Probabilistic Network Programming

Nate Foster
Cornell

Plan

I: Introduction

- Semantics Primer
- Software-Defined Networking

II: NetKAT

- Language Design
- Formal Semantics

III: Probabilistic NetKAT

- Formal Semantics
- Applications

Quiz

What does *this* program do?

```
n := 0
while !(x = 0)
do
  n := n + 1;
  (x := 0 ⊕ x := 1);
```

Quiz

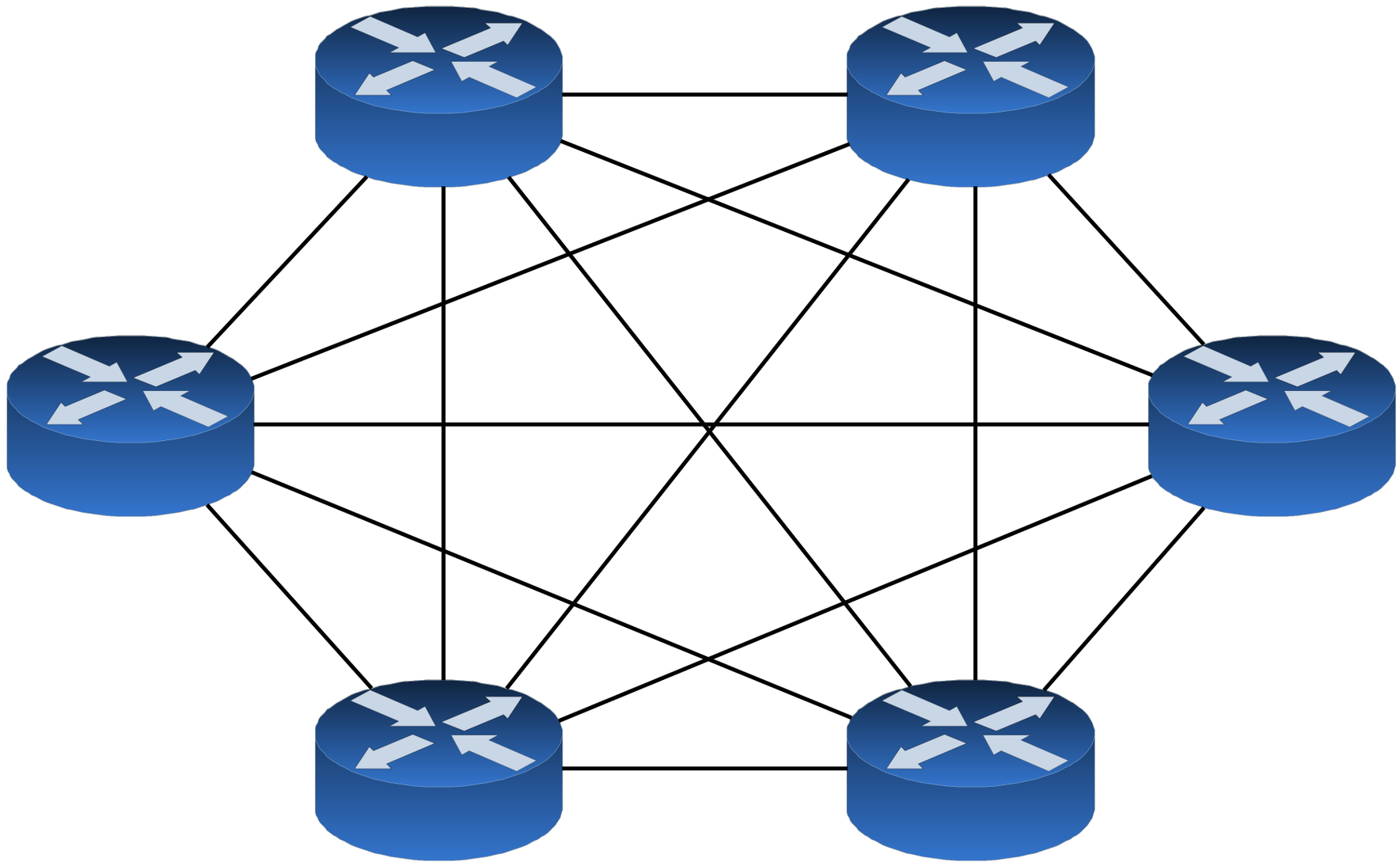
What does *this* program do?

```
n := 0
while !(x = 0)
do
  n := n + 1;
  (x := 0 ⊕ x := 1);
```

