With a little help from Structured Friends

Aarti Gupta

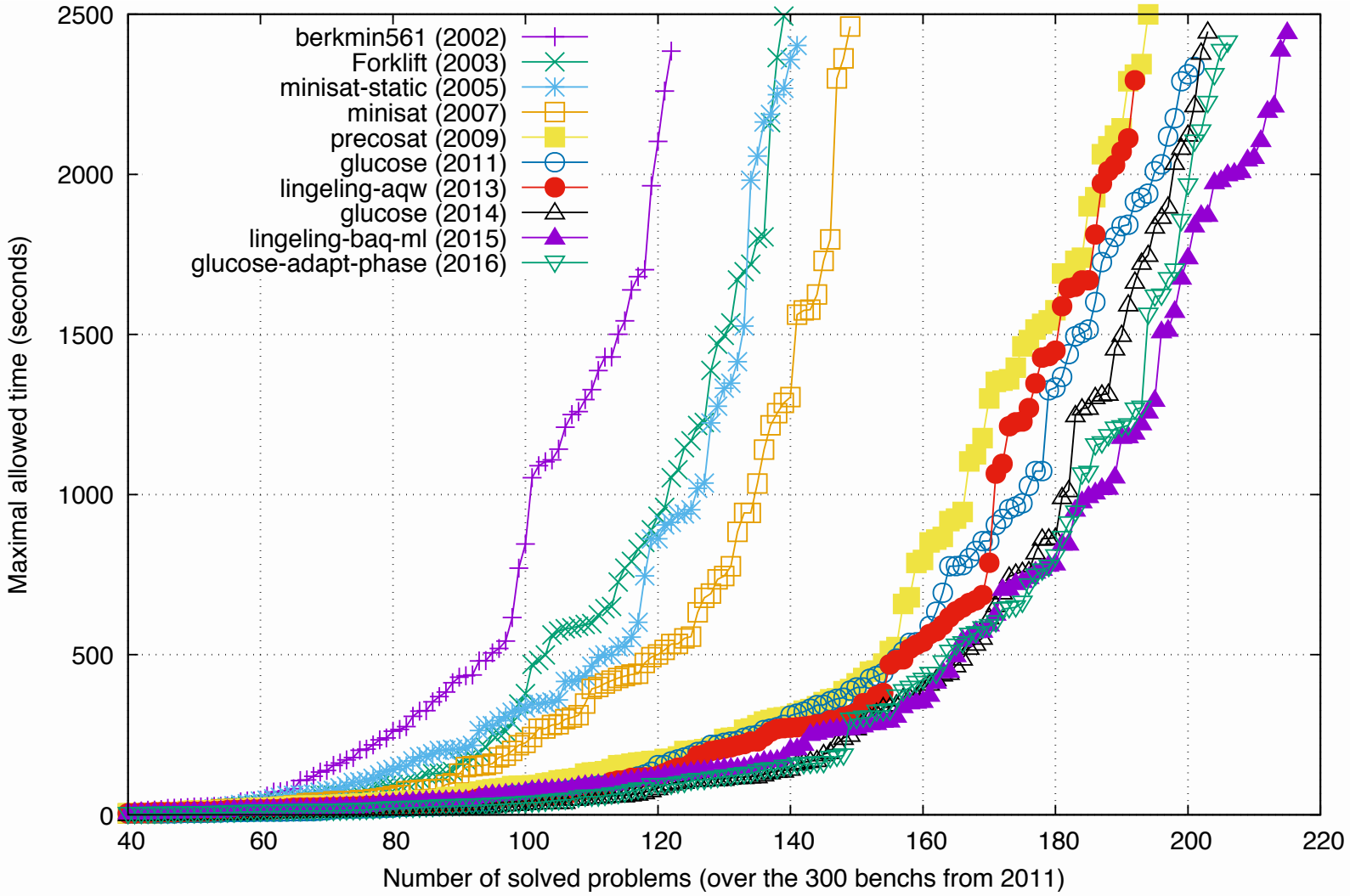
Princeton University

The 23rd International Conference on Theory and Applications of Satisfiability Testing

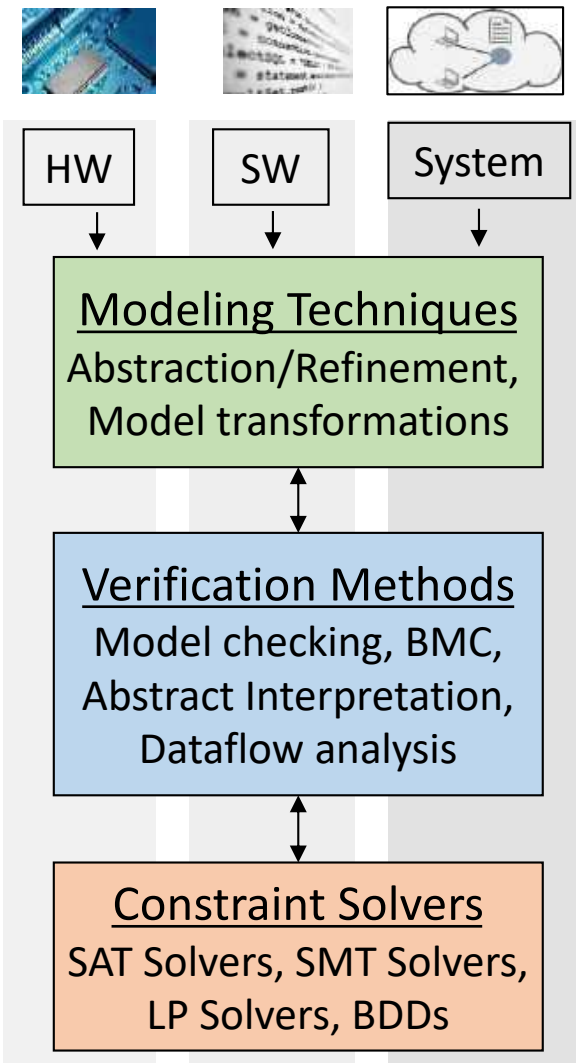
Credit: Joao Marques-Silva (SAT Workshop, Banff, 2018)

SAT solver evolution

[Source: Simon 2015]



