LINK ERROR DETECTION AND FAILURE RECOVERY IN

SOFTWARE DEFINED NETWORKING

1. THILLAIRAJA S.T.S.(2012103611)

2. SUNIL N.K. (2012103602)

3. SWETHA V.S.(2012103078)

under the guidance of

**Dr. V. Mary Anita Rajam, Associate Professor**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**ANNA UNIVERSITY**

**CHENNAI - 600 025.**

**1. PROBLEM STATEMENT**

The present networking system is decentralized and the functionality is distributed. While this admits more freedom to respond to a failure event, it ultimately means that each controller application must include its own recovery logic, which makes the code more difficult to write and potentially more error-prone.Software Defined Networks makes the network centralized and execute the task in a systematic fashion. It has two main parts. ie., Data plane and Control plane. The whole network is monitored using SDN Controller which is considered to be the brain of the network. Any modification in the network such as updations and deletions are only done via the Controller. Our project is mainly focused on link error detection and recovery. A network is can be said reliable if there is no packet drop and its communication is done without any delay. Both of these are affected if a link is failed and so the links in the network must be kept resilient.This project deals with an unique mechanism to overcome the link failures. Our project ensures that the packet is rerouted properly to the destination node even if the link fails and the restoration policy of the state is taken into consideration. This will make the network reliable and this is done in a virtualized environment called mininet in order to provide with better results.

**2. INTRODUCTION**

The aim of SDN is to provide open interfaces enabling development of software that can control the connectivity provided by a set of network resources and the flow of network traffic though them, along with possible inspection and modification of traffic that may be performed in the network.

With SDN, the **applications can be network aware**, as opposed to traditional networks where the network is application aware (or rather, application ambivalent):

Traditional (i.e. non-SDN) applications only implicitly and indirectly describe their network requirements, typically involving several human processing steps, e.g., to negotiate if there are sufficient resources and policy controls to support the application.

**Traditional networks** (e.g. the current Internet and its services like web browsing, media streaming) do not offer a (dynamic) way to express the full range of user requirements, for example throughput, delay, delay variation or availability. Packet headers can encode priority requests, but network providers typically do not trust user traffic markings. Therefore some networks try to infer the users requirements on their own (e.g. through traffic analysis), which may incur additional cost and sometimes leads to misclassification. SDN offers the ability for a user to fully specify its needs in the context of a trusted relationship that can be monetized.

Traditional (i.e. non-SDN) networks do not expose information and network state to the applications using them. Using an SDN approach, SDN Applications can monitor network state and adapt accordingly.

The **control plane is (1) logically centralized** and **(2) decoupled from the data plane**. The SDN Controller summarizes the network state for applications and translates application requirements to low-level rules.This does not imply that the controller is physically centralized.

For performance, scalability, and/or reliability reasons, the logically centralized **SDN Controller** can be distributed so that several physical controller instances cooperate to control the network and serve the applications. Control decisions are made on an up-to-date global view of the network state, rather than distributed in isolated behavior at each network hop. With SDN, the control plane acts as a single, logically centralized network operating system in terms of both scheduling and resolving resource conflicts, as well as abstracting away low-level device details, e.g., electrical vs. optical transmission.

**The SDN Controller has complete control of the SDN Datapaths**, subject to the limit of their capabilities, and thus does not have to compete/contend with other control plane elements, which simplifies scheduling and resource allocation. This allows networks to run with complex and precise policies with greater network resource utilization and quality of service guarantees. This occurs through a well-understood common information model (e.g. as the one defined by OpenFlow).

It is a logically centralized entity in charge of (i) translating the requirements from the SDN Application layer down to the SDN Datapaths and (ii) providing the SDN Applications with an abstract view of the network (which may include statistics and events). An SDN Controller consists of one or more NBI Agents, the SDN Control Logic, and the Control to Data-Plane Interface (CDPI) driver. Definition as a logically centralized entity neither prescribes nor precludes implementation details such as the federation of multiple controllers, the hierarchical connection of controllers, communication interfaces between controllers, nor virtualization or slicing of network resources.

In our system, the process of detecting the link failure in a network and providing a secure path for the packets. Mostly, the alternate path provided is pre-computed already and is stored in the table.This is a monotonous case , as the networks in existence today are solely dependent on this scenario.The existing system uses a working and a transient plane.Working plane is used till the link failure occurs.Transient plane comes into play when a link is failed. The transient plane is like a tree in which the failure node is the root.

**3. RELATED WORKS**

3.1 Open Flow Path Protection in Software-Defined Networks:

**3.1.1 OpenFlow-Based Segment Protection in Ethernet Networks,**

**IEEE/OSA Journal of Vol 5 ,No 1, September 2013**

Carrier-grade Ethernet networks, industrial area networks and some local area networks (LANs) have to provide a resilient spring back in case of a network failure. Open Flow Architecture is generally based on Ethernet cables. Thus link failure in these networks are common. Thus in OpenFlow-Based Segment Protection in Ethernet Network, the Open Flow architecture is enhanced to support segment based rerouting algorithm for efficient transfer of messages if there are link failures. This mechanism is efficient in choosing the backup path (secondary path) in case of an intermediate link failure. The choosen (secondary) path is calculated within few tens of milliseconds. The above method takes in loads of memory space and is also considered to be inefficient when it comes to memory handling. Also, the time taken to recover is approximately 1ms and that is quite high.

