffic across multiple paths, overwhelming the target link with a high volume of seemingly innocuous requests. This is important due to the presence of anomaly-based intrusion prevention systems. These systems can include statistical-based techniques to detect attacks. Network traffic activity is monitored and analyzed to create a profile that captures its stochastic behavior. This profile incorporates various metrics such as traffic rate, protocol-specific packet counts, connection rates, and unique IP addresses[33]. During the anomaly detection process, two datasets are compared: the currently observed profile, representing real-time traffic behavior, and a previously trained statistical profile. By evaluating the differences between these profiles, an anomaly score is computed, indicating the level of irregularity for a specific network event. When the anomaly score exceeds a predefined threshold, the intrusion detection system is triggered to flag the occurrence of an anomaly [33]. Therefore, if only two devices were to send a significant amount of traffic to each other, the anomaly-based intrusion prevention system would block the connection and the attack would be rendered ineffective. This strategy aims to exploit the link's limited capacity and exhaust its resources, causing congestion and potential service disruption. However, the challenge lies in accurately determining the routes that packets will take during the attack.

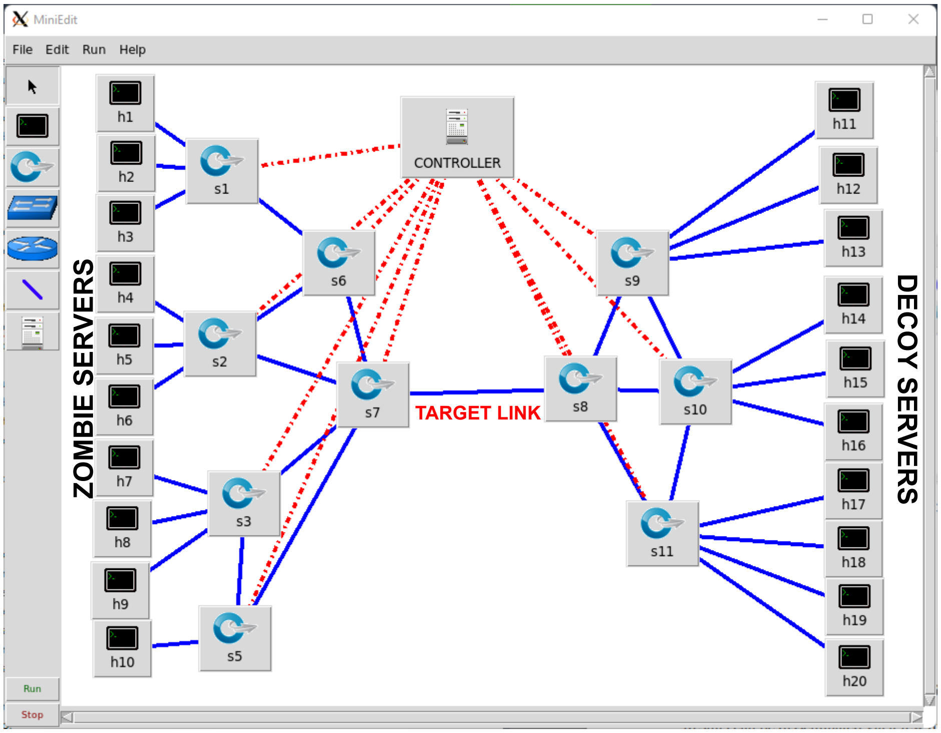


Figure 3.1: Sample Execution Network

A crossfire attack can be executed on a testbed using MiniNet. As discussed before, MiniNet has SDN capabilities and is a good way to simulate network environments. A small-scale Crossfire attack will be executed on this testbed, with 1 shared link being targeted. Ideally, 25-50 hosts will be used to route data through 1 shared link, degrading it quickly. To build the network, MiniEdit will be used. A network similar to Figure 3.1 will be used to simulate the first attack. For example, the nodes h1-h10 act as zombie servers. These servers in a real attack are compromised by the adversary. These zombie servers will send low-intensity requests to the public decoy servers h11-

h20. The public decoy servers are random web servers that are legitimate but are located across the shared link. This will force all of these requests to go through the link between s7 and s8, flooding it. Even though no host is sending a large amount of data, the link is unable to keep up with all of the requests and is degraded. During the execution of the attack, data will be collected to show clear degradation of connection due to the attack. After creating the first attack, some legitimate data will be mixed in with some hosts acting as adversaries and others acting as legitimate users. This data will be lower intensity, and take the form of legitimate requests to an endpoint. Ideally, the combination of these two separate data sources should give enough information to distinguish a crossfire attack from a legitimate flash mob. Variations of the base crossfire will be executed to create differences and similarities between general traffic and attack traffic.

3.2 Detection Model

After the attack is executed and sufficient data is gathered, our focus shifts towards identifying discernible patterns within the collected data that can effectively distinguish attack traffic from legitimate traffic. This process entails employing a range of techniques and statistical tests to discriminate between the two types of traffic: adversarial and benign. Initially, we conduct a standard analysis of variance to determine the capability of the variables employed in accurately detecting attack traffic. Since this model yields problematic results, we proceed to create alternative standard statistical models to explore additional avenues of analysis. These advanced techniques can potentially uncover complex patterns and relationships within the data, enabling more sophisticated identification and differentiation of attack and legitimate traffic.

Detecting a crossfire attack directly can be a formidable task, particularly when dealing with encrypted packets in HTTPS. The encryption of packet contents prevents inspection, leaving only the headers as accessible information. As a result, detecting and mitigating crossfire attacks often relies on analyzing these headers to gain insights and make informed decisions. In this section, we explore the methodologies employed for detecting crossfire attacks by focusing on the analysis of packet headers. By examining the characteristics and patterns encoded within the headers, researchers aim to develop models capable of classifying adversarial traffic and identifying potential indicators or anomalies that differentiate normal network behavior from malicious activity.

Given the encryption of packet contents in HTTPS, traditional methods of packet inspection and content analysis are rendered ineffective. As a result, researchers and security professionals have turned their attention to packet headers as a valuable source of information for detecting and mitigating crossfire attacks. The analysis of headers allows for the examination of key fields and attributes that can reveal important insights about the nature of network traffic, such as source and destination addresses, protocol information, timing parameters, and flags.

By leveraging the information contained within packet headers, we attempted to create models to classify adversarial traffic and identify patterns associated with crossfire attacks. This involves collecting extensive datasets of packet headers, employing statistical analysis techniques like Analysis of Variance (ANOVA), and applying machine learning algorithms, such as neural networks, to uncover hidden patterns and relationships within the data. The goal is to create predictive models that can accurately differentiate between normal and adversarial packets based solely on header information.

However, it is important to acknowledge the challenges and limitations of header-based detection. The inability to directly inspect packet contents restricts the depth of analysis, and the reliance on headers alone may not capture the full complexity of network behavior. Additionally, factors like significant delays and congested links can introduce uncertainties and affect the accuracy of classification models. Ongoing research and development are necessary to refine and enhance header-based detection approaches, ensuring their effectiveness in real-world scenarios.

3.3 Traffic Optimization Model

Finally, to create a more robust network, we use a deep learning optimization library to increase network bandwidth across the entire network. To do this, we first give the optimization library our network topology to create many different possible network scenarios. In an SDN framework, the network is controlled by a centralized controller, which makes decisions regarding traffic routing and flow management. Gurobi optimization can be utilized within this controller to determine the optimal paths for network traffic based on various objectives and constraints. The objective of the optimization model is to minimize network congestion. The use of Gurobi optimization enables efficient resource allocation and traffic management within an SDN. Finding optimal routes minimizes network congestion, reduces latency, and enhances overall network performance. Additionally, it allows network administrators to dynamically adapt to changing traffic patterns and network conditions, ensuring efficient utilization of network resources.

The traffic optimization process consists of two stages or phases. Initially, the traffic is optimized using a Binary Linear Program (BLP). In this program, each pair of nodes requires the demand to be routed through one of the available routes between them. The routing must ensure that the forwarded flows on any link do not exceed its capacity, and only one route is chosen for each pair of nodes. To represent the selection of routes, binary variables are used. A binary variable of 1 indicates that the corresponding route is chosen, while 0 indicates it is not. Therefore, the Linear Program can be formulated as follows:

$$\text{minimize} \qquad F(x) = r \tag{3.1}$$

$$\text{s.t.} \qquad \sum_{p \in P_d} u_{kp} x_{dp} = h_d, \qquad d \in D \tag{3.1a}$$

$$\sum_{p \in P_d} u_{kp} = 1, \qquad d \in D \tag{3.1b}$$

$$\sum_{d \in D} \sum_{p \in P_d} \delta_{dpl} x_{dp} \leq c_l r, \qquad l \in L \tag{3.1c}$$

where: x_{dp} is the flow on path p for demand d; u_{kp} is the binary variable; h_d is the volume for demand d; c_l is the capacity for link l; P_d is the number of candidate paths for demand d and in our case we use three paths only; δ_{dpl} = 0, 1 is a link-path indicator, with δ_{dpl} = 1 if path p for demand

d uses link l, and 0 otherwise.

Three constraints are applied. The demand constraint (3.1a) ensures that all demands are satisfied over some paths. The one path constrained (3.1b) ensures that only one path is chosen among the three routes for a pair of nodes. The capacity constraint (3.1c) ensures that load does not exceed the link capacity where $r \leq 1$, after solving (3.1).

However, the complexity of a Linear Program can become very large as the network size, particularly the number of nodes, increases. To address this issue, we limit the selection to three paths for each source-destination pair. The use of three paths has been proven to be effective and has a negligible impact on the optimal solution [34, 35, 36].

Even with this reduced number of routes, the complexity can still increase with larger networks. To overcome this challenge, we propose a Neural Network solution that predicts the optimal routes for each source-destination pair, matching the solution obtained by solving the Linear Program. This solution consists of two steps: an offline phase and an online phase.

In the offline phase, the Linear Program mentioned earlier is used to map a large number of varying input demands to their corresponding output, which is the best set of routes. This data is collected and used to train a Neural Network model. This trained model can then predict the best set of routes given specific input demands or traffic patterns.

The optimal use of this model is within a Software-Defined Network (SDN) setting, where the model can be installed in the controller. The controller periodically updates the installed routes on switches based on the predictions of the Neural Network. The frequency of route updates is determined by the network operator, and there is a trade-off involved. Setting the frequency too high may occupy the switches frequently, impacting performance. Conversely, setting the frequency too low may result in congestion on certain routes due to changes in traffic patterns, also affecting performance. Therefore, the network operator must choose an appropriate value to maintain a balance between network operation and performance. The graph below Figure 3.2 illustrates the entire process.

