Quality of Service with Software Defined Network.

Andrea Pittaro, Matteo Maso Department of Information Engineering, University of Padova – Via Gradenigo, 6/b,
35131 Padova, Italy
Email: {pittaroa,masomatt}@dei.unipd.it

***Abstract*—Quality of Service (QoS) in existing network architectures is an ongoing problem. A lot of researchers and industries tried to solve this problem and found different solutions but some of them are too expensive other failed or were not implemented. Software Defined Networking (SDN) paradigm has emerged in response to traditional networking architecture's limitations. The main advantage of SDN is a centralized global network view. This particular feature pushed researchers to keep on improving QoS.**

In this report we will focus on two different aspects; all problems we faced during our attempt to create a based SDN application able to support a QoS service and the results we achieved.

OpenFlow standard and python POX controller, which is largely spread, were used as instruments to make this experience possible, despite the limit of using a new type of tecnology still in the research phase.

I. Introduction

Software-defined networking (SDN) is a way to manage networks that separates the control from the forwarding plane. SDN offers a centralized view of the network, giving an SDN Controller the ability to act as the "brain" of the network. The SDN Controller relays information to switches and routers via southbound APIs, and to the applications with northbound APIs. One of the most well-known protocols used by SDN Controllers is OpenFlow, widely accepted even if it isn't the only SDN standard. Centralized, programmable SDN environments can easily adjust to the rapidly changing business'needs. SDN can reduce costs and restrict wasteful provisioning, as well as provide flexibility and innovation for networks.

A. *OpenFlow*

During 2006 ,thanks to M. Casado, a new network architecture was developed, called Ethane. On this conncept was born, during 2011, the Open Network Foundation, a no profit association that would innovate SDN and standardize OpenFlow protocol whit relatives tecnology [7]. The first version of OpenFlow is 1.0.0 was released on 2009 but the last standardized version v1.5.1 was approved only on march 2015 [8].

Openflow 1.0 messages

Openflow defines a number of messages between a controller and different switches. Now there will be presented some messages we exploited [2]:

- *Hello*
  This message is sent both from the controller and the switch who is looking for a controller after it is turned on. If the switch sent the message, the controller, after receiving it, will send the FeatureReq packet to that switch.

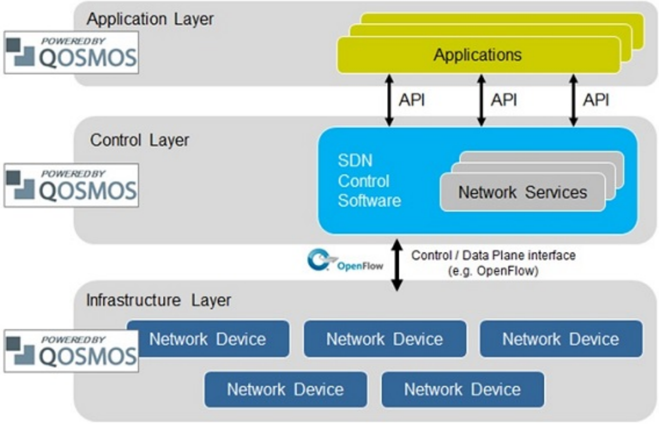

Figure 1: *OpenFlow*

- *FeatureReq*
  This is the message that the controller sends to a switch to get to know its abilities. The switch respond with a FeatureRes and when this last message reaches the pox controller, the ConnectionUp event is raised. With the featureRes the controller will know how many ports a switch has and their properties (e.g. if they are connected, link speed,...), the flow table capabilities and the supported statistics.
- *PortStatus*
  this message is sent from a switch to the controller, to notify that some ports has changed their status. It's useful to check if a port has someone connected to it.
- *StatsReq*
  The controller will send this message to request the statistics. The payload of the message contains the type of statistic the controller wants to know. The switch then will answer with the Stats reply, if it supports that type of stat. If not, The switch will send a Bad_Request message.
- *PacketIn*
  This message is sent from the switch to the controller everytime a packet arrives without matching any rule. The controller can decide to ignore that type of message, insert this message in a PacketOut message, or set a flow rule for packets like this using a FlowMod.
- *PacketOut*
  This message is sent from the controller to forward a packet, which is the payload of the PacketOut message, to some ports.
- *FlowMod*

This message set, delete or modify a rule inside a switch.This is the message that allows routing.

