Programmable Network Using OpenFlow for Network Researches and Experiments

HIDEyuki Shimonishi*, Yasuhito Takamiya*, Yasunobu Chiba*, Kazushi Sugyo*, Youichi Hatano*, Kentaro Sonoda*, Kazuya Suzuki*, Daisuke Kotani**, and Ippei Akiyoshi*

*Cloud Systems Research laboratory, NEC Corp.
1753 Shimonumabe, Nakahara, Kawasaki, Kanagawa 211-8666, Japan, h-shimonishi@cd.jp.nec.com
**Graduate School of Informatics, Kyoto University.
Yoshida-Honmachi, Sakyo, Kyoto 606-8501 Japan

ABSTRACT

We explain what OpenFlow is and how it is used for network researches and experiments, as well as software platform for that. OpenFlow has been proposed as a means for researchers, network service creators, and others to easily design, test, and deploy their innovative ideas in experimental or production networks to accelerate research activities on wired or wireless network technologies. Rather than having programmability within each network node, the separated OpenFlow controller provides network control through pluggable software modules. In this paper, we introduce our OpenFlow programming framework Trema, which focuses on productivity of network experiments and covers an entire development cycle of programming, testing, debugging, and deployment. Then, we introduce some of our experiments including seamless handover between WiFi and WiMAX, multicast video streaming, and network access control.

Keywords: OpenFlow, Software Defined Network, network virtualization, network controller

1 INTRODUCTION

The Internet has evolved to accommodate a variety of services including real-time communication, broadcasting, and content delivery as well as computer-to-computer communication. These services place very diverse demands on the networks. For example, real-time video delivery requires high bandwidth and a low packet loss rate whereas non-real time video delivery requires best effort performance but still high network bandwidth. Accordingly, a number of new technologies, including ones for quality of service (QoS), mobility, security, and traceability, need to be added to the traditional network paradigm.

This has opened up new era of network researches and experiments and led to the creation of a number of initiatives focused on the "Future Internet." The idea is to give researchers, network service creators, and others a way to easily develop, test, and deploy their innovative ideas in a large network infrastructure. These initiatives include the National Science Foundation's Future Internet Design (FIND) [1] and the European Commission's Seventh Framework Programme (FP7) [2].

Large-scale testbed facilities have also been funded to accelerate related research activities. They included the Global Environment for Network Innovations (GENI) [3] project sponsored by the National Science Foundation and the Japan Gigabit Network (JGN-X) [4] testbed sponsored by the National Institute of Information and Communications Technology. Network virtualization technologies, such as slice-based facility architecture [5] or FlowVisor [16], enable researchers to share the testbed simultaneously. Rather than aiming at a one-size-fits-all network, this technology aims at creating a diverse network structure that does not rely on a single unchanging technology. The result is an evolutionary cycle in which a variety of virtual networks are easily created, some of which soon disappear and some of which become widely used. This birth-and-death and natural selection process would promote the continuous evolution of network architectures.

To accelerate such innovation process, technologies separating control and forwarding have been proposed [6-8]. They realize programmability in a separated controller rather than having programmability within each network node. This enables independent evolution of the control plane, which is used to implement a wide variety of control algorithms and has a relatively short evolutionary cycle, and the data plane, which supports faster packet delivery and has a relatively long evolutionary cycle. Among these technologies, OpenFlow has been accepted widely and used in large network testbeds like GENI and JGN-X. OpenFlow defines atomic behaviors for flow handing within each switching element and an interface for manipulating the behaviors from a separate controller. While this idea is not very new, its specific design has several advantages. For example, OpenFlow defines a flow as a set of arbitrary combinations of packet header fields, so it is applicable to flow-based fine-grained control as well as to aggregated control using a destination address or tunnel label.

With these technologies, researchers can easily design, test, and deploy their innovative ideas in experimental or production networks. Programming at a (possibly centralized) controller improves such research productivity, however, a network systems is actually a distributed system consists of controllers, hosts, switches, and other network equipments, and thus, debugging and testing is not that easy. Researchers have to setup their test environment with hosts and switches that are controlled by their controller.