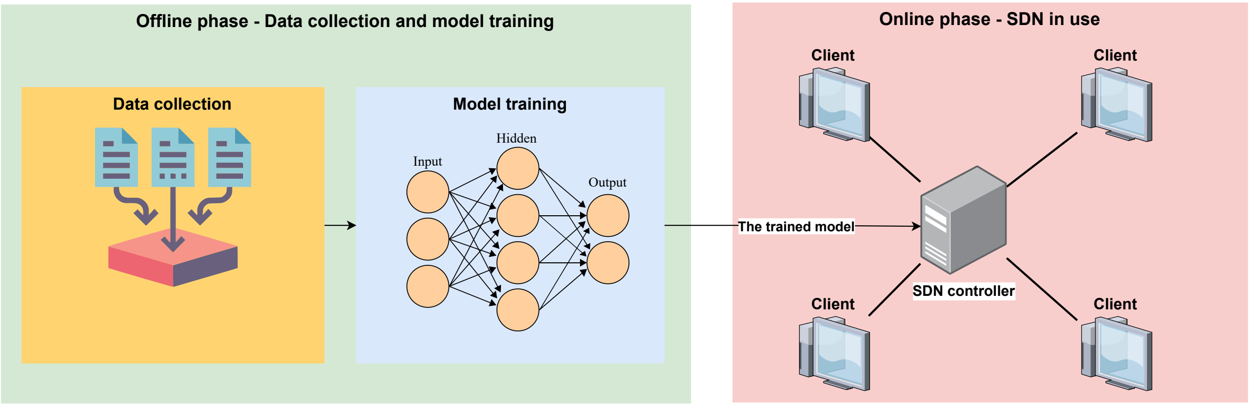


Figure 3.2: Training Methodology for Binary Linear Programming

Please note that the proposed solution enhances the efficiency of traffic optimization by combining the power of Linear Programming and Neural Networks in a complementary manner within an SDN framework.

It is worth mentioning that to solve the set of Binary Linear Programs, we have utilized Gurobi, an efficient Linear Programming solver. With Gurobi, we can handle the optimization process effectively. We apply this solver to a range of Binary Linear Programs, where we vary the input traffic while keeping the topology constant.

In an SDN framework, if the network's topology is relatively small and the time required to find the optimal solution is reasonable, Gurobi optimization can be employed within the centralized controller. This enables the controller to make informed decisions regarding traffic routing and flow management, based on various objectives and constraints. By utilizing Gurobi, the optimal paths for network traffic can be determined, aiming to minimize network congestion.

However, in cases where the topology is large and finding the optimal solution using Gurobi may be time-consuming, an alternative approach can be adopted. In such scenarios, a trained neural network can be employed to enhance the performance of the network. The neural network can assist in dynamically adapting to changing traffic patterns and network conditions, ensuring efficient utilization of network resources.

In summary, Gurobi optimization serves as a powerful tool for solving Binary Linear Programs within an SDN framework. Its usage within the centralized controller depends on the size of the topology and the time required to find the optimal solution. In cases where Gurobi optimization is not feasible due to a large topology or lengthy optimization process, a trained neural network can be leveraged to enhance network performance.

Chapter 4

Testbed Development

Throughout this thesis, various tools have played a crucial role in facilitating research, experimentation, and analysis. These tools have been carefully selected based on their specific purposes, setups, benefits, and drawbacks, ensuring their alignment with the objectives of the study. In this section, we provide an overview of three prominent tools: Mininet, Ryu SDN Controller, and Gurobi Optimization Software. Each of these tools serves distinct purposes and offers unique capabilities that contribute to the advancement of our research.

4.1 Mininet Simulation Environment

Mininet is a network simulator that allows for prototyping large networks on constrained resources. This allows us to create large software-defined networks without the need for expensive networking hardware. Mininet uses lightweight virtualization with an extensible CLI and API. This allows for a rapid prototyping workflow. Mininet allows for easy porting to real-world hardware on the first try since the simulator uses real-world protocols such as OpenFlow. A python API is provided for easy creation and sharing of networks can be done through a lone python script [37].

Mininet is an open-source network emulator that provides a virtual environment for network research, education, and testing. It allows users to create and experiment with virtual networks on a single machine, enabling the evaluation of network protocols, testing of networking algorithms, and simulation of network topologies. By leveraging software-defined networking (SDN) principles, Mininet facilitates the creation of complex network configurations, while its integration with SDN controllers enables dynamic network programming.

At its core, Mininet employs Linux network namespaces, virtual Ethernet pairs, and process-based isolation to create lightweight virtual networks. This approach allows for the emulation of various network elements such as switches, routers, hosts, and links, all running as virtual entities on a single host machine. By utilizing these techniques, Mininet provides a cost-effective and scalable alternative to physical network setups, enabling researchers and students to conduct network experiments without the need for expensive hardware.

Mininet is particularly valuable in the field of network research and education. It provides a controlled and reproducible environment for studying the behavior of networks, evaluating network protocols, and testing new networking algorithms. Researchers can easily create and modify network topologies, simulate different network configurations, and observe the impact of their changes. This enables them to gain insights into network performance, scalability, and resilience, while also allowing for the optimization of network resources and the enhancement of overall network efficiency.

Furthermore, Mininet's integration with SDN controllers, such as OpenFlow, enhances its capabilities for network experimentation. Users can connect Mininet topologies to an SDN controller, enabling dynamic control over the network behavior. This programmability aspect of Mininet facilitates the prototyping and evaluation of SDN applications and network management strategies. Researchers can program the network behavior on-the-fly, observe real-time changes, and analyze the impact of different control algorithms or network policies.

In addition to research and education, Mininet serves as a valuable tool for network testing and validation. By emulating various network scenarios, users can assess the performance, scalability, and resilience of their network designs before deploying them in a production environment. Mininet allows for the easy creation and modification of network topologies, making it possible to simulate different network configurations and evaluate their impact on network performance. This helps identify potential bottlenecks, optimize network resources, and enhance overall network efficiency.

Mininet's flexibility and extensibility are further enhanced by its Python-based interface, which allows users to programmatically control and interact with network elements. This feature facilitates the automation of network experiments and enables the integration of Mininet with other tools and frameworks. Researchers and developers can leverage the Python interface to automate complex network setups, perform repetitive tasks, and integrate Mininet with existing network management systems or test frameworks.

4.1.1 Setup

In this thesis. we use the VM (Virtual Machine) distribution of Mininet. This allows for granular control over the resources allocated to Mininet and allows for multiple network simulators to run on the same hardware. To install Mininet, first VirtualBox must be installed. Virtualbox is a VM Hypervisor that allows for VMs to be managed on any operating system. To setup Mininet:

1. **Download Virtual Box**: VirtualBox is openly available on their website. Download the version that corresponds to the test bed operating system

2. **Download Mininet OVF**: Download the latest Mininet Virtual Machine image (usually in OVA or OVF format).

3. **Import the Mininet Virtual Machine image into VirtualBox**:

   a. Open VirtualBox and click on the "File" menu.

   b. Select "Import Appliance" from the dropdown menu.

   c. In the import dialog, click the folder icon and browse to the location where you downloaded the Mininet Virtual Machine image.

   d. Select the Mininet Virtual Machine image file and click "Open".

   e. Review the appliance settings and click "Import" to start the import process.

4. **Configure the Mininet Virtual Machine**:

a. Once the import process is complete, select the imported virtual machine in the VirtualBox main window.

b. Click on the “Settings” button to open the virtual machine settings.

c. Adjust the settings as needed, such as memory allocation and network settings.

d. Click “OK” to save the changes.

5. **Start the Mininet Virtual Machine**

   a. Select the Mininet Virtual Machine in the VirtualBox main window.

   b. Click on the “Start” button to boot up the virtual machine.

6. **Log in to the Mininet Virtual Machine**:

   a. Once the virtual machine has started, a login prompt will appear.

   b. Enter the username and password for the Mininet virtual machine. The default username is “mininet” and the password is “mininet”.

4.1.2 Benefits

Mininet, as an open-source network emulator, offers numerous benefits that contribute to its widespread adoption in the networking community. This section highlights some of the key advantages and advantages of using Mininet for network research, education, and testing.

1. **Cost-Effective**: One of the primary benefits of Mininet is its cost-effectiveness. By providing a virtual network environment on a single machine, Mininet eliminates the need for expensive physical network equipment. Researchers, students, and network engineers can conduct experiments, test network configurations, and evaluate protocols without incurring significant hardware costs. This accessibility promotes innovation and enables a broader audience to engage in network-related activities.

2. **Scalability**: Mininet allows for the creation of scalable network topologies, accommodating a wide range of network sizes and complexities. Users can easily design and modify networks with varying numbers of hosts, switches, and routers. This scalability is particularly useful in testing large-scale network deployments and evaluating network protocols under different traffic loads. Mininet’s ability to scale the network environment helps researchers and engineers analyze the behavior of networks in diverse scenarios and identify potential bottlenecks or performance limitations.

3. **Reproducibility**: Mininet facilitates reproducibility in network research and experimentation. Researchers can capture the state of network emulation, including the network topology, configurations, and traffic patterns, and share it with others. This capability allows for the replication of experiments, verification of results, and collaboration among researchers. The ability to reproduce network conditions is crucial for validating research findings and building upon existing work.

4. **Control and Programmability**: Mininet's integration with SDN controllers, such as OpenFlow, empowers users with enhanced control and programmability over the network behavior. Users can dynamically modify the network flow, configure routing protocols, and experiment with network policies in real-time. This programmability aspect is invaluable for evaluating the performance of SDN applications, exploring different network management strategies, and testing novel network algorithms. The ability to program network behavior on-the-fly accelerates the development and evaluation of new networking concepts.

5. **Flexibility and Extensibility**: Mininet offers a flexible and extensible platform for network experimentation and development. Its Python-based interface allows users to leverage the extensive Python ecosystem and libraries for network automation, analysis, and integration with external tools. Researchers can easily automate experiments, customize network behavior, and integrate Mininet with existing network management systems or test frameworks. This flexibility enables the seamless integration of Mininet into various research and development workflows, enhancing productivity and enabling novel approaches to network experimentation. Mininet facilitates automation through its Python-based interface, enabling users to write scripts and automate repetitive tasks, thus saving time and effort.

6. **Educational Value**: Mininet's user-friendly interface and simplicity make it an ideal tool for network education and training. Students can gain hands-on experience in networking principles, protocols, and configurations by creating and managing virtual networks using Mininet. The ability to visualize and interact with network topologies in a controlled environment enhances understanding and fosters active learning. Mininet serves as an effective platform for teaching networking concepts, enabling students to apply theoretical knowledge in practical scenarios.

7. **Easy Sharing of Scripts**: Mininet allows for the easy sharing of scripts and configurations. Researchers and users can share their network setups, experiments, and network topologies as scripts or configuration files. This ease of sharing promotes collaboration, accelerates research progress, and allows others to replicate experiments and build upon existing work easily.

8. **Multiple Instances on One Machine**: Mininet supports running multiple instances on a single machine, either through the use of multiple virtual machines (VMs) or through the isolation capabilities of the underlying operating system. This enables users to create and test complex network scenarios involving multiple network topologies simultaneously. The ability to run multiple instances on one machine enhances productivity

4.1.3 Drawbacks

While Mininet offers numerous benefits for network research, education, and testing, it is important to consider some of the limitations and drawbacks associated with its usage. This section discusses several key challenges that users may encounter when working with Mininet.

