Toward Network Access Control With Software-Defined Networking

Dennis Gamayunov, Ivan Platonov, Ruslan Smeliansky
Applied Research Center for Computer Networks
Moscow, Russia
dgamayunov@arccn.ru, iplatonov@arccn.ru, rsmeliansky@arccn.ru

Abstract—**This paper addresses the network access control problem in dynamically changing network topologies using Software-Defined Networking (SDN). In this article we propose an approach to network infrastructure migration from traditional topology with dedicated firewall appliances to SDN L2 infrastructure, which implements the original filtering policy with OpenFlow switches and SDN controller only. The proposed approach allows maximizing throughput of the switched network preserving the reachability graph of the original network. Proposed algorithm was implemented as an application for POX controller and supports rule translation from Cisco ACL syntax. Experimental evaluation with physical SDN testbed built upon NEC PF series switches confirms applicability of the proposed method.**

Keywords-SDN; OpenFlow; firewall; network security; access control

I. Introduction

In the early days of basic Internet protocols development no native support for access control was provided at the network level. It was expected that applications would connect to each other in the global network without any restrictions. Along with the growth of commercial use of Internet mechanisms for L3 (and higher) network access control became necessary for normal operations, and packet filtering solutions were developed (including software implementations in operating systems) — firewalls, Intrusion Prevention Systems (IPS), network antiviruses, application layer proxy servers, including WAF — web application firewalls.

Unfortunately, nowadays with evolved networking technologies, enormous growth of Internet throughput and a shift from fixed client devices towards mobile networking (we have over 1 billion connected smartphones already in early 2013, and only about 200 million fixed devices) efficiency of existing access control solutions reduces. More expensive devices are required for every new version of Ethernet protocol providing the same level of network granularity as five or ten years ago. In terms of client devices mobility, network configuration is changing rapidly and the information about network topology changes could not be used directly for access control. That is why the problem of network access control based on the information about the expected behavior (flows) of network applications is becoming more and more important.

New SDN concept allows us not only to save the same network level access control functionality as traditional solutions may provide, but also to implement it much more efficiently. Let us consider a firewall, which implements access control between applications based on address information, port numbers (types of applications) and other service header fields. The use of complex rules is primarily caused by firewall installation in one certain point of network topology. The rules syntax should allow us to accurately distinguish the flows originating from different applications and clients. Because of the complexity and richness of the policy language the logic of firewalls becomes more complex; they need to perform more operations with each packet header to resolve which action to perform. At the same time it is known that source based filtering performed closer to the source node and destination based filtering closer to the destination node or application allows simplifying the policy rules and therefore making filtering logic cheaper.

In this paper we show that any given access control policy which specifies reachability matrix on a network graph (implemented in firewall configuration) may be implemented as an SDN Flow Policy which preserves the reachability matrix between applications and maximizes the throughput of the L2 networking infrastructure. We show that using SDN it is possible both to escape the necessity for dedicated hardware firewalls and to maximize the overall L2 network throughput. We will formally show how to solve the problem of migration from traditional network to SDN while preserving the nodes reachability (the connection matrix).

This paper is organized as follows. In section II we describe the proposed approach and analysis of its applicability. In section III we give quantitative results, which were obtained by probing our tool on an example of network topology. Section IV gives a survey of related works and section V closes the paper.

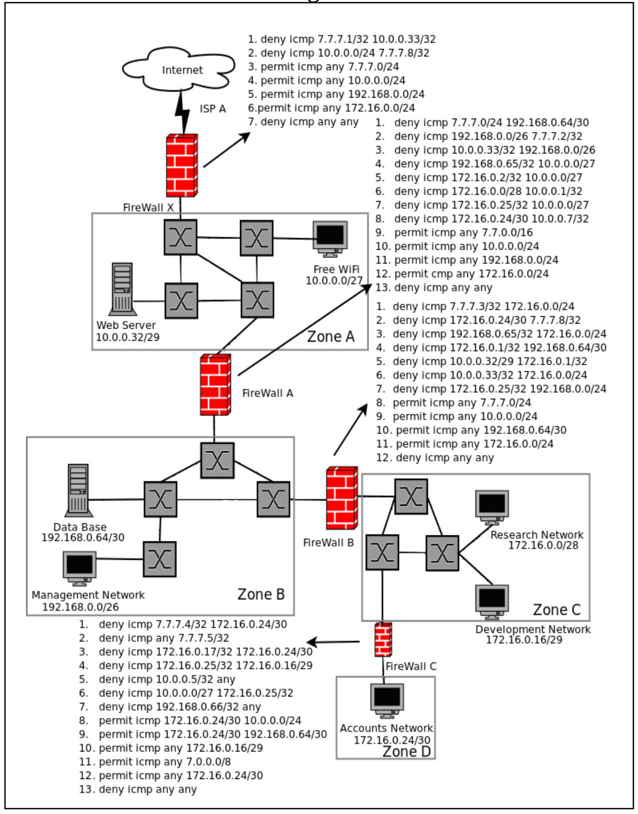
II. Proposed approach

The main purpose of this research is to investigate the possibility of firewall rules optimization in the process of migration from classic network architecture to its SDN implementation. The topology in Figure 1 was chosen as an example of our method evaluation. It consists of a large amount of subnets (Free WIFI, Web Server, Data Base, Management Network, Research Network, Development Network and Accounts Network), switches and firewalls. There is also a separate subnet called «Internet». We define it as a subnet with a network address that does not intersect with any address of the subnets above. Some redundant links were added to maintain reliability in the network — it is possible to make more than one route between most of the subnets.