Debugging can be even more troublesome because users need to collect states of all related network components and analyze them.

The contribution of this paper is twofold:

1) We introduce our open source OpenFlow programming framework Trema [9-10], which focuses on productivity of network experiments to accelerate research activities on wired or wireless networks technologies.
2) An independent repository called TremaApps [11], which distributes practical/experimental controllers. TremaApps would be a good starting point for easily developing real-world controllers.

Unlike other OpenFlow controller platform such as NOX/POX [12], Beacon, [13], Floodlight [14], and ONIX [15], uniqueness of Trema is that it heavily focuses on productivity of network researchers. To this end, Trema is not just an OpenFlow controller but an OpenFlow programming framework, which covers an entire development cycle of programming, testing, debugging, and deployment.

Trema provides a platform part of an OpenFlow controller, a.k.a. network OS, and modularized programming framework on top of it. Trema is used by researchers, network operators, or service creators to develop their own OpenFlow controllers with multiple control modules. They can easily develop their own control modules using Ruby or C, or arrange the modules developed by other users. Trema provides equivalent programming APIs for both Ruby and C. This multiple language support is intended for smooth migration from rapid prototyping using Ruby to high-performance implementation using C for deployment at production networks.

Trema includes an integrated network emulator, which consists of pseudo hosts and virtual switches, so that users can easily test a new controller with this network emulator and then seamlessly deploy to production environment. For easier debugging in a distributed system called network, Trema provides an integrated debugging environment that collects state information from all parts of the system. Lastly, to enable seamless operation during entire development process, Trema provides integrated operation environment to configure and operate network emulator, debugger and controller through configuration file, command line interface, and interactive shell.

2 OPENFLOW

2.1 Basics

The OpenFlow protocol supports the programming of various switch behaviors at the flow level. The "Open" means that the interface for externally controlling the switches is open, enabling anyone to participate in modifying the switch functions. The "Flow" means that the control is based on a flow, which can be arbitrarily defined. As shown in Fig. 1, user programs on the controller can perform various network control tasks, including routing, path management, and access control, and add flow entries to the flow tables in the switches. When a packet arrives at a switch, the switch searches for a flow entry matching the packet and performs the actions specified by the entry.

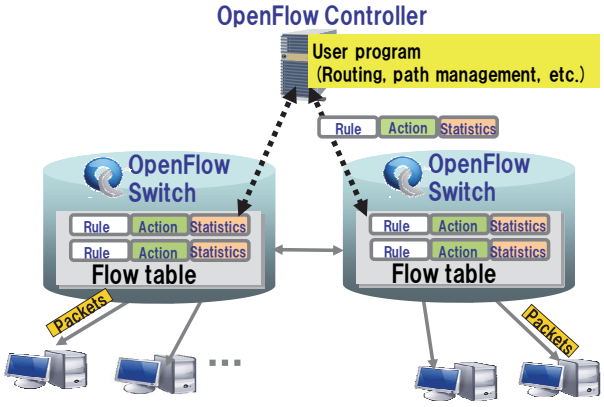


Fig. 1: Illustration of OpenFlow architecture

2.2 Flow and actions

A "flow" can be flexibly defined using arbitrary parts of a packet header, whereas classical switches and routers use only specific parts of the header. The header parts used for flow matching include

- Ingress port (either physical or logical port)
- MAC source/destination address
- Ethernet type
- VLAN id and priority
- IP source/destination address
- IP protocol
- Type of service
- Transport layer source and destination port.

When a packet matches a flow entry, one or more actions are applied, including

- sending the packet to one or more physical ports
- redirecting the packet to the controller
- placing the packet in a specific switch queue, which may have QoS control
- dropping the packet
- modifying specific fields in the header.

Therefore, the behaviors of OpenFlow switches are not limited by the classical layered architecture; for instance, various types of flow entries can be mixed in a switch. For example, the following flow entry emulates the broadcast operation of an Ethernet switch.

