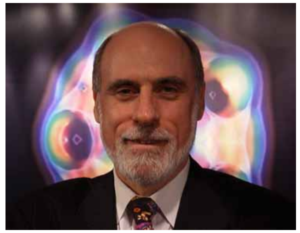


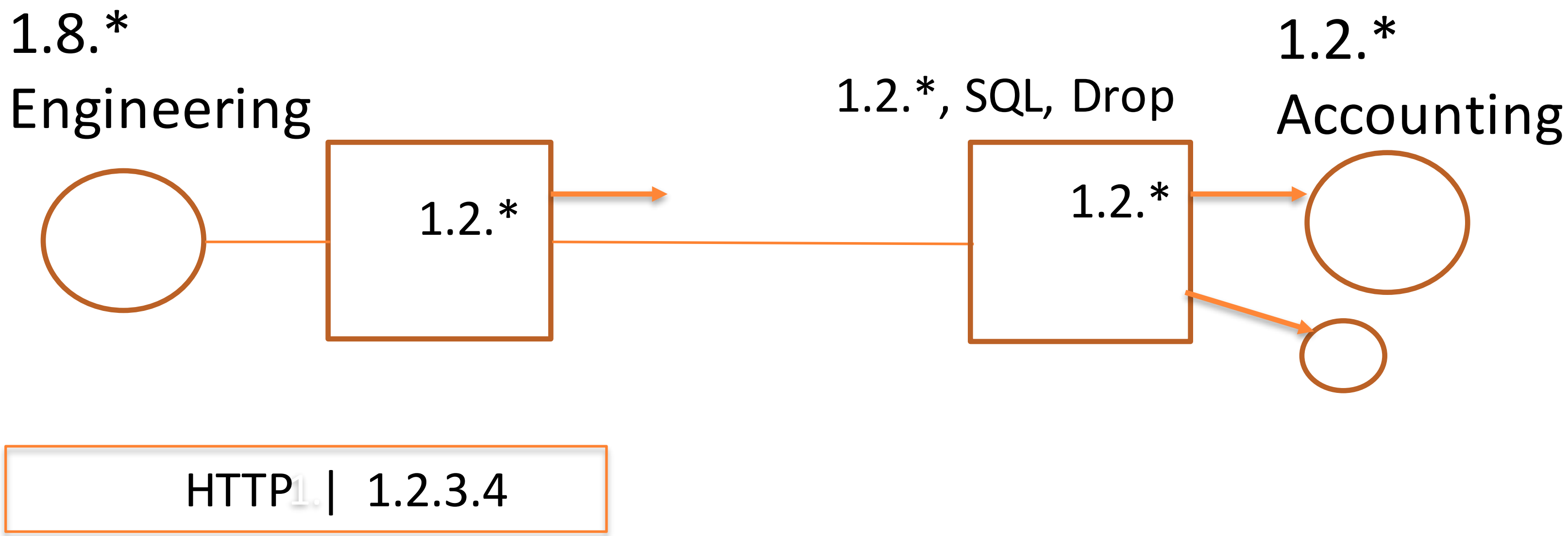
TOOLS

FOR PUBLIC CLOUDS,
PRIVATE CLOUDS,
ENTERPRISE
NETWORKS, ISPs, . . .

NETWORK VERIFICATION: WHEN CLARKE MEETS CERF

George Varghese
UCLA
(with collaborators from CMU, MSR, Stanford, UCLA)

Model and Terminology



- Routers, links, interfaces
- Packets, headers
- Prefix match rules, manually placed Access Control (ACL) rules

Problem with Networks today

- Manual Configurations: Managers override default shortest paths for security, load balancing, and economic reasons
- Data Plane + Control Plane: Vendor-specific knobs in both
- Problem: Manually programming *individual* routers to implement *global* policy leads to cloud failures

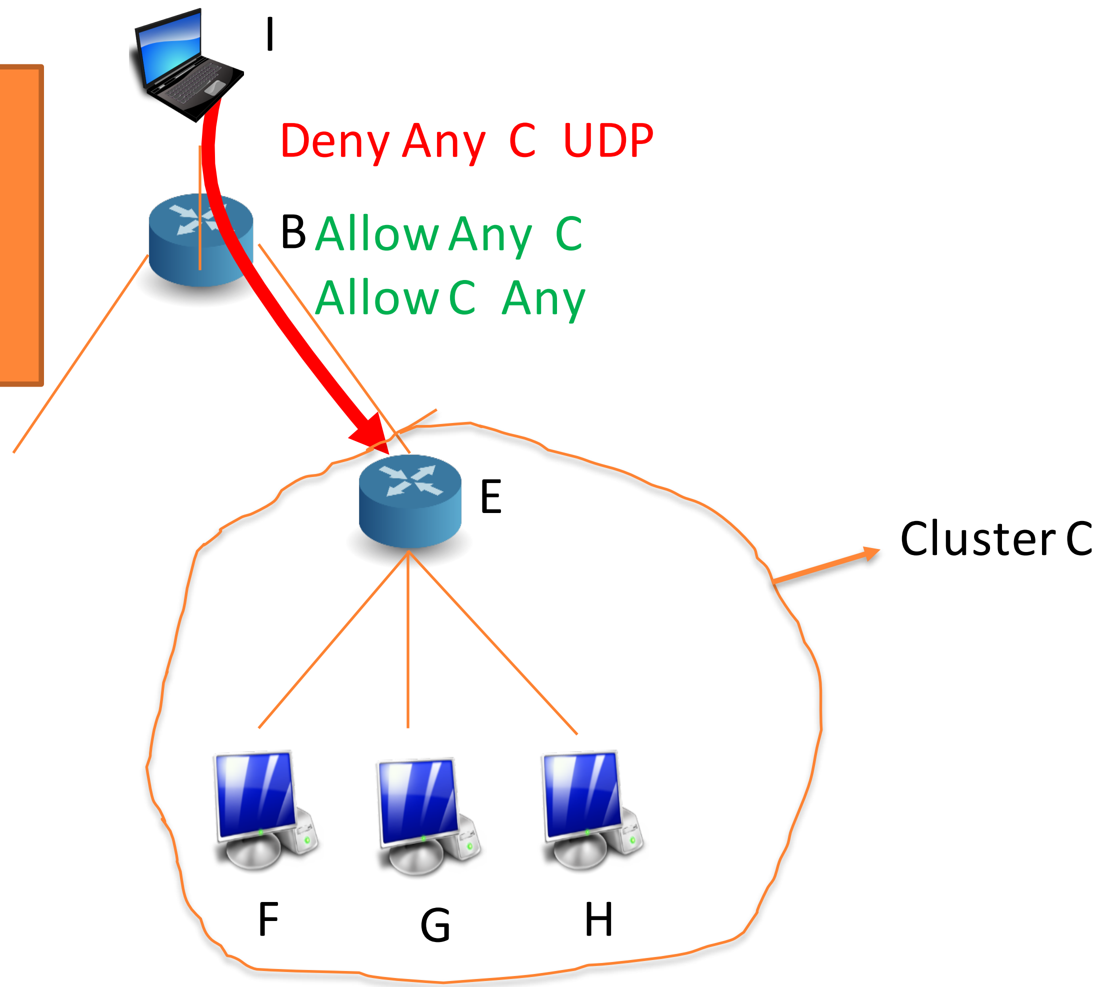
Manual Traffic “steering knobs”

- Data forwarding/Data Plane:
  - Access Control Lists (predicates on headers)
  - VLANs (a way to virtualize networks)
  - MAC Bridging Rules (ACLs at the Ethernet Level)
- Routing/ Control Plane:
  - Communities: equivalence classes on routes via a tag
  - Static routes: a manager supplied route
  - Local preference: “priority” of a route at this router regardless of global cost of the route

Managers use all these knobs for isolation, economics

Why manual reasoning is hard

POLICY
- Internet and Compute can communicate
- Internet cannot send to controllers



DNS Services are now blocked!

