EasyWay: Simplifying and automating enterprise network management with SDN/OpenFlow

Alexander Shalimov
Applied Research Center for Computer Networks
Moscow State University
ashalimov@arccn.ru

Danila Morkovnik
Applied Research Center for Computer Networks
Moscow State University
dmorkovnik@arccn.ru

Sergey Nizovtsev
Applied Research Center for Computer Networks
Moscow State University
snizovtsev@arccn.ru

Ruslan Smeliansky
Applied Research Center for Computer Networks
Moscow State University
smel@arccn.ru

ABSTRACT

This paper presents new high level abstractions to manage the whole network at once: "names", "groups", "paths" instead of low level traditional IPs, subnets, routes. We called this approach as semantic network management where administrators simply say what they want from the network but not how and a SDN/OpenFlow controller will automatically program the network elements. All low level details are hidden from administrators (e.g., choosing IP addressing scheme). We implemented the proposed approach in a new SDN/OpenFlow-based network management system for enterprise networks called EasyWay.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.2.3 [**Network Operations**]: Network management, Network monitoring

General Terms

Algorithms, Design, Measurement, Performance

Keywords

SDN, OpenFlow, Network management system, Administration, Automation

1. INTRODUCTION

Modern enterprise network infrastructure is very complex: a lot of varied network elements, complicated topology, different routing and security policies. Network administrators are individuals that are mainly responsible for the maintenance of such networks including network monitoring and troubleshooting, configuring switches and routers, setting up new users and hosts. The main issues is that network administrators do all of these tasks by hands from black console terminals where they tediously print hundreds of commands! They still have limited number of tools for network debugging like ping, tcpdump, traceroute. Moreover, configuration APIs for network elements are different for every vendor (Cisco, Arista, NEC, Extreme, etc) that requires additional trainings.

CEE-SECR '14, October 23 - 24 2013, Moscow, Russian Federation.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2889-0/14/10 ...$15.00.
http://dx.doi.org/10.1145/2687233.2687251

To help network administrators there are several Network Management Systems (NMS) such HP OpenView [1], Cisco Prime [2], Open NMS [3]. They are designed to monitor activity of discovered network devices using SNMP protocol: topology, utilization, throughput, latency, and so on. But network administrator still need to configure networks devices manually and to add new rules into switching/routing tables by hand.

SDN/OpenFlow is the most revolutionary technology in the area of computer networks of recent years [4]. It allows us to configure flow tables for all network devices remotely from the logically centralized point called OpenFlow controller. In SDN/OpenFlow, we don't need to configure routes manually and to enter hundreds of commands — we just might specify what we want and OpenFlow controller will do the rest. This provides great possibilities to automate and simplify network management.

Despite the potential benefits, all existing OpenFlow controllers like Pox [5], FloodLight [6], OpenDaylight [7] have very pure low level interfaces to the network. They only display a topology with abilities to show and change flow tables remotely. Administrators should care about each single devices separately.

In this paper we introduce new high level abstractions to manage the whole network at once. The paper is structured as follows. Section 2 explains the proposed approach called semantic network management. Section 3 describes IP addressing scheme and contains IP choosing algorithm to reduce number of flows on switches. Section 4 describes implementation details of the system called EasyWay that is a noval SDN/OpenFlow-based Network management system. Section 5 shows the result of our experimental evaluation of the proposed algorithm and the implemented system.

2. PROPOSED APPROACH

2.1 Motivation

The key concept is to introduce higher levels of abstractions for network administration than existed now. These new abstractions should really move network administration towards to network management where we can simple control the whole network and have network operators instead of networks administrators. SDN/OpenFlow is *easier way* to make this step forward.

In SDN/OpenFlow, we don't need to configure routes manually and enter hundreds commands, specify OSPF[1] and VRRP[2] on top of RSTP[3], and so on. We should simple "draw" a needed path between network elements and an OpenFlow controller does the rest by sending appropriate commands to overseen network's switches. You even can just specify who can speak with whom, the controller will choose paths for you according to company's routing policies.

