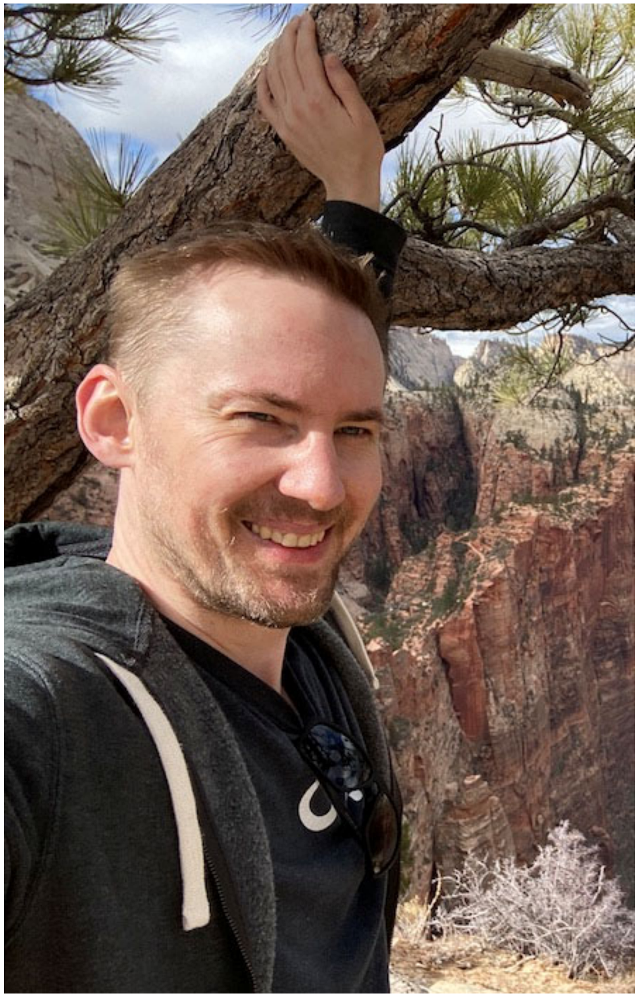


NV: A Framework for Modeling and Verifying Network Configurations

LangSec 2020

David Walker
Princeton University

Collaborators

Nick Giannarakis

Devon Loehr

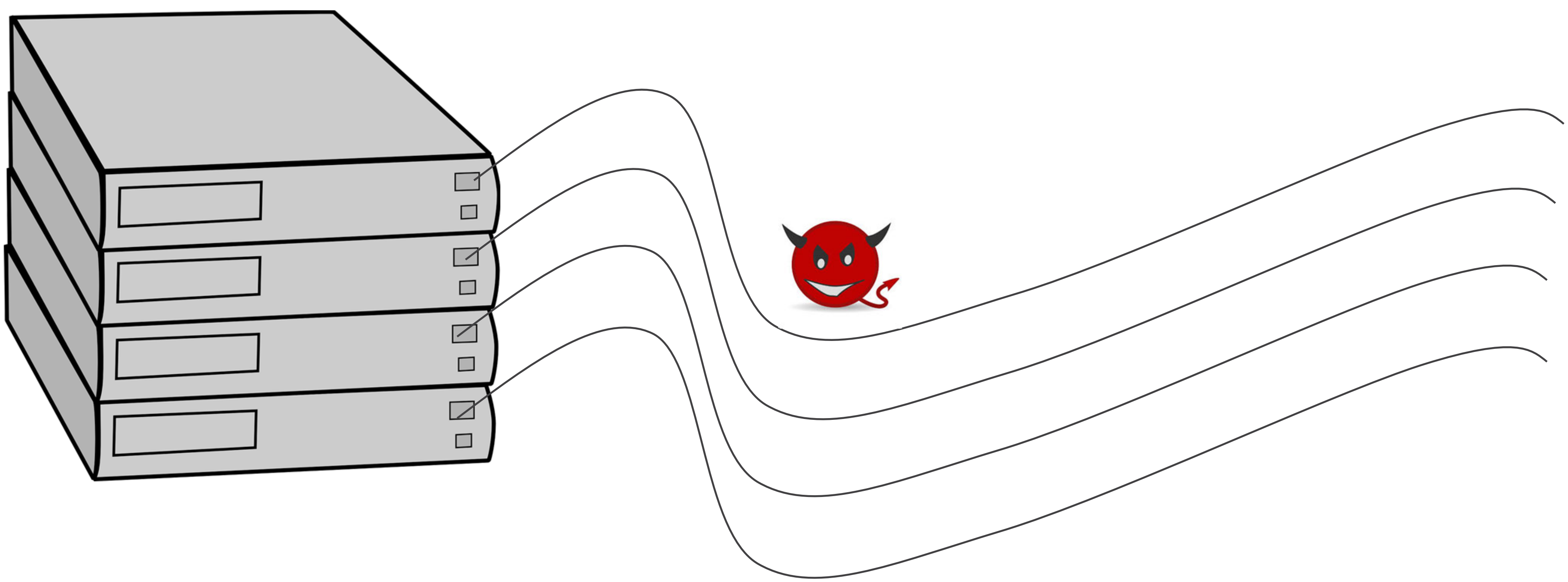
Tim Thijm

Ryan Beckett
(Microsoft)

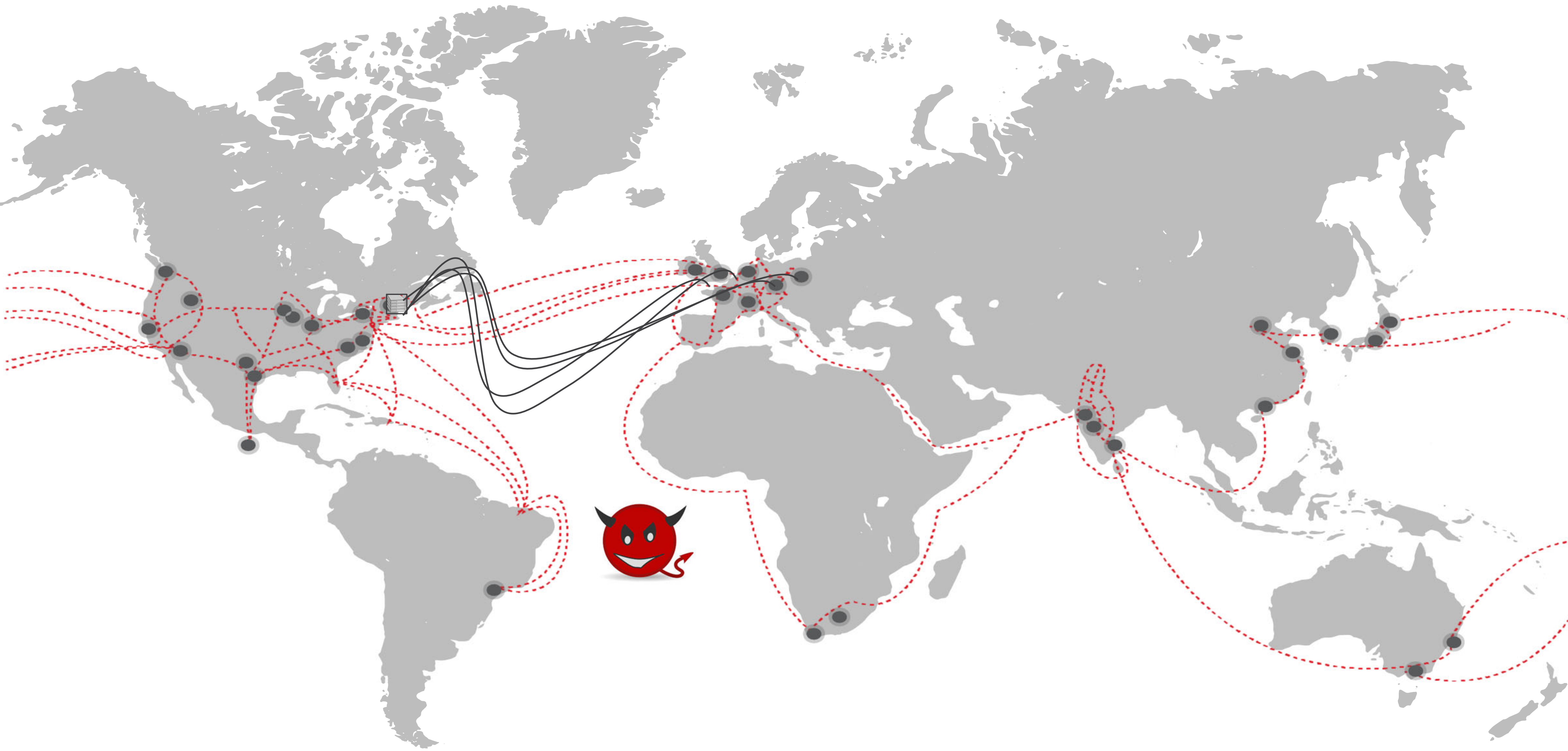
Aarti Gupta

Ratul Mahajan
(UW)

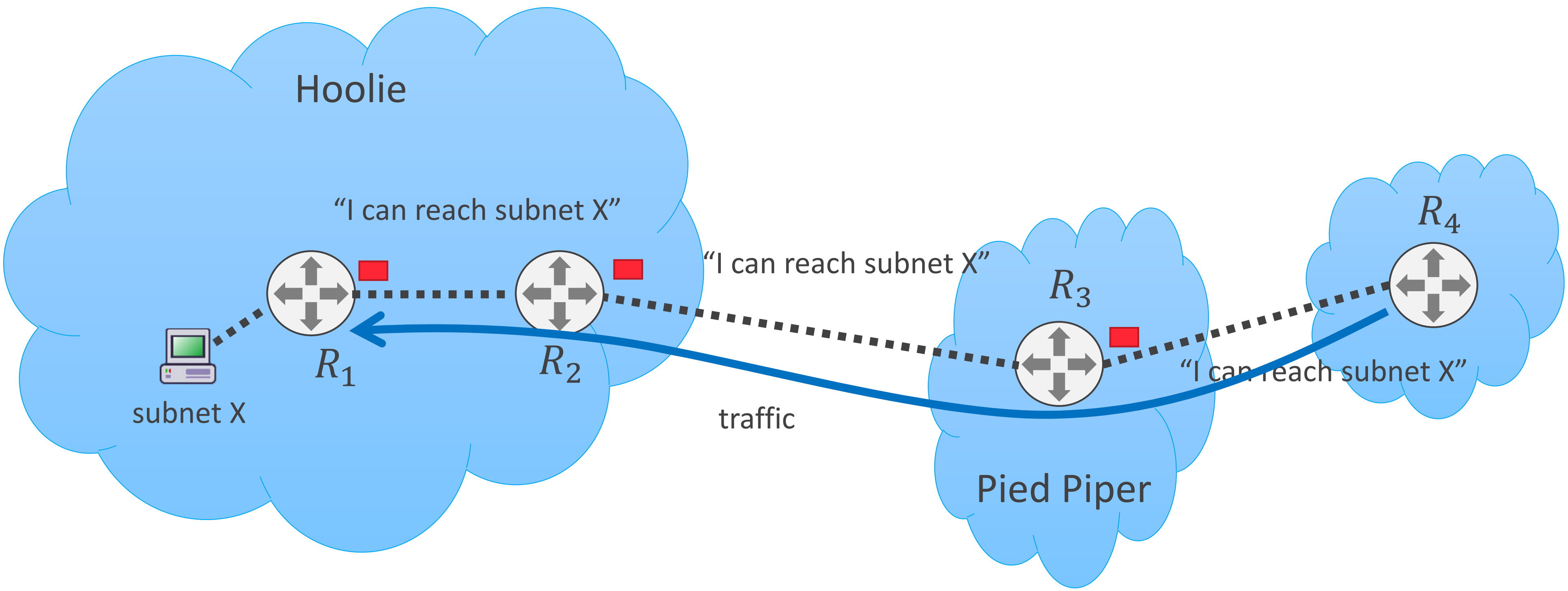
Language-Based Security



Language-Based Security for Networks

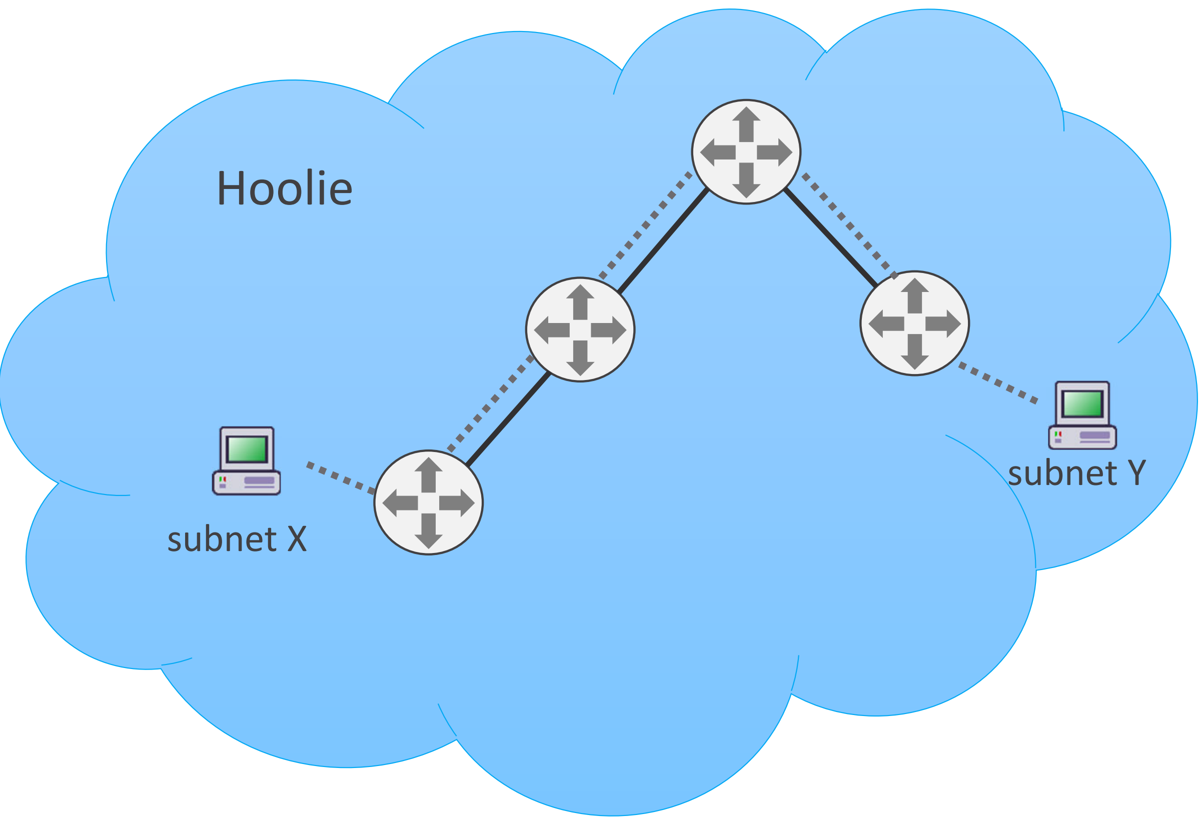


Routing 101

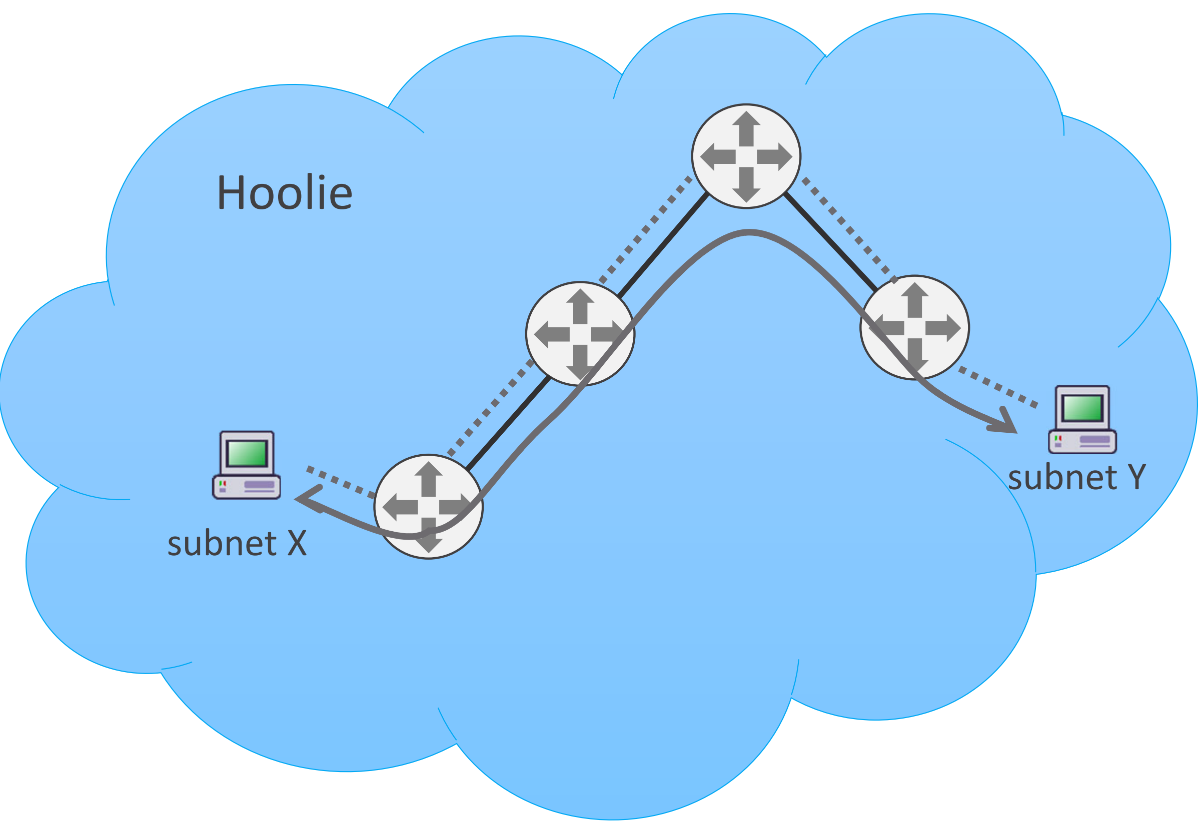


Hoolie
“I can reach subnet X”
“I can reach subnet X”
R_3
R_4
R_1
R_2
subnet X
“I can reach subnet X”
traffic
Pied Piper

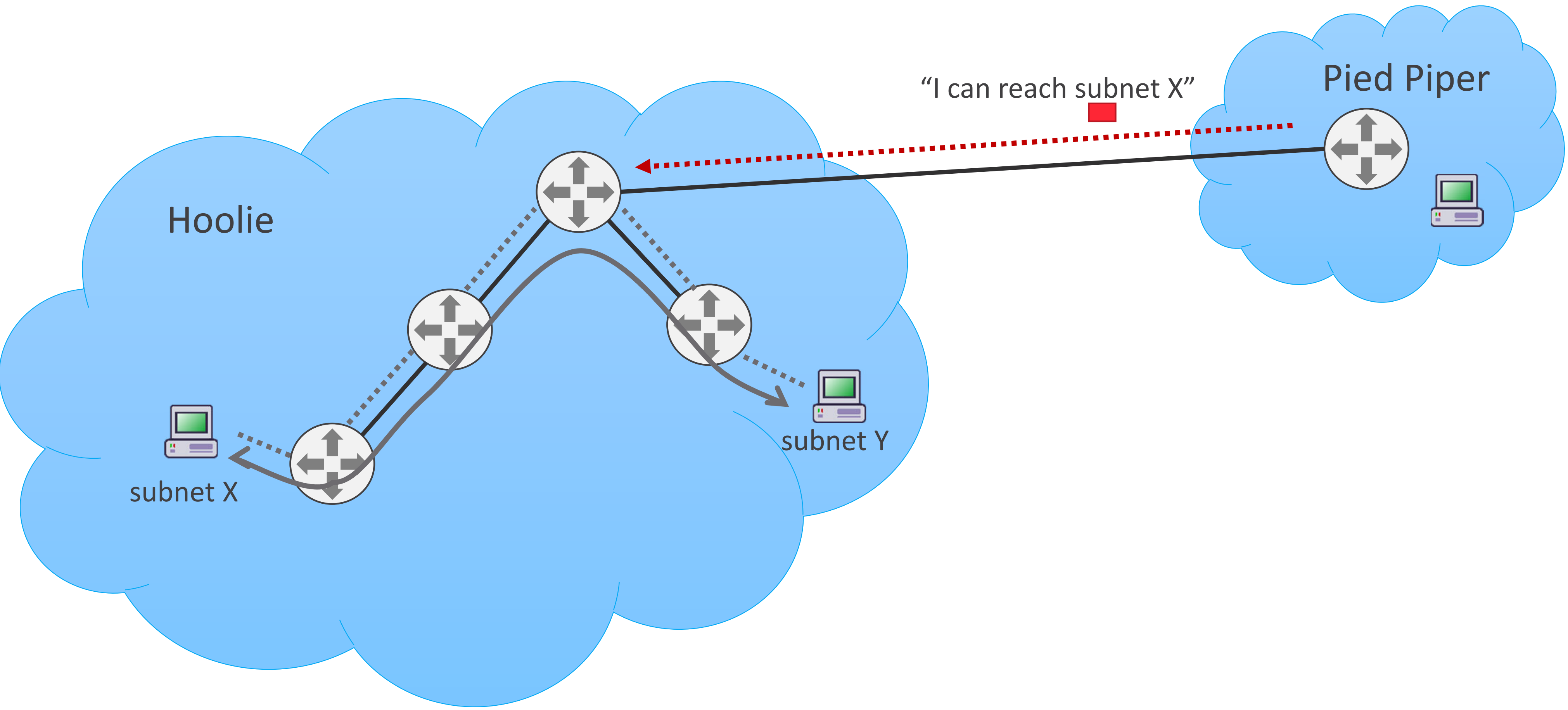
An Example Route Hijack



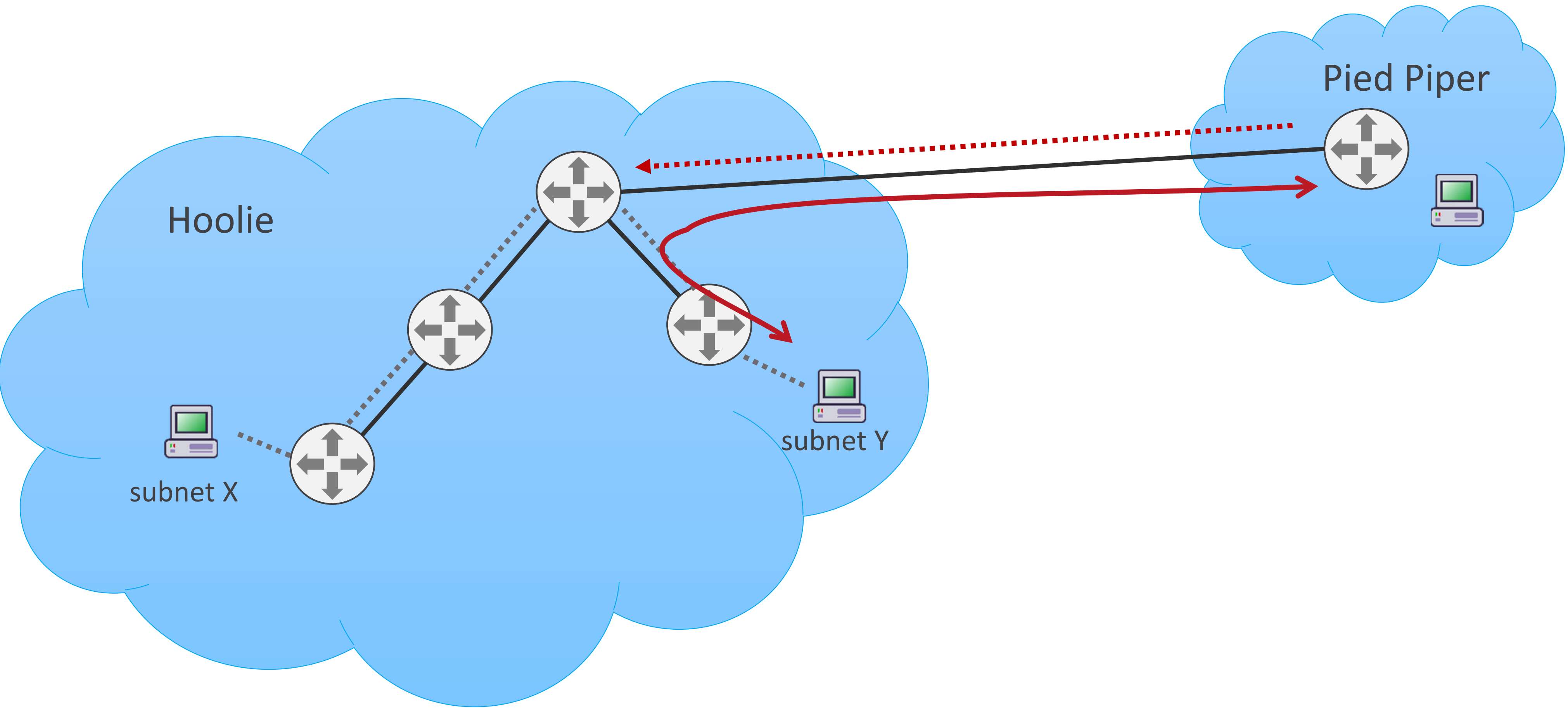
An Example Route Hijack



An Example Route Hijack



An Example Route Hijack



This Kind of Thing Happens Too Often

Why?

Networks are:

- Large (100K+ LOC)
- Distributed
- Low-level
- Multiple vendors
- Subject to failures

Too much for humans to handle

```
interface Ethernet0
  ip address 172.16.0.0/31
  ... configuring topology

ip route 192.168.1.0 255.255.255.0 192.168.2.0
... static routes

bgp router 1
  redistribute static
  neighbor 172.16.0.1 remote-as 2
  neighbor 172.16.0.1 route-map RMO out
  ... Configuring BGP connections

router ospf 1
  redistribute static metric 20 subnets
  distance 70
  network 192.168.42.0 0.0.0.255 area 0
  ... Configuring OSPF connections

ip community-list standard comm1 permit 1:2 1:3
ip prefix-list pfx permit 192.168.2.0/24
route-map RMO permit 10
  match community comm1
  match ip address prefix-list pfx
  set local-preference 200
route-map RMO permit 20
  set metric 90
  ... Configuring routing policies
```

We need automated analysis!

Generic Network Models

To model the many ad hoc vendor languages in a uniform way

[Griffin 2002, Sobrinho 2005]

[SIGCOMM 2017, SIGCOMM 2018, PLDI 2020]

Effective Abstractions and Efficient Algorithms

To analyze these model at scale

[POPL 2020, PLDI 2020]

Network Models

Routing Algebra

[Griffin 2002, Sobrinho 2005]

Topology: (V, E)

Algebra: $(S, \oplus, f, init)$

set of routes
(protocol messages)

merge
$\mathrm{S} \rightarrow S \rightarrow S$
(select preferred route)

transfer
$\mathrm{E} \rightarrow S \rightarrow S$

initial route
$\mathrm{V} \rightarrow S$

Given an algebra, one can ***simulate*** it, looking for its ***solutions***.

Routing Example (Idealized BGP)

(no route)

S = { ∞ } U { (*preference*, *path*, *set of tags*) }

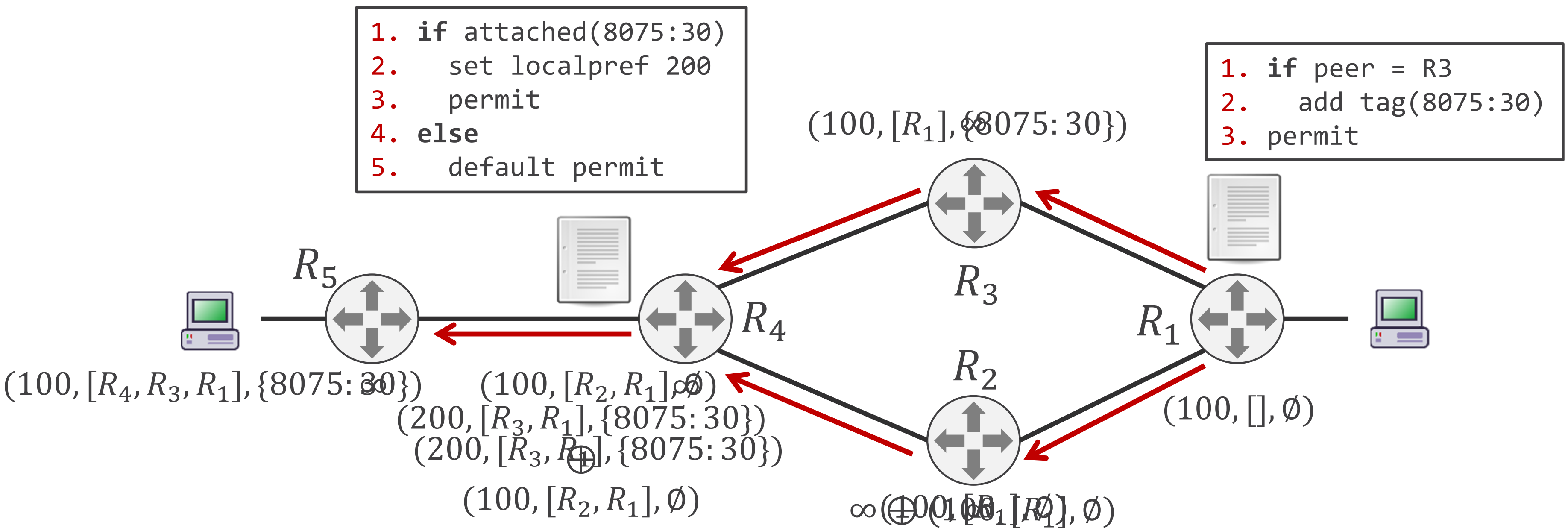
$\oplus$ = "select the most preferred route"
(route with higher preference, shorter path)

f (src,dst) = add src to path;
adjust preference, tags according to configuration

init = given by configuration

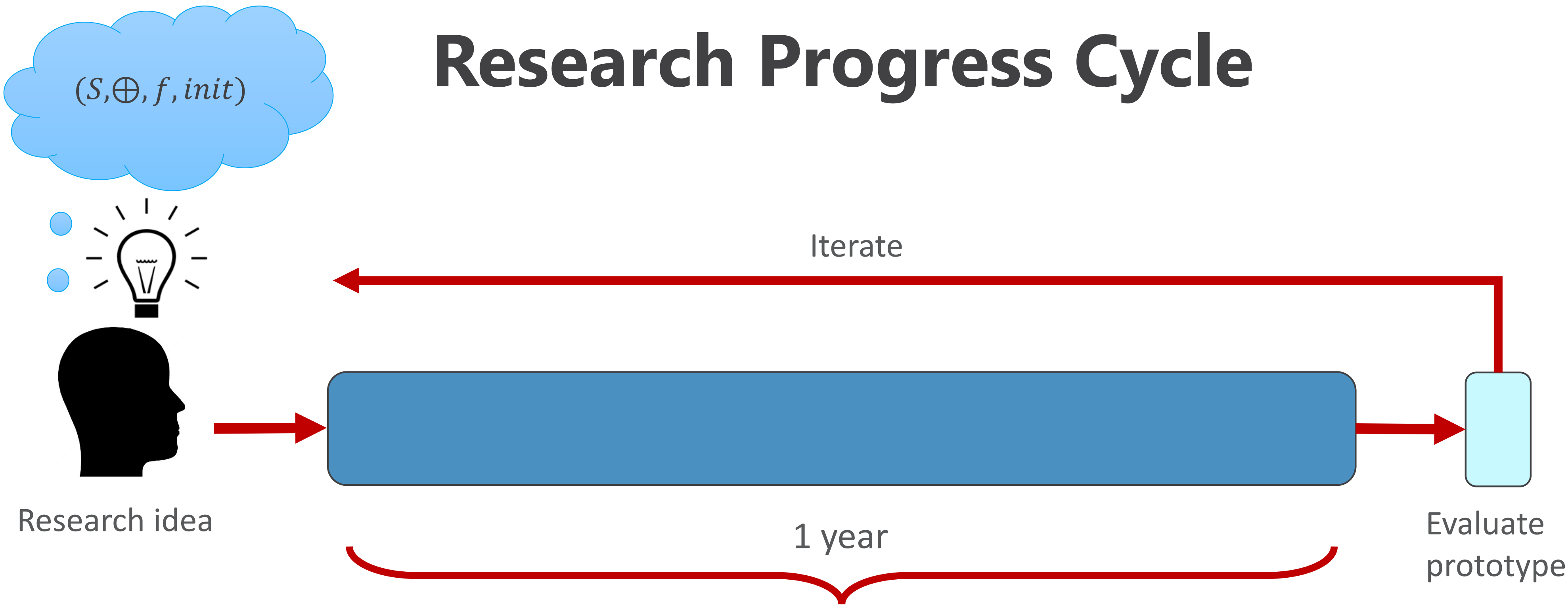
Routing Example (Idealized BGP)

messages S = { ∞ } U { (*preference*, *path*, *set of tags*) }



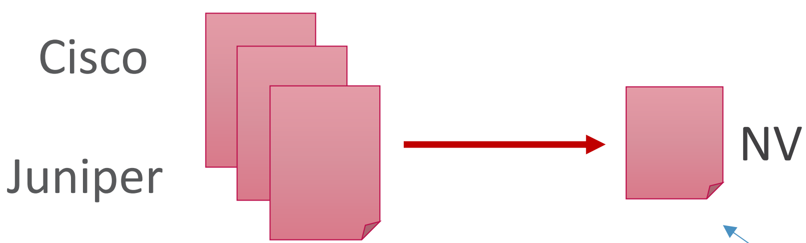
Further propagation of routes causes no change? We have found a ***solution***.

Research Progress Cycle

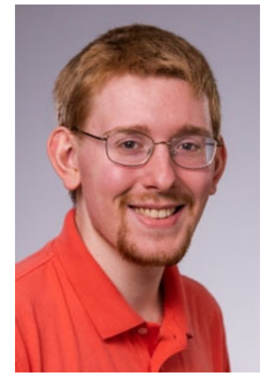


Cisco (IOS, NX-OS) Juniper, Arista
BGP, OSPF, ISIS, RIP, iBGP Route Reflectors, Redistribution,
Conditional advertisement, aggregation, ACLs, MPLS, GRE, …

NV: A Language for Modelling Networks



Nick Giannarakis



Devon Loehr

Ryan Beckett
(Microsoft)

- ad hoc
- non-uniform
- non-compositional
- complex
  - *23+ commands to set protocol fields*

- standard
- uniform
- compositional
- concise
  - *1 command to get a record field*

NV Language

idealized_bgp.nv

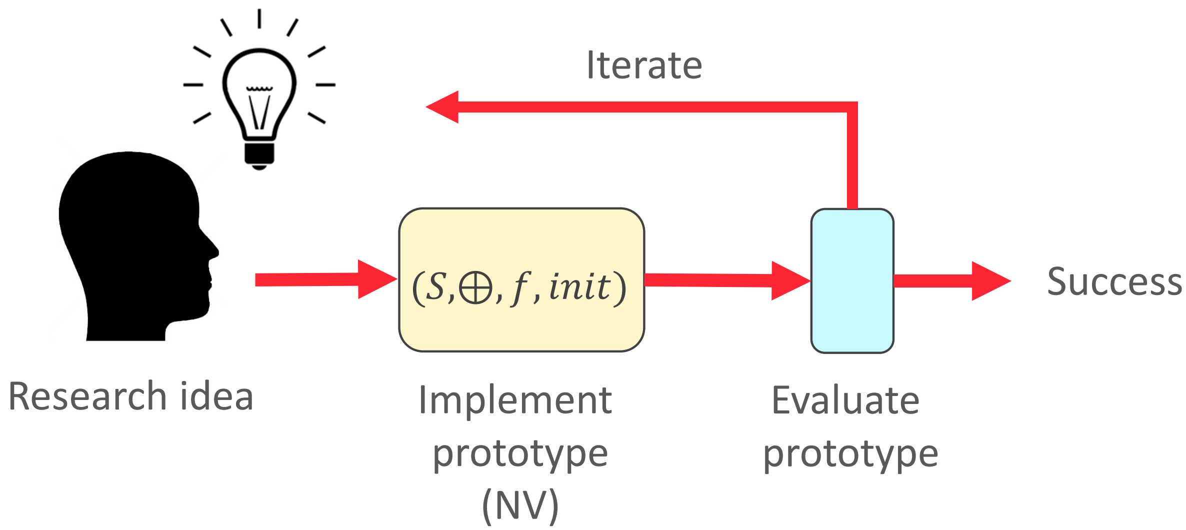
```
let nodes = 5;
let edges = { 1-2; 1-3; 2-4; 3-4; 4-5; }

type route = {pref:int; len:int; orig:node; tags:int set}
type message = option[route]


let init n = if n = 1 then Some {pref=100; len=0; orig=1; tags=empty;}
                      else None

let f e m =
  let protocol m = {pref=m.pref; len=m.len + 1; orig=orig; tags=tags;} in
  let config e m = ... in
  m |> protocol |> config e

let merge n m1 m2 = if is_preferred m1 m2 then m1 else m2
```

NV Language

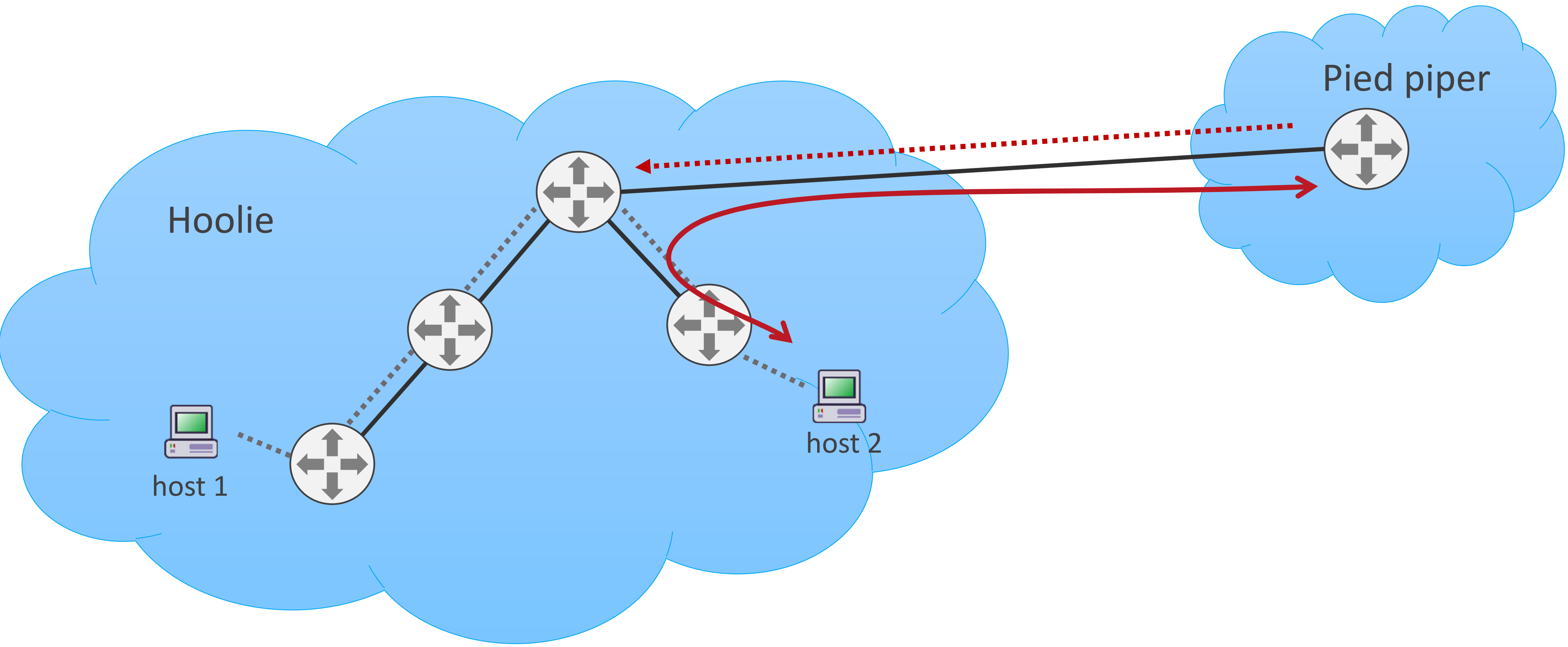
idealized_bgp.nv

```
let nodes = 5;
let edges = { 1-2; 1-3; 2-4; 3-4; 4-5; }

let init n = ...
let f e m = ...
let merge n m1 m2 = ...

let sol = solution {init=init; trans=f; merge=merge;}

(* Does router R5 have a route to R1? *)
let prop sol =
  match sol[5] with
    None -> false
  | Some {pref=_; len=_; orig=n; comm=_;} -> (n = 1)

assert prop(sol);
```

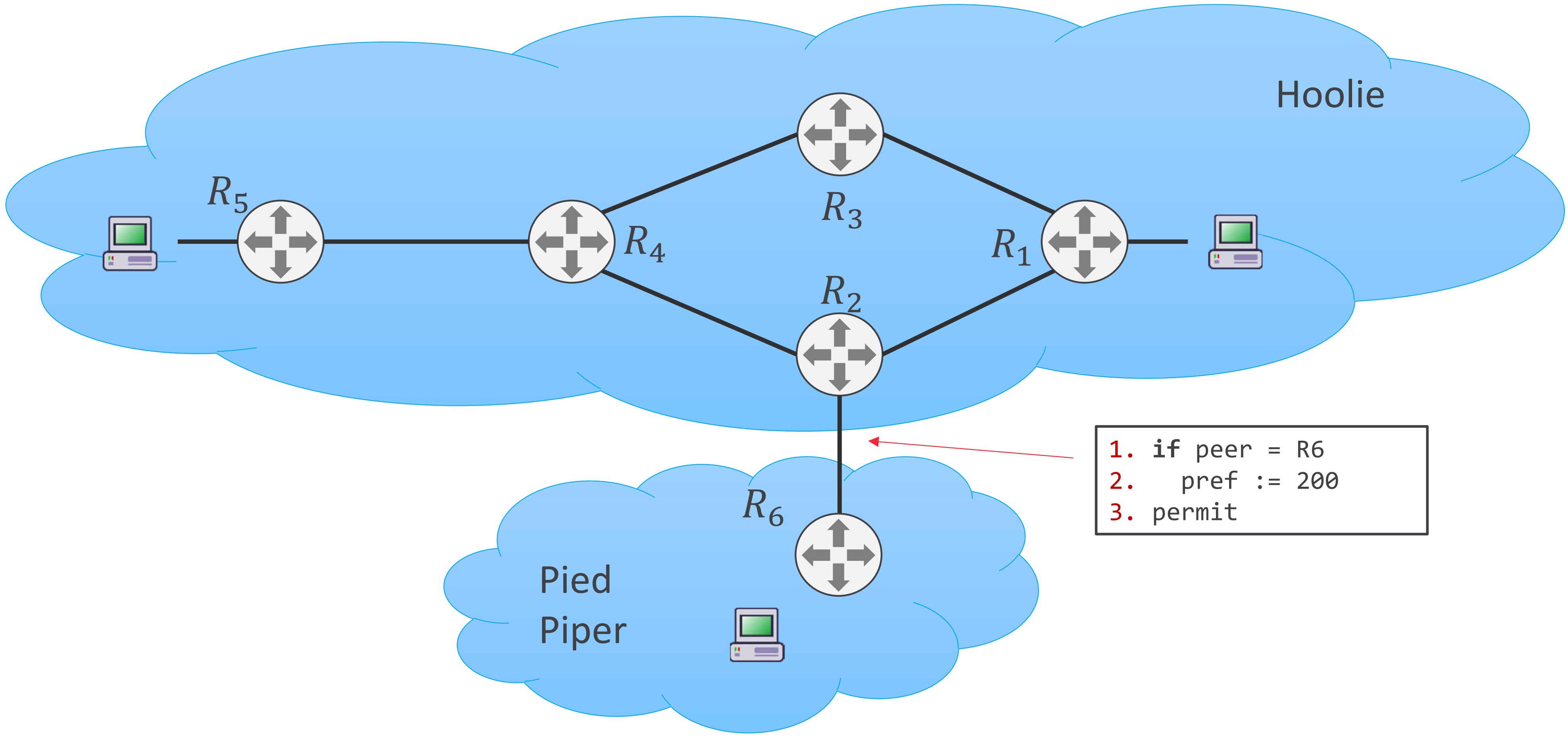
The Power of Language: Exploring New Models



Recall: A BGP Hijack



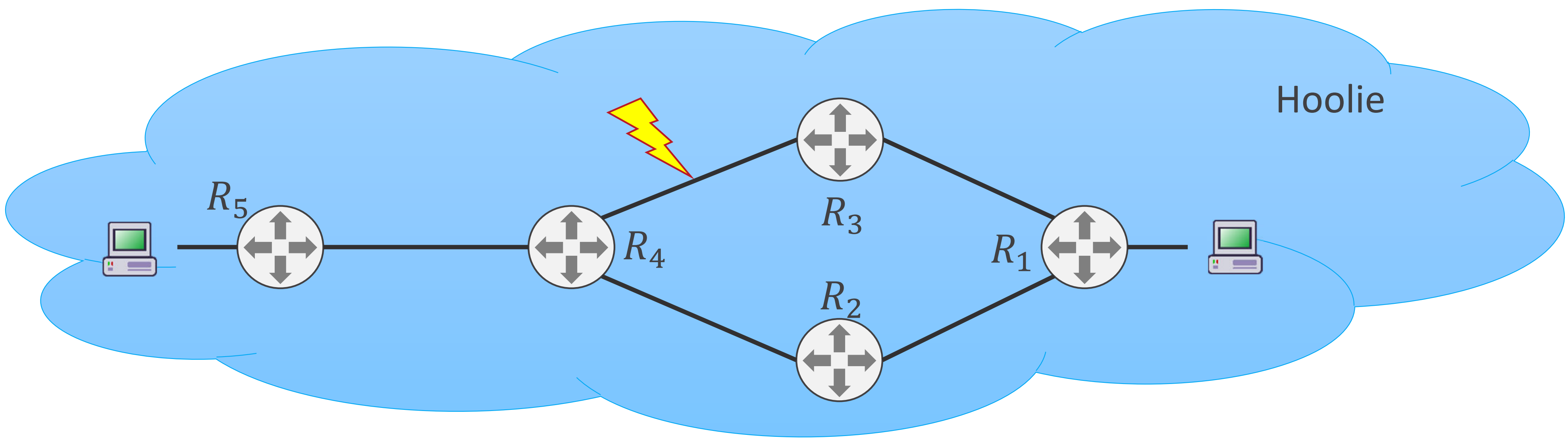
Can Pied Piper Hijack Hoolie?



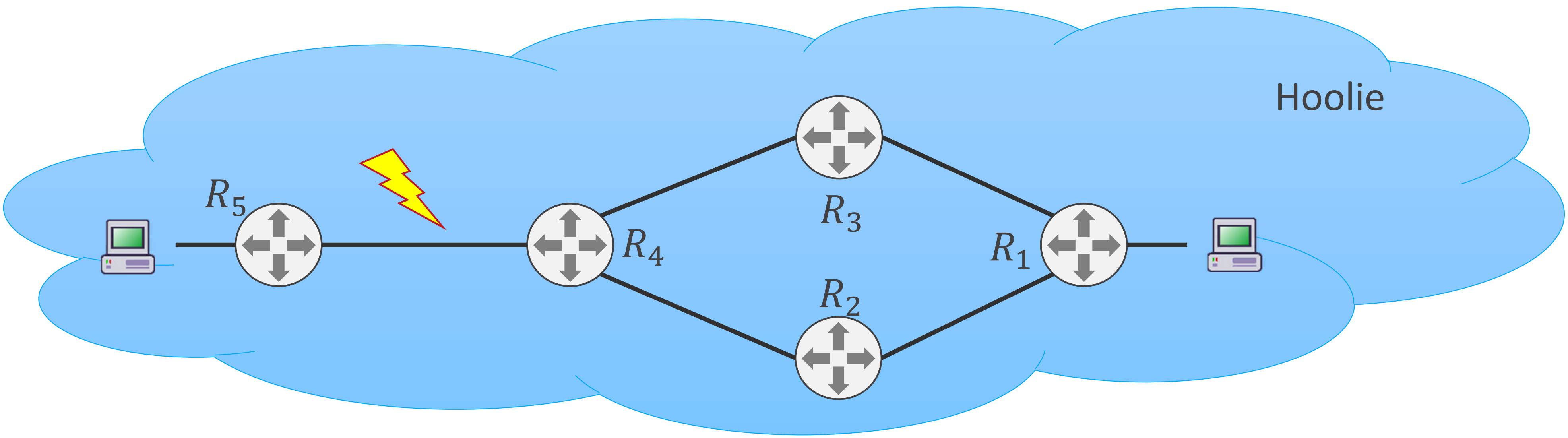
Can Pied Piper Hijack Hoolie?

```
let nodes = 6
let edges = { 1-2; 1-3; 2-4; 3-4; 4-5; 6-2;}

type route = {pref:int; len:int; orig:node; tags:int set}
type message = option[route]

symbolic u : route  (* unknown route *)
require u.orig = 6;

let init n = if n = 6 then Some u else ...

let f e m =
  let protocol m = ... in
  let config e m = match e with | 6~2 -> {pref=200; ... } | _ -> ... in
  m |> protocol |> config e

assert prop(sol);
```

Is Hoolie's Network Fault Tolerant?



Is Hoolie's Network Fault Tolerant?



duh ...

Is Hoolie’s Network Fault Tolerant?

```
let nodes = 5
let edges = { 1-2; 1-3; 2-4; 3-4; 4-5}

type route = {pref:int; len:int; orig:node; tags:int set}
type message = option[route]

symbolic failure : edge  (* the failed edge *)

let f e m =
  let fail e m = if e = failure then None else m in
  let protocol m = ... in
  let config e m = ... in
  m |> fail e |> protocol |> config e

assert prop(sol);
```

Aside: Eliminating Symbolic Values

```
type message = option[route]

symbolic failure : edge

let f e m =
  let fail e m = if e = failure then None else m in
  ...
```

```
type message = dict[edge, option[route]]

let f e m =
  let fail e m = mapif (fun e -> e = failure then None else m) m
  ...
```

Aside: Eliminating Symbolic Values

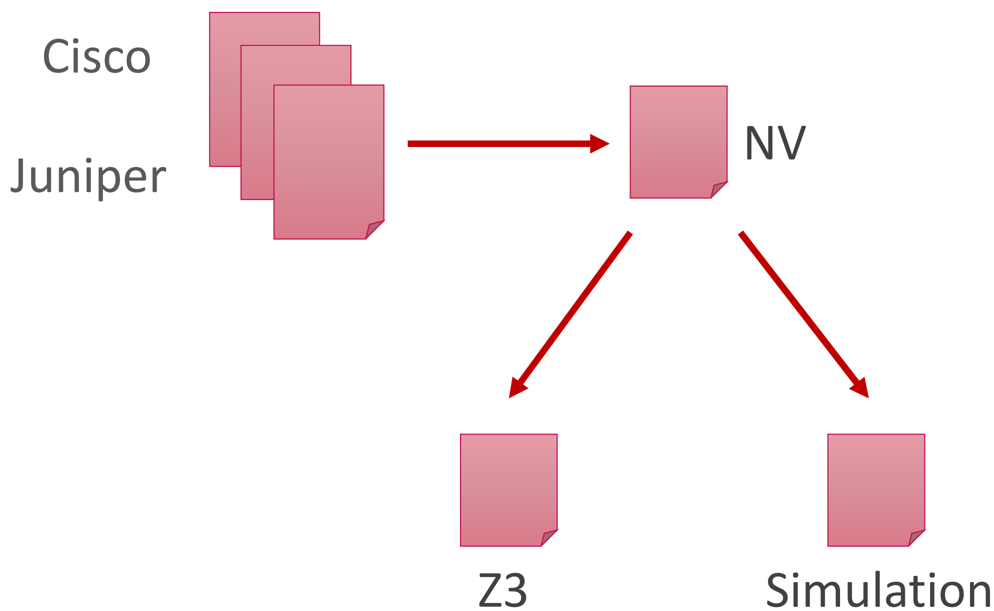
```
type message = option[route]

symbolic failure : edge

let f e m =
  let fail e m = if e = failure then None else m in
  ...
```

```
type message = dict[edge, option[route]]

let f e m =
  let fail e m = mapif (fun e -> e = failure) (fun m -> None) m
  ...
```

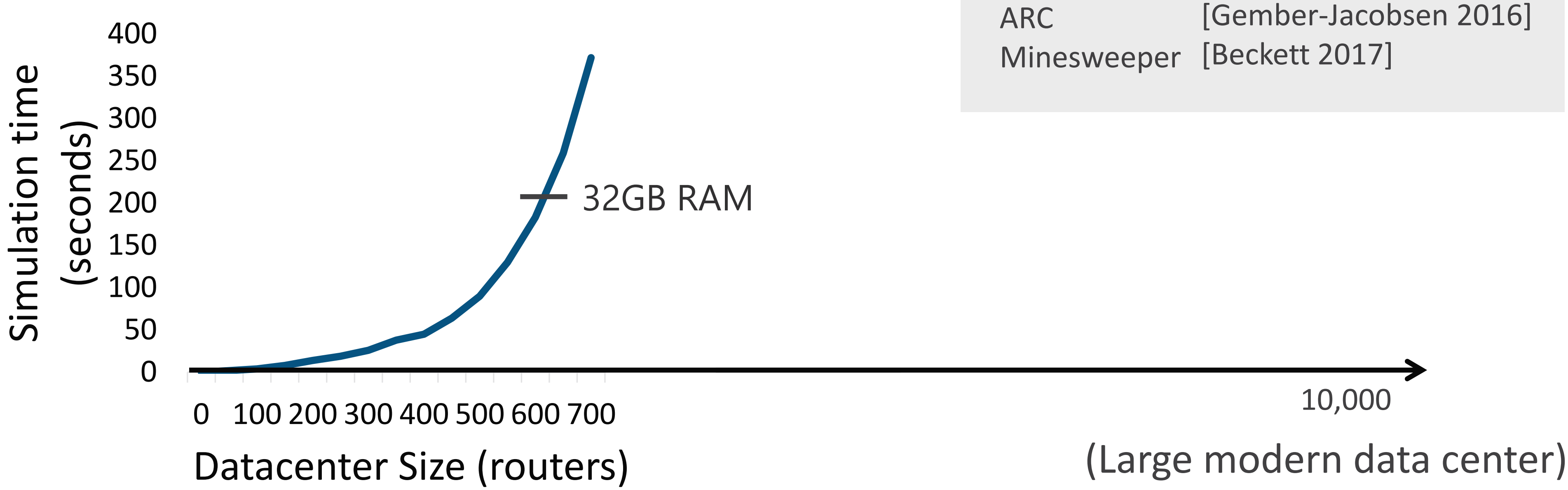
More Realistic Networks

```
type ospf =
  {ad: int; weight: int; areaType: int4; areaId: int;}

type bgp =
  {ad: int; lp: int; aslen: int; comms: set[int16]; origin: int;}

type rib_entry = {
  connected  : option[edge];
  static     : option[edge];
  ospf       : option[ospf];
  bgp        : option[bgp];
  selected   : option[int2]
}

type prefixV4 = { ip: int32; len: int5; }

type attribute = dict[prefixV4, rib_entry]
```

NV Tools



The Scalability Problem



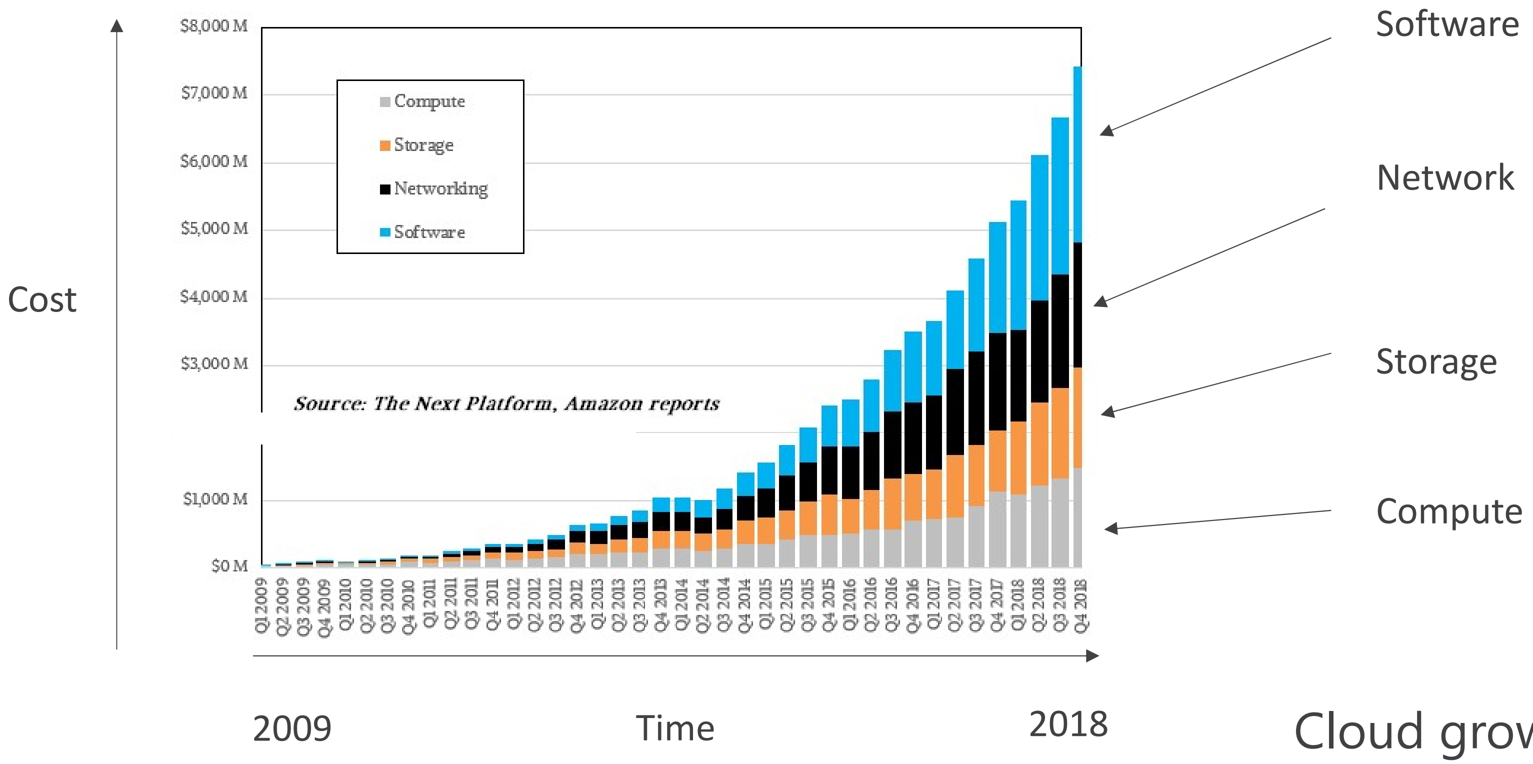
control plane simulation

CBGP [Mai 2011]
Batfish [Fogel 2015]

control plane verification

ARC [Gember-Jacobsen 2016]
Minesweeper [Beckett 2017]

The Scalability Problem (AWS)



Cloud growth by quarter (AWS)

228x growth in networks in a decad

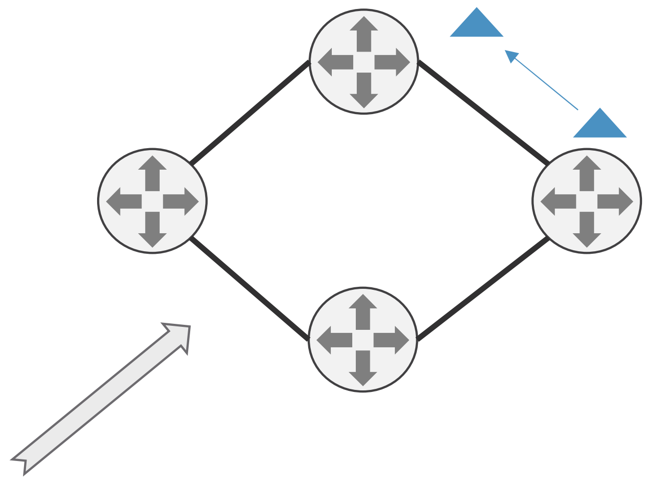
Effective Abstractions & Efficient Algorithms

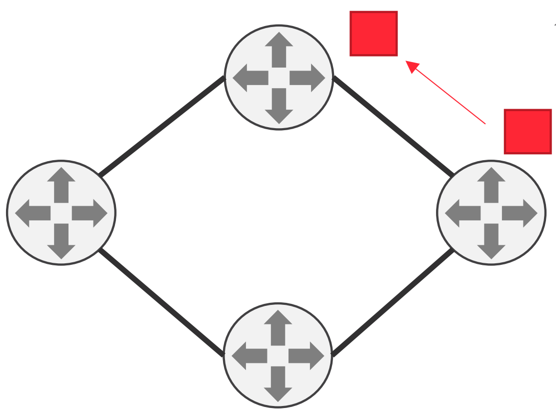
Abstract Interpretation of Routing Algebras

Ryan Beckett

Aarti Gupta

Ratul Mahajan





Message Abstraction: asympototic improvements in time and space

Abstract Interpretation of Routing Algebras

Idealized BGP

option[(preference,
path,
tag set)]

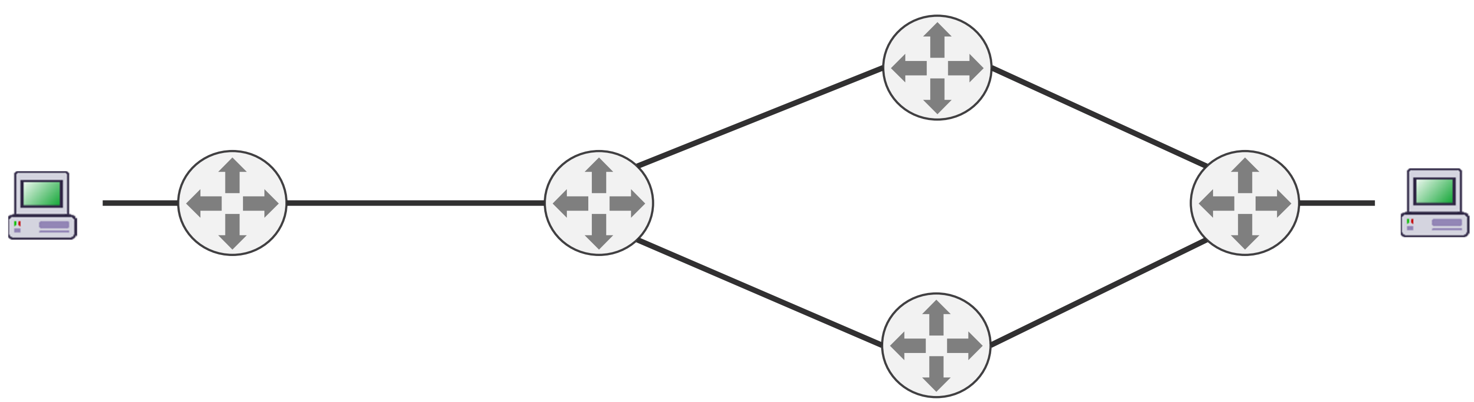
Base Model

option[(preference,
length,
origin,
tag set)]

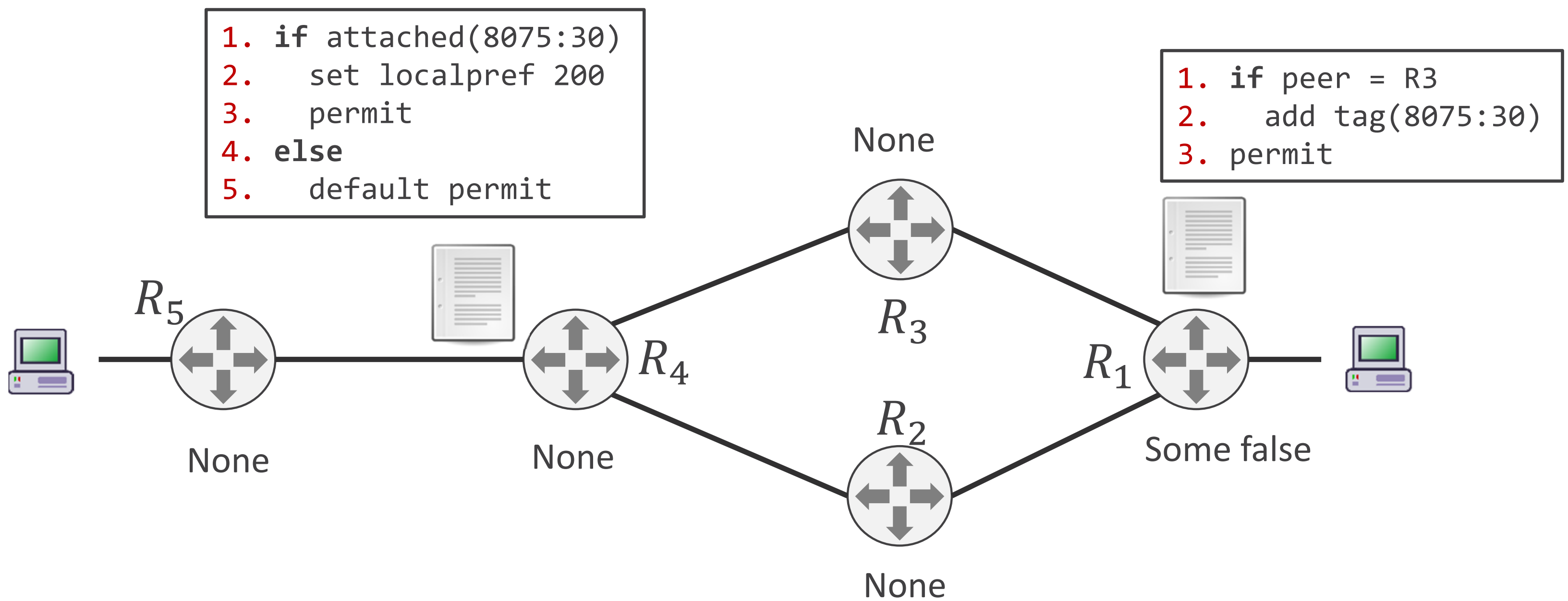
Abstract Model

option[tag abstraction]

true, false, *

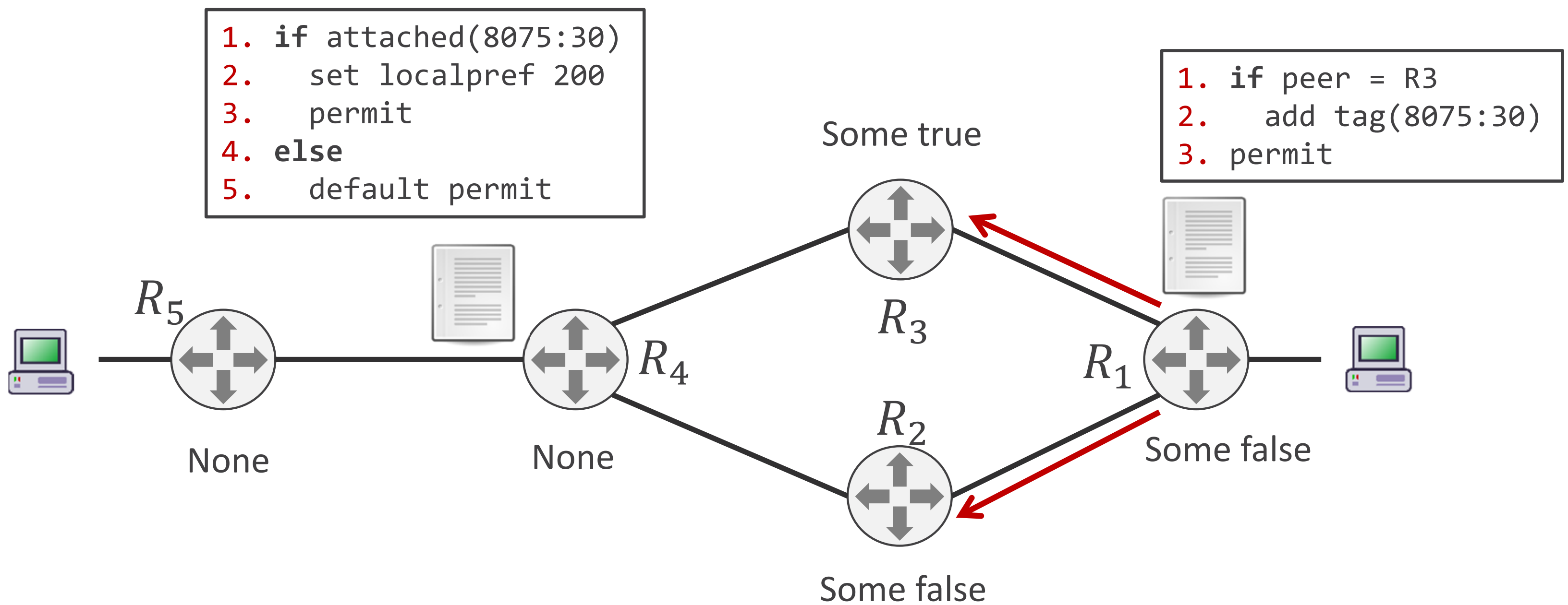


Abstract Interpretation of Routing Algebras



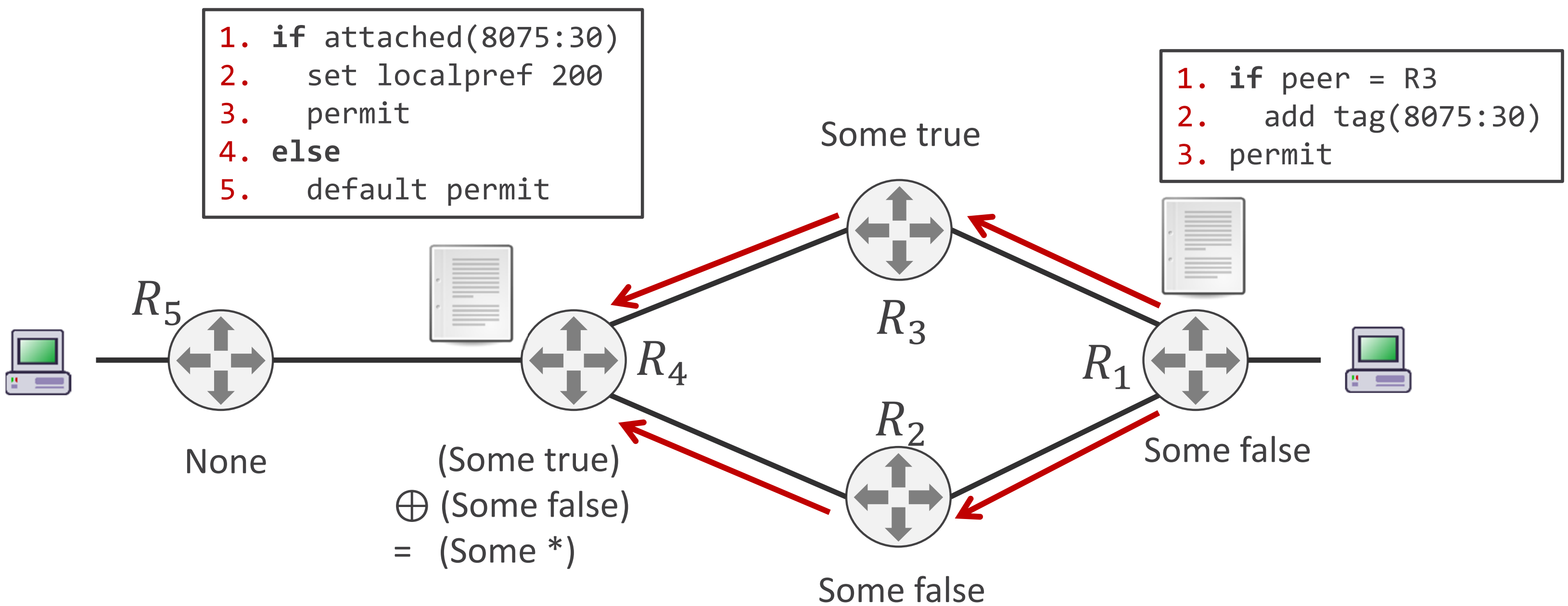
Property: Does R5 obtain any route?

Abstract Interpretation of Routing Algebras



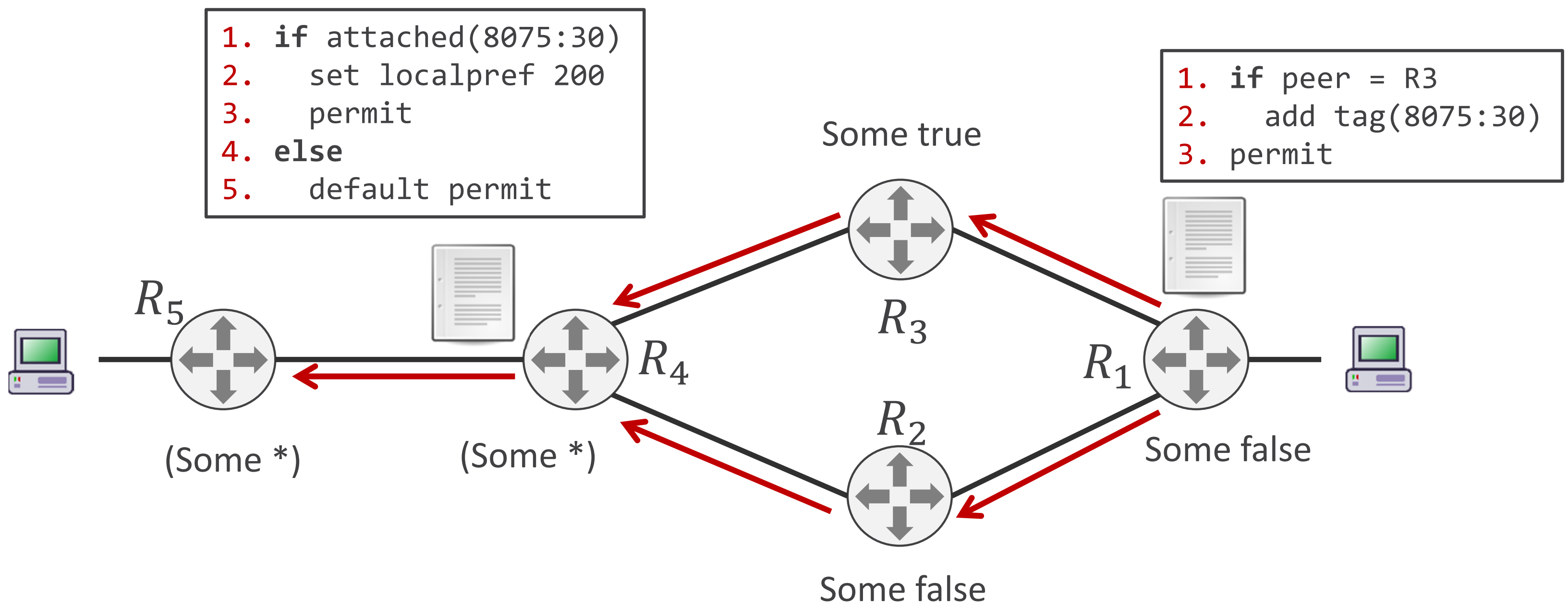
Property: Does R5 obtain any route?

Abstract Interpretation of Routing Algebras



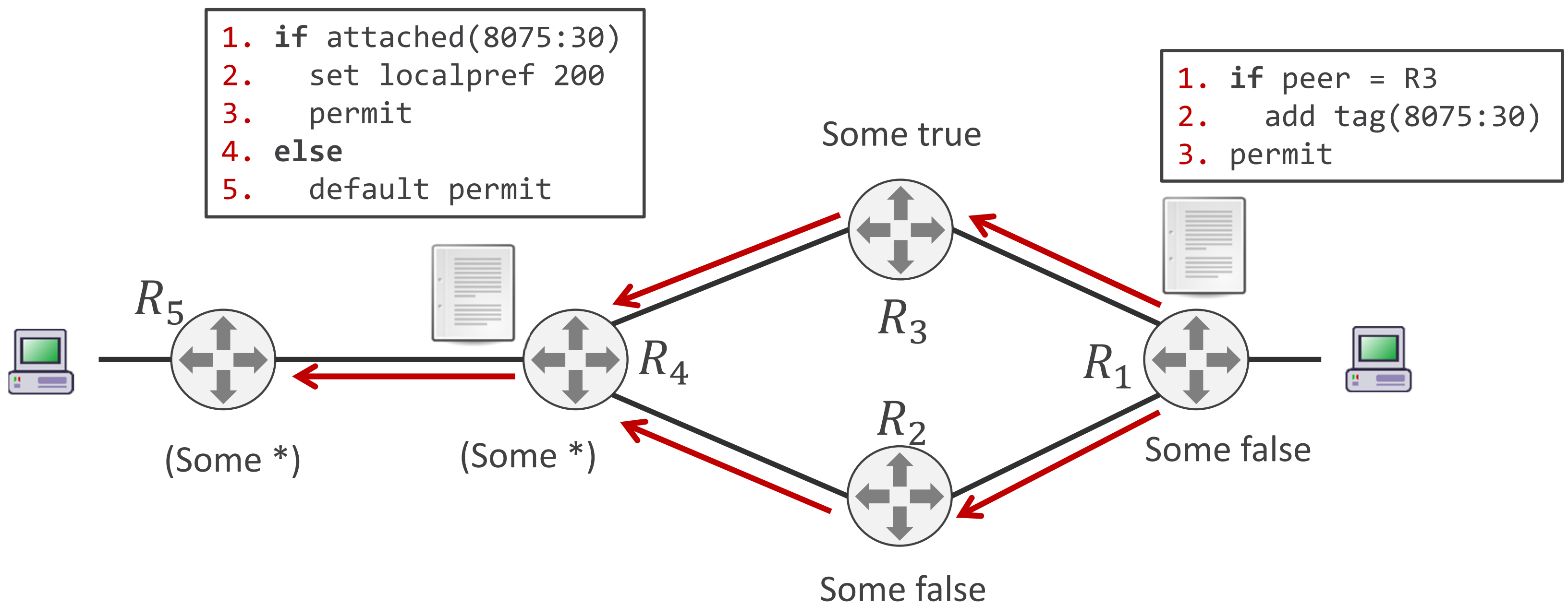
Property: Does R5 obtain any route?

Abstract Interpretation of Routing Algebras



Property: Does R5 obtain any route?

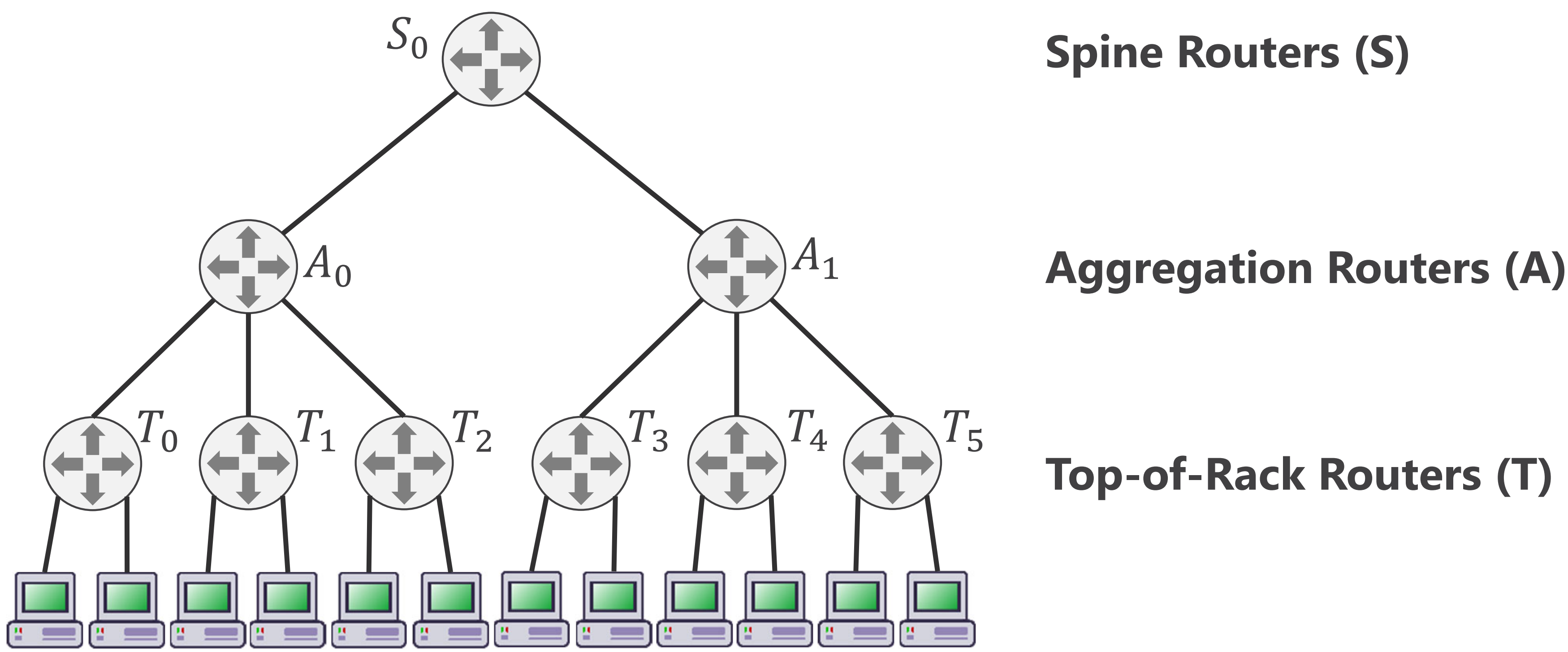
Abstract Interpretation of Routing Algebras



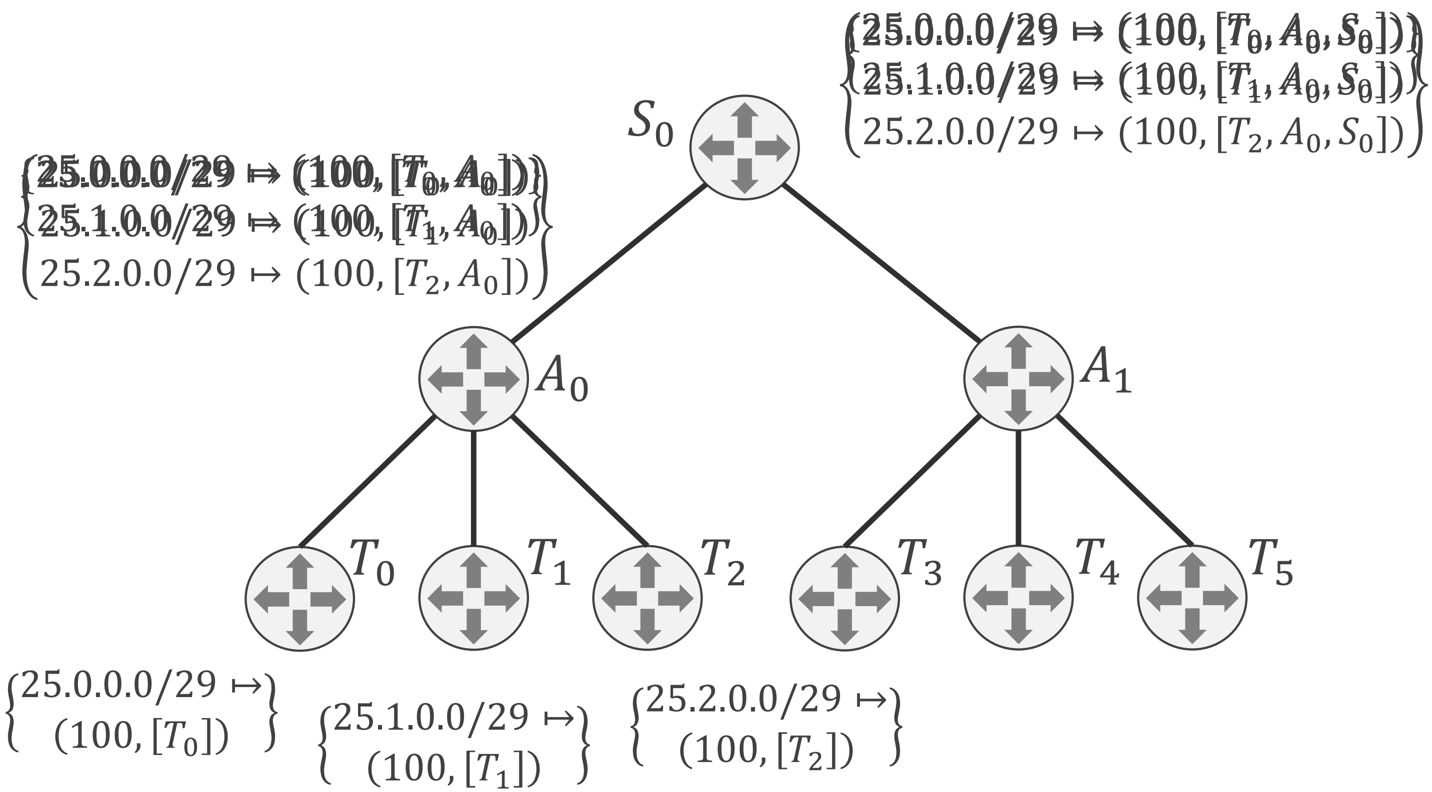
Property: Does R5 obtain any route?

Yes

Example 2: Datacenter Simulation



Example 2: Datacenter simulation

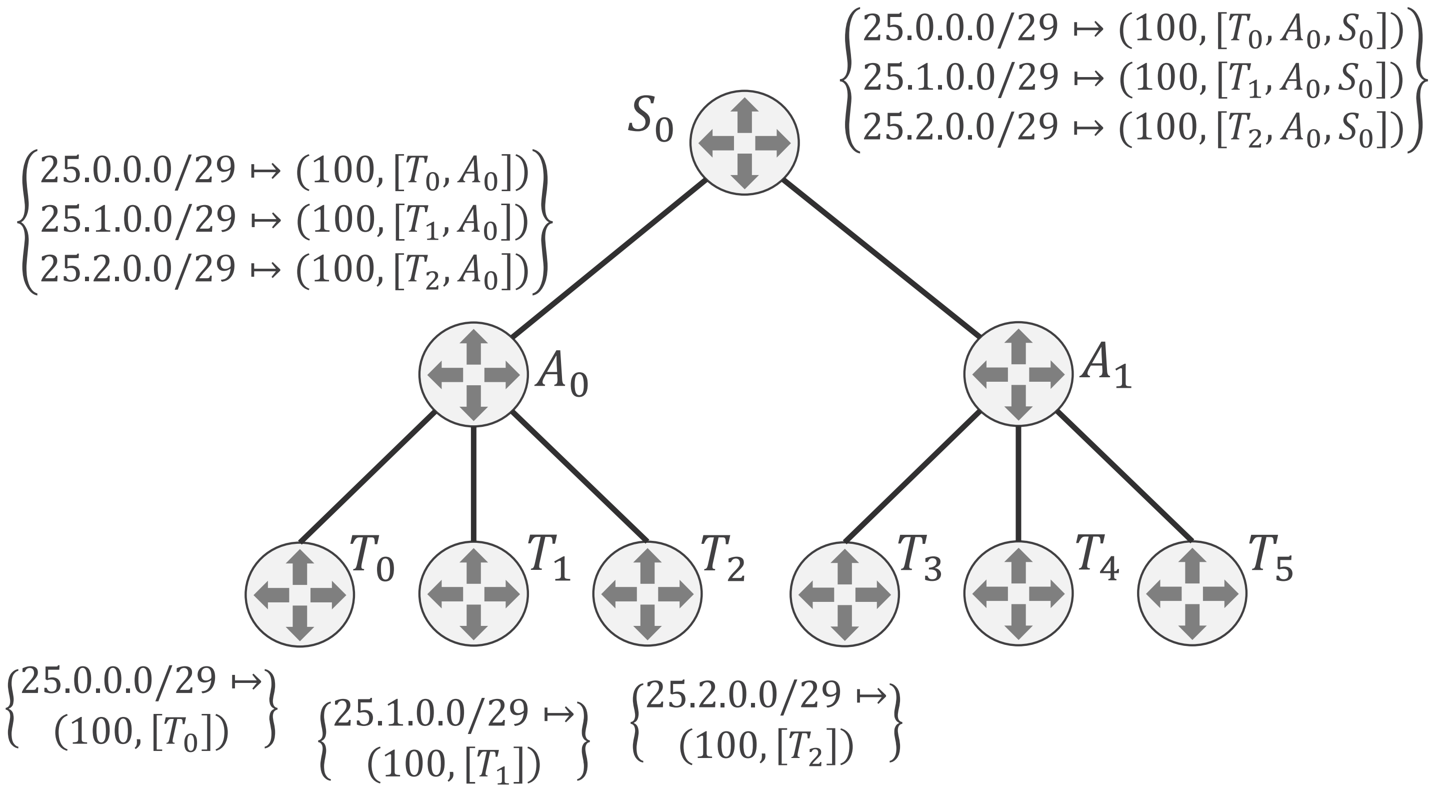


Edges: $n\sqrt{n}$

Destinations: n

Complexity: $n^2\sqrt{n}$

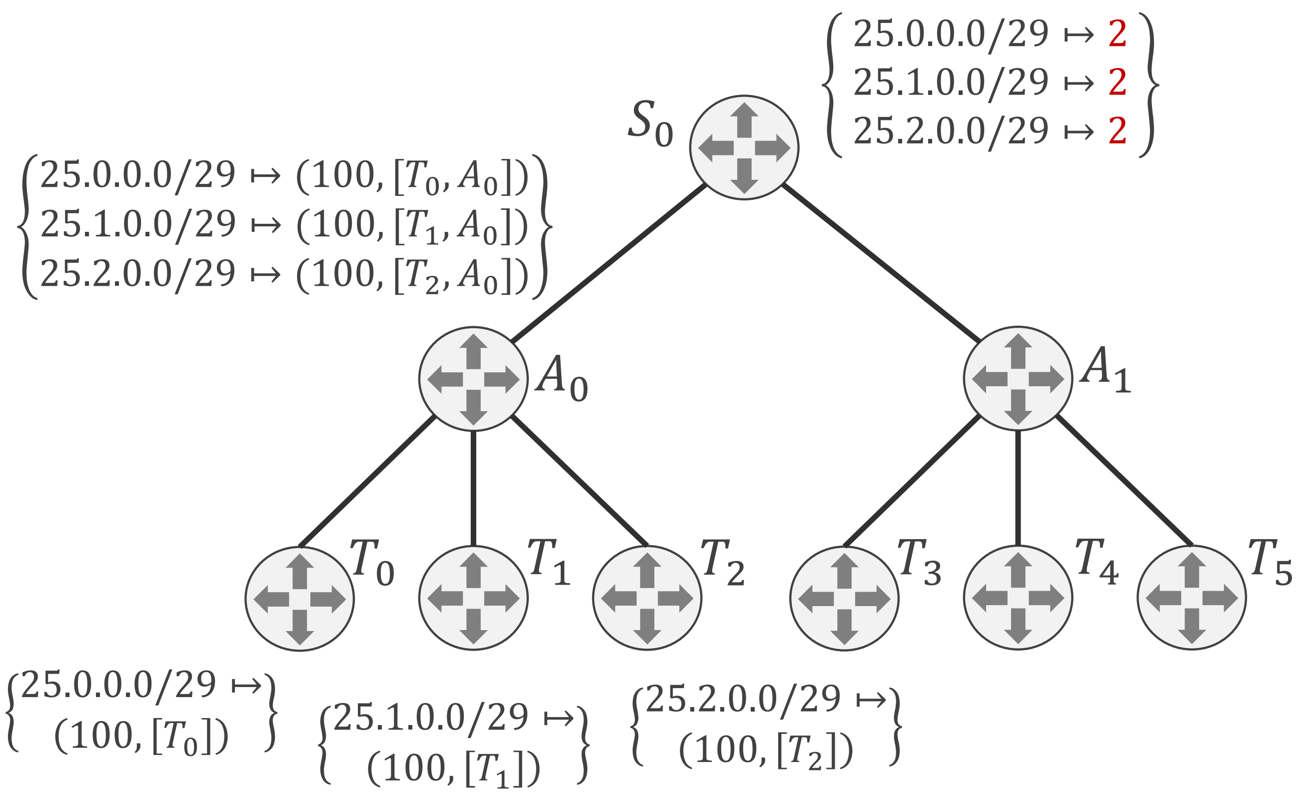
Example 2: Datacenter Simulation



Abstraction:

pref * path --> length

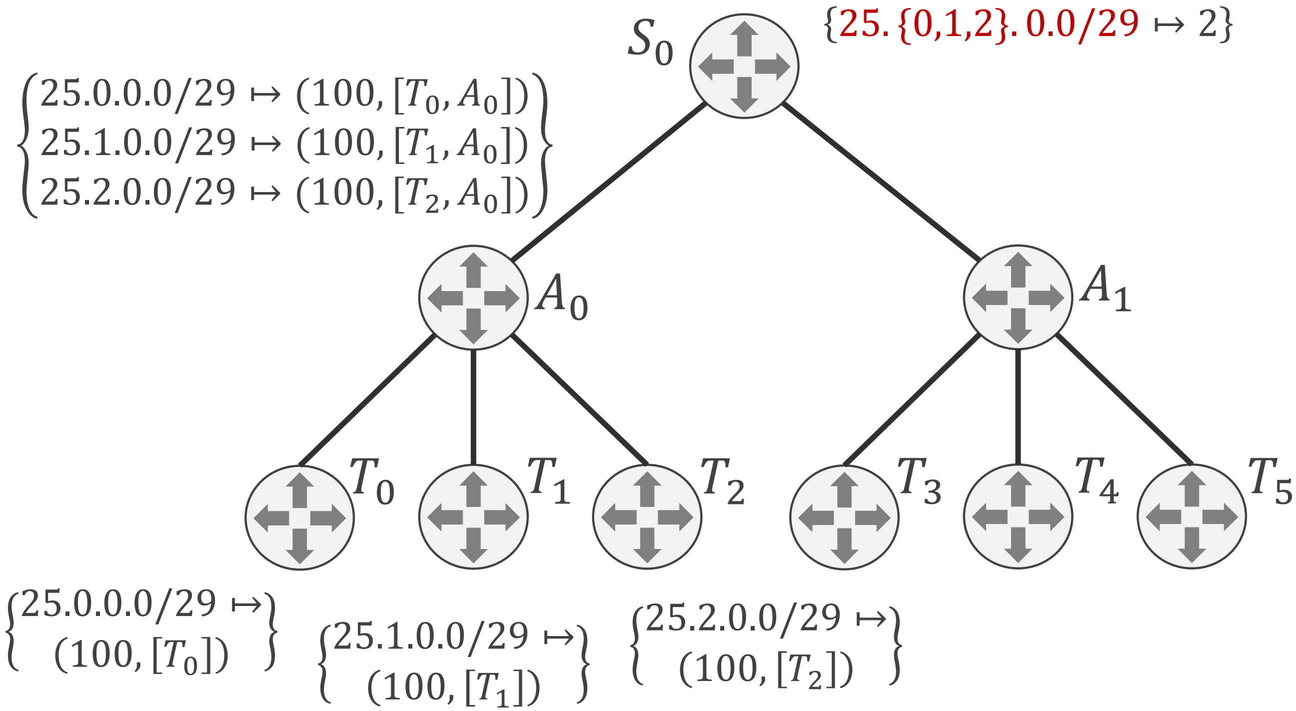
Example 2: Datacenter Simulation



Abstraction:

pref * path --> length

Example 2: Datacenter Simulation

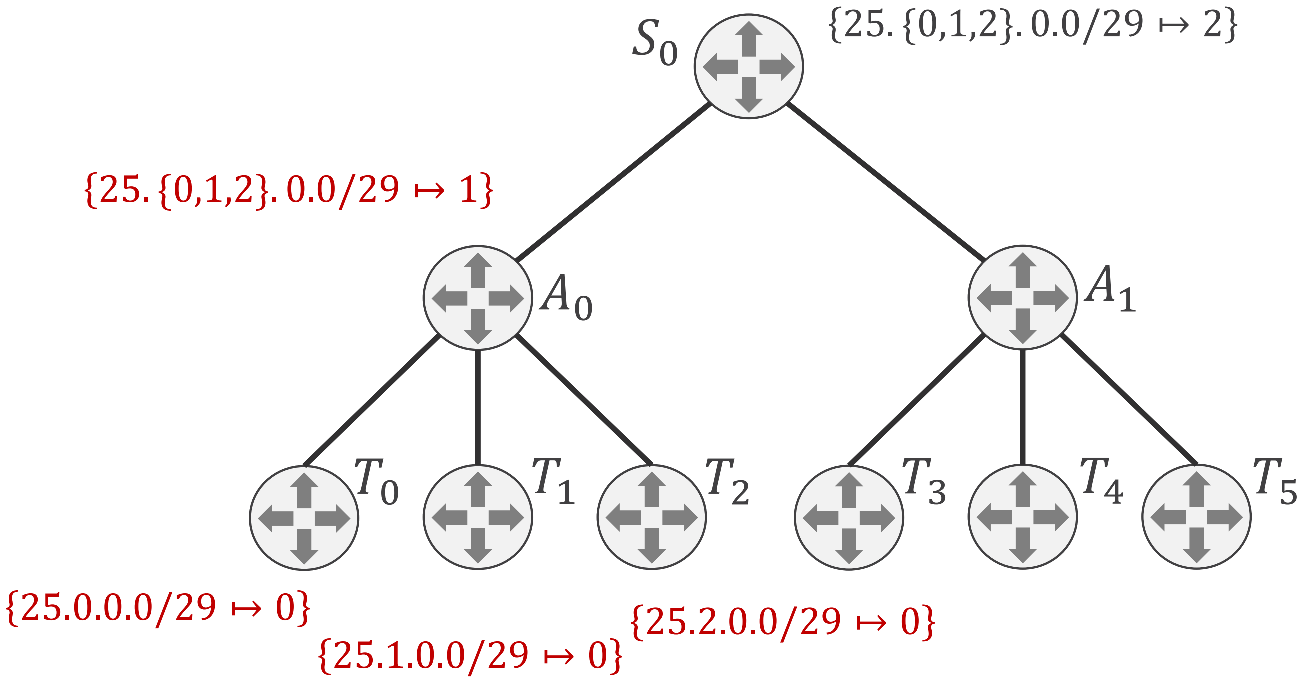


Abstraction:

pref * path --> length

Represent dictionaries efficiently using multi-terminal BDDs

Example 2: Datacenter Simulation

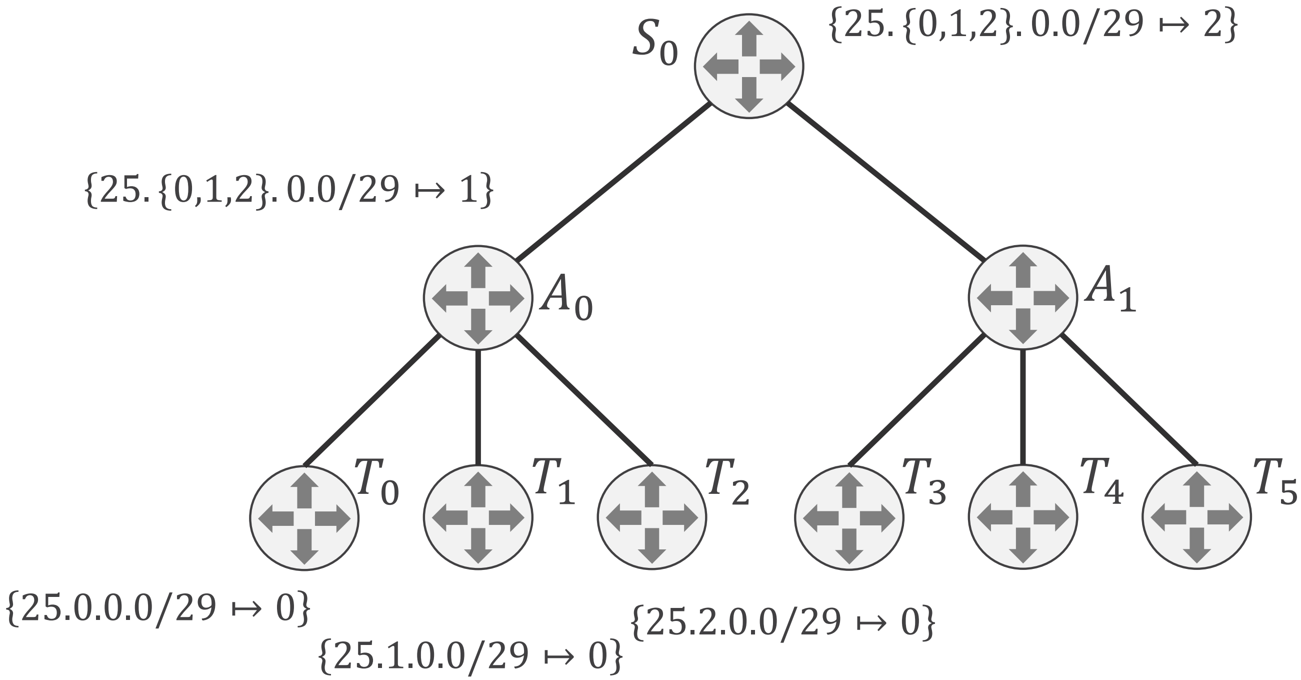


Abstraction:

pref * path --> length

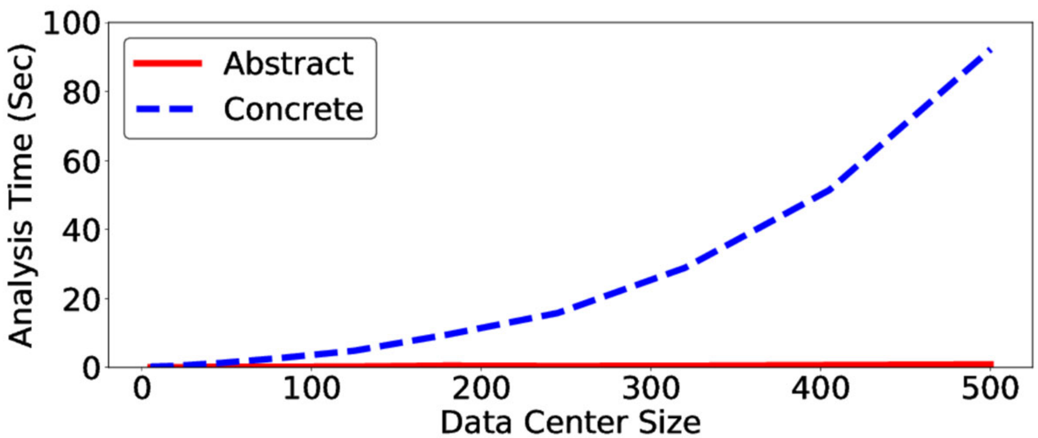
Represent dictionaries efficiently using multi-terminal BDDs

Example 2: Datacenter Simulation



Complexity: $n\sqrt{n}$

Experimentally, Synthetic Data Centers



Simulation time vs. data center size
for verifying all-pairs connectivity

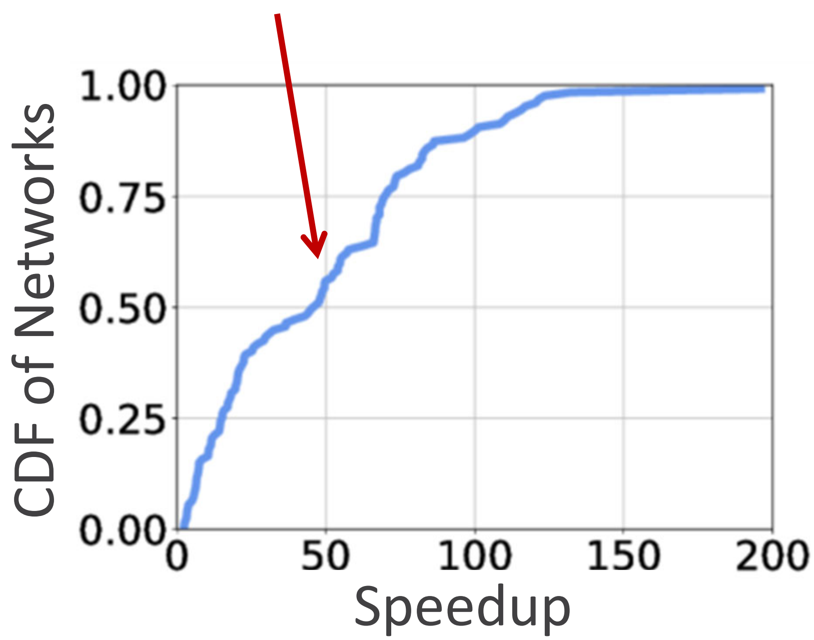
Experimentally, Real Networks

Considered 127 production networks at Microsoft

- Run multiple protocols (BGP, OSPF, connected, static, …).
- Networks use many protocol features.
  - Route redistribution, custom pref, tags, regex filters, ACLs etc.
- 1K to 100K lines of configuration per device.
- Networks have ~10 to 1000 routers.

Speedup compared to concrete simulation

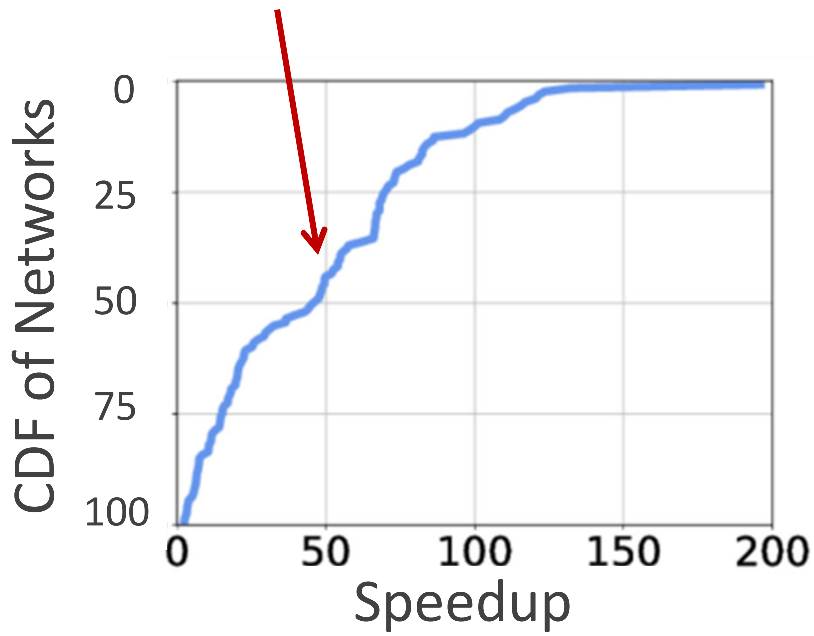
Half of networks have
more than 50x speedup



Speedup grows as
network size grows.

Speedup compared to concrete simulation

Half of networks have
more than 50x speedup

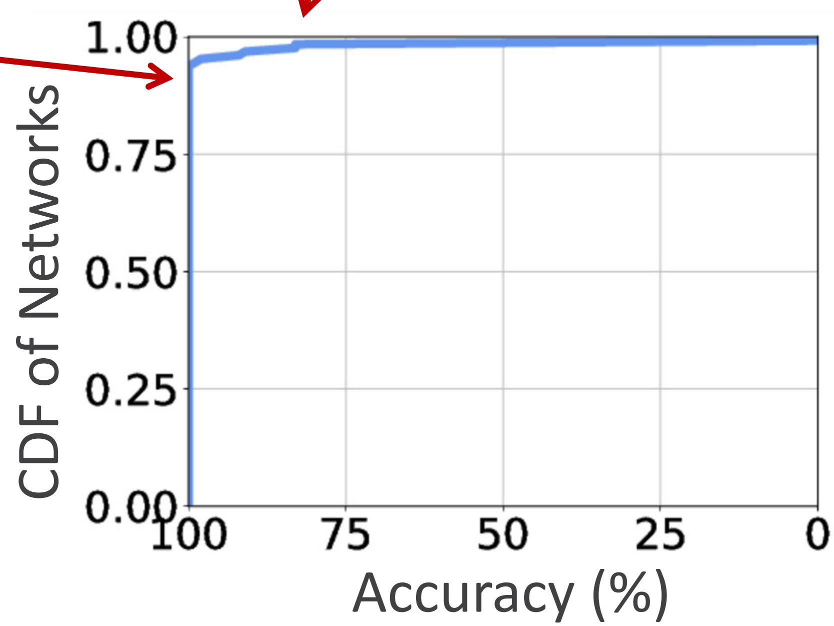


Speedup grows as
network size grows.

Abstraction precision on production networks

For the remaining 5% of networks, can prove reachability for the majority of destinations

Can prove reachability for all destinations for 95% of networks



Wrap-Up

Further Reading

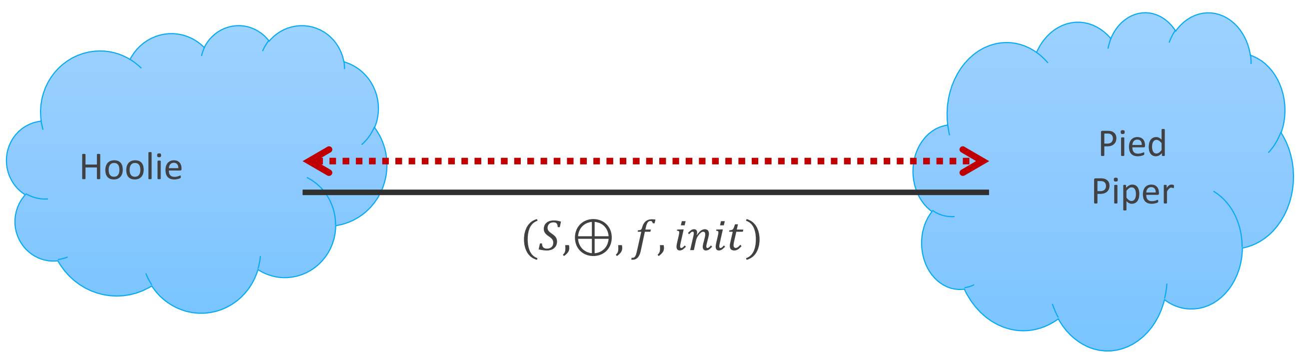
- Stable paths, routing algebras [Griffin et al ToN 2002; Sobrinho ToN 2005]
- Batfish [Fogel et al. NSDI 2015] [batfish.org]
- Network Verification (MineSweeper) [Beckett et al, SIGCOMM 2017]
- Network Abstract Interpretation [Beckett et al, POPL 2020]
- NV [Giannarakis et al, PLDI 2020] [github.com/NetworkVerification]
- Graph-based reasoning (ARC) [Gember-Jacobson et al., SIGCOMM 2016]
- NetVerify.fun – a blog about network verification
- Data plane analysis (HSA, Veriflow, NetKAT, ...) [...]

Conclusions

Network reliability is more important than ever

~2008-2014: Researchers solve the (stateless) data plane verification problem

~2014-2023: Conjecture: Researchers solve the (basic) control plane verification problem



www.github.com/NetworkVerification

Thanks!