Load Distribution of an OpenFlow Controller for Role-based Network Access Control

Takayuki Sasaki, Yoichi Hatano, Kentaro Sonoda,
Yoichiro Morita, Hideyuki Shimonishi, Toshihiko Okamura
NEC
1753 Shimonumabe Nakahara-ku
Kawasaki Japan
{t-sasaki, y-hatano, k-sonoda, y-morita, h-shimonishi, t-okamura}@{fb, ct, cp, dc, cd, da}.jp.nec.com

Abstract—**Network attacks have been coming from not only outside of an organization but also internal networks in recent years due to malware infected clients and malicious insiders. Therefore, a firewall on the network boundary is insufficient for preventing such attacks. To prevent the attacks, we have developed a network access control system using OpenFlow. The system monitors whole internal networks and performs access control on the basis of Role Based Access Control (RBAC) on OpenFlow architecture. In the system, however, one problem is that the controller may become a performance bottleneck of the system for large scale network, because the controller monitors and controls all traffics in the network. In this paper, we propose an architecture which evaluates RBAC rules at OpenFlow switch side for load distribution. Furthermore, we evaluate its feasibility and performance, and show that the architecture can reduce the size of dynamically distributed rules by 93% in an ideal case.**

Keywords— Network Access Control, RBAC, OpenFlow, Software Defined Network

I. INTRODUCTION

Network attacks have been coming from outside networks such as internet, and the attacks are prevented by a firewall at the border between inside and outside of a network. But in recent years, the attacks also have been coming from internal network due to malware infected clients and malicious insiders. Therefore, a firewall at the border of the networks is insufficient for preventing such attacks. We can identify two difficulties to tackle the problem. One difficulty is monitoring points. All traffics from outside can be monitored at the border between outside and inside, but on the other hand traffics in an inside network do not pass through a single point, thus we need to monitor whole internal network. The other difficulty is creation of access control rules. Considering traffics from outside, few servers are open to outside network, thus we only need to specify rules regarding such servers. However, in the internal network, there are many clients and servers compared with the above case, and we need to specify complex and a lot of rules regarding client-sever communications.

To tackle the difficulties, we have developed an access control system [1] using OpenFlow [2]. For whole network monitoring, we leverage OpenFlow architecture, in which an OpenFlow controller manages all OpenFlow switches. And, for supporting the rule writing, we adopt Role Based Access Control (RBAC) [6]. RBAC defines roles as collections of rights, and roles are assigned to users, then it performs access control on the basis of the assigned roles. To reduce difficulty of policy writing, role definition and role assignment can be specified by different administrators. For example, the human resource managers are responsible for the role assignment to users and the server administrators are responsible for the role definition.

However, the proposed system has a scalability problem due to centralized architecture, because the OpenFlow controller is a single point to be responsible for control of whole network. Specifically, it monitors all traffics by receiving queries from OpenFlow switches, and it performs policy decision and returns a result of the decision to the OpenFlow switches.

For this scalability problem, we propose an architecture which enforces RBAC rules at OpenFlow switches. The architecture offloads a load of the OpenFlow controller using following ideas.

- **Policy decision at switch side:** This idea distributes the load of the controller to switches. OpenFlow architecture performs all access control decisions at the controller, on the other hand the proposed architecture performs the decisions at switch side according to pre-distributed rules.
- **Pre-distribution of the rules:** For temporal load balancing, the controller dynamically distributes only role assignments to the switches, and role definitions are distributed in advance.

For evaluating its feasibility and performance, we preliminarily implement the architecture using a multiple-table function of OpenFlow specification. Evaluation results show proposed architecture is feasible on the OpenFlow 1.3 reference implementation [3], and it reduce 93% of dynamically distributed rules.