Others important responsibilities of network administrators that can be simplify and automated with SDN/OpenFlow are in an area of the network designing: establishing IP addressing scheme (dividing the network into subnets in order to structure network administration) and naming entities in the network (assigning names to the hosts, because it's much easier to work with understandable names).

In our approach, first of all, we specify names for all entities in the network: hosts , servers, switches, etc. We also let users to pick their host names. Then an operator can group hosts and specify the paths for the whole group. Based on groups and specified paths (or routing policies) we automatically choose IP addresses for hosts in order to reduce the number of entries into switches' flow tables. For example, for group of hosts that should have only an access to Internet, it makes sense to use the same subnet like 10.0.1.∗ that required only one rule in the core switch.

2.2 Semantic network management

To recap above mentioned ideas, the proposed approach to network management allows network operators to work with high level terms of "names", "groups", and "paths" instead of low level IPs, subnets, and routes.

- **NAMES.** Working with understandable names associated with the hosts is much easier for the operator than keeping in mind their IP or MAC addresses.
- **GROUPS.** Hosts might be combined into groups in case we need to specify the same policy for them. E.g. "the machines in the classroom have access only to the Internet".
- **PATHS.** Instead of configuring routes for each single device in the network, the operator can "draw" the paths between groups and hosts through the whole network specifying who can speak with whom.

[1]The Open Shortest Path First (OSPF) is a link-state routing protocol for IP networks.
[2]The Virtual Router Redundancy Protocol (VRRP) is a computer networking protocol that provides for automatic assignment of available IP routers to participating hosts (used for automatic default gateway selections on an IP subnetwork).
[3]The Rapid Spanning Tree Protocol (RSTP) is a network protocol that ensures a loop-free topology for any bridged Ethernet local area networks with faster convergence.

We called this approach as *semantic network management.*

3. IP CHOOSING PROBLEM

As was already explained above, administrators don't need to choose IP addresses or setup several DHCP servers. They specify groups that can be located in the different network segments. E.g. you might have two developers teams located in the different floors, but they still should have visibility of each other, and have an access to the same resources. With groups all restrictions should be specified ones, not several times for different segments.

This situation is kindly different from the traditional networks. The often-quoted "Yakov's Law" states "Addressing can follow topology or topology can follow addressing; choose one" âĂŞ– Y. Rekhter. In SDN/OpenFlow, you don't tight to topology, you can use any addressing scheme and get a lot of benefits from there.

In this section, we introduces two sequential ideas on what we can do with IP addressing scheme in semantic network management area. First, we present IP addressing scheme that works well with notion of groups. You can specify that the same host can be in several groups at the same time. The second idea is how to automatically choose what groups should have close IP addresses to get maximum benefits.

3.1 IP addressing scheme

OpenFlow allows to set not only specific values in the fields of flow tables, but also a set of values. This is obtained by utilization of a bit mask. Bit mask is a string of bits that is used for masking.

When combining several hosts in a group, we give them the IPs that can have common bit mask. In this case, instead of a single flow entry in the flow table for each host we can add only one flow entry with their common IP-bitmask.

In the proposed model, we use LAN with 10.0.0.0/8 IPs. IP address of each host in network is divided into four parts: group's pool (GP), group ID (GID), host's pool (HP), and host ID (HID) (see Figure 1). Group's pool is 4 bits long, group ID is 8 bits, host's pool is 6 bits, and host ID is the rest 6 bits.

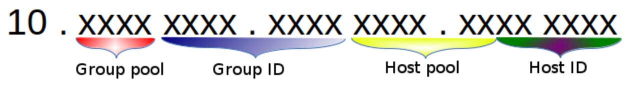

Figure 1: IP address notion

The above lengths are not hardly fixed. It is possible to change them depending on a network topology. For example, in a network with a large number of users with similar policies we can reduce the number of groups (GP and GID) and, respectively, increase the number of different users (HP and HID).

