CAPSTONE PROJECT

I2RS ARCHITECTURE IN SDN AND ITS USE CASES

Project Supervisor : Juned Noonari

Name : Anshul Arora

Faculty : Master of Science in Internetworking

ABSTRACT

With the introduction of virtualization and software defined networking, the way network functions has been completely revolutionized. These days, one controller is required to control and monitor hundreds of devices, a job which usually required a human administrator. This process is fast and simple, but still the way the controller interacts with the router or any networking device is still not fully optimized in terms of the ease, the speed at which new policies are implemented over the networking device. It can further be evolved with the help of I2RS protocol.

I2RS protocol can utilize existing protocols to inject routes directly into the local routing information base of a networking device. With the help of topology manager, it can provide full-fledged view of the network topology. This can be helpful in taking decisions regarding the network. With the access to local RIB the feedback loop between sensing the network and implementing new policies can be significantly tight. This can make the network completely real time.

The I2RS protocol provides several use cases like distributed reaction to the attacks, within data center routing, route control via indirection, policy based filtering of the unknown traffic and so on. All these use cases are based upon the present need of the network. They have been formulated after analyzing the inabilities of the existing protocols like CLI, NETCONF, etc. However, these use cases require slight modifications in the networking technologies, listed in the requirements for the use cases.

In this report, a full study on the architecture of the I2RS, its compatibility with existing SDN, its business drivers, Use cases and applications has been done. Further, a demo of an SDN topology along with how it can be improved with the inclusion of I2RS has also be included.

Table of Content

Section 1- Introduction

[1]A network device consists of a data plane and a control plane, where control plane is a switch fabric connecting various ports of the device and control plane is the brain or CPU of the device. Therefore, a simple topology of 4 switches connected with each other would look like figure 1this in a conventional scenario.

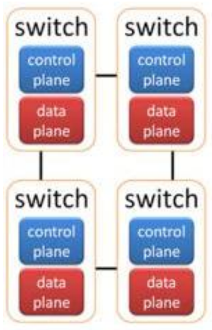


Figure 1

Control Plane - [2]The control plane in the process of learning the network through routing protocols like RIP, OSPF, etc. Control plane packets are destined to or locally originated by the router itself. This is the major difference between the concept of the control and data plane.

Routing update, like an OSPF update, going to the router is process switched, which means that the general purpose CPU has to handle it. In other words, every packet received by the router is processed independent of other packets which consumes CPU cycles. Management protocols, like Telnet, SSH, SNMP, etc. could be considered part of the control plane, but are more properly considered part of the Management Plane, which is a specific subset of the control plane.[2]

Data Plane - [3]The data plane (sometimes known as the user plane, forwarding plane, carrier plane or bearer plane) is the part of a network that carries traffic. The data plane does not perform any computing of its own , but just sends the packets over the network based upon the forwarding table which has already been built with the help of the results of the computations and processing of update packets done by the control plane.[3]

Management Plane - [4]Management plane is used by the user to interact with the networking device. Interfaces, IP addresses and routing protocols (RIP, OSPF, etc) are configured through management plane protocols like CLI, NETCONF or certain northbound APIs.
Management plane protocols like SNMP can be used to monitor the operation, performance, interface counters, etc for the networking device.[4]

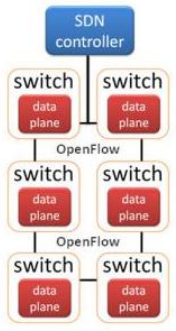


Figure 2

[5]Here each switch has its own control plane (CPU) and data plane (forwarding plane). This kind of topology is what we have all been used to. However, when the number of devices grow , to a number such as 1000 or 50,000 then configuring each switch separately can be a slow and tiring job. With the help of Software defined networking, the deployment and monitoring of these devices can be simplifies, centralized and automated.[5]

This is where SDN comes into scene. What SDN does is that it separates the control plane from the switches, leaving behind just the data plane.[6]Software-defined networking (SDN) separates the data and control planes, removes the control plane from networking device and runs it in software instead, which allows for programmatic access and, as a result, makes network administration much more easy [6].

Figure 2 shows a standard SDN scenario where there is one centralized control plane to push commands to four switches (4 data planes).The control plane and the data plane are connected though some southbound interface like openflow as specified in the figure. Since now there is just one controller taking care of 6 switches, it leads to a faster service provisioning, less costly network operation and management, and flexibility.

I2RS(Interface to Routing System)

Traditionally, the routing protocols running on the router will perform route calculations for a particular prefix/length based upon. All these calculations are performed by the control plane and the routes that are generated after the routing protocol completes its calculation are based upon certain algorithm. Each routing protocol has a predefined algorithm which it utilizes to calculate the routes over a network.

[7]However, there can be certain forwarding requirements in industries which are beyond the scope of traditional routing protocols and their predefined algorithms. This calls for the need of a method which can allow a network administrator to modify the routing table either manually or with the help of some application based upon his special needs. This is where I2RS can be useful.[7]

[7]I2RS stands for “Interface to the Routing System”. The general idea is to provide an interface that fits in between routing protocols such RIP, IS-IS and the RIB (routing

information base) on a networking device, modifying the routing table. In other words, I2RS provides a programmable interface with can inject routes using an application directly into the RIB, bypassing the conventional routing protocols. [7]

Section2: SDN and I2RS architecture

2.1 Major components of SDN

[8]Figure 3 shows SDN architecture as seen by ONF (Open Networking Foundation). The main goal of SDN is to provide open interfaces which can help in developing of software applications which can control the network connectivity between network devices and perform flow modification and also provide statistics.[8]

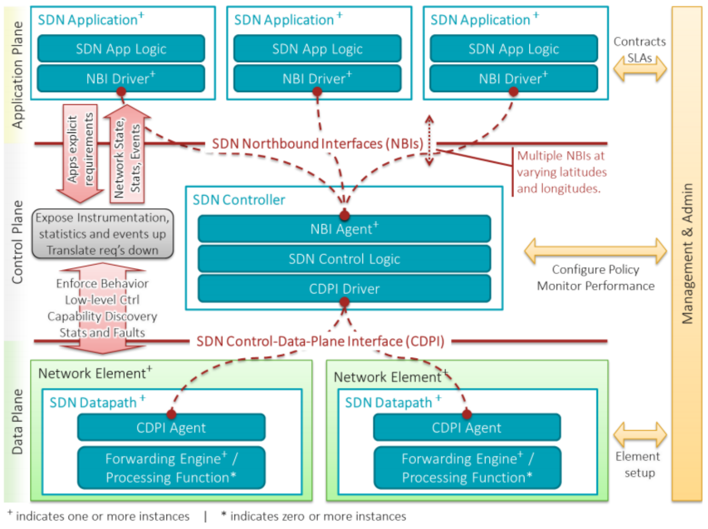


Figure 3[2]

SDN consists of the following major components: -

- **SDN Application -**
  [9]An SDN application is a toolcreated to perform a task in a software-defined networking (SDN) environment. SDN applications can replace and extend the functions that are already performed through firmware in the hardware devices of a traditional network.[9]

  [8]SDN applications are the programs which specify their networking needs to the SDN controller through a north bound interface. An SDN application consists of SDN app logic and an NBI driver.

  An SDN Application comprise of one SDN Application Logic and one or more NBI Drivers. SDN Applications may themselves expose another layer of abstracted network control, thus offering one or more higher-level NBI(s) through respective NBI agent(s). [8]

- **SDN Controller**–[8] SDN controller performs the task of processing the requests ofSDN application and producing the desired result in network devices. Also, it provides an abstract level of network scenario back to the SDN application. It consists of an SDN control logic, an NBI agent and a CDPI driver.[8]

  [10]An SDN Controller is the brain of the network in an SDN environment. It is the central control point in the SDN network, transmitting information to the networking devices through southbound APIs and the SDN applications through northbound APIs.[10]

  An SDN Controller usuallyhasseveral pluggable modules that can perform a variety networkingfunctions. Some of the tasks including listing the devicesthat are within the network along with their capabilities, collecting network statistics, etc. Extensions can be addedtoadd more the functionality and run more advanced features, such as running algorithms to analyze and configuring new policieson the network.

  [10]Some of the most common protocols used by SDN Controllers to interact with the devicesareOpenflow and OVSDB. Other less common protocols used by controllers are YANG or NetConf.

  Many new protocols are being developed to be used along with the SDN controllers. For example, the Internet Engineering Task Force (IETF) working group – theInterface to the Routing System (i2rs**)** – is developing an SDN standard that enables an SDN Controller to leverage proven, traditional protocols, such as OSPF, MPLS, BGP, and IS-IS.[10]

- **SDN Datapath** - [8]SDNdatapath consists of a CDPI agent and some data forwarding engines and functions. The main role of these functions or engines is to forward data between external datapath interfaces or inside the network element between the internal processing functions.[8]

  [11]The datapath of an OpenFlow Switch consists of a Flow Table, and an action associated with each flow entry. In simpler words, SDN datapath can be seen as data plane of conventional networking environment. It simply performs the task of forwarding data packets based upon certain routes calculated by the SDN controller (which acts as a control plane for SDN environment.[11]

- **SDN Control to data plane interface (CDPI)** - As the name suggests, CDPI is an interface between SDN controller and data plane. It provides the following functionalities : -

  (i) Programmatic control of all forwarding operations- It allows the network administrator to control the forwarding decisions with the help of software applications. The controller simply processes the scripts running in it to make forwarding decisions. These forwarding decisions are then sent to the forwarding plane with the help of controller to data plane interface. After this, the forwarding table is populated with the routes from the controller and then used for forwarding the data.

(ii) Capabilities advertisement - In a network, to run a feature or a service , it is important to first find out that if the service is supported by all the routers of the network. These services are called the capabilities of the router and each router must be able to know the capabilities of its neighboring router.

For example,[12]BGP uses additional field in its open message while peering to know about the capabilities of the neighboring router[12]. Similarly, OSPF uses an opaque type LSA for it advertising its capabilities to the neighbors.

In SDN environment, this task is performed with the help of controller to data plane interface. All the capabilities related to each routing protocol are processed by the control plane and then later on sent to the data plane with the help of CDPI interface. The networking device, then simply forwards its capabilities to its neighbors as directed by the controller.