Why automated reasoning is imperative

- Challenges: 2^{100} possible headers to test!
  - Scale: devices (1000s), rules (millions), ACL limits (< 700)
  - Diversity: 10 different vendors, > 10 types of headers
  - Rapid changes (new clusters, policies, attacks)
- Severity: (2012 NANOG Network Operator Survey):
  - 35% have 25 tickets per month, take > 1 hour to resolve
  - Welsh: vast majority of Google "production failures" due to "bugs in configuration settings"
  - Amazon, GoDaddy, United Airlines: high profile failures

As we migrate to services ($100B public cloud market), network failure a debilitating cost.

Simple questions hard to answer today

- Which packets from A can reach B?
- Is Group X provably isolated from Group Y?
- Is the network causing poor performance or the server?
- Why is my backbone utilization poor?

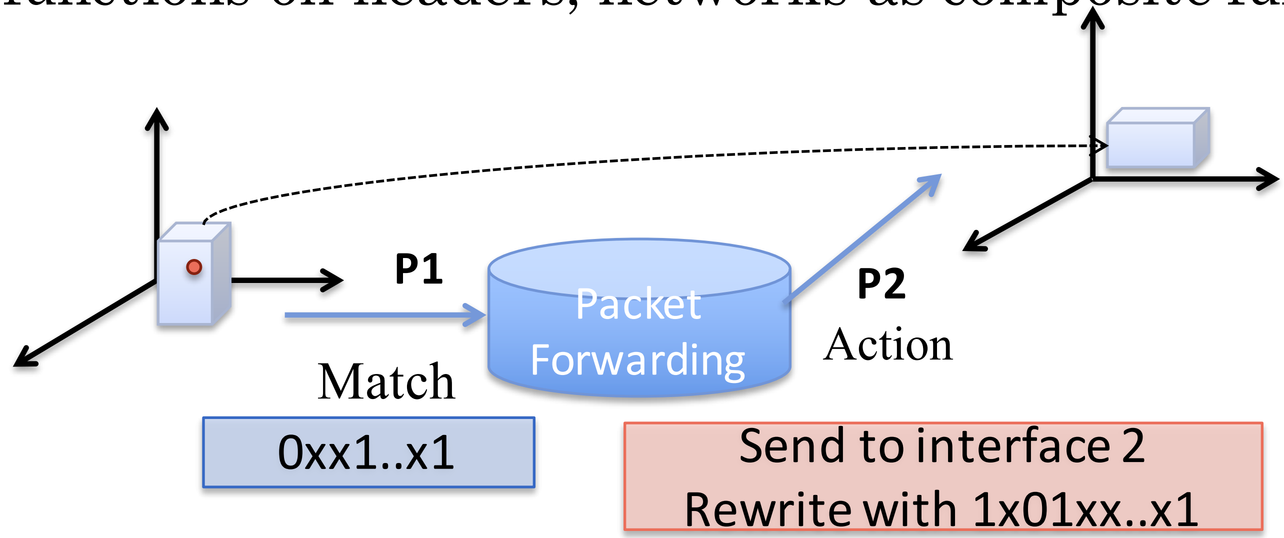
NEED BOTTOM UP ANALYSIS OF EXISTING SYSTEMS

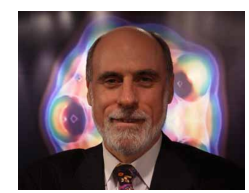
Formal methods have been used to verify (check *all* cases) large programs and chips (FMCAD!)

Can we use formal methods across *all* headers, & inputs for large clouds?

Approach: Treat Networks as Programs

- Model header as point in header space, routers as functions on headers, networks as composite functions





CAN NOW ASK WHAT THE EQUIVALENT OF *ANY* PROGRAM ANALYSIS TOOL IS FOR NETWORKS

Problems addressed/Outline

- Classical verification tools can be used to design static checkers for networks but do not scale
  - Part 1: Scaling via Symmetries and Surgeries (POPL 16)
- Bugs exist in the routing protocols that build forwarding tales
  - Part 2: Control Plane Verification (OSDI 2016)
- A vision for Network Design Automation (NDA)

Scaling Network Verification

(Plotkin, Bjorner, Lopes, Rybalchenko, Varghese, POPL 2016)

- exploiting regularities in networks
- symmetries and surgeries

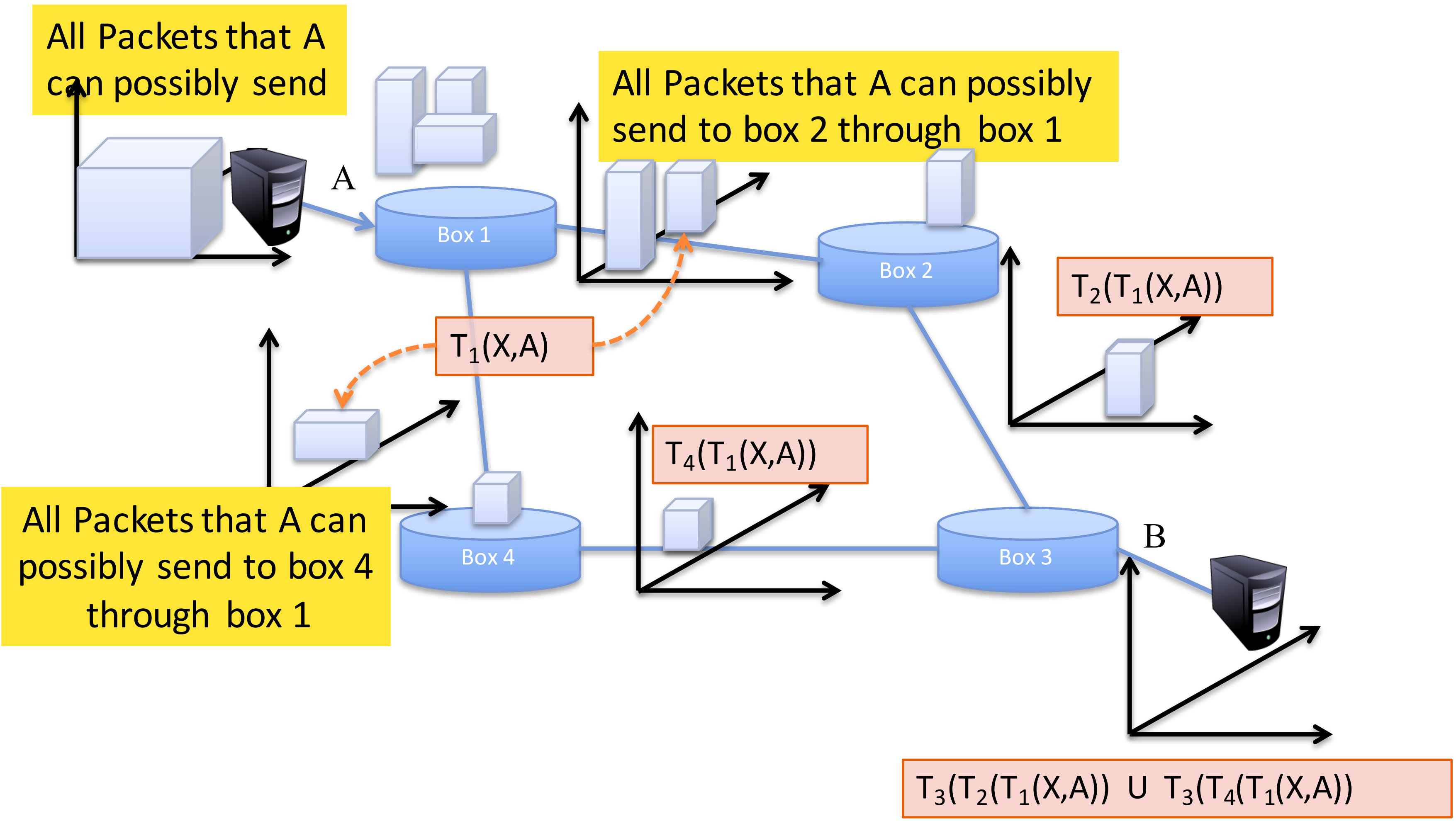
Formal Network Model [HSA 12]

