ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Dimitrios Gkounis
dgkounis@gmail.com

# Cross-domain DoS link-flooding attack detection and mitigation using SDN principles

Master Thesis MA-2013-18
October 14, 2013 to April 13, 2014

Advisors: Vasileios Kotronis, Dr. Xenofontas Dimitropoulos
Supervisor: Prof. Bernhard Plattner

**Abstract**

The Denial of Service (DoS) attacks pose a major threat to Internet users and services. Since the network security ecosystem is expanding over the years, new types of DoS attacks emerge. The DoS link-flooding attacks target to severely congest certain network links disrupting Internet accessibility to certain geographical areas and services passing through these links. Since crucial services like financial and government services depend on real-time Internet availability, the consequences of DoS link-flooding attacks become detrimental. Among the diverse DoS link-flooding attacks, the Crossfire attack is worthwhile to focus on when designing a DoS link-flooding attack countermeasure due to its effectiveness while it remains undetected. In this master thesis, we propose a detection and mitigation technique by combining Software Defined Networking (SDN) and network security principles. Since current defence solutions and techniques are unable to deal with the Crossfire attack, we use SDN features, such as flow rerouting, flow-level management and control and monitoring centralization, which provide by definition higher flexibility in defeating such complex DoS attacks. We design an online traffic engineering mechanism based on a strategy that enables both mitigation and detection of the Crossfire attack. A working prototype is implemented based on the proposed technique and evaluated on an emulated pure SDN environment.

**Acknowledgements**

I would like to thank Professor Bernhard Plattner who gave me the opportunity to conduct my Master Thesis at the Communications Systems Group of the ETH Zurich. I am very glad that I had the chance to explore the fascinating areas of Software Defined Networking and Network Security during the past six months. I am also grateful to Dr. Xenofontas Dimitropoulos and PhD candidate Vasileios Kotronis for their valuable help and feedback and for the fruitful discussions we had throughout this thesis. Without their support, the completion of this thesis would not have been possible.
Finally, I would like to express my gratitude to my family for their unconditional love and support and their encouragement to make my dreams come true all these years.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

A Denial of Service (DoS) attack is defined as an attempt by an attacker to prevent access to online information or services to legitimate users [1]. The DoS attack strategy is based on the asymmetry between the Internet hosts' and victim's resources. The attacker exploits the resources of thousands of compromised machines, called bots, which exceed the victim's limited resources. Therefore, the victim is not able to serve the requests made by both the bots and the legitimate users.
The DoS attacks constitute a significant threat to the Internet. The security researchers constantly develop defence countermeasures but the attackers in turn modify their attack mechanisms circumventing current security solutions. Thus, the DoS attack methods evolve in terms of complexity over the years [2].
New types of DoS attacks which are even more detrimental for network services have recently emerged. The DoS link-flooding attacks aim to flood specific network links disrupting all services passing through them. The DoS attacks against Spamhaus [3] constitute a recent example of these DoS attack mechanisms. Even though the attacks started targeting specific servers, they evolved to flood network links in several Internet Exchange Points (IXPs), indirectly affecting the services using these IXPs. The DoS link-flooding attack strategy has higher complexity in comparison with other DoS attacks since the network links leading to specific target services should be identified. However, the consequences of the DoS link-flooding attacks outweigh the complexity of their mechanism. Considering that critical activities, such as electric power distribution, financial transactions, government services and industrial remote access networks, depend on Internet connectivity, the DoS link-flooding attacks become of higher importance.
Recently, the advent of the Software Defined Networking (SDN) has become a breakthrough in the Computer Networking field. This architecture allows decoupling of the control and data plane, enables logically centralised network controllers to manage whole networks and promises to reduce complexity in today's network operations. Therefore, the SDN principles are a promising asset in the attempt to counter complex DoS attacks.

## 1.1 Motivation

As discussed, the DoS link-flooding attacks pose a major threat among the diverse DoS attack strategies. The Crossfire [4] and Coremelt [5] DoS link-flooding attacks cannot be easily blocked by current defence countermeasures. These attacks use valid bot IP addresses and therefore cannot be detected by IP spoofing filters [6]. Furthermore, legitimate traffic is used, either between pairs of bots (Coremelt attack) or between bots and public servers (Crossfire attack). In addition, current defence approaches cannot easily distinguish the attack traffic from the legitimate one since the Crossfire attack uses low-intensity attack flows. The targets of the Coremelt attack are backbone links within several network domains (Autonomous Systems - ASes). In the Crossfire attack, the targets are network links around a certain geographical region which belong to several ASes and Internet Service Providers (ISPs). This makes a single ISP unable to stop the attack.
The current online and offline traffic engineering mechanisms cannot counter these DoS link-flooding attacks since the offline ones react late, i.e., generate new routes in hours or even days, and the online ones can lead to routing instabilities [9] especially when ISP collaboration is needed to block an attack as in the case of the Crossfire attack. The feasibility of ISP coordination is

also an issue due to competitive and privacy concerns. Even if the online traffic engineering approaches could be applied, the Crossfire attack could bypass them by targeting different sets of network links at a time.
The Crossfire attack, compared to the Coremelt attack, poses more challenges to take into account when developing a security solution [4]. The Crossfire attack is robust against route shifts and it avoids triggering alarms for potential attacks by changing the set of target links every time interval given by the attacker. In addition, its efficiency, in contrast to Coremelt attack, does not depend on how the bots are geographically distributed. The Crossfire attack also requires ISP collaboration to be defeated. Thus, the Crossfire is a more interesting attack to focus on in order to devise a detection and mitigation solution.
The Crossfire attack potential countermeasures are ISP collaboration and reactive rerouting of flows traversing flooded network links [4]. We look at how SDN can be used in the context of such practices. The SDN principles and particularly the OpenFlow protocol which implements them provide high flexibility in managing a network [10, 11]. In SDN, the network traffic is identified, monitored, controlled and managed on a flow level. SDN enables real-time (reactive) flow management which can be modified based on the network response and on demand changes of the user's or the network application's requirements. Current IP networks do not offer this level of granularity in network control and management. Furthermore, the logically centralized network controller in an SDN environment has full knowledge of the global network state and control over it. Therefore, traffic engineering policies can be easily imposed all over the network. On the contrary, today's network devices have limited visibility usually only within their immediate network neighbourhood and should be individually configured. The SDN characteristic of centralised controllers which have full network knowledge and control also facilitates ISP collaboration. It is beneficial to exchange aggregate information in a hierarchical fashion only among network controllers rather than among all network devices. For all the above-mentioned reasons, we employ SDN to counter the Crossfire attack.

## 1.2 Related Work

### 1.2.1 DoS Attack Detection and Mitigation using SDN

At the current time, there is a small number of published papers having proposed methods on DoS attack detection and mitigation using SDN. Indicatively, in [12], the authors present a low-overhead technique for traffic analysis using Self Organizing Maps to classify flows. This mechanism is deployed on an SDN (NOX-controlled [13]) network to enable DoS attack detection. Our approach differs from this as our proposed solution not only performs DoS attack detection but also mitigation. The detection and mitigation mechanisms are coupled in our work. We have designed an online traffic engineering technique which primarily conducts mitigation. However, the mitigation approach uses a clever strategy that also enables detection. Moreover, we employ SDN principles in order to distinguish (and mitigate) the malicious traffic from the legitimate one taking into consideration that the attacker generates legitimate low-rate traffic flows and not easy-to-detect heavy-hitters. In their work, regular abnormal traffic is detected. In addition, they focus only on a local network domain. On the other hand, we conduct detection and mitigation within multiple network domains. In particular, our solution works both locally and in a client-ISP collaborative environment.

### 1.2.2 DoS Link-Flooding Attack Countermeasures

There is not any significant work on defence solutions against the DoS link-flooding attacks since the high complexity of their characteristics has not allowed them to commonly occur yet. However, the potential effects of such attacks are detrimental in terms of costs for Internet services and can surpass the cost of developing such attacks when disrupting services which highly depend on the Internet, such as industrial networks, financial and government services etc. The published Coremelt [5] and Crossfire [4] attacks highlighted the feasibility and the consequences of this type of DoS attacks. Therefore, a recent paper [14] is a first approach towards defeating DoS link-flooding attacks. In CoDef [14], the authors propose a cooperative method for identifying low-rate attack traffic. The traffic source and target ASes communicate with each other and low-rate malicious traffic ASes are identified by not complying with multiple rerouting requests

generated by the target ASes. Our work has also the same goal but the fundamental conceptual difference is that we do not explicitly communicate with the traffic sources. We apply our defence solution and we monitor how the attacker implicitly reacts. Another major difference is that we focus on the Crossfire attack while their approach cannot deal with it. Furthermore, we enable collaboration between the target ASes and their direct upstream providers since we first focus on solving the problem within a local domain (the target AS) and then we expand our security mechanism on an inter-domain level. We believe this is a more feasible scenario considering the current Internet state (multihoming practices of large ASes on the edge hosting numerous Internet services attractive for such attacks). In CoDef, the authors mention SDN as a possible mechanism for implementing their solution, while we explore in depth the power and capabilities of SDN taking its principles into account both in our design and implementation. However, our work has similarities with CoDef. We both use rerouting to mitigate and detect the DoS link-flooding attack (both locally and on an inter-domain level in our case - only on an inter-domain level in CoDef). The potentially malicious traffic is rate-limited in both works. We limit the rates of the potentially malicious sources while they provide higher bandwidth guarantees in legitimate traffic. Finally, we both employ inter-ISP coordination to tackle the attack, though in a different way (different reaction and coordination algorithms).

## 1.3 The Task

In this thesis, a prototype of a cooperative detection and mitigation service for link-flooding attacks will be developed, focusing on the Crossfire attack. An SDN environment with a controller per network domain is assumed and inter-controller communication will be deployed to facilitate ISP collaboration. Moreover, detection and mitigation techniques, such as flow rerouting, rate-limiting and ranking, between and within network domains will be implemented. Finally, the performance of the proposed attack countermeasure will be evaluated on a network emulation platform (Mininet [15]) in terms of feasibility, efficiency and accuracy.
The contributions of this thesis are:
**Detection and Mitigation Technique against the Crossfire Attack**: We present our security solution which combines online Traffic Engineering (TE) and network security techniques. The Crossfire attack is a powerful attack and multiple mechanisms are needed to defeat it. Our mechanism is an online TE strategy which is performed under the control of a centralized vantage point (the network controller) which allows us to deal with routing instabilities that current practices (such as implicit OSPF weight tuning) may cause. The detection and mitigation mechanisms are coupled and interdependent in our approach.
**Implementation of Working Prototype**: We implement a working prototype of the proposed technique and we also evaluate it on an emulated testbed (Mininet). We show the experimental results of its performance evaluation to highlight the feasibility, efficiency and accuracy of our approach. The prototype can also be deployed in a production testbed.
**Incentives**: We outline a number of incentives for ISP collaboration and for deploying the proposed security service within an enterprise network.

## 1.4 Overview

The rest of the thesis is organized as follows: In the first part of chapter 2, an overview of the Crossfire attack is described, while the second part gives an overview of the Software Defined Networking. Chapter 3 presents the proposed solution to counter this type of DoS link-flooding attack. At first, we introduce our attacker model based on the Crossfire attack and then we explain how our detection and mitigation approach is designed. Chapter 4 gives implementation details of our attacker and defender. Chapter 5 shows the evaluation results of our attack countermeasures and discusses the conclusions and insights drawn from them. In chapter 6, future tasks related to this work are proposed. Chapter 7 summarizes the thesis.

# Chapter 2

# Background

The first part of this chapter presents an overview of the Crossfire attack. More details about this DoS link-flooding attack can be found in [4]. An overview of the Software Defined Networking is described in the second part. Further information can be found in [10, 11].

## 2.1 The Crossfire Attack

The aim of the Crossfire attack is to block legitimate users from accessing a certain geographical area of the Internet, the *Target Area*. This area usually contains hosts offering important services to the rest of the Internet and is surrounded by multiple public servers, the *Decoy Servers*. The concept of the attack is based on the fact that specific network links, the *Target Links*, lead to both the *Decoy Servers* and the *Target Area*. Therefore, an attacker can employ bots to flood the *Target Links* by sending traffic only to the *Decoy Servers*. As a consequence, when the *Target Links* are flooded, the *Target Area* becomes unreachable from the rest of the Internet. The intended victim is not aware of the attack since there is not any attack traffic destined to the *Target Area*.
To launch the Crossfire attack, three main steps are needed: the link map construction, the attack setup and the bot coordination. These steps are briefly described in the following subsections.

### 2.1.1 Link Map Construction

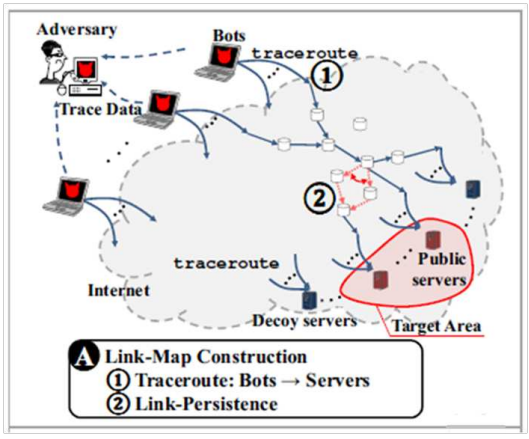



Figure 2.1: Link Map Construction [4]

The first step of the Crossfire attack is the link map construction (Figure 2.1). The attacker builds a map of the network links along the paths from her bots to both the decoy servers and the public servers in the target area using *traceroutes* and processing their results. Each bot executes multiple *traceroutes* to a destination to determine whether the same links are traversed each time or not. Some ISPs often load-balance the traffic passing through their network, resulting

in different routes between the same pairs of nodes. If this is the case in the data gathered by the *traceroutes*, the corresponding network links are not considered as candidate target links to be flooded by the attacker (due to the implicit link "protection" through load-balancing).

