

Network Working Group
Internet-Draft
Intended status: Informational
Expires: December 2012

M-K. Shin
K-H. Nam
ETRI
M. Kang
J. Choi
Korea U.
June 29, 2012

Formal Specification for Software-Defined Networks (SDN)
draft-shin-sdn-formal-specification-01

Abstract

This document discusses formally verifiable networking framework for software-defined networks (SDN). In SDN, incomplete or malicious programmable entities could cause break-down of underlying networks shared by heterogeneous devices and stake-holders. Formally verifiable networking can provide a logic-based framework to unify the design, specification, verification, and implementation of SDN. This framework describes formal specification and verification process for SDN. In addition, we present two examples of formal specification for a part of SDN using a process algebra called Algebra of Communicating Shard Resources (ACSR) and Z specification language.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on September 1, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Formally Verifiable Networking Framework for SDN	3
3. Formal Specification for SDN	4
4. Examples	6
4.1 Formal Specification of SDN using ACSR	6
4.2 Formal Specification of SDN using Z	8
4.3 Subtle Ambiguities Discovery and Correctness Verification	9
5. Security Considerations	10
6. Acknowledgements	10
7. References	10
7.1 Normative References	10
7.2 Informative References	10
Authors' Addresses	12

1. Introduction

Software-defined networking (SDN) is emerging and intensively discussed as one of the most promising technologies to introduce network virtualization within data centers, enterprise networks, mobile networks, etc. [I-D.nadeau-sdn-problem-statement],[b-OpenFlow]. SDN is defined as a new networking approach which enables network operators and/or application/service providers to add their own processing, control, program, etc. through open network interfaces and network abstraction into their networks that they can control and manage the slicing and virtualization of the networks. With SDN, network operators and application/service providers can introduce a new capability easily by writing a simple software program.

To design and implement networks that conform to the design goals of SDN network topology, the structure and behavior of the networks need to be formally specified to prevent from misinterpreting of the intended meanings and to avoid inconsistency in the networks. Furthermore, SDN networks can be used for safety-critical systems such as avionic and automotive systems, nuclear power plants, and medical devices, and those systems should be verified to guarantee their reliability and security properties, otherwise catastrophic disaster could be occurred. Other areas that the SDN networks can be applied, such as private clouds and data centers, also benefit from formal specification and analysis since any inconsistencies in the systems and unexpected errors that could not be caught during design process can result in network breakdown or system failure, which can lead to tremendous commercial loss [b-Nam12].

This document discusses formally verifiable networking framework for SDN. Formally verifiable networking can provide a logic-based framework to unify the design, specification, verification, and implementation of SDN. This framework presents formal specification and verification process for SDN.

2. Formally Verifiable Networking Framework for SDN

SDN network operators and application/service providers can introduce a new capability by writing a simple software program. In SDN, incomplete or malicious programmable entities could cause break-down of underlying networks shared by heterogeneous devices and stakeholders. Formally verifiable networking in SDN can reduce any inconsistency or misunderstanding of the meaning of components and mechanisms because formal specification removes ambiguity in the informal specifications. Furthermore, formal specification can be applied to verification methods such as theorem proving, process

algebraic analysis, model checking, and static analysis.

Figure 1 illustrates an overview of the formally verifiable networking framework for SDN, which consists of the three components, formal specification, verification methods, and implementation. SDN network operators and application/service providers design an abstract network model (e.g., virtual network topology) of desired properties informally. After then, the SDN network operators and application/service providers write down formal specification for the properties, which finally verifies that implementation (e.g., SDN control software) satisfies these properties.

In general, traditional methods of realizing network protocols and devices are based on community agreements of informal specification of such mechanisms. As depicted in Figure 1, those processes can be improved by applying formal methods in the development process of SDN. Targets of specification can range from conceptual model of components or mechanisms for SDN, logical switch/router models, network protocols, user-defined topologies of virtual networks, and so on. Informal specification of those targets can be encoded in formal specification languages that can best reflect the features of targets among the existing methods for formal specification. The formal specification can be textual form or graphical representation only if their semantics are defined formally and unambiguously. Once the specifications are described formally, system and protocol designer can check the existence of inconsistencies and possible errors in the specification with the help of formal methods experts or supporting tools. Any type of formal verification methods can be applied to this validation and verification process while each has pros and cons for this purpose. One may use theorem proving such as HOL, Isabelle, Coq, PVS with the help of assistant tools and experts, others can take advantage of full automation of this process by specifying important properties in temporal logics and feed them into model checking tools such as SPIN and SMV [b-Nam12].

