Securing Communication in Multiple Autonomous System Domains with Software Defined Networking

Vijay Varadharajan*, Kallol Krishna Karmakar†, Udaya Tupakula*
Advanced Cyber Security Research Centre
Faculty of Science, Macquarie University, Sydney, Australia
[vijay.varadharajan,udaya.tupakula]@mq.edu.au*, kallol.karmakar@students.mq.edu.au†

***Abstract*—In this paper we proposed policy based security architecture for securing the communication in multiple Autonomous System (AS) domains with Software Defined Networks (SDN). We will present a high level overview of the architecture and detail discussion on some of the important components for securing the communication in multiple AS domains. A key component of the security architecture is the specification of security policies that are to be enforced on the SDN communications whether they are intra or inter-domain. We will present example scenarios to demonstrate the operation of the security architecture to enable end-to-end secure communication within a single AS domain and for multiple AS domains. We have justified the model using ONOS controller.**

***Index Terms*—Software Defined Networking(SDN) Security, OpenFlow, Policy Control.**

I. Introduction

As enterprise networks and data centres expand in size and complexity, they pose greater administrative and management challenges. Increasingly, current networks are highly heterogeneous with many different devices, from small sensors and appliances to network devices such as routers to many different clients and servers and peripherals. Furthermore, these devices use different network technologies such as fixed, wireless and mobile networks. In such a complex environment, proprietary vendor specific software and tools for managing network devices such as switches, routers and gateways, the mobility of users and devices, the dynamic variation in networks due to failure of devices and network links, as well as dramatic increase in security attacks and shortage of skilled professionals are posing serious challenges. Software Defined Networks (SDN) is an emerging technology [1] that offers a promising approach to meeting some of these challenges. SDN is rapidly emerging as a disruptive technology, poised to change communication networks much the same way cloud computing is changing the compute world. It is altering the texture of modern networking, moving away from the current control protocols dominant in the TCP/IP Internet stack, towards something more flexible and programmable.

It is potentially changing the way networking will be conducted in the future, by enabling devices that are open and controllable by external open software, unlike today's proprietary network equipment that has protocols embedded into them by vendors. In this sense, SDN opens up new avenues of research to realize network capabilities that were impossible or extremely cumbersome before, thereby helping to make future networks more manageable and practicable. The separation of the control plane from the data plane by SDN results in the network switches becoming simpler forwarding devices with the control logic implemented in software in a logically centralized Controller. This decoupling in the SDN enables the design of new innovative network functions and protocols. First, it is simpler and less error-prone to modify network policies through software, than via low-level device configurations. Second, a control program can automatically react to spurious changes of the network state and thus maintain up to date high-level policies. Third, the centralization of the control logic in the Controller with network wide knowledge can help to simplify the development of sophisticated network functions. Although SDN offers several advantages to deal with complexities in current networks, a critical issue in SDN at present is that of security; the current state of the art in security in SDNs is still at its infancy [2]. Securing networks is becoming more challenging to businesses, especially with bring your own devices (BYOD), increased cloud adoption and the Internet of Things.

In this paper we present the design and implementation of security architecture for end-to-end communication in SDN. A key component of the security architecture is the specification of security policies that are to be enforced on the SDN communications whether they are intra or inter-domain. First we will consider a Policy based Security Architecture for intra-domain interactions and then we will address the inter-domain communications enabling end to end SDN services across multiple domains. We will present a high level overview of the architecture and then describe each component in detail. We have justified the model using ONOS controller.

In Section II, we describe the Policy based Security Architecture for end-to-end services and detail discussion on the components that are related to end-to-end secure communication in multiple domains. Section III presents the implementation of our model using ONOS SDN Controller, Mininet and Virtual Box. Then we present some of the results for inter-domain scenarios. Section IV considers some related work and Section V concludes.

II. Policy based Security Architecture for SDNs

In this section, we will present an overview of the security application architecture for SDN Controllers and detail discussion on some of the important components that are used

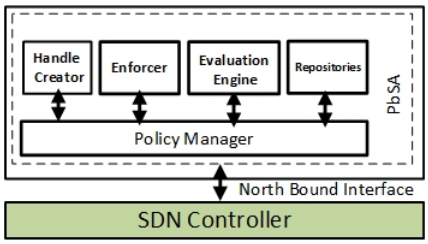

Fig. 1: Policy based Security Architecture for SDN

for securing the end-to-end communication in multiple AS Domains. Section II-A describes the network setup and presents an overview of the security application architecture for SDN Controller. Topology repository and Policy Repository are the important subcomponents of the PbSA that are related to end-to-end security in multiple AS domains. Hence Section II-B describes the topology repository and Section II-C describes the policy repository. We will also describe the language based specification for security policies that is developed for communication between the Controllers. Finally, Section II-D considers some of the cases for conflict resolution.