Forwarding table

- `match fields`: fields that compare packets in order to look for correspondences. They contain the in_port, the packet's header and others optional fields;
- `priority`: this field is used when there are rows with more than one correspondence;
- `counter`: this field was improved every time a correspondence with a packet is matched;
- `istruction`: a set of actions that the switch add on *Action-set* of the matched packet or some instruction in order to modify the pipeline flux;
- `timeout`: this is the maximum time afther which the rule expire;
- `flag`: with this field we can differenciate similar voices in order to managed multiple table with similar rules;

Match Constraints these are the principal matching method that we can use with OpenFlow v1.0:

- `in_port`: the port of swith;
- `dl_src/dl_dst`: 802.3 MAC source and destination of packet;
- `dl_type`: 802.3 EtherType;
- `nw_tos`: IPv4 ToS;
- `nw_proto`: IPv4 Protocol;
- `nw_src/nw_dst`: IPv4 Source/Destination addresses;
- `tp_src/tp_dst`: TCP/UDP Source/Destination ports;

Action With OpenFlow v1.0 we can do basic tasks in practice, the most important are for example:

- `Output`: set output port;
- `SetDLSrc/Dst`: modify packet's MAC address;
- `SetNWSrc/Dst`: modify packet's IPv4 address;
- `SetNWTos`: change ToS fields of message;
- `SetTPSrc/Dst`

From OpenFlow v1.1 on we can modify Time To Live and other options that allow us to do something more complex.

B. Known Limitations

SDN networks are known to be underperforming with low traffic, which means that if nodes tries to connect with each other to exchange little traffic, the network will have a higher delay and will perform worst than a traditional network. We tried to reduce this underperformance with the usage of the wildcards: Every rule needs at least the source ip or the destination ip. If one of them is in a network (e.g. the internet traffic is in the network 0.0.0.0/0 excluded 192.168.10.0/24), we can redirect all the traffic from an internal host to the internet with a predefined path. This means that only one PacketIn will be sent to the controller if a host is surfing the network.

Another limitation is the no default routing from a source to a destination: when an host is plugged to a network of SDN switches, it won't know the destination. The only way it can discover the host is by flooding the network with that packet, but this makes each switch raise a PacketIn message to the controller slowing down the startup of the connection. This thing doesn't happen with traditional routing, because each router knows the network disposition, but doesn't know each host how much traffic is doing. For that the administrator need to put a firewall. The last one isn't needed, as the controller can set a switch to drop all the packets from/to a destination.

C. Related Work

The work presented in "Efficient topology discovery in OpenFlow-based Software Defined Networks" by F. Pakzad has been implemented to improve the controller speed and efficiency as it make it more reactive [11].
The introduction to the statistics power in a SDN network using the OpenFlow protocol has been found in the paper called "Getting traffic statistics from network devices in an SDN environment using OpenFlow" by D.J. Hamad & others [14].
For traffic analysis we haven't found any paper describing the characteristics of some types of flows, so we used our PCs to sniff the traffic.
Traffic analysis will be discussed in section IV-D.

II. Needs and Goals

The main purpose of this experience is to develop a small SDN network in a lab with physical hardware in order to understand how to improve a Quality of Service in a SDN scenery with a real scenario. Our network have to manage traffic with a global purpose, in order to evolve according to different parameters.

Moreover the SDN network have to efficiently interact with the world wide web.