- 1 - Model sets of packets based on relevant header bits, as subsets of a $\{0,1,*\}^L$ space – the Header Space
- 2 – Define union, intersection on Header Spaces
- 3 – Abstract networking boxes (Cisco routers, Juniper Firewalls) as transfer functions on sets of headers
- 4– Compute packets that can reach across a path as composition of Transfer Functions of routers on path
- 5. Find all packets that reach between every pair of nodes and check against reachability specification

All Network boxes modelled as a Transfer Function:

$$T : (h, p) \to \{(h_1, p_1), \ldots, (h_n, p_n)\}$$

Computing Reachability [HSA 12]

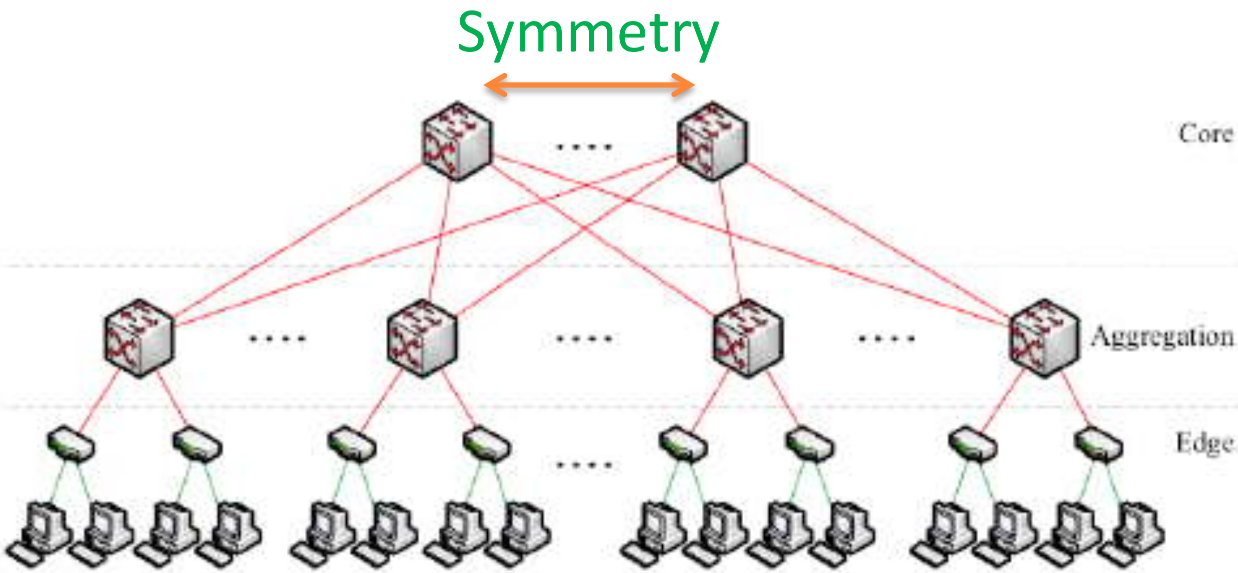


COMPLEXITY DEPENDS ON HEADERS, PATHS, NUMBER OF RULES

Unfortunately, in practice . . .

- Header space equivalencing: 1 query in < 1 sec. Major improvement over standard verification tools like SAT solvers and model checkers
- But our data centers: 100,000 hosts, 1 million rules, 1000s of routers, 100 bits of header
- So N^2 pairs takes 5 days to verify all specs.

Exploit Design Regularities to scale?



Common data center interconnect topology. Host to switch links are GigE and links between switches are 10 GigE.

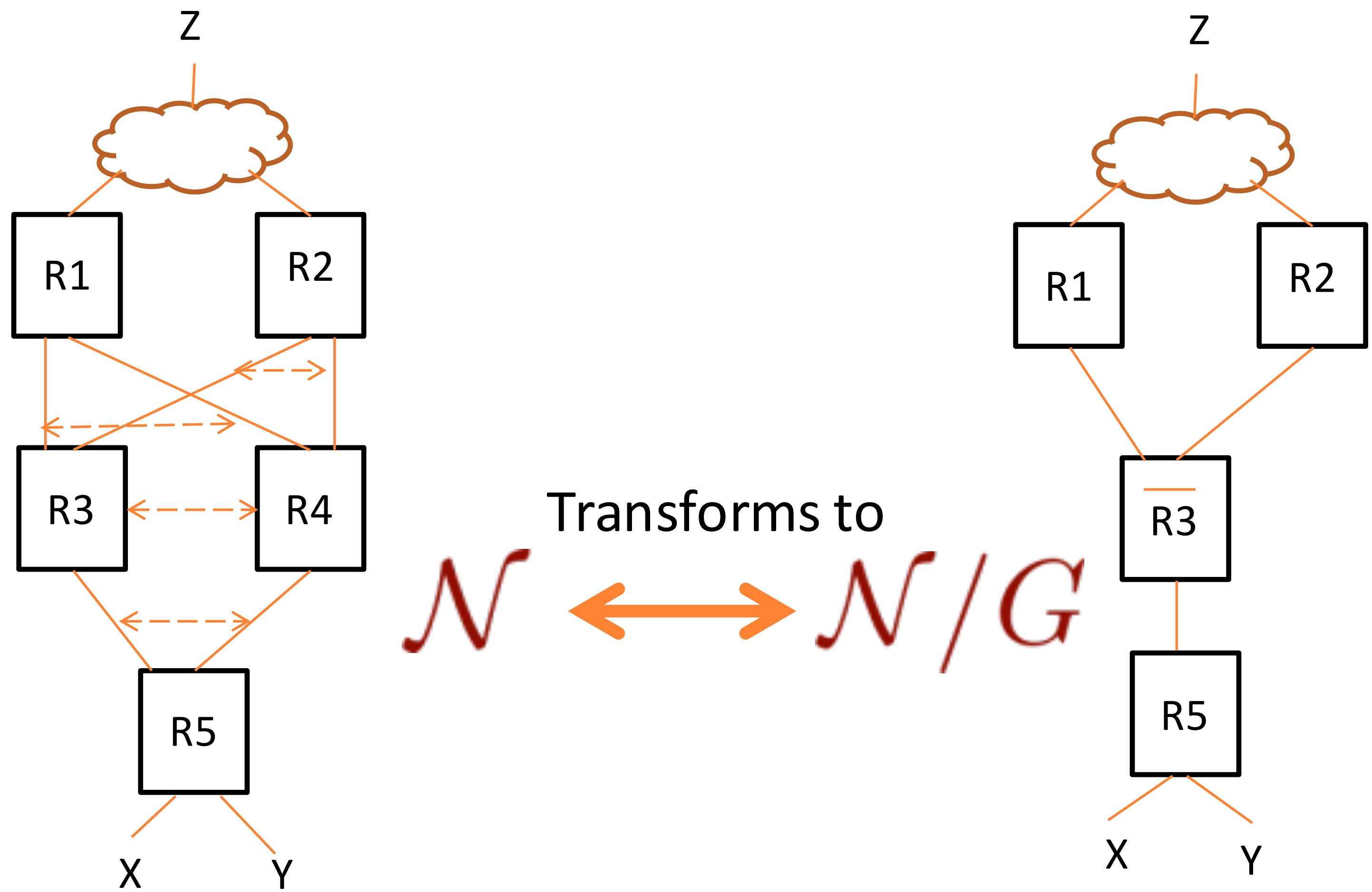
Can exploit regularities in rules and topology (not headers):