**3.1.2 Fault-Tolerant OpenFlow-based Software Switch Architecture with LINC Switches for a Reliable Network Data Exchange**,**Research and Educational Experiment Workshop (GREE), 2014 Third GENI 19-20 March 2014**

Moreover, switches are essential for forwarding the packets in a local area network and if a switch fails, then the packets are not able to reach their destination. The new trend in Software Defined Networking (SDN) has made the use of software switches quite popular. These Software switches are required to be resilient to failure. This Fault-Tolerant OpenFlow-based Software Switch Architecture with LINC Switches for a Reliable Network Data Exchange explains one mechanism for fault tolerance of LINC (Link Is Not Closed), an open source OpenFlow switch, which is written in Erlang programming language. We leverage the built-in concurrency, and fault-tolerant features of Erlang to realize a fault-tolerant distributed LINC switch system·The LINC switches took only a little more than half the time the hardware switches took to connect hosts to the fault-tolerant system. When failover happens, the controller modifies the flow entries in the LINC switches which causes the packets to be sent to the new switch’s input port almost instantaneously. Erlang system shows some ease of programmability and faster deployment. The future Work is that by making efficient use of the Erlang Distributed System, the Fault Tolerant System can be improved further.

3.2 Scalable Resilience for Software-Defined Networking :

**3.2.1 Using Loop-Free Alternates with Loop Detection, Network Softwarization (NetSoft), 2015 1st IEEE Conference**

It provides alternatives for both the scalability and resilience issues in OpenFlow Networks. OpenFlow switches store their flow tables in expensive, limited Ternary Content-Addressable Memory (TCAM) due to which the stored tables cannot be large, thus forwarding packets in line speed. Most resilience mechanisms require additional entries thus the implementation in OpenFlow may quickly exceed the available TCAM. Loop-Free Alternates (LFAs) are a standardized mechanism for fast reroute in IP networks which do not require additional entries. However, some LFAs cause loops when some node or multiple nodes failures occur. This renders additional links unusable. But if we were to exclude such LFAs, it would reduce the protected destination coverage even further. To overcome this, a scheme is designed to detect the loops caused by LFAs. This maximizes the protection coverage because the LFAs can be used without creating loops. This paper describes how LFAs and the loop detection scheme can be implemented in OpenFlow networks with only little packet overhead and a single additional entry per switch. They are simple and have no additional forwarding entries. And had no flow tables. The ternary content-addressable memory was used to store the flow tables. The TCAM is very expensive. Hence, here, its cost is also reduced thus giving maximized protection.Some problems not dealt here are that they mostly cannot protect all traffic and some of them cause micro-loops in case of node failures or multiple failures. Loop detection just helps to prevent loops for LFAs, but it cannot protect traffic for which no LFAs exist.

**3.2.2** **Using** **Orion:-A Hybrid Hierarchical Control Plane of Software-Defined Networking for Large-Scale Networks,** **2014 IEEE 22nd International Conference on Network Protocols.**

The three layers of the Orion model provide the hybrid hierarchical control layer for large scale networks. The Orion has the area controller which is responsible for collecting physical device information and link information, managing the intra-area topology and processing intra-area routing requests and updates. The domain controller synchronizes the global abstracted network view through a distributed protocol. The Domain Routing Management Sub-Module of the domain controller computes the global shortest path. When a PacketIn message reaches the area controller, the area routing management sub-module checks the source address and destination address of the message. If the destination address is in the area, the area controller employs Dijkstra algorithm to compute intra-area path and if is not present in the area, the area controller sends the source address and the destination address of the message to the domain controller, and stores the message to a waiting buffer with index. The domain controller computes the routing path for the flow and sends the routing result to the area controller· The CPU utilization of the domain controller is 40%. The domain controller of Orion costs lower than an OpenFlow based SDN controller and the load of the area controller of Orion is between the load of the controller in the flat architecture and the lowest area controller in the abstracted hierarchical SDN architecture. One disadvantage is the memory overhead due to the maintenance of area and domain controllers.

3.3 Openflow Path Failures and Re-routing:

**3.3.1 Europe-wide demonstration of fast network resto- ration with OpenFlow ,** **IEICE Commun. Express, vol. 3, no. 9, pp. 275–280, 2014.**

During the data transmission, there may occur many number of the data failures in the data path. Segment protection is the key feature used to reroute the data in the secondary route which may or may not be the best or optimal path. Independent Transient Plane (IPL) is designed in Europe-wide demonstration of fast network resto- ration with OpenFlow which reduces the path complexity and maintains the security of the data. This work results in the most efficient protection in the secondary path and optimal solution to the data failure problem. One advantage here is the designed mechanism deals with the data packets in all paths si multaneously. The data packets which are lost during the transmission in the malicious path are gained by this designed. In some cases those data are sent to the destination in the legitimate path in retransmission phase but the regained data packets are not considered as legitimate; therefore there is a possibility of malicious data present in that regained data.