II. APPROACH

A. OpenFlow Architecture and its Problem

Traditional L2 or L3 switch has a data plane for forwarding ethernet frames/packets and a control plane for decision making. To improve flexibility and management capability of a network, OpenFlow architecture decouples these two planes. OpenFlow switch is responsible for the data plane, and the OpenFlow controller is responsible for control plane. Specifically, when the OpenFlow switch receives a packet, the switch issues an inquiry to the OpenFlow controller to know how the packet should be handled. This inquiry is called "packet in" and it contains information of the packets such as MAC address, IP address and TCP/UDP port number. Using the information of the "packet in", the OpenFlow controller makes decisions such as "forward", "drop" and "modify", and the controller returns the decision to the switch. Specifically, the controller issues a "flow mod" message for notifying the decisions as a flow entry. A flow entry comprises a match field to specify condition and an action field to specify the decision. Then, the switch stores the flow entries, and handles the packets according to the flow entries.

In the OpenFlow architecture, the OpenFlow controller is a single point to make decisions, thus in large network the controller must process all inquiries from OpenFlow switches. Therefore, in large network, the OpenFlow controller may become a bottleneck of the system

B. *Approaches of Performance Improvement*

For distributing the load, horizontal load distribution, vertical load distribution, and temporal load distribution are well-known approaches used for distributed systems.

Horizontal load distribution: This approach improves the controller performance using parallelization techniques such as multi-threads/multi-core [4][5]. Furthermore, multiple controller architecture [9] has been proposed. This approach can improve the controller performance to 1~10M flow/sec.

In this paper, we focus on following two approaches, but these approaches can be simultaneously used with the horizontal load distribution.

Vertical load distribution: This approach offloads the load by moving some functionalities to another layer. Based on this approach, the policy decision functionality can be performed by OpenFlow switches for reducing the controller load. However, we can identify a limitation of the OpenFlow switches, which have a limited memory area to store rules, therefore size of distributed rules should be small. Furthermore, as for reducing distribution cost of the controller, the rule size should be small. However, the OpenFlow architecture sends rules for each client-server communication, thus the distribution size would become large in case of heavy traffics.

Temporal load distribution: This approach distributes a peak load by preprocessing. Proactive mode is a kind of this approach, and it distributes rules in advance for reducing queries from OpenFlow switches. However, access control rules can not be distributed in advance, because the rules should be distributed after authentication which identify who uses the client. For example, access rights of a shared client depend on a log-on user of the client.

III. ARCHITECTURE

A. *Overview*

To distribute the load of the OpenFlow controller, we introduce following mechanisms corresponding to the approaches mentioned in the previous section.

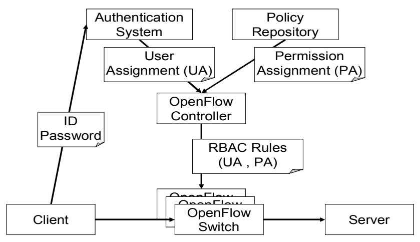


Fig. 1. Architecture

Functional load distribution to the switches: A part of policy decision processes is moved from the OpenFlow controller to the OpenFlow switches. Specifically, access control rules are distributed to the OpenFlow switches and the switches perform policy decision according to the rules. Furthermore, for reducing the rule size, the proposed architecture distributes RBAC rules.

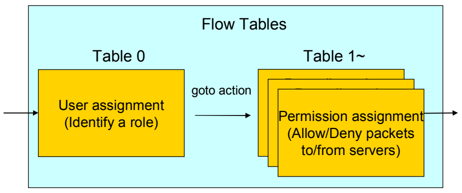
Temporal load distribution of rule distribution: Rule distribution phase is divided into two steps; before authentication and after authentication. This step distributes a peak of the load by temporal load balancing using a structure of RBAC, which comprises user assignment (role assignment) and permission assignment (role definition). The former part is distributed after authentication, but on the other hand the latter part is distributed before authentication.

