1. Introduction

Software Defined Networking (SDN) is an architectural approach that optimizes and simplifies network operations by more closely binding the interaction (i.e., provisioning, messaging, and alarming) among applications and network services and devices, whether they be real or virtualized. It often is achieved by employing a point of logically centralized network control—which is often realized as an SDN controller—which then orchestrates, mediates, and facilitates communication between applications wishing to interact with network elements and network elements wishing to convey information to those applications. The controller then exposes and abstracts network functions and operations via modern, application-friendly and bidirectional programmatic interfaces.

So, as you can see, software-defined, software-driven, and programmable networks come with a rich and complex set of historical lineage, challenges, and a variety of solutions to those problems. It is the success of the technologies that preceded software-defined, software-driven, and programmable networks that makes advancing technology based on those things possible. The fact of the matter is that most of the world's networks—including the Internet—operate on the basis of IP, BGP, MPLS, and Ethernet.

Virtualization technology today is based on the technologies started by VMware years ago and continues to be the basis on which it and other products are based. Network attached storage enjoys a similarly rich history.

I2RS has a similar future ahead of it insofar as solving the problems of network, compute, and storage virtualization as well as those of the programmability, accessibility, location, and relocation of the applications that execute within these hyper virtualized environments.

Although SDN controllers continue to rule the roost when it comes to press, many other advances have taken place just in the time we have been writing this book. One very interesting and bright one is the Open Daylight Project. Open Daylight's mission is to facilitate a community-led, industry-supported open source framework, including code and architecture, to accelerate and advance a common, robust software-defined net-working platform. To this end, Open Daylight is hosted under the Linux Foundation's

umbrella and will facilitate a truly game changing, and potentially field-leveling effort around SDN controllers. This effort will also spur innovation where we think it matters most in this space: applications. While we have seen many advances in controllers over the past few years, controllers really represent the foundational infrastructure for SDN-enabled applications. In that vein, the industry has struggled to design and develop controllers over the past few years while mostly ignoring applications. We think that SDN is really about operational optimization and efficiency at the end of the day, and the best way to achieve this is through quickly checking off that infrastructure and allowing the industry to focus on innovating in the application and device layers of the SDN architecture

2. SDN Overview

2.1 Architecture

Software Defined Networking is a new architecture that has been designed to enable more agile and cost-effective networks. The Open Networking Foundation (ONF) [6] is taking the lead in SDN standardization, and has defined an SDN architecture model as depicted in Figure 2.1

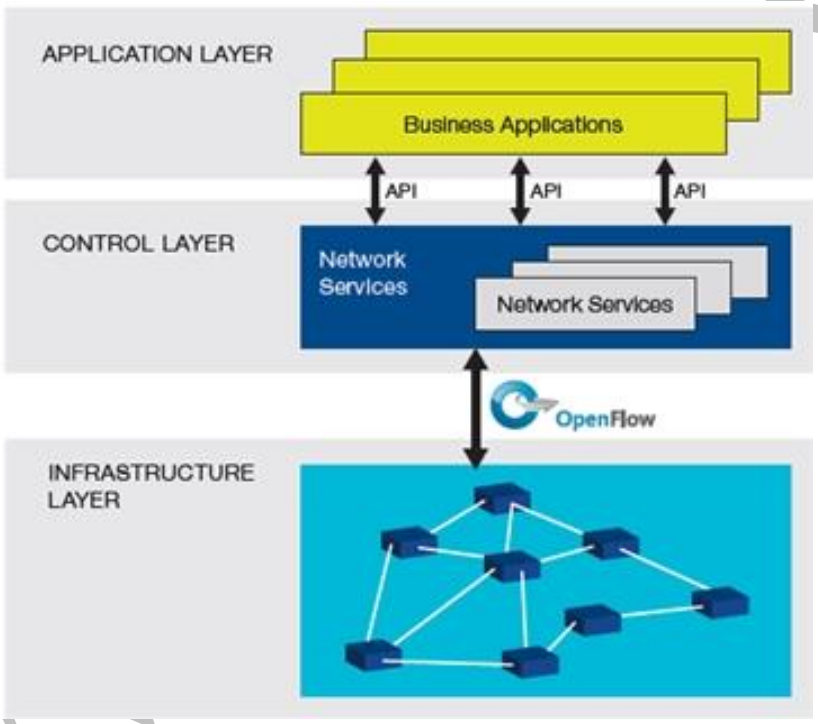


Fig 2.1 SDN Architecture

The ONF/SDN architecture consists of three distinct layers that are accessible through open APIs:

- **The Application Layer** consists of the end-user business applications that consume the SDN communications services. The boundary between the Application Layer and the Control Layer is traversed by the northbound API.
- **The Control Layer** provides the consolidated control functionality that supervises

the network forwarding behavior through an open interface.

- **The Infrastructure Layer** consists of the network elements (NE) and devices that provide packet switching and forwarding.