### 2.1.2 Attack Setup

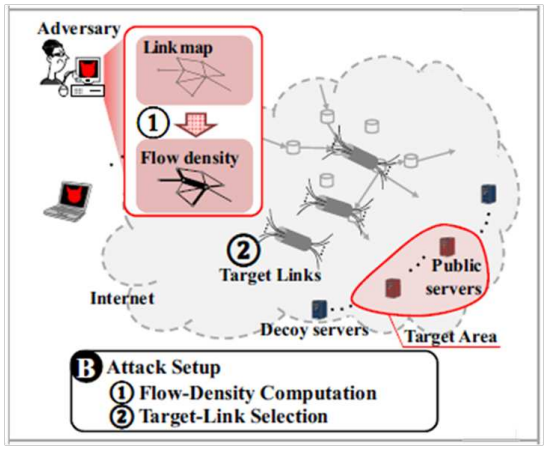



Figure 2.2: Attack Setup [4]

After the link map has been constructed, the attacker uses the links of the stable routes of the link map to determine the target links (Figure 2.2). The candidate target links are categorized based on the largest number of routes and flows passing through them, the *flow density*, and leading to the target area. If a certain link is used by a large number of flows, then its flooding can effectively disrupt the access to the target area. The attacker selects multiple disjoint groups of potential target links and floods one each time. The feature of dynamically changing the set of links to flood enhances the undetectability of the attack. The final target is to concurrently flood all the links of a certain set of target links each time in order to fully disrupt target area access for legitimate traffic.

### 2.1.3 Bot Coordination

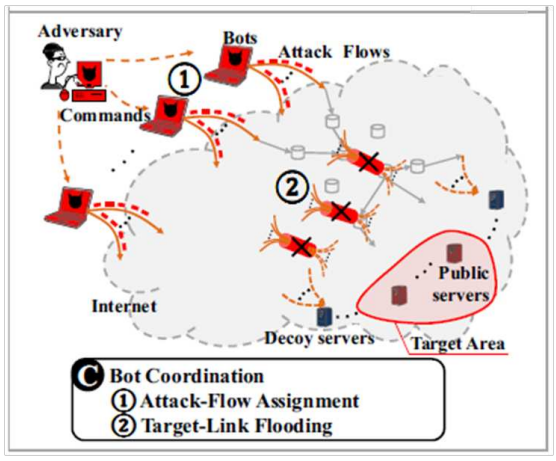



Figure 2.3: Bot Coordination [4]

In the last part of the attack, the attacker assigns to her bots both the decoy servers to send traffic to and their corresponding flow rates and then the bots start the flooding (Figure 2.3). The assignment is made in a way that each flow from a bot to a decoy server has low bandwidth demand while the selected target links are flooded by the aggregate flow rate which exceeds the

target link bandwidth. The flows generated by the bots have low-rate so that no current security solution can classify the traffic as malicious. Furthermore, the traffic needed to flood the selected target links is evenly distributed to multiple bots and decoy servers. Thus, the servers in the target area are unable to detect the attack as no attack traffic is destined to the target area. The decoy servers are also unable to identify the attack since the attack traffic is distributed in a large number of decoy servers and there is not a high enough bandwidth increase in each server to trigger an alarm. After the assignment has finished, the bots start generating the attack traffic. The adversary can repeatedly execute the bot coordination part of the attack by changing the set of the target links in order to prolong its duration and "mislead" the defender.

## 2.2 Software Defined Networking

Software Defined Networking is a novel network architecture which allows separation of the network control functions, the *control plane*, such as tracking the topology, computing routes and installing forwarding tables, from the forwarding functions, the *data plane*, such as forwarding, buffering, filtering, marking packets. Therefore, the network control can directly be programmatically configured having abstract knowledge about the underlying infrastructure. Furthermore, the *control plane* is logically centralised maintaining a global network view. The SDN deployment simplifies the network functions, operation and maintenance and enables the development of open standards and thus rapid innovation. In addition, the use of SDN reduces the need of intelligence in the network devices since the control decisions are made by the centralised network controller.

### 2.2.1 OpenFlow

OpenFlow is a communication interface and mechanism between the control plane and the data plane in an SDN environment. A typical SDN environment consists of multiple *switches* and network *controllers* (Figure 2.4). Each switch contains a *flow table* and a *secure channel* interface. The flow table processes and forwards traffic flows using sets of flow matching rules, the flow entries. The flow rules contain header fields to identify certain traffic packets and actions to perform on the corresponding packets. Each entry in the flow table also contains a list of counters which hold statistical information about the matching packets. The network control plane decides how to treat packets which have no matching flow rules in the flow table of a switch. A controller installs and removes flow entries in a flow table through the secure channel of the switch using OpenFlow-based messages. OpenFlow is currently the only standardized protocol which enables Software Defined Networking and can be integrated on current physical and virtual networks.

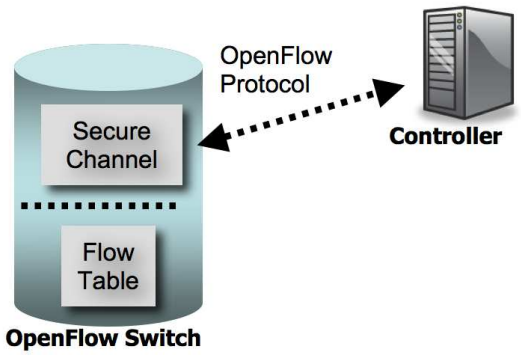



Figure 2.4: OpenFlow-based Switch-Controller Communication [16]

# Chapter 3

# Attacker and Defender Model

Before discussing how the Crossfire attack is modelled and how our detection and mitigation approach is designed, we briefly describe the environment we are working in. A multihomed enterprise network is considered to be protected by our solution against the Crossfire attack. This countermeasure is offered by collaborating Internet Service Providers (ISPs) as a service to the customer, the enterprise network. It is safe to consider that an enterprise network is multihomed as in general the enterprise networks use more than one ISP to access to the Internet in order to achieve higher throughput, fault-tolerance etc. We consider that we work on a SDN environment, where the network devices of the enterprise and the providers' networks are managed by their corresponding controllers. This SDN principle simplifies ISP collaboration. Dealing with the attack locally precedes ISP collaboration, and it helps scale up our approach since each domain does its best to handle the attack first on a local level and then on a higher scale by involving more ISPs. This also helps to mitigate the propagation of routing instabilities to the outside world, when the outside world does not need to know about local TE changes. More details on the exact approach will be explained later in this chapter.
Multiple ISPs should collaborate to counter this attack considering that the attack traffic will derive from multiple network domains. The incentives for assuming that the involved ISPs collaborate are the benefits from providing the security service in terms of revenue, the reputation they earn in the network security area and the fact that the ISPs, based on our design, do not need to disclose valuable information, such as their topologies, routing policies etc., which would violate their privacy and put their business at risk. An ISP "sees" the enterprise network and the other ISPs as a cloud and has only knowledge about its peering links to the other ISPs and the enterprise network.

## 3.1 Attacker Model: Launching a Reactive Crossfire Attack

In this section, we describe how the attacker in the Crossfire Attack is modelled in the context of this thesis.
In general, the attacker behaves as described in section 2.1. However, in order to evaluate our proposed solution to the attack, we have to make some assumptions on how the attacker reacts to specific cases, e.g., rerouting, which were not fully addressed in [4].
The attacker starts with setting up and launching the attack following the same steps as in the Crossfire attack. Thus, the attacker constructs a link-map for a certain target area, sets up the attack finding the target links based on their flow density and then assigns to her bots the decoys servers to send traffic to. The attacker continues with constantly monitoring the previously constructed link-map and the corresponding flow-density until any changes, i.e., topology changes due to reroutings, are detected. The attacker can then determine if any rerouting along the paths between her bots and the decoy or target servers has happened. If this is the case, the attacker sets up again and relaunches the attack. The attacker has a fixed attack budget and should allocate her resources as efficiently as possible. The attack budget is defined as the total number of bots the adversary employs to launch the attack and in our case is considered to be fixed. The attacker model is summarized in Figure 3.1.

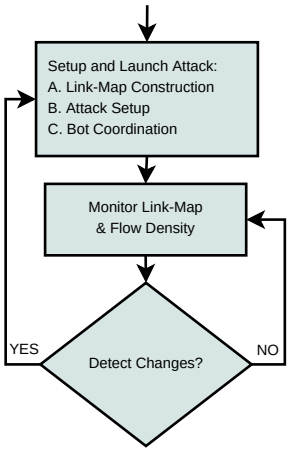



Figure 3.1: Attacker Model

There are many strategies of decoy server assignment to bots. A strategy is selected considering that the attack traffic should be evenly distributed between the bots and the decoy servers. Thus, the bots should not generate too much traffic as they may be considered suspicious by the defender and each decoy server should not receive a large amount of traffic so as an attack alarm is not triggered. In this way, the defence approach becomes difficult, especially in terms of reliable detection of the bots. We further discuss about assignment strategies as well as our custom one in section 4.1.3.

## 3.2 Defender Model: Detection and Mitigation Approach

In order to detect and mitigate the Crossfire attack, we divide the problem in two parts: the local approach that is enabled when the attack occurs in the local domain and the inter-domain approach which is executed when the local approach is unable to handle the problem on its own or when the attack happens on an inter-domain level. The inter-domain approach enhances the detection and mitigation capability of the local one.
We consider that a Crossfire attack may arise either within the local, enterprise, network or on the peering links between the local domain and its ISPs. In the scope of this thesis, we assume that a Crossfire attack cannot happen within an ISP network due to the abundance of its resources. We also consider that attack traffic does not originate from the local, enterprise network.

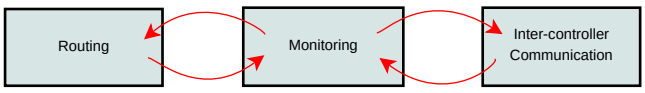



Figure 3.2: Defender Components Interaction

Taking into consideration the above mentioned parts of our approach, we further define the components of our defender model. This categorization is necessary to help the reader understand the principles of our approach. The Monitoring component conducts measurements to identify severe congestions derived by a potential Crossfire attack and classify the congested traffic within the network domain. The Routing component deals with the congested traffic during the Crossfire attack and the Inter-domain (Inter-controller) component defines the interface among the local domain and the collaborating ISPs. The components interact with each other in a specific way which will be described later via examples. A simple model of the components interaction is illustrated in Figure 3.2.
In Figure 3.3, our defence approach in a local domain is presented. Every link in the local network is constantly monitored. In case of a severe link congestion, we check if another one has happened in the past (i.e., if our security algorithm has at least been executed once). This is because we would like to correlate a current congestion with an old one. If this mechanism is run for the first

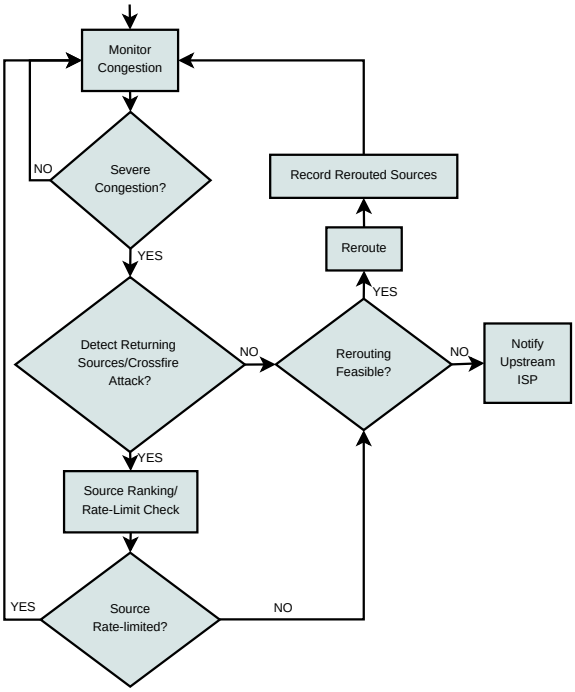
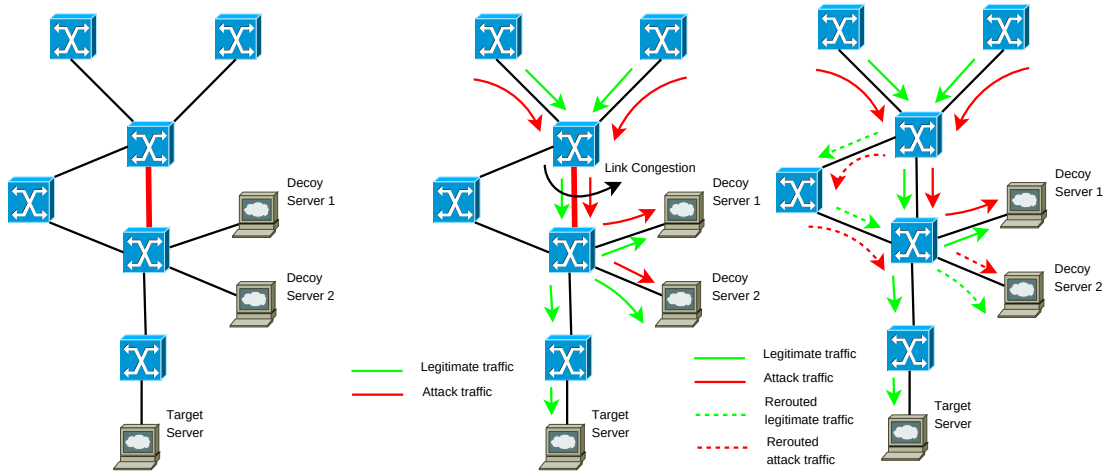



