Soft MUD

Implementing Manufacturer Usage Descriptions on OpenFlow SDN Switches

Mudumbai Ranganathan, Doug Montgomery, Omar El Mimouni
Advanced Networking Technologies Division
National Institute of Standards and Technology
Gaithersburg, Maryland, USA
E-mails: {mranga, dougm, omarilias.elmimouni}@nist.gov

***Abstract* -- A Manufacturer Usage Description (MUD) is a generalized network Access Control List that allows manufacturers to declare intended communication patterns for devices. Such devices are restricted to only communicate in the manner intended by the manufacturer, thus reducing their potential to launch Distributed Denial of Service attacks. We present a scalable implementation of the MUD standard on OpenFlow-enabled Software Defined Networking switches**.

Keywords- IOT; MUD; Network Access Control.

I. INTRODUCTION

Internet of Things (IoT) devices (henceforth called "devices") are special purpose devices that have dedicated functions. Such devices typically have communication requirements that are known to the device manufacturer. For example, printer might have the following requirement: Allow access for the printer (LPT) port, local access on port 80 (HTTP) and deny all other access. Thus, anyone can print to the printer, but local access would be required for the management interface which runs on port 80 as a web server. All other access would be in violation of the intended use of the device. The idea behind the Manufacturer Usage Description (MUD) [1] is to declare the intended communication pattern to the network infrastructure using a generalized network Access Control List (ACL) which is specified by the manufacturer, the integrator or the deployer of the device. These are realized as network access controls, by which the device can be constrained to the intended communication patterns.

MUD provides an effective defense against malicious agents taking control of the device and subsequently using it to launch attacks against the network infrastructure. It can also prevent compromised devices from attacking other devices on the network. Thus, MUD substantially reduces the threat surface on a device to those communications intended by the manufacturer.

Because the manufacturer cannot know deployment parameters of devices such as device IP addresses and IP addresses of device controllers, MUD defines class abstractions, using which, the MUD ACLs are defined. For example, the manufacturer may state an intent that devices can only communicate with other devices on the local network, or may state an intent that devices may only communicate with other devices made by the same manufacturer, or that devices may communicate with other devices made by a specific manufacturer on a defined port, or that devices may communicate with specific internet hosts or combinations of the behaviors above. To enable such generality, ACLs are defined with placeholders known as *classes*. These place holders are associated with Media Access Control (MAC) or IP addresses when the ACL is deployed on the switch.

In brief, the system works as follows: A device is associated with a MUD URL. The MUD URL is a locator for the MUD ACL file. The MUD server fetches the MUD file for the device from the manufacturer site, verifies its signature and installs network access controls using whatever mechanism the network switches and firewalls provide. There are several mechanisms that may be available for enforcing access control; for example, iptables could be used or the switch may already support an implementation of network ACLs.

In this paper, we describe a scalable design and implementation of the MUD standard on OpenFlow 1.5 [2] capable Software Defined Network (SDN) switches. An OpenFlow switch supports flow rules that are logically arranged in one or more flow tables in the data plane. The switch connects to one or more controllers that can install flow rules in the switch either reactively, when a packet is seen at the controller, or proactively when the switch connects to the controller. Flow rules have a MATCH part and an ACTION part. The MATCH part can match on different parts of the IP and TCP headers. The ACTION part forwards or drops the packet or sends it to the next table. As packets hit flow rules, metadata can be associated with the packet to provide a limited amount of state as packet processing proceeds from one table to the next. More details are found in [2].

In related work, Hamza et al. [3] consider how MUD may be used with real-world devices to build an Intrusion Detection System (IDS). They present a simulation based on captured trace data. Details on how to organize flow tables to implement MUD are not presented in their work. The focus of our work is different; in our work, we describe how to implement MUD and demonstrate that it can be done in a scalable fashion.

 ISBN: 978-1-61208-695-8

The rest of this paper is organized as follows: In Section II, we outline our design; Section III provides an analysis of our design; Section IV describes our implementation; Section V presents emulation results followed by Section VI which gives measurement on a commercially available home/small business router.

Note that certain commercial equipment, instruments, or materials are identified in this paper to foster understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

II. DESIGN SKETCH

MUD Access Control Entries (ACEs) can be divided into two categories – those that define intent for communication between a device and a named host, and those that define intent for communication between a device and other classes of devices. The former kind presents no scalability challenges and can be easily implemented using MAC and destination IP address match rules. The main challenge with MUD arises when implementing ACEs that define intent for communication between classes of devices or between the device and hosts on the local network.

