ARTEMIS$_{\text{ONOS}}$: SDN-based Real-Time Detection and Automatic Mitigation of BGP Prefix Hijacking

Dimitrios Mavrommatis

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Assistant Prof. *Xenofontas Dimitropoulos*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been supported by the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS).

University of Crete
Computer Science Department

ARTEMIS$_{\text{ONOS}}$: SDN-based Real-Time Detection and Automatic Mitigation of BGP Prefix Hijacking

Thesis submitted by
Dimitrios Mavrommatis
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Dimitrios Mavrommatis

Committee approvals: _____
Xenofontas Dimitropoulos
Assistant Professor, Thesis Supervisor

_____
Maria Papadopouli
Professor, Committee Member

_____
Sotiris Ioannidis
Research Director, Committee Member

Departmental approval: _____
Antonios Argyros
Professor, Director of Graduate Studies

Heraklion, June 2018

ARTEMIS$_{\text{ONOS}}$: SDN-based Real-Time Detection and Automatic Mitigation of BGP Prefix Hijacking

Abstract

Prefix hijacking is a persistent and serious threat for the Internet's routing system, having a technical and financial impact on a global scale. On the one hand, the research community has developed several sophisticated prefix hijacking detection techniques, which nevertheless lack wide adoption. On the other hand, network operators usually follow simple, tested practices, albeit with their own limitations (e.g., slow mitigation speed).

In the current work, we present a Software Defined Networking (SDN) application which is built upon the principles of the prototype ARTEMIS, such as self-monitoring, and utilizes ONOS, a carrier-grade SDN Operating System. The application is called ARTEMIS$_{\text{ONOS}}$; it uses modern, publicly available streaming services to monitor the BGP control plane in real-time, and accurately detects different types of hijacks. Moreover, it reacts automatically with a configurable mitigation countermeasure.

ARTEMIS$_{\text{ONOS}}$ is an official application of ONOS, and leverages several advantages of SDN. In particular, it provides the following features. ARTEMIS$_{\text{ONOS}}$ is developed as a modular application on top of the OSGi framework, containing a monitoring, a detection and a mitigation module. It achieves high availability and scalability through the distributed architecture of the network control plane, and is agnostic to the network infrastructure (BGP speakers, data-plane devices, etc) that it operates on, allowing for easy deployment and reduced operational complexity. Although ARTEMIS$_{\text{ONOS}}$ is an SDN application, it is fully compatible with BGP, and is thus ready to be used in operational environments.

We evaluate our work by implementing a framework that emulates prefix hijacks. We show that ARTEMIS$_{\text{ONOS}}$ detects the hijack and starts the mitigation process within milliseconds. On the contrary, mitigation is achieved in seconds; the time required for BGP to fully converge. Despite ARTEMIS$_{\text{ONOS}}$ being –in principle– a reactive application, in some cases it is faster than the propagation of the actual hijack event, protecting some networks (the ones "close" to the victim) almost proactively.

ARTEMIS$_{\text{ONOS}}$: Εφαρμογή SDN για ανίχνευση και αυτόματη αντιμετώπιση επιθέσεων BGP Prefix Hijacking σε πραγματικό χρόνο

Περίληψη

Το BGP Prefix Hijacking είναι μια συνεχής απειλή για το σύστημα δρομολόγησης του Διαδικτύου, έχοντας σημαντικές τεχνολογικές και οικονομικές επιπτώσεις παγκοσμίως. Η ερευνητική κοινότητα έχει απαντήσει με εξειδικευμένες τεχνικές ανίχνευσης και αντιμετώπισης, οι οποίες δεν έχουν τύχει ευρείας αποδοχής. Αντιθέτως, οι μηχανικοί δικτύων συνήθως ακολουθούν απλές, δοκιμασμένες πρακτικές, που όμως έχουν περιορισμούς (π.χ. ταχύτητα).

Σε αυτή την εργασία, δημιουργούμε μια SDN εφαρμογή που βασίζεται στο πρωτότυπο ARTEMIS και αξιοποιεί το ONOS, ένα δικτυακό λειτουργικό σύστημα τεχνολογικής αιχμής. Η εφαρμογή ονομάζεται ARTEMIS$_{\text{ONOS}}$ και χρησιμοποιεί σύγχρονες, κοινώς διαθέσιμες υπηρεσίες BGP streaming και ανιχνεύει με ακρίβεια διαφορετικούς τύπους hijack. Επιπλέον, αντιδρά αυτόματα με τα κατάλληλα αντίμετρα.

Το ARTEMIS$_{\text{ONOS}}$ είναι μια επίσημη εφαρμογή του ONOS και αξιοποιεί όλα τα οφέλη που προσφέρουν τα δίκτυα SDN. Συγκεκριμένα, το ARTEMIS$_{\text{ONOS}}$ είναι υλοποιημένο ως αρθρωτή εφαρμογή πάνω στο OSGi framework, και χωρίζεται σε ξεχωριστά modules για την ανίχνευση και την επίλυση των επιθέσεων. Επιτυγχάνει υψηλή διαθεσιμότητα και κλιμάκωση μέσω της κατανεμημένης αρχιτεκτονικής του επιπέδου ελέγχου του δικτύου. Επίσης, λειτουργεί ανεξαρτήτως την διαδικτυακής του υποδομής (π.χ. συσκευές δικτύωσης), επιτρέποντας να έχει εύκολη ανάπτυξη αλλά και μειωμένη πολυπλοκότητα. Επιπλέον, αν και η εφαρμογή είναι βασισμένη σε SDN, είναι συμβατή με το BGP, συνεπώς έτοιμη να χρησιμοποιηθεί σε επιχειρησιακό επίπεδο.

Αξιολογούμε την εργασία μας υλοποιώντας μια πλατφόρμα που εξομοιώνει τις επιθέσεις BGP Prefix Hijacking. Το ARTEMIS$_{\text{ONOS}}$ ανιχνεύει την επίθεση και εκκινεί την διαδικασία αντιμετώπισης εντός μερικών χιλιοστών του δευτερολέπτου. Αντιθέτως, η πλήρης αντιμετώπιση της επίθεσης επιτυγχάνεται σε δευτερόλεπτα, το χρόνο που απαιτείται από το BGP για να συγκλίνει. Παρόλο που το ARTEMIS$_{\text{ONOS}}$ είναι μια εφαρμογή που αντιδρά μετά τη εκκίνηση της επίθεσης, η αντιμετώπιση της σε ορισμένες περιπτώσεις είναι ταχύτερη από τη διάδοση της, προστατεύοντας ορισμένα δίκτυα σχεδόν προληπτικά.

Acknowledgements

I would like to take this opportunity to thank the following people for help and support:

- My supervisor Professor Xenofontas Dimitropoulos for letting me be part of his team, INSPIRE group, and also participate as a teaching assistant in his courses that helped me a lot to develop as an individual.
- The members of my dissertation committee, Professors Maria Papadopouli and Sotiris Ioannidis.
- The group members Dr. Vasileios Kotronis and Lefteris Manassakis for the continuous support and contributions on my work but also my other colleagues like Michalis, Alex, Manos and Petros for the helpful advices.
- The Institute of Computer Science (FORTH-ICS) and Telecommunications and Networks Laboratory which provided the means to carry on with my work.
- Last, but definitely not least, I would like to devote this work to all my family for their support and belief during these years.

The project leading to this application has received funding from the EU Research Council Grant Agreement no. 338402

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The Border Gateway Protocol (BGP) [61] governs inter-domain routing on the level of Autonomous Systems (ASes), and enables approx. 60k ASes to exchange routing information and follow inter-domain paths according to their desired policies [8]. Due to its distributed architecture and lack of security and authorization, BGP has proven very hard to debug and protect; e.g., control-plane mechanisms can be exploited by malicious users to "hijack" traffic destined to a legitimate AS and consequently intercept it, blackhole it or otherwise manipulate it. Such attacks can last for multiple hours or even days, sometimes without the affected AS even noticing, especially in cases of Man-in-the-Middle (MitM) incidents.

In this thesis we focus on BGP prefix hijacking which is the act of manipulating BGP via the announcement of prefixes that do not belong to the hijacker. By broadcasting fraudulent prefix announcements, the hijacker poisons the Routing Information Base (RIB) of its peers, which in turn spread the (fake) news all over the Internet. Sometimes this incident may be due to simple misconfigurations (e.g., the infamous "route leaks" [28, 7]). The usual result is a division of the Internet in two AS-level sets: one that routes traffic to the prefix through the hijacker, and the other routing as expected through the legitimate AS. The larger the hijacked ("polluted") set, the greater the impact.

Several methodologies have been proposed for detecting and mitigating prefix hijacking incidents [44, 45, 48, 50]. However, none of them are considered production-ready or easily applicable to the current structure and operational practices of an ISP.

In this work, we build upon the methodology and early encouraging results of legacy ARTEMIS [33], which proposed a systematic approach for detecting and mitigating prefix hijacking in real-time, starting with an automatic prefix de-aggregation counter-measure. The main idea behind

ARTEMIS is to provide a self-operated application that detects and mitigates BGP prefix hijack attacks without the need to share internal network information to other third party companies. The application has three services that follow a modular design pattern. These services can run independently and be extended to support any additional need of the network operator. They are: (i) a monitoring service, that is responsible for receiving control-plane information, (ii) a detection service, that can identify hijacks given the control-plane information and lastly (iii) a mitigation service, that can connect to any BGP network device on the network and mitigate the attack.

In fact, we implement and refine ARTEMIS concepts on top of ONOS [19]; a popular, production-grade network operating system (cf. Section 2.3.5.1) that follows the SDN principles. We chose ONOS because it is a globally-contributed open-source project that offers applications and abstracted interfaces, such as SDN-IP or Reactive-Routing, that were useful and were eventually employed in our implementation. We describe this implementation as a dynamic application in Section 4.

We next evaluate the application using multiple virtualized network emulation environments, generated with Mininet [15] and GNS3 [12] (cf. Section 5) automation scripts that we implemented. We show that our application is highly scalable, easy-to-deploy (and configure), and provides all the benefits of the systematic approach of legacy ARTEMIS while remaining portable and adjustable to the requirements of the ISP operator.

1.2 Contributions

We implemented an open-source SDN application, called $\text{ARTEMIS}_{\text{ONOS}}$; this application detects and mitigates BGP hijacks inside an SDN-enabled AS in a matter of milliseconds. This application was accepted by the ONOS technical steering team and is now bundled and deployed with ONOS as a default application. Also, this work was presented during the ONOS Build 2017 conference in Seoul, Korea. We also created a wiki page that explains this application [3] and a corresponding demo [2].

Our second contribution is an open-source ExaBGP control-plane monitor which is easily deployed with Docker. It is a light-weight BGP speaker that can connect with iBGP to any other BGP router within the operator's network and provide a Northbound Interface that will forward the control plane information through the data plane to a set of subscribers. This control-plane monitor was used by $\text{ARTEMIS}_{\text{ONOS}}$ in our evaluation framework to learn about observed BGP updates in an emulated topology.

Our third contribution is an automated AS topology generator for GNS3. In this generator you can specify if an AS is SDN-enabled or not; it will then create the topology based on the links and AS relationships the user is

providing. It can be used by other researchers to create topologies quickly, when Mininet is not a viable approach, and evaluate their work.

Lastly, we were contributors on the open-source ONOS project. We patched bugs and extended features to applications that were eventually used by ARTEMIS$_{\text{ONOS}}$. These applications were responsible for enabling Software Define Networks to talk BGP and making them co-exist with today's Internet.

Chapter 2

Background

In this chapter we focus on the background of this thesis. We first give a brief explanation of Internet Routing with emphasis on the Border Gateway Protocol (BGP), and how we monitor it through the help of control-plane monitors. Then, we introduce the reader to Software-Defined Networks and Network Operating Systems such as ONOS, and end with some information about tools we used for our evaluation framework (ExaBGP, Quagga, GNS3, Mininet and Docker).

2.1 Internet Routing

The Internet today is sometimes considered as a single network, but in reality it is divided in multiple networks called Autonomous Systems (ASes). To identify the Autonomous Systems, the Internet Assigned Numbers Authority (IANA) allocates unique AS Numbers (ASN) to Regional Internet Registries (RIRs) for each AS (e.g. Google [AS 15169], Comcast [AS 7922]). Each AS has internal and external routers where they use Internal Gateway Protocols (IGP) to route intra-domain traffic and External Gateway Protocols (EGP) to route inter-domain traffic.

Communication between hosts is achieved by sending packets from one router to another, repeatedly until they reach their final destination. Each AS has hosts that are identified by their unique IP address in order to communicate with other hosts. An IP address is a numerical label that is assigned to network devices that communicate in the Internet through the IP protocol, and is used to identify the device and address the location. An IP address might be PUBLIC where it is reachable from the whole Internet, or PRIVATE where it is only routable inside a private network.

These individual IP addresses are grouped together into prefixes which are originated, or owned, by the AS. Prefixes are a logical subdivision of an IP network and can be expressed in Classless Inter-Domain Routing (CIDR)

notation written as the first address of a network, followed by a slash character (/), and ending with the bit-length of the prefix. For example in Figure 2.1, if the network prefix 192.0.2.0/24 is originating from AS numbered 65000 or else ASN 65000, then the host will have an IP addresses inside that prefix (e.g. 192.0.2.100) and Border Gateway Protocol will propagate to other connected ASes the information that this prefix originates from ASN 65000.

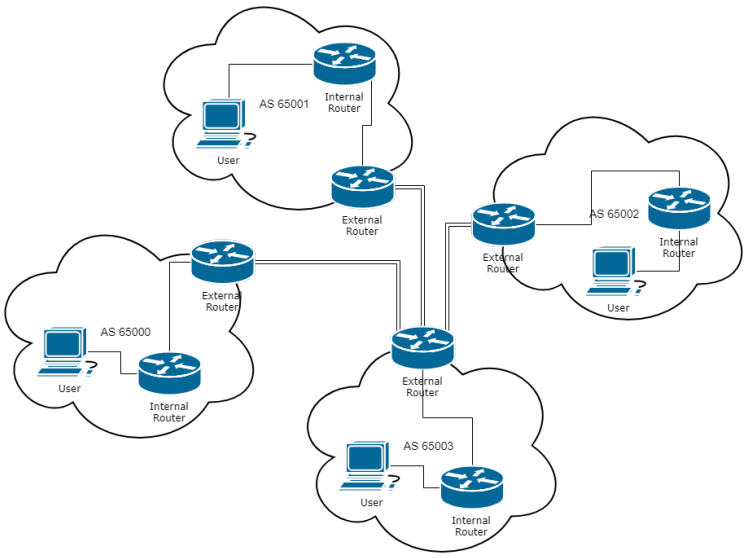

Figure 2.1: Internet Topology with 4 ASes

2.1.1 BGP

In this thesis we explore a security vulnerability that exists in the Border Gateway Protocol. For this reason we will provide an extensive description of BGP, how it works and why it has this security issue.

The Border Gateway Protocol version 4 [61] (BGPv4) is a standardized External Gateway Protocol designed to exchange routing information between ASes in order to handle intra-domain and inter-domain traffic.

2.1.1.1 BGP Selection Algorithm

BGP is classified as a path vector protocol and makes routing decisions based on the BGP Best Path Selection Algorithm. A list of best path selection criteria is listed below, sorted in order of preference:

1. **Weight**
2. **Local Preference**
3. **Network or Aggregate**

4. **Shortest AS PATH**
5. **Lowest origin type**
6. **Lowest multi-exit discriminator (MED)**
7. **eBGP over iBGP**
8. **Lowest IGP metric**
9. **Multiple paths**
10. **External paths**
11. **Lowest router ID**
12. **Minimum cluster list**
13. **Lowest neighbor address**

From these criteria we will focus on the **shortest AS PATH** criterion. When the BGP routing mechanism cannot decide on adopting a route based on weight, local preference and the route does not originate from a local or aggregated network, then BGP decides which route is best based on the lowest number of ASes it needs to go through. For example, if a BGP speaker receives multiple routes for an IP prefix; (a) AS1,AS2,AS3 and (b) AS1,AS4,AS2,AS3 and they do not have any other higher criterion to discriminate, then route (a) will always be chosen for having only 3 ASes in-between opposed to the 4 ASes of route (b).

It is important here to note that routers always use the longest prefix match to select an entry from the routing table. Considering that a forwarding table has the entries (i) 192.168.20.16/28 and (ii) 192.168.0.0/16, then if address 192.168.20.19 needs to be looked up because it matches with both entries, routers will prefer the most specific /28 prefix always, irrespective of the other tie-braking attributes.

2.1.1.2 AS Relationships

AS relationships can be divided in two major categories, (i) customer-provider relationship and (ii) peer-to-peer relationship. Forwarding traffic through an AS has a price that varies based on this relationship, where in case (i) the customer pays the provider to send and receive traffic and in case (ii) both exchange each other's traffic for free (typically this is called "settlement-free" peering). In general, the route with the lowest cost or the highest pay-off is always preferred.