3. Formal Specification for SDN

We discuss formal specifications about virtual network topology of SDN. The two researches that are most closely related to our work are NetCore [b-NetCore12] and NDlog [b-NDlog11]. But each has different perspectives. NetCore, the Network Core Programming Language, is a high-level and declarative language for expressing packet-forwarding policies and has a formal semantics. Network Datalog (NDlog) is distributed recursive query language used for querying network graphs.

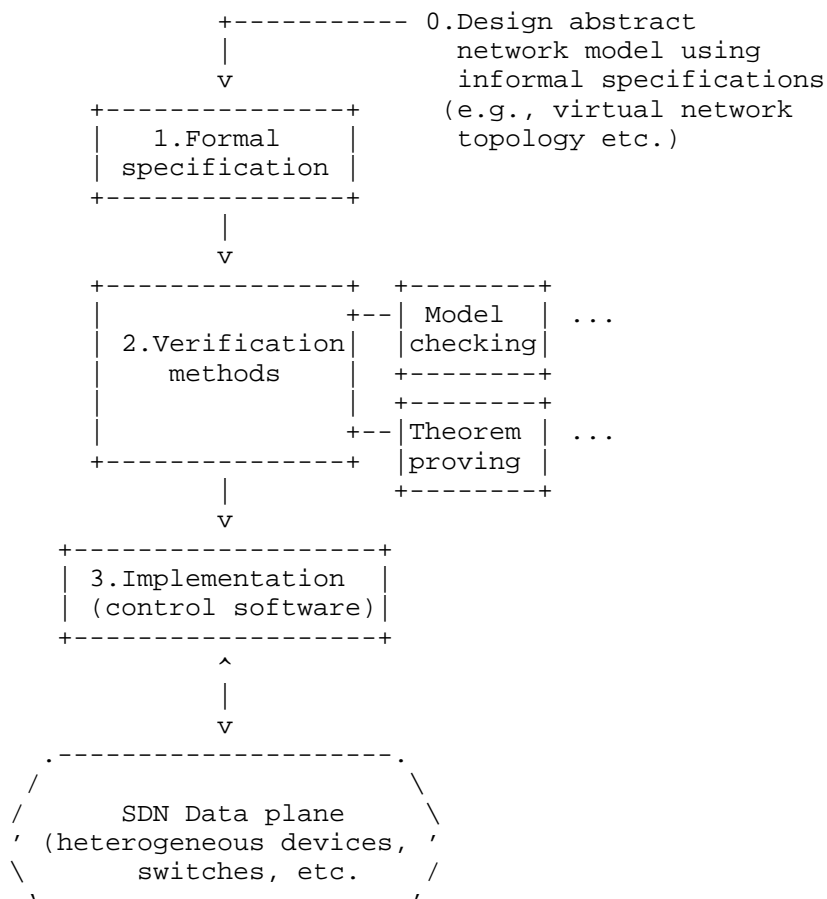


Figure 1 Formally verifiable networking framework for SDN

We consider developing a new formal specification language to accommodate various requirements of SDN networks and properties, if necessary. At this moment, we use process algebra ACSR (Algebra of Communicating Shared Resources) [b-ACSR95] and Z specification language formally, as examples. To provide a correct and efficient solution for forwarding packets on the SDN, ACSR can express processes running concurrently and communicating switches and a controller. Forwarding packets can be modeled as prioritized synchronization of events in ACSR. In addition, Z specification for SDN is focused on each switch and controller for emphasis on their functionality. Based on this, we are researching to verify the OF/SDN through the analysis of the requirement for OpenFlow.

4. Examples

4.1 Formal Specification of SDN using ACSR

This clause describes an example of ACSR specification of SDN. In our process algebraic approaches, network entities are represented by processes in ACSR.

ACSR is a formal specification and verification methods for behavior modeling using concepts of processes, resources, events, and priorities. ACSR, like other process algebras, consists of (1) a set of operators and syntactic rules for constructing process; (2) a semantic mapping which assigns meaning or interpretation to processes; (3) a notion of equivalence or partial order between process; and (4) a set of algebraic laws that allows syntactic manipulation of processes. ACSR uses two distinct action types to model computation: time and resource-consuming actions, and instantaneous events.

ACSR distinguish two types of actions: those which consume time, and those which are instantaneous. Timed actions may require access to system resources, e.g., network bandwidth etc. In contrast, instantaneous actions provide a synchronization mechanism between concurrent processes. In this document, we use only instantaneous action to model SDN.

Packet forwarding is specified as event sending and receiving. Packet matching with rules are represented by synchronization between input and output events. We provide a demonstration of ACSR specification of an example virtual networks.

Let the example virtual networks have a topology shown in Figure 2. The topology consists of a single switch and three hosts (H1, H2, and H3).