A. Policy based Security Application

A distributed network typically consists of a number of Autonomous System domains (ASs) and each domain has packet forwarding devices such as routers, gateways and switches, and end hosts which are connected to users. There may be a hierarchy of SDN controllers but for simplicity, we assume that each domain contains one SDN Controller. End hosts are connected to the forwarding devices. For clear specification of policies, we will assume that each AS has a separate entry and exit gateways. These are OpenFlow supported forwarding devices. The traffic generated by the end hosts and forwarded by the network devices within the network is subjected to policies specified in the Controller. The distributed SDN environment has both intra-domain and inter-domain communications. In an intra-domain communication, the traffic from source to destination passes through devices with a single SDN domain and the requested services are provided by the servers and devices in the same domain. In this case, the traffic and service requests are subjected to security policies in the SDN Controller of that domain. Inter-domain communications involving multiple domains requires cooperation between SDN Controllers, as the communications are subject to security policies in multiple Controllers.

First we will consider a Policy based Security Architecture for intra-domain interactions and then we will address the inter-domain communications enabling end to end SDN services across multiple domains. We will present a high level overview of the architecture and then describe some of the important components in detail.

Figure 1 shows the Policy based Architecture for securing the communication in a SDN domain. The Policy based Security Architecture can either form part of the SDN Controller or can run as a Security Application on top of the SDN Controller. We have designed and developed the Policy based Security Architecture as a Security Application running on top a SDN Controller for flexibility reasons. We will refer to this Policy based Security Application (PbSA). PbSA is implemented in the NorthBound API interface of the Controller. As PbSA is designed in a modular fashion, the components of PbSA can be implemented on a single host or distributed over multiple hosts.

We assume that each AS domain is controlled by a single logical SDN Controller. Each Controller maintains a repository for storing the information related to the topology and security policies. Topology repository and Policy repository are the important components that are used for establishing secure end-to-end communication between the hosts in multiple AS domains. The topology repository stores the topological information of the neighboring domains. It also stores the Security Labels of each SDN controller. Security Labels are statically assigned based on SDN controller making and reputation of that particular AS domain. Policy Repository is used for storing the security policies for communication between the nodes. Policies are stored in a XML repository. Each policy in Policy Repository is termed as one Policy Expression. These Policy Expressions control the AS domain activity as well as the incoming packet activity in that particular domain. A detail discussion on these repositories is presented in the following sub Sections. The core component of PbSA is a Policy Manager, which manages every single operation of the security system. It is the coordinator of the whole AS policy routing system. Policy manager makes use of Evaluation Engine to evaluate the incoming network traffic (eg packet in messages) and determine the relevant security policies for that specific traffic. Following the evaluation, the Policy Manager determines the flow rules which are then conveyed to the Enforcer module. The Enforcer is used for enforcing the policies related to the flows. A Packet Handle Creator module creates the necessary handles for AS domains which is piggy backed with the payload from the Policy Manager. These handles are used to check the authenticity of the packet and the enforcement of policies at the switches which is discussed in detail in Section II-C2.

Now we described the Topology and Policy repositories in detail.

B. Topology Repository

For inter-domain routing of traffic across multiple domains requires an SDN Controller in one AS domain to have knowledge of other Controllers in other AS domains. We have used a pre-built topological map of different AS domains. We have included an additional security attribute, a Security Label for each AS domain SDN Controller. The intention of the Security Label is to reflect the level of secureness that particular Controller have. In our current architecture, this Security Label is hard wired (static) and is specified at the time of installation of the Controller. They are statically assigned based on device meta data such as vendor, making, manufacturer reputation etc. For this implementation purpose only, we have chosen four Security Labels where, SL1 being the least and SL4

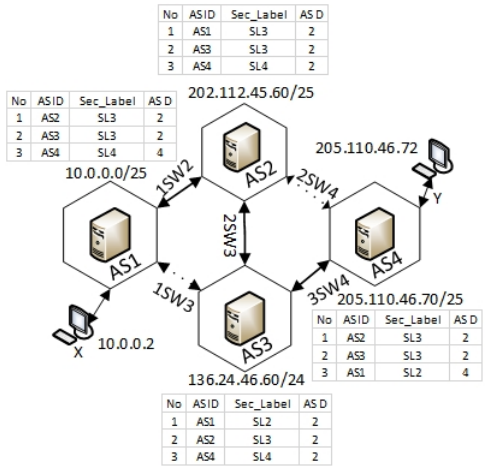

Fig. 2: Topology Views at the Controllers

being the highest Security Label. Hence each AS domain SDN Controller now has the ability to create the topological information as well as the levels of security associated with all the neighbouring AS domain Controllers in this Repository. Consider the distributed SDN environment shown in Figure 2 and the associated Topology Repositories are shown in Tables with in the Figure. Each hexagon in Figure 2 represents an AS domain. We have represented the AS Gateways using Gateway OpenFlow Switches. The OpenFlow Switches are represented using the notation ([source AS ID]SW[destination AS ID]). Hence the Switch connecting AS1 to AS2 will be represented as 1SW2. In each topology table, *AS_ID* is the identity of a particular AS, *Sec_Label* is the Security Label of the AS domain, AS D represents the distance in terms of number of ASs between the source and destination.

C. Policy Repository and Security Policy Specifications

Policy Repository is used for storing the security policies. A key component of the security architecture is the specification of security policies that are to be enforced on the SDN communications whether they are intra or inter-domain. Policies stored in the repository are granular and can also help in managing the SDN network domain, such as managing QoS of video traffic, blocking the P2P traffic for better network performance etc.