3.1.1 Requirements of the model

- All hosts in one group get the same group parameters (GP and GID).

- Different groups can not have the same group parameters.
- Groups that can be subsequently combined, should have the same group pool (GP).
- Each switch for some groups has unique host pool (HP).
- All hosts which have the same group and are connected to a switch must have the same host pool.
- Hosts which have the same group and connected to different switches must have different host pool.
- All hosts which have the same group and connected to some switches must have different host ID.

We can aggregate only those flow entries that have the same action, e.g. send packet to the same output port – *action: output(p)*.

3.1.2 Group pool and Group ID

Group ID is an identifier of a group in a group pool. We select 4 bits in address space for group pool. So we can use 16 group pools. We select 8 bits for group ID. Every bit corresponds to the same group: the first bit, that set in "1", corresponds to the first group, the fifth bit with "1" to the fifth group. So, in one group pool we can utilize only 8 groups. It was made in order to have the opportunity to mask different groups.

Why we need ability for masking of groups:

- This is the possibility to further reducing the number of flow entries in the flow tables. Flow entries on some switch for groups, which have the same GP value and action set, can be aggregated into one flow entry.
- This enables multi-group hosts, i.e. hosts that belong to several groups. Maximum simultaneous group value for host is defined by number of bits which was selected to GID parameter in IP address. Host may consist only in those groups that have the same pool.

The main purpose of utilizing the GP is to increase the maximum amount of groups. Without group pool parameter we can mask all groups together, but the maximum value of groups is $4 + 8 = 12$ groups. With GP parameter we have 16 non-overlapping sets of groups. In every set we have 8 groups. So maximum value of groups is $16 * 8 = 108$ groups. The main restriction of this method is the we can mask only the groups in the same group pool.

3.1.3 Host pool

Host pool allows to avoid extreme increase number of flow entries in "border-group" switches or edge switches. "Border-group" switch has several ports with hosts from the same group. Example of this case is shown in a figure 2. In particular, this happens when the same group is in several different network segments.

Assuming there is a certain routing policy from outside of this segment to Group-2 (figure 2). If we utilize (group parameters; host ID) addressing in this model, switch-A is not able to determine where to send received packets from an external network, because hosts with the same Group-2 is in two output ports: to switch-B and to switch-C. In this case we must add new flow entry on switch-A for every host from Group-2.

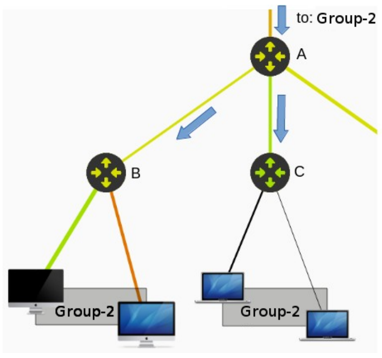

Figure 2: Host pool example

To prevent the increase of the amount of flow entries in flow tables we add additional parameter –âĂŞ host pool. In a case when elements of group are in different network segments, we allocate for them the unique values of host pool. When packet is received, the switch can determinate the next switch unambiguously by analyzing the IP address of destination.

Generally, the host pool of Group-X is number of switch to which hosts of Group-X are connected.

3.1.4 Host ID

This parameter determines the host in the specific group and in the specific host pool. In the same host pool there must not be the coinciding host IDs. Otherwise, several devices in the network will have identical IP.

Host IDs are assigned dynamically in sequence. If more than 63 devices with the same group are connected to one switch, the switch releases the second host pool.

3.2 IP distribution algorithm

In this section, we describe the algorithm which solves the problem of automatic choosing of IP addresses for hosts. This algorithm is based on IP addressing proposed in the previous section. The purpose is to minimize the number of flow entries in flow tables on switches. The task can be divided into two subtasks:

1. Hosts should receive such IP addresses so that we could select a general bitmask for them;
2. Combining groups in such group pools to maximize number of the aggregated flows at all pools.