**3.3.2 Fast Recovery in Software-Defined Networks**, **IEEE Software Defined Networks (EWSDN), 2014 Third European Workshop on 1-3 Sept. 2014**

Fast Recovery in Software-Defined Networksimplements a failover scheme with per-link Bidirectional Forwarding Detection sessions and preconfigured primary and secondary paths computed by an OpenFlow controller It uses the usual way of detecting failures in Ethernet networks like if an acknowledgement is not received within 50-150ms , then the link is said to be broken. Uses two steps of process –The first step involve a fast switch-initiated recovery based on preconfigured forwarding rules guaranteeing end-to-end connectivity. The second step involves the controller calculating and configuring new optimal paths.A lower detection time due to decreased session round trip time (RTT) with the removal of false positives. As each session spans a single link, false positives due to network congestion can be easily removed by prioritizing the small stream of control packets. But the problem is that the memory is not efficiently handled since the flow table should store two ways of possible communication and Redundant routing information is stored in the group tables.

**3.3.3 Detour Planning for Fast and Reliable Failure Recovery in SDN with OpenState,Design of Reliable Communication Networks (DRCN), 2015 11th International Conference on 24-27 March 2015**

Detour Planning for Fast and Reliable Failure Recovery in SDN with OpenState provides a secure and reliable path if a particular node or a link fails. A protection scheme is given in this paper which calculates the backup paths in prior and the routing used here is MPLS which ensures zero packet loss after the link failure is detected. Also in this paper, the forwarding rules are done autonomously without the use of the controller always. The recovery is done with the help of tagged indexes making the routing simpler. The centralized controller is not frequently accessed since few forwarding rules are done independently and no packets are lost. However this mechanism cannot be expanded to multiple link failures. Also if the tag indices are lost, then the packet drop numbers will increase.

**3.3.4** **Openflow Path Fast Failover Fast Convergence Mechanism,** **Network Research Proceedings of the Asia-Pacific Advanced Network 2014 v. 38, p. 23-28**

Openflow Path Fast Failover Fast Convergence Mechanism deals with a fast and efficient failover mechanism for redirecting traffic to more optimal backup path when there is a link failure or congestion problem in SDN. It also proposes a local pre-calculated path dataset mechanism in OpenFlow controller to allow fast network convergence. The central OpenFlow Controller computes the main and the best backup path based on the current network topology. OpenFlow controller is said to have a local dataset of path information and in case of a link failure or congestion in a path, the affected switch sends port-status message to the controller and the controller checks the flow entries affected by the failure. The controller pushes the main and the backup path to the OpenFlow switches and will recalculate the less congested backup path after it is updated periodically by the network. Once the controller get the notification about a link failure, it will perform simple lookup in its local dataset to find whole flow entry that affected by the failure. Finally, the affected entries will be deleted from the flow table and the pre-computed less congestion backup path will be selected. The controller then updates the flow entries of all switches and incorporates the new backup plan. The single backup path for every main path in only one single switch flow table reduces the possibilities of flow table explosion and the network traffic is redirected to alternate optimal path. But one drawback is the memory overhead due to the recalculation of less congested path in the controller.

**3.3.5 Efficient routing for traffic offloading in Software-defined Network,** **International Workshop on Software Defined Networks for a New Generation of Applications and Services (SDN-NGAS-2014)**

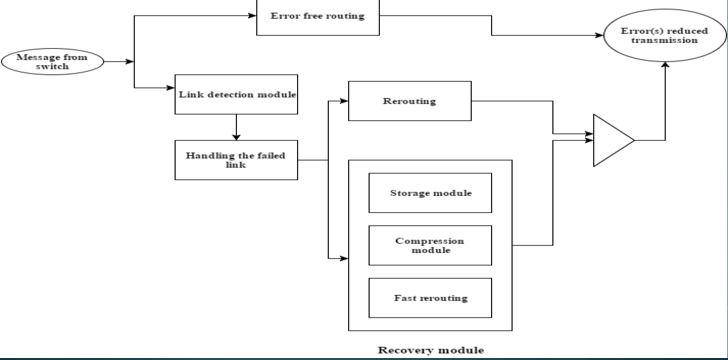
One efficient re-routing is the Automatic Re-routing with Loss Detection architecture (ARLD) proposed in Efficient routing for traffic offloading in Software-defined Network paper works on the assumption that if a packet loss occurs at a link , it is mainly due to congestion. The controller treats this link as a bottleneck link. The Openflow protocol has a stats message which delineates the status of each node and each port to the SDN controller. The Re-routing module of the SDN controller computes an alternate path as a bypass route. The Re-routing module updates the virtual topology by eliminating the node at which the packet loss occurred and finds an alternative route without the switch that dropped the packet. After examining the availability of the path, the module returns the alternate route to controller and controller distributes new forwarding rules to each switch and updates the flow table. In this way the packet loss at the switches due to congestion can be reduced to yield better performance. With the help of an Openflow based SDN architecture , the controller detects the packet loss at switches in a shorter time and the model reduces the packet loss by providing a better performance. However when dropped packet is routed to bypass route the traffic on that route may cause congestion on that link. This effect would be a serious problem in larger sized network. Also, In the proposed architecture, the average loss rate is reduced and this might decrease average latency of each flow, although measuring delay is not conducted due to the low timing realism of mininet emulation.