1. **Limited Physical Network Simulation**: Although Mininet provides a virtual network environment, it does not fully replicate the characteristics of a physical network. Certain aspects of network behavior, such as timing constraints, interference, and hardware-specific nuances, may not be accurately emulated. This limitation can affect the realism and accuracy of certain network experiments, especially those that heavily rely on precise timing or physical layer interactions.

2. **Resource Intensive**: Running large-scale network simulations with complex topologies and high traffic volumes can be resource-intensive. Mininet's emulation capabilities rely on the resources of the underlying host machine, including CPU, memory, and network interfaces. As the size and complexity of the emulated network increase, the resource requirements also escalate. Users may encounter limitations in terms of available system resources, which can impact the scalability and performance of Mininet-based experiments.

3. **Limited Protocol Support**: While Mininet supports a wide range of network protocols and offers flexibility in terms of customizing network behavior, there may be instances where certain protocols or features are not fully supported. It is important to verify the compatibility of desired protocols and features with Mininet's capabilities before conducting experiments. Users may need to resort to workarounds or manual configurations to emulate specific protocol behaviors, which can be time-consuming and may introduce inaccuracies.

4. **Lack of Real-world Network Variability**: Mininet's virtual network environment may lack the variability and unpredictability of real-world networks. Network behavior in Mininet is typically deterministic, which means that the same input and conditions will produce the same output consistently. However, real networks are subject to dynamic changes, varying traffic patterns, and unpredictable events. This limitation may hinder the ability to accurately assess the resilience and adaptability of network protocols and algorithms in real-world scenarios.

5. **Learning Curve and Lack of Tutorials**: Mininet can have a steep learning curve, particularly for users who are new to networking or unfamiliar with Linux operating systems and SDN principles. The availability of comprehensive and beginner-friendly tutorials for Mininet is limited, which may pose challenges for users trying to grasp the concepts and effectively utilize the tool. The lack of accessible learning resources can hinder the adoption and utilization of Mininet for network experimentation.

6. **Incomplete/Outdated Documentation**: The documentation for Mininet may be incomplete or outdated in some cases. Users may encounter challenges when seeking detailed information or troubleshooting specific issues. Outdated documentation may not accurately reflect the latest features, functionality, or best practices, leading to confusion or misinterpretation. This can result in additional effort and time spent on finding alternative sources of information or seeking community support.

7. **Unmaintained**: Mininet's development and maintenance may vary over time, and users may encounter instances where certain features or functionalities are no longer actively supported or updated. This lack of ongoing maintenance can result in compatibility issues with newer

operating systems, dependencies, or SDN controllers. Users may need to invest additional effort in adapting Mininet to their specific environment or consider alternative network emulation solutions.

4.2 Ryu SDN Controller

The RYU SDN Controller is an open-source software platform developed in Python that facilitates the development and management of Software-Defined Networking (SDN) applications. This section provides an overview of the RYU SDN Controller, highlighting its key features, architecture, and its significance in the SDN landscape. The RYU SDN Controller serves as a flexible and programmable framework for researchers, network operators, and developers to create and deploy SDN applications capable of controlling the behavior of SDN-enabled networks. Its simplicity, ease of use, and extensive support for various SDN protocols and OpenFlow versions have contributed to its widespread recognition. RYU's Controller architecture is designed to be modular and extensible, enabling developers to easily customize and extend its functionality. It follows a component-based structure, where each module handles specific tasks or functions within the SDN application. This modular design simplifies the development process and encourages code reuse, fostering a collaborative and efficient SDN application ecosystem.

4.2.1 Setup

1. **Set up the Python environment**:

   a. Ensure that Python is installed on your system. Ryu requires Python 3.5 or later.

   b. If Python is not installed, download and install it from the official Python website.

2. **Install Ryu using pip**:

   a. Open a terminal or command prompt.

   b. Run the following command to install Ryu: `pip install ryu`

4.2.2 Benefits

The RYU SDN Controller provides numerous benefits to researchers, network operators, and developers in the Software-Defined Networking (SDN) domain. This section explores the key advantages of using the RYU SDN Controller and how it enhances network control, flexibility, and innovation.

1. **Flexibility and Customizability**: RYU offers a highly flexible and customizable platform for building SDN applications. Its architecture allows developers to tailor the controller to specific network requirements and experiment with different network management strategies. With RYU, users have the freedom to design and implement custom SDN applications that align with their specific use cases and objectives.

2. **Protocol Support**: RYU provides extensive support for various SDN protocols, including OpenFlow, NETCONF, OF-config, and more. This broad protocol compatibility allows RYU to integrate with a wide range of SDN devices and switches, enabling seamless control and management of diverse network infrastructures. The ability to work with multiple protocols enhances interoperability and simplifies the integration of RYU into existing network environments.

3. **Ease of Use**: RYU is designed with simplicity in mind, making it user-friendly and accessible. Its intuitive interface and well-documented APIs facilitate the development process, enabling developers to quickly grasp SDN concepts and start building applications efficiently. RYU's ease of use lowers the learning curve, allowing researchers, network operators, and developers to focus on their specific network goals.

4. **Programmability**: RYU's programmable nature empowers developers to exert granular control over network behavior. Leveraging Python as its programming language, RYU enables rapid application development and facilitates the integration of third-party libraries and tools. This programmability extends the capabilities of RYU, enabling the creation of sophisticated SDN applications tailored to specific networking requirements.

5. **Lightweight and Extensible**: RYU is lightweight in terms of resource utilization, making it suitable for various deployment scenarios. Its efficient resource management ensures optimal performance even in resource-constrained environments. Additionally, RYU's extensibility allows developers to incorporate additional functionalities and features as needed, further enhancing its capabilities and adaptability.

6. **Real-World Deployment**: RYU has been successfully deployed in real-world applications across various industries, including telecommunications, data centers, and enterprise networks. Its versatility and compatibility with different SDN protocols make it suitable for a wide range of network environments. RYU empowers organizations to implement SDN solutions that enhance network agility, scalability, and management efficiency.

4.2.3 Drawbacks

While the RYU SDN Controller has provided significant contributions to the Software-Defined Networking (SDN) landscape, it is important to consider its drawbacks. This section explores some of the key limitations associated with RYU.

1. **Learning Curve**: RYU involves a steep learning curve, particularly for users new to SDN concepts and programming. The complex nature of SDN and the intricacies of RYU's architecture require a solid understanding of network protocols, Python programming, and SDN principles. The lack of comprehensive tutorials and learning resources can further hinder the learning process.

2. **Limited GUI Support**: RYU primarily relies on a command-line interface (CLI) and lacks extensive graphical user interface (GUI) support for configuration and monitoring. Users

who prefer visual representations and intuitive GUI-based management interfaces may find RYU less user-friendly compared to controllers with robust GUI capabilities.

3. **Scalability Concerns**: RYU may face challenges in scaling to large and complex network environments. As the size of the network or the number of SDN devices increases, RYU's performance may be affected, potentially leading to performance bottlenecks. Users operating large-scale SDN deployments with high traffic volumes should carefully evaluate RYU's scalability and performance capabilities.

4. **Limited Vendor Support**: RYU's open-source nature means that it may lack direct support from networking equipment vendors. Users relying on specific vendor-dependent features or proprietary extensions may encounter difficulties in integrating their hardware with RYU. The lack of vendor-specific support can limit interoperability and advanced functionalities in certain networking environments.

5. **Lack of Maintenance**: One significant drawback is that RYU is no longer actively maintained. The absence of ongoing development and updates raises concerns about the controller's compatibility with newer networking technologies and protocols. The lack of maintenance may result in outdated features, potential security vulnerabilities, and limited community support for bug fixes or enhancements.

6. **Advanced Features and Functionality**: RYU may not offer the same extensive set of advanced features and functionalities as some other SDN controllers. Certain specialized use cases or specific requirements may necessitate additional features that may not be readily available in RYU. Users with complex SDN environments or specific network management needs should carefully assess whether RYU can meet all their requirements.

4.3 Gurobi Optimization Library

Gurobi is a state-of-the-art mathematical optimization solver renowned for its powerful algorithms and cutting-edge solver technologies. Developed by Gurobi Optimization, LLC, it is designed to solve a wide range of optimization problems, including linear programming (LP), mixed-integer programming (MIP), and quadratic programming (QP), among others. Gurobi is known for delivering high-performance optimization solutions that combine speed, accuracy, and robustness. We use this optimization to create pre-determined network load scenarios to determine the best route for all packets.

4.3.1 Setup

1. **Obtain a License**: Obtain a valid Gurobi license from the Gurobi website or request an academic license if applicable.

2. **Download Gurobi**:

a. Navigate to the ”Download Center” or ”Downloads” section.

b. Select the appropriate Gurobi version for your Linux distribution and download the Gurobi installation package in tar.gz format.

3. **Install Gurobi**:

a. Navigate to the directory where the Gurobi installation package is downloaded.

b. Use the following command to extract the contents of the tar.gz archive: `tar -xzvf gurobi*.tar.gz`

c. Run the Gurobi installer as the superuser using the following command: `sudo python3 setup.py install`

4.3.2 Benefits

The utilization of Gurobi in network traffic engineering brings numerous benefits to optimize and enhance network performance. By leveraging Gurobi's powerful optimization capabilities, network operators can address complex challenges associated with traffic management and achieve efficient utilization of network resources. The benefits of using Gurobi for network traffic engineering can be summarized as follows:

1. **Traffic Flow Optimization**: Gurobi enables the optimization of network traffic flows by formulating mathematical models that consider factors such as network congestion, bandwidth requirements, and latency constraints. By finding optimal traffic routing and resource allocation solutions, Gurobi helps minimize congestion, reduce packet loss, and optimize the overall flow of network traffic. This results in improved network performance, reduced latency, and enhanced user experience.

2. **Load Balancing**: Gurobi aids in load balancing by optimizing the distribution of network traffic across multiple paths and links. Through its optimization capabilities, Gurobi can allocate network resources effectively, ensuring balanced utilization and preventing bottlenecks. Load balancing optimization facilitates optimal resource allocation and improves network capacity, leading to better performance and increased scalability.

3. **Resource Efficiency**: Gurobi aids in optimizing the utilization of network resources, such as bandwidth and computing power. By formulating resource allocation problems as optimization models, Gurobi determines the optimal allocation of resources across network devices and links. This optimization minimizes resource wastage, ensures efficient resource utilization, and reduces operational costs. Network operators can achieve better resource efficiency and cost-effectiveness by leveraging Gurobi's optimization capabilities.

4. **Scalability**: Gurobi's scalability allows it to handle large-scale network traffic engineering problems with a high number of variables and constraints. Whether dealing with massive

traffic volumes or complex network topologies, Gurobi's efficient algorithms and solver technologies can effectively optimize traffic management and resource allocation. This scalability ensures that Gurobi can be applied to real-world network environments, accommodating growing network demands and ensuring optimal performance as networks expand.

5. **Flexibility and Customization**: Gurobi offers a flexible modeling environment, supporting various programming languages and providing APIs and libraries for easy integration into existing network management systems. This flexibility allows network operators to customize optimization models to their specific requirements, incorporating unique constraints, objectives, and network policies. Gurobi's customization capabilities enable tailored solutions that align with specific network traffic engineering needs.

