Formal Verification of Computer Switch Networks

Sharad Malik; Department of Electrical Engineering; Princeton Univeristy
(with Shuyuan Zhang (Princeton), Rick McGeer (HP Labs))

SDN: So what changes for verification?

- Previously
  - System complexity precluded formal modeling and verification
  - Relied exclusively on testing based techniques
    - traceroute, ping, tcpdump, wireshark
- Now
  - Hardware
    - Switch network is purely hardware (finite state)
    - Can apply hardware verification techniques
  - Software
    - Centralized control algorithm, easier to analyze
- However
  - Hardware
    - Large network size
      - Switches: From tens to hundreds
      - Rules per switch: From hundreds to thousands
  - Software
    - Interacts with distributed hardware

Hardware Snapshot Verification

- Verify the static network state at a single instance of time
  - A snapshot of a dynamic system
  - Do not consider network performance, e.g. delay, bandwidth, …
- Verify consistency of updates separately
  - Reitblatt, Foster, Rexford, and Walker. 2011. Consistent updates for software-defined networks: change you can believe in!. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks* (HotNets-X)
- Rationale
  - Network state change (rule deletion/addition/change at a switch)[1]
    - Tens of events per second
  - Packet arrival rate
    - Millions of arrivals per second

[1] Gude, N., Koponen, T., Pettit, J., Pfa, B., Casado, M., McKeown, N., Shenker, S.: "Nox: towards an operating system for networks"

Talk Goals/Outline

- Review specific verification efforts
  - Formalisms
    - Modeling
    - Verification Tasks
  - Emphasis on verification engines
    - Model checking
    - Symbolic simulation
    - SAT based propositional logic verification
  - With insights on their applicability
- From verification to design synthesis
  - Formal methods based optimal synthesis of network components

Packet State ≡ System State

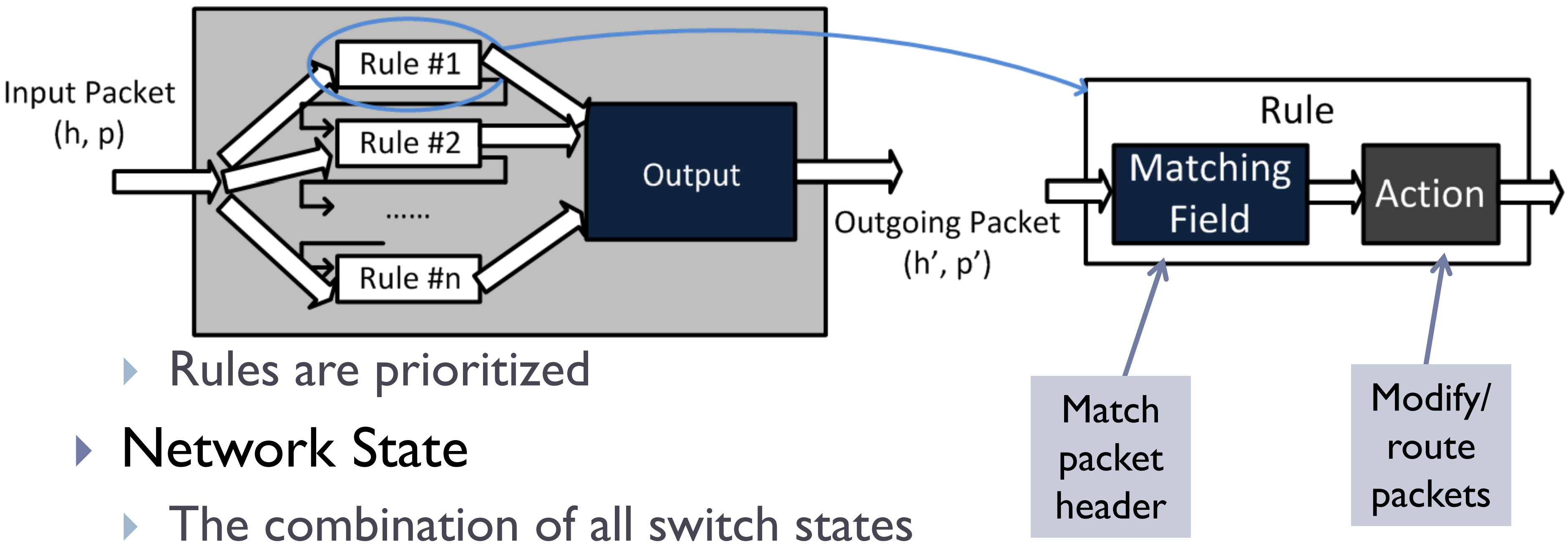
- Verification is packet centric
- Packet State
  - (packet header, packet location)
    - (h,p)
  - Ignore payload
  - Packet state transitions during network traversal
- State Space Size
  - Packet Header

Bit #	0~31	32~63	64~79	80~95	96~103	104~207
Pkt	Src IP	Dst IP	Src port	Dst port	Protocol	Src IP’, …… , Proto’

  - Packet Location
    - Global Port ID
      - Stanford campus network: 47 ports, 6 bit encoding

Network State

- Switch State
  - Set of rules defining how a packet is processed
  - Routing Information Base, Forwarding Information Base, Access Control List, Forwarding Table, Configuration Policies…



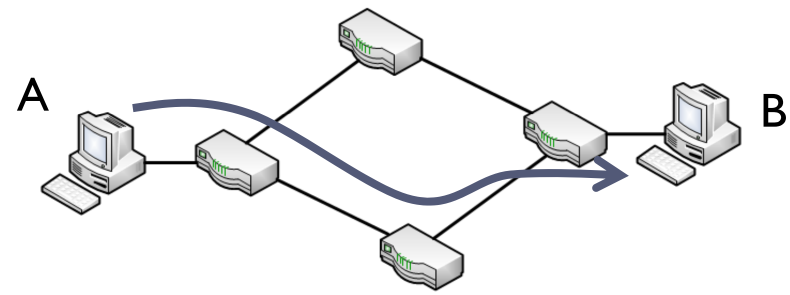
  - Rules are prioritized
- Network State
  - The combination of all switch states
  - Fixed → Snapshot verification

Talk Goals/Outline

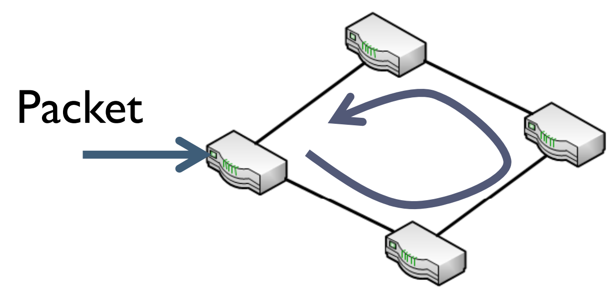
- Review specific verification efforts
  - Formalisms
    - Modeling
    - Verification Tasks
  - Emphasis on verification engines
    - Model checking
    - Symbolic simulation
    - SAT based propositional logic verification
  - With insights on their applicability
- From verification to design synthesis
  - Formal methods based optimal synthesis of network components

Network Properties

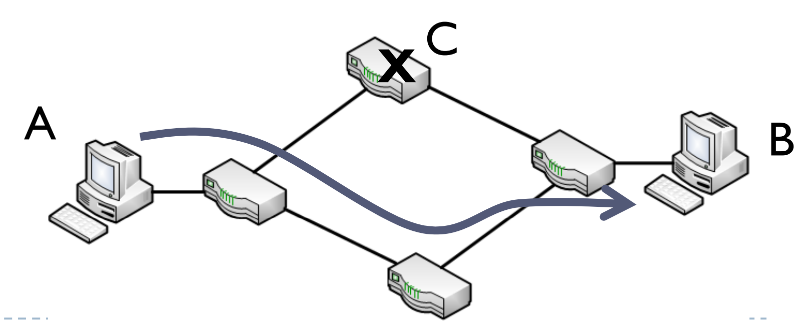
- Reachability Checking:
  - Check if a packet can always reach B from A.



- No Forwarding Loop:
  - Make sure there is no packet that can reach the same switch/port more than once during its lifetime.

