Securing BGP over Software Defined Networks

by *Rubén Hervás Fernández*
Supervisor: Marcelo Yannuzzi
Co-supervisor: René Serral Gracia

December 3, 2014

Acknowledgment I want to specially thank René Serral, Marcelo Yannuzzi, Genís Riera and Manos for the help in developing this project.

Abstract A causa de a la mancança de seguretat en el protocol BGP al llarg dels anys hi ha hagut diferents propostes per intentar millorar la seguretat al intercanviar paquets entre sistemes autònoms. Aquestes solucions no han pogut ser portades a terme degut a l'altíssim cost d'introduir-les a la xarxa o a la limitació de capacitat de computació dels dispositius. Amb l'aparició de les Software Defined Networks(SDN) han aparegut noves possibilitats de gestionar els dispositius de xarxes i ha obert una possibilitat a interactuar amb aquests dispositius d'una manera nova. En aquest document hem aprofitat aquest nou paradigma per crear una solució al problema de seguretat de BGP tot utilitzant la plataforma SDN the Cisco, OnePK. La solució proposada té la característica que no cal cap gran canvi a la xarxa ni canviar els dispositius. A més s'ha creat un entorn de proves per demostrar que la solució era possible.

Abstract Due to the lack of security in the BGP protocol over the years there have been various proposals trying to improve security in exchanging packets between Autonomous Systems. These solutions could not be carried out due to the high cost of deploying to the network or due to the limited computing capacity of the devices. With the advent of Software Defined Networks (SDN) new possibilities have emerged for managing networks and devices. This fact has created the opportunity to interact with these devices in whole different way. In this paper use the SDN paradigm to improve the security of BGP based on the Cisco SDN platform, OnePK. The proposed solution has the characteristic that do not need big changes to the network or the devices. A test environment has been created to prove that the feasibility of the solution.

Contents

Chapter 1

Introduction

Nowadays the Internet main edge routing protocol is BGP, the main protocol responsible for routing the traffic between the ASs. Although it performs such an important task, BGP was designed to use a mutual trust mechanism when exchanging routes. For that reason many solutions have been proposed in order to improve the security of BGP. Although there are proposed very interesting ideas to solve this problem they are not viable due to vast expense in changing the core network or high processing power needed in devices that is not available. With the appearance of SDN networks a new way of managing network devices has been discovered. Using this idea we propose an SDN architecture that places the necessary security actions on a Server controller, thus not consuming resources of the network devices. Moreover, the only assumption we make for the devices is that they are SDN-enabled.

We have implement this architecture using OnePK[1], the Cisco SDN platform, combined with a Server application that is able to secure BGP without any need to change the network devices. As the system is developed over OnePK, and with respect to the percentage of Cisco routers in the core network[1] we believe that our system can be a viable solution in the future as there would be no need to apply changes to the devices.

The solution is based in taking out the decision process of BGP to a Server controller that interacts with OnePK applications. This applications are interacting with the device and they give the information needed by the Server. Besides the functionalities shown in this project there is still a lot of work to do with this types of systems as they offer an interesting new way of dealing with network devices and network management.

[1] `http://www.cn-c114.net/577/a692668.html`

1.1 Motivations

The main motivation of this project is to propose an extensible and efficient mechanism to offload the security provisioning of the legacy Border Gateway Protocol (BGP) protocol that is used widely in the Internet. One of the key factors of the solution is that it will neither modify nor extend the legacy BGP protocol. To achieve this, we propose a Software Defined Network (SDN) solution. Such solution intercepts the BGP packets and transparently assessing their validity of the BGP packets and performing the necessary actions, i.e., drop them or accept them.

This solution is implemented on top of One Platform Kit (OnePK), a Cisco platform that is still under development. To complete the solution a web user interface has been created to provide the user with an easy way to configure prefix rules and Autonomous System (AS) Path rules to any node in a managed network. By now the solution has some limitations inherent to the Application Programming Interface (API) provided by OnePK. Despite these limitations a testbed has been deployed where the correctness of the system has been proved as well as performance experiments have been performed.

1.2 Goals

The main idea of the solution is to add security to the BGP protocol but without modifying the protocol itself. In order to achieve this goal we have used a SDN platform, OnePK. Using a SDN platform we want to create a BGP route validation system without any interference in the regular BGP functioning. Additionally, the validation process is intended to provide a user friendly interface to provide a network administrator with a set of components for creating prefix rules and AS path rules and attach them to a specific node of the network. Furthermore the user interface will be able to give a clear view of the network and the user will be able to access it in a comfortable way.

1.3 Planification

For the execution of this project there were many challenges but one of them has been the most critical point of this project as without it the execution of the project was not possible. This challenge is the inability of OnePK to intercept packets before they are processed by the Border Gateway Protocol (BGP) process of a router or before they enter the Routing Information Base (RIB) of the router.When this project started the tool used for testing this feature was the All in One Virtual Machine (VM) that Cisco provides. The version of OnePK was 0.9, so at very early stages and after many tests and study of the platform we concluded that it was not

possible to intercept BGP packets with the APIs that were provided in the 0.9 release. But as the project was evolving new versions of OnePK were released and at the 1.3 version that was released the 19th of September 2014. In that version, new OnePK APIs were provided that gave the ability to manage the packets before they entered the BGP process. Nevertheless OnePK still does not provide a BGP API. We faced this challenge implementing a workaround that is explained in 4.4 section. Very recently it was announced that the next All in One VM release will contain the OnePK BGP API, so it is possible that our workaround is no more necessary.

1.3.1 Gantt chart

Figure 1.1 presents the evolution of the project. As it can bee seen the Packet Interception task was the blocker of all the project, without which the rest of the tasks could not be initiated since it was unknown whether they were required.

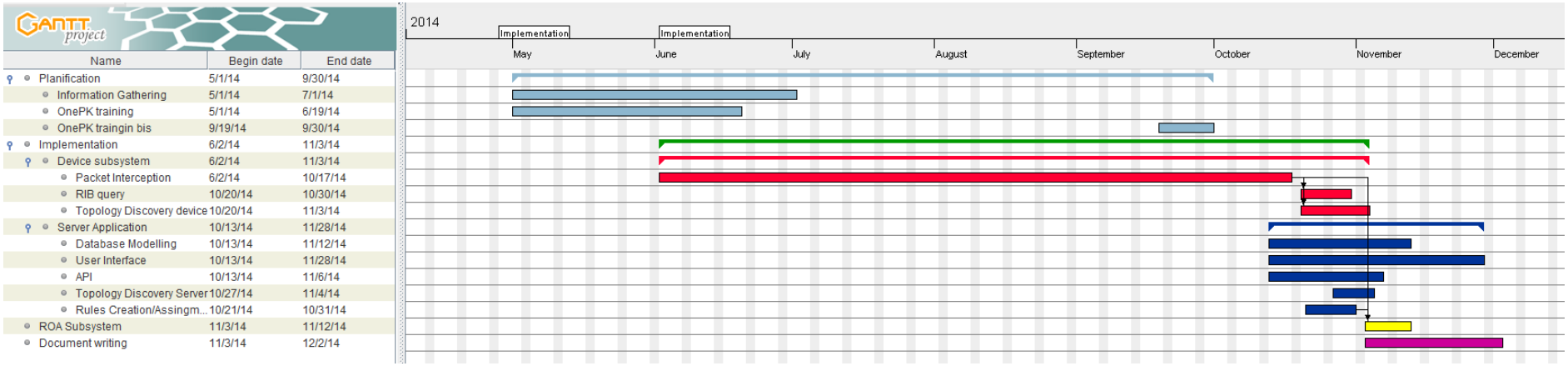

Figure 1.1: Gantt Chart

Chapter 2

Background

2.1 BGP

In this section an explanation of the BGP protocol is provided. To ensure better understanding of the problem solved in this document, we also present a brief introduction to the protocol and the way it functions.

2.1.1 History

BGP is one of the most important and most used protocols over the Internet. BGP is a path vector protocol used in order to exchanges route between autonomous systems. BGP was first standardized in 1989, originally defined in Request for Comments (RFC) 1105[2]. The current version, BGPv4, was adopted in 1995 and is defined in RFC 4271[3]. From now on in this document BGP will be referring to BGPv4.

BGP makes routing decisions using paths, policies and rules defined by the network administrator and the information that receives from the BGP neighbours. BGP has shown that is scalable and robust enough to work in big networks.

2.1.2 How it Functions

In order to understand why BGP is broadly used in the Internet an overview of how the Internet is organized needs to be presented.

Internet is an interconnected set of networks that exchange routes with each other so that every

network is reachable by the other networks. Because of the vast size of the Internet nowadays this simple fact implies that there are thousands of networks interconnected with each other. Networks are managed by different companies or governments which need to exchange the routes they have, or know, in order for the Internet to work properly. To provide the Internet with such capability there are different ways that the networks are interconnected. The first entities created to achieve that were the Network Access Points (NAPs), which are public network exchange facilities where Internet Service Providers (ISPs) connect to each other. The exchange of routes between different countries or ISPs takes place in the NAPs, although it is not the only place where this can be done. As the Internet has been growing some ISPs decided to directly interconnect their networks, since NAPs were getting congested. ISPs usually create direct interconnections in case where both ISPs are economically benefited.

Bearing this overview of how Internet works in mind there are two ways of route exchanges necessary:

1. Intradomain Routing : This routing is done between the routers of the same ISP which usually using the same routing protocol. This can be done using Interior Border Gateway Protocol (iBGP), Open Shortest Path First (OSPF) or the protocol that the ISP chooses.

2. Interdomain Routing : This routing takes place between two different autonomous systems from different organizations(ISP, Governments). The main Exterior Gateway Protocol (EGP) used currently is BGP. In this type of routing more complex policies and interests are taken into account as this can lead to some unpleasant situations for the ISP, which as a company intents to provide their customers with the best service they can offer while at the same time being profitable .

In figure 2.1 a graphical representation of how the Internet from a BGP perspective can be seen.

BGP neighbours, which are called peers, create a Transmission Control Protocol (TCP) connection using port number 179 in order to communicate with each other.TCP ensure features like the reliability of the connection, retransmission of the packets, thus BGP does not need to take care of such issues. As mentioned previously, in the exchange of routes between different ASs there many interests of advertising or not certain routes to another AS. If route exchange is not done carefully it may lead to a congestion of the network of some ISP as it may be routing traffic which is supposed not to be routed trough this network. This has been a very controversial issue as it can lead to failures in the ISP network or even more dramatical failures, making part of the Internet unreachable for the ISP customers.

Another concept important for the understanding of the routes exchange complexity between AS is the that of transit traffic. This traffic comes from other ASs and goes to other ASs. So neither the source nor the destination is inside the AS is traversing. For an ISP this traffic can incur in extra cost of infrastructure maintenance without any benefit in the transit traffic is not

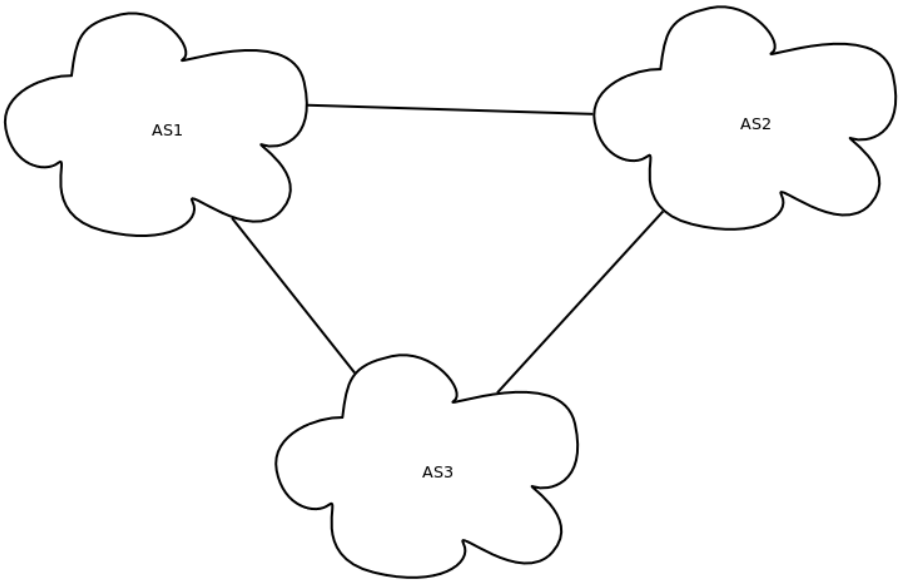

Figure 2.1: Interconnected Autonomous Systems

controlled and previously accorded with the other ISPs. In figure 2.2 a non-transit AS scenario is shown where the ASs does not advertise the networks learned from the BGP neighbours.

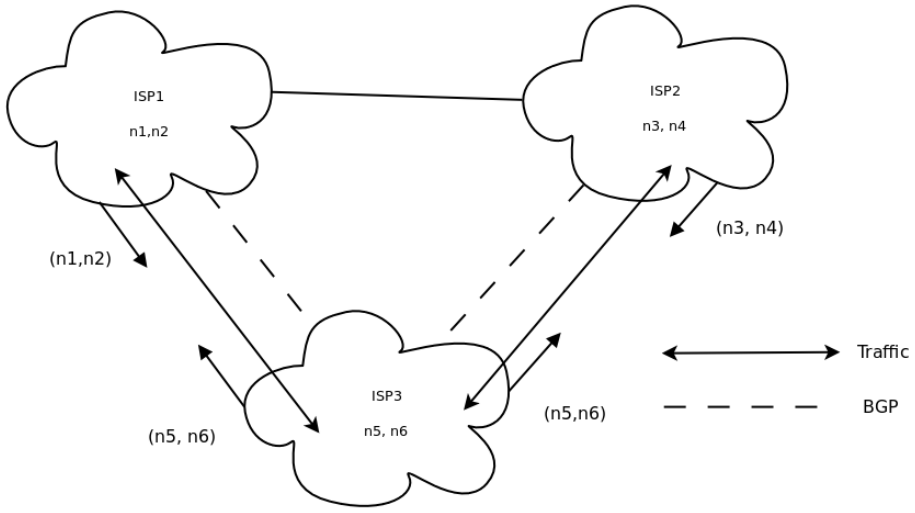

Figure 2.2: Nontransit Autonomous system

Figure 2.3 presents a transit AS scenario where the networks learned from the BGP neighbours are advertised.

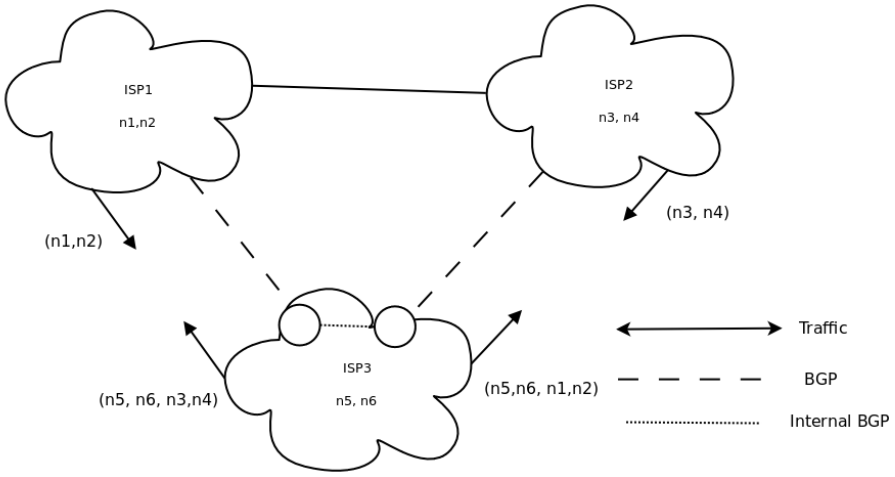

Figure 2.3: Transit Autonomous system

2.1.3 Vulnerabilities

In this section we provide a list of all the attacks that BGP is vulnerable to. This list is taken from [4].

1. Confidentiality Violations: The routing data carried in BGP is carried in cleartext, so eavesdropping is a possible attack against routing data confidentiality. (Routing data confidentiality is not a common requirement.)