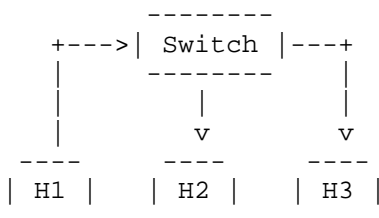


Figure 2 Example virtual topology

An abstract view point is used to model packets by abstracting out

all detailed data in the packets. We assume that there are several types of packets forwarded between nodes in the networks as follows:

Sender	Receiver	Types of packets
H1	Switch	packet1, packet2, packet3
Switch	H2	packet4
Switch	H3	packet5, packet6

We also assume that there are three types of rules in switch which are rule1, rule2 and rule3. Packets are matched to rules in the way as follows. The larger number indicates the higher priority. Note that packet1 can be matched to both rule2 and rule3 and packet2 can be matched to both rule2 and rule3.

Rule	Priority	Packets matched	Action on matching
rule1	4	packet1	action1
rule2	3	packet1	action2
rule2	3	packet2	action2
rule3	6	packet2	action3
rule3	6	packet3	action3

Action	Description
action1	output packet4 to H2 through outPort1
action2	output packet5 to H3 through outPort2
action3	output packet6 to H3 through outPort2

The process 'Network' represents the example system being specified. 'Network' consists of H1, H2, H3, and Switch, which are composed using the parallel operator because they are running in parallel and interacting each other. The specification of the process 'System' is as follows:

```
Network = (H1||H2||H3||Switch)
          \{inPort,outPort1,outPort2,activatingRule1,
            activatingRule2,activatingRule3};
```

In an example topology in Figure 2, we assume that there are three hosts which are specified as ACSR processes 'H1', 'H2' and 'H3'. H1

transmits to Switch packets such as packet1, packet2, and packet3. H2 receives packets such as packet1 and packet3. H3 receives packets such as packet1 and packet2.

```
H1 = (inPort!1,1).(inPort!2,1).(inPort!3,1).Host1;
H2 = (outPort1?packet,1).Host2;
H3 = (outPort2?packet,1).Host3;
```

The switch in the example network consists of 'InputModule', 'FlowTable', and 'OutputModule'. The specification of 'Switch', 'InputModule', 'FlowTable', and 'OutputModule' are as follows:

```
Switch = (InputModule||FlowTable(1,1,0)||OutputModule)
        \{rule1,rule2,rule3,rule0,action1,action2,action3};

InputModule = (inPort?packet, 1).(
  cket = 1) -> ((rule1!,1).InputModule + (rule2!,1).InputModule)
+ (packet = 2) -> ((rule2!,1).InputModule + (rule3!,1).InputModule)
+ (packet = 3) -> (rule3!,1).InputModule
+ (rule0!packet,0).InputModule
);

FlowTable(r1,r2,r3) =
(r1 = 1) -> (rule1?,4).(action1!,99).FlowTable(r1,r2,r3)
+ (r2 = 1) -> (rule2?,3).(action2!,99).FlowTable(r1,r2,r3)
+ (r3 = 1) -> (rule3?,6).(action3!,99).FlowTable(r1,r2,r3)
+ (activatingRule1?,99).FlowTable(1,r2,r3)
+ (activatingRule2?,99).FlowTable(r1,1,r3)
+ (activatingRule3?,99).FlowTable(r1,r2,1);
+ (rule0?packet,0).('requestRuleForPacket?packet,99).
                    FlowTable(r1,r2,r3);

OutputModule = (action1?,999).(outPort1!4, 1).OutputModule
+ (action2?,999).(outPort2!5, 1).OutputModule
+ (action3?,999).(outPort2!6, 1).OutputModule;
```

We describe a portion of formal specification of the informal SDN specification using ACSR. ACSR can express processes running concurrently and communicating the switches and controller. Forwarding packets can be modeled as prioritized synchronization of events in ACSR. But the disadvantages of ACSR is that it is hard to categorize classification of data packets.

