Formal Specification and Programming for SDN

relevant ID:
draft-shin-sdn-formal-specification-01

Myung-Ki Shin, Ki-Hyuk Nam
ETRI

Miyoung Kang, Jin-Young Choi
Korea Univ.

Proposed SDN RG Meeting@IETF 84 - Vancouver, BC, Canada

What's *Formal Specification* ?

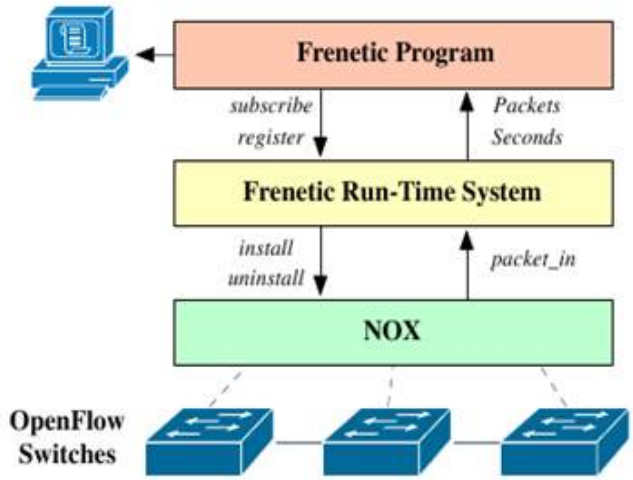
- Some definitions from academia
  - A formal specification is a specification expressed in a language whose semantics are formally defined, as well as vocabulary and syntax.
  - The need for a formal semantic definition means that the specification language must be based on logic, mathematics, etc., not natural languages.
- Formal verification
  - The act of proving or disproving the correctness of designs or implementations with respect to requirements and properties with which they must satisfy, using the formal methods or techniques

Why it is Necessary in SDN ?

- SDN network operators and application/service providers can introduce a new capability by writing a simple software program.
  - Incomplete or malicious programmable entity could cause break-down of underlying networks shared by heterogeneous devices and stake-holders.
  - Any misunderstanding or diverse interpretations should be avoided.
- Formal specification can be applied to verification methods such as theorem proving, model checking, static analysis, etc.

SDN Programming – Relevant Works (1/3)

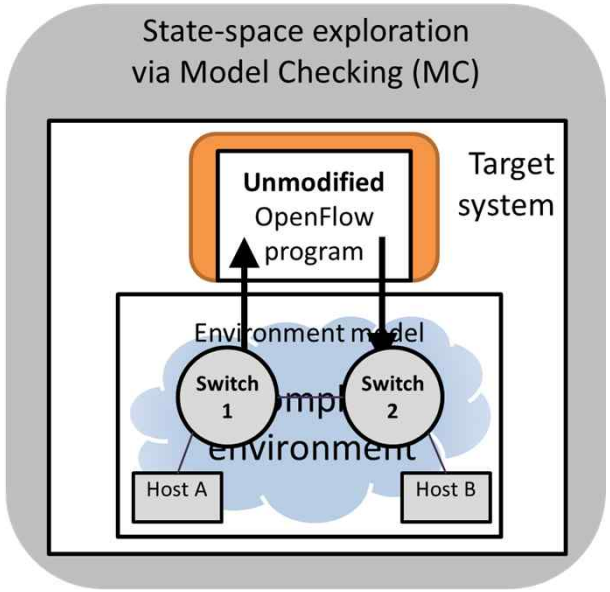
- Frenetic and NetCore
  - A high-level programming language that can be used to write OpenFlow applications running on top of NOX.
  - Neither NOX or Frenetic perform correctness checking of updates, limiting their ability to help in detecting bugs in the application code or other issues that may occur while the network is in operation.



Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker, "Frenetic: A network programming language," in Proc. ACM International Conference on Functional Programming, September 2011

SDN Programming – Relevant Works (2/3)

- NICE (No bugs In Controller Execution)
  - NICE performs symbolic execution of OpenFlow applications and applies model checking to explore the state space of an entire OpenFlow network.
  - NICE is a proactive approach that tries to figure out invalid system states by using a simplified OpenFlow switch model.
  - It is not designed to check network properties in real time



Marco Canini, Daniele Venzano, Peter Peresini, Dejan Kostic, and Jennifer Rexford, "A NICE way to test OpenFlow applications," in Proc. Networked Systems Design and Implementation, April 2012

SDN Programming – Relevant Works (3/3)