Figure 3.3: Defender Model

time, then we check if a rerouting is possible to bypass the congested link. If the local topology allows an alternative path and this path has the required resources, then a selection of the congested flows is rerouted and the corresponding sources are recorded. The flows are rerouted based on a specific strategy, the implementation of which is described in section 4.2.2. We employ destination-based rerouting for the congested flows since the number of the destinations of the congested traffic is far less than the number of the congested flows. We have to remember that in the Crossfire attack a link is flooded by a large number of low-rate traffic flows. Our rerouting strategy leads part of the congested traffic in routes disjoint to the previously calculated by the attacker target links and potentially disjoint to the new ones (the calculated ones after the attacker has detected and reacted to the route shifts). The design goal of the rerouting is to ”force” a potential attacker to persist in flooding target links using the same sources but in traffic destined to different destinations (decoy-servers). The congestion is mitigated and malicious traffic can be detected if the same sources persist in further congestion events. The current and the past flooding events are correlated, and if the above-mentioned desired reaction of the potential attacker is identified, then the corresponding sources are marked as suspicious, incrementing a corresponding counter. This traffic engineering mechanism enforces mitigation of the congested traffic and may lead to the detection of the malicious traffic. We always attempt to reroute the congested flows until their sources which are responsible for multiple congestions are rate-limited when their suspiciousness degree exceeds a threshold. The higher the score of a source, the more confident we are to detect the malicious traffic with higher probability. The concept of scores is necessary in order to avoid considering legitimate sources as malicious as much as possible. The upstream ISPs may help in the local detection and mitigation approach when there is not any rerouting capability in the local domain. The corresponding ISPs reroute the congested traffic through either another peering link or through a collaborating ISP if another peering link does not exist or does not have the required resources. In this case, the local algorithm works as before.

Our defender model in the case of a Crossfire attack which aims at the peering links of the local network with its direct ISPs is quite similar. The ISPs monitor the peering links with the local network and reroute the traffic in the event of a congestion. The rerouting is feasible since its bandwidth requirements are met by another peering link of the same or a collaborating ISP. The ISPs may identify and rate-limit the malicious traffic in the peering links as in the case of our local security approach. This is due to the fact that the malicious sources will persist in congesting the inter-domain links. This detection and mitigation approach requires communication and

collaboration among the direct providers of the enterprise network.

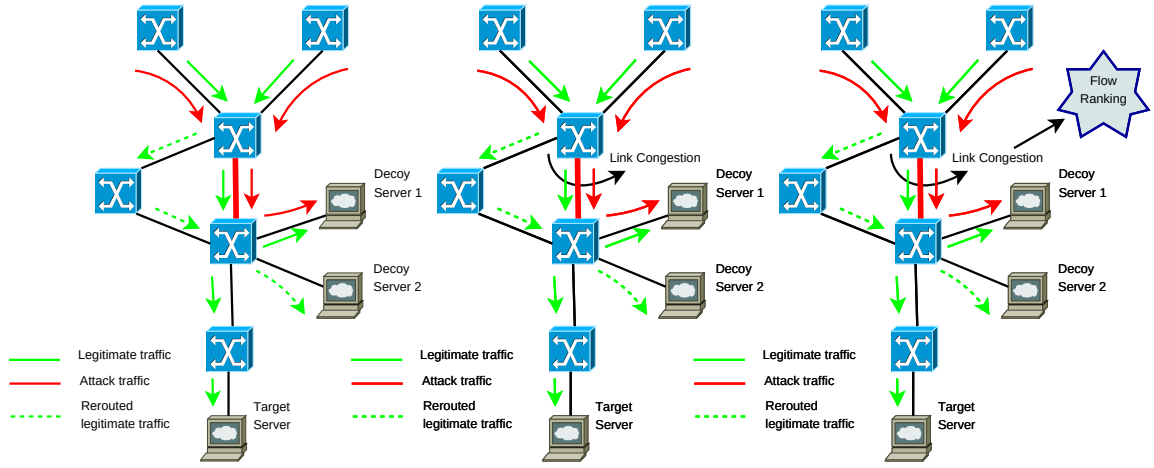



(a) Sample Topology (b) Target Link Congestion (c) Congestion Rerouting

Figure 3.4: Local Rerouting Example

To further explain our defender model in more detail, we will use an example (Figures 3.4, 3.5 and 3.6). Let us consider a simple topology with a target server, two entry switches and two decoy servers (Figure 3.4a). Taking into account the *Attack Setup* step of the Crossfire attack, described in section 2.1.2, the target link of this topology and of this designated target area is the one marked in red colour. The attacker therefore assigns to her bots to send traffic to decoy servers 1 and 2 and launches the attack to flood the target link (section 2.1.3). There is also legitimate traffic with destination to both the decoy and target servers (Figure 3.4b).
The target link is severely congested and for example the decoy server 2 is the destination of the congested flows which is selected for rerouting based on our classification mechanism. This mechanism attempts to implicitly identify the decoy servers. It is based on the assumption that the attacker will attempt to evenly distribute her flows among the decoy servers. Thus, the flows towards the decoy servers will occupy homogeneous bandwidth on the flooded link. We assume that the bandwidth of flows leading to target servers does not follow the same distribution. The congested flows are aggregated by their destination and the aggregated flows are classified according to homogeneous occupied bandwidth in the target link. Then, the destinations with high homogeneity level and contributing to a large part of the congestion bandwidth are chosen to be rerouted, so the path towards the target area may not be affected, i.e., rerouted. This ensures that malicious traffic detection is enabled in subsequent link congestions. If our mechanism also changes the routes to the target servers, the malicious traffic may be detected but after more executions of our detection algorithm than in the previous case. In any case, any congestion is successfully mitigated. Consequently, the traffic, legitimate and malicious one, to decoy server 2 is now routed towards a path that is disjoint with the flooded link (Figure 3.4c). We have to note that in the case of a more complex topology and in the presence of severe congested links within the same time interval, the reroutings should take into account the concurrent flooded network links. This is necessary since the new routing paths should not contain the already congested links. Our approach will also find routing paths that are able to accommodate the congested flows in terms of bandwidth as much as possible. The flows of a congested link could be rerouted in multiple network paths. We should note that this is not equivalent to multipath routing per flow but per group of rerouted flows. Each individual flow still follows a single path from the available ones, albeit different than its initial one due to the occurrence of the rerouting event. The sources that were rerouted are recorded along with the rerouted paths.
The rerouting will change the link-map of the attacker. When this change is identified, the attacker will set up again and relaunch the attack (Figure 3.5a) based on the attacker model (Figure 3.1). The calculated target link will be the same as before in this example. The attacker now assigns to her bots to send traffic only to decoy server 1 since the decoy server 2 is reached through a path which is disjoint with the target link. The attacker will stop sending traffic to decoy server 2 considering that the new path along this server does not contain the target link

and that she has a fixed attack budget, as mentioned in section 3.1. Our design is based on the fact that the time of the attacker's readjustment is higher than that of the defender's reaction. The time it takes the defender to react is faster than the time that the attacker identifies and responds to the defender's reaction. Therefore, our approach alleviates the problem.

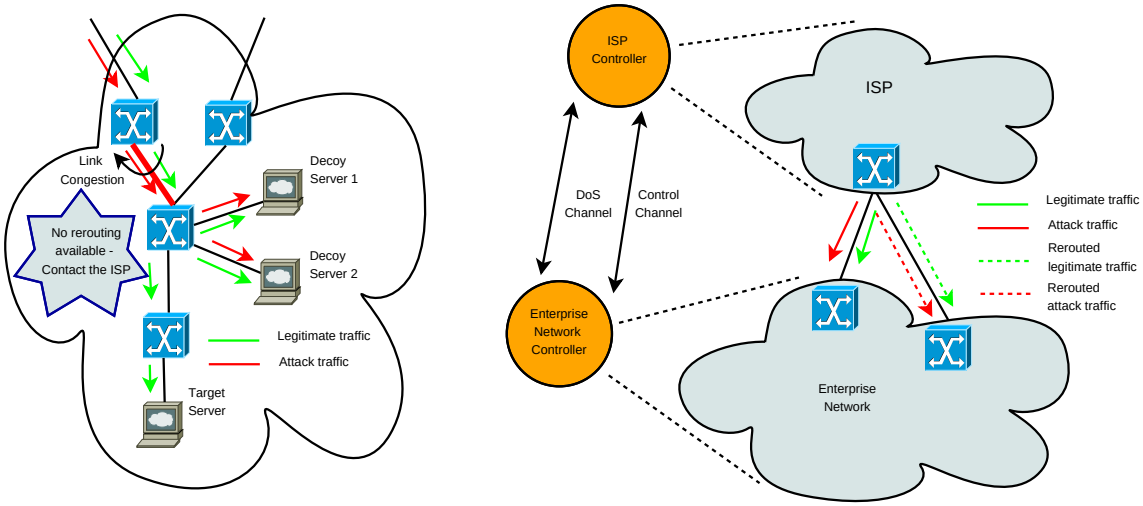



(a) Attack Resetup (b) Severe Recongestion (c) Suspicious Source Marking

Figure 3.5: Local Rerouting Example - cont.

This rerouting event may increase the cost of the attack as some bots may be required to send more flows to the same decoy servers, decoy server 1 in this example, to flood the target link in case no more decoy servers come into play as in this example. Our approach attempts to increase the probability of detecting the attack at the decoy servers by implicitly forcing the attacker to assign more flows to certain decoy servers (more bandwidth is received by the decoy servers).
The target link is flooded and the sources of the flows that were previously rerouted are present in the congested link (Figure 3.5b). Since the rerouted sources stopped sending traffic towards the rerouting paths and are now present in the flooded link, we can assume that these are suspicious sources (Figure 3.5c). Thus, these sources are marked in our attempt to detect the malicious traffic. As long as the attack is persistent and our topology allows multiple reroutings, the malicious traffic can be distinguished with higher certainty. When the suspiciousness rank of some sources surpasses a threshold that allows us to determine that they are malicious with high probability, then they can be rate-limited. To consider that a source is suspicious, we should take into account that subsequent link congestions should not take place along the just previously rerouted paths. In this way, we ensure that traffic sources in a flash crowd will not be considered malicious. In case a flash crowd causes a severe congestion, the corresponding sources will not stop sending traffic towards a rerouted destination. In case of a flash crowd during the Crossfire attack, then multiple rounds of our algorithm should be executed to be able to distinguish the malicious traffic. Up to now, we consider that the granularity of the traffic flows is on the level of the IP addresses (source IP address, destination IP address tuples). However, in large scale enterprise networks where thousands of flows are present, it is easier to manage flows on an IP prefix level, essentially employing aggregation in order to deal with the issue of high volume state management. Our approach in this case does not change but our algorithm should run more rounds than when managing traffic flows as IP address tuples in order to determine that a prefix is malicious.
The congested traffic is rerouted as before as long as the topology allows it. In case there are not any further alternative routes for the congested flows or the topology does not allow any secondary paths, as in Figure 3.6a, then the corresponding upstream ISP should be notified to extend our detection and mitigation attempts. The controller of the local network communicates with the one of the provider network to reroute the congested flows through another peering link of this provider or of a collaborating one and notifies the local network about the rerouting state of the reported flows (Figure 3.6b). The local algorithm continues its execution as before. The advantage of our approach is that both the local network and the collaborating providers are allowed to not disclose information about their networks, such as their topologies, routing

policies etc.




(a) Infeasible Local Rerouting (b) Inter-Domain Rerouting

Figure 3.6: Inter-Domain Rerouting Example

# Chapter 4

# Attacker and Defender Implementation

As discussed in Chapter 3, we design our security approach assuming a pure SDN environment. To implement our approach on this environment, we use the OpenFlow protocol which is currently the only one that enables Software Defined Networking in a clean and standardised fashion. As mentioned in section 2.2.1, the network control functionalities in a typical OpenFlow-based SDN environment are performed by logically centralised network controllers. There is currently an increasing number of frameworks that enable the development of OpenFlow controller applications. The choice of the controller platform depends on the needs of the controller applications, the programming language we would like to use, etc [17] . POX is a Python-based OpenFlow controller framework [18, 19]. It currently supports OpenFlow version 1.0, is widely used and supported and its learning curve is easy. However, POX has slow performance. It allows easy and fast development of OpenFlow controller applications when there are not any controller performance requirements and thus is preferred in research, experimentation and demonstrations [19, 17, 18]. For the above-mentioned reasons, POX is chosen as the OpenFlow controller platform to implement our security approach. We have to note that we work on the *carp* branch of POX since it was the most recent release when this thesis started.

## 4.1 Attacker: Launching a Reactive Crossfire Attack

In section 3.1, we described how the Crossfire attacker is modelled. In this section, we will explain how we implement it based on this model.