We have adopted a simple language based approach to specify the security policies; in particular, we have chosen the policy based routing syntax specified in RFC1102 [3] as the basis for our security policy specifications. Policies are rules that specify whether packets follow a particular path or paths in the network and the conditions under which the packets follow these paths. In our language, we have policy terms specifying a range of attributes associated with the flow and the entities in the SDN. These include the following:

- Flow Attributes: Flow ID, sequence of packets associated with the Flow, type of packets, Security Profile indicating the set of security services that are to be associated with the packets in the Flow
- Autonomous System Domain Attributes: AS Identities such as Source AS and Destination AS Identities, Identities of Entry and Exit Gateway/Switch to AS, AS Type (e.g. Commercial Domain, Government Domain) and Security Label associated with the AS.
- Switches Attributes: Identities of the Switches and Security Label of the Switches
- Host Attributes such as Identities of Hosts such as Source Host ID and Destination Host ID
- Constraints such as Flow Constraints (FlowCons) and Domain Constraints (DomCons) associated with a Flow
- Services (for which the Policy Expression applies),
- Time Validity (the period for which the Policy Expression is valid) and
- the Path (In case of intra-domain it indicates a specific sequence of switches and in the case of inter-domain communications it indicates the sequence of Autonomous Systems traversed by a Flow)

The flow constraints are conditions that apply to specific flows or sets of flows. For instance, a constraint might specify the flow of packets of a specific type (e.g video) should only go through a set of switches that can provide a certain bandwidth; or from a security point of view, a constraint could be that a flow should only go through AS domains that are at a particular Security Level. Domain constraints are applied to all flows within a domain. They are used to specify domain-wide policies. For instance, there could be a domain wide security policy which may specify that all flows should be protected for integrity, as part of the security profile. These constraints are used as part of the actions associated with the Policy Expressions.

Alternatively it is also possible to enforce specific paths by explicitly specifying the set of switches through which a flow must go through or a specific set of AS domains that should be traversed.

The policy language has wild cards in its syntax enabling specification of policies that can apply to sets or groups of entities and services. When a Policy Expression is satisfied, then the associated action is performed which could be simple as just allow or deny the request. Hence using these policy terms, one is able to specify different sets of Policy Expressions to deal with a range of scenarios in both intra-domain and inter-domain communications in a distributed SDN environment.

1) **Intra Domain Scenarios:** First let us consider some simple intra-domain scenarios in a single SDN domain. Using the policy terms mentioned above, a simplified Policy Expression template could be as follows:

$PE_i = < FlowID, SourceAS, DestAS, SourceHostIP, DestHostIP, SourceMAC, DestMAC, User, FlowCons, DomCons, Services, Sec-Profile, Seq-Path >:< Actions >$

where i is the Policy Expression number.

In general, we will have a number of Policy Expressions stored in the SDN Policy Repository. Such a template will enable us to specify a range of policies for different users,

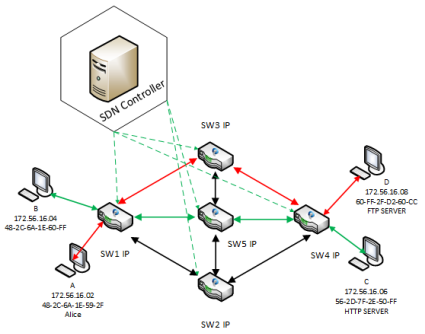

Fig. 3: Intra-Domain Example Scenario

from different locations, accessing different services using different devices following different paths. We will illustrate the use of the Policy Expression using 3 examples below.

Example 1: User Alice accessing a Service from a Specific Device Let us assume that Alice has registered her device with the IP address and the MAC address of the specific device are 172.56.16.02 and 48-2C-6A-1E-59-2F respectively (e.g. as part of BYOD policy) as shown in Figure 3. Then the Policy Expression is as follows:

$PE_2 = < *, *, *, 172.56.16.02, *, 48-2C-6A-1E-59-2F, *, *, Alice, *, 80, conf, * >:< Allow >$

This scenario is more specific to organizations where employees try to access from their assigned access points.

According to the Policy Expression, this Policy is associated with a user Alice. Whenever she tries to access via HTTP traffic from the specified host machine, she will be provided a confidentiality service by the Controller. The AS related conditions, host traffic destinations have wild cards. Flow IDs as well as the switch sequences for path have also got wild cards in this scenario.

Example 2: Accessing Specific Services This scenario considers access to specific services such as HTTP and FTP to be allowed for any user. For example, a Hospital may maintain a patient database in an FTP server and any doctor can access the patient database from his or her assigned access point. One may have additional constraints such as the communications between the patient database server and the access points should be protected for confidentiality and integrity. The Policy Expression for such a scenario could be as follows:

$PE_1 = < *, *, *, *, *, *, *, *, *, *, (20; 21; 22; 23), \{Conf, Intg\}, * >:< Allow >$

Policy Expression specifies that the FTP traffic from any host to any destination within the hospital is protected for confidentiality and integrity.