Random Choice

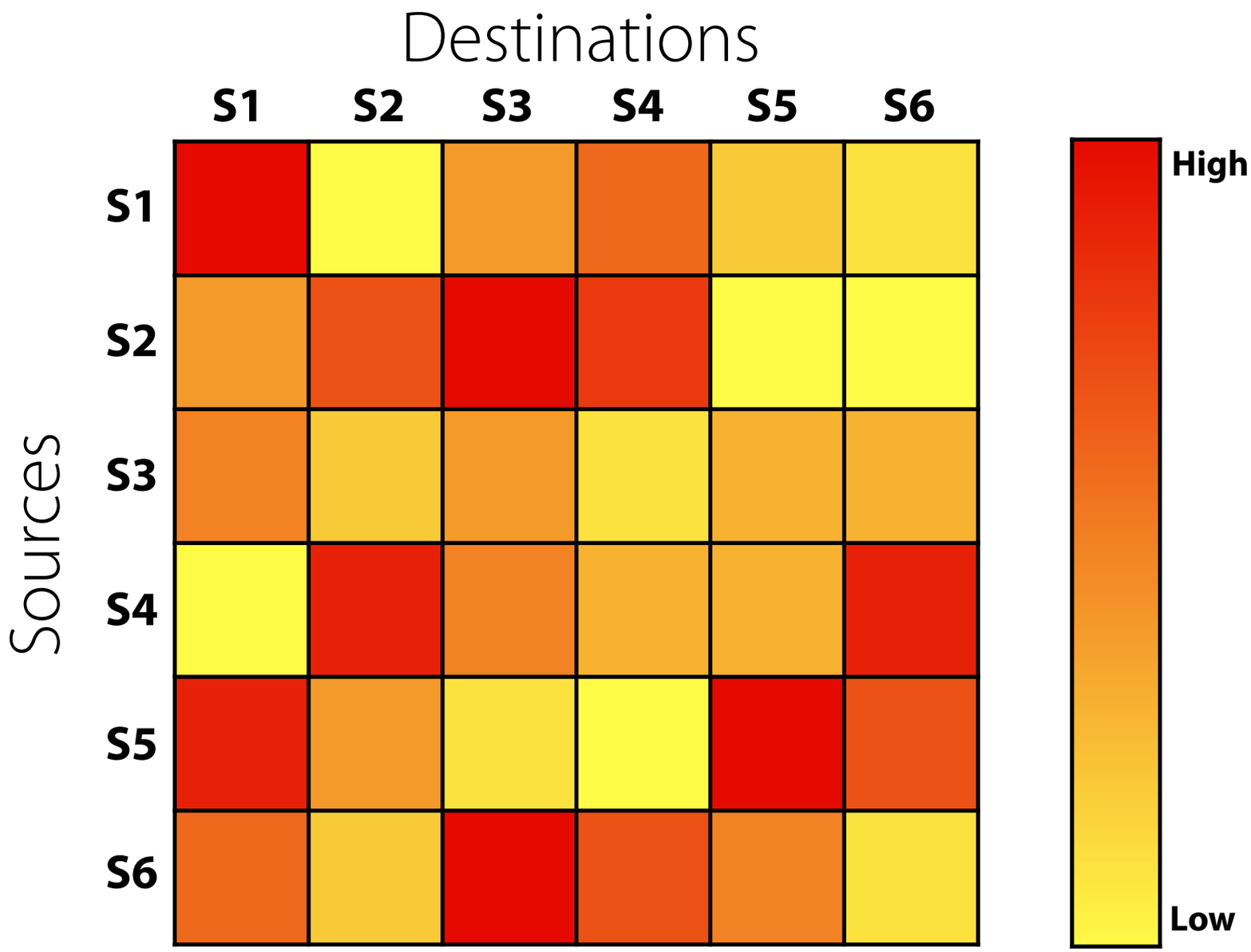
Randomized Routing

Topology



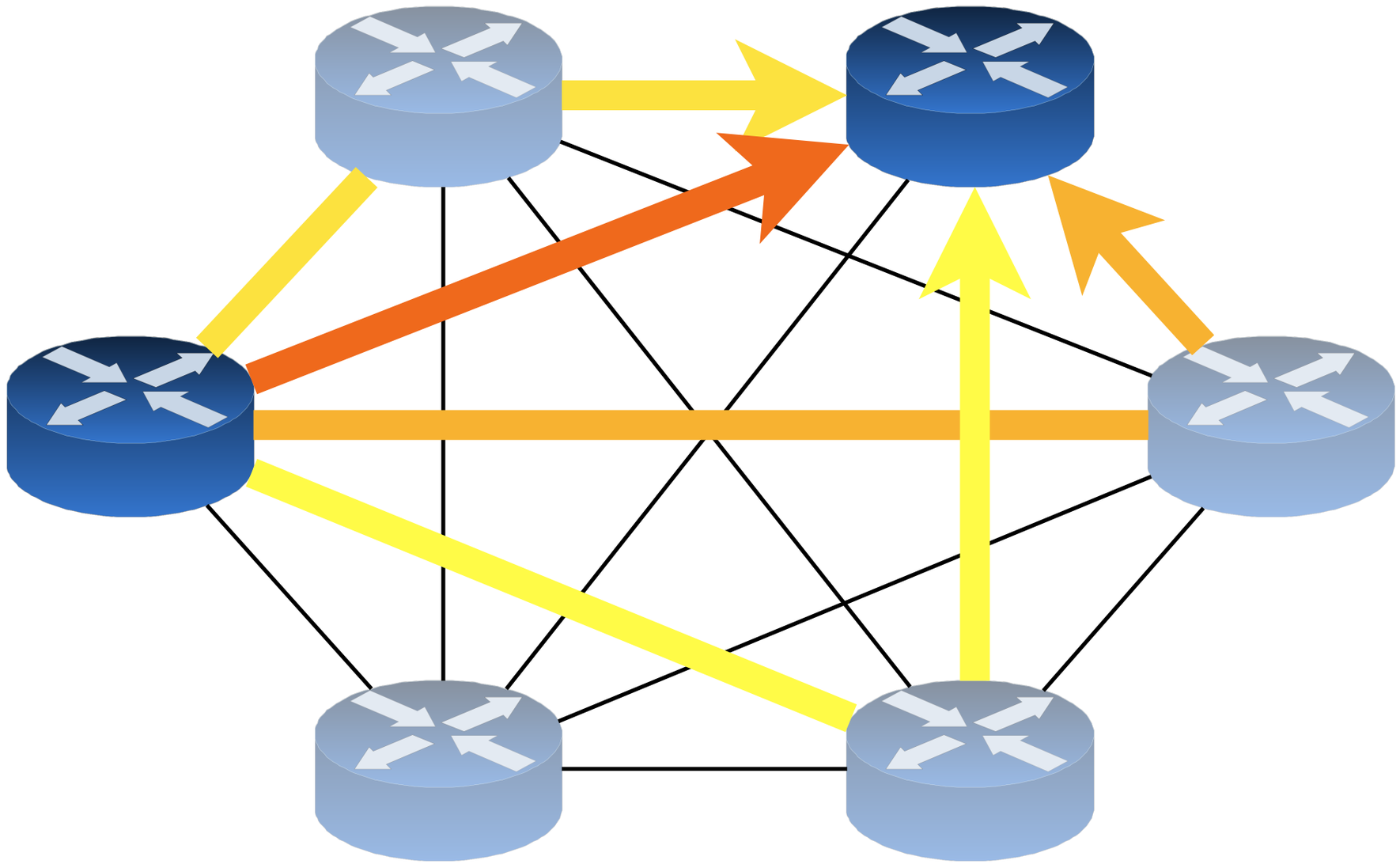