To realize above ideas, we design an architecture which processes RBAC rules at the switch side (Fig. 1). An authentication system authenticates users by ID and password and sends user assignments to the OpenFlow controller. A policy repository stores permission assignments and sends them to the OpenFlow controller. Then, the OpenFlow controller inputs RBAC rules (user assignments and permission assignments) into the OpenFlow switches. According to the RBAC rules, the OpenFlow switches perform packets filtering.

B. *Policy Decision by OpenFlow Switches*

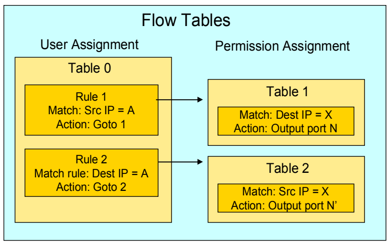
In the OpenFlow architecture, an OpenFlow controller is a Policy Decision Point (PDP), and OpenFlow switches are Policy Enforcement Points (PEP). In the proposed architecture, a part of PDP functionalities is delegated from the OpenFlow controller to the OpenFlow switches. Specifically, network access control rules are distributed to the OpenFlow switches from the OpenFlow controller, and the rules are enforced by the switches. However, a memory of the OpenFlow switch is limited. For example hardware OpenFlow switch PF5240 can store 160,000 rules, and OpenFlow 1.3 reference software switch (default setting) can store 1,024 rules. Therefore rules within the memory size are moved to the OpenFlow switches and evaluated by them. The rest of the rules are still in the OpenFlow controller and evaluated by the controller when it receives inquiries from OpenFlow switches

For reducing rule size, the OpenFlow controller distributes RBAC rules which comprise two parts. One is permission assignment to describe rights, and its structure of each entry is (Role, Object, and Action). The other part is user assignment which shows assignments of roles to users, and its structure of each entry is (User, Role). For controlling network using RBAC, we treat a source IP address as the user and a destination IP address as the object. The action is "pass packet"



Flow Tables
Table 0
Table 1~
User assignment (Identify a role)
goto action
Permission assignment (Allow/Deny packets to/from servers)

Fig.2 Flow table structure



Flow Tables
User Assignment
Permission Assignment
Table 0
Rule 1 Match: Src IP = A Action: Goto 1
Rule 2 Match rule: Dest IP = A Action: Goto 2
Table 1
Match: Dest IP = X Action: Output port N
Table 2
Match: Src IP = X Action: Output port N'

Fig.3 An example of flow tables

or “drop packet”. For example, there are a client (192.168.0.1) and a server (192.168.0.2). In case that a permission assignment is (role1, 192.168.0.2, pass packet) and a user assignment is (192.168.0.1, role1), then an access from the client to server is accepted. Here, “role1” is an arbitrary name of the role.

RBAC can reduce the size of the rules compared with ACL which is used by common firewalls. ACL comprises tuples of (source IP address, destination IP address, action), thus the rule size is

$$\textit{size of ACL} = \textit{\# of src IP addresses} * \textit{\# of dest IP addresses}$$

$$= \textit{\# of clients} * \textit{\# of servers}$$

As for RBAC, assuming a simple situation where a user has one role and a server is specified by one role, the size of RBAC rules is sum of user assignments (UA) and permission assignments (PA). Therefore, its size is calculated as follows.

$$\textit{size of RBAC} = \textit{\# of UA} + \textit{\# of PA} = \textit{\# of clients} + \textit{\# of servers}$$

The rule size of the RBAC is the sum of the clients and servers, on the other hand, the size of ACL is the product of those. Therefore, we can reduce rule size using RBAC.

C. *Pre-distribution of RBAC Rules*

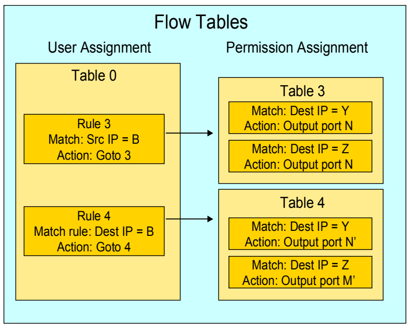
Using RBAC rules, the proposed architecture distributes permission assignments and user assignments at different timings for temporal load balancing. In the proposed architecture, a user’s role is associated with a clients IP address at the authentication step. Thus, user assignments (client IP address and role) need to be distributed after the user authentication. On the other hand, permission assignments can be distributed before the authentication, because permission assignments do not contain information about users (clients), and contain only objects (servers) and actions. By distributing permission assignments and user assignments at different timings, the load of the OpenFlow controller is leveled in terms of time.