- Reduce fat tree to "*thin tree*"; verify reachability cheaply in thin tree.
- How can we make this idea precise?

Logical versus *physical* symmetry

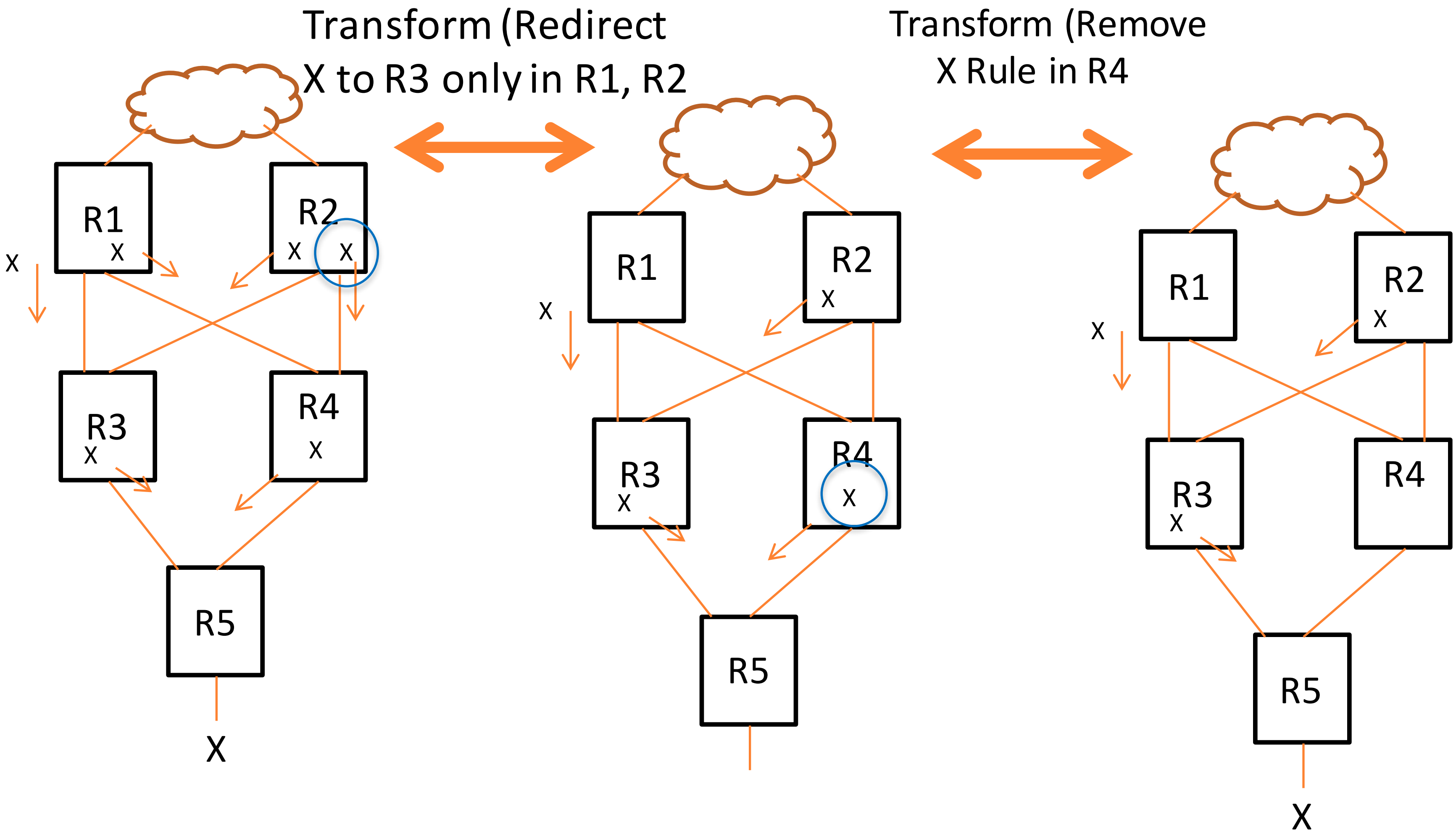
- (Emerson-Sistla): Symmetry on *state* space
  $h@p \to^*_{\mathcal{N}} h'@p' \iff \pi_{\mathcal{N}}(h@p) \to^*_{\mathcal{N}} \pi_{\mathcal{N}}(h'@p')$
- (Us): Factor: symmetries on *topology, headers*
  Define symmetry group *G* on *topology*
  Then $\mathcal{N} \sim \mathcal{N}/G$ (via bisimulation)
- Theorem: Any reachability formula *R* for $\mathcal{N}$ holds iff *R'* holds for quotient network $\mathcal{N}/G$