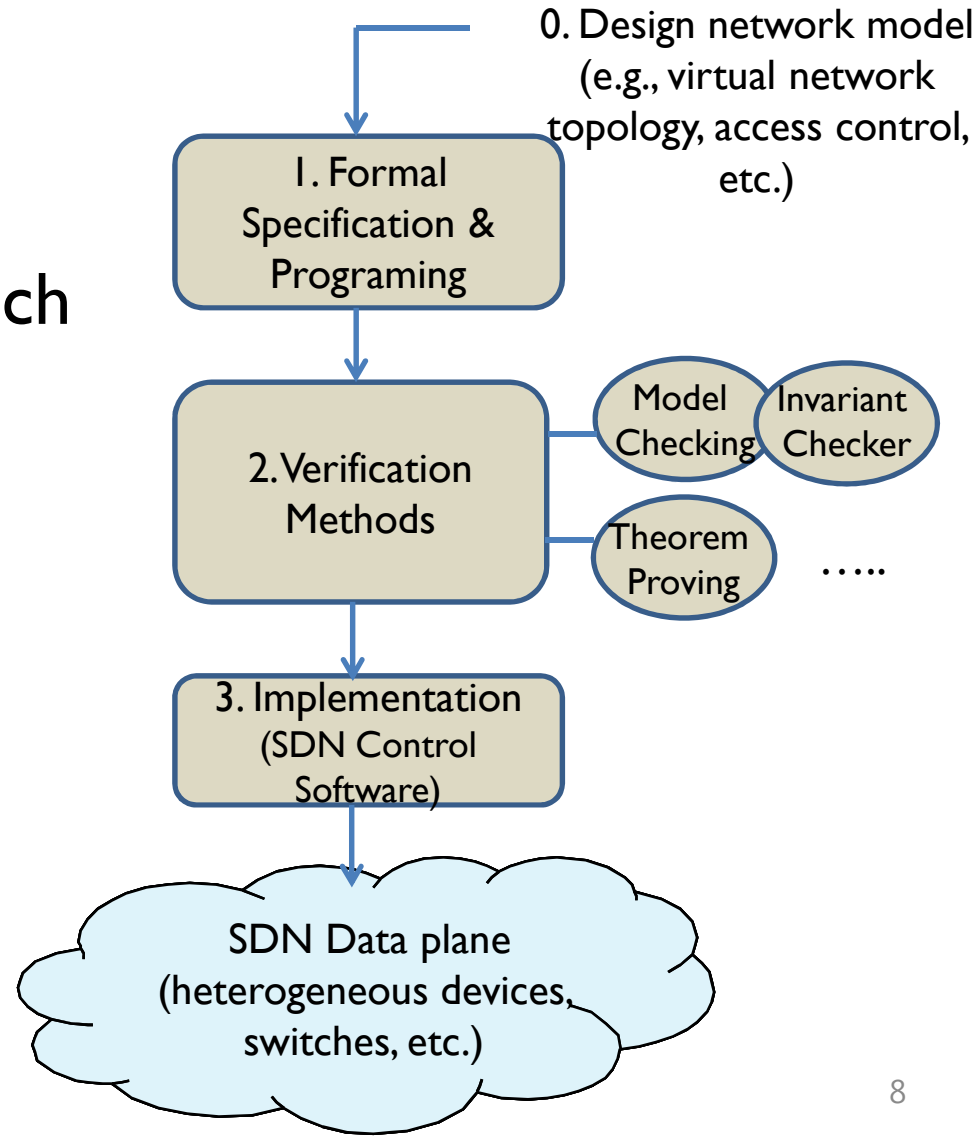
- Nettle
  - Functional reactive programming for OpenFlow networks using HASKELL language
  - http://haskell.cs.yale.edu/
- ONRC
  - http://onrc.stanford.edu/
- HotSDN Workshop (SIGCOMM2012)
  - Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schesinger, David Walker, "Abstractions for Network Update," HotSDN, 2012.
  - …

Formal Specification Languages

- SDL (Z.100)
  - Standard specification language suitable for real-time and reactive systems, from requirements to implementation
  - Too big for SDN?
- Z Language
  - Z could be focused on each switch and controller for emphasis on their functionality
  - It is difficult to specify various states of large networks
- ACSR (Algebra of Communicating Shard Resources)
  - ACSR can express processes running concurrently and communicating the switches and controller
  - Forwarding packets can be modeled as prioritized synchronization of events in ACSR
  - It is hard to categorize classification of data packets

Our Approach - Common Framework

- We discuss the formally verifiable networking framework for SDN, which consists of the three components
  - Formal specification and programming,
  - Verification methods, and
  - Implementation (SDN control software and applications)



We Assume that SDN has

- *Three-Tier* Architecture, including
  - *Tier-1* : Forwarding entities and any software/hardware components comprising of them
  - *Tier-2* : Control and management entities for the *Tier-1*
  - *Tier-3* : Applications and services that take advantage of the infrastructures based on *Tier-1* and *Tier-2*.