Sample network topology is divided into Zones (A, B, C and D). Network traffic inside each zone is not filtered by any firewall.

The given topology shows the structure of a real LAN and has complex access control policy, which is provided by three hardware firewalls. Also some hosts in network topology could additionally have application firewalls (hosts in Accounts Network, for example).

Network topology is represented by an undirected graph with vertexes as network nodes (hosts, routers and firewalls) and edges as links between them. In Figure 1 every subnet is shown as a single host for clarity. All firewalls in this sample network could also perform routing and implement rule-based network traffic filtering.



```
1. deny icmp 7.7.7.1/32 10.0.0.33/32
2. deny icmp 10.0.0.0/24 7.7.7.8/32
3. permit icmp any 7.7.7.0/24
4. permit icmp any 10.0.0.0/24
5. permit icmp any 192.168.0.0/24
6.permit icmp any 172.16.0.0/24
7. deny icmp any any
```

```
1.  deny icmp 7.7.7.0/24 192.168.0.64/30
2.  deny icmp 192.168.0.0/26 7.7.7.2/32
3.  deny icmp 10.0.0.33/32 192.168.0.0/26
4.  deny icmp 192.168.0.65/32 10.0.0.0/27
5.  deny icmp 172.16.0.2/32 10.0.0.0/27
6.  deny icmp 172.16.0.0/28 10.0.0.1/32
7.  deny icmp 172.16.0.25/32 10.0.0.0/27
8.  deny icmp 172.16.0.24/30 10.0.0.7/32
9.  permit icmp any 7.7.0.0/16
10. permit icmp any 10.0.0.0/24
11. permit icmp any 192.168.0.0/24
12. permit cmp any 172.16.0.0/24
13. deny icmp any any
```

```
1.  deny icmp 7.7.7.3/32 172.16.0.0/24
2.  deny icmp 172.16.0.24/30 7.7.7.8/32
3.  deny icmp 192.168.0.65/32 172.16.0.0/24
4.  deny icmp 172.16.0.1/32 192.168.0.64/30
5.  deny icmp 10.0.0.32/29 172.16.0.1/32
6.  deny icmp 10.0.0.33/32 172.16.0.0/24
7.  deny icmp 172.16.0.25/32 192.168.0.0/24
8.  permit icmp any 7.7.7.0/24
9.  permit icmp any 10.0.0.0/24
10. permit icmp any 192.168.0.64/30
11. permit icmp any 172.16.0.0/24
12. deny icmp any any
```

```
1.  deny icmp 7.7.7.4/32 172.16.0.24/30
2.  deny icmp any 7.7.7.5/32
3.  deny icmp 172.16.0.17/32 172.16.0.24/30
4.  deny icmp 172.16.0.25/32 172.16.0.16/29
5.  deny icmp 10.0.0.5/32 any
6.  deny icmp 10.0.0.0/27 172.16.0.25/32
7.  deny icmp 192.168.0.66/32 any
8.  permit icmp 172.16.0.24/30 10.0.0.0/24
9.  permit icmp 172.16.0.24/30 192.168.0.64/30
10. permit icmp any 172.16.0.16/29
11. permit icmp any 7.0.0.0/8
12. permit icmp any 172.16.0.24/30
13. deny icmp any any
```

Figure 1. Example of classic topology.

The approach for migration from initial (classic) topology to SDN with security policy preservation proposed in this article consists of several steps:

Input: The existing network topology and a rule set for each firewall.

Algorithm of migration to SDN Topology:

1) *Rule optimisation:*

a) *Discovery and removal of intra firewall anomalies;*

b) *Discovery and removal of inter firewall anomalies;*

2) *Construction of a single logical (virtual) firewall for every subnet in topology;*

3) *Calculation of the minimum required number of OpenFlow switches and SDN topology selection;*

4) *Translation of firewall rules to flow rules for each OF switch.*

The result: SDN access control policy of rule sets for every OpenFlow switch in SDN topology.

Below we will consider each step of this algorithm, provide implementation details and describe the related problems.

A. *Rule Optimisation*

Every filtering rule is represented by a set of patterns, which describe the possible values of corresponding fields of packet header in real network traffic. Each field (pattern) could have a single value or a range of values. Moreover, each rule defines actions, which will be conducted by the firewall for packets that match the rule. The most common actions are «permit» and «deny» further packet transmission. Most research papers operate with the definition of a filtering rule as a 7-element tuple:

<Sequence number><Action><Protocol><Source Address><Source Port><Destination Address><Destination Port>

Examination of rules during the packet analysis is performed according to their sequence number (from lowest to highest). If the packet does not match any rule, then default action is performed. There are different models of firewalls available. Some of them produce action of the «first match» rule, e.g. Cisco Access Control List [5], and some of them produce action of the «last match» rule, e.g. Packet Filter OpenBSD [24]. We assume that all firewalls in our model work as «first match» and the default action is «deny».

The compliance of firewall rules sequence is critical in the definition of filtering policy, because examination of rules is sequential. If rules are completely different, their sequence is irrelevant, however, such situation is unlikely to occur when merging multiple rule sets from production network firewalls. If the original sequence is not respected some rules may be overlapped by other rules. The research provided by A. Wool in [1] shows, that the most modern firewalls contain critical errors.

In our model the filtering policy is represented and stored as a single root tree on every firewall. Each node in the policy tree represents a field and each branch originated from the node represents a possible value of associated field. Every route in this tree that starts in the root and ends in the leaf presents a unique rule in the policy. Rules that have the same values in some nodes share the same branches, which present these values. The policy tree model allows defining relations and discovering anomalies between rules easily and also accelerating the matching by cutting misleading branches.