Rule: MAC DA = broadcast

Action: OUTPUT = flood

Also, the following entry may be used for IP forwarding.

Rule: MAC DA = MAC address of a router
Ethernet type = IPv4
IP DA = destination host

Action: MAC DA = next hop MAC address
OUTPUT = physical port to next hop

The following entry redirects packets having an HTTP port number to a specific path.

Rule: Dest. TCP port = HTTP

Action: OUTPUT = physical port X

2.3 Design variations

The OpenFlow specifications are flexible enough to support many design variations of the behavior model.

- Reactive vs. proactive

An OpenFlow controller can be reactive by dynamically injecting flow entries when a new flow arrives at the switch. Or, it can be proactive by statically injecting flow entries in advance into the arriving packets.

- Fine grain vs. aggregated

A flow entry can be fine grain, i.e., per TCP/IP session, or aggregated, i.e., per IP destination or tunnel. If the controller runs an IP routing protocol, for example, it creates aggregated flow entries for IP destinations and injects them into the switches proactively.

- Centralized vs. distributed

An OpenFlow controller can be centralized by having a control server control all the switches. Or multiple controllers can be deployed to cooperatively control the network for scalability and redundancy.

2.4 OpenFlow for network experiments

Experiments in network research are generally done by using experimental facilities isolated from production environment; however, experiments in production networks have several advantages, such as using real traffic load for experiments, large and wide area test environment setup, seamless migration from experiments to production operation. OpenFlow with network virtualization enables these kinds of experiments in a shared production network.

For example, as shown in Fig. 2, a class in a university would teach how to program in a network, and then exercise an implementation in a real campus network. Laboratory experiments also share the campus network facility for larger and lively testing. Students and researchers can use their own controller for their office traffic like web browsing or e-mail.

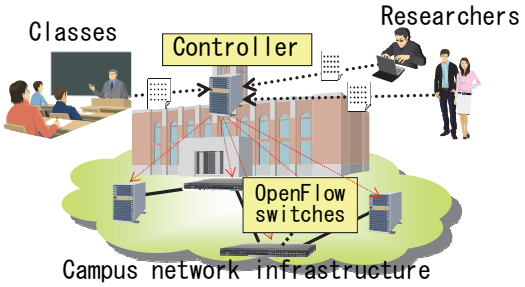


Fig. 2: OpenFlow in campus networks

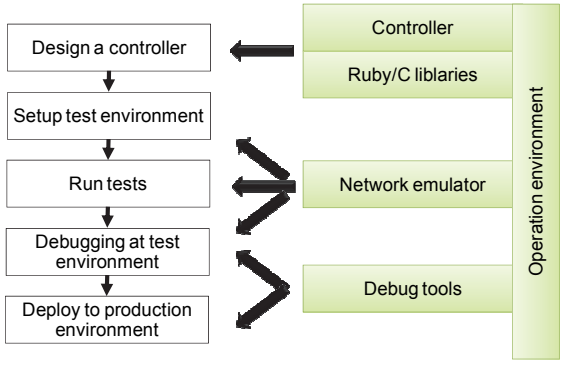
3 OPENFLOW PROGRAMMING FRAMEWORK TREMA

3.1 Trema overview

Trema is an OpenFlow programming framework, which covers entire development cycle of development processes to improve productivity of research activities on wired or wireless networks technologies.

Figure 3 shows a typical development process associated with Trema structure. A user design and develop a controller at a first place, then sets up a test environment and configures the controller to test the controller. If the controller gets any problems, appropriate debugging needs to be done. Then the controller will be switched from the test environment to production environment for its real operation. During the operation, ruining states has to be properly monitored to check the healthiness or possible bugs in the controller.

Trema is designed to cover all these development processes. Therefore, as shown in Fig. 4, Trema framework is composed of these blocks including OpenFlow controller, network emulator, operation environment, and debugging support, which are described in the following.