2. Replay: BGP does not provide replay protection of its messages.

3. Message Insertion: BGP does not provide protection against insertion of messages. However, because BGP uses TCP, when the connection is fully established, message insertion by an outsider would require accurate sequence number prediction (not entirely out of the question, but more difficult with mature TCP implementations) or session-stealing attacks.

4. Message Deletion: BGP does not provide protection against deletion of messages. Again, this attack is more difficult against a mature TCP implementation, but is not entirely out of the question.

5. Message Modification: BGP does not provide protection against modification of messages. A modification that was syntactically correct and did not change the length of the TCP payload would in general not be detectable.

6. Man-In-The-Middle: BGP does not provide protection against man- in-the-middle attacks. As BGP does not perform peer entity authentication, a man-in-the-middle attack is a piece of cake.

7. Denial Of Service: While bogus routing data can present a denial of service attack on the end systems that are trying to transmit data through the network and on the network infrastructure itself, certain bogus information can represent a denial of service on the BGP routing protocol. For example, advertising large numbers of more specific routes (i.e., longer prefixes) can cause BGP traffic and router table size to increase, even explode.

2.1.4 Decision Process Summary

As explained in [5] the BGP decision process consists of set of steps when choosing the best route to send the traffic. This explanation is tied to the BGP Cisco implementation which may differ from another BGP implementation:

1. If the next hop is inaccessible, the route is ignored. (This is why it is important to have an Interior Gateway Protocol (IGP) route to the next hop.)
2. Prefer the path with the largest weight (weight is a Cisco proprietary parameter, local to the router).
3. If the weights are the same, prefer the route with the largest local preference value.
4. If there are no locally originated routes and the local preference is the same, prefer the route with the shortest AS_PATH.
5. If the AS_PATH length is the same, prefer the route with the lowest origin type (where IGP is lower than EGP and EGP is lower than INCOMPLETE).
6. If the origin type is the same, prefer the route with the lowest MED value if the routes were received from the same AS (or if BGP always-compare-med is enabled).
7. If the routes have the same MED value, prefer Exterior Border Gateway Protocol (EBGP) paths to iBGP paths.
8. If all the preceding scenarios are identical, prefer the route that can be reached via the closest IGP neighbour-that is, take the shortest internal path inside the AS to reach the destination. (Follow the shortest path to the BGP NEXT_HOP.)
9. If the internal path is the same, the BGP ROUTER_ID will be a tiebreaker. Prefer the route coming from the BGP router with the lowest RID. With Cisco IOS, the RID is the loopback address if one is configured; otherwise, it's the highest Internet Protocol (IP) address on the router. RID determination is vendor-specific.

2.1.5 Packet Types

BGP has different types of packet that are used for different purposes. In this section the BGP packet types of packet are explained. For further information the reader can refer to [6].

1. OPEN message : when the TCP connection is established this is the first message that the BGP speaker sends to the BGP neighbors. In Figure 2.4 the format of an OPEN message is shown.

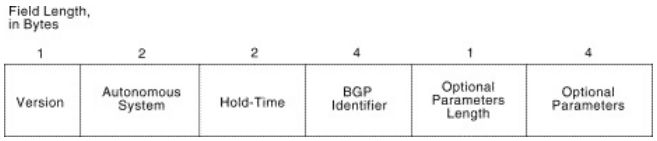

Figure 2.4: Open packet format

2. UPDATE message: This type of messages are used for exchanging prefixes and routes between BGP neighbors. UPDATE messages are used to advertise routes or to withdraw routes. An UPDATE message could update and withdraw routes simultaneously. Figure 2.5 show the fields that an UPDATE message can have and its format.

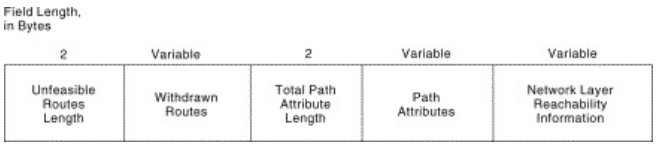

Figure 2.5: Update packet format

The minimum length of an UPDATE message is 23 bytes, 19 bytes of header plus 2 byte for the withdrawn routes length plus 2 bytes for the total path attribute length.

UPDATE messages can advertise one set of path attributes at most to multiple destinations. All path attributes contained in a given UPDATE message apply to all destinations carried in the Network Layer Reachability Information field of the UPDATE message.

An UPDATE message can list multiple routes to withdrawn from service. Each route is identified by its destination(IP Prefix).

If the message only advertise routes to withdrawn the message will not include path attributes or Network Layer Reachability Information. contrariwise, it may advertise only a feasible route, in which case the WITHDRAWN ROUTES field need not be present.

An UPDATE message should not include the same address prefix in the withdrawn routes and Network Layer Reachability Information fields. However, a BGP speaker must be able to process UPDATE messages in this form. A BGP speaker should treat an UPDATE message of this form as though the withdrawn routes do not contain the address prefix.

3. KEEPALIVE message : BGP does not use TCP keepalive messages. Instead of that KEEPALIVE messages are exchanged often enough. The time interval of these messages is negotiated between the two peers. The length of the message is 19 bytes and the format is this, figure 2.6 shows the format of a KEEPALIVE message:

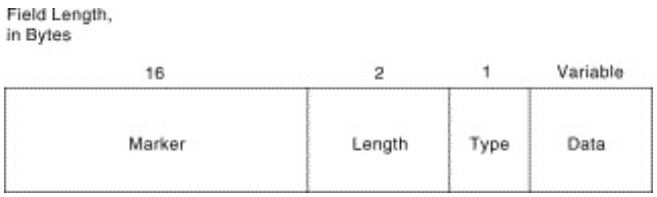

Figure 2.6: Keepalive packet format

In the case of the KEEPALIVE messages the data field is empty.

4. NOTIFICATION message : A NOTIFICATION message is sent when a error is produced and the BGP session is closed immediately afterwards. Figure 2.7 shows the format of a NOTIFICATION message:

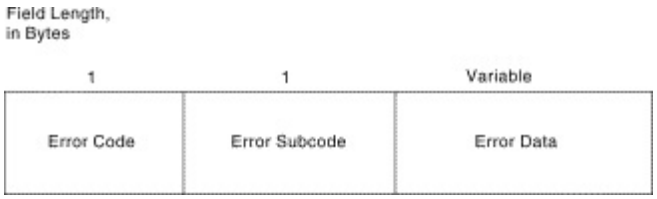

Figure 2.7:

2.1.6 Finite State Machine

The establishment of a BGP session is done with a set of steps that are graphically illustrated in 2.8. We now present a brief explanation of each step:

1. *Idle* : In this state when the BGP session initializes all resources and it is not accepting BGP connections.

2. *Connect* : In this state the BGP process is waiting for a TCP connection to move on to *OpenSent* state.

3. *Active* : This is a transition state. If the connection in the state *Connect* was unsuccessful the Connect timeout will be reset and will go to the *Connect* state again.

4. OpenSent : This is the state of the state machine once the TCP connection has been successful. The router sends an OPEN message and waits for one in return in order to transition to the *OpenConfirm* state.

5. Established : Once the router receives the previous states has finished successfully and it receives the confirmation of the *OpenConfirm* State the router goes to this state. In this state the router can receive and send *Keepalive*, *Update* and *Notifications* from peer to peer.

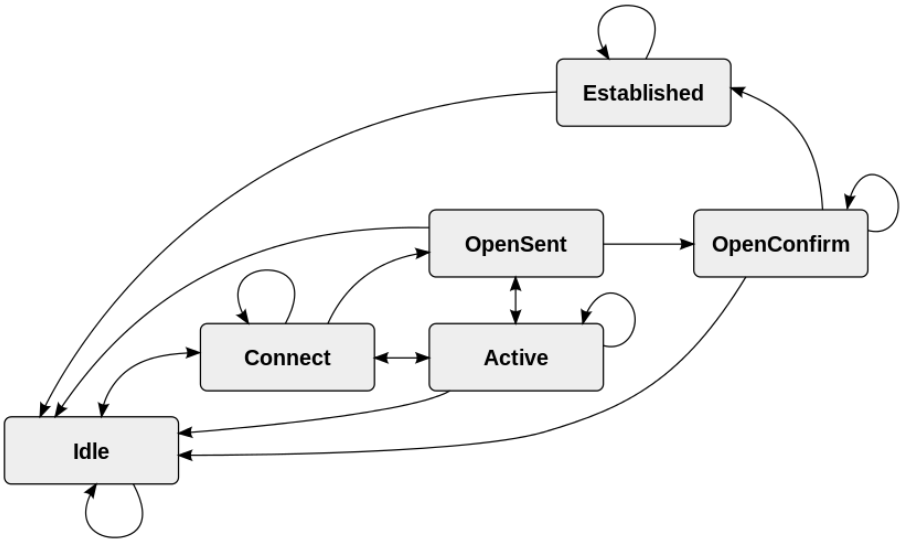

Figure 2.8: BGP state machine

2.2 ROA and RPKI

2.2.1 RPKI

As explained in [7], Resource Public Key Infrastructure (RPKI), also known as Resource Certification is a system that uses public key infrastructure designed to secure the Internet's routing infrastructure.

This system allows Local Internet Registries to request a a digital certificate listing the Internet number resources they hold. RPKI allows to map resource information. In this document we use the AS number and Prefix information RPKI can announce. With this system it is possible to validate the origin of an announced prefix. In particular, RPKI is used to secure BGP through BGP Secure (BGPSEC), as well as Network Discovery Protocol (NDP) for Internet

Protocol version 6 (IPv6) through the Secure Neighbour Discovery Protocol (SEND). Work on standardizing RPKI is currently (late 2011) ongoing at the Internet Engineering Task Force (IETF)[1] in the Secure Inter-Domain Routing (SIDR) working group, based on a threat analysis which was documented in RFC 4593[8]. The standards cover BGP origin validation, while work on path validation is underway. Several implementations for the prefix origin validation already exist.

2.2.2 ROA

A Route Origination Authorization (ROA) states which AS is authorised to publish certain IP prefixes. In addition, it can also determine the maximum length of the prefix that the AS is authorised to advertise.

Maximum Prefix Length The maximum prefix length is an optional ROA field. When not defined, the AS is only authorised to advertise exactly the prefix specified. Any announcement of different prefix (less or more specific) will be considered invalid. This is a way to enforce aggregation and prevent hijacking through the announcement of a more specific prefix. When present, this specifies the length of the most specific IP prefix that the AS is authorised to advertise. For example, if the IP address prefix is 10.0/16 and the maximum length is 22, the AS is authorised to advertise any prefix under 10.0/16, as long as it is no more specific than /22. In this specific example, the AS would be authorised to advertise 10.0/16, 10.0.128/20 or 10.0.252/22, but not 10.0.255.0/24.

RPKI Route Announcement Validity When a ROA is created for a certain combination of origin AS and prefix, this will have an effect on the RPKI validity of one or more route announcements. An announcement can be:

1. Valid : The route announcement is covered by at least one ROA.
2. Invalid : The prefix is announced from an unauthorised AS. This means:
   (a) there is a ROA for this prefix for another AS, but no ROA authorising this AS; or
   (b) this could be a hijacking attempt; or
   (c) the announcement is more specific than is allowed by the maximum length set in a ROA that matches the prefix and AS
3. Unkown : The prefix in this announcement is not covered (or only partially covered) by an existing ROA.

[1] `https://www.ietf.org/`

2.3 Software Defined Networks

Software Defined Networking (SDN) is an approach to networking that aims to introduce abstractions for Networking that were missing for a long time. The main idea is that a network should be seen as a set of resources. SDN proposes that all network components should be translated into a set of resources reachable by a defined API. That way, layers of abstraction can be introduced, simplifying the life of network researchers and developers. This approach tries to introduce in the Networking field the abstraction techniques used in many Computer Science fields, like Computer Architecture or Operating Systems. The most prominent application of SDN currently, is Network Virtualization, since virtualization is strongly connected with the idea of abstraction. To this date, there exists only one main representative of the SDN approach, the OpenFlow protocol but Cisco is building its approach which is called OnePK, this platform has been used for the development of this project.

2.3.1 OnePK

One Platform Kit (OnePK) is a cross-platform API that gives the opportunity to create applications for Cisco devices. It provides several half-documented interfaces in order to communicate with the device.

Cisco provides with the All in One VM that includes the OnePK SDK, sample apps, tutorials and vmcloud which lets you virtualize routers. The VM has everything a developer may need in order to start creating applications for Cisco devices. This is the environment that has been used for doing the testbed and the experiments.

OnePK has 3 three SDKs, one written in C, which is the most advanced and has more functionalities than the other ones, one written in Java and a newer one written in Python, OnePK Service Sets provide APIs that are bound to a communications library that interacts with its platform-specific counterpart on the network device by means of a secure Remote Procedure Call (RPC) channel. In this project the C and the Python SDKs have been used.

In order to better understand the OnePK system an overview of the architecture is provided in Figure 2.9. As it can be seen in the image this architecture provides a service implementation that the onePK client-server model makes available in a platform-independent and device-independent way.

These are all the API that have been used in order to achieve the goal of this project:

1. Datapath Service Set (DPSS) : Enables an application developer to hook up to the packet flow through a Cisco switch or router and extract packets from that flow of packets. These packets may be either copied from the data path to an application or they may be punted

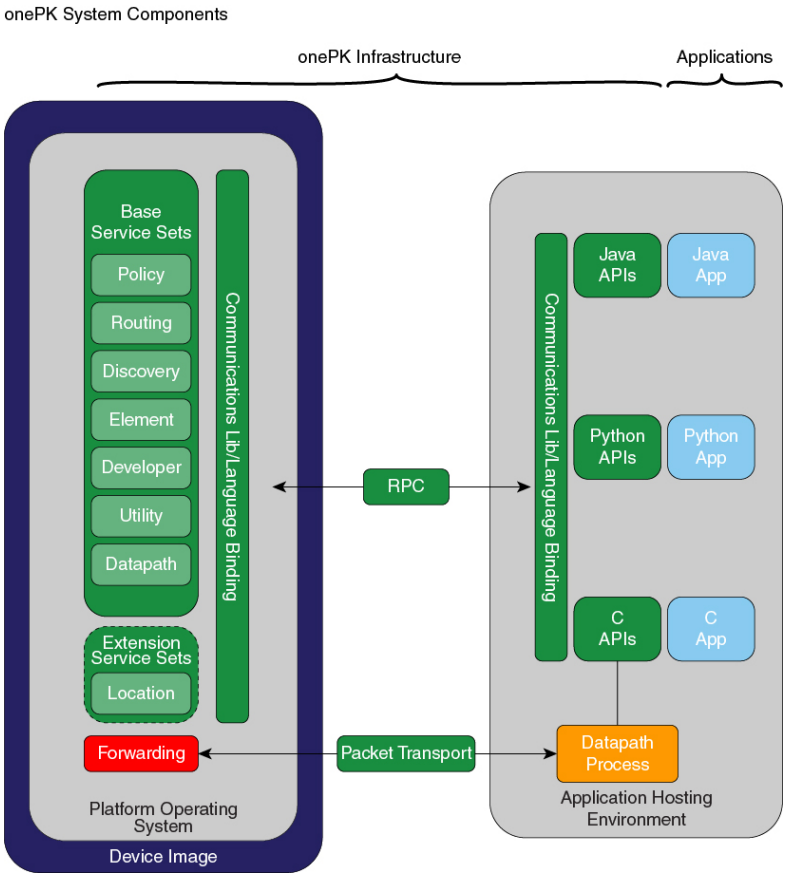

Figure 2.9: OnePK architecture

or diverted to the application. The key difference between diverting and copying is that when a packet is diverted it does not continue on to its destination until the application returns it to the data path, modified or unmodified.

2. Routing Service Set : Provides multiples sub-services. The current supported sub-services are:

   (a) RIB : Provides access to the content of the RIB. The application can:

      i. read the content of RIB

      ii. get notification when the route state changes (up/down) in the RIB

   (b) Application Route Table (ART) : Keeps track of and provides high availability for routes originated from the application. Handles registration with RIB and inserts those routes to the RIB. The application can:

      i. add and remove route from the application route table

      ii. get notification when the application route is promoted or demoted in the RIB

Figure 2.10 diagram depicts the interactions of the application and the RIB and Application Route Table sub-services.