4.2. Formal Specification of SDN using Z

This clause describes an example of the Z specification for SDN that is comprised of lots of switches and a controller managing them. In

this specification, we focus on each one switch and controller for emphasis on their functionality of them. For an example, switch keeps a flowtable for handling input packets. This table has a fulfillment action for some packets, and can be modified by the controller

```
Table_Type == HEADER_FIELD x N x F ACTION_TYPE
```

The table is made up of a packet header, applied counter and actions. The counter plays a role of priority, so when one packet matches more than two elements, actions having a higher counter value will be applied.

```
PORT ::= port1 | port2 | port 3 | port 4
PORT_STATUS ::= active | inactive
```

Switch also has ports for input or output and each port has a state (active or inactive). In this specification, we assume that the switch has four ports.

```
ACTION ::= Forward | Enqueue | Drop | Modify_Field
OPTIONAL_ACTION ::= ALL | CONTROLLER | LOCAL | TABLE | IN_PORT | NONE
ACTION_TYPE: ACTION <-> OPTIONAL_ACTION
ACTION_TYPE Forward = {ALL, CONTROLLER, LOCAL, TABLE, IN_PORT}
ACTION_TYPE Enqueue = NONE
ACTION_TYPE Drop = NONE
ACTION_TYPE Modify_Field = NONE
```

Actions stored in table are Forward, Enqueue, Drop and Modify Field, and forward action has four optional actions, ALL (meaning broadcast), LOCAL (multicast), IN PORT (unicast) and Controller (to controller).

```
forwardToController
headerNoMatchAction
packet: PACKET_TYPE
packet.header = packet?.header
packet.contents = packet?.contents
packet.action = {(Forward, CONTROLLER)}
controller.packet = {packet}
```

Z specification for SDN is focused on each switch and controller for emphasis on their functionality and it is possible of limited verification for SDN using Z specification. It can specify forwarding packets in limited hosts and switches, but it is difficult to specify various states of large networks in the real-world.

4.3 Subtle Ambiguities Discovery and Correctness Verification

We could find the overlooked subtleties in SDN networks during transforming from the topology and properties to ACSR and Z specification. Once a formal specification is made, it can be used for validating the network topology.

For example, in order to prove that the correctness of the ACSR and Z specifications, we can show that the specification has no deadlock. If the specification has no deadlock, we can claim that the network topology runs forever without any stop, which means the packet flow is well modeled.

5. Security Considerations

TBD

6. Acknowledgements

The authors would like to thank theory and formal methods lab members in Korea University for their process algebraic specification support.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

7.2. Informative References

[I-D.nadeau-sdn-problem-statement] Nadeau, T., and P. Pan, "Software Defined Network (SDN) Problem Statement", draft-nadeau-sdn-problem-statement-00 (work in progress), September 2011.

[b-OpenFlow] OpenFlow Switch Specification 1.3,
<https://www.opennetworking.org/>.

[b-Kang12] M. Kang et al., Formal Specification for Software-Defined Networks (SDN), CFI'12 (submitted), 2012.

[b-Nam12] K-H. Nam, et al., Draft Document of Y.FNsdn-fm "Requirements of formal specification and verification methods for software-defined networking, ITU-T (work in

progress), 2012.

[b-NetCore12] C. Monsanto, N. Foster, R. Harrison, and D. Walker, A Compiler and Runtime System for Network Programming Languages, POPL Jan.2012. To appear.

[b-NDlog11] A. Wang, L. Jia, C. Liu, B. Loo, O. Sokolsky, and P. Basu, Formally Verifiable Networking, 2011.

[b-ACSR95] J. Choi, I. Lee, and H. Xie, The Specificatoin and Schedulability Analysis of Real-Time Systems Using ACSR, 16th IEEE Real-Time Systems Symp.(RTSS'95), Dec. 1995.

Authors' Addresses

Myung-Ki Shin
ETRI
161 Gajeong-dong Yuseng-gu
Daejeon, 305-700
Korea

Phone: +82 42 860 4847
Email: mkshin@etri.re.kr

Ki-Hyuk Nam
ETRI
161 Gajeong-dong Yuseng-gu
Daejeon, 305-700
Korea

Phone: +82 42 860 5729
Email: nam@etri.re.kr

Miyoung Kang
Korea University
Anam-dong, Seongbuk-gu
Seoul, 136-713
Korea

Phone: +82 2 3290 3200
Email: mykang@formal.korea.ac.kr

Jin-Young Choi
Korea University
Anam-dong, Seongbuk-gu
Seoul, 136-713
Korea

Phone: +82 2 3290 3200
Email: choi@formal.korea.ac.kr

Internet Research Task Force
Internet-Draft
Intended status: Informational
Expires: December 29, 2012

H. Yin
H. Xie
T. Tsou
Huawei Technologies
D. Lopez
P. Aranda
Telefonica I+D
R. Sidi
ConteXtream
June 27, 2012

SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS)
across Multiple Domains
draft-yin-sdn-sdni-00.txt