SAT/SMT-based Reasoning



Layered Framework

- Modeling techniques
- Verification and analysis methods
- Constraint solvers

Application domains

- Software, multi-threaded programs
- Hardware/embedded/hybrid systems
- *SoCs: hardware-firmware, accelerators*
- *Networks*

Automated reasoning

- *Verification, Synthesis, Security*

this talk

Many Gaps

Model Checking		Satisfiability solvers
2QBF		Propositional SAT
PSPACE complexity		NP complexity
Transition systems		FOL/propositional formulas
Invariants		Interpolants

Bridges over the gaps

- Iterative unrolling of transition systems, e.g., BMC [Biere *et al.* 99]
- Symbolic model checking, e.g., interpolant-based [McMillan 03]
- Verification Condition (VC) generation, e.g., Dafny [Leino 10]
- Iterative invariant generation, e.g., IC3 [Bradley 11]
- …

Theme today

Despite tremendous advancements in SAT/SMT solvers, there are benefits to be gained by leveraging the *structure* of the underlying domain models in verification/synthesis.

Approach 1: use a suitable encoding that matches well with the structure of the domain model

Approach 2: use domain insights to guide search for invariants in verification

- Three recent projects that illustrate these techniques

Application 1
Network Verification

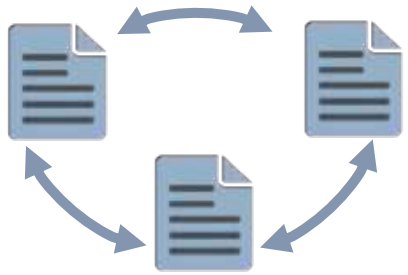
Ryan Beckett

Ratul Mahajan

David Walker

Network Design

Control Plane



Protocol + Configuration

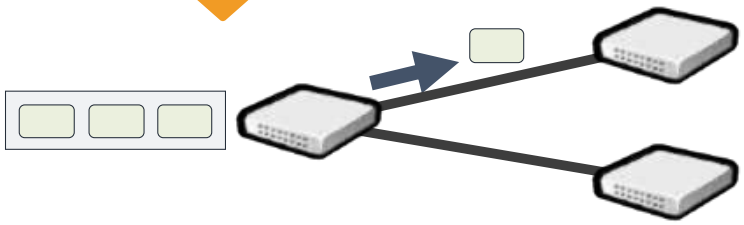
Control Plane Protocols
RIP, OSPF, BGP, …

Data Plane

7.2.0.*	port 1
9.2.*.*	port 2

7.2.0.*	port 5
9.2.*.*	port 3

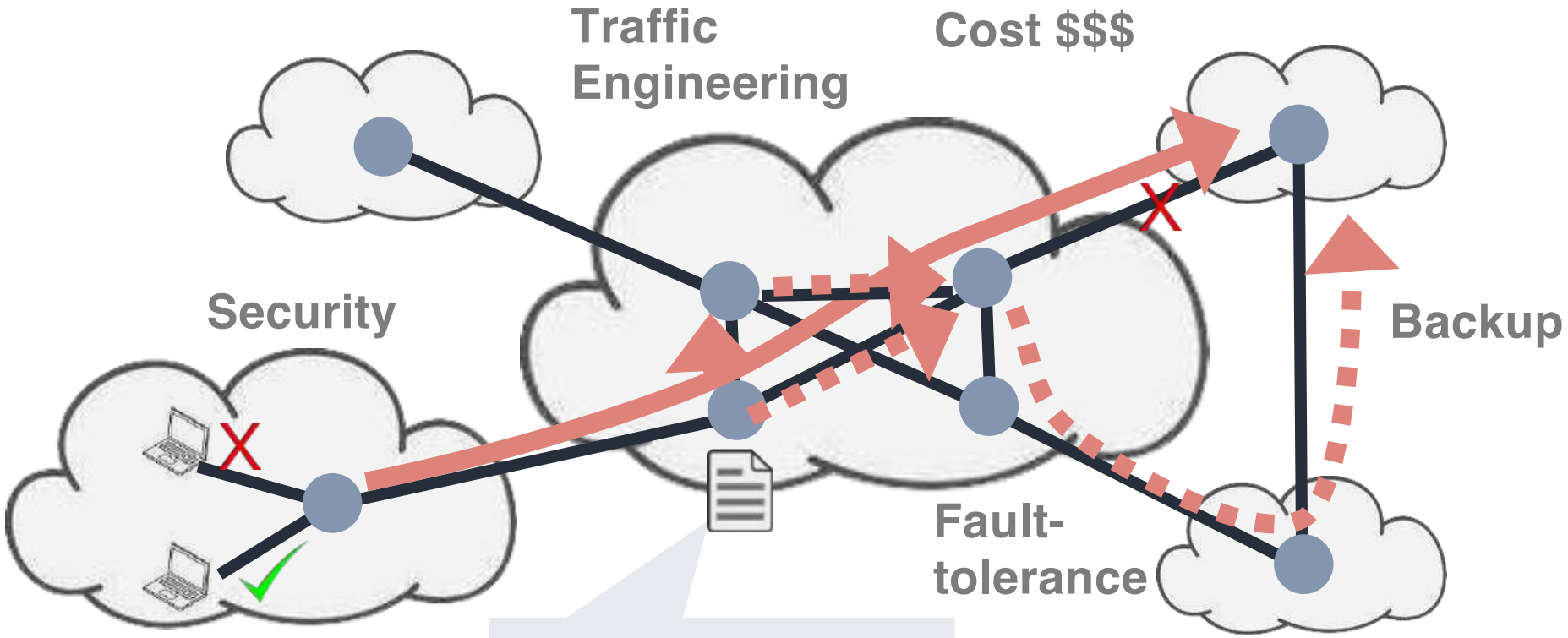
Forwarding Tables



Runtime Behavior

Network Control Plane

Primary goal is to get traffic from point A to point B but …



Configuration
Distributed program
Vendor-specific
Assembly-language
Thousands of lines

Misconfiguration is a BIG problem

Some solutions

Data Plane Verification

Anteater	[Mai 2011]
HSA	[Kazemian 2012]
Veriflow	[Kurshid 2013]
NoD	[Lopes 2015]
Symmetries	[Plotkin 2016]
…	