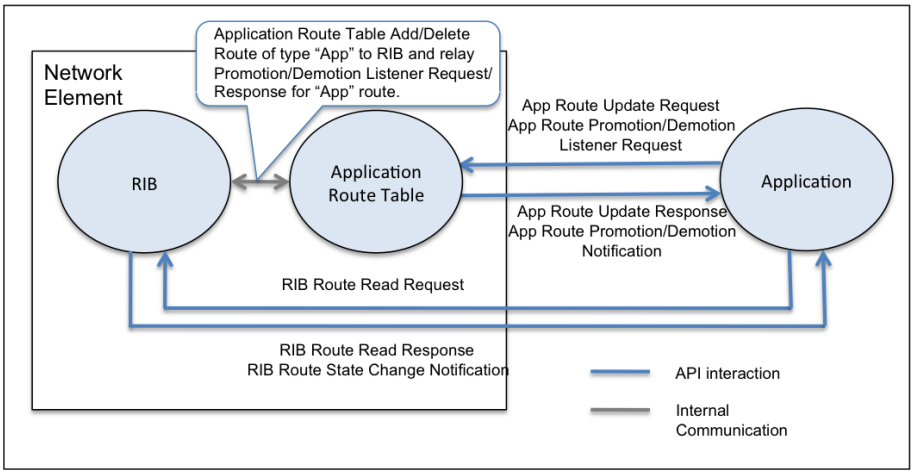

Figure 2.10: Routing Service Set

3. Virtual Terminal Line (VTY) Service Set : Enables the application to open a VTY, execute one or more commands, retrieve the results, query the parser state if the write failed, and close the VTY.

Using OnePK or another similar tool it is possible to deploy innovative routing and switching protocols or, as it is done in this project, secure BGP.

Chapter 3

Problem Definition and State of the Art

In this chapter the problem tackled in this document will be presented and explained. Moreover, we present and comment the existing implemented solutions.

3.1 Problem Definition

As we have explained above, BGP by its nature is based in a mutual trust mechanism between the ASs. BGP, by default, does not validate neither the origin nor the content of the routes announced. This has become a very important problem since the BGP is widely adopted as main EGP for the Internet.

As a result of these vulnerabilities many attacks or attempts of attack are performed every year. For example, it is possible for a malicious organization to announce routes that does not belong to its AS and the routers that are at the border of a legal AS will not have any mechanism to validate the announced routes. The fake routes will be accepted, hence the routers could be routing the traffic of their AS to a malicious router or a network where the malicious organization could, in the best case, drop the traffic. The malicious organization would also be able to analyse the traffic. As is explained in [9], a malicious organization could be interested in getting information of the network topologies and exploiting it for criminal operations and waging cyberwar. These are only some examples of the possible damages of attacks on interAS routing. More scenarios and explanations of the possible damages of interAS attacks can be found in [4][9][10].

A very interesting and relatively new interAS attack is the route leak problem. Nevertheless, in order to explain the route leak problem in BGP, a small introduction to the relationships

between ASs must be done. Although BGP is not aware of business relationships it provides a complex set of rules and policies that can model this kind of business oriented relationships. There are three possible kinds of business relationship between ASs:

1. Client - Provider : In this relationship one of the AS is provider of another AS, hence the traffic routed by the Client AS towards the Provider AS is charged to the client. In this type of relationship the provider usually announces all the known routes, since his economical benefit is proportional to the amount of traffic routed by its AS.
2. Peer - Peer : In this relationship both ASs exchange routes but the traffic routed towards each AS up to an agreed threshold is not charged by any of the peers. Usually, ASs in this relationship do not announce all the known routes, as there might exist economical reasons for not doing so.
3. Sibling Link : In this relationship both ASs provide connectivity to each other without charges. This relationship is very rarely encountered, compared to the two other relationships.

A correct route exporting policy an AS should follow the following three rules, also referred as valley-free rules, according to [10]:

1. Rule R.1. :"Routes learned from Customers can be further advertised to other Customers, Peers and Providers."
2. Rule R.2. : "Routes learned from Peers can be further advertised to Customers only."
3. Rule R.3 : "Routes learned from Providers can be further advertised to Customers only."

The route leak problem is still very recent in the Internet community, thus the formal definition of the problem has not been defined yet. In [11] the author defines the route leak problem as the advertisement of a non-customer route over a peer or a provider link. A route leak does not imply an announcement of an invalid AS path or route. A route leak is a violation of the business policy agreement between two ASs. To facilitate the understanding of the problem we depict a route leak situation in figure 3.1.

Situations similar to the ones described above have happened by accident recently. In 2008 a Pakistan Telecom blocked Youtube for the entire world due to a BGP misconfiguration. The company was trying to block Youtube locally and mistakenly announced the new routing information to an ISP at Hong Kong. This information was propagated throughout all the AS. Another situation, this time relevant to the a route leak problem, happened in November 5, 2012. Google services experienced an outage for almost half an hour because of a route leak

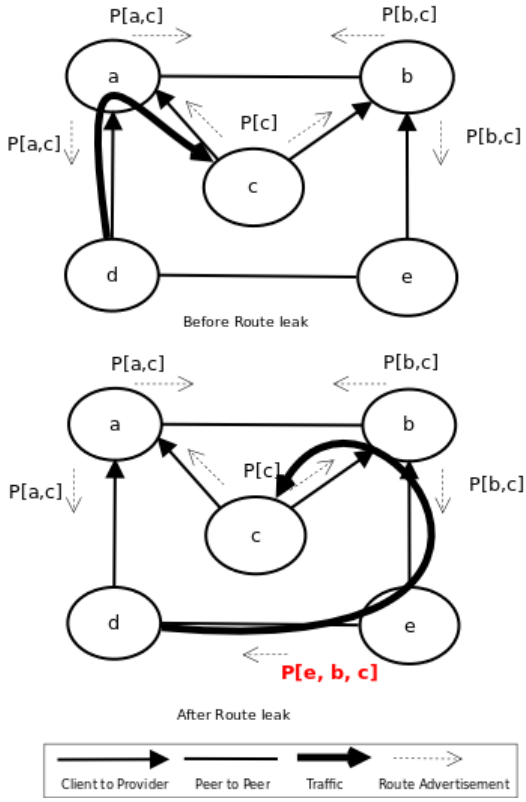

Figure 3.1: Example of Route Leak problem

that happened between two ASs. This route leak in combination with economical reasons attracted a lot of traffic in a specific AS that was not supposed to be routed throughout it. As a result, the network of that AS could not cope with the amount of traffic and eventually started dropping the traffic.

BGP attacks are very serious threat since they are very hard to detect and they can affect massively the performance of the whole Internet. The attackers can be very tempted by the fact in a few hours they can rerout the whole Internet traffic through their network while detecting it may take even months [12] presents important evidence that this can be done in an easy and hardly detectable way.

Many solutions have tried to address the presented BGP security problems, but up to now none of them has been deployed in a real large scale environment. This is a result of the vast expenses and changes in the infrastructure that these solutions entail. Moreover, the majority of the available solutions do not address the route leak problem, which is becoming increasingly important.

3.2 State of the Art

In this section we are going to present the most important works concerning BGP security enhancements.

3.2.1 Secure BGP

Secure BGP (sBGP) was first presented in 1996 by BGP experts that intented to secure the exchange of inter domains routes using BGP. sBGP is one of the major security contributions to the BGP protocol as it offers a relatively complete implementation. sBGP architecture uses three security mechanisms. First an RPKI infrastructure is used to validate the ownership of the UPDATE messages that come from any AS. Second, a new, optional, BGP transitive path attribute is employed to carry digital signatures (inside "attestations") covering the routing information in a BGP UPDATE message. These signatures, along with certificates from the sBGP RPKI enable the receiver of a BGP routing UPDATE to verify the address prefixes and path information that it contains. Third, IPsec is used to provide data and partial sequence integrity, as well as to enable BGP routers to authenticate each other for exchanges of BGP control traffic. This solution, though not completely secure, makes it a lot more difficult to a malicious organization to announce bad IPs.

The attestations of the routes increase the processing power and memory consumption considerably. Noticing that routers not always have a very high computational power this can lead to contention, leading to problems in traffic routing. This case is more obvious in the scenario of a network that has a high exchange rate or when a when router is reset.

Another drawback of sBGP is the necessity for a centralized Public Key Infrastructure (PKI) system. The verification process introduces also performance overheads since every time a route is received the public key of the originator should be obtained in order to decode the packet and verify that its validity.

In order to deploy this solution to the Internet the routers would need to be modified. Additionally, the ISPs of the core network would need to replace the routers with new routes able to execute the sBGP protocol. Moreover, a PKI infrastructure between ISPs is necessary incurring additional costs. As a results of all these, a transition to sBGP would be very expensive.

Many experts have said that despite the fact that sBGP solves the BGP security problem it is too resource consuming to deploy it to a real carrier, or at least not right now.

3.2.2 Secure Origin BGP

Secure Origin BGP (soBGP) is another solution that tries to secure BGP, addressing the drawbacks of sBGP, mostly the ones concerning performance. More specifically t is focused to relieve the processing load needed when a route is validated and the overhead created by every advertised UPDATE with a locally generated router attestation.

To improve performance, soBGP tries only to validate that the UPDATE messages received come from a feasible inter AS path from the BGP speaker to the destination AS. Consequently, the validation becomes weaker compoared to sBGP which uses the attestation mechanism to validate that the UPDATE has traversed all the ASs.

soBGP was an intention to tackle the main disadvantages of sBGP and it successfully managed to be more deployable. While sBGP verifies the route originator and its authorization, soBGP uses A new BGP message is used to carry security information and it has fixed additional scalability requirements. Nevertheless, the security guarantees provided by soBGP are very limited and inaccurate.

3.2.3 Pretty Secure BGP (psBGP)

Pretty Secure BGP [13] introduces the idea that the proposals related to the authentication of the use of an address in a routing context must either rely on the use of signed attestations that need to be validated in the context of a PKI, or rely on the authenticity of information contained in the Regional Internet Registrys (RIRs). The weakness of relying on RIRs is that they lack accuracy for the current authenticity of the information that is represented in a route registry objects. The information may have been accurate at the time the information it was introduced into the registry, but this may no longer be the case at the time the information is accessed by a relying party. Moreover, soBGP states that a PKI can not be constructed in a hierarchical deterministic manner because of the indeterminate nature of some forms of address allocations. This is much a contradiction lately when the very same protocol specification proposes and assumes to use the same PKI infrastructure to map AS numbers hierarchically in a PKI context, thus assuming AS numbers can have this hierarchical PKI design path but IP prefixes do not.

Chapter 4

Architecture and Implementation

In order to solve the problem described in the previous chapter a validation system has been implemented. During this process we faced several challenges. Most of them were related with the features of OnePK SDK. The OnePK SDK framework is still under development, thus bugs exist, documentation is still poor and some features are not yet implemented. In order to understand if the solution was feasible using this framework we initially did a study of the platform. At the beginning of this thesis we started with the OnePK SDK v0.9 which did not provide the necessary tools to implement the solution that were needed in order to implement our solution, like BGP packet interception. As presented later, a basic component of this solution is based on intercepting route insertions in the RIB before the update of the RIB occurs. This was the hardest step in the process of implementing this solution as it was unknown if a solution could be done. Nevertheless, during this study of the platform we also performed some initial tests and we elaborated more our architecture. Finally, the OnePK SDK v1.3 was released(9 September 2014) which included a service API which gives developers the ability to intercept packets. From that point on, we were able to proceed normally with the implementation of the proposed system. To implement our solution we used the All-In-One VM offered by Cisco, which contains the OnePK SDK installed and vmcloud, a tool that is used for virtualized networks.

It needs to be mentioned that this solution contains a limitation, again related with Cisco's software lack of features, that is explained in the 4.4 section.

4.1 Proposed solution

The innovation of the solution proposed in this document is moving the decision process of the BGP daemon out of the router. As a result, any validation taking place would be transparent to the BGP daemon of the router, not affecting its performance. This approach is very powerful as it is not necessary to change the BGP implementation, a problem that proved to be a huge barrier

for deploying in real networks the solutions described above. As OnePK is being developed by Cisco, one of the major routers and switches producer of the core network devices, the solution could be easily deployed in many parts of the core network without big expenses. Many Cisco devices will be able to run OnePK and thus will be possible to deploy easily this solution in a real network.

Figure 4.1 depicts the idea of the proposed solution. The idea can be applied to any SDN enabled set of devices and using any available server technology. As shown, the decision process of the BGP daemon is taken out of the router, making our solution generic enough to be installed in any SDN-enabled device without any changes. In our solution we adopt OnePK as our SDN technology. The OnePK applications communicate with the device using the OnePK API explained in the 2.3.1 section. The information gathered by the applications is sent to the Server application and are used to build the network representation. Using the Web User Interface the user is able to set rules to the devices of the network and using this rules the decision is taken. In our solution we adopted the Django framework as a Web server. There is also another entity in this solution which is the ROA authority that helps in the decision process.

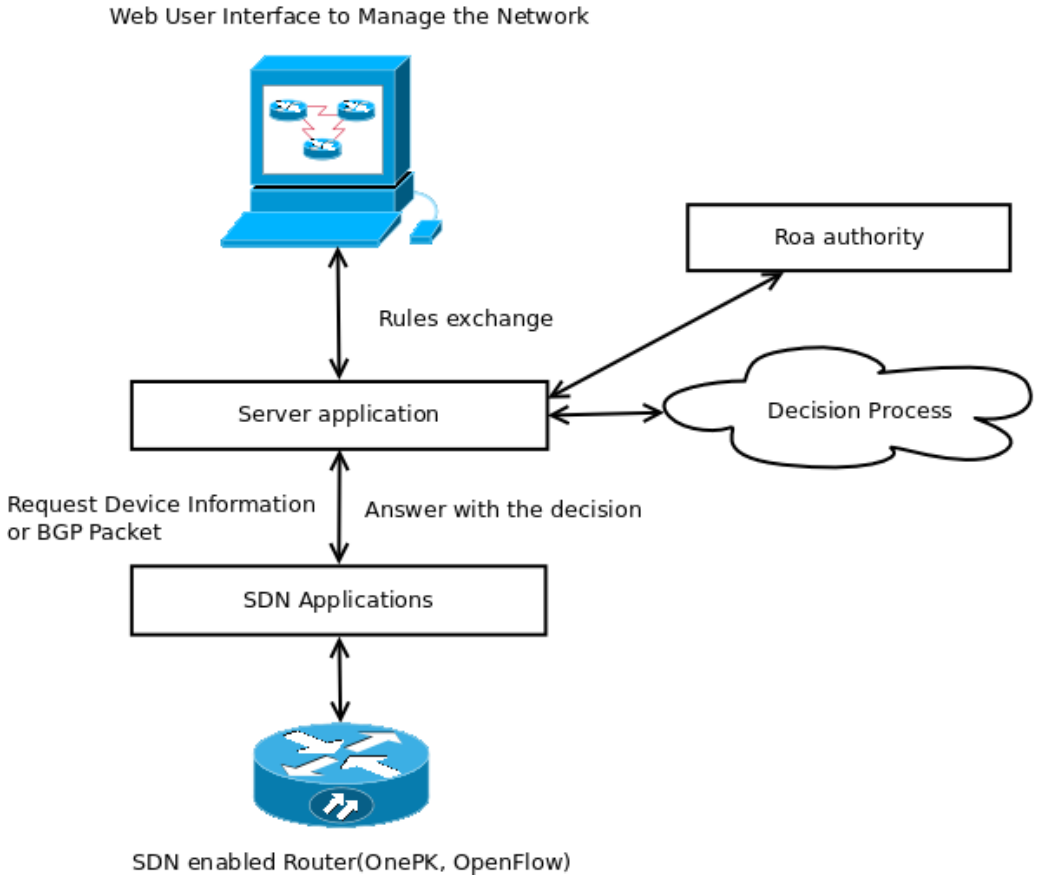

Figure 4.1: Proposed Solution

The solution is achieved by a combination of various components that deal with different problems and all together they compose the entire system. These parts are:

1. Packet Interception
2. RIB Update
3. Topology Discovery
4. Route Validation
5. Network management
6. ROA System

In this section all the different parts are going to be described briefly. Our description takes into account the environment of our specific implementation (OnePK), but the components and their functionality would be similar no matter the technical decisions made.