4.3.3 Drawbacks

While using Gurobi for traffic engineering can be beneficial, there are certain drawbacks that need to be taken into account. These drawbacks are as follows;

1. **Complexity of Model Formulation**: Gurobi requires the formulation of mathematical optimization models to represent network traffic engineering problems. The process of formulating these models can be complex, requiring expertise in mathematical modeling and optimization techniques. Network operators and engineers may need to invest time and effort in understanding and formulating the models correctly, which can be a challenge, particularly for those without extensive optimization background.

2. **Computational Resources**: Gurobi's optimization algorithms and solver technologies are computationally intensive, especially for large-scale network traffic engineering problems. Solving complex optimization models may require substantial computational resources, including memory and processing power. Network operators must ensure that their systems have sufficient resources to handle the computational demands of Gurobi, especially in scenarios with high network traffic volumes or complex network topologies.

3. **Solution Interpretation**: While Gurobi provides optimal solutions to network traffic engineering problems, interpreting and implementing these solutions can be challenging. The solutions generated by Gurobi may involve changes in traffic routing, resource allocation, or network configurations. Network operators must carefully interpret and translate these solutions into actionable changes within the network infrastructure. Implementing complex optimization solutions in a live network environment requires careful planning and coordination to avoid disruption or unintended consequences.

4. **Dependency on Problem Formulation**: The effectiveness of Gurobi heavily relies on the accuracy and completeness of the mathematical model formulation. Incorrect or incomplete modeling can lead to suboptimal solutions or infeasible outcomes. Network operators need to invest significant effort in accurately representing the network traffic engineering problem in the optimization model. Any errors or oversights in the model formulation can impact the quality of the optimization results.

5. **Cost**: Gurobi is a commercial optimization solver and, as such, has associated licensing costs. Depending on the scale and complexity of the network traffic engineering requirements, the licensing fees for Gurobi may pose a financial burden. Organizations need to evaluate the cost-effectiveness of using Gurobi compared to alternative optimization solvers or in-house solutions.

6. **Learning Curve and Support**: Gurobi's utilization in network traffic engineering may involve a learning curve, especially for individuals who are not familiar with optimization techniques or the Gurobi software. Acquiring the necessary knowledge and skills to effectively use Gurobi may require training or external expertise. Additionally, while Gurobi provides documentation and support resources, the availability of comprehensive tutorials or community support specifically focused on network traffic engineering may be limited.

Chapter 5

Experiment & Results

5.1 Executing a Crossfire Attack

The execution of a crossfire attack, in theory, appears straightforward. By flooding a specific link, the attacker aims to overwhelm it with traffic. This process involves identifying potential routes that utilize the targeted link and generating low-intensity requests across those routes to flood the link effectively. However, in practice, executing a crossfire attack can be challenging due to the limited availability of information regarding the specific routes taken by packets. The lack of public access to this crucial routing data presents a significant hurdle for attackers attempting to orchestrate such attacks. In this section, we delve into the intricacies of executing a crossfire attack, exploring the methodologies used to overcome these obstacles and the implications of this type of attack on network performance and security.

5.1.1 Network Topology

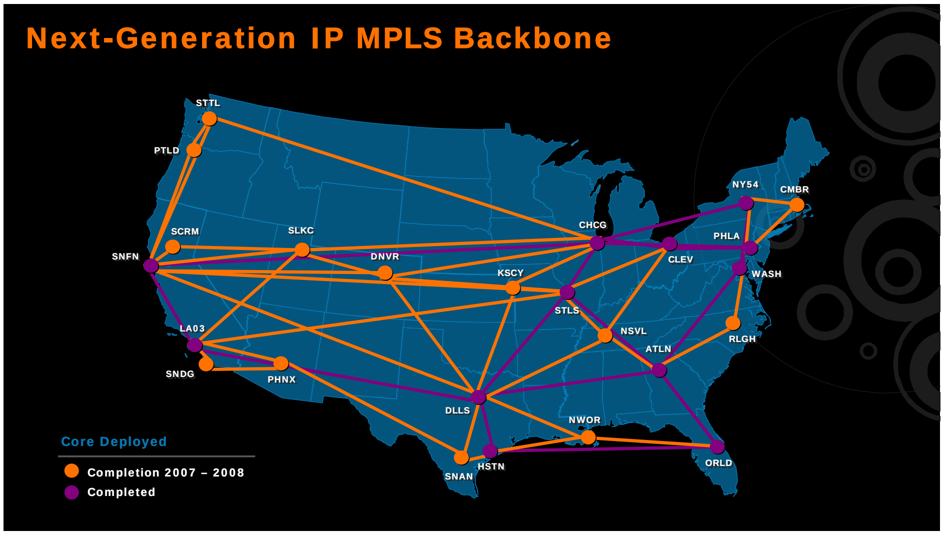


Figure 5.1: Test Network Architecture [38] [1]

Our crossfire attack was executed on a network simulated by the MiniNet network simulator and the RYU SDN controller. These tools allowed for the creation and management of a virtual

[1] Reproduced with express permission from authors

network environment for experimentation and analysis. To set up the network, a Python script was employed, which utilized the ATT North America Backbone network from The Internet Topology Zoo [38] as the basis for the network configuration. The network topology, as depicted in Figure 5.1, provides a visual representation of the structure and interconnections of the various network components. It showcases the arrangement of links within the simulated network. To introduce additional functionality and explore specific scenarios, a Mobile Edge Computing (MEC) server was strategically placed between the ORLD and CLEV nodes. The MEC server served as a centralized computing platform that brought computing resources closer to the network edge, enabling efficient processing and analysis of data generated within the network. In this particular setup, the MEC server had a direct connection between the ORLD and CLEV nodes, facilitating seamless communication and data exchange between them. The choice of the ATT North America Backbone network from The Internet Topology Zoo [38] was driven by its complexity and size, which allowed for a more realistic simulation of network traffic. By utilizing a network with a sufficient number of components and diverse connections, researchers and analysts could better understand and evaluate the performance, scalability, and security aspects of network systems under various conditions

5.1.2 Simulation Environment Setup

Table 5.1 outlines the tools used for the simulation and their version. We followed these steps to simulate the testing on Mininet:

1. Obtain a graph for the topology from Topology Zoo [38]
2. Manually create a MiniNet python script based on the visualization of the graph.
3. Modify the previously created Python script to generate random traffic demand between pair of hosts for the benign traffic.
4. Determine nodes across the target links and add them to the adversary list
5. Have the adversary list begin flooding low-intensity requests across the shared link
6. Collect data using Wireshark

5.1.3 Attack Methodology

Once the network setup is complete, the testing scenario involves a sequence of events. First, a ping request is sent from the NY54 node, which represents an external connection, to the MEC (Multi-Access Edge Computing) server. This initial interaction confirms the connectivity between these nodes.

Following the establishment of the network, servers within the network begin initiating HTTP connections randomly across the infrastructure. Approximately 80% of the servers are engaged

Table 5.1: Tools used in Simulation and their version

Tools	Version
Mininet	2.3.0-210211-ubuntu-20.04.1-legacy-server-amd64-ovf
Ryu	4.3.4
Python	3.8.10
Gurobi	3.8.10
Python	3.8.10
VirtualBox	6.1.38_Ubuntu r153438
Ubuntu Host	20.04.6 LTS

in requesting HTTP resources at any given time. This random traffic generation simulates unpredictable and legitimate network activity, replicating real-world usage patterns.

After a period of 30 seconds dedicated to legitimate traffic flows, the attack commences. Multiple zombie servers (compromised devices controlled by the attacker), start streaming video traffic over TCP (Transmission Control Protocol) connections to each other. TCP is deliberately chosen for this attack to obscure the nature of the traffic being transmitted. To further obfuscate the content, the videos are streamed over HTTPS (Hypertext Transfer Protocol Secure), making it difficult to distinguish the packets as video traffic based on their packet types.

Each zombie server strategically selects its destinations, ensuring that the attacking video packets pass through, but do not end at, either the ORLD or CLEV nodes. This strategic routing aims to block external connections to the MEC server. Consequently, the switches connecting these networks experience congestion due to the significant volume of data flowing through each link.

The attack is executed in three phases, with a third of the zombie servers initiating the attack during each phase. This staged approach helps distribute the attack traffic and potentially evade detection or mitigation measures.

Throughout the entire process, packet headers are captured using Wireshark, a widely used network protocol analyzer. Despite the use of HTTPS, which encrypts the content of the packets, the packet headers remain visible. Therefore, in this scenario, the network would only have access to information contained in the packet headers to analyze and identify the attack.

5.1.4 Results

After running the network for 5 minutes, the ping round trip times were collected and plotted in Figure 5.2, showcasing the temporal changes in the network's responsiveness. To provide a clearer representation of the data, a moving average was also plotted as a line, indicating the overall trend. During the initial stage, prior to any active attacks, the ping round trip times remained relatively stable, with values ranging between 50 to 100 milliseconds. This indicated a reasonably efficient network performance and responsiveness. However, as the attack commenced and the zombie servers came into play, the average ping round trip times experienced a noticeable surge. The ping increased to approximately 500 milliseconds, demonstrating a significant delay in communication between the source and destination. This jump in response time highlighted the impact of the attack

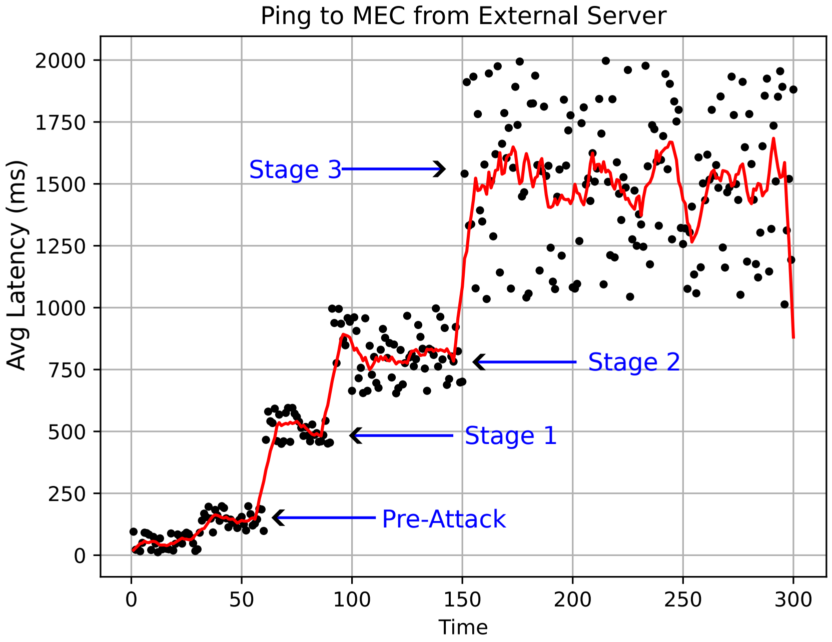


Figure 5.2: Average Ping Over Time

on the network's overall performance, causing a tangible slowdown in data transmission. Following the first phase, the attack intensified further, leading to a subsequent increase in the average ping round trip times. The values climbed to around 800 milliseconds, further exacerbating the latency issues experienced by the network. This escalation in response time served as an indicator of the mounting strain on the network's resources and its ability to handle the incoming traffic effectively. Finally, as the attack progressed into its third phase, the ping round trip times reached a notable peak, with the values hovering around 1500 milliseconds. This significant increase in latency pointed to a severe degradation in the network's performance, with the delay between request and response becoming increasingly pronounced. The prolonged response times were indicative of the network's struggle to cope with the heightened load imposed by the ongoing attack, causing substantial disruptions to normal operations.