(iii) Statistics reporting -The SDN controller receives the all the analytical data through its data plane from its neighboring router through the CDPI from the network hardware. This data may be used to generate statistics for the network administrator and includes details like packet loss, network congestion, neighbor relationships, etc.

This is very crucial for orchestrating networking policies over the network. It acts as a feedback between the existing network policies and the future policies. For example, if the network administrator sees that there is too much packet loss over a link, then he can failover to the redundant link which has less packet drop.

(iv) Event notification - [13]Event notification is a capability that enables you to define certainsituations that cause the networking device to send notifications to the network administrator.[13]

Once events are sets through the controller, the interface between the controller and the hardware device (CDPI) monitors the working of the device. In case the event already specified by the administrator comes up, a notification is sent over the interface to the controller which notifies the administrator about it.

- **SDN Northbound Interface** - SDN northbound interface is the interface between the SDN controller and an SDN client , which is accessible to a user. It provides a topological view of the network to an administrator and also directly controls the behavior of the network.

  Software programs called northbound applications are implemented above SDN controller. The northbound API provides a network abstraction interface to the applications which lie above the controller in SDN stack. Term northbound is used for the interface that lies above the controller and the term southbound is used for the interface that lies below the SDN controller.

  Most of the Northbound interfaces implemented today are open and vendor-neutral. An example of one such interface is Openstack.

- **Interface driver and agents -** [8]Each interface consists of a driver agent pair . A driver is present in the northbound interface and an agent is present in the south bound interface. Since, the northbound interface is associated with SDN application which has to control a network behavior, it must have a network driver. On the other hand, a south bound interface is associated with a network element hence it must have a network agent to take commands from the controller and process them.[8]

- **Management & Admin**: The Management plane includesthe tasks that are useful in managing the network device and cannot be done with the control or data plane. For Example, givingresources (like bandwidth) to clients, physical equipment setup, managing reachability over network.

I2RS aims at reaching the RIB of the network element through an asynchronous and enabling real time modification of RIB entries in order to directly control behavior of routing protocols and consequently the whole network topology. The SDN architecture stated above can be modified to include the drivers of I2RS architecture.

2.2 Drivers of I2RS architecture

i) An asynchronous and programmatic interface that provides fast, real time access for minute operations.

[8]SDN provides a solution to easily configure and manage a large number of networking devices with the help of applications. However, as the network complexity is growing with the increase in the number of devices, the need to automate even the simplest of operations is important. Along with that, we need the interfaces and data models which are faster than the existing ones. We need interfaces which support real-time and asynchronous accessing of routing database.

Till date, most of the existing protocols rely upon use of CLI or NETCONF which makes changes into a routers configuration file to make changes. This can be a very slow process considering the growing need for fast operations because the controller will need to first process the configuration files and then implement routing policies out of it. After this, it will add new routes to the routing table. The feedback loop can be quite large and can cause a lot of delay between sensing the network and implementing new policies.

A solution for this slow process is presented by I2RS which aims at entering a network device's RIB and modifying it directly. Here, we are skipping the step of writing into configuration file which makes the process really fast. However, this is not possible with present interfaces and calls for the need of asynchronous and programmatic interface which can provide such real time capabilities.

This makes it one of the most important business drivers of the I2RS. Addition of I2RS to the existing SDN environment can lead to even faster and easier configuration and implementation of routing policies on network devices.[8]

ii) Access tostructured information or state which is presently not accessed by any existing control protocol or method.

[8]Presently there is no way to access structured information from a networking device. This information can be useful to get an abstract view of the network topology. This kind of facility is provided by I2Rs. For example, one such information can be the forwarding table of a device.

Forwarding table is obtained from the routing information base. All the routing protocols maintain a Routing Information Base (RIB) which contains their best routes to all the learned prefixes. The routing devices reads all the routes from the RIB and builds a forwarding table which consists of the best routes for all the known destinations after comparing the routes from all the routing protocol. The forwarding information base is used by the data plane to forward packets over the network.

What I2RS offers is a way to access the active forwarding table of the networking device. This can save a lot of computations and time if we can directly inject routes into the forwarding table without having to go through the process of learning routes, building RIB and then forming a FIB. This kind of capability can speed up the process of injecting new routes into the forwarding table. This kind of capability has not yet been offered by any existing protocol. It can make SDN even more powerful and suitable for large complex networks. [8]

iii) Ability to subscribe to structured event notifications on a router.

[8]I2RS provides ability to get notifications in case of an event on the networking device. This event can be programmed and can vary from arrival of a packet from a particular subnet or when a packet is received on a specific interface or loss of route to a particular network.

For example, this can be particularly useful in a data center with redundant servers for disaster recovery. In this kind of scenario, a customer will back up its data at a server in a remote location. Therefore, in case of a disaster leading to shutting down of the physical network of the customer, the customer can fail over to the virtual network offered by the server in the remote location. However, things can turn pretty bad if the remote server also fails. In such situations, the central controller monitoring the server can set an event for this situation. When the event triggers, the controller can send default routes for the backup server to the customer and turn it active.[8]

[14]Events can be useful for handling several situations. It can be used like if block in programming languages. For example, if a particular event is true then execute a particular action. Further, events can be used in form of interrupts that is when an event occurs, the interrupt handler should be able to perform an action specific to that event. The interrupt handler exists in the control plane of the networking device.[14]

To conclude, this is one of the most important drivers for the I2RS architecture. Such kind of capabilities make I2RS and SDN combination a really powerful one. Since, I2RS provides several features to SDN environment which are not being offered by any other existing protocol, while at the same time handling a complex network of hundreds of thousands of networking equipments can become a nightmare without them. This calls for an urgent need for the development of an I2RS kind of protocol.

2.3 I2RS architecture

2.3.1 Overview of the architecture

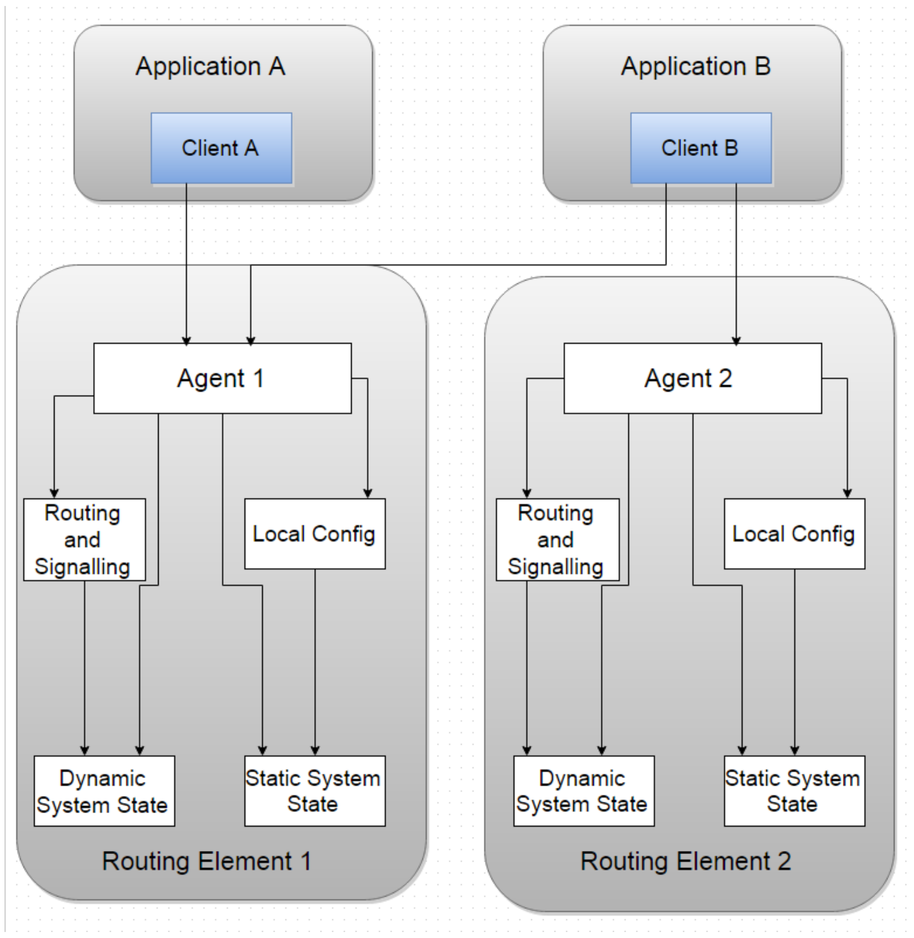


Figure 4 [15]

Figure 4 provides an overview of the I2RS architecture. It shows how I2RS clients get access to the I2RS services with the help of I2RS clients. A single client can be used to provide access to multiple applications. In the above figure, both client A and B are providing access to one application each.

Applications can have access to the services either through local or remote clients. In figure 4, applications A and B are using local clients. We can have a case in which a remote application can have access through client A or B.

Each I2RS client can access one or more I2RS agents. In figure 4, client B is accessing agent 1 and 2 , whereas client A is accessing agent 1. In a similar fashion, an I2RS agent can provide service to one or more applications. For example, agent 1 is serving both client A and client B in the figure.

Agents and clients in the I2RS architecture communicate with the help of asynchronous transmission. As a result, a single client can send request to multiple agents simultaneously. Similarly, a single agent can receive and process requests from several clients at the same time.

- **Routing Element**

  [16]A routing element is a basic router which can implement routing protocols like BGP, OSPF, etc. It does not necessarily need to have a forwarding plane. Examples of a routing element are:i) a router with forwarding plane and ability to run routing protocols such as BGP, ISIS, OSPF, RIP, etc. ii) A BGP speaker functioning as a route reflector. iii) A server that runs ISIS, BGP or OSPF and remotely controls a forwarding plane.[16]

  A routing element is managed locally, with the help of CLI or SNMP. The routing client can be considered as a standard networking device with I2RS supported interface. I2RS provides access for read and write to certain data in the routing element.

  A routing element, as shown in figure 4 consists of several important components like routing and signaling block, I2RS agent, local configuration, dynamic routing state and static routing state. All these components are crucial for offering the services promised by I2RS protocol.

- **Routing and Signaling**
  [16]Routing and signaling is the most important part of the routing element. It is the part that implements internet routing functionality. It includes routing protocols like RIP, BGP, and OSPF. Apart from that it acts as the Routing information base (RIB) management layer.