(a) Typical development process (b) Trema structure

Fig. 3: Typical development process and Trema structure

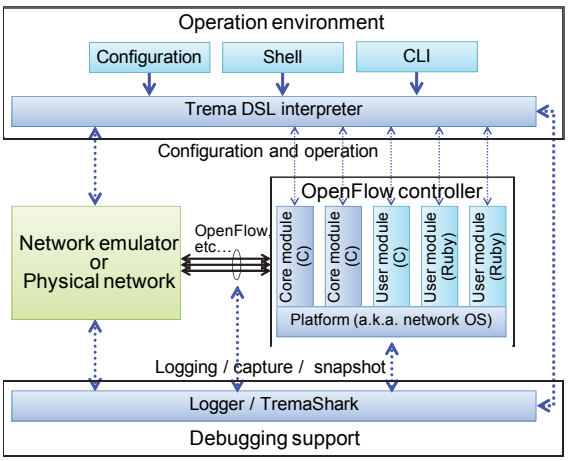


Fig. 4: Trema high-level architecture

3.2 Programming with Trema

Trema provides a platform part of an OpenFlow controller (this part is sometimes referred to as "network OS"), and modularized programming framework on top of it. Users develop their own OpenFlow controllers by collecting some core modules, such as OpenFlow switch manager, included in the Trema framework and user modules developed by themselves.

User modules can be written in Ruby or C. Trema provides equivalent programming APIs for both Ruby and C. This multiple language support is intended for smooth migration from rapid prototyping using Ruby to high-performance implementation using C for deployment at production networks.

Figure 5 shows an example of a Ruby program for a repeater hub controller. When the controller receives a *packet_in* message from a switch, it sends *flow_mod* message back to the switch to instruct it to flood subsequent packets in the same flow, then it sends *packet_out* message to flood the first packet of the flow. The code reads quire smoothly almost like a pseudo code but it is actually an executable Trema program. Trema Ruby APIs are carefully designed to eliminate commonly used repetitive expressions and write the code concise; for example:

- Coding by convention: when the controller receives a *packet_in* message, a handler function named *packet_in* is automatically called. There's no need to write a code to parse and dispatch the messages.
- Default options: OpenFlow messages like *flow_mod* or *packet_out* have many parameters to be specified. Trema API requires specifying only the parameters that is different from the default values.
- Syntactic sugar: *match* structure, which defines a flow with arbitral combination of packet header fields, can easily be extracted from *packet_in* messages using *ExactMatch.from* expression.

```
class RepeaterHub < Controller                  # Create a new controller class

 def packet_in datapath_id, message             # Packet-in received handler
  send_flow_mod_add(
    datapath_id,
    :match => ExactMatch.from( message ),                    <= Send flow_mod
    :actions => ActionOutput.new( OFPP_FLOOD )
  )
  send_packet_out(
    datapath_id,
    :packet_in => message,                                   <= Send packet_out
    :actions => ActionOutput.new( OFPP_FLOOD )
  )
 end
end
```

Fig. 5: Repeater-hub program in Ruby

3.3 Network abstraction and high-level APIs

Trema provides basic APIs for OpenFlow protocol handling, but users may want some high-level APIs for easier development. For example, users may want an API obtaining network topology, rather than directly handling OpenFlow messages to obtain it. Abstracting network components and high-level APIs for them should be extensible and thus they should not be tightly bound to the platform. The modularized programming framework of Trema enables to develop network abstractions as a module structure. As shown in Fig.6, use modules call high-level APIs provided by the abstraction modules such as *Topology management* or *Path management*. These modules can be interpreted as an abstraction layer and user application layer, but they are actually implemented in a flat structure for extensibility. Abstraction layers can be hierarchical, for instance, *Link discovery* module is used by *Topology management* module, which is used by *Routing* module.