### 4.1.1 Link-Map Construction and Monitoring

The attacker selects a certain geographical area as a target and builds a map of its network links (i.e., a *link-map*). The attacker runs multiple traceroutes from her bots to public servers inside and around the target area, gathers all the results and processes them to construct the link-map. In addition, the attacker executes traceroutes periodically to verify whether there are any modifications to her constructed link-map or not. In case of a map change, the adversary uses the updated data to set-up and launch her attack over again.

### 4.1.2 *Flow Density* Computation - *Target Link* Selection

The attacker processes the link-map she has built to select the target links to flood (Figure 4.1). The attacker computes how many times a network link is traversed, i.e., the *flow density* of a link, along the multiple routes from her bots to the target area servers. Then, she selects the network link with the highest density as a target link. When a target link is determined, the bot to target servers paths which include it are removed from the set of bot to target servers paths which is taken into account to determine a target link. The algorithm runs again until the attacker has selected as many target links as needed for cutting-off the target area from the rest of the Internet. This process chooses the link with the highest flow density in each execution. This characteristic guarantees that the chosen target link maximizes the effects of the attack. The

algorithm also enables selecting disjoint target links which are flooded in sets to avoid causing topology changes within a certain target area. If the adversary keeps constantly flooding a single set of target links, route changes will arise from link loss detection mechanisms in conventional routing protocols (OSPF, IS-IS, BGP) [4]. This feature helps in keeping the attack undetectable.

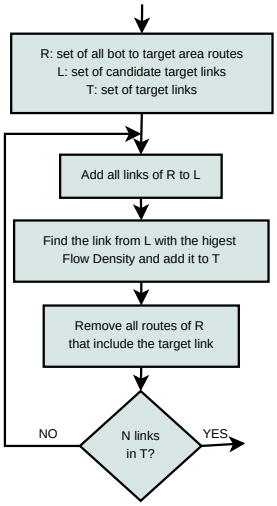



Figure 4.1: Target Link Selection Algorithm [4]

### 4.1.3 Bot Assignment Strategy

After the adversary has selected the target links to flood, she has to assign to her bots the decoy servers to send traffic to. Each bot generates low-rate traffic to a decoy server in the Crossfire attack in order not to be distinguished from legitimate sources and therefore not to be detected. Thus, each bot should generate flows leading to multiple decoy servers so as the aggregate traffic of a large number of bots is capable of flooding the target link. The total malicious traffic received by a decoy server should not have a high bandwidth demand to avoid raising an alarm for a potential attack. Hence, the selected bot assignment strategy should distribute the traffic between the bots and the decoy servers as evenly as possible. This is not a trivial problem and many strategies can be developed.

Taking into account that the target links can be in different geographical locations as well as the bots exist in multiple network domains and physical areas, we assume that the attacker assigns her bots to a set of decoy servers based on their proximity. In this way, the attacker maximizes the consequences of the Crossfire attack due to reduced delay between the bots and the assigned decoy servers. Therefore, to assign bots to decoy servers, we first need a map of the bots and the corresponding accessible decoy servers. As this task is out of the scope of this thesis, we employ a simplistic approach towards this issue. As we are aware of the bot locations and the topology characteristics on our experimental evaluation, we are able to efficiently assign sets of bots in corresponding sets of decoy servers. A more sophisticated approach could use a mapping from the IP addresses of both the bots and the decoy servers to their geolocations [20].

Considering the previously mentioned bot assignment criteria, we define a custom bot assignment scheme (Figure 4.2). After the clustering which we discussed has been completed, the decoy servers that are eligible to receive traffic by a set of bots (i.e., the routes towards them traverse a given target link) are mapped to corresponding First-In First-Out (FIFO) queues. Then, the set of bots of the cluster is placed in every FIFO queue. We have to note that all eligible decoy servers are accessible by all bots that are assigned to participate to this cluster. The number of flows which are necessary to occupy the total link capacity of, i.e., to flood, the given target link is afterwards computed. We assume that the attacker can predict the target link's bandwidth with high precision and that she is aware of the size of her generated flows. Based on the calculated required number of malicious flows, the attacker chooses a FIFO queue from the ones that were previously constructed to assign a single flow to one of its bots. After a single bot, which is contained in a queue, has been assigned to send traffic to the corresponding decoy server, the

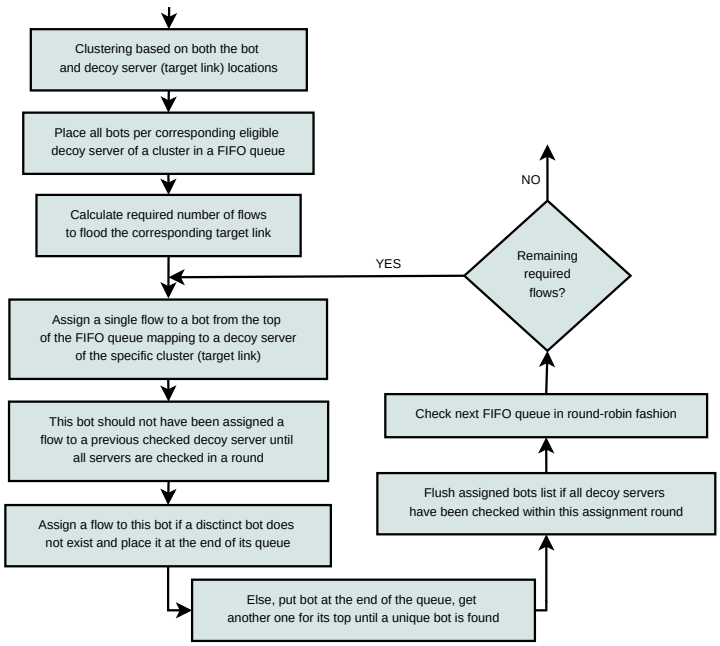



Figure 4.2: Bot Assignment Strategy

next FIFO queue in round robin fashion is considered. An assignment round is completed when every decoy server is included as destination to the corresponding assigned flows. During an assignment round, the scheme attempts to assign flows to distinct bots to ensure that all given bots of a cluster are assigned around the same number of flows. In case the assignment scenario does not allow any other unique bots to be assigned within an assignment round (due to the remaining required number of flows and the numbers of both the bots and decoy servers), then the bot under consideration is assigned a flow. Then, the next decoy server in order is considered as the destination in a to-be-assigned flow. When a flow assignment is attempted, the scheme selects a bot from the top of the FIFO queue and then this bot is placed at its end regardless of whether this bot is assigned to send a flow in the current selection or not. The algorithm runs until the number of the assigned flows is equal to the number of the required ones. The features of our custom bot assignment strategy enable as equal distribution of the flows with destinations to the eligible decoy servers as possible. This scheme also attempts to evenly allocate the required number of flows to the set of bots. Considering that the issue of equally allotting flows between bots and decoy servers is challenging as well as it is a minor task in the scope of this thesis, we did not undertake to devise a more sophisticated scheme.

### 4.1.4 Attack Traffic Generation

In addition to the previously discussed assignment phase, the rates of the corresponding assigned flows should be determined. Based on the aforementioned assignment criteria and assuming that the decoy servers are publicly accessible web servers, we consider that each attack flow consists of sending one HTTP GET message per second. Hence, the calculation of the number of flows which is necessary to flood a target link, mentioned in the previous section, depends on this unit flow-rate. However, in case of multiple topology changes around a given cluster of bots and decoy servers, it is possible that multiple attack flows are generated between the same pair of endpoints to flood a target link (i.e., the flow-rates are increased - $n$-fold rate regarding the mentioned unit flow-rate). In case of topology changes, the decoy servers' routes could shift to network paths disjoint with the target link calculated by the attacker. Thus, the number of the eligible to receive attack traffic decoy servers for a given cluster may be reduced and the corresponding flow-rates of the bots should be increased to be able to flood the target link.
We have to note that no potential attack alarm is raised at the decoy servers since the considered web traffic is a legitimate one and by taking into account the characteristics of our custom assignment strategy.

The bots are afterwards ordered by the adversary to launch the attack based on the assigned decoy servers and their corresponding flow-rates.
We implement the traffic generation using Scapy, a program that easily crafts network packets [21]. It is Python-based as the OpenFlow controller framework we use, i.e., POX. We first tried to generate web traffic using the GNU WGET program [22], but due to bugs and errors from using it within our emulation environment we ended up into using Scapy.

## 4.2 Defender: Detection and Mitigation Approach

The OpenFlow controller framework we employ to develop our security approach, i.e., POX, is based on an event-driven programming model. The controller registers for events and the developer implements event handlers [17].
The POX-based controller functionalities are implemented on multiple *Components* [19]. Therefore, our defence solution depends on *Components* which are provided by the POX framework for core functions and useful characteristics as well as on *Components* which we develop for the needs of our proposed scheme.
Considering the architecture of the proposed defender model which is defined in section 3.2, we will explain in detail how each of the architecture's components are implemented in the following sections.

### 4.2.1 Monitoring

As briefly discussed in section 3.2, the Monitoring component is responsible for identifying severe network link congestions. To enable this feature, the component measures periodically the bandwidth of the links within the considered network. It is implemented using the *Port Statistics* capability of the OpenFlow protocol. OpenFlow enables requesting information about the physical ports of a switch and the response message contains the transmitted bytes of the considered ports. By periodically parsing this data, the transmitted bandwidth of a given port can be determined. We assume that a single network link is connected to a port of a switch. Thus, the bandwidth of a port in a switch determines the bandwidth of the corresponding network link. It has to be mentioned that bandwidth measurements are made and link flow information is retrieved using the OpenFlow protocol. This implementation decision is not binding. One could use an external mechanism for these purposes (e.g., NetFlow) that would collaborate with the defence mechanism imlpemented in OpenFlow. We chose to use OpenFlow for bandwidth measurements and flow data collection since we wanted to build a complete network application that runs using OpenFlow.
In the case of the enterprise network, all its network links are monitored whereas the peering links to the enterprise network are only monitored within a provider network domain. This is due to our assumption in section 3.2 that a Crossfire attack may occur either in a local domain or on its peering links with its ISPs.
The actions of the monitoring component constitute the first step in our attempt to detect and mitigate the Crossfire attack. When a link congestion occurs, information about the flows passing through the congested link is requested from the corresponding port and switch. This is enabled by employing the *Individual Flow Statistics* mechanism of OpenFlow. The congested flows are grouped by distinct destination Internet Protocol (IP) addresses. Bandwidth measurements are then conducted to the congested link for each group by using the *Aggregate Flow Statistics* feature of OpenFlow. Our goal is to identify the bandwidth distribution of the aggregate flows. The groups are sorted in a way that the destinations occupying around the same bandwidth at the congested link are higher in order (Figure 4.4). This is because the attacker aims to equally distribute the number of flows which are required to flood a target link among the decoy servers. Therefore, this classification mechanism is a heuristic in the attempt to exclude target servers to be placed in higher order than many decoy servers. The aggregate flows in a high order will be selected for rerouting as long as their bandwidth requirements meet a rerouting bandwidth threshold. The selection of the destinations to reroute is completed as long as the total bandwidth demands of all the selected for rerouting groups reach the defined threshold or the next group in order does not fit within the threshold in terms of bandwidth. We assume that the target servers may not occupy around the same bandwidth as the decoy servers. Thus, this heuristic avoids to reroute flows with destinations leading to target servers. Even in the case where the heuristic

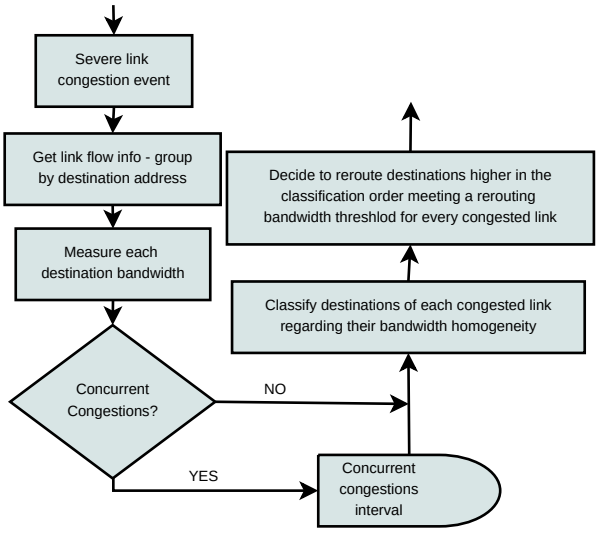



Figure 4.3: Link Congestion Monitoring

is mislead and leads to unwanted target area traffic rerouting, the detection algorithm does not break but needs more rounds in order to clearly identify attack traffic.