Assume each node can send and receive traffic at rate **r**

Demand Matrix



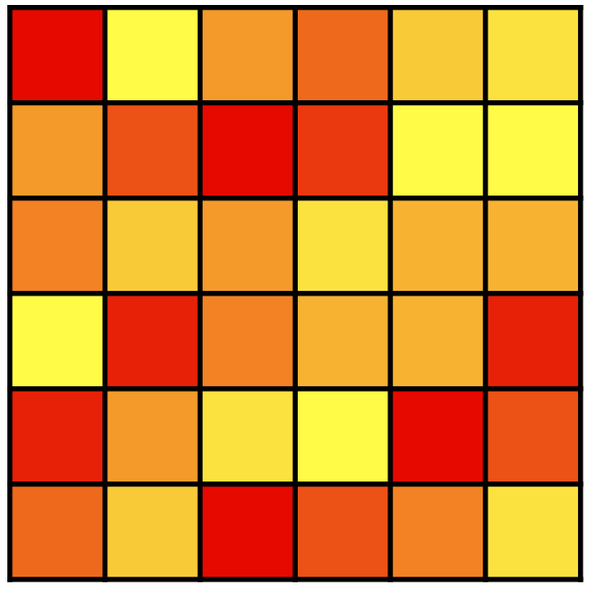
The D_{ij} represents the demand from S_i to S_j