According to this model, an SDN architecture is characterized by three key attributes:

- **Logically centralized intelligence.** In an SDN architecture, network control is distributed from forwarding using a standardized southbound interface: OpenFlow. By centralizing network intelligence, decision-making is facilitated based on a global (or domain) view of the network, as opposed to today's networks, which are built on an autonomous system view where nodes are unaware of the overall state of the network.
- **Programmability.** SDN networks are inherently controlled by software functionality, which may be provided by vendors or the network operators themselves. Such programmability enables the management paradigm to be replaced by automation, influenced by rapid adoption of the cloud. By providing open APIs for applications to interact with the network, SDN networks can achieve unprecedented innovation and differentiation.
- **Abstraction.** In an SDN network, the business applications that consume SDN services are abstracted from the underlying network technologies. Network devices are also abstracted from the SDN Control Layer to ensure portability and future-proofing of investments in network services, the network software resident in the Control Layer

2.2 SDN Models

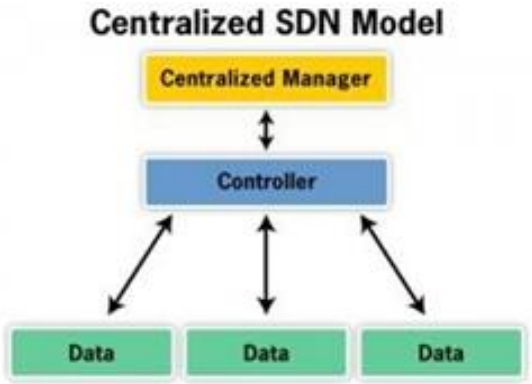
2.2.1 Centralized SDN Model

In the centralized model [4], a centralized manager with a single controller talks to distributed data planes. The route and data flow definitions are all done within that centralized manager. The centralized approach simplifies managing complex flows that are related to specific applications. It also makes setting priority queues around a specific application, for example voice data over Web browsing traffic, because along the path, regardless of the nodes in the network, you can easily say, "voice has priority." But there are issues with scaling the centralized approach, particularly in web-based or

cloud environments. As networks scale and become more distributed, centralized control requires bigger, more costly systems with more powerful CPUs, larger storage

and a more resilient infrastructure with less failure tolerance. Bottom line: the centralized approach can be unwieldy and expensive.

The Centralized SDN architecture model as depicted in Figure 2.2.1



2.2.1 Centralized - a manager with a single controller talking to distributed data planes

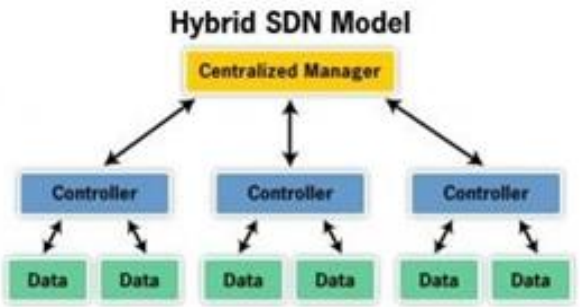
2.2.2 Distributed SDN Model

In the distributed model [4], a centralized manager talks combined distributed controller and data planes. The centralized purist could easily argue this approach and ask, "Why have a distributed network, when just one network with central control would work?" The distributed model with a centralized management interface makes it look as if you are controlling the system as a whole, even though you are really individually managing each node in the distributed network.

From the implementation view, there is a lot more complexity in deploying a distributed network. It may not be possible to get to all of the nodes to create the same path with the same prioritization - so there are timing and synchronization issues. The good news about distributed SDN networks is that they are evolutionary and easy to scale, as networks expand rather than the traditional centralized model of rip and replace. It is also easier to apply policies to individual departments for specific applications with this approach. The Distributed SDN architecture model as depicted in Figure 2.2.2

2.2.3 Hybrid SDN Model

There is value in both centralized and distributed approaches, which is why we advocate a hybrid SDN model. In the hybrid model, you still have a centralized manager, but are distributing separate controller and data planes. The hybrid SDN model [4] leverages the benefits of the simple control of managing specific data flows as in the centralized model with the scalability and resiliency of the distributed model. The Hybrid SDN architecture model as depicted in Figure 2.2.3



2.2.3 Hybrid - a manager talking to distributed controllers that are talking to distributed data planes

Defining policies is key to any network, and meeting an application's specific policy requirements across a large network is very complex. In a hybrid SDN, the ideal approach is to have tightly bound policies in respect to certain applications for more control, but loosely coupled networks for greater agility and flexibility in the management of those networks. The hybrid approach allows for more manageable policy definition in the sense that you can start small, or more locally, and then as you expand in size, apply those policies among more networking components. For example, voice traffic may have a more general global policy, however, if you want another application's traffic to take priority (such as video traffic during a webinar broadcast) then you can define the policy to one or more sets of network paths without addressing it across the entire network. Security prioritizations have the same needs as general policies. You may want global security requirements to be managed in a distributed fashion, but specific local security policies to be managed more centrally. It is much more likely that network automation (which translates into lower costs) will happen at the controller level which is more tightly bound to geography, while simpler global

policies and security configuration will occur at the centralized management level.

3. OpenFlow Protocol

Over time, many software defined networking (SDN) protocols will likely emerge, but for now, the OpenFlow is the mostly commonly used SDN language. In an SDN with a centralized, the OpenFlow [5] protocol carries the message between SDN controllers and the underlying network infrastructure, bringing network applications to life. So far, vendors and enterprises have made swift advancements in OpenFlow product development and network design strategies. In this tutorial, learn about the basics of the OpenFlow protocol, as well as OpenFlow SDN controllers and applications already in testing and production.

3.1 OpenFlow protocol basics: A look under the hood

OpenFlow is a software-defined networking protocol that can be used to centrally control switches and traffic flows in a network.

In a conventional network, switches handle both high-level routing (the control path) and packet forwarding (the data path) [5]. In some SDNs, the control plane is decoupled from the physical network and placed into a centralized controller. These controllers use OpenFlow to communicate with all components on the network.

Using this combination of technologies, engineers can manage the network as a whole rather than as a number of individual devices. This model also allows for more effective use of network resources. The controller and switches communicate via the OpenFlow protocol.