2.1.1.3 BGP Message Types

BGPv4 [61] as a protocol, runs its functions through the exchange of messages. It uses four types of messages:

1. **OPEN** - Messages used to initiate a session between Routers. An OPEN message is always send when the TCP session is established and includes the following fields:

Version	Autonomous System	Hold-Time	BGP Identifier	Optional Parameters Length	Optional Parameters

Table 2.1: BGP message OPEN fields

   - **Version** - Specifies BGP version, default 4.
   - **Autonomous System** - Provides the AS number of the sender. If the AS number of the receiver and sender is the same then the session is iBGP, else is eBGP.
   - **Hold-Time** - This is a timeout timer that indicates the maximum amount of time that can elapse before the transmitter is assumed to be offline. If two neighbors have different timers the minimum timer is accepted as the hold time.
   - **BGP Identifier** - Provides the BGP Identifier of the sender which is an IP address.
   - **Optional Parameters Length** - Indicates the length of the next field, Optional Parameters or zero if not present.
   - **Optional Parameters** - A list of optional parameters, like authentication or support for other protocols.

2. **KEEPALIVE** - A heartbeat message that keeps the session alive. It is usually equal to 1/3 of the specified hold time.

3. **UPDATE** - Messages used to advertise and/or withdraw routes. The fields of these messages in more detail:

Unfeasible Routes Length	Withdrawn Routes	Total Path Attribute Length	Path Attributes	Network Layer Reachability Information

Table 2.2: BGP message UPDATE fields

- **Unfeasible Routes Length** - Indicates the length of the withdrawn routes field or zero if is not present.
- **Withdrawn Routes** - A list of IP address prefixes that became unreachable and will be withdrawn.
- **Total Path Attribute Length** - Indicates the length of the path attributes field or zero if is not present.
- **Path Attributes** - Attributes of the advertised path. Some possible attributes for a path are; (i) Origin: attribute that describes the origin of the advertisement, (ii) AS PATH: a list of AS numbers that describes the sequence path to the origin AS that announces the prefix, (iii) Next Hop: an IP address that specifies the next IP address to be used as a next hop for packets destined to advertised prefix, and multiple others like Mult Exit Disc, Local Preference, Atomic Aggregate, etc.
- **Network Layer Reachability Information** - A list of IP address prefixes for the advertised routes.

4. **NOTIFICATION** - A notification message that is generated when an error is detected. A BGP notification message consists of:

Error Code	Error Subcode	Error Data

Table 2.3: BGP message Notification fields

- **Error Code** - A code that corresponds to the type of the error that occurred. Sample errors can be a message header error, an `open` message error, etc.
- **Error Subcode** - A more specific error code that describes better the reported error.
- **Error Data** - Additional data that help diagnose the reason of the notification message.

2.1.1.4 Prefix Hijacking

To configure BGP the network operator provides a configuration, which is usually manually generated, where it defines the BGP neighbors, originated AS prefixes and other routing policies. Unfortunately, manual configuration is prone to human errors, which can cause problems to BGP operation. Also, BGP routers accept any BGP message without any origin authentication by default. This, in a trusted environment, allows for fully decentralized routing. In a real-world scenario though we cannot trust all BGP peers due to the presence of potentially vulnerable security holes. One of these security issues that we are going to focus on is BGP prefix hijacking.

BGP prefix hijacking is the illegitimate takeover of IP prefixes by corrupting Internet routing tables maintained using the Border Gateway Protocol (BGP). As mentioned before, the route selection is based on the shortest AS PATH and the longest prefix is always preferred (assuming no differentiation of routes based on policies). If the routing table has a routing rule for a /23 prefix, a hijack can be achieved by announcing two /24 sub-prefixes of the previous prefix, which will cause all the traffic to go to the hijacker due to longest prefix match rule. On the other hand, if hijacker decides to announce the same /24 prefix, the traffic will be divided in two; based on the shortest AS PATH on each router the traffic will either go to the original AS or to the hijacker AS. These attacks are viable because there is no origin authentication and all BGP updates are accepted as legitimate causing traffic to be redirected to the malicious AS as depicted in Figure.2.2.

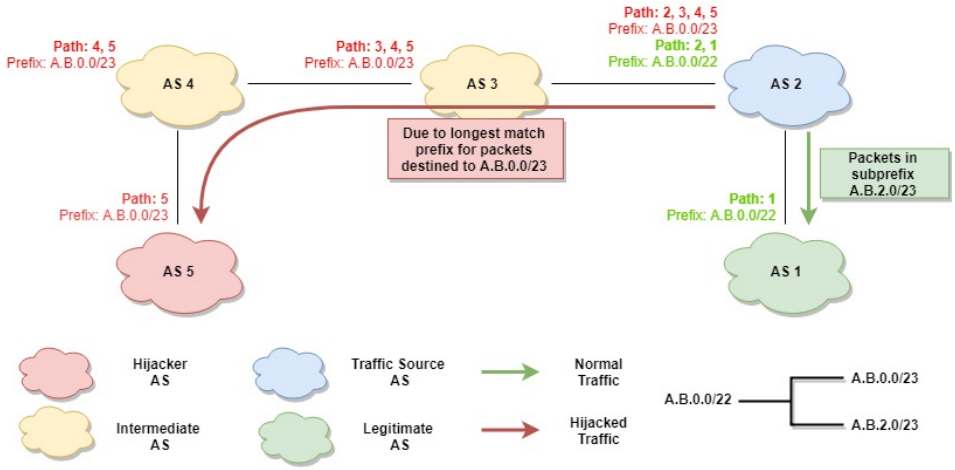

Figure 2.2: Subprefix hijack

This type of attack, as mentioned before, can be accidentally caused from misconfiguration or for profit by malicious ASes. For example, a prefix hijack attack can redirect and steal sensitive traffic [13] or even black-hole the traffic to cause Denial of Service (DoS).

Prefix hijack attacks can be divided to several different types but we are going to focus on (a) Type-0 hijacks where the hijacker pretends to be the legal origin of the prefix and announces either the same prefix or subprefixes and (b) Type-1 hijacks where a false AS PATH is constructed with the hijacker inside the AS PATH in order to execute a Man-in-the-Middle attack to the traffic that will eventually pass through it. Examples of hijacks:

- An AS announces that it originates a prefix that it does not actually originate (Type-0).
- An AS announces a more specific prefix than what may be announced

by the true originating AS (Type-0).

- An AS announces that it can route traffic to the hijacked AS through a shorter route (Type-1).

There are multiple solutions proposed to solve these issues (e.g. Resource Public Key Infrastructure) that we are going to analyze more in chapter 3.

2.2 Control Plane Monitoring

In this thesis, control-plane monitoring plays a major role, as it is one of the three components used to create ARTEMIS$_{\text{ONOS}}$. In routing, the control-plane carries the control information with control packets that originate from or are destined for a router. Multiple protocols that are used for control plane management exist. Unfortunately, there is no easy way to monitor this type of traffic when having multiple devices. For this reason, people tried to find a way to monitor the already existing protocols without changing them, by introducing devices that trick the network to send the control-plane traffic to them through data-plane to be easier to monitor them.

Specifically, for BGPv4 one can create an emulated BGP router that connects with a BGP speaker and is configured as an internal neighbor that speaks iBGP. This way, any BGP message that is received on the BGP speaker will be forwarded to the emulated BGP router. By doing this for all BGP speakers, we have a centralized BGP monitor that we can access, retrieving all the logged BGP traffic, or even stream it in real time through the data-plane. Many different implementations of these control plane monitors exists with some of them being:

BGPMON

BGP monitoring system [4] is a control plane monitor implemented by Colorado State University. It provides live BGP feeds from several BGP routers hosted in RouteViews [24] sites, and other peers worldwide. Unfortunately, it was not available at the time of writing this thesis and was not used.

RIPE RIS

RIPE's Routing Information System (RIS) [23] has 21 route collectors (RCs) spread globally. These route collectors collect BGP updates from more than 300 peering ASes. RIPE RIS provides an API that lets users to subscribe easily to BGP live feeds for specific prefixes. We ended up using RIPE RIS control-plane monitors as our default monitors because of their responsiveness and easy to use interface.

Looking-Glass Servers

Looking-Glass servers are a real-time source of routing and BGP related information for network administrators. Looking Glass servers are deployed in different parts of the Internet and allow on-line checking of prefixes, collected from the BGP speaking routers. These Looking-Glass servers do not have a standardized API which makes it very difficult to implement a universal interface to parse them. Periscope [14] provided an API for a plethora of Looking-Glass servers but because of the big delay and the query limitation that they have it was not optimal for this thesis.

Custom Monitor - ExaBGP

This is a control-plane monitor that we implemented and used to evaluate our work because we needed a control-plane monitor that works in an emulated environment. ExaBGP [11] is an open-source BGP router implemented in Python, which provides a convenient way of adding new script-like control functions on top of it. We transformed ExaBGP to a control plane monitor by parsing BGP messages into `json` format and implementing a `socket.io` service that lets clients request prefix-specific BGP traffic.

2.3 Software Defined Networking

In this section we will give a brief description of Software Defined Networking because ARTEMIS$_{\text{ONOS}}$ is build to work on a SDN environment. First, we focus on the current state of Ethernet switching in subsection 2.3.1 and then introduce the concepts of Software Defined Networking (SDN) in subsection 2.3.2. Then we focus on explaining; (i) OpenFlow [71] as the communication protocol that enables SDN concepts in Ethernet networks in subsection 2.3.3, (ii) Open vSwitch [20] as an open-source virtual switch implementation that speaks OpenFlow and is used as the data-plane OpenFlow-enabled switch to evaluate our methodology with in subsection 2.3.4 and lastly, (iii) the purpose of the SDN controller and more specifically Open Networking Operating System (ONOS) and its services that we used in this thesis (Intent Framework, SDN-IP and Reactive-Routing) in subsection 2.3.5.

2.3.1 Ethernet Switching

Today's Ethernet switches are often quite static, slow to change and dedicated to single services. They follow a packet-switched model having information only about their neighbors. The main purpose of the Ethernet switches is to interconnect nodes, while exchanging Internet Protocol [56] (IP) data-packets. When an Ethernet switch receives a data-packet it goes

through a method that decides what action should be applied to this packet. The actions are decided based on the Forwarding Table of the device, which contains a list of MAC addresses that map incoming traffic and destination address to an outgoing port, but in case that there is no match found then the packet is flooded over all ports. In more detail the main functions of an Ethernet switch are:

- **Populate Forwarding Table** - The most basic function of the Ethernet switch is to learn the MAC addresses that corresponds to each port. Whenever the Ethernet switch receives a data-packet, an Ethernet frame exists which contains both the source and the destination addresses. When the source address is not present in the Forwarding Table, it is added to the corresponding port. When the destination address is not present, the Ethernet switch generates an ARP [55] request on all ports except the incoming port asking about the destination address. If the destination address is connected to the Ethernet switch, it will receive an ARP reply having the answer it needs to update the Forwarding Table and then forward the data-packet.

- **Forward data-packets** - The main function for an Ethernet switch is to forward traffic to the correct outgoing port. As explained before, the data-packet is forwarded if it has a positive match in the Forwarding Table. This tells us that an Ethernet switch has only visibility of its neighbors and cannot have a global view to further optimize traffic flow selections.

- **Spanning Tree Protocol** - When an entry is missing from the Forwarding Table the traffic is flooded over the network until it reaches the destination. This flooding technique can cause routing loops in the network and in order to prevent them switches employ the Spanning Tree Protocol [34] (STP), standardized in IEEE 802.1D.

- **Data-plane Separation** - In a computer network we usually have multiple groups and users. For this reason, data plane traffic should be easily separated or partitioned between these different groups. Virtual Local Area Network [32] (VLAN) is used to separate data traffic between different applications. Each Ethernet switch can have multiple virtual networks that follow different forwarding policies and can be indicated with a VLAN identifier that comes with the data-packet. In this manner, an Ethernet switch that supports VLANs can be used for multiple groups, removing the need of having higher complexity networks with additional switches.

Besides the functions described above, we notice a trend of managing the flows of the data-packets over the network. For this reason other protocols, such as Simple Network Monitor Protocol [63] (SNMP), are used

for management purposes. On manageability level, configuring each switch independently is unwanted and inflexible because small errors can cause big network failures.

2.3.2 Network Abstraction with SDN and Applications

The SDN philosophy evolves around having different switch configurations based on the network services that are run and the QoS needs that need to be met. To embrace this philosophy, a centralized entity must perform the configuration changes on all of the switches through a management plane and have an abstracted interface that translates the network state based on the network services. c With SDN one can create a network that handles many services in a dynamic fashion, allowing to consolidate multiple services into one common infrastructure. A comparison is depicted in Figure 2.3. Software-Defined Networking (SDN) is an open vendor-neutral network architecture approach that enables the network to be centrally controlled by decoupling the control plane from the data plane and offloading all control functions to a logically centralized entity, the SDN Controller [72].

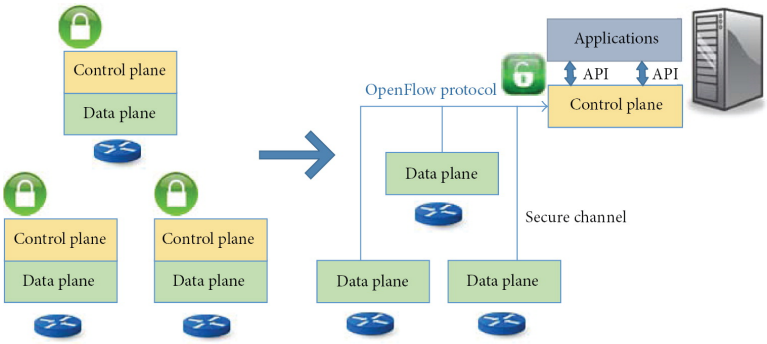

Figure 2.3: Comparison between conventional networks and SDN [72]

The SDN concept is divided in three layers. A short description of the planes is given below in relation to Figure 2.4:

- **Infrastructure Layer** - This is the bottom layer, contained in the data plane with the network forwarding equipment. Unlike todays architecture, that forwarding architecture relies on a new layer, the control layer where the SDN Controller resides. The network forwarding equipment are simple packet forwarding devices that are controlled through the Control Layer and have no innate decision-making capabilities.

- **Control Layer** - Middle layer responsible for configuring the infrastructure layer; it does that by receiving service requests from the top

layer, i.e., the application layer. It involves the controller who is in charge of configuring the forwarding plane based on the network services needs. The control layer and infrastructure layer communicate with each other, usually with OpenFlow, although other protocols exist. OpenFlow will be explained further in subsection 2.3.3.

- **Application Layer** - The top layer, i.e., the application layer, is where management, cloud and/or business applications place their demands on the network through the control layer, through an API the controller provides. The advantage over legacy networks is the ability to dynamically allocate resources and handle application needs.

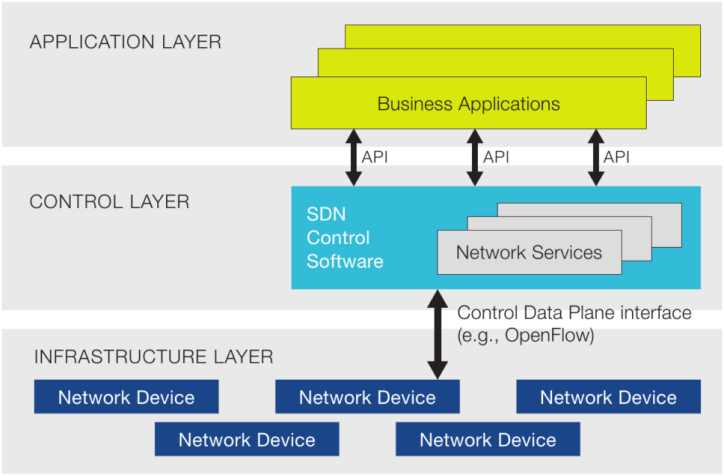

Figure 2.4: SDN Architecture [25]

By centralizing the control logic to the controller, not only is the management of the switches simplified but also the need of reconfiguring the switches manually and locally by hand is removed. Also, it offers the possibility of optimizing traffic, increasing robustness and enhancing security by having a global view of the network. Where Ethernet switches usually act on a local level and have information only for its neighbors, a central control logic acts on a global level. Besides that, having the option to implement all the existing routing or security protocols on a central control logic removes the need of having expensive middle boxes, such as routers, firewalls and load balancers. All these advantages come at a cost; centralizing all the decision-making to a single unit might cause an unwanted additional delay in packet forwarding and in an extreme scenario cause a Denial of Service on itself. Multiple controller implementations fixed this issue by introducing the idea of clusters of controllers, opposed to the solution of only one controller, to distribute the work, provide high-availability, better performance,

fault-tolerance and remove the potential overhead due to overloading.