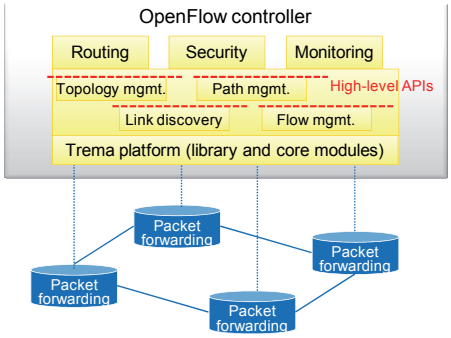


Fig. 6: Network abstraction and high-level APIs

3.4 Network emulator

Trema provides an integrated network emulator, which consists of pseudo hosts and virtual switches, so that users can easily test their controller. After the test, the controller is seamlessly deployed to production environments without modifying any configurations. The controller is always waiting for any switches to connect whether they are virtual ones or physical ones, so deploying the controller is just plugging a network cable to the controller.

The emulator also allows connecting virtual and physical switches, thus physical switches can be integrated in a test environment. In the same way, virtual switches can also be used in a part of production networks.

The advantage of this integrated emulator over other independent network emulators like Mininet [17] is the productivity of the development cycle. During the testing phase, its configuration and operation are integrated with Trema's operation environment to run the test as quick as possible. In a debugging phase, monitoring of running states of the network emulator and the controller are integrated for easier debugging.

3.5 Debugging support

For easier debugging in a distributed system called network, Trema provides an integrated debugging support to collects state information from all parts of the system. In addition to standard logging system, TremaShark [18] enables system wide state monitoring.

As shown in Fig.7, TremaShark monitors any messages and events from any components of the target system. Any messaging (or API call) among control modules in a controller, syslog messages from switches and hosts, packet captures from network interfaces or tap devices, and any text messages are collected and serialized. They are held in a circular buffer for real-time monitoring, or stored in a pcap file for off-line analysis. They are parsed and displayed in a single timeline at a Wireshark [19] terminal with Trema plug-in.

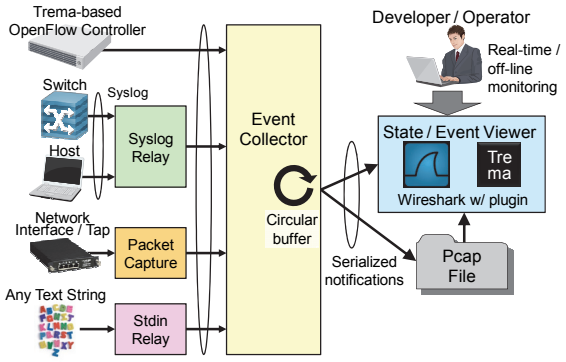


Fig. 7: TremaShark

3.6 Operation environment

To enable seamless operation of all through the development process, Trema provides an integrated operation environment to configure and operate network emulator, debugger, as well as controller modules as shown in Fig. 6. These components are managed through configuration file, command line interface, or interactive shell. The details can be found at Trema tutorials [20, 21].

3.7 TremaApps

An independent repository called TremaApps [11] distributes practical/experimental controllers, which would be a good starting point for developing real-world controllers. For example, Routing Switch controller, which abstracts a mush of OpenFlow switches into a single virtual layer 2 switch and provides end-to-end shortest path packet delivery, is found in the repository. Sliceable Routing Switch emulates multiple virtual layer 2 network slices.

TremaApps also provides independent control modules, such as *Topology* and *Flow Manager*. As discussed in Section 4.3, users would include these modules and develop their own control modules that use these high-level APIs.

4 RESEARCH ACTIVITY EXAMPLES

4.1 WiFi-WiMAX handover

One of the applications of OpenFlow to wireless research area that we have studied earlier is a seamless handover between WiFi and WiMAX. In this scenario, to maximize total wireless capacity utilization, a centralized controller instructs which mobile station should use which access channel based on global utilization information, rather than a mobile station individually selects a channel with its local information. OpenFlow is used to integrate controls of both routing in wired network and selection of wireless channels. When the controller indicates to switch to a different wireless channel, routing path in the OpenFlow network is also changed to reach a new access point or base station.

