INVITED PAPER *Special Section on New Paradigm on Content Distribution and Sharing*

Architecture, Implementation, and Experiments of Programmable Network Using OpenFlow

Hideyuki SHIMONISHI[†a)], ***Member***, **Shuji ISHII**[†], **Lei SUN**[†], ***and*** **Yoshihiko KANAUMI**[††], ***Nonmembers***

SUMMARY We propose a flexible and scalable architecture for a network controller platform used for OpenFlow. The OpenFlow technology was proposed as a means for researchers, network service creators, and others to easily design, test, and virtually deploy their innovative ideas in a large network infrastructure, which will accelerate research activities on Future Internet architectures. The technology enables the independent evolution of the network control plane and the data plane. Rather than having programmability within each network node, the separated OpenFlow controller provides network control through pluggable software. Our proposed network controller architecture will enable researchers to use their own software to control their own virtual networks. Flexibility and scalability were achieved by designing the network controller as a modularized and distributed system on a cluster of servers. Testing showed that a group of servers can efficiently cooperate to serve as a scalable OpenFlow controller. Testing using the nationwide JGN2plus network demonstrated that high-definition video can be delivered through OpenFlow-based point-to-point and point-to-multipoint paths.

key words: *OpenFlow, network controller, distributed system*

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 TCP/IP communications paradigm.

This has 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 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 (JGN2plus) [4] testbed sponsored by the National Institute of Information and Communications Technology. The slice-based facility architecture [5], which supports network virtualization, enables researchers to share a testbed and test their individual ideas. Rather than aiming at a one-size-fits-all network, this architecture 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.

In addition to network virtualization, programmability is a key to accelerating innovation. Rather than having programmability within each network node, having programmability in a separate controller has been proposed [6]–[8]. 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. For example, OpenFlow [6], which is widely used in many campus networks, 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 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-grain control as well as to aggregated control using a destination address or tunnel label. Also, it can easily be migrated from the current network hardware and host because it does not require placing a new label in a packet header.

We have designed, implemented, and tested a programmable network controller that uses OpenFlow. It can be used by network operators, service creators, and researchers to create their own virtual networks. These users can easily create their own network controllers by flexibly arranging control modules provided by other users or ones they have constructed themselves. In addition to flexibility, we also focused on the scalability of the network controller. Since a single server may be sufficient for controlling small enterprise networks, whereas a cluster of servers may be needed for large networks, we designed the controller as a distributed system on a cluster of servers. Unlike other OpenFlow controllers such as HyperFlow [14] and ONIX

Manuscript received February 28, 2011.
Manuscript revised May 30, 2011.

[†]The authors are with System Platforms Researches Laboratories, NEC Corp., Kawasaki-shi, 211-8666 Japan.
[††]The author is with Carrier Network Business Planning Division, NEC Corp., Tokyo, 108-80001 Japan.
a) E-mail: h-shimonishi@cd.jp.nec.com

DOI: 10.1587/transcom.E94.B.2715

[15], which enable homogeneous clustering, the proposed network controller is much more flexible, so the control modules can be distributed to multiple servers ether homogeneously or heterogeneously to maximize processing efficiency.

In this paper, we first describe OpenFlow and discuss its application to content delivery services. We then introduce a model for network virtualization in which various service networks are instantiated as independent virtual networks. These virtual networks have their own slice of the shared data plane, as well as their own control plane, which is implemented as a set of control programs. Next we describe our proposed programmable network controller. Finally, we present performance results demonstrating that a group of servers can efficiently cooperate to serve as an OpenFlow controller, confirming that it is scalable for large and complex networks. We also present results for video streaming over the nationwide JGN2plus network using 17 OpenFlow switches at 10 locations. They show that high-definition video can be transmitted through OpenFlow-based point-to-point and point-to-multipoint paths.

2. OpenFlow and Application to Content Delivery

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 host management, 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.