**3.4 IP Fast Reroute for Single and Correlated Failures with rMRC:**

**3.4.1 Relaxed Multiple Routing Configurations:IP Fast Reroute for Single and Correlated Failures , IEEE Transactions on network and service management, vol 6, no. 1, MARCH 2009.**

IP Fast Rerouting here provides us the methodology employed to recover from link failures in a network with the help of Relaxed Multiple Routing Configuration (rMRC). The MRC and the rMRC guarantee link or node failure from biconnected topologies. Backup topologies can be constructed using different methods and the number of states required in a router will increase with multiple backup paths. In the rMRC the requirements of the network topologies are relaxed. The difference between rMRC and MRC is that in conventional MRC the instantaneous recovery from the node failure is done by isolating the affected nodes. Instead, rMRC computes the shortest path without the failed link in the backup topology where the detecting node itself is isolated. Using the backup topology where the detecting node is isolated ensures that the traffic cannot loop back to the detecting node but still enables the rMRC forwarding to reach the destination node. The presented algorithm can guarantee link and node fault tolerance with fewer backup topologies than MRC. As relaxed backup topologies do not isolate all links, there is more flexibility in rMRC than in MRC to decrease the number of backup topologies. However the problem here is rMRC’s ability to spread traffic over more links can sometimes have a dramatic impact in a sparsely connected network topology.

**4. HIGH LEVEL BLOCK DIAGRAM**

****

**5. DETAILED MODULE DESIGN INDICATING THE INTERMEDIATE DELIVERABLE(S)**

**Detection Module :**

**Detection of link failure**

* Link between nodes is considered to be the framework for any network and so

preserving the link is cardinal.

* If the network link is error free ,it reroutes and finally transmits the packets without
* any loss.
* If a link failure is detected, the nodes between the failed link is identified. After the detection of link failure, input is then passed to the recovery module,and the
* open flow protocol intimates the SDN controller.

**PSEUDO CODE:**

1. for each switch in a network
   * 1. sending & receiving beacon signals to nearby switches
     2. IF  (link failure detected)
        + 1. identify the failed link between switches
          2. link array[] = store the failed link
          3. initialize link flag bit =1
          4. IF (packet loss > buffer limit in switches)

stop the packet transmission from the source.

pass the packet flow to  recovery module

modify the routing table and initialize the flag bit =1.

intimate the SDN Controller.

* + 1. ELSE
       - 1. re routing the packets without loss.

**Recovery Module :**

* Storing of dropped packets in compression state.
* Handling the failed link using the alternate back path (the packets which are not left
* from the source).
* Stored packets are sent using fast rerouting technique.
* Packets that are not left from the source are rerouted in pre-calculated alternate
* path.
* Packets that are transmitted before the detected failed link are stored in the
* switch in the compressed state . The stored packets are then decompressed and fast rerouted to the destined node.
* SDN controller controls all the rerouting paths using Dijkstra’s algorithm.

**PSEUDO CODE:**

IF (link flag bit is set)

* + - 1. Packets which haven’t started from the source are need to be stopped.
      2. Re route the packets from the source in alternated precalculated path.

IF (flag bit is set)

Packets which had started from source are made to stored in a switch in a compressed state before the failed link which is stored in link array[].

Stored switch intimates SDN Controller using OpenFlow protocol.

SDN Controller decompresses the packets

Finding the next alternate shortest path is done by using Shortest Path Algorithm.

**PATH CALCULATION MODULE:**

The module helps in calculating the optimal path between two hosts. The path is calculated with the help of shortest path algorithm and ports calculation algorithm. A test case that is implemented in this module is Debugger function. Inputs to the module are two variables, src and dest. The source host is stored in the variable src and the destination host is stored in the variable dest.

**THE SHORTEST PATH ALGORITHM:**

The variable graph contains the network topology defined by the network administrator. The variable src is stored in the variable start and the variable dest is stored in the variable end. The output is the shortest path between the start and the end nodes. The output is stored in the variable tmp-path.

def spath(graph,start,end,path=[]):

temp-path=[start]

add tmp-path to queue

while q is not empty

if last-node == end

for link node in graph[last -node]

dequeue q to tmp-path

last-node=tmp-path[len(tmp-path)-1]

print tmp-path

tm-path is a valid path

if link node not in tmp-path

newpath = tmp-path+ link-nod

q.enqueue(new path)

return tmp-path

**THE PORTS CALCULATION ALGORITHM:**

The algorithm helps in calculating the in-port and out-port of the switches along the path between the source and the destination.