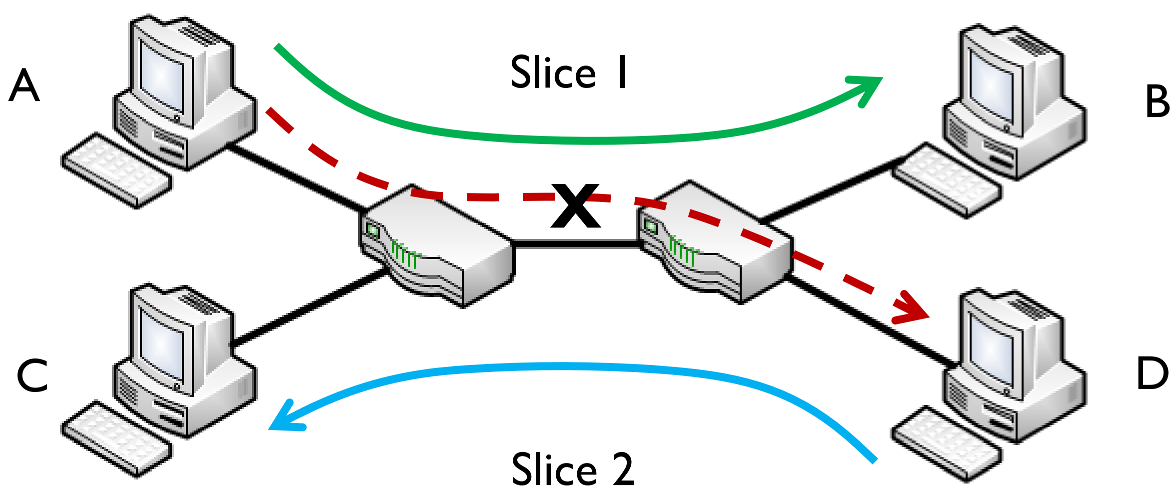


- Packet Destination Control:
  - Make sure a packet can/cannot go through certain switches/hosts.



Slice Isolation

- Network infrastructure shared by several users (e.g. corporations)
  - Virtualization
  - Each user network defined by a slice
- Network Slice[2]:
  - Network space (packet headers × location)
  - Forwarding rules
- Slice Isolation: Definition
  - A packet that belongs to Slice 1 should not enter Slice 2.



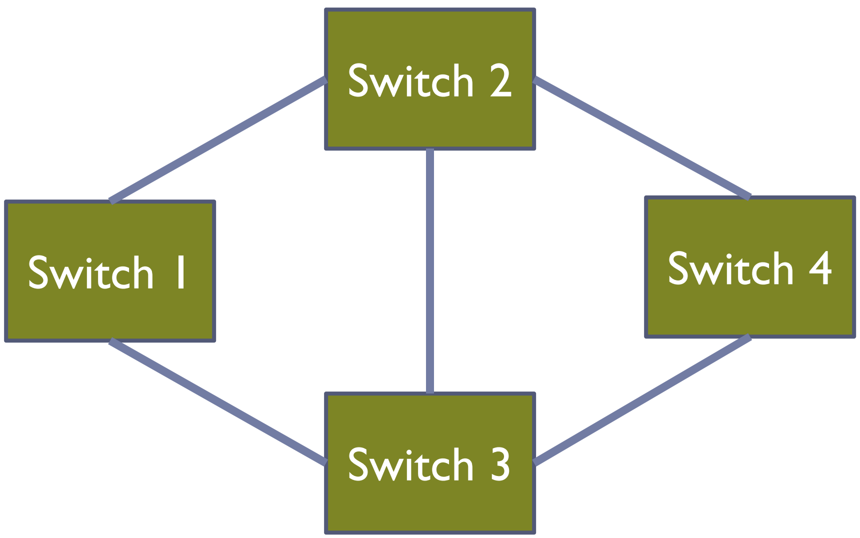
[2] Kazemian, P., Varghese, G., McKeown, N.: "Header space analysis: static checking for networks"

Talk Goals/Outline

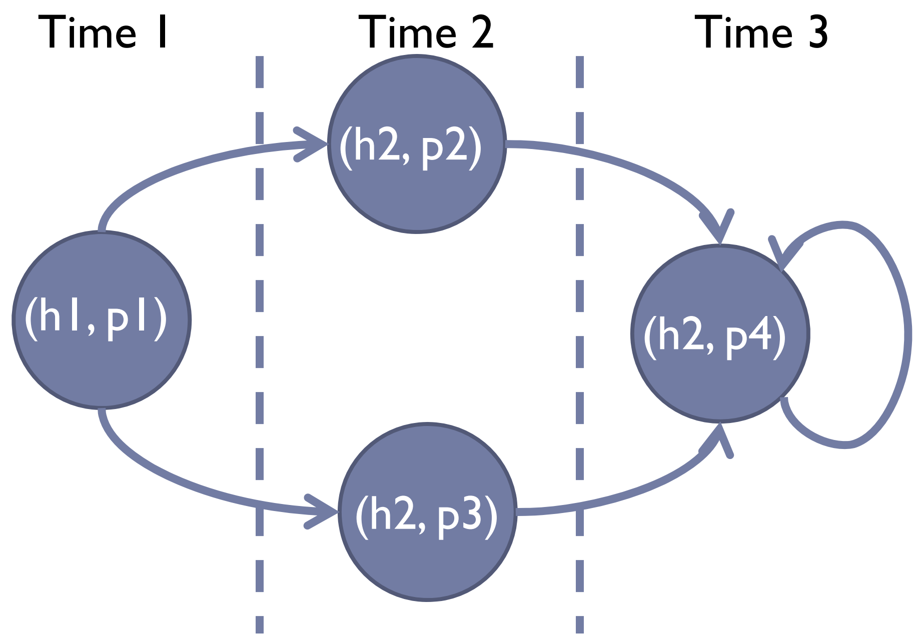
- Review specific verification efforts
  - Formalisms
    - Modeling
    - Verification Tasks
  - Emphasis on verification engines
    - Model checking
    - Symbolic simulation
    - SAT based propositional logic verification
  - With insights on their applicability
- From verification to design synthesis
  - Formal methods based optimal synthesis of network components

Model Checking Based Verification

- Transition of packet states
  - Given a packet, FSM based approaches model how the packet transitions during its lifetime.