Some solutions

Data Plane Verification

- Anteater [Mai 2011]
- HSA [Kazemian 2012]
- Veriflow [Kurshid 2013]
- NoD [Lopes 2015]
- Symmetries [Plotkin 2016]
- …

our work

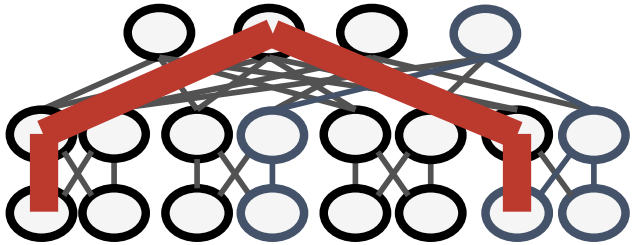
Control Plane Simulation

- C-BGP [Quotin 2005]
- Batfish [Fogel 2015]
- …

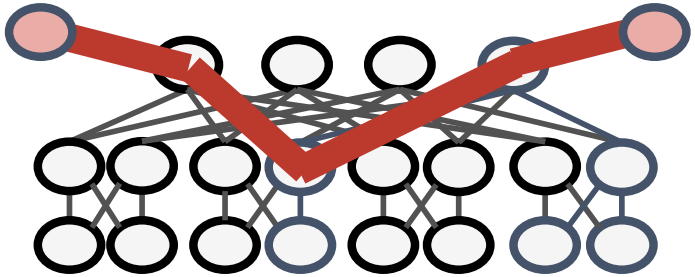
Control Plane Verification

- Bagpipe [Weitz 2016]
- ARC [Gember-Jacobsen 2016]
- ERA [Fayaz 2017]
- **Minesweeper [Beckett 2017]**
- …

