Open access • Proceedings Article • DOI:10.1109/DRCN.2011.6076899

Enabling fast failure recovery in OpenFlow networks — Source link

Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet ...+1 more authors

Institutions: Ghent University

Published on: 19 Dec 2011 - Design of Reliable Communication Networks

Topics: OpenFlow, Carrier grade, Forwarding information base, Router and Switchover

Related papers:

- OpenFlow: enabling innovation in campus networks
- OpenFlow-based segment protection in Ethernet networks
- OpenFlow: Meeting carrier-grade recovery requirements
- Scalable fault management for OpenFlow
- Software defined networking: Meeting carrier grade requirements

Enabling Fast Failure Recovery in OpenFlow Networks

Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet and Piet Demeester
Ghent University - IBBT, Department of Information Technology (INTEC),
Gaston Crommenlaan 8, bus 201, 9050 Ghent, Belgium
e-mail: {firstname.lastname}@intec.ugent.be

***Abstract*—OpenFlow is a novel technology designed at Stanford University which aims at decoupling the controller software from the forwarding hardware of a router or switch. The OpenFlow concept is based on the approach that the forwarding information base (FIB) of a switch can be programmed via a controller which resides at a separate hardware. The goal is to provide a standardized open management interface to the forwarding hardware of a router or switch. The aim of a project SPARC "SPlit ARchitecture Carrier grade networks" is to deploy OpenFlow in carrier grade networks. Reliability is a major issue to deploy OpenFlow in this networks. This work proposes the addition of a fast restoration mechanism in OpenFlow and evaluates the performance by comparing the switchover time and packet loss to existing restoration options in a current OpenFlow implementation.**

Keywords: OpenFlow, Restoration, Protection, Carrier Grade Networks

I. Introduction

The aim of the OpenFlow architecture [**?**] is to provide a standardized open management interface to the forwarding hardware of a router or switch, particularly to test an experimental protocol in the network we use every day. This is based on the fact that most modern routers/switches contain FIB (Forwarding Information Base) and FIB is implemented using TCAMs (Ternary Content Addressable Memory). The OpenFlow provides a protocol to program this FIB via "adding/deleting" entries in a FlowTable. The FlowTable is an abstraction of a FIB. In OpenFlow networks, all the logic is performed on a centralized system, called the OpenFlow controller which manages the OpenFlow switches using the OpenFlow protocol (Fig. 1). Thus an OpenFlow switch consists of a FlowTable; which performs packet lookup and forwarding, and a secure channel to an external controller.

A Flow Entry in the FlowTable consists of (1) a "packet header" that defines the flow, (2) "action" which defines how the packet should be processed, and (3) "statistics" which keep track of the number of packets; bytes for each flow; and the time since the last packet matched per flow. The controller installs these Flow Entries in FlowTables of OpenFlow switches. Incoming packets processed by the OpenFlow switches are compared against the FlowTable. If a matching Flow Entry is found, actions for that entry are performed on the packet. If no match found, the packet is forwarded to the controller over the secure channel. The controller is responsible to determine how the packet should be handled; either by adding no Flow Entries or by adding valid Flow Entries in the OpenFlow switches.

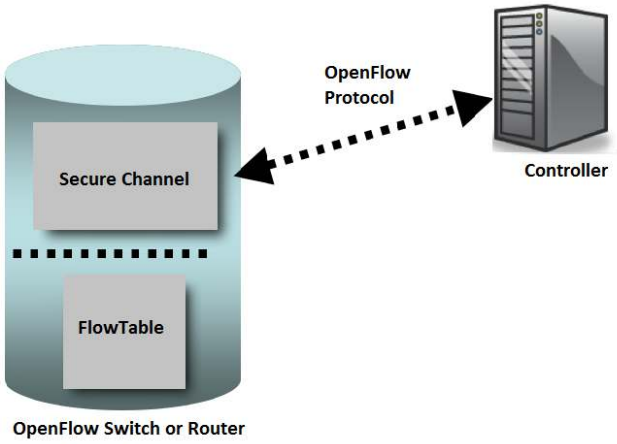

Fig. 1. OpenFlow Architecture

The secure channel between the OpenFlow switch and controller is a transport layer security (TLS) channel. Thus, packets between OpenFlow switches and controller contain OpenFlow header above transport layer header.

There are two different OpenFlow enabled software switches which can be installed in our testbeds. One is Stanford's software reference design, and the other is Open vSwitch (http://openvswitch.org/) implementation. We use Open vSwitch for our emulation as it is a production quality, multi-layer virtual switch which is designed to enable massive network automation.

Many attempts are already been carried out to make an OpenFlow controller. These are NOX [**?**], Beacon [**?**], Onix [**?**], Helios [**?**] and Maestro [**?**]. This work uses the NOX as an OpenFlow controller. This is because it is Open-Source, widely used and provides a simplified platform for writing network control software in C++ or Python.