3.2.1 Input Data

Proposed algorithm adds flows proactively, i.e. before the network functioning administrator initially sets combining hosts in groups and policies (with paths) between groups:

- G – union of groups, N – number of groups in one group pool;
- $g_k = \{H_{i_1}, \ldots, H_{i_n}\}$ – hosts belong to groups g_k;
- $g_m \to g_k$ via $\{K_{i_1}, \ldots, K_{i_n}\}$ – routing policy between groups g_m and g_k. Packets from group g_m to g_k will pass through switches $K_{i_1}, \ldots, K_{i_n}$ sequence. Note if

there is a policy $g_m \to g_k$, it means that there is a policy $g_k \to g_m$ via $\{K_{i_n}, \dots, K_{i_1}\}$.

Port p is called "out-port" for switch K_{i_k} and policy $g_m \to g_k$ via $\{K_{i_1}, \dots, K_{i_n}\}$, if switch K_{i_k} is connected with switch $K_{i_{k+1}}$ by port p. It means that a flow entry on switch K_{i_k} will be added with action $output(p)$ – "send packet to port p". Group g_m is called "source-group" for port p.

Based on policies created by the administrator the initial data sets will be prepared for calculating algorithm. Initial data sets have an appearance: $Set(j) = (\{g_{i_1}, \dots, g_{i_n}\}, k)$. Each initial data set (j) defines count (k) out-ports for all switches and the maximum union of source-groups for set is $g_{i_1}, \dots, g_{i_n}$.

Let:

- $Set(j).set = g_{i_1}, g_{i_n}$;
- $Set(j).ports = k$;

Define the function $Subset(p_k, Set(i))$ with input parameters:

- $p_k = \{g_1, , g_l\}$ is union of some groups;
- Set(i) is initial data set.

Function defines the reduction of flow entries on switches according to $Set(i)$, if we use group pool p_k. Function can be calculated as:

1. $counter = 0$;
2. $\forall g \in p_k$: if $g \in Set(i).set \to Increment(counter)$;
3. $return(counter - 1) * Set(j).ports$.

The task of distribution of groups is to find disjoint subsets of all groups (p_1, p_n), which are group pools, in order to maximize the following objective function:

$$F = \sum_{i=1}^{|G|/N} \sum_{j=1}^{|Set|} Subset(p_i, Set(j));$$

3.2.2 Steps

Algorithm is based on iterative search. On each step some group pool is determined as set which aggregate the largest number of flows. Algorithm is looking for local maximum on each step in objective function.

```
1. Define array $overlap[]$;
2. $i = 1; G_1 = G; TmpSet = \bigcup Set(j), \forall j$
3. Choose the group pool $p_i$. Select an arbitrary group $g_{new} \in G_i$ and add it into the pool $p_i$;
4. Analyze all initial data sets in TmpSet:
   (a) if $Set(j) \in TmpSet$ and $g_{new} \in Set(j).set$:
       i. $\forall g_k \in Set(j).set$ and $g_k \neq g_{new} \to overlap[k] = overlap[k] + Set(j).ports$
       ii. Delete $Set(j)$ from $TmpSet$;
5. In $TmpSet$ we choose the index $k$, for which $overlap[k]$ is maximum;
6. Add the group $g_k$ into the group pool $p_i$;
7. If the pool is not full:
   (a) $overlap[k] = 0; g_{new} = g_k$;
   (b) goto step 4;
8. $p_i$ is a full group pool with an optimal allocation of groups on step $i$;
9. $G_{i+1} = G/p_i$;
10. If $G_{i+1} \neq \emptyset$:
   (a) $TmpSet = \bigcup Set(j), \forall j$;
   (b) i = i + 1;
   (c) goto steo 3;
11. Algorithm successfully distributed groups into pools.
```

3.2.3 Algorithm analysis

The proposed algorithm has some bottlenecks. Firstly, it is not fast and now we can utilize it only proactively, because collecting the initial data sets ($\forall j\ Set(j)$) takes a long period of time. Secondly, at the moment the algorithm is not adjusted for the dynamic changes in the network topology. If the topology (switches, but not users) quickly changes, proposed distribution of groups into the group pools can become non optimal. In this case we should restart algorithm to enumerate distribution, and, therefore, hosts might get new IP addresses. Thus, we will have to rewrite the flow entries on the switches.