  All the routing protocols perform their respective routing algorithms to learn the routes to prefixes. Further, they populate their own RIB which consists of all best known routes for the prefixes. It is the responsibility of the routing and signaling block of the routing element to manage the RIB of each routing protocol and to compare the routes from all the protocols in order to forward the best route to each destination to the forwarding information base. [16]

- **Local Configuration**
  [16]A routing element will perform the task of running routing protocols like OSPF, BGP, ISIS on it. The location configuration can be implemented by using existing protocols like NETCONF, SNMP, etc.[16]

Local configuration block of the routing element represents the commands that are stored in RAM. These configurations control the working of the networking device. Therefore, the most common way to implement routing policies and routing protocols is through the use of local configuration which can be accessed for writing using CLI or NetConf.

Over conventional SDN environment, I2RS provides facility to do automated configuration and monitoring of the forwarding database, while still giving some provision to write on the configuration files.

- **Dynamic System State**
  [16]In order to access several statistical details like data flow, counters, logs and local events, the I2RS agent needs to access a state beyond the state of the routing subsystem. This state is required by the network based on I2RS. However, it is not included in the routing subsystem.[16]

  This kind of information state is important for implementing event based notifications, capability advertisement, statistical analysis of the network, etc. The I2RS agent has direct access to the dynamic system state. This direct access makes it possible for the applications running over I2RS to extract all the information stated above without any difficulty. However, they are required to request the agent for any such information for security purposes. If the clients are allowed to have a direct access to the dynamic or static system state then there can be a serious threat to the network.

- **Static System State**
  [16]This system state is also beyond the information provided by the routing subsystem. This state provides information like queuing at the interface. This information is necessary to form an abstract view of the topology.[4]

  Another example of the static system state can be information like traffic at an interface. This can help avoid network congestion which can cause serious down time in a network. Just like the dynamic system state, static system state is also accessed directly by the I2RS agent. Application clients request for the information from static state through the agent. Direct access to the static state is also prohibited for protecting the network from outside attacks.

- **I2RS Agent**
  [16]An I2RS agent performs the task of communicating with the I2RS clients in order to service their requests. It provides the supported I2RS services by extracting the information from the routing element. I2RS has direct communication with different sub-parts of the routing element and also with the

clients. An I2RS agent has full knowledge of the I2RS protocol and is contacted by applications through I2RS clients.[16]

- **I2RS Client**
  [16]I2RS client uses the I2RS protocol to communicate with the I2RS agent and with other elements of the routing subsystem to accomplish various services requested by the user. A client communicates with the I2RS agent to extract information from the elements in the routing system. This may include dynamic or static state information like events, logs, traffic, etc. Further, the client may also want to write some information into the routing element in order to make implement some routing policy.[16]

- **I2RS Services**
  In I2RS protocol, a service may refer to a set of functions which are related to each other and work to access one of the states of the routing system. For example, a set of functions which are working to access the RIB of the networking device in order to read it or modify it may be called as a 'RIB service'. Similarly there can be services of many of other I2RS related information bases such as a FIB service, that is, a service which is called by the client in order to access the forwarding database of the networking device.

As can be seen in figure 4, an agent can communicate with several clients whether or not it serves their requests. The agent communicates with the clients to update them with timely notifications. Timely notifications are very important for the client applications to have the latest view of the network.

Since there can be multiple clients accessing the same, there can be situations where two clients requests to modify information in the routing element at the same time. This is prohibited and to deal with such conflicts, the agent prioritizes the clients if they arrive at the same time. Else, the agent serves them on a first come first servebasis.

The I2RS architecture aims at building a protocol which is simple. In other words, I2RS should be easy to deploy and robust even while implement on complex network topologies. The IETF work force for I2RS focuses on making everything simple and easy to use. Further, another important goal for the I2RS architecture is extensibility. It is clear that I2RS is currently being built to solve a relatively smaller set of issues. However, there should be provision for addition of more features and compatibility with other data model if required in the future.

2.3.2 Security Considerations

[16]This I2RS architecture illustrates interfaces that requiresome serious security measures. As an architecture, I2RS has beendesigned to use some existing

protocols. Examples of the existing protocol which the I2RS work grouphas selected to use are NETCONF and RESTCONF. The I2RS protocol architecture requires including security for theexisting protocols so that they can be utilized in the I2RS architecture.After modifying the existing protocols like NETCONF and RESTCONF to fit the I2RS requirements, they will be scrutinized to match the security requirements for the I2RS architecture.

As the I2RS architecture utilizes some existing protocols, this part describes the ideal security for I2RS environmentalong with some details on :i) identity and authentication, ii)authorization, and iii) client redundancy. Each existing protocol selected to be used by the work group will be thoroughly checked to fir the I2RS security environment.

Firstly, the desired security requirements are described. The I2RS Agent part of thearchitecture is a trusted part of that Routing Element. For example, itcan be part of an application package which has been signed by the distributer for the agent or the agent is installed already by some operator on the routing element. The I2RS Agent is required to have its own authentication mechanism by which it canauthenticate the identity of the I2RS clients which try to associate with it. To support many and fast interactions between the I2RSAgent and I2RS Client, it is considered that the I2RS Agent can alsocache that particular I2RS Clients are trusted and don't have to go through authentication check for every interaction.

On other hand, an I2RS client should also be able to authenticate its I2RS agent. An I2RS client should have full confidence that it is connected to a correct agent so that it's requests don't end up reaching an attacker or someone else. It has to make sure that it's read/write requests are handled by the correct routing element.

Integrity, confidentiality, and replay protection are importantat different levels and different aspects of the I2RS client and agent communication. The communication link that is established between the client and the agent should be secured for integrity and confidentiality as this link will be used in the future for read/write as well. Therefore, connecting to a non-trusted client or agent can be quite hazardous which requires the implementation of above mentioned security aspects on the communication channels.

Integrity,confidentiality, and replay protection are not necessary for all types of interactions happening within the I2RS architecture. For example, when a client asks for the prefix announcement from an OSPF process from the routing element, it does not have to have replay protection or confidentiality but it must have integrity , that is , all the data being requested must be transmitted error free.Further, packets containing the details about statistics do not even have to have integrity. For instance, an I2RS client sometimes asks for certain event notifications from an agent. Therefore, to avoid the data from being compromised to an attacker, such requests should only be taken from the clients which have already been authenticated and stored in the cache of the agent. Any other client requesting for such information should not be entertained.[16]

- **Identity and Authentication**

  [16]As said above, the communication between the I2RS agent and client should be fully authenticated. All the packets transferred between the two should be checked for integrity that is the data should not be damaged or modified on the way. In an idealarchitecture, even information collection and notification should beprotected.[16]

  [16]I2RS clients work for the applications. These applications need not be authenticated. However, there should be some identification mechanism for the applications. These identifiers will be passed to the I2RS agent by the client and will help in uniquely identifying each application. This kind of identification is needed to support operations such as troubleshooting and maintaining log files.[16]

- **Authorization**

  [16]All processes using I2RS such as modification and extraction of system information, shouldbe done after proper authorization. This kind of authorization is based upon the kind of role provided by the I2RS client and agent for performing the specific tasks. For example, operations involved in writing the data are identified in similar type while the processes involved in reading data in another type.

  Upon authenticating an I2RS client, the agent receives the identity provided by the client and links to the specific role that has to be performed for the client. The role can be anything depending on the type of task asked to be performed. It can vary from accessing RIB to extracting logs. Therefore, there can be several identities with the same role.[16]

- **Client Redundancy**
  [16]I2RS should be able to support redundant clients.One requirement for this can be tohave a primary and a secondary network application. Both the applications have same identifier but still should be able to authenticate successfully.I2RS does not need to have a continuous transport session and can open multiple transport connections at a time. This can be helpful for handling multiple clients at the same time. Therefore, for handling the logging information for each session, the agent needs to maintain the identifier along with the basic log information corresponding to each identifier.

<u>How I2RS fits in SDN architecture</u>

The I2RS architecture has to be able to link to the present SDN architecture. To do so the I2RS client interact with the SDN controller. The SDN controller will interact with theI2RS agent on the routing system. Since all the control plane work is done by the SDN controller.

The SDN controller will do the job of relaying the request of I2RS clients to the I2RS agent. The I2RS agent will now communicate with the client indirectly and serve all the requests made by the client to the controller, which will relay the service back to the client application.

The client and the agent should be authenticated by the SDN controller. Further agent should have a mechanism to authenticate the controller so that it does not expose its information to an attacker posing as a controller. There need to be addition of functionality in the present SDN controllers in order to give support to the I2RS protocol

Section 3- Demo of SDN and need for I2RS

In this section we will implement a basic network topology using open vswitch controlled by a pox controller on mininet. As can be seen in the Figure 5, three hosts are connected to an open vswitch . The switch here, does not have a brain of its own and will provide us just the data plane for the data to flow. The controller will provide the brain to the switch and tell it how to forward the packets.

We are going to run a basic SDN based network using mininet and pox controller and see the result for different types of behaviors installed into the OVS by the controller. Later, we will see how I2RS can benefit the SDN based network shown in figure 5 and how we can be benefitted by different features offered by the I2RS protocol

POX Controller - [17]POX controller at its core, is a platform for development of network controller software using Python language. POX is being used to communicate with the open vswitch in the network. Further, it is used for controller design, network virtualization, debugging and designing networks.[17]

Mininet - [18]Mininet is a software application which is used to create realistic virtual network topologies, running real kernel, switch and applications with the use of a single command and within a few milliseconds.[18]

Open vSwitch - [19]OpenvSwitch, also known as OVS, is an industry standard open source distributed multilayer virtual switch. OVS does the task of providing a switching stack for hardware virtualization environment. It also does the task of communicating with the network controllers in the SDN environment.[19] It acts as a hardware device with just the forwarding plane of its own. The control plane for the switch is provided by the SDN controller. Open vswitch is being used as the virtual switch by mininet in the network topology stated in figure 5.

In the network shown below, we will use the pox controller to control the network shown in figure 5. Initially, we will try to run the topology without any controller and see how the network behaves. We will ping all hosts from all hosts and see if the open vswitch sends the icmp packets to correct destination or not.