Example 3: Path Based Policy for Services This scenario can specify specific paths for specific services requested from the hosts. Consider for example, we are running HTTP and FTP servers on two machines with IP 172.56.16.06 and 172.56.16.08 respectively. We have created a OpenFlow switch matrix with five switches as shown in Figure 3. Two Hosts 172.56.16.02 and 172.56.16.04, we want to guide their flows through this switch matrix based on the services they are trying to access. The Policy Expressions could be as follows:

PE_5=< $*, *, *, 172.56.16.04, 172.56.16.06, 48 : 2C : 6A : 1E : 60 : FF, *, *, *, *, 80, \{Conf, Intg\}, (SW1; SW5; SW4)$ >:< $Allow >$

PE_6=< $*, *, *, 172.56.16.02, 172.56.16.08, 48 : 2C : 6A : 1E : 59 : 2F, *, *, *, *, (20; 21; 22; 23), Conf, (SW1; SW3; SW4)$ >:< $Allow >$

First Policy Expression says that flows from host with IP 172.56.16.04 and MAC 48:2C:6A:1E:60:FF accessing the HTTP server on IP 172.56.16.06, should be protected for confidentiality and integrity and forwarded via OpenFlow switch SW1->SW5->SW4.

According to the second Policy Expression, flows to the FTP server running on IP 172.56.16.08 from host with the IP address 172.56.16.02 and MAC address 48:2C:6A:1E:59:2F are to be forwarded via a OpenFlow switch path SW1->SW3->SW4. This Policy Expression also instructs the Controller to protect the flow for confidentiality.

2) **Inter Domain Scenarios:** Let us now consider a simplified Policy Expression template using the policy terms mentioned above.

$PE_i^{AS_k} =< FlowID, SourceASAttributes, DestASAttributes, SourceHostIP, DestHostIP, SourceMAC, DestMAC, User, FlowCons, DomCons, Services, Sec - Profile, Seq - Path >:< Actions >$

where k is the AS ID and i is the Policy Expression number. In an inter-domain scenario, the various attributes in the Policy Expression are as follows:

Source AS Attributes include the following:

- AS Domain ID
- Source Subnet(SRC_{SUB}) = Source Subnet address from which the incoming packets originate.
- Source AS Controller Entry Switch (SRC_{ENT}) = Address of the entry OpenFlow Switch connected to the SDN Controller of the AS domain
- Source AS Type (SRC_{Type}) = Examples include government domain GOV or commercial domain COM or education domain EDU
- Security Label (SRC_{SL})= Security Label of Source AS (optional)

Destination AS Attributes include the following:

- Destination Subnet (DST_{SUB}) = Destination Subnet address where the packets are to be transferred.
- Destination AS Type (DST_{Type}) = Type of the Destination AS
- Security Label (DST_{SL}) = Security Label of the destination AS (optional)

Source/ Destination Host specific parameters:

- Source Device IP (SRC_{IP}) = IP address of the host machine from where the packet originates.

- Source Device MAC (SRC_{MAC}) = MAC address of the host machine from where the packet originates.
- Destination Device IP (DST_{IP}) = IP Address of the destination host machine.
- Destination Device MAC (DST_{MAC}) = MAC Address of the destination host machine.