A. Connectivity

First thing to do is provide connectivity, letting the controller know the connection of a host and avoiding packet to be lost while the first add the computer to its graphes and try to add the rules to let the traffic go from the source to the destination. This has been done doing a *controlled flooding*: when a PacketIn event arrives at the controller, this look into the data structure that saves the nodes connected.
If source and destination nodes are already found, it set the flow rules from the source node' switch to the destination node switch. To the latter, a PacketOut message is prepared with inside the packet arrived inside the PacketIn event and with destination the port of the destination switch connected to the destination node. Then the message is sent to the destination switch, so that the first packet is not lost. If, instead, the destination node is not found, the controller sends to the switch that generated the PacketIn event a PacketOut message with destination all the ports of the switch itself and with payload, the message that arrived within the PacketIn message. To avoid

the network to be filled of the only packet sent, we kept the information regarding the switches on which the packet that was flooded: if a PacketIn message arrives at the controller and both the packet and the switch got a PacketOut message within 2 seconds, the controller doesn't do anything, as this is a loop.
So this approach allowed us to make the network unaware that it's not a legacy one and allowed connectivity to be provided.

B. Adaptability of the network

An adaptable Network has to real-time reroute traffic in order to ensure a lot of demands. A general efficient Network has to manage some tasks, for example avoiding links congestion, reroute traffic when some malfunction accure on a link and running a link path algorithm. In addition our network has to be adaptable in order to ensure a QoS type of service.

To reach this gools we needed to have a global knowledge on the network and we had to modify the routing table on every routerboard with a global purpose. The performance of a single connection and entire network can ensure various type of metrics. In order to ensure the traffic shaping based on network informations, there is a need of an automatically updatable structure, where the link statistics can be saved to allow the best path for a data stream. The best metric will be described below.

C. Exterior comunication

Our network needed to be able to communicate with the external internet to be useful for a demo which demonstrate the capabilities of a SDN network. The first thing we needed to accomplish was to make the controller aware of what switch and which port of the switch is connected to the external internet: to do so, we allowed our controller to be able to receive and process in-line parameters. We provided two parametes to the controller, so that it could recognize where the traffic to the internet need to be sent. Theese two parameters were the mac address of the port of a switch that is connected to the link and the network address of our network, to let the controller know if a request from a host was meant to another host within the network or the data stream was intended to the internet. We provided only one gateway, as the Routeboard had few ports available for the packet switching and only four SDN-capable switches to be able to simulate a real network that was generating traffic.

III. Instrument

A. Controller

The controller that we use is POX, the software is opensource and it is written with Python, based on SDN and supports OpenFlow. The software uses the version 2.7 of Python. It's repository, together with lot of documentation regarding the project can be found at these links https://github.com/noxrepo/pox, https://openflow.stanford.edu/display/ONL/POX+Wiki. The only problem in using this controller is that it supports only the first version of OpenFlow1.0.
The controller runs inside a VirtualMachine on our personal computer and the VM is offered by SDN hub on this site: http://sdnhub.org/tutorials/sdn-tutorial-vm/. This VM is a 64-bit Ubuntu 14.04 that has a number of SDN software and tools installed.

B. Router Board

Figure 2: *Experience scenario*

The router board that we used is MikroTik miniROUTER RouterBOARD 450G. This Router Board virtualizes an OpenFlow switch that supports OpenFlow v1.0 [9].

- CPU AR7161 680MHz
- Memory 256MB DDR SDRAM onboard memory
- Data storage 512MB onboard NAND memory chip, microSD card slot (on reverse)
- Ethernet Three 10/100/1000 Mbit/s Gigabit Ethernet ports supporting Auto-MDI/X
- Serial port One DB9 RS232C asynchronous serial port

The **Operating System** that they use is MikroTik RouterOS [10].
One limit of this routerboard is that it has only 5 ethernet ports and this is a limit because one of these ports must be reserved for the connection with the controller, that is why we have to connect both host and other routerboards using only 4 ports.

IV. Results

A. Controller

We developed a controller that can handle a mesh network with loops, avoiding packets flooding. The controller stores the switch and the connection between them and knows which hosts are connected to each switch port starting from the hypothesys that only one host is connected to each port, unless the port is the one connected to the internet. With the use of statistics, the controller gets the status of the network and, if necessary, tries to adjust the traffic and to put in the link with the higher path loss the torrent traffic. On the other hand it sets up the connection for two gaming hosts going through a link with less delay and tries to redistribute all the traffic in a way that all the nodes have an average throughput as lowest as possible.

