Scalable Verification of Stateful Networks

Aurojit Panda, Ori Lahav, Katerina Argyraki, Mooly Sagiv, Scott Shenker
UC Berkeley, TAU, ICSI
Roadmap

• Why consider stateful networks?
• The current state of stateful network verification?
• VMN: Our system for verifying stateful networks.
• Scaling verification.
Why consider *stateful* networks?
Network State Increasingly Common

- 1/3rd of deployed network devices are middleboxes
- These are typically stateful (e.g., firewalls, caches, etc.)
- NFV will only make these more common
Network State Increasingly Common

• 1/3rd of deployed network devices are middleboxes
  • These are typically stateful (e.g., firewalls, caches, etc.)
  • NFV will only make these more common
• Later in this conference: stateful programming for P4 switches.
  • SNAP: Stateful Network-Wide Abstractions for Packet Processing
Network State Increasingly Common

• 1/3rd of deployed network devices are middleboxes
  • These are typically stateful (e.g., firewalls, caches, etc.)
  • NFV will only make these more common
• Later in this conference: stateful programming for P4 switches.
  • SNAP: Stateful Network-Wide Abstractions for Packet Processing
• Bottomline: Stateful is increasingly relevant.
Verification Checks Invariants

- We look at Reachability/Isolation invariants (same as stateless verification)
Verification Checks Invariants

- We look at Reachability/Isolation invariants (same as stateless verification)
- Packets from host A cannot reach host B
Verification Checks Invariants

• We look at Reachability/Isolation invariants (same as stateless verification)
  • Packets from host A cannot reach host B
• But statefulness raises some important issues:
Verification Checks Invariants

• We look at Reachability/Isolation invariants (same as stateless verification)
  • Packets from host A cannot reach host B
• But statefulness raises some important issues:
  • Invariants include temporal aspects.
Verification Checks Invariants

- We look at Reachability/Isolation invariants (same as stateless verification)
  - Packets from host A cannot reach host B
- But statefulness raises some important issues:
  - Invariants include temporal aspects.
  - Storing state can result in spooky action at a distance.
User 1 receives no packets from server 0 unless a connection is initiated.
User 1 receives no packets from server 0 unless a connection is initiated.

Standard Reachability

Temporal Property
User 0

User 1

Firewall

deny user1 server0

Cache

Server 0

Server 1

User 1 receives no packets from Server 0
User 1 receives no packets from Server 0
User 1 receives no packets from Server 0
User 1 receives no packets from Server 0
User 1 receives no packets from Server 0
User 1 receives no data from Server 0
Roadmap

• Why consider stateful networks?
• The current state of stateful network verification?
• VMN: Our system for verifying stateful networks.
• Scaling verification.
Network Verification Today

• Lots of existing work has looked at network verification.
Network Verification Today

- Lots of existing work has looked at network verification.
- Switches: Static forwarding rules in switches.
  
  HSA, Veriflow, NetKAT, etc.
Network Verification Today

• Lots of existing work has looked at network verification.
• Switches: Static forwarding rules in switches.
  HSA, Veriflow, NetKAT, etc.
• SDN Controller: Code generating these rules.
  Vericon, FlowLog, etc
Network Verification Today

• Lots of existing work has looked at network verification.
• Switches: Static forwarding rules in switches.
  
  HSA, Veriflow, NetKAT, etc.
• SDN Controller: Code generating these rules.
  
  Vericon, FlowLog, etc
• Testing for stateful networks
  
  Buzz: Generate packets that are likely to trigger interesting behavior.
Network Verification Today

- Lots of existing work has looked at network verification.
- Switches: Static forwarding rules in switches.
  - HSA, Veriflow, NetKAT, etc.
- SDN Controller: Code generating these rules.
  - Vericon, FlowLog, etc
- Testing for stateful networks
  - Buzz: Generate packets that are likely to trigger interesting behavior.
- Verification for stateful networks
  - SymNet: Uses symbolic execution to verify networks with middleboxes.
Roadmap

• Why consider stateful networks?
• The current state of stateful network verification?
• VMN: Our system for verifying stateful networks.
• Scaling verification.
VMN: System for scalable verification of stateful networks.
VMN Flow

1. Model each middlebox in the network
2. Build network forwarding model
3. Logical Invariants
4. SMT Solver (Z3 from MSR)
   - Invariant Holds
   - Example of violation
Modeling Middleboxes

• One approach: Extract model from code
Modeling Middleboxes

- One approach: Extract model from code
- **Problem**: At the wrong level of abstraction.
Modeling Middleboxes

- One approach: Extract model from code

**Problem**: At the wrong level of abstraction.

- Code written to match bit patterns in packet, etc.
Modeling Middleboxes

• One approach: Extract model from code

• **Problem**: At the wrong level of abstraction.
  • Code written to match bit patterns in packet, etc.
  • Configuration is in terms of higher level abstractions
Modeling Middleboxes

- One approach: Extract model from code

- **Problem**: At the wrong level of abstraction.
  - Code written to match bit patterns in packet, etc.
  - Configuration is in terms of higher level abstractions
    - E.g., source and destination addresses, payload matches regex, etc.
Modeling Middleboxes

• One approach: Extract model from code

• **Problem**: At the wrong level of abstraction.