IV. DESIGN AND IMPLEMENTATION

For feasibility evaluation, we implement an enforcement mechanism of RBAC rules using multiple table function of an OpenFlow switch, which has been supported from OpenFlow specification version 1.1. Furthermore, we implement two convert scripts as a controller for converting RBAC rules into rules of OpenFlow switches.

A. *Architecture of OpenFlow Switch*

An OpenFlow switch has flow tables to store rules called “flow entry” which comprises a match filed and an action field.



Flow Tables
User Assignment
Permission Assignment
Table 0
Rule 3 Match: Src IP = B Action: Goto 3
Rule 4 Match rule: Dest IP = B Action: Goto 4
Table 3
Match: Dest IP = Y Action: Output port N
Match: Dest IP = Z Action: Output port N
Table 4
Match: Dest IP = Y Action: Output port N'
Match: Dest IP = Z Action: Output port M'

Fig.4 An example of flow tables (two servers)

The match field describes a condition to execute the action and it can specify network protocol information such as source/destination MAC address and IP address. The action field specifies the action for the packet. For example, we can specify the action such as “output X” (output the packet from port X), “drop” (drop the packet), and “goto X” (jump to table X). When an OpenFlow switch receives a packet from a client, the OpenFlow switch searches a flow entry matching the packet. Then the OpenFlow switch performs action specified by the action field of the entry. For example, in case that a flow entry has “src IP =A” as match field and “output X” as action, the OpenFlow switch transfers packets from IP A to the specified port X of the switch.

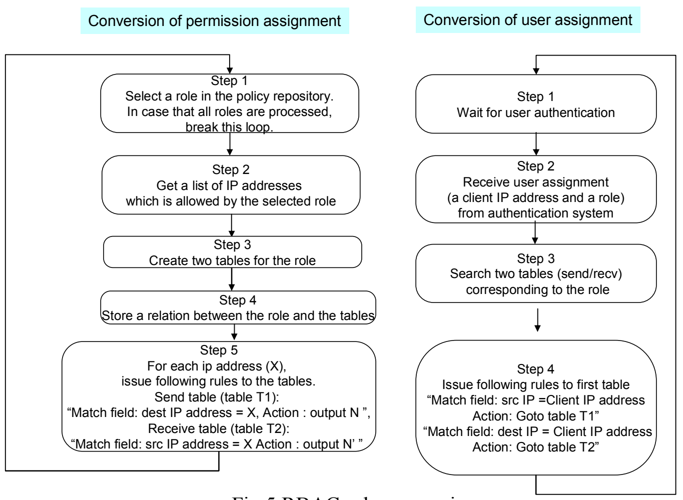
Using above OpenFlow specification, we implement an enforcement mechanism of RBAC rules on OpenFlow switch, and also we implement RBAC rule/flow entry converters.

B. *RBAC Rules Evaluation by OpenFlow Switch*

As described above, the OpenFlow switch can have some tables to store flow entries and jump to a table to be evaluated using “goto” action. For evaluating RBAC rules, we use first table (table 0) for user assignments to identify a role of a client, and following table (table 1~) for permission assignments to specify allowable server IP addresses. And the first table and following tables are connected with “goto” action (Fig. 2).

