Logic-based Synthesis of Software Defined Network Programs

Chen Chen, Limin Jia, Wenchao Zhou and Boon Thau Loo
University of Pennsylvania, Carnegie Mellon University, Georgetown University
Software Defined Networks

Control-plane

Network Operating System

Data-plane
Properties in SDN

– Correct static network configuration
  • Reachability: Node A is reachable from node B
  • Loop-freeness: No forwarding loop in the network

– Correct configuration update
  • Consistency: reachability and loop-freeness are not violated during update

– Correct implementation of the controller
  • Two-phase update: internal ports are updated before ingress ports
  • Non-repetitive configuration: each flow entry is installed only once

Challenge: State explosion
Our solution

• Observation
  – Not all properties are topology-dependent

• Logic-based synthesis of controller programs
  • Use a declarative language to encode the controller program 
    and switch functionality
  • A sound proof system for the language to prove topology-
    independent properties
  • Proof → synthesis
System specification

• Declarative programming:
  – Network Datalog (NDLog)
  – A set of derivation rules

\[
\text{rc1 flowMod(@swc, smac, iport) :-}
\]
\[
\text{ofConn(@ctl, swc),}
\]
\[
\text{ofPacket(@ctl, swc, iport, smac, dmac).}
\]

• Rule head: flowMod()
• Rule body: ofConn(), ofPacket()
• Location specifier: @
System specification in NDLog
Verify topology-independent properties

• Program logic:
  – Prove invariant properties of NDLog programs
  – Sound w.r.t. NDLog operational semantics

• Example:
  – Ethernet MAC learning
  – Property: flow installation follows table miss
    • Whenever a forwarding entry with destination “m” is installed on a switch s, there must have been a forwarding table miss at s before, triggered by a packet whose src address is “m”
Flow installation follows table miss

• Property specification:
  – First-order logic
  – flowEntry(swc, m, t) => ∃t’, flowMiss(swc, m, t’) & t’ < t

• Proof:
  – Switch invariant:
    • flowEntry(swc, m, t) => ∃t1, flowMod(swc, ctl, m, t1) & t1 < t
  – Controller invariant:
    • flowMod(swc, ctl, m, t) => ∃t2, ofPacket(ctl, swc, m, t2) & t2 < t
  – Switch invariant:
    • ofPacket(ctl, swc, m, t) => ∃t3, flowMiss(swc, m, t3) & t3 < t
Next step: synthesis

r1 ofPacket(@ctl, swc, m) :-
   flowMiss(@swc, m).

r2 flowEntry(@swc, m) :-
   flowMod(@swc, ctl, m).

flowEntry(swc, m, t) =>
∃t’, flowMiss(swc, m, t’) & t’ < t.

r1 flowMod(@swc,ctl,m) :-
   ∃fPacket(@ctl,swc,m).


Thank you & Questions?
State of the art

- Related work
  - Header Space Analysis: Static Checking For Networks (①)
    • P. Kazemian, et al. NSDI 2012
  - VeriFlow: Verifying Network-Wide Invariants in Real Time (①, ②)
    • A. Khurshid, et al. NSDI 2013
  - Abstractions for Network Update (②)
    • M. Reitblatt, et al. SIGCOMM 2012
  - A NICE Way to Test OpenFlow Applications (debugging ①, ②, ③)
    • M. Canini, et al. NSDI 2012
  - A Balance of Power: Expressive, Analyzable Controller Programming (③)

- Main challenge:
  - Network state explosion:
    • Packets, flow tables, ACLs...
Verify topology-dependent properties

Boolean satisfiability problem + SAT solver

Topology-dependent properties
Open issue

• Implementation choices:
  – Compile NDLog programs into low-level execution code
  – Invoke Openflow APIs directly from NDLog programs

• Proof automation:
  – Automatic invariant specification
  – Automatic proof process

• Program synthesis:
  – Proof $\rightarrow$ program
Properties category

- Topology-dependent properties
  - Reachability
  - No black holes in the network
  - Correct ordering of rule issuing to a number of switches
  - VLAN isolation

- Topology-independent properties
  - First packet of each flow is never dropped
  - Correct ordering of rule issuing to a single switch
  - Packets are either forwarded or processed by controller
  - The controller eventually processes all OpenFlow messages
  - Loop-freeness with BGP