Initial thoughts on Requirements of SDN Programming (1/2)

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

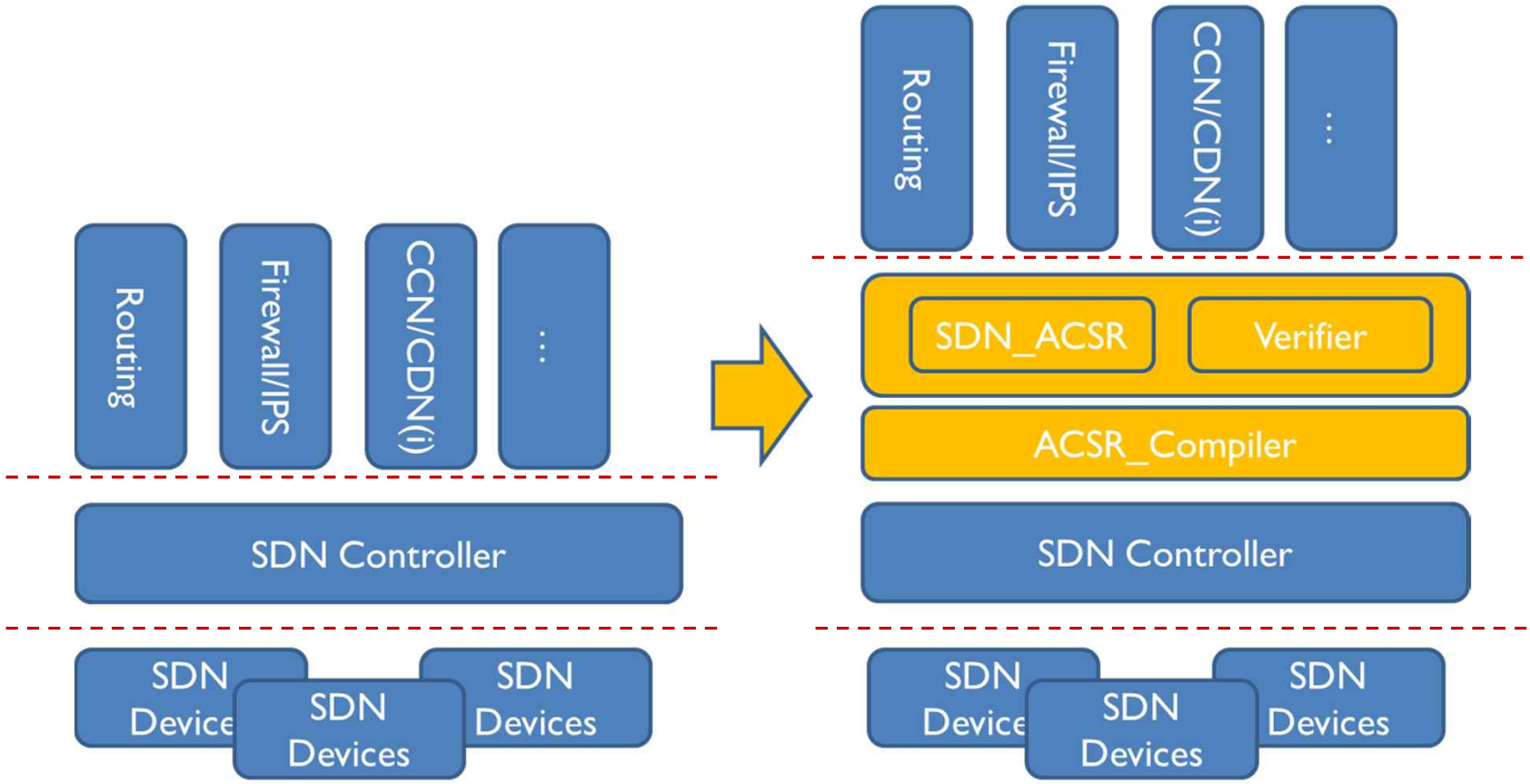
Initial thoughts on Requirements of SDN Programming (2/2)

- 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
- 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 behaviors of packet flows and data traffic in
  - Formalisms that reflect networks and systems behaviors.
  - Diverse languages and tools based on the conceptual model