Topological Group Symmetry



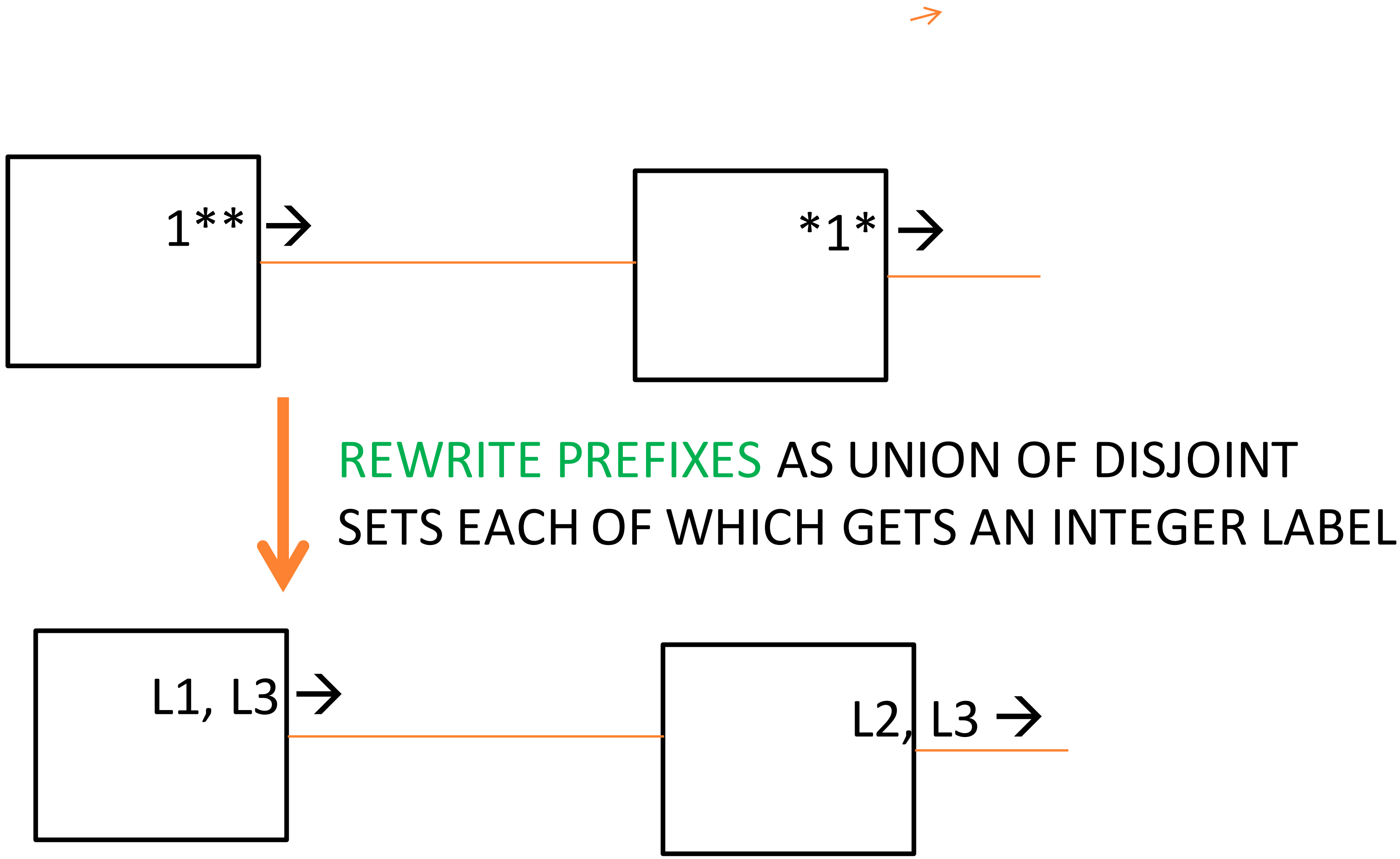
REQUIRES *PERFECTLY* SYMMETRICAL RULES AT R3 & R4.
IN PRACTICE, A FEW RULES ARE DIFFERENT.

Near-symmetry → rule (not box) surgery

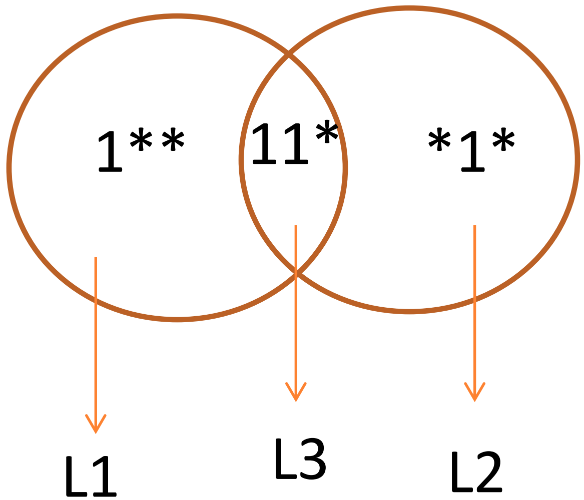


Instead of removing boxes, "squeeze" out redundant rules iteratively by redirection and removal. How to automate?

Step 1: Compute header equivalence classes (Yang-Lam 2013)

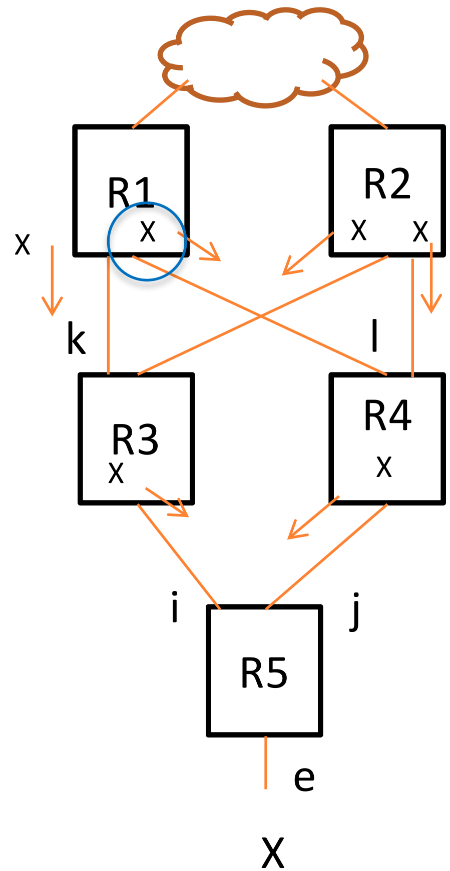


Computing labels in linear time



Efficiently compute labels using a graph on sets that we call a ddNF, takes linear time on our datasets

Step 2: compute interface equivalence classes via Union-Find



$k \equiv_X l$

$i \equiv_X j$

$e \equiv_X e$

For each header equivalence class, find all equivalent interfaces

Exhaustive verification solutions

- Header equivalence classes: 2^{100} → 4000
- Rule surgery: 820,000 rules → 10K rules
- Rule surgery time → few seconds
- Verify all pairs: 131→ 2 hours
- 65 x improvement with simplest hacks. With 32-core machine & other surgeries → 1 minute goal

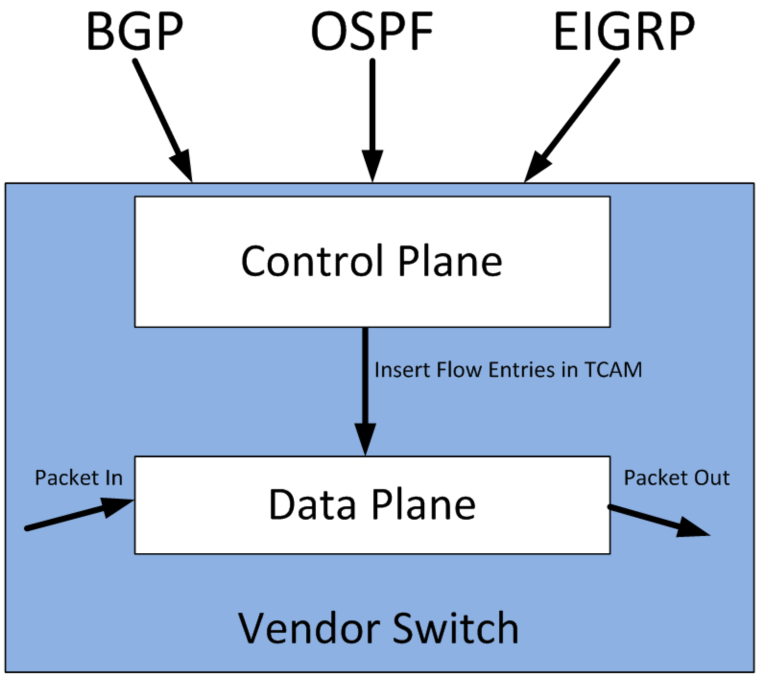
→ Can do periodic rapid checking of network invariants. Simple version in operational practice