SPARC (SPlit ARchitecture Carrier grade networks) [**?**]

is a project, aimed at the investigation and implementation of a new split in the architecture of the future internet and its building blocks. Unique for the SPARC project is the implementation of scalable carrier class functionalities based on the OpenFlow concepts. So the aim of the SPARC is to deploy OpenFlow in carrier grade networks. The term "carrier grade" [**?**] describes a set of functionalities and requirements that architecture should support in order to fulfill the operational part of network operators. The requirements are (1) Scalability (2) Reliability (3) Quality of Service (QoS) and (4) Service Management. Carrier grade networks should be able to detect and recover from incidents without impacting users. Hence a requirement is added in the carrier grade network so that it should recover from failure within 50 ms sub interval [**?**]. Keeping this requirement in mind, this work is carried out to enable fast recovery in OpenFlow networks.

Resilience mechanisms [**?**] to be used in carrier grade networks to handle recovery can be divided into Restoration and Protection. In case of protection, the paths are preplanned and reserved before a failure occurs. Hence when a failure occurs, no additional signaling is needed to establish the protection path. However, in case of restoration, the recovery paths can be either preplanned or dynamically allocated, but resources are not reserved until failure occurs. Thus, when a failure occurs additional signaling is needed to establish the restoration path. Protection is a proactive strategy while restoration is a reactive strategy.

In this paper, we compare restoration options available in OpenFlow networks. Furthermore, this work proposes the addition of a fast restoration mechanism for OpenFlow networks and evaluates its performance by comparing the switchover times and packet loss to the existing restoration options in a current OpenFlow implementation.

The rest of paper is organized as follows: Section II presents current mechanisms available in OpenFlow networks for handling recovery. It also discusses limitations of existing mechanisms in handling failures. Section III describes our fast restoration option that can be enabled in OpenFlow networks. Section IV gives the emulation environment. Section V presents analysis of results and finally section VI concludes.

II. Current Mechanisms in handling Failures in OpenFlow Networks

This section discusses existing mechanisms implemented at OpenFlow and NOX software to recover from a link failure. It also discusses limitations of these mechanisms in enabling fast recovery in OpenFlow networks. The OpenFlow protocol [**?**] defines the control messages which are exchanged between OpenFlow switches and controller. Two messages are important for this work (1) "packet in" and (2) "packet out" message. When the packet is received by the datapath and needs to send this to the controller, it is sent via "packet in" messages. On the other hand, when controller needs to send the packet out through datapath of OpenFlow switch, it sends "packet out" message.

A. OpenFlow Mechanisms for handling failures

OpenFlow follows an on-demand approach. Flow Entries are not added proactively in switches. When a data packet arrives at an OpenFlow switch and it does not match any Flow Entry, it requests the Flow Entry from the controller via sending "packet in" message. However, if the Flow Entry is present then OpenFlow switches directly forward the data packet according to the Flow Entry, without contacting the controller.

The failure can be recovered if new correct Flow Entry is added in OpenFlow switches after the failure occurs. We can say that recovery from the failure depends on the time when the OpenFlow switch again requests the Flow Entry from the controller. Thus, recovery from failure depends on the life of flow-entries in FlowTable.

The life of Flow Entries is associated with two intervals called "idle timeout" and "hard timeout" [**?**]. Idle timeout indicates the time when the Flow Entries should be removed due to lack of activity. It is the time interval of a Flow Entry with which the OpenFlow switch has not received the packet of a particular flow of that entry. Hard timeout indicates the maximum life of Flow Entries, regardless of activity. OpenFlow switches remove their Flow Entries from the FlowTables after the expiration of one or both the intervals.

The more the value of these intervals, the more time the OpenFlow switches will take to recover from failure. The lesser the value, more packets will be sent to controller to refresh the Flow Entries.

B. NOX mechanisms for detection of failures

The recovery from failures is possible with a new Flow Entry only if the controller also knows about the failure. Otherwise, the controller may add an incorrect entry in the OpenFlow switches. Thus, recovery depends not only on hard and idle timeout but it also depends on mechanisms running in the network to detect the failure. This section describes the mechanisms available to NOX for the detection of link failure.

NOX implements L2-Learning and routing mechanisms to recover from a failure. L2-Learning is implemented in C++ and Python. The former is called L2-Learning Switch and the latter is called L2-Learning Pyswitch. These behave differently to recover from a failure.

1) L2-Learning Switch: L2 Learning switch operates by maintaining a mapping between the MAC (Media Access Control) addresses and the physical ports of OpenFlow switches by which they can be reached. These mappings are learned by monitoring the source addresses of incoming

packets. It updates or adds the source MAC address and incoming port in its MAC table once it receives the packet.

Besides this, it matches the destination address (DA) of the packet with the addresses available in the MAC table. If the address matches then it adds the Flow Entry in the FlowTable of the OpenFlow switch so that the packet can be forwarded via the port, defined in the mapping of the MAC table. Otherwise, if the DA is a broadcast, multi-cast, or unknown uni-cast, it sends an OpenFlow packet to OpenFlow switch to flood the packet out of all ports in the spanning tree, except the incoming port.

An L2-Learning switch does not currently implement the aging timer logic. In aging timer logic [**?**], when a switch learns a source address, it timestamps the entry. Every time the switch sees a frame from that source, it updates the timestamp. Now if it does not hear from that source before an aging timer expires, it removes the entry from its MAC table.