Later, we will run a script of hub in the controller so that it controls the vswitch to work as a hub. We will again do the ping test and see the flows in the vswitch using dpctl. Further, we will run script of an L2 learning switch on the controller and again do the ping test and check the dpctl flows on the vswitch. The results of the vswitch as a hub will be compared with the results of vswitch as a L2 switch.

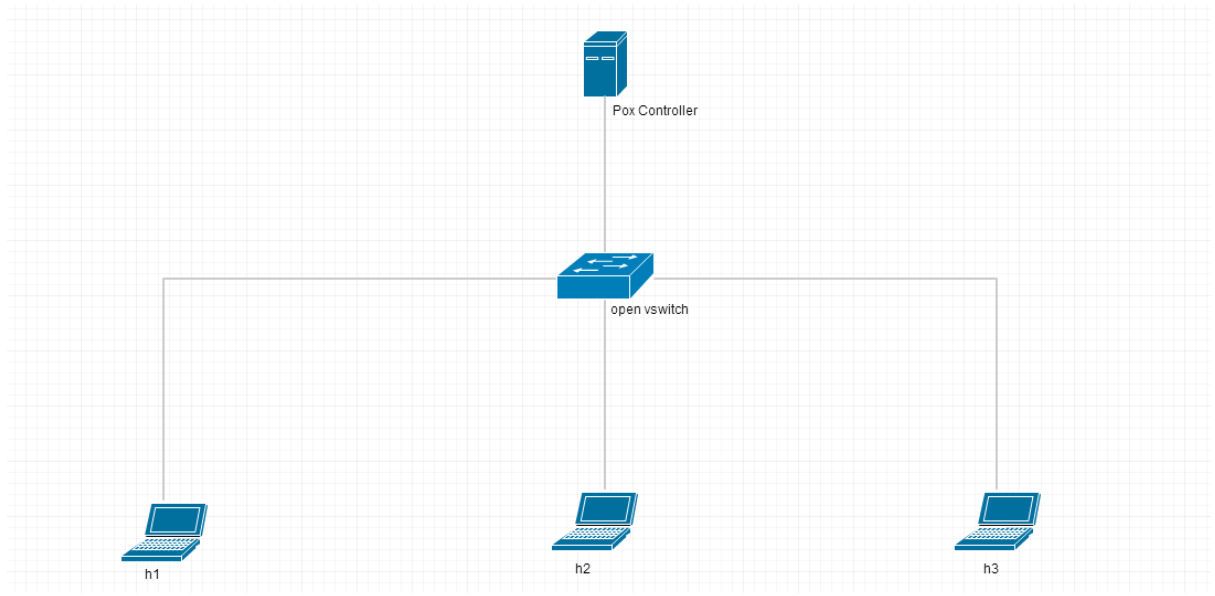
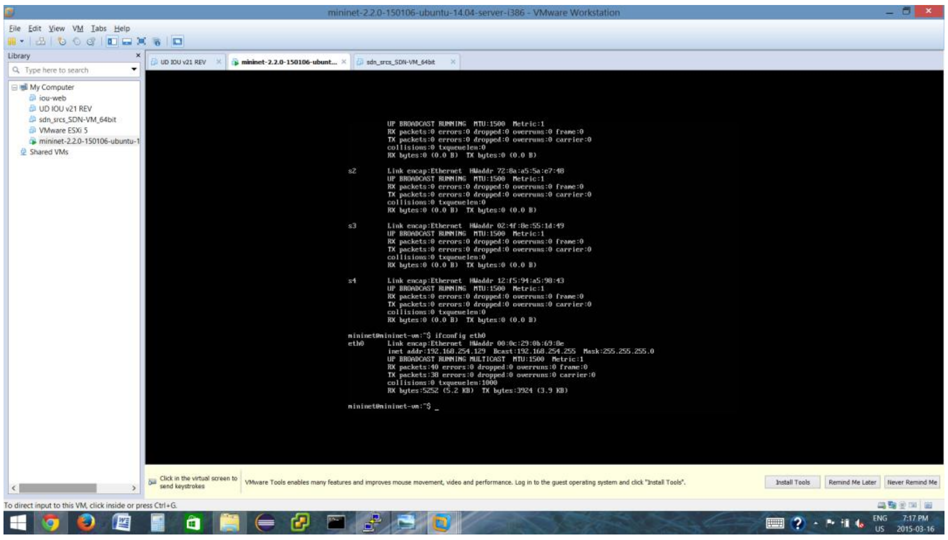


Figure 5

Firstly, we test run the mininet virtual machine as seen in Figure 6. We will use mininet to simulate the open vswitch and hosts. The pox controller will also be run on the mininet. Mininet's eth0 interface is used to establish an SSH session. This is done on a VMware workstation running on windows 8 desktop. The mininet is running as a virtual machine on VMware workstation 10. After establishing a telnet session with mininet VM using putty or terminal, mininet is ready to make network topologies as shown in figure 7.

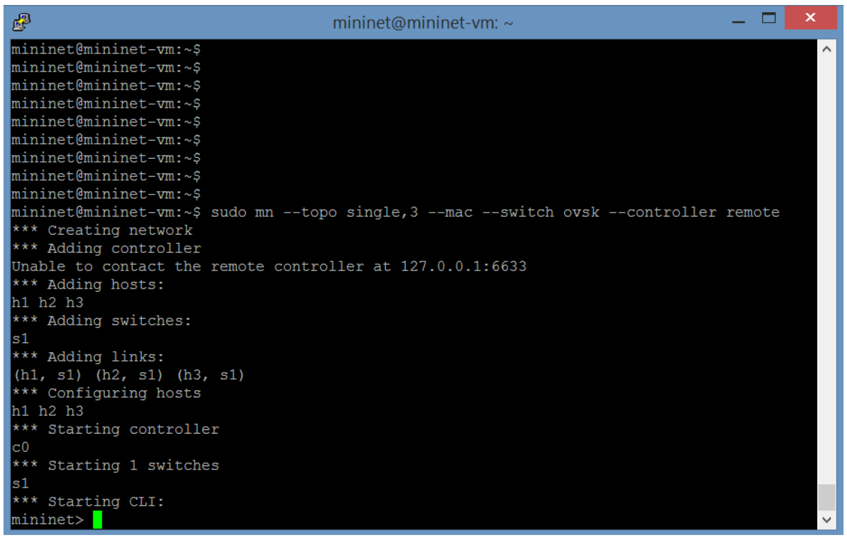

mininet-2.2.0-150106-ubuntu-14.04-server-i386 - VMware Workstation
File Edit View VM Tabs Help
Library
My Computer
iou-web
UD IOU v21 REV
sdn_srcs_SDN-VM_64bit
VMware ESXi 5
mininet-2.2.0-150106-ubuntu-1
Shared VMs
Click in the virtual screen to send keystrokes
VMware Tools enables many features and improves mouse movement, video and performance. Log in to the guest operating system and click "Install Tools".
Install Tools
Remind Me Later
Never Remind Me
To direct input to this VM, click inside or press Ctrl+G.

```
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

s2        Link encap:Ethernet  HWaddr 72:8a:a5:5a:e7:48
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

s3        Link encap:Ethernet  HWaddr 02:4f:8e:55:1d:49
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

s4        Link encap:Ethernet  HWaddr 12:f5:94:a5:90:43
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet@mininet-vm:~$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0c:29:0b:69:8e
          inet addr:192.168.254.129  Bcast:192.168.254.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:40 errors:0 dropped:0 overruns:0 frame:0
          TX packets:38 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5252 (5.2 KB)  TX bytes:3924 (3.9 KB)

mininet@mininet-vm:~$ _
```

Figure 6

<u>CASE 1 - No script running on the controller</u>

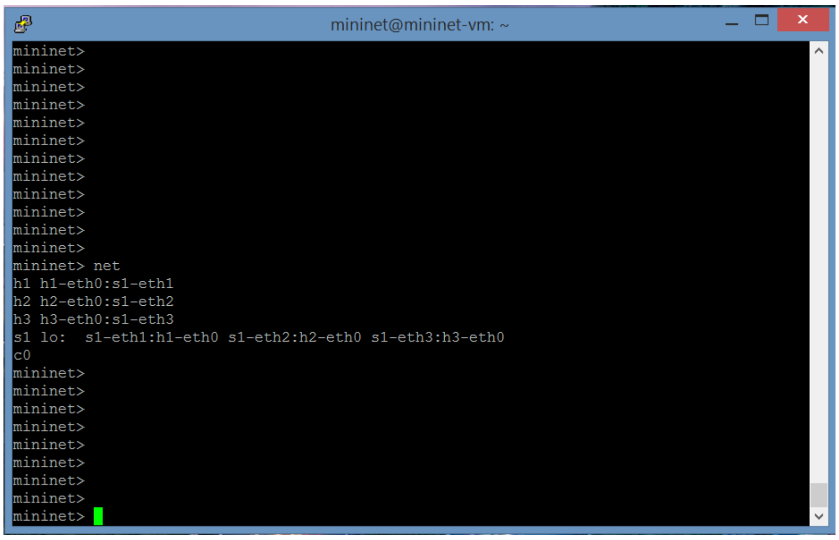
Figure 7 shows the use of a simple command to create the topology. Here it can be seen that a single, 3 host topology is being created with an open vswitch represented by ovsk and a controller which is remote. Here --mac means that mac address should be assigned to the hosts." -- controller remote" means that the SDN controller is remotely located



```
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

Figure 7

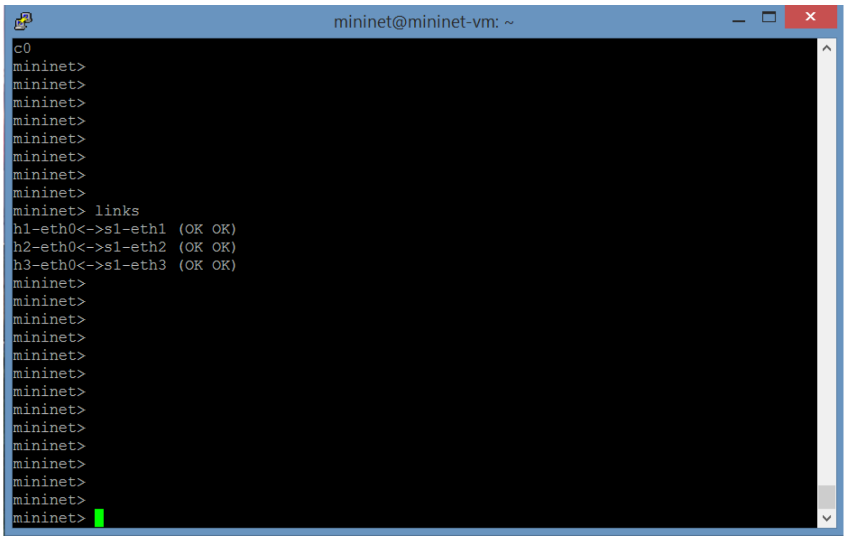
Figure 8 verifies the creation of the three host topology with h1,h2 and h3 being the three hosts where as s1 is the open vswitch. Controller is represented by c0. Currently, the switch does not have any instructions from the controller therefore it has no configuration. The controller is empty as well because we have not initialized the pox controller yet. This is done using net command in the mininet. It simply specifies the network that mininet is currently simulating by showing a node for each network device that exist in the network topology. In this case, it is showing the devices in our 3 host topology along with the controller and the switch.