Ongoing work

Limitation	Research Project
Booleans only (Reachability)	*Quantitative* Verification (QNA)
No *incremental* way to compute header equivalence classes	New data structure (ddNFs) Venn diagram intersection
Data plane only: no verification of routing computation	*Control Space Analysis (*second part of talk)
Correctness faults only (no *performance* faults)	Data-plane tester ATPG (aspects in Microsoft clouds)
Stateless Forwarding Only	Work at Berkeley, CMU

Progress in Data Plane Verification

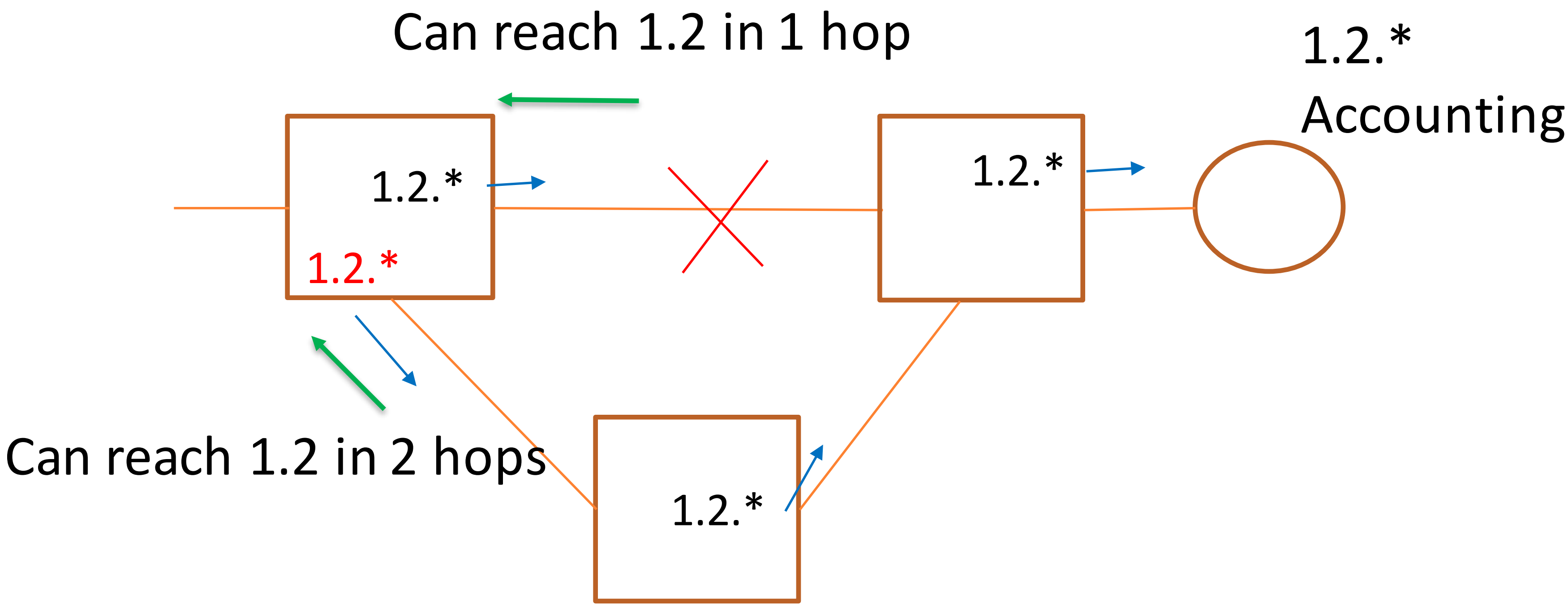
- FlowChecker (UNC 2009): reduces network verification to *model checking*. Not scalable
- Anteater (UIUC 2011): reduces to *SAT solving*. One counterexample only
- Veriflow (UIUC 2012): Finds all headers using header equivalence classes
- HSA(Stanford 2012): Header Space Analysis
- Atomic Predicates(UT 2013): Formalizes Header ECs and provides algorithm to precompute them
- NoD(MSR 2014): Reduces to *Datalog*, new fused operator
- Surgeries (MSR 2016): Exploits symmetries to scale



Topic 2: Control Plane Verification

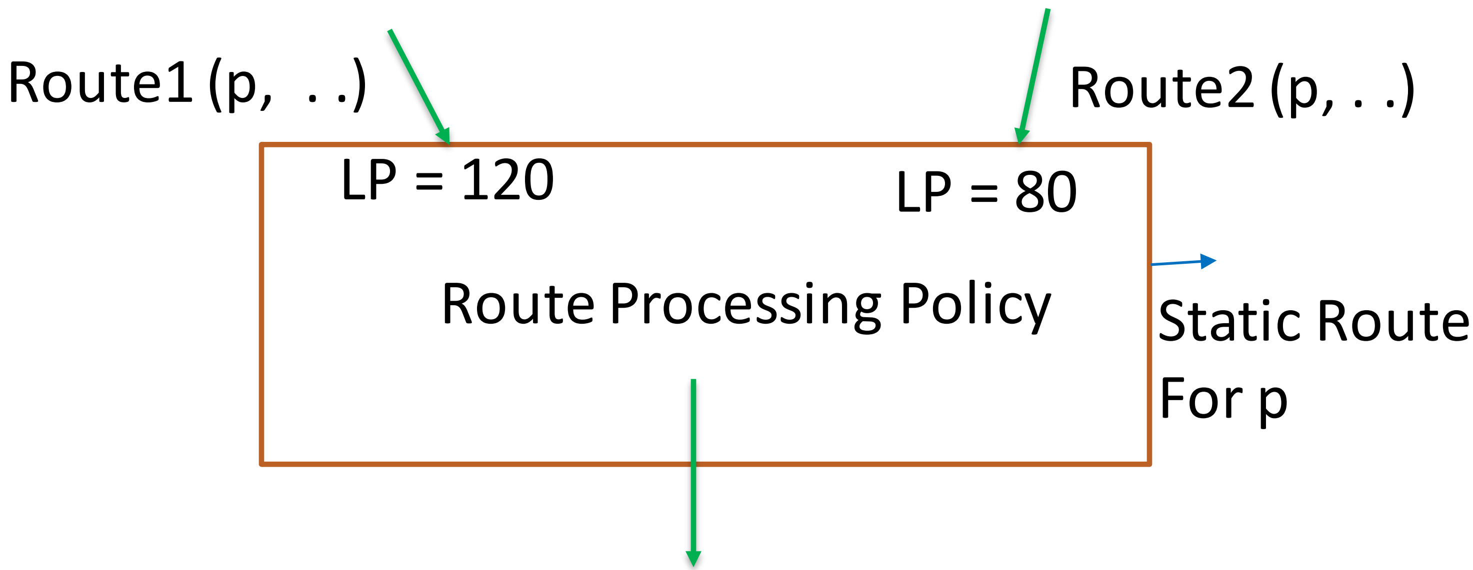
Fayaz et al, OSDI 2016

But there is also a Control Plane



- Data Plane (DP): Collection of forwarding tables and logic that forward data packets, aka Forwarding
- Control Plane (CP): Program that takes failed links, load into account to build data plane, aka Routing

BGP Routing: Beyond shortest path



- Static Routes take precedence
- Then come local preferences at this router (higher wins)
- Then comes some form of path length
- And more . . .