4. IMPLEMENTATION

4.1 EasyWay

We have implemented the proposed ideas to network management in the system called EasyWay. It works on top of Runos OpenFlow controller [8]. RUNOS is based on libfluid code base developed by CpqD for Open Networking Foundation [9]. All controller-side code is written on C++11. The GUI is written on HTML5 and JavaScript. From RUNOS point of view, Easyway is an network application that widely leverages different services like like device and link monitoring, topology, routing with QoS support, DNS proxy, DHCP proxy, ARP proxy, BGP, load balancing, firewall, ACL, NAT, and OpenFlow rules composer.

EasyWay is publicly available on https://github.com/ARCCN/easyway.

4.2 Features

Let us consider the following use case to demonstrate benefits of using EasyWay as a network management system. The demo company is middle multipurpose company with the following needs: it has a classroom where students should have an access only to training materials; company's guest should have access to the Internet via WiFi, but not to the internal resources; only HR department will have access to the database with personal data of employees; the boss will have guaranteed high bandwidth for video conference calls; load balancing of customers' requests along back-ends servers; restricting access from outside to the enterprise network (ACL). In traditional networks, it takes several weeks to setup and configure the network infrastructure with these policies. The proposed system allows to setup these requirements within 10 minutes.

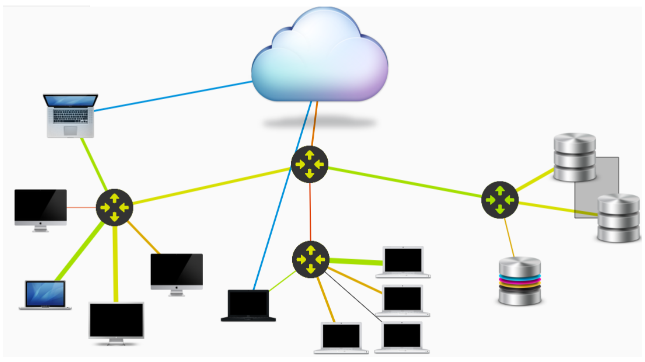
Figure 3: EasyWay example.

Figure 3 shows a sample view of EasyWay. The main features are:

- **Topology discovery.** The network map includes information about network devices (type, name, characteristics, etc) and their interconnections. It also detects host migration for both wired and wireless clients.
- **Network monitoring.** The system automatically monitors link utilization, rtt, load of network elements. The width of the links in the topology means the channels' bandwidth, the color indicates channel' load level (black - not in use, green - almost free, yellow - busy, red - critical).
- **Naming.** The operator can specify the name for end-user machines or let users to pick their names (DNS and DHCP agents).
- **Grouping.** The operator can group the hosts into one logical unit. Groups also have names. Group can be distributed over the network.
- **Path selection.** The operator can specify how the path should go or just specify the destination and the system will choose the actual path (based on shortest-path routing).
- **QoS support.** The operator can specify the needed QoS setting for the path. The path's width means the actual bandwidth dedicated to the path.
- **Load balancing.** The system supports flow balancing on OpenFlow switches. The feature can be enabled for every switch by selecting the LB item (load balancer) in the switch's drop down menu and then choosing the links for balancing.
- **Firewall/ACL.** The system allows the operator to specify ACL rules on OpenFlow switches with one of the prespecified rules like "allow HTTP traffic".
- **NAT.** The operator may enable NAT feature on a core switch and specify the public IP for address translation. We have implemented only subset of functionality required by different RFCs: only endpoint independent mapping and filtering with limited support of ALGs.