Overall, the plotted data in Figure 5.2 illustrates the adverse effects of the attack on the network's ping round trip times. The upward trajectory of the moving average line clearly demonstrates how the average ping gradually worsens as the attack progresses through its different phases, highlighting the detrimental impact of such attacks on network performance and stability.

5.2 Detecting a Crossfire Attack

In the following subsections, we delve into the specific techniques used to detect crossfire attacks by analyzing packet headers. We explore the data collection process, statistical analysis methods, and the implementation of neural networks to develop predictive models. By understanding these

Table 5.2: Initial ANOVA Full Model Test

Model	-LogLikelihood	DF	ChiSquare	Prob>ChiSq
Difference	176.18978	12	352.3796	<.0001
Full	408.10276			
Reduced	584.29254			

techniques, we can gain insights into the challenges and advancements in detecting and mitigating crossfire attacks in today's encrypted network environments.

5.2.1 Classifying Adversarial Traffic

Model Setup

After conducting the experiment outlined in Section 5.1, a comprehensive data collection process was initiated to gather packet headers for every packet transmitted throughout the network. A substantial amount of data was collected, comprising approximately 30,000 packets. These packets were captured and recorded for further analysis and evaluation. To extract meaningful insights and uncover patterns within the collected packet data, the dataset was processed and imported into JMP Pro, a powerful statistical analysis software. By utilizing JMP Pro's extensive capabilities, researchers were able to explore the dataset and perform a range of advanced analytical techniques. In the initial analysis, an Analysis of Variance (ANOVA) was employed to assess the statistical significance of various factors or variables within the packet headers. ANOVA allowed for the identification of any significant differences or relationships between different groups or categories of packets, providing valuable insights into the effects of different network conditions or configurations on packet behavior. Additionally, a neural network was implemented to further investigate and model the complex relationships within the collected packet data. Neural networks, as a machine learning technique, are capable of identifying hidden patterns and learning from complex datasets. By training a neural network on the packet headers, we aim to develop a predictive model that could effectively classify and interpret different packet characteristics or behaviors.

ANOVA

After conducting the initial analysis of variance (ANOVA) on the collected packet data, the results were summarized and presented in Table 5.2 and Table 5.3. These tables provided insights into the significant predictors of adversarial packets within the dataset.

ANOVA, short for Analysis of Variance, is a statistical technique used to compare the means of two or more groups or treatments and determine if there are significant differences among them. It is a parametric test that assesses the variability within and between groups to draw conclusions about population means. ANOVA provides a comprehensive framework for understanding the sources of variation and making statistical inferences based on the observed data.

The fundamental principle behind ANOVA is the partitioning of total variance into different components: the between-group variance and the within-group variance. The between-group vari-

Table 5.3: Initial ANOVA Parameter Estimates

Term		Estimate	Std Error	ChiSquare	Prob>ChiSq
Intercept	Unstable	-6.18762050	89752.103	0.00	0.9999
tcp.window_size		0.00057399	8.7147e-5	43.38	<.0001
tcp.len		-0.00051130	0.0003489	2.150	0.1428
tcp.stream		-0.08826740	0.0343179	6.620	0.0101
tcp.flags[0x00000010]	Unstable	-15.9381260	89752.102	0.000	0.9999
tcp.flags[0x00000011]	Unstable	9.78867015	116551.53	0.000	0.9999
- tcp.flags[0x00000012]	Unstable	12.0348821	117261.45	0.000	0.9999
tcp.flags[0x00000018]	Unstable	-15.0194920	89752.102	0.000	0.9999
tcp.analysis.ack_rtt		-63.1695000	7.2849454	75.19	<.0001
frame.time_relative		0.11828939	0.0393738	9.030	0.0027
frame.time_delta		1.88407238	1.1200274	2.830	0.0925
tcp.time_relative		25.6221427	3.1284271	67.08	<.0001
tcp.time_delta		3.24388117	4.7044249	0.480	0.4905

ance represents the variability among the means of different groups or treatments, indicating the extent to which group means differ from each other. On the other hand, the within-group variance captures the inherent variability within each group, reflecting the amount of variation that can be attributed to random fluctuations or individual differences within the groups.

To perform ANOVA, certain assumptions must be met. These include the assumption of independence, where the observations within each group are independent of each other. Additionally, the assumption of normality states that the data within each group should follow a normal distribution. Lastly, the assumption of homogeneity of variances requires that the variability within each group is approximately equal.

ANOVA produces an F-statistic, which measures the ratio of the between-group variability to the within-group variability. By comparing this statistic to a critical value based on the chosen significance level, researchers can determine whether the observed differences among group means are statistically significant or simply due to chance. If the F-statistic exceeds the critical value, it indicates the presence of significant differences between at least some of the groups.

The notation “Prob $>$ ChiSq” refers to the p-value associated with the chi-square test statistic in statistical analysis. The chi-square test is a statistical test used to determine whether there is a significant association between categorical variables. It compares the observed frequencies in different categories with the frequencies that would be expected under a null hypothesis of independence or homogeneity.

The p-value is a measure of the evidence against the null hypothesis. It represents the probability of observing a test statistic as extreme as, or more extreme than, the one calculated from the data, assuming that the null hypothesis is true. In the case of the chi-square test, a smaller p-value indicates stronger evidence against the null hypothesis and suggests that there is a significant association between the categorical variables.

When the p-value is reported as “Prob $>$ ChiSq,” it means that the p-value is less than a certain

Table 5.4: Revised ANOVA Full Model Test

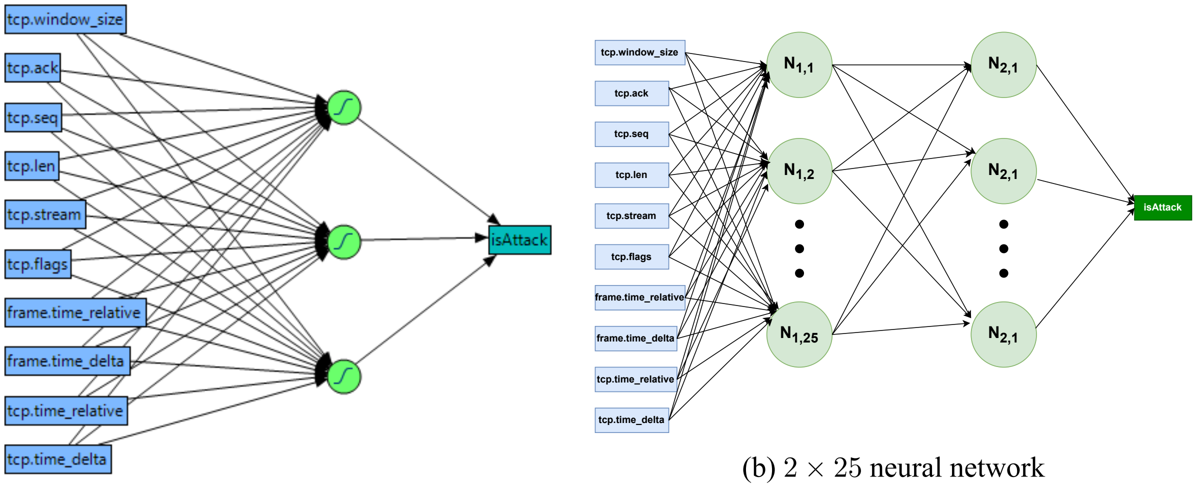
Model	-LogLikelihood	DF	ChiSquare	Prob>ChiSq
Difference	105.98066	4	211.9613	<.0001
Full	478.31188			
Reduced	584.29254			

Table 5.5: Revised ANOVA Parameter Estimates

Term	Estimate	Std Error	ChiSquare	Prob>ChiSq
Intercept	-13.9094560	3.0268445	21.120	<.0001
tcp.window_size	0.000396320	6.8154e-5	33.810	<.0001
tcp.analysis.ack_rtt	-38.0040650	3.7611744	102.10	<.0001
frame.time_relative	0.01735100	0.0023833	53.000	<.0001
tcp.time_relative	15.8395049	2.3819797	44.220	<.0001

significance level, often denoted by α (e.g., $\alpha = 0.05$). The ChiSq in the notation refers to the chi-square statistic used in the test. The specific p-value is typically not explicitly mentioned but can be understood to be smaller than the significance level α.

From the data in Table 5.2 and Table 5.3, it was observed that the window size, ack_rtt, and time_relative variables exhibited the largest influence on classifying packets as adversarial. To refine and streamline the analysis, non-significant factors were subsequently eliminated from the ANOVA. This process aimed to focus on the most relevant and influential variables. The updated ANOVA results were then presented in Table 5.4 and Table 5.5. reflecting the reduced set of predictors. The revised ANOVA indicated that the combined factors of window size, ack_rtt, frame time, and tcp time remained significant in determining whether a packet was adversarial or not. These variables collectively formed the basis of the model for classifying packets as adversarial. However, it should be noted that this model may not be practical in scenarios involving nodes with significant delays. In such cases, the presence of significant delays could result in the model classifying all packets as adversarial, potentially leading to the blocking of all connections. This situation could exacerbate the impact of an attack, as legitimate connections might also be blocked due to the model's overly cautious classification. Furthermore, during an attack, the delay experienced by packets traversing congested links could pose a challenge. In such situations, the model's tendency to classify all packets as adversarial might exacerbate network congestion and intensify the adverse effects of the attack. These considerations highlight the limitations and potential drawbacks of the ANOVA-based model. It underscores the need for further refinement and exploration of alternative approaches that can address the issues related to significant delays and congested links. By developing more robust and adaptive models, it may be possible to enhance the effectiveness and practicality of packet classification in the context of adversarial attacks.