As a basis for the rules optimization method two algorithms proposed by E. Al-Shaer et al in [2], [3] were adopted. Both of them describe relations between firewall rules. Work [2] describes the method of intra firewall anomalies discovery and the technique of inserting the rules in the policy tree and work [3] is its logical extension and describes the method of inter firewall anomalies discovery. Rules that cause an anomaly and do not affect reachability of any two nodes in topology should be removed. Modeling relations between the rules in firewalls are necessary for providing analysis of the rule set and development of the technique of anomaly discovery and removal. As a result of providing this step of migration to SDN topology the number of rules on each firewall either remains the same (in case of anomaly absence) or decreases (in case of anomaly removal).

B. Construction of logical firewall for each subnet in topology

The next step, necessary to perform migration from a classic network topology to SDN, is collecting and generalizing filtering rules for each subnet in topology. Filtering rules may be categorized based on the source address (src_addr) and destination address (dst_addr) fields values (fields <protocol>, <src_port>, <dst_port> in terms of provided classification are not significant and could take any valid value):

1) deny <protocol> const_src_addr <src_port> const_dst_addr <dst_port>;

2) permit <protocol> const_src_addr <src_port> const_dst_addr <dst_port>;

3) deny <protocol> src_addr <src_port> const_dst_addr <dst_port>;

4) permit <protocol> src_addr <src_port> const_dst_addr <dst_port>;

5) deny <protocol> const_src_addr <src_port> dst_addr <dst_port>;

6) permit <protocol> const_src_addr <src_port> dst_addr <dst_port>;

7) deny <protocol> src_addr <src_port> dst_addr <dst_port>;

8) permit <protocol> src_addr <src_port> dst_addr <dst_port>.

The detailed description of each type of rules is presented in Appendix 1.

Statement 1: Each rule in the firewall filtering policy belongs to exactly one type. Other types of rules do not exist according to the specified classification.

The problem of constructing a unified logical firewall between any two points may be reduced to the problem of redistribution of rule sets of two firewalls, which are located between two subnets. A similar situation is presented in Figure 1. In case of more than two firewalls lying on the route between two considered subnets, two firewalls which are the nearest to the source subnet are united in one logic firewall on each step and each following firewall is added to the constructed logic firewall.

Typically, filtering rules are applied to a specific interface (port) of network device either on the input direction (in) or the output direction (out). Thereby, we need to find all filtering tables which could be met on the packet's route. For this purpose the logical firewall which is an aggregation of all filtration rules on all possible routes between every two end points of topology (subnets) is constructed using recursive traversal of the graph. We will define an ordered sequence of network devices and corresponding input and output ports which take part in packet transmission as a "route". If filtering rules on two different routes between subnets are different, i.e. if an end point on one route is reachable and on the other is not, the warning message "Conflict Found" is displayed and computation stops. In our example topology of three hardware and one application firewalls (Figure 1) this situation is not possible; however, the implemented application allows this case.

When we have discovered all routes, we then find all firewalls on them. Rules for which "Source Address" field value is not a subset or superset of the source subnet and "Destination Address" field value is not a subset or superset of the destination subnet are deleted from the policy tree of each found firewall.

The policy tree of the most upstream firewall which is left after rule removing is taken as a basis of the source subnet's policy tree. The remaining rules from other firewalls are added into this tree.

For rules with "permit" action located on downstream firewalls we introduce the concept of "compressing" and "extending" rules. We will call a rule "compressing", if it permits only part of network traffic, which could reach the downstream firewall because the next rule that could be matched by the same traffic has a "deny" action and every field in the first rule is a subset of each corresponding field in the second rule. We will call a rule "extending" if, vice versa, it could permit transition of all packets which could come to this firewall. Each field of an "extending" rule is equal or a superset of each corresponding field in an upstream firewall rule with action "permit".

The analysis algorithm, which is applied in constructing a single logical firewall, is presented further:

1) Define the type of rule;

2) Insert the rule in the policy tree of the logical firewall according to its type:

- *Type 1.* If the rule belongs to the first type, it means that network traffic, which could be matched by this rule, has already been matched by a rule with "permit" action on all previous firewalls. Therefore, it is possible to add rules of this type at the beginning of the logic firewall's list, because network traffic that matches this rule will be obligatory dropped.
- *Type 2.* In case of an "extending" rule, there is no need to insert it, because the security policy may change and the firewall may start to allow transmission of packets that were dropped previously. If the rule is "compressing", it is necessary to insert it before the rule, which is its superset, and delete all the following rules after it (for which it is a subset).
- *Type 3.* To accomplish a minimal number of filtration rules it is desirable to insert rules of this type in the destination subnet's policy tree, because

then it will be analyzed once, i.e. when the packet is addressed exactly to this subnet.

- *Type 4.* Whereas packets from multiple subnets could be matched by "Source Address" field value, a rule of this type is "extending" for each of them. Therefore, it could be assigned to the source subnet and processed according to the second type rules, i.e. to be removed.
- *Type 5.* While searching for a place to insert this type of rule we can not define exactly which "permit" rule was matched on the upstream firewall. In case of its insertion without changing, the security policy may change (subnets, which were available because of other "permit" rules, could become unavailable, if we insert a rule of this type before a "permit" rule, or, vice versa, subnets, which were unavailable because of the presence of this rule, could become available). Thereby, for correct adding of rules of this type in the logic firewall, they need to be specified. We have to substitute the network address of the considered source subnet and insert this type of rules according to the instruction for the first type rules.
- *Type 6.* In this case all considerations are similar to Type 5 rules.
- *Type 7.* It is possible to include this type of rules to the source subnet policy tree having preliminarily specified the destination address. In this case, if the analyzed rule is the last rule on the list and has a form "deny <protocol> any <src_port> any <dst_port>", it does not need to be inserted.
- *Type 8.* Rules of this type are "extending" rules, so we should not insert them.