Figure 1 shows an example "same-manufacturer" ACE. This indicates the intent that the device may communicate with other devices made by the same manufacturer.

```
{
  "name": "myman0-todev",
   "matches": {
         "ietf-mud:mud": {
           "same-manufacturer": [
             null
           ]
         }
        },
        "actions": {
          "forwarding": "accept"
        }
}
```

Figure 1. Example of a "same-manufacturer" ACE.

Similarly, an ACE can be set up that indicates that the device may communicate with other devices on the local network on a specific port. Such ACEs present scalability problems when naively implemented. For example, if the Same Manufacturer ACE were implemented as MAC to MAC flow rules, there can be $O(N^2)$ rules in the flow table (where N is the number of devices belonging to the manufacturer that are associated with the switch). This is unfeasible as an implementation strategy because switches may be limited in ternary content-addressable memory. Similarly, an explosion of rules will result if the Local Networks ACE were implemented in a single table using MAC address to destination IP match flow rules. We seek a solution that is memory scalable and operator friendly. We make the following assumptions:

- Device Identification: Devices are identified using their MAC addresses on the local network and are dynamically associated with MUD URLs at run-time.
- Flexibility: MAC addresses of devices that will be managed at a switch are not known to the network administrator a priori.
- Network Administration: The network administrator configures information about the network - such as the range of local addresses and the controller classes for the Domain Name System (DNS), Network Time Protocol (NTP) and Dynamic Host Configuration Protocol (DHCP) and device controller.

To achieve scalability and flexibility, ACEs are implemented using SDN flow rules in three flow tables. The source and destination MAC address are classified in the first two flow tables and metadata is associated with the packet. The third table implements the MUD ACEs with rules that stated in terms of the packet classification metadata that is assigned in the first two tables.

The flow pipeline is as shown in Figure 2, with the packet being finally sent to a table that implements L2Switch flow rules which is provided by another application.

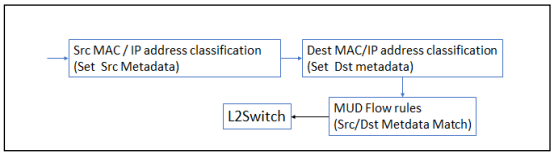

Figure 2. Flow Pipeline structure

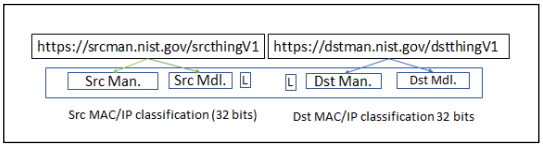

Figure 3. Source and destination metadata assignment.

The OpenFlow metadata field consists of 64 bits. We organize this as two 32-bit segments. Each 32-bit segment encodes a triple <*manufacturer, model, local-networks flag (L)* > as shown in Figure 3. The manufacturer and model are determined from the MUD URL and the local-network flag is determined from Source / Destination IP address and network configuration.

The packet classification metadata rules are reactively inserted when packets arrive at the switch as follows: When

the switch connects to the controller, the source and destination classification tables are initialized with low priority rules that unconditionally send IP packets up to the controller. This generates a *PacketIn* event at the controller which then inserts Source (or destination) MAC match rules that assign metadata to the packet and forward to the next table at a higher priority. Subsequent packets that match on the same MAC will have metadata associated with it and be forwarded to the next table without controller intervention. The next stage is to do the same for the destination MAC match rule.

The controller maintains a table associating MAC address with a MUD URI. This mapping is learned dynamically during DHCP processing, i.e., when the device sends out its own MUD URL when requesting an address (using the newly defined DHCP options 161) or it can be configured by the administrator for the device if DHCP support has not been implemented on the device. Each manufacturer (i.e., the "authority" portion of the MUD URL) and model (i.e., the entire MUD URL) is assigned a unique integer, which is placed in the metadata as shown in Figure 3. The controller also has knowledge of what constitutes a "local network" (typically the local subnet) which is assigned a bit in the metadata. If a MAC address does not have a MUD URL associated with it (e.g., a laptop) then it is assigned an implementation reserved metadata classification of UNCLASSIFIED that cannot be assigned to any real MUD URL.The next table implements the MUD ACEs. Note that at this stage of the pipeline, metadata has already been associated with the packet. MUD rules are implemented as ACCEPT rules. That is, if the metadata assigned to a packet matches the match part of the mud rule, it is sent to the next table.