The authors of [54] discuss about resource allocation and policy enforcement concerning security on enterprise networks, where SDN can control resources based on the network requirements without the need for specific hardware. As mentioned before, SDN can substitute the functionality of different middle boxes, making them obsolete. The centralized and unified control authority yields considerable improvements for enterprise networks enabling them to control and manage the network. Now, policies and configuration do not need to be applied locally, but are instead assigned globally through the controller.

Another application discussed in [54] focused on energy consumption in data centers where additional physical hardware are needed for the provided network services. By moving from using additional physical hardware to implementing the network services on the SDN network, the operator achieves much lower energy consumption and lower operating costs for the data centers. It is important to mention here that there have been multiple projects around SDN'izing data centers, with one of them being CORD [10], which combines NFV and SDN to provide network operators a platform to manage their Central Offices using declarative modeling languages for agile, real-time configuration of new customer services.

The SDN concept is expandable also to wireless access networks. For WLAN it can provide the possibility of seamlessly roaming over multiple wireless infrastructures without the user noticing. Besides WLAN, there have been works like xRAN [27] which aims to introduce SDN into the mobile infrastructure and standardize an open alternative to the traditionally closed, hardware-based Radio Access Network architecture.

Home and small business networks can use SDN as well as explained in [54]. Logging the network activity and using additional security modules is not an easy task and that is where SDN shines. Modules with user friendly interfaces can be installed on the application plane and logging becomes much easier because of having everything going through the central control logic.

The SDN philosophy is not here to create new protocols and re-define standards, but to improve the networks by providing a framework that can co-exist with existing networks. Unfortunately, SDN are not yet widely accepted and deployed due to the complexity of SDN'izing existing networks.

2.3.3 OpenFlow

OpenFlow [71] is an open protocol, created by Open Networking Foundation, and is used to configure forwarding devices inside the infrastructure layer by an SDN Controller. OpenFlow offers a plethora of capabilities making it ideal to researchers wanting to experiment with novel ideas and innovations.

A wide collection of open-source software is available in the form of OpenFlow controllers (like ONOS which will be discussed in subsection 2.3.5.1), as well as physical and virtual switch implementations (Open vSwitch will be discussed further in subsection 2.3.4.

The control plane and forwarding plane communicate in three different ways [71]:

1. **Controller-to-Switch** - These messages are sent by the controller for initialization or configuration purposes. There are seven different type of messages: (1) Handshake, (2) Switch Configuration, (3) Modify State, (4) Queue Configuration, (5) Read State, (6) Send Packet and (7) Barrier.

2. **Asynchronous** - usually intended to update the controller w.r.t. network events or changes of state. There are four type of events that can trigger an asynchronous message from the switch to the controller; (1) Flow removal, (2) Port status, (3) Packet-In message and (4) Error messages.

3. **Symmetric** - Can be sent from both entities without solicitation and can be Hello, Echo Request/Reply, and Vendor messages.

In a simple SDN network scenario the forwarding devices start with some empty routing tables, called Flow Tables. Each row of a Flow Table is a Flow Entry. These tables can be populated by flow entries either proactively or reactively. Whenever a switch receives a packet, a meta data is created containing an Action List, Action Set or both. When a packet matches a flow rule inside a flow table, an action is added to the list or set and execution moves to the next Flow Table. Example of Actions are “forward the packet to port X”, “drop the packet”, “go to Group Table A” or even “modify the headers”. In case a packet does not match any entry that exists inside the Flow Table, the switch will send an asynchronous message to the controller to ask what to do with this packet (Figure 2.5).

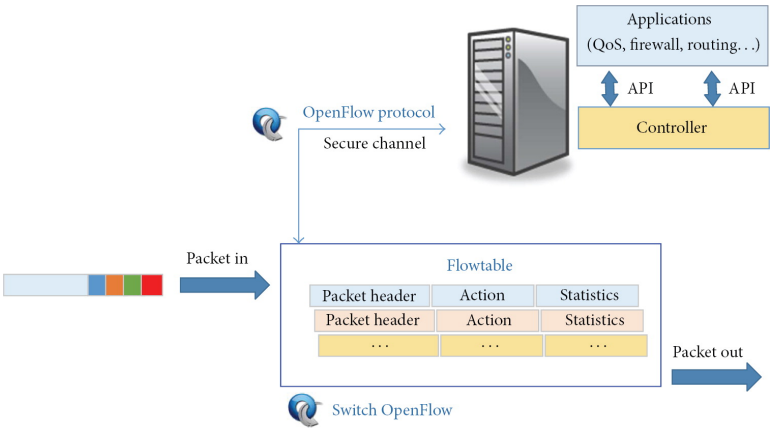

Figure 2.5: OpenFlow architecture [72]

On the other hand, the SDN Controller, based on the current active policies and services responds back to the switch with a message that will add a new entry to the Flow Table of the device so that the packet can match and processed. Flow Tables contain Flow Entries with six parameters [71]:

- **Match Fields** - The criteria which the packets match against. Criteria include parameters from data-packet headers and meta data from previous tables. Some of the criteria which are used to match incoming packets are depicted in Table 2.4.

Match Field	Layer	Description
Ingress Port	Physical	Incoming ports
IP	Network	IPv4/IPv6 source and destination addresses
MPLS	Network	MPLS label

Table 2.4: Flow Entry Match Fields

- **Priority** - If a packet matches multiple Flow Entries, the one with the highest priority is selected.
- **Counters** - Whenever a packet matches this flow Entry the counter increments. This way, the controller can keep track of the traffic going through to determine network policies or for network monitoring.
- **Instructions** - If the packet matches, instructions are added to add Actions to the List or Set or change the Pipeline processing. Some actions can be seen in Table 2.5.
- **Timeouts** - A hard timeout after which this flow will be removed and an idle timeout that removes this flow if it does not get any packet match within that interval.

Action	Associated Data	Description
Push VLAN header	Ethertype	Push a new VLAN header onto the packet.
Push MPLS header	Ethertype	Push a new MPLS header onto the packet.
Output	-	If not group action, then forward the packet to specified port.
Drop	-	Drop all packets that match this flow.

Table 2.5: Flow Entry Actions

- **Cookie** - Random cookie chosen by the controller to filter flow statistics, flow modification and flow deletion.

2.3.4 Open vSwitch

The Open Virtual Switch [20] (OVS) is a virtualized OpenFlow switch which was not specifically designed to enable the SDN philosophy, but is widely used to test OpenFlow implementations and benefit from flexible SDN configurations. OVS can be configured as a normal Layer-2 switch or a Layer-2 switch controlled by a local OpenFlow controller. Configuration can be supplied by the OpenFlow controller; alternatively, Flow Rules can be supplied manually by hand. Open vSwitch is composed of three different modules, two user-space modules and one kernel-space:

1. **ovsdb-server** - A database server which keeps the configuration persistent through the system.

2. **ovs-vswitchd** - The core component of the device which establishes the communication with the controller.

3. **openvswitch_mod.ko** - A kernel module that manages forwarding and tunneling.

Rules associated with each flow entry either contain actions or modify the processing of the pipeline (jumping from one flow table to another, in sequence). When the processing pipeline does not specify any next table, the packet is usually modified and forwarded, as shown in Figure 2.6.

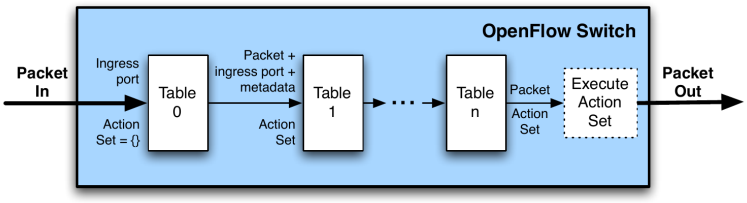

Figure 2.6: Forwarding plane of an OpenFlow Switch [71]

2.3.5 The SDN Controller

As mentioned before, the SDN controller is a centralized, software-based entity that controls the network devices that exist in the infrastructure layer. It takes advantage of the global view it has over the network for running network services and applications. As it is shown if Figure 2.7, the SDN controller is located in the Control Plane which sits between the Data Plane and Application Plane. However, in order to communicate with the controller, proper interfaces must be defined on both sides. Between the Application and Control plane the controller usually employs a Northbound Interface (NBI), while between the Data and Control plane another interface, called Southbound Interface (SBI), is used. Both of these interfaces can be designed to use any available communication protocol, for example on the SBI side it can be OpenFlow and on the NBI side it can be a REST framework. For this thesis, we decided to use the Open Networking Operating System as our SDN controller and we explain further why this was our choice.

2.3.5.1 Open Networking Operating System

The Open Network Operating System [19, 30] (ONOS) is a Software- Defined Networking (SDN) OS, built for service providers, offering scalability, high availability, and performance. Following the SDN mantra, ONOS utilizes high-level abstractions to make it easy for developers to create applications and services over its distributed core. It is based on a well-defined architecture and has quickly matured to be feature-rich over the years (interfacing with classic IP routing is one of these features). Moreover, multiple real-world deployments [16] exist, validating ONOS as a production-ready, high-quality platform, suitable for demanding network environments.

Figure 2.7 shows the architecture of ONOS, where we can distinguish the "distributed" nature of the controller. ONOS is a collection of different services which can be split in three different categories; northbound applications, core applications and southbound protocols. Let us look each category in detail:

- **Northbound Applications** - ONOS is an open-source controller and supports various application categories such as control, configuration

and management applications. Software contributors globally participate in ONOS development; anyone can propose a new application that can be packaged with ONOS after it passes through a Technical Steering Team meeting. Among the applications that are published, some of them are Reactive-Routing, SDN-IP and ARTEMIS$_{\text{ONOS}}$ (which we analyze and evaluate in the current thesis).

ONOS applications work based on information present at the core-layer - via sending and receiving command requests and responses, and event-handling. The core-layer exposes a service interface for each application, which can be used by applications to exchange information between them, such as flow rules, application state, intents and also to add listeners to specific events.

- **Distributed Core** - ONOS's distributed core is built to offer scalability, high availability and performance. To achieve this, ONOS includes (i) a gossip-based protocol and a RAFT implementation to synchronize multiple instances of the controller, (ii) a state of the process's logical clock to have partial ordering, (iii) usage of distributed queues and, (iv) Atomix for cluster membership management.

- **Southbound Protocols** - ONOS supports multiple southbound protocols like OpenFlow, NETCONF, OVSDB, etc. It uses the concept of providers to hide the complexity of each protocol on the controller's side. Anybody can develop a protocol and register it to the core. Following the registration, the communication between devices and providers is done by either (a) event notifications or (b) issuing commands from the core itself.

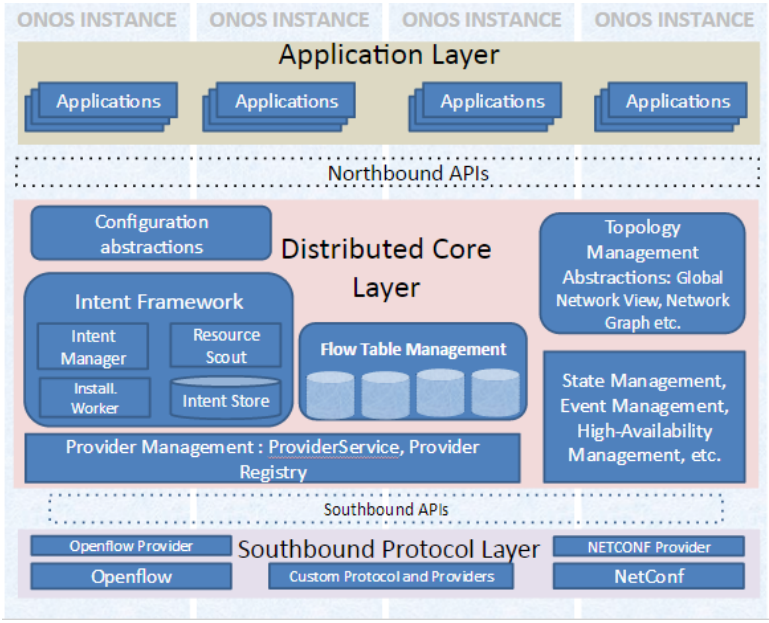

Figure 2.7: ONOS Architecture [26]

Next, we will give brief explanations of all the applications and features that ONOS is packaged with and were used in ARTEMIS$_{\text{ONOS}}$.

ONOS Clustering

ONOS, unlike multiple other controllers, follows a distributed (by design) architecture and thus, it can be deployed as a collection of controllers that coordinate with each other and provide fault-tolerance, data-integrity and better load management. The Cluster Coordination of ONOS is achieved by using a consensus algorithm called RAFT.

SDN-IP Application

Connection between ASes (Autonomous Systems) in the Internet today is universally done via the Border Gateway Protocol version 4 [61] (BGPv4), as explained previously in Section 2.1. Since SDN does not talk to BGP by default, an ONOS application called SDN-IP [51] was introduced for an SDN AS to communicate with other legacy ASes via BGP.

SDN-IP [51] enables an SDN controller to speak BGP by giving the controller the ability to talk and understand iBGP, enabling communication with a BGP speaker. By adding the ONOS controller as an iBGP peer to the BGP speaker's configuration, the speaker is going to forward all the control plane messages that it receives to the SDN-IP application, if so configured. Since the external ASes are not directly connected to the BGP speakers but to OpenFlow switches, the SDN-IP application preemptively installs

flow rules to route the BGP traffic to the BGP speakers. In Figure 2.8 we see that the external networks are each connected to an OpenFlow switch that forwards the traffic to the (closest) BGP speaker, which afterwards communicates with the SDN-IP application.

The SDN-IP application keeps all the BGP routes information to a local RIB (Route Information Base) and installs the needed flows and intents so it can route inter-domain traffic but also intra-domain traffic through another application called Reactive-Routing.

External ASes see the SDN AS as a simple BGP speaker but in reality the SDN network may contain multiple BGP speakers, and several SDN controllers or else a cluster. This architecture provides high availability and scalability; where the BGP speakers are decentralized.

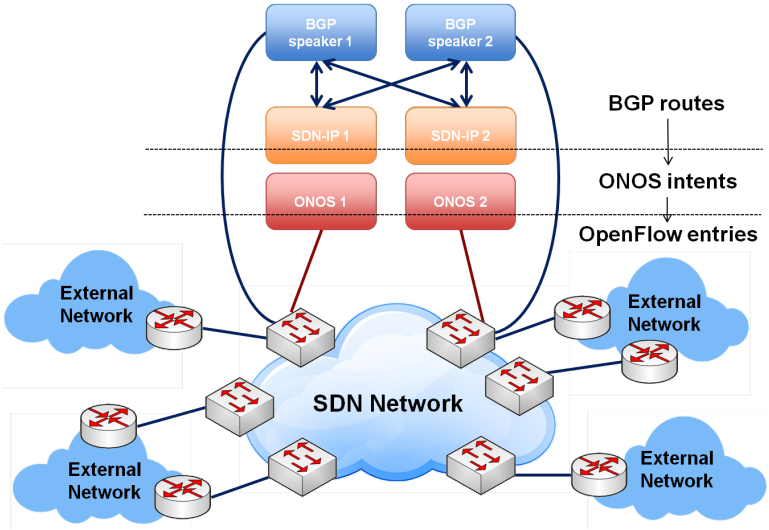

Figure 2.8: SDN-IP Architecture

Reactive-Routing

Reactive Routing is a complimentary application to SDN-IP. It is used to install reactively all the rules needed to route traffic from within the network to the Internet and the other way around when a new BGP routing entry is added on the SDN-IP routing table.

Also, reactive-routing is responsible for routing traffic between hosts that both reside in the SDN network.

Intent Framework

The Intent Framework [18], implemented in ONOS, allows applications to specify their network control desires in form of a policy. These policies follow some specifications which are eventually translated, into flow rules in

the data plane. These actions may result in changes in the infrastructure layer, such as setting up new tunnel links, installing flow rules on a switch, etc.

Some of the important intent types are:

- **Host Intent** - creates bi-directional connectivity between hosts.
- **Tunnel Intent** - creates tunnels of any type (MPLS, GRE, etc) between two connect-points, which can either be a host or a switching device.
- **Point Intent** - Installs forwarding flows that connects two points (point-to-point).
- **Multi-to-Single Intent** - Maps and installs flows for multiple sources to a single connection-point. For example a /24 prefix destined to a specific port of an SDN switch.
- **Single-to-Multi Intent** - Single source, multiple destination connectivity intent.

2.4 Quagga

The evaluation of our tool is in a virtual environment, so Quagga was selected as the best solution, providing the best emulated normal router functionality as a BGP speaker.

Quagga [22] is a network routing software suite that implements multiple routing protocols like OSPF, RIP and BGP, built on Unix platforms. There is a zebra daemon that through an API communicates with Quagga to form the required architecture, but also to run other daemons for the routing protocols that are enabled by the user. Quagga offers a `vty` shell, which can be used to configure each protocol daemon that is running. The CLI configuration is similar to a traditional router.

2.5 Network Emulation

Network emulation is the act of replicating every aspect of the device's behavior in test environments. Emulation enables us to have complex topology testing without the need to wire up a physical network. Moreover, it can support multiple concurrent developers to work independently on the same topology.

