Network Abstraction Model Achieves Simplified Creation of SDN Controllers

KOIDE Toshio, ASHIDA Yuta, SHIMONISHI Hideyuki

Abstract

The advent of SDN has removed the constraints brought about by rigid control protocols, thereby enabling a flexible description of control software as an SDN controller, which meets the needs of network operators. Further simplification in creating SDN controllers will become an issue in the need to create more SDN controllers quickly. This paper proposes a network abstraction model that makes it possible to simplify the description for the virtualization of network resources and the implementation of complex path control algorithms while absorbing differences in control protocols between network devices. It is also shown that an SDN controller can be created quickly and simply using the SDN controller platform prototype that was designed based on the proposed model.

Keywords

SDN, SDN controller, OpenFlow, OpenFlow controller, network virtualization

1. Introduction

SDN (Software-Defined Networking) that achieves flexible control of information communication networks by using software implemented with common programming languages has recently been attracting attention. One actual example is network control using OpenFlow[1], which is being standardized by the ONF (Open Networking Foundation)[2]. Conventionally, construction and administration of information communication networks have been carried out with combination and configuration of dedicated network devices that incorporate network control and data processing functions. This makes it difficult for operators to change the control functions flexibly, thereby posing problems such as inability to quickly cope with the new requirements for advanced network virtualization and visualization that has not been an issue hitherto. The situation is now being improved thanks to the increasing availability of suitable environments, spurred on by the advent of technologies such as OpenFlow, whereby control functions are separated from network devices and they can be written in common programming languages as SDN controllers (**Fig. 1**).

Moreover, when attempting to acquire an SDN controller that reflects the administration policy of each network, new programming skills will be required for the flexible control. For the easy migration to SDN, it is desirable to simplify the procedure to create SDN controllers in order to avoid overemphasis on programming skills. Furthermore, although the concept of SDN enables success by using various types of network other than OpenFlow, such as the GMPLS network and IP-VPN, it is desirable to use a virtualization method that is not affected by differences in network types. Independent re-implementation according to the network type and compli-

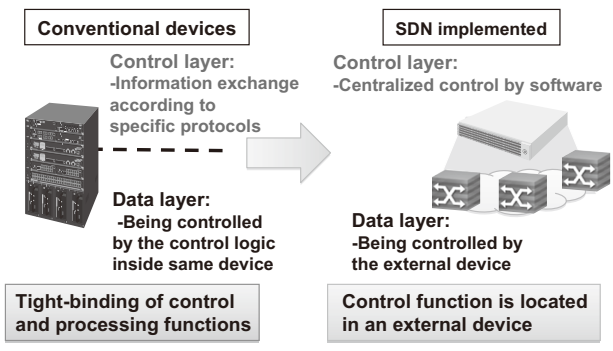


Fig. 1 SDN achieved by separation of control layers and data layers.

cated network controls that extend over multiple network types is thereby avoided.

Proposed in this paper is a network abstraction model that aims to facilitate creation of an SDN controller and to absorb the differences between network types. The proposed model will facilitate connection of functional groups required for the creation of SDN controllers by providing such functions in advance as general-purpose software components. It will also facilitate hiding the differences between lower network types while enabling creation of SDN controllers by simple operation of the assembly components. This paper also explains the design concept of the prototype that has been implemented in order to confirm project feasibility.

2. Network Abstraction Model

A simple yet universally effective network abstraction model is described below.

In developing an SDN controller, the ability to control the overall system that extends over multiple switches - in other words, to grasp the network topology and control the packet transfer paths extending over multiple switches - will be more important than setting each switch according to the protocol. Also required is a dynamic optimization responding to changes in the network conditions. We designed a network abstraction model of optimal scale that can observe and control the entire network as well as can dynamically control the network in response to changes of the network conditions within the observed view. In our model, the network conditions are defined by three types of data: topology, a flow group and a packet group (**Fig. 2**). The network changes are represented by event notifications. Not like the flow entry regulated in OpenFlow; the "flow" mentioned in Fig. 2 is a concept that encompasses transfer paths across the entire network. Furthermore, a data model that comprehensively represents the conditions of the entire network is defined. By expressing network control using the operation of that data, we have achieved the separation of network control from the protocols of the physical layer. **Fig. 3** shows an example of these expressions as common hash objects.