```
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0
c0
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
```

Figure 8

After the topology has been established in the mininet, it is important to verify the status of all the links between the network devices. Links command in mininet does the job of describing the current status of the links. In our case, all the links are up and ready to send packets over them as can be seen in figure 9

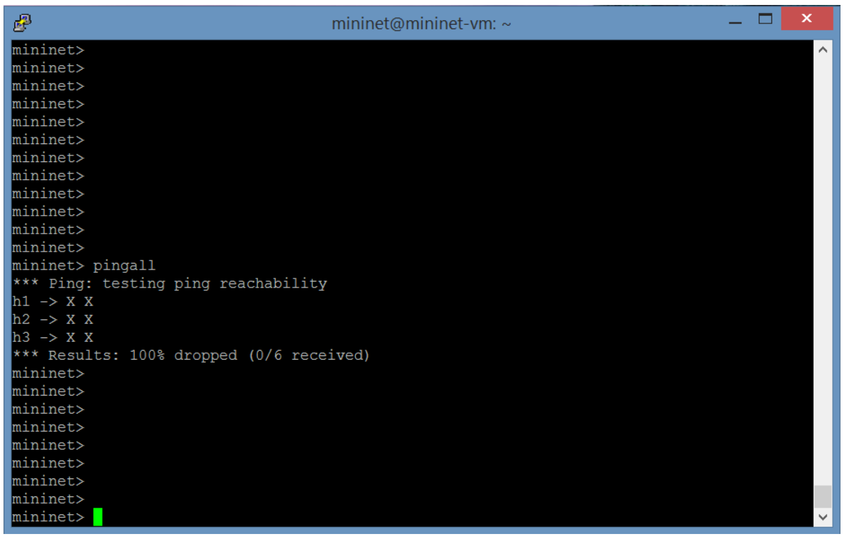


```
c0
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
h3-eth0<->s1-eth3 (OK OK)
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
```

Figure9

The next step is to try and ping all the hosts from all the hosts. This is done using pingall command. Since the switch being an open vswitch has to wait for commands from the controller which it has not got yet, it will drop all the packets. The switch does not know what to do with the packets. We have not initialized any controller. When the controller is initialized it will communicate with the switch and tell it how to behave.

Till then, the switch will act as a blank switch. It can be seen in figure 10 that all the packets have been dropped.

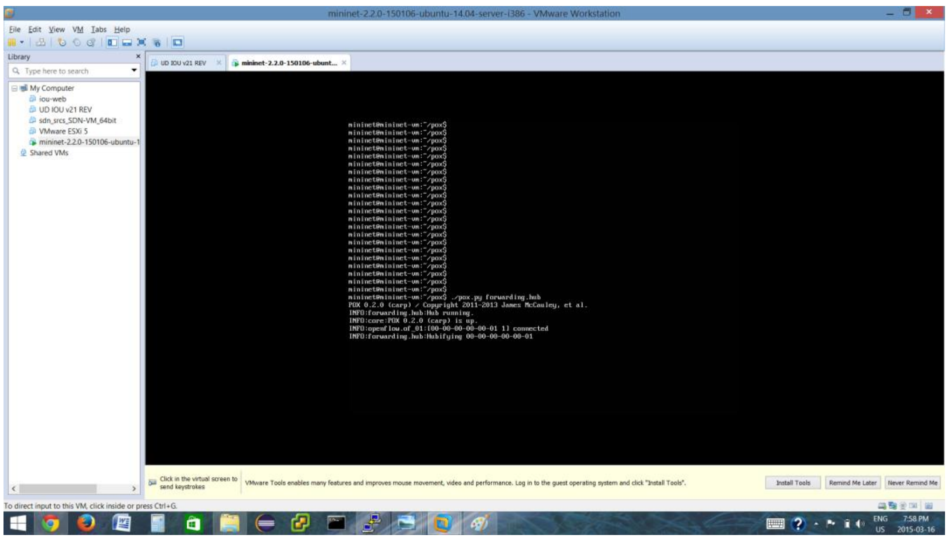


```
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
```

Figure 10

CASE 2 - Controller running hub.py script

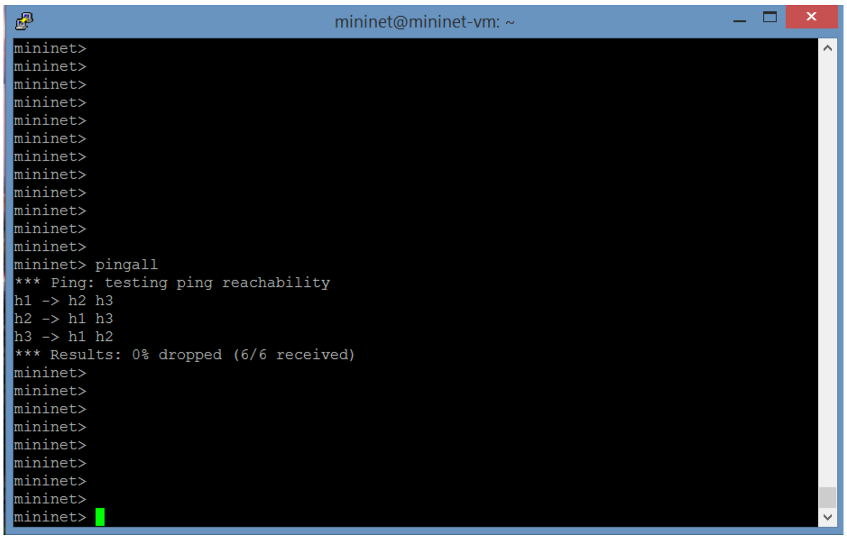
In order to give some instructions to the switch on how to forward the packets, we will initialize a pox controller on the mininet virtual machine. Firstly, we will make the open vswitch to act as a hub. what hub does is that it when it receives a packet on port it just floods it to all other ports. Therefore, in the flow entries of a hub we will see that there will be just one entry and that will be to flood. This makes a hub much more simpler than a switch; however, this also makes it more prone to collisions because a hub floods its incoming packets to all its ports except for the port on which the packet was received.



```
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$ ./pox.py forwarding.hub
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:forwarding.hub:Hub running.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
```

Figure 11

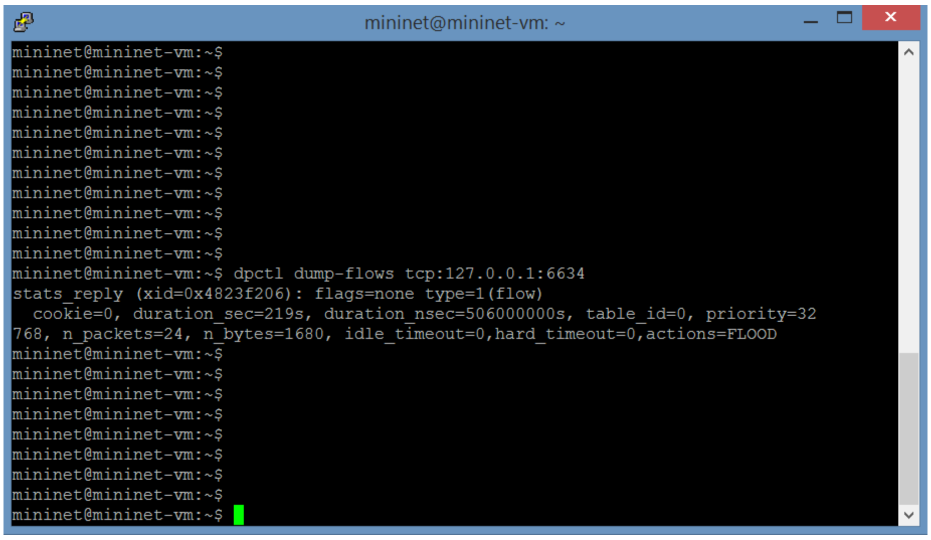
Since the controller has started with a script called hub.py, we can expect the open vswitch to behave like a hub. We can verify it by using the pingall statement again. The pingall statement worked this time and none of the packets were dropped as seen in figure 12. Now the vswitch will act like a hub until stopped by the controller. The switch is offering its data plane for the packets to flow. The control plane of the switch is manipulated by the poxcontroller, This is what software defined networking is all about. However, this process requires to speed up because the controller still has to access the .config file in the switch. This is why i2rs is here. If there are hundreds of thousands of open vswitches being controlled by the controller at the same time then we can save so many seconds of time if we have access to the RIB table.