With network emulation we can setup different network topologies to evaluate our tool's performance by creating a Internet graph with multiple ASes, and setting up the routing protocols/policies for each of them in order to emulate a BGP hijack event.

The tools that we used to emulate a network topology and run ARTEMIS$_{\text{ONOS}}$ were GNS3, Docker and Mininet. To give the reader the needed background to understand the use of these tools, we

2.5.1 Graphical Network Simulator 3

Graphical Network Simulator 3 [12] (GNS3) is a network emulator that is used by many big companies and allows the combination of virtual and real devices. It is open-source with a graphic front-end and can co-exist with other emulation programs, like Qemu, VMware, VirtualBox and Docker.

GNS3 offers a wide range of network devices and tools for emulating network devices, simulating different scenarios. Something important to note is that GNS3 is emulating and not simulating; meaning that the devices act the same as if it was a physical device. For this thesis we build the wanted topologies in GNS3 using Docker containers and evaluate our tool on different scenarios. To build these topologies we used the Python API that is provided by GNS3, in order to automate the topology generation and configuration of all the devices.

2.5.2 Docker

Container virtualization (known as containerization) uses Linux's kernel functionality to isolate processes and their resources from each other. These isolated environments are called containers. The difference with classic (e.g., hypervisor-based) virtualization is that it shares the same OS kernel by combining kernel features, such as Kernel Namespaces and Control Groups, so there is no need of a hypervisor. In Figure 2.9 the hypervisor virtualization is compared to containerization, and we give a short description of their biggest differences; (i) a container has smaller footprints of applications since there is no additional OS in-between, (ii) smaller footprint can be translated to better scalability and, (iii) a container offers better performance and start-up times because it runs directly on the kernel, and is preferred when performance is a critical requirement.

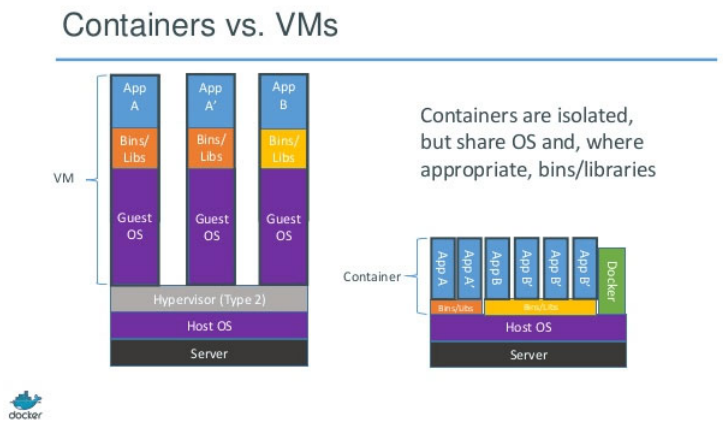

Figure 2.9: Container vs Virtual Machines [9]

Regarding Kernel Namespaces and Control Groups, the following are worth mentioning. Kernel Namespaces provide isolation between different groups of processes by dividing their kernel space into multiple environments, while Control Groups (or `cgroups`) are used to limit hardware resources like CPU usage, disk performance and memory.

Docker is an open-source platform which allows applications to be deployed inside software containers that utilize the core concepts of container virtualization that we described above. With Docker, you can manage infrastructure like you manage applications. Moreover, it offers easy shipping, testing and code deployment which can lower the delay between development and release.

For the implementation of this thesis, we used Docker because we wanted to build a "mini"-Internet that has multiple ASes. The small footprint that Docker containers have solve any scalability issues that we might have come across if we used hypervisor virtualization.

2.5.3 Mininet

Existing OpenFlow-supported devices are expensive, as it is not profitable for the vendors to mass-produce them because of the low demand; this disincentivizes companies from investing to SDN. Also, high-cost devices make experiments very difficult to implement and test new ideas. To combat this limitation, a virtualized approach was proposed. The most known SDN emulator is called Mininet. Mininet [15] is a network emulator like GNS3 but was firstly introduced to support SDN environments. It uses a container approach like Docker to create the nodes and enables SDN development on any laptop or PC. Nodes can be any linked set of virtual hosts, switches, controllers. Mininet hosts run standard Linux network software and ultimately, provide an environment for research, rapid prototyping of software-defined

networks, testing, and debugging. Some of the key aspects of Mininet are:

- Simple and inexpensive network testbed for developing and testing OpenFlow applications.
- Provides a framework that enables multiple developers to work concurrently on the same topology.
- Supports regression tests, that are repeatable and easily deployed.
- Generates complex topologies, without the need of wiring up physical devices.
- Provides a command line interface that is topology-aware.
- Provides out-of-the-box topologies, that can be parameterized to fit any need without additional programming.
- Offers a Python API and documentation to experiment and automate the network creation.

In summary, Mininet is a useful tool for SDN emulating purposes. It offers an easy way to build topologies programmatically and guarantees applicability of tested applications that will perform the same in a real world scenario. Also, it allows large-scale network emulation because it virtualizes less and shares more. Mininet is used in this thesis to build our topology and evaluate the performance of our tool in cases where GNS3 and Docker were not efficient.

Chapter 3

Related Work

In the following, we summarize works on detection and mitigation techniques on BGP (sub-)prefix hijacking and also other SDN approaches that aim to improve Internet routing.

3.1 Detection of BGP Hijacks

The purpose of the detection techniques are to discover any suspicious activity in Internet routing and raise alarms. These methods can be classified based on the type of information they monitor as: (i) control plane, (ii) data plane, and (iii) combination of both (hybrid).

Control plane approaches primarily collect BGP updates or routing tables from BGP monitors and inform the origin-AS of a prefix, if a malicious intent is recognized. These approaches are lightweight because they receive BGP feeds passively.

An early work by [31] focused on detecting (sub-)prefix hijacks with the use of a static prefix ownership map or taking advantage of optional BGP message attributes. Another research by Qiu et al. [58] suggested a protocol that is used between ASes and can alert an origin AS when a prefix suspiciously originates by multiple ASes. A prefix hijack real-time alert system was presented by Lad et al. [48] which ASes can use to register their prefixes and get alerted if a different AS origin is detected in a BGP update message. Simirarly to [48], Qiu et al. [57] used instead historical routing data and prefix ownerships with validation through various heuristics. Siganos and Faloutsos [68] collected information from the Regional Internet Registries and the Internet Routing Registry to correlate with data announced in the BGP update messages. Other approaches [40, 42] introduced a distributed architecture using Chord-based DHTs [69] that needed ASes to participate and build the ground truth. Heaberlen et al. [39] use trusted sources for relationships between ASes and proposes a solution where

all BGP update messages and routing policies are logged and shared between the neighboring ASes. To avoid revealing this –sensitive– information, other approaches [38, 53, 75] were presented, where they used cryptography. However, in contrast to ARTEMIS, most of the approaches use third-party databases to build the ground truth, which may not always represent the reality, thus resulting if false-positive hijacks. Furthermore, ARTEMIS does not need additional protocols [58] or support from neighboring ASes in order to run [39, 38, 53, 75]. Lastly, ARTEMIS can spot advanced type-N ($N > 1$) hijack events that are harder to detect.

Data plane approaches typically depend on active measurements of the Internet, e.g., using pings/traceroutes, to detect changes on AS paths and reachability of a prefix. Zhang et al. [74] presented iSPY, a system that monitors the connectivity of a prefix and alerts the AS if there is a reachability issue. Following a same method, Zheng et al. [76] detect prefix hijacks by observing paths and discover whether a significant variation is detected. Both [74, 76] employ vantage points outside of the target prefix but close to the AS. To compensate the overhead of permanent active probing, Quan et al. [60] following the same methods, applied Bayesian inference on the probing data. Hiran et al. [29] collected passively RTT by end-users and use them to identify routing anomalies when an outlier RTT was present. Balu et al. [41], complimentary to [29], took into account the number of intermediate hops to reduce the quantity of false positives. While some of these approaches [74] can run without the need of third-party entities and be deployed by the network operator itself, they are not capable of detecting reliably prefix hijacks. In cases of link failures or prefix hijacks on sub-prefixes where a probe is not present, these systems will report falsely that there is a hijack, or even miss it completely (false negative). Finally, being active measurement approaches, they are more heavyweight than the control plane approaches.

Hybrid approaches such as HEAP [62], Argus [66], and Hu et al. [43] combine control and data plane to overcome limitations that are introduced when used separately. HEAP [62] and Hu et al. [43] are detecting sub-prefix hijack attacks that blackhole or imposture the data plane traffic, thus missing Man-in-the-Middle (MitM) attacks. On the other hand, Argus [66] considers only blackholing and not imposture or MitM hijack events. BGP-mon [5] is the most popular commercial detection tool that combines information on both planes; however, being offered as a third-party service, each alert needs to be handled manually by each AS network operator. In contrast, ARTEMIS introduces an approach that handles all three categories of prefix hijacking.

3.2 Mitigation of BGP Hijacks

Mitigation techniques aim to defend the victim against the ongoing prefix hijack attack. Prototype ARTEMIS [65] that our SDN application implements uses prefix-deaggregation –when feasible– or outsources the mitigation to helper ASes that will act as Multi-origin ASes.

Early studies focused on introducing cryptography to BGP, such as S-BGP [45], RPKI [49], BGPSEC [50], and Whisper and Listen [70]. Karlin et al. [44] proposed `pretty good BGP` (pgBGP) which adds a verification process for any suspicious route that is observed. Qiu et al. [59] followed a similar outsourcing approach to ARTEMIS [65], which selects ASes to use as relays when a hijack is present. As shown in [65], ARTEMIS does not require any large-scale coordination and can achieve a highly impactful outsourced mitigation using a small number of carefully selected helper ASes, in contrast to other approaches. Also, cryptographic approaches require changes on the existing protocols and require global adoption, which is proven to be infeasible due to technological, political and economic factors.

3.3 SDN on Internet Routing

Many researchers tried to introduce SDN in order to solve known security and performance issues in Internet routing [37, 52, 47, 46, 6, 36].

Kotronis et al. [46] proposed SIREN, a BGP-SDN emulation framework that can evolve BGP by providing a birds-eye view over several different networks. A proof of concept was also demonstrated that aimed to improve the slow convergence time of BGP. An evaluation methodology regarding acceleration of BGP convergence on SDN was carried out by Sermpezis and Dimitropoulos [6] where they analyze the effects of centralization on inter-domain routing. Gupta et al. [37] created SDX, an SDN-enabled Internet eXchange Point (IXP). Authors believed that SDN could revolutionize wide-area traffic delivery and solve known inter-domain routing problems by introducing a new wider range of policies, simplifying network management and enabling new networked services without the need of changing the current technology. Similarly, an IXP-based system that establishes QoS route paths through path stitching is described in [47]. In [51], the authors discuss a solution for incremental deployment of SDN networks and how they can seamlessly peer with IP networks.

On the security side, Wang et al. [73] implement FloodGuard, a Denial of Service (DoS) attack prevention extension in SDN. Similar work was presented by Gkounis et al. [36], where they show an SDN-based approach to mitigate different types of DDoS attacks, called `Crossfire` attacks. A survey on SDN security is presented by Shin et al. [67] that provide insights for future research in this area.

Chapter 4

ARTEMIS over SDN

$\text{ARTEMIS}_{\text{ONOS}}$ follows the same principles as legacy ARTEMIS [33] which consists of three components: a monitoring, a detection and a mitigation service. In our implementation we follow the same methodology and implement these three services while taking advantage of Software Defined Networking principles, mechanisms and applications.

4.1 Methodology

In this section we explain our methodology and why we decided to move legacy ARTEMIS concepts to a Software Defined Networking solution, how we took advantage of ONOS applications and abstractions it offers and finally, the three services of ARTEMIS.

4.1.1 Open Networking Operating System (ONOS)

Legacy ARTEMIS [33] was implemented over python and was deployed as a package of scripts. We decided to introduce ARTEMIS into Software Defined Networks to overcome any automation limitation that the legacy approach had. ONOS was chosen as the best choice to implement $\text{ARTEMIS}_{\text{ONOS}}$ as it provides applications and abstractions that would greatly enhance ARTEMIS's features.

We wanted $\text{ARTEMIS}_{\text{ONOS}}$ to be a modular application and because ONOS runs over Apache Karaf [1] it offers a modular runtime environment where all the applications can co-exist with each other and exchange information between them when needed.

Also, as mentioned in section 2.1.1.4 because each device in the network needs to be configured manually in a decentralized manner, it is very prone to misconfigurations. ONOS offers a bird's eye view of the network and lets the operator control the configuration of the devices through an abstraction layer that is agnostic to the underlaying protocols, eventually configuring

the devices through a logically centralized entity, the controller. Except for the configuration of the devices, configuration of the applications is loaded through a core application that is packaged with ONOS, making it easy to spot any misconfiguration or update an application with new variables.

Apart from the advantages on the modularity and the centralized view and configuration of the network, ONOS is packaged with some applications that are globally-contributed and open-source that let Software Defined Networks co-exist with the legacy networks. The applications that we used from ONOS are SDN-IP and Reactive-Routing, detailed in the following,

4.1.1.1 SDN-IP and Reactive-Routing

In the background section, in subsection 2.3.5.1 we explained that SDN-IP enables an SDN AS to connect with legacy ASes through BGP. For our work, we used SDN-IP to keep track of the connected BGP speakers and peers but also the routing tables. This information was very important and was used by the detection service to see if there is a hijack incoming to the routing table and by the mitigation service to select which BGP speaker to use to mitigate the hijack.

Reactive-Routing is a required application for the SDN-IP application to work. It installs all the flow rules proactively so the BGP peers can communicate with the internal BGP speakers but also installs flows re-actively to provide reachability for the hosts inside the AS to the hosts in the Internet. In Figure 4.1 we show how these applications interact when the network is initialized.

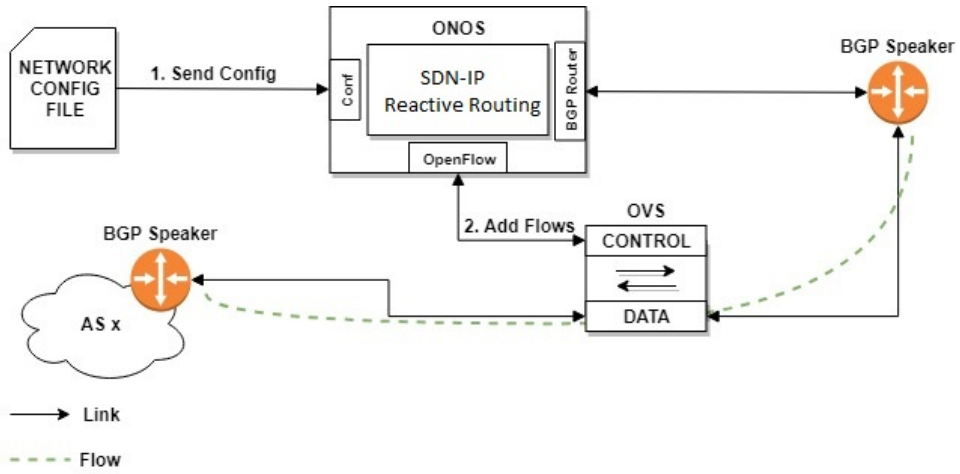

Figure 4.1: SDN-IP and Reactive Routing initialization

4.1.2 ARTEMIS'$_{\text{SONOS}}$ Services

4.1.2.1 Monitoring

ARTEMIS'$_{\text{SONOS}}$ monitoring service is run locally and its purpose is to subscribe to multiple control plane monitors for the prefixes it owns. By having multiple control plane monitors that are spread across the globe we have visibility to cover a big percentage of the Internet.

A network operator needs to provide a configuration file to the monitoring service that lists all the control plane monitors that will be used by the service and as the output of the service, all BGP Update messages for the prefixes specified in the configuration will be forwarded to all of its listeners.

4.1.2.2 Detection

The detection service receives two inputs and has one output. As inputs it has (a) a configuration file that includes information about the prefixes owned by the operator, such as the Origin AS and the first-hop neighbors of the AS, and (b) the output of the Monitoring Service, which are the BGP Update messages received from the control-plane monitors.

Having these two inputs, the service can compare the AS PATH fields of the BGP Update messages with the ones provided in the configuration and detect, if present, a hijack attempt. This event will be the output of this service and be forwarded to the subscribers of the detection service.

As we explained in subsection 2.1.1.4 a hijacker can either do an attack on the same prefix or take advantage of the longest prefix matching algorithm and attack a sub-prefix. A sub-prefix hijack is the most dangerous type of an attack because it can potentially pollute the whole Internet. Also, it is the most problematic because whenever an AS needs to announce a longer prefix or de-aggregate it will trigger a false positive hijack alert.