3.2 Using the OpenFlow protocol for network programmability

Why go with OpenFlow SDN? Many believe that OpenFlow SDN will enable the kind of network programmability necessary to better manage server virtualization environments.

Server virtualization has made the IT infrastructure much more dynamic. When a server administrator moves a virtual machine from one server to another, the

network must be able to automatically adjust VLANs, Quality of Service policy and access control lists [5]. But traditional networks are static and don't support this fluidity. So when a virtual machine is ready to be moved on the network, changes must be made manually.

OpenFlow and software-defined networks change that. Together they can make the network more responsive and adaptable to the rest of the IT infrastructure. Most importantly, using a centralized controller and OpenFlow, engineers can coordinate the forwarding of data across all network devices, enabling automation and granularly managed dynamic provisioning for virtualized environments and cloud networks.

3.3 OpenFlow in action: Many vendors, many strategies

OpenFlow started as a Stanford University research project back in 2008, but vendors and large enterprises started productizing the technology and implementing SDN in 2011. Data center mega-user Google built its own SDN switches and was the first to build a global software-driven network. Meanwhile vendors like Cisco and Brocade have released OpenFlow-friendly products, in addition to technology that will depend on alternate SDN approaches.

3.4 OpenFlow SDN controllers: Choices emerge

Both vendors and the open source community are developing OpenFlow controllers that will be used to centrally manage routers and switches in an SDN. FlowVisor[5] was created as a tool for researchers to quickly and flexibly experiment with new SDN ideas and tools in a large production environment. It slices up physical networks through an abstraction layer. The controller acts as a transparent proxy between a network of OpenFlow switches and other standard OpenFlow controllers, and manages bandwidth, CPU utilization and flow tables. While FlowVisor has been deployed in production environments around the country, it is not necessarily enterprise-ready. For example, it lacks a prompt command-line interface or Web-based administration, so users must manage configuration files to push changes out.

Floodlight also had its start in a research environment. The OpenFlow controller was built on work that began at Stanford University and UC Berkeley and now continues among a community of open source developers, along with engineers at Big Switch Networks Inc. Floodlight has been tested with both physical and virtual OpenFlow-compatible switches. It also supports networks where groups of OpenFlow-compatible switches are connected through conventional, non-OpenFlow switches.

3.5 OpenFlow SDN applications go well beyond the data center

Controllers are only one part of OpenFlow architecture [5]. In fact, the SDN market will adopt a three-tiered architecture, according to Kyle Forster, co-founder of Big Switch Networks. The first tier will consist of physical network equipment that are built to be OpenFlow-friendly, such as Ethernet switches and routers. The middle tier will consist of the controllers. The top tier will involve northbound applications that direct security, management and other specific functions through the controllers. Some vendors will play in a single tier while others will participate in multiple tiers.

3.6 OpenFlow applications will take on network monitoring and management

OpenFlow applications will play a key role in network monitoring and management, going well beyond current tools. With OpenFlow controllers, engineers will gain a centralized view of the entire network configuration along with control of every component, even in a dynamic virtual environment. Unlike traditional network monitoring and management tools, OpenFlow provides a powerful tool set for configuring the network in a positively controlled system with multiple feedback loops for accuracy and confirmation.

3.7 Benefits of OpenFlow-Based Software-Defined Networks

For enterprises and carriers alike, SDN makes it possible for the network to be a competitive differentiator, not just an unavoidable cost center. OpenFlow-based SDN

technologies [3] enable IT to address the high-bandwidth, dynamic nature of today's applications, adapt the network to ever-changing business needs, and significantly reduce operations and management complexity.

The benefits that enterprises and carriers can achieve through an OpenFlow-based SDN architecture include:

- **Centralized control of multi-vendor environments:** SDN control software can control any OpenFlow-enabled network device from any vendor, including switches, routers, and virtual switches. Rather than having to manage groups of devices from individual vendors, IT can use SDN-based orchestration and management tools to quickly deploy, configure, and update devices across the entire network.
- **Reduced complexity through automation:** OpenFlow-based SDN offers a flexible network automation and management framework, which makes it possible to develop tools that automate many management tasks that are done manually today. These automation tools will reduce operational overhead, decrease network instability introduced by operator error, and support emerging IT-as-a-Service and self-service provisioning models. In addition, with SDN, cloud-based applications can be managed through intelligent orchestration and provisioning systems, further reducing operational overhead while increasing business agility.
- **Higher rate of innovation:** SDN adoption accelerates business innovation by allowing IT network operators to literally program—and reprogram—the network in real time to meet specific business needs and user requirements as they arise. By virtualizing the network infrastructure and abstracting it from individual network services, for example, SDN and OpenFlow give IT—and potentially even users— the ability to tailor the behavior of the network and introduce new services and network capabilities in a matter of hours.
- **Increased network reliability and security:** SDN makes it possible for IT to define high-level configuration and policy statements, which are then translated down to the infrastructure via OpenFlow. An OpenFlow-based SDN architecture eliminates the need to individually configure network devices each time an end

point, service, or application is added or moved, or a policy changes, which reduces the likelihood of network failures due to configuration or policy inconsistencies. Because SDN controllers provide complete visibility and control over the network, they can ensure that access control, traffic engineering, quality of service, security, and other policies are enforced consistently across the wired and wireless network infrastructures, including branch offices, campuses, and data centers. Enterprises and carriers benefit from reduced operational expenses, more dynamic configuration capabilities, fewer errors, and consistent configuration and policy enforcement.