In the absence of the aging timer logic, an L2-Learning switch does not have a way to remove the entry from its MAC table in the presence of a link failure. However, it has a way to update the entries, if the packet is received from some other port. This may be possible if L2-Learning starts flooding. Packets are flooded in two cases (1) when the MAC table does not contain the entry related to DA (2) if DA is a broadcast or multicast address. Due to absence of the aging timer logic, case (1) never occurs once the MAC table entry is filled with an entry, but case (2) can occur.

When data packets are transmitted in the network, ARP (Address Resolution Protocol) also runs in parallel if static permanent ARP entries are not added at the source to store the MAC address of destination nodes. The destination address of a packet with an ARP request for unknown MAC address is a broadcast address. Thus, ARP request may become the reason of establishing new path in L2-Learning. ARP requests with broadcast address are sent in network when the "node reachable time" of the ARP entry expires. Thus establishment of new path in L2-Learning may depend on the time when the ARP request with broadcast address is resent in the network. This time is a random value between $[\frac{1}{2} \times baseReachableTime, \frac{3}{2} \times baseReachableTime]$ [**?**]. Thus until the end nodes send ARP requests in network, wrong Flow Entries may be added in the OpenFlow switches after a failure occurs. Thus, recovery may be delayed by client initiated ARP requests.

End hosts do not send ARP requests with broadcast address when a permanent entry is already present in ARP table to stop the ARP traffic between client and server. Hence, in the absence of packets with the destination MAC address as broadcast or multicast address, L2-Learning switch may not recover from failure.

2) L2-Learning PySwitch: PySwitch implements above L2-Learning switch mechanism. Beside this, it also implements aging timer logic.

The value of the aging timer may also delay the establishment of the restoration path. The NOX release with Pyswitch implementation does not suffer this delay as it keeps this value equal to the idle timeout of Flow Entries. Logically, the value of idle timeout should be less than hard timeout. Once a Flow Entry is added, the next packet is not forwarded to NOX controller in a time less than idle timeout. Thus, as the aging timer is equal to idle timeout, it will always expire before the next packet reaches this Pyswitch. Thus the packet reaching to controller after adding Flow Entry may always be flooded in the network. Thus, recovery in this case depends totally on idle and hard timeout value.

3) Routing Mode: Routing mode [**?**] installs the Flow Entries by constructing the shortest path between end hosts. It uses four mechanisms to construct the shortest path between end hosts. The mechanisms are discovery, topology, authentication and routing.

Discovery mechanism uses the "packet in" and "packet out" messages (defined by OpenFlow Protocol) to run the discovery protocol among OpenFlow switches. Discovery mechanism transmits "packet out" to transmit LLDP (Link Layer Discovery Protocol) packets among OpenFlow switches. When the OpenFlow switch receives these "packet out" message, LLDP packets are sent to the respective output port. When the corresponding OpenFlow switch receives LLDP packet, it sends the "packet in" message to the NOX to say about the detected link.

The topology mechanism provides in-memory records of all the links currently up in the network. On the other hand, the authenticator mechanism keeps an in-memory record of all the authenticated hosts and users in the network.

Finally, the routing mechanism keeps track of all the shortest-path routes between authenticated hosts in the network. A packet is forwarded to the NOX when there is no Flow Entry in OpenFlow switch to forward it to destination. If the authenticator mechanism does not yet know about the source, the source is authenticated first. If the destination is known, the routing mechanism adds the shortest path Flow Entries in the OpenFlow switches. Otherwise, if a destination is unknown then it is located by flooding the packet out of every datapath port except the incoming port. Once the destination is known then the shortest path is built from the topology database.

Link addition and failure detection in the routing mode of NOX depends on discovery sent and timeout interval, respectively. Discovery sent interval is the time after which

it sends LLDP "packet out" message to connected OpenFlow Switch. Discovery timeout interval is the time within which if it does not receive the LLDP "packet in" message, it declares the link as lost.

C. Loop free Technology at OpenFlow Environment

Recovery mechanisms require a redundant path in OpenFlow Switch topology. Controller mechanisms may require flooding of data packets when destination is unknown. Furthermore, ARP requests with broadcast address are always flooded in network. So, recovery requires spanning tree protocol (STP) [**?**] or any other loop free technology at OpenFlow switch to run recovery experiment in OpenFlow Networks. (STP is a Data Link Layer protocol and is standardized as IEEE 802.1D)

OpenFlow reference software [**?**], implemented by Stanford provides a way to enable IEEE 802.1D in its networks. To enable IEEE 802.11, switch should first support IEEE 802.1D protocol. Now to use IEEE 802.1D of a switch, OpenFlow software should be enabled with -stp option. Those switches that do support it are expected process all 802.1D packets locally before performing flow loopup. A switch that implements STP much set the OFPC_STP bit in the 'capability' field of its OFPT_FEATURE_REPLY message. OFPT_FEATURES_REPLY [**?**] is a one of the message exchanged between OpenFlow Switch and controller.