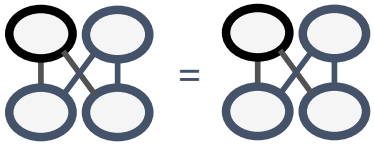
Network Properties



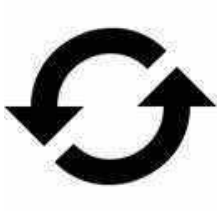
reachability



no transit



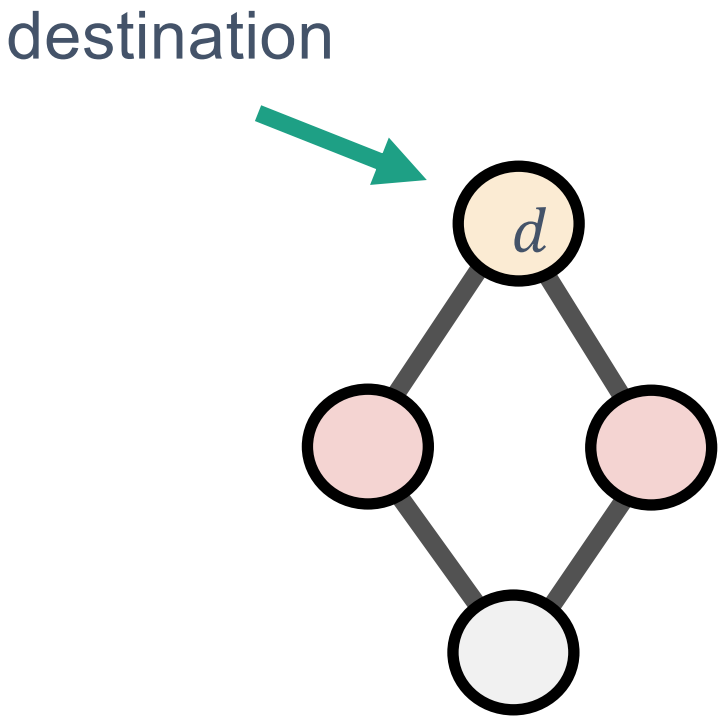
Router or subnet equivalence



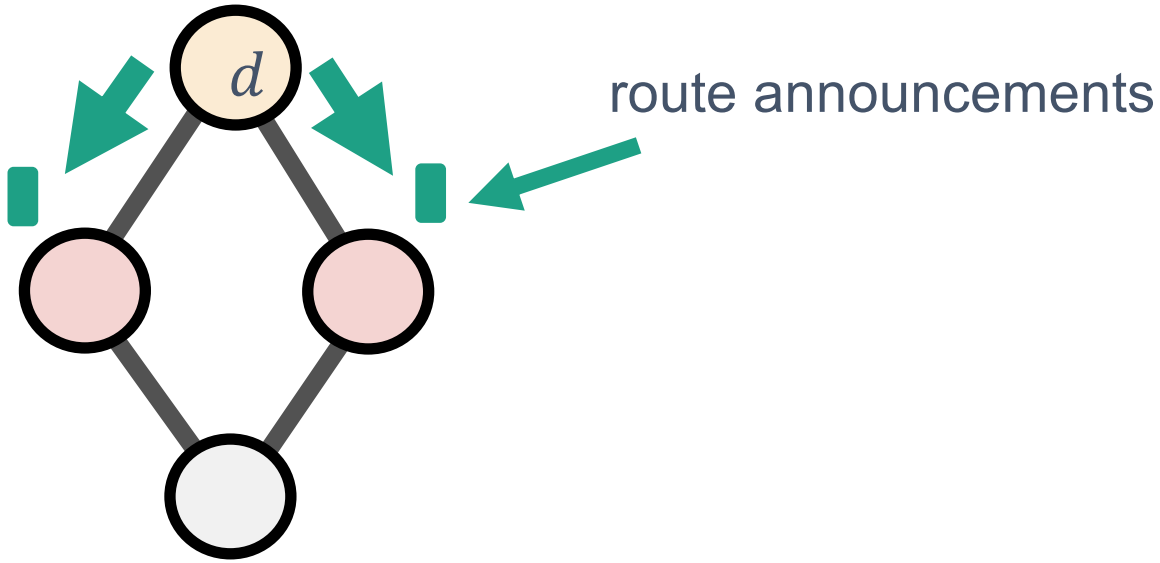
no loops

no black holes

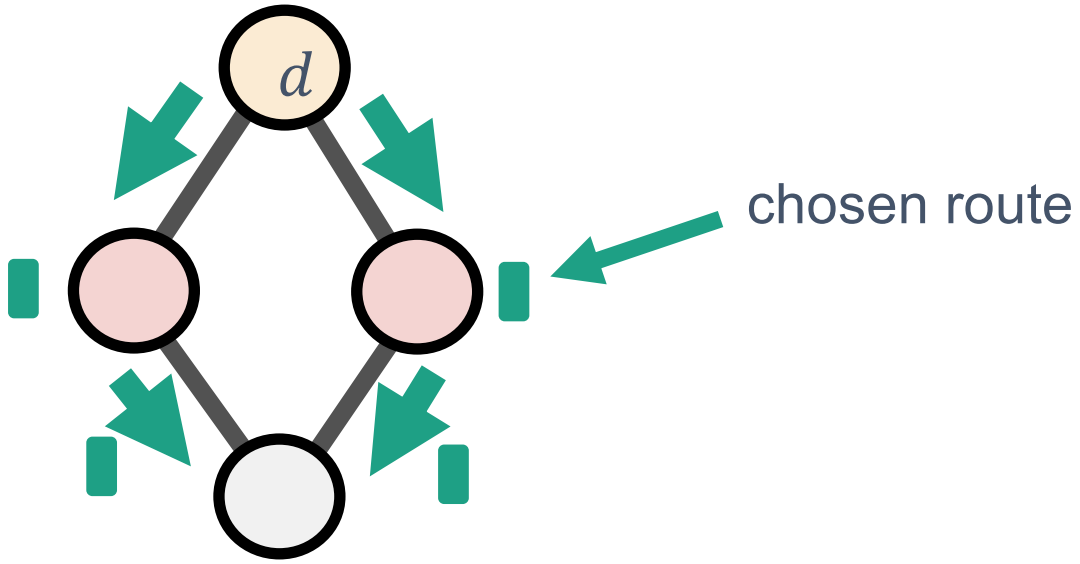
A Generic Routing Protocol



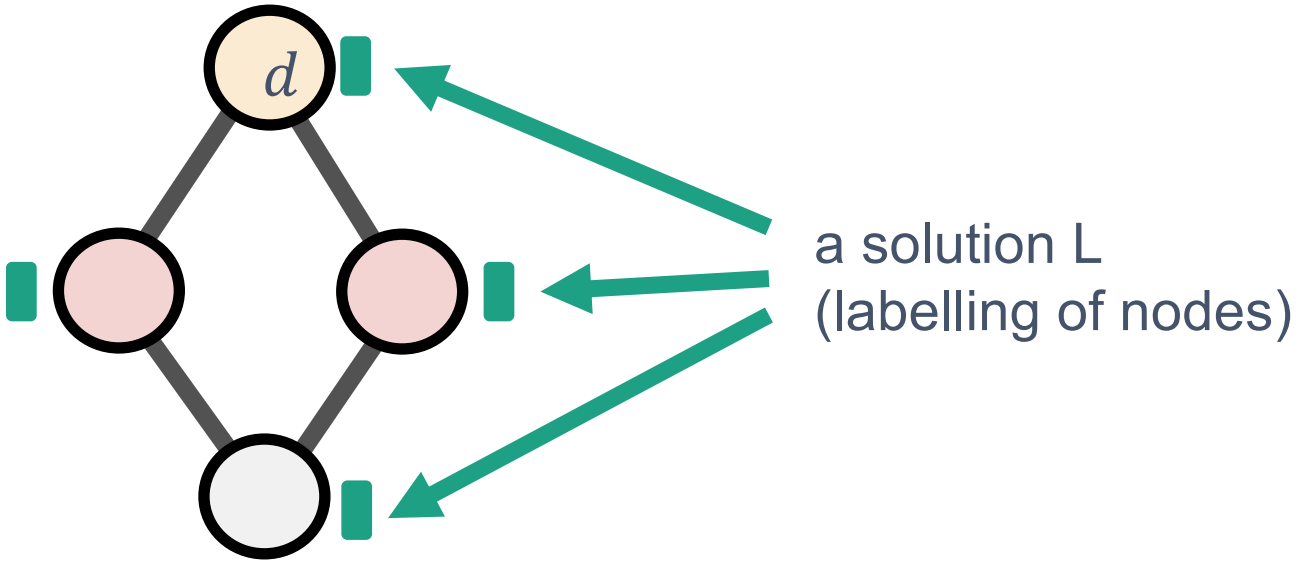
A Generic Routing Protocol



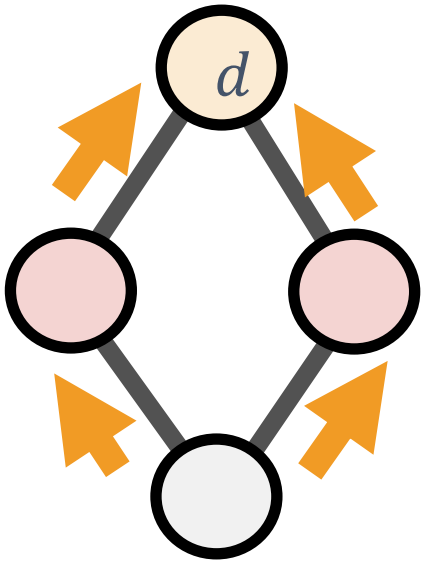
A Generic Routing Protocol



A Generic Routing Protocol

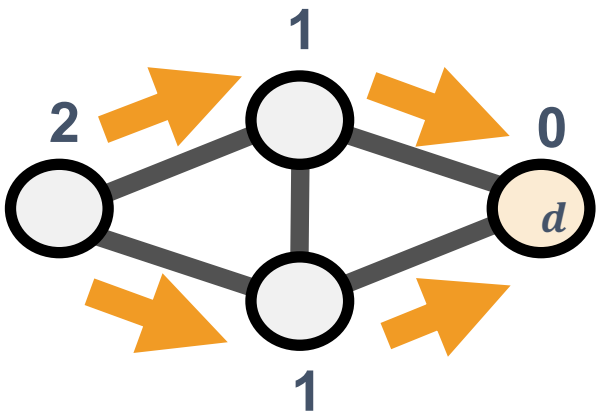


A Generic Routing Protocol



visual representation /
flow of traffic:

A Generic Routing Protocol



Idealized RIP:
A simple routing protocol

- The origin creates an ***initial announcement*** stating it has a path to destination *d*
- Other nodes that receive the announcement pass it on to their neighbors, ***possibly modifying it***
- When nodes receive multiple announcements, they ***choose a best one***
- Eventually (hopefully), the system converges to a ***stable solution***: all nodes are happy

[Griffin and Sobrinho 05]

Stable Paths Problem (SPP)

[Griffin *et al.* 2002]

Imperative, Stateful Program

Logic Model

Choices (P, u) = …
Best (u) = …

Each node is locally stable

- Primarily used to reason about **network convergence**
- Not applied for verifying ***network configurations of protocols***

Minesweeper: Domain Insights

Network protocols are designed to generate stable paths
i.e., routers exchange messages to make best choice, which stays stable

- **Our Idea**

Capture network control plane behavior in terms of logical constraints, such that *satisfying solutions are stable paths* in the network

- **Analogy to Program Verification**

Program: Satisfying solution represents a path in the program graph
Network: Satisfying solution represents *stable* paths in the network graph

But: arbitrary graphs in network topology, a single solution corresponds to many paths (possibly a stable routing tree)

Minesweeper: Key Choices

Choice 1: Model routing *graphs*, not paths at a time

- Too many paths, but all paths share the same graph
- In the data plane, reasoning is done per-packet because there is no interference between packets along different paths;
  but routing messages along different paths *interact* in the control plane – modeling this interaction can be expensive!

Choice 2: Don't compute states due to exchange of routing messages, but perform *search* on final stable states

- Familiar lesson from symbolic model checking vs. bounded model checking
- Solve a *search* problem over state space, use modern SAT/SMT solvers
- Often scales better than *computing* sets of states iteratively

Minesweeper Approach

- **SRP** (Stable Routing Paths) – a general, logical control plane model
- Framed in terms of **local** route processing constraints at a node

- Applies translation to **SMT-based logic** for **verification**
- Heavily **optimized** to make the resulting tool practical

Ryan Beckett, Aarti Gupta, Ratul Mahajan, David Walker:
A General Approach to Network Configuration Verification. SIGCOMM 2017: 155-168

Minesweeper: SRP Model

Topology: $G = (V, E, d)$ where $d: dest \in V$

Attributes: $A_\perp = A \cup \{\perp\}$

Preference relation: $\prec: A \times A$

Transfer function: $transfer: E \times A \rightarrow A_\perp$

Initial value: $a_d \in A_\perp$

protocol

configuration

- An SRP solution is a labeling: $L: V \rightarrow A_\perp$
- **A solution is an attribute assignment where each node is happy**

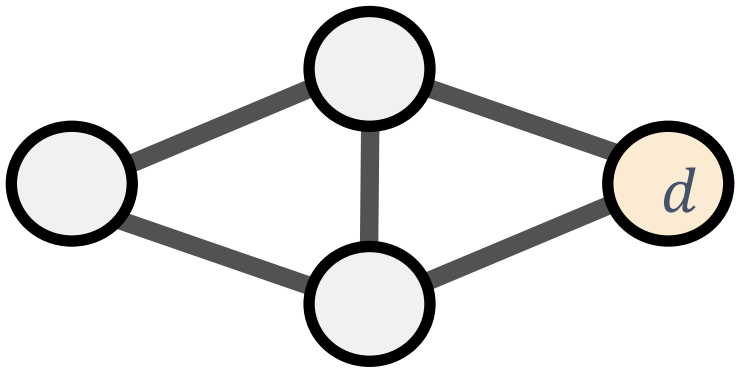
SRP Models for Popular Protocols

- Many network protocols are used in practice
  - RIP
  - OSPF
  - BGP (eBGP, iBGP)
  - Static Routing

- Minesweeper handles them all as SRPs
  - uniform model allows handling fancy features like route redistribution etc.

Example SRP

Routing Information Protocol (RIP)



Attributes:

$\mathrm{A} = \{0..15\}$

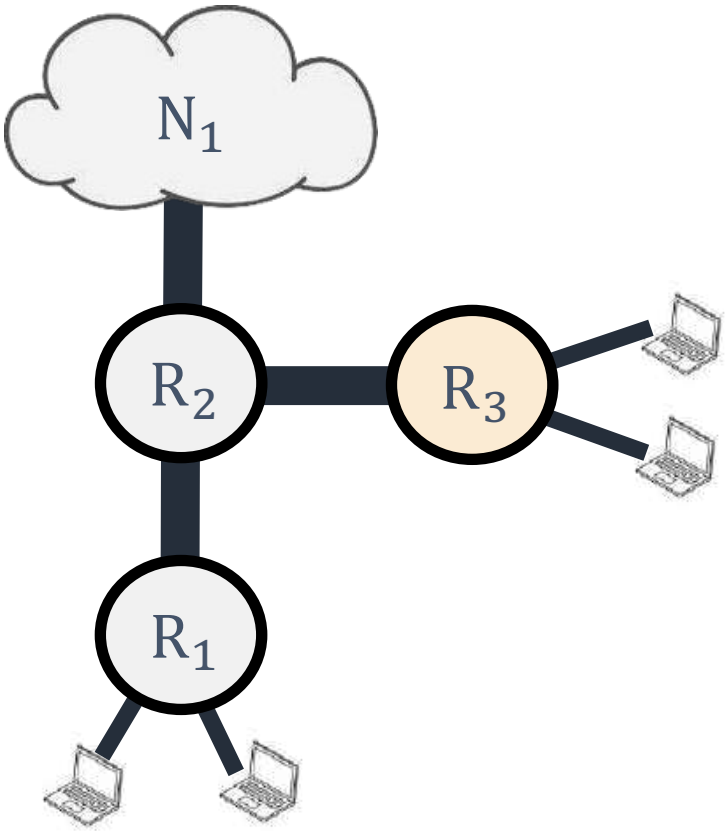
Preference relation:

$\mathrm{a} \prec \mathrm{b} \Leftrightarrow \mathrm{a} < \mathrm{b}$

Transfer function:

$$\mathrm{transfer}(\mathrm{e},\mathrm{a}) = \begin{cases} \perp & \text{If a=15} \\ \mathrm{a}+1 & \text{otherwise} \end{cases}$$

SMT Encoding for Verification



"Does P hold in the network?"