Real Network

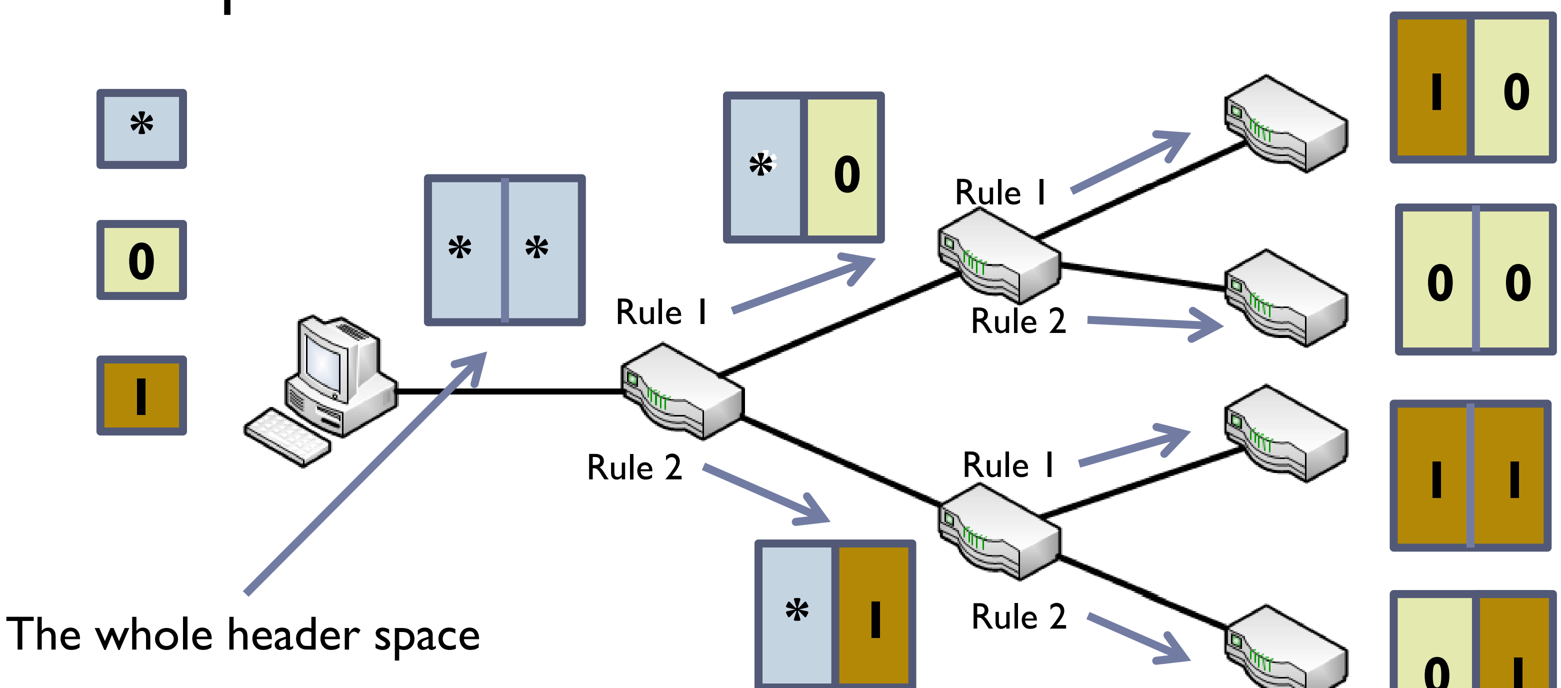


Transition Model

- Properties specified using temporal logic formulas
  - CTL: Computation Tree Logic

Header Space Analysis: Ternary Symbolic Simulation Implementation

- Can follow a symbolic packet through the network
- Example:



- Limitation
  - No clean formalism to express/check properties

Reachability Analysis

- Packets can reach from A to B
- Model Checking Based Approach
  - CTL Property
    - $(p=A) \rightarrow AF\ (p=B)$
- Ternary Symbolic Simulation
  - Follow the symbolic packet along all possible paths

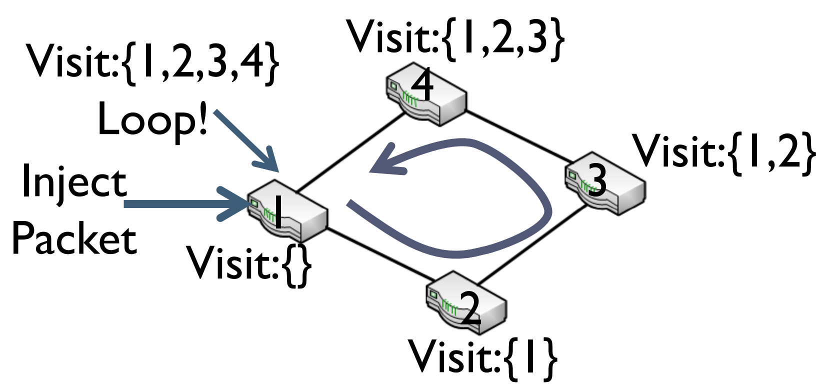
AF: Along **A**ll paths there some **F**uture state

Forwarding Loop

- Model Checking
  - ConfigChecker:
    - ¬((p=A) → EX (EF (p=A)))
  - Atomic Update:
    - AF (p=drop V p=(outside world))

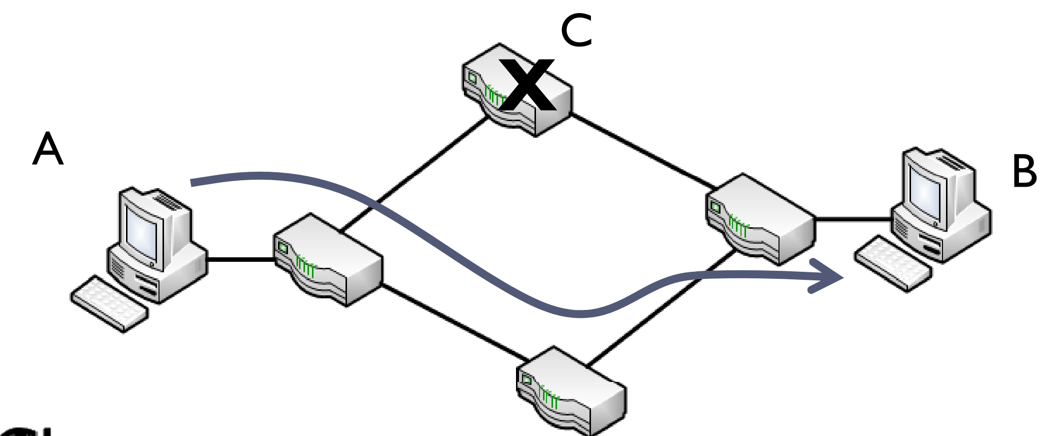
drop, outside world are encoded as some port ID

- Ternary Symbolic Simulation
  - Maintain a visit history.
  - Whenever we have visited the current switch before, a loop is found.
  - Book-keeping overhead



Packet Destination Control

- Example:
  - All packets from A get to B without reaching C.

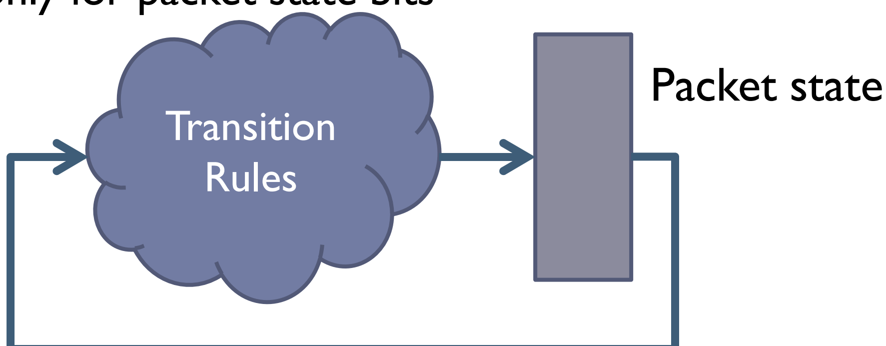


- Model Checking
  - $(p{=}A) \rightarrow (AF(p{=}B) \wedge AG(p{!=}C))$
- Ternary Symbolic Simulation
  - Inject the whole space at A ([* * *…* *])
  - Returns false if a packet gets to C or ends up with places other than B

Experimental Evidence: BDD Based Model Checking

BDD: Binary Decision Diagram

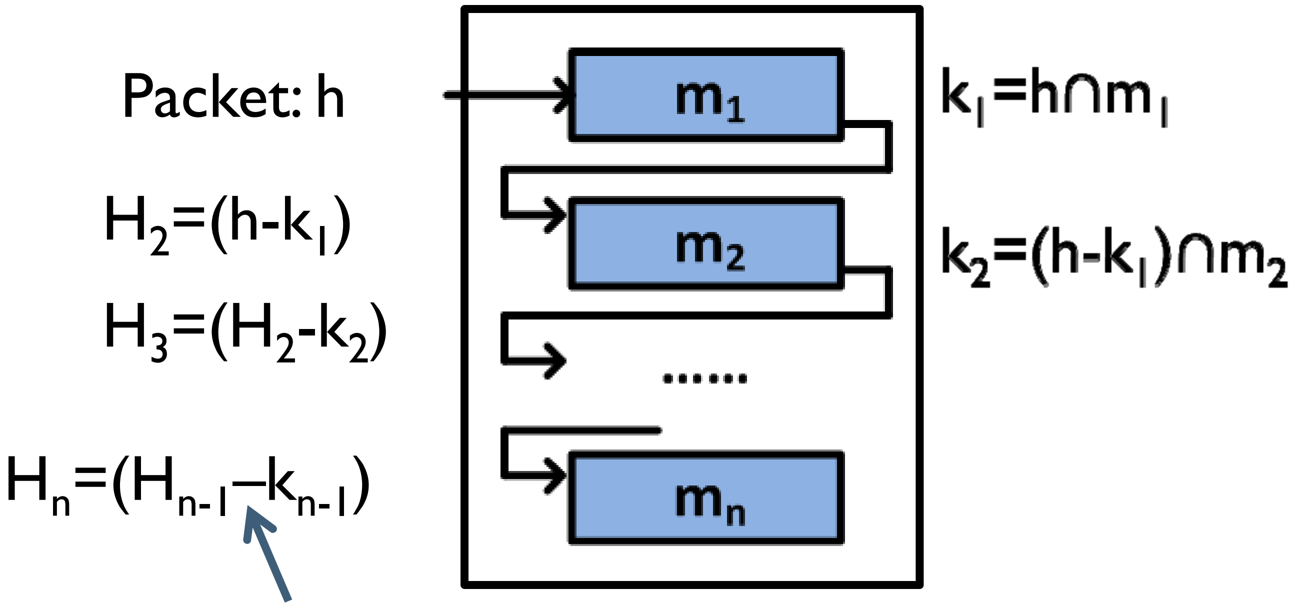
- Scalability:
  - # of variables in transition relation
    - Header bits: OpenFlow v1.1 → 15 matching fields → 356 matching bits
    - Network size: 47 ports (as in Stanford campus) → 6 bits
- Experimental Result:
  - ConfigChecker: 111 bits for header + (largest) 4000 nodes
  - Atomic Update: 64 bits header + Hundreds of switches + hundreds of thousands of rules → over an hour
- Why does this even work?
  - Space: Largest part of the system is the rules
    - BDD variables only for packet state bits



  - Time: Shallow transition systems. Packets go through relatively few hops.