- **More granular network control:** OpenFlow's flow-based control model allows IT to apply policies at a very granular level, including the session, user, device, and application levels, in a highly abstracted, automated fashion. This control enables cloud operators to support multi-tenancy while maintaining traffic isolation, security, and elastic resource management when customers share the same infrastructure.
- **Better user experience:** By centralizing network control and making state information available to higher-level applications, an SDN infrastructure can better adapt to dynamic user needs. For instance, a carrier could introduce a video service that offers premium subscribers the highest possible resolution in an automated and transparent manner. Today, users must explicitly select a resolution setting, which the network may or may not be able to support, resulting in delays and interruptions that degrade the user experience. With OpenFlow-based SDN, the video application would be able to detect the bandwidth available in the network in real time and automatically adjust the video resolution accordingly.

4. Distributed Control Behavior of SDN Reviewed

According to the design of current SDN's leading technology, flows can be programmed (installed) by the controller. This means that the controller is the only entity that is responsible for installing and maintaining flows on the network equipment. For simplification let's call this type of flow installation the "controller to equipment flow installation". The controller to equipment flow installation has many advantages like having tight control over all of the equipment by the controller.

However, the advantages of the controller to equipment flow installation come with some cost. First is the probability that the controller would be a source of bottle neck in the whole system. This can be confirmed by, Michael Jarschel et al. who concluded [7] that "*When using OpenFlow in high speed networks with 10 Gbps links, today's controller implementations are not able to handle the huge number of new flows.*". Second, by limiting the flow manipulation to "*the controller to equipment*" installation method OpenFlow can miss some opportunities that the "*network equipment to network equipment*" can provide.

For the previously mentioned reasons, a new method for installing flows, that is, the "network equipment to equipment flow installation" (Ne-NeFI) method. Through using this method, the controller does not have to program (install) flows to each one of network equipment one by one; instead it can ask the equipment to spread this flow to other equipment on behalf of the controller, this can be useful in cases where the controller needs to program non critical-start up time flows. And thus relieving some load off the controller. Also, the network equipment to equipment flow installation method can be used to make the OpenFlow network more self-aware by having the network equipment cooperate and carry loads for each other upon the need and traffic situation by having the overloaded equipment delegating some of its flows to another network equipment.

4.1 Protocol of the Distributed Control of SDNs

In order to enable distributed control of SDNs, represented by the network

equipment to equipment flow installation to be adopted to the OpenFlow Protocol [1], three new packets have to be introduced (see Fig. 4.1). First one is, equipment to equipment (e-e) flow installation request, abbreviated as "e-e request". While the second is; the e-e flow installation reply, abbreviated as "e-e reply". The third is the e-e flow installation acknowledgement or negative acknowledgement, abbreviated as "e-e ACK/NACK".

The first packet is the e-e flow installation request. This packet holds an OpenFlow header, list of flows to be programmed, address of the equipment that sent the request and an identification value, address and identification of the originator of the request (who requested for the flows to be programmed in the first place, i.e. controller or an equipment), Level of Flow Installation, the Time To Live (TTL) of the IP protocol, and a temporary identifier for this request. Where, the Level of Flow Installation (LFI) is somewhat similar to the TTL field in the IP protocol. Where, the TTL in IP protocol indicates how many hops a packet can travel, and it will be decremented when passing a network equipment and the packet will be discarded when the value reaches 0. While, the LFI, it is decremented each time an equipment relays the e-e request (sends further the broadcast). And thus it is very similar to the TTL value except that the LFI is part of the proposed protocol of the distributed control of the SDNs, while TTL is part of the underlying IP protocol. LFI and TTL are used together to control the propagation of the e-e flow installation request.

The second packet is the e-e flow installation reply. It contains an OpenFlow protocol header, the identification of the e-e flow installation request, the reply to the request which can be either an acceptance or a rejection, address and self-identification of the equipment that sent the reply.

And the third packet is the e-e Acknowledgement or the e-e Negative Acknowledgement (e-e ACK/NACK). Its purpose is clear; to confirm to the equipment that sent the reply that its reply has been received, and accepted or rejected.

5. Security Issues in SDN

5.1 Network Security Challenges

IT infrastructure is rapidly moving to the cloud, creating a dramatic technology shift in the data center. This shift has significantly influenced user behavior: end users now expect anytime, anywhere access to all their data. Additionally, network operations are being transformed from operator-intensive management towards greater automation.

The data center of the future is emerging as a highly virtualized environment that must address a diverse set of user needs, including anytime, anywhere access to their data, the consumerization of IT (BYOD) and increased reliance on cloud services.

Security concerns are consistently identified as a major barrier to this data center transformation. While protecting user data is of paramount importance, mobility and virtualization pose new threats that must be understood and secured. And the human factor continues to lead to unnecessary downtime, expense, and unauthorized intrusion.

5.2 Threat model

The proposed distributed control security methods seeks to provide robust protection against both insider threats (authenticated network equipment), and outsider threats (end hosts or an unauthenticated network equipment). We also assume that an attacker might be able to use different points within the same network to charge his attack.

5.3 Security Requirements/Goals

The main goals and requirements of the proposed security methods are as follows:

- Allowing the transfer of flow table entries form one network equipment to another, in a way that prevents any malicious user form obtaining any information related to that flow entry or disclosing its contents. And thus preventing any malicious user from obtaining any knowledge about the network or its operation or control.

- Enabling a smooth operation of the distributed control of SDN. This requires, the security methods to be able to protect the distributed control's protocol, so that no attack could be charged to jeopardize the operation of the SDN's distributed behavior.

