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 languages.

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].
Table of Contents

1. Introduction ....................................................3
2. SDN Discussion and Assumption .................................3
   2.1 Requirements for SDN Programming ..........................4
3. Formally Verifiable Networking Framework for SDN .............5
4. Formal Specification and Verification for SDN ...............6
5. Examples ........................................................7
   5.1 Formal Specification of SDN using ACSR ...................7
   5.2 Formal Specification of SDN using Z ......................10
   5.3 Subtle Ambiguities Discovery and Correctness Verification ..12
6. Security Considerations ........................................12
7. Acknowledgements .............................................12
8. References .....................................................12
   8.1 Normative References .....................................12
   8.2 Informative References ...................................13
Authors’ Addresses ................................................14
1. Introduction

Software-defined networking (SDN) is emerging and intensively discussed as one of the most promising technologies to make networks programmable and flexible and 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 so that they can directly control and manage their networks and resources to best serve their customers’ needs. 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. SDN Discussion and Assumption

SDN architectural issues are not fully investigated yet, because it is very difficult to build a single SDN architecture to accommodate and harmonize various requirements from network operators, application/service providers and vendors’ perspectives. Therefore, it is assumed that SDN has three-tier architecture as illustrated in Figure 1.
2.1 Requirements for SDN Programming

This document presents initial thoughts on requirements of SDN application programming.

R1: Guarantee that the design and implementation of SDN devices conforms to the standards, correctness and safety properties.

R2: Check consistency and safety of their network configurations and virtual and physical topologies against any properties to be satisfied with such as:
- No loops and/or blackholes in the network
- Logically different networks cannot interfere with each other (e.g., traffic isolation)
- New or update configurations conforms to properties of the network and do not break consistency of existing networks (e.g., network updates)

R3: Support formal semantics in high-level languages, APIs and underlying protocols for SDN
- Properties that need to be satisfied with by the SDN should be described in notations with formal semantics

R4: Support conceptual models to reason about networks defined, configured, implemented by software and hardware for SDN more precisely.
- Timing models that capture essential properties and
3. 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 2 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 2, 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
4. Formal Specification and Verification 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.
In this document, we propose to add SDN language compiler and tools such as verifier on top of SDN control software. Figure 3 illustrates formal specification and verification tools between SDN control software and applications.

We consider developing new formal specification language with simple and minimum semantics to support new properties of SDN networks. At this moment, we use process algebra ACSR (Algebra of Communicating Shard Resources) \cite{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 SDN through the analysis of the requirement for OpenFlow.
5. Examples

5.1 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 processes; 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 actions.

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 4. The topology consists of a single switch and three hosts (H1, H2, and H3).

---\| Switch |---\
|    \|        |    \v    \\
|   ----  ----  ----
|   H1  |   H2  |   H3 |

Figure 4 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:

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
<th>Types of packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
<td>Switch</td>
<td>packet1, packet2, packet3</td>
</tr>
<tr>
<td>Switch</td>
<td>H2</td>
<td>packet4</td>
</tr>
<tr>
<td>Switch</td>
<td>H3</td>
<td>packet5, packet6</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Priority</th>
<th>Packets matched</th>
<th>Action on matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule1</td>
<td>4</td>
<td>packet1</td>
<td>action1</td>
</tr>
<tr>
<td>rule2</td>
<td>3</td>
<td>packet1</td>
<td>action2</td>
</tr>
<tr>
<td>rule2</td>
<td>3</td>
<td>packet2</td>
<td>action2</td>
</tr>
<tr>
<td>rule3</td>
<td>6</td>
<td>packet2</td>
<td>action3</td>
</tr>
<tr>
<td>rule3</td>
<td>6</td>
<td>packet3</td>
<td>action3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action1</td>
<td>output packet4 to H2 through outPort1</td>
</tr>
<tr>
<td>action2</td>
<td>output packet5 to H3 through outPort2</td>
</tr>
<tr>
<td>action3</td>
<td>output packet6 to H3 through outPort2</td>
</tr>
</tbody>
</table>