Figure 8 shows our implementation of the system architecture. WiFi APs and WiMAX BSes have OpenFlow switch module, which are controlled by a centralized controller, to switch packets among multiple wired and wireless interfaces, and mobility module to inform the controller about wireless link information. In the controller, mobile agent modules manages location of mobile stations and makes handover decision based on various information such as wireless link status, population at each AP/BS, signaling from mobile station, to maximize wireless

capacity utilization. OpenFlow module establishes end-to-end path in the wired network part.

Our demonstration system is illustrated in Fig. 9, where a streaming server is sending a video stream to a mobile station via WiFi or WiMAX network. The controller monitors wireless resource utilizations and if WiFi network is crowded, for example, it indicates to switch to WiMAX network and changes the path in the OpenFlow switch network. We have observed almost no packet losses, and thus degradation of video quality, during the handover.

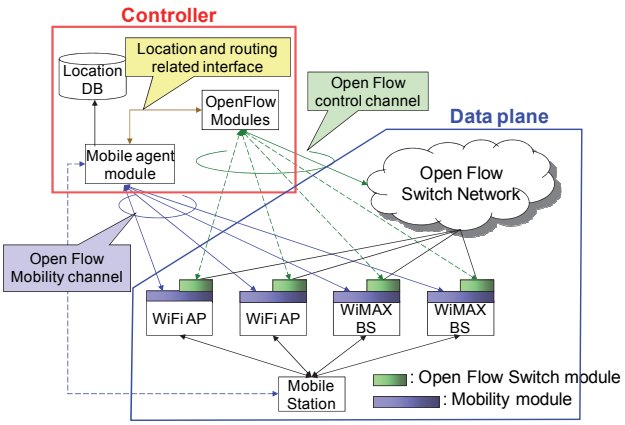


Fig. 8: WiFi-WiMAX seamless handover system

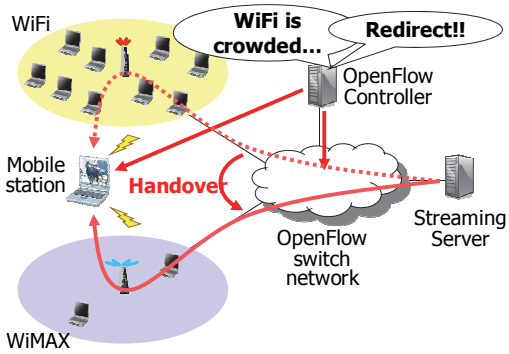


Fig. 9: Demonstration system

4.2 Multicast streaming

Looking at video streaming in wired network part, OpenFlow network is also useful to deliver IP multicast streaming. In a traditional IP network environment, it takes time to reconstruct multicast tress when a switch failure or a link failure occurs because multicast trees cannot be reconstructed until unicast path is stabilized, therefore significant packet losses and hence video quality degradation cannot be avoided. One approach for this problem is to use redundant trees, but algorithms to compute redundant tree require centralized computation. We have been using OpenFlow to realize redundant multicast trees under IP multicast protocols [22]. We have presented a design of an OpenFlow controller supporting IP multicast protocols, e.g. snoops IGMP messages to manage multicast recipient groups, and a method to set up multiple multicast trees for fast tree switching.

To evaluate the proposed method, we setup a test environment consists of NEC IP8800 OpenFlow enabled Ethernet switches and Linux servers for a sender and receivers. As shown in Fig. 10, we have tested with three different topologies having 5, 7, and 9 switches. We sent 30Mbps DV stream from the sender. Figure 11 shows the switching time to the backup tree and the number of lost packet when we intentionally disabled one of the links. The results indicate that multicast packet delivery using OpenFlow is quite robust to network failures and the degradation of video quality is quite limited.

Although our implemented code is not open to public, Simple Multicast found in TremaApps could be some help to researchers who develop multicast controllers.