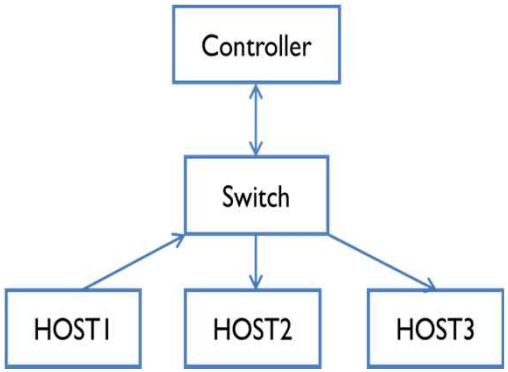
Case Study : SDN Modeling using ACSR

- SDN modeling using ACSR specification
  - Example-1 : OpenFlow 1.0 spec. verification
  - Example-2 : Invariant Property Checker of SDN topology (access control example)

Adding SDN-ACSR/Tools between SDN Controller and Apps



Example-1: ACSR Specification of OF1.0 based Example Topology



An example topology

Sender	Receiver	Types of packets
Host1	Switch	packet1, packet2, packet3
Switch	Host2	packet1', packet3'
Switch	Host3	packet1'', packet2'

```
System = (Host1 || Host2 || Host3 || Switch || Controller)
        \ {inPort, outPort1, outPort2, activatingRule1, activatingRule3,
        activatingRule3};
```

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

```
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
    );
```

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

```
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);
```

……..

Subtle Ambiguities in OF1.0 Spec.

① An entry that specifies an exact match(i.e., it has no wildcards) is always the highest priority. All wildcard entries have a priority ones. If multiple entries have the same priority, the switch is free to choose any ordering. [OF1.0]

- Same packets may have different rules ?
- Resolved in OF 1.1+ as setting "CHECK_OVERLAP" bit

② For all packet that do not have a matching flow entry, a packet-in event may be sent to the controller [OF1.0] (send OFPT_FLOW_MOD to a switch)

But, no specification regarding delays between controllers and multi-switches

- Not resolved yet ?

Example-2: ACSR Specification of Access Control Property

① No loops

- $M_{noloop} = \{\}:M_{noloop} + (packetin?,1).P(0)$
- $P(t) = (t < TLIMIT) \rightarrow (drop?,1).M_{noloop} +$
  $(t < TLIMIT) \rightarrow (world?,1).M_{noloop} +$
  $(t < TLIMIT) \rightarrow \{\}:P(t+1)$

② Blocklist (the packets cannot traverse)

- $M_{blocklist} = \{\}:M_{blocklist} + (packet1in?,1).R(0)$
- $R(t) = (t < TLIMIT) \rightarrow (s24?,1).NIL +$
  $(t < TLIMIT) \rightarrow \{\}:R(t+1) +$
  $(t = TLIMIT) \rightarrow M_{blocklist}$

③ Route (the packets reach a switch)

- $M_{route} = \{\}:M_{route} + (packet1in?,1).R(0)$
- $R(t) = (t < TLIMIT) \rightarrow (s4?,1).M_{route} +$
  $(t < TLIMIT) \rightarrow \{\}:R(t+1)$

Checking Property Invariance

- SDN_ACSR verifier and tools could check the invariant properties related to access controls

$$(\mathrm{Sys} \,||\, \mathrm{M}_{\mathrm{noloop}} \,||\, \mathrm{M}_{\mathrm{blocklist}} \,||\, \mathrm{M}_{\mathrm{route}}) \approx \{\}^{\infty}$$

Discussion and Next Step

- Is "proposed SDNRG" interested in this topic ?
- Investigate relevant works and challenging issues
  - Develop or standardize new language ?
  - Or, define simple/minimum semantics for SDN ?
- Develop a *common* framework document for formally verifiable networking of SDN
  - Should be integrated with SDN architecture or framework works ?

References

- Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker, "Frenetic: A network programming language," in Proc. ACM International Conference on Functional Programming, September 2011
- Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schesinger, David Walker, "Abstractions for Network Update," HotSDN, 2012.
- Marco Canini, Daniele Venzano, Peter Peresini, Dejan Kostic, and Jennifer Rexford, "A NICE way to test OpenFlow applications," in Proc. Networked Systems Design and Implementation, April 2012
- A. Wang, L. Jia, C. Liu, B. Loo, O. Sokolsky, and P. Basu, Formally Verifiable Networking,2011.
- J. Choi, I. Lee, and H. Xie, The Specification and Schedulability Analysis of Real-Time Systems Using ACSR, 16th IEEE Real-Time Systems Symp.(RTSS'95), Dec. 1995.
- Nick McKeown, "Making SDNs Work," ONS2012
- 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.
- M. Kang et al., Formal Specification for Software-Defined Networks (SDN), CFI'12 (accepted), 2012.