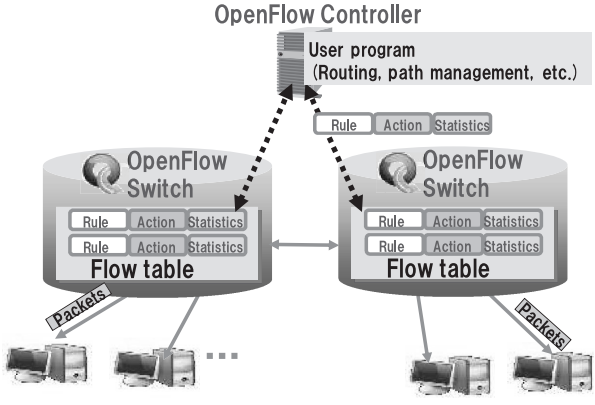


Fig. 1 Illustration of OpenFlow architecture.

2.2 Flow and Action

But what is a "flow," and what is an "action"? 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
Others = any
Action: OUTPUT = flood

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

Rule: MAC DA = MAC address of virtual router
Ethernet type = IPv4
IP DA = destination host
Others = any
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. The flow entry is removed when the flow ends. 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 like OSPF (open shortest path first), 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 or federation.

2.4 Application of OpenFlow to Video Delivery Services

Flow-by-flow control using OpenFlow would be useful for various types of video delivery service, as shown in the following examples. Since their control mechanisms are implemented as control programs on a controller, these examples can be easily deployed without constructing overlay networks or modifying routers.
- High-quality and real-time video delivery
A dedicated path with a QoS guarantee is allocated to a specific video delivery flow. The controller calculates the path and injects flow entries and QoS configurations into the switches. For reliable service, the controller may set up bi-cast or tri-cast flows by instructing the switch to copy the packets to multiple concurrent paths.
- Pseudo-real-time video file transfer
Web-based video streaming would be categorized as this type of transfer. A QoS guarantee might not be needed, but sufficient bandwidth is required for stable streaming. In this case, the controller selects an appropriate path from various path candidates that satisfy the transfer bandwidth requirement.
- Point-to-multipoint streaming
Single-rooted path trees are created for multicasting. OpenFlow switches on the tree branches copy the packets to the multiple links. This may be an easier and more cost-effective solution than IP multicasting or application-layer multicasting because OpenFlow provides optimum layer 2 trees for each video stream. Also, a centralized controller can easily calculate the optimum tree.
- Video cache delivery
Video content transfer from an origin server to the cache servers can be given low priority. The controller may actually calculate a least loaded path, one that even if detoured does not disturb other communications.

3. Network Virtualization Model

3.1 Network Virtualization Based on OpenFlow

A network virtualization mechanism enables the coexistence of various content delivery services and other application-specific services as well as regular IP-based services. While widely used network virtualization techniques, such as VLAN and tunneling, separate only the name space and possibly the bandwidth, control separation

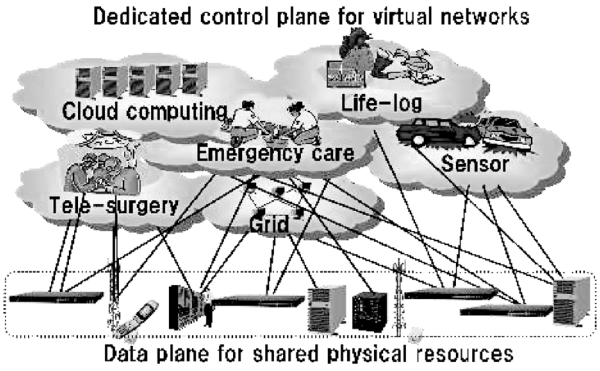


Fig. 2 Model for designing virtual network.