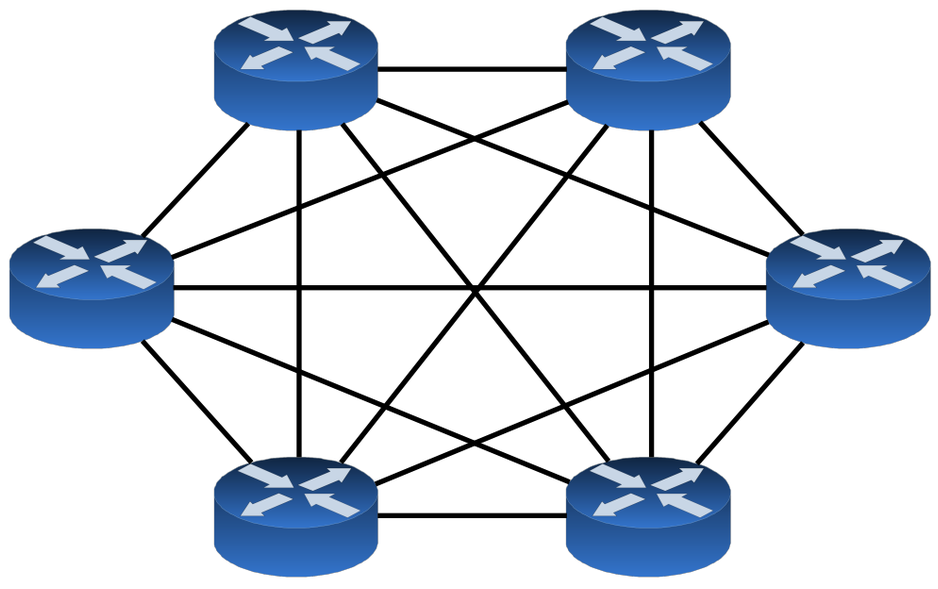
Routing Scheme



Assigns the demand to paths while respecting link capacities

Demand Matrix

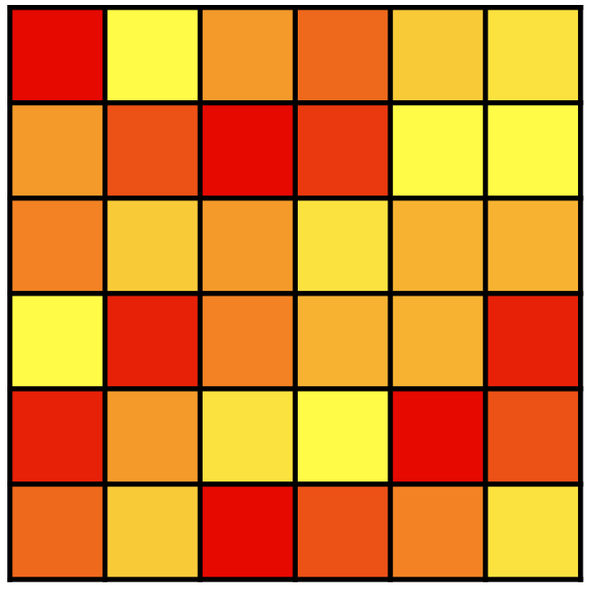


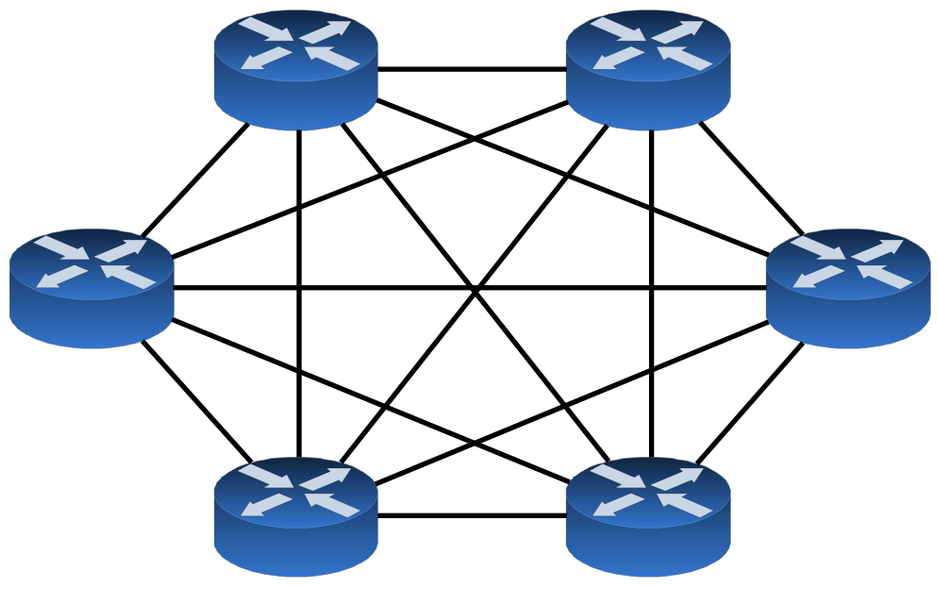