Figure 4.4: Example of our Classification Mechanism

Then, the routing component is triggered by a message from the monitoring one to deal with the congested traffic. The message contains information about the congested link such as the aggregate flows' destination IPs that were selected to be rerouted, their bandwidth and their corresponding source IPs. It also contains a matrix with the available bandwidth of every network link of the considered network domain to facilitate the rerouting decision of the routing component. Furthermore, it also includes a matrix of the level of suspiciousness of some source addresses. This matrix enables rate-limiting to flows with sources whose rank exceeds a predefined threshold. In the following paragraphs within this section, we will explain how this matrix is constructed and its use. In case of multiple congested links within the same time interval, these link congestions are handled together as a batch, so as a single message is sent to the routing component. Figure 4.3 presents this part of our algorithm. The control is then passed to the routing component, which is explained in section 4.2.2.
After the routing component has rerouted the selected congested flows, the monitoring component is triggered again. This paragraph describes the other functionality of the monitoring component that monitors the reroutings made by the routing component. The rerouted flows and the network links towards the new paths which are disjoint with the previous routes are recorded. We aim at identifying potential changes in the behaviour of the rerouted flows in the event of a subsequent congestion. We expect that the attacker may stop sending traffic towards the new routes as these may not include potential target links. She may reassign her bots to send traffic to other destinations instead. When a subsequent congestion occurs in a non rerouted link, the *Individual Flow Statistics* mechanism of OpenFlow is used as discussed before. However, in this case it requests information not only about the flows passing through the congested link but also about

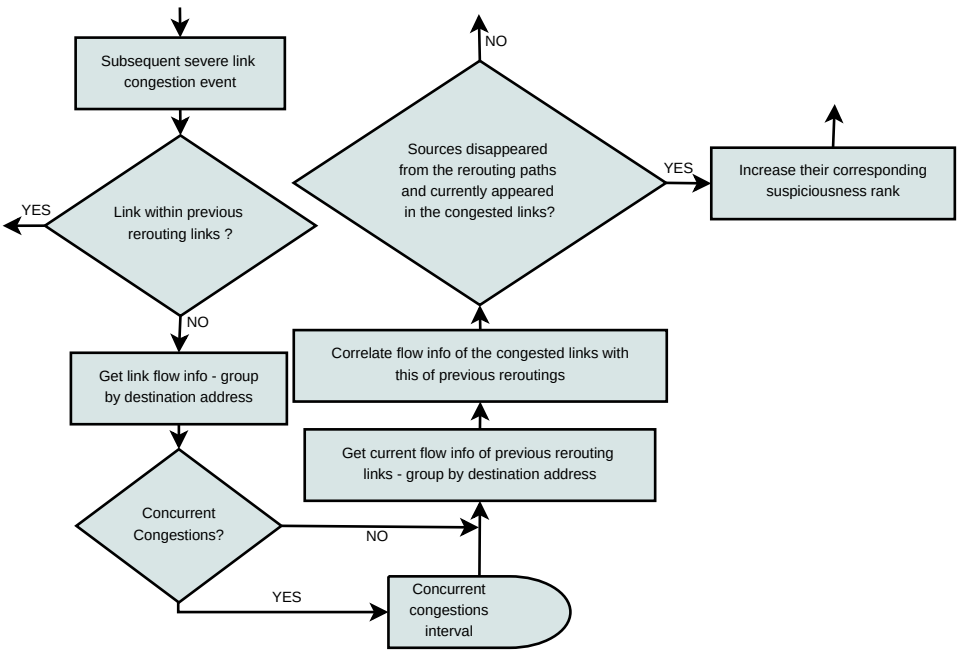



Figure 4.5: Rerouting Monitoring

the flows passing through the recorded rerouted routes. For scalability reasons, the information about the rerouted flows is requested by a single switch, the first one towards the corresponding recorded network paths. The new congestion may appear in a different link with respect to the link of a previous congestion and many concurrent congestions may occur within the same time interval which should be handled together as a batch as discussed before. Therefore, the correlation between the recorded flows and the flows which are present in the current flooded links is made after the time that is considered to deal with concurrent network congestions has elapsed. We first check whether the recorded rerouted flows are still present at the rerouted paths or not and then we verify whether the flows that disappeared from the rerouted network links are now present in the current flooded links. If this is the case, the corresponding source IPs are marked as suspicious, incrementing a corresponding counter. We have to note that in a correlation round between the set of the rerouted links and the set of the concurrent target links, we only increment the score of the suspicious sources by one even if these sources are successfully correlated more than once within the same correlation round. This is a conservative approach but it provides flexibility in avoiding false positives, e.g., when the suspicious source is an IP prefix from which both bots and legitimate hosts originate. This scheme also deals with the congestions due to flash crowds. After a rerouting is completed, the rerouted flows that may congest the rerouted links will not be considered as suspicious. If the attacker reacts in as many reroutings as possible, she makes us more confident to detect the malicious traffic with higher probability. Figure 4.5 shows this part of our methodology.

### 4.2.2 Routing

The routing component is enabled by the monitoring one in the presence of network link congestions. The monitoring component reports to the routing one the flows to be rerouted along with their corresponding bandwidth requirements and a matrix containing the available bandwidth of all the network links. A matrix which contains some source addresses and their score of being considered as malicious is also reported. The routing component attempts to find new routing paths for the congested flows taking into consideration both the bandwidth requirements of the flows and the bandwidth availability of the network links along the alternative network routes. It also sets a rate limit to flows with certain source addresses. This feature is enabled by installing two queues in every port of a switch. The first queue is the default one which has the size of the nominal network link bandwidth and the second one has a reduced size compared to the first to achieve setting a limit to the rates of the flows which are placed in this queue.

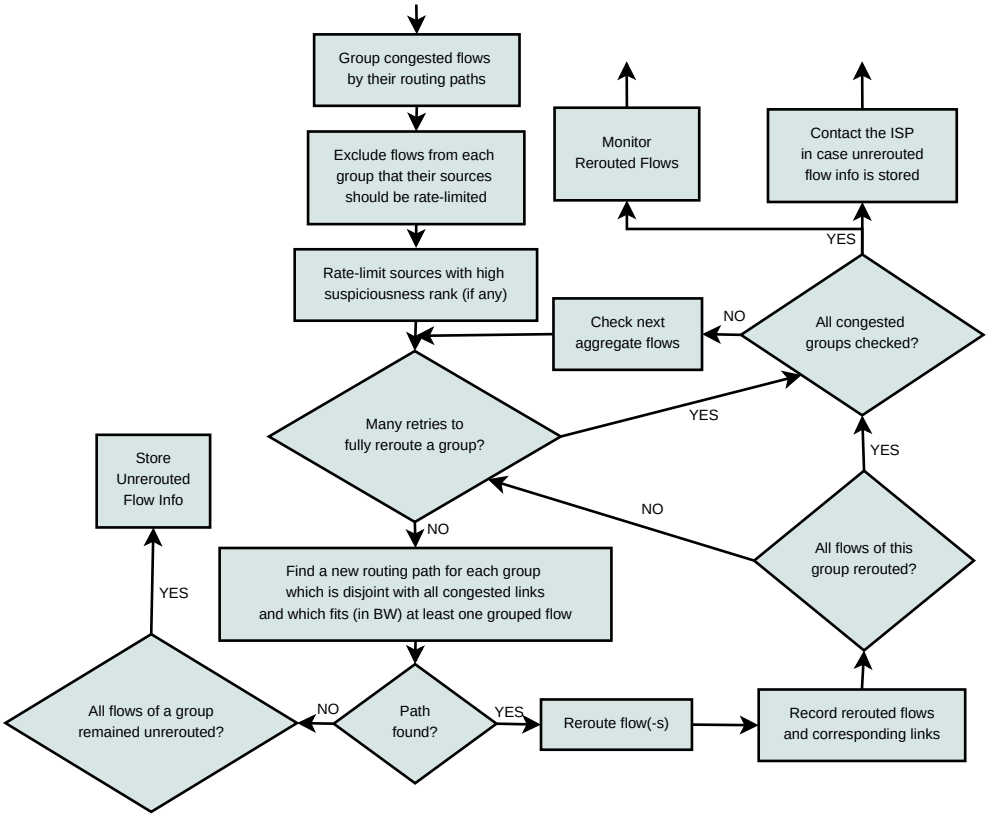



Figure 4.6: Congested Flows Rerouting

A new path is generated considering all the reported congested links. Thus, the new path will not involve another already congested link. To optimize the search of a new routing path for the reported flows, the flows are aggregated based on their existing path. It is first checked whether some of the considered flows should be rate-limited or not based on the corresponding matrix and then the groups are formed. For each group, the routing component attempts to find a new path which will be disjoint with all the reported congested links. If such path is found, then the available bandwidth of all the new path's network links is checked. If the links can accommodate flows from the selected group, these flows are immediately rerouted. In case that flows from this group do not further fit in this new path due to its limited remaining available bandwidth, the routing component attempts to discover a new path that will carry the rest. If there is not another rerouting possibility or the component has made many attempts to reroute the remaining unrerouted flows of this group and still not all of them are successfully rerouted, then the next group of flows is examined. This happens for scalability reasons. In case that not even a single flow of a group is rerouted due to no rerouting possibilities, then the corresponding ISP should be notified to reroute these flows in an inter-domain level. We choose not to contact the ISP if at least a single flow of a group is rerouted, as we consider that this successful rerouting partly alleviates the congestion. We have to mention that the OpenFlow reaction time in detecting and mitigating a congestion is orders of magnitude less than the time that takes the attacker to readjust to this rerouting. The attacker has to execute multiple traceroutes to reconstruct her link-map in this case. Therefore, a Crossfire attack cannot have a permanent duration.

The information about the flows which were successfully rerouted, their corresponding new routing links which are disjoint with their previous paths and the information about the first hop towards the new routes are all reported back to the monitoring component in the attempt to detect the malicious traffic. The last part of section 4.2.1 explains how the reported information is used. Figure 4.6 illustrates the actions of the routing component.

### 4.2.3 Inter-controller Communication

The POX *Messenger* component provides an interface that allows external services to communicate with POX using bidirectional JSON-encoded messages [23]. The communication is enabled by selecting one of the *transports* (TCP socket and HTTP transport in the *carp* branch of POX). POX and external software exchange messages via certain *channels* which are determined when a connection between them is established. The *Messenger* component also provides a scheme that enables POX to automatically process messages received on a channel, called *ChannelBot* [24]. Taking advantage of the benefits of the *Messenger* component, we build a custom Messenger-based framework that employs TCP socket transport to enable POX inter-controller communication. The *Messenger* component offers a simple client script (test_client.py) to test the communication with POX. Based on this sample script, we enable the communication between POX controllers using a client-server model. In addition to the simplicity of this approach, we employ this model because we might not know which controller initiates first a connection to a peer. As a result, each POX controller implements both a client and a server to enable inter-controller communication. Each controller employs its server to receive messages and responds using its client. In particular, messages are received by a *ChannelBot* which notifies the server about the reception of messages. The server then decides how to process them and can send back its replies by passing the corresponding responses to its client. The server interacts with the *Monitoring* component of our application either by sending information to it or by getting data from it that needs to be transmitted to other POX controller (Defender Model Architecture - Figure 3.2). Figure 4.7 illustrates the details of the inter-controller communication interface.

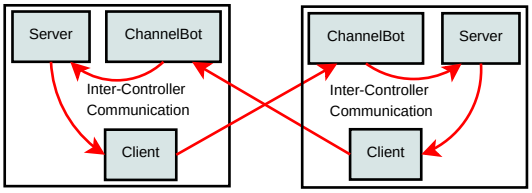



Figure 4.7: Inter-Controller Communication Interface

We assume that POX controllers are connected in pairs imitating the Border Gateway Protocol (BGP) model [25]. Thus, each pair of controllers uses unique channels to communicate. Each connection of the client of a controller to the server of another controller is determined by a unique channel. Therefore, the communication between a pair of controllers employs two channels, one for each communication direction. We consider two types of channels between every pair of controllers, the *Control* channel that is used for exchanging connectivity and reachability information and the *DoS* channel that is used for exchanging DoS-specific messages. We have to remember that we assume that a single controller is responsible for a network domain. The *Control* channel remains always active and a *DoS* channel is created whenever a DoS event occurs within a network domain and the controller of another network should be notified. Before a channel, either *Control* or *DoS*, is created, we employ the *ChannelBot* framework to authenticate the peers. The client of a controller connects with the *AuthChannelBot* of the other controller through a default channel to request a communication channel, either a *Control* or a *DoS* one. The *AuthChannelBot* generates and sends a nonce to the client and the client replies with the Message Authentication Code (MAC) of the nonce. We assume that each pair of controllers share a secret key. Therefore, the controller that initiated the connection is successfully authenticated and a communication channel for this direction is created. After the authentication phase has completed, a *ControlChannelBot* or a *DoSChannelBot* is responsible for receiving messages from the created *Control* or *DoS* channel respectively.
In our approach, the OpenFlow controllers exchange different types of messages depending on the role each controller has. The controller of the enterprise network exchanges messages with its peer on a direct upstream provider and the upstream providers' controllers can also communicate with each other. Since we consider an enterprise network connected to multiple direct upstream providers as the environment for our approach (chapter 3), attack traffic may originate from the direct upstream provider networks. As discussed, we assume no traffic is generated within the enterprise network. When link flooding occurs within the enterprise network, the congested flows which cannot be rerouted within the enterprise network are reported by the network controller to

the corresponding upstream providers' controllers in order to be rerouted through another peering link between the enterprise network and the corresponding provider networks. The enterprise network controller reports the flows to be rerouted, their bandwidth requirements and a matrix which contains the sources that are marked as suspicious and their suspiciousness level. Since the attacker reacts to the defender's rerouting when she identifies the route changes, the new target links may potentially be in the peering links to the local network. Therefore, the suspiciousness information is used when such an event occurs.
An ISP's controller which received a rerouting request by the enterprise network attempts to reroute the reported flows using an algorithm similar to the one described in section 4.2.2. The provider selects to which peering link the flows are rerouted. The decision is made based on the bandwidth availability of the alternative peering links and the bandwidth requirements of the reported flows. The provider attempts to reroute as many flows as possible. After the rerouting has completed, the provider stores the information about the flows that were successfully rerouted and the corresponding rerouting peering links and notifies the enterprise network by sending the same information.
In the event of a peering link congestion after a rerouting has been previously completed due to a reported enterprise network link congestion or an old peering link flooding, the provider's controller attempts to correlate the stored information about the previous rerouting with the information about the current congestion in the same way as the correlation which happens in the enterprise network domain and was previously described in the last paragraph of section 4.2.1. In case of a successful correlation, the controller of the provider network sends an asynchronous message to the enterprise network controller to update the scores of the sources marked by this correlation on its local suspiciousness matrix. Of course, the controller of the provider network also updates its local counters. The provider's controller enforces a limit to the rate of the flows whose source ranks surpass a predefined threshold in the same way as described in section 4.2.2. There is also the case in which a provider cannot reroute any of the flows passing through specific peering links (in case a congestion occurs in the peering links or of a reported enterprise network congestion) due to lack of alternative peering links or of available resources on them and a physically connected collaborating provider should be contacted. This type of interaction will be taken into account in order to extend our current approach in a future work.