The variable path is the path calculated by shortest path algorithm. l1[]and l2[] are the list of hosts and switches having connection between eachother. l3[] is the list of port number through which l1[i] (each node) is connected to l2[i] (each node )

ports=[]

for j in range(0,len(path)):

for i in range(0,len(l1)):

if l1[i]==path[j]:

if l2[i]==path[j+1]

ports.append(l3[i])

return ports

**6. IMPLEMENTATION DETAILS (20%) AND THE CORRESPONDING RESULTS/SNAPSHOTS**

1.Open the Terminal application on your computer (Windows users will use PuTTY).

In the Terminal window, use the SSH client on your computer to start an SSH session to the Mininet VM.

**ssh -Y mininet@192.168.56.101**

In the Terminal window, which is now running an SSH session connected to the Mininet virtual machine, start an Xterm:

**xterm -sb &**

2.The MiniEdit script is located in Mininet’s examples folder. To run MiniEdit, execute the command:

**$ sudo ~/mininet/examples/miniedit.py**

MiniEdit has a simple user interface that presents a canvas with a row of tool icons on the left side of the window, and a menu bar along the top of the window.

3.Add eight switches and three controllers using the same method: Click on the Switch tool and add switches, then click on the Controller tool and add controllers.

4.To start the simulation scenario, click the Run button on the MiniEdit GUI. In the terminal window from which you started MiniEdit,

you will see some messages showing the progress of the simulation startup and then the Miniedit CLI prompt

(because we checked Start CLI box in the MiniEdit preferences window).

5.First, check the switch configurations in the network simulation to verify that everything is set up correctly.