Default unconditional high priority rules are initially inserted that allow interaction of the device with the reserved ports for DHCP and NTP. These are inserted into the MUD rule table on switch connect with the controller. The DHCP match rule has a "send to controller" Action part so that the controller may extract the MUD URL from the DHCP request if it exists. This enables the controller to associate a MUD URL with the source MAC address based on the DHCP request.

After the MUD URL is associated with the device, the MUD profile is retrieved by the SDN controller and flow rules that implement the MUD ACEs are inserted into the MUD table in the following order:

- High priority source (or destination) metadata and TCP Syn. flag match drop action rule to enforce TCP connection directionality. MUD ACEs can specify which end of a TCP connection is the initiator. For such MUD ACEs, we insert rules that drop packets where the connection is initiated from the wrong direction.
- Lower priority Rules that match on source metadata and destination IP addresses for access to specific named hosts or classes of hosts (e.g. Controller or my-controller) as specified by the MUD ACEs.
- Rules that match on source IP address and destination metadata for inbound packets to the IOT device as specified by the MUD ACEs.
- Rules that match on source and destination metadata for allowing access to manufacturer or model or local network classes as specified by the MUD ACEs.
- Lower priority Drop rule for packets that match on Source Model metadata but do not match on one of the rules above.
- Lower priority Drop rule for packets that match on Destination Model metadata but do not match on one of the higher priority rules above.
- Lower priority default UNCLASSIFIED packet pass through rule. The default MUD behavior allows all packets that are metadata tagged as UNCLASSIFIED to pass through the pipeline.

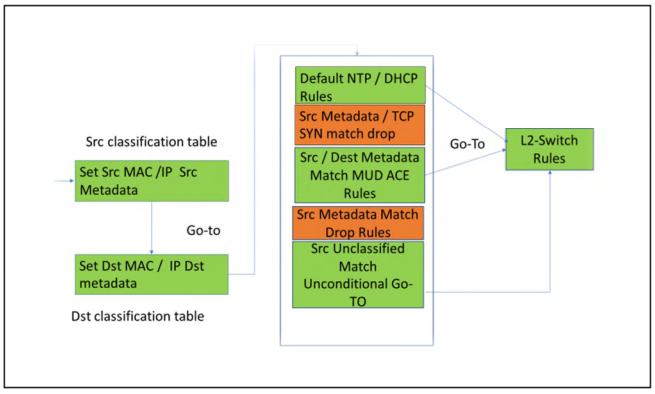

Figure 4. Detailed Flow Pipeline structure. The first two tables are classification tables which assign metadata. The third table is the MUD rules table. Drop rules are color coded Red.

III. ANALYSIS

The scheme we have described above is dynamic and memory scalable with O(N) rules for N distinct MAC addresses at switch. By dividing the rules into packet classification rules and MUD rules which are dependent on the metadata assigned on the first two tables, MUD rules can be installed independently of packet classification. The devices may appear at the switch prior to the MUD rules being installed or vice versa. This allows for dynamic configuration i.e. the MUD ACL table can be changed dynamically at run time without needing to re-configure the rules in the first two tables that classify the packets and vice versa. The packets may be initially marked as UNCLASSIFIED and later when they are associated with a MUD profile (using the DHCP or other mechanism outlined

 ISBN: 978-1-61208-695-8

in the MUD specification), the appropriate metadata is assigned to them.

Because the MUD ACEs are expected to be relatively static and few, the flows in the MUD rule table have hard timeouts to match the cache timeout in the MUD file. This can be in the order of days. The packet classification flow rules have short (configurable) idle timeouts. This limits the size of the table and allows for dynamic adjustment of the table when MAC addresses appear and disappear at the switch. The shorter the idle timeout for the classification rules, the less time it takes for reconfiguration and the less time it takes to purge the table from unreferenced entries. However, the shorter the timeout, the more overhead by way of communication with the controller due to the increased number of *PacketIn* events at the controller. We present experimental results in section V.

Our scheme, as described thus far, requires that a packet must be processed at the controller and a rule installed before packet processing may proceed. The initial rule in the MAC address classification stage that is installed when the switch connects, sends the packet to the controller but not to the next table. Thus, a packet may not proceed in the pipeline before it can be classified. This may be necessary if strict ACL-dictated behavior is required but there are some resultant performance consequences i.e., a disconnected or failed controller causes a switch failure because no packets from a newly arriving device can get through prior to the classification rule being installed.