ARTEMIS$_{\text{ONOS}}$, due its nature of being self-operated, returns 0 false positives and 0 false negatives for type-0 and type-1 hijacks. For hijacks of type-N (where $N > 1$), because the network operators do not always know the second (or longer)-hop neighbors on their AS PATH (plus this path may be frequently updated), the approach requires a special handling detection-wise, which will not be covered by this thesis.

For the basic sub-prefix, type-0 and type-1 hijacks accuracy is very high because the tool runs based on the provided configuration which includes an up-to-date list of all owned prefixes and their AS neighbors. When a prefix hijack takes place ARTEMIS$_{\text{ONOS}}$ will detect it through a control-plane monitor. Our approach however can detect hijacks where attackers announce an illegitimate AS PATH so it can carry the attack on the data plane by being a Man-in-the-middle (Type-1). Furthermore, we will illustrate how ARTEMIS$_{\text{ONOS}}$ detects these different types of exact prefix hijacks.

The configuration file that we mentioned has the following information per prefix:

- **Origin AS** - the AS that the prefix originates from.
- **Neighbor ASes** - the ASes that are direct neighbors of the origin AS.

For every BGP update message it receives from the monitor service, ARTEMIS$_{ONOS}$ extracts the AS PATH field and compares the announced prefix as well as the first and second AS in the AS PATH. This way, ARTEMIS$_{ONOS}$ can detect all Type-0 and Type-1 hijacks for the prefixes that are specified inside the configuration file and are visible to the control-plane monitors.

If there is an update message for an owned prefix which is not matched within the configuration then we have a hijack with 0% of it being a false positive. A new event will be generated that includes information about the type of the hijack, the prefix that is hijacked, a timestamp and the fraudulent AS PATH, which is the output of this service.

4.1.2.3 Mitigation

After the detection service detects a hijack event, we need to quickly mitigate the hijack. Since we have as an input the hijack events generated from the detection service, which follow a specific structure, we can extract from it the type of the hijack.

In the mitigation service we can define actions in order to mitigate each event automatically based on different factors. Sermpezis et al. [64] conducted a survey which asked operators how they mitigate ongoing hijacks; they answered that either they contact other networks (88%) and/or the engage in prefix de-aggregation (%68). ARTEMIS$_{ONOS}$'s aim is to remove the need of manual contact with other networks or manually changing the configurations to do a de-aggregation. Instead, it proposes an automated approach.

Prefix de-aggregation is the act of inserting more-specific prefixes into the BGP table of a router (to have them advertised to its peers) in order to do traffic engineering, which in our case, is to re-claim the traffic stolen by the hijacker. This way polluted ASes that have wrong routes for the hijacked prefix in their BGP routing table will re-establish legitimate routes since longest prefix matches are always preferred by BGP.

The de-aggregation approach is very simple and can be operated by the network itself without any additional cost. The problem arises when a hijack event involves a /24 prefix which cannot be de-aggregated because operators usually filter prefixes that are more specific than /24. To mitigate these hijacks we can employ additional networks that run ARTEMIS$_{ONOS}$

or an agent that supports ARTEMIS$_{ONOS}$'s communication scheme, and outsource the mitigation to them.

Outsourcing BGP announcements follows similar principles as the DDoS protection security model. The main idea is that the victim AS, when it gets hijacked, communicates with a friendly AS and asks for help. The friendly AS then announces the hijacked prefix on the victim's behalf. Then the friendly AS will attract the hijacked data-plane traffic and redirect it through an out-of-band tunnel to the victim AS.

4.2 Implementation

In this section we explain in detail the general work-flow and the third party applications of ONOS we used. Then we will focus more on the three services of ARTEMIS$_{ONOS}$ application; the monitoring, the detection, and the mitigation service and how they communicate with each other (Figure 4.2). We will extensively describe the design pattern that we used on the services and the interfaces that are reachable by other applications. Following the event-listener design pattern allows for other services to use the ARTEMIS$_{ONOS}$'s services and/or extend the capabilities of the application. This approach is spread through all three services; so we can have a modular design.

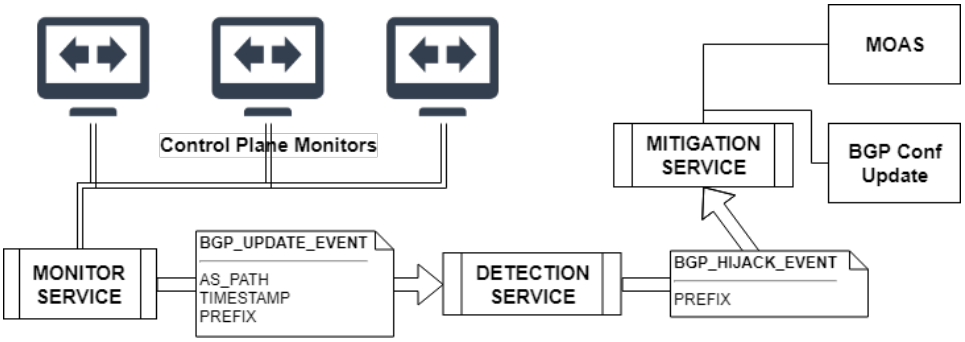

Figure 4.2: Services lifecycle

4.2.1 Open Networking Operating System (ONOS)

4.2.1.1 SDN-IP and Reactive-Routing

As we explained in the methodology, Reactive-Routing is a complementary application to SDN-IP. When we start these two applications Reactive-Routing and SDN-IP:

- Install rules **pro-actively** to handle transit traffic between the BGP peers,

- Install rules **pro-actively** to handle traffic coming from the Internet to talk to a host that is part of the SDN network,
- Install rules **re-actively** for communication between two hosts, and
- Install rules **re-actively** when an internal host wants to talk to the Internet.

For the Reactive-Routing application the network operator needs to provide a configuration file that specifies the following (sample configuration can be found in Appendix A.2):

- **PUBLIC** and **PRIVATE** prefixes used inside the SDN network. PUBLIC prefixes are the ones announced by the BGP speakers and are reachable from the Internet, where PRIVATE are local prefixes which are only used inside the SDN network.
- the ports of the SDN switches that will act as gateways for each prefix, and
- a MAC address that will be used as a Virtual Gateway address. Virtual Gateway is needed because we do not have layer-3 routers on the SDN data plane; therefore, the switches need to reply to the ARP requests of the hosts asking about their next hop to the gateway.

On the other hand, in the configuration of the SDN-IP application we provide the ports of the switches that the BGP speakers and BGP peers are attached to. By doing this, Point-to-Point intents will be installed between the speakers and the peers for the BGP traffic which will enable the communication between them through eBGP. Now whenever a new BGP update message reaches the BGP speakers it will be forwarded to the SDN-IP application through iBGP.

4.2.2 ARTEMIS$_{\text{ONOS}}$'s Services

4.2.2.1 Monitoring

ARTEMIS$_{\text{ONOS}}$'s monitoring service can connect to any BGP route collector (e.g., based on exaBGP [11]), the streaming service of RIPE RIS [23], and any additional source required by the network operator, as long as the monitor interface is implemented inside the application.

These route collectors provide real-time BGP update messages for the specified prefixes, that the application needs to monitor. The monitoring service can be parameterized via a straightforward network configuration file, where the network operator provides the prefixes under protection, as well as the monitors to connect to. Additionally, the configuration includes

the inferred neighbors of the ASN in order to detect (sub-/exact-prefix) hijacks of type 0, 1 and N ($N > 1$).

When receiving a BGP Update message from a monitor, the service will generate a BGP_UPDATE_EVENT and will notify all listeners about the message. In a default scenario where the network operator runs ARTEMIS$_{\text{ONOS}}$ with all of its services, this means that the Detection's service listener will receive the BGP_UPDATE_EVENT and process it accordingly.

Monitors Interface. ARTEMIS$_{\text{ONOS}}$ natively supports exaBGP-based and RIPE RIS control-plane monitors. However, the monitoring service provides a Java interface (Monitors.java) that can be implemented to support any control-plane monitor. To introduce a new control-plane monitor the developer needs to provide the following methods:

- **startMonitor**: a method that connects and subscribes to a control plane monitor based on the given prefixes. For example, the implementation for RIPE RIS monitors uses a `socket.IO` client and a special crafted message to subscribe.
- **stopMonitor**: a method that gracefully stops the monitor.
- **{set/get}Prefix**: a setter/getter method for the IP prefix that will be monitored.
- **{set/get}Host**: a setter/getter method for the IP of the monitor to connect to.
- **isRunning**: a method that checks the state of the monitor to see if it running. It can be used as a health check method for the monitor.

Event Dispatcher. ONOS has an event delivery service that ARTEMIS$_{\text{ONOS}}$ uses to dispatch its events. Whenever a control-plane message is received on a monitor, a new BGP_UPDATE_EVENT is generated that includes the BGP update message. Another service can attach listeners to these types of events and use them accordingly. For example, the detection service handles these events by registering an ARTEMIS$_{\text{ONOS}}$ event listener. The BGP_UPDATE_EVENT message fields are:

- **path**: A list of ASes, known as AS PATH.
- **prefix**: The announced IP prefix that this BGP update corresponds to.
- **timestamp**: Timestamp of this BGP update message.

4.2.2.2 Detection

The detection service has a listener that tracks new BGP_UPDATE_EVENT messages. When a BGP_UPDATE_EVENT is received, the service checks the AS PATH attribute to verify that the origin AS and the neighbors are legitimate as per configuration. When an unknown AS is listed as origin in the AS PATH, or there is an unknown neighbor, we safely assume that we have a BGP hijack and generate a BGP_HIJACK_EVENT to notify the mitigation service.

For example, the configuration might have a prefix of A.B.C.D/8 and as a legitimate AS origin, AS1, with first neighbor AS2 and second neighbor AS3. This means that if it receives a BGP update message for this prefix (or sub-prefixes of it) but has as origin some other ASX, the detection service will trigger the mitigation service because there is a Type 0 BGP Hijack. This approach is effective for type-0/1 and any sub-prefix hijacks. Type-Ns ($N > 1$) require explicit care but we do not focus on them here.

4.2.2.3 Mitigation

The mitigation service is triggered when the listener receives a BGP_HIJACK_EVENT. This event includes the hijacked prefix which, based on the prefix length, decides to either change the BGP configuration of the BGP speakers to de-aggregate (see subsection 4.2.2.3) or outsource the mitigation by using a MOAS setup (see subsection 4.2.2.3).

De-aggregation The SDN-IP application provides an interface that enlists all the active BGP speakers that are connected to the SDN network. By iterating over this list $\text{ARTEMIS}_{\text{ONOS}}$ chooses the supported BGP speakers and connects to them. Then it changes the BGP configuration of the BGP speaker to announce the new sub-prefixes and initiate the de-aggregation.

Mitigation Interface Following the same design as the detection service, we have a Java interface (`BgpSpeakers.java`) that can be implemented to support any BGP speaker. This interface only has one method that needs to implemented called `announceSubPrefixes`. The parameters of this method are the two sub-prefixes that the BGP speaker will use for deaggregation-based mitigation. The implementation includes the functions needed to connect to the speaker and announce them.

Currently, we support Quagga routers as BGP speakers for the de-aggregation phase. The Quagga router offers a CLI that is reachable over telnet. The mitigation service will connect to the desired BGP speaker through telnet and update the BGP configuration to announce the new sub-prefixes. However, dynamic configuration of different devices (including

BGP speakers) is under development in the ONOS project; therefore there is no actual limitation of the approach with respect to the speaker's type.

Outsourcing to MOAS When the hijacker announces a /24 prefix the mitigation service cannot de-aggregate to two /25 sub-prefixes because network operators usually filter prefixes more specific than /24. To bypass this filtering we implemented an east-west protocol where the hijacked AS can connect to other ARTEMIS$_{\text{ONOS}}$'s instances or ARTEMIS$_{\text{ONOS}}$'s agents and request help in a form of a service from them, labeling them as multi-origin AS (MOAS) helper.

After a handshake with the MOAS helper (Figure 4.3), the victim AS will send information regarding the prefix that is hijacked to the MOAS helper, which in turn updates the BGP configuration of the BGP speakers in order to announce the hijacked prefix itself. We decided that the handshake should be as fast as possible so we would avoid any additional delays that a longer one might have added. The victim AS will also announce the same prefix so it can attract all the traffic that is closer to it than the hijacker.

In order for this solution to work, the MOAS helper should have a tunnel installed beforehand, connecting the MOAS helper with the victim AS on the data plane. This tunnel will be used when the MOAS helper attracts the hijacked traffic that was originally destined to the victim AS. Then, MOAS helper ARTEMIS$_{\text{ONOS}}$ will install a Multi-to-Single point intent that encapsulates the packets to go through the tunnel and then eventually get routed to the original destination.

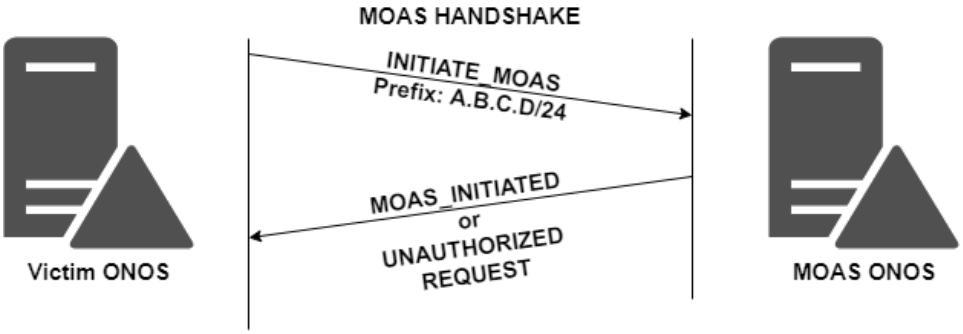

Figure 4.3: MOAS handshake

It is important to note here, that only the forward path traffic goes through the tunnel in order to reach the victim AS. Optimally, this defense mechanism provides a BGP hijack solution where the only additional cost is the delay that the packet needs to go through the MOAS helper on the forward path, Figure 4.4.

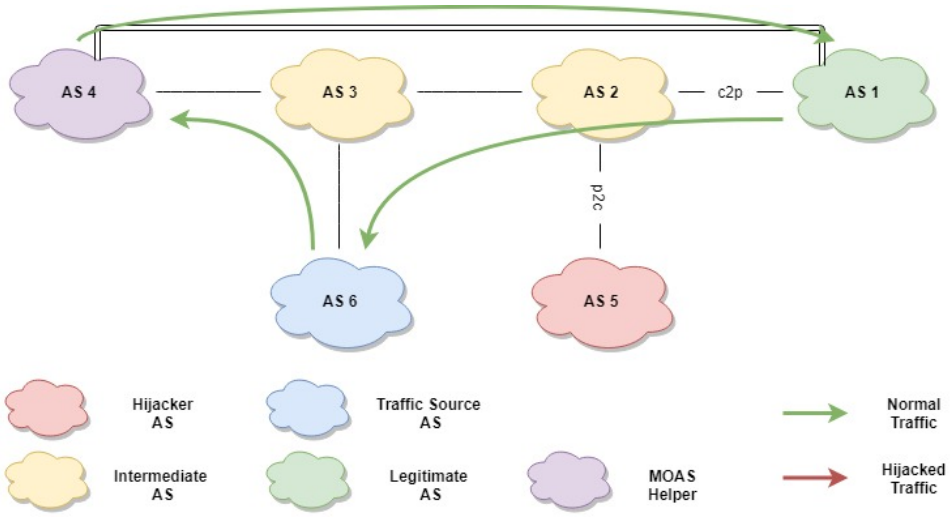

Figure 4.4: Traffic after MOAS mitigation approach

Chapter 5

Evaluation Framework

5.1 Overview

In this chapter we explain the methodology that we used to evaluate our tool and the results we observed. We wanted to emulate realistic hijack attempts and to evaluate the two mitigation scenarios which are: (a) de-aggregating from within the AS and (b) outsourcing the BGP announcements to another AS (MOAS). For (a) we used Mininet in order to emulate the topologies and test the tool's behavior, and for (b) we used GNS3, because we needed to achieve the same goal while having multiple ONOS instances. The latter is not supported by Mininet; also, GNS3's graphical interface makes it easier to setup multiple topologies and debug them. The emulated nodes inside GNS3 are docker containers that include Quagga BGP routers, Open vSwitch switches, hosts, normal switches, ExaBGP control plane monitors and lastly the ONOS controllers.

There were many implications that did not let us run our experiments in the real Internet. We did not have access to an AS to attach our controller to their BGP speaker and it would have been impossible to do a legitimate hijack without polluting the whole Internet. PEERING [21] offers a testbed that let researchers access and interact with the BGP routing system and gives us the theoretical ability to produce a hijack, but due to the inability to test the outsourcing mitigation technique we did not choose it as our evaluation method.

We chose to emulate Internet topologies and then do the hijack events there. Because the experiments were setup and run in an emulated environment and we did not have access to external control plane monitors that are deployed on the real Internet, we implemented our own control plane monitor on top of an ExaBGP router. This control plane monitor offers a `Socket.IO` Northbound Interface that clients can subscribe to and ask for the control plane traffic that corresponds to a specific prefix. When an

ExaBGP control plane monitor is specified in the ARTEMIS$_{ONOS}$'s configuration, ARTEMIS$_{ONOS}$ sends a subscribe request to the control plane monitor for each owned prefix; afterwards, it will receive any BGP update message corresponding to them.