- **Topology (Fig. 3 (a))**
  The topology consists of network identifiers, lists of the nodes and lists of the links. The node comprises the node identifiers and the lists of the communication ports. The link is composed of the link identifiers, the source nodes/communication port identifiers and the destination nodes/communication port identifiers.
- **Flow group (Fig. 3 (b))**
  A flow group is composed of multiple flow information

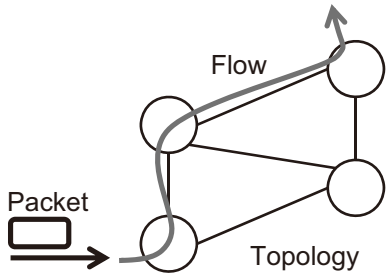


Fig. 2 Conceptual diagram of topology, flow, and packets.

```
{
"network": "network1",
"nodes": {
"1": {
 "id": "1",
 "ports": {
  "1": { "id": "1", "out_link":"1", "in_link":"3" },
  "2": { "id": "2", "out_link":null, "in_link":null },
 }
},
"2": {
 "id": "2",
 "ports": {
  "1": { "id": "1", "out_link":null, "in_link":null },
  "2": { "id": "2", "out_link":"2", "in_link":"4" },
 }
}
},
"links": {
"1": { "id":"1", "src_node":"1", "src_port":"1",
                 "dst_node":"2", "dst_port":"3" },
"2": { "id":"2", "src_node":"2", "src_port":"2",
                 "dst_node":"1", "dst_port":"4" },
}
}
```

(a)

```
{
  "1": {
    type: "BasicFlow",
    id: "1",
    match: {
      in_node:"1",
      in_port:"1",
      dl_dst:"66:55:44:33:22:11"
    },
    path: ["1", "2", "3"],
    edge_actions: {
      node1: [{"output":"2"}, {"output":"3"}],
    }
  },
  "2": {
    ...
  },
  ...,
  "priority": ["2", "5", "1", "3"]
}
```

(b)

```
{
  "data": <packet_data>,
  "node": "1",
  "port": "5",
  "time": 1353397038
}
```

(c)

```
{
  "event_type": "PortChanged",
  "node": "1",
  "port": "2",
  "action": "update",
      "old": {"id": "2", "out_link":null, "in_link":null},
      "new": {"id": "2", "out_link":"5", "in_link":"6"}
}
```

(d)

Fig. 3 Expression examples based on the network abstraction model.

and flow priorities. Flow information consists of flow identifiers and objects describing about the processing rules of these flows. The processing rules are comprised of "match": the packet conditions to be processed, "path": the transmission path of the packet and "actions": the processing actions of the packet. The path is expressed in the list of the link identifiers, and the priority of the flows is expressed by order according to the list of the flow identifiers.

- **Packet group (Fig. 3 (c))**
  A packet group consists of lists of input packets and output packets respectively. Packet information is comprised of node identifiers, communication port identifiers, packet headers and payloads. The reception time is also added to the input packets.
- **Event (Fig. 3 (d))**
  Information changes occurred in the topology, flow group and packet group are informed to the components according to the events. Using these events, each component perceives the changes of the network in real time thereby enabling an immediate response.

3. Design of the SDN Controller Platform

Based on the proposed network abstraction model, we have designed and implemented a prototype of an SDN controller platform. Using this prototype, the simplification of SDN controller creation is verified below. Also with this prototype, each component that comprises the SDN controller is defined based on our network abstraction model.

Components can be considered as being divided into two classifications: the network component and the logic component. The network component is the one comprised of a database that maintains network conditions based on the above-mentioned network abstraction model. The logic component is the component that controls the data in the network components by connecting to one or more network components. Although the purpose of the logic component may vary, it may be roughly classified into ones that perform network abstraction, modification and control.

A basic SDN controller can be created simply by connecting the network components and logic components defined above as if they were Lego blocks. This procedure enables creation of SDN controllers and the administration of networks even by users who have no knowledge of the physical devices that comprise the networks or their control protocols.

Fig. 4 shows the configuration example of SDN controller by combining logic components. These logic components include the driver that abstracts the respective network domains

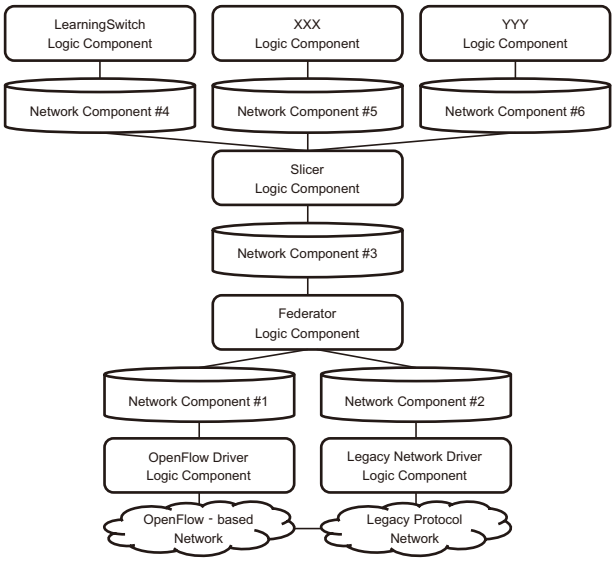


Fig. 4 SDN controller configuration example.