We have used some principal functions released with POX controller:

- `pox.openflow.discovery.launch()` this function runs an algorithm in order to discover the network. Every time a new topological change occurs this function returns an event on POX with which we can update our topological knowledge. It does not provide the host event [1].

- `pox.openflow.spanning_tree.launch()` this function is necessary with a presence of loop on topology because with a spanning_tree algorithm it avoids the broadcast storm of messages.
  When we set up this option `OFPP_FLOOD` like output port's action in a table's switch row if some spanning_tree algorithm is working the `OFPP_FLOOD` is only theoretical in the sense that the algorithm excludes some ports during the flooding option, in order to avoid loop [1].

B. *Real Lab Scenario*

The topological scenario in which we have set up experience is as follows:

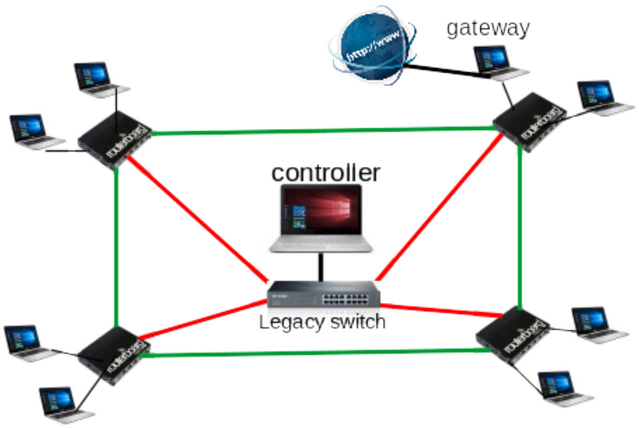

Figure 3: *Experience scenario*

We have use 4 MikroTik Routerboards that were provided us by the Communication's Network Lab in the department of information engineering to University of Padua. The 4 routerboards are connected each other in order to design a topological loop. Then every sigle board connects two hosts and is connected with a legacy switch which provides logically connection with controller. The last thing is a host that made gateway's function; we have connected its with both routerboard and legacy network and set up this host like default gateway on the controller.

C. *Graph and Network's structures*

In order to achieve the realtime adaptability and statistics storage we developed ad-hoc structures. We used a specific Python's library called NetworkX [6]. NetworkX allow us to create a graph in witch we can storage suitable nodes that represent resplectively switches or hosts. This topological structure is updated live thanks to connection events and links discover offered by POX's libraries:

- `_handle_LinkEvent`: from openflow.discovery an event of this type gives us two types of information; a new link is added and a still present link is expired;
- `_handle_ConnectionUp`: from core.openflow this event joins up when a new switch is added on the network and gives us some information of the switch, in particular its ports configuration;
- `_handle_ConnectionUp`: from core.openflow like the event explained above comes up when a switch is disconnected from the network;
- `_handle_port_status`: from core.openflow this event occures when on a particular switch there is a change about its ports, for example a new host has just been connected or the link is expired;

The link on our network besides has many parameters in witch we can storage network's performance. In order to update every link's parameters we had to create ad-hoc functionality since there are not proper functions constructed before.

This type of structure allow us to run Dijkstra's polynomial algorithm that discover the shortest path according to various parameters.

The following picture shows the visual output of our laboratory network's topology offered by the controller, we can see 4 different switches and all of the other hosts differentiated with their IP addresses.