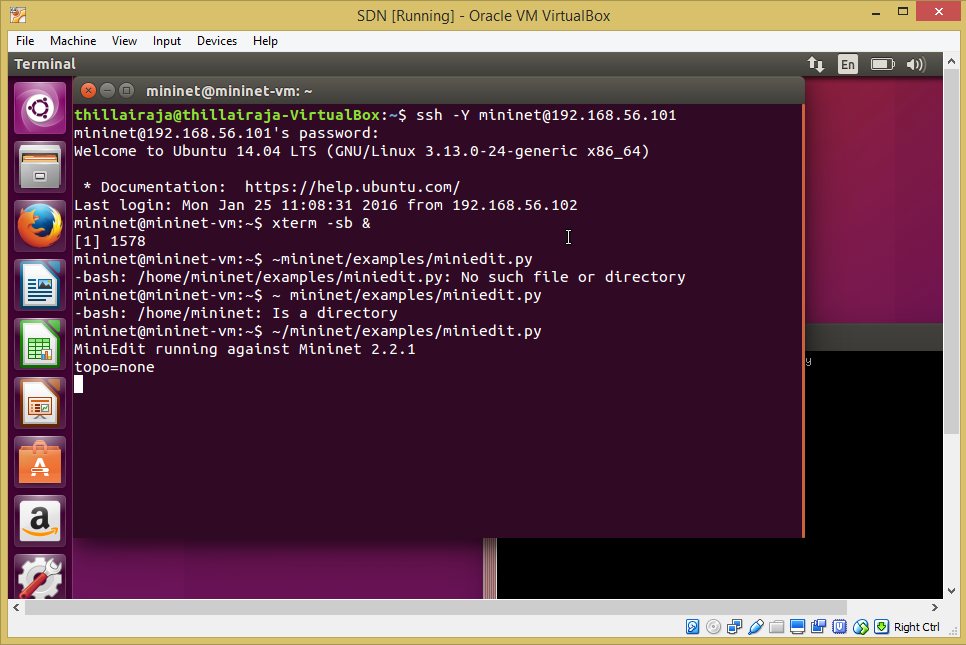
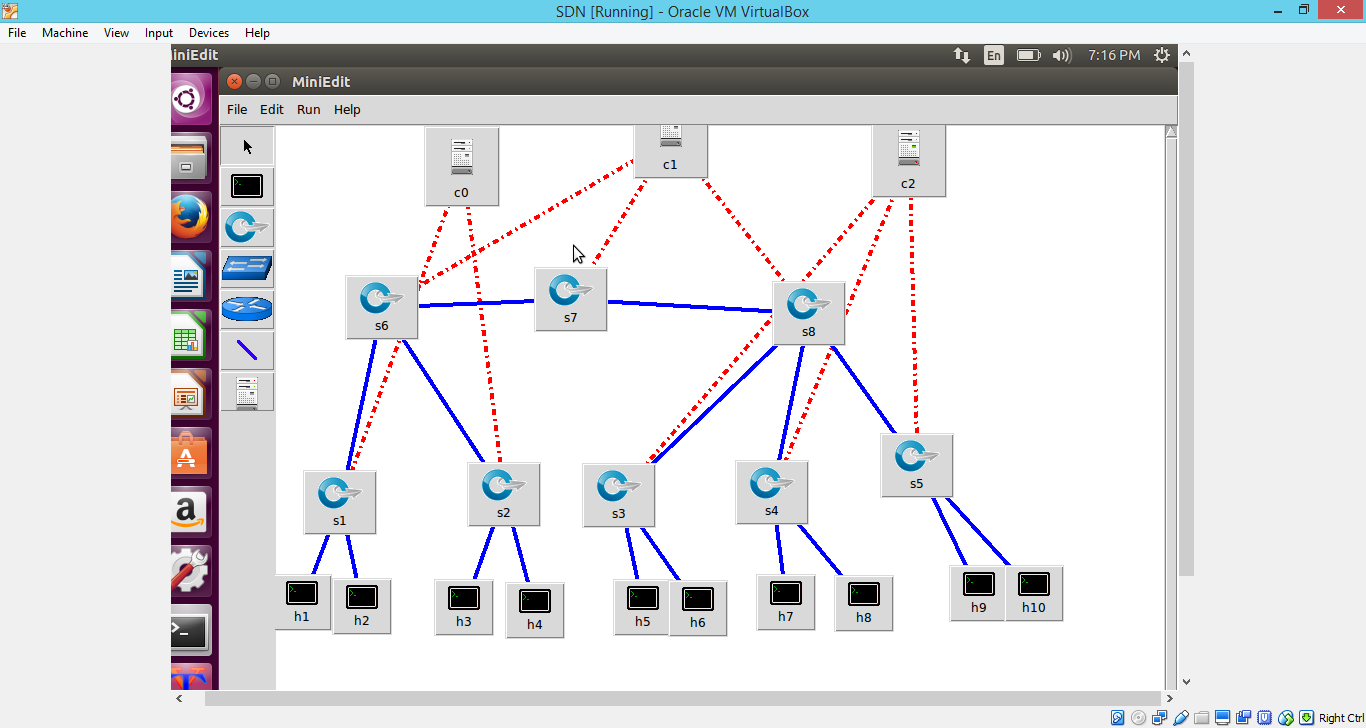
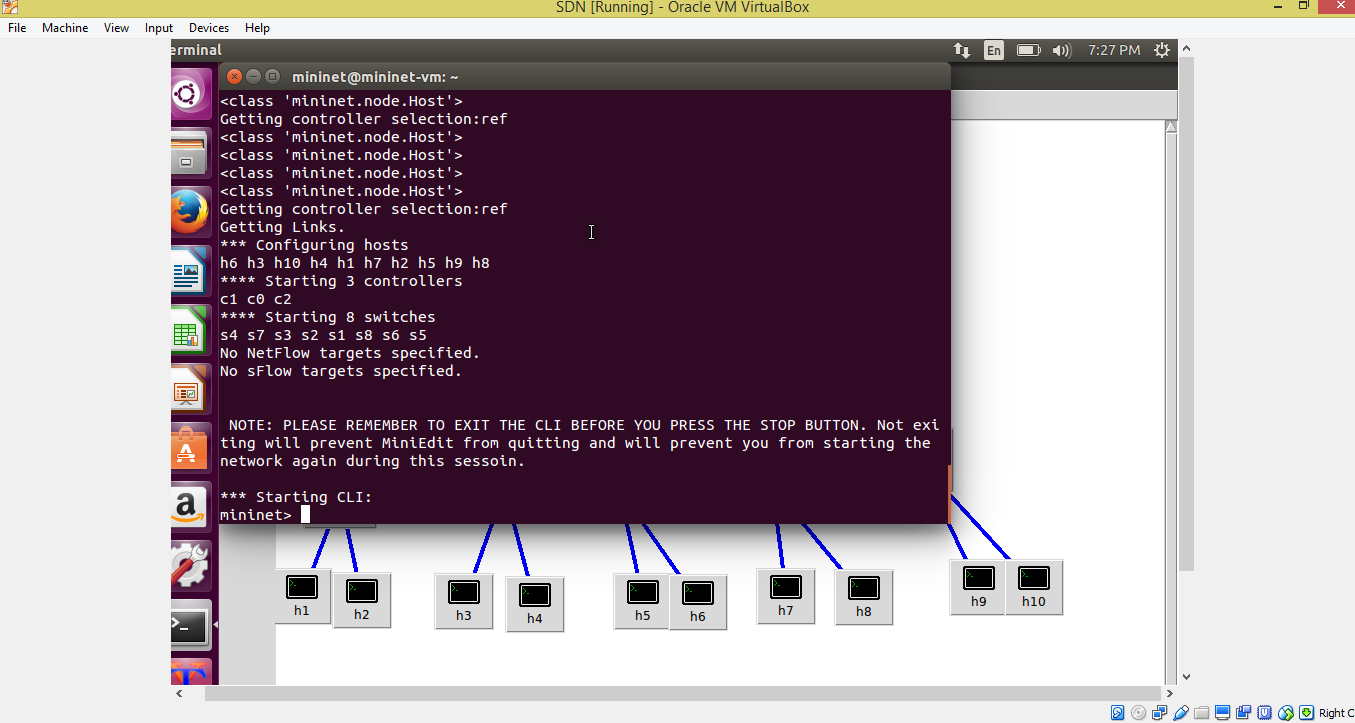