FlowCons and DomCons: As mentioned earlier, FlowCons and DomCons represent conditions that need to be satisfied by the path taken by the traffic in the SDN environment. The Path (AS_{SEQ}) is used to explicitly specify a list of AS Domains that is to be traversed by the traffic.
Packet Type (PKT_{TYP}) indicates the type of the incoming packets. In some cases, port numbers are used to indicate the type.
Time (T^{PE_i}) represents the duration time for which a particular Policy Expression is valid. A wild card implies the policy is valid from the start and there is no time limit.
Actions: Each Policy Expression has an action associated with it. The FlowCons and DomCons impose certain conditions associated with the actions. For instance, if there is a flow constraint which requires that this specific flow of packets of a specific type should only go through a set of switches that can provide a certain bandwidth, then the action requires that appropriate flow rules be dynamically configured into a set of switches enabling a path that satisfies this constraint. Similarly, from a security point of view, if there is a flow constraint that requires only AS domains that are at a particular Security Label be traversed by the specific traffic in question, then the action will require the selection of a path with appropriate AS domains. An action can also have some attributes. For instance, Destination Exit Switch (DST_{EXT}) attribute associated with an action indicates that the exit switch that should be taken by the traffic when the policy is satisfied.
Example: Secure Policy based Routing in Inter-Domain Scenarios The PbSA's Topology Repository in each SDN Controller of the AS domain stores information about the neighbouring AS domain SDN controllers. Here in policy based routing of SDN controller AS domains packets are being guided by the policies stored in the Policy Expression Repository.
In this section, we consider secure policy driven routing and communications in inter-domain scenarios involving multiple AS domains and SDN Controllers. Routing process begins from the host that is generating the packets and the request which is the source of the communication. This source end host could be any client such as a mobile device. The initial packet from the end host is sent by the switch (to which this end host is connected) to the SDN Controller in the AS domain. The host packet contains all the usual network and service parameters such as the source address, the packet type. The PbSA application in the Controller extracts the relevant parameters from the incoming packets and uses the Policy Repository and the Policy Manager to determine whether the relevant Policy Expressions are satisfied. We have seen some examples of the Policy Expressions above in the previous section. If the Policy Expressions are valid for the incoming packets, then PbSA will enforce the specified actions as flow rules in the appropriate data plane devices such as switches to transfer the packets.
With inter-domain communications, there is a need to transfer the packets to other AS domains. For such inter-domain scenarios, we introduce two further concepts. First is the notion of a *Handle*, which the the PbSA creates and tags it to the Packet. The Handle consists of a list of visited AS domain IDs. The Packet + Handle is then transferred to the next AS Domain Controller. Similar process is being repeated as the packet goes through all the transit AS domain SDN Controllers until the packet reaches its destination. The second concept is that of a *Policy Transfer Token*. A Policy Transfer Token contains the constraints that are transferred from the current AS domain to the subsequent transit AS domains which need to be satisfied as the packet is being transferred. These constraints need to be taken account in addition to the constraints of the transit domains. For instance, if there is a constraint that the traffic should only pass through AS domains of Security Label greater than a certain threshold, then this constraint needs to be satisfied by subsequent transit domains. Suppose an AS domain SDN Controller (with AS ID = 10) has a constraint that packets should only be forwarded through a path of AS domain SDN Controllers that have a Security Label higher that say $SL3$. In distributed systems, in general, it is not possible for one domain Controller to know about policies of other domains. Hence there is a need to transfer the policy constraints, which are communicated via Policy Transfer Tokens. In this paper, we assume that the AS domain Controllers are trusted and that if and only if the policies can be satisfied in the domain, the receiving Controller will accept the packets. In terms of notation, we denote the Handle as $H_i^{AS_k}$ which is tagged to the packet , where i is the Handle number for a particular communication *i* and *k* is the ID of the AS domain SDN Controller which created the Handle. Similarly, the Policy Transfer Token is denoted as $PTT_i^{AS_k}$, where again i denotes the specific communication and k denotes the ID of the AS domain. Hence an AS domain SDN Controller creates the Augmented Packet using the original Packet as well as the Handle and the Policy Transfer Token.
Let us now illustrate the above concepts using a simple inter-domain scenario in Figure 2, which shows two domains AS1 and AS4 connected via domains AS2 and AS3. Each AS domain has a SDN Controller and PbSA with its Policy and Topology Repositories as described above. The Tables beside each of the AS domains in Figure 2 represent the Topology Repository. We have not counted the intermediary node for hop counts between AS domain for simplicity. The Table I shows the policies in each of these AS domains stored in the Policy Repositories of PbSA.
In this scenario, the host machine X (with IP address 10.0.0.2) wishes to communicate host machine Y (with IP address 205.110.46.72). As X and Y reside in two different AS domains, communication between them occurs via transit AS domains. At first, packets from X goes to the SDN Controller

TABLE I: Stored Policy Expressions

AS ID	Policy Expression
AS1	$PE_1^{AS1} =< *, (10.0.0.0/24, EDU, SL2), (205.110.46.70/25), 10.0.0.2, *, *, *, *, *, *, (80, 443), conf, * >:< (SL2+ =) >$
AS2	$PE_4^{AS2} =< *, (10.0.0.0/24, EDU, SL2), (205.110.46.70/25), 10.0.0.2, *, *, *, *, *, *, (80, 443), conf, (AS3) >: (SL2+ =) >$
AS3	$PE_2^{AS3} =< *, (10.0.0.0/24, EDU, SL2), (205.110.46.70/25), *, *, *, *, *, *, *, (80, 443), conf, * >: (SL2+ =) >$
AS4	$PE_1^{AS4} =< *, (10.0.0.0/24, EDU, SL2), (205.110.46.70/25), 10.0.0.2, 205.110.46.72, *, *, *, *, *, (80, 443), conf, *) >:< Allow >$

of AS1. As the policy expression PE1 in AS1 matches with this particular network traffic, HTTP & HTTPS traffic originating from 10.0.0.2 is allowed to go to Y in the subnet 205.110.46.70/25. However let us assume that there is a flow constraint which specifies communications between X and Y must occur only through domains which have Security Labels greater than or equal to $SL2$. This will result in the traffic routed through the AS2 domain via the OpenFlow Switch *1SW2*(Connected to AS1). A similar process occur in AS2 and the traffic will be sent to Y in AS4.

D. Security Policy Conflicts and Error Handling

As discussed above, in inter-domain scenarios, each AS domain has its own security policies within its PbSA in the SDN Controller. When inter-domain communications occur, the constraints attached to specific flows are transferred as part of Policy Transfer Tokens in our architecture. This leads to two types of mismatches. One type is the actual conflicts between policies in different domains and the other type is due to mismatch in capabilities available in different domains. Conflicts in policies can occur if the policies associated with a flow arising from one domain contradicts with policies in another domain. There are several ways in which such conflicts can arise. For instance, a simple case of conflict could be that the traffic coming out of Domain I may need to be destined to a host in Domain K, which is passing through Domain J. However Domain J may have a policy which blacklists Domain K and hence it will not send any traffic to any host in Domain K. As security policies in each domain is independent and not known to other domains, Domain I will not be aware of the security policies of Domain J. Another example could be that Domain J requires for all traffic being transferred in its domain, the traffic must be signed, that is, there must be source authentication to counteract spoofing. However traffic from Domain I may not have such constraint, and hence once again a policy conflict arises. Then there is the second type, which is due to different capabilities available in different domains. For example, there may be a flow constraint which requires a certain minimum bandwidth for the traffic. The transit domain, e.g. Domain J, may not be able to satisfy this requirement. It is necessary for the security architecture to handle both these types of policy conflicts and mismatches in capabilities between domains.