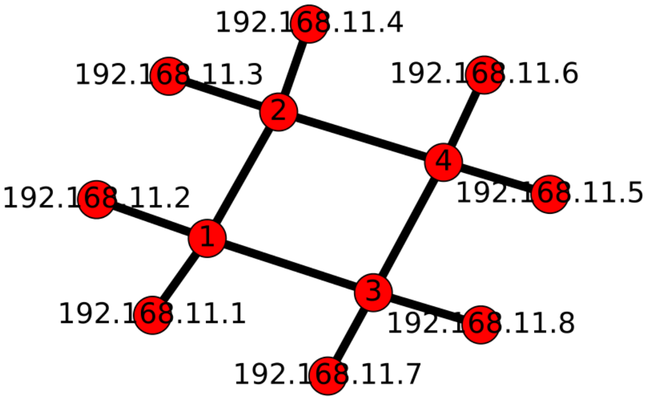

Figure 4: *Experience scenario*

D. *Traffic's Analysis*

We used laptop connected via GigabitEthernet interface and Ethernet cable cat.5a to receive the traffic from the destination host. Our laptop has been connected via Wi-Fi to our access point via protocol 802.11n. No other traffic sources were active during the testing, so the router was handling only the traffic from laptop when sniffing traffic. The connection used was an ADSL2+ via copper. We did 3 type of traffic analysis: Gaming traffic, Skype® and torrent traffic. We analyzed #Mb for the gaming traffic, #Mb for the VoIP traffic and 3GB for the torrent traffic. The analysis involved the average number of packets per second, the packet size, the protocol (TCP or UDP).

The traffic analysis of the torrent protocol shows that the average packet size is the MTU (i.e. 1500bytes) and the protocol tryies to exploit all the channel capabilities. The traffic is made via UDP packets with the port range in the ephimeral ports (2000_- 65535). Moreover, the node is linked to a lot of IPs with multiple connection: this means that if someone wants to find a p2p traffic, it can look to the statistics and see if the traffic is loading all the channel, to how many IPs, an host is connected to and if it uses UDP. For the VoIP we saw that the protocol uses UDP and the traffic, after a transition time, goes only between two hosts. The average packet size is the MTU and the protocol tryies to exploit all the channel for a better QoE. On another test, we tried the VoIP while a download was ongoing, and we saw that the quality decreased to allow communication. The traffic is between the UDP port 80.

For every networks performance we have tryed to develop a specific solution. The OpenFlow main goal is to centralize controll and software complexity so the routerboards can made only some easy function in order to contain the hardware complexity and mantainly a good speed [12]. This led to the necessity of stratagems utilization.

1) Delay: Monitoring round-trip time provides important insights for network troubleshooting and traffic engineering. The common monitoring technique is to actively send probe packets from selected vantage points hosts or middleboxes. In traditional networks, the control over the network routing is limited, making it impossible to monitor every selected path. The emerging concept of Software Defined Networking simplifies network control. However, OpenFlow, the common SDN protocol, does not support RTT monitoring as part of its specification. In this experience we have tryed to emply an tecnic to escape this problem [13].
The idea is to measure the time that occure from sending a packet from controller to a switch and the time in which the packet return from an other switch.

First we have to discovery round trip time from controller to every single switch on our topology. In order to do this on the controller we start a timer, we send a proper packet to the switch specifing the `OFPP_CONTROLLER - Send to the controller` option and when the message return to the controller we stop the timer and calculare RTT for the switch.

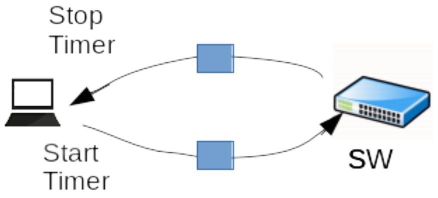

Figure 5: *RTT between controller and OF switch*

If we need to calculate the delay on a specific link, the rtt has to calculate with the headed switch of this link.

Now we create a proper message with unique id. number, let's start a timer `TM1` and then send this message to a first switch specifing the output port. Then when the packet comes to the second switch it identify the particular message and sends to the controller that stops it's dedicated timer `TM2`.
This part required a particolar flow entry on the second switch in order to send to the controller only some particulare packet.