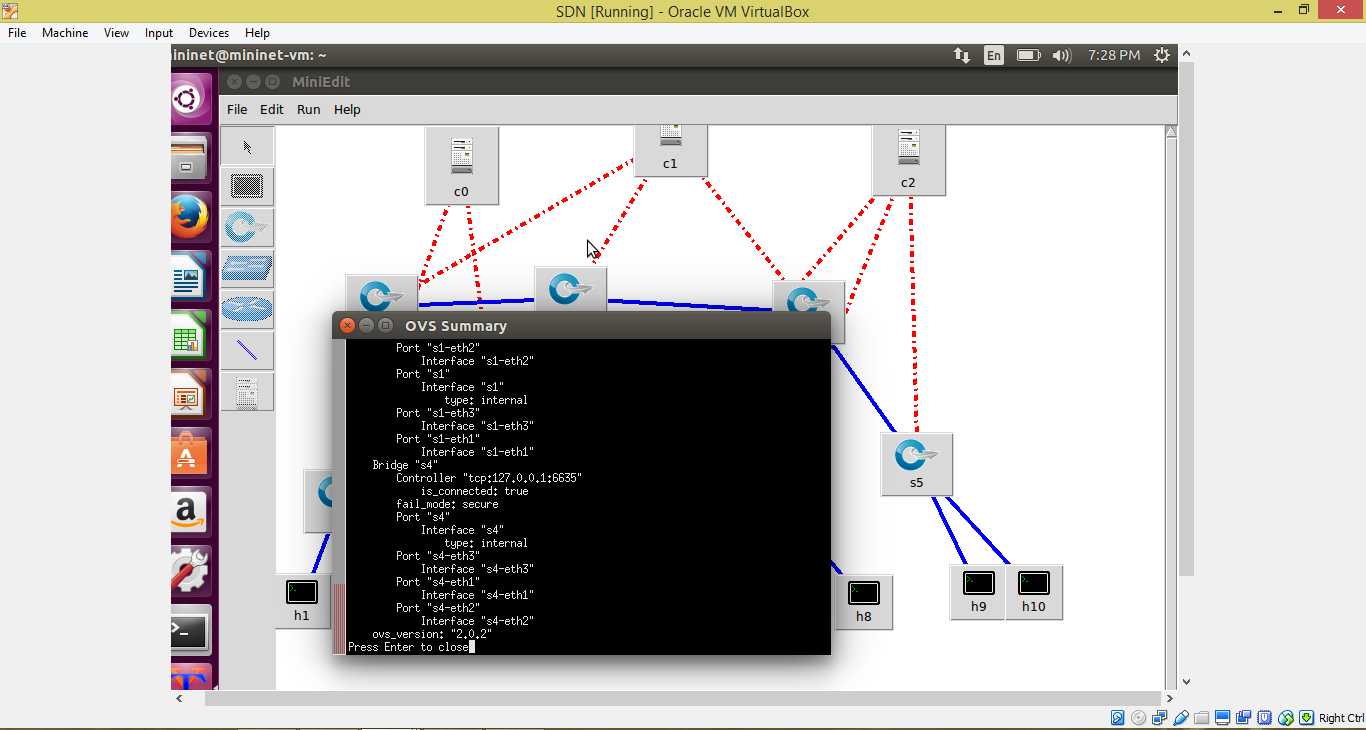
You can run the MiniEdit menu command, Run . Show OVS Summary to see an listing of switch configurations.