- Protecting the whole SDN network from any attack that might use the distributed control to affect the normal operation of the SDN. The importance of this requirement is obvious, since the original design of the centralized (central controller to any equipment) the OpenFlow [8] is secured by using Transport Layer Security (TLS) [9]. And thus, any propose to extend the centralized control model must be able to maintain the security of the whole network.

6. Design of the Security Method

In order to designing a successful security method for the distributed control of the SDN, special care must be taken enable the designed algorithm to achieve the security requirements (described in Subsection 5.3) while being able to operate under the threat model described in Subsection 5.2.

To shed more light on the design of our proposed security method for the distributed control of SDNs; Subsection 6.1 shows the main components of the security method. While Subsection 6.3 describes in details the algorithms used to compose the desired security scheme. Finally, Subsection 6.4 shows the how the proposed security method can be applied, using a scenario based example.

6.1 Building Blocks

The main building blocks that the security method of the distributed behavior of SDN, are explained as follows:

1. Trust Manager

It is the entity; that is responsible for assuring a secure binding of the unique ID of each network equipment with the public key of that network equipment. Along with the binding, the second responsibility of the trust manager is to periodically distribute a certificate list to all of the network equipment in its domain. Where this certificate list; contains the digital certificates of all of the equipment in the domain of the trust manager. The third responsibility of the trust manager is to manage the list of trusted network equipment within its domain by listening to threat warning reports sent by the network equipment within its domain in case of a suspected attack or a confirmed one (as will be explained in the next Subsection). And then, responding to the threat warnings by suspending the certificate of the attacker network equipment, and sending an updated certificate list - that does not contain the certificate of that attacker - to the network equipment within its domain.

Through our design of the proposed security method, we assume that the public key signature algorithms used in the trust manager and the network equipment are secure and

no malicious user is able to fraud a valid signature nor he is able to recover the secret key of any component of the scheme. We also assume that the communication between the network equipment and the trust manager is also secure.

2. Network Equipment

Network equipment are same as the regular SDN equipment that are either routers or switches that supports OpenFlow or any other SDN technology. And in addition to their regular tasks those network equipment have to perform additional operations in order to use a secure distributed control of SDNs. Those additional operations are; digitally signing e-e requests, verifying digital signature of the distributed control, receive and store the certificate list distributed by the trust manager, and reporting any threats (activity or message exchange the is expected to be of a malicious attacker) to the trust manager.

6.2 Proposed Security Mechanism

This Subsection shows the conditions forming the main methods of security. And thus, they are the main essence of how attacks are prevented.

1. Using Trust Manager: it will send and update a list of trusted devices. E.g. list of (device ID, public key).
2. Using start time and end time for each e-e request.
3. Having every device that relays or receives the e-e request broadcast to check the signature of device that originated the e-e request, and thus make sure that the e-e request received is exactly same as it left the device that originated it.
4. Having every device that relays the e-e request broadcast, to check the signature of the previous device that relayed the e-e request, knowing that it must be a direct neighbor.
5. Having every device that relays the broadcast to sign the e-e request.
6. The flow table entries will be encrypted when they are transferred form one network equipment to the other, so that no eavesdropper can expose their contents.
7. Each network equipment will hold counters, one for each network equipment in the certificate list. Where this counter counts the number of e-e requests that does not

pass the conditions of the main methods of security shown in this chapter.

6.3 Algorithms

Section 6.1 explained about the main components of the security method of the distributed control of SDNs. While, this Section; explains the details of the algorithms of the security method, and discusses their steps in details.

We first start by showing the list of variable and primitive functions used in the algorithms of the security method in Table 6.3. Then, we start by explaining the details of the algorithm used by the originator of the e-e request to create and send the request, as shown in SEND_REQUEST algorithm. After that, we will explain the details of the algorithms used for by the network equipment that receives the e-e request; that are: RECEIVE_BROADCAST algorithm, which calls the CHECK_INCOMING_REQUEST algorithm and the RELAY_BROADCAST algorithm. After that, the later algorism will be explained respectively. Finally, we will explain the RECEIVE_REPLY algorithm, which shows the steps taken by the originator of the e-e request upon receiving a reply to his request.

```
1:  Function SEND_REQUEST ()

2:      Req = create_req ()
3:      self_sign = sign (private_self , Req)
4:      sigList = create_signature_list ()
5:      Append ( sigList , (D_self , self_sign) )
6:      Broadcast (Req, sigList)
```

Fig 6.3.1 SEND_REQUEST Algorithm

The SEND_REQUEST algorithm (shown in Fig.6.3.1), is a simple algorithm, which is followed by the originator of the e-e request, to create the e-e request, as shown in line 2. After that, the originator will digitally sign the e-e request and add that as the first signature to the sign list of the e-e request, shown in lines 3 through 5. The Final step in this algorithm is for the originator to broadcast the e-e request, as shown in line 6.

```
1:  Function RECEIVE_BROADCAST (req, sigList)
    // req = the e-e request
    // sigList = {(D1 ,Sig1), ...., (Dn ,Sign)}

2:      CHECK_INCOMING_REQUEST (req, sigList)
3:      if (can_Accept_Request (req))
4:          rep = create_Reply (req)
            // rep = e-e reply
5:          repSig = sign(private_self , rep)
6:          reqSig = sign (private_self , req)
7:          rep = Append (rep, repSig, reqSig)
8:          send (get_Src (req) , rep)
9:      end if

10:     decrement (LFI)
11:     if (LFI > 0)
12:         RELAY_BROADCAST (req, sigList)
13:     end if
```

Fig 6.3.2 RECEIVE_BROADCAST Algorithm.