It is left to the discretion of a switch to handle STP action appropriately. However, Open vSwitch does not provide –stp option to enable STP. The switches that do not support IEEE 802.1D spanning tree depend on the controller to enable a basic spanning tree at the switch level. Current NOX releases do not implement any loop free technology in their releases. A basic spanning tree mechanism is built by Glen Gibb [**?**] for NOX which attempts to build a spanning tree within an OpenFlow network. However, this mechanism is made for NOX release version 0.5. The structure of higher releases are different from version 0.5 in the terms of xml and meta files. Thus this basic spanning tree requires some modification to integrate in higher NOX releases.

In the absence of STP or any other loop free technology, flooded packets travel in a loop. First, flooded packets persist indefinitely in the network cycle causing congestion. Secondly, mechanism for finding destination path by flooding may not function correctly because a switch may receive packets from a source via multiple ports.

III. Fast Restoration for OpenFlow Networks

Fast Restoration in an OpenFlow network requires an immediate action of the controller on a link change event. This section first gives an overview and then explains the algorithmic approach for fast restoration.

A. Overview

Fast recovery is possible if the incorrect Flow Entries are flushed from all the OpenFlow switches and new entries are installed immediately after detecting the link failure in the existing path. This is possible with the help of controller only if (1) the controller knows about the failure (2) the controller remembers the older path which was established by adding Flow Entries in OpenFlow switches (3) the controller is able to calculate a new path (4) the controller knows all the current flows in the network.

B. Algorithmic Approach

We explain fast restoration in OpenFlow networks using pseudo code written below. Pseudo code shows the action performed by controller on receiving link change event. The link change event can occur if either of OpenFlow Switch or controller run link "addition/failure" algorithm in its level and raise link change event at controller.

Pseudo Code for Fast Restoration

```
1. Given: Link Change Event on Controller
2. for each calculated path (P) via Controller
3.         if(Path P affected by link change )
4.                 Calculate the new available path (P1)
5.                 if(Flow Entries added in OpenFlow (OF)
                   switches w.r.t P)
6.                         Delete the Flow Entries from each OF
                           whose entry is incorrect due to affected
                           path P
7.                         Establish path P1 in OF Switches by
                           adding Flow Entries in each OpenFlow
                           switch
```

In fast restoration, the controller performs a set of actions to restore the path between affected hosts. First action is to check that whether its calculated older paths among end hosts are affected or not (line 2 and 3 of pseudo code). If these are affected then it calculates the new path for those end hosts (line 4). Besides this, it also checks that if it has added Flow Entries in OpenFlow switches regarding the older faulty path (line 5). If so then it deletes the Flow Entries from all the OpenFlow switches regarding the older path (line 6). Then, it adds Flow Entries in all the OpenFlow switches for the new path (line 7).

IV. Emulation Environment

We assume in our testing that the link "failure/addition" detection mechanism is present at data plane level. We manually make ethernet interface down and up for the "link change event" in our network. Port parameters changes by doing this. OpenFlow switches detects this change and reports the change to controller. Our emulated nodes take average of 108 ms time to detects this change. Thus this change is known to OpenFlow switches after this interval.

We call our emulated restoration as predetermined restoration. This is because in our emulation, the administrator provides all the paths to the end hosts with the priority to each path. The paths are provided to controller at the beginning of experiment. The controller chooses the available path which has highest priority.

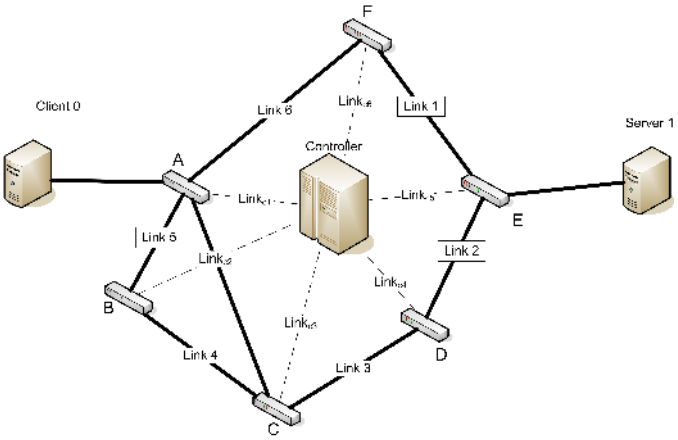

Fig. 2. OpenFlow Network Environment

Immediate recovery requires a redundant path in topology. We create the emulation setup shown in Fig. 2 to test the behavior of OpenFlow Networks in the presence of redundant path in the topology.

A. *Testing of Emulation Environment*

We use Ubuntu version 9.04 for the installation of Open vSwitch version 1.1.0 and NOX version 0.9.0 (zaku). We send 11000 ping packets from Client 0 to Server 1 in the interval of 10 ms. We calculate the number of ping replies received by Client 0. Hard and idle timeout for Flow Entries is set as 20 seconds and 10 seconds, respectively.