For the topologies that we generated, we employed two types of ASes; (a) a normal AS that contains a BGP speaker and a simple host; these ASes could be either intermediate ASes or hijackers, and (b) a SDN-enabled AS that contains a BGP speaker, an OpenVSwitch, an ONOS controller, an ExaBGP monitor and a simple host. More specifically, the SDN-enabled AS has (i) a Quagga BGP speaker that connects with the ExaBGP monitor to forward the control plane traffic through iBGP, (ii) the OpenVSwitch that interconnects the AS with the rest of the Internet and (iii) the ONOS controller which uses ARTEMIS$_{ONOS}$, Reactive-Routing and SDN-IP applications to coordinate all these network devices. These types and associated devices are sufficient to illustrate the concept of ARTEMIS$_{ONOS}$.

Note that the ONOS controller must have access to the BGP speaker, to the control plane monitors and to any other ONOS controller that will communicate with it. For simplification, we used a global switch in our topologies through which the ONOS controllers and the control plane monitors communicate. These connections, in the real Internet, are meant to be out-of-band connections. Through these connections, control plane monitors connect to the controllers to transfer the control plane information. Moreover, ARTEMIS$_{ONOS}$'s instances talk with each other in case an outsourcing mitigation is needed.

Before we explain independently each attack and mitigation scenario, we will analyze the steps that are the same in both cases.

When we start the emulation, all of the ASes have a BGP speaker with a BGP configuration that specifies the prefixes that they own and the neighbors that they have. So, in the beginning, BGP will converge and the ASes will populate their local routing tables. The protected SDN-enabled AS will not be visible to the outer Internet because when initialized, ONOS only runs the core applications, so we need to activate reactive-routing, SDN-IP and ARTEMIS$_{ONOS}$.

By activating these applications and providing the network configuration for each of them we trigger the following actions inside the SDN network:

- The SDN-IP will connect with the BGP speaker through iBGP and wait for any BGP update that it receives.

- A point-to-point intent will be installed to forward all the BGP traffic from the outside BGP peers to the BGP speaker of the SDN network. This means that the BGP speaker of the SDN network will announce through eBGP the prefix that it owns, but also receive all the routing information concerning the other connected ASes; this information will

be then forwarded to the SDN-IP application. So, after this step the protected SDN-enabled AS will be reachable from the Internet and BGP will converge.

- A Multi-to-Single intent will be installed to route any traffic coming from the Internet destined to the internal hosts and a Point-to-Multi intent for the other way around, and
- ARTEMIS$_{\text{ONOS}}$'s detection service will subscribe to all the control plane monitors specified in the configuration.

Now the protected SDN-enabled AS is connected to the legacy ASes and has exchanged all the routing information through BGP. Also, all hosts are now reachable and can exchange data plane traffic between them.

The configuration files that we provide to these applications and their fields are:

SDN-IP Configuration (Appendix A.3)

- The ports on the OpenVSwitch that the BGP speakers are attached to.
- The IPs of the BGP peers that are connected to this AS.

Reactive-Routing Configuration (Appendix A.2)

- The PUBLIC and PRIVATE IP prefixes with their gateway IP.
- A virtual gateway MAC address.

ARTEMIS$_{\text{ONOS}}$ Configuration (Appendix A.1)

- The prefixes that the protected AS owns and we want to monitor for BGP prefix hijacks.
- The AS paths that are legitimate for each prefix.
- A list of all control plane monitors to use, as well as their interface type (RIPE RIS, ExaBGP, etc).

To make it easier for the reader to understand which network entity is which, we are going to label them with the following subscripts (note that: R →router, H →host, EXA →ExaBGP monitor, $ONOS$ →ONOS, AS →AS):

- AS_p, R_p, H_p, $ONOS_p$ - We label with the letter 'p' the network devices that are inside the protected SDN-enabled AS that runs ARTEMIS$_{\text{ONOS}}$.

- AS_h, R_h, H_h - Letter 'h' will correspond to devices that are inside the hijackers AS.

- AS_t, EXA_t, H_t - Network entities labeled with letter 't' will correspond to the devices that are inside the AS that generates the traffic to the protected AS.

- AS_m, EXA_m, $ONOS_m$ - Letter 'm' corresponds to the friendly Multi-Origin AS (MOAS).

- **default** - No letter means that this is an intermediate AS, with the purpose of adding additional nodes on the AS graph (e.g., for large-scale experiments).

5.2 De-aggregation Scenario

For this test scenario we wanted to have an AS with a control plane monitor, a protected SDN-enabled AS that runs ARTEMIS$_{ONOS}$, SDN-IP and Reactive-routing, a random number of intermediate ASes to test the difference of the BGP convergence when the AS graph changes, and the hijacker AS. The hijacker AS is always positioned closer to the AS that the traffic will originate from, so it can hijack traffic based on the shortest path to reach it. Lastly, all the links between the network devices for this type of experiment have only 1ms delay.

5.2.1 Experiment Setup

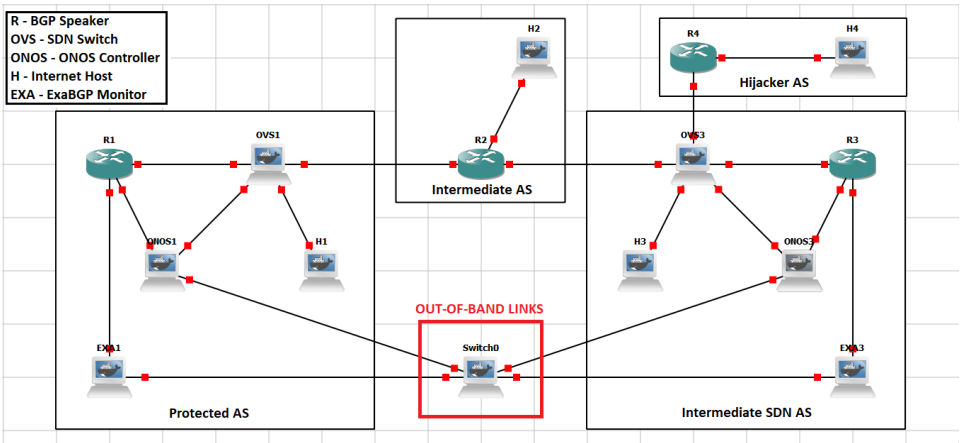

Figure 5.1: GNS3 Demo Topology

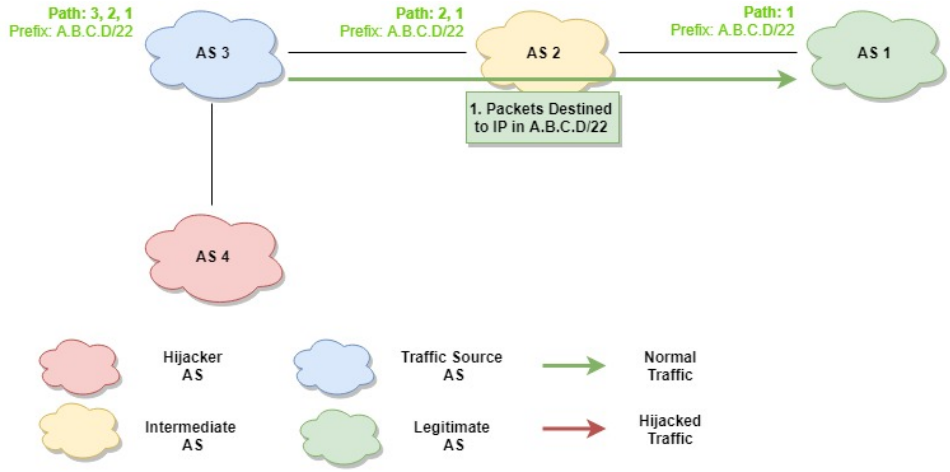

Figure 5.2: Abstracted demo topology of 4 ASes

A simple topology that can be used to test the de-aggregation approach is depicted in Figure 5.1 and abstracted in Figure 5.2. In this topology we have 4 ASes; three of them are normal ASes and the other one is the SDN-enabled AS. They are connected like this: $AS1_p$ - AS2 - $AS3_t$ - $AS4_h$. The protected AS in this topology is $AS1_p$, the intermediate AS is AS2, the intermediate AS with the control plane monitor and the traffic generator is $AS3_t$, and the hijacker is $AS4_h$.

Also, each AS has a prefix that it originates based on their AS number. For an AS with number X the prefix that will originate from them will be 10.x.0.0/22 and the –single– host of each AS always has IP 10.x.0.100.

After routes towards the prefixes of all ASes, including the SDN ASes, have converged in BGP, we are going to start sending data plane traffic from $H3_t$ to $H1_p$. When a data plane packet reaches the OpenVSwitch ($OVS1_p$) the packet will be forwarded to the port where $H1_p$ is, because $OVS1_p$ already has rules to route this traffic; the latter rules were pro-actively installed by the reactive-routing application.

While the traffic is exchanged, the hijacker $AS4_h$ will announce a sub-prefix of $AS1_p$. In this case, a sub-prefix of 10.1.0.0/22 would be 10.1.0.0/23. We also change the IP of $H4_h$ to be 10.1.0.100 so that it receives the traffic that was originally destined to $H1_p$.

When we change the BGP configuration of $R4_h$ to announce the 10.1.0.0/23 prefix we keep a timestamp t_h which signals the start of the hijack. This BGP update message will eventually pollute all BGP routing tables of other ASes into thinking that 10.1.0.0/23 prefix originates now from $AS4_h$ and not from $AS1_p$. The data plane traffic will get redirected from $H1_p$ to $H4_h$, which signals a successful hijack. We will denote this event with timestamp t_s. In Figure 5.3 we see a simple representation of a successful hijack.

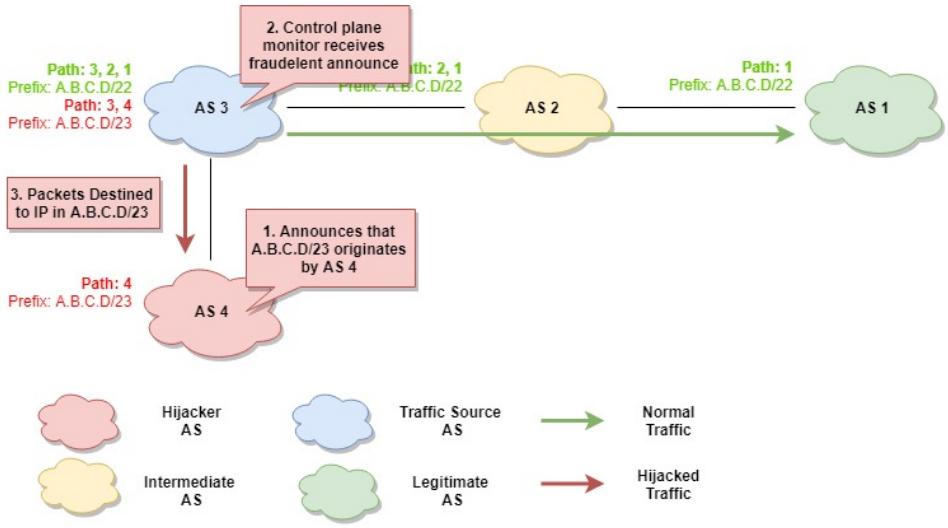

Figure 5.3: Hijacker announces illegitimate sub-prefix

When the illegitimate BGP update message reaches BGP speaker $R3_t$, it is also going to be forwarded through iBGP to the ExaBGP monitor, $EXA3_t$. The ExaBGP monitor will then send this message to the subscribers of that prefix. Because the protected $AS1_p$ had subscribed for prefix 10.1.0.0/22 which includes (i.e., is a super-prefix of) 10.1.0.0/23, the monitor will forward the BGP update message to the subscribed $ARTEMIS_p$.

The monitor service of $ARTEMIS_p$ will receive the BGP update message and add it to a queue of the detection service. Because the BGP update message will have an illegitimate origin AS of $AS3_h$, this will generate a BGP_HIJACK_EVENT that will be sent to the mitigation service.

The mitigation service will inspect the hijacked prefix that is /23 and will decide to follow the de-aggregation approach by announcing through $R1_p$ two /24 prefixes. When the announcement is done, we keep a timestamp denoted as t_m. After some time t_d, the BGP update message will reach $R3_t$ and converge, and the data plane traffic will return back to the destined $H1_h$. When BGP converges on $AS4_h$ which started the hijack we also keep a final timestamp t_e.

As seen in Figure 5.4 the de-aggregation approach results in retrieving back all the stolen traffic from the hijacker.

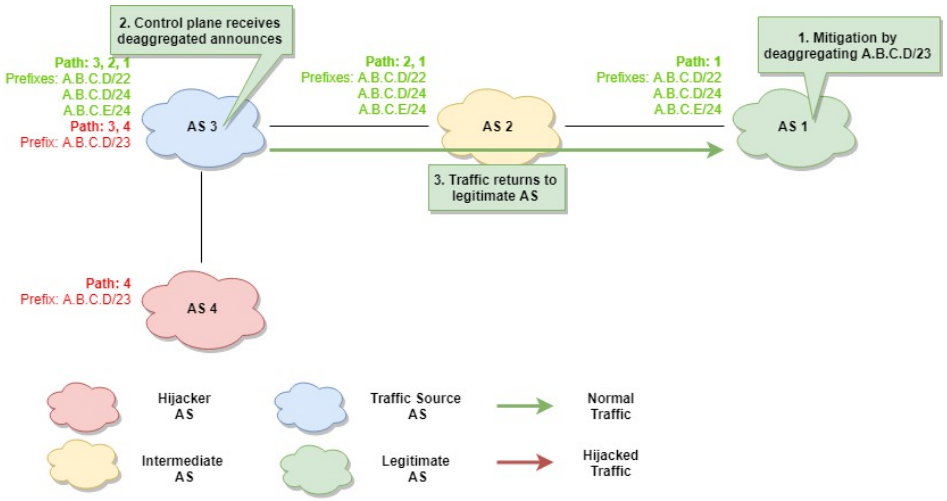

Figure 5.4: Successful mitigation through de-aggregation

5.2.2 Results

After experimenting with different topologies, we gather the timestamps mentioned before and split them in four stages:

1. **Stage 1** - is the time difference between t_h and t_s, which translates to the time the hijack needs to reach a control plane monitor.
2. **Stage 2** - is the time between t_s and t_m and indicates the ARTEMIS$_{\text{ONOS}}$'s reaction speed from detecting to mitigating.
3. **Stage 3** - represents the BGP convergence time for the de-aggregated prefixes to reach the control plane AS that detected the hijack (t_m to t_d).
4. **Stage 4** - this is the time needed for the legitimate BGP update to reach the hijacker (t_d to t_e).

For the sample topology that was described in subsection 5.2.1, we run our ARTEMIS$_{\text{ONOS}}$ setup 10 times and gather the timestamps to calculate each stage which are listed in Figure 5.5.

By calculating the averages we form a stacked bar plot in Figure 5.6 with which we compare the time of each stage so that we can evaluate the performance of our tool and what can be considered the overhead of our methodology. We can see that when a control plane monitor is one hop away from the hijacker, it takes around 4 seconds to receive the polluted BGP update. Then, the control plane monitor sends the polluted BGP update which will be handled by the ARTEMIS$_{\text{ONOS}}$ application and eventually mitigate the hijack through announcing the more specific prefixes. The

average time that ARTEMIS$_{ONOS}$ takes to detect and mitigate is in the order of milliseconds (1-2ms), which is not visible in the original Figure 5.6, but you can see it if we zoom in at Figure 5.7. Then, the BGP convergence time for the de-aggregated prefixes that signs the end of the mitigation, is shown as the overhead, with an average time of 45 seconds. For the last stage, the BGP update message reaches the source of the hijack from the control plane monitor, after 5 seconds.

With these results, we can see that the ARTEMIS$_{ONOS}$ application reacts almost instantaneously when a control plane monitor receives the hijack in a matter of milliseconds, and the only large and unavoidable delay is the BGP convergence of the de-aggregated prefixes.

We also run our methodology with other topologies that followed the sample one but with adding additional intermediate ASes between the AS_p and AS_t. When we have a huge topology with multiple intermediate ASes, thanks to ARTEMIS$_{ONOS}$'s fast reaction speed and mitigation, the BGP updates of the de-aggregated prefixes sometimes reach ASes closer to AS_p faster than the polluted BGP updates. Typically, ASes that are closer to AS_p than AS_h will not prefer routing to the hijacker as they have already a longer prefix to match upon (generated during the mitigation process) but also a shorter AS PATH.