For mutual communication between clients and servers, we deploy two tables for each role. One is for sending packets from clients and the other is for receiving packets from servers. Fig.3 shows an example of flow tables. In this case, we assume that a client (IP A) communicates with a server (IP X). For this, we need to configure three tables; one table is for the user assignment and two tables are for the permission assignment. The first table (table 0) has two rules to associate the client with the role. Specifically, rule 1 and rule 2 show that a client (IP A) has a role corresponding to table 1 and table 2. The second table (table 1) allows packets to server (IP X) and the third table (table 2) allows packets from the server. These rules are enforced as follows. When the client sends a packet (src IP = A, dest IP = X), it matches rule 1 in the table 0. Then the OpenFlow switch refers to table 1 according to “goto” action. Next, the packet matches a rule in the table 1. Finally the packet is sent from port N specified by the rule. Moreover, packets from the server match rule 2 in the table 0, and the packets are allowed by a rule in table 2.

A role can contain permissions for access multiple servers. For example, table 3 and table 4 in figure 4 correspond to a role



Conversion of permission assignment
Step 1
Select a role in the policy repository.
In case that all roles are processed, break this loop.
Step 2
Get a list of IP addresses which is allowed by the selected role
Step 3
Create two tables for the role
Step 4
Store a relation between the role and the tables
Step 5
For each ip address (X), issue following rules to the tables.
Send table (table T1):
"Match field: dest IP address = X, Action : output N ",
Receive table (table T2):
"Match field: src IP address = X Action : output N' "
Conversion of user assignment
Step 1
Wait for user authentication
Step 2
Receive user assignment (a client IP address and a role) from authentication system
Step 3
Search two tables (send/recv) corresponding to the role
Step 4
Issue following rules to first table
"Match field: src IP =Client IP address Action: Goto table T1"
"Match field: dest IP = Client IP address Action: Goto table T2"

Fig.5 RBAC rule conversion

to access server Y and server Z. In this case, a client (IP = B) has the role and it can communicate with server Y and Z.

C. RBAC Rules Conversion to Flow Entries

In this section, we describe rule conversion from RBAC rules to flow entries. We implement two convert scripts to convert RBAC rules. Permission assignment convert script (PA converter) retrieves permission assignments from the policy repository and issues flow entries to create tables which specify permissions of roles. User assignment convert script (UA converter) retrieves user assignments from the authentication system, and issues flow entries to associate each IP address of clients with the tables corresponding to the roles created by the PA converter.

1) Conversion of permission assignment

A role is converted into two tables; one is for sending packet and the other is for receiving packet. Moreover, for conversion of user assignment, relations between a role and two tables are stored in a database. The PA converter performs following steps for the conversion (Fig. 5). (Step 1)PA converter selects a role in the policy repository. (Step 2)PA converter retrieves IP addresses which are permitted by the role. (Step 3)PA converter selects two non-used flow tables. One is for sending packet to servers and the other is for receiving packet from servers. (Step 4)PA converter stores the relation between the role and the two tables to a database. The format is (role name, table number). (Step 5) For each IP address retrieved at step 2, PA converter issues following flow entries to the table for sending.

Match field: destination IP address = X Action: output N

X is an IP address retrieved at step 2 and N is port number of the OpenFlow switches. Here, N should be determined according to route information, but in this paper we set this value manually. For receiving packets from the server, following flow entry is also issued to the table for receiving.

Match field: source IP address = X Action output N' (port connected with the client)

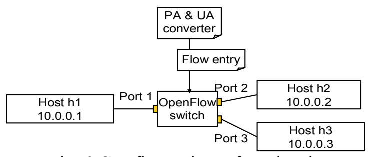


Fig.6 Configuration of evaluation

In this case, the tables of PA directly specify output ports, but it is also possible that the tables of PA refer to a routing table which specifies output ports for each host.