```
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
```

Figure 12

Next, we will see the flow table entries in the open vswitch using the dpctl dump-flows command. We can see that there is just one entry which says actions=FLOOD. It means that whatever the vswitch will receive on its port it is going to flood it to all its ports.

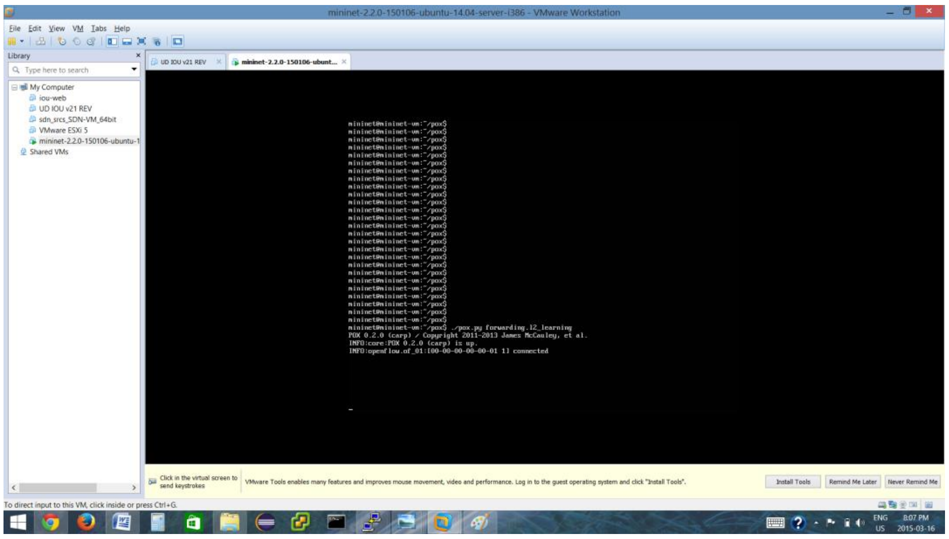


```
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x4823f206): flags=none type=1(flow)
  cookie=0, duration_sec=219s, duration_nsec=506000000s, table_id=0, priority=32
768, n_packets=24, n_bytes=1680, idle_timeout=0,hard_timeout=0,actions=FLOOD
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
```

Figure 13

CASE 3 - Controller running the L2 learning switch script

Now the next step is to change the behavior of the switch. We can make it work like a switch instead of a hub now. All we have to do is to run the script of l2_learning on the pox controller. This will stop the hub behavior of the vswitch and turn it into a learning switch. Now the switch will learn about mac addresses of different hosts and install them in its mac table whenever it receives a packet from a host. A switch is a little more complex and much more reliable than a hub.



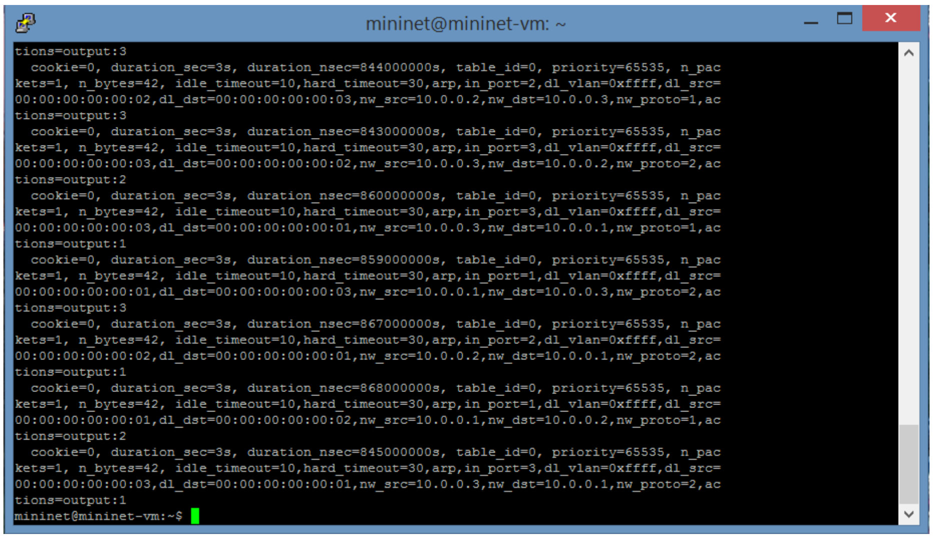
mininet-2.2.0-150106-ubuntu-14.04-server-i386 - VMware Workstation

```
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$ ./pox.py forwarding.l2_learning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
```

Figure 14

Next, we can again run the pingall command. The pingall will give same result as it gave for a hub. However, the flow table entries for a switch will be different from a hub.

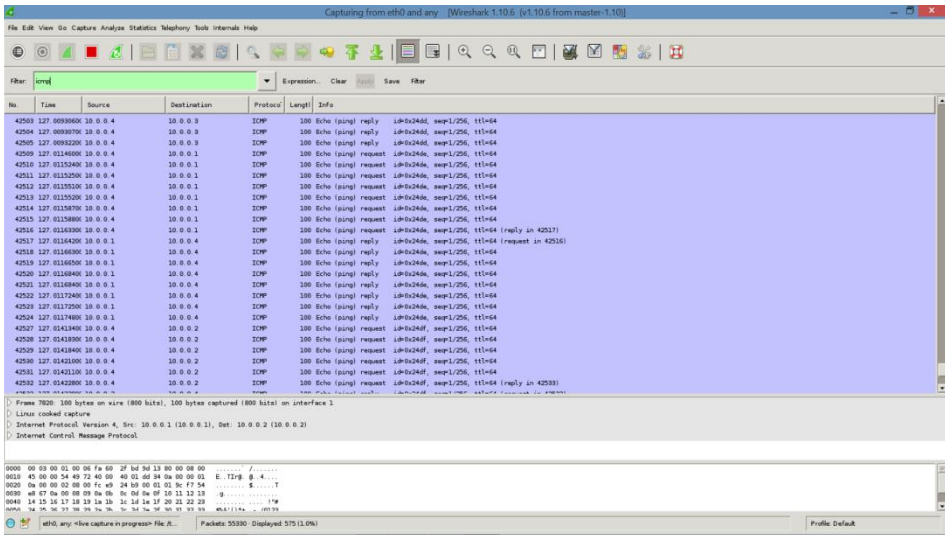
In case of hub there was just one entry which flooded all packets to all ports. The switch has a different entry for each destination mac address. It will only flood if there is no entry in its mac table. If it has an entry in its mac table it means it knows about the destination port and will flood the packet only to that single port.



```
tions=output:3
  cookie=0, duration_sec=3s, duration_nsec=844000000s, table_id=0, priority=65535, n_pac
kets=1, n_bytes=42, idle_timeout=10,hard_timeout=30,arp,in_port=2,dl_vlan=0xffff,dl_src=
00:00:00:00:00:02,dl_dst=00:00:00:00:00:03,nw_src=10.0.0.2,nw_dst=10.0.0.3,nw_proto=1,ac
tions=output:3
  cookie=0, duration_sec=3s, duration_nsec=843000000s, table_id=0, priority=65535, n_pac
kets=1, n_bytes=42, idle_timeout=10,hard_timeout=30,arp,in_port=3,dl_vlan=0xffff,dl_src=
00:00:00:00:00:03,dl_dst=00:00:00:00:00:02,nw_src=10.0.0.3,nw_dst=10.0.0.2,nw_proto=2,ac
tions=output:2
  cookie=0, duration_sec=3s, duration_nsec=860000000s, table_id=0, priority=65535, n_pac
kets=1, n_bytes=42, idle_timeout=10,hard_timeout=30,arp,in_port=3,dl_vlan=0xffff,dl_src=
00:00:00:00:00:03,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.3,nw_dst=10.0.0.1,nw_proto=1,ac
tions=output:1
  cookie=0, duration_sec=3s, duration_nsec=859000000s, table_id=0, priority=65535, n_pac
kets=1, n_bytes=42, idle_timeout=10,hard_timeout=30,arp,in_port=1,dl_vlan=0xffff,dl_src=
00:00:00:00:00:01,dl_dst=00:00:00:00:00:03,nw_src=10.0.0.1,nw_dst=10.0.0.3,nw_proto=2,ac
tions=output:3
  cookie=0, duration_sec=3s, duration_nsec=867000000s, table_id=0, priority=65535, n_pac
kets=1, n_bytes=42, idle_timeout=10,hard_timeout=30,arp,in_port=2,dl_vlan=0xffff,dl_src=
00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_proto=2,ac
tions=output:1
  cookie=0, duration_sec=3s, duration_nsec=868000000s, table_id=0, priority=65535, n_pac
kets=1, n_bytes=42, idle_timeout=10,hard_timeout=30,arp,in_port=1,dl_vlan=0xffff,dl_src=
00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_proto=1,ac
tions=output:2
  cookie=0, duration_sec=3s, duration_nsec=845000000s, table_id=0, priority=65535, n_pac
kets=1, n_bytes=42, idle_timeout=10,hard_timeout=30,arp,in_port=3,dl_vlan=0xffff,dl_src=
00:00:00:00:00:03,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.3,nw_dst=10.0.0.1,nw_proto=2,ac
tions=output:1
mininet@mininet-vm:~$
```

Figure 15

We can now verify the icmp packets on wireshark running on the mininet interfaces in our network. Figure 16 shows various ICMP packets from different hosts going over the network.


Capturing from eth0 and any [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: icmp
Expression... Clear Apply Save Filter