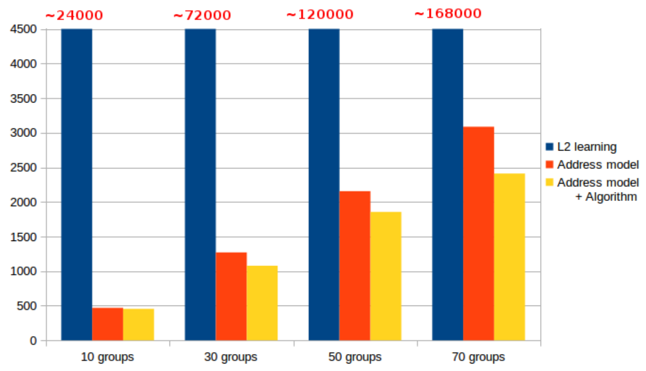

Figure 4: IP addresses algorithms

5. EXPERIMENTAL EVALUATION

5.1 IP addresses

We have three ways to fill flow tables in the switches:

- Traditional "L3 Learning Switch" mode can be used. In order to provide communications between hosts we must add new flow entry on all switches for each pair of hosts.
- To reduce number of flow entries we can use IP addressing and due to bitmask it is possible to aggregate different flow entries into one flow entry. We will consider IP addressing model proposed above. Utilizing this method we are masking IP source address on the border switches and IP source and destination addresses on core switches.
- The proposed algorithm for smart distribution groups into group pools can be used in addition to IP addressing. Due to this algorithm we can use masking not only for IP addressed of hosts in groups, but also for different groups in group pool.

When we apply the first method (L3 Learning), we should add flow entry for each pair of IP-addresses. We will assume Group-N contains $|G_n|$ hosts. If Group-1 can communicate with Group-2 we must create $|G_1| * |G_2|$ flow entries on each switch between them. If Group-1 communicates with set of groups it is necessary to use $\sum_{\forall i G_1 \rightarrow G_i} |G_i|$ instead $|G_2|$. It means that summary number of rules for communication of two groups is:

- $count\_of\_rules\_for\_Group - 1 = |G_1| * \sum_{\forall i G_1 \rightarrow G_i} |G_i| * count\_of\_hops$
- $summary\_count\_of\_rules = \sum_{\forall k} count\_of\_rules\_for\_Group - k$

When we apply the second and third methods, we have huge reduction of flow entries on core switches and relatively large reduction on border switches.

We consider network topology consisting from varied quantity of groups. Other parameters are fixed: each group consists of 20 hosts; each host belongs to only one group; each group (i.e. hosts in this group) can communicate with 2 other groups; the average number of hops between groups is 3 switches.

Figure 4 shows number of rules (flow entries) on all switches in the network. The abscissa shows the number of groups in the network. The ordinate shows the total number of rules. The proposed ways to IP addresses shows enormous benefits compared to simple distribution. The proposed algorithm is 20% than just picking up IP addresses according to the proposed scheme. Note, our IP addressing scheme is close to traditional IP addresses if we forget about groups in our model.

5.2 Network communication overhead

When we utilized the application to monitoring of the network, there is an additional (overhead) traffic and messages between switches and the controller. This overhead can be divided into three types:

- LLDP traffic and, respectively, PacketIn messages are received by the controller. This type of
- Overhead initiated by LinkDiscovery and Topology applications;
- ARP messages are initiated by switches to check host's presence in the network; Multipart messages are sent by the controller to switches to get port statistics;

LLDP traffic appears every 10 seconds. According to received PacketIn messages, controller understands actual network topology, i.e. switches and links between them:
– $overhead.LLDP = ((count\_of\_border\_sw) * 25 + (count\_of\_core\_sw) * 4)/10$. We assume average number of active ports on border switches is $(500/20) = 25$ and on core switches is 4.
– $overhead.PacketIn = ((count\_of\_border\_sw) * 1 + (count\_of\_core\_sw) * 4)/10$. Border switches receive only one LLDP message, because it is connected to one another switch.