Control versus Data Plane Verification

Program types:

- $ControlPlane$: $(Config{\times}Env) \rightarrow \ ForwardTable$
- $DataPlane$: $(ForwardTable{\times}Header) \rightarrow FwdResult$

Data Plane verification for fixed Forwarding Table f

$$\forall h: \ \ Header: \Phi(h, DataPlane\,(f, h))$$

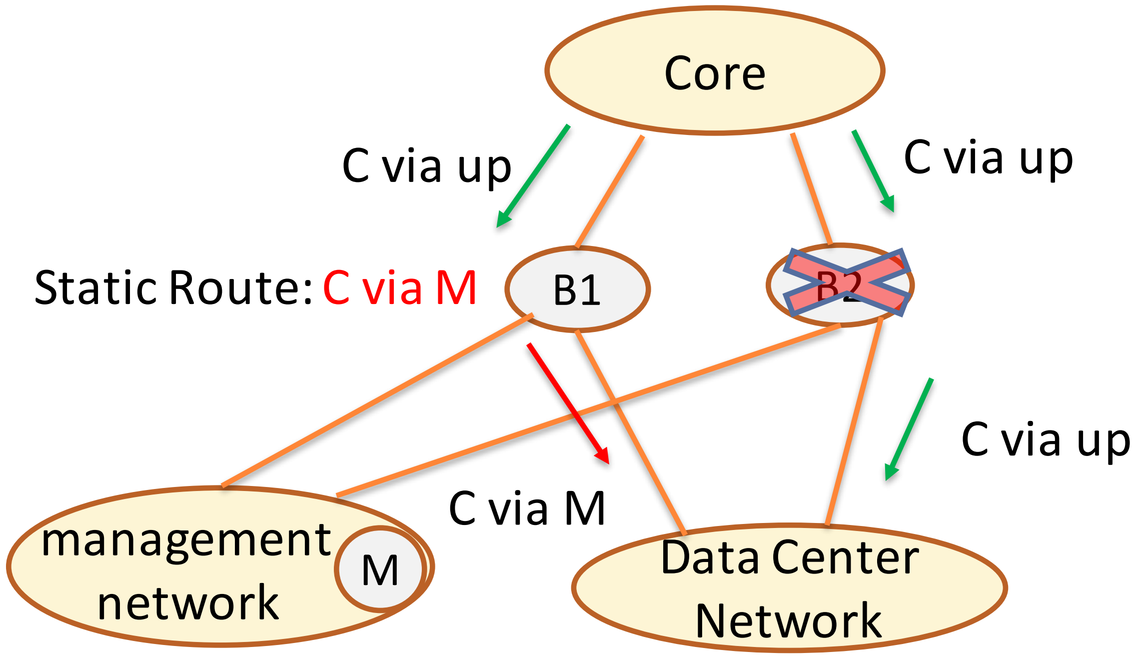
Control plane verification for configuration c

$$\forall e, h: \ \Phi(h, DataPlane(ControlPlane(c, e), h)))$$

Or

$$\forall e: \mathrm{P}((ControlPlane(c, e))$$

Errors manifest as *Latent* Bugs



Buggy static route causes B1 to propagate wrong route to C. Works fine till . . .

Specification: $\forall e$ (routing messages received)

PropagatedRoute (B1, e) = PropagatedRoute (B2, e)

Symbolic Execution of Route Propagation

- Model BGP Code in Router using C
  - Can now do symbolic execution
  - Many tools, we used Klee for a prototype
- Can encode symbolic route packets:

Prefix	Local Preference	AS Path	. . . .

- Then propagate routes as in Header Space.
- Encoding routers in Klee, we found . . .

Using Klee to uncover latent bug

Create symbolic attribute

scope a field for faster verification

```
unsigned int sym_route_ad;
klee_make_symbolic(&sym_route_ad,sizeof(sym_route_ad),"sym_route_ad");
klee_assume(sym_route_ad >= 0);
klee_assume(sym_route_ad <= 5);
memcpy(&sym_route.ad,&sym_route_ad,sizeof(sym_route.ad));
```

```
struct Route A_output;
struct Route E_output;

tf_helper(0,sym_route);
tf_helper(1,sym_route);

A_output = RIBout[0][0];
E_output = RIBout[1][0];
```

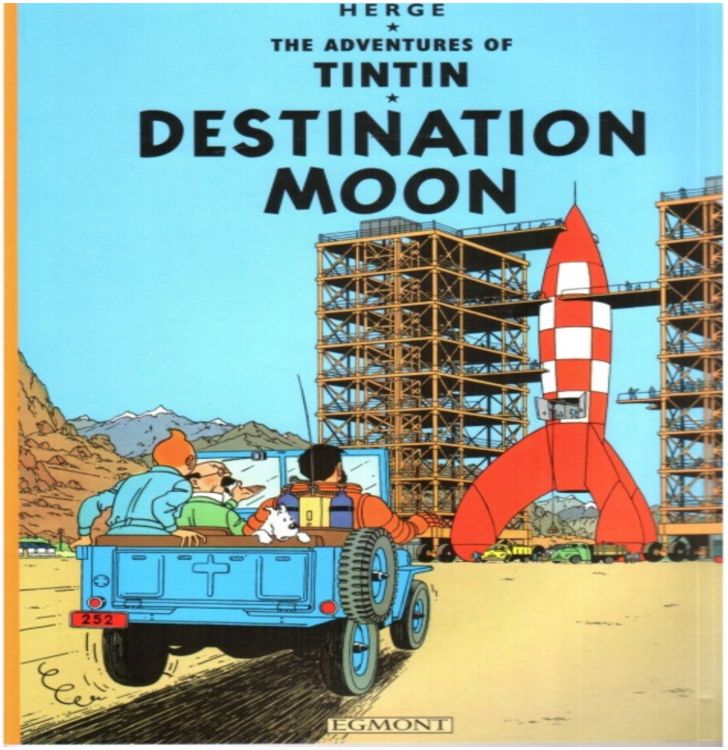
KLEE assertion

```
klee_assert(A_output.IP == E_output.IP);
```