The main algorithm for handling the security method within the network

equipment receiving e-e request is the RECEIVE_BROADCAST algorithm (shown in Fig. 6.3.2). This algorithm is called after receiving an e-e request, and it starts by checking if the incoming e-e request matches point 2) through 5) of the methods show in Subsection 6.2 by calling in line 2 the CHECK_INCOMING_REQUEST (shown in Fig. 6.3.3, and explained next). After that, if the received e-e request is found to be correct then the algorithm will check in lines 3 to 9, if equipment in which it runs is capable of serving this e-e request (that is; the equipment is capable to receive new flow table entries form the sender of the e-e request) . And if it can receive the new flow table entries then the algorithm will generate an e-e reply (line 4) that will be signed (line 5). Also, if the equipment is willing to accept the request it has to generate a signature of the received e-e request (line 6); so that the originator of the e-e request can verify that the sender of the reply did receive the original e-e request that the originator did send. The final step of accepting to serve an e-e request is to send the e-e reply, its signature, and the signature of the e-e request; to the originator of the e-e reply (lines 7, 8). And in case of the network equipment is not capable of serving the e-e reply by itself then it will decrement the lifetime (LFI) of the e-e reply by one, sign it, and broadcast it again as shown through lines 8 to 11.

After explaining about the main algorithm related to receiving the e-e request that is the RECEIVE_BROADCAST. We will continue to explain one of the algorithms that the main algorithm calls; that is the CHECK_INCOMING_REQUEST algorithm as shown in Fig 6.3.3. Where, in this algorithm, it first starts in line 2 to check if it received the e-e request from a direct neighbor, and if it received the e-e request form a network equipment that was included in the certificate list - that was received from the trust manager - and in case that any of those conditions is not satisfied then the packet will be dropped as shown in line 3. After the e-e request passes the test in line 2, the validity will be tested of both the in its signature list (line 8). If both are correct, then the CHECK_INCOMING_REQUEST algorithm finishes. Elsewise, if the signature of the originator of the e-e request is found to be invalid (line 6), then the request will be dropped. Same thing will happen in case that the last signature in the signature list ($Sig7_n$ of sigList) is found to be invalid (line 9).Furthermore, in that case,

the counter of the dropped e-e requests related to the network equipment that

broadcasted the false e-e request will be incremented (line 10) (as explained in point) of Subsection 6.2). And in case that this counter increases over a predefined threshold that is the Tolerate_Limit (line 11), the algorithm will generate a threat warning report and send that to the trust manager (line 12). Where in its turn the trust manager will count for a number of threat warning reports before removing the network equipment form the certificate list and thus preventing the suspected malicious equipment form doing further malicious activities. It should be clarified here that choosing the value of the Tolerate_Limit, or the threshold upon which the trust manager removes an equipment form the certificate list.

The second algorithm called by RECEIVE_BROADCAST is the RELAY_BROADCAST shown in Fig. 6.3.4. The RELAY_BROADCAST algorithm is relatively simple. It starts by signing the e-e request using the current network equipment's public key. After that the signature is appended to the signature list (sigList) in the e-e request, and finally the e-e request is broadcasted again.

Finally, the RECEIVE_REPLY algorithm, shown in Fig. 6.3.5; explains the steps followed by the originator of the e-e request upon receiving an e-e reply to his request. It starts by checking the validity of both the signature of the e-e reply and the signature of its corresponding e-e request, if they were signed by the sender of the e-e reply or not (line 2). If any of those two signatures fails, then the received reply will be dropped (line 3). In case the two signatures are valid, the next step will be to check if the e-e request has been already satisfied or not. If it was satisfied (lines 6 to 11); then a negative acknowledgement (NACK) will be sent, after signing it to the sender of the reply. On the other hand, if the request has not been satisfied (lines 12 to 18); then a positive acknowledgement (ACK) will be sent to the e-e reply sender, after singing it.

```
1:  Function RECEIVE_REPLY (rep, repSig, reqSig)
    // rep = the e-e reply
    // repSig = the signature of the reply, by its sender.
    // reqSig =  the signature of the received request, by the sender
                 of the reply.

2:      if (check_Signature_If_Not_Valid (rep, repSig ) OR
        check_Signature_If_Not_Valid (originalReq, reqSig ) )
3:          drop (rep)
4:      end if
5:      else
        // if both signatures are verified correctly
6:          if ( is_req_satisfied (originalReq) )
7:              NACK = create_neg_ack(rep)
8:              nackSig = sign (private_self , NACK)
9:              NACK = Append (NACK, nackSig)
10:             send (get_Src (rep) , NACK)
11:         end if
12:         else
            // the request has not been accepted before
13:             ACK = create_ack(rep)
14:             ackSig = sign (private_self , ACK)
15:             ACK = Append (ACK, ackSig)
16:             send (get_Src (rep) , ACK)
17:
18:         end else
19:     end else
```

Fig 6.3.5 RECEIVE_REPLY Algorithm

6.4 Usage Scenario

To further explain how the proposed security method works, the following scenario follows the steps of the distributed control along with the proposed security method. This scenario is show as a series of steps shown by figures Fig. 6.4.2 through Fig. 6.4.5. In those figures, each router is identified by its color. The medal shape represents the digital signature, where the color of the medal's ribbon represents the router that signed it by having that router's color. And the red tick symbol represents either a verified digital signature, or verifying that the e-e request was sent by a direct neighboring network equipment. Also, in this scenario we assume that the certificate list has been already distributed to the whole network, where this is represented by showing the symbol of the certificate list over the whole network. While the numbers in circles represents the step number in the explanation of each figure.