```
No.    Time         Source     Destination  Protocol Length Info
 42503 127.0093060( 10.0.0.4   10.0.0.3     ICMP     100 Echo (ping) reply    id=0x24dd, seq=1/256, ttl=64
 42504 127.0093070( 10.0.0.4   10.0.0.3     ICMP     100 Echo (ping) reply    id=0x24dd, seq=1/256, ttl=64
 42505 127.0093220( 10.0.0.4   10.0.0.3     ICMP     100 Echo (ping) reply    id=0x24dd, seq=1/256, ttl=64
 42509 127.0114600( 10.0.0.4   10.0.0.1     ICMP     100 Echo (ping) request  id=0x24de, seq=1/256, ttl=64
 42510 127.0115240( 10.0.0.4   10.0.0.1     ICMP     100 Echo (ping) request  id=0x24de, seq=1/256, ttl=64
 42511 127.0115250( 10.0.0.4   10.0.0.1     ICMP     100 Echo (ping) request  id=0x24de, seq=1/256, ttl=64
 42512 127.0115510( 10.0.0.4   10.0.0.1     ICMP     100 Echo (ping) request  id=0x24de, seq=1/256, ttl=64
 42513 127.0115520( 10.0.0.4   10.0.0.1     ICMP     100 Echo (ping) request  id=0x24de, seq=1/256, ttl=64
 42514 127.0115870( 10.0.0.4   10.0.0.1     ICMP     100 Echo (ping) request  id=0x24de, seq=1/256, ttl=64
 42515 127.0115880( 10.0.0.4   10.0.0.1     ICMP     100 Echo (ping) request  id=0x24de, seq=1/256, ttl=64
 42516 127.0116330( 10.0.0.4   10.0.0.1     ICMP     100 Echo (ping) request  id=0x24de, seq=1/256, ttl=64 (reply in 42517)
 42517 127.0116420( 10.0.0.1   10.0.0.4     ICMP     100 Echo (ping) reply    id=0x24de, seq=1/256, ttl=64 (request in 42516)
 42518 127.0116630( 10.0.0.1   10.0.0.4     ICMP     100 Echo (ping) reply    id=0x24de, seq=1/256, ttl=64
 42519 127.0116650( 10.0.0.1   10.0.0.4     ICMP     100 Echo (ping) reply    id=0x24de, seq=1/256, ttl=64
 42520 127.0116840( 10.0.0.1   10.0.0.4     ICMP     100 Echo (ping) reply    id=0x24de, seq=1/256, ttl=64
 42521 127.0116840( 10.0.0.1   10.0.0.4     ICMP     100 Echo (ping) reply    id=0x24de, seq=1/256, ttl=64
 42522 127.0117240( 10.0.0.1   10.0.0.4     ICMP     100 Echo (ping) reply    id=0x24de, seq=1/256, ttl=64
 42523 127.0117250( 10.0.0.1   10.0.0.4     ICMP     100 Echo (ping) reply    id=0x24de, seq=1/256, ttl=64
 42524 127.0117480( 10.0.0.1   10.0.0.4     ICMP     100 Echo (ping) reply    id=0x24de, seq=1/256, ttl=64
 42527 127.0141340( 10.0.0.4   10.0.0.2     ICMP     100 Echo (ping) request  id=0x24df, seq=1/256, ttl=64
 42528 127.0141830( 10.0.0.4   10.0.0.2     ICMP     100 Echo (ping) request  id=0x24df, seq=1/256, ttl=64
 42529 127.0141840( 10.0.0.4   10.0.0.2     ICMP     100 Echo (ping) request  id=0x24df, seq=1/256, ttl=64
 42530 127.0142100( 10.0.0.4   10.0.0.2     ICMP     100 Echo (ping) request  id=0x24df, seq=1/256, ttl=64
 42531 127.0142110( 10.0.0.4   10.0.0.2     ICMP     100 Echo (ping) request  id=0x24df, seq=1/256, ttl=64
 42532 127.0142280( 10.0.0.4   10.0.0.2     ICMP     100 Echo (ping) request  id=0x24df, seq=1/256, ttl=64 (reply in 42533)

▷ Frame 7820: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface 1
▷ Linux cooked capture
▷ Internet Protocol Version 4, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.2 (10.0.0.2)
▷ Internet Control Message Protocol

0000  00 03 00 01 00 06 fa 60  2f bd 9d 13 80 00 08 00   ........ /.......
0010  45 00 00 54 49 72 40 00  40 01 dd 34 0a 00 00 01   E..TIr@. @..4....
0020  0a 00 00 02 08 00 fc e9  24 b9 00 01 01 9c f7 54   ........ $......T
0030  e8 67 0a 00 08 09 0a 0b  0c 0d 0e 0f 10 11 12 13   .g...... ........
0040  14 15 16 17 18 19 1a 1b  1c 1d 1e 1f 20 21 22 23   ........ .... !"#