KLEE finds counterexample: `sym_route.prefix = C`

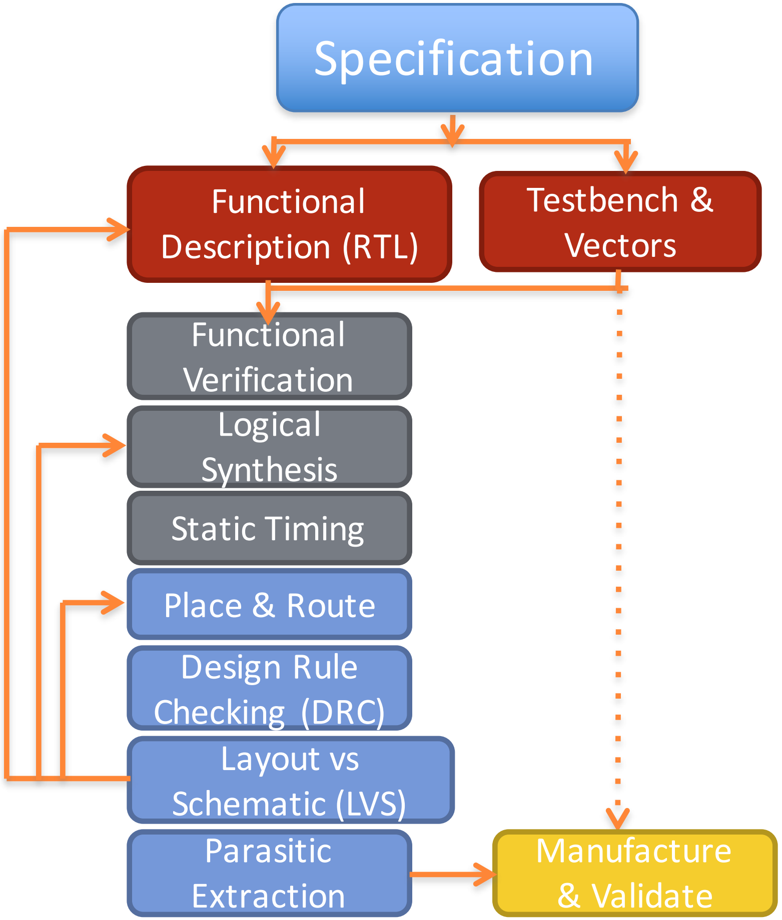
Progress in Control Plane Validation

- RCC (MIT 2005): static checker for common BGP faults (mostly syntactical, cannot catch deeper bugs)
- Batfish (MSR, UCLA 2015): computes data plane for 1 BGP environment (cannot reason across environments)
- ARC (MSR, Wisconsin 2016): For a rich class of BGP operators, can reason across *all* failures
- ERA (CMU, MSR, UCLA 2016): Reasons across a subset of maximal environments to find bugs
- Bagpipe (Washington 2016): Reasons about BGP only and for a sunset of topologies
- NetKat (Princeton, Cornell 2014): *Data* plane synthesis
- Propane : (Princeton, MSR, 2016): *Control* plane synthesis

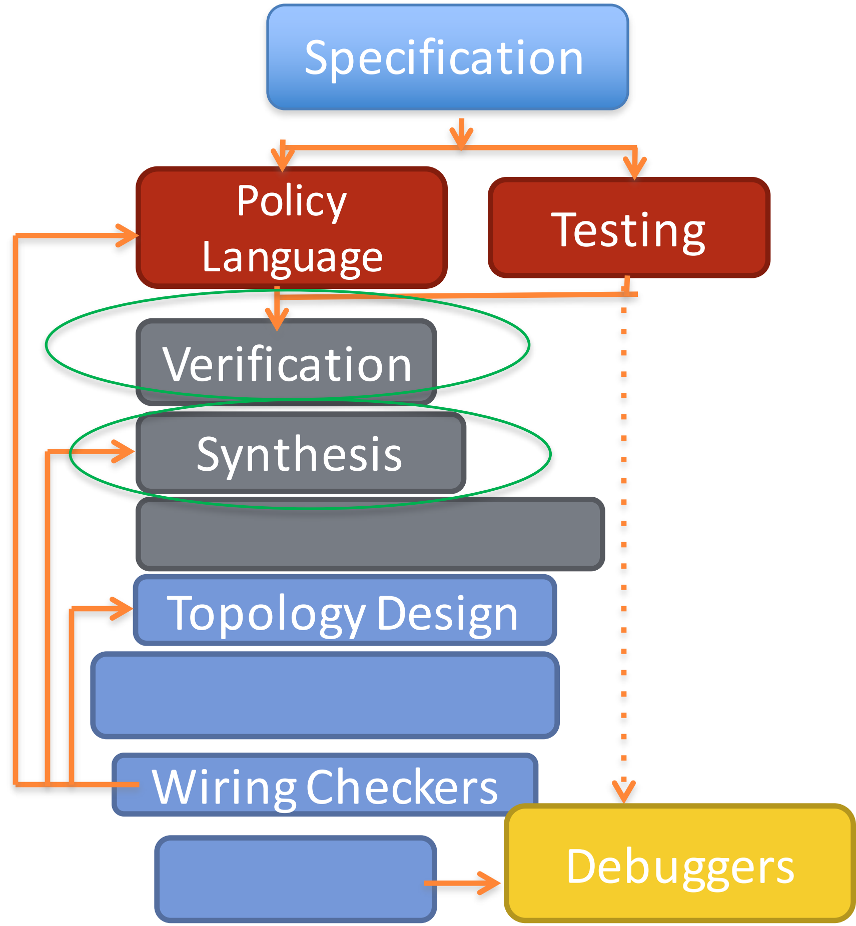


NETWORK DESIGN AUTOMATION?

Digital Hardware Design as Inspiration



Electronic Design Automation
(McKeown SIGCOMM 2012)



Network Design Automation
(NDA)?

NDA: Broader Research Agenda

- Bottom up *(analysis):*
  - Run time support (automatic test packets?)
  - Debuggers (how to "step" through network?)
  - Specification Mining (infer reachability specs?)
- Top Down *(synthesis):*
  - Expressivity (load balancing, security policies?)
  - Scalable specifications (network types?)
  - New Optimization Problems (stochastic?)

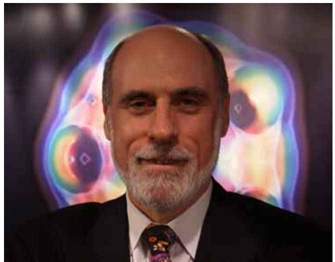
Yawn. We have seen it all years ago!

Verification Exemplar	Network Verification Idea
Ternary Simulation, Symbolic Execution [Dill 01]	Header Space Analysis [Kazemian 2013]
Certified Development of an OS Sel4 [Klein 09]	Certified Development of an SDN Controller [Guha 13]
Specification Mining [Bodek 02]	Mining for Enterprise Policy [Benson 09]
Exploit Symmetry in Model Checking [Sistla 09]	Exploit Symmetry in Data Centers [Plotkin 16]

Yes, but scale by exploiting domain

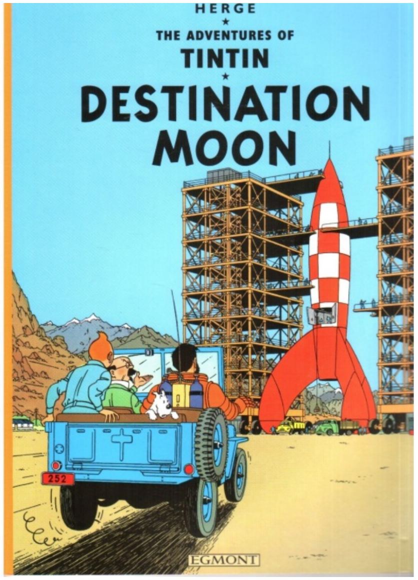
Technique	Structure exploited
Header Space Analysis	Limited negation, no loops, small equivalence classes
Exploiting Symmetry	Symmetries in physical topology
ATPG (Automatic Test Packet Generation)	Network graph limits size of state space compared to KLEE
Netplumber (incremental network verification)	Simple structure of rule dependencies

Requires Interdisciplinary work between formal methods and networking Researchers

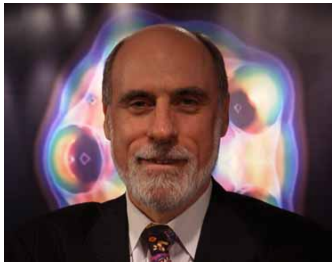


Conclusion

- Inflection Point: Rise of services, SDNs
- Intellectual Opportunity: New techniques
- Working chips with billion transistors. Large networks next?



Thanks



- (MSR): N. Bjorner, N. Lopes, R. Mahajan, G. Plotkin,
- (CMU): S. Fayaz, V. Sekar
- (Stanford): P. Kazemian, N. McKeown
- (UCLA): A. Fogel, T. Millstein