Packet Interception This is the crucial feature of this solution as without it the validation of the routes would have been impossible to be achieved. We achieve that using the Datapath Service API of the OnePK platform. In 4.3 we provide a more detailed explanation of the calls and code used in order to achieve this functionality.

RIB Update This feature is achieved by an application that uses the Python SDK of OnePK. This application retrieves the RIB entries of the device every 5 seconds retrieving pushes the information to the API of the Server application. For the implementation of this feature we used the Routing Service API of the OnePK platform.

Topology Discovery This feature is achieved by an application in each device that uses the Python SDK of OnePK and is running every 10 seconds. The application uses the VTY service API provided by OnePK.

Route Validation This feature is achieved as a combination of many agents. Both the client and the server side of our system are involved. The validation is done in the server side and the decision is forwarded to the device. The application on the devices which processes decides the action that should be applied to the packet.

Network Management This is a server side feature providing a graphical interface in a browser. Through this component the user is able to monitor the nodes of the network, set prefix and AS path rules to the nodes and see almost in real time the UPDATE messages and if they are accepted or revoked by the nodes. The interfaces also shows the originator and the destination node of the messages. To provide even more information to the user, we also present statistics of the revoked and accepted packets of each node, thus users can identify which nodes are receiving more often bad or invalid packets.

ROA System This component of the system is provided by an external implementation that was done by the RIPE NCC [14]. As explained in [14], RIPE NCC provides a package so everyone(although focused on RIPE NCC members) can run its own certificate authority. We are using the ROA system for testing purposes in this thesis without syncing our authority with the official authorities as we are not a RIPE NCC member.

4.1.1 Comparison between State of the art solutions and our solution

In this paragraph we compare the different existing solutions with our solution. As shown in 4.1.1,the main advantage of our solution is easy deployablity. Although we do not provide a specific system to prevent route leaks, the most common cause of route leaks are policy misconfigurations of and our solution gives the network administrator a easier way to apply policies and check them. Moreover, since the BGP packet is unpacked and stored in a well structured database, making to more difficult to create misconfigured policies.

	Trust Model	Path Auth	Origin Auth	Deployed	Route Leak Protection	Deployability
sBGP	Centralized	Strong	Strong	No	No	Difficult
soBGP	Web of Trust	None	Strong	No	No	Difficult
psBGP	Centralized	Strong	Weak	No	No	Difficult
Prop. Solution	Centralized	Policy rules	Strong	No	Ease policy configuration	Easy

4.2 System Architecture

In this chapter we describe in more detail the system architecture and the interaction between the different components. In the solution proposed the system can be split as three categories:

1. Device component : In this category are all the processes that run in the router device.
2. Server component : In this category is the server application that interacts with the device components.

3. ROA component : This category contains the application that runs the certificate authority.

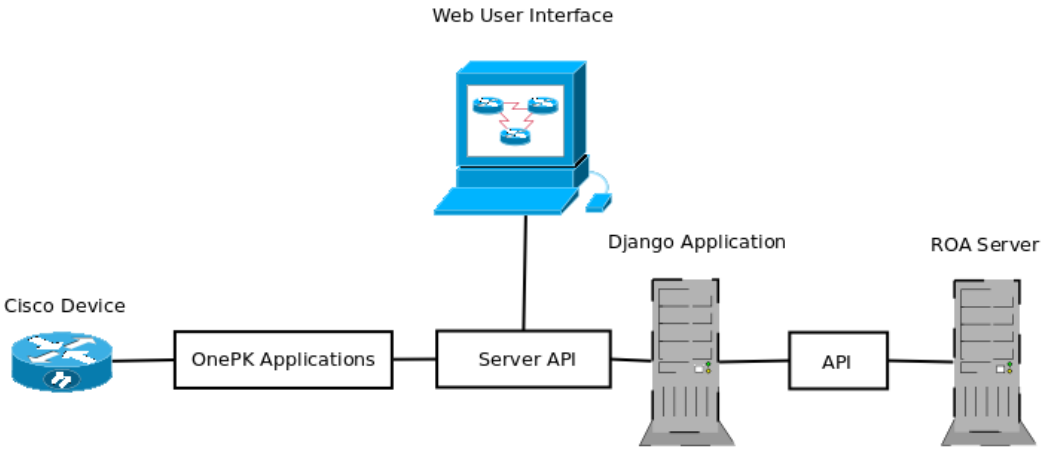

Figure 4.2: System Architecture

4.2.1 Device Component

This component is based in three applications that run in each router device.

The first and most important application is written in C and is listening for incoming packets. After obtaining the payload of the BGP packet it pushes that information to the server application. This application uses the Datapath API provided by OnePK. First it connects to the device and then it registers a function for every new incoming packet. If the received packet is a BGP packet it sends the payload to the server where it will be processed. The second application that runs on the devices is a Python application which using the Routing Service API polls every 5 seconds for all the routes in the RIB. The information obtained is sent in JSON[15] format to the server application where it is processed and attached to the right node. The third and last device component is the BGP topology discovery application, a Python application that using VTY Service API sends commands to the router and parses its output. Using the received output it retrieves the necessary information concerning the router neighbours and sends it to the server application in JSON format.

4.2.2 Server Component

This component is a Django[16] application. In order to provide communication between the Device component and the Server component an API has been implemented. For this purpose

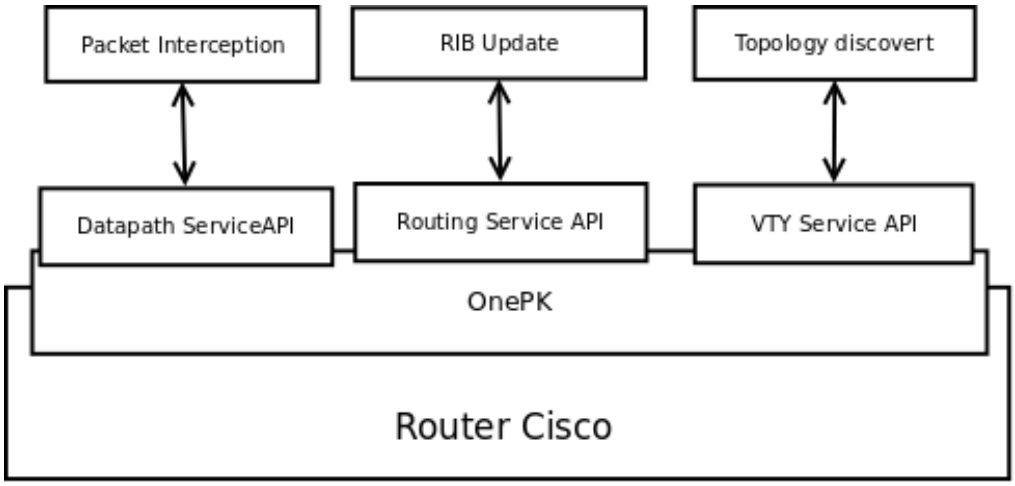

Figure 4.3: System Architecture

we used the Django RESTframework[17], although the most important calls of the interaction with the Device component have been implemented using custom components. While Django RESTframework plugin offers a good manner to create an API for the main Resources that may exist in any application the main API calls have been done with custom templates and custom view calls that will be explained in the implementation chapter.

4.2.3 ROA Component

This component is based in a RIPE NCC implementation that is provided in [14] and can be installed in some systems. In order to use it for this thesis we had to add the routes to a whitelist that is provided by a web interface that comes with the server implementation. The interaction between the Server component and the ROA component is done using a REST API and the response is provided in JSON format.

4.3 Implementation

In this chapter the implementation of our system will be explained in detail as well as the challenges encountered and the technologies used. The solution is going to be explained using the same categorization and structure as the previous section to facilitate the reader.

4.3.1 Device Component

As explained in the previous section, the Device component is implemented using three applications that run in each device.

Packet Interception Application This is a C application that is constantly running on the devices. Using the Datapath Service API from OnePK a programmer is able to register a function *callback* every time a package is received in the device. As explained in the OnePK documentation[18] the function call is the following:

```
onep_status_t dpss_tutorial_create_ip_pmap (
        onep_network_element_t *elem,
        onep_dpss_pak_callback_t callback,
        onep_dpss_pkt_action_type_e action,
        onep_policy_pmap_handle_t *pmap_handle,
        onep_policy_pmap_op_t *pmap_op,
        onep_policy_op_list_t *pmap_op_list,
        onep_policy_op_list_t *cmap_op_list,
        onep_policy_cmap_handle_t *cmap_handle,
        onep_policy_cmap_op_t *cmap_op,
        onep_acl_t ** acl);
```

For this thesis we based our code in the DatapathTutorial.c that comes with the All in One VM that is provided by Cisco. Below we present the *callback* function that is called every time a packet is received by the device which is registered with the function.

```
void dpss_display_pak_info_callback(
                onep_dpss_traffic_reg_t *reg,
                struct onep_dpss_paktype_ *pak,
                void *client_context,
                bool *return_packet);
```

In this function we check for the protocol, ports and destination of the packet intercepted and we push the packet to the server in case it is a BGP packet. This application uses libcurl to issue http requests to the server and the cJSON[19] library is used to create and parse the responses from the server. In the Appendix A we provide the code for these functions and the parse response function.

RIB Update Application This is a Python application that runs on each device and is querying the RIB of the device every 5 seconds as explained in section 4.2.1. The queries to the RIB of the device are issued using the Routing Service API as described in [18]. The information obtained is pushed to the server using httplib and urllib Python libraries.

The information of the RIB of the device is sent in JSON format to the /rib/ API endpoint of the server who parses the information and attaches it to the proper node of the network. To help the server associates the information received with the correct node, a router identifier is sent together with the RIB data. An example of the format of the information sent to the API is shown in section A.3.

BGP Topology Discovery Application This is a Python application that runs on each device and uses the VTY Service API to obtain information about the BGP neighbours and other device specific data. To achieve that the application is sending native router commands that are executed through the VTY Service and parses the replies. To send this information to the server we use the urllib and httplib Python libraries. This process is repeated every 10 seconds providing the user with a pseudo real-time representation of the state of the network. The application provides information that allows the users to understand the originators of bad route announcements and take the appropriate actions to mitigate any misconfiguration or attempt of attack.

4.3.2 Server Component

In this chapter we describe the technologies used for the server, presenting more in depth the most important parts of the implementation.

Django Application The Server component is mostly implemented by a Django application. Django is an open-source project being developed since 2005 up to nowadays and serves as a web framework which provides developers with a fast way of building elegant web applications quickly. For the implementation of the server Django v1.7 has been used deploying both CBV(Class Based Views) and FBV(Functional Based Views). Django uses the MVC(Model-View-Controller) pattern.

Server API This section provides a description of the API endpoints implemented for the interaction with the device applications. For a more detailed example of the requests and responses the user can refer to the Appendix A.

To facilitate the presentation of the API we adopt a generic example that includes all the main resources of the server and then we will continue with a more detailed explanation of each separate endpoint.

For the main resources of the server:

1. Node : This resource is the model is used by the server to represent a router in the network.

2. Packet : This resource is the model used by the server to represent a packet in the network.
3. Prefix Rule: This resource is the model used by the server to represent a prefix rule that is attached to a router in the network.
4. AsPath Rule: This resource is the model used by the server to represent an AS path rule that is attached to a router in the network.
5. Route : This resource is the model used by the server to represent a route in the RIB of a router in the network.
6. Interface : This resource is the model used by the server to represent an interface of a router in the network.

For this resources a uniform API is provided using the Django RESTframework. All of the resources expose the following endpoints:

1. BASEURL/resource/new/ : This endpoint is used to create an object of the specific resource. All the required fields of the specific resource should be sent in JSON format.
2. BASEURL/resource/{id}/ : This endpoint is used to retrieve the details of the resource with this specific ID in JSON format.
3. BASEURL/resource/edit/{id} : This endpoint is used to edit the resource with this specific ID.
4. BASEURL/resource/delete/{id} : This endpoint is used to delete the resource with this specific ID.

For every main resource in the Server, besides this generic API calls, our API provides more specific API endpoints. Below we describe these endpoints. Examples of the requests and responses to this API endpoints are provided in the Appendix A.

1. BASEURL/evaluate/RouterId/SourceIP/DestinationIP/SourcePort/DestinationPort/ : This endpoint is used by the Packet Interception application. A request to this endpoint invokes the process that starts the decision procedure, where the BGP packet is parsed, evaluated and the response is in JSON Format.
2. BASEURL/statistics/RouterId/ : This endpoint can be used to retrieve the statistics of a specific router, a functionality mostly utilized by the Web User Interface.
3. BASEURL/update/info/router/RouterId/ : This endpoint is used by the BGP Topology Discovery application in order to maintain up-to-date the information of the BGP neighbours and how they are connected with each other.

4. BASEURL/rib/ : This endpoint is used by the RIB Update application in order to update the RIB information of a router in the network.

Packet Parsing In order to decode the packet we used the dpkt library[20]. Since the library does not support decoding 32bit format updates we introduced the necessary modifications, which are shown in the Appendix A.

User Interface The User Interface is implemented on top of the VisJS library[21], the Bootstrap[22] framework, the jQuery[23] library and the Django template system in order to achieve the interaction between the DOM elements of the web user interface. This components provides a friendly User Interface for network management. Below we include screenshots of the different features provided through this Web User Interface.

A notification system informs the user when a packet is sent from one node to another with a pop-up window at the bottom right corner. The color of the pop-up window declares if the packet is accepted, green, or rejected, red. Figure 4.4 depicts a notification when a packet is intercepted and processed by the application. In this example the packet is accepted.

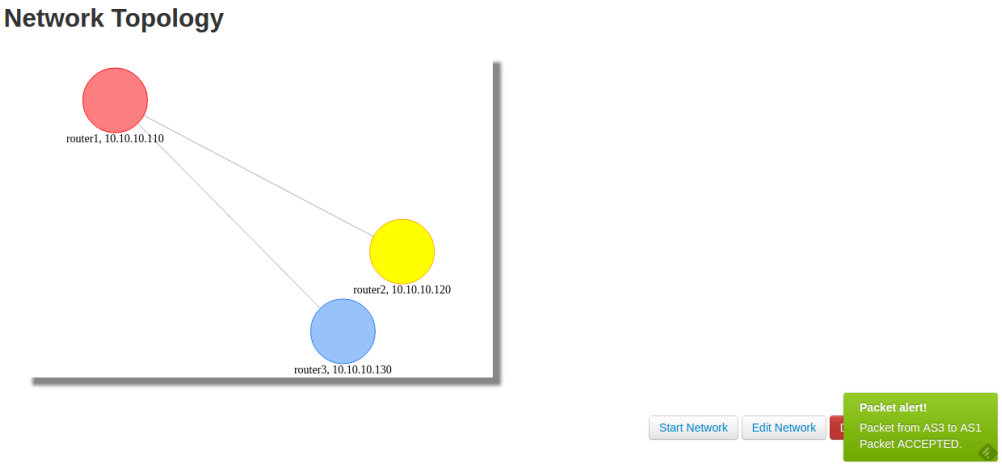

Figure 4.4: Packet notification

Figure 4.5 presents a sample of how the RIB entries shown in the application.

Figure 4.6 is a sample of the packet statistics available.

Figure 4.7 shows the user interface to create a prefix rule for a node.