To address this problem, we loosen up the interpretation of the ACE specification. We define a "relaxed" mode of operation where packets can proceed in the pipeline while classification flow rules are being installed. This may result in a few packets being allowed to proceed, in violation of the MUD ACEs with the condition that the system will become eventually compliant to the MUD ACEs.

To implement this behavior, the initial rule installed in the packet classification table with infinite timeout, allows the packet to proceed through the pipeline and delivers the packet to the controller simultaneously. If the controller is offline or fails during rule installation, the packet is sent to the next table with the initial rule and there is no disruption. When the controller comes online again, it will get a packet notification and install the appropriate rule – thus restoring MUD compliant behavior. Thus, the switch becomes resilient to controller failures, with the failure mode being to allow communication.

However, there is another source of potential disruption that must be addressed: Because the source and destination MAC addresses are classified using two tables, it is possible that the source MAC address classification rule exists in the table, while the destination MAC address classification rule has not yet been inserted into the next table. If the MUD rules have been inserted already, this will result in dropped packets in the MUD rules table until the destination table is populated, because the fall through action for Source MAC - classified packets that do not match a MUD ACE rule is to drop the packet.

We address this issue by defining reserved metadata classifications as follows:

- UNCLASSIFIED: The MAC address does not belong to any known MUD URL. For example, if the packet is emitted with a source address belonging to a laptop, for which no MUD rules exist, then its source MAC address is UNCLASSIFIED.
- UNKNOWN: The MAC address has been sent to the controller and is pending classification. The default rule that is installed when the switch connects to the controller sends the packet to the controller on IP match and stamps the packet with metadata of UNKNOWN.

The classification tables each have rules that send the packet up to the controller while setting the corresponding metadata (for source or destination MAC) to UNKNOWN and forwarding the packet to the next stage. The MUD rules table has rules that permits packets that are UNKNOWN in source or destination MAC classification to proceed to the next stage. The scheme is as shown in Figure 5.

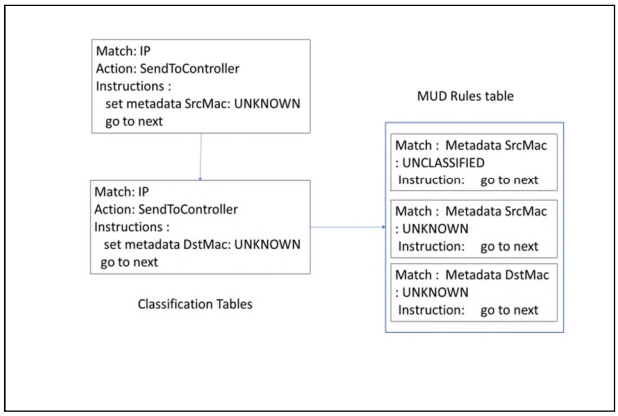

Figure 5. Temporary classifications label added to prevent blocking of flow pipeline during configuration.

On *PacketIn*, the controller pushes flow rules to correctly classify the packet. Because these packet classification rules are pushed at a higher priority than the default send to controller rule in the classification tables, the metadata will change from UNKOWN to the actual classification determined by the controller when the flow rule is installed. In the meanwhile, the pipeline is not blocked.

This "eventually compliant" mode of operation avoids packet drops and provides controller failure resiliency; however, there some limitations: (1) A few packets that violate the MUD rules could get through prior to the

 ISBN: 978-1-61208-695-8

classification rule being installed at the switch. This could result in a temporary violation of the ACEs. (2) TCP direction enforcement for short flows, which depends upon detection of TCP SYN flags and correct classification of MAC addresses, is not possible to enforce at the switch until a flow rule that classifies the packet is installed. We quantify these limitations in the next sections.

IV. IMPLEMENTATION

Our implementation [4] uses the OpenDaylight (ODL) SDN controller [5]. The configuration information for the system, which includes the MUD file and ACLs file are presented as north-bound API, are generated using the ODL YANG tools. The association between MUD URL and MAC can be configured directly or inferred by the controller by examining interactions between IOT devices and the DHCP server. For the performance measurement experiments, we directly configured the MAC to MUD URL association.