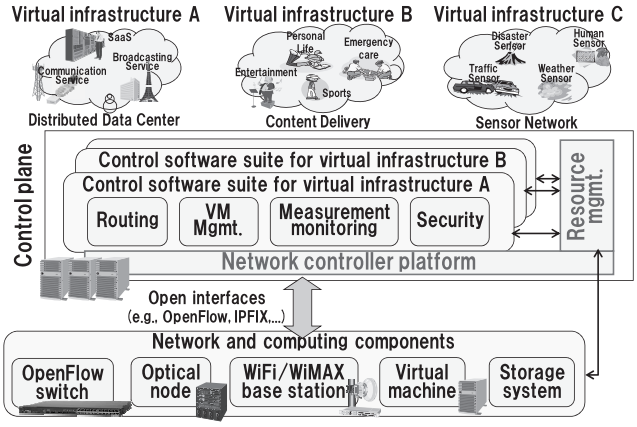


Fig. 3 Control model.

is also needed for the video delivery examples described above to enable variations in flow path control. The model we used for designing our virtual network is shown in Fig. 2. It combines network control using OpenFlow with resource slicing. As shown in the figure, physical resources such as OpenFlow switches, servers, and other network and IT components, are sliced and represented as virtual resources. For example, a virtual OpenFlow switch is a virtualized switch instance having a subset of the links and bandwidth of the physical switch. A virtual network is created by gathering such virtual resources and adding a control plane to control them.

3.2 Control Model

In this virtual network model, programs for controlling the various virtual networks are installed in the controllers. The control programs may or may not cooperate. For example, a virtual network might provide basic layer 2 point-to-point connectivity service while another one provides video delivery service using a connectivity service, so their control programs would cooperate. On the other hand, two video delivery services might be allocated separate connectivity services, so their control programs would not cooperate.

Such a control model is shown in Fig. 3. The control programs for the virtual networks are managed using a soft-

ware platform in the network controller. The platform has three features in particular.

- Modularity

Control programs are modularized as independent software modules for efficient development. A user creates his or her own virtual network by collecting control modules for routing, network measurement, VM migration control, and so on. The platform ensures both isolation and communication among the modules.

- Virtualization and isolation

The platform provides managed communication among the modules of different virtual networks to ensure their secure cooperation. It also manages allocation of virtual resources to the virtual networks for both cooperation and isolation among them.

- Integrated control

The control and data planes are connected using open interfaces including OpenFlow, sFlow [10], and IPFIX [11], so that various types of standard equipment can be included in the network system. The network controller abstracts the complexity of using multiple interfaces to the physical components and provides open APIs to the application programs.

4. Programmable Network Controller

4.1 Architecture

Unlike other control models like Tesseract [9], we do not define a specific network model in our architecture. Instead, we define abstract "service" and "control module" models, as shown in Fig. 4. Since the network model itself is a research target, the network controller should provide a platform on which users can run their own models.

4.1.1 Service

A service is used to control a virtual resource. It is defined as the combination of a data structure with the APIs needed to control the resource. For example, the "OpenFlow Switch" service contains a data structure for managing the switch configuration and a flow table, and APIs for flow table adding/modifying/deleting/showing. A "path management" service contains a data structure for managing a set of existing flow paths, and APIs for path adding/modifying/deleting/showing.

A service may or may not have a direct relationship with a specific physical resource. For example, an OpenFlow switch service is used to control a physical OpenFlow switch while a path management service has no physical substance called "path." A "path" is a virtual instance created by a control module (the "path manager") using a set of OpenFlow switch services.

Therefore, we define a service as an instance recursively created by control modules. As shown in Fig. 4, the control model is structured as an iteration of services and control modules. "Resource manager" and "service repository" describe the structure of the specific control plane, as shown in Fig. 5. The service repository is a registry used to create a mapping of services and control modules. When a control module creates a new service, it registers the service in the service repository with its name, location, and other attributes. The resource manager manages which control module uses which resource that represented as a service.

4.1.2 Control Module

A control module is used to instantiate a service. When a service is used by other control modules, its associated control module is called and the control program of the module is activated to process the service. During the processing, the control module may use other services. This recursive iteration of services and control modules is the essence of the control model.