Experimental Evidence: Ternary Symbolic Simulation

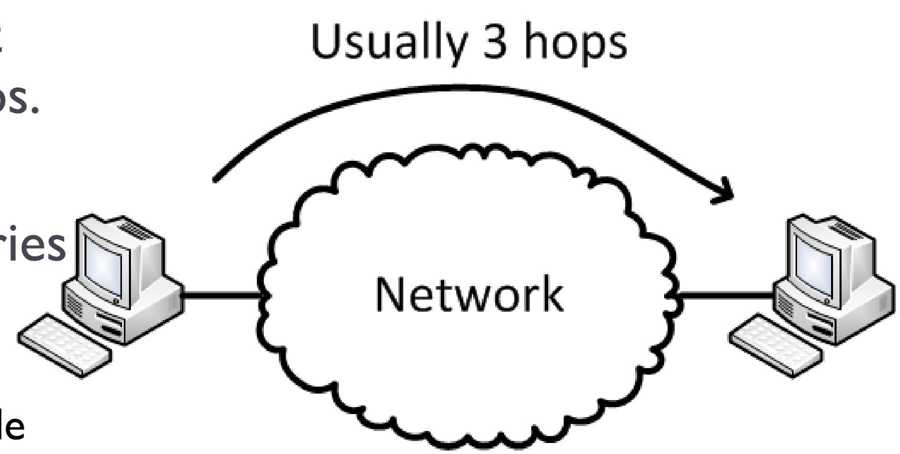
- Potential Difficulty:



  - Operation “-” is expensive in ternary symbolic simulation
    - It is equivalent to DNF complementation.

Experimental Evidence: Ternary Symbolic Simulation

- Experimental result:
  - Stanford campus network:
    - 2 backbone routers + 14 zone routers + 10 switches
    - # of forwarding rules after compression: 4,200 (originally 757,000)
  - Loop Detection on 30 ports: 560 seconds
- Why does this even work?
  - Shallow transition system: A packet reaches its destination in a few hops.
  - Rule overlaps are small
  - Limited number of packet trajectories
    - Exploited in incremental verification
      - Khurshid, Zhou, Caesar, and Godfrey. 2012. VeriFlow: verifying network-wide invariants in real time. HotSDN '12



Talk Goals/Outline

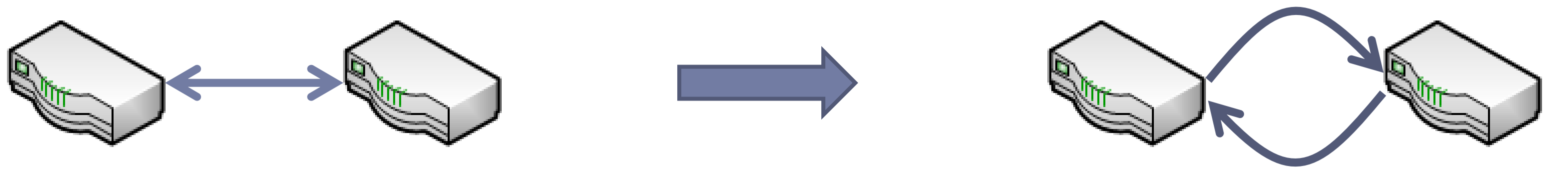
- Review specific verification efforts
  - Formalisms
    - Modeling
    - Verification Tasks
  - Emphasis on verification engines
    - Model checking
    - Symbolic simulation
    - SAT based propositional logic verification
  - With insights on their applicability
- From verification to design synthesis
  - Formal methods based optimal synthesis of network components

From Model Checking to SAT

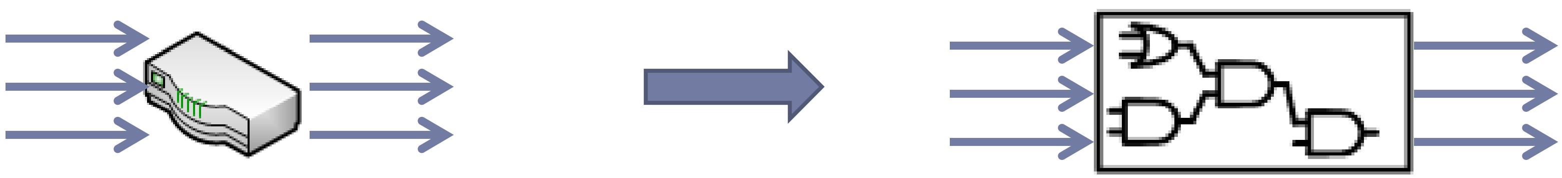
- Model Checking vs. SAT
  - Higher in the complexity hierarchy
- Ternary Symbolic Simulation
  - Properties are hard to specify
  - Book-keeping overhead (e.g. check forwarding loop)
- Can we model the network as a combinational circuit?
  - Propositional logic model
  - SAT based property checking

SAT Based Verification: An Overview

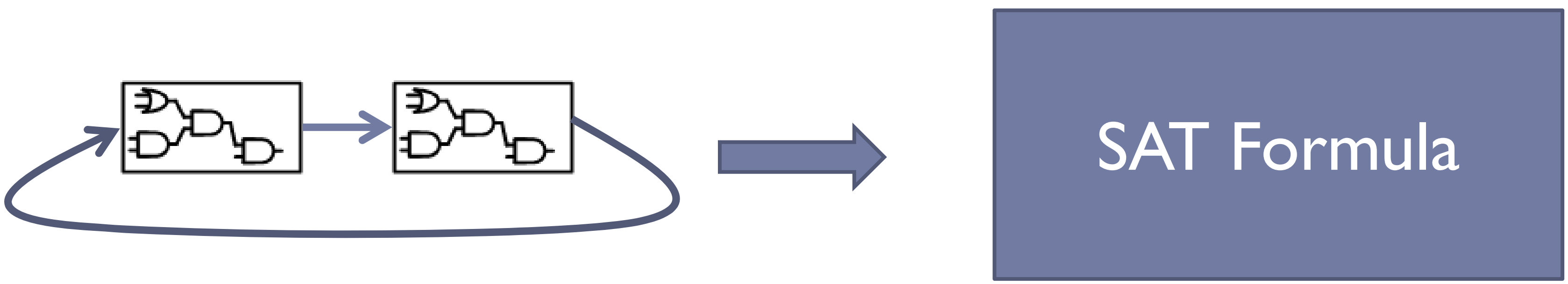
- Split one bidirectional link into two unidirectional links