2) Conversion of user assignment

User assignments are converted into flow entries in the first table. A rule in the table identifies a role by IP address and specifies the role using "goto" action to jump to the tables created by PA converter. The table for user assignment is created by following steps. (Step 1)UA converter waits for user authentication. (Step 2)UA converter receives a user assignment (a client IP address and its role) from an authentication system. (Step 3)UA converter searches two tables corresponding to the role in the database created by PA converter. (Step 4)UA converter issues following two flow entries to jump to the specified tables. Here, A is IP address of the client, and T1 and T2 are the table identified at step 3

Rule 1 for send in table 0 Match field: source IP = A (client IP address) Action: goto T1 (a table for sending packets)

Rule 2 for receive in table 0 Match field: destination IP = A (client IP address) Action: goto T2 (a table for receiving packets)

V. Evaluation

A. Feasibility Evaluation

We evaluate feasibility using Mininet [8], which is a network simulator for OpenFlow. Fig.6 describes a network simulated by Mininet. In this network, three hosts are simulated and connected with a software OpenFlow switch. We assume that host h1 is a client and h2 and h3 are servers. For allowing a communication between client h1 and server h2, we input following permission assignment to the policy repository.

(Role, permitted IP address, action) = ("role1", 10.0.0.2, "pass ")

Then, the PA converter executes following commands to issue flow entries.

```
dpctl unix:/tmp/s1 flow-mod cmd=add,table=1 eth_type=0x800,ip_dst=10.0.0.2 apply:output=2
dpctl unix:/tmp/s1 flow-mod cmd=add,table=2 eth_type=0x800,ip_src=10.0.0.2 apply:output=1
```

"dpctl" is a command which inputs a flow entry to the OpenFlow software switch by issuing "flow_mod". In this case, a table for sending is table 1 and a table for receiving is table 2, thus two rules are generated. Specifically, a rule in the table 1 passes the packets only to the server (10.0.0.2) connected with port 2 of the switch. On the other hand, a rule in table 2 passes the packets only from the server.

Next, we assume a situation when the client h1 is authenticated.

```
(IP address , Role)=(10.0.0.1 , "role1")
```

Then, the UA converter retrieves above authentication information and executes following commands to configure the first table (table 0) for a user assignment. First rule specifies "goto table 1" for sending packets from h1 (10.0.0.1), and second rule specifies "goto table 2" for receiving packets.

```
dpctl unix:/tmp/s1 flow-mod cmd=add,table=0 eth_type=0x800,ip_src=10.0.0.1 goto:1
dpctl unix:/tmp/s1 flow-mod cmd=add,table=0 eth_type=0x800,ip_dst=10.0.0.1 goto:2
```

Then, we check whether communications are accepted or rejected using "pingall" command of the Mininet. The result of the command is described blow. Before authentication all commutations are blocked at the switch. After authentication, we can identify h1 (client) and h2 (server) can communicate with each other, however the h1 can not communicate with h3 which is not specified by the rules.

Pingall result before authentication

```
h1 -> X X #h1 can not communicate with h2 and h3
h2 -> X X #h2 can not communicate with h1 and h3
h3 -> X X #h3 can not communicate with h1 and h2
```

Pingall result after authentication

```
h1 -> h2 X #h1 can only communicate with h2
h2 -> h1 X #h2 can only communicate with h1
h3 -> X X #h3 can not communicate with h1 and h2
```

B. Performance Evaluation

Next, for evaluating performance, we estimate the size of ACL rules and RBAC rules. For simplicity, we assume a user has only one role and each server is specified by one role. Furthermore, we assume that rules are specified using IP addresses and we ignore port numbers of TCP and UDP. Size of the ACL rules and RBAC rules are calculated as follows. Here, *C* is the number of clients, *P* is average number of servers allowed by a role, and *R* is the number of roles.

$$\text{Size of ACL rules} = C * P \qquad (1)$$

$$\text{Size of RBAC rules} = C + R * P \qquad (2)$$

For calculating ACL size, we count only "allowable" servers(*P*), instead of all servers (*R*P),* because denied servers would be covered by default deny policy.