A control module directly connected to physical resources is called a "component manager." A component manager is used to abstract a physical resource and provide it as a service. For example, an OpenFlow component manager creates a virtual instance of an OpenFlow switch and provides it to other modules as a service. VM migra-

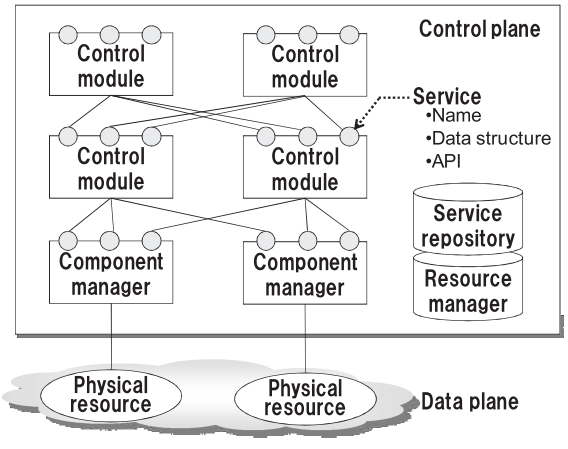


Fig. 4 Basic structure of control model.

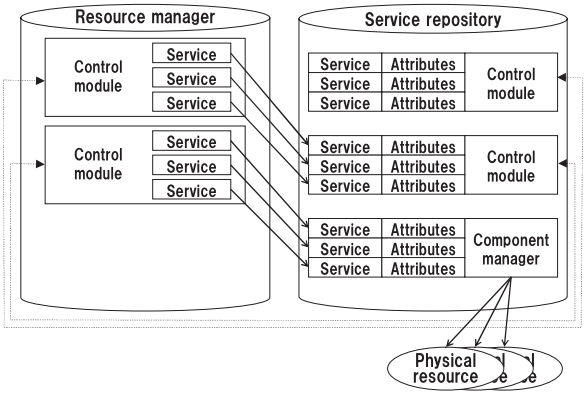


Fig. 5 Service repository and resource manager.

tion control and the IPFIX interface are other examples of a component manager.

4.1.3 Virtualization

A control module can provide multiple services for a physical resource and is thus a mechanism for virtualization (or isolation) of logical resources. For example, an OpenFlow component manager creates multiple OpenFlow switch services for a physical switch, some of which are allocated to a virtual network and others are allocated to other virtual networks. In this case, the OpenFlow component manager ensures isolation of ports and bandwidth among the switch services sharing the same physical switch. While this is similar to the function provided by FlowVisor [12], it provides a more integrated and unified way for resource virtualization, not only for OpenFlow switches but also for other physical and logical resources. For example, path manager module creates two isolated sets of paths as different services. The OpenFlow component manager and path manager module are both used for network virtualization but at different layers. Then a set of switches, in which the path manager module creates edge-to-edge paths, are abstracted into one virtual switch and represented as a logical "OpenFlow switch" service.

4.2 Implementation

We implemented our programmable network controller as a middleware suite on top of Linux servers. As illustrated in Fig. 6, the network controller platform is the basic platform for the distributed system. Each control module is implemented as an independent process on the platform and placed in one of the servers in the controller cluster. All the modules can be placed on one server for controlling a small network, or they can be distributed to multiple servers for controlling a large network. The modules are connected through a server-client system. A control module wanting to use a service calls the messaging API specifying the name of the service. The messaging API refers to the service repository to get the information for the control module providing the service and sets up a connection between these modules to control the service. Whether these modules are running on the same server or not, the messaging API conceals the physical assignment, so the modules can be distributed to any of the servers.

The platform also provides a set of libraries for a specific class of applications, such as OpenFlow or network measurement. Users develop their own control modules using the APIs provided by these libraries.

4.3 Application Example

Figure 7 shows an application example in which two virtual networks are created on a controller platform. Six virtual switch instances are created using three physical OpenFlow switches. Three of them are allocated to virtual network A, and three are allocated to virtual network B. Each virtual network has its own set of control modules.

