VeriCon: Towards Verifying Controller Programs in SDNs

Thomas Ball, Nikolaj Bjorner, Aaron Gember, Shachar Itzhaky, Aleksandr Karbyshev, Mooly Sagiv, Michael Schapira, Asaf Valadarsky
Guaranteeing network invariants

• Network should always satisfy some invariants

• Difficult to write an SDN application that always guarantees such invariants
Limitations of existing approaches

1. Establish existence, but not absence, of bugs
   - **NICE** *(finite-state model checking)*: unexplored topologies may cause bugs to be missed
   - **HSA** *(check network snapshots)*: snapshots may not capture situations in which bugs exist

2. Runtime overhead
   - **VeriFlow & NetPlumber** *(check in real-time)*: bugs only identified when app is actually running
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
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
Core SDN (CSDN) language

• Define and initialize relations
  – Topology: \( \text{link} (S, O, H) \quad \text{link}(S_1, I_1, I_2, S_2) \)
  – Forwarding: \( S.\text{ft}(\text{Src} \rightarrow \text{Dst}, I \rightarrow O) \)
    \( S.\text{sent}(\text{Src} \rightarrow \text{Dst}, I \rightarrow O) \)

• Write event handlers: \( \text{pktIn}(S, Pkt, I) \)
  – Update relation
  – Install rule (insert into \( ft \))
  – Forward packet (insert into \( sent \))
  – If-then-else
Stateful firewall in CSDN

\[
\text{rel } \text{tr}(SW, HO) = \{ \}
\]

\[
\text{pktIn}(s, \text{pkt}, \text{prt}(1)) \rightarrow
\]

\[
\text{s.forward}(\text{pkt}, \text{prt}(1), \text{prt}(2))
\]

\[
\text{tr.insert}(s, \text{pkt.dst})
\]

\[
\text{s.install}(\text{pkt.src} \rightarrow \text{pkt.dst}, \text{prt}(1), \text{prt}(2))
\]

\[
\text{pktIn}(s, \text{pkt}, \text{prt}(2)) \rightarrow
\]

\[
\text{if } \text{tr}(s, \text{pkt.src}) \text{ then }
\]

\[
\text{s.forward}(\text{pkt}, \text{prt}(2), \text{prt}(1))
\]

\[
\text{s.install}(\text{pkt.src} \rightarrow \text{pkt.dst}, \text{prt}(2), \text{prt}(1))
\]
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
Stateful firewall invariants

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

  \[
  \exists U, T : HO, S : SW, P : PK. \\
  link(S, prt(2), U) \land link(S, prt(1), T) \land \\
  P.src = U \land P.dst = T \land S.sent(P, prt(2), prt(1))
  \]

- **Safety**: For every packet sent from a host $U$ to a host $T$ there exists a packet sent to $T'$ from $U$

  \[
  I_1 = \exists P' : PK. P'.dst = P.src \land S.sent(P', prt(1), prt(2))
  \]
Counterexample

$I_1$ is not inductive—not all executions starting from an arbitrary state satisfy the invariant
Additional firewall invariants

• Flow table entries only contain forwarding rules from trusted hosts

\[ I_2 = S. ft(Src \rightarrow Dst, prt(2), prt(1)) \Rightarrow \exists P': PK.P'.dst = Src \land S.send(P', prt(1), prt(2)) \]

• Controller relation \( tr \) records the correct hosts

\[ I_3 = tr(S, H) \Rightarrow \exists P : PK.P.dst = H \land S.send(P, prt(1), prt(2)) \]

• \( I_1 \land I_2 \land I_3 \) is inductive
## Non-buggy verification examples

<table>
<thead>
<tr>
<th>Program</th>
<th>Program LOCs</th>
<th>Topo Inv.</th>
<th>Safety + Trans Inv.</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firewall</td>
<td>8</td>
<td>1</td>
<td>3 + 0</td>
<td>0.12</td>
</tr>
<tr>
<td>Stateless Firewall</td>
<td>4</td>
<td>1</td>
<td>2 + 0</td>
<td>0.06</td>
</tr>
<tr>
<td>Firewall + Host Migration</td>
<td>9</td>
<td>0</td>
<td>3 + 0</td>
<td>0.16</td>
</tr>
<tr>
<td>Learning Switch</td>
<td>8</td>
<td>1</td>
<td>4 + 2</td>
<td>0.16</td>
</tr>
<tr>
<td>Learning Switch + Auth</td>
<td>15</td>
<td>2</td>
<td>5 + 3</td>
<td>0.21</td>
</tr>
<tr>
<td>Resonance (simplified)</td>
<td>93</td>
<td>6</td>
<td>5 + 2</td>
<td>0.21</td>
</tr>
<tr>
<td>Stratos (simplified)</td>
<td>29</td>
<td>12</td>
<td>3 + 0</td>
<td>0.09</td>
</tr>
</tbody>
</table>
## Buggy verification examples

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Counterex Host + Sw</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auth: Rules for unauth host not removed</td>
<td>3 + 2</td>
</tr>
<tr>
<td>Firewall: Forgot part of consistency inv</td>
<td>5 + 3</td>
</tr>
<tr>
<td>Firewall: No check if host is trusted</td>
<td>6 + 4</td>
</tr>
<tr>
<td>Firewall: No inv defining trusted host</td>
<td>6 + 4</td>
</tr>
<tr>
<td>Learning: Packets not forwarded</td>
<td>1 + 1</td>
</tr>
<tr>
<td><strong>Resonance: No inv for host to have one state</strong></td>
<td><strong>11 + 4</strong></td>
</tr>
<tr>
<td>StatelessFW: Rule allowing all port 2 traffic</td>
<td>4 + 2</td>
</tr>
</tbody>
</table>
Future work

• Assume events are executed atomically
  – Enforceable using barriers, with performance hit
  – Consider out-of-order rule installs

• Rule timeouts
  – App handles timeout events to update its $ft$ relation and check invariants
  – Need to reason about event ordering
Summary of VeriCon

• Verifies network-wide invariants for any event sequence and all admissible topologies

• Guarantees invariants are satisfied, or provides a concrete counterexample

• Application with 93 LOC and 13 invariants is verified in 0.21s

http://agember.com/go/vericon