Every 30 seconds border-switches sends ARP messages to hosts to get information about their presence. If host didn't reply to the ARP request, it means that host disconnected. Number of ARP messages doesn't depend on number of switches and can be calculated as: $overhead.ARP = ((count\_of\_hosts) * 2)/30$; // request and reply if the host exists

And every second controller requests statistics per port on all switches. So, controller sends multipart message (type is $OFPMP\_PORT\_STATS$) and gets the statistics on the switches. Analyzing this information, application can display actual congestion of the network. the Number of messages depends on the number of switches in the network topology: $overhead.Multipart = (count\_of\_switches) * 2$; //request and reply

We consider network topology consisting from the fixed quantity of hosts and varied quantity of core switches. In our topology we have 500 active hosts located in various network's segments and 20 border-switches.

Figure 5 shows packet's overhead arising every second in the network. In average, this is 200 OpenFlow messaged per second.

5.3 Performance

EasyWay works mostly proactively. If a new host appears, the administrators must specify group, nameas, additional policies, and the system proactively install all necessary rules down to the network. Also EasyWay works in

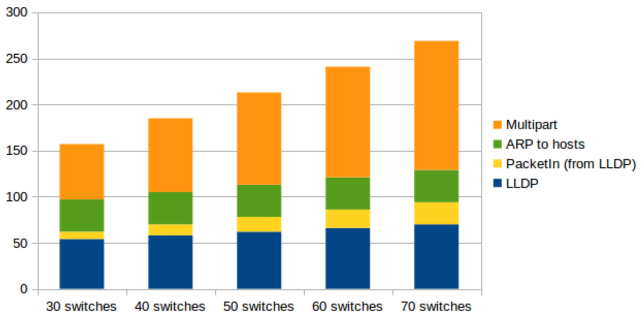

Figure 5: Overheads in the network

active mode in the following scenarios. Administrators can specify actions to apply for every new hosts appeared in the dedicated network segment. This works well in guests WiFi zone. Some applications works in the active mode: Load Balancer, NAT. Thus we need to know how fast our system in terms of throughput and latency.

Throughput (cps)	100K
Latency (us)	300

Table 1: The performance characteristics

Tables 1 shows the average numbers. Note this is not raw IO performance of RUNOS controller. This is an application performance running on a single core.

6. CONCLUSION

We introduce new high level abstractions to have control over the whole network at once calles semantic network management. All low level details are hidden from the eyes of network administrators. The paper shows benefits the enterprise can get having SDN networks. We have implemented the proposed ideas in a new SDN/OpenFlow-based network management system for enterprise networks called EasyWay.

From logically point of view our approach is very close to high level programming languages for SDN networks like Pyretic [10]. The differences is our goal was to simplify network administrating from the user graphical side with higher encoded functionality.

7. REFERENCES

[1] HP OpenView, http://openview.hp.com
[2] Cisco Prime Infrastructure, http://www.cisco.com/c/en/us/products/cloud-systems-management/prime-infrastructure/index.html
[3] Open NMS - Enterprise-Grade Network Management System, http://www.opennms.org/
[4] M. Casado, T. Koponen, D. Moon, S. Shenker. *Rethinking Packet Forwarding Hardware.* In Proc. of HotNets, 2008
[5] POX. http://www.noxrepo.org/pox/about-pox/
[6] Floodlight. www.projectfloodlight.org/floodlight/
[7] OpenDaylight. http://www.opendaylight.org/
[8] RUNOS. https://github.com/ARCCN/runos
[9] LibFluid. opennetworkingfoundation.github.io/libfluid
[10] J. Reich, C. Monsanto, N. Foster, J. Rexford, D. Walker. *Modular SDN Programming with Pyretic.*Communications Magazine, IEEE 38(5):128-134, 2013.