Network graph In order to create the netgraph we used the VisJS library. This library provides a way to visualize data as a network of nodes. To facilitate visualization and management

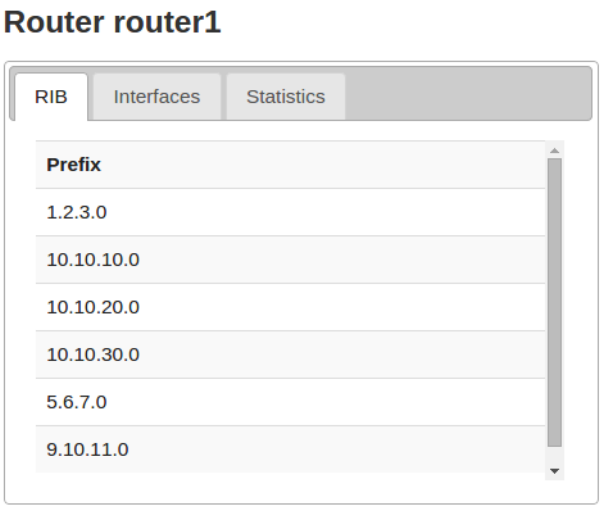

Figure 4.5: RIB table

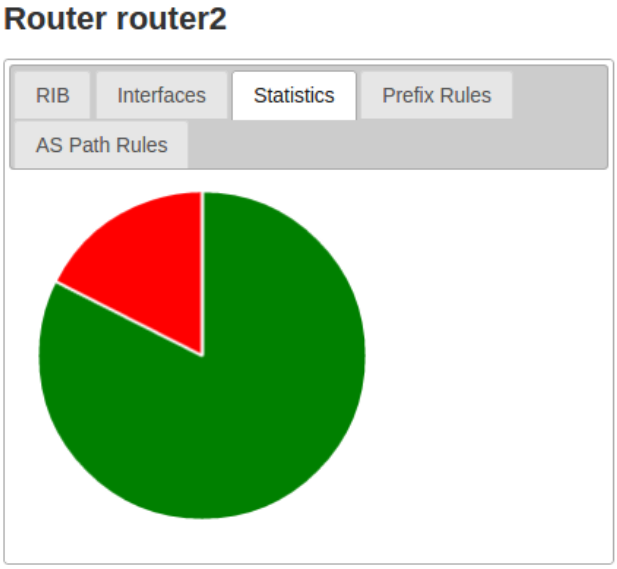

Figure 4.6: Packet Statistics

of the network nodes to the user we also introduced jQuery functionalities.

As shown in Figure 4.8, an interactive network topology is provided. The nodes have different color depending on the AS they belong to. When the user selects a node, a new window appears containing real-time information(RIB, interfaces) and statistics of the specific node, but also allowing the user to insert prefix and AS path rules.

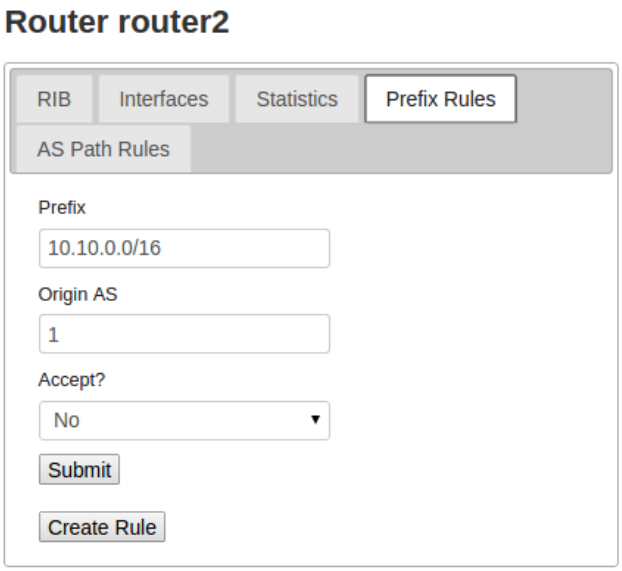

Figure 4.7: Prefix Rule UI

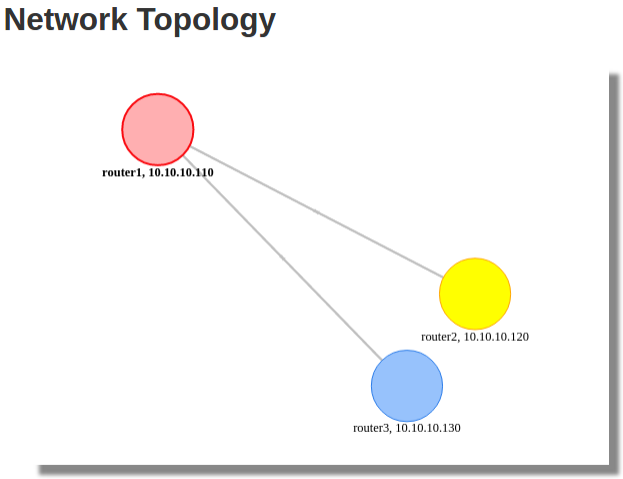

Figure 4.8: Network graph

4.3.3 ROA Component

This component, as explained above, is a RIPE NCC implementation of a certification entity. We have installed the ROA package in the All in One VM. This component offers an API in order to validate the origin of a route based comparing the information with its database. Following we provide a description of the endpoint offered by this components:

- /BASEURL/api/v1/validity/ASN/IP/LENGTHPREFIX : Using this endpoint the system is able to validate the origin of a received route. The response of this endpoint is in JSON format and gives information about the validation of the route.

The Server component uses this endpoint every time it receives an UPDATE in order to validate the origin of the route. In the next step of the process, the server checks the prefix and AS rules associated with the destination Node. To summarize the validation process, we provide a two-step validation:

1. The origin of the route is validated using the ROA component
2. The information is validated based on the stored rules associated to the node in question.

4.4 Limitation

OnePK SDK is still under development and currently there is no BGP API implementation, though Cisco has announced that it will be included in the next version of the OnePK SDK. In order to overcome this limitation, our system uses intermediate routers that intercept the packet before it is received and processed by the BGP process. When a BGP packet is intercepted it is sent to the server where it is processed. Figure 4.9 presents the described limitation and our solution.

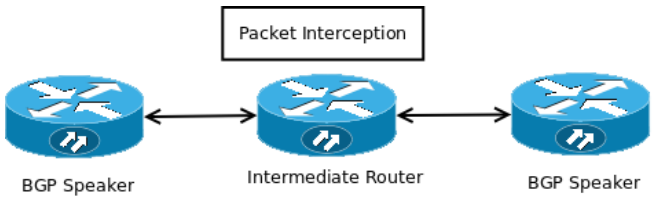

Figure 4.9: Limitation of the solution

In this point, we would also like to point out that due to lack of time and access to real Cisco routers the implemented system has been tested and benchmarked inside the environment of Cisco All-in-One VM.

Chapter 5

Experimental Results

In this chapter we introduce the experimental environment deployed as well as the validity and scalability experiments performed. We also present the results obtained and we discuss them briefly.

5.1 Testbed

In order to test the proposed solution in this document a testbed with the resources available has been deployed. As explained in section 4.4, an intermediate router needs to be placed between two BGP speakers in order to intercept the BGP packets. Thus, the testbed necessarily contains an extra intermediate route for every link between BGP speakers. As also explained in section 4.4, all the testbed routers are simulated inside one All-in-One VM in one physical host.

The testbed consists of 6 BGP speakers and 4 intermediate routers for the interception of the BGP packets. Figure 5.1 shows the testbed structure.

All the routers are virtualized, running inside the All-in-One VM provided by Cisco. The network topology is simulated using the vmcloud tool. In the Appendix A all the configuration for this testbed is provided, to facilitate the replicability of our experiments.

The Server application was executed in the Host OS. For a more realistic Server Application set up we avoided using the development server provided by Django and adopted our own web-server and application server solution. The web-server chosen is nginx [24]. Nginx is an open source reverse proxy server for various protocols. It has been proved that nginx works well in high concurrency environments and uses low memory. For the application server we chose Gunicorn[25]. Gunicorn is a Python Web Server Gateway Interface HTTP Server for Unix. Gunicorn has been proved to work well with Django and has an easy set up procedure.

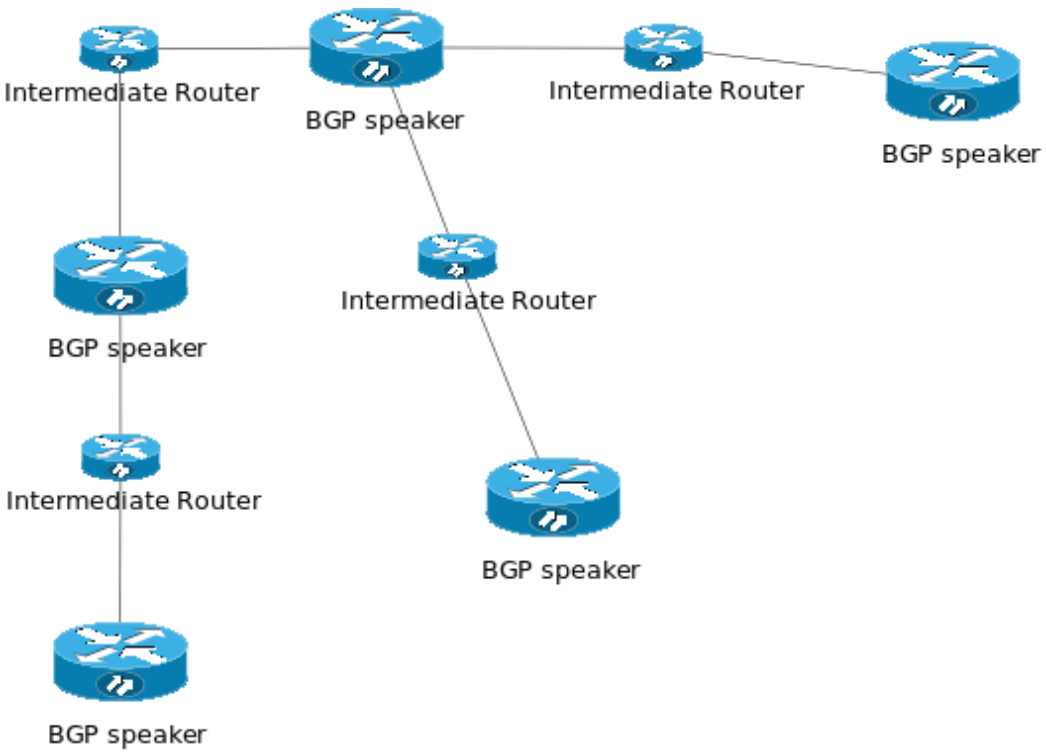

Figure 5.1: Testbed Structure

Furthermore, Gunicorn has also proved to be light resource consuming and fairly fast.

All the experiments have been executed in a personal laptop, with CPU *Intel i5 2410M @ 2.3GHz* and *16GB* RAM.

5.1.1 Validation Tests

After the testbed was set up a scenario was tested in order to validate the proper system operation.

Prefix and BGP Path Hijacking Test

This scenario simulates an attempt of Prefix Hijacking. After the network enters a stable state one of the routers starts to sends invalid Prefixes. The execution of the scenario proves that the system is able to detect those UPDATES and treat them appropriately. We provide below a step by step explanation of how the system is able to detect the malicious Prefix.

The first step in in order to execute the scenario we need to configure the ROA system. The

ROA system has a whitelist where valid prefixes for specific ASs can be configured. If we want the system to evaluate as valid an announced prefix the first step is to create an entry to the ROA whitelist. In Figure 5.2 some examples of entries in the whitelist are shown.

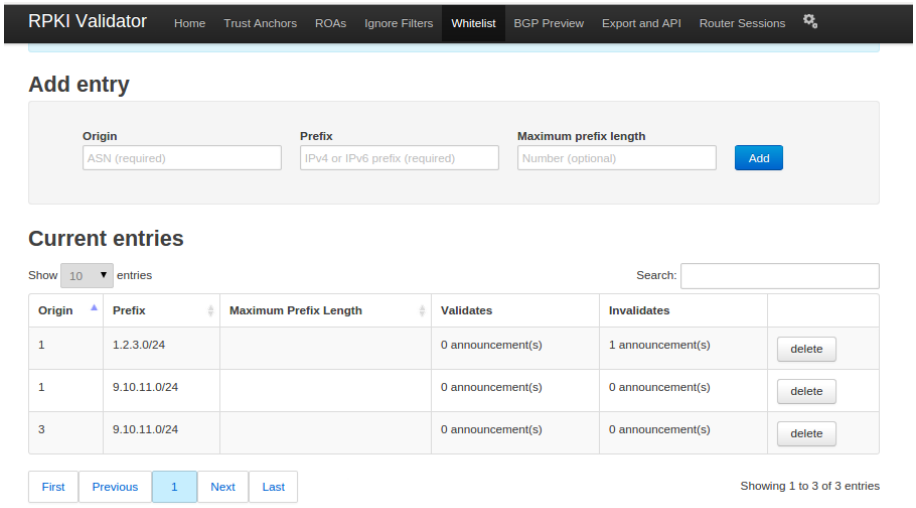

Figure 5.2: ROA whitelist

After creating the entry in the ROA system whitelist a BGP update is intercepted by the interception packet application and then pushed to the Server. The Server decodes the packet and forwards it to the ROA system API where it will be evaluated. In Figure 5.3 an example of a valid evaluation of a BGP update is shown.

If the ROA validation was successful then the prefix rules of the node will evaluate the packet. In Figure 5.4 an example of a set of rules assigned to a Node is shown.

Besides the prefix rules of the nodes also a BGP path validation is done. In Figure 5.5 an example of a set of rules assigned to a Node is shown.

After all these processes are executed automatically upon a packet interception a pop up notification showing the user if the packet was accepted or rejected will be shown. The notification provides also the information of the origin and destination of the packet. Moreover, the system also keeps track of the received packets and shows a graph with the rejected and accepted packets. An example of the packet notification is shown in figure 5.6.

As we can see in this example, AS1 is trying to announce an invalid prefix to AS2 and the system is able to detect it. In this example the ROA system validates the route but due to a prefix rule in the AS2 the packet is rejected.

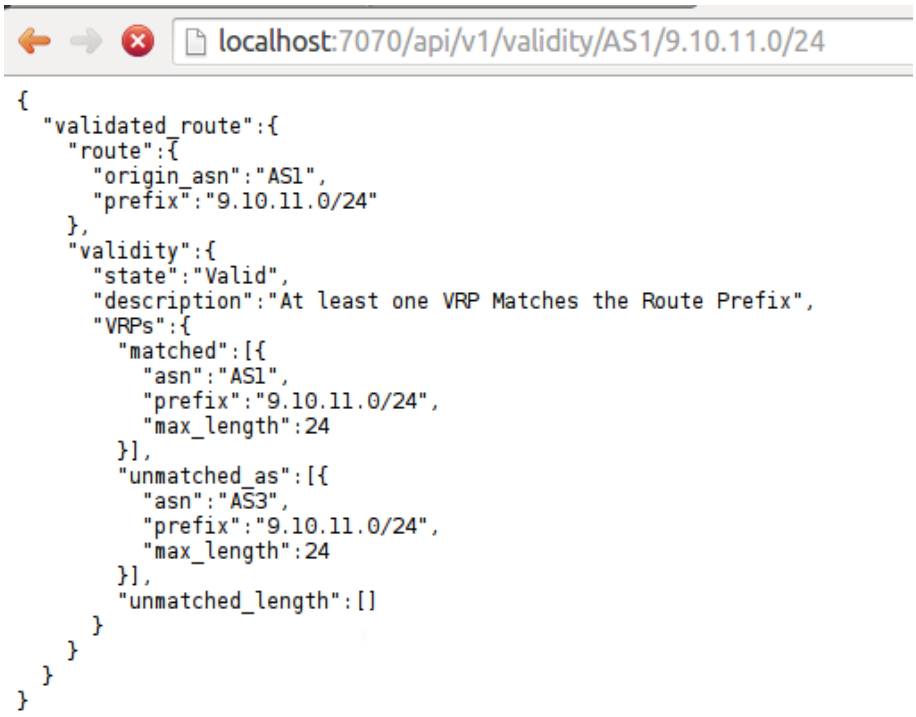

Figure 5.3: Example of a response of the ROA system

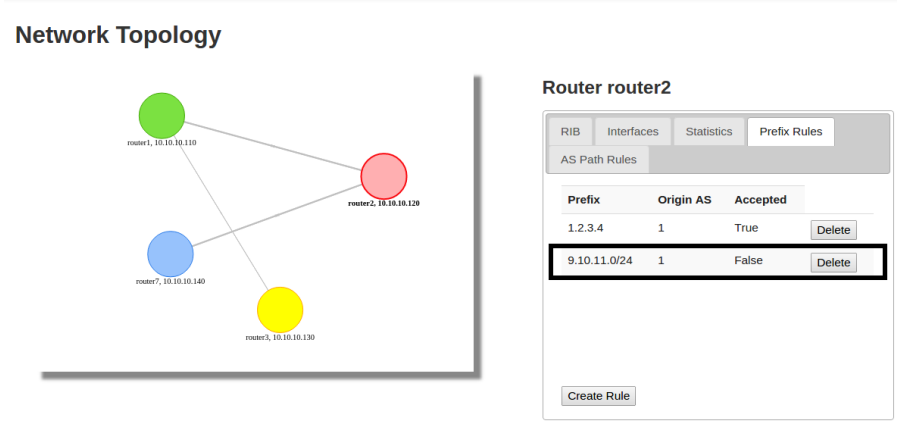

Figure 5.4: List of Prefix Rules assigned to a Node

5.2 Scalability of the solution

Maintaning the set up and the tools explained above, we performed stress testing of the Server app. All the experiments have been executed in a personal laptop, the characteristics of it are:

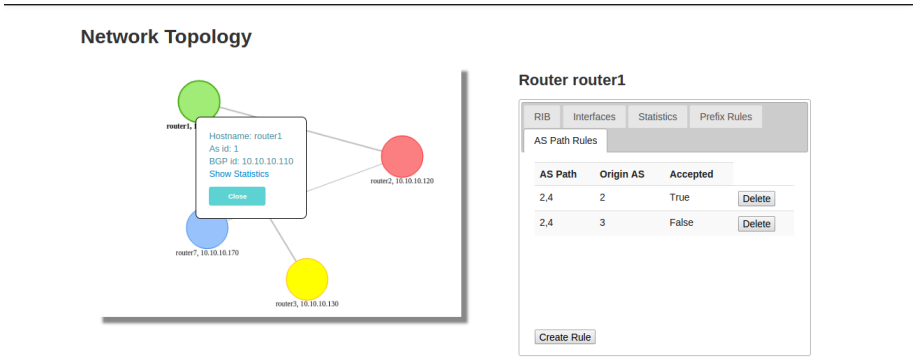

Figure 5.5: List of AS Path Rules assigned to a Node

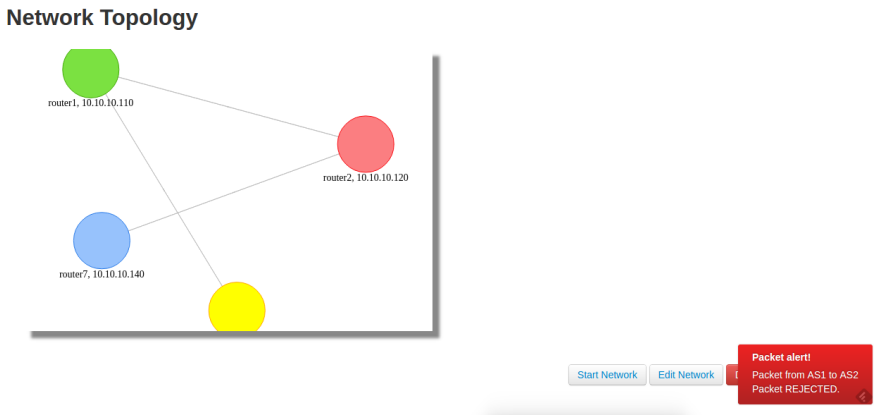

Figure 5.6: Rejected Packet Notification Example

1. Laptop
   (a) CPU : Interl i5 @ 2.3GHz
   (b) RAM : 16Gb

The objective of this experiment was to test how many nodes could a single instance of Server application serve without errors.

The stress testing tool we used is the Apache Benchmark (ab)[26]. The configuration of the Gunicorn application server for this experiment is provided in Appendix A. In order to simulate the scenario three instances of the ab were executed concurrently simulating the behaviour of the three applications running in the devices and sending requests to the Server application with a mock data generated for this purpose.

To obtain the representative response times we have run the ab benchmark three times for every situation and used the arithmetic mean of the three values obtained. As can be seen in figure

5.7 all of the applications have very similar response time so none of them is the main bottleneck of the system. After executing three instances of ab with level of concurrency fifty, the server is failing and not responding to all the requests and causing also a a significant drop in the response time.

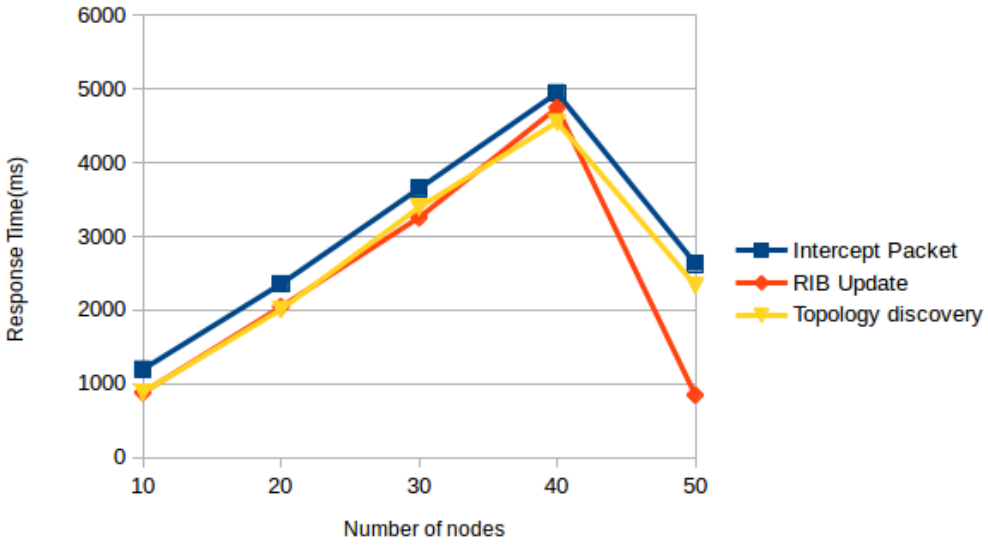

Figure 5.7: Scalability analysis of the Server Application

The result we can obtain from the figure is that the Server application saturates when more than 40 nodes are issuing concurrently requests from all their Device applications.

We have used ab which does not model a realistic scenario in our system. The reason of this unrealistic scenario is how ab makes the request, ab does all the requests at the same time, in a realistic scenario the device applications will be doing requests at different time intervals so it is very unlikely that all the nodes will send the request at the same time. Besides the unrealistic workload there is also the added burden that the ROA system is running in the same laptop that is running the benchmark and the server application. Nevertheless, this can be considered the worst case scenario. As a result we have proved that in the worst case scenario the Server application can manage at least 40 connected nodes.

Chapter 6

Conclusion

In this project it has been proven that a solution to secure BGP transparently to the protocol is possible. The solution presented in this document is a early stage of a very powerful tool for SDN networks. The main idea of the solutions is to take out of the device the decision process of the BGP protocol. To achieve this goal we have used OnePK that offers Service Sets API that allow us to interact with the devices. The main competitive advantage between the solution proposed here and the state of the art solutions presented is that is easily deployable as it can be used in any network that have OnePK compatible devices, and Cisco is one of the biggest core network device providers. We have made a solution using OnePK applications running in Cisco devices that combined with a Server application. And we have done a experiment in order to show that it works and a performance experiment to test the scalability of the solution.

In conclusion it has been shown that SDN platform can lead to very impressive an innovative ways of dealing with network devices, both more comfortable and powerful for the management of networks.

6.1 Future work

Because of the early stage of OnePK there is a lot of future work to be done. The most important one is after the BGP API is released and if it is possible adapt this solution without the limitations explained in the document. Besides this very important improvement the Server application could be done with a bidirectional communication mechanism so the Server could trigger some actions proactively. And as Cisco is releasing new versions of OnePK other interesting features could be adapted to this system. Besides this work a lot of optimization work is still pending to be done so maybe the solution can be more scalable.

Acronyms

ab Apache Benchmark

API Application Programming Interface

ART Application Route Table

AS Autonomous System

BGP Border Gateway Protocol

BGPSEC BGP Secure

DPSS Datapath Service Set

EGP Exterior Gateway Protocol

EBGP Exterior Border Gateway Protocol

iBGP Interior Border Gateway Protocol

IETF Internet Engineering Task Force

IGP Interior Gateway Protocol

IP Internet Protocol

IPv6 Internet Protocol version 6

ISP Internet Service Provider

NAP Network Access Point

NDP Network Discovery Protocol

OnePK One Platform Kit

OSPF Open Shortest Path First

PKI Public Key Infrastructure

RFC Request for Comments

RIB Routing Information Base

RIR Regional Internet Registry

ROA Route Origination Authorization

RPKI Resource Public Key Infrastructure

sBGP Secure BGP

soBGP Secure Origin BGP

SDN Software Defined Networking

SEND Secure Neighbour Discovery Protocol

SIDR Secure Inter-Domain Routing

TCP Transmission Control Protocol

VM Virtual Machine

VTY Virtual Terminal Line

Bibliography

[1] Cisco, "Onepk." http://www.cisco.com/c/en/us/products/ios-nx-os-software/onepk.html.

[2] K. Lougheed and Y. Rekhter, "Border Gateway Protocol (BGP)." RFC 1105 (Experimental), June 1989. Obsoleted by RFC 1163.

[3] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)." RFC 4271 (Draft Standard), January 2006.

[4] S. Murphy, "BGP Security Vulnerabilities Analysis." RFC 4272 (Informational), January 2006.

[5] B. Halabi, *Internet Routing Architectures*. Indianapolis, IN: Cisco Press, 1997.

[6] BGP Cisco report, "Border Gateway Protocol (BGP)," tech. rep., Cisco, 1999.

[7] Wikipedia, "RPKI." http://en.wikipedia.org/wiki/Resource_Public_Key_Infrastructure.

[8] A. Barbir, S. Murphy, and Y. Yang, "Generic Threats to Routing Protocols." RFC 4593 (Informational), Oct. 2006.

[9] C. Meinel, "Attacking and Defending the Internet with Border Gateway Protocol (BGP)." http://www.ciscopress.com/articles/article.asp?p=1237179, 2008.

[10] M. Siddiqui, D. Montero, R. Serral-Gràcia, and M. Yannuzzi, "Self-Reliant Route Leak Identification in Inter-Domain Routing," tech. rep., Advanced Network Architectures Lab, 2014.

[11] B. Dickson, "Route Leaks - Requirements for Detection and Prevention thereof. draft-dickson-sidr-route-leak-reqts," 2012.

[12] A. Pilosov and T. Kapela, "Stealing The Internet." https://www.defcon.org/images/defcon-16/dc16-presentations/defcon-16-pilosov-kapela.pdf, 2008.

[13] T. Wan, E. Kranakis, and P. C. van Oorschot, "Pretty secure bgp, psbgp.," in *NDSS*, The Internet Society, 2005.

[14] Ripe NCC, "RIPE NCC RPKI Test Environment." http://www.ripe.net/lir-services/resource-management/certification/rpki-test-environment.

[15] G. Inc., "Json." http://www.json.org/.

[16] D. community, "Django framework." https://www.djangoproject.com/.

[17] T. Christie, "Django rest framework." http://www.django-rest-framework.org/.

[18] Cisco, "OnePK API Reference." https://developer.cisco.com/media/onePKCAPI-v1-1-0.

[19] D. Gamble, "cjson." http://sourceforge.net/projects/cjson/.

[20] dugsong@gmail.com, "Dpkt." https://code.google.com/p/dpkt/.

[21] Community, "Visjs." http://visjs.org/.

[22] Twitter, "Bootstrap." http://getbootstrap.com/.

[23] jQuery Foundation, "jquery." http://jquery.com/.

[24] I. Sysoev, "Nginx." http://nginx.org/.

[25] B. Chesneau, "Gunicorn - python wsgi http server for unix." http://gunicorn.org/.

[26] Community, "ab - Apache HTTP server benchmarking tool." http://httpd.apache.org/docs/2.2/programs/ab.html.

Appendix A

Source Code, Requests Configurations

In this chapter all the code and extra information for the better understanding of the thesis is provided.