The scenario first starts by having network equipment named R1 - shown in red color - initiating a distributed control behavior that is the Ne-NeFI. This initiation shown in step 1 of Fig. 2, where R1 broadcasts an e-e request after signing it.

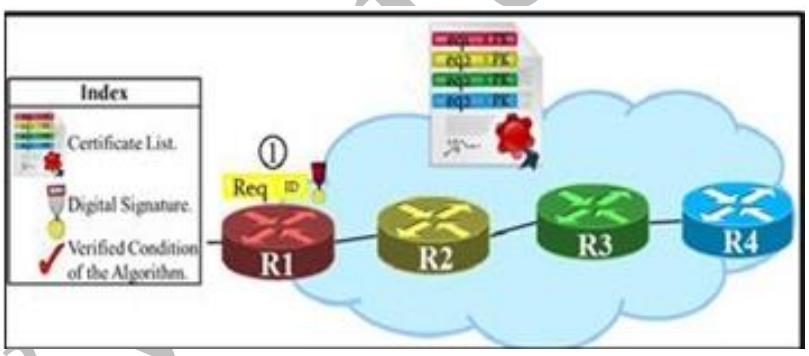


Fig 6.4.2 Request Sending

After that, in Fig. 6.4.3, network equipment R2 receives the e-e request, checks that it was received from its neighbor, and that the signature is an authentic one of R1; as shown in step 1. However, for the sake of illustration, both of R2 and R3 are not capable of serving the e-e request. And Thus R2 will broadcast the e-e request after signing it. Next, in step 2, network equipment R3 will does the same steps followed by R2 in step 1, and will end broadcasting the e-e request.

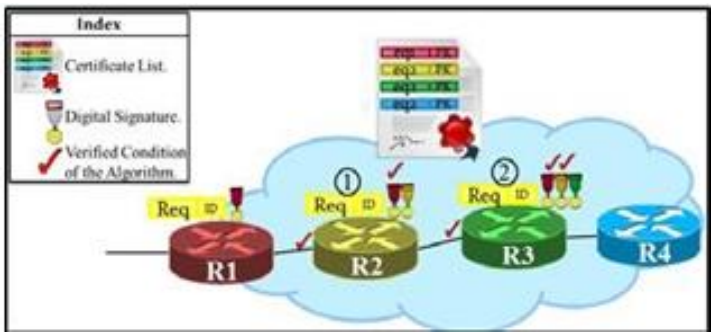


Fig 6.4.3 Relaying Request

Next, in Fig.6.4. 4 step 1, R4 receives the e-e request, and verifies that it was sent by a neighbor equipment, and that it was properly signed. Then, R4 decides to serve this e-e request. And thus, it will send an e-e reply after signing it to the initiator of the Ne-NeFI's e-e request that is R1. As shown in step 2.

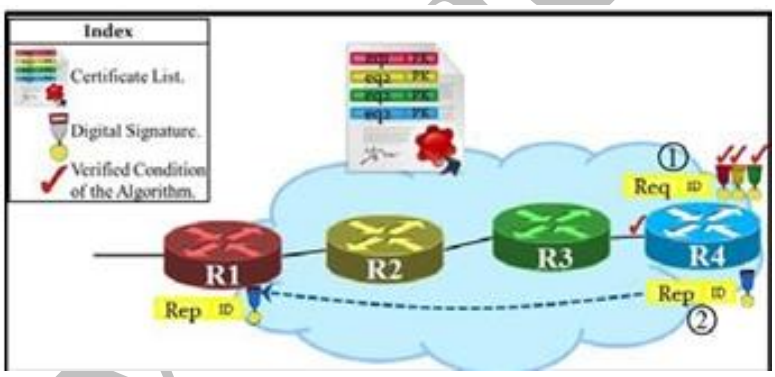


Fig 6.4.4 Replying for Request

Finally, Fig. 6.4.5 shows in step 1, that R1 - the initiator of the Ne-NeFI - receives the e-e reply sent by R4, and checks that it was properly signed by R4. After that, in step 2, R1 will send an e-e ACK to R4 after signing it, so that R4 can verify that the acknowledgement was sent by R1.

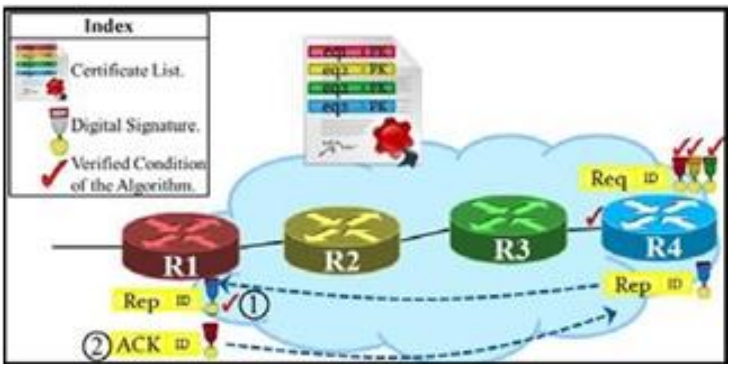


Fig 6.4.5 Accepting Reply

7. Attack Resistance

Based on the threat model presented in Subsection 5.1, and the requirements illustrated in Subsection 5.2; the proposed security methods are designed to provide robust security, resisting many security threats. In the rest of this Section we considered various classes of security threats and showed how the proposed security methods provide protection against them.