Only 97, 9 and no replies are received using Pyswitch, Routing and L2-learning switch, respectively. The existing mechanisms behave very badly in this emulation environment as they suffer loop problems in the presence of redundant path. We emulate OpenFlow in ethernet switches which need tree topology for those packets which are flooded in network. In the absence of tree topology, packets travel in loop. Thus, switches receive packets from a source via multiple looping ports. This is the reason that L2-Learning and routing mechanism do not function properly without the presence of any loop free technology. As predetermined restoration does not allow OpenFlow switches to flood any packets, all the 11000 ping replies are received by Predetermined restoration. Predetermined restoration does not search destination by flooding and also it does not flood ARP packets. When ARP request comes, it sends an OpenFlow packet to forward the packet from a particular output port as defined in the predetermined paths. Thus ARP requests in predetermined restoration also do not suffer looping problem in the absence of any loop free technologies.

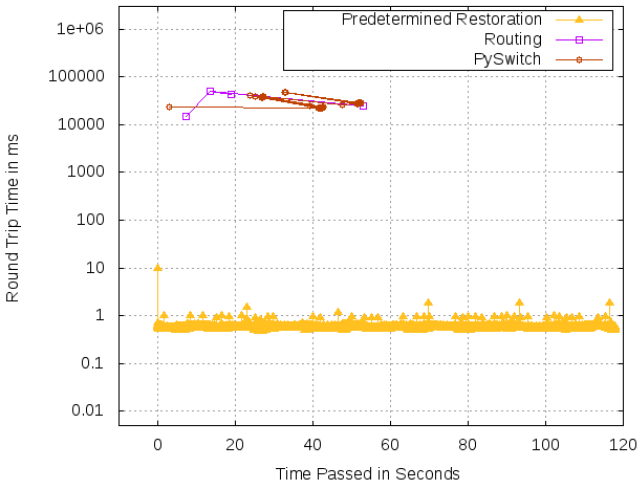

Fig. 3. Round Trip Time in Loop Topology

Fig. 3 shows the round trip time of ping replies received by Client 0. It shows that the round trip time of a ping packet is more than 15 seconds in Routing mode and Pyswitch. However, in case of predetermined restoration, ping reply takes less than 10 ms to arrive.

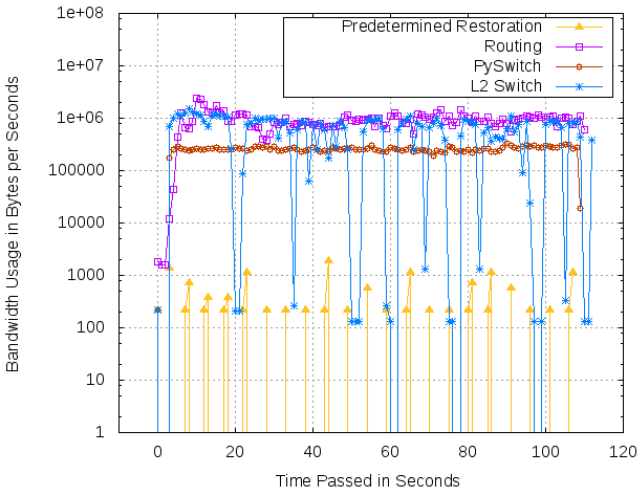

Fig. 4. Bandwidth usage (Bytes per seconds) in loop topology

Fig. 4 shows the bandwidth usage of the NOX line ($Link_{c1}$ in Fig. 2) when 64 byte ping packet is transmitted in networks. Bandwidth usage includes all the packets including periodic echo request and echo replies between the NOX and Open vSwitches. Fig. 4 shows that even though 64 bytes are transmitted in the interval of 10 ms, the bandwidth usage of NOX is more than 10 Mbps for all the existing mechanisms.

This is because packets travel in loop in the absence of any loop free technology. It also shows that bandwidth usage of NOX line via predetermined restoration is comparable to the sent packet size at rate of 10 ms.

Our aim in this paper is not to remove the loop problem by predetermined mechanism. Loop problem can be removed by using any loop free mechanism, for example by building spanning tree in topology. Our aim is to show failure recovery time in the OpenFlow network when outgoing path is affected by link failure. Our proposed fast restoration requires a mechanism for calculating paths. So, we use predetermined mechanism to calculate the path.

B. Emulation Scenario

In order to compare the performance of existing mechanisms with our proposed fast restoration mechanism (via Predetermined Restoration), we believe that existing mechanism should also behave normally without loop problem. So, we break the links (Link 1, Link 2 and Link 5) in topology, shown in Fig. 2, such that data packet should not travel in loop. Loops with data packets are created when Open vSwitches flood the packets. Packets are flooded when controller does not know the destination. Thus we create a topology such that when the NOX replies the Open vSwitches to flood the packet then loop should not be present in our topology. Secondly, loops are also created when ARP requests