Restrict attention to *feasible* demand matrices

- Send: $\mathbf{\Sigma_i D_{ji} < r}$
- Receive: $\mathbf{\Sigma_j D_{ij} < r}$

Demand Matrix



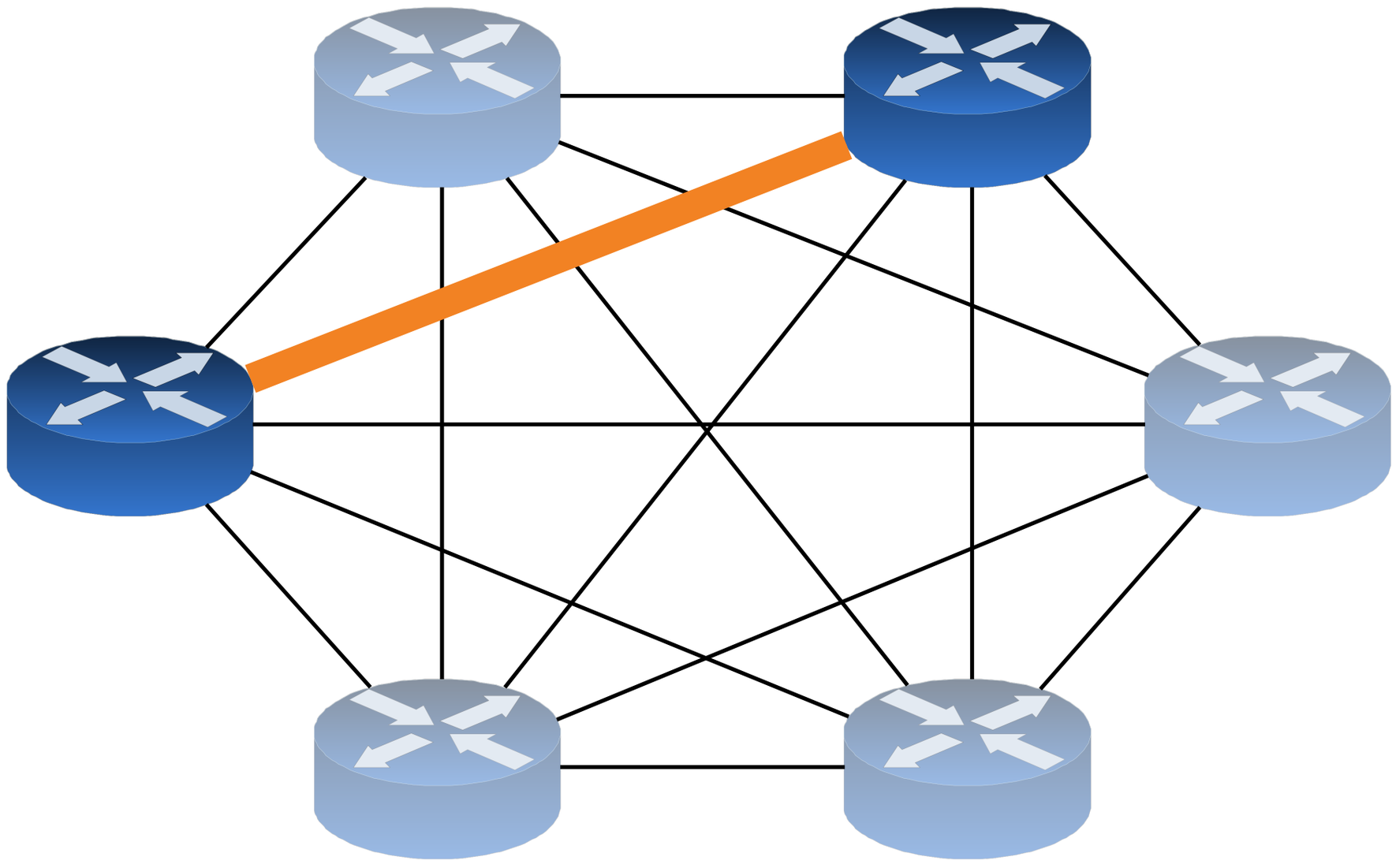


Restrict attention to *feasible* demand matrices

- Send: $\sum_i D_{ji} < r$
- Receive: $\sum_j D_{ij} < r$

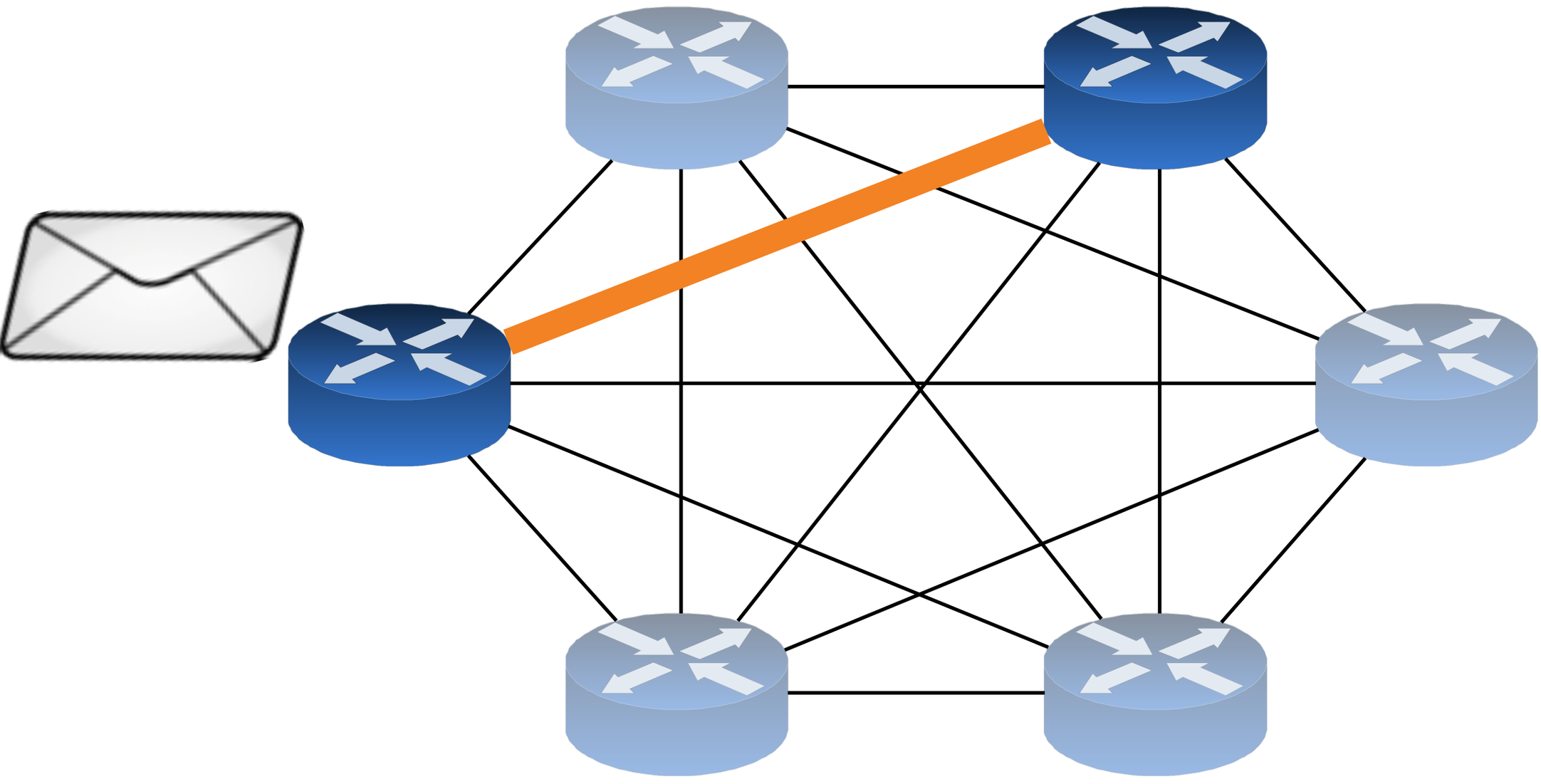
Theorem [Keslassy '05]. Mesh with capacities **2r/N** is optimal design that routes all feasible demand matrices

Shortest-path Routing



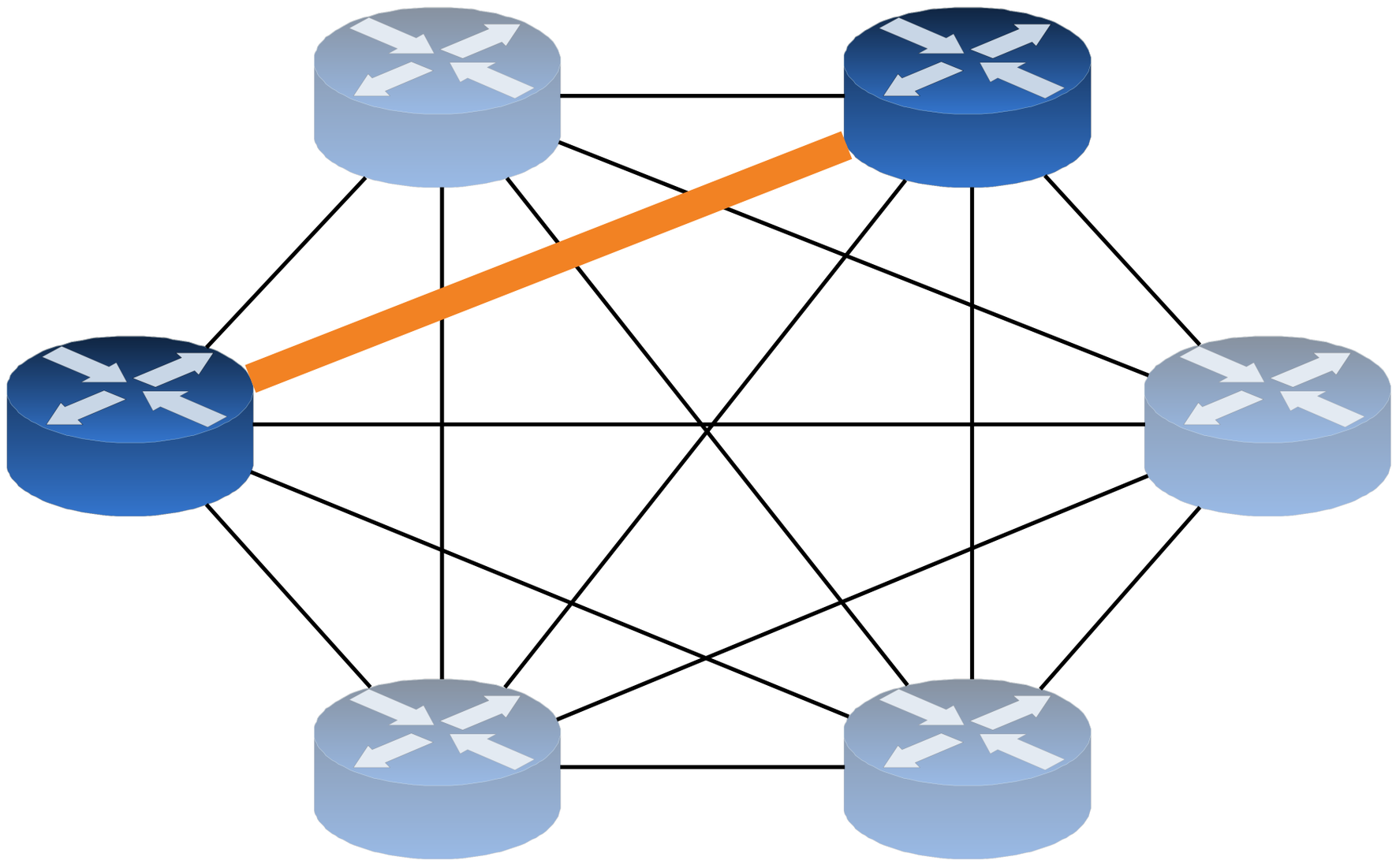
Shortest-path routing cannot route all feasible demands.

Shortest-path Routing



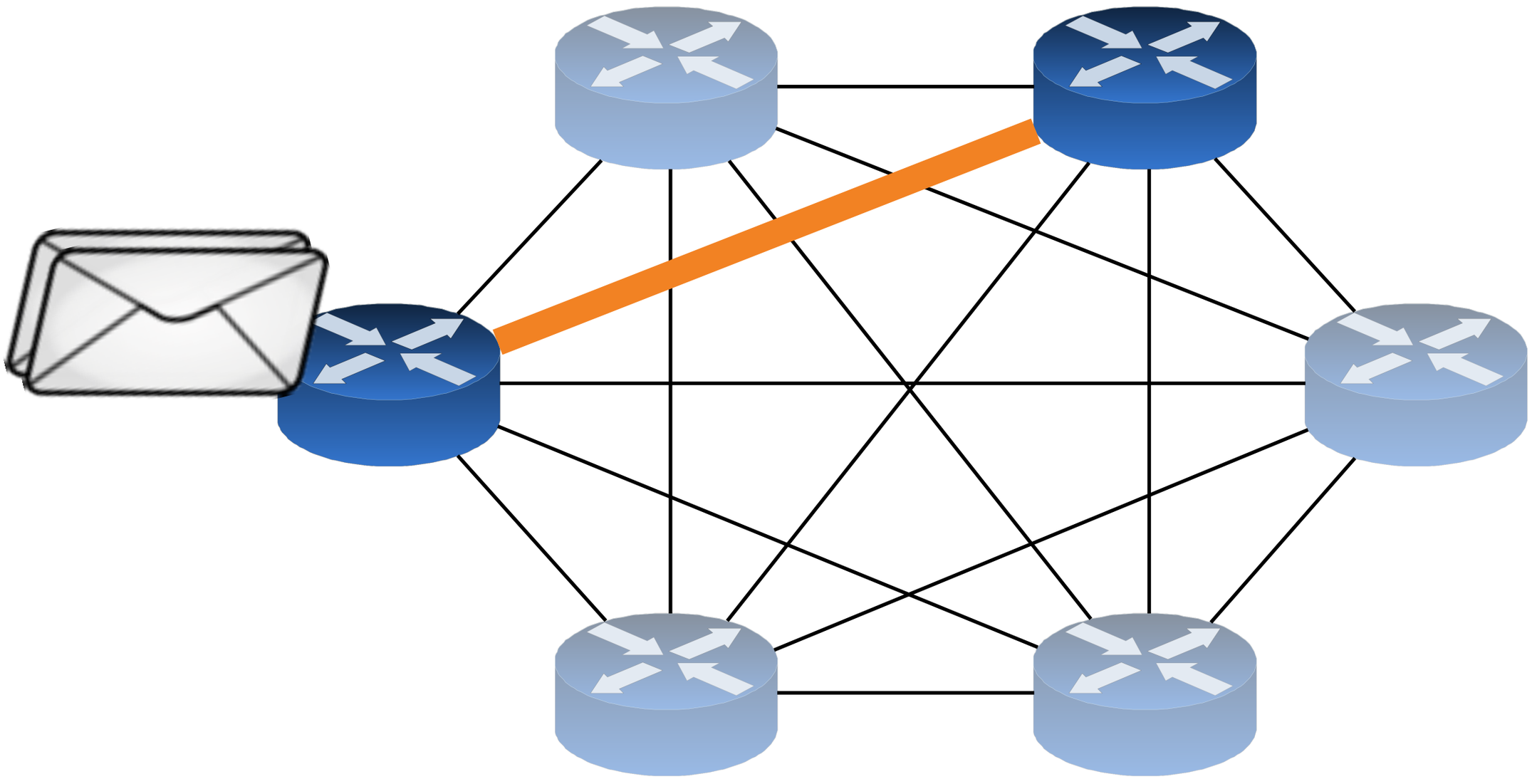
Shortest-path routing cannot route all feasible demands.

Shortest-path Routing