A.1 onep_status_t dpss_tutorial_create_ip_pmap Code

```
/*
 * Example function to create a simple ACL and Policy Map
 */
onep_status_t dpss_tutorial_create_ip_pmap (
    onep_network_element_t *elem,
    onep_dpss_pak_callback_t callback,
    onep_dpss_pkt_action_type_e action,
    onep_policy_pmap_handle_t *pmap_handle,
    onep_policy_pmap_op_t *pmap_op,
    onep_policy_op_list_t *pmap_op_list,
    onep_policy_op_list_t *cmap_op_list,
    onep_policy_cmap_handle_t *cmap_handle,
    onep_policy_cmap_op_t *cmap_op,
    onep_acl_t ** acl)
{
    onep_ace_t *ace40 = 0;
    onep_acl_t *onep_acl = 0;
    onep_collection_t *result_list = 0;
    onep_iterator_t *iter = 0;
    onep_policy_action_holder_t *ah = 0;
    onep_policy_action_t *dp_action = 0;
```

```
 onep_policy_entry_op_t *entry_op;
 onep_policy_match_holder_t *mh = 0;
 onep_policy_match_t *match = 0;
 onep_policy_table_cap_t *table_cap = 0;
 onep_status_t rc = ONEP_OK;
 onep_status_t destroy_rc = ONEP_OK;

 /* create a simple ACL, ip any any */
 rc = onep_acl_create_l3_acl(AF_INET, elem, &onep_acl);
 if(rc != ONEP_OK) {
   fprintf(stderr, "\nError in onep_acl_create_l3_acl: %d, %s\n",
         rc, onep_strerror(rc));
   goto cleanup;
}

//Create ACE40(seq=40, permit)
 rc = onep_acl_create_l3_ace(40, TRUE, &ace40);
 if(rc != ONEP_OK) {
   fprintf(stderr, "\nError in onep_acl_create_l3_ace: %d, %s\n",
         rc, onep_strerror(rc));
   goto cleanup;
}

//Set ACE40 src prefix
 rc = onep_acl_set_l3_ace_src_prefix(ace40, NULL, 0);
 if(rc != ONEP_OK) {
   fprintf(stderr, "\nError in onep_acl_set_l3_ace_src_prefix : %d
      , %s\n",
         rc, onep_strerror(rc));
   goto cleanup;
}

//Set ACE40 dest prefix
 rc = onep_acl_set_l3_ace_dst_prefix(ace40, NULL, 0);
 if(rc != ONEP_OK) {
   fprintf(stderr, "\nError in onep_acl_set_l3_ace_dst_prefix: %d,
      %s\n",
         rc, onep_strerror(rc));
   goto cleanup;
}

//Set ACE40 dest port
```

```
 rc = onep_acl_set_l3_ace_protocol(ace40, proto);
 if(rc != ONEP_OK) {
   fprintf(stderr, "\nError in onep_acl_set_l3_ace_protocol: %d, %
      s\n",
          rc, onep_strerror(rc));
   goto cleanup;
}
//Set ACE40 src port
 rc = onep_acl_set_l3_ace_src_port(ace40, 0, ONEP_COMPARE_ANY);
 if(rc != ONEP_OK) {
   fprintf(stderr, "\nError in onep_acl_set_l3_ace_src_port: %d, %
      s\n",
          rc, onep_strerror(rc));
   goto cleanup;
}
//Set ACE40 dest port
 uint16_t bgp_port = 179;
 rc = onep_acl_set_l3_ace_dst_port(ace40, 0, ONEP_COMPARE_ANY);
 if(rc != ONEP_OK) {
   fprintf(stderr, "\nError in onep_acl_set_l3_ace_dst_port: %d, %
      s\n",
          rc, onep_strerror(rc));
   goto cleanup;
}
//Add ACE40 to ACL
rc = onep_acl_add_ace(onep_acl, ace40);
if(rc != ONEP_OK) {
   fprintf(stderr, "\nError in onep_acl_add_ace: %d, %s\n",
          rc, onep_strerror(rc));
   goto cleanup;
}
/*
 * Get traffic action table
 */
 rc = dpss_tutorial_find_datapath_table(elem, &table_cap);
 if(rc != ONEP_OK) {
   goto cleanup;
 }
 /*
  * Create a policy using the class just created.
  */
```

```
/* 1. Create the op_list */
rc = onep_policy_pmap_op_list_new(&pmap_op_list);
if(rc != ONEP_OK) {
  fprintf(stderr, "\nError in onep_policy_pmap_op_list_new: %d, %
     s\n",
        rc, onep_strerror(rc));
  goto cleanup;
}
/* 2. Add the network element */
rc = onep_policy_op_add_network_element(pmap_op_list, elem);
if(rc != ONEP_OK) {
  fprintf(stderr, "\nError in onep_policy_op_add_network_element:
     %d, %s\n",
        rc, onep_strerror(rc));
  goto cleanup;
}
/* 3. Add pmap create operation to list */
rc = onep_policy_pmap_op_create(pmap_op_list, table_cap, &pmap_op
   );
if(rc != ONEP_OK) {
  fprintf(stderr, "\nError in onep_policy_pmap_op_create: %d, %s\
     n",
        rc, onep_strerror(rc));
  goto cleanup;
}
/* 4. Add an entry */
if(onep_policy_table_cap_supports_sequence_insertion(table_cap)){
    rc = onep_policy_pmap_op_entry_insert_sequence(pmap_op, 200,
       &entry_op);
    if(rc != ONEP_OK) {
      fprintf(stderr, "\nError in
         onep_policy_pmap_op_entry_insert_sequence: %d, %s\n",
            rc, onep_strerror(rc));
      goto cleanup;
    }
} else {
    rc = onep_policy_pmap_op_entry_insert_end(pmap_op, &entry_op)
       ;
    if(rc != ONEP_OK) {
      fprintf(stderr, "\nError in
         onep_policy_pmap_op_entry_insert_end: %d, %s\n",
            rc, onep_strerror(rc));
```

```
      goto cleanup;
    }
  }
  if (onep_policy_table_cap_supports_persistent(table_cap)) {
    rc = onep_policy_pmap_op_set_persistent(pmap_op, "onep-dp-
       tutorial-pmap");
    if(rc != ONEP_OK) {
      fprintf(stderr, "\nError in
         onep_policy_pmap_op_set_persistent: %d, %s\n",
            rc, onep_strerror(rc));
      goto cleanup;
    }
  } else {
    rc = onep_policy_pmap_op_set_transient(pmap_op);
    if(rc != ONEP_OK) {
      fprintf(stderr, "\nError in
         onep_policy_pmap_op_set_transient: %d, %s\n",
            rc, onep_strerror(rc));
      goto cleanup;
    }
  }
  if (onep_policy_table_cap_supports_cmap(table_cap)) {
    /*
     * Create a class based on the ACL.
     */

    /* 1. Create the op_list */
    rc = onep_policy_cmap_op_list_new(&cmap_op_list);
    if(rc != ONEP_OK) {
      fprintf(stderr, "\nError in onep_policy_cmap_op_list_new: %
         d, %s\n",
              rc, onep_strerror(rc));
      goto cleanup;
    }
    /* 2. Add the network element */
    rc = onep_policy_op_add_network_element(cmap_op_list, elem);
    if(rc != ONEP_OK) {
        fprintf(stderr, "\nError in
           onep_policy_op_add_network_element: %d, %s\n",
              rc, onep_strerror(rc));
        goto cleanup;
    }
```

```
/* 3. Create a specific operation on the list */
rc = onep_policy_cmap_op_create(cmap_op_list, table_cap, &
   cmap_op);
if(rc != ONEP_OK) {
     fprintf(stderr, "\nError in onep_policy_cmap_op_create:
        %d, %s\n",
            rc, onep_strerror(rc));
     goto cleanup;
}
if (onep_policy_table_cap_supports_persistent(table_cap)) {
    rc =  onep_policy_cmap_op_set_persistent(cmap_op, "onep-
       dp-tutorial-cmap");
    if(rc != ONEP_OK) {
      fprintf(stderr, "\nError in
         onep_policy_cmap_op_set_persistent: %d, %s\n",
            rc, onep_strerror(rc));
      goto cleanup;
    }
}
/* 4. Get the match holder for the operation instance */
rc = onep_policy_cmap_op_get_match_holder(cmap_op, &mh);
if(rc != ONEP_OK) {
  fprintf(stderr, "\nError in
     onep_policy_cmap_op_get_match_holder: %d, %s\n",
        rc, onep_strerror(rc));
  goto cleanup;
}
/* 5. Add an access list match */
rc = onep_policy_match_add_access_list( mh, (
   onep_policy_access_list_t *)onep_acl, &match);
if(rc != ONEP_OK) {
  fprintf(stderr, "\nError in
     onep_policy_match_add_access_list: %d, %s\n",
        rc, onep_strerror(rc));
  goto cleanup;
}
/* 6. Submit the operation. */
rc = onep_policy_op_update(cmap_op_list);
if(rc != ONEP_OK) {
  fprintf(stderr, "\nError in onep_policy_op_update 1: %d, %s
     \n",
        rc, onep_strerror(rc));
```

```
      goto cleanup;
    }
    /* 7. Find the cmap_handle we just created */
    rc = onep_policy_op_list_get_list(cmap_op_list, &result_list)
       ;
    if(rc != ONEP_OK) {
        fprintf(stderr, "\nError in onep_policy_op_list_get_list
           : %d, %s\n",
              rc, onep_strerror(rc));
        goto cleanup;
    }
    rc = onep_collection_get_iterator(result_list, &iter);
    if(rc != ONEP_OK) {
        fprintf(stderr, "\nError in onep_collection_get_iterator
           : %d, %s\n",
              rc, onep_strerror(rc));
        goto cleanup;
    }
    cmap_op = (onep_policy_cmap_op_t *)onep_iterator_next(iter);
        if (!cmap_op) {
          fprintf(stderr, "\nError in getting policy op\n");
          goto cleanup;
     }
     rc = onep_policy_cmap_op_get_handle(cmap_op, cmap_handle);
        if(rc != ONEP_OK) {
          fprintf(stderr, "\nError in creating class map : %d, %s
             \n",
                rc, onep_strerror(rc));
          goto cleanup;
    }
    /* 5. Set the cmap on the entry */
    rc = onep_policy_entry_op_add_cmap(entry_op, *cmap_handle);
    if(rc != ONEP_OK) {
      fprintf(stderr, "\nError in onep_policy_entry_op_add_cmap:
         %d, %s\n",
            rc, onep_strerror(rc));
      goto cleanup;
    }
} else {
    rc = onep_policy_entry_op_get_match_holder(entry_op, &mh);
    if(rc != ONEP_OK) {
```

```
      fprintf(stderr, "\nError in
         onep_policy_entry_op_get_match_holder: %d, %s\n",
            rc, onep_strerror(rc));
      goto cleanup;
    }
    /* 5. Add an access list match */
    rc = onep_policy_match_add_access_list( mh, (
       onep_policy_access_list_t *)onep_acl, &match);
    if(rc != ONEP_OK) {
      fprintf(stderr, "\nError in
         onep_policy_match_add_access_list: %d, %s\n",
            rc, onep_strerror(rc));
      goto cleanup;
    }
}
/* 6. Try and add an action */
rc = onep_policy_entry_op_get_action_holder(entry_op, &ah);
if(rc != ONEP_OK) {
  fprintf(stderr, "\nError in
     onep_policy_entry_op_get_action_holder: %d, %s\n",
        rc, onep_strerror(rc));
  goto cleanup;
}
if (action==ONEP_DPSS_ACTION_COPY) {
  printf ("Adding ONEP DPSS Action Copy\n");
  rc = onep_policy_action_add_copy(ah, callback, NULL, &dp_action
     );
  if(rc != ONEP_OK) {
     fprintf(stderr, "\nError in onep_policy_action_add_copy: %d,
        %s\n",
           rc, onep_strerror(rc));
     goto cleanup;
  }
}
if (action==ONEP_DPSS_ACTION_DIVERT) {
  printf ("Adding ONEP DPSS Action Divert\n");
  rc = onep_policy_action_add_divert(ah, callback, NULL, &
     dp_action);
  if(rc != ONEP_OK) {
     fprintf(stderr, "\nError in onep_policy_action_add_divert: %
        d, %s\n",
           rc, onep_strerror(rc));
```

```
      goto cleanup;
    }
  }
  if (action==ONEP_DPSS_ACTION_PUNT) {
    printf ("Adding ONEP DPSS Action Punt\n");
    rc = onep_policy_action_add_divert(ah, callback, NULL, &
       dp_action);
    if(rc != ONEP_OK) {
       fprintf(stderr, "\nError in onep_policy_action_add_divert: %
          d, %s\n",
             rc, onep_strerror(rc));
       goto cleanup;
    }
    rc = onep_policy_action_set_stateful(dp_action);
    if(rc != ONEP_OK) {
       fprintf(stderr, "\nError in onep_policy_action_set_stateful:
          %d, %s\n",
             rc, onep_strerror(rc));
       goto cleanup;
    }
  }
  /* 7. Submit the operation. */
  rc = onep_policy_op_update(pmap_op_list);
  if(rc != ONEP_OK) {
     fprintf(stderr, "\nError in onep_policy_op_update: %d, %s\n",
           rc, onep_strerror(rc));
     goto cleanup;
  }
  /* 8. Find the pmap_handle we just created */
  rc = onep_policy_op_list_get_list(pmap_op_list, &result_list);
  if(rc != ONEP_OK) {
    fprintf(stderr, "\nError in onep_policy_op_list_get_list: %d, %
       s\n",
          rc, onep_strerror(rc));
    goto cleanup;
  }
  rc = onep_collection_get_iterator(result_list, &iter);
  if(rc != ONEP_OK) {
    fprintf(stderr, "\nError in onep_collection_get_iterator: %d, %
       s\n",
          rc, onep_strerror(rc));
    goto cleanup;
```

```
    }
    pmap_op = (onep_policy_pmap_op_t *)onep_iterator_next(iter);
        if (!pmap_op) {
          fprintf(stderr, "Error in getting pmap_op\n");
          rc = ONEP_FAIL;
          goto cleanup;
         }
    rc = onep_policy_pmap_op_get_handle(pmap_op, pmap_handle);
        if(rc != ONEP_OK) {
          fprintf(stderr, "\nError in onep_policy_pmap_op_get_handle:
             %d, %s\n",
                rc, onep_strerror(rc));
          goto cleanup;
    }
   /* Return the acl we created */
   *acl = onep_acl;
   printf("Successfully created acl.\n");
   printf("Done creating policy handle.\n");
   cleanup:
   if(cmap_op_list) {
          destroy_rc = onep_policy_op_list_destroy(&cmap_op_list);
          if(destroy_rc != ONEP_OK) {
                fprintf(stderr, "\nError in
                   onep_policy_op_list_destroy: %d, %s\n",
                        destroy_rc, onep_strerror(destroy_rc));
          }
    }
   if(pmap_op_list) {
          destroy_rc = onep_policy_op_list_destroy(&pmap_op_list);
          if(destroy_rc != ONEP_OK) {
              fprintf(stderr, "\nError in
                 onep_policy_op_list_destroy: %d, %s\n",
                      destroy_rc, onep_strerror(destroy_rc));
           }
   }
   return rc;
}

/*
 * Simple packet callback that will just display some information per
 * packet. Can be used for diverted or copied packets and doesn't try
     to
```

```
 * take any action on the packet.
 */
void dpss_display_pak_info_callback(onep_dpss_traffic_reg_t *reg,
    struct onep_dpss_paktype_ *pak, void *client_context, bool *
       return_packet) {
        static int count = 1;    /* packet counter*/

    onep_status_t          rc;
    onep_dpss_fid_t        fid;
    char                   ipv = 0;
    uint16_t               src_port = 0;
    uint16_t               dest_port = 0;
    char                   *src_ip = NULL;
    char                   *dest_ip = NULL;
    char                   l4_protocol[5];
    char                   l4_state[30];
    bool                   *fragmented = false;
    uint32_t               size_payload = 0;
    uint8_t                *start_payload;
    int i = 0;
    strcpy(l4_protocol,"ERR");
    strcpy(l4_state,"ERR");
    char *content = NULL;
    uint8_t *payload;
    rc = onep_dpss_pkt_get_flow(pak, &fid);
    if( rc == ONEP_OK ) {
        rc = dpss_tutorial_get_ip_version(pak, &ipv);
        if( rc != ONEP_OK ) {
            fprintf(stderr, "Error in get ip version: code[%d], text
               [%s]\n",
                    rc, onep_strerror(rc));
        }
        rc = dpss_tutorial_get_ip_port_info(pak, &src_ip,
                                            &dest_ip,
                                            &src_port,
                                            &dest_port,
                                            l4_protocol,
                                            ipv);
        if( rc != ONEP_OK ) {
          fprintf(stderr, "Error in get ip port info: code[%d], text
             [%s]\n",
                rc, onep_strerror(rc));
```

```
    }
    dpss_tutorial_get_flow_state(pak, fid, l4_state);

} else {
    fprintf(stderr, "Error getting flow ID. code[%d], text[%s]\n"
        ,
           rc, onep_strerror(rc));
}
count++;
free(src_ip);
free(dest_ip);

rc = onep_dpss_pkt_is_fragmented(pak, &fragmented);
if(rc == ONEP_OK){
    if(!fragmented) {
        rc = onep_dpss_pkt_get_payload(pak, &start_payload);

        rc = onep_dpss_pkt_get_payload_size(pak, &size_payload);
        if((src_port == 179 || dest_port == 179) && size_payload
           > 0){
            char *url = "http://localhost:8000/evaluate/";
            content = do_web_request(url, payload, size_payload)
                ;
            cJSON* request_json = NULL;
            /* assuming the response is a string */
            request_json = cJSON_Parse(content);
            int bool_return = parse_object(request_json, &
               return_packet, &pak);
            char *hex_data = cJSON_GetObjectItem(request_json, "
               data")->valuestring;
            if(bool_return == 1) {
                            *return_packet = true;
            }
            else{
                *start_payload = "";
            }
        }
    }
} else {
    fprintf(stderr, "Error knowing if its fragmented. code[%d],
       text[%s]\n",
    rc, onep_strerror(rc));
```

```
    }
    return;
}
// END SNIPPET: callback_info
```

A.2 Parsing object Code

```
int parse_object(cJSON *request_json,
bool *return_packet,
struct onep_dpss_paktype_ *pak)
{
    onep_status_t        rc;
    int bool_return = NULL;
    cJSON* index = NULL;
    cJSON* optional = NULL;
    cJSON* hex_data = NULL;
    // request_json = cJSON_Parse(root);
    uint8_t *binary_data = 0;
    bool_return = cJSON_GetObjectItem(request_json, "return")->valueint;

    return bool_return;
}
```

A.3 Rib Information

```
{
    "local_bgp_id": "1",
    "prefixes": [
        {
            "address": "192.168.1.10",
            "netmask": "24"
        }
    ]
}
```

A.4 Topology discovery Request Example


```
{
    "Neighbors": [
        {
            "router_id": "10.10.10.120",
            "ip": "10.10.60.120",
            "bgp_state": "BGP state = Established",
            "as_id": "2"
        }
    ],
    "hostname": "Router7"
}
```


A.5 Evaluate Response Example


```
{
    "packet": {
        "dst_port": 27087,
        "src_port": 179,
        "type": 2,
        "name": "p1",
        "protocol": "bgp"
    },
    "data": "
       ffffffffffffffffffffffffffffffff003b020000001c4001010040020e020300000001
       ",
    "return": 1
}
```