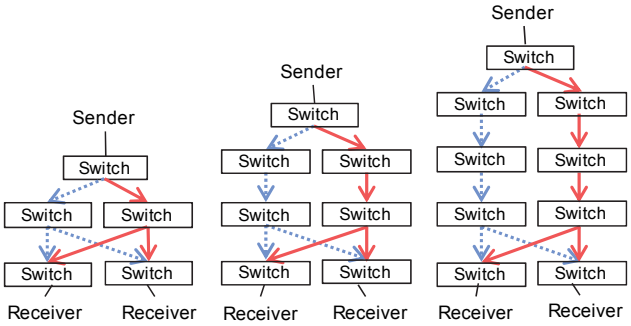


Fig. 10: Network topology

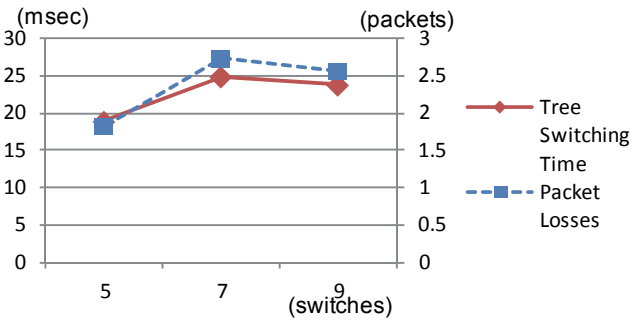


Fig. 11: Evaluation results

4.3 Network access control

Access control to a network, specific hosts, or services, is an important feature of a network. Wireless access points have such functionality to let users access to the network. Firewalls and routers also generally have access control list capability to block specified flows of packets. However,

they only check packets that exactly go though the devices, thus if terminals move around the network or switch to/from wireless and wired accesses, access control may not be applied. In mobile environments, access control must be applied for any packets from any locations to any locations.

OpenFlow can provide "default-deny" type of communications, namely a controller dynamically sets up a path for an authorized flow and other flows are blocked by default. As shown in Fig.12, when a packet of a new flow arrives at any switches in the network, it is sent to the controller. Filtering rule is looked-up with any combinations of packet header fields according to associated priority. If a matched rule is found, its associated action is applied to the flow.

We have evaluated this system using Trema to see the effects of forcing this any-to-any access control. Figure 13 shows RTTs between host A and B with this access control. When we only set exact match rules, which exactly specifies a flow with all applicable packet header fields, no performance degradation is observed regardless of the number of rules because hash search performance is not affected by the number. If we use wildcard matching, which one or more part of the header fields are wildcarded, performance degradation is still negligible when the number of rules is less than 10-100K.