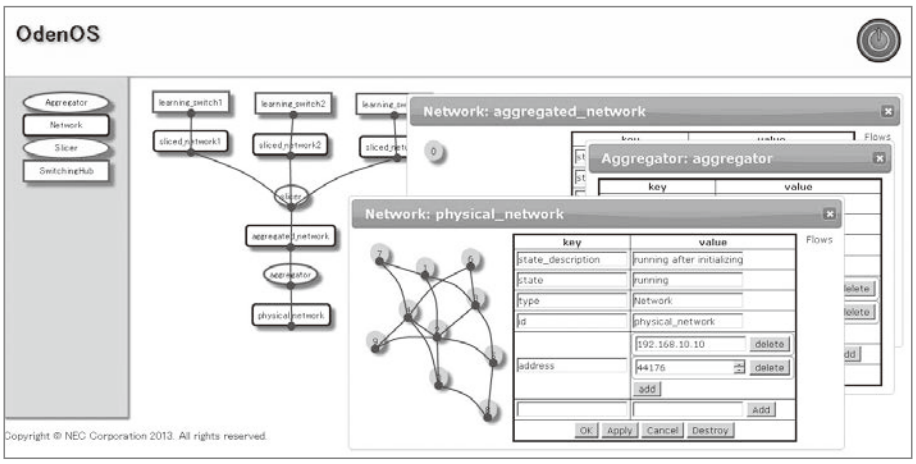


Fig. 5 Creation of an SDN controller of GUIs and visualization of networks.

of the different control systems, the federator that integrates them and composes them as one network, and the slicer that slices them further into multiple users (control logic). Simply by combining standard components, the user can even create a high-function SDN controller that achieves integration of multiple domains via different physical network control protocols, construction of virtual networks, and application of path control algorithms that vary according to the virtual network.

Furthermore, our current prototype is provided with GUIs whose components can be operated via the Web with a view to further simplifying the creation of SDN controllers. We have also confirmed that the user can create an SDN controller by freely assembling these components on a browser. It is also expected that, in addition to its capability of creating an SDN controller, our prototype will develop into an administration support tool in order to display the topology information gathered by each network component on GUIs (**Fig. 5**).

4. Conclusion

This paper has proposed a simple yet universal network abstraction model and described the design and implementation of our prototype of an SDN controller platform based on this proposal. We have also confirmed that, through the implementation of the prototype, the creation of a SDN controller is possible with just a mouse click on a Web browser.

Based on our prototype implementation we plan to increase the component groups and also to improve the network abstraction model and interface specifications by implementing advanced components that can be applied to actual use cases. We also plan to accommodate such components in the actual environments.

5. Acknowledgement

This research was partially supported by the Ministry of Internal Affairs and Communications with a contract entitled “R&D for fundamental technology for energy-saving network control compatible to changing communication status” in FY2013.

* OpenFlow is a trademark of registered trademark of Open Networking Foundation.

Reference

1) N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: enabling innovation in campus,” SIGCOMM Comput. Commun. Rev., vol.38, no.2, 2008.
2) Open Networking Foundation, http://www.opennetworking.org/

Authors’ Profiles

KOIDE Toshio
Assistant Manager
Knowledge Discovery Research Laboratories

ASHIDA Yuta
Knowledge Discovery Research Laboratories

SHIMONISHI Hideyuki
Principal Researcher
Knowledge Discovery Research Laboratories

Information about the NEC Technical Journal

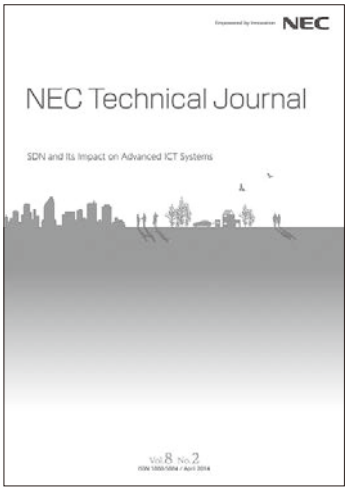
Thank you for reading the paper.
If you are interested in the NEC Technical Journal, you can also read other papers on our website.

Link to NEC Technical Journal website

Japanese

English

Vol.8 No.2 SDN and Its Impact on Advanced ICT Systems



Vol.8 No.2
April, 2014

Special Issue TOP