## 4.3 Auxiliary Components

In order to dynamically discover a network topology and detect any addition or removal of a network link, POX provides a custom component, the *openflow.discovery*, which sends custom LLDP messages to identify topology changes. The *Monitoring* component should be aware of the links that are included in the topology since, as we discussed before, it monitors the network links. Therefore the *Monitoring* component depends on the *Discovery* one.

### 4.3.1 Flow Routing Management

We consider that our defence mechanism against the Crossfire attack is deployed on networks which have a baseline routing policy. The traffic on these networks is identified on a flow level. For simplicity, we assume that flows prefer to be routed along the Shortest Path. Thus, we develop a custom component that implements the Dijkstra algorithm to discover shortest path routes along a network topology. This component depends on the *Discovery* component to dynamically discover the topology of a network. We also assume that the network controller holds information about which traffic passes through and how this traffic is routed within a network domain. This knowledge facilitates flow management in the presence of the Crossfire attack. In our implementation, the controller aggregates traffic flows by their source and destination switch pairs and by the network topology information that each set of flows "sees". The default topology for all flow groups is the full real network topology. The controller stores information about the adjacent nodes of all nodes in the topology. In case the Crossfire attack occurs, the *Routing* component which runs on top of the custom routing management component employs the network controller's knowledge to mitigate severe network congestions. The *Routing* component (described in section 4.2.2) creates virtual topologies on top of the default one by removing the congested network links from the default topology information. In this way, a shortest path that bypasses the flooded network links can be found while keeping the default routing information

intact. The congested flows are mapped to the corresponding virtual topologies and then the custom routing management component groups them as discussed and generates their shortest path routes considering their virtual topologies. Therefore, the *Routing* component enforces routing policies which supersede the baseline routing policy in order to mitigate severe congestions. We have to mention that the type of the baseline routing policy (Shortest Path routing in this case) is not binding to our defence approach. However, our solution depends on the existence of an underlying flow routing mechanism similar to the one that was previously described.

### 4.3.2 Enabling *Traceroute* Capability in POX and OpenFlow

To build the link-map, the attacker runs multiple traceroutes around the target area (section 4.1.1). In general, there is not a built-in mechanism either in POX or OpenFlow that would support executing traceroutes. As mentioned in the beginning of this chapter, POX implements OpenFlow version 1.0 which does not support the installation of any rules at the switches that would enable them to decrement the value of the Time-To-Live (TTL) field of an IP packet. Thus, IP packets (and hence traceroute packets) must be first forwarded to the OpenFlow controller which decreases their TTL value and then sends them back to the switches to forward them to their next hop. Furthermore, the packet library of POX (pox.lib.packet) and more specifically the source file which enables processing *icmp* packets in POX does not include an ICMP Time Exceeded packet structure. This structure is necessary to identify intermediate hops when running traceroute to a destination. Accordingly, to enable this capability, we implement a custom packet struct based on the other ICMP packet structures in POX. These are the necessary steps towards enabling traceroutes in POX and OpenFlow version 1.0. We have to highlight that there is a limitation in the scalability of this approach. The execution of traceroutes using the OpenFlow controller as proxy server adds a significant overhead and imposes load in the form of *Packet-In*s and *Packet-Out*s to the network controller. Furthermore, each probe of a traceroute should first pass from the OpenFlow controller before it is forwarded to its next hop. Therefore, the latencies displayed for each hop in a traceroute result are not indicative of the latency to reach this hop. Of course, in practice the controller would not suffer this extra load and traceroute latencies would be indicative since there are more scalable mechanisms used in practice to deal with traceroute responses (ICMP agents on switches, distributed traceroute responders, etc.). The traceroute controller application was specifically developed for exhibition and demonstration reasons and to provide a classic means for an attacker to collect information about the IP-level topology of a network. Enabling traceroute in SDN to gather this IP-level information is of course not designed to help the attacker, but to enable an external source to troubleshoot routing problems. Traceroutes are useful not for attack reasons but for troubleshooting and we do not think that simply disabling traceroutes everywhere is a sound idea (we would not have so straightforward attacks but also not so straightforward routing troubleshooting).

# Chapter 5

# Experimental Evaluation

In this chapter, we present the results produced by evaluating the performance of our defence approach. At first, we describe how we set up our emulations. Then, we give details about our performance metrics and we present the results in the corresponding sections. We afterwards mention potential origins of induced error to our measurements and we finally discuss about the conclusions drawn from our results.

## 5.1 Setup

To evaluate how our defence solution reacts against the Crossfire attack, we conduct emulations. We have built a working prototype based on our attack countermeasures and we would like to assess its performance as if it is deployed in a production environment. We deploy *Mininet* [15] for network emulation. It is an easy and lightweight network emulator for developing OpenFlow-based network control applications. As we discussed in chapter 4, we use *POX* as our OpenFlow-based development platform. *POX* and *Mininet* are pre-installed in a single Virtual Machine which can be easily downloaded and installed [26]. We use VirtualBox as our virtualization software [27] since the VM is a VirtualBox image by default and its installation instructions assume that it runs on VirtualBox [26]. The VM has a Linux 64-bit Operating System (Ubuntu 12.10) and contains Mininet version 2.0.0 and the *master* branch of POX. As mentioned in chapter 4, the POX's *carp* branch is used to develop our application. Thus, this branch has to be installed [28]. We have to note that the default switches in a Mininet network are based on Open vSwitch (OVS) [29], an open source virtual switch, which enables the use of OpenFlow. The default OVS version in Mininet version 2.0.0 is 1.4.3 which is the one used in our OVS configurations (section 4.2.2). We have to point out that both the POX controller and the Mininet network for our emulations run within the same VM since our measurements depend on timestamps provided by both tools. If POX and Mininet run in separate VMs, it is highly possible that the corresponding timestamps would not be synchronized. Our system that hosts this VM has a dual-core 2.67 GHz processor and 4 GB RAM. To conduct our emulations, we have assigned the use of two cores and 1.5 GB memory to the VM through VirtualBox configuration.
For the rest of this chapter, we will conduct measurements assessing the performance of our defence solution in case the Crossfire attack occurs within an enterprise network. We use a theoretical network topology (Figure 5.1) for the role of the enterprise network. This topology highlights the Crossfire attack and the proposed countermeasure.
The theoretical topology consists of four entry switches and it is assumed that the attacker considers the server at the bottom of this topology as the target server (area). The rest of the servers in this topology are considered by the attacker as decoy servers. All the networks links in the topology have a capacity of 200 kbps. Furthermore, three bots and two legitimate hosts are connected to each one of the four border switches of the network. We consider that these hosts generate traffic originated from the Internet. Thus, the links connected to the entry switches have a 100 ms delay. Bots generate flows of 4 kbps unit flow rate. To emulate legitimate traffic, we assume that users' arrival and service time follow an exponential distribution. Regarding the bots, it is assumed that certain bots are clustered during the bot assignment phase (section 4.1.3). The bots accessing the enterprise network through the first two and the rest two border switches are clustered accordingly. We have also to remember that it is considered that the attacker employs

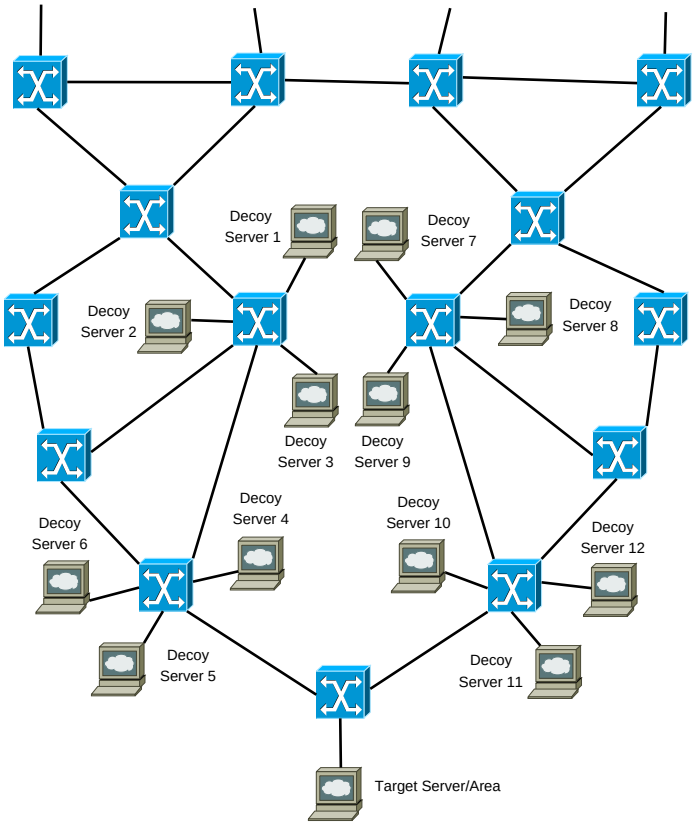



Figure 5.1: Theoretical Topology

a fix number of bots to launch the Crossfire attack.
We construct the given topology in Mininet and we configure the network link delay and capacity using the TCLink class which Mininet offers. We configure the queues in every OVS switch of the network before Mininet is launched. As discussed, we use two queues to enforce a rate limit to potentially malicious flows. The default queue for all flows has the same size as the corresponding network link capacity and the queue to which the malicious flows are assigned for rate-limiting has half the size of the default one. We have downscaled our experiments since our computer and hence the VM used for our emulations have limited resources.
Our defence approach involves configuring many parameters. We mention their default values used during our emulations. The monitoring component conducts bandwidth measurements to each network link of the topology every 2 seconds. It also estimates the bandwidth of all the grouped flows passing through a congested link by making measurements every 1 sec and for 3 seconds in total. We set the time interval that is used to handle as a batch all the link congestions which occur within it to 5 seconds. The congested flows that are selected to be rerouted form at most the 40 % of the total link occupied bandwidth. A severe link congestion is identified when the bandwidth the link carries exceeds 95 % of its nominal capacity. We consider that a source address is malicious when its level of suspiciousness reaches 3. Table 5.1 accumulates all the aforementioned default configuration values for our emulations.
Before we start conducting our emulations, we execute *Ping* commands from all the legitimate users and bots to every server within the theoretical topology. It is necessary for bots and users to be aware of a map of IP to MAC addresses to avoid significant traffic overhead during our experiments in the forms of ARP messages and Packet-Ins.

## 5.2 Reaction Times

The design of our security mechanism against the Crossfire attack is based on the fact that the defender reacts fast and the attacker cannot respond quickly to the defender's reaction. Thus, the attack is mitigated. In this section and in the following subsections, we evaluate the accuracy

| Parameter | Default Value |
|---|---|
| Network Link Capacity | 200 kbps |
| Unit Flow Rate | 4 kbps |
| External Network Link Delay | 100 ms |
| Network Link BW Measurements Interval | 2 secs |
| Destination BW Measurements Interval | 1 secs |
| Total Destination BW Measurements Time | 3 secs |
| Concurrent Link Congestions Time Interval | 5 secs |
| Congested Link Occupied BW Rerouting Threshold | 40 % of nominal link BW |
| Severe Link Ccongestion BW Threshold | 95 % of nominal link BW |
| Suspiciousness Level Rate-Limit Threshold | 3 |

Table 5.1: Default Experimental Configuration

of this approach.

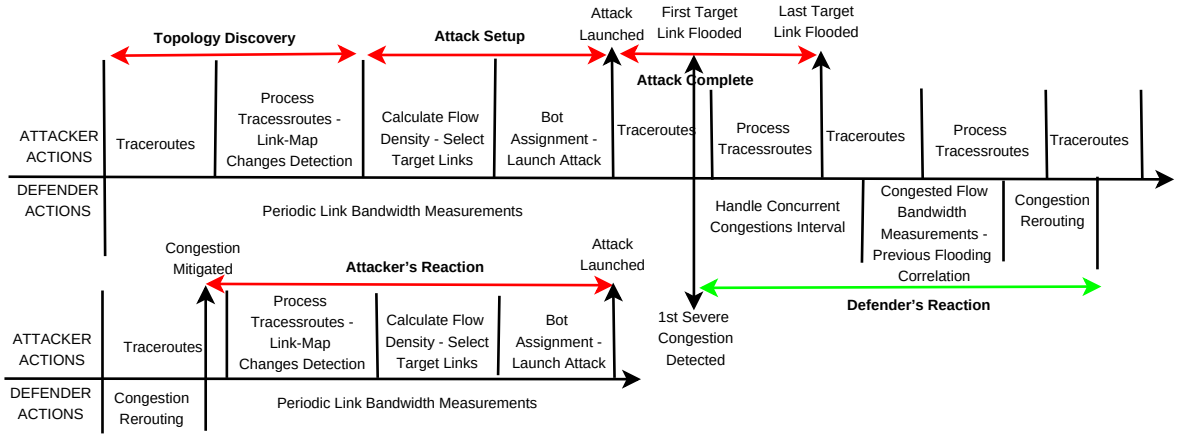



Figure 5.2: Attacker's and Defender's Event Sequence Diagram

In Figure 5.2 which is based on what we have already discussed on chapter 3, the sequential steps of the attacker and the defender actions are presented. We will explain these actions in detail in the following subsections.