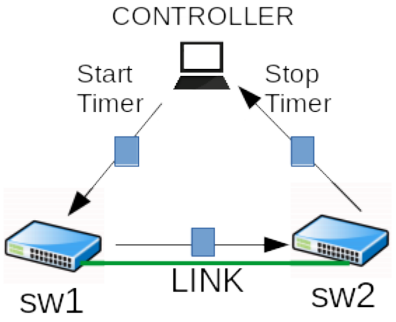

Figure 6: *Pck from controller to itself following a particular way.*

At this point we can calculate the delay of the link.

delay = (TM2 - TM1) - RTT1/2 - RTT2/2

Teorethically this method must works but in practice there are a lot of delay introduced from computational congestion on the virtualized controller. What we observe is a estimated delay like a few seconds which is not the case, infact if we made some `ping test` from one host to another host in this SDN network we realize delay around 3ms.

2) Link Capacity: The first data that was collected was the actual link capacity, which depends both on the cable used to connect different devices and the effective capacity for each ethernet port linked by the cable itself. Everytime a device supporting openflow connects to the controller, it gives to the latter information about its status and its capabilities. One of them is the link capacity in three ways: the first is the *supported* capacity, which is the phisical maximum capacity, the second is the *advised* capacity, which name describes itself and the latter is the *current* one. We choose to save the last one, as it gives the most useful about the link state. This information, moreover, is the one that non-SDN switches use to choose the *metric* parameter. The simulation with Mininet showed us that this type of information can be used, but the Routeboards didn't support it. As a matter of fact, we weren't able to exploit this capacity in our network realization.

3) Throughput: Using the flow statistics, we analyzed the traffic going through a link and, with a simple division, we got the normalized throughput. We exploited the information given by the switches with the match structure: this means that we used the throughput for the TCP traffic, the one for the UDP and for each of them we could know toward which IP the traffic is directed or is from. This was exploited with mininet simulations, but not in the set up in the laboratory. With the latter, we used the statistics as an indicator of the bitrate of a link and divided the host to categories based on it.

4) Packet error rate: The port statistics given by the switches contains different informations about the state of each port in a switch. To classify a link for its reliability, we divided the number of packets with erroneous symbols that couldn't be corrected with a FEC by the total number of packets that were sent, getting the packet error rate. The choice we made when exploiting this statistic was privileging the gaming and normal traffic, wherease the heavy traffic was directed to links with highest packet error rate. As the switches exchanges a packet before recognizing most of the error, both because the phisical error are corrected using blocks coding and other error are found using a *Frame Check Sequence* like CRC, the byte error were considered worse as those are indirect measures themself and the error derived from the calculations could grow up.

E. Route changing

For each statistic we got and each parameter we obtained, we changed the weight of the graphs. Every graph we used had a specific weight, so the Dijkstra algorithm for a weighted graph gave different minimum path between two hosts. From the parameter we saved, we choose to change the route to reach the destination and viceversa. The traffic follow the same path in both the directions and depends only on the type of traffic: we choose to set the minimum unweighted path if no statistics were available, the minimum throughput path for the standard traffic, the minimum delay for the gaming one and the maximum packet error rate for the heavy traffic generated by a host. Two threshold were chosen based on the traffic analysis described on Sec. IV-D which are 200 bytes/s for the gaming traffic and one megabyte/s when the node is considered in heavy traffic. Those two threshold were made as they are average values and considering that our network is made of few really fast link, so on one side the lower limit is sufficient not to consider gaming traffic to be a normal traffic, wherease it's easy to reach the high level of traffic load between two machines. The changing of the path were made only if the last change happened before one minute because the torrent traffic has, in its protocol, a discovery algortithm. This means that if a lot of changes were made, most of the traffic generated by the peer to peer, is the control one and the result is that the load in the network will be heavy and last longer.