Network Encoding (SRP): N

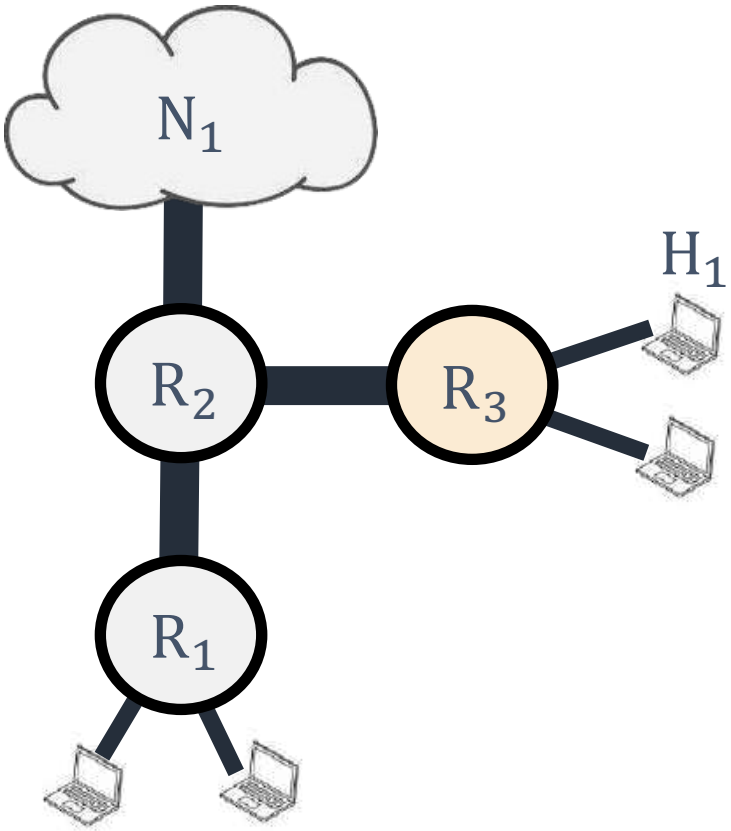
$\wedge$

Network Property (Negated): ¬P

Satisfiable: Property violation

Unsatisfiable: Property holds for all data planes

Example: Reachability Property



"Can router R1 reach host H1?"

$$\text{canReach}_{R3} \leftrightarrow \text{forwards}_{R3,H1}$$

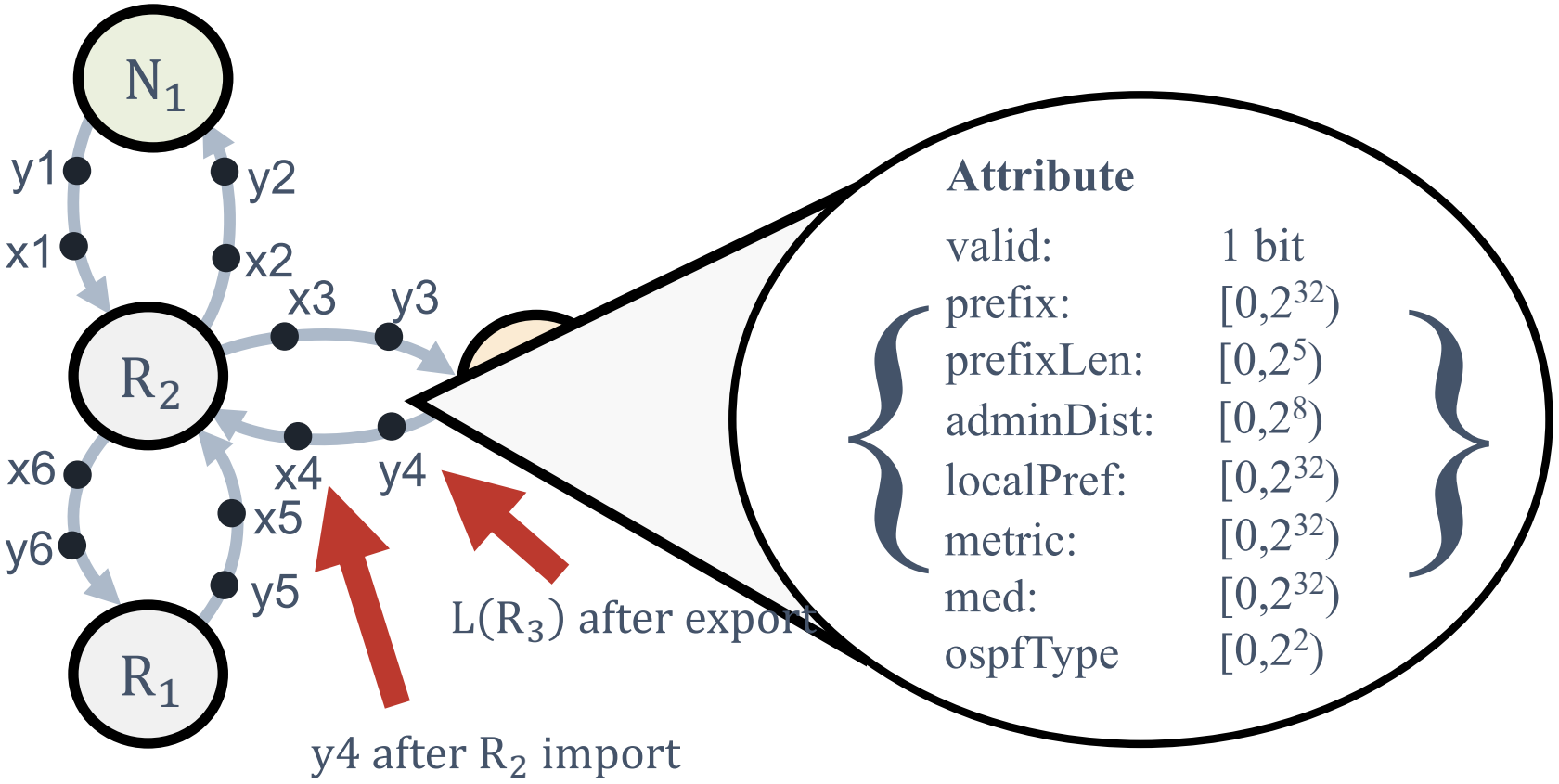
$$\text{canReach}_{R2} \leftrightarrow (\text{forwards}_{R2,R3} \wedge \text{canReach}_{R3}) \vee (\text{forwards}_{R2,R1} \wedge \text{canReach}_{R1}) \vee (\text{forwards}_{R2,N1} \wedge \text{canReach}_{N1})$$

$$\text{canReach}_{R1} \leftrightarrow \text{forwards}_{R1,R2} \wedge \text{canReach}_{R2}$$

