SRL
SOFTWARE RELIABILITY LAB

Networked Systems
ETH Zürich

# Programmable Networks with Synthesis



Ahmed ElHassany


Petar Tsankov


Laurent Vanbever


Martin Vechev

# Network Misconfigurations are Common

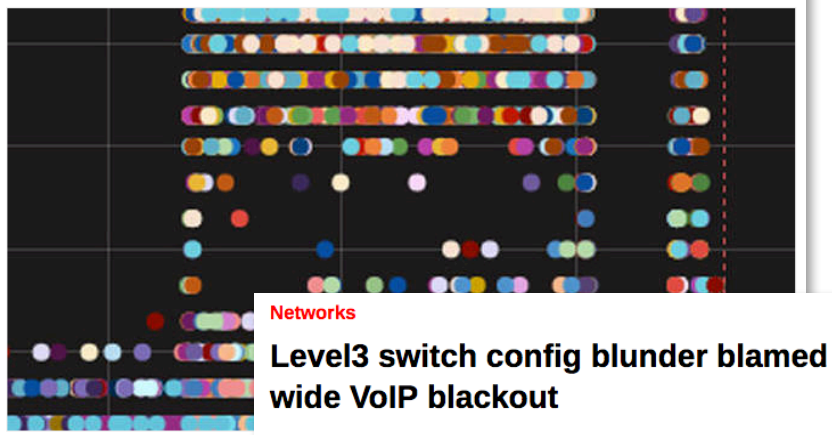
## Amazon server outage affects millions of companies and causes online chaos

MARCH 1, 2017 12:39PM

AMAZON'S giant servers crashed today causing chaos for millions of companies and people that use the cloud service and affected everythin larger web sites to people's smarthomes and even library catalogues.

## CloudFlare apologizes for Telia screwing you over

Unhappy about massive outage



Ouch, th

21 Jun 2016 at 20:34, Kieren Mc

## Amazon's massive AWS outage was caused by human error

One incorrect command and the whole internet suffers.

BY JASON DEL REY | @DELREY | MAR 2, 2017, 2:20PM EST

TWEET SHARE LINKEDIN


Sean Gallup / Getty

Amazon today blamed human error for the the big AWS outage that took down a bunch of large internet sites for several hours on Tuesday afternoon.




## The summer of network misconfigurations

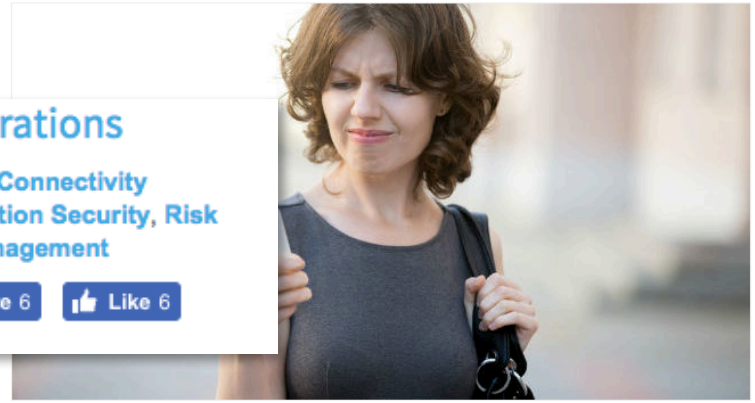
by Joanne Godfrey on August 11, 2016 in Application Connectivity Management, Firewall Change Management, Information Security, Risk Management and Vulnerabilities, Security Policy Management

EMAIL Share 32 Tweet Share 6 Like 6

Networks

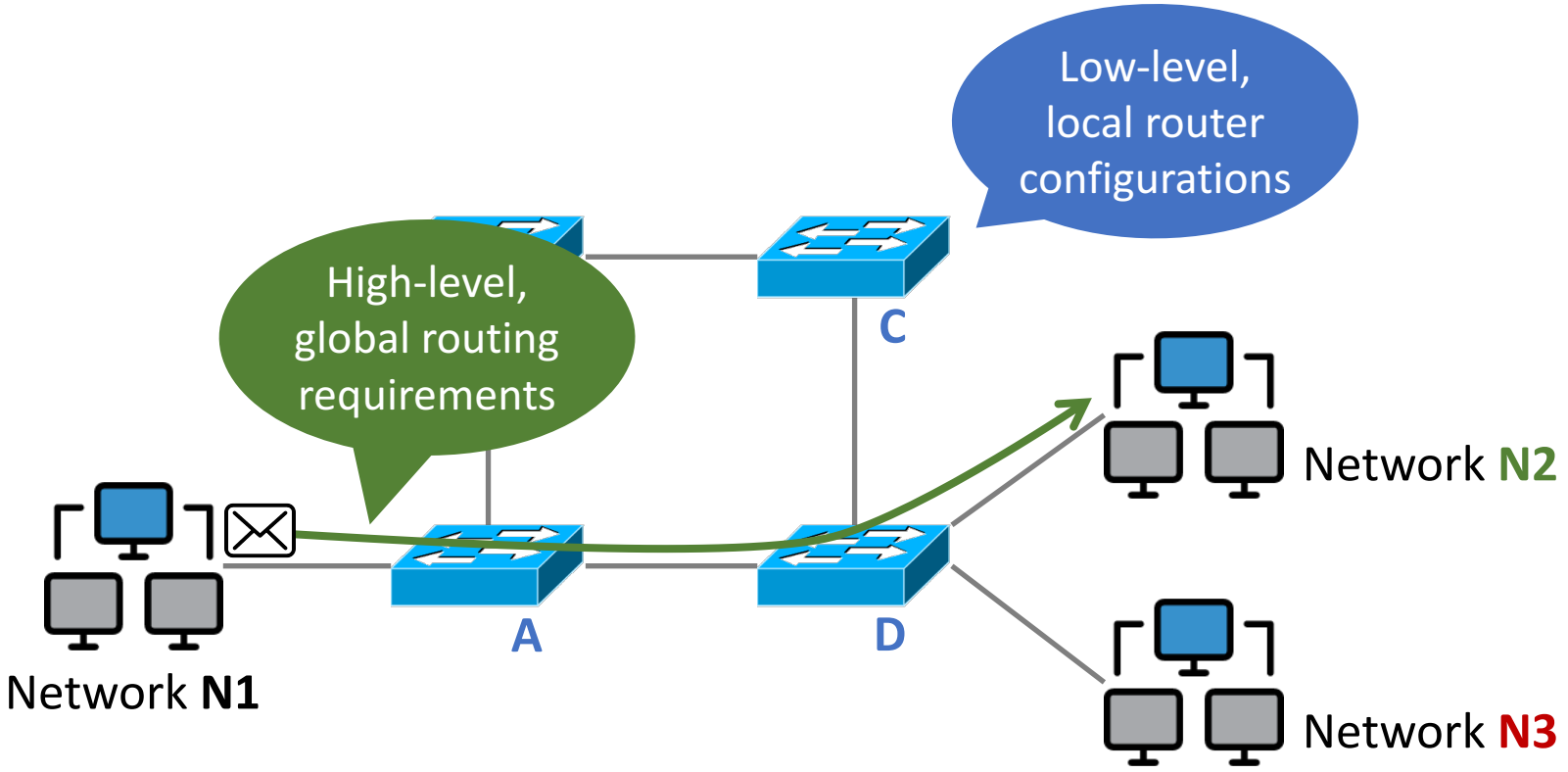
## Level3 switch config blunder blamed for US-wide VoIP blackout