For the most upstream firewall the filtration rules order does not change while constructing a single firewall, the rules could only be specified according to their type.

In case of absence of rules with "permit" action for two considered subnets on any firewall, all the rules which are related to subnets are deleted, and single a rule with "deny" action and addresses of source and destination subnets is inserted into the logical firewall.

The relations between rules have been already defined in work [2]. The comparison of rules in the policy tree with the inserted rule is provided according to the algorithm described in A.

As a result of bypassing all subnets we get (k - 1) policy trees, where k is the number of subnets. All these trees do not intersect with each other because of strictly specified source and destination address fields. Therefore, we could merge them in one policy tree by sequential rules placement one after another. The last rule "deny <protocol> any <src_port> any <dst_port>" does not need to be placed anywhere except the last tree.

Finally, after providing step 3 we obtain a filtering rule tree for each subnet in our network topology. However, it could contain anomalies which need to be removed. The resulting policy tree is the basis of the SDN access control policy and rule generation for OpenFlow switches.

It should be pointed out that rules with the same network addresses, but different ports both on the source side and on the destination side are completely different and we can not compare them.

C. Estimation of the required number of OpenFlow switches and topology selection

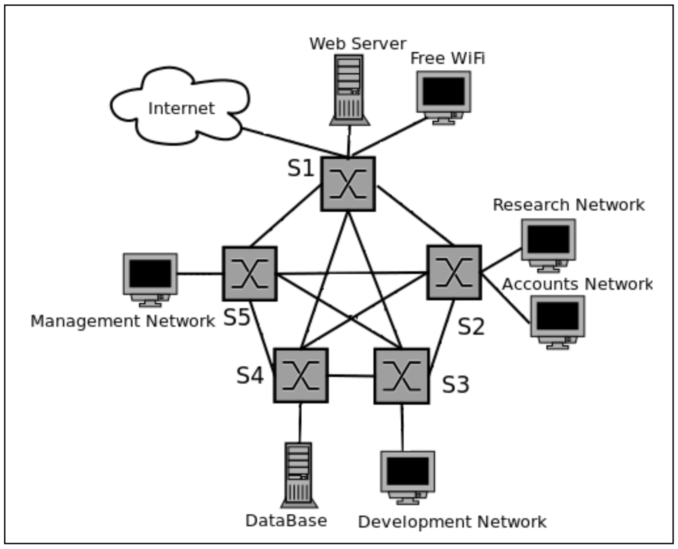


Figure 2. Example of resulting topology.

For migration to the SDN-based topology we need to choose its structure. The most common examples of topologies are tree, fully connected graph, lattice, star, ring, 3-dimensional cube and etc. Topology is selected depending on the network size, reliability, resilience and congestion requirements. For our example of topology we chose a fully connected graph structure because we assume that the number of hosts in the network is small, however the speed of data transmission is a critical factor. It is known that in a fully connected graph topology every packet can reach its destination point in three hops.

Then according to the choice it is necessary to calculate the number of ports for connecting switches with each other. For example, in a fully connected graph topology which consists of n switches, (n-1) port will be occupied on each switch.

The next step in topology replacement is the process of computing the minimal number of OpenFlow switches, which will take part in SDN topology. This problem is a sort of "Bin-Packing Problem", described in [4]. It consists in distribution of subnets with a variety number of hosts between switches with a variety number of ports in such a way that the number of occupied switches would be minimal. It is known as NP-complete problem in computational complexity theory. However, the use of heuristics (the so-called "greedy" bin packing algorithm) significantly reduces computational complexity. Herewith, discovery of optimal solution is not guaranteed, but accurate results are obtained on practice.

After solving this problem, we can unambiguously associate with every subnet the unique switch identifier (DPID) and a group of ports to which this subnet is connected. In case of several subnets connected to a single switch it is necessary to unite their policy trees. For this purpose we consistently insert all rules from one subnet policy tree into another using the technique described above. Thereby, after this step the filtration rule tree is associated with each OF switch that is taking part in SDN topology.

In case of our topology example, SDN will consist of 5 OF switches with different number of ports (2 48-port switches and 24-, 16- and 8- port switches one at a time). The example of the resulting SDN is shown in Figure 2.

D. *Translation of filtration rules to flow rules*

SDN switching equipment supports flow routing tables (Flow Tables) in which processing rules for packet flows are installed. The final step of migration from a traditional topology to an SDN paradigm is installing flow rules into OF switches flow tables.

Appendix 2 contains Table 3 in which Extended Cisco ACL syntax [5] is associated with fields of flow rules in accordance with OpenFlow Specification. With the “action” term in OpenFlow context we designate an operation that forwards packets to the specified port or modifies the packets (e.g. decreases TTL field value or changes MAC/IP source or destination addresses). Each rule may have multiple actions.

Every rule field in the Flow Table contains either a defined value or ANY value which matches any content of the corresponding packet field. If a switch supports setting arbitrary bit masks for the source and/or destination Ethernet address fields and for the source and/or destination IP address fields, they could define the match more precisely. In addition to packet headers, input port and metadata fields may be taken into account. Metadata may be used for transmitting information between switch tables.