  • Code written to match bit patterns in packet, etc.
  
  • Configuration is in terms of higher level abstractions
    
    • E.g., source and destination addresses, payload matches regex, etc.
    
    • Operators think and configure in terms of these abstractions.
Modeling Middleboxes

- One approach: Extract model from code

- **Problem**: At the wrong level of abstraction.
  - Code written to match bit patterns in packet, etc.
  - Configuration is in terms of higher level abstractions
    - E.g., source and destination addresses, payload matches regex, etc.
  - Operators think and configure in terms of these abstractions.
  - Verify invariants written in these terms.
Example Middlebox Configuration

• Drop all packets from connections transmitting infected files.
• How to define infected files: bit pattern for all worms: not really accurate
• Also not how operators think about this.
Modeling Middleboxes

• Take a different tack: model specified in terms of classification oracle.
  
  • Oracle responsible for classifying packet.

• We are not verifying implementation (nor is anyone else).
Modeling Middleboxes

• Take a different tack: model specified in terms of classification oracle.
  
  • Oracle responsible for classifying packet.

• We are not verifying implementation (nor is anyone else).

• Model specifies forwarding behavior in terms of these abstractions.
  
  • Need to know forwarding behavior to reason about reachability.

• Require that any state that affects forwarding behavior also specified.
Modeling Middleboxes
Modeling Middleboxes

Classify Packet

Determines what application sent a packet, etc. Complex, proprietary processing.
Modeling Middleboxes

Determines what application sent a packet, etc. Complex, proprietary processing.

Update state required for classification.
Modeling Middleboxes

- **Classify Packet**
  - Determines what application sent a packet, etc. Complex, proprietary processing.

- **Update Classification State**
  - Update state required for classification.

- **Update Forwarding State**
  - Update forwarding State.
Modeling Middleboxes

- Classify Packet
  - Update Classification State
    - Update Forwarding State
      - Forward Packet

Determines what application sent a packet, etc. Complex, proprietary processing.
Update state required for classification.
Update forwarding State.
Always simple: forward or drop packets.
Modeling Middleboxes

Oracle: Specify data dependencies and outputs

- Classify Packet
  - Update Classification State
    - Update Forwarding State
      - Forward Packet

Determines what application sent a packet, etc. Complex, proprietary processing.

Update state required for classification.

Update forwarding State.

Always simple: forward or drop packets.
Modeling Middleboxes

Oracle: Specify data dependencies and outputs
- Determines what application sent a packet, etc.
  Complex, proprietary processing.
- Update state required for classification.

Forwarding Model: Specify Completely
- Update forwarding State.
- Always simple: forward or drop packets.
Modeling Middleboxes

- Classify Packet
- Update Classification State
- Update Forwarding State
- Forward Packet
Modeling Middleboxes

Dependencies
See all packets in connection (flow).

Outputs
Is packet infected.
Modeling Middleboxes

Dependencies
See all packets in connection (flow).

Outputs
Is packet **infected**.

```java
if (infected) {
    infected_connections.add(packet.flow)
}
```
Modeling Middleboxes

Classify Packet

Update Classification State

Update Forwarding State

Forward Packet

Dependencies
See all packets in connection (flow).

Outputs
Is packet infected.

if (infected) {
  infected_connections.add(packet.flow)
}

if (packet.flow not in infected_connections) {
  forward (packet);
}
Modeling Middleboxes

\( \text{infected\_connection}(\text{flow}(p)) \)
\[ \implies (\Diamond \text{rcv}(n, p')) \wedge \]
\[ \text{flow}(p') = \text{flow}(p) \wedge \]
\[ \text{infected}(p) \]

\( \text{snd}(n, p) \implies \)
\[ (\Diamond \text{rcv}(n, p)) \wedge \]
\[ \neg \text{infected\_connection}(\text{flow}(p))) \]
VMN Flow

Model each middlebox in the network

Build network forwarding model

Logical Invariants

SMT Solver (Z3 from MSR)

Invariant Holds

Example of violation
Network Transfer Functions

- Kazemian 2012 developed the idea of a network transfer function.
  - A single function modeling the behavior of the entire network.
- VMN models static elements in the network using a transfer function.
Network Transfer Function
Network Transfer Function

\[
f(p, \text{port}) = \begin{cases} 
(p, f) & \text{if } \text{port} = A \land (\text{dst}(p) = C \lor \text{dst}(p) = D) \\
(p, c) & \text{if } \text{port} = f \land \text{dst}(p) = C \lor \text{dst}(p) = D \\
(p, C) & \text{if } \text{port} = c \land \text{dst}(p) = C \\
(p, D) & \text{if } \text{port} = c \land \text{dst}(p) = D \\
\ldots
\end{cases}
\]
Roadmap