We have customized ICMP error messages to handle these two types of mismatches in our security architecture. These error messages ripple back to the PbSA enhanced SDN Controller in the source AS domain via the same path taken by the original traffic. This can be achieved using the handle which contains the relevant parameters about the original path taken by the traffic. The ICMP error messages contain the hash of the relevant parameters such as the source and destination IP and service information belonging to the original packet for which the mismatch occurred. This implies that domain SDN Controllers need to store these parameters for all the packets being transferred for a period of time. In our architecture, we store them in the form of hashed digests. When a particular error message comes to an AS domain SDN Controller, it extracts the hash from the ICMP error message and identifies the required traffic by matching it with the stored hashes. The different types of mismatches within the error message have the following parameters:

$MismatchMessage = [Cause\ of\ Mismatch] + [Domain\ where\ Mismatch\ Occurred] + Hash(PacketHeader, ServiceInfo, Payload)$

III. Implementation

Here, we highlight the implementation details of the SDN network and PbSA application. In Section III-A we describe the functional modules of PbSA application and Section III-B illustrates implemented Policy Repository. Finally, we represent our findings in Section III-C.

We have validated our Policy based Security Architecture for SDN using Open Network Operating System(ONOS). We have extended the SDN-IP application in ONOS by adding extra modules for policy control in SDN inter-domain. We have simulated the environment using Mininet and ONOS running on Oracle VM BOX. For simulation, we are using a Core i7 - 4790 @3.60 GHz CPU with 32 GB of RAM. The setup is similar to Figure 2 . We have added bridged mode Ethernet adapters to the VM pairs running within VM box to make communication channels between each pair and each ONOS VM talking to Mininet VM. We have created ONOS and Mininet VM pairs, which acts as a single SDN AS domains. Each AS domain SDN controller runs PbSA. We have activated PbSA in all the ONOS SDN controllers. Policies are installed and stored in a XML repository. We have used Java parser to update the XML policy repository. We now present some specific implementation details.

A. Application Modules

Figure 4 shows the different modules used in the Policy based Security Application. We have combined our application with SDN-IP, the ONOS built-in application which is used for BGP listening. In particular we made use of SDN-IP features such as BGP route selection, update, maintaining sessions for creating & updating our topology repository. We present brief overview of the modules related to our implementation.

SDN-IP core consists of a software router which captures best BGP routes from BgpSessionManager. BgpSessionManager maintains sessions, updates and selects the best BGP routes.

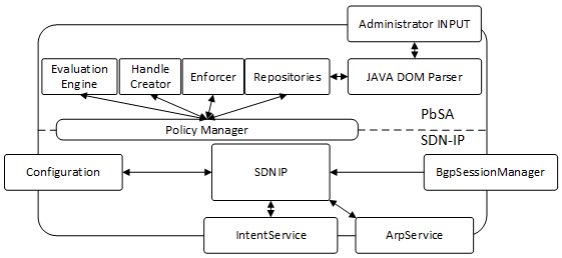

Fig. 4: Software modules for the Policy based Security Application

```
<policies>
    <id>1</id>
    <flowID>*</flowID>
    <srcasatt>
        <asid>*</asid>
        <assub>10.0.0.0/24 </assub>
        <astyp> EDU</astyp>
        <astrulavel>TL2</astrulavel>
    </srcasatt>
    <dstasatt>
        <asid>*</asid>
        <assub>205.110.46.70/25</assub>
        <astyp>EDU</astyp>
        <astrulavel>TL2</astrulavel>
    </dstasatt>
    <hostatt>
        <srcip>10.0.0.2</srcip>
        <dstip>205.110.46.72</dstip>
        <srcmac>*</srcmac>
        <dstmac>*</dstmac>
        <user>*</user>
    </hostatt>
    <flowcons>(80, 443)</flowcons>
    <domcons>*</domcons>
    <seq>*</seq>
    <secprofile>conf</secprofile>
    <action>Allow</action>
</policies>
```

Fig. 5: AS4 PbSA policy repository

ArpService is used for resolving MAC addresses. ONOS core configurations are captured by the Configuration module. IntentService submits the requests to the ONOS controller in the form of intents. With the help of these modules SDN-IP listens to BGP requests and chooses the best route from them. SDN-IP suffers from some of the traditional issues which we have mentioned in the related section. Our target here is to use SDN-IP to implement our proposed policy based secure interdomain architecture.

The next phase is implementation of different modules of our application. Here we have used an XML Repository for storing the policies. To update, modify and read the stored XML policies we have used JAVA DOM XML parser module. Policy Manager is the core of the Policy based Security Application. It maintains all the modules in the application. Enforcer helps Policy Manager in enforcing flow rules. It also interacts with the topological information from SDN-IP and updates our Topological Repository. Policy Manager is aware of the topological information. Before setting up any new flows between the available domains, Policy Manager checks the XML policy repository. Evaluation Engine module helps Policy Manager in the decision making process. It checks the policies and sends the particular action to the Enforcer, which is conveyed by Policy Manager to the intent services. These intent requests are captured by the ONOS controller and appropriate action is being taken. Handle creator is used to create the packets with policy tokens.