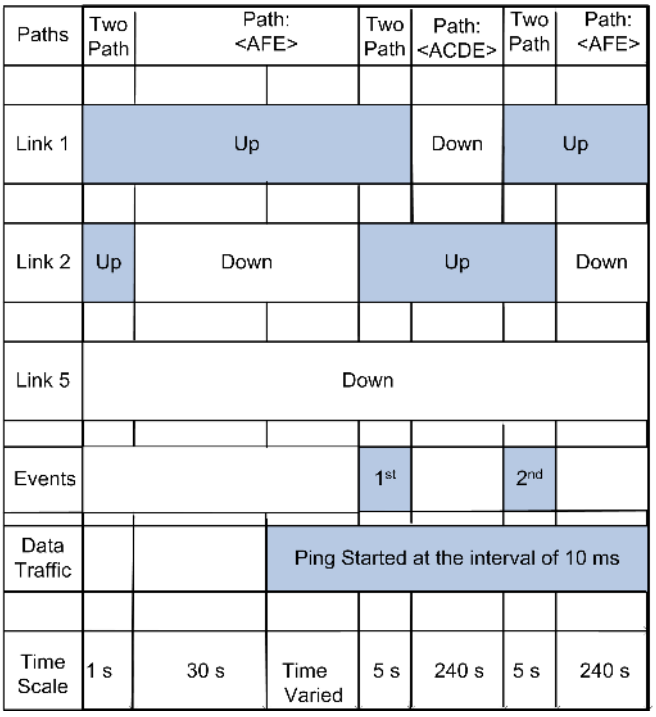

Fig. 5. Time Analysis for Link Events happening in Experiment

with broadcast address are sent. Thus to this, we manually add static ARP entry at Client and Server node to know their MAC addresses. However, as L2-Learning Switch can only be recovered from failure if ARP requests with broadcast address are sent. We enable ARP in case of L2-Learning and will see that how it recovers from failure in the presence of ARP. Base reachable Time [**?**] which is responsible for ARP entry to declare as stale is 60 seconds. Results via disabling ARP for L2-Learning are also shown.

Value of hard timeout and idle timeout for all the mechanisms is kept as 120 and 10 seconds, respectively. We also keep the value of hard timeout equal to infinite to show the difference of recovery time in our results.

Fig. 5 gives time analysis of the events happening in our emulation networks (Fig. 2). We break the link 5 at the beginning of experiment. We break the link 1 and link 2 after a time interval shown in Fig. 5. Link 1 and link 2 are up and present at the beginning of experiment. We break link 2 after the span of 1 second. Thus now there is only one path left to reach the Server. The path is $< AFE >$. After 30 seconds of time span, we start ping from client 0 at the interval of 10 ms. Ping packets travel path $< AFE >$ to reach the server as only one path is available now to reach server.

We give two link events in our experiment, the time when we make link 2 up and link 1 down is the 1st event and the time when link 1 is made up and link 2 down is the 2nd event. Events in our emulation scenario last 5 seconds, where we make change in outgoing path by doing link up and down of two different paths $< AFE >$ and $< ACDE >$. Two looping path $< AFE >$ and $< ACDE >$ are available in this 5 seconds interval but this does not give rise to looping of data packets. This is because Flow Entries are already established in OpenFlow switches when the events are given in our emulation.

We vary the time of first "link addition/failure" (1^{st} Event in Fig. 5) to show the variation in our results. This time gives the time left to flush the Flow Entries from Open vSwitches. The path changes to $< ACDE >$ after 1st event. We wait for 240 seconds to give the second "link addition/failure" (2^{nd} Event in Fig. 5) in our experiment. Thus, after 2nd event, ping path again changes to $< AFE >$.

V. Analysis of Results

This section deals with the calculation of "flow switchover time", number of packet drops and round trip time. "Flow Switchover Time" is the time spent by flow to switch to another available path.

A. Mathematical Analysis of the Flow Switchover Time

This section gives the mathematical formula for the calculation of "flow switchover time" for routing and L2-learning mechanism.

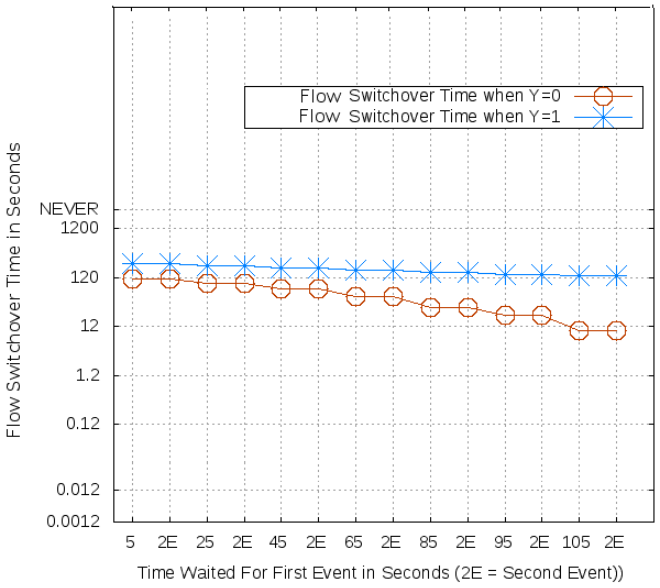

Fig. 6. Flow Switchover Time (Mathematical Formula)

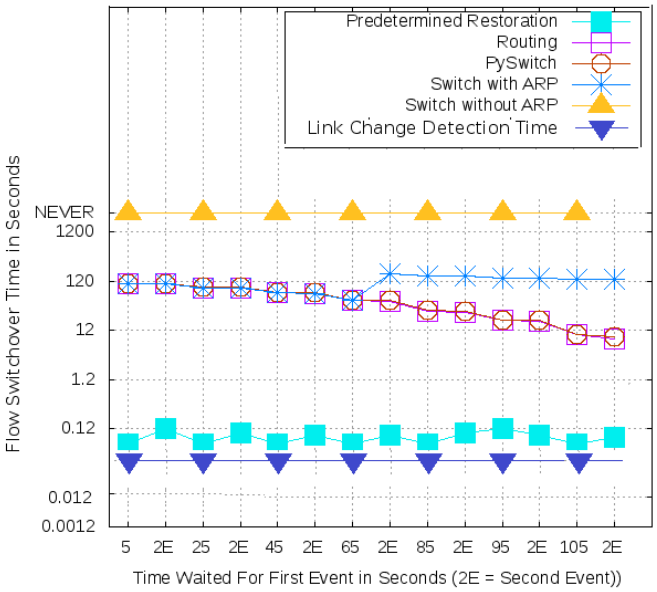

Fig. 7. Flow Switchover Time (Emulation)

Let the value of the hard timeout is equal to T and the first Flow Entry is added at time t_0. Suppose the traffic is going from (Client) 0 to (Server) 1 through the path $< 0AFE1 >$ (Fig. 2) and link between F and E gets broken at time t_1.

Recovery from this failure depends on the time t when controller detects the link failure and also the time when OpenFlow switches A and E will delete the older incorrect Flow Entries from their FlowTable. Present existing mechanisms depend on idle and hard timeout to flush the entries from OpenFlow Switches. In our emulation scenario, packets are sent in 10 ms interval and idle timeout is 10 seconds. Thus idle timeout in A will never expire. So, it will flush its older incorrect Flow Entries after its hard timeout expire. But E will flush entries after idle timeout is expired. As value of hard timeout is more than idle timeout, switching to new path in this scenario depends on hard timeout. Thus, "flow switchover time" can be given by Eq. **??**.

$$R_T = T - t_1 + \lfloor \frac{t - t_0}{T} \rfloor \times T \qquad (1)$$

The graphical notation of Eq. **??** is given in Fig. 6. It shows the "flow switchover time" of existing mechanisms when $Y = \lfloor \frac{t-t_0}{T} \rfloor = 0$ and when $Y = \lfloor \frac{t-t_0}{T} \rfloor = 1$.

Fig. 6 and 7 show integer value and 2E in X-axis. Integer value let say x in X-axis shows the time of first event when it is given after the x seconds of ping initiated. 2E after let say x in X-axis shows the time of second event when first event is given after x seconds of time when ping initiated.

B. Emulation Results

Fig. 7 gives the "flow switchover time" of our fast restoration and existing mechanisms via emulation. It shows that "flow switchover time" of Routing and Pyswitch is almost similar to the result obtained via mathematical graph by virtue of Y=0 (Fig. 6). It also shows that "L2-learning switch with ARP enable" works similar to line with Y=0 from X axis value 5 to 65. But after that it behaves as the line with Y=1. This is because ARP is not sent in that hard timeout interval. So, now L2-Learning switch has already added incorrect Flow Entries in OpenFlow Switches. Thus now it has to wait for second timeout to expire these entries.

Fig. 7 also shows that in the absence of ARP, L2-Learning Switch never switch to other path in the presence of failure.

Fig. 7 shows that our proposed fast restoration always switches to other path within an interval of 12 ms ("flow switchover time" - "link change detection time"), no matters how much time is left to expire the hard timeout. This is because fast restoration does not depend on expiration of hard timeout to flush the Flow Entries from OpenFlow switches. It immediately flushes the Flow Entries once the controller detects the failure and establishes a new path.

Our predetermined restoration results show the possible fast recovery when the controller has to take part in the switchover decision. Fig. 7 gives the "flow switchover time" when hard timeout is 120 seconds. However, we also calculate the "flow switchover time" when hard timeout is infinite. The existing mechanisms do not change path when hard timeout is infinite. However, predetermined restoration switches to other path within 12 ms when hard timeout is infinite.

The "flow switchover time" in our emulation results show that our proposed restoration recovers more fast than all the existing mechanisms. This is because it takes immediate action on link failure. We believe that our restoration mechanism enables fast recovery in true sense when all the decision is taken by controller which is installed in different hardware.

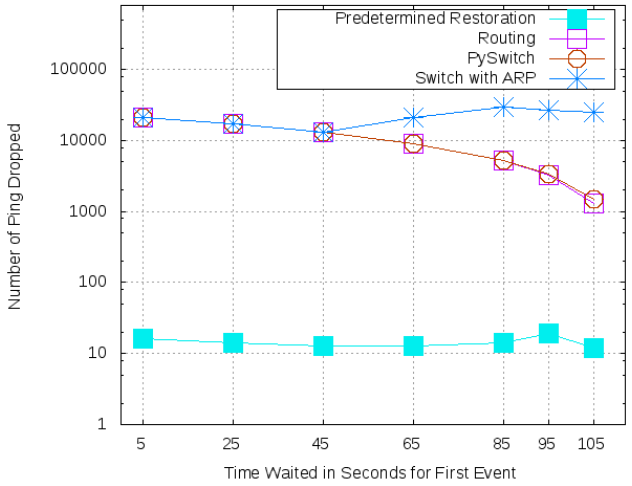

Fig. 8. Total Number of Ping Drop via Link Change Event

Fig. 8 gives the total number of ping drops via the "link change events" given in our experiment. The reason of these drops is explained in Fig. 7 where flow has taken some time to switch to other path. In case of existing mechanisms, more than 1000 packets get dropped. However, less than 20 packets are dropped in our fast restoration mechanism.