One of the advantages of Extended Cisco ACL syntax is the possibility of flexible setting of a range of source and destination ports. OpenFlow protocol specification does not support this feature. Another ACL advantage is the possibility of setting the time activity range for rules, e.g. on weekdays from 10 am to 7 pm. However, this functionality may be implemented by the SDN controller application which is involved in installing/removing rules in Flow Tables. On the contrary, the possibility of simultaneous packet filtration based on L2-L4 header fields exists in OF protocol, for example matching network IP address and MAC address in the same rule. Also it is possible to work with packets which belong to a specified VLAN or have a specified MPLS label.

In other cases rule fields either match flow fields precisely or could be implemented by protocol specification or functionality of application that works on the controller. Thereby, Extended Cisco ACL syntax could be taken as a basis of input language of traffic filtration rules in input topology.

Functions of traffic filtration in our tool were implemented in two options: classic Extended Cisco ACL and Reflexive Cisco ACL. In case of the classic variant of implementation the rule flows that permit all incoming traffic with action “CONTROLLER” were installed on every port which connects the switch to another switch. In the reflexive variant of implementation all new flow rules for each new connection with the highest priority were installed on all switches which participate in packet transmission (proactive mode). Because of this, the required time for searching the matching rule is significantly reduced.

II. EXPERIMENTAL RESULTS

The proposed approach was implemented as application for the POX SDN controller [25] using Python 2.7 programming language. The POX controller supports protocol OpenFlow version 1.0.

As a testbed switch 48-GbE NEC PF 5820 was used [26]. This model supports OpenFlow version 1.0, up to 80000 L2-layer flow records and up to 750 L2-tuple flow records. A laptop (2 Core 2.2 GHz CPU, 2048 Mb RAM) with OS Ubuntu 12.04 was used as a server for SDN controller application.

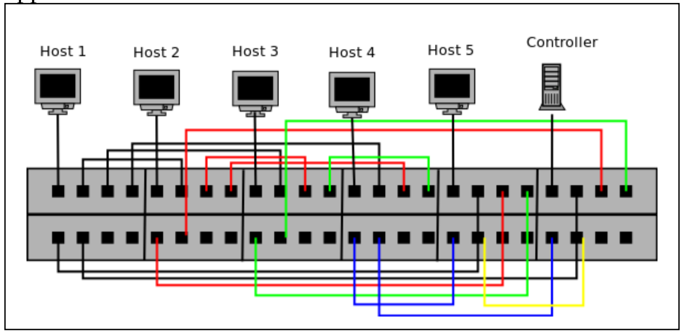


Figure 3. Experimental testbed.

NEC switch supports partitioning into several virtual switches; however all of them are using a common Flow Table. In our testbed setup the NEC hardware switch was divided into 6 virtual switches with 8 ports on each. Then virtual switches were connected with each other to create a full mesh network topology. The scheme of the testbed is shown in Figure 3.

Hosts 1-5 were connected to the virtual switches using ports 1, 9, 17, 25 and 33 accordingly. Every host was associated with an IP address from different subnets in the example topology. The server was connected to the 6-th virtual switch (port 41).

At first we reproduced the example of the described above topology and checked the correctness of installation and translation of filtration rules from Cisco ACL syntax to flow rules.

Then we performed 2 experiments. The obtained results are approximate and are performed as an average number for 5 measurements.

1) *The measurement of rule installation time depending on the number of rules.*

TABLE I. THE MEASUREMENT OF TIME OF RULES INSTALLATION DEPENDING ON THE NUMBER OF RULES.

Number	Number of rules	Required time, ms
1	100	0.02
2	200	0.04
3	300	0.06
4	400	0.08
5	500	0.1

According to the obtained results the required time of rule installation depends linearly on the number of rules.

2) The measurement of the delay time of new flow transmission depending on the location of the first match rule in the Flow Table.

For providing this experiment we used a "ping" utility that allows determining the time of ICMP Echo-Reply packet arrival.

TABLE II. THE DELAY TIME IN CASE OF THE FIRST MATCH RULE'S LOCATION IN THE FIRST PLACE.

Variant of realization	Required time, ms
Standard	35
Reflexive	220 (1st packet), 0.2 (subsequent packets)

We also performed measurements for cases when the first match rule was located in the last place (for 100, 200, 300, 400, 500 and 600 rules). For both variants of filtration implementations the delay time remained approximately the same (about 40 ms and 250 ms accordingly) because the required time for searching in Flow Table that is located in Ternary Content Addressable Memory (TCAM) is minimal.

We may conclude from our experiments that the delay time for packet transmission for classic implementation remains constant irrespective of the number of flow rules and for reflexive realization the delay time for packet transmission increases, however, after installing rule flows on every switch the packet transmission time is estimated at 0.2 ms, which is 150 times less.

The implementation of the described method may be found at https://gitorious.org/sdn-network-access-control/.

III. RELATED WORK

There is a lot of already published research in the area of network access control, provisioning and management. In this section we will provide the major works that intersect with the topic of our research in five basic areas – packet filtering policy modeling, conflict discovery in security policy, distributed firewall policy management, firewall performance optimization and modeling the network reachability. We will also describe the existing solutions for providing SDN security.

For rule representation and storing some models have been proposed. The method shown by S. Hazelhusrt in [6] uses Binary Decision Diagrams (BDD) for optimization of packet classification. Another model presented by B. Hari et al in [7] uses the space of tuples. A set of several filters is generalized in one tuple which is stored in a hash-table. Binary trees in several dimensions described by V. Srinivasan in [8] are used for modeling filters. The geometric model proposed by D. Eppstein et al [9] is used for rules representation which includes 2 tuples. Since all of these models were implemented for packet filtering optimization in high-load networks they are too complex to use in the security policy analysis. An approach based on filtering policy tree construction was presented by E. Al-Shaer et al in [2]. It was chosen for our implementation because of simplicity and enough descriptive power for the security policy analysis.