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:

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

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:

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

\[ \text{InputModule} = (\text{inPort?packet}, 1). (\text{packet} = 1) \rightarrow ((\text{rule1!},1).\text{InputModule} + (\text{rule2!},1).\text{InputModule}) + (\text{packet} = 2) \rightarrow ((\text{rule2!},1).\text{InputModule} + (\text{rule3!},1).\text{InputModule}) + (\text{packet} = 3) \rightarrow (\text{rule3!},1).\text{InputModule} + (\text{rule0!packet},0).\text{InputModule}; \]

\[ \text{FlowTable}(r1,r2,r3) = \]
\[ (r1 = 1) \rightarrow (\text{rule1?},4).(\text{action1!},99).\text{FlowTable}(r1,r2,r3) + (r2 = 1) \rightarrow (\text{rule2?},3).(\text{action2!},99).\text{FlowTable}(r1,r2,r3) + (r3 = 1) \rightarrow (\text{rule3?},6).(\text{action3!},99).\text{FlowTable}(r1,r2,r3) + (\text{activatingRule1?},99).\text{FlowTable}(1,r2,r3) + (\text{activatingRule2?},99).\text{FlowTable}(r1,1,r3) + (\text{activatingRule3?},99).\text{FlowTable}(r1,r2,1); + (\text{rule0?packet},0).('\text{requestRuleForPacket?packet},99).\text{FlowTable}(r1,r2,r3); \]

\[ \text{OutputModule} = (\text{action1?},999).(\text{outPort1!4}, 1).\text{OutputModule} + (\text{action2?},999).(outPort2!5, 1).\text{OutputModule} + (\text{action3?},999).(outPort2!6, 1).\text{OutputModule}; \]

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.

5.2. 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
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.

\[
\text{Table\_Type} \equiv \text{HEADER\_FIELD} \times N \times F \times \text{ACTION\_TYPE}
\]

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.

\[
\text{PORT} ::= \text{port1} \mid \text{port2} \mid \text{port}3 \mid \text{port}4
\]

\[
\text{PORT\_STATUS} ::= \text{active} \mid \text{inactive}
\]

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.

\[
\text{ACTION} ::= \text{Forward} \mid \text{Enqueue} \mid \text{Drop} \mid \text{Modify\_Field}
\]

\[
\text{OPTIONAL\_ACTION} ::= \text{ALL} \mid \text{CONTROLLER} \mid \text{LOCAL} \mid \text{TABLE} \mid \text{IN\_PORT} \mid \text{NONE}
\]

\[
\text{ACTION\_TYPE} : \text{ACTION} \leftrightarrow \text{OPTIONAL\_ACTION}
\]

\[
\text{ACTION\_TYPE Forward} = \{\text{ALL, CONTROLLER, LOCAL, TABLE, IN\_PORT}\}
\]

\[
\text{ACTION\_TYPE Enqueue} = \text{NONE}
\]

\[
\text{ACTION\_TYPE Drop} = \text{NONE}
\]

\[
\text{ACTION\_TYPE Modify\_Field} = \text{NONE}
\]

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).

\[
\text{forwardToController}
\]

\[
\text{headerNoMatchAction}
\]

\[
\text{packet: PACKET\_TYPE}
\]

\[
\text{packet.header }= \text{packet?.header}
\]

\[
\text{packet.contents}=\text{packet?.contents}
\]

\[
\text{packet.action}=\{(\text{Forward, CONTROLLER})\}
\]

\[
\text{controller.packet}=\{\text{packet}\}
\]

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.
5.3 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.

6. Security Considerations

TBD

7. Acknowledgements

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

8. References

8.1. Normative References


8.2. Informative References


“Requirements of formal specification and verification methods for software-defined networking, ITU-T (work in progress), 2012.


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