A.6 Vmcloud topology configuration


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<topology xmlns="http://www.cisco.com/VIRL" xmlns:xsi="http://www.w3.
    org/2001/XMLSchema-instance" schemaVersion="0.3"
```

```
xsi:schemaLocation="http://www.cisco.com/VIRL http://cide.cisco.
com/vmmaestro/schema/virl.xsd">
 <node name="router1" type="SIMPLE" subtype="vios" location="
    188,263" vmImage="/usr/share/vmcloud/data/images/vios.ova">
     <extensions>
         <entry key="bootstrap_configuration" type="String">/home/
            cisco/vmcloud-example-networks/3node/router1.con</
            entry>
         <entry key="import_files" type="String">/home/cisco/
            vmcloud-example-networks/3node/router1.p12</entry>
     </extensions>
     <interface name="GigabitEthernet0/0" />
     <interface name="GigabitEthernet0/1" />
     <interface name="GigabitEthernet0/2" />
     <interface name="GigabitEthernet0/3" />
     <interface name="GigabitEthernet0/4" />
     <interface name="GigabitEthernet0/5" />
 </node>
 <node name="router2" type="SIMPLE" subtype="vios" location="
    488,319" vmImage="/usr/share/vmcloud/data/images/vios.ova">
     <extensions>
         <entry key="bootstrap_configuration" type="String">/home/
            cisco/vmcloud-example-networks/3node/router2.con</
            entry>
         <entry key="import_files" type="String">/home/cisco/
            vmcloud-example-networks/3node/router2.p12</entry>
     </extensions>
     <interface name="GigabitEthernet0/0" />
     <interface name="GigabitEthernet0/1" />
     <interface name="GigabitEthernet0/2" />
     <interface name="GigabitEthernet0/3" />
     <interface name="GigabitEthernet0/4" />
 </node>
 <node name="router3" type="SIMPLE" subtype="vios" location="
    371,407" vmImage="/usr/share/vmcloud/data/images/vios.ova">
     <extensions>
         <entry key="bootstrap_configuration" type="String">/home/
            cisco/vmcloud-example-networks/3node/router3.con</
            entry>
         <entry key="import_files" type="String">/home/cisco/
            vmcloud-example-networks/3node/router3.p12</entry>
     </extensions>
```

```
    <interface name="GigabitEthernet0/0" />
    <interface name="GigabitEthernet0/1" />
    <interface name="GigabitEthernet0/2" />
</node>
<node name="vmc_lan_1" type="SEGMENT" location="374,520" />
<node name="eth1" type="ASSET" location="671,235">
    <interface name="none0" />
    <interface name="none1" />
</node>
<node name="lan_ex" type="SEGMENT" location="722,161" />

<node name="router4" type="SIMPLE" subtype="vios" location="
   250,360" vmImage="/usr/share/vmcloud/data/images/vios.ova">
    <extensions>
        <entry key="bootstrap_configuration" type="String">/home/
           cisco/vmcloud-example-networks/3node/router4.con</
           entry>
        <entry key="import_files" type="String">/home/cisco/
           vmcloud-example-networks/3node/router4.p12</entry>
    </extensions>
    <interface name="GigabitEthernet0/0" />
    <interface name="GigabitEthernet0/1" />
    <interface name="GigabitEthernet0/2" />
    <interface name="GigabitEthernet0/3" />
    <interface name="GigabitEthernet0/4" />
</node>

<node name="router5" type="SIMPLE" subtype="vios" location="
   250,360" vmImage="/usr/share/vmcloud/data/images/vios.ova">
    <extensions>
        <entry key="bootstrap_configuration" type="String">/home/
           cisco/vmcloud-example-networks/3node/router5.con</
           entry>
        <entry key="import_files" type="String">/home/cisco/
           vmcloud-example-networks/3node/router5.p12</entry>
    </extensions>
    <interface name="GigabitEthernet0/0" />
    <interface name="GigabitEthernet0/1" />
    <interface name="GigabitEthernet0/2" />
    <interface name="GigabitEthernet0/3" />
</node>
```

```
<node name="router6" type="SIMPLE" subtype="vios" location="
    250,360" vmImage="/usr/share/vmcloud/data/images/vios.ova">
     <extensions>
          <entry key="bootstrap_configuration" type="String">/home/
              cisco/vmcloud-example-networks/3node/router6.con</
              entry>
          <entry key="import_files" type="String">/home/cisco/
              vmcloud-example-networks/3node/router6.p12</entry>
     </extensions>
     <interface name="GigabitEthernet0/0" />
     <interface name="GigabitEthernet0/1" />
     <interface name="GigabitEthernet0/2" />
     <interface name="GigabitEthernet0/3" />
</node>
<node name="router7" type="SIMPLE" subtype="vios" location="
    250,360" vmImage="/usr/share/vmcloud/data/images/vios.ova">
     <extensions>
          <entry key="bootstrap_configuration" type="String">/home/
              cisco/vmcloud-example-networks/3node/router7.con</
              entry>
          <entry key="import_files" type="String">/home/cisco/
              vmcloud-example-networks/3node/router7.p12</entry>
     </extensions>
     <interface name="GigabitEthernet0/0" />
     <interface name="GigabitEthernet0/1" />
     <interface name="GigabitEthernet0/2" />
     <interface name="GigabitEthernet0/3" />
</node>
<node name="router8" type="SIMPLE" subtype="vios" location="
    250,360" vmImage="/usr/share/vmcloud/data/images/vios.ova">
     <extensions>
          <entry key="bootstrap_configuration" type="String">/home/
              cisco/vmcloud-example-networks/3node/router8.con</
              entry>
          <entry key="import_files" type="String">/home/cisco/
              vmcloud-example-networks/3node/router8.p12</entry>
     </extensions>
     <interface name="GigabitEthernet0/0" />
     <interface name="GigabitEthernet0/1" />
     <interface name="GigabitEthernet0/2" />
     <interface name="GigabitEthernet0/3" />
</node>
```

```
<node name="router9" type="SIMPLE" subtype="vios" location="
    250,360" vmImage="/usr/share/vmcloud/data/images/vios.ova">
     <extensions>
          <entry key="bootstrap_configuration" type="String">/home/
              cisco/vmcloud-example-networks/3node/router9.con</
              entry>
          <entry key="import_files" type="String">/home/cisco/
              vmcloud-example-networks/3node/router9.p12</entry>
     </extensions>
     <interface name="GigabitEthernet0/0"/>
     <interface name="GigabitEthernet0/1"/>
     <interface name="GigabitEthernet0/2"/>
     <interface name="GigabitEthernet0/3"/>
     <interface name="GigabitEthernet0/4"/>
</node>

<!--<connection src="/topology/node[1]/interface[1]" dst="/
    topology/node[2]/interface[1]"/>-->
<connection src="/topology/node[1]/interface[2]" dst="/topology/
    node[7]/interface[1]"/>
<!-- Router intermig entre router1 i router 3 -->
<connection src="/topology/node[7]/interface[2]" dst="/topology/
    node[3]/interface[1]"/>
<connection src="/topology/node[7]/interface[3]" dst="/topology/
    node[4]"/>
<connection src="/topology/node[7]/interface[4]" dst="/topology/
    node[6]"/>
<!-- <connection src="/topology/node[1]/interface[2]" dst="/
    topology/node[3]/interface[1]"/> -->
<connection src="/topology/node[1]/interface[3]" dst="/topology/
    node[4]"/>
<connection src="/topology/node[2]/interface[2]" dst="/topology/
    node[4]"/>
<connection src="/topology/node[3]/interface[2]" dst="/topology/
    node[4]"/>
<connection src="/topology/node[3]/interface[3]" dst="/topology/
    node[6]"/>
<connection src="/topology/node[5]/interface[1]" dst="/topology/
    node[6]"/>
```

```
<connection src="/topology/node[1]/interface[4]" dst="/topology/
    node[6]"/>
<connection src="/topology/node[2]/interface[3]" dst="/topology/
    node[5]/interface[2]"/>

<connection src="/topology/node[2]/interface[1]" dst="/topology/
    node[8]/interface[1]"/>
<connection src="/topology/node[1]/interface[1]" dst="/topology/
    node[8]/interface[2]"/>
<connection src="/topology/node[8]/interface[3]" dst="/topology/
    node[4]"/>
<connection src="/topology/node[8]/interface[4]" dst="/topology/
    node[6]"/>

<connection src="/topology/node[2]/interface[5]" dst="/topology/
    node[9]/interface[1]"/>
<connection src="/topology/node[10]/interface[1]" dst="/topology/
    node[9]/interface[2]"/>
<connection src="/topology/node[9]/interface[3]" dst="/topology/
    node[4]"/>
<connection src="/topology/node[9]/interface[4]" dst="/topology/
    node[6]"/>
<connection src="/topology/node[10]/interface[2]" dst="/topology/
    node[4]"/>
<connection src="/topology/node[10]/interface[3]" dst="/topology/
    node[6]"/>

<connection src="/topology/node[1]/interface[6]" dst="/topology/
    node[11]/interface[1]"/>
<connection src="/topology/node[11]/interface[2]" dst="/topology/
    node[12]/interface[1]"/>
<connection src="/topology/node[11]/interface[3]" dst="/topology/
    node[4]"/>
<connection src="/topology/node[11]/interface[4]" dst="/topology/
    node[6]"/>
<connection src="/topology/node[12]/interface[2]" dst="/topology/
    node[4]"/>
<connection src="/topology/node[12]/interface[3]" dst="/topology/
    node[6]"/>

<connection src="/topology/node[12]/interface[5]" dst="/topology/
    node[7]/interface[5]"/>
```

```
</topology>
```

A.7 Routers configuration files

Router1

```
version 15.3
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname Router1
!
boot-start-marker
boot-end-marker
!
!
!
no aaa new-model
mmi polling-interval 60
no mmi auto-configure
no mmi pvc
mmi snmp-timeout 180
!
!
!
!
!
!
ip cef
no ipv6 cef
ipv6 multicast rpf use-bgp
!
multilink bundle-name authenticated
!
!
!
username cisco privilege 15 password 0 cisco
!
redundancy
```

```
!
!
!
!
!
!
!
interface GigabitEthernet0/0
 ip address 10.10.20.110 255.255.255.0
 duplex auto
 speed auto
 no shutdown
!
interface GigabitEthernet0/1
 ip address 10.10.30.110 255.255.255.0
 no shutdown
 duplex auto
 speed auto
!
interface GigabitEthernet0/2
 ip address 10.10.10.110 255.255.255.0
 no shutdown
 duplex auto
 speed auto
!
interface GigabitEthernet0/3
 ip address dhcp
 no shutdown
 duplex auto
 speed auto
!
interface GigabitEthernet0/4
 ip address 1.2.3.4 255.255.255.0
 duplex auto
 speed auto
 no shutdown
!
!
interface GigabitEthernet0/5
 ip address 10.10.80.110 255.255.255.0
 duplex auto
 speed auto
```

```
 no shutdown
!
ip route 10.10.40.130 255.255.255.255 10.10.30.140
!
ip route 10.10.50.120 255.255.255.255 10.10.20.150
!
ip route 10.10.90.190 255.255.255.255 10.10.80.180
!
router bgp 1
 bgp router-id 10.10.10.110
 bgp log-neighbor-changes
 neighbor 10.10.50.120 remote-as 2
 neighbor 10.10.40.130 remote-as 3
 neighbor 10.10.90.190 remote-as 6
 !
 address-family ipv4
  network 1.2.3.0 mask 255.255.255.0
  neighbor 10.10.50.120 activate
  neighbor 10.10.50.120 ebgp-multihop 2
  neighbor 10.10.50.120 next-hop-unchanged
  neighbor 10.10.40.130 activate
  neighbor 10.10.40.130 ebgp-multihop 2
  neighbor 10.10.40.130 next-hop-unchanged
  neighbor 10.10.90.190 activate
  neighbor 10.10.90.190 ebgp-multihop 2
  neighbor 10.10.90.190 next-hop-unchanged
 exit-address-family
!
ip forward-protocol nd
!
!
no ip http server
no ip http secure-server
!
!
!
!
control-plane
!
banner exec ^C
**********************************************************************
```

```
* vIOS - Cisco Systems Confidential                                    *
* Unauthorized use or distribution of this software is expressly       *
* Prohibited.                                                          *
**********************************************************************^
   C
banner incoming ^C
**********************************************************************

* vIOS - Cisco Systems Confidential                                    *
* Unauthorized use or distribution of this software is expressly       *
* Prohibited.                                                          *
**********************************************************************^
   C
banner login ^C
**********************************************************************

* vIOS - Cisco Systems Confidential                                    *
* Unauthorized use or distribution of this software is expressly       *
* Prohibited.                                                          *
**********************************************************************^
   C
!
line con 0
 password cisco
 login
line aux 0
line vty 0 4
 password cisco
 login
 transport input all
!
onep
! Added by griera in order to run DataPathTutorial
```

```
! datapath transport gre sender-id 2 interface GigabitEthernet0/2
 service set vty
!
 transport type tls localcert demoTP disable-remotecert-validation
 start
!
!
! IOS PKI will fail to import the tftp file if we attempt this before
! the config has been fully applied. So if we just do:
!   crypto pki import demoTP pkcs12 [location] [etc...]
! We would see something similar to this in the boot log:
!   *Nov 29 19:27:32.415: CRYPTO_PKI: Copying pkcs12 from flash1://
   bootstrap_admin.con
!   *Nov 29 19:27:32.492: %PKI-6-PKCS12IMPORT_FAIL: PKCS #12 Import
   Failed.
! Therefore we use a short delay before loading the pkcs12 file:
!
event manager applet load_identity
 event timer countdown name Delay time 20
 action 0.0 cli command "enable"
 action 1.0 cli command "config terminal"
 action 2.0 cli command "file prompt quiet"
 action 3.0 cli command "crypto pki import demoTP pkcs12 flash2://
    router1.p12 password cisco"
 action 4.0 syslog msg "Loaded bootstrap identity certificate"
!
end
```

A.8 Patch for dpkt

```
class ASPathSegment32bit(dpkt.Packet):
    __hdr__ = (
        ('type', 'B', 0),
        ('len', 'B', 0)
        )
    def unpack(self, buf):
        dpkt.Packet.unpack(self, buf)
        l = []
        for i in range(self.len):
```

```
            AS = struct.unpack('>I', self.data[:4])[0]
            self.data = self.data[4:]
            l.append(AS)
        self.data = self.path = l
    def __len__(self):
        return self.__hdr_len__ + \
               4 * len(self.path)
    def __str__(self):
        as_str = ''
        for AS in self.path:
            as_str += struct.pack('>I', AS)
        return self.pack_hdr() + \
               as_str
```

A.9 Gunicorn Configuration file

```
#!/bin/bash

NAME="bgp-server" # Name of the application
DJANGODIR=/path/to/app # Django project directory
SOCKFILE=/path/to/app/run/gunicorn.sock  # we will communicte using this u
USER=user           # the user to run
GROUP=group         # the group to run as
NUM_WORKERS=9       # how many worker processes should Gunicorn spawn
DJANGO_SETTINGS_MODULE=name_app.settings # which settings file should Djang
DJANGO_WSGI_MODULE=name_app.wsgi         # WSGI module name

echo "Starting $NAME as `whoami`"

# Activate the virtual environment
cd $DJANGODIR
source ../bin/activate
export DJANGO_SETTINGS_MODULE=$DJANGO_SETTINGS_MODULE
export PYTHONPATH=$DJANGODIR:$PYTHONPATH

# Create the run directory if it doesn't exist
RUNDIR=$(dirname $SOCKFILE)
test -d $RUNDIR || mkdir -p $RUNDIR

# Start your Django Unicorn
# Programs meant to be run under supervisor should not daemonize themselves
```

```
exec gunicorn ${DJANGO_WSGI_MODULE}:application \
  --name $NAME \
  --workers $NUM_WORKERS \
  --user=$USER --group=$GROUP \
  --bind=unix:$SOCKFILE \
  --log-level=debug \
  --log-file=-
```