For the analysis of traffic filtering several techniques (both for centralized and decentralized systems) were proposed. A technique for anomaly discovery in the policy of a single firewall was shown by E. Al-Shaer et al in [2]. A method based on Decision Diagrams was described by M. Gouda et al in [10] for setting a sequence of rules which is ordered, complete (every packet in the network matches one or more rules on the list) and compact (redundant rules are absent). There are also some analyzers and tools for managing the network security policy, e.g. Fang [11], Firmato [12], Fireman [13] et al. Also the problem of filtration rules optimization was solved by A. Tapdiya in [14] using heuristics based on genetic algorithms.

In the area of distributed firewalls current research predominantly focuses on the policy of distributed firewalls management. The technology of global policy management described by J. Guttman in [15] defines the global language for policy setting and filtration rules creation with an algorithm for its verification. A technique for anomaly discovery in a multi firewall environment, which consists of several firewalls, was proposed by E. Al-Shaer et al in [3].

In the area of theoretical presentation of network topology, its network elements and construction of a reachability graph of end nodes there have also been published several works. The work presented by G. G. Xie et al in [16] is based on a graph theory and the network is modeled as a triple that consists of routers, their physical links and a set of functions for defining packet filtration and transformation rules. Firewalls and NAT devices could be expressed using this technique. However, this approach is exclusively theoretical and there are no experimental results. Also this approach could be used to present only static NAT and filtration rules based on destination address and it does not take into account the existing connection-oriented and non connection-oriented protocols. To eliminate some restrictions of work [16] its addition [17] has been published. A more general model is used to describe firewalls, packet filtration and transformation rules, e.g. adding the opportunity to process policies that depend on the source address and filtration stages.

A technique of using Firewall Decision Diagrams (FDD) for precise computing network reachability was described by A. R. Khakpour et al in [18]. Implemented model supports packet routing, filtration and rule transformation. The tool could be used to obtain reachability of two network nodes using SQL-like language.

All previous works were focused on the network as an object, which is managed by a single participant, thus they require full knowledge of infrastructure and its rules. In real scenarios it is not always feasible, especially when several subnets are managed by several participants. The approach proposed by F. Chen et al in [19] has been implemented to solve this problem. In its solution firewalls need to exchange their ACLs, previously encrypted and encoded to avoid unauthorized access, with neighbors. Rules are consequentially compared using binary prefixes and simple logical operations to find their intersection and, as a result, network reachability. This realization does not require a precomputational phase, but does not support any transformation network devices.

There is a single solution that provides network security in SDN. It is a SNAC (Simple Network Access Control) controller [20]. SNAC is an OF controller that is oriented toward building corporate networks. It is based on the NOX controller version 0.4 and has a flexible language for policy definition, a user-friendly interface for setting network devices and event monitoring. At first SNAC was an Open Source project (last release v.0.4.2) and was developed by Big Switch Network company under GPLv2 license. However, then it became a closed project, developed by Nicira Networks company. SNAC allows setting a network using Formal Modeling Language (FML).

The main component of the controller is the application Policy Manager. It unites the functions of establishing the connectivity and security policy compliance in high-level categories. The system defines entities, related to network (physical switches and physical ports on switches) and entities, related to clients (hosts and users). Each entity is associated with unique information and authentication policies. All hosts that appear in the network are automatically redirected to the controller for authentication.

The information supplied by OS contains statistics for every entity in the network: the number of active entities, the total number of registered entities (both active and inactive) and the number of entities that have been seen in the network and the system has not any registration information; policy statistics: the average number of denied network flows, the number of rules in the current policy, the number of configured rules-exceptions; recent events: all events that were registered by the system and happened higher than L2 layer. All events are associated with a priority number from 1 (the highest priority) to 5 (the lowest priority). However, SNAC does not perform migration from classic topology to SDN with security policy protection and we do not know anything about methods of its implementation not to mention optimization.

IV. Conclusion

In this work we have shown the methodology of migration from a network with traditional architecture which uses dedicated hardware firewalls to SDN topology. Also the approach for solving the problem of filtration rules migration from dedicated devices to L2 switched network preserving the reachability matrix and maximizing L2 network throughput was proposed. The described algorithm reduces the number of analyzed rules in packet transmission between subnets by removing anomalies and transferring every rule to the switches closest to the traffic source. In case of each subnet connecting to the dedicated OF switch, the number of processed rules in the worst case would be minimal.

According to our topology example (Figure 1), the network packet with the source IP address 7.7.7.7 and destination IP address 172.16.0.25 has to go through 3 hardware firewalls and one application firewall. As a result 41 rules would be analyzed. It is important to say that we describe the worst case where every packet matches the penultimate rule (the last rule is the default “deny” rule). After migration to SDN the number of processed rules for the same case is significantly reduced — which is 7 (for the best case) and 32 (for the worst case).

A POX controller is used for implementation of a network access control application prototype. Its performance is significantly less than similar solutions on C++, Java or Ruby. For production-grade implementation of the proposed approach in real networks it is necessary to port it to another controller, e.g. NOX.