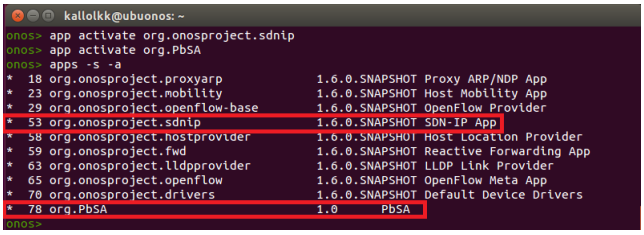

Fig. 6: Active ONOS applications

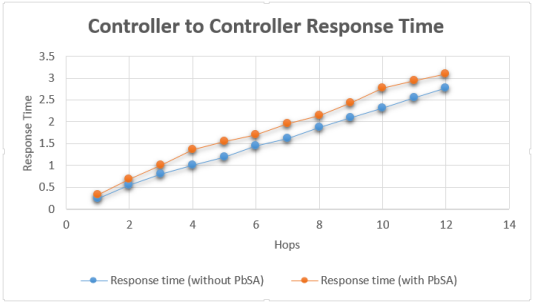

Fig. 7: Controller to Controller Response Time

B. Repository

Figure 5 shows policy repository of AS4 (VM Pair 4. Policy expresses that packets originated from subnet 10.0.0.0, host 10.0.0.2 with any (*) MAC address, whose destination subnet is 205.110.46.70/25 will be forwarded to Host Y with IP 205.110.46.72. The traffic must be HTTP/HTTPS traffic.

C. Findings

Figure 6 shows the Policy based Security Application (PbSA) running in VM pair 2. We have activated the same applications in all the ONOS VMs. We have extended the simulations for increasing number of AS domains. The default AS hop length for a vast majority of Internet path is 10 [4]. Figure 7shows the controller to controller response time. The graph shows a comparison between the policy based approach and normal approach of communication between the controllers for varying number of AS domains. There is a minor increase in the overhead with PbSA (10-20% slow response).

We have also calculated the throughput for a single ONOS running our application. First, we have calculated the normal throughput of a ONOS instance by activating drivers, OpenFlow and Forwarding application only. For this we have used Cbench, it is a SDN controller benchmarking tool. After that, we have activated other necessary apps including PbSA and calculated the throughput for them. We represent the throughput comparison of both the experiments in Figure 8. We have varied the switch count and ran the experiments. Due to PbSA, throughput reduced to 2-10% (depends on switch size & flow rule), but we compromise it to security.

To check the CPU load incurred due to our application during active processing time we have used JProfiler. ONOS core and applications running over NBI, displays certain informations at the bash shell of the host OS. These messages are called ONOS tail-log messages. We have marked three zones (1. ONOS normal working zone, 2. Device discovery and driver

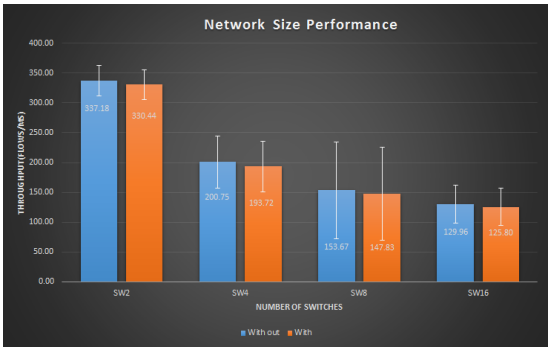

Fig. 8: Throughput Performance

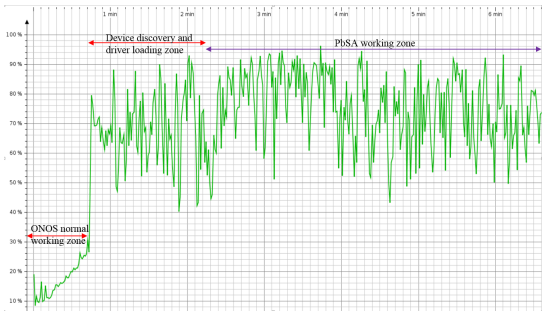

Fig. 9: CPU usages

loading zone, and, 3. PbSA working zone.) in the graph, based on the tail-log messages and their time stamp. In PbSA working zone (figure 9), spikes show the processing time load due to our application.

IV. Related Work

Inter-domain routing is the main backbone in today's network infrastructure. There are approximately 50000 autonomous domains. As SDN is getting popular day by day the need for changing the existing networking world technologies is becoming inevitable. Policy based control in inter-domain SDN is a new area. Here we categorize our discussion into three groups: i) Policy based routing in legacy network, ii) Need and state of inter-domain policy based routing in SDN, and iii) Current state of policy enforcement in SDN intra-domain.

i) Policy based routing in legacy network: Policy based routing for legacy network was first introduced by D. Clark in RFC1102[3] during 1989. This paper discusses how policies can be used to route packets in Autonomous Domains. D. Clark proposed a simple policy syntax which is able to control the Autonomous Domains. However this approach does not consider the security policies between the Autonomous domains and lacks specification of policies at fine granular level.