V. EMULATION EXPERIMENTS

To measure scalability of the implementation, our experimental scenario on MiniNet [6] consisted of 100 devices on one switch all belonging to the same manufacturer randomly exchanging messages. A device randomly picks another device and sends 10 pings, then sleeps randomly with an exponentially distributed average sleep time of 5 seconds. Our goal is to measure the memory scaling as the idle timeout of flow rules is altered. The following chart shows sum of the maximum number of rules in the source and destination classification tables for different values of the idle timeout.

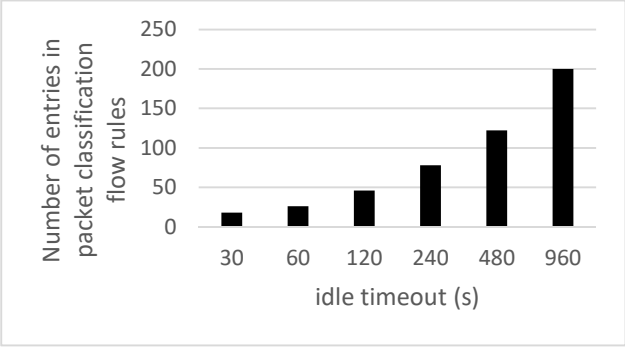

Figure 6. Packet Classification tables size variation with idle timeout. The MUD Rules table is a constant size and has infinite timeout.

In all cases, it is possible to implement the system with just a few rules in the classification table at the expense of an increasing number of *PacketIn* events processed at the controller.

To quantify the overhead involved with *PacketIn* processing under load, we measured the number of packets seen at the controller per burst of packets sent by varying the idle timeout settings for the packet classification rules. The results are shown in Figure 7. The maximum number of *PacketIn* events per burst of pings reaches a maximum of about 6 packets with the classification flow idle timeout set to 15 seconds – 6 packets are processed at the controller under these load conditions before the flow is pushed to the switch. The time it takes for the flow to appear at the switch is the window within which ACE violations can occur in the Relaxed ACL model and is hence significant. Measurements on an actual switch are presented next.

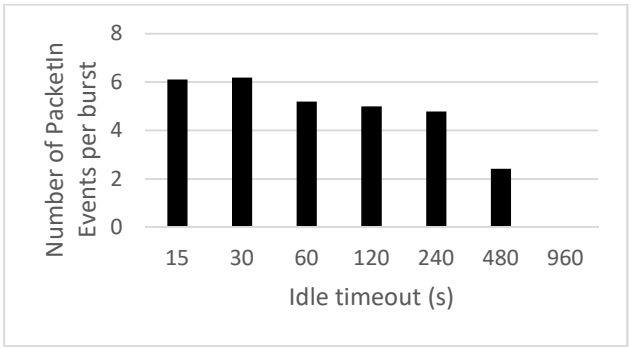

Figure 7. Maximum Number of packetIn events (observed at the controller) per burst of packets.

VI. MEASUREMENTS ON AN OMNIA TURRIS ROUTER

To measure how well our implementation would perform on commercially available hardware that may be a part of a home or small business, we tested our implementation an Omnia Turris [7] router that supports OpenVSwitch [8]. Raspberry Pi devices were used for load generation.

We installed a MUD Profile using the DHCP mechanism which allows the device to be accessed on port 80 from any machine on the local network. The device can access www.nist.gov on port 443. All other access is denied. The baseline performance of the router was measured. Relaxed ACLs were used to install the flow rules. Then, using iPerf [9], we measured the bandwidth with the MUD rules installed under different scenarios. This gives an indication of the overhead involved with MUD rule processing. As previously described, relaxed ACLs give us some advantages i.e., resilience to controller failures and reduced latency for packets that do not violate ACLs. However, packets may get through in violation of an ACL until the time a packet classification rule is pushed to the switch and appears in the switch table as a flow. How many packets get through before further communication is blocked? We used iPerf to perform an experiment where the device initiates an outbound connection with a peer on the local network and sends packets to it. As the "attempted bandwidth" is increased, more packets make it through the pipeline before being blocked, reaching a maximum of about 3 MB total leakage before the flow rules are applied, as shown in Figure 8.