For estimating *C, P* and *R*, we captured traffics at a core switch of our laboratories network used by about 300 researchers. First, for estimating *C*, we counted the IP

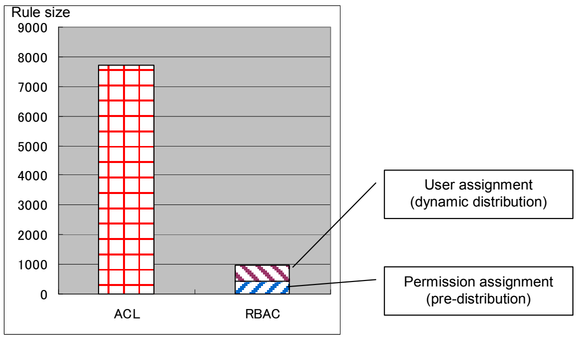


Fig.7 Rule size

addresses which sent SYN packets and we identified C = 517. Next, we estimated size of ACL by counting the number of source IP and destination IP pairs of SYN packets in the captured data. Then we identify ACL size = 7695, and size of RBAC rule is calculated as follows. Here we assume R=30, because the core switch is used by about 30 research teams.

$$P = \text{Size of ACL} / C = 7695/517 = 14.9$$

$$\text{Size of RBAC rules} = 517+30*14.9 = 964$$

$$\text{Rule reduction ratio} = (7695 - 964)/7695=87\%$$

From this information, we estimated the number of rules and we identified that RBAC reduces rule size by 87%. The rule size is less than memory limitation of PF5240 and a reference software switch, thus these switches can store all rules in this case. Furthermore, the half of RBAC rules (permission assignments) can be distributed in advance, thus our architecture reduces the size of dynamically distributed rules by 93% compared with ACL rules (Fig. 7).

VI. Discussion

A. Effectiveness in Case of Using Netmask

In the previous section, we show effectiveness based on per-IP address. However, ACL rule can use netmask for specifying a range of IP addresses. The OpenFlow also can use the mask for the match field of a flow entry. In case of using the mask for aggregating server IP addresses, the mask directly reduces the size of total ACL rules (see equation 1 in previous section). On the other hand, as to RBAC rules, the mask only reduces the size of permission assignment (see equation 2), thus the sizes become as follows.

$$\text{Size of ACL rules} = C * P / N$$

$$\text{Size of RBAC rules} = C + R * P / N$$

, where N is the number of IP addresses covered by a mask. Therefore the efficiency of RBAC depends on the mask, for example in case of N=8, the reduction ratio becomes 40%, and N>14 RBAC does not reduce the rule size. As to the client, the mask reduces the rule size as well as server.

As described above, the efficiency of RBAC depends on the rules, thus as future work, we will evaluate the efficiency using security policies of real network.

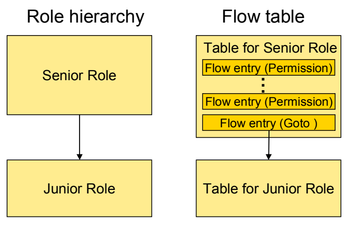


Fig.8 Role hierarchy

B. Limitation of Multiple Role Assignment

RBAC can assign multiple roles to a user so that it can handle a situation where a person belongs to some divisions. For example, when a person belongs to a research division and a sales division, two roles are assigned to the user. However, the proposed design can not handle this situation due to limitation of OpenFlow specification. In our design, a table corresponds to a role, thus multiple tables need to be evaluated for multiple role assignment. However, OpenFlow switch searches a flow entry using "highest-priority match" algorithm which performs only one action in a table, thus it can not jump multiple tables corresponding to the roles. To support multiple roles, we can create a new role which has permissions of multiple roles, but it increases rule size.

C. Hierarchy RBAC

Hierarchy RBAC [7] is extension of RBAC and its role has senior-junior relation, and a senior role also has permissions of junior roles. For example, there are a sales manager role as senior role and a sales common clerk role as junior role, then a sales manager can perform both manager tasks and clerk tasks.