- **Man in the Middle Attacks**

This class of attacks can occur in different variations. First, is the case where an eavesdropper whether internal or external, captures legitimate packet from a distributed control exchange. In this case, the effect of such attack is very limited, since the attacker will not be able to alter any contents of the distributed control, since it is digitally signed. And based on the assumption that the public key signature algorithms are secure, then it will be inapplicable for the attacker to resign the distributed control packets since the attacker does not possess the private key of the previous equipment that have relayed the packet, nor that of the initiator.

Furthermore, even if the attacker was an internal one and did legitimately sign the packet - because he was listed in the certificate list - after tampering with its original contents. This can be carried out only in the case of the e-e request being relayed, because in the case of e-e reply and e-e ACK this will not be possible since only the sender will sign; and thus, it will not be possible for the attacker to resign the packet. Similarly, in case of the e-e request, each network equipment that relays, or accepts the e-e request; will check the signature of the originator of the e-e request in addition to the last signature in the signature list (sigList). And so, if any malicious user tampers with the original e-e request, this will cause the e-e request to be dropped. Thus, any tampering with the contents of the distributed control protocol will be detected, and renders any subsequent attacks to be impossible.

- **Resource Exhaustion Attacks**

This attack also can occur in different variations. The first one can be the case that an attacker being either an internal or an external eavesdropper will copy legitimate e-e request and resend them to a distant part of the network for the purpose of consuming the flow table entries of many equipment all over the network. However, according to the proposed security methods, such attacks are rendered impossible since that the algorithm for any relaying network equipment will make sure that the e-e request was received form a direct neighbor, if not then the e-e request will be discarded. And thus this type of resource exhaustion will not be possible.

Another variation of the resource exhaustion attack is the simple case of having an internal equipment sending a large number of e-e requests for the purpose of exhausting the available space of the flow tables of other equipment within the network. In this case, according to the proposed security methods, neighbor network equipment will send threat warnings to the trust manager, who in turn will remove the malicious sender form the certificate list, and thus will stop the attack and prevent any further attacks to be done by that malicious equipment.

While another more sophisticated variation, where an attacker might have control over one or more internal network equipment. In such case, an attacker can copy a legitimate e-e request, encapsulate it, and send it to another network equipment - under control of the attacker - in a more distant part of the network, in order to exhaust the resources of network equipment in that distant part of the network. In such attacks, the attacker can maintain the e-e request in its original form without tampering it or its signature; after that the attacker can legitimately add the signature of the distant attacker equipment to the signature list. Thus, if a network equipment receives the request form the distant attacker equipment; and the receiver, will find an authentic e-e request with its original signature, and will find that it received this e-e request by a direct neighbor. However, such attacks are made unfeasible, since, the e-e request will be more likely to be served/accepted by another network equipment that is nearer to the originator of the e-e request, and in this case the originator will send a negative acknowledgement to cancel the installation to the

distant sender of the e-e reply. And thus such attacks are infeasible since they require big efforts of the attacker to charge such attacks, while their effect will be minor.

8. Conclusion

Providing future Internet with technologies that enable it to play its role is extremely important. Because of that, many researchers are studying technologies to be the future Internet enabling technologies. SDN is one of the candidate future Internet technologies, as it provides compelling functionalities that enable smarter applications to be built. However, there have been many concerns regarding its scalability; as well as of its key enabler OpenFlow, especially, due to its dependence on a central controller. And thus, many efforts were done to overcome this problem. Following the well-established fact; that all sensitive computer networking operations must be secure, in this work we propose methods for securing the distributed control behavior of the SDNs. In order to get a fully secure hybrid control, since the centralized control is already secure by means of TLS.

In order to achieve the desired security for the distributed control, we designed security methods and algorithms. Where the proposed methods require; according to our design, a centralized trust manager to distribute a list of trusted equipment along with their public keys. In addition to the centralized trust manager, the network equipment must be able to perform digital signature, signature verification, and reporting any threat warnings to the trust manager. In more details, the equipment to originate and send the e-e request has to sign it. While each network equipment relying the e-e request must make sure that it received a verified e-e request from direct neighboring equipment, and the e-e request have not been tampered. And thus, we can make user that the genuine e-e request did traverse a trusted path. After that, if the e-e request have reached a network equipment that is willing to accept this request, it will send a signed reply along with a signature the e-e request that it received to the equipment that originated the e-e request. Thanks to which, the originator of the e-e request, upon receiving the e-e reply can make user that it came from an authentic equipment, and that equipment did receive the original genuine e-e request. Finally, the equipment that originated e-e request will reply to the e-e reply, by either an acknowledgement or a negative acknowledgement. And thus by following the previous steps it is possible to secure

the distributed control of the hybrid control model of SDNs, thus enjoying the benefits of the hybrid control without jeopardizing the whole network.

References

[1] O. M. Othman and K. Okamura, "Securing the distributed control of SDN," in the international conference COMPSAC 2013 - International Journal of Computer Science and Network Security, September 2013.

[2] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," Open Networking Foundation, 2012.

[3] SDN-Software-Defined-Networks by Thomas D Nadeau & Ken Gray.

[4] www.opennetworking.org.

[5]http://searchsdn.techtarget.com/guides/OpenFlow-protocol-tutorial-SDN-controllers-and-applications-emerge

[6] Open Networking Foundation-ONF Solution Brief October 8, 2013.

[7] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll and P. Tran-Gia, "Modeling and performance evaluation of an OpenFlow architecture," in Proceedings .

[8] T. Dierks and E. Rescorla, "rfc5246: The Transport Layer Security (TLS) Protocol Version 1.2," The Internet Engineering Task Force, 2008.

[9] "OpenFlow Switch Specification, Version 1.1.0," 2011.