(a) 2 × 3 neural network

(b) 2 × 25 neural network

Figure 5.3: Neural Network Diagrams Using Hyperbolic Tangent Nodes

Table 5.6: Confusion Matrix for Training Data for 1 × 3 neural network

		Predicted	
		Adversarial	Benign
Actual	Adversarial	19206	33
	Benign	242	4928

Table 5.7: Confusion Matrix for Validation Data for 1 × 3 neural network

		Predicted	
		Adversarial	Benign
Actual	Adversarial	6373	12
	Benign	72	1679

Table 5.8: Confusion Matrix for Training Data with 2 × 25 neural network

		Predicted	
		Adversarial	Benign
Actual	Adversarial	19308	28
	Benign	140	4933

Table 5.9: Confusion Matrix for Validation Data with 2 × 25 neural network

		Predicted	
		Adversarial	Benign
Actual	Adversarial	6375	10
	Benign	31	1720

Neural Network

In addition to the ANOVA analysis, a neural network model was developed based on the selected criteria. The architecture and structure of the neural network were visualized in Figure 5.3, providing a diagrammatic representation of its layers and connections. To evaluate the performance of the neural network model, confusion matrices were constructed for both the training and validation datasets, as shown in Table 5.6 and Table 5.7, respectively. The confusion matrices allowed for a comprehensive assessment of the model's predictive capabilities by displaying the classification results. In the training data, the neural network exhibited high accuracy in correctly predicting legitimate packets, achieving an accuracy rate of 99.82%. This indicates that the model accurately identified the vast majority of non-adversarial packets. For attack packets, the model achieved a correct prediction rate of 95.3%, meaning it accurately identified a significant portion of the adversarial packets during training. In the validation data, which serves as an independent dataset to assess the model's generalization capability, the neural network performed similarly well. It correctly identified 99.81% of legitimate packets, demonstrating its ability to accurately classify non-adversarial traffic. For attack packets, the model achieved a correct prediction rate of 95.88%, indicating its capacity to correctly identify a substantial portion of adversarial packets in unseen data. The high accuracy rates achieved by the neural network model in both training and validation data highlight its effectiveness in distinguishing between legitimate and adversarial packets. The model's ability to correctly classify a large majority of legitimate packets provides assurance that it can accurately identify normal network traffic, minimizing the risk of false positives. Similarly, the model's capability to accurately detect a significant portion of attack packets suggests its potential for effectively identifying malicious network activity. These results demonstrate the promising performance of the neural network model in packet classification. However, it is essential to continually monitor and evaluate the model's performance using real-world data, as the efficacy of such models can vary depending on the specific characteristics and dynamics of the network environment. Regular updates and refinements to the model can further enhance its accuracy and reliability in distinguishing between legitimate and adversarial packets. Other models were also trained, such as a 2-layer, 25-node neural network was trained. Tables 5.8 and 5.9 show the training and validation data for the 25x25 neural network. Since the performance of this network was only slightly better than the original neural network, it would make sense to use the original neural network to minimize the processing power for each packet, thereby reducing the decision time.

5.3 Optimizing Traffic using Binary Linear Program

To evaluate the effectiveness of our optimization model, referred to as OptimizingDemandsWithGurobi in Section 3.3, we conducted experiments on various network typologies. To create these network scenarios, we employed a random network generation approach that allowed us to generate networks with different sizes and topologies.

For our evaluation, we generated random networks with 5, 10, 50, and 100 nodes, each representing unique challenges and characteristics. These networks were designed to emulate real-world

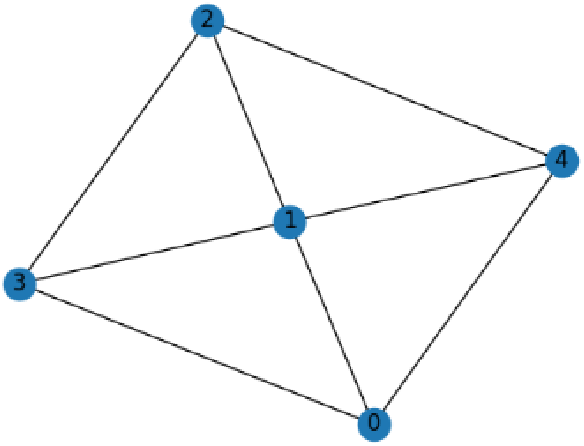


Figure 5.4: Sample Gurobi Topology

scenarios, where nodes represent network devices and links represent connections between them.

In our optimization process, we utilized the powerful optimization solver Gurobi. By leveraging Gurobi's optimization capabilities within our OptimizingDemandsWithGurobi model, we were able to determine optimal routing decisions while considering factors such as network congestion and link capacity. This approach enabled us to assess the scalability and performance of our model across a wide range of network sizes and topologies.

By comparing the routing decisions produced by our OptimizingDemandsWithGurobi model against the default shortest-path routing, we could evaluate the effectiveness of our approach in improving network performance and resilience.

5.3.1 Network Topologies

In order to assess the effectiveness of our optimization model, we conducted a series of experiments on various network typologies. We purposely selected networks of different sizes to evaluate both the scalability and performance of our approach. Specifically, we executed the optimization library on networks comprising 5, 10, 50, and 100 nodes.

As an illustrative example, Figure 5.4 showcases the 5-node topology that we utilized for this evaluation. This particular topology was thoughtfully designed to incorporate multiple paths between any source and destination within the network. By incorporating these alternative routes, we aimed to maximize the network's robustness and fault tolerance, enabling efficient rerouting in the event of potential disruptions.

To thoroughly examine the impact of our optimization approach, we replicated these experiments across networks of varying scales. We started with a small-scale network of 5 nodes, progressively increasing the complexity and size of the networks to 10, 50, and finally 100 nodes. This systematic exploration allowed us to evaluate how well our model adapted and performed in different network scenarios. By running our Gurobi optimization model on these diverse network typologies and comparing its routing decisions with the non-optimized shortest path routing, we were able to assess the efficacy and efficiency of our approach. These experiments provided valuable insights into the scalability and performance of our optimization model, enabling us to fine-tune and improve its capabilities for real-world network deployments.

5.3.2 Methodology

Model Generation

The generation of our Gurobi optimization model was facilitated by employing a readily available script obtained from GitHub [35]. This script served as a powerful tool that streamlined the process of creating the model by leveraging the network topology information provided to us. To initiate the optimization process, we supplied the script with the network topology represented in the graphML format. This format allowed us to conveniently capture the connectivity and relationship between nodes in a structured and machine-readable manner. With the topology information at hand, we executed the script, which commenced the generation of the Gurobi optimization model. It is important to note that as the size of the network topology increased, the computational time required for the Gurobi optimization process exhibited an exponential growth pattern. This observation is consistent with the inherent complexity associated with solving optimization problems on larger networks. Nevertheless, despite the increased computational demand, the script efficiently managed and executed the optimization tasks, producing the desired results.

Once the Gurobi optimization model was generated, the script produced two crucial output files in CSV (Comma-Separated Values) format. The first output file contained the network scenarios, providing a comprehensive overview of the various potential configurations that could be considered. Each scenario represented a different set of routing decisions, allowing us to explore a wide range of possibilities for optimizing the network's resilience against crossfire attacks. The second output file contained information about the available paths within the network. This file captured the viable routes and their respective characteristics, such as latency, bandwidth, or any other relevant metrics. By having access to this data, we gained valuable insights into the network's existing capabilities and potential routing options. This information formed the basis for evaluating the performance and effectiveness of different routing decisions within the generated network scenarios.

Routing using Ryu

Once the Gurobi optimization model was generated and the output CSV files containing network scenarios and available paths were obtained, further steps were taken to convert this data into a format that could be understood by Mininet and Ryu. To accomplish this, we employed a data processing script designed specifically for this purpose. The data processing script was responsible for parsing the CSV files and transforming the relevant information into a format compatible with Mininet and Ryu. This involved extracting details about the network topology, routing decisions, available paths, and any other pertinent data required for network emulation and control. The output from the data processing script was then piped into our custom Ryu controller. Ryu is an open-source software-defined networking (SDN) framework that enables the development of network applications and controllers. By directing the processed data to the Ryu controller, we established the necessary communication and control between the simulated network and the Gurobi optimization model.

Following the setup of the Ryu controller, we utilized the Mininet auto-net script to facilitate the creation and teardown of network scenarios for each of the generated network configurations. The auto-net script, located in Appendix A.1, played a crucial role in automating the deployment and management of the network scenarios. The auto-net script operated in a sequential manner. First, it initiated the Ryu controller, establishing the control plane for the network. Subsequently, it triggered the network creation process within Mininet, deploying the network infrastructure based on the specified topology and routing decisions obtained from the Gurobi optimization model. Once the network was successfully launched, the auto-net script proceeded to execute the defined traffic scenarios for a predetermined duration. This entailed simulating the traffic flows, emulating network communication, and monitoring the performance and behavior of the network under various conditions. Throughout this process, bandwidth measurements were recorded using the iperf tool, allowing us to quantitatively assess the network's performance in terms of data transfer rates.

Upon completing the specified duration for each traffic scenario, the auto-net script initiated the network teardown process. This involved dismantling the virtual network infrastructure created within Mininet, ensuring the resources were released and made available for subsequent scenarios. Concurrently with the execution of network scenarios, the auto-net script continuously captured and recorded the bandwidth measurements obtained from the iperf tool. This data was vital for evaluating the impact of different routing decisions on the network's performance, enabling us to compare and analyze the efficiency and effectiveness of the Gurobi optimization model.

5.3.3 Results

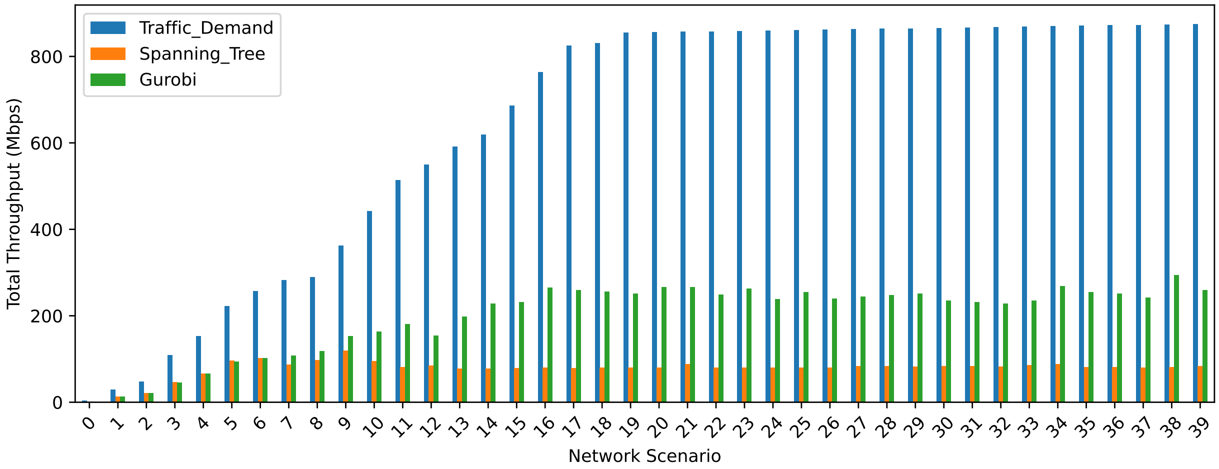


Figure 5.5: Network Bandwidth

For the 5 node network, the results of the Gurobi optimization are presented in Figure 5.5. The graph illustrates the performance comparison between the Gurobi-optimized network and the

Table 5.10: Traffic Demands by Network Scenario

Network Scenario	Total Bandwidth (Mbps)	Average Bandwidth (Mbps)
0	3.548	0.142
1	29.833	1.193
2	48.407	1.936
3	109.075	4.363
4	153.473	6.139
5	222.560	8.902
6	257.327	10.293
7	282.930	11.317
8	289.446	11.578
9	362.989	14.520
10	442.276	17.691
11	514.479	20.579
12	550.033	22.001
13	591.462	23.658
14	619.085	24.763
15	686.898	27.476
16	764.495	30.580
17	825.347	33.014
18	831.233	33.249
19-39	855.173	34.207

simple network configuration. Initially, for network scenarios 1-10, both the Gurobi optimized network and the simple network exhibit similar performance. This similarity can be attributed to the low overall load on the network during these scenarios. As the network scenarios progress, the overall load gradually increases. In the initial scenarios, the shortest path in the network is not fully saturated, resulting in the Gurobi optimization making the same routing decisions as the simple network. However, beyond scenario 10, the Gurobi-optimized network begins to outperform the simple network. This improvement can be attributed to the fact that as the network load increases, the shortest path may no longer be the fastest path due to link saturation. The Gurobi optimization algorithm considers alternative routing decisions that alleviate congestion and improve overall network performance.