Although the problem of discovering the optimal sequence of filtering rules from all linear combinations is an NP-complete problem, some heuristics have been proposed, e.g. a heuristic based on changing the order of disjoint rules according to the probability of their activation was proposed by E. W. Fulp in [21]. Several algorithms for rule sorting and a method for merging two policies were shown by A. Tapdiya et al in [22]. A technique for dynamic rule reordering depending on collected traffic statistics was described by E. Al-Shaer et al in [23]. The results shown in these papers are close to optimal. According to the OpenFlow specification, counters could be stored for each Flow Table, flow, port, queue, group and event container. OpenFlow-compatible counters could be implemented in software by examining the hardware counters, which have a more limited range. Also we plan to implement our own methods for rule order optimization considering the information from the counters.

The packet pipeline processing using multiple Flow Tables appeared in protocol OpenFlow version 1.2. However, all existing hardware OF switches support only version 1.0. POX also supports only protocol version 1.0. Thereby, the implementation of packet processing using several Flow Tables is impossible now. However, when switches supporting the new version of protocol emerge on the market it will be possible not to unite several policies to store them in a single table, but to locate each policy tree in its own Flow Table in case of connecting several subnets to one switch. As a result we suppose that the speed of search in Flow Table would increase significantly.

Despite the fact that the widely used protocol OpenFlow version 1.0 does not provide the mechanisms of network security provision, they could be implemented through traffic filtration on L2 layer, what is more the total filters' throughput will rise compared to existing hardware solutions.

References

[1] A. Wool, “Trends in Firewall Configuration Errors: Measuring the Holes in Swiss Cheese,” in IEEE Internet Computing, vol. 14, pp. 58–65, 2010.

[2] E. Al-Shaer and H. Hamed, “Modeling and Management of Firewall Policies,” in IEEE eTransactions on Network and Service Management, vol. 1-1, April 2004.

[3] E. Al-Shaer, H. Hamed, R. Boutaba and M. Hasan, “Conflict Classification and Analysis of Distributed Firewall Policies,“ in IEEE Journal on Selected Areas in Communications, vol. 23, No. 10, October 2005.

[4] D. S. Johnson, “Near Optimal Bin-Packing Algorithms,“ Massachusetts Institute of Technology, Dept. of Mathematics, 1973

[5] Extended Cisco ACL syntax, http://www.cisco.com/en/US/products/sw/secursw/ps1018/products_tech_note09186a00800a5b9a.shtml

[6] S. Hazelhusrt, “Algorithms for Analyzing Firewall and Router Access Lists,” in Technical Report TR-WitsCS-1999, Department of Computer Science, University of the Witwatersrand, July 1999.

[7] B. Hari, S. Suri and G. Parulkar, “Detecting and Resolving Packet Filter Conflicts,” in Proc. of IEEE INFOCOM’00, March 2000.

[8] V. Srinivasan, S. Suri and G. Varghese, “Packet Classification Using Tuple Space Search,” in Computer ACM SIGCOMM Communication Review, October 1999.

[9] D. Eppstein and S. Muthukrishnan, “Internet Packet Filter Management and Rectangle Geometry,” in Proc. of 12-th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), January 2001.

[10] M. Gouda and A. Liu, “Firewall design: consistency, completeness, and compactness,” in Proc. of the 24th IEEE International Conference on Distributed Computing Systems, Tokyo, Japan, March 2004, pp. 320-327.

[11] A. Mayer, A. Wool, and E. Ziskind, “Fang: A firewall analysis engine,” in Proc. of the 2000 IEEE Symposium on Security and Privacy (S&P 2000), May 2000, pp. 177.

[12] Y. Bartal, A. Mayer, K. Nissim and A. Wool, “Firmato: A novel firewall management toolkit,” in ACM Transactions on Computer Systems, vol. 22, no. 4, Nov. 2004, pp. 381 – 420.

[13] L. Yuan, J. Mai, Z. Su, H. Chen, C.-N. Chuah and P. Mohapatra “FIREMAN: A Toolkit for Firewall Modeling and Analysis,” in Proc. IEEE Symposium on Security and Privacy, May 2006.

[14] A. Tapdiya, “Firewall policy optimization and management,” in Master’s thesis, Wake Forest University, Computer Science Department, 2008.

[15] J. Guttman, “Filtering Posture: Local Enforcement for Global Policies,” in Proc. of 1997 IEEE Symposium on security and Privacy, May 1997.

[16] G. G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. Greenberg and G. Hjalmtysson, “On static reachability analysis of ip networks,” in INFOCOM ’05 Proc. of the 24-th Annual Joint Conference of the IEEE Computer and Communications Societies, Miami, USA, 2005, pp. 2170–2183.

[17] S. Bandhakavi, S. Bhatt, C. Okita, and P. Rao, “Analyzing end-to-end network reachability,” in IM ’09 Proc. of the 11-th IFIP/IEEE International Conference on Symposium on Integrated Network Management, Long Island, USA, 2009, pp. 585–590.

[18] A. R. Khakpour and A. X. Liu, “Quantifying and Querying Network Reachability,” in CDCS ’10 Proc. of the 2010 IEEE 30-th International Conference on Distributed Computing Systems, Genoa, Italy, 2010, pp. 817–826.

[19] F. Chen, B. Bruhadeshwar, and A. X. Liu, “A cross-domain privacy-preserving protocol for cooperative firewall optimization,” in INFOCOM ’11 Proc. of the 30-th IEEE International Conference on Computer Communications, Shanghai, China, 2011, pp. 2903–2911.

[20] SNAC, http://www.openflow.org/wp/snac/

[21] E. W. Fulp, “Optimization of Network Firewall Policies using Directed Acyclical Graphs” in Proc. of IEEE Internet Management Conference, 2005.