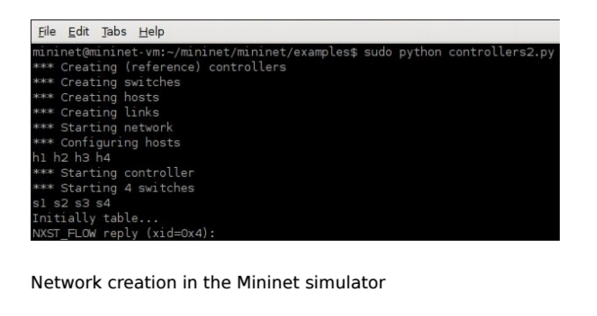
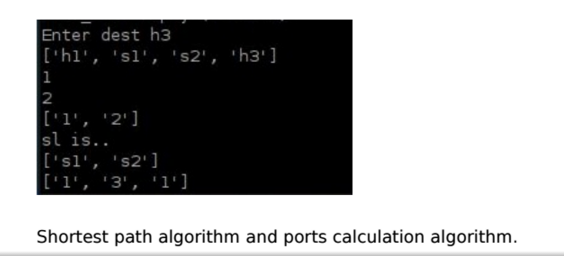
In this case, we can verify that each switch is listening to the correct controller on the correct port.

6.Then, run a ping command to send traffic between host h1 to h8.

**mininet > pingall**

In the MiniEdit console, you see the results of the ping command



****

**7. METRICS FOR EVALUATION:**

**Buffer space overhead**

A threshold, which specifies the maximum number of packets in each switch, is determined when the network is created. For each request, before the switch is loaded up to the buffer limit, the packets already present is compared against the threshold. If it exceeds the threshold,the packets are rerouted through the shortest path.

A threshold, which specifies the maximum number of packets in each switch, is determined when the network is created. For each request, before the switch is loaded up to the buffer limit, the packets already present is compared against the threshold. If it exceeds the threshold,the packets are rerouted through the shortest path.

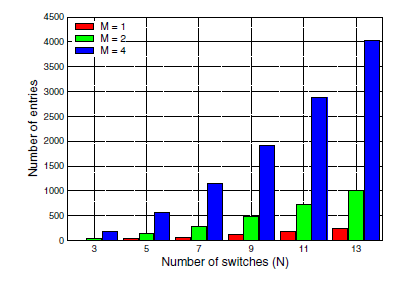
A threshold, which specifies the maximum number of packets in each switch, is determined when the network is created. For each request, before the switch is loaded up to the buffer limit, the packets already present is compared against the threshold. If it exceeds the threshold,the packets are rerouted through the shortest path.

A threshold, which specifies the maximum number of packets in each switch, is determined when the network is created. For each request, before the switch is loaded up to the buffer limit, the packets already present is compared against the threshold. If it exceeds the threshold,the packets are rerouted through the shortest path.

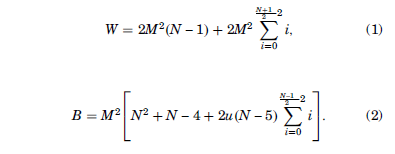
A threshold, which specifies the maximum number of packets in each switch, is determined when the network is created. For each request, before the switch is loaded up to the buffer limit, the packets already present is compared against the threshold. If it exceeds the threshold,the packets are rerouted through the shortest path.

A threshold, which specifies the maximum number of packets in each switch, is determined when the network is created. For each request, before the switch is loaded up to the buffer limit, the packets already present is compared against the threshold. If it exceeds the threshold,the packets are rerouted through the shortest path.

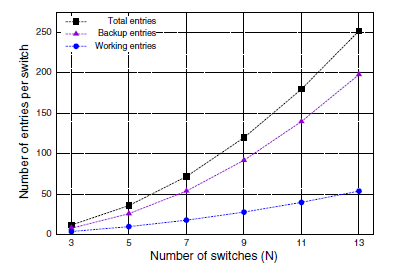
A threshold, which specifies the maximum number of packets in each switch, is determined when the network is created. For each request, before the switch is loaded up to the buffer limit, the packets already present is compared against the threshold. If it exceeds the threshold,the packets are rerouted through the shortest path.

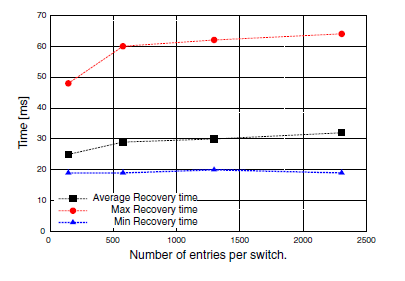


N values higher than 13 are not supported by the server running Mininet. Equations (1) and (2), Respectively detail the amount of working entries W and backup entries B in each switch of a ring network as a function of N and M, in the any-to-any communication worst case:









**8. REFERENCES**

1. Nattapong Kitsuwan , Seamos McGettrick , Frank Slyne , David B. Payne , Marco Ruffini , “Independent Transient Plane Design for Protection in openFlow-Based Networks”, IEEE/OSA Journal, Vol 7 , March 2015.
2. Niels L. M. van Adrichem, Benjamin J. van Asten and Fernando A. Kuipers, “Fast Recovery in Software-Defined Networks” in IEEE Software Defined Networks (EWSDN), 2014 Third European Workshop on 1-3 Sept. 2014
3. Yonghong Fu, Jun Bi, Kai Gao, Ze Chen, Jianping Wu and Bin Hao, “Orion: A Hybrid Hierarchical Control Plane of Software-Defined Networking for Large-Scale Networks” , 2014 IEEE 22nd International Conference on Network Protocols.
4. Nor Masri Sahri and Koji Okamura, “Openflow Path Fast Failover Fast Convergence Mechanism” , Network Research Workshop Proceedings of the Asia-Pacific Advanced Network 2014 v. 38, p. 23-28
5. Tarik ˇ Ciˇci´c, Audun Fosselie Hansen, Amund Kvalbein, Matthias Hartmann, R¨udiger Martin, Michael Menth,Stein Gjessing, and Olav Lysne, ”Relaxed Multiple Routing Configurations: IP Fast Reroute for Single and Correlated Failures” , IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, VOL. 6, NO. 1, MARCH 2009.