- Switch can be modeled as acyclic combinational logic

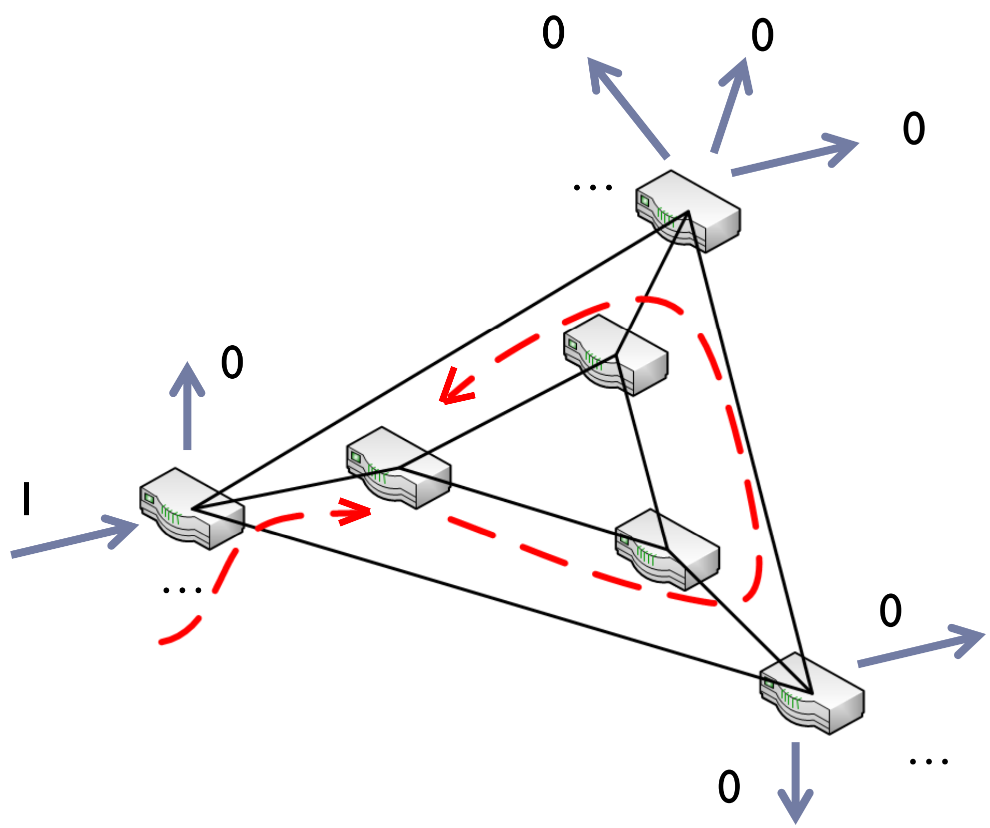


- Use traditional hardware verification techniques.



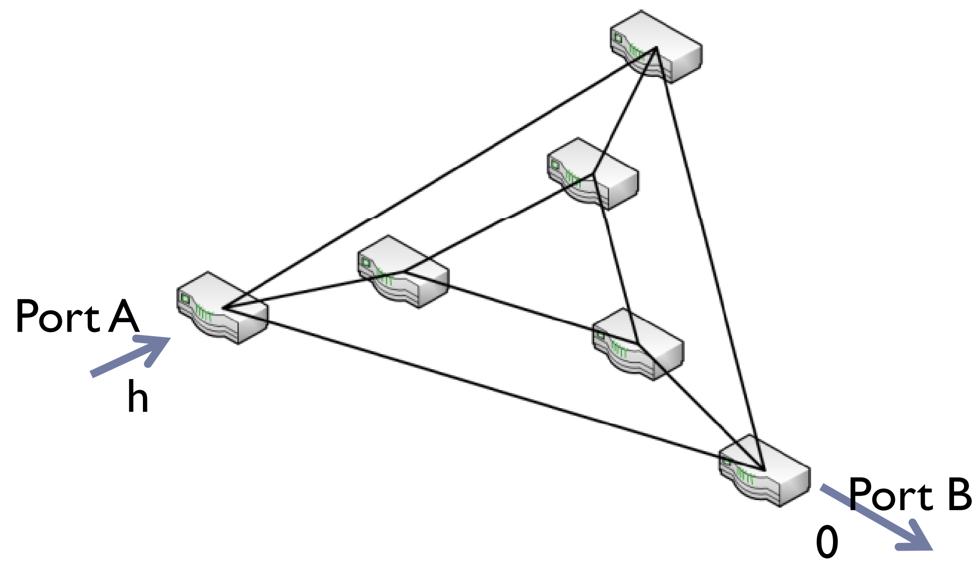
Encoding Property: Find A Forwarding Loop

- Forwarding Loop:
  - The same packet shows up at the same switch twice, not necessarily with the same header format
- Assumption:
  - There is a packet entering the network
- Constraint:
  - No packet gets out.
  - No packet is dropped.
- Return:
  - SAT: find forwarding loop
  - UNSAT: no forwarding loop



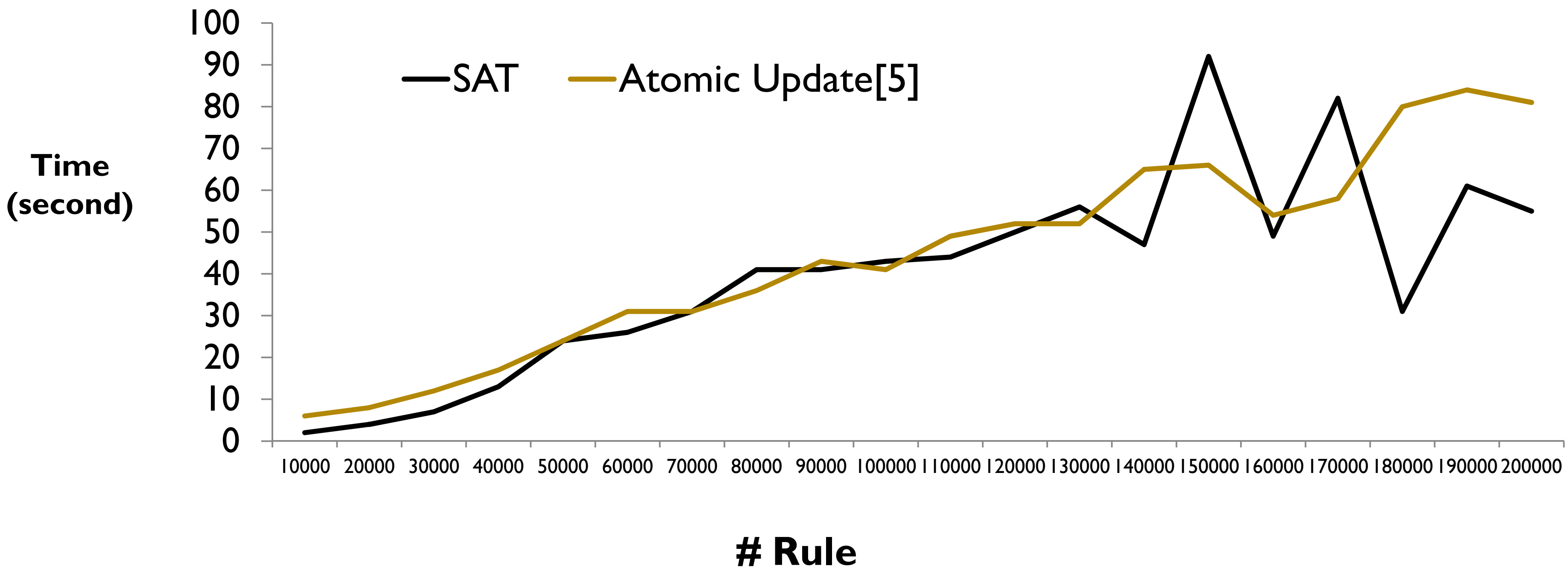
Encoding Property: Reachability Checking

- Example properties:
  - Packets with format h=10xx… will always get to B from A.
- Constraint:
  - Packet h=10xx… enters the network at port A
  - No packet shows up at port B
- Return:
  - SAT: Reachability fails
  - UNSAT: Reachability holds