[22] A. Tapdiya, E. W. Fulp, “Towards Optimal Firewall Rule Ordering Utilizing Directed Acyclical Graphs,” in Proc of 18-th International Conference on, August 2009.

[23] E. Al-Shaer, M. El-Alfy and S. Z. Selim, “Dynamic Rule-ordering Optimization for High-speed Firewall Filtering,” in Proc. of IEEE International Conference on Computer Systems and Applications, 2007.

[24] Packet Filter OpenBSD, http://www.openbsd.org/faq/pf/

[25] POX, http://www.noxrepo.org/pox/about-pox/

[26] Nec Switch, http://www.nec.com/en/global/prod/pflow/images_documents/ProgrammableFlow_Switch_PF5820.pdf

V. APPENDIX 1

Classification of network traffic filtration rules based on values of fields “Source Address” and “Destination Address”:

1) deny <protocol> const_src_addr <src_port> const_dst_addr <dst_port>, where both the source and destination IP addresses can match packets that belong to one subnet in the given topology, e.g. rule 3 in Firewall A (Figure 1).

2) permit <protocol> const_src_addr <src_port> const_dst_addr <dst_port>, where both the source and destination IP addresses can match packets that belong to one subnet in the given topology, e.g. rule 9 in Firewall C (Figure 1).

3) deny <protocol> src_addr <src_port> const_dst_addr <dst_port>, where the source IP address can match packets that belong to several subnets and IP destination address can match packets that belong to one subnet in the given topology. A special case of the given rule type is deny <protocol> any <src_port> const_dst_addr <dst_port>. For example rule 2 in Firewall X (Figure 1).

4) permit <protocol> src_addr <src_port> const_dst_addr <dst_port>, where the source IP address can match packets that belong to several subnets and IP destination address can match packets that belong to one subnet in the given topology. A special case of the given rule type is permit <protocol> any <src_port> const_dst_addr <dst_port>. For example rule 10 in Firewall B (Figure 1).

5) deny <protocol> const_src_addr <src_port> dst_addr <dst_port>, where the source IP address can match packets that belong to one subnet and IP destination address can match packets that belong to several subnets. A special case of the given rule type is deny <protocol> const_src_addr <src_port> any <dst_port>. For example rule 1 in Firewall B (Figure 1).

6) permit <protocol> const_src_addr <src_port> dst_addr <dst_port>, where the source IP address can match packets that belong to one subnet and IP destination address can match packets that belong to several subnets. A special case of the given rule type is permit <protocol> const_src_addr <src_port> any <dst_port>. For example rule 8 in Firewall C (Figure 1).

7) deny <protocol> src_addr <src_port> dst_addr <dst_port>, where both the source and destination IP addresses can match packets that belong to several subnets. A special case of the given rule type is deny <protocol> any <src_port> any <dst_port>.

8) permit <protocol> src_addr <src_port> dst_addr <dst_port>, where both the source and destination IP addresses can match packets from several subnets. A special case of the given rule type is deny <protocol> any <src_port> any <dst_port>. For example rule 10 in Firewall A (Figure 1).

VI. Appendix 2

TABLE I. Table of fields of Extended Cisco ACL syntax and associated values of fileds in flow rules

Description	**Extended Cisco ACL**		**OpenFlow**	
	Field name	***Necessity***	***Field name***	***Bit length***
Telnet authentication before establishing a new session.	dynamic	Optional	Only while using implemented application.	
Setting the time range after which, in case of inactivity, the rule will be deleted.	timeout	Optional	Every rule has two attributes: Idle-timeout - rule will be deleted after specified period after its last usage. Hard-timeout – rule will be deleted after specified period of time irrespective of its use. In case of absence of these values, rule will be present in Flow Table constantly.	
Action that will be performed in match case.	deny / permit	Obligatory	The list of actions, which could be applied to packets, is defined in protocol specification.	
L3 layer protocol	Protocol	Obligatory	IPv4 Protocol / ARP opcode	8
Source IP address and mask	source source-wildcard	Obligatory	IPv4 source address	32
Source port or port range	Operator [port]	Optional	Transport source port / ICMP type	16
Destination IP address and mask	Destination destination-wildcard	Obligatory	IPv4 destination address	32
Destination port or port range	operator [port]	Optional	Transport source port / ICMP type	16
Permission for TCP packets transmission that are part of established TCP session.	established	Optional	Only while using implemented application.	
Comparison of packets by precedence field value	precedence	Optional	Absent	
Comparison of packets by TOS field value	tos	Optional	IPv4 ToS bits	6
Possibility of writing logs in case of rule match	log / log-input	Optional	Only in case of removing the rules.	
Setting the time range of rule activity	time-range	Optional	Only while using implemented application that would install/remove flow records depending on current time.	
Comparison of packets by ingress port.	Rule is installed on specified interface and specified direction (in/out).		Ingress port	32
Metadata used for transmitting information between Flow Tables	Not required		Metadata	64
Source MAC address	It is possible to set in Cisco MAC ACL.		Ethernet source address	48
Destination MAC address	It is possible to set in Cisco MAC ACL.		Ethernet destination address	48
Type of Ethernet frame	It is possible to set in Cisco MAC ACL.		Ethernet type	16
Comparison of packets by VLAN ID label	Not possible		VLAN ID	12
Comparison of packets by VLAN Priority label	Not possible		VLAN Priority	3
Comparison of packets by MPLS label	Not possible		MPLS Label	20
Comparison of packets by MPLS traffic class.	Not possible		MPLS traffic class	3