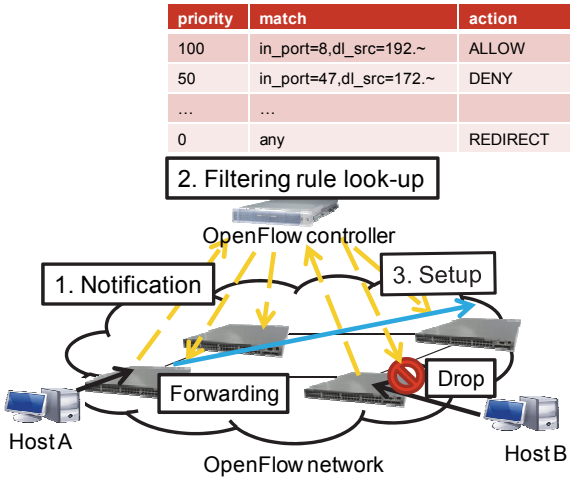


Fig. 12: Access control in OpenFlow network

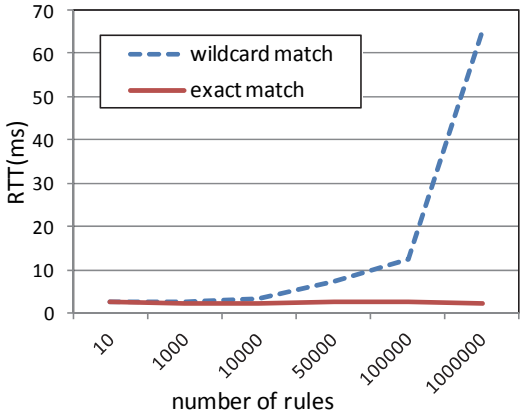


Fig. 13: Observed RTT between hosts

5 CONCLUSION

In this paper, we introduced OpenFlow and its application for network researches and experiments on wired or wireless network technologies. OpenFlow is quite useful for researchers to easily deploy their innovative ideas in experimental or production networks. Programming at a controller improves research productivity and network virtualization technologies help researchers share a large testbed and production facilities. We also introduced our OpenFlow programming framework Trema, which focuses on productivity of network researches and experiments. Trema is designed to cover an entire development cycle of programming, testing, debugging, and deployment. We also touched network abstraction and high-level APIs, as well as TremaApps.

Then, we introduced some of our experiments including seamless handover between WiFi and WiMAX, multicast video streaming, and network access control. In these cases, we have shown a centralized wireless and wired resource control for efficient wireless link capacity, robust multicasting, as well as network access control for highly mobile terminals, which has shown that OpenFlow is quite useful for wireless and wired network researches.

REFERENCES

[1] NSF NeTS FIND Initiative, http://www.nets-find.net/
[2] "Seventh Framework Programme (FP7)", http://cordis.europa.eu/fp7/dc/index.cfm
[3] "GENI",available at http://www.geni.net/
[4] "JGN2plus", http://www.jgn.nict.go.jp/english/index.html
[5] L. Peterson, S. Sevinc, J. Lepreau, R. Ricci, J. Wroclawski, T. Faber, and S. Schwab, "Slice-Based

Facility Architecture", http://www.cs.princeton.edu/~llp/arch_abridged.pdf
[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," SIGCOMM Comput. Commun. Rev., vol.38, no.2. 2008.
[7] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe, "The case for separating routing from routers," in Proc. ACM SIGCOMM Workshop on Future Directions in Network Architecture, 2004.
[8] M. Casado, T. Garfinkel, A. Akella, M. Freedman, D. Boneh, N. McKeown, and S. Shenker, "SANE: A protection architecture for enterprise networks," in Usenix Security, 2006.
[9] "Trema: Full-Stack OpenFlow Framework in Ruby and C", http://trema.github.com/trema/
[10] Trema repository, https://github.com/trema/trema
[11] TremaApps repository, https://github.com/trema/apps
[12] NOX, POX, available at http://www.noxrepo.org/
[13] Beacon, https://openflow.stanford.edu/display/Beacon/Home
[14] Floodlight, http://floodlight.openflowhub.org/
[15] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A Distributed Control Platform for Large-scale Production Networks. In the Proc. of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10), 2010.
[16] Sherwood, R., Gibb, G., Yap, K.-K., Appenzeller, G., Casado, M., McKeown, N., and Parulkar, G, FlowVisor: A Network Virtualization Layer. Tech. Rep. OPENFLOW-TR-2009-01, OpenFlow Consortium, 2009.
[17] Mininet, http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet
[18] Y. Chiba and H. Shimonishi, "Network Debugger: A Unified Tool for Diagnosing Network Controlling Applications", World Telecommunications Congress 2012 Workshop on Software Defined Networks (SDN) and OpenFlow, 2012.
[19] Wireshark, available at http://www.wireshark.org/
[20] Trema Tutorial, 13rd GENI engineering conference, http://groups.geni.net/geni/wiki/GEC13Agenda/TremaTutorial, 2012
[21] Trema Tutorial, 2nd Open Networking Summit, http://opennetsummit.org/talks/ONS2012/shimonishi-mon-trema.pdf, 2012
[22] D. Kotani, K. Suzuki, and H. Shimonishi, "A design and implementation of OpenFlow Controller handling IP multicast with Fast Tree Switching", to be presented at SAINT2012. 2012