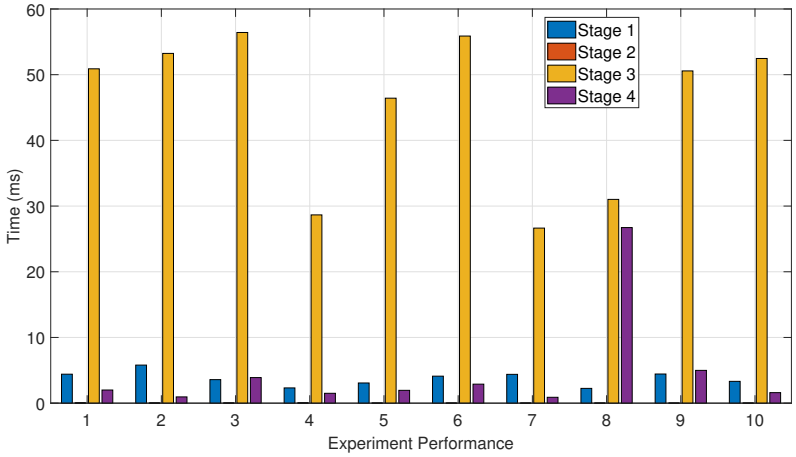

Figure 5.5: Bar plot of all BGP hijacking stages

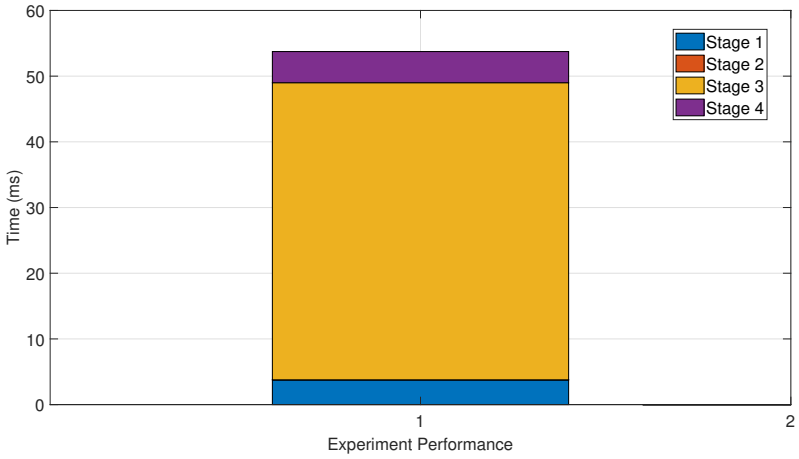

Figure 5.6: Stacked plot of all BGP hijacking stages

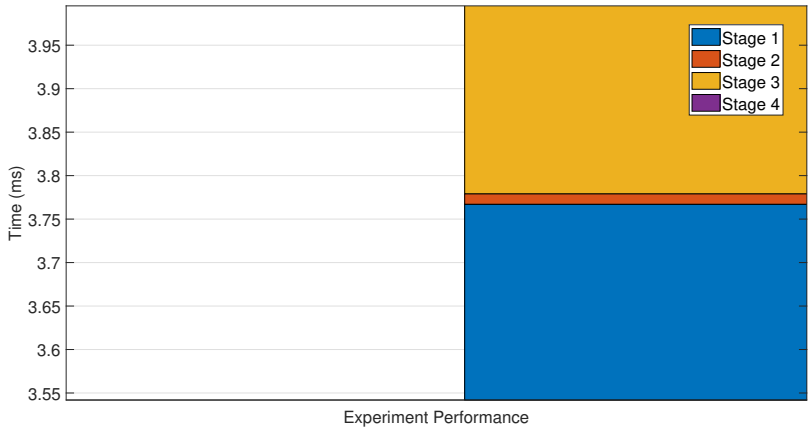

Figure 5.7: Zoomed-in Figure 5.3 to show ARTEMIS$_{\text{ONOS}}$'s time

5.3 Outsourcing Mitigation Scenario

More specific prefixes than /24 tend to be filtered from the network operators to avoid having huge BGP routing tables, which can affect the speed but also need more memory. Therefor, in this test scenario, the main difference is that the hijacker announces a more specific prefix /24 and the protected AS needs not only to announce the /24 prefix, but also to outsource the BGP announcements in order to get back the data plane traffic that is closer to the hijacker through tunneling. Also, for the emulation platform we used GNS3 instead of Mininet because Mininet cannot support multiple instances

of ONOS running simultaneously. The links between the ASes have now a fixed delay of 10ms, while the links between the network devices inside an AS a delay of 1ms. We decided to have different delays so we can measure the difference when a packet is routed through another path, or in this case through the MOAS helper, AS_m.

5.3.1 Experiment Setup

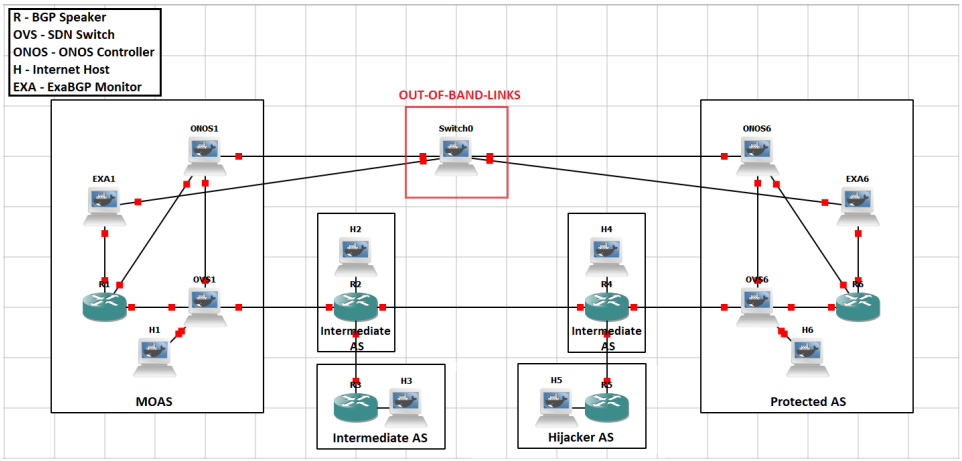

Figure 5.8: MOAS Helper Demo Topology

The simplest topology to test the outsourcing approach and introduce the Multi-Origin AS_m, is depicted in Figure 5.8. Before we move on with the hijack experiment, we need to state that a tunnel is setup upfront between $AS6_p$ and $AS1_m$, and is going to be used to route the data plane traffic to the legitimate prefix owner after the hijack.

Also, another difference from the previous scenarios is that the hijacker and the protected AS have the same neighboring AS, AS4. This neighboring AS has BGP policies for the protected AS and the hijacker AS so it will always prefer the hijacker when the AS_PATH length is the same. To achieve this in practice, the hijacker AS is set as a CUSTOMER to the intermediate AS where the protected AS is set as a PROVIDER.

Here, the hijacker $AS5_h$ is going to hijack a /24 sub-prefix of $AS6_p$. When $AS5_h$ announces the /24 sub-prefix at t_h, and after BGP converges with the illegitimate route, $H3_t$ which is the traffic origin will talk with $H5_h$ instead of $H6_p$; this event is timestamped as t_s. The illegitimate BGP update message will also reach $AS1_m$ which in our case is a friendly AS that runs ARTEMIS$_{\text{ONOS}}$ as well and communicates with the ARTEMIS$_{\text{ONOS}}$ instance of $AS6_p$. $R1_m$ will send the BGP update message to $EXA1_m$ and through the out-of-band link to $ONOS6_p$.

ARTEMIS$_{\text{ONOS}}$ in $AS6_p$ will now receive the BGP update message on the monitoring service and fire up a BGP_UPDATE_EVENT for the detection service. The detection service will see that the origin AS on the BGP update message do not match the legitimate ASes and because now a /24 prefix is hijacked, it will use the outsourcing method by contacting a MOAS helper and also update its BGP configuration to announce the /24 prefix as well.

Through ARTEMIS$_{\text{ONOS}}$, $ONOS6_p$ will talk with $ONOS1_m$ and request help for the /24 prefix, timestamped as t_m. Then, $ONOS6_p$, will announce the attacked /24 prefix which will cause the $AS3_t$ to update the BGP routes to forward traffic to the $AS1_m$ instead of the hijacker AS, timestamped as t_d.

$AS1_m$ will install flow rules that will redirect through the mentioned pre-installed tunnel all incoming traffic of the attacked prefix to the protected $AS6_p$.

5.3.2 Results

The stages that we used for the previous mitigation technique are still the same, except for adding an intermediate **Stage 2.5** which is the time the MOAS helper needs to announce the prefix. This stage's execution time varies according to the delay that exists between the two ASes (the protected AS and the MOAS helper). Because they exchange only one message, in order to start the mitigation, we only need to measure the forward path delay. For the sample topology that is shown in Figure 5.8, during **Stage 2.5** it takes 20ms for the request to reach the MOAS helper through the out-of-band link and 1ms to announce and install the flows to redirect the tunneled traffic.

When BGP converges, we have two possible ways for the hijacked traffic to return back to the protected $AS6_p$. If the traffic (a) is closer to the MOAS helper it will go through it as a relay and then be forwarded to the protected AS, or (b) if it is closer to the protected AS it will route to it directly.

In the sample topology the hijacked traffic that is closer to $AS1_m$ will go through it and then will be sent through the tunnel. If we check the forward path of a data plane packet from $AS3_t$, we will see that it goes like this: $AS3_t$ - AS2 - $AS1_m$ - AS2 - AS4 - $AS6_p$ and when it returns: $AS6_p$ - AS4 - AS2 - $AS3_t$. This means that because the packet goes through the MOAS helper and then tunneled to the $AS6_p$ it adds an additional delay on the forward path of the data plane packet but not on the return path, eventually acting as a relay AS only for the one-way traffic.

As seen in Table 5.1, we observe the difference of the packet delays before and after the MOAS mitigation. The additional delay that is added is the cost of the packet going through $AS1_m$ as a relay only for traffic that is closer to the MOAS helper. In our sample topology the additional delay is

the delay of adding two additional AS hops only for the forward path, which is around 20ms. Mitigating the hijack for the cost of some milliseconds can be considered acceptable.

	Forward Path	Forward Path Delay (ms)
Before	AS3 - AS2 - AS4 - AS6	32ms
After	AS3 - AS2 - Tunnel(AS1 - AS2 - AS4) - AS6	54ms
	Return Path	Return Path Delay (ms)
Before	AS6 - AS4 - AS2 - AS3	32ms
After	AS6 - AS4 - AS2 - AS3	32ms

Table 5.1: Packet delay difference when going through a MOAS tunnel.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

We implemented ARTEMIS$_{\text{ONOS}}$ as a modular application that has three services that can be run independently and uses two pre-existing applications of ONOS, SDN-IP and Reactive-Routing. We show that it successfully detects and mitigates BGP prefix hijacks of any prefix length for type-0 and type-1 hijacks.

The application was accepted by the ONOS technical steering team and is now already packaged and released on-par with ONOS. It is the first SDN application that battles with BGP hijacks and has received feedback by many researchers. A short introduction and demo was presented in ONOS Build 2017 in Seoul, Korea and a demo video is released that explains how it works [2].

With the results we gathered from our evaluation framework we showed that the application takes milliseconds to detect and initiate mitigation of the BGP hijack attack, while the overhead is the BGP convergence which has three times higher order of magnitude.

Also, using an SDN-enabled network instead of a legacy one, automates the traffic rerouting by using the intent framework and also significantly decreases the time it takes to (re-)configure the network devices.

6.2 Future Work

Despite the fact that legacy ARTEMIS was first tested on a real-world, non-SDN environment with the basic mitigation strategy of prefix de-aggregation in mind, our application can support several extensions related to its monitoring, detection and mitigation modules due to its highly modular design.

These extensions can be served as extra services built over the ONOS platform. We intend to implement the following extensions:

6.2.1 AS Graph Discovery

We intend to automate the network configuration generation, where the tool is set to an idle discovery state and instead of searching for BGP Hijacks, it discovers the AS-level topology for the specified prefixes. Also, it will be possible to populate this topology through BGP control-plane archives to speed-up the AS discovery. This will allow the simplification of the configuration process and the detection of advanced hijacks.

6.2.2 Dynamic Router Configuration

NETCONF[35] is a network management protocol that provides mechanisms to install, manipulate, and delete the configuration of network devices.

Support for dynamic configuration of devices (in our application the BGP speakers) is a new feature currently under development by the ONOS's Dynamic Configuration Brigade [17]. This will potentially support a large number of available routers and will enhance the capabilities of ARTEMIS$_{\text{ONOS}}$'s mitigation service.

6.2.3 PEERING Testbed

Following the footsteps of the initial ARTEMIS approach, we plan to employ the PEERING testbed [21] to announce prefixes on the real Internet using hybrid real-world and virtualized SDN setups.

The challenge with this approach is that we cannot install SDN software on PEERING Points of Presence (PoP). This means that we will need to have a local SDN setup that monitors the real Internet and outsources, if a hijack is detected, the mitigation to a PEERING PoP.

Appendix A

Network Configurations

A.1 ARTEMIS$_{\text{ONOS}}$ Configuration

This is a sample network configuration file for ONOS's ARTEMIS$_{\text{ONOS}}$ application.

```
"org.onosproject.artemis" : {
    "artemis": {
        "prefixes": [
            {
                "moas": [ ],
                "paths": [
                    {
                        "neighbor": [
                            {
                                "asn": 65002,
                                "neighbor": [
                                    65001
                                ]
                            }
                        ],
                        "origin": 65004
                    }
                ],
                "prefix": "40.0.0.0/8"
            }
        ]
        },
        "moas": { },
        "monitors": {
            "exabgp": [
                "192.168.1.2:5000"
```

```
            ],
            "ripe": []
        }
    }
}
```

Explanation of Fields

- **prefixes**: List consisting of prefixes with their AS-PATH information and (optionally) legitimate MOAS ASes.
  - **moas**: List of IPs that will be used to request help when an outsourced mitigation is required.
  - **prefix**: A CIDR representation of the prefix that is monitored/protected.
  - **paths**: List of dictionaries that contain the ASN of the protected AS (origin), along with a list of dictionaries for the neighbors.
  - **neighbor**: List of dictionaries that contain each neighbor's ASN and a list of ASNs for the neighbor's neighbor. For example, in the demo topology, the protected (origin) AS65004 sees the AS65002 as a first-hop neighbor, and AS65001 as a second-hop neighbor (resulting in the legitimate announced path AS65004 - AS65002 - AS65001).

    Note: While the operator can supply the origin and first-hop neighbor ASNs as ground-truth in the configuration, the N-hop ($N > 1$) neighbor information is planned to be generated automatically by ARTEMIS$_{\text{ONOS}}$ in future versions of the tool, based on the received BGP updates.

- **moas**: List of IPs that this service will accept outsourced mitigation requests from. It is used as a security check to validate the source of the mitigation request. In future work this will be changed to use asymmetric cryptography instead, to avoid IP spoofing attacks.

- **monitors**: List of the route collectors that ARTEMIS$_{\text{ONOS}}$ is using for monitoring. Currently it supports RIPE and ExaBGP route collectors through the `socket-io` interface, and is extendable to include more monitoring services/APIs.

  RIPE Route Collectors have specific identifiers ("rrc18", "rrc19", "rrc20", "rrc21"). You can configure them following this example:
  "ripe" : ["rrc18", "rrc19"].

  An ExaBGP Route Collector (RC) is implemented inside the tutorial folder (/onos/tools/tutorials/artemis/). You can host such an RC locally by running an ExaBGP instance with the exabgp.conf and

server.py files provided (the latter will require modifications in directory paths). In the demo topology we have an ExaBGP speaker running on 192.168.1.2:5000, monitoring the BGP control plane from the perspective of AS65001.

A.2 Reactive-Routing Configuration

This is a sample network configuration file for ONOS's Reactive-Routing application.

```
{
    "ports" : {
        "of:00000000000000a1/4" : {
            "interfaces" : [
                {
                    "ips"  : [ "201.0.0.254/24" ],
                    "mac"  : "00:00:00:00:00:01"
                }
            ]
        },
        "of:0000000000000a13/4" : {
            "interfaces" : [
                {
                    "ips"  : [ "213.0.0.200/24" ],
                    "mac"  : "00:00:00:00:00:01"
                }
            ]
        }
    },
    "apps" : {
       "org.onosproject.reactive.routing" : {
            "reactiveRouting" : {
                "ip4LocalPrefixes" : [
                    {
                        "ipPrefix" : "201.0.0.0/24",
                        "type" : "PUBLIC",
                        "gatewayIp" : "201.0.0.254"
                    }
                    {
                        "ipPrefix" : "213.0.0.0/24",
                        "type" : "PRIVATE",
                        "gatewayIp" : "213.0.0.254"
                    }
                ],
```

```
                "ip6LocalPrefixes" : [
                ],
                "virtualGatewayMacAddress" : "00:00:00:00:00:01"
            }
        }
    }
}
```

Explanation of Fields

- **ports**: Include configuration to match switch ports with an IP and a MAC address. Each port is identified with the OpenFlow ID of the switch and the port number.
- **apps/reactiveRouting/ip{4/6}LocalPrefixes**: List of all prefixes that exist inside our AS.
  - **ipPrefix**: A CIDR notation of the prefix owned.
  - **type**: Type of prefix, it can either be PUBLIC if it is reachable from the outside or PRIVATE if it is not.
  - **gatewayIP**: An IP that is the gateway IP of this specific prefix.
- **apps/reactiveRouting/virtualGatewayMacAddress**: A MAC address that is used to answer all ARP requests about gateway IPs of all IP prefixes.