- The OpenFlow component manager module controls the OpenFlow switches using TCP/SSL and provides OpenFlow switch service to the other modules.
- The OpenFlow distributor module dispatches OpenFlow messages from one module to another. For example, messages provided through the OpenFlow switch service are examined and dispatched to the topology discovery module if they are related to the link layer discovery protocol (LLDP) or switch link information, and they are dispatched to the application layer network (ALN) manager module if they are related to a new flow, the address resolution protocol (ARP), the dynamic host configuration protocol (DHCP), or other control protocols.
- The topology discovery module generates LLDP packets and sends them to the links by sending them to the OpenFlow switch service. It maintains the dynamic link status and provides network topology information as a service to the other modules.
- The path manager module provides path management service. When it receives a path setup request, it identifies a path between the specified source and destination using the topology information provided by the topology man-

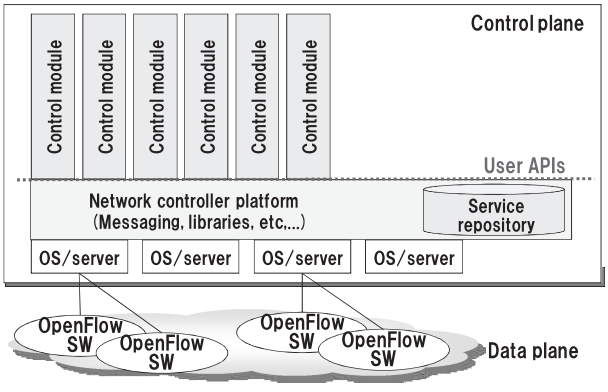


Fig. 6 Network controller implementation.

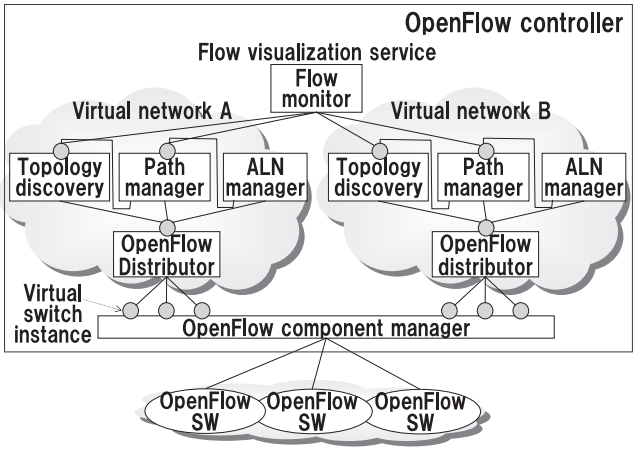


Fig. 7 Application example.

ager. Then it sends flow setup messages to the OpenFlow switches via the OpenFlow distributor and OpenFlow component manager. It also provides its path information as a service for visualization or management purposes.

- The ALN manager module is the main module controlling the network. It has functionalities for host management, subnet management, ARP and DHCP handling, and so on to provide basic layer 2 and layer 3 communications. It receivers various types of packets from the switch services and, when it decides to set up a new flow or to send a packet, it calls the path management service or OpenFlow switch service, respectively.

Independent of the two virtual networks is a flow visualization service for both networks that uses the information from the topology and path management services.

These two virtual networks do not necessarily have the same modules. If one provides a video delivery service, it may have a specialized path manager module that sets up a QoS-aware path and an ALN manager module that provides access control to a video server.

4.4 Distributed System

The example shown in Fig. 7 can be deployed as a distributed system by placing different modules on different severs. This model is different from the ONIX and HyperFlow models because it can support a heterogeneous distributed system in which a module with frequent state updates is not necessarily distributed so as to avoid inefficiency in sharing the state. In this example, the OpenFlow component manager or path manager modules may not be distributed because they maintain flow information, which is updated for each flow setup. In contrast, the ALN manager module maintains host and subnet information, which is modified only when hosts are moved so multiple ALN manager modules can be load-balanced without incurring the overhead of state sharing.