Preliminary Results

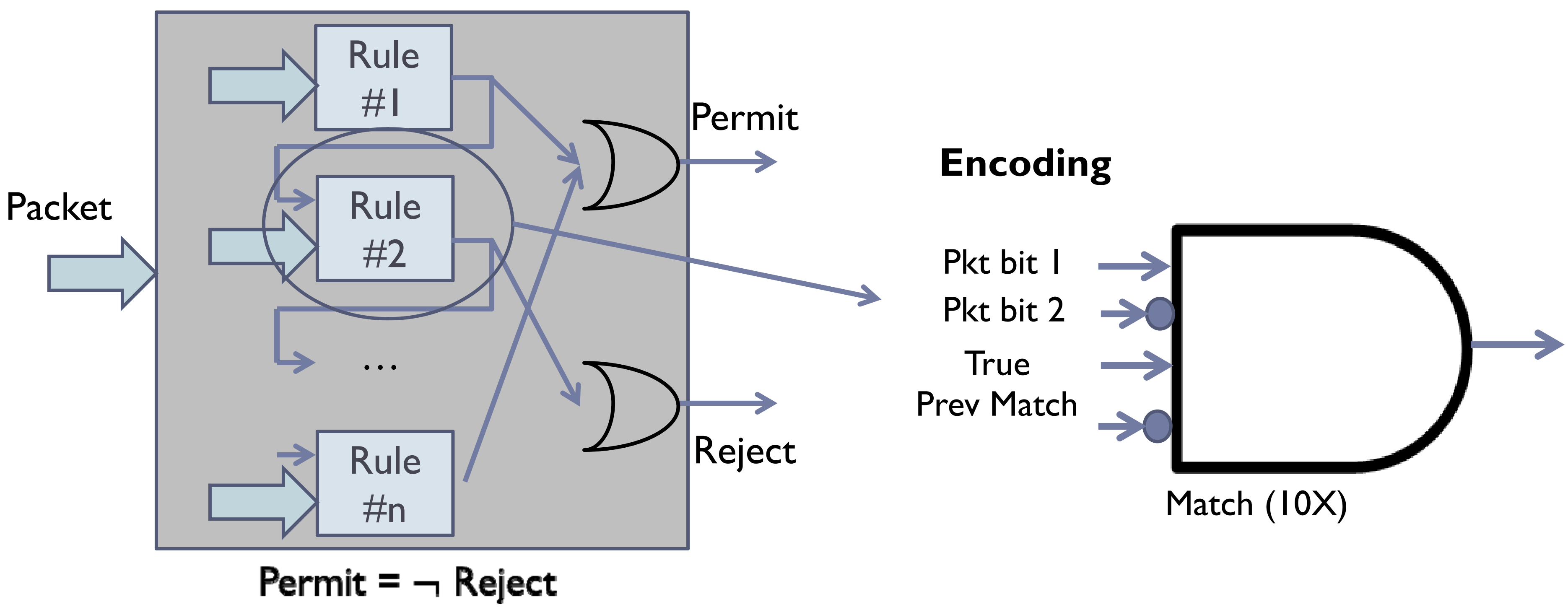
- Forwarding Loop
  - Waxman topology
    - 10 switches+1000 hosts
    - Policy: shortest path between certain port pairs
    - Property: Check if there is forwarding loop.
  - 200 switches + 1000 hosts + 300,000 rules → 11 minutes
  - 200 switches + 1000 hosts + 750,000 rules → 3 hours and 48 minutes
  - 200 switches + 1000 hosts + 2,700,000 rules → Run out of memory



[5] Reitblatt, M., et al..: "Abstractions for network update"

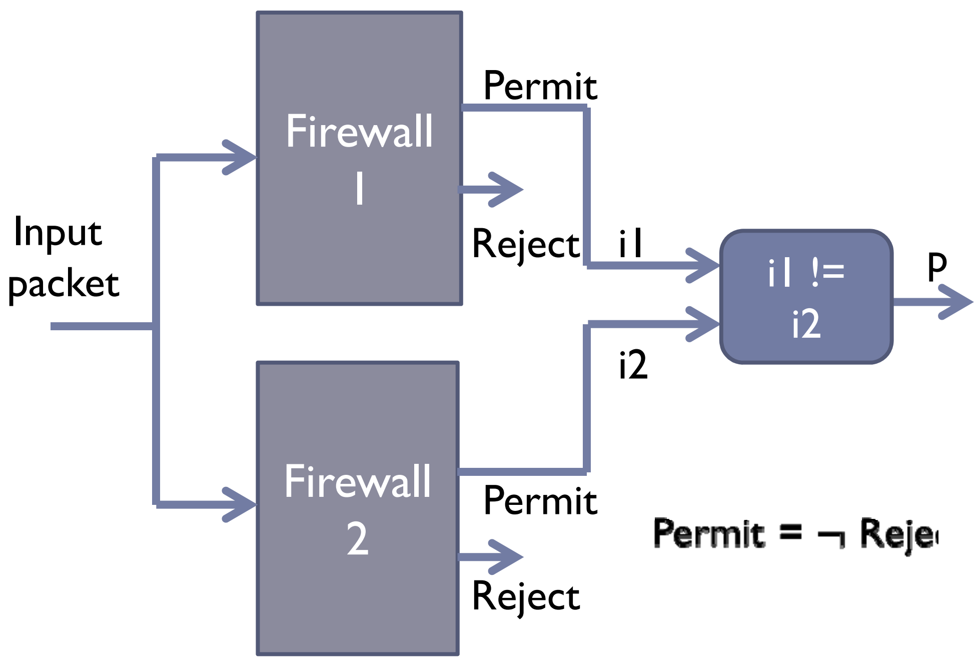
SAT Based Firewall Verification

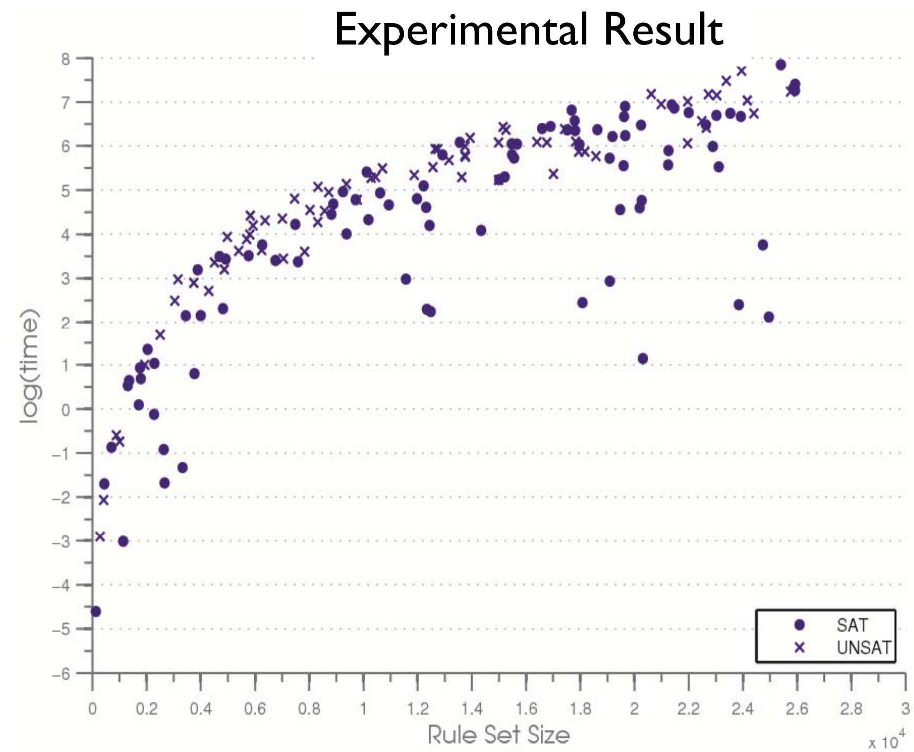
- Firewall
  - Inputs: Incoming packet
  - Outputs: "accept" or "reject" action
- Firewall Encoding



Firewall Equivalence Check

- Feed the same input to the two firewalls and check if the two outputs can differ.

