VeriCon: Towards Verifying Controller Programs in Software-Defined Networks

Thomas Ball*, Nikolaj Bjørner*, **Aaron Gember†**, Shachar Itzhaky‡, Aleksandr Karbyshev¬, Mooly Sagiv‡, Michael Schapira° and Asaf Valadarsky°

**Microsoft Research, †University of Wisconsin, ‡Tel Aviv University, ¬Technische Universität München, °Hebrew University*

Motivation

Software defined networking (SDN) aims to simplify network management by removing the control plane from switches and running custom control applications at a logically central controller. Unfortunately, writing control applications that **always** maintain a set of network invariants (e.g., the network does not contain forwarding loops or blackholes) is a challenging task.

Prior work [1, 2] uses finite-state model checking and network snapshots to identify bugs in control applications. They can find errors, but they cannot guarantee the absence of errors.

Overview

VeriCon verifies network-wide invariants for ***any*** event sequence and ***all*** admissible topologies

SDN application in *Core SDN* + Topology constraints & invariants in first order logic → Verify conditions using the Z3 theorem prover → Guarantee invariants are satisfied **OR** Concrete counter-example

Types of Invariants

- **Topology**: define admissible topologies } *assumed to hold initially*
- **Safety**: define the required consistency of network-wide states } *checked initially & after each event*
- **Transition**: define the effect of executing event handlers

Learn More!

http://agember.com/go/vericon

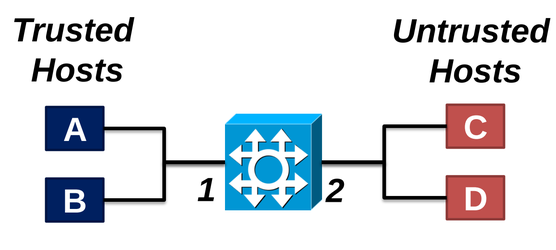
Core SDN (CSDN)

- Define and initialize relations: **rel** $r(\ )$ **init** $r = (\ )$
  - Topology relations: $link(S,O,H)$ $path(S,O,H)$
  - Forwarding relations: $ft(S,P,I,O)$ $fr(S,P,I,O)$
- Write packet-in event handlers: **pktIn**(S,P,I)
  - Update defined relations: r.**insert**$(\ )$ r.**remove**$(\ )$
  - Install rules (ft.**insert**): S.**install**(P,I,O)
  - Forward packet (fr.**insert**): S.**forward**(P,I,O)
  - Conditionals: **if** *Cond* **then** *Cmd** **else** *Cmd**

Verification Time

Program	LOCs	Topo Inv.	Safety + Trans Inv.	Time (sec)
Firewall	8	1	3 + 0	0.12
Stateless Firewall	4	1	2 + 0	0.06
Firewall + Host Migration	9	0	3 + 0	0.16
Learning Switch	8	1	4 + 2	0.16
Learning Switch + Auth	15	2	5 + 3	0.21
Resonance (simplified)	93	6	5 + 2	0.21
Stratos (simplified)	29	12	3 + 0	0.09

Example: Stateful Firewall

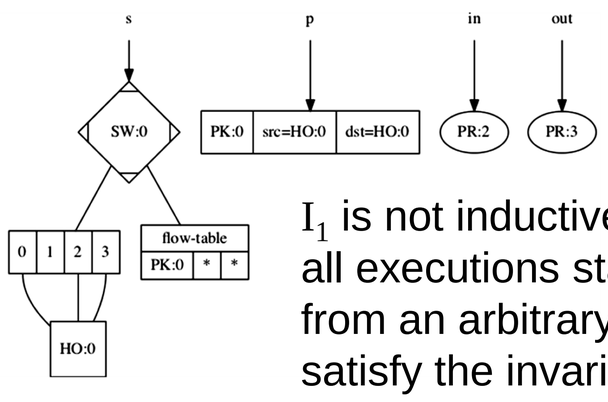


- Always forward from trusted to untrusted hosts
- Only forward from untrusted to trusted hosts if a trusted host previously sent a packet to the untrusted host

Application in Core SDN

rel tr(*SW*, *HO*)
pktIn(*sw*,*pkt*,*prt*(1)) →
  sw.**forward**(pkt,*prt*(1),*prt*(2))
  tr.*insert*(*s*,*pkt.dst*)
  sw.**install**(*pkt*,*prt*(1),*prt*(2))
pktIn(*sw*,*pkt*,*prt*(2)) →
  if tr(*sw*,*pkt.src*) **then**
    sw.**forward**(*pkt*,*prt*(2),*prt*(1))
    sw.**install**(*pkt*,*prt*(2),*prt*(1))

Counterexample



I_1 is not inductive—not all executions starting from an arbitrary state satisfy the invariant

$I_1 \wedge I_2 \wedge I_3$ is inductive

Invariants

Topology Invariant

- At least one switch with ports $prt(1)$ & $prt(2)$; a packet P is forwarded from an untrusted host U to a trusted host T

$$\exists U,T{:}\,HO, S{:}\,SW, P{:}\,PK.\ link(S,prt(2),U) \wedge link(S,prt(1),T) \wedge P.src = U \wedge P.dst = T \wedge fr(S,P,prt(2),prt(1))$$

Safety Invariants

- For every packet sent from an untrusted host U to a trusted host T there exists a packet sent to U from T

$$I_1 \stackrel{\text{def}}{=} fr(S,P,prt(2),prt(1)) \Rightarrow \exists P'.\ P'.dst = P.src \wedge fr(S,P',prt(1),prt(2))$$

- Flow table entries only contain forwarding rules from trusted hosts

$$I_2 \stackrel{\text{def}}{=} ft(S,P,prt(2),prt(1)) \Rightarrow \exists P'{:}\,PK.P'.dst = P.src \wedge fr(S,P',prt(1),prt(2))$$

- Controller relation tr stores the correct hosts

$$I_3 \stackrel{\text{def}}{=} \mathtt{tr}(S,H) \Rightarrow \exists P : PK.P.dst = H \wedge fr(S,P,prt(1),prt(2))$$

References

[1] Kazemian, P., Varghese, G., and McKeown, N. Header Space Analysis: Static Checking For Networks. In *Network Systems Design and Implementation (NSDI)*. 2012.
[2] Canini, M., Venzano, D., Peres, P., Kostic, D., and Rexford, J. A NICE Way to Test OpenFlow Applications. In *Network Systems Design and Implementation (NSDI)*. 2012.