Our design can describe role hierarchy using "goto" action at end of a senior role table and it refers to a junior role table (Fig. 8). First, a table corresponding to senior role is searched. In case that there are no rule which matches a packet in the senior role table, the OpenFlow switch searches second table corresponding to the junior role according to "goto" action. However, there is a limitation of multiple junior roles. A senior role can not have multiple junior roles due to same reason described in multiple role assignment. The OpenFlow switch can execute only one action in a table, therefore we can not specify two or more "goto" actions for simultaneous evaluation of junior role tables.

VII. RELATED WORK

For improving controller performance, parallelization techniques such as multi-threads/multi-core [4][5] has been proposed. Furthermore, HyperFlow [9] has been proposed. HyperFlow is multiple controller architecture to improve scalability and controllers share events using a publish/subscribe system. FlowN [10] is architecture for virtual network solutions such as tenant, and it improves scalability using a lightweight virtualization of a controller and a database technology for virtual-physical mapping. In this paper, we proposed the architecture for load distribution, and it can be used in combination with these techniques for further performance improvement.

As for load distribution, DIFANE [11] is a distributed flow management architecture for scalable networking. It has layered switch architecture (ingress switch, egress switch and authority switch) and distributes wildcard rules for reducing rule size. DevoFlow [12] also distributes wildcard rules to reduce the number of switch-controller interactions. The difference is that we adopt RBAC rules not only for rule size reduction but also for separation of distribution timing and for easy rule writing.

VIII. CONCLUSION

To control whole internal network, we have developed the role-based access control system using OpenFlow. In this paper, for applying the system to large scale network, we propose a load distribution architecture which performs RBAC at network switches for horizontal load distribution. Furthermore, for temporal load distribution, it distributes permission assignments and user assignments at different timings. Moreover, we evaluate its feasibility and performance, and conclude that it reduces 93% of dynamically distributed rule in an ideal case. However, the efficiency depends on network environment such as the number of clients/servers and access control rule such as netmask, thus as future work, we will evaluate its performance in a real network environment. Furthermore, for multi-switch environment, we will develop a rule distribution method to identify a switch which each rule should be distributed to, on the basis of the network topology. For this purpose, we will determine network topologies and system architecture which our method suits for. Tree and star network topology would be suitable, because we need only to distribute RBAC/ACL policies to the central switch. However, mesh topology has many paths, thus we need more sophisticated approach of policy distribution based on location of clients/servers and the network topology.

REFERENCES

[1] Yoichiro Morita et al, Policy composition and distribution methods for IT/NW integrated access control (Japanese), SCIS 2012

[2] OpenFlow, http://www.openflow.org/

[3] OpenFlow 1.3 software switch reference implementation, https://github.com/CPqD/ofsoftswitch13

[4] Andreas Voellmy et al., Scalable Software Defined Network Controllers, SIGCOMM'12,

[5] Amin Tootoonchian et al., On Controller Performance in Software-Defined Networks, Hot-ICE'12

[6] Ravi S. Sandhu et al., Role-Based Access Control Models, IEEE Computer, Volume 29, Number 2, February 1996, pages 38-47.

[7] Ravi Sandhu et al., The NIST Model for Role-Based Access Control: Towards A Unified Standard, ACM workshop on Role-based access control, Pages 47 – 63, 2000

[8] Mininet, http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Minine

[9] Amin Tootoonchian et al., HyperFlow: A Distributed Control Plane for OpenFlow, INM/WREN'10 Proceedings of the 2010 internet network management conference on Research on enterprise networking

[10] Dmitry Drutskoy et al, Scalable Network Virtualization in Software-Defined Networks, IEEE Internet Computing, Issue 99, 2012

[11] Minlan Yu et al., Scalable Flow-Based Networking with DIFANE, SIGCOMM 2010

[12] Andrew R. Curtis et. al., DevoFlow: scaling flow management for high-performance networks, SIGCOMM '11 , Pages 254-265