eth0, any: <live capture in progress> File: /t...   Packets: 55330 · Displayed: 575 (1.0%)   Profile: Default
```

Figure 16

In the above example, running I2RS between the controller and the hosts can improve the performance and efficiency of the network. The improvement might not be noticeable in case of the above example with few devices. However, in case of data centre where there are hundreds of thousands of hosts connected to thousands of forwarding devices, there can be a significant difference in performance and efficiency. The read write cycles onto the network devices can become so small that the operations may appear to be real time.

Section 4- Comparison of CLI , NETCONF and I2RS.

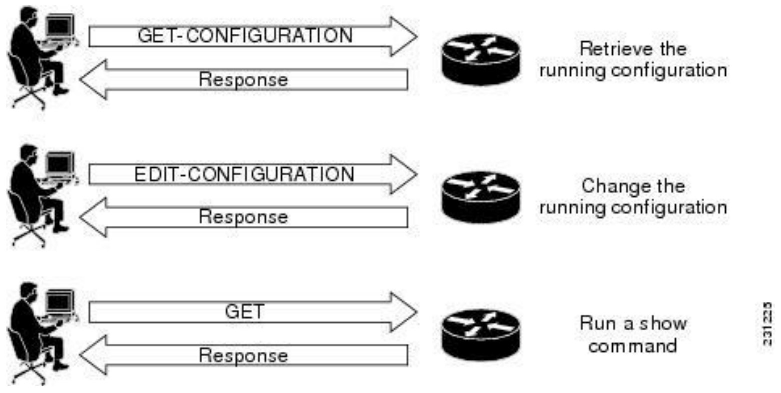


Figure 17

[20]To begin with we have CLI as a method to control the behavior of a networking device. This process is very conventional and requires an administrator to directly enter the commands into networking devices. Here, a user enters commands in the form of human readable text. However the process of using CLI is very slow and is most of the time not automated. The only way to control the routing protocols like BGP, OSPF, etc is to enter some predefined commands which are supported by the device firmware. The CLI is reliable but very slow. Imagine having 100 devices in a data center. Going to each device and entering commands on one to one basis can be a very tiring task.

In order to make the task of programming and managing the network devices faster, NETCONF was introduced. NETCONF defines a mechanism through which configuration data can be managed. It provides a way to retrieve, upload and manipulate configuration data. To achieve this, NETCONF uses remote procedural calls (RPC) encoded in the XML format. These RPCs are sent toserver which then replies with configuration data encoded in the XML format.

NETCONF does a great job when it comes to dealing with configuration data, but does not help much when it comes to RIB or controlling the policies of the routing protocols directly. The feedback loop between network elements, statistical gathering and network programming is still very slow. I2RS is an attempt to speed up this loop.

Till now most of the protocols attempted at controlling the network through the configuration files. I2RS goes a little deeper and attempts at injecting routes directly into the routing information base (RIB) of a router. This makes the feedback loop very fast. It is possible to take statistical data from a router and implement changes in its RIB accordingly in a real time manner. I2RS enables an application called I2RS client which can talk to RIB and routing protocols directly.

Apart from its ability to modify RIB directly, I2RS also has a topology manager feature which enables the administrator to have an overview of the network. Earlier, a user had access to the partial network topology offered by the routing protocols . It was partial in the sense that RIP will provide information about routers running RIP only or BGP will give an overview of BGP topology only. With I2RS topology manager, it is possible to get a global view of the entire network topology. The topology manager collects topology information from all sort of sources like network elements, routing protocols, inventory collection and statistics collection.

Further, I2RS provides a conceptual or an operational environment where the routing protocols like RIP, BGP, etc can be controlled by external factors. I2RS has the ability to control the policy behavior of the routing protocols which can affect the routes generated by the protocols.[20]

Section 5- Business Drivers, and I2RS applications and Use Cases

5.1 I2RS Business Drivers

I2RS aims at breaking the paradigms of conventional data retrieval techniques. The growth of SDN and Yang's data model triggers the need for a more generic data exchange model. However, it does not aim at replacing any of the existing protocols of data distribution but on the contrary to utilize on them in order to create the new data exchange model.

Apart from the features of the I2RS which have been specified above, there has been demands for Pub/Sub compatibility. A pub/ sub format is the one in which the sender acts as a publisher and in place of sending the data to a particular receiver the data is categorized into classes. The sender might not have any information about the receiver. The receivers are called as subscribers they receive data by associating themselves to a particular class. The receiver might not be having any information about the sender.

In addition to these, I2RS use cases have specific requirements . A few of them have been mentioned below :-

- [21]The I2RS interface must let userssubscribe for specific data with following ways: data can be added synchronously or asynchronously depending the need of the user.
- The interface should let a user subscribe to his desired parts of the data model.
- Keep track of theroutes injected in the routing information base of each forwarding device. It should also be able to send real-time notifications upon addition or deletion of a route.
- The I2RS client should have mechanism to ask the agent to send notification to the client in case the bgp prefixes stated by the client gets added or deleted. The agent should be able to inform the client with the help of subscribing the client to such updates.
- The interface should have some kind of way to inform its clients about important changes in IGP routes. It can be done with the help of subscribing the clients to the agent.
- The agent should provide a way for subscription of the client to listen to notifications regarding MPLS LDP state modifications.
- I2RS must be able to extract large amount of data from the network related to the network topology and network statistics while putting minimal burden on the CPU and memory.[21]

Apart from these requirements there are several other requirements such as need for pub/sub interface security, time sensitivity of delivery of data, etc.

5.2 I2RS Use Cases

There are several requirements a programmatic interface offered by I2RS can fulfil. The interface can go directly into the control plane used to discover best path in a network and also interact with the routing information base. These requirements listed below are all the requirements which are needed by different use cases.

Use Case Requirements

- [22]Keep track of the routes injected in the routing information base of each forwarding device. It should also be able to send real-time notifications upon addition or deletion of a route.

- To be able to put routes into RIB of each networking device. It should also install the following information in the routes : - The source and destination address, exit interface, next-hop, local preference, etc

- The I2RS protocol should be able to install a discard route into routing table in order to drop all the traffic for a particular prefix.

- The quality to communicate with various routing policies configured on the forwarding devices in order to tell them about the policies configured by the routing protocols. This communication should be done over some existing protocol like NETCONF.

- The need to interact with various other protocol which work at network traffic level in order to analyze the path performance.

- To be able to access the RIB of each networking device for reading the fields such as source and destination prefix, next-hop, local preference, metric, AD value, etc.

- To be able to access the tables of other local protocols working in the device. This type of accessrequires the need of an import/export interface which can deliver the information from different protocols in a generic representation. An interpretation which can be understood by all the protocols and helps in inter-communication between the protocols[22]

5.2.1 Use Case 1 - Distributed Reaction to Network Based Attacks

[22]It can be quite challenging to modify the control plane to reroute traffic for a destination without disturbing the standard configuration (filters,metrics, and other policy mechanisms). This can be challenging for a network administrator experiencing an attack in his network. The feature to redirect some flows of traffic into the analyzer and back on the fly is quite crucial in such situations. The following network diagram provides an illustration of theproblem.

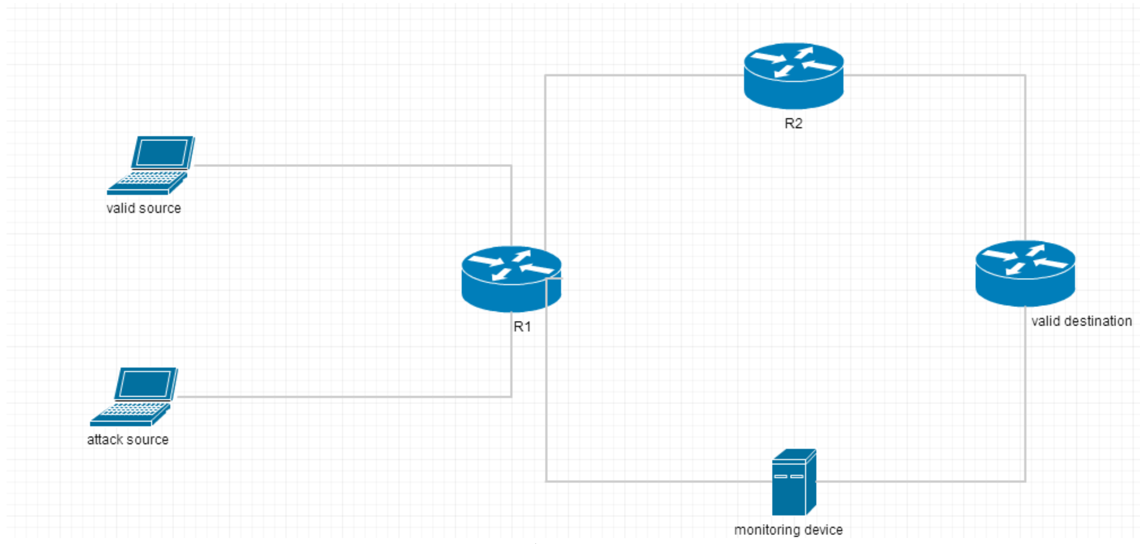


Figure 18

Modifying cost between R1 and R2 to move attack traffic through the analyzer will also make authentic traffic move through the analyzer. Passing valid traffic through the analyzer poses the problem of jitter, delay for the real traffic and also of overload for the analyzer.

An I2RS controller can be installed in between the detection of the attack and the control plane in order to decide which traffic has to be sent over to other devices and which traffic not.[22]

5.2.2 Use Case 2 - Remote Service Routing

[22]In hub and spoke topology there is usually an issue between the routes stored at hubs, and optimal routing via the overlay network. Most common solution isto store the routes at some kind of a centralized routing server. This server contains all the reachable destinations with their next hop information stored. We just need some kind of protocol for communication between the spokes and the centralized server.

An I2RS solution would use the same concept as above but with a separate control plane. Remote sites would register (or advertise throughsome standard routing protocol, such as BGP), the reachabledestinations at each site, along with the address of the router (orother device) used to reach that destination. These would, asalways, be stored in a route server (or several redundant routeservers) at a central location.

When a remote site sends a set of packets to the central locationthat are eventually destined to some other remote site, the centrallocation can forward this traffic, but at the same time simplydirectly insert the correct routing information into the remotesite's routing table. If the location of the destination changes,the route server can directly modify the routing information at theremote site as needed.

An interesting aspect of this solution is that no new and specializedprotocols are needed between the remote sites and the centralizedroute server(s). Normal routing protocols can be used to notify thecentralized route server(s) of modifications in reachabilityinformation, and the route server(s) can respond as needed, based onlocal algorithms optimized for a particular application or network.For instance, short lived flows might be allowed to simply passthrough the hub site with no reaction, while longer lived flows mightwarrant a specific route to be installed in the remote router.Algorithms can also be developed that would optimize traffic flowthrough the overlay, and also to remove routing entries from remotedevices when they are no longer needed based on far greaterintelligence than simple non-use for some period of time.[22]

5.2.3 Use Case 3 - Within Data Center Routing

[22]Data Centers have grown into huge networks of thousands of servers and hosts. Data Centers run BGP with ECMP,ISIS or other protocols to connect all the devices. Data centers these days are designed around 3 level topology with server, top-of-rack switches, aggregationswitches, and router facing the Internet.

An important characteristic of the data center is to read routes from the central routing servers. The feedback loop between learning about new topology changes and installing these changes into the routing table and initializing new policies is a very crucial feature of the data center environment. Without I2RS, data centers are using extra physical topologies or logicaltopologies to work around the features.

An I2RS solution would use the same elements but a separate control plane. It would use a I2RS programmatic interfaces to read and write routing information into the RIB of networking devices.Access via I2RS could have a fastfeedback loop and cold save a lot of time.[22]

5.2.4 Use Case 4 - Route control via Indirection

[20]We can have an application to install routes directly into RIB. For example, we can have an application installed route in the RIB when BGP tries to resolve its IGP next hop. Redistribution of a route from one protocol to another is done by RIB. That route can be a route installed by I2RS.[20]

5.2.5 Use Case 5 - Policy-Based Routing of Unknown Traffic

[20]A static route can be installed into RIB to route the unknown traffic. The static route installed in RIB can be used to route the unrecognised traffic toward an application. The static route can send the traffic to a server which is being pointed to by another route. This can be done using indirection. The input traffic can be recognised and then whatever action is necessary can be taken.[20]

5.2.6 Use Case 6 - Services with fixed hours

[20]This type of use case is needed in case of applications where the service is required only for a certain period of time. Here, the route installed is for a particular amount of time and when it is not needed anymore it is deleted from the RIB. For example, an application could install a specific route in a router's RIB and advertise the associated prefix via a routing protocol. The application can then remove the route when it is no longer needed.[20]

Section 6 - Recent IETF developments on I2RS

[23]The I2RS IETF group is dedicated towards building a high-level architecture that gives details of the basic building blocks required to enable i2rs to achieve its use cases , that will further help in developing accurate data models and protocols for the interfaces.[23]

[24]The working group is keen on working on the following fields in the near future :

i) High-level architecture for I2RS which also includes the solutions for security and policies.

ii)Some important Use Cases which are under development are listed:

- Direct interactions with the routing information base(RIB) but to allow reading and writing into the RIB but no direct communication with the forwarding information base (FIB).

- Filter based RIB is used to match prefix/length in the IP packet . This way RIB can filter all the IP addresses that need to be blocked. The matches in the RIB can be sequenced and programmed to perform specific actions like to forward the packet to a particular router, etc. This work will be coordinated with various IETF work groups which involve routing, security and management.

- Implementation of policies in BGP and control and monitoring of the overall working of the protocol.

- Traffic engineering, that is, controlling and optimization of traffic flow over the network with the help of more detailed description of the network possible because of I2RS implementation.

- Route specific reaction for the network based attack so that the protocol only routes the attacked network through a new path while leaving all the policies and configurations for the rest of the network intact.

- The feature to get information from the network about the network topology. injection and creation of new topologies can be considered as a separate task that the IETF task force is working upon recently. [24]

Section 7 - Future Scope

I2RS promises to be a solution to several requirements in the field of networking. However, it is still a relatively new technology with a lots of research still going on. It is a very promising innovation and can prove to be revolutionary in terms of how routes are installed and manipulated in the network devices.

It would be quite interesting to see how I2RS is implemented into real networking devices and how the existing interfaces of the forwarding devices are modified to accommodate I2RS. Next step of the project should be to implement I2RS on a virtual or real router and to modify their policies and install routes based upon the need. Further, I2RS can be used to modify the local RIB of the virtual or real device. Lastly, the tight feedback loop offered by I2RS can be examined and compared with the feedback loop provided by the existing protocols.

I2RS can change the way networking is performed and can prove to be extremely useful in tuning the network to achieve maximum performance. However, this comes at the cost of installing compatible interfaces and fulfilling other requirements mentioned before, but, with the amount of improvement that we will get , it looks affordable.

References

[1] https://www.citrix.com/content/dam/citrix/en_us/documents/oth/sdn-101-an-introduction-to-software-defined-networking.pdf

[2] http://blog.ine.com/2011/06/15/control-plane-vs-data-plane/

[3] http://searchsdn.techtarget.com/definition/data-plane-DP

[4] http://blog.ipspace.net/2013/08/management-control-and-data-planes-in.html

[5] http://www.techradar.com/news/networking/determining-the-need-for-software-defined-networking-1253463

[6] http://searchsdn.techtarget.com/definition/control-plane-CP

[7] http://packetpushers.net/show-181-intro-to-i2rs-with-joel-halpern-russ-white/

[8] https://datatracker.ietf.org/doc/draft-clemm-i2rs-yang-network-topo/?include_text=1

[9] http://www.cohodata.com/blog/2014/01/22/data-storage-and-sdn-an-application-example/

[10] https://www.sdxcentral.com/resources/sdn/sdn-controllers/

[11] https://www.clear.rice.edu/comp529/www/papers/tutorial_4.pdf

[12] https://tools.ietf.org/html/rfc5492

[13] http://www.cisco.com/c/en/us/td/docs/wireless/wcs/4-0/configuration/guide/wcscfg40/wcsovrv.html

[14] https://www.serfdom.io/docs/recipes/event-handler-router.html

[15] http://doi.ieeecomputersociety.org/10.1109/MIC.2013.76

[16] https://tools.ietf.org/html/draft-ietf-i2rs-architecture-08

[17] http://www.noxrepo.org/pox/about-pox/

[18] http://mininet.org/

[19] http://openvswitch.org/

[20] SDN: Software Defined Networks. An Authoritative Review of Network Programmability Technologies.By Thomas D. Nadeau, Ken Gray.

[21] https://tools.ietf.org/html/draft-voit-i2rs-pub-sub-requirements-00

[22] http://www.ietf.org/archive/id/draft-white-i2rs-use-case-06.txt

[23] https://datatracker.ietf.org/meeting/92/agenda/i2rs-drafts.pdf

[24] https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf