Scalable Fault Management for OpenFlow

James Kempf, Elisa Bellagamba, András Kern,
Dávid Jocha, Attila Takacs
Ericsson Research
james.kempf@ericsson.com

Pontus Sköldström
Acreo AB,
Stockholm, Sweden
ponsko@acreo.se

***Abstract*— In the OpenFlow based split architecture, data-plane forwarding is separated from control and management functions. Forwarding elements make only simple forwarding decisions based on flow table entries populated by the controller. While OpenFlow does not specify how topology monitoring is performed, the centralized controller can use Link-Layer Discovery Protocol (LLDP) messages to discover link and node failures and trigger restoration actions. This monitoring and recovery model has serious scalability limitations because the controller has to be involved in the processing of all of the LLDP monitoring messages. For fast recovery, monitoring messages must be sent with millisecond interval over each link in the network. This poses a significant load on the controller. In this paper we propose to implement a monitoring function on OpenFlow switches, which can emit monitoring messages without posing a processing load on the controller. We describe how the OpenFlow 1.1 protocol should be extended to support the monitoring function. Our experimental results show that data plane fault recovery can be achieved in a scalable way within 50 milliseconds using this function.**

Keywords-component; OpenFlow, protection switching, OAM, MPLS

I. Introduction

Split architecture is the concept of decoupling the control functions from the forwarding and data processing elements and defining an open programmable interface (API) between them. Forwarding elements make only simple forwarding decisions programmed by a remote controller. Such split allows the independent design of control and forwarding, which fosters novel control paradigms and leads to Software Defined Networking (SDN) concept. The OpenFlow [1] realizes the above API through modeling a switch as a collection of input and output ports and a single flow table. This flow table stores definitions of packet flows forwarded by the switch. The latest specification, OpenFlow 1.1 [2], updates this model by introducing a packet processing pipeline formed by up to 256 flow tables and a group table. With the new multi-table extension, scalable multi-stage packet processing is possible without flow table size explosion. Previously, flows requiring similar processing actions, like VLAN tagging, required recording of this action multiple times separately in the flow table. With the group table, these actions only need to be recorded once in the group table and hence updating such actions only requires a single update to the group table while flow table entries do not need to be modified. These improvements enhanced the capabilities and scalability of OpenFlow.

OpenFlow has been deployed in large scale experimental test-beds in the US (GENI) [3], Japan (JGN2plus) [4], and in the EU (OFELIA) [5]. But research has moved beyond these experimental networks and now considers the application of OpenFlow in operator production environments. The authors in reference [6] argue that OpenFlow provides the means for a successful unified control plane for packet and circuit transport networks. Two multi-layer use cases are highlighted. The first use case is packet links with adaptive bandwidth, implemented by monitoring the traffic over IP layer links then compensating increased demand through additional TDM slot allocation. The second use case is automatic optical route bypass in which underlying wavelength paths are dynamically established to reduce the load posed on intermediate IP hops. The authors in reference [7] take these application ideas further and include service characteristics into the consideration for lower layer path selection. The concept of service-aware recovery schemes is introduced whereby, depending on the service, recovery is provided by different layers. This work suggests that OpenFlow could play a role in multi-layer traffic engineering for transport networks.

In this paper we discuss how to support transport networks with OpenFlow from a management perspective, i.e. scalable fault management and path protection. In Section II, we briefly review requirements for scalable fault management in transport networks and discuss how they can be supported with OpenFlow 1.1. We argue that centralizing all the functions needed for transport network fault management in a remote controller according to the canonical OpenFlow model will lead to unacceptable performance, and that a judicious partitioning of the functionality between the switch and the controller will lead to more acceptable scalability. Section III presents our experimental modification of the OpenFlow 1.1 switch model to support scalable fault management. In Section IV, we discuss how our fault management scheme could be applied to MPLS-TP [9], a variant of MPLS used for transport networks. Section V presents some experimental results illustrating how our fault management scheme could satisfy the performance requirements for transport networks while reducing the load on the controller. Finally, we draw some conclusions in Section VII.

This work was partially sponsored by the SPARC FP7 EU project: http://www.fp7-sparc.eu

II. OpenFlow for Transport Networks

Because a transport connection aggregates traffic, high reliability is an important requirement. Reliability requires automatic recovery mechanisms that are triggered by Operations, Administration, and Maintenance (OAM) tools to re-establish connectivity when a path failure occurs.

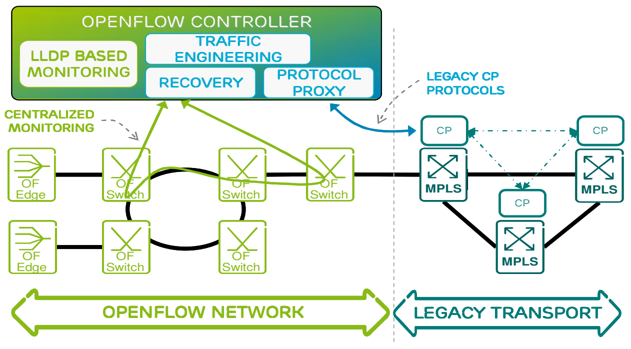

Figure 1. Transport network support in OpenFlow

Recovery is categorized into restoration and protection. For restoration, detour or alternate paths are computed and configured only after a failure is detected; making this method relatively slow. For protection in contrast, a backup path is configured parallel to the working path; hence when a failure is detected, a fast switchover minimizes traffic disruption. Transport applications demands a 50 millisecond recovery time; requiring protection to be supported by any transport network technology. Later, the traffic engineering function can calculate recovery paths to balance traffic more evenly when possible.

Figure 1 illustrates how the transport OAM functions could be realized in the canonical OpenFlow architecture, where all control plane functions are centralized. The monitoring application sends probe messages over the switches to discover data-plane links and maintain the network topology. The format of these probes is inline with the Link Layer Discovery Protocol (LLDP) [12]. While this mechanism is applicable to detect link or node failures within sub-seconds time frame, it is not applicable to monitor the health of links or end-to-end tunnels for protection. The centralized LLDP based monitoring model has serious scalability limitations because the controller must be involved in the processing of all the probe messages. For the required 50 milliseconds recovery time in transport networks, connectivity monitoring needs to run with approximately a 10 millisecond frequency. This means that the controller must be able to emit, receive and process around 100 monitoring packets per second per link and per end-to-end tunnel. In a transport network it is not unusual to have hundreds of links and thousands of tunnels. Hence the controller must be able to handle and react to millions of monitoring packets per second just to monitor the health of the network. This is not just a burden on the controller; it also imposes an unacceptable load on the control network.

While centralized link health monitoring for protection is clearly not scalable, the traffic engineering module running on the controller can still be used for restoration. Once a failure is discovered an alternate path can be requested from this module and then configured via the OpenFlow protocol. Protection on the other hand requires specific processing in the switches. That is, switches must detect failures and take local action to quickly reroute flows. The fast failover group entry in OpenFlow 1.1 [2] provides the means for local rerouting in the switches. However, due to the lack of connectivity monitoring in the switches, a particular switch can only react to local link failures. End-to-end tunnel protection cannot be realized with OpenFlow today.

III. Improved Fault Management for OpenFlow

To overcome the scalability limitations of centralized connectivity monitoring, we propose to relax the separation of control operations to include connectivity monitoring OAM in the switch. To ensure fast detection of any impairment along the path, the connectivity monitoring must operate in a pro-active fashion. The source end of the monitored flow, which is the entry point of a transport tunnel, periodically emits probe packets and interleaves them into the data flow running along that path. The destination end extracts the probe packets from the data flow. If the destination end stops receiving the probe packets for a long enough period (referred to as the detection time) or a received packet indicates that the source endpoint detected some errors in the other direction (remote defect indication), some node or link in the monitored flow is assumed to have failed, and the destination end node switches over to an alternate path. The detection time is typically defined as the product of the interval between two probe packets and the number of consecutive probe packets whose loss would indicate the failure of the flow. This latter multiplier is employed to avoid false positive failure indications in the case of big jitter or loss.

In order to implement improved fault management efficiently in OpenFlow we applied the strict integration of the above functions with the OpenFlow switch forwarding. We propose extending the OpenFlow 1.1 switch forwarding model with new entities and we also show what protocol extensions are essential to provision these novel entities. Nevertheless, we do not intend to provide a comprehensive protocol specification, rather an insight on the necessary switch model and protocol updates.

A. *Generating Monitoring Messages within the Switch*

To reduce the amount of processing required in the switch we allow multiple monitored entities using the same message rate to share a monitoring packet generator, separating packet generation from formatting (i.e. filling in identifiers or timers). A packet generator for a particular frequency emits a template packet. The template packet enters the processing pipeline through a virtual port and is treated like a regular packet. The flow table is programmed with a flow rule that matches the template packet, specifically the “In Port” field matches the virtual port number of the packet generator, while the other fields match on packet header fields (e.g., to identify the OAM type). The flow action instructs the switch to forward the packet to a group table entry, which replicates and forwards the packet to further group table entries. These latter entries implement the actions to fill in the missing fields of the currently empty packet: timer values, identifiers, associated signals, like Remote Defect Indication.

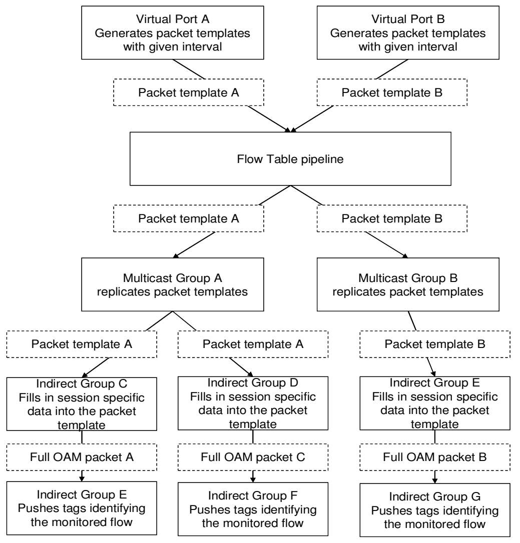
Virtual Port A Generates packet templates with given interval
Virtual Port B Generates packet templates with given interval
Packet template A
Packet template B
Flow Table pipeline
Packet template A
Packet template B
Multicast Group A replicates packet templates
Multicast Group B replicates packet templates
Packet template A
Packet template A
Packet template B
Indirect Group C Fills in session specific data into the packet template
Indirect Group D Fills in session specific data into the packet template
Indirect Group E Fills in session specific data into the packet template
Full OAM packet A
Full OAM packet C
Full OAM packet B
Indirect Group E Pushes tags identifying the monitored flow
Indirect Group F Pushes tags identifying the monitored flow
Indirect Group G Pushes tags identifying the monitored flow

Figure 2. Ingress-side OAM Packet Generation

Once the packet has been filled in, additional headers may be added to the packet and the packet is sent out on the same port as the monitored tunnel. Typically the ingress of the monitored tunnel can be identified with a set of actions (pushing tags, setting header values etc.). These actions can be encoded in a group table entry. Implementing OAM packet insertion requires the packet filler group table entry to be configured to pass the packet to the tunnel ingress encoding entry. Figure 2 illustrates the process.

The OpenFlow switch sees the packet generators as virtual ports. Therefore to configure them we can adopt the configurable virtual port extension of reference [8]. This extension only allows virtual ports to act as packet modifiers but not to create packets on their own. A new sub-type of virtual ports is required, an active virtual port, that allows packet generation. To configure the details of an active virtual port, we define the "create packet generator" action. This action specifies the packet sending interval as well as encoding basic encapsulation options.

B. *Reacting to Monitoring Messages in the Switch*

Figure 3 illustrates how OAM packets are processed on the egress side of the monitored tunnel. Incoming OAM packets enter the switch through ports like any other packets. First the monitored tunnel including the OAM packets is identified with a flow table entry. The typical actions associated to that entry are removing the tunnel tags and passing the packet to the next stage, where the payload is further processed. Demultiplexing the OAM packets must be done at this phase as well. Demultiplexing is done either as part of popping the tunnel tag or expressed as an additional rule deployed in a later flow table. The OAM packets can be terminated at a group table entry containing a single action bucket. At this point the content of the OAM packet is processed and the timers associated to the OAM termination points are updated. These steps are implemented in a single atomic action, the "terminate OAM packet" action.

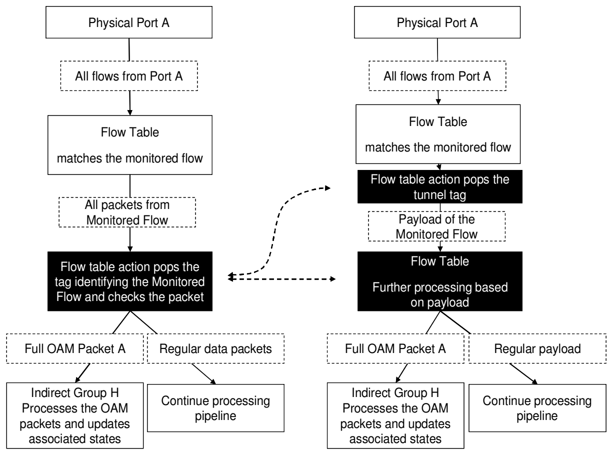
Physical Port A
All flows from Port A
Flow Table matches the monitored flow
All packets from Monitored Flow
Flow table action pops the tag identifying the Monitored Flow and checks the packet
Full OAM Packet A
Regular data packets
Indirect Group H Processes the OAM packets and updates associated states
Continue processing pipeline
Physical Port A
All flows from Port A
Flow Table matches the monitored flow
Flow table action pops the tunnel tag
Payload of the Monitored Flow
Flow Table Further processing based on payload
Full OAM Packet A
Regular payload
Indirect Group H Processes the OAM packets and updates associated states
Continue processing pipeline

Figure 3. Egress-side OAM Signaling Reception

The exact steps to be executed as well as the fields to be considered depend on the OAM type; therefore, one OAM termination action is defined per OAM type. The OAM termination action implements a state machine with appropriate timers. Depending on its state, the action affects the state of the containing action bucket. When a proper monitoring packet is received, the containing bucket is alive. But if the detection time expires without receiving any monitoring packets, the containing action bucket is set to down, indicating the transport tunnel has failed, and the controller is notified.

Binding the OAM packet check to popping the tunnel tag extends the scope of the tag removing actions beyond OpenFlow 1.1, with checking whether the remaining packet is an OAM one and relaying the caught OAM packets to the proper OAM processing entity. This requires an extended version of the pop-tag actions. Such extended versions encode not only the header fields required for popping the tag (e.g., TTL generation), but also defines an identifier of the OAM processing group table entry. On the other hand, when catching OAM packets with the flow table entry, only one action is needed, to redirect the matching OAM packets to the processing group table entry. Such an action is already provided by the OpenFlow 1.1. However, the OAM packets won't be identified with header fields covered by the standard OpenFlow matching fields, so OpenFlow must also be extended with these new fields.

Upon the occurrence of any monitoring and protection events the switch notifies the controller. OpenFlow Error message was modified to support a generic *Notification* message. Two notification messages are defined: first to report any changes to the status of the monitoring entity and a second to report the protection reactions. This separation is considered in order to allow mixed scenarios, where controller driven restoration is combined with switch performed continuity monitoring.

C. *Monitoring-based Fast Failover Switching*

The fast failover group entry in OpenFlow 1.1 consists of two or more action buckets with a well defined order. The first action bucket describes what to do with the packet during normal operation. If this bucket is declared as unavailable then the packets are treated according to the second action bucket. The status of the buckets can depend on the status of a physical port or on other buckets. This makes it easy to associate the monitoring capability to the protection switching mechanism. The action buckets of the Fast Failover Group entry watch the status of the appropriate OAM termination group entry. This configuration operates only in case of bidirectional tunnels, but transport networks rarely use unidirectional transport tunnels.

IV. OpenFlow For MPLS Transport Profile

A. *Fault Managent in MPLS Transport Profile*

In MPLS-TP [9] protection switching is performed by the endpoints of the working path and the backup path. To stay synchronized, the endpoints may notify each other on protection actions, using either a data-plane associated protocol, like Protection State Coordination or a mechanism that is part of the control plane (e.g., the GMPLS Notify message). To detect Label Switch Path (LSP) failures the MPLS-TP OAM supports a continuity check packet flow. The recommended implementation is based on the Bidirectional Forwarding Detection (BFD) protocol [10]. BFD packets are transmitted in-band on the same tunnel as the monitored data flow in order to share fate with it. MPLS-TP uses Generic Associated Channels (G-ACh), to interleave different typed data channels into the LSP. In order for a Label Edge Router (LER) to distinguish normal data-traffic on the LSP from associated channel traffic, a special label, the Generic Associated Label (GAL) is used.

B. *Adding BFD and protection support to OpenFlow*

The OpenFlow protection switch scheme presented in Section III was designed before the OpenFlow 1.1 specification stabilized. Therefore, the current implementation is based on an extension of the OpenFlow 1.0 reference switch implementation that supports MPLS forwarding [8]. As a consequence, instead of using group table entries to implement the monitoring packet construction procedure, we relied on configurable virtual ports. This means that not only the packet generator but all other packet construction steps are implemented with virtual ports and configured through virtual port management commands. In addition, the lack of multiple tables prohibited implementing the detached BFD exception mechanism. Therefore, an updated "pop MPLS tag" action implements both, removing the MPLS tag and also catching the BFD frames. The latter action is done by checking if the next label is GAL and identifying the BFD packets by reading the type field of the associated channel header. Finally the BFD packets are passed to a dedicated module in the switch control plane directly instead of a BFD termination group table entry.

The lack of a dedicated fast failover group table entry required addition of a BFD state database that stores all state information related to the BFD sessions (outgoing MPLS LSP virtual port numbers, timer values, paired sessions, etc.). A failure detection module processes all received BFD packets and updates the associated state entry. It also monitors if the timers associated to these entries are expired. Upon timer expiration, first it identifies the BFD session data whose timer is expired, and based on that it locates the BFD session of the associated backup path. If the backup path is operational it requests the flow table updater module to update all flows using the failed LSP to the backup one. The flow table updater iterates through all the installed rules in the flow table looking for rules containing an output action referring to the virtual port number of the failed LSP. Whenever such an entry is found the port number in the output action of this entry is updated to refer to the backup LSP instead. Once the whole table has been examined and updated the failover is complete. Finally, the flow table updater notifies the controller of any changes in the BFD states or protection switching using *Notification* messages.

V. Experimental Results

To evaluate our scheme, we measured the failover time in a test bed consisting of two OpenFlow LERs with two LSPs between them; one working and one backup (see Figure 4). The switches were implemented as modified OpenFlow 1.0 soft switches on Linux, and the controller was modified version of the NOX open source controller [11] running on Linux. The working and backup LSPs are monitored by two BFD sessions, both running with a packet transmission interval of 10 ms and a detection multiplier (timeout) of three, giving a maximum detection time of 30 ms. From the source we transmit constant bit rate traffic with roughly 800 packets per second across the working LSP. This results in emitting packets at 1.25 milliseconds intervals. At the sink side we capture the incoming traffic and measure the inter-arrival time of the packets. To trigger failover we simply unplug the network cable of the current working path. The captured inter-arrival time of the constant bit rate packets before and after recovery at the sink is an approximation of the full protection time.

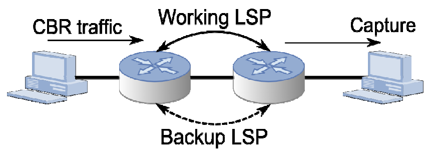

Figure 4. Measurement setup

As depicted on Figure 5, the inter-arrival time is around 1.25 milliseconds with minor jitters during normal operation. But when the cable was unplugged the delay increased to about 27-32 milliseconds. Based on 76 failover events we calculate a 95% confidence interval for the failover time to be 28.2 ± 0.5 milliseconds.

The expected value for the detection time is slightly counter-intuitive; it is actually lower than the required detection time. This is caused by the fact that with BFD packets we are not counting lost packets but rather set a timeout every time we receive a BFD packet. In this particular case the detection time is between 30 and 20 milliseconds. If a BFD packet is the last received packet, the timer was just set and will expire 30 ms later. On the other hand, if the first lost packet is a BFD packet it has been at most 10 ms since the timer was set and a minimum of 20 ms left until it expires. Given the accuracy of our timers and the timer jitter in the off-the-shelf PCs used in the experiment, an average protection time of 28 ms is a reasonable result.

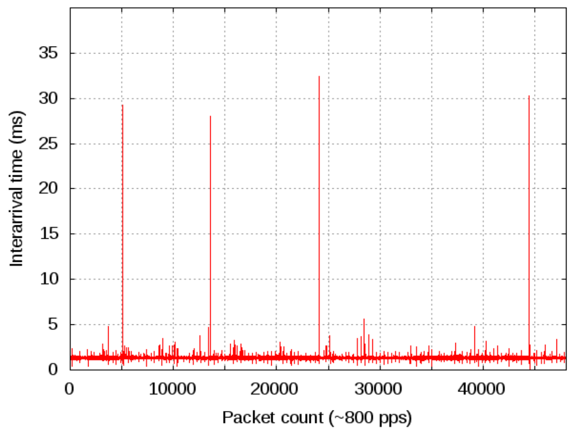

Figure 5. CBR packet inter-arrival times for four failover events

VI. Offloading the Controller

As discussed earlier, placing control functions at the switches instead of in a remote, central control entity is expected to enhance scalability through offloading the controller. This gain is achieved through redefining the role of the LLDP based centralized monitoring (Figure 6). As first step, LLDP is not used any longer for the continuity check, which relaxes the time constraints from milliseconds to seconds. Then, by using switch monitoring features together with link down notifications, the new link detection and failure declaration are decoupled. As a consequence there is no need to consider accidental packet losses and jitters, which allows further increase of packet sending intervals.