Abstract

This draft proposes a protocol SDNi for the interface between Software Defined Networking (SDN) domains to exchange information between the domain SDN Controllers. It defines the concept of a SDN domain; its need, what are its components and how SDNi helps in inter domain communication.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 29, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	4
2. Motivation Behind SDN	5
3. SDN Architecture	6
4. SDN Functionality	8
5. SDN Domains	9
6. SDNi	10
6.1. SDNi Message	10
6.2. SDNi Protocol	11
6.3. Practical Considerations	11
7. Conclusions	11
8. Acknowledgements	12
9. Security Considerations	12
10. IANA Considerations	12
11. Informative References	12
Authors' Addresses	12

1. Introduction

Software Defined/Driven Networking (SDN) is generally defined as an emerging network architecture where network control is decoupled from forwarding and is directly programmable. This migration of control, formerly tightly bound in individual network devices, into accessible computing devices enables the underlying infrastructure to be abstracted for applications and network services, which can treat the network as a logical or virtual entity.

SDN focuses on four key features:

- o Separation of the control plane from the data plane.
- o A centralized controller and view of the network.
- o Open interfaces between the devices in the control plane (controllers) and those in the data plane.
- o Programmability of the network by external applications.

The centralized view and the separation of the control plane and the data plane means that the SDN controller can create a physical topology of how nodes are connected and, based on some combination of algorithms, create paths through the network. Finally, the paths are programmed into the devices' forwarding engines. That allows the SDN controller to better manage traffic flows across the entire network and react to changes quicker and more intelligently. How well the controller defines those paths is, of course, critical to the operation of an SDN.

The controller is akin to a computer's operating system (a network-control operating system, if you will) on which applications are written. It communicates with the underlying network hardware (switches and routers) through a protocol such as OpenFlow. Different types of SDN controller software, then, are analogous, in certain respects, to Windows, Linux, and Mac OS X. What's more, just like those computer-based operating systems, today's controller software is not interoperable.

As much as OpenFlow and SDN have been conflated and intertwined in the minds of many, we need to understand that SDN controllers are at higher layer of network abstraction, providing a platform for network applications and a foundation for networking programmability. OpenFlow is just an underlying protocol that facilitates interaction between the controller and data-forwarding tables on switches.

Due to the great potentials of SDN and the unique requirements of

Data Center Networks (DCNs), Data Centers are likely to become a first place where SDN could be deployed. We envision that SDN could be gradually adopted by enterprise networks and then by carrier networks due to its unique, attractive features. When deploying SDN in large- scale distributed networks, we expect to see a collection of deployments limited to portions of a bigger network (referred to as SDN domains). In other words, the operator of a large-scale enterprise / carrier network should divide the whole networks into multiple connected SDN domains, where each of such domains corresponds to a relatively small portion of the whole network. Such a divide-and- conquer methodology not only allows gradual deployment and continuous evolution, but also enables flexible provisioning of the network.

With the deployment of multiple SDN domains comes the issue of exchanging information between these domains. This draft defines the concept of an SDN domain and proposes a protocol SDNi as a protocol for interface between inter SDN domains.

1.1. Terminology

While the definition of software define/driven networks is still "nebulous" to some extent, we refer to as SDN the networks that implement the separation of the control and data/forwarding planes and software defined/driven interactions between these two separated planes. SDN provides capability of network openness in that applications can program networking devices via APIs. The software defined/driven interactions could be similar to OpenFlow or the like. However, how the two separated planes interact is not a focus of this draft.

Networking devices: networking devices are hardware devices or software elements capable of forwarding data packets and communicating with Network Operating System (NOS).

Network Operating System (NOS): NOS is software responsible for monitoring and controlling a network system. It has whole knowledge of the underlying network topology and resources. NOS provides a programmatic platform based on the abstraction of the underlying network devices to applications by means of appropriate virtual elements, controlled by APIs that provide access to network functions and common functionalities. It facilitates network access and control to applications by translating their requests to actions on networking devices. The traditional device control plane can be built within NOS or above NOS.

SDN controller: SDN controller represents a SDN open platform, and responsible for managing the platform. With or without NOS, SDN

controller implements SDN's functionality. A general SDN controller is a software entity that opens SDN APIs to applications, and translates application's request to actions on network devices using proprietary protocol or OpenFlow -like protocol with or without NOS's supports. The SDN controller can reside in an NOS or run separately on a server in network.

SDN controller-based applications: Applications interacting with underlying networks via SDN controller. They can run inside the controller or they can be (virtual) appliances distributed throughout the network and serve some (specific) high-function and have the controller configure the network devices to steer traffic through those elements selectively according to SDN policy.

Some controllers leverage OpenFlow exclusively, and others are (or will be) capable of accommodating other protocols and mechanisms to achieve the same practical result. A normal OpenFlow controller can be integrated within NOS or run within SDN controller depending on implementation.