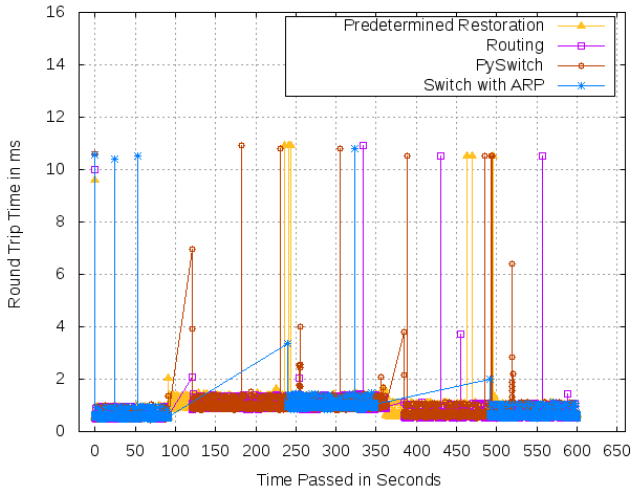

Fig. 9. Round Trip Time in ms

Fig. 9 gives the round trip time of ping packets when time waited for first event (Fig. 5) is 85 seconds. It shows the spikes in round trip time. When the packets are sent to the controller for deciding the path then ping packets take more time in getting reply. OpenFlow network delays some of the packet as it is dependent on controller which is situated far away from OpenFlow Switches to take the action. Furthermore, it also depends on how much controller is loaded at the time when it receives the request from OpenFlow switches.

VI. Conclusion and Future Work

OpenFlow architecture allows us to implement restoration options in OpenFlow networks which are much faster than MAC reconvergence (Routing and L2-Learning PySwitch) or the client-initiated recovery using a new ARP request (L2-Learning Switch). Our fast restoration mechanism can be integrated in any mechanism where controller is able to detect the failure by some means. In our fast restoration mechanism, flow is able to switch to another path within 12 ms interval regardless of the time left to expire timeouts of the Flow Entries. We tested this restoration for a single ping flow. Therefore, restoration can also be tested for muliple flow in networks. Furthermore, as Automated Protection Switching (APS) is more faster than restoration, Openflow can also be tested with this mechanism, where we may remove the need to contact the controller after a failure.

Acknowledgment

This work was partially funded by the European Commission under the 7th Framework ICT research Programme projects SPARC and OFELIA.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Peterson, J. Rexford, S. Shenker, J. Turner, "OpenFlow: Enabling innovation in campus networks", SIGCOMM, Rev. 38(2), 69-74, 2008.
[2] N. Gude, T. Koponen, J. Pettit, B. Paffa, M. Casado, N. McKeown and S. Shenker, "NOX: Towards and Operating System for networks", In ACM SIGCOMM, CCR, 2008.
[3] Beacon: A java-based openflow control platform. See http://www.openflowhub.org/display/Beacon/Beacon+Home, Nov 2010.
[4] Teemu, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks", In OSDI, Oct 2010.
[5] HIDEyuki Shimonishi, Shuji Ishii, Yasunobu Chiba, Toshio Koide, "Helios: Fully distributed OpenFlow controller platform", GEC, 2010.
[6] Z. Cai, A. L. Cox, T. S. Eugene, "Maestro: A system for scalable OpenFlow control", Rice University Technical Report TR10-08, 2010.
[7] SPARC: http://www.fp7-sparc.eu/
[8] MEF: http://metroethernetforum.org/index.php
[9] B. Niven-Jenkins, D.Brungard, M. Betts, N. Spreche, "MPLS-TP Requirements draft-ietf-mpls-tp-requirement-10", 2009
[10] Jean Philippe Vasseur, Mario Picavet, Piet Demeester, "Network Recovery: protection and restoration of optical, SONET-SDH, IP and MPLS", Morgan Kaufmann, 2004.
[11] OpenFlow Specification 1.0: http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf
[12] STP via OpenFlow Specification: http://www.openflow.org/wk/index.php/OpenFlow_0.8.9_release_notes
[13] K. Kompella, and Y. Rekhter, "RFC 4761: Virtual Private LAN service (VPLS) Using BGP for Auto-Discovery and Signalling", 2007
[14] ARP Manual: http://linux.die.net/man/7/arp
[15] NOX Documentaion: http://noxrepo.org/manual/app-index.html
[16] ISO DIS 10038 MAC Bridges.
[17] NOX Basic Spanning Tree Implementation: http://www.openflow.org/wk/index.php/Basic_Spanning_Tree