Figure 8 shows an implementation of ALN manager modules distributed to multiple servers. These modules share an information base in the database module. The other modules are placed on sever 1. Dynamic message filtering is implemented in the OpenFlow distributor module for dispatching messages to the ALN manager modules. For example, when an ALN manager module sends an ARP request message, the corresponding reply message is delivered to the same ALN manager module.

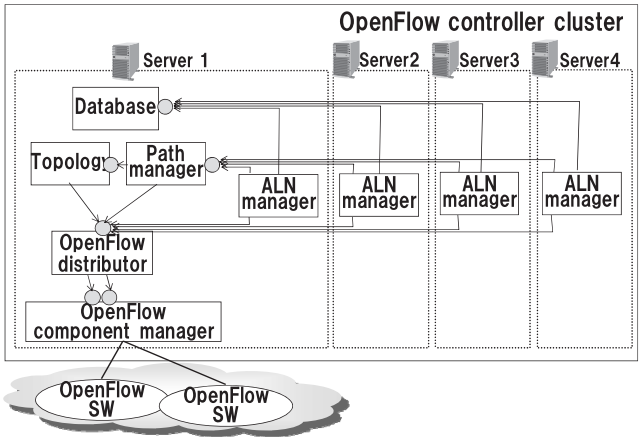


Fig. 8 Example implementation of ALN manager modules distributed to multiple servers.

5. Evaluation

5.1 Four-Server Implementation

We implemented the distributed controller shown in Fig. 8 on four servers and tested its performance. Server 1 had all the modules while the others had only the ALN manager module. Each server had a 4-core 2.40-GHz Intel Xeon processer and 8-GB main memory. Four virtual OpenFlow-enabled switch processes and two virtual host processes were launched on the other servers using OpenVswitch [13] and other related utilities for building an emulation network.

We first ran the four servers to balance the loads on the ALN manager modules, and then we shut down the servers one by one. Before shutting down a server, we reconfigured the OpenFlow distributor module so that the messages to be dispatched to that server would be dispatched to the other servers.

Figures 9 and 10 show the relationship between system resource (CPU and memory) utilization on Server 1 and the total number of servers under three system workloads: 1000, 750, and 500 flow setups per second (fps). As shown in Fig. 9, the CPU utilization on Server 1 changed negligibly as the number of servers was increased from 1 to 4. This is because the processing cost of the ALN managers is not significant and the cost of message passing among the servers is small. As shown in Fig. 10, the memory utilization on Server 1 decreased about 23% as the number of servers was increased from 1 to 4 for fps = 1000. This is because the flow information stored on Server 1 was moved to the other servers. In fairly large-scale networks in which the ALN managers would have many complicated tasks and high workloads, the virtual networks cannot be easily controlled using a single server but they can be using the proposed distributed system.

5.2 Nationwide-Network Implementation

We also implemented the controller on the nationwide JGN2plus network in Japan. We deployed 17 OpenFlow switches at 10 locations in 5 major cities and used them to deliver uncompressed or compressed high-definition video captured at the 2010 Sapporo Snow Festival and at an Okinawa Professional Baseball Camp. Figure 11 shows the physical topology of the network. The OpenFlow switches and legacy L2 switches are respectively shown as dark gray and white boxes. We used IEEE 802.1 tunneling (QinQ) to overlay the OpenFlow network on the existing JGN2plus

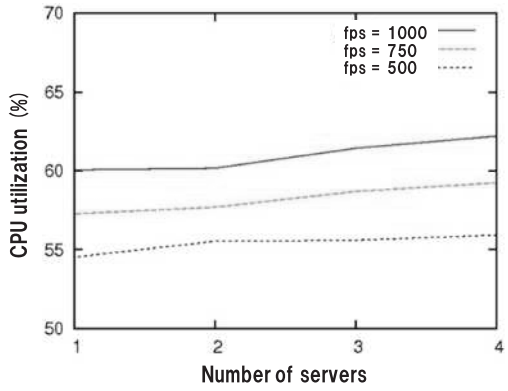


Fig. 9 CPU utilization of Server 1.