Table 5.10 details the network load for each network scenario in our experiment. The "Total Demand (Mbps)" column represents the total amount of traffic demand in megabits per second (Mbps), while the "Average Demand (Mbps)" column represents the average demand per scenario. Starting from network scenario 0, the total demand gradually increases. In scenario 0, the total demand is 1.43075 Mbps, with an average demand of 0.05723 Mbps. As we progress through the network scenarios, both the total demand and average demand continue to rise. By network scenario 19, the network becomes fully saturated, with a total demand of 344.82785 Mbps and an average demand of 13.793114 Mbps. In this state, further network scenarios do not increase the total demand but rather alter the traffic distribution without changing the overall demand. The data in Table 5.10 provides insights into the increasing network load as the scenarios progress, allowing for a comprehensive understanding of the network's capacity and the demands placed upon it.

Toward the end of the experiment, the Gurobi-optimized network achieves significantly higher bandwidths compared to the simple network. Specifically, the Gurobi optimization achieves bandwidths of 120 Mbps, while the simple network only achieves around 40 Mbps. This represents a substantial increase in throughput, with the Gurobi optimization resulting in over a three-fold improvement in bandwidth. By enhancing the overall throughput and robustness of the network using identical infrastructure, the Gurobi optimization demonstrates its effectiveness in maximizing network performance. It identifies alternative routing decisions that consider the network's current state and load distribution, leading to improved bandwidth utilization and reduced congestion.

Unfortunately, when attempting to apply the Gurobi optimization to larger networks, technological challenges were encountered. The Mininet simulator and Ryu controller experienced crashes, preventing the execution of the optimization on these larger network configurations. Despite efforts to resolve these issues, the lack of documentation and time constraints limited the ability to address and overcome these problems.

Chapter 6

Conclusion & Future Work

As availability attacks grow in complexity, scale, and number, additional research is needed to create methods to defend against such attacks. One attack, the crossfire attack, is particularly difficult to detect as adversarial traffic flows are currently indistinguishable from legitimate traffic flows. The holistic flow perspective granted by software-defined networks may assist in detecting such an attack. The proposed thesis aims to find a novel method of detecting a crossfire attack. Current methods treat attacks and legitimate flashmobs the same, causing issues when a large influx of users attempt to connect to a site legitimately. The crossfire attack continues to be virtually undetectable since no end nodes are being targeted.

In this thesis, we have explored the field of network traffic engineering and its significance in ensuring efficient and optimized utilization of network resources. We have examined the challenges faced by network operators in managing traffic flow, load balancing, resource efficiency, and scalability, and we have discussed the potential benefits of using optimization solvers such as Gurobi in addressing these challenges. Through our analysis, we have identified the powerful optimization capabilities offered by Gurobi and how they can contribute to traffic flow optimization, load balancing, resource efficiency, and scalability in network traffic engineering. By formulating mathematical models and leveraging Gurobi's algorithms and solver technologies, network operators can achieve optimal traffic routing, resource allocation, and utilization, resulting in improved network performance, reduced latency, and enhanced user experience.

However, we have also acknowledged certain drawbacks associated with using Gurobi in traffic engineering. The complexity of model formulation, computational resource requirements, solution interpretation, dependency on accurate problem formulation, licensing costs, and learning curve are important considerations that network operators need to take into account when adopting Gurobi or any commercial optimization solver.

Additionally, we have presented a case study involving a crossfire attack on a simulated network environment. By executing the attack and analyzing the network's responsiveness, we have demonstrated the detrimental impact of such attacks on network performance and stability. The results highlighted the increase in ping round trip times, indicating the strain on network resources and the subsequent degradation of performance. Furthermore, we discussed the challenges of detecting crossfire attacks, particularly in encrypted HTTPS traffic. By analyzing packet headers and employing statistical analysis techniques such as ANOVA, researchers can classify adversarial traffic based on header information, contributing to the development of effective detection mechanisms.

In conclusion, network traffic engineering plays a crucial role in optimizing network performance, resource utilization, and user experience. Optimization solvers like Gurobi offer powerful capabilities to address complex traffic management challenges. However, careful consideration of the associated drawbacks and challenges is necessary. By conducting experiments, collecting and analyzing data, and employing advanced analytical techniques, we can enhance our understanding of network behavior and develop effective strategies to detect and mitigate adversarial traffic. These findings contribute to the field of network traffic engineering and can aid network operators in building more robust and efficient network infrastructures.

Future Work

Future work can be broken into a few different research areas:

1. **Enhancing Attack Detection**: While the developed neural network model shows promising results in classifying adversarial packets based on header information, further research can focus on improving the accuracy and robustness of the detection system. This can involve exploring additional features or variables in the packet headers that can provide better discrimination between normal and adversarial traffic. Additionally, the model can be fine-tuned and optimized to handle specific network conditions or attack patterns more effectively.

2. **Handling Congested Links and Delays**: The ANOVA-based model exhibited limitations in scenarios with significant delays or congested links, potentially leading to false positives or exacerbating network congestion. Future work can address these challenges by developing more adaptive models that can dynamically adjust their classification thresholds based on network conditions. Advanced techniques, such as reinforcement learning or adaptive algorithms, can be explored to enhance the model's ability to handle varying network environments.

3. **Real-time Detection and Response**: The experiment primarily focused on the offline analysis of packet headers. To make the detection system more practical, future work can explore real-time detection and response mechanisms. This can involve integrating the detection model into network devices or controllers to continuously monitor network traffic and identify crossfire attacks in real-time. Efficient mechanisms for immediate response and mitigation can also be developed to minimize the impact of such attacks on network performance.

4. **Network Robustness Optimization**: The Gurobi optimization library was mentioned as a potential approach to enhance network robustness against crossfire attacks. Further work can explore different optimization techniques and strategies to identify optimal network configurations that maximize throughput and minimize vulnerabilities. This can involve considering various factors, such as link capacity, routing protocols, network topology, and traffic patterns, to design resilient networks that can withstand crossfire attacks and other forms of adversarial activities. Additionally, additional work would be needed to fix the problems with our current implementation on larger networks.

5. **Evaluation on Real-world Networks**: The experiment was conducted on a simulated network environment. To validate the effectiveness and applicability of the proposed methods, future work should involve conducting experiments on real-world networks. By analyzing

network traffic and conducting experiments in diverse environments, the detection and defense mechanisms can be evaluated for their performance, scalability, and adaptability. Real-world evaluations can also provide insights into the practical challenges and limitations of the proposed approaches.

6. **Collaborative Defense Strategies**: Crossfire attacks can have severe consequences on network performance and stability. Future research can explore collaborative defense strategies, where multiple network entities, such as switches, routers, and SDN controllers, work together to detect and mitigate crossfire attacks. Collaborative approaches can leverage the collective intelligence and resources of the network to identify attack patterns, share information, and coordinate response actions more effectively.

7. **Adaptive and Self-learning Systems**: As attackers constantly evolve their techniques, future work can focus on developing adaptive and self-learning systems that can continuously analyze network traffic, detect new attack patterns, and dynamically update defense mechanisms. Machine learning and AI algorithms can be employed to build intelligent systems that can autonomously adapt to emerging threats and improve their detection and response capabilities over time.

8. **Privacy and Ethical Considerations**: With the increasing reliance on network monitoring and analysis for attack detection, future research should also address privacy and ethical considerations. Developing techniques that can effectively detect and mitigate attacks while protecting user privacy and ensuring compliance with legal and ethical norms is crucial. Striking the right balance between network security and privacy preservation is an important area for further exploration.

By addressing these future research directions, the field can advance the detection and defense mechanisms against crossfire attacks, improving network security and resilience in the face of evolving threats.

Appendix A

Networking Scripts

These scripts were used to run the MiniNet simulations for the Gurobi-optimized routing decisions.

A.1 Auto Network Script

```
#!/usr/bin/env python

import multiprocessing
import os
import shutil
import time
import logging
from runNetwork import myNetwork
def main():
    logging.basicConfig(level=logging.DEBUG)
    for i in range(0, len(os.listdir("TR"))):
        if os.path.exists("TR.txt"):
            os.remove("TR.txt")
        shutil.copy(f"TR/TR_{i}.txt", "TR.txt")
        if os.path.exists("/home/mininet/ryu/app/FL.txt"):
            os.remove("/home/mininet/ryu/app/FL.txt")
        shutil.copy(f"FL/FL_{i}.txt", "/home/mininet/ryu/app/FL.txt")
        process1 = multiprocessing.Process(target=runRyu)
        process2 = multiprocessing.Process(target=runNetwork, args=(i,))
        print(f"Starting Iteration {i}")
        process1.start()
        time.sleep(2)
        process2.start()
        process2.join()
        process1.terminate()
        process1.join()
        print("PROCESS KILLED")
        os.system("mn -c")
        print(f"Ending Iteration {i}")
        i += 1
def runRyu():
    os.system("cd ~/ryu/app && ryu-manager multi.py --observe-links")

def runNetwork(i):
    if not os.path.exists(f"DELAY/DELAY_{i}"):
        os.makedirs(f"DELAY/DELAY_{i}")
    myNetwork(i)

if __name__ == "__main__":
    main()
```

A.2 Mininet Script

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call
import time
import csv
import numpy as np
TRAF_File = "TR.txt"
REFERENCE_BW = 128
PORT = 1025
def myNetwork(iterationnum):
    T_VALUE = 10
    TRAF_File = "TR.txt"
    REFERENCE_BW = 128
    PORT = 1025
    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/12',
                   autoSetMacs = True
                 )

    info( '*** Adding controller\n')

    c0 = net.addController(name='c0',
                        controller=RemoteController,
                        ip='127.0.0.1',
                        protocol='tcp',
                        port=6633)

    info( '*** Add switches\n')