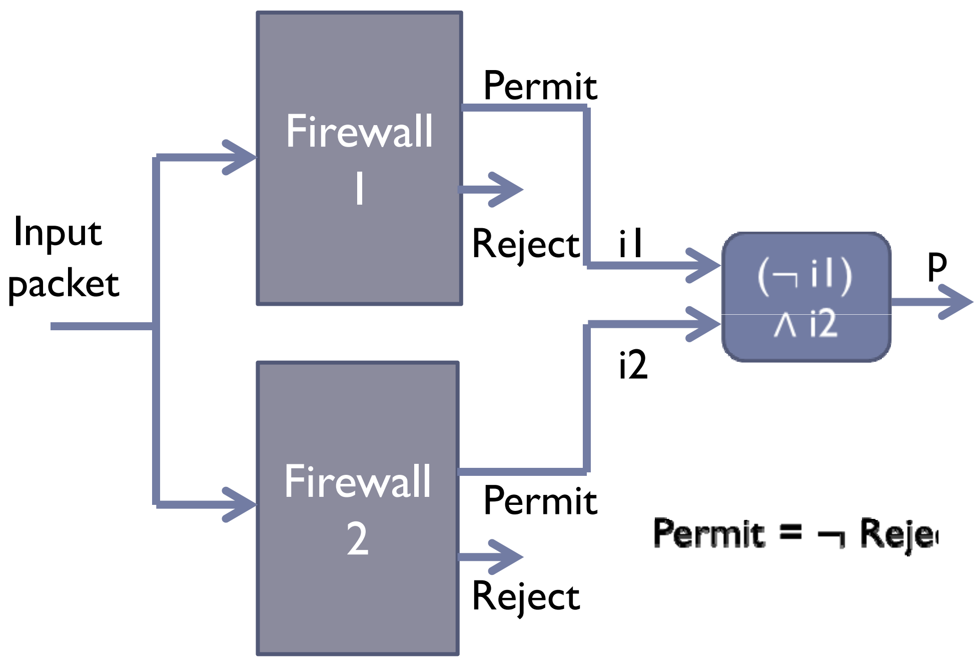




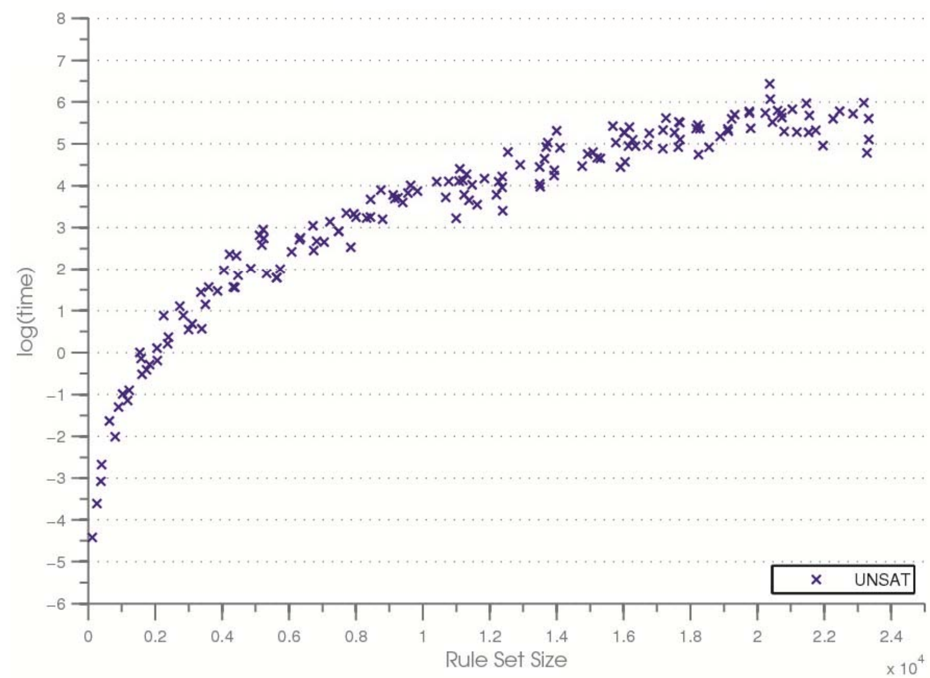
Classbench for firewall generation

Firewall Inclusion Check

- **Check if one firewall is stricter (i.e., drops more packets) than the other.**
  - If $(\neg i1) \wedge i2$ is satisfiable, Firewall 2 is not stricter than Firewall 1.



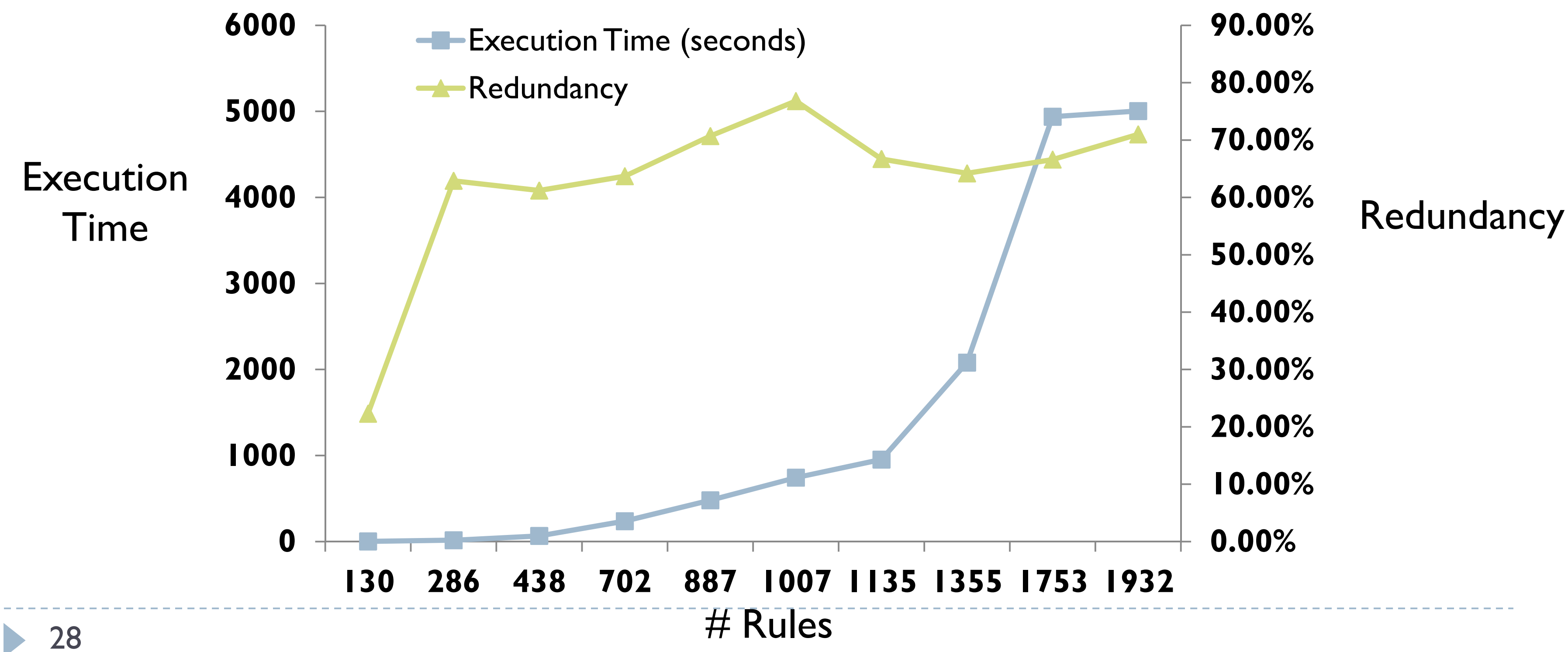
Experimental Result



Classbench for firewall generation

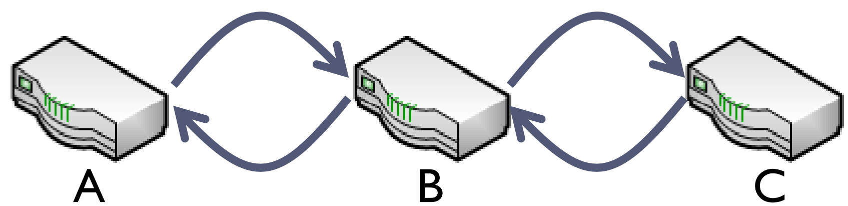
Firewall Redundancy Removal

- Single rule redundancy checking
  - Delete it and check the equivalence of the new firewall with the old old
  - If they are equivalent, delete the rule
- Sequentially iterate over all rules



Other SAT Formulations: Anteater[6]

- **Primary property: Reachability checking**
  - Based on the connectivity between two adjacent switches.
- **Example:**