Tsudik et al. [5] introduced security into Clark policy based routing theme. He introduced $n(n-1)$ symmetric key distribution in the work for encrypting polices. However this techniques distributes all the domains related policies between the AS domains.

Our research work is based on D. Clark[3] RFC, but we focus on the security aspects in SDN controlled Autonomous Domains. Our policies are much more granular and XML based policy storage makes the matching operation much faster. Compared to Tsudik, our work only distributes policies related to specific flows between the AS domain SDN Controllers which incurs less overhead. This aspect of our work is novel from Tsudiks work.

ii) Need and state of inter-domain policy routing in SDN networks: You et al. in [6] has used SDN to conduct multipath routing of flows in inter-domain. This approach does not consider security issues either in SDN nor in inter-domain flow communication. Machado et at. in [7] have mentioned about a Policy Authoring Framework for SDN. This framework deals with only QoS of the network services. In current state of art many SDN controllers do not have support for BGP. Peter et al. in [8] discusses briefly about Broader Gateway Protocol (BGP) based routing weaknesses in legacy network. He also mentions why BGP is not suitable for SDN. Finally, they have proposed an application based routing architecture. Another approach [9] uses Routing Control Platform (RCP) for the SDNs. RCP is a logically centralized BGP route selector and speaker, which is separate from the IP forwarding plane. This work does not focus on the security policy control of the Autonomous Domains.

Recently developed controllers such as OpenDaylight and Open Network Operating System (ONOS) SDN controllers has some limited features to establish communication between Autonomous Domain SDN controllers using BGP. OpenDaylight uses SDNi protocol[10], which only exchanges state information between Autonomous Domain controllers. On the other hand, ONOS uses SDN-IP[11] to communicate with other Autonomous Domain SDN controllers. Though this application mimics BGP protocol, it lags many of the core features of BGP[11].

iii) Current state of intra-domain policy enforcement in SDN controllers: Some important works regarding intra-domain policy control is found in [12], [13], [14], [15], [16], [17]. They have different weakness like granular control, security and complexity in specifying the Policy Expression.

From above discussion it is evident that there is need for work related to secure communication between the Autonomous Domain SDN controllers. Our work proposed a security architecture for end-to-end servers in multiple AS domains.

V. Conclusion

In this paper we presented the design and implementation of security architecture for end-to-end services in SDN. A key component of the security architecture is the specification of security policies that are to be enforced on the SDN communications whether they are intra or inter-domain. We have presented Policy based Security Architecture for securing communications in single AS SDN domain and then extended the architecture for securing communication in multiple AS SDN domains. We have also discussed how our model can handle errors and conflicts that can occur in inter-domain environment.

REFERENCES

[1] O. N. Foundation, "Software-defined networking: The new norm for networks," https://www.opennetworking.org/images/stories/downloads/sdnresources/white-papers/wp-sdn-newnorm.pdf[Accessed12Dec. 2015].

[2] D. Kreutz et al., "Towards secure and dependable software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 55–60.

[3] D. Clark, "Policy routing in internet protocols. request for comment rfc-1102," *Network Information Center*, 1989.

[4] CAIDA, "As path lengths," https://www.caida.org/research/traffic-analysis/fix-west-1998/aspathlengths.

[5] D. Estrin and G. Tsudik, "Security issues in policy routing," in *Security and Privacy, 1989. Proceedings., 1989 IEEE Symposium on*. IEEE, 1989, pp. 183–193.

[6] L. You, L. Wei, L. Junzhou, J. Jian, and X. Nu, "An inter-domain multi-path flow transfer mechanism based on sdn and multi-domain collaboration," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 758–761.

[7] C. C. Machado, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, "Policy authoring for software-defined networking management," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 216–224.

[8] P. Thai et al., "Decoupling policy from routing with software defined interdomain management: interdomain routing for sdn-based networks," in *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on*. IEEE, 2013, pp. 1–6.

[9] C. E. Rothenberg et al., "Revisiting routing control platforms with the eyes and muscles of software-defined networking," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 13–18.

[10] H. Yin et al., "Sdni: A message exchange protocol for software defined networks (sdns) across multiple domains," *IETF draft, work in progress*, 2012.

[11] A. Koshibe, "Appliction for onos," https://wiki.onosproject.org/display/ONOS/SDN-IP+Architecture.

[12] N. Foster et al., "Frenetic: A network programming language," in *ACM SIGPLAN Notices*, vol. 46, no. 9. ACM, 2011, pp. 279–291.

[13] T. L. Hinrichs et al., "Practical declarative network management," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 1–10.

[14] J. Reich et al., "Modular sdn programming with pyretic," *Technical Reprot of USENIX*, 2013.

[15] A. Voellmy and P. Hudak, "Nettle: Taking the sting out of programming network routers," in *Practical Aspects of Declarative Languages*. Springer, 2011, pp. 235–249.

[16] A. Voellmy et al., "Maple: simplifying sdn programming using algorithmic policies," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 87–98.

[17] Y. Ben-Itzhak, K. Barabash, R. Cohen, A. Levin, and E. Raichstein, "Enforsdn: Network policies enforcement with sdn," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 80–88.