    S1 = net.addSwitch('S1', cls=OVSKernelSwitch, dpid='0000000000000001')
    S2 = net.addSwitch('S2', cls=OVSKernelSwitch, dpid='0000000000000002')
    S3 = net.addSwitch('S3', cls=OVSKernelSwitch, dpid='0000000000000003')
    S4 = net.addSwitch('S4', cls=OVSKernelSwitch, dpid='0000000000000004')
    S5 = net.addSwitch('S5', cls=OVSKernelSwitch, dpid='0000000000000005')
    info( '*** Add hosts\n')

    H1 = net.addHost('H1', cls=Host, ip='10.0.0.1', defaultRoute=None)
    H2 = net.addHost('H2', cls=Host, ip='10.0.0.2', defaultRoute=None)
    H3 = net.addHost('H3', cls=Host, ip='10.0.0.3', defaultRoute=None)
    H4 = net.addHost('H4', cls=Host, ip='10.0.0.4', defaultRoute=None)
    H5 = net.addHost('H5', cls=Host, ip='10.0.0.5', defaultRoute=None)
    info( '*** Add links\n')

    local_link = {'bw':1000.0, 'delay':'0ms'}
    net.addLink(S1, H1, cls=TCLink, **local_link)
    net.addLink(S2, H2, cls=TCLink, **local_link)
    net.addLink(S3, H3, cls=TCLink, **local_link)
    net.addLink(S4, H4, cls=TCLink, **local_link)
    net.addLink(S5, H5, cls=TCLink, **local_link)

    CityLink2 = {'bw':128.0, 'delay':'5.8229ms'}
    net.addLink(S1, S2, cls=TCLink, **CityLink2)
    net.addLink(S1, S4, cls=TCLink, **CityLink2)
    net.addLink(S1, S5, cls=TCLink, **CityLink2)
    net.addLink(S2, S3, cls=TCLink, **CityLink2)
    net.addLink(S2, S4, cls=TCLink, **CityLink2)
    net.addLink(S2, S5, cls=TCLink, **CityLink2)
    net.addLink(S3, S5, cls=TCLink, **CityLink2)
    net.addLink(S3, S4, cls=TCLink, **CityLink2)
    info( '\n*** Starting network\n')

    net.build()
    net.start()
    info( '\n*** Post configure switches and hosts\n')

    S1.cmd('ifconfig S1 10.0.8.1')
    S2.cmd('ifconfig S2 10.0.8.2')
    S3.cmd('ifconfig S3 10.0.8.3')
    S4.cmd('ifconfig S4 10.0.8.4')
    S5.cmd('ifconfig S4 10.0.8.5')
    readCSV(net, iterationnum)
    net.stop()
def readCSV(net, iterationnum):
    PORT = 1025
    info("STARTING TRAFFIC FLOWS\n")
    arr = np.loadtxt(TRAF_File).tolist()
    for i in range(len(arr)):
        for j in range(len(arr[i])):
            if arr[i][j] == 0:
                continue
            bw = REFERENCE_BW * arr[i][j] * 4
            net.get(f"H{j+ 1}").cmd(f"iperf -S -u -p {PORT} -D")
            net.get(f"H{i + 1}").cmd(f"iperf -c 10.0.0.{j + 1} -u -p {PORT} -t 60 -r -d  -b {REFERENCE_BW * 4 * arr[i][j]}M | grep -Po '[0-9.]*(?= Mbits/sec)'> DELAY/DELAY_{iterationnum}/H{j+ 1}_H{i+ 1}.txt &")
            PORT += 1
    time.sleep(65)
if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

References

[1] S. Balaji, Karan Nathani, and R. Santhakumar. IoT Technology, Applications and Challenges: A Contemporary Survey. *Wireless Personal Communications*, 108(1):363–388, Sep 2019.

[2] Mahesh Kavre, Aditya Gadekar, and Yash Gadhade. Internet of Things (IoT): A Survey. In *2019 IEEE Pune Section International Conference (PuneCon)*, pages 1–6, 2019.

[3] Nisha Panwar, Shantanu Sharma, and Awadhesh Kumar Singh. A Survey on 5G: The Next Generation of Mobile Communication. *Physical Communication*, 18:64–84, 2016.

[4] A. Gupta and R. K. Jha. A Survey of 5G Network: Architecture and Emerging Technologies. *IEEE Access*, 3:1206–1232, 2015.

[5] Cheng-Xiang Wang, Fourat Haider, Xiqi Gao, Xiao-Hu You, Yang Yang, Dongfeng Yuan, Hadi M. Aggoune, Harald Haas, Simon Fletcher, and Erol Hepsaydir. Cellular architecture and key technologies for 5G wireless communication networks. *IEEE Communications Magazine*, 52(2):122–130, 2014.

[6] Aliya Shaikh and Maninder Jeet Kaur. Comprehensive Survey of Massive MIMO for 5G Communications. In *2019 Advances in Science and Engineering Technology International Conferences (ASET)*, pages 1–5, 2019.

[7] Neil Briscoe. Understanding the OSI 7-layer Model. *PC Network Advisor*, 120(2):13–15, 2000.

[8] Mark Patton, Eric Gross, Ryan Chinn, Samantha Forbis, Leon Walker, and Hsinchun Chen. Uninvited Connections: A Study of Vulnerable Devices on the Internet of Things (IoT). In *2014 IEEE Joint Intelligence and Security Informatics Conference*, pages 232–235. IEEE, 2014.

[9] S. Sullivan, Alessandro Brighente, S. A. P. Kumar, and M. Conti. 5G Security Challenges and Solutions: A Review by OSI Layers. *IEEE Access*, 9:116294–116314, 2021.

[10] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. Mobile Edge Computing: A Survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2017.

[11] Milan Patel, Yunchao Hu, Patrice Hédé, Jerome Joubert, Chris Thornton, Brian Naughton, Julian Roldan Ramos, Caroline Chan, Valerie Young, Soo Jin Tan, and et al. Mobile-Edge Computing – Introductory Technical White Paper, Sep 2019.

[12] Preet Sanghavi, Sheel Sanghvi, and Ramchandra S Mangrulkar. Software-Defined Networks A Brief Overview and Survey of Services. *Software Defined Networking for Ad Hoc Networks*, pages 1–31, 2022.

[13] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14–76, 2014.

[14] Jacob H. Cox, Joaquin Chung, Sean Donovan, Jared Ivey, Russell J. Clark, George Riley, and Henry L. Owen. Advancing Software-Defined Networks: A Survey. *IEEE Access*, 5:25487–25526, 2017.

[15] Slavica Tomovic, Milica Pejanovic-Djurisic, and Igor Radusinovic. SDN Based Mobile Networks: Concepts and Benefits. *WIRELESS PERSONAL COMMUNICATIONS*, 78(3):1629 – 1644, 2014.

[16] Bo Yi, Xingwei Wang, Keqin Li, Sajal k. Das, and Min Huang. A Comprehensive Survey of Network Function Virtualization. *Computer Networks*, 133:212 – 262, 2018.

[17] Alcardo Alex Barakabitze, Arslan Ahmad, Rashid Mijumbi, and Andrew Hines. 5G Network Slicing Using SDN and NFV: A Survey of Taxonomy, Architectures and Future Challenges. *COMPUTER NETWORKS*, 167:106984, 2020.

[18] Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy. Network Innovation Using Openflow: A Survey. *IEEE communications surveys & tutorials*, 16(1):493–512, 2013.

[19] Mohammed A. Alqarni. Benefits of SDN for Big Data Applications. In *2017 14th International Conference on Smart Cities: Improving Quality of Life Using ICT IoT (HONET-ICT)*, pages 74–77, 2017.

[20] Juniper Networks. *Readiness, Benefits, and Barriers: an SDN Progress Report*. 2014.

[21] Dmitrij Melkov and Sarunas Paulikas. Security Benefits and Drawbacks of Software-Defined Networking. In *2021 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream)*, pages 1–4. IEEE, 2021.

[22] Bushra Alhijawi, Sufyan Almajali, Hany Elgala, Haythem Bany Salameh, and Moussa Ayyash. A Survey on DoS/DDoS Mitigation Techniques in SDNs: Classification, Comparison, Solutions, Testing Tools and Datasets. *Computers and Electrical Engineering*, 99, 2022.

[23] Abdulaziz Aldaej. Information Security and Distributed Denial of Service Attacks: A Survey. *2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA), Electrical and Computing Technologies and Applications (ICECTA), 2017 International Conference on*, pages 1 – 6, 2017.

[24] Jasmeen Kaur Chahal, Abhinav Bhandari, and Sunny Behal. Distributed Denial of Service Attacks: A Threat or Challenge. *New Review of Information Networking*, 24(1):31 – 103, 2019.

[25] Raihan ur Rasool, Hua Wang, Usman Ashraf, Khandakar Ahmed, Zahid Anwar, and Wajid Rafique. A survey of Link Flooding Attacks in Software Defined Network Ecosystems. *Journal of Network and Computer Applications*, 172:102803, 2020.

[26] Min Suk Kang, Soo Bum Lee, and Virgil D. Gligor. The Crossfire Attack. In *2013 IEEE Symposium on Security and Privacy*, pages 127–141, 2013.

[27] Charan Gudla and Andrew H. Sung. Moving Target Defense Application and Analysis in Software-Defined Networking. In *2020 11th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 0641–0646, 2020.

[28] Panos Kampanakis, Harry Perros, and Tsegereda Beyene. SDN-based solutions for Moving Target Defense network protection. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pages 1–6, 2014.

[29] Abdullah Aydeger, Nico Saputro, Kemal Akkaya, and Mohammed Rahman. Mitigating crossfire attacks using sdn-based moving target defense. In *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, pages 627–630, 2016.

[30] Jared M Smith and Max Schuchard. Routing Around Congestion: Defeating DDoS Attacks and Adverse Network Conditions via Reactive BGP Routing. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 599–617, 2018.

[31] Muoi Tran, Min Suk Kang, Hsu-Chun Hsiao, Wei-Hsuan Chiang, Shu-Po Tung, and Yu-Su Wang. On the Feasibility of Rerouting-Based DDoS Defenses. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1169–1184, 2019.

[32] Akash Raj Narayanadoss, Tram Truong-Huu, Purnima Murali Mohan, and Mohan Gurusamy. Crossfire attack detection using deep learning in software defined its networks. In *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, pages 1–6, 2019.

[33] Pedro Garcia-Teodoro, Jesus Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-Based Network Intrusion Detection: Techniques, Systems and Challenges. *Computers & Security*, 28(1-2):18–28, 2009.

[34] Mohammed I. Salman and Bin Wang. Boosting performance for software defined networks from traffic engineering perspective. *Computer Communications*, 167:55 – 62, 2021.

[35] Mohammed I. Salman and Bin Wang. Near-Optimal Responsive Traffic Engineering in Software Defined Networks based on Deep Learning. *Future Generation Computer Systems*, 135:172–180, 2022.

[36] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. Semi-oblivious traffic engineering: The road not taken. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation*, NSDI'18, page 157–170, USA, 2018. USENIX Association.

[37] Bob Lantz, Brandon Heller, and Nick McKeown. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6, 2010.

[38] S. Knight, H.X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The Internet Topology Zoo. *Selected Areas in Communications, IEEE Journal on*, 29(9):1765 –1775, October 2011.