Network Verification and Synthesis: Lessons from Hardware (and Software) Verification and Synthesis

Sharad Malik
Cornell-Princeton Center for Network Programming
6/20/2016
Need Strong Practical Motivation

High cost of failure
• Need for first silicon success
  • High mask costs
• Product recalls
  • Intel Pentium FDIV Bug 1994
  • Total cost: $475 million

Down time and security breach costs compelling for Network Verification
Scalability is Key

- Behavioral Spec
- Behavioral Synthesis
- RTL Description
- Functional Spec
- Functional + Timing Spec
- Logic Synthesis
- Gate-level Netlist
- Detailed Logic Design
- Layout Synthesis
- Physical Layout
- Full Chip Equivalence Checking is Routine
Watch the complexity barrier: PSPACE-complete vs. NP-complete

Model Checking
State space exploration:
Need to store sets of states

Bounded Model Checking
Propositional Logic, SAT based analysis:
Search, but no state storage

\[ X(t+1) = f(X(t), I) \]
Snapshot Verification

- Verify the static network state
  - A snapshot of a dynamic system
  - A single SDN rule configuration
  - No performance verification

- Network state change (rule deletion/addition/change at a switch)[1]
  - Tens of events per second
  - Packet arrival rate
  - Millions of arrivals per second

Modeling/Analysis Challenge

• Even for a single packet entering a network, a link may see multiple packets

• Switch output not a combinational function of its inputs

Need to store sets of values
Adapting Modeling/Analysis

• Limit packet flow to a *single path for a single packet* through the network

• Captures only part of the network behavior

• What good is this?
Goal: Counterexamples for Property Failures

Single Path Single Packet Counterexample

Suffices for

• Functional Properties:
  • Reachability checking
    • Waypointing
    • Blacklisting

• Functional/Performance Properties:
  • Forwarding loop

• Security Properties:
  • Slice isolation
    • virtualization context
Evaluation

Setup
• SAT solver: Minisat
• Stanford backbone network
  • 16 routers with full network functions (VLAN, ACL, ...)
  • ≈ 15,000 rules
  • 129 seconds to find a forwarding loop
    • Header Space Analysis (HSA): 758 seconds
    • Uses Ternary Symbolic Simulation
• Synthetic benchmarks for scalability experiments
  • Fat tree topology
  • Shortest path routing
  • Depth-first-search to generate matching rules
• Vary
  • # of switches: N
  • # of routes: P
  • # of packet header bits: H
Evaluation

• Property
  • Forwarding loop check

• Setup
  • Vary
    • # Routes
    • # of Header bits
  • HSA: Header Space Analysis
  • SAT: SAT-based method

• Observations
  • Sub-exponential growth with number of routes
  • Low dependence on header size
Evaluation

• Property
  • Reachability check

• Setup
  • Vary
    • # Routes
    • # of Header bits
  • HSA: Header Space Analysis
  • SAT: SAT-based method

• Observations
  • Sub-exponential growth with number of routes
  • Low dependence on header size

Small number of equivalence classes of packets
Controller Verification: Challenges

- Large number of packets alive in network
  - Large buffer state
  - Large interleaving state
- Large number of rules installed in switches
  - Large network state

![Routing Table Diagram]

- Port 1: inPkt.src = Host_1
- Port 2: inPkt.src = Host_3
- Port 3: inPkt.src = Host_k
- Port_p: inPkt.src = Host_r
- Port_q: inPkt.src = Host_a

outPort(inPkt) =
Abstractions are Key

- State Space Traversal
  - PSPACE-completeness
- Abstractions to manage state-explosion
  - Over-abstractions
    - No false negatives

Figure Source: Valeria Bertacco
Abstraction: Handling Large Number of Packets

Environment packets ($\text{pkt}_e$) simulate the affect of an unbounded number of packets.
Evaluation

• Verified a learning switch
  • No packet gets into a loop in the network

• A buggy stateful firewall example
  • No source host gets unnecessarily blocked by the firewall
  • Detected known bug: a host did get blocked
Synthesis

**Hardware**

- Compile-time optimization of circuits

**Software**

- Program sketching [ASPLOS’06, ICSE’10, …]
  - Fill in program holes


From Verification to Synthesis: Firewall Case Study

• Firewall Equivalence Checking
  • $\mathcal{P} = \mathcal{F} \downarrow A \not\approx \mathcal{F} \downarrow B$
    • $\mathcal{P}$ satisfiable $\rightarrow$ not equivalent
    • $\mathcal{P}$ unsatisfiable $\rightarrow$ equivalent
Firewall Synthesis

• Firewall Synthesis
  • Firewall with the fewest rules for a given specification

• Symbolic Firewalls
  • Represents all firewalls with $k$ rules

\[ R = \{(r_{1,0}, r_{1,1}, r_{1,2}, ...), (r_{2,0}, r_{2,1}, r_{2,2}, ...),
      (r_{3,0}, r_{3,1}, r_{3,2}, ...), ..., (r_{k,0}, r_{k,1}, r_{k,2}, ...)\} \]

- Each assignment to $R$ specifies one firewall
Firewall Synthesis

- Find an $R$, if one exists, such that for all $B$, $g$ holds
- Binary search for minimum $k$
- Practical QBF (and special purpose) solvers do not scale well

$\exists R \forall B \ (g)$

Quantified Boolean Formula (QBF)

Watch the complexity barrier!
QBF is PSPACE-complete

$\mathcal{F} \downarrow A \equiv \mathcal{F} \downarrow B$

Packet $B = \{ b_{\downarrow 1}, b_{\downarrow 2}, ..., b_{\downarrow N} \}$

Symbols $R = \{ (r_{\downarrow 1,0}, r_{\downarrow 1,1}, r_{\downarrow 1,2}, ...), ((r_{\downarrow 2,0}, r_{\downarrow 2,1}, r_{\downarrow 2,2}, ...), 3,0, r_{\downarrow 3,1}, r_{\downarrow 3,2}, ...), ..., ((r_{\downarrow k,0}, r_{\downarrow k,1}, r_{\downarrow k,2}, ...)) \}$

Binary search for minimum $k$
Summary

Verification
• Scalability barriers
  • NP-complete vs. PSPACE-complete
• Implementation verification is invaluable
• Abstractions are key

Synthesis
• Compile-time optimization opportunities
• Patch in holes
  • Debugging
    • Fixing configuration files
    • Network updates
• Large-scale synthesis?