SDN-capable devices: networking devices are capable of communicating with NOS/SDN controller using proprietary protocol or standard protocol, like OpenFlow. The devices which do not support the proprietary protocol or OpenFlow are normal devices.

An SDN domain is a portion of a network infrastructure, consisting of numerous connected networking devices that are SDN-capable and NOS that supports proprietary protocol or OpenFlow. A SDN domain controller (SDN controller) can cover multiple NOSs, means it can communicating with multiple NOSs via some proprietary protocol or NOS APIs, or it can communicates with OpenFlow-enabled devices directly if OpenFlow protocol is supported in the controller. An SDN domain can be as small as a sub-network of a dozen devices or as large as a medium/large data center network. An network can consist of one or many SDN domains. Hence an inter-SDN domain protocol is needed.

2. Motivation Behind SDN

There has been a great deal of innovation in networking software but not that much in networking. Networks are still stuck in the past with routing algorithms that change very slowly or with primitive network management. Computation and storage have been virtualized, creating a more flexible and manageable infrastructure, but networks are still hard to manage. Networking is built with limited view of application needs. Networks used to be simple: basic Ethernet/IP straightforward, easy to manage but new application requirements have led to control plane complexities. Each control requirement leads to

new mechanisms and so networks continue to grow more complex.

By simplifying network's structures and allowing applications to take part in the control of networks, SDN provides innovative principles of networking system construction. With the trends of the network development that moving from networks to more and more services and applications, applications are taking more roles in network routing decisions and affecting the traditional network functionalities and behaviors. As an interaction and control point between applications/services and underlying networks, SDN controller is a key point in SDN networks and clearly a SDNi protocol among controllers is clearly a necessity in our opinion.

3. SDN Architecture

As software-defined networking gains traction, vendors and enterprises will generally adopt a three-tiered architecture as seen in Figure 1.