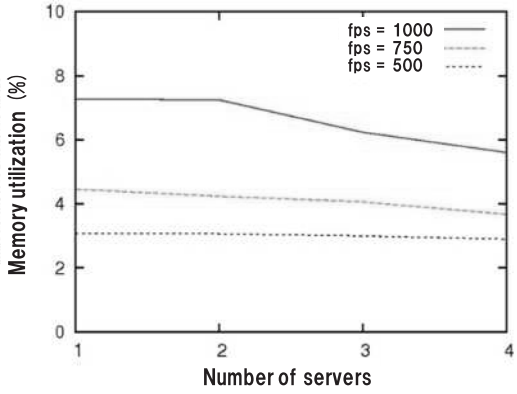


Fig. 10 Memory utilization of Server 1.

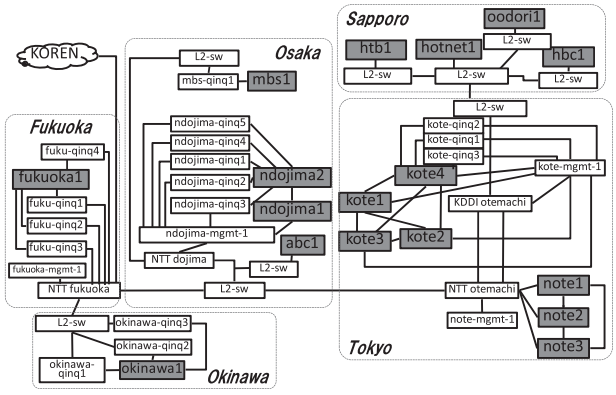


Fig. 11 Physical topology.

Ethernet-based network. Figure 12 shows the logical topology of the OpenFlow switches in the network.

In this experiment, we modified the path manager module to set up point-to-multipoint paths for multicasting the video streams. We also conducted bi- and tri-casting, in which packets are copied to multiple paths at an OpenFlow switch and merged into one stream at a downstream node, to evaluate streaming reliability.

The quality of the video was subjectively evaluated by observing it on a receiver screen, and the frame rate and

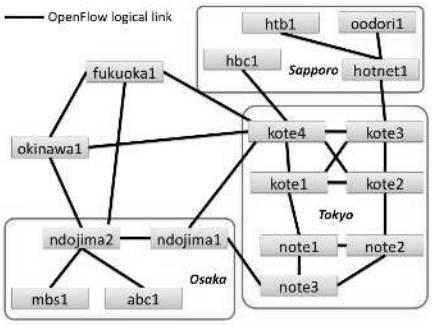


Fig. 12 Logical topology.

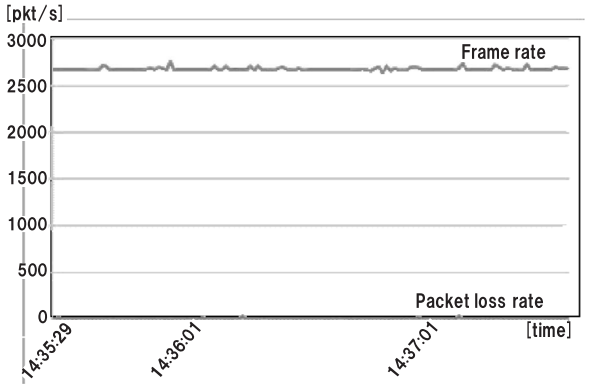


Fig. 13 Sample video stream results.

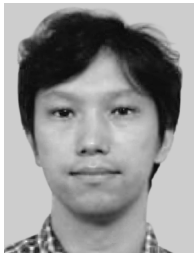
packet loss rate were also measured. The results for one of the video streams from Sapporo to Osaka are plotted for two minutes in Fig. 13. The frame rate was high and stable during the stream, and packet loss rate was quite negligible, resulting in high-quality video reception. The average measured flow setup delay was 7.2 ms. Using the OpenFlow thus provides more flexible and easier video delivery than using IP multicasting and a much shorter flow setup time than with other techniques.