### 5.2.1 Reaction Time of Attacker

In Figure 5.2, the Crossfire attack is divided in three parts: the dynamic topology discovery, the set-up of the attack and the completion of the attack. The adversary constantly monitors the target area in an attempt to discover the target topology and any dynamic changes on it. In the event of route changes, she sets-up and launches the attack. The attack is considered successful when all the calculated target links are flooded. We have to note that after the attack has been launched and while attack traffic is being sent, the attacker still keeps track of the target area. In the dynamic topology discovery part of the attack, the attacker runs multiple traceroutes and constructs the map of the network links around the target area. Since the attacker may execute traceroutes while topology changes are in progress, this part of the attack may have to be executed many times until a stable link-map is built. We assume that running two extra sets of traceroutes when a link-map change has been detected can verify that a stable link-map will be constructed. Considering that executing traceroutes in OpenFlow adds a significant overhead to the network controller, we run sequential traceroutes (one host each time) instead of parallel ones (all hosts concurrently) to be able to successfully build a map of the network links around the target area. However, this strategy results in a high amount of time needed for executing all the traceroutes necessary to build a link-map. Therefore, instead of constantly running traceroutes to detect route changes and also running two extra sets of traceroutes to verify the stability of the contructed link-map, we periodically run a single set of traceroutes under the assumption that

topology changes only occur within the time interval between two consecutive sets of traceroutes. To represent the parallel traceroutes in theory with the sequential traceroutes in practice, we store data about the minimum and maximum time for executing a single from a set of traceroutes. It is then assumed that the time needed to run the a set of traceroutes is equal to the average of its minimum and maximum values. Although this data doesn't show exactly how much time a stable link-map construction needs, it helps us as to estimate the reaction time of the attacker. We believe that the execution of traceroutes until a stable link-map is determined spends most of the time needed to successfully launch a Crossfire attack due to the Round Trip Time between a bot and a decoy or target server.

In the attack set-up, the flow density of the links of the link-map is computed and the links to flood are selected. The attacker also assigns her bots to send traffic to certain decoy servers and then orders them to launch the attack. The attack is considered successful (attack completion) when all the calculated target links are flooded.

In the following paragraphs, we measure the attack setup and the attack success times and the time it takes the attacker to react to our defence mechanism. We conduct multiple emulations modifying the attacker's unit flow rate to verify how the corresponding times are affected.

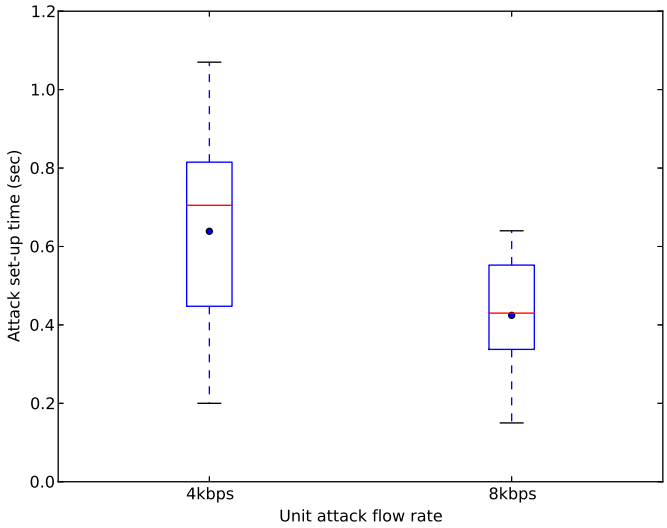



Figure 5.3: Attack Setup Time

In Figure 5.3, the boxplots of the attack setup time when generating attack traffic of 4 and 8 kbps as unit flow rates are presented. It is clear that the time is decreased when increasing the unit flow rate since the attack setup includes the bot assignment phase. In this phase, the attacker calculates the number of flows (and then assigns the flows to her bots) which is required to flood a certain target link. As discussed, we assume that the bots use a fixed unit flow rate and that the adversary is aware of the target link capacity. By increasing the unit flow rate, less flows are needed to flood a target link and therefore the bot assignment scheme is completed faster.

In Figure 5.4, the boxplots of the attack success (complete) time when generating attack traffic of 4 and 8 kbps are presented. It is clear that the time is decreased when increasing the unit flow rate since the attack is launched in a higher rate. Even though the amount of the cumulative traffic in the target links is the same in both cases, the higher packet length that we use in order to increase the unit flow rate is responsible for this behaviour. We have to mention that this measurement is biased by our link bandwidth measurements mechanism. It is assumed that an attack is successful when all the target links are identified as severely congested by our bandwidth measurements mechanism. This measurement clearly depends on how often the network links are polled for their occupied bandwidth.

In Figure 5.5, the boxplots of the reaction time of the attacker when generating attack traffic of 4 and 8 kbps are presented. The reaction time is presented in Figure 5.2 and is defined as the time interval between a successful response to the attack (congestion mitigation) by the defender's

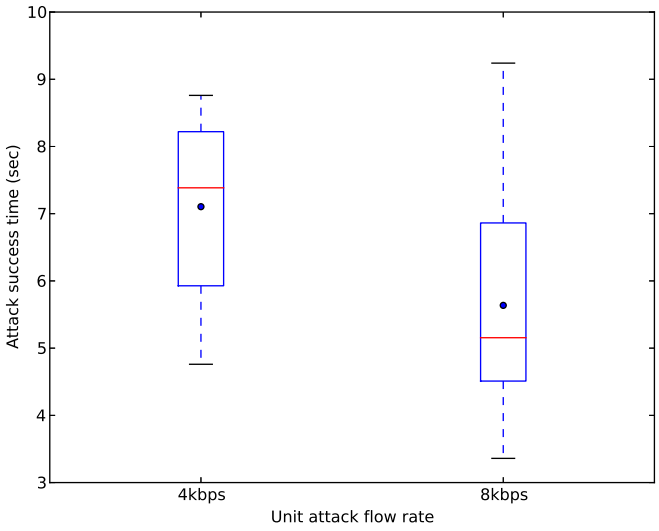



Figure 5.4: Attack Success Time

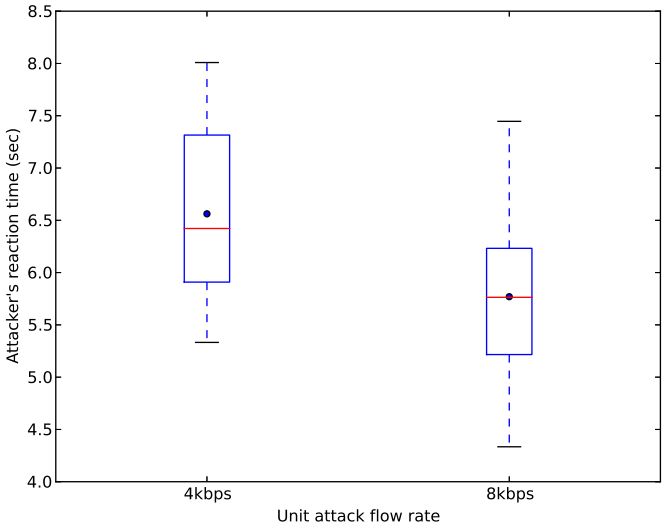



Figure 5.5: Attacker's Reaction Time

mechanism and until when an attack is launched once again. Since the time spent for topology discovery (traceroutes and link-map contruction) is highly topology-dependent, the attacker's reaction time mainly depends on the attack setup time. Therefore, it is clear why the reaction time of the attacker has the same behaviour as the attacker setup time when modifying the unit flow rate. It is noteworthy to mention that as described in the first paragraph of this section, this is an estimate of the attacker's reaction time since we run traceroutes in a sequential instead of in a parallel fashion.

This metric is highly important for assessing the performance of our defence countermeasure against the Crossfire attack. In section 5.5, the attacker's reaction time is compared with the one of the defender to evaluate the accuracy of our security solution.

### 5.2.2 Reaction Time of Defender

As shown in Figure 5.2, the reaction time of the defender is defined as the time interval between when the first severe congestion is detected by the defender and the time when all the concurrent congestions are mitigated (rerouted). It is assumed that many severe congestions may occur within the same time interval to successfully prevent access to the target area. The boxplots of the reaction time of the attacker when generating attack traffic of 4 and 8 kbps are presented in Figure 5.6.

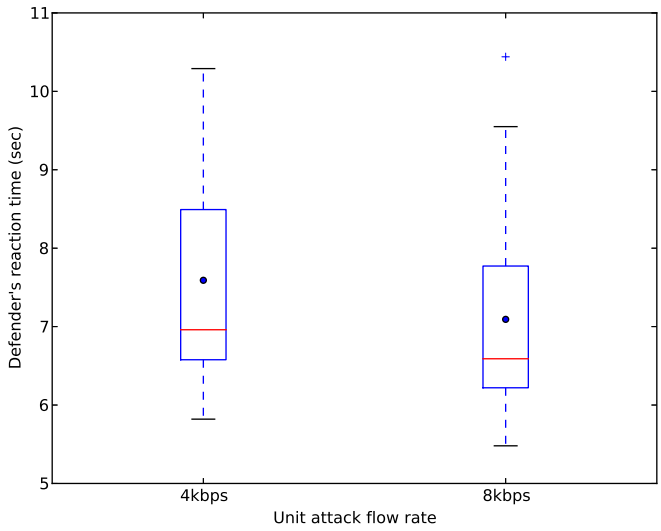
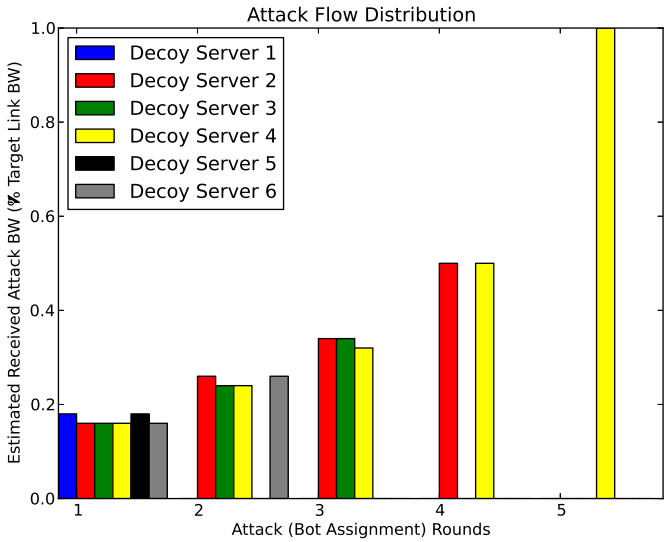



Figure 5.6: Defender's Reaction Time

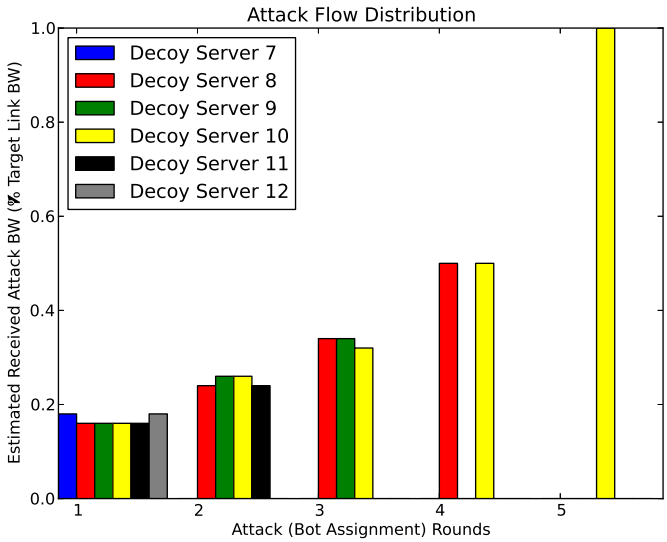
There is not any clear correlation between the defender's reaction times when using a different unit flow rate since there is not much difference between the defender's reaction times. Our OpenFlow-based defence mechanism receives around the same amount of flow information in the event of a severe link congestion in both cases. Our controller application then decides how to reroute the congested traffic. The rerouting decision has nothing to do with the unit rate of the generated flows when using this topology due to its abundance of resources. A rerouting path is always directly found without having to search for many paths to accommodate the flows to be rerouted. Thus, the insignificant difference between the defender's reaction times may arise from various sources of error (section 5.4).