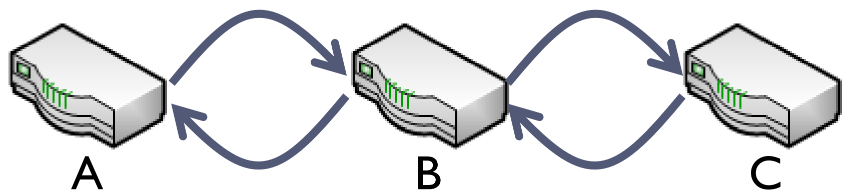


  - Connectivity between A and B:
    - P(A, B) = (dst_IP=B) V (dst_IP=C)
    - P(B,A) = (dst_IP=A)
  - Connectivity between B and C:
    - P(B, C) = (dst_IP=C)
    - P(C, B) = (dst_IP=A) V (dst_IP=B)

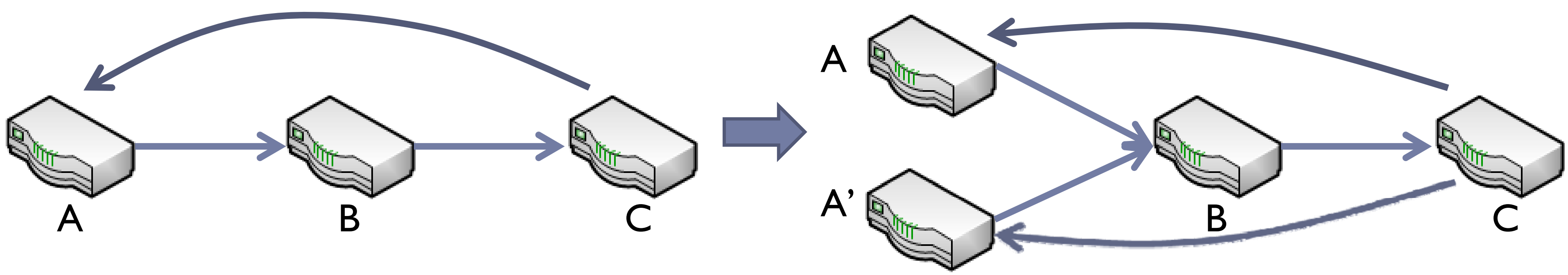
[6] Mai, H., Khurshid, A., Agarwal, R., Caesar, M., Godfrey, P.B., King, S.T.: "Debugging the data plane with anteater"

Property Checking for Anteater

- Reachability



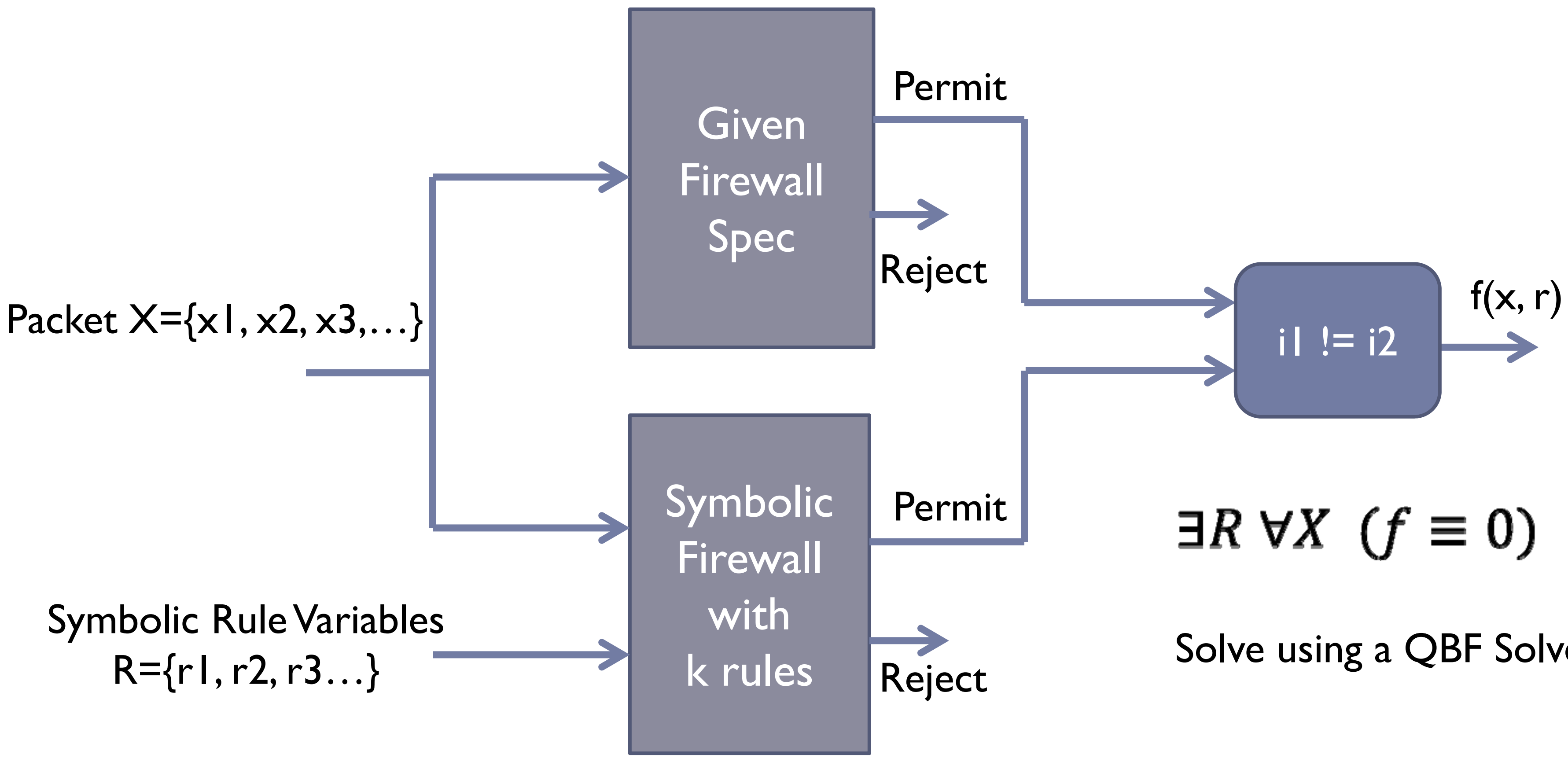
  - Path Analysis
    - P(A, B) ∧ P(B, C)
- Forwarding Loop
  - Create a dummy switch
  - Check reachability from A to A'
  - Need to do this for all switches in the loop, for all loops



Talk Goals/Outline

- Review specific verification efforts
  - Formalisms
    - Modeling
    - Verification Tasks
  - Emphasis on verification engines
    - Model checking
    - Symbolic simulation
    - SAT based propositional logic verification
  - With insights on their applicability
- From verification to design synthesis
  - Formal methods based optimal synthesis of network components

Firewall Synthesis



$$\exists R \ \forall X \ \ (f \equiv 0)$$

Solve using a QBF Solver

Current QBF Solvers don't scale ☹

Wrap Up

- Summary
  - Reviewed emerging Symbolic Simulation/Model Checking/SAT based approaches.
- Challenges
  - Speed
    - Ternary Symbolic Simulation: 10 switches + 2 backbone router,s a total of 4,200 forwarding rules (after compression) → 10 minutes.
    - Model Checking Based (using NuSMV): Hundreds of switches + hundreds of thousands of rules → Over an hour.
    - Current SAT Based Propositional Property Checking: Similar in scale
  - What we need:
    - Verification between two network updates → continuous verification
      - Explore incremental verification techniques
  - Network Application Verification
    - Opportunities for tailored software verification techniques

References

[1] Kazemian, P., Varghese, G., McKeown, N.: "Header space analysis: static checking for networks"

[2] Al-Shaer, E., Marrero, W., El-Atawy, A., ElBadawi, K.: "Network configuration in a box: towards end-to-end verification of network reachability and security"

[3] Al-Shaer, E., Al-Haj, S.: "FlowChecker: configuration analysis and verification of federated OpenFlow infrastructures"

[4] Reitblatt, M., Foster, N., Rexford, J., Schlesinger, C., Walker, D.: "Abstractions for network update"

[5] Mai, H., Khurshid, A., Agarwal, R., Caesar, M., Godfrey, P.B., King, S.T.: "Debugging the data plane with anteater"

[6] Gude, N., Koponen, T., Pettit, J., Pfa, B., Casado, M., McKeown, N., Shenker, S.: "Nox: towards an operating system for networks"