- Why consider stateful networks?
- The current state of stateful network verification?
- VMN: Our system for verifying stateful networks.
- Scaling verification.
Networks are Large

• Networks are huge in practice
  • For example Google had 900K machines (approximately) in 2011
  • ISPs connect large numbers of machines.
• Lots of middleboxes in these networks
  • In datacenter each machine might be one or more middlebox.
• How do we address this?
Scaling Techniques Thus Far

- Abstract middlebox models
  - Simplify what needs to be considered per-middlebox.
- Abstract network
  - Simplify network forwarding.
Those Techniques are not Enough
Those Techniques are not Enough

- TACAS 2016: Network verification with state is EXPSPACE-complete.
Those Techniques are not Enough

- TACAS 2016: Network verification with state is EXPSPACE-complete.
- Practically for us SMT solvers timeout with large instances.
Those Techniques are not Enough

- TACAS 2016: Network verification with state is EXPSPACE-complete.
- Practically for us SMT solvers timeout with large instances.
- Other methods also do not handle such large instances
  - Symbolic execution is exponential in number of branches, not better.
Those Techniques are not Enough

- TACAS 2016: Network verification with state is EXPSPACE-complete.
- Practically for us SMT solvers timeout with large instances.
- **Other methods also do not handle such large instances**
  - Symbolic execution is exponential in number of branches, not better.
- Our techniques work for small instances, what to do about large instances?
Scaling Verification

• Challenge: Run verification on a subnetwork of size independent of network.

• Avoid instability and scale to arbitrary network sizes.
Scaling Verification

- Challenge: Run verification on a subnetwork of size independent of network.
  - Avoid instability and scale to arbitrary network sizes.
- Goal: Identify subnetwork where verification results translate to whole network.
Network Slices

- **Slices**: Subnetworks for which a bisimulation with the original network exists.
- Ensures equivalent step in subnetwork for each step in the original network
- Slices are selected depending on the invariant being checked.
Network Slices

Invariant: RR cannot access data from Coyote’s server
Network Slices

Invariant: RR cannot access data from Coyote’s server
Network Slices

Invariant: RR cannot access data from Coyote’s server
Network Slices

Invariant: RR cannot access data from Coyote’s server
Network Slices

Establishes a bisimulation between slice and network. Allows us to prove invariants in the slice.

Invariant: RR cannot access data from Coyote’s server.
Cannot always find such a slice.
Finding Slices: Flow Parallel Middleboxes

- To achieve performance, many middleboxes are flow parallel
  - State from one connection cannot affect another connection.
  - Example: Stateful firewall.
- For networks with only flow parallel NFs
  - Only need to consider paths between hosts.
  - Network slices whose slice is independent of network size.
Finding Slices: Origin Equivalence

- Middleboxes like caches don’t distinguish where a request originates
  - More generally, state is shared, but origin does not matter.
- In this case, need to ensure that all states in the network can appear in a slice.
  - Pick one member from each policy group.
- Scalable if increasing network size does not increase number of policy groups
Symmetry: Going Beyond Slices

• Slices merely reduce the size of the problem for each invariant
  • Number of invariants is still a problem.
• Rely on the observation that lots of hosts in networks are symmetric
  • Policies largely applied to groups of hosts (departments, etc.)
• Can use this symmetry to reduce number of invariants checked
Evaluation Setup: Datacenter

- Consider AWS like multi-tenant datacenter.
- Each tenant has policies for private and public hosts.
- Three verification tasks
  - Private hosts for one tenant cannot reach another
  - Public host for one tenant cannot reach private hosts for another
  - Public hosts are universally reachable.
Verification Time (Datacenter)

![Verification Time Graph]

- **Time (S)**: The x-axis represents the number of tenants, ranging from 0.01 to 100,000.
- **Priv-Priv**: Represented by blue bars.
- **Pub-Priv**: Represented by yellow bars.
- **Priv-Pub**: Represented by red bars.

The graph shows the verification time for different datacenter setups with varying numbers of tenants.
Role of Symmetry

- Consider a private datacenter
  - User verification to prevent some bugs from a Microsoft DC (IMC 2013)
- Bugs include
  - Misconfigured firewalls
  - Misconfigured redundant firewalls
  - Misconfigured redundant routing
- Measure time to verify as a function of number of symmetric policy groups
Verification Time (With Symmetry)

Graph showing the verification time for different categories: Rules, Redundancy, Traversal, and # of Policy Equivalence Classes. The x-axis represents the number of policy equivalence classes, and the y-axis represents time in seconds (S). The graph includes box plots for each category, indicating the distribution of verification times.
Conclusion

• Verifying stateful networks is increasingly more important.
• The primary challenge is scaling to realistic network.
• Splitting network into smaller verifiable portions is necessary.