Property: canReach_{R1}

Encoding Transfer Function

Attributes are like states
Transfer function encodes how state is updated along a link



SMT theories: bit vectors, LIA

Common Network Design Features

Features	Implemented
OSPF Intra-area	✓
OSPF Inter-area	✓
eBGP Local-pref	✓
eBGP Communities	✓
eBGP MEDs	✓
eBGP Path Prepending	✓
eBGP Aggregation	✓

Continued…	
iBGP	✓
Route Reflectors	✓
Static Routes	✓
Route Redistribution	✓
Multipath Routing	✓
Access Control Lists	✓
IPV6	✗

Properties Supported

Properties	Implemented
Reachability	✔
Bounded Path Length	✔
Equal Path Lengths	✔
Disjoint Paths	✔
Multipath Consistency	✔
Routing Loops	✔
Black Holes	✔
ECMP Load Balancing	✔
Router Equivalence	✔

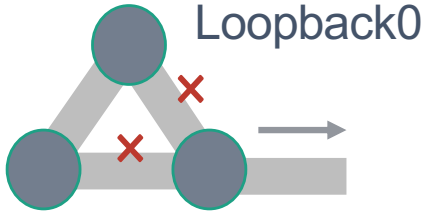
- Limitations:
  Does not support **convergence, quantitative/probabilistic properties**
  Checking **multiple destinations** is expensive

Minesweeper Evaluation

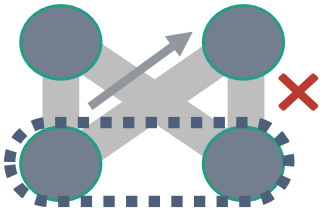
- Can Minesweeper find real bugs?
  - Ran on a collection of 152 legacy networks
  - 1—23K lines of configuration each
- How well does Minesweeper scale?
  - Tested on a collection of synthetic data center benchmarks
  - Compared verification time across a wide variety of properties

Evaluation: Bug Finding

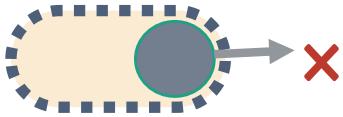
- **Management interface reachability**
  - Found 67 violations of the property
  - Example: BGP peer sends /32 with length 2



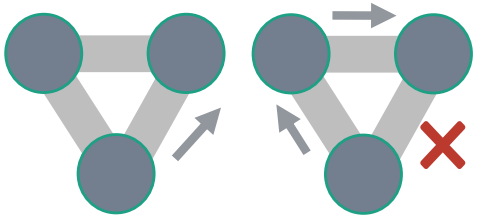
- **Local equivalence of routers**
  - Found 29 violations
  - Example: ACL has missing entry



- **Blackholes occur only at the network edge**
  - Found 24 violations of the property

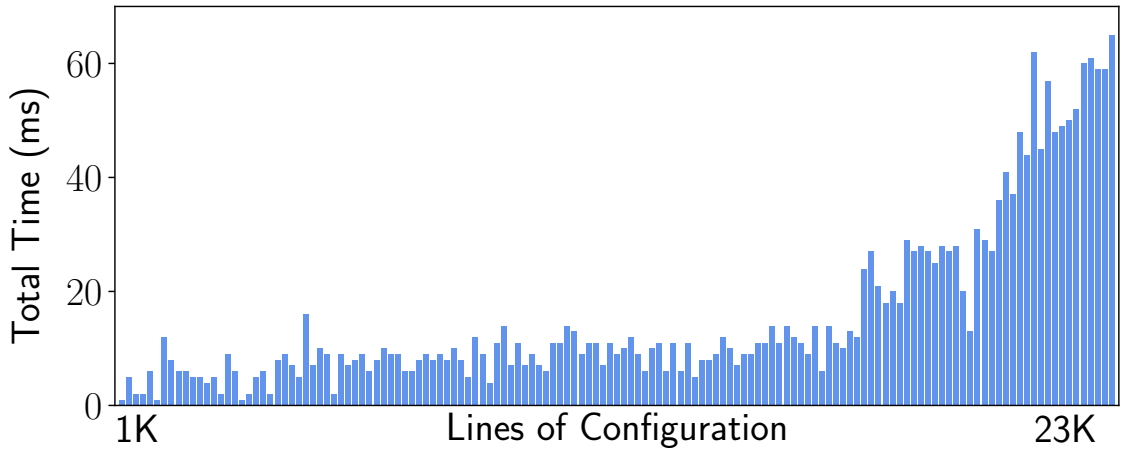


- **Reachability is the same after any 1 failure**
  - Found no violations of the property