6. Conclusion

We introduced the basics of OpenFlow and described examples of its application to video delivery services. We proposed a flexible and scalable network controller architecture in which users (network operators, service creators, researchers, etc.) can easily design their own network functions and deploy them in a large network infrastructure. We implemented and tested a programmable network controller using OpenFlow. Testing on the nationwide JGN2plus network showed that it is flexible and scalable. Future work includes developing various applications on the proposed network controller to verify the functionality of its architecture. Opening its source code will enable researchers to conduct their own research using it and share their software asset to help with each other.

References

[1] NSF NeTS FIND Initiative, http://www.nets-find.net/
[2] "Seventh Framework Programme (FP7)," available at http://cordis.europa.eu/fp7/dc/index.cfm
[3] "GENI," available at http://www.geni.net/
[4] "JGN2plus," available at 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," available at 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, pp.69–74, 2008.
[7] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe, "The case for separating routing from routers," 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," Usenix Security, 2006.
[9] H. Yan, D.A. Maltz, T.S. Eugene Ng, H. Gogineni, H. Zhang, and Zheng Cai., "Tesseract: A 4D network control plane," I Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI'07), 2007.
[10] P. Phaal, S. Panchen, and N. McKee, "InMon corporation's sFlow: A method for monitoring traffic in switched and routed networks," Internet Engineering Task Force, RFC-3176.
[11] "IP Flow Information Export (ipfix)," Internet Engineering Task Force, http://www.ietf.org/dyn/wg/charter/ipfix-charter.html
[12] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "FlowVisor: A network virtualization layer," Tech. Rep. OPENFLOW-TR-2009-01, OpenFlow Consortium, 2009.
[13] "Open vSwitch — An Open Virtual Switch," available at http://www.openvswitch.org
[14] 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," Proc. 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10), 2010.
[15] A. Tootoocian, Y. Ganjali, HyperFlow, "A distributed control plane for OpenFlow," Proc. NSDI Internet Network Management Workshop/Workshop on Research on Enterprise Networking (INM/WREN), 2010.



Hideyuki Shimonishi received M.E. and Ph.D. degrees from the Graduate School of Engineering Science, Osaka University, Osaka, Japan, in 1996 and 2002. He joined NEC Corporation in 1996 and has been engaged in research on traffic management in high-speed networks, switch and router architectures, and traffic control protocols. As a visiting scholar in the Computer Science Department at the University of California at Los Angeles, he studied next-generation transport protocols. He now works in NEC's System Platforms Research Laboratories, engaged in research on technologies for future Internet architectures including OpenFlow.

Shuji Ishii received an M.E. degree from the Graduate School of Computer Science, University of Electro-Communications, Tokyo, Japan, in 1992. He joined NEC Corporation in 1995 and now works in NEC's System Platforms Research Laboratories. He has been engaged in the development of the IPv6 (IPSec) protocol stack for routers and hosts at NEC as well as research on software architectures for the OpenFlow controller.

Lei Sun received B.E. and M.E. degrees from the Department of Computer Science and Technology, Tsinghua University, Beijing, P. R. China, in 2001 and 2004 and a Ph.D. degree from the Department of Computer Science, Waseda University, Tokyo, Japan, in 2010. He worked as a research associate at the Research Institute of Open Source Software, Waseda University, and is now a researcher in NEC's System Platform Laboratories.

Yoshihiko Kanaumi received an M.E. degree from the Graduate School of Engineering Science, Osaka Prefecture University, Osaka, Japan, in 1998 and is currently a Ph.D. candidate in the Graduate School of Engineering, University of Tokyo, Tokyo, Japan. He joined NEC Corporation in 1998 and now works in NEC's Carrier Network Business Planning Division. He has been engaged in the development of hardware architectures for routers and of SDH/SONET, ATM, and IP networks. He has also been involved in research at the National Institution of Information and Communication Technology on the operation and management of the Future Internet, including the OpenFlow controller.