## 5.3 Attack Cost Increase

As mentioned, the adversary attempts to equally distribute among her bots and the decoy servers the required number of flows (bot IP, decoy server IP tuples) when aiming to flood a target link. This strategy ensures that the Crossfire attack remains undetected at the decoy servers by conventional security solutions. Due to the design of our solution, the congested traffic is rerouted in network paths disjoint to the flooded links. The attacker then identifies changes on her link-map and she readjusts her attack properly by recalculating the target links. Our approach employs destination-based rerouting for the congested flows destined to decoy servers. Therefore, the set of decoy servers that are included in bot-to-decoy-server routes which traverse a target link may change (may be reduced) due to the defender's reaction. This is a very topology-dependent issue. The security solution attempts to disrupt the initial distribution during the bot assignment phase of the attack when the attack was launched for the first time. The scheme attempts to "force" the attacker to increase the assigned number of flows destined to certain decoy servers. In this way, the total traffic received by the decoy servers is increased raising the probability that the attack is detected at the decoy servers. This is defined as an increase to the cost of the attack. In case a given set of bots are assigned to flood a target link, as in our case (clustering - section




(a) Assigned Attack Flow Distribution - 1st Target Link




(b) Assigned Attack Flow Distribution - 2nd Target Link

Figure 5.7: Attack Cost Increase

4.1.3), and the set of decoy servers to receive attack traffic is reduced by our reroutings, then each decoy server of this set will now receive higher amount of malicious traffic so as the attack keeps successfully flooding a target link. In Figure 5.7, it is clear that our assumption is valid, but we have to mention that this a highly topology-dependent feature. To fully understand this meaning of this figure, we have to take into account the topology we used in our emulations (Figure 5.1). In our emulation, the attacker calculates the same two target links each time due to the position of the target and the decoy servers within the theoretical topology. The defender reroutes portions of the attack traffic by changing the routes towards some decoy servers which are selected by our classification mechanism. The target server routes remain unchanged since it does not receive highly homogeneous bandwidth compared with that of the other servers within this topology. The attacker then responds to this shift but due to nature of this topology keeps flooding the same target links and therefore sending traffic to the unrerouted decoy servers. As a result, the bandwidth these servers receive is increased in each attack round.

We have also to mention this graph is based on data during the bot assignment phase of the

attacker each time the attacker (re-)launches the attack. We can notice from the figure that five attack rounds exist. Due to the nature of this topology (the attacker calculates the same target links every time), the suspiciousness level of the bots reaches three (equal to the default rate-limit threshold) at the forth round. Thus, the Crossfire attack sources will be detected in the fifth round when a link flooding generated by the attack is identified by our scheme.

## 5.4 Sources of Error

The results we presented in the previous sections are biased by various factors. First of all, Mininet has limitations in generating high-rate traffic. Since Mininet hosts depend on the shared underlying Linux Operating System, they schedule traffic to be generated using the TCP stack of Linux OS. This makes Mininet by definition not an accurate way to timestamp and control the generated traffic and this is an important limitation. Also, the popen() interface in Mininet had an unexpected behaviour when trying to launch traffic from each of the Mininet hosts. Our developed Shell script for generating traffic was unexpectedly stopping after some time. This was the reason why we used the cmd() method instead, but additionally we had to employ a custom mechanism to ensure that the generating process would not be killed. Mininet does not offer a clean and standard way of generating high rate traffic, so our approach probably induced an error to our measurements. Furthermore, Scapy, the program we used for generating packets, imposes an overhead to the generated traffic and therefore to the results. Scapy was constantly executed by Mininet hosts during our emulations since it was an important part of the traffic generation. Moreover, both POX and Mininet run in the same Virtual Machine during our emulations. Since they shared the same resources, this may have affected our results. In addition, our findings may have been influenced by the resources as well as the running processes of the computer which hosts the VM we used for our emulations.

## 5.5 Insights

In this section, we discuss the conclusions which arise from our results. In our approach, the defender reacts to a potential Crossfire attack and the attacker readjusts to this reaction in order to maintain the attack active. The correlation between the reaction times of the defender and the attacker determines how fast the Crossfire attack is mitigated and can be detected. Our SDN-based mechanism reacts fast against the Crossfire attack while the attacker has to readjust to our mitigation strategy. The attacker's reaction time (Figure 5.5) therefore determines how long the attack stays ineffective. On the other hand, the defender's reaction time (Figure 5.6) determines how long the attack is effective upon its identification in the form of severe link congestions until its mitigation. The reaction time of the defender also determines how fast in total the Crossfire attack sources can be detected while the attack is in progress. Since our solution "forces" the adversary to respond each time and provides detection of the malicious sources after many executions of our algorithm, an approach which offers fast reaction leads to faster detection in total.

The Crossfire is a distributed DoS attack in which multiple target links are flooded within a certain time interval (concurrently) to disrupt the target area's Internet connectivity. Therefore, we consider a time interval in our approach that severe link congestions which occur within it as handled together as a batch. This considered interval is necessary to enable severe congestion mitigation in a correct way since by using this mechanism it is not possible to route the congested traffic along a network path which includes a target link that is about to flood. In these emulations, we have set this time interval to 5.0 seconds. We have also taken this time interval into account when calculating the reaction time of the defender. That is the reason why the Figure 5.6 displays high values. It is clear by comparing Figures 5.6 and 5.5, that if we do not take the concurrent link congestions time interval into account, the defender reacts much faster than the attacker validating the accuracy of our approach.

# Chapter 6

# Future Work

Designing, implementing and evaluating an attack countermeasure to a reactive Crossfire attack requires a lot of effort. Therefore, a number of tasks could be done to further extend our work. Since our model involves configuring many parameters, an evaluation that takes into account more of them could be performed to gain a deep understanding of the parameter interaction on our model. Moreover, experiments could be run on real network topologies such as those derived by the Internet Topology Zoo project [30]. Also, evaluating the inter-domain coordination using realistic client-upstream ISP pairs. In addition, we were using OpenFlow for conducting bandwidth measurements. Clearly, OpenFlow is not designed for this purpose, so another monitoring tool for finer-grained bandwidth measurements could also be applied to our solution (e.g., NetFlow). Furthermore, we assume that the baseline routing in our work is based on Shortest Path First. It would be interesting to research on how our approach could be modified when policy-based routing is enforced in a network domain. Flow rules with different priorities could be installed in an OpenFlow network for this purpose but the routing policy interaction remains an open problem in Software Defined Networking. Since we faced many bugs and errors while trying to generate traffic using Mininet, one could attempt to generate traffic using a different approach and maybe use another emulation platform to assess our solution. It would be also interesting to enhance our solution by integrating other security mechanisms, such as a Hierarchical Heavy Hitters detection approach [31].
Our work mainly focused on devising a solution to defeat the Crossfire attack on a local level. A simple model was also built for extending our defence mechanism from a local to an inter-domain level. Thus, research could be further conducted about how to counter the Crossfire attack on an inter-domain level and a more extensive model could be designed and implemented. In the inter-domain part of our model, we assumed that all the enterprise network's upstream direct providers collaborate to provide a defence solution to the Crossfire attack. It would be a desirable task to investigate countermeasures when some direct ISPs do not collaborate and also when some higher level (e.g., Tier-2) providers collaborate with each other and with lower level (e.g., Tier-3) ISPs to defeat the Crossfire attack. Furthermore, our solution is deployed in a pure SDN environment in our work. It would be a challenge to enable compatibility with legacy equipment for our network application, especially on an inter-domain level (e.g., communication either between enterprise network running SDN and direct ISP using legacy equipment or among collaborating ISPs which some of them run SDN and the rest work on a legacy environment). Since multiple controllers per network domain may be used in large scale networks for scalability and fault-tolerance reasons, it would be also interesting to verify how our approach is extended in this case. One last future task could be the identification of the potential Target Area and Decoy Servers in order to find the rerouting paths that are disjoint with both the current and the potentially subsequent target links. In our work, we assume that we are not aware of the Target Area and the Decoy Servers. Thus, the rerouting based on our approach may potentially lead to a path including a newly calculated target link. In this case, the detection of the Crossfire attack could not be possible directly based on our model and more rerouting "detection-mitigation" rounds would be required to potentially identify portions of the attack sources.

# Chapter 7

# Summary

In this thesis, an online traffic engineering scheme towards both detecting and mitigating the Crossfire attack was designed and developed. SDN and network security characteristics were used to counter this attack since current defence mechanisms are unable to defeat it. A working prototype was built and evaluated on an emulation platform (Mininet) and can be integrated in a production testbed. It was assumed that we work on a pure SDN environment. The performance evaluation results indicated that our detection and mitigation mechanism can detect and mitigate the Crossfire attack. In particular, our scheme is executed fast resulting in successful mitigation of the Crossfire attack since the attack's responsiveness is slow due to its inherent characteristics. The attacker uses traceroutes to launch an easily implemented and low-cost attack at which no sophisticated tools or knowledge are needed. On the other hand, the effectiveness of the attack is limited by the overhead of the Round Trip Time between the bots and the servers in and around a certain target area when attempting to adjust to topology changes. Our scheme causes topology changes to mitigate the attack and can also detect the malicious sources by monitoring the attacker's reaction to the network route shifts generated by our mechanism. Our solution was mainly focused on the case when the Crossfire attack occurs within an enterprise network. Our technique was also briefly designed and implemented to expand on a inter-domain level in order to enhance the mitigation and detection capability of the local domain but still more work can be done. Our proposed solution was based on the fact that the attacker uses traceroutes to enable dynamic topology discovery. This imposes a limitation to our solution when the attacker has for example full topology knowledge due to social engineering, has hacked the BGP tables or the OpenFlow controllers of the enterprise network or has gained access to OSPF routers. However, our scheme conducts mitigation and may be able to conduct detection in those cases. The implementation of the whole solution was a very challenging task and is worth of more future work in terms of parameterization, topology exploration and attack strategies.

# Bibliography


[1] US-CERT. Security Tip (ST04-015): Understanding Denial-of-Service Attacks. *http://www.us-cert.gov/ncas/tips/ST04-015*. Last visit on March 21, 2014.

[2] J. Mirkovic, and P. Reiher. A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. *ACM SIGCOMM Computer Communications Review*, Volume 34, Number 2, April 2004.

[3] J. Markoff, and N. Perloth. Firm Is Accused of Sending Spam, and Fight Jams Internet. *http://www.nytimes.com/2013/03/27/technology/internet/online-dispute-becomes-internet-snarling-attack.html*. Last visit on March 21, 2014.

[4] M. S. Kang, S. B. Lee, and V. D. Gligor. The Crossfire Attack. *In Proceedings of IEEE Symposium on Security and Privacy*, 2013.

[5] A. Studer, and A. Perrig. The Coremelt Attack. *In Proceedings of the 14th European Symposium on Research in Computer Security*, 2009.

[6] P. Ferguson. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. *RFC 2827*, 2000.

[7] A. Yaar, A. Perrig, and D. Song. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. *In Proceedings of the IEEE Security and Privacy Symposium*, 2004.

[8] R. Moskowitz, and P. Nikander. Host Identity Protocol (HIP) Architecture. *RFC 4423*, 2006.

[9] K. Levanti. Routing management in network operations. *Ph.D. dissertation, Carnegie Mellon University*, 2012.

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, March 2008.

[11] Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. *https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf*, ONF White Paper, April 13, 2012.

[12] R. Braga, E. Mota, and A. Passito. Lightweight DDoS flooding attack detection using NOX/OpenFlow. *In Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, Pages 408-415, October 2010.

[13] NOXRepo.org. About NOX. *http://www.noxrepo.org/nox/about-nox/*. Last visit on March 28, 2014.

[14] S. B. Lee, M. S. Kang, and V. D. Gligor. CoDef: Collaborative Defense Against Large-Scale Link-Flooding Attacks. *In Proceedings of ACM CoNEXT*, December 2013.

[15] Mininet Team. Mininet. *http://mininet.org/*. Last visit on April 3, 2014.

[16] Specification. OpenFlow Switch. *Version 1.0.0 (Wire Protocol 0x01)*. December 31, 2009.

[17] N. Feamster. Software Defined Networking. *https://www.coursera.org/course/sdn*. Last visit on April 5, 2014.

[18] NOXRepo.org. About POX. *http://www.noxrepo.org/pox/about-pox/*. Last visit on April 5, 2014.

[19] Open Networking Lab. POX Wiki. *https://openflow.stanford.edu/display/ONL/POX+Wiki*. Last visit on April 5, 2014.

[20] MaxMind Inc. GeoIP. *www.maxmind.com*. Last visit on April 7, 2014.

[21] Philippe Biondi. Scapy. *http://www.secdev.org/projects/scapy/*. Last visit on April 7, 2014.

[22] GNU Operating System. GNU Wget. *https://www.gnu.org/software/wget/*. Last visit on April 7, 2014.

[23] Open Networking Lab. POX Messenger. *https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-messenger*. Last visit on April 8, 2014.

[24] James McCauley. POX Messenger Source Code. *https://github.com/noxrepo/pox/blob/carp/pox/messenger/__init__.py*. Last visit on April 8, 2014.

[25] BGP4.AS. BGP: the Border Gateway Protocol, Advanced Internet Routing Resources. *http://www.bgp4.as/*. Last visit on April 9, 2014.

[26] OpenFlow Team. OpenFlow Tutorial, Required Software Installation Instructions. *http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial#Pre-requisites*. Last visit on April 10, 2014.

[27] Oracle. VirtualBox. *https://www.virtualbox.org/*. Last visit on April 10, 2014.

[28] Open Networking Lab. POX, Selecting a Branch/Version. *https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-SelectingaBranch%2FVersion*. Last visit on April 10, 2014.

[29] Open vSwitch. Open vSwitch, An Open Virtual Switch. *http://openvswitch.org/*. Last visit on April 10, 2014.

[30] S. Knight, H.X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The Internet Topology Zoo. *IEEE Journal on Selected Areas in Communications*, Volume 29, Number 9, Pages 1765-1775, October 2011.

[31] L. Jose, M. Yu, and J. Rexford. Online measurement of large traffic aggregates on commodity switches. *In Proc. Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, March 2011.