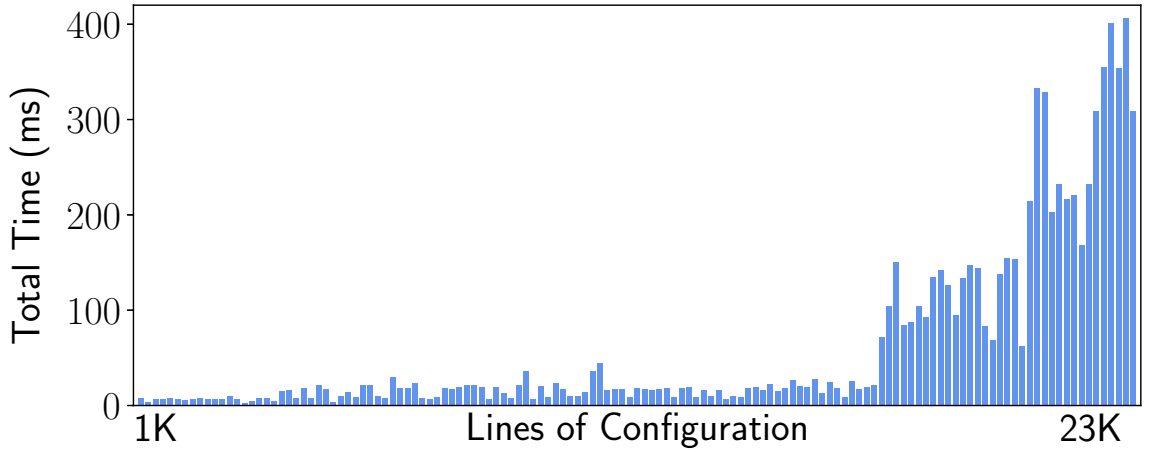


Evaluation: Scalability

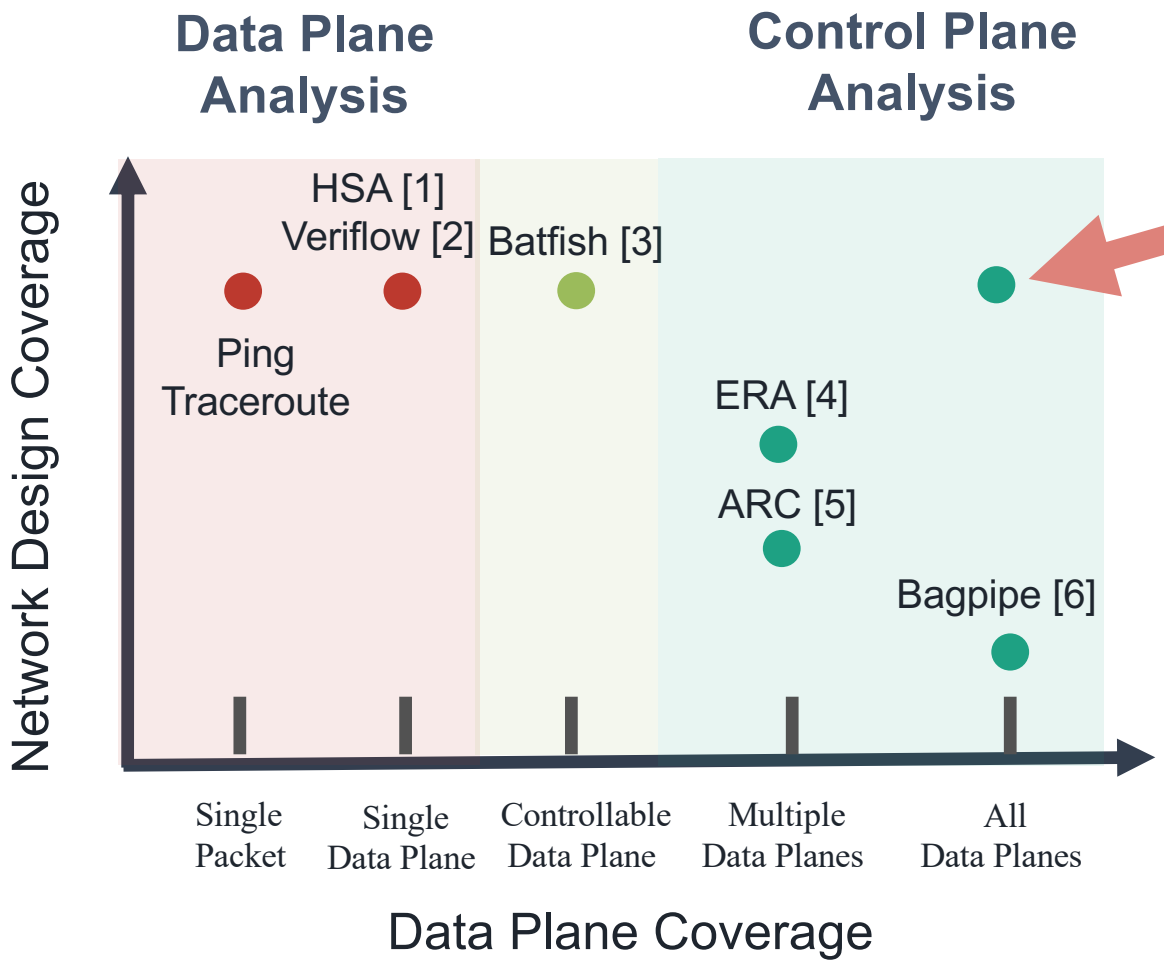
Management interface reachability **< 60 ms**



Local equivalence of routers (for all n comparisons) **< 400 ms**



Network Verification Landscape



[1] Kazemian et al, NSDI'12
[2] Khurshid et al, NSDI'13
[3] Fogel et al, NSDI'15
[4] Fayaz et al, OSDI'16
[5] Gember-Jacobson et al, SIGCOMM'16
[6] Weitz et al, OOPSLA'16

Minesweeper: Lessons (re-)Learned

- Build the logic-based core model first
  - modeling the network control plane stable behavior enabled direct use of SMT technology for verification

- Don't abstract too early
  - our model is rich in detail, captures many features (e.g., local preferences, route redistribution) considered important by network practitioners

- Build logic-based abstractions, compositional methods, CEGAR, <*your-favorite-method*> on top of the core
  - Symmetry-based abstractions [Beckett *et al.* SIGCOMM 2018]
  - Abstract Interpretation [Beckett *et al.* POPL 2020]
  - Failure analysis [Giannarakis *et al.* CAV 2019]
  - NV Programmable Platform [Giannarakis *et al.* PLDI 2020]

Future Opportunities

- Many challenges
- Scalability
  - Minesweeper does well for about ~600 node networks, but data centers have *tens of thousands* of nodes
  - ➢ Faster solvers
  - ➢ Graph-based theory solvers
  - ➢ Verification-based machinery (CEGAR, CHC solvers)
- Quantitative/probabilistic properties
  - Some existing efforts (e.g., Probabilistic NetKat, ApproxFlow)
  - ➢ Model counting techniques
  - ➢ Probabilistic verification
- Automated synthesis with verification
  - ➢ Optimization modulo theories
  - ➢ Leverage machine learning + deductive techniques