F. ARP solution

During our analysis and in our tests, we prove that in a SDN mesh network, the ARP packets are useless. This type of packets is useful for a multipoint to multipoint connection where the connection is set between a HUB and a set of computers or between a bridge and a computer. The first case is no more in use in common networks as it higher the collision range. The bridges limits their collision range to the port. With openflow each switching unit can act as a bridge or as a router. In both cases analyses the header of the L2 packet L3 packet and some of the L4 header. If it has the rule to forward the packet, it will act as declared in the most specific matching rule. Otherwise it will send to the controller a PacketIn and the controller will act as programmed. If an host want to communicate to another host whithin the network, it sends the packet to the line and then the switch will check in its tables and execute the actions of the match found. For an IP network, it will check the path to reach a destination and then send to the defined output port. The mac address in this case became useless, as the connection is point to point. So, knowing the destination layer2 address is useless as the switch can act regardless of it. The only thing the switch need to know is the layer2, layer 3 addresses and, at most, the layer 4 address. This can be seen by a reader as a violation of the layers in the ISO/OSI stack, but to exploit the potentials of the SDN approach, this is necessary.

G. Default Gateway

Analyzing the traffic while we were trying the controller on real machines in the lab, we have tried to set the default gateway to other machines that were connected to the SDN network, but not to the internet. We proved that the controller can redirect correctly the traffic to the internet. This means that computers don't need anymore to set a gateway to reach another network as the controller and the switches will redirect the traffic properly, according to the rules set by the first. The controller must only know which switch ports are connected to the internet and then add them to the path. I this way, the controller can set a path to reach it and provide connectivity with a traffic shaping policy and/or differentiate the traffic from and to different networks.

H. Ip switching in pox

During our tests, both with Mininet and in the lab, we found out that the core of POX, when a PacketIn event occoures, switches the source and the destination addresses, both MAC and IP, if one of the hosts is already known. We weren't able to understand in which point of the code of POX and why it happens. The problem was partially solved, checking out the parameters known by the controller, i.e. if the two host are connected to two different switches and the controller switches the source with the destination address, we can switch it back; the result is that the PacketIn event now has correct informations. If the two host are in the same switch, the problem is solved using a controlled packet flooding, described in section II-A.

V. Conclusions

Conclusions are a superbrief summary of what has been done and highlighting of the "take home message" ehi #TODO

References

[1] POX https://openflow.stanford.edu/display/ONL/POX+Wiki

[2] OpenFlow http://flowgrammable.org/sdn/openflow/

[3] POX's repository https://github.com/noxrepo/pox

[4] Virtual Machine http://sdnhub.org/tutorials/pox/

[5] Router Board https://www.senetic.it/product/RB450G

[6] NetworkX python's library https://networkx.github.io/

[7] Open Networking Foundation https://www.opennetworking.org/sdn-resources/technical-library

[8] v1.5.1 OpenFlow report https://www.opennetworking.org/sdn-resources/technical-library

[9] RouterBOARD 450G MikroTik documentation http://www.routerboard.sk/files/pdf/rb450g_manual.pdf

[10] RouterBOARD MikroTik Software https://mikrotik.com/software

[11] Efficient Topology Discovery in Software Defined Networks F. Pakzad, M. Portmann, W. L. Tan and J. Indulska School of ITEE, The University of Queensland Brisbane, Australia https://www.researchgate.net/publication/271508903_Efficient_Topology_Discovery_in_OpenFlow-based_Software_Defined_Networks

[12] OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks Niels L. M. van Adrichem, Christian Doerr and Fernando A. Kuipers Network Architectures and Services, Delft University of Technology Mekelweg 4, 2628 CD Delft, The Netherlands https://pdfs.semanticscholar.org/ea94/7a9334484ee4c3bb6bcfb4d1590aece1b178.pdf

[13] Efficient Round-Trip Time Monitoring in OpenFlow Networks A. Atary A. Bremler-Barr Interdisciplinary Center Herzliya http://www.deepness-lab.org/pubs/infocom2016_grami.pdf

[14] Getting traffic statistics from network devices in an SDN environment using OpenFlow Diyar Jamal Hamad , Khirota Gorgees Yalda, and Ibrahim Tanner Okumus http://itas2015.iitp.ru/pdf/1570195931.pdf;