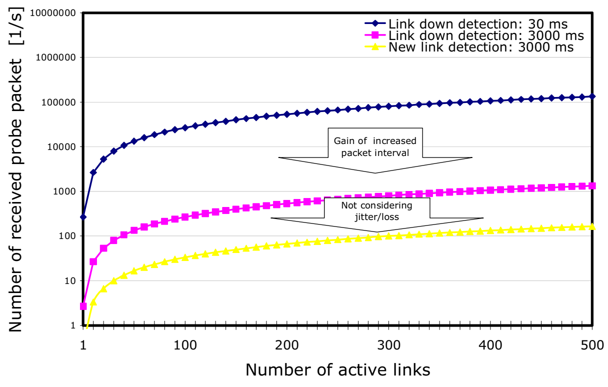

Figure 6. Decrease of packet processing load by using BFD for link monitoring.

VII. Conclusions

In the split architecture, forwarding elements make only simple forwarding decisions based on flow table entries populated by a controller. This model has serious scalability limitations for fault management, because the controller has to be involved in the processing of all the monitoring messages.

To overcome the scalability limitations of centralized fault management, we proposed to relax the separation of control and forwarding operations slightly. We argued that OAM, in particular connectivity monitoring is a function that needs to be distributed and deployed close to the data-plane to provide a scalable way for data-plane connectivity monitoring and protection switching. We proposed to place a general message generator and processing function on OpenFlow switches. We described how the OpenFlow 1.1 protocol should be extended to support the monitoring function. In our implementation, we followed the fault management operation of MPLS-TP. Through experiments we provided a proof that data-plane fault recovery within 50 milliseconds can be reached in a scalable way with our proposed extensions. As a next step we will extend our open-source OpenFlow 1.1 switch implementation [13] with the proposed connectivity monitoring functions.

References

[1] "OpenFlow Switch Specification: Version 1.0.0 (Wire Protocol 0x01)", http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf, December 31, 2009.

[2] "OpenFlow Switch Specification: Version 1.1.0 (Wire Protocol 0x02)", http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf, February 28, 2011.

[3] Global Environment for Network Innovations (GENI) http://www.geni.net/

[4] JGN2plus research and development testbed network http://www.jgn.nict.go.jp/

[5] OpenFlow in Europe: Linking Infrastructure and Applications (OFELIA), http://www.fp7-ofelia.eu/

[6] S. Das, G. Parulkar, and N. McKeown, "Unifying Packet and Circuit Switched Networks", IEEE Globecom, Nov 30 – Dec. 4 2009.

[7] S. Das, Y. Yiakoumis, G. Parulkar, N. McKeown, P. Singh, D. Getachew, and P. D. Desai. "Application-aware aggregation and traffic engineering in a converged packet-circuit network" In Optical Fiber Communication Conference and Exposition (OFC/NFOEC), March 2011.

[8] J. Kempf, S. Whyte, J. Ellithorpe, P. Kazemian, M. Haitjema, N. Beheshti, S. Stuart, and H. Green, "OpenFlow MPLS and the Open Source Label Switched Router", International Teletraffic Congress, September, 2011.

[9] M. Bocci, S. Bryant, D. Frost, L. Levaru, and L. Berger, " A Framework for MPLS in Transport Networks", RFC 5921, Internet Engineering Task Force, July, 2010.

[10] R. Aggarwal, K. Kompella, T. Nadeau, and G. Swallow, "Bidirectional Forwarding Detection (BFD) for MPLS Label Switched Paths (LSPs)", RFC 5884, Internet Engineering Task Force, July, 2010.

[11] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, amd S. Shenker, "NOX: Towards an Operating System for Networks", CCR, July, 2008.

[12] "IEEE Standard for Local and Metropolitan Area Networks – Station and Media Access Connectivity Discovery", IEEE Std. 802.1ab, September, 2009.

[13] OpenFlow 1.1 Switch Implementation available at: https://openflow.stanford.edu/display/of11softswitch and https://github.com/TrafficLab/of11softswitch