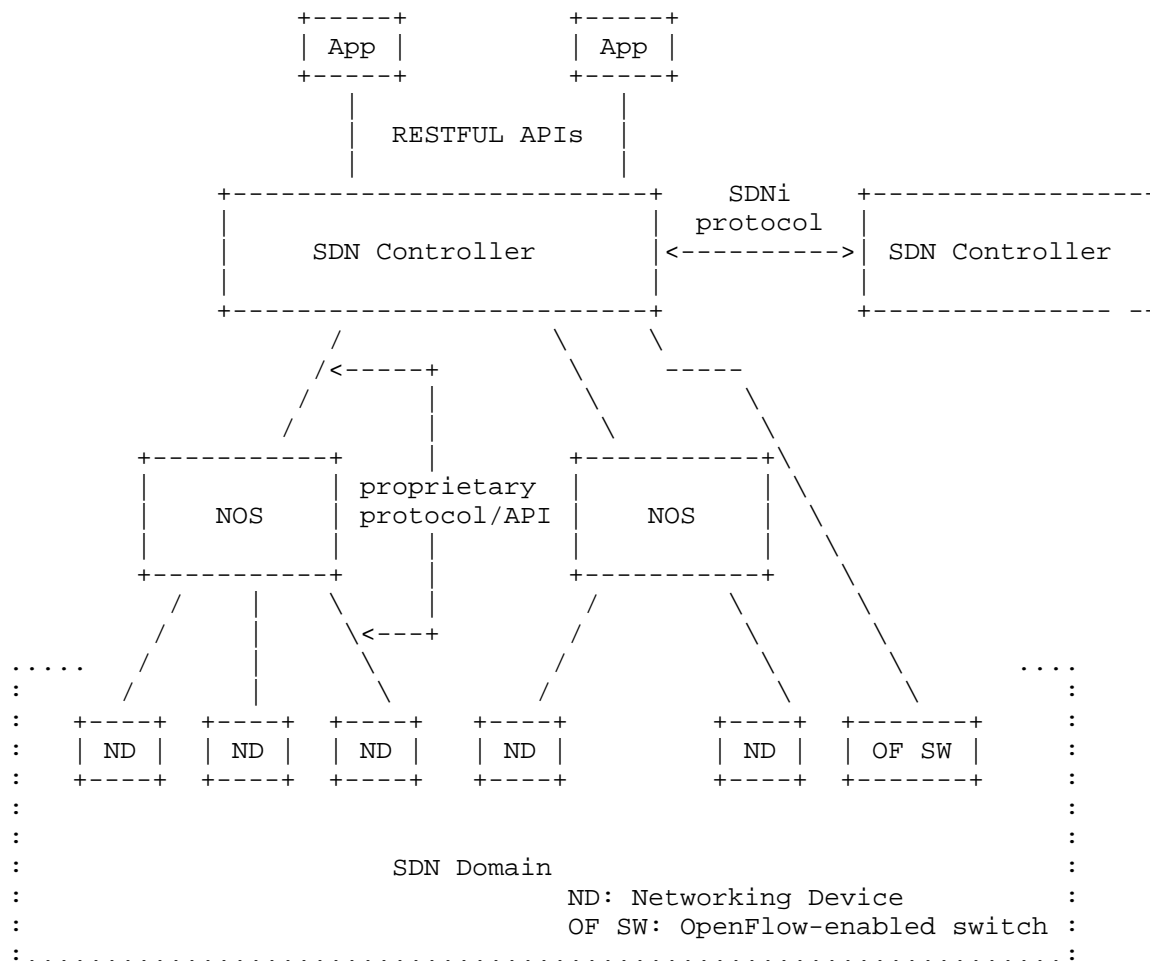


Figure 1: An Architecture For SDN

The architecture's first tier will involve the physical network equipment, including Ethernet switches and routers. This forms the dataplane. The middle tier consists of the controllers that facilitate setting up and tearing down flows and paths in the network leveraging information about capacity and demand obtained from the networking gear through which the traffic flows. This forms the control plane, NOS and SDN controller/platform over it. The top tier will involve applications to direct security, management and other specific functions through the controller. However, instead of relying on proprietary software running on each switch to control its forwarding behavior, switches in a SDN architecture are controlled by

a Network OS (NOS) that collects information and manipulates their low-level forwarding plane, providing an abstract model of the network topology to the SDN controller hosting the applications. Applications can adapt the network behavior to suit specialized requirements, for example, providing network virtualization services that allow multiple logical networks to share a single physical network - similar to the way in which a hypervisor allows multiple virtual machines to share a single physical machine.

These controller-based applications will serve the same roles that physical appliances play in the network today. For example, network architects who are building software-defined networks could deploy applications like a virtual load balancer, a virtual intrusion detection system (IDS), or a virtual firewall on a controller. The application could tap into information the controller possesses about traffic patterns, application data and capacity. Cloud computing applications could be a big beneficiary of software-defined networking and OpenFlow because these technologies make provisioning in a multi-vendor virtual environment much simpler. For instance a controller-based load balancing application could automate the movement of workloads among virtual machines by using the controller's view of the capacity of individual network devices.

SDN controller is a software module in a SDN domain; logically it represents the highest control point in a SDN domain network, and physically it can reside in NOS or run independently at a separate server. SDN controller provides north-bound APIs to user applications, and interacts with NOS to collect network information for applications.

4. SDN Functionality

This section provides a detailed explanation of the role of SDN controller in SDN.

SDN is constructed through the separation of control plane and forwarding plane of networking devices. The decoupling is achieved by turning the network control problem into a state management problem, and only exposing the state to be managed to the application, and not the mechanisms by which it arrived. Applications operate over control traffic, flow table state, configuration state, port state, counters, etc. However, how this data is pulled into the controller, and how any changes are pushed back down to the switch is not the application's concern. It can just use the controller API to modify network state which in turn will be translated to necessary commands on the relevant devices to have the new state correctly applied on the network.

SDN controller should provide following functionality:

- o Provide northbound APIs for applications so they can program underlying network properties such as bandwidth assignments, network isolation, routing properties, redundancy properties, or security requirements.
- o Provide covered SDN domain network topology view to applications so that applications can define data flow and flow path within the view.
- o Respond and adapt to the network conditions and provide event-handling mechanisms for applications so that they can receive feedback and adapt accordingly.
- o Have covered SDN domain network services and resources views so that the controller can coordinate application's requests.
- o Work with NOS/control plane to collect underlying device information and translate application's request to actions on network devices and feed into NOS/control plane.
- o Act as the controller of physical devices, such as the OpenFlow control mechanism if such mechanism is not implemented in NOS[1].

5. SDN Domains

An SDN domain is a portion of the network determined by the network operator. It has a SDN controller to control the domain, it can cover multiple NOS or communicate with some SDN-capable devices directly. Each NOS typically consists of numerous inter-connected SDN-capable devices. An example of such NOS is illustrated in [ONIX]. The SDN controller aggregates network topology views from multiple NOS and maintain a global network view of networks covered by the domain. The controller is responsible for dispatching and disseminating application's request to corresponding NOS. Two SDN domains are adjacent if there is a physical link between two underlying networks.

Inside each SDN domain, its controller may define domain-specific policies on information importing from devices, aggregating, and exporting to external entities. Such policies may not be made public; therefore, other domain controllers may not know the existence of such policies for any given SDN domain.