Network dropped calls because it was told to

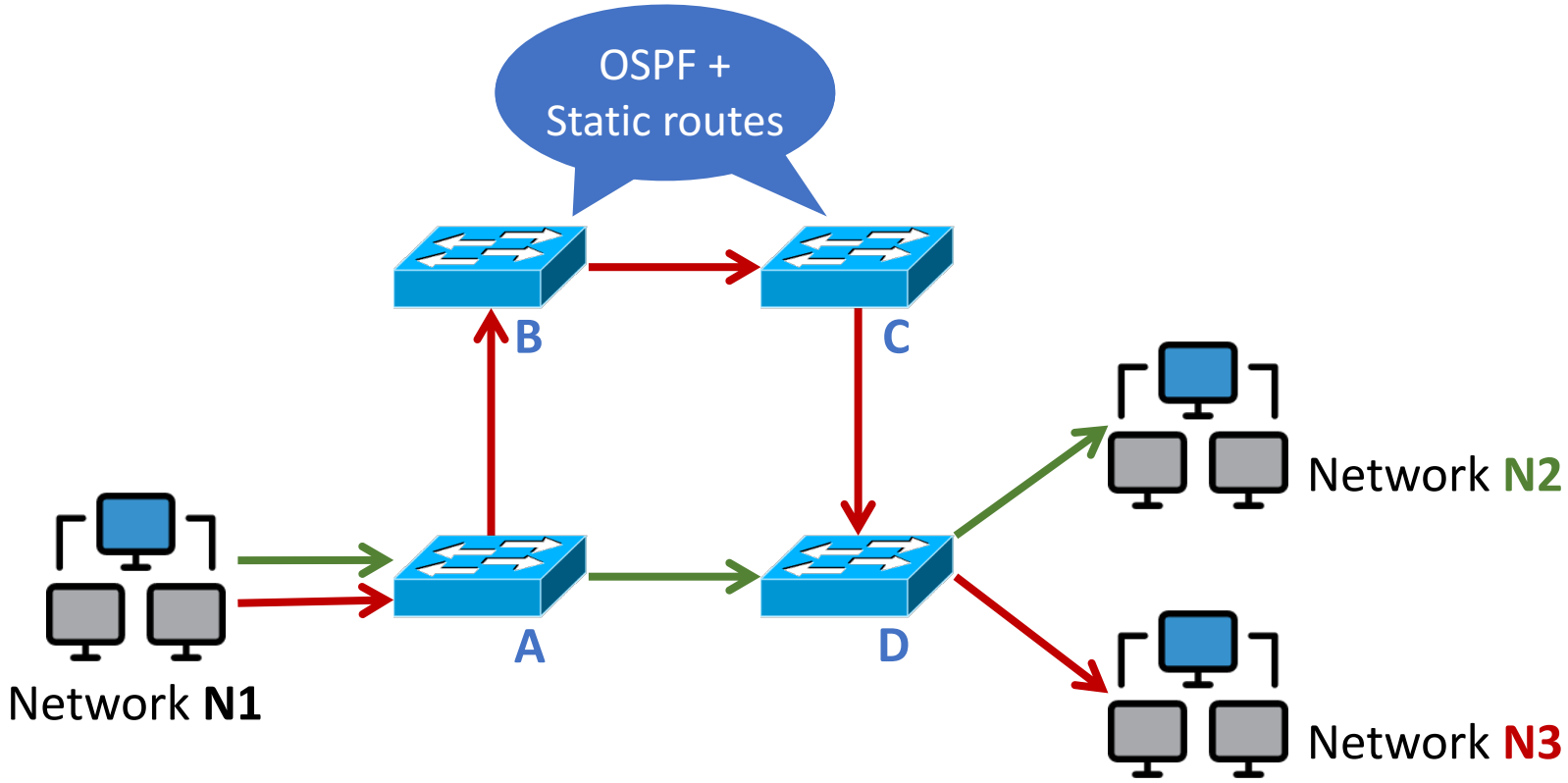


5 Oct 2016 at 11:33, Shaun Nichols

# Example: What Makes Network Configuration Hard?



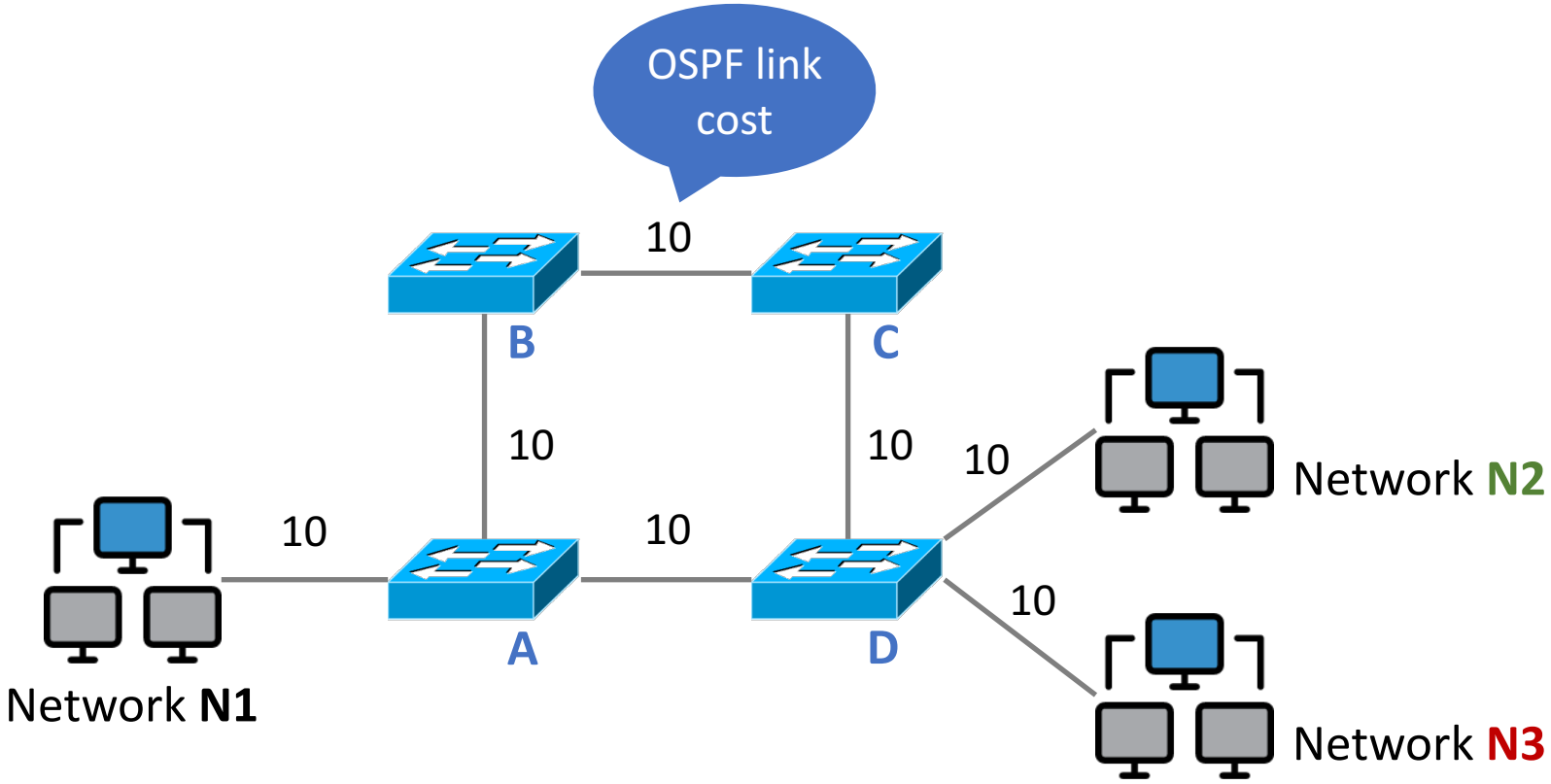
# Example




R1: Packets from **N1** to **N2** must follow the path **A → D**

R2: Packets from **N1** to **N3** must follow the path **A → B → C → D**

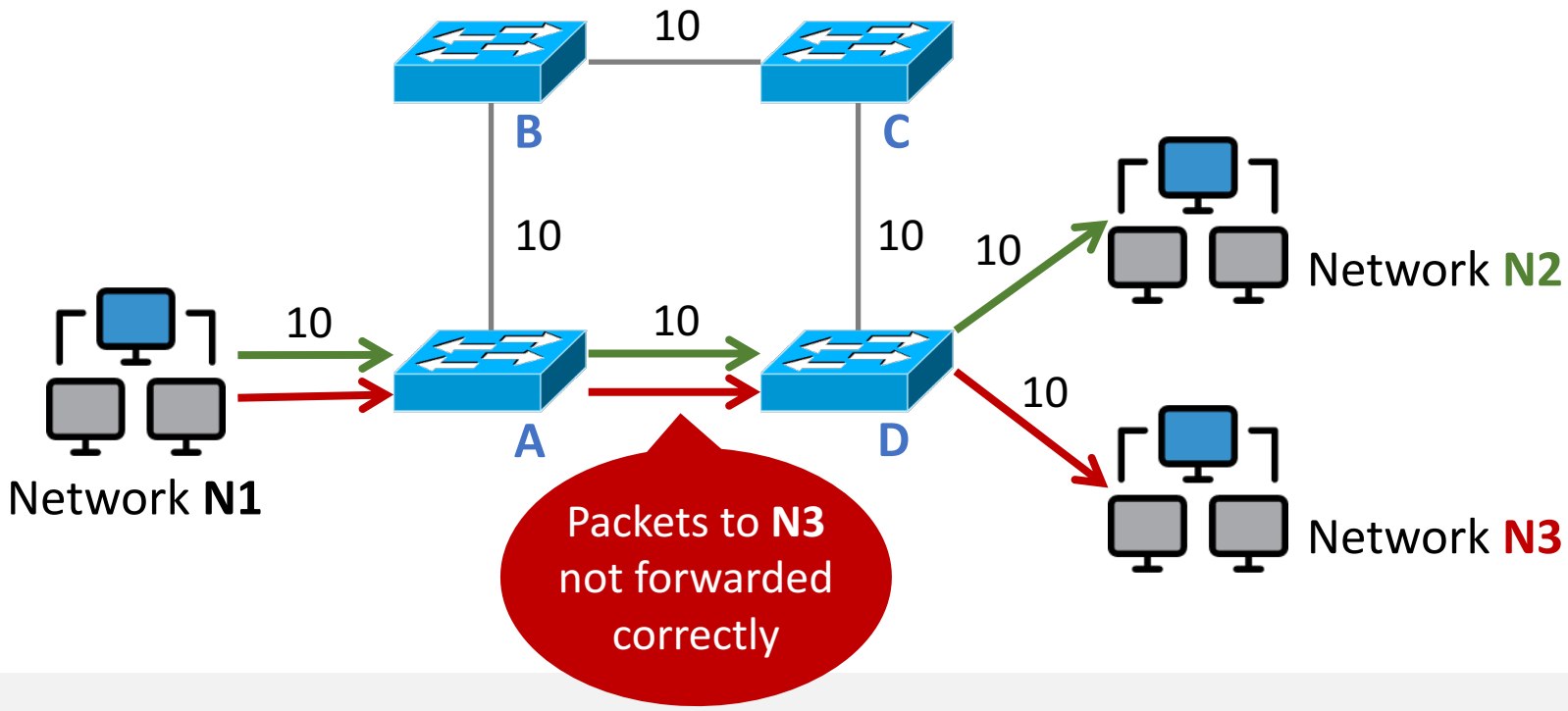
# Example




R1: Packets from **N1** to **N2** must follow the path **A → D**

R2: Packets from **N1** to **N3** must follow the path **A → B → C → D**

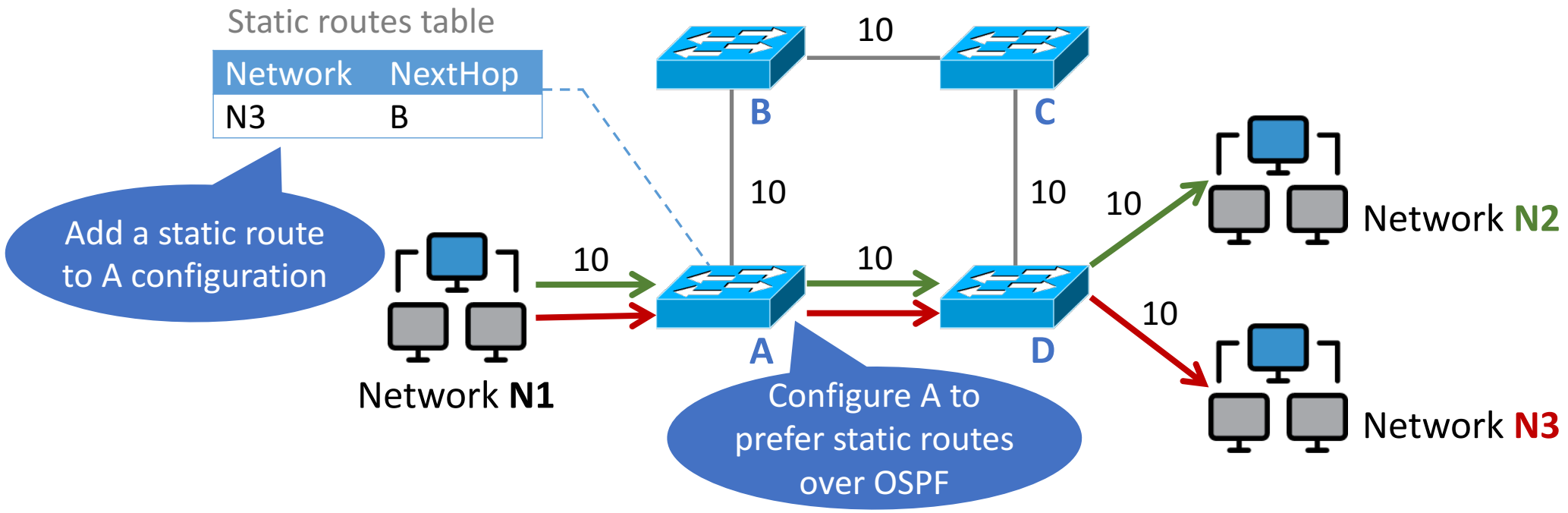
# Example




✔ R1: Packets from **N1** to **N2** must follow the path **A → D**

✘ R2: Packets from **N1** to **N3** must follow the path **A → B → C → D**

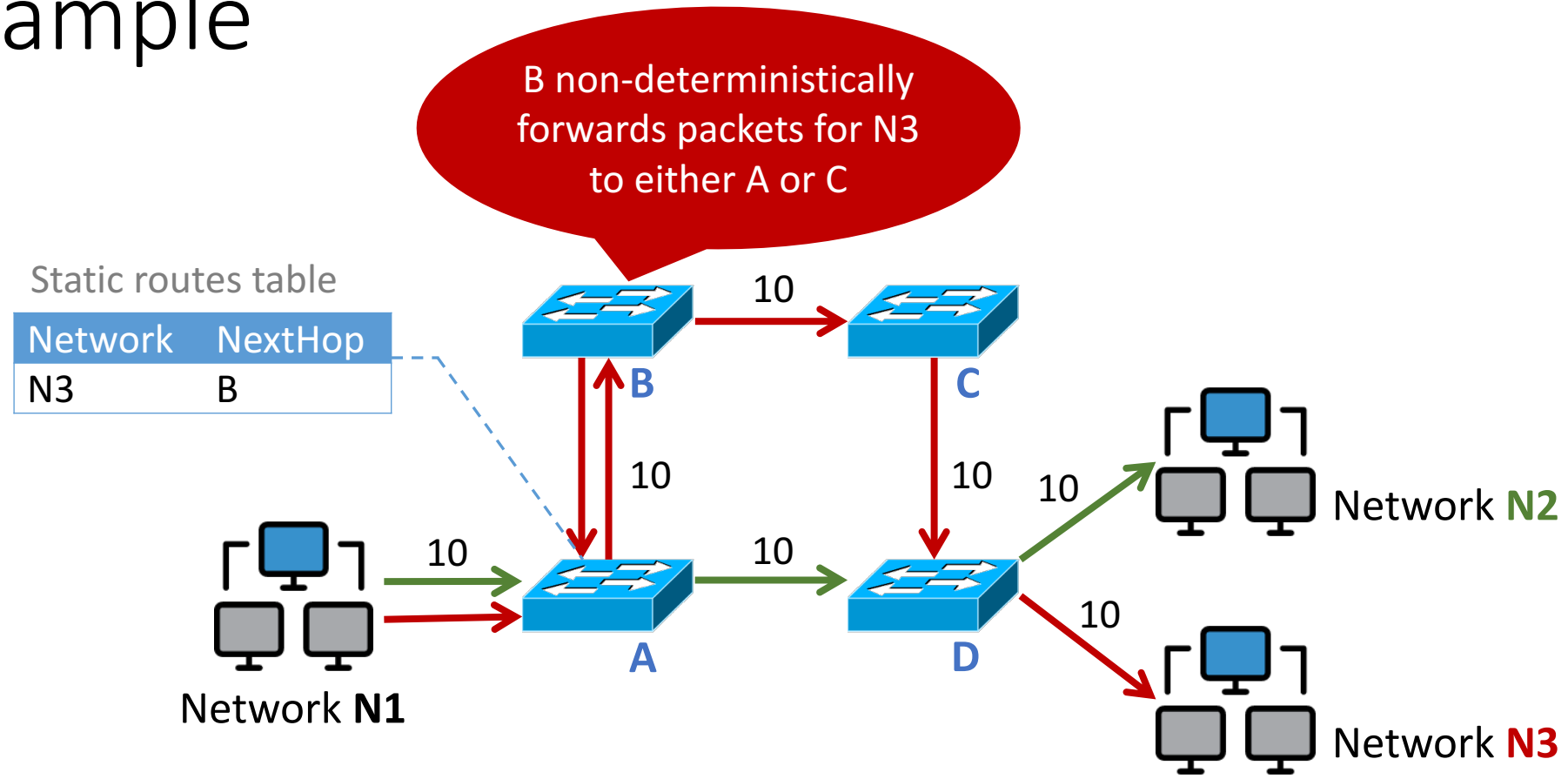
# Example




✔ R1: Packets from **N1** to **N2** must follow the path **A → D**

✘ R2: Packets from **N1** to **N3** must follow the path **A → B → C → D**

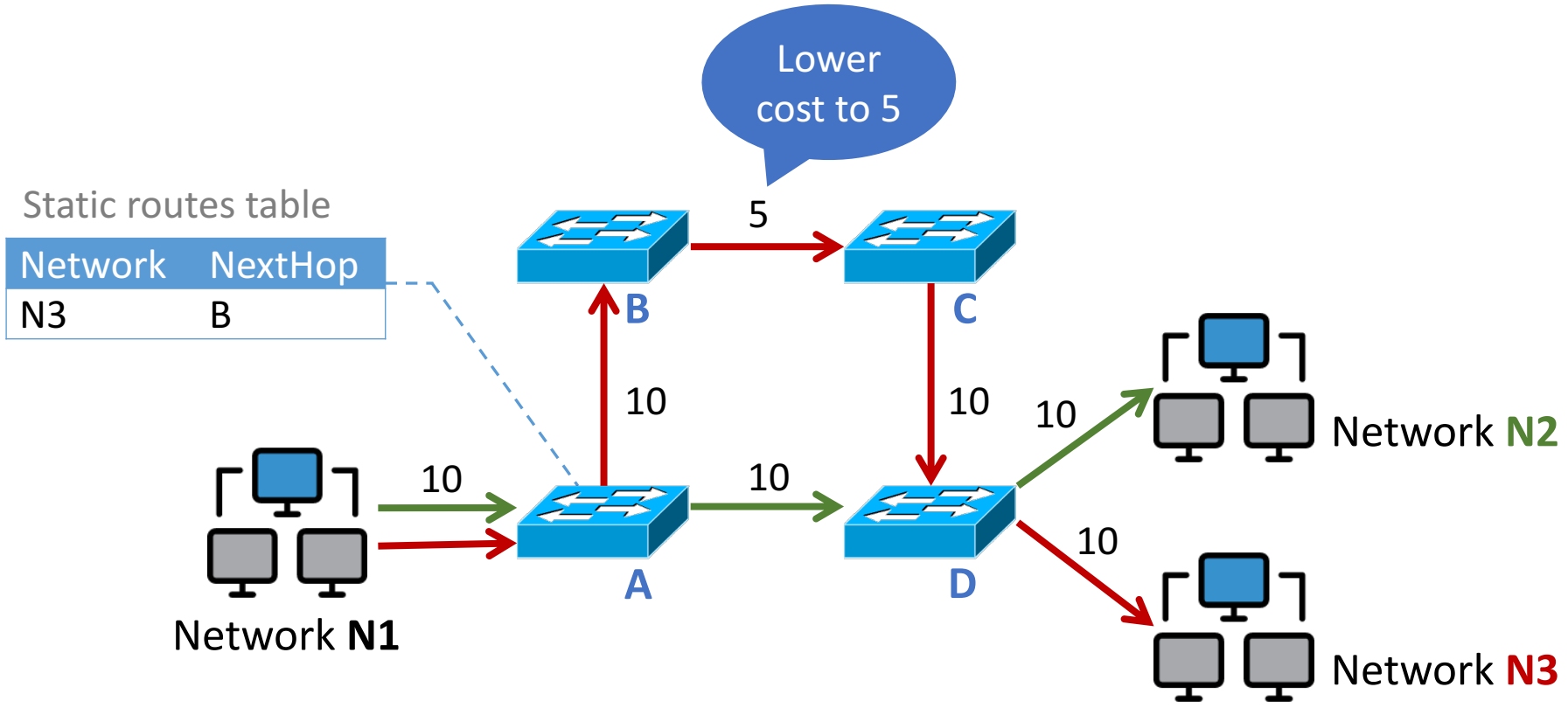
# Example




✔ R1: Packets from **N1** to **N2** must follow the path **A → D**

✘ R2: Packets from **N1** to **N3** must follow the path **A → B → C → D**

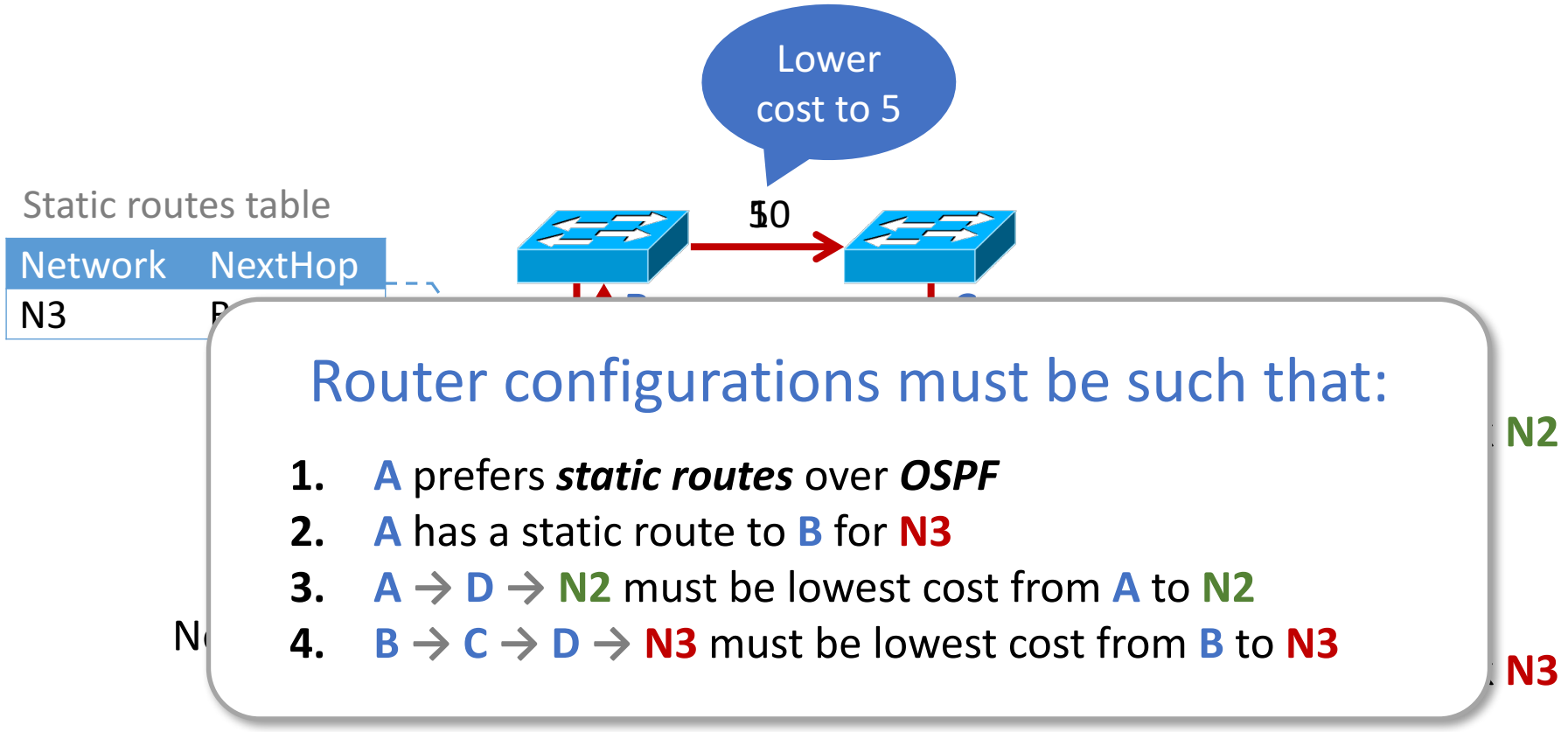
# Example




✔ R1: Packets from **N1** to **N2** must follow the path **A → D**

✘ R2: Packets from **N1** to **N3** must follow the path **A → B → C → D**

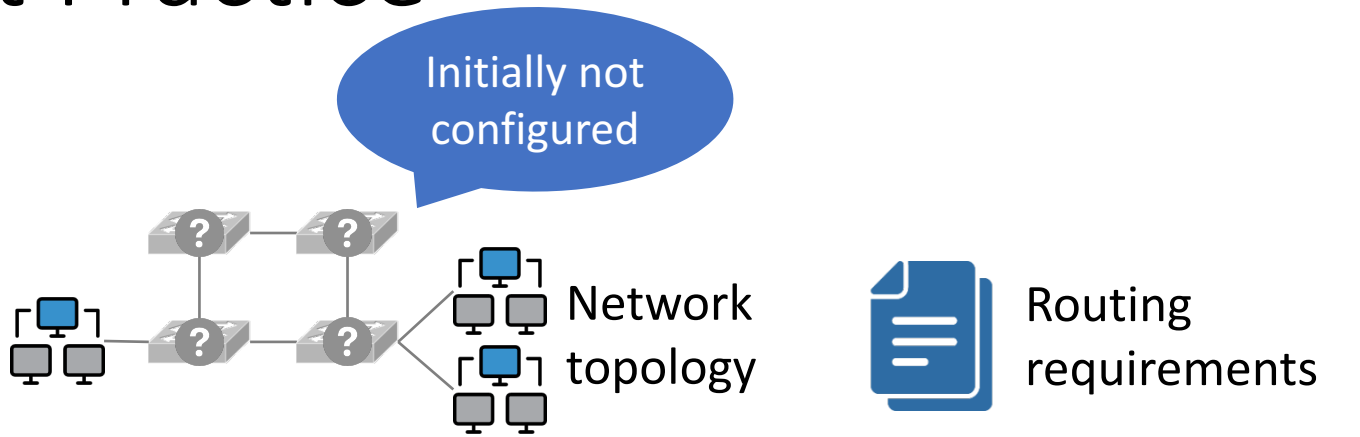
# Example



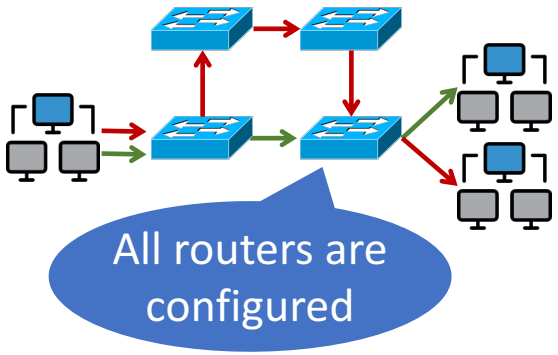

✔ R1: Packets from **N1** to N2 must follow the path A → D

✘ R2: Packets from **N1** to N3 must follow the path A → B → C → D

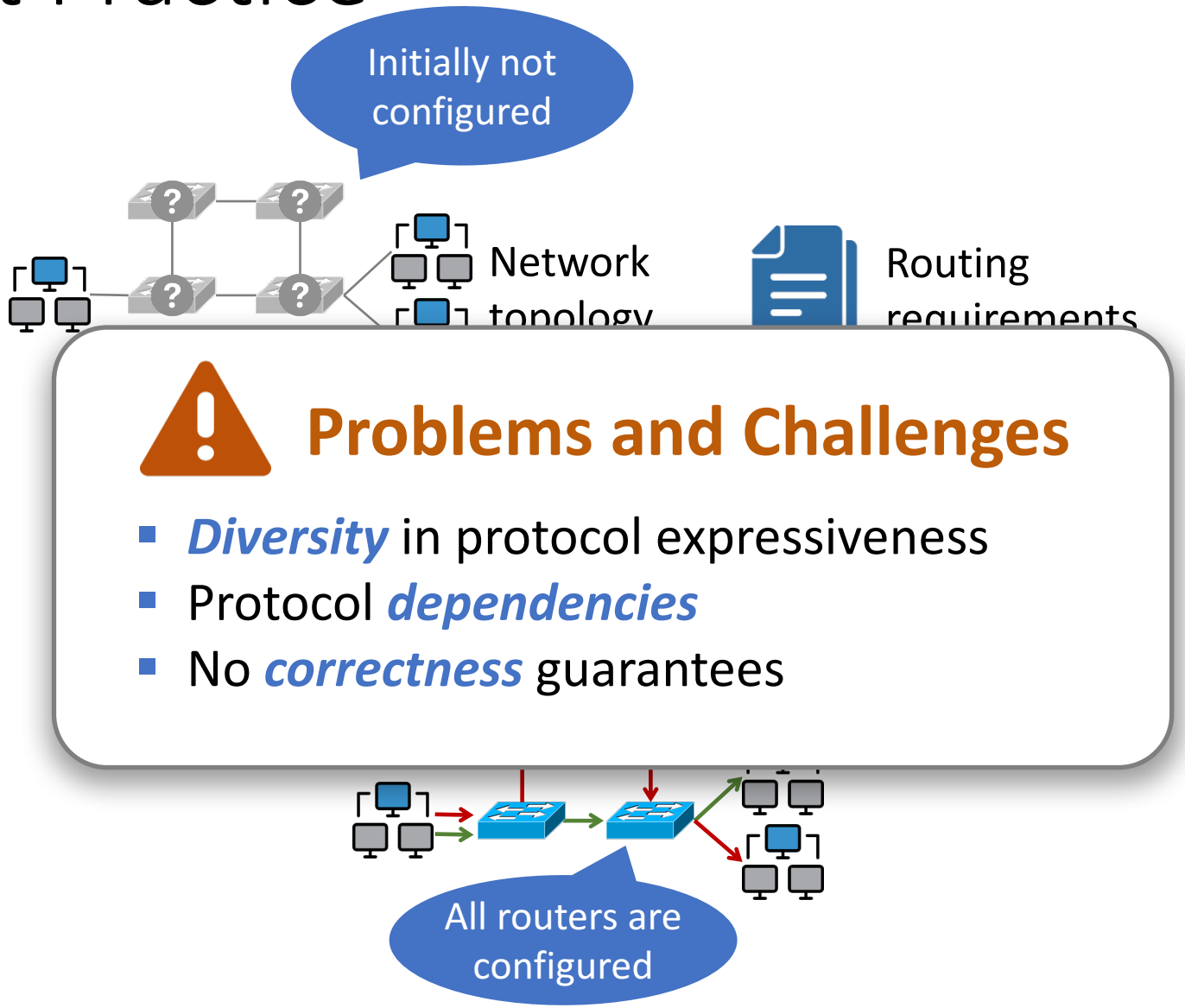
# Current Practice



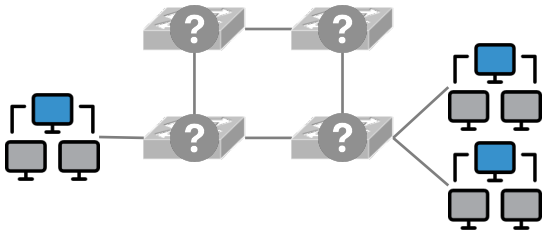




# Current Practice



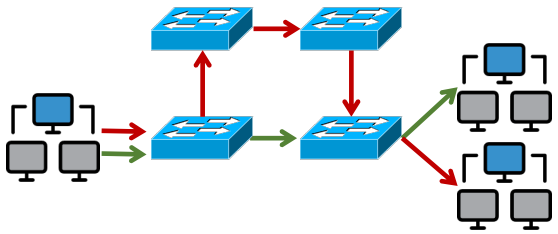
# Wanted: Programmable Networks with Synthesis



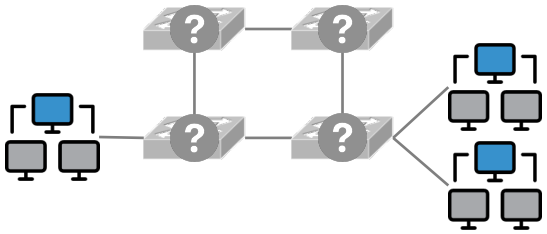
Network topology

Routing requirements

Operators manually configure each router



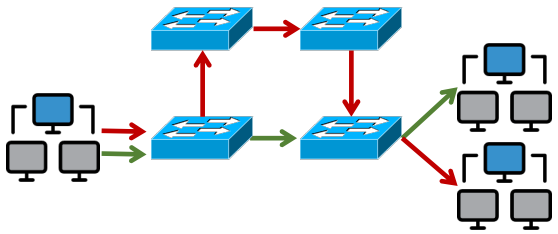
# Wanted: Programmable Networks with Synthesis



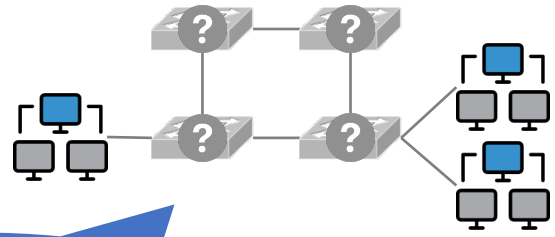
Network topology

Routing requirements

Automatically configure routers with synthesis



# Wanted: Programmable Networks with Synthesis



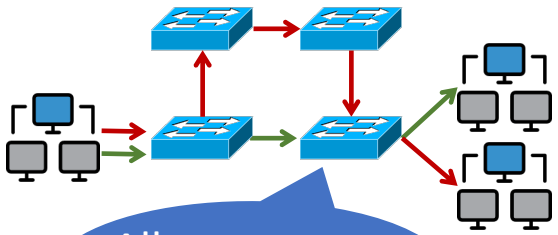
Network topology

Routing requirements

How is the behavior of routers captured?

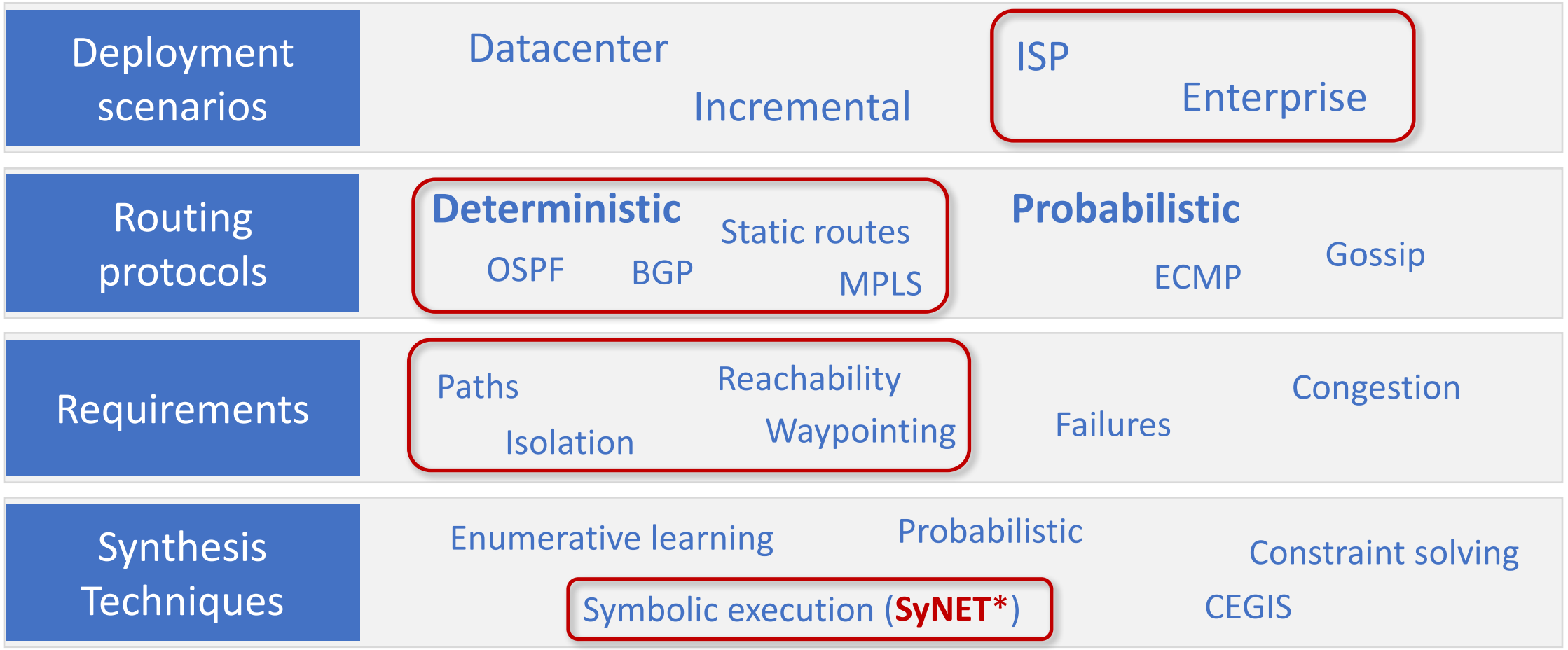
Automatically configure routers with synthesis

How to find a configuration that conforms to the requirements?



All routers are configured

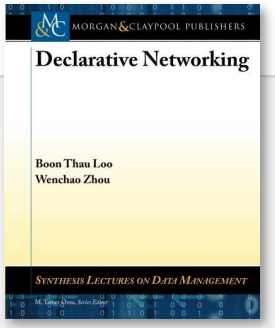
# Programmable Networks with *Synthesis*: Dimensions




***SyNET**: http://synet.ethz.ch

# Capturing Network Behavior

**Key idea:** Express routing protocols, along with their dependencies, in ***stratified Datalog***



# Datalog (Graph Reachability)

**Input**

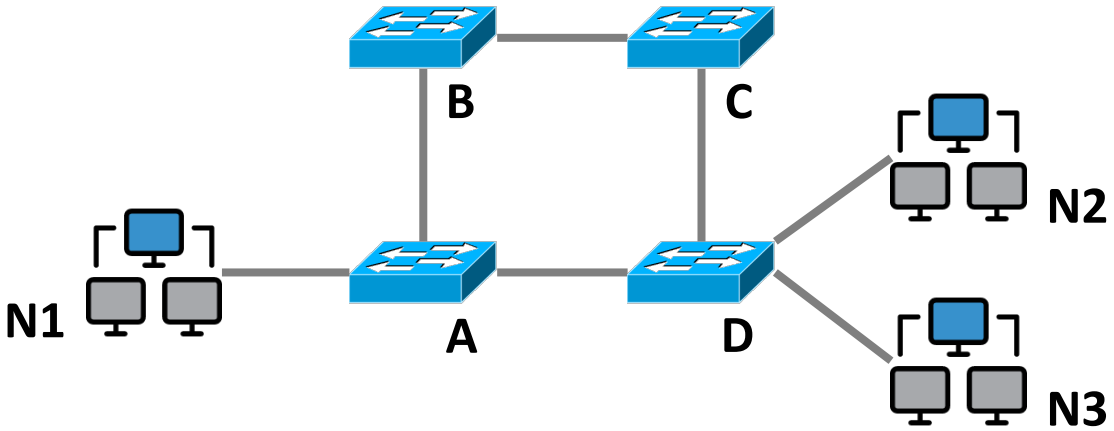
$link(n1, a)$
$link(a, b)$
…

**Program**

$path(X, Y) \leftarrow link(X, Y)$
$path(X, Y) \leftarrow link(X, Z), path(Z, Y)$

**Query**

$path(n1, n2)?$




Can we capture the network's forwarding plane?

# Datalog (Shortest-path Routing)

## Input

Add OSPF cost to links

$link(n_1, a, 10)$

Captures the router configuration

## Program

$path(Router, Net, Net, Cost) \leftarrow$
$\quad link(Router, Net, Cost)$

$path(Router, Net, NextHop, C_1 + C_2) \leftarrow$
$\quad link(Router, NextHop, C_1),$
$\quad path(NextHop, Net, X, C_2)$

$sp(Router, Net, NextHop, min\langle C \rangle) \leftarrow$
$\quad path(Router, Net, NextHop, C\ )$

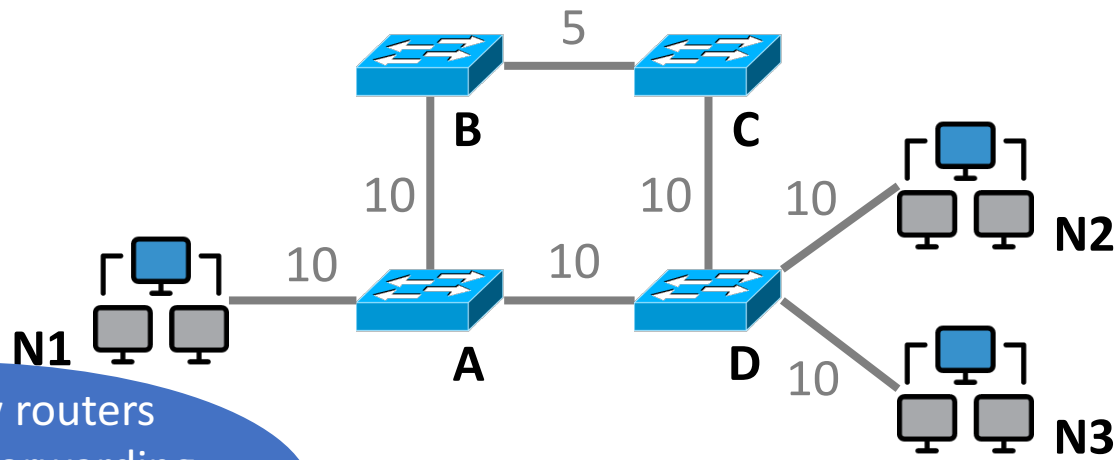
Captures how routers compute their forwarding entries using routing protocols

$fwd(Router, Net, NextHop) \leftarrow$
$\quad sp(Router, Net, NextHop\ , C)$

## Query

Checks a property on the forwarding plane

$fwd(a, n_2, ?\ )$



# Routing Requirements

**Paths**

*Packets for traffic class $TC$ must follow the path $r_1 \rightarrow \cdots \rightarrow r_n$*

$$fwd(r_1, tc, r_2) \wedge \cdots \wedge fwd(r_{n-1}, tc, r_n)$$

# Routing Requirements

**Paths**

*Packets for traffic class $TC$ must follow the path $r_1 \rightarrow \cdots \rightarrow r_n$*

$$fwd(r_1, tc, r_2) \wedge \cdots \wedge fwd(r_{n-1}, tc, r_n)$$

---

**Traffic isolation**

*The paths for two distinct traffic classes $tc_1$ and $tc_2$ do not share links in the same direction*

$$\forall R_1, R_2 . fwd(R_1, tc_1, R_2) \Rightarrow \neg fwd(R_1, tc_2, R_2)$$

# Routing Requirements

**Paths**

*Packets for traffic class $TC$ must follow the path $r_1 \rightarrow \cdots \rightarrow r_n$*

$$fwd(r_1, tc, r_2) \wedge \cdots \wedge fwd(r_{n-1}, tc, r_n)$$

---

**Traffic isolation**

*The paths for two distinct traffic classes $tc_1$ and $tc_2$ do not share links in the same direction*

$$\forall R_1, R_2.\, fwd(R_1, tc_1, R_2) \Rightarrow \neg fwd(R_1, tc_2, R_2)$$

---

**Reachability**

Packets for traffic class $tc$ can reach router $r_2$ from router $r_1$

$$reach(r_1, tc, r_2)$$

# Routing Requirements

**Paths**

*Packets for traffic class $TC$ must follow the path $r_1 \rightarrow \cdots \rightarrow r_n$*

$$fwd(r_1, tc, r_2) \wedge \cdots \wedge fwd(r_{n-1}, tc, r_n)$$

---

**Traffic isolation**

*The paths for two distinct traffic classes $tc_1$ and $tc_2$ do not share links in the same direction*

$$\forall R_1, R_2.\, fwd(R_1, tc_1, R_2) \Rightarrow \neg fwd(R_1, tc_2, R_2)$$

---

**Reachability**

Packets for traffic class $tc$ can reach router $r_2$ from router $r_1$

$$reach(r_1, tc, r_2)$$

---

**Loop-freeness**

The forwarding plane has no loops

$$\forall TC, R.\, \neg reach(R, TC, R)$$

# Network-wide Configuration Synthesis

# Network-wide Configuration Synthesis

**Network specification** $N$
(OSPF, BGP, MPLS, …)

**Routing requirements** $R$
(isolation, reachability, reliability)

Datalog ***program*** $P$

Datalog ***query*** $Q$

**Synthesis problem:**
*Find a configuration* $C$ *such that* $N$ *configured with* $C$ *satisfies* $R$

**(Input) Synthesis problem:**
*Find an input* $I$ *such that* $P, I \models Q$

**Network-wide configuration** $C$
(protocol configurations for routers)

Datalog ***input*** $I$

# Network-wide Configuration Synthesis

**Network specification** $N$
(OSPF, BGP, MPLS, …)

**Routing requirements** $R$
(isolation, reachability, reliability)

Datalog ***program*** $P$

Datalog ***query*** $Q$

**Synthesis problem**
*Find a configuratio*
*configured with* $C$ *s*

**problem:**
*ch that* $P, I \vDash Q$

**Problems:**

- No input synthesis tools for Datalog
- Problem is undecidable

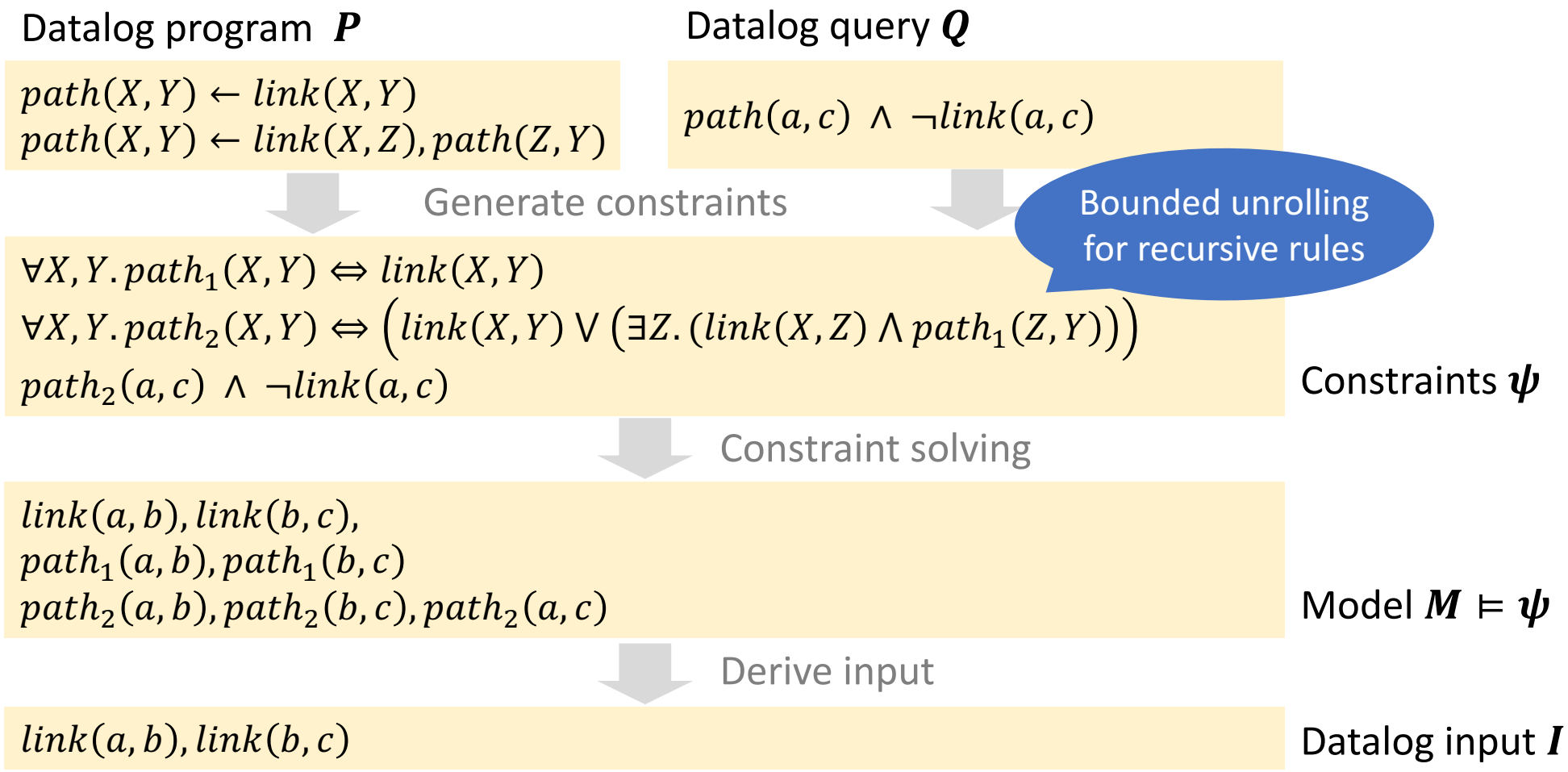
**Network-wide configuration** $C$
(protocol configurations for routers)

Datalog ***input*** $I$

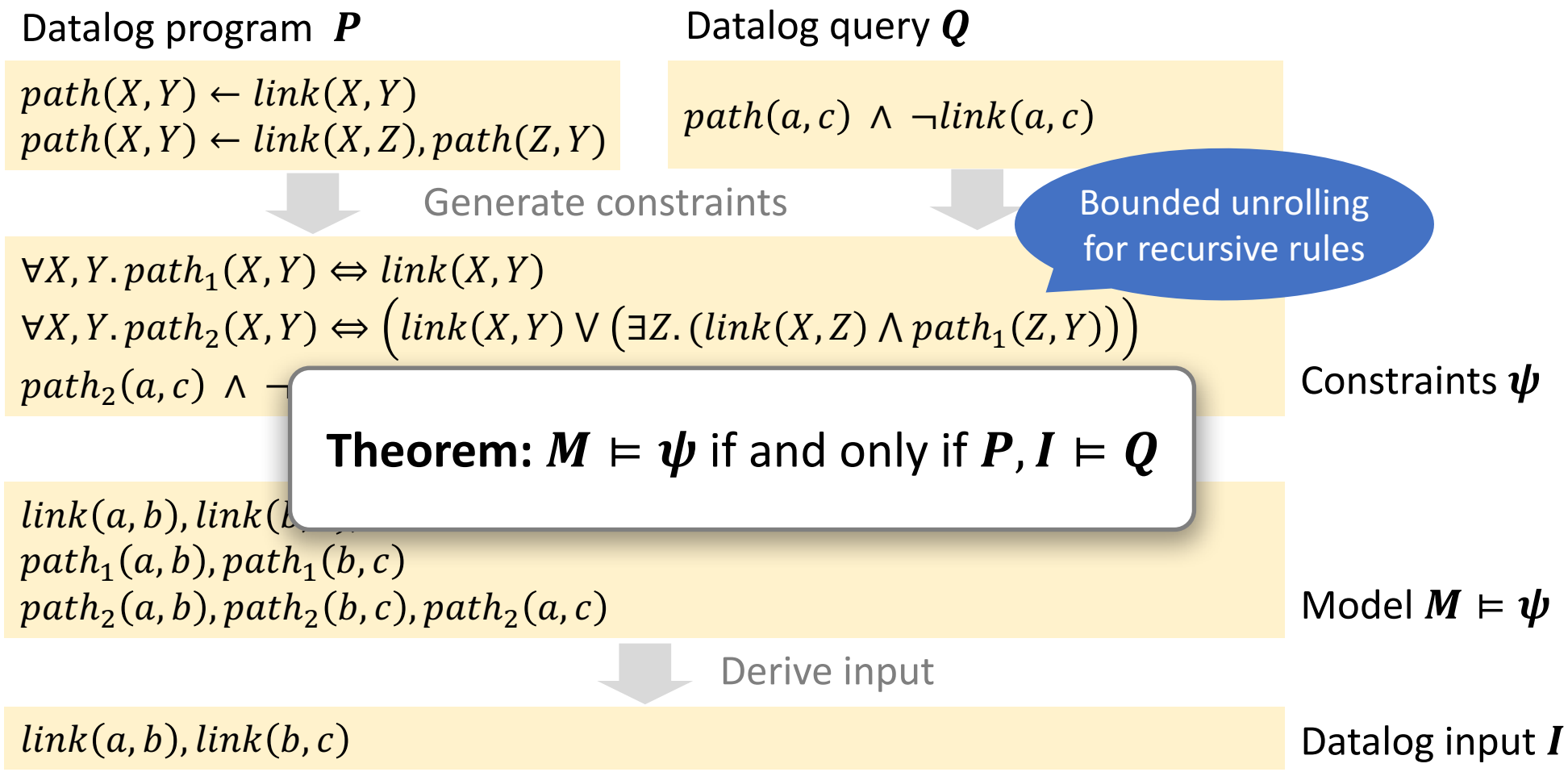
# Input Synthesis for Datalog

**Key idea:** Reduce to solving logical constraints

# Input Synthesis for Datalog (via Constraint Solving)



# Input Synthesis for Datalog (via Constraint Solving)



# Generating Constraints

$link(N1, A, ?)$

...

$path(Router, Net, Net, Cost) \leftarrow$
$link(Router, Net, Cost)$

$path(Router, Net, NextHop, C_1 + C_2) \leftarrow$
$link(Router, NextHop, C_1),$
$path(NextHop, Net, X, C_2)$

$sp(Router, Net, NextHop, min\langle C \rangle) \leftarrow$
$path(Router, Net, NextHop, C)$

$fwd(Router, Net, NextHop) \leftarrow$
$sp(Router, Net, NextHop, C)$

$fwd(a, n_2, d) \wedge fwd(a, n_3, b) \wedge \ldots$



$\forall X, Y. path_1(X, Y) \Leftrightarrow link(X, Y)$
$\forall X, Y. path_2(X, Y)$
$\Leftrightarrow \left(link(X, Y) \vee \left(\exists Z. (link(X, Z) \wedge path_1(Z, Y))\right)\right)$
$path_2(a, c) \wedge \neg link(a, c)$

# Generating Constraints

$link(N1, A, ?)$

...

**Input**

$path(Router, Net, Net, Cost) \leftarrow$
$link(Router, Net, Cost)$

$path(Router, Net, NextHop, C_1 + C_2) \leftarrow$
$link(Router, NextHop, C_1),$
$path(NextHop, Net, X, C_2)$

**Box 1**

$sp(Router, Net, NextHop, min\langle C \rangle) \leftarrow$
$path(Router, Net, NextHop, C)$

**Box 2**

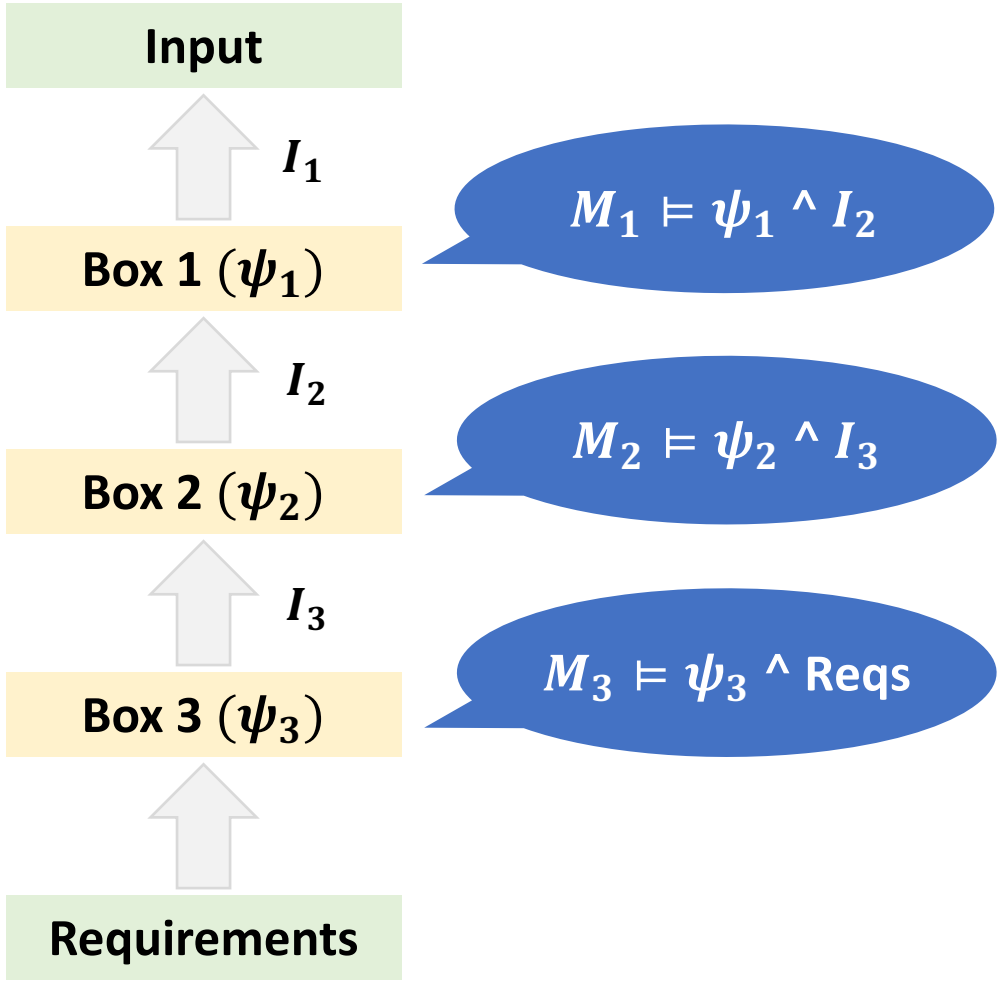
$fwd(Router, Net, NextHop) \leftarrow$
$sp(Router, Net, NextHop, C)$

**Box 3**

$fwd(a, n_2, d) \ \wedge \ fwd(a, n_3, b) \wedge \ ...$

**Requirements**

# Synthesis Algorithm (Naïve Approach)



# Synthesis Algorithm (Naïve Approach)

## Naïve approach does NOT work

1. Not all valid SMT models are valid network configurations.
2. The search space of valid inputs is huge.

# Synthesis Algorithm (Our Approach)

## Input Constraints

*Ensure the synthesized configurations are valid*

$Ex. OSPF\ Link\ costs\ are\ positive\ numbers$

# Synthesis Algorithm (Our Approach)

**Input Constraints**

*Ensure the synthesized configurations are valid*

$Ex. OSPF\ Link\ costs\ are\ positive\ numbers$

---

**Domain specific constraints**

*Reduce the search space*

$Ex. Routers\ only\ forward\ to\ neighbours$

# Synthesis Algorithm (Our Approach)

**Input Constraints**

*Ensure the synthesized configurations are valid*

$Ex. OSPF\ Link\ costs\ are\ positive\ numbers$

---

**Domain specific constraints**

*Reduce the search space*

$Ex. Routers\ only\ forward\ to\ neighbours$

---

**Learn from mistakes**

*Don't reuse the same invalid input*

$\boldsymbol{M_{boxi}} \models \boldsymbol{\psi} \wedge \mathbf{I_{i+1}} \wedge \neg \mathbf{I`_i}$

# Synthesis Algorithm (Our Approach)

**Input Constraints**

*Ensure the synthesized configurations are valid*

$Ex. OSPF\ Link\ costs\ are\ positive\ numbers$

---

**Domain specific constraints**

*Reduce the search space*

$Ex. Routers\ only\ forward\ to\ neighbours$

---

**Learn from mistakes**

*Don't reuse the same invalid input*

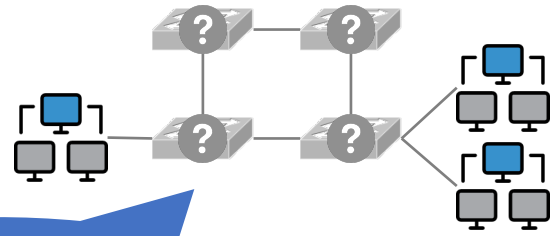
$M_{boxi} \models \psi \wedge I_{i+1} \wedge \neg I`_{i}$

---

**Partial evaluation**

*Don't compute already known values*

# Network-wide Configuration Synthesis



Network topology

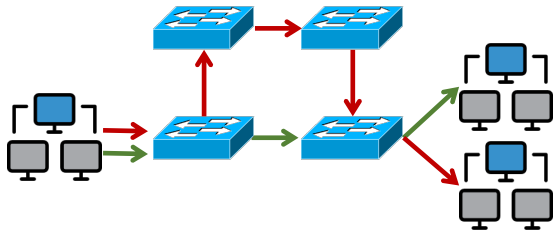
Routing requirements

Constraints on the forwarding plane computed by the routers

Encode as a Datalog program

Automatically configure routers with synthesis

Via reduction to input synthesis for Datalog



Datalog input identifies correct configuration

# Implementation

# Implementation

## The SyNET system (http://synet.ethz.ch)

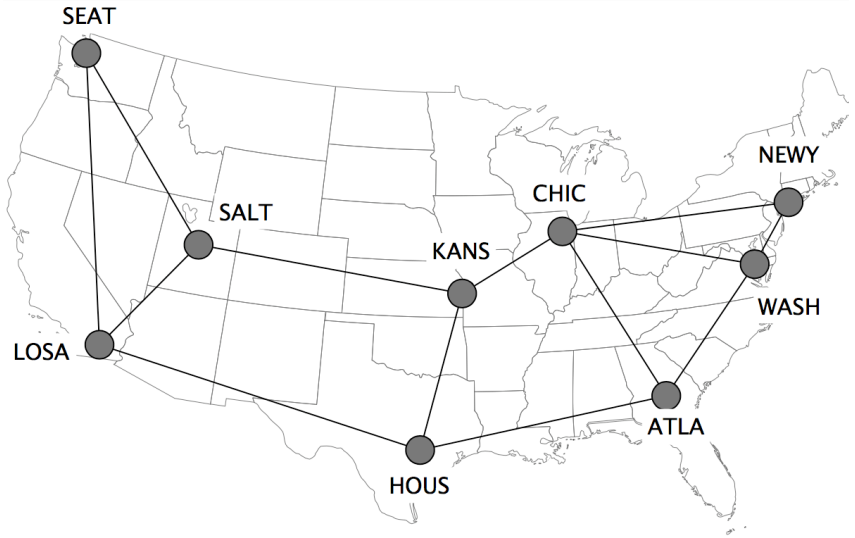
- Written in **Python** ($\approx 4K$ lines of code)
- Protocols encoded in **stratified Datalog** ($\approx 100$ rules)
- Uses the **Z3** constraint solver
- Outputs **CISCO** configurations
- Supports **BGP**, **OSPF**, and **static routes**

```
! A snippet from router A
interface f0/1
   ip address 10.0.0.2 255.255.255.254
   ip ospf cost 10
   description "To B"
interface f0/0
   ip address 10.0.0.0 255.255.255.254
   ip ospf cost 65530
   description "To C"
interface f1/0
   ip address 10.0.0.4 255.255.255.254
   ip ospf cost 65530
   description "To D"
!
```

**CISCO** configuration output by SyNET

# Experiments

# Experiment



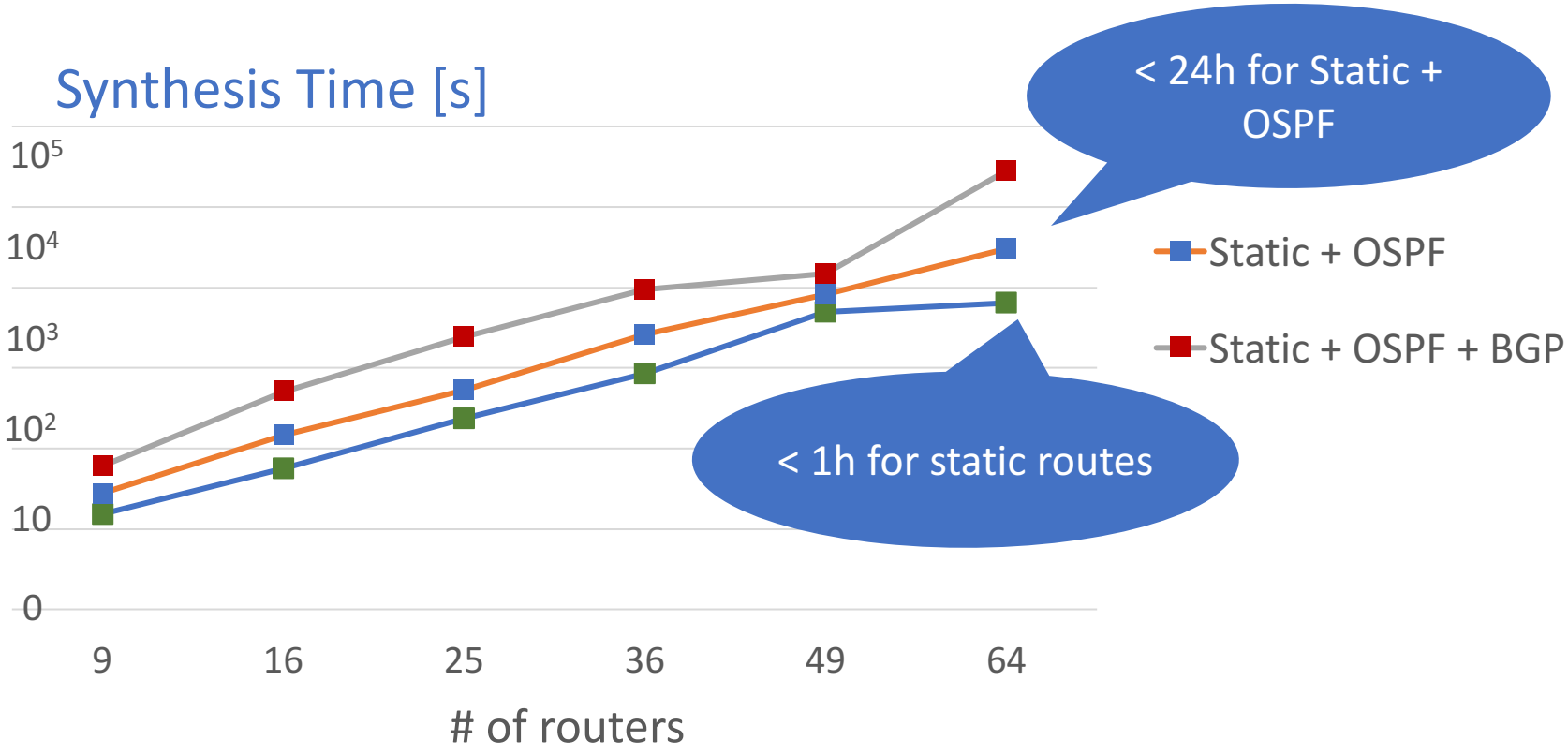

US-based network connecting major universities and research institutes

| Protocols / # Traffic classes | 1 class | 5 classes | 10 classes |
|---|---|---|---|
| Static | 1.3s | 2.0s | 4.0s |
| Static + OSPF | 9.0s | 21.3s | 49.3s |
| Static + OSPF + BGP | 13.3s | 22.7s | 1m19.7s |

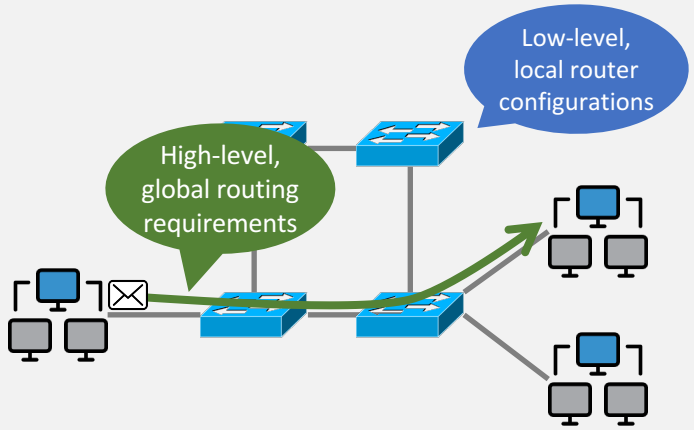
Synthesis Times

# Scalability Experiment

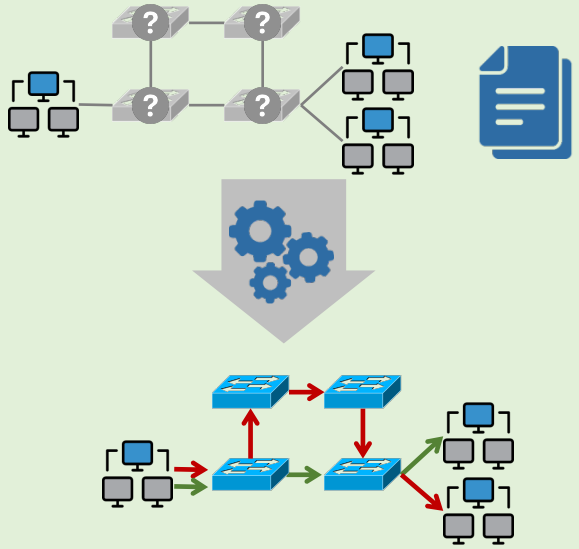
- Grid topologies with up to 64 routers
- Requirements for 10 traffic classes



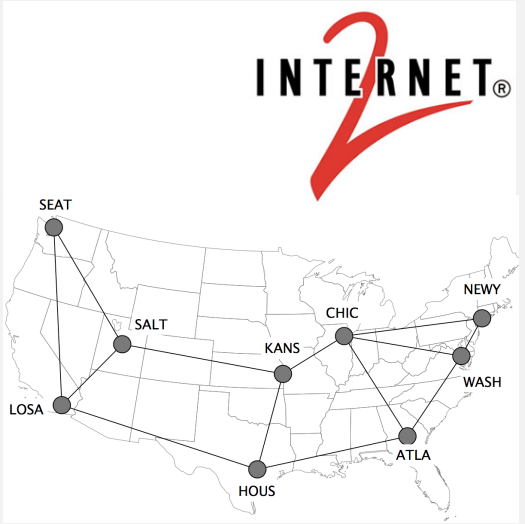
# Summary: Programmable Networks with Synthesis




**Global requirements vs local configurations**



**Network-wide configuration synthesis**




**Approach scales to realistic problems**