A.3 SDN-IP Configuration

This is a sample network configuration file for ONOS's SDN-IP application.

```
"org.onosproject.router": {
            "bgp": {
                "bgpSpeakers" : [
                    {
                        "name" : "speaker1",
                        "connectPoint" : "of:00002a45d713e141/4",
                        "peers" : [
                            "150.1.3.1"
                        ]
                    }
                ]
            }
        }
```

Explanation of Fields

- **bgp/bgpSpeakers**: List of structures that correspond to each BGP speaker
  - **name**: An alias for the specific BGP speaker.
  - **connectPoint**: The Openflow ID of the switch and the port that this BGP speaker is connected to.
  - **peers**: List of IPs that this BGP speaker is peering with.

A.4 ExaBGP control-plane monitor

We implemented a lightweight docker container that contains an EXABGP control-plane monitor which connects with a BGP Speaker and has a Socket.IO interface to propagate all control-plane messages.

The operator may initialize the monitor by providing the local IP address, the IP address of the BGP Speaker and the AS Number through environment variables. Then he can subscribe to the monitor by connecting on port 5000 as a socket.IO client and send a subscribe message.

To Subscribe to the ExaBGP monitor:

```
Event: exa_subscribe
Message: prefix as string (e.g. '10.0.0.0/8')
```

Messages sent from ExaBGP monitor to the subscriber:

```
Event: exa_message
Message:
    'type': Type of the BGP message (Now it only sends BGP Update messages;
    you can change server.py to send also withdraw messages)
    'timestamp': Timestamp
    'peer': Peer of the BGP message
    'host': String identifier of the monitor (default 'exabgp')
    'path': BGP Update AS Path
    'prefix': Prefix that corresponds to the BGP Upate message
```

Appendix B

ONF Internship

As a part of my internship at Open Networking Foundation, except the work on ARTEMIS$_{\text{ONOS}}$, I was a member of multiple brigades focusing on ONOS and M-CORD.

B.1 M-CORD

M-CORD is an open source reference solution for carriers deploying mobile wireless networks. It is built on SDN and NFV technologies while it includes both virtualization of RAN functions and a virtualized mobile core (vEPC).

B.1.1 eXtensible Radio Access Networks (xRAN)

I was the lead developer and brigade lead for the xRAN project (as ONOS community continued to grow over time, ONOS formed brigades which are teams that coordinate to achieve a shared goal). xRAN organization standardized southbound and northbound interfaces in order to SDN'ize the RAN architectures. I designed and implemented these interfaces on top of ONOS but also the controller logic. The end-product demo was presented in MWC Americas 2017 in San Francisco with a custom eNB developed by Radisys.

B.1.2 ONOS in-between Control and User plane of vEPC

INTEL implemented a VNF that merges Service Gateway and Packet Gateway of RAN while decoupling the control plane and user plane. I introduced an SDN application that lays in-between the two planes which provides an API to control and monitor each plane but also add any other implementation for the data-plane. We presented at MWC 2018 in Barcelona where we replaced the data plane implementation of INTEL with a P4-based programmable fabric and open source EPC.

B.2 ONOS

On the side of ONOS, I implemented ARTEMIS$_{\text{ONOS}}$ but also was part of other brigades and contributed fixes and patches to some bugs.

B.2.1 ISSU

I was a member and contributor of In-Service Software Upgrade brigade for ONOS. We implemented a new protocol, following the same logic as CISCO routers, that would support an in-service upgrade. I focused primarily on patching primary-backup and RAFT protocols that were implemented through atomix library.

B.2.2 Other Bugs & Futures

Finding bugs was a daily event and with the help of the brigades we fixed/patched a lot of them. Most of them that I came across and resolved were in services such as REST, RESTCONF, SDN-IP, Reactive-Routing, BGP-Router, etc.

Bibliography

[1] Apache karaf. https://karaf.apache.org/.

[2] ARTEMIS ONOS Demo. https://www.youtube.com/watch?v=UouzKz8sUFw.

[3] ARTEMIS ONOS Wiki. https://wiki.onosproject.org/display/ONOS/ARTEMIS%3A+an+Automated+System+against+BGP+Prefix+Hijacking.

[4] BGPmon (Colorado State University). http://www.bgpmon.io.

[5] BGPmon (commercial). http://www.bgpmon.net.

[6] Can SDN accelerate BGP convergence? a performance analysis of inter-domain routing centralization, author=Sermpezis, Pavlos and Dimitropoulos, Xenofontas, journal=arXiv preprint arXiv:1702.00188, year=2017.

[7] Chinese isp hijacks the internet. http://www.bgpmon.net/chinese-isp-hijacked-10-of-the-internet/.

[8] CIDR REPORT for 18 Jun 18. http://www.cidr-report.org/as2.0/.

[9] Containers vs VMs: Which is better for cloud deployments? https://www.sdxcentral.com/cloud/containers/definitions/containers-vs-vms/.

[10] CORD (Central Office Re-architected as a Datacenter): reinventing central offices for efficiency and agility. http://opencord.org/.

[11] exabgp: The BGP swiss army knife of networking. https://github.com/Exa-Networks/exabgp.

[12] GNS3. https://gns3.com/.

[13] Hacker Redirects Traffic from 19 Internet providers to steal Bitcoins. https://www.wired.com/2014/08/isp-bitcoin-theft/.

[14] Looking Glass API documentation. http://www.caida.org/tools/utilities/looking-glass-api/.

[15] Mininet. http://mininet.org/.

[16] ONOS Deployments. https://wiki.onosproject.org/display/ONOS/Deployments/.

[17] ONOS: Dynamic Configuration Brigade. https://wiki.onosproject.org/display/ONOS/Dynamic+configuration+brigade/.

[18] ONOS Intent Framework. https://wiki.onosproject.org/display/ONOS/Intent+Framework.

[19] Open Networking Operating System (ONOS). http://onosproject.org/.

[20] Open vSwitch. http://openvswitch.org/.

[21] The PEERING testbed. https://peering.usc.edu.

[22] Quagga. http://www.nongnu.org/quagga/.

[23] RIPE RIS. http://www.ris.ripe.net/.

[24] The Route Views Project. http://www.routeviews.org/.

[25] SDN Resources: Understanding the SDN Architecture and SDN Control Plane. https://www.sdxcentral.com/sdn/definitions/inside-sdn-architecture/.

[26] SDN series part seven: ONOS. https://thenewstack.io/open-source-sdn-controllers-part-vii-onos/.

[27] xRAN.org. http://www.xran.org/.

[28] Youtube hijacking: A ripe ncc ris case study. https://www.ripe.net/publications/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study.

[29] Karan Balu, Miguel L Pardal, and Miguel Correia. DARSHANA: Detecting route hijacking for communication confidentiality. In *Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on*, pages 52–59. IEEE, 2016.

[30] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al. Onos: towards an open, distributed sdn os. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6. ACM, 2014.

[31] Kevin Butler, Toni R Farley, Patrick McDaniel, and Jennifer Rexford. A survey of BGP security issues and solutions. *Proceedings of the IEEE*, 98(1):100–122, 2010.

[32] Kenneth K Chan, Philip W Hartmann, Scott P Lamons, Terry G Lyons, and Argyrios C Milonas. Virtual local area network, April 18 1989. US Patent 4,823,338.

[33] Gavriil Chaviaras, Petros Gigis, Pavlos Sermpezis, and Xenofontas Dimitropoulos. ARTEMIS: Real-time Detection and Automatic Mitigation for BGP Prefix Hijacking. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 625–626. ACM, 2016.

[34] Alan Elder and Jonathan Harrison. Spanning tree protocol, October 1 2015. US Patent App. 14/673,652.

[35] R Enns, M Bjorklund, J Schoenwaelder, et al. IETF RFC 6241: Network Configuration Protocol (NETCONF).

[36] Dimitrios Gkounis, Vasileios Kotronis, and Xenofontas Dimitropoulos. Towards defeating the crossfire attack using sdn. *arXiv preprint arXiv:1412.2013*, 2014.

[37] Arpit Gupta, Laurent Vanbever, Muhammad Shahbaz, Sean P Donovan, Brandon Schlinker, Nick Feamster, Jennifer Rexford, Scott Shenker, Russ Clark, and Ethan Katz-Bassett. SDX: A software defined internet exchange. *ACM SIGCOMM Computer Communication Review*, 44(4):551–562, 2015.

[38] Alexander JT Gurney, Andreas Haeberlen, Wenchao Zhou, Micah Sherr, and Boon Thau Loo. Having your cake and eating it too: Routing security with privacy protections. In *Proceedings of the 10th ACM workshop on hot topics in networks*, page 15. ACM, 2011.

[39] Andreas Haeberlen, Ioannis C Avramopoulos, Jennifer Rexford, and Peter Druschel. Netreview: Detecting when interdomain routing goes wrong. In *NSDI*, volume 2009, pages 437–452, 2009.

[40] Rahul Hiran, Niklas Carlsson, and Nahid Shahmehri. PrefiSec: A distributed alliance framework for collaborative BGP monitoring and prefix-based security. In *Proceedings of the 2014 ACM Workshop on Information Sharing & Collaborative Security*, pages 3–12. ACM, 2014.

[41] Rahul Hiran, Niklas Carlsson, and Nahid Shahmehri. Crowd-based detection of routing anomalies on the internet. In *Communications and Network Security (CNS), 2015 IEEE Conference on*, pages 388–396. IEEE, 2015.

[42] Rahul Hiran, Niklas Carlsson, and Nahid Shahmehri. Collaborative framework for protection against attacks targeting BGP and edge networks. *Computer Networks*, 122:120–137, 2017.

[43] Xin Hu and Z Morley Mao. Accurate real-time identification of ip prefix hijacking. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 3–17. IEEE, 2007.

[44] Josh Karlin, Stephanie Forrest, and Jennifer Rexford. Pretty good BGP: Improving BGP by cautiously adopting routes. In *Network Protocols, 2006. ICNP'06. Proceedings of the 2006 14th IEEE International Conference on*, pages 290–299. IEEE, 2006.

[45] Stephen Kent, Charles Lynn, and Karen Seo. Secure border gateway protocol (s-bgp). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, 2000.

[46] Vasileios Kotronis, Adrian Gämperli, and Xenofontas Dimitropoulos. Routing centralization across domains via SDN: A model and emulation framework for BGP evolution. *Computer Networks*, 92:227–239, 2015.

[47] Vasileios Kotronis, Rowan Klöti, Matthias Rost, Panagiotis Georgopoulos, Bernhard Ager, Stefan Schmid, and Xenofontas Dimitropoulos. Stitching inter-domain paths over IXPs. In *Proceedings of the Symposium on SDN Research*, page 17. ACM, 2016.

[48] Mohit Lad, Daniel Massey, Dan Pei, Yiguo Wu, Beichuan Zhang, and Lixia Zhang. PHAS: A Prefix Hijack Alert System. In *Usenix Security*, 2006.

[49] Matt Lepinski, Richard Barnes, and Stephen Kent. An infrastructure to support secure internet routing. 2012.

[50] Matt Lepinski and K Sriram. BGPSEC protocol specification. Technical report, 2017.

[51] Pingping Lin, Jonathan Hart, Umesh Krishnaswamy, Tetsuya Murakami, Masayoshi Kobayashi, Ali Al-Shabibi, Kuang-Ching Wang, and Jun Bi. Seamless interworking of SDN and IP. page 2.

[52] Pingping Lin, Jonathan Hart, Umesh Krishnaswamy, Tetsuya Murakami, Masayoshi Kobayashi, Ali Al-Shabibi, Kuang-Ching Wang, and Jun Bi. Seamless interworking of SDN and IP. In *ACM SIGCOMM computer communication review*, volume 43, pages 475–476. ACM, 2013.

[53] Shishir Nagaraja, Virajith Jalaparti, Matthew Caesar, and Nikita Borisov. P3CA: Private anomaly detection across ISP networks. In

International Symposium on Privacy Enhancing Technologies Symposium, pages 38–56. Springer, 2011.

[54] Bruno Astuto A Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, 16(3):1617–1634, 2014.

[55] David Plummer et al. An Ethernet address resolution protocol (RFC 826). *Network Working Group*, 1982.

[56] Jon Postel et al. RFC 791: Internet protocol, 1981.

[57] Jian Qiu, Lixin Gao, Supranamaya Ranjan, and Antonio Nucci. Detecting bogus BGP route information: Going beyond prefix hijacking. In *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on*, pages 381–390. IEEE, 2007.

[58] Sophie Y Qiu, Fabian Monrose, Andreas Terzis, and Patrick D McDaniel. Efficient techniques for detecting false origin advertisements in inter-domain routing. In *Secure Network Protocols, 2006. 2nd IEEE Workshop on*, pages 12–19. IEEE, 2006.

[59] Tongqing Qiu, Lusheng Ji, Dan Pei, Jia Wang, and Jun Xu. Towerdefense: Deployment strategies for battling against ip prefix hijacking. In *Proc. IEEE ICNP*, pages 134–143, 2010.

[60] Lin Quan, John Heidemann, and Yuri Pradkin. Trinocular: Understanding internet reliability through adaptive probing. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 255–266. ACM, 2013.

[61] Y Rekhter, T Li, and S Hares. IETF RFC 4271: A Border Gateway Protocol 4 (BGP-4). *Reston: Internet Society*, 2006.

[62] Johann Schlamp, Ralph Holz, Quentin Jacquemart, Georg Carle, and Ernst Biersack. HEAP: Reliable Assessment of BGP Hijacking Attacks. *IEEE JSAC*, 34(06):1849–1861, 2016.

[63] JD Case M Fedor ML Schoffstall and C Davin. RFC 1157: Simple network management protocol (SNMP). *IETF, April*, 1990.

[64] Pavlos Sermpezis, Vasileios Kotronis, Alberto Dainotti, and Xenofontas Dimitropoulos. A Survey among Network Operators on BGP Prefix Hijacking. *arXiv preprint arXiv:1801.02918*, 2018.

[65] Pavlos Sermpezis, Vasileios Kotronis, Petros Gigis, Xenofontas Dimitropoulos, Danilo Cicalese, Alistair King, and Alberto Dainotti. ARTEMIS: Neutralizing BGP Hijacking within a Minute. *arXiv preprint arXiv:1801.01085*, 2018.

[66] Xingang Shi, Yang Xiang, Zhiliang Wang, Xia Yin, and Jianping Wu. Detecting prefix hijackings in the Internet with Argus. In *Proc. ACM IMC*, 2012.

[67] Seungwon Shin, Lei Xu, Sungmin Hong, and Guofei Gu. Enhancing network security through software defined networking (SDN). In *Computer Communication and Networks (ICCCN), 2016 25th International Conference on*, pages 1–9. IEEE, 2016.

[68] Georgos Siganos and Michalis Faloutsos. Neighborhood watch for internet routing: Can we improve the robustness of internet routing today? In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1271–1279. IEEE, 2007.

[69] Ion Stoica, Robert Morris, David Liben-Nowell, David R Karger, M Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking (TON)*, 11(1):17–32, 2003.

[70] Lakshminarayanan Subramanian, Volker Roth, Ion Stoica, Scott Shenker, and Randy Katz. Listen and whisper: Security mechanisms for bgp. In *Proc. NSDI*, 2004.

[71] ONF TS. OpenFlow switch specification. page 206.

[72] Ángel Leonardo Valdivieso Caraguay, Alberto Benito Peral, Lorena Isabel Barona López, and Luis Javier García Villalba. SDN: Evolution and opportunities in the development IoT applications. 10(5):735142.

[73] Haopei Wang, Lei Xu, and Guofei Gu. Floodguard: A dos attack prevention extension in software-defined networks. In *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*, pages 239–250. IEEE, 2015.

[74] Zheng Zhang, Ying Zhang, Y Charlie Hu, Z Morley Mao, and Randy Bush. iSPY: detecting ip prefix hijacking on my own. In *ACM SIGCOMM CCR*, volume 38, pages 327–338, 2008.

[75] Mingchen Zhao, Wenchao Zhou, Alexander JT Gurney, Andreas Haeberlen, Micah Sherr, and Boon Thau Loo. Private and verifiable interdomain routing decisions. *ACM SIGCOMM Computer Communication Review*, 42(4):383–394, 2012.

[76] Changxi Zheng, Lusheng Ji, Dan Pei, Jia Wang, and Paul Francis. A light-weight distributed scheme for detecting ip prefix hijacks in real-time. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 277–288, 2007.