 ISBN: 978-1-61208-695-8

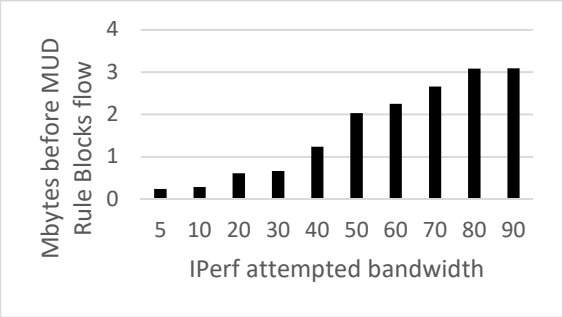

Figure 8. Relaxed ACL TCP packet leakage before ACL application. This is the amount of data that gets through before iperf stops.

Thus, if devices are expected to communicate infrequently and in short bursts, it is better to use the strict ACL model. Otherwise, it is possible that the communication may complete before the MUD ACE flow rules intervene.

Finally, we measured the overhead of strict ACLs on connection establishment. We initiate a connection from a local network resident device to a server accepting connections on port 80 on the MUD-compliant IOT device and measure the overhead in TCP connection establishment with and without relaxed ACL support over 100 attempts. The results are summarized in Table 1.

TABLE I. MAX TCP CONNECTION ESTABLISHMENT TIME FOR STRICT AND RELAXED ACLs

ACL Model	Max Connection establishment time (s)	Standard deviation (s)
Relaxed	0.002	.0003
Strict	2.0	.15

Significantly worse performance for strict ACL is caused by packets being dropped before flow rules are pushed. Dropping packets when the TCP connection is being established adversely impacts the connection establishment time. Note that this phenomenon only occurs when the rule is first installed because of the round trip to the controller before the installation of the rule.

VII. CONCLUSION

In this paper we presented the design and implementation of the MUD standard on OpenFlow switches, thereby demonstrating its implementation feasibility – even on limited memory devices. Our design is model driven, resilient to controller failure and allows for dynamic re-configuration. Our design uses $O(N)$ flow rules for N distinct MAC addresses seen at the switch.

An open question is how to set the idle timeout for the flows. Our timeout policy for the experiments described in this paper was to set the timeout the same for all devices – which makes sense in this case given a homogenous communication pattern with equal probability that a randomly selected pair of MAC addresses will communicate. In general, the communication between devices and between devices and its controller or host is not likely to be uniform. To achieve best utilization of the switch flow table memory, the idle timeout should be set high for MAC addresses that have a high probability of being referenced and set low for MAC addresses that have a low probability of being referenced [10]. It would be useful to extend the MUD standard to provide hints for communication frequency and length of communication burst for different MUD ACEs so that the controller can use this information to optimize timeouts and pick the appropriate management strategy on the classification flow table.

Our future work includes setting timeouts adaptively and combining MUD with an IDS to develop a comprehensive enterprise security architecture.

REFERENCES

[1] E. Lear, R. Droms, and D. Romascanu, “Manufacturer Usage Description Specification,” Internet Engineering Task Force Work in Progress, Jun. 2018. https://datatracker.ietf.org/doc/draft-ietf-opsawg-mud/ [Accessed: March 2019]

[2] Open Networking Foundation, “OpenFlow Switch Specification, Version 1.5.1 (Protocol version 0x06)”, https://www.opennetworking.org/software-defined-standards/specifications/ [Accessed: March 2019]

[3] A. Hamza, H. H. Gharakheili, and V. Sivaraman, “Combining MUD Policies with SDN for IoT Intrusion Detection,” in Proceedings of the 2018 Workshop on IoT Security and Privacy, 2018, pp. 1–7.

[4] nist-mud - NIST SDN MUD implementation, https://github.com/usnistgov/nist-mud [Accessed: March 2019]

[5] OpenDaylight, SDN Controller https://www.opendaylight.org [Accessed March, 2019]

[6] MiniNet - An instant Virtual Network on Your Laptop. http://www.mininet.org [Accessed: March 2019]

[7] Omnia Turris https://omnia.turris.cz/ [Accssed: March 2019]

[8] OpenVSwitch: Production Quality Multilayer, Open Virtual Switch., https://openvswitch.org Sep-2018. [Accessed: March, 2019]

[9] J. Dugan, S. Elliott, B. Mah, J. Poskanzer, and K. Prabhu, “iPerf - The ultimate speed test tool for TCP, UDP and SCTP.” https://iperf.fr [Accessed: March 2019]

[10] A. Vishnoi, R. Poddar, V. Mann, and S. Bhattacharya, “Effective switch memory management in OpenFlow networks,” Proc. 8th ACM Int. Conf. Distributed Event-Based Systems, 2014, pp. 177-188.

 ISBN: 978-1-61208-695-8