When receiving an application request for certain network resources, the SDN controller should decide if the request can be satisfied. If

it can be satisfied, the controller should instruct the devices in its domain by updating them with a set of data/forwarding rules so that the devices could implement such rules and fulfill the request.

6. SDNi

SDNi is a protocol for interfacing SDN domains. It is responsible for coordinating behaviors between SDN controllers and exchange control and application information across multiple SDN domains. SDNi is an "east-west" protocol between SDN controllers, as an analogy to OpenFlow being a "north-south" protocol between NOS and Network devices. As such, SDNi should be a protocol implemented by the NOS (the same as OpenFlow or any other control protocols as mentioned above) and how the applications/SDN controllers that run in the NOS use this protocol (i.e. what is the API to use it) is out of scope of this document.

SDNi protocol should be able to

- o Coordinate flow setup originated by applications, containing information such as path requirement, QoS, SLA etc. across multiple SDN domains.
- o Exchange reachability information to facilitate inter-SDN routing. This will allow a single flow to traverse multiple SDNs and have each controller select the most appropriate path when multiple such paths are available.

SDNi depends on the types of available resources and capabilities available and managed by the different controllers in each domain. Hence it is important to implement SDNi in a descriptive and open manner so that new capabilities offered by different types of controllers will be supported.

Since SDN in essence allows for innovation, it is important that data exchanged between controllers will be dynamic in nature, i.e. there should be some meta-data exchange that will allow SDNi to exchange information about unknown capabilities.

6.1. SDNi Message

The types of messages exchanged via SDNi can be:

- o Reachability update
- o Flow setup/tear-down/update request (including application capability requirement such as QoS, BW, latency etc.)

- o Capability update (including network related capabilities such as BW, QoS etc. and system and software capabilities available inside the domain).

6.2. SDNi Protocol

SDNi is used to exchange information between SDN domains that are under the control of a single network operator or collaborating operators. One way to implement SDNi is to use extension of BGP and SIP over SCTP protocols to exchange information.

6.3. Practical Considerations

SDN domains are built by network operator for flexible administration purpose. It depends on the scale of underlying network that the operator decides how to divide whole network into SDN domains. For some small-scale data centers, only one SDN domain may be enough. For a Service Provider with large carrier network, it is most likely better to divide the network into SDN domains as centralizing the control onto a single NOS/controller will create a bottleneck. For example, Operator can divide its network into different SDN domains based on physical locations. It can lease such part of its network to local content provider, etc. Such deployment scenario requires an SDN controller providing powerful network service capability to applications.

Also defining domains and interconnections between them involves more than simple connections between SDN boxes; it requires to consider various aspects such as to work out how their network topologies connect together, which neighbor controller boxes to talk to and how to get their addresses, what rights and policies control the conversation etc.

Other aspects to be considered are who is allowed to deploy 'programs' on the SDN infrastructure what actions can a 'program' execute depending on who deployed them: the SDN network manager is likely in control to deploy 'programs' on the SDN infrastructure. The focus then is on how the deployed programs might affect other domains and what mechanism we want to use to communicate this effect to the other domains.

7. Conclusions

Add any conclusions

8. Acknowledgements

The authors would like to thank Aditi Vira for editing the draft.

9. Security Considerations

- o What actions are 'applications' allowed to execute?
- o How is the security policy propagated to other SDN domains.
- o An application should not be able to do a DoS attack (either willingly or unwillingly)

10. IANA Considerations

This document makes no specific request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

11. Informative References

- [1] Xie, H., Yin, H., Tsou, T., and V. Hilt,
"draft-xie-alto-sdn-use-cases-00.txt", June 2012.

Authors' Addresses

Hongtao Yin
Huawei (USA)
2330 Central Expy
Santa Clara, CA 95050
USA

Phone:
Email: Hongtao.yin@huawei.com

Haiyong Xie
Huawei & USTC
2330 Central Expy
Santa Clara, CA 95050
USA

Phone:
Email: Haiyong.xie@huawei.com

Tina Tsou
Huawei Technologies (USA)
2330 Central Expressway
Santa Clara, CA 95050
USA

Phone:
Email: Tina.Tsou.Zouting@huawei.com

Diego R. Lopez
Telefonica I+D
Don Ramon de la Cruz, 84
Madrid, 28006
Spain

Phone:
Email: diego@tid.es

Pedro Andres Aranda
Telefonica I+D
Don Ramon de la Cruz, 84
Madrid, 28006
Spain

Phone:
Email: pedro.aranda@tid.es

Ron Sidi
ConteXtream
3600 West Bayshore Rd.
Palo Alto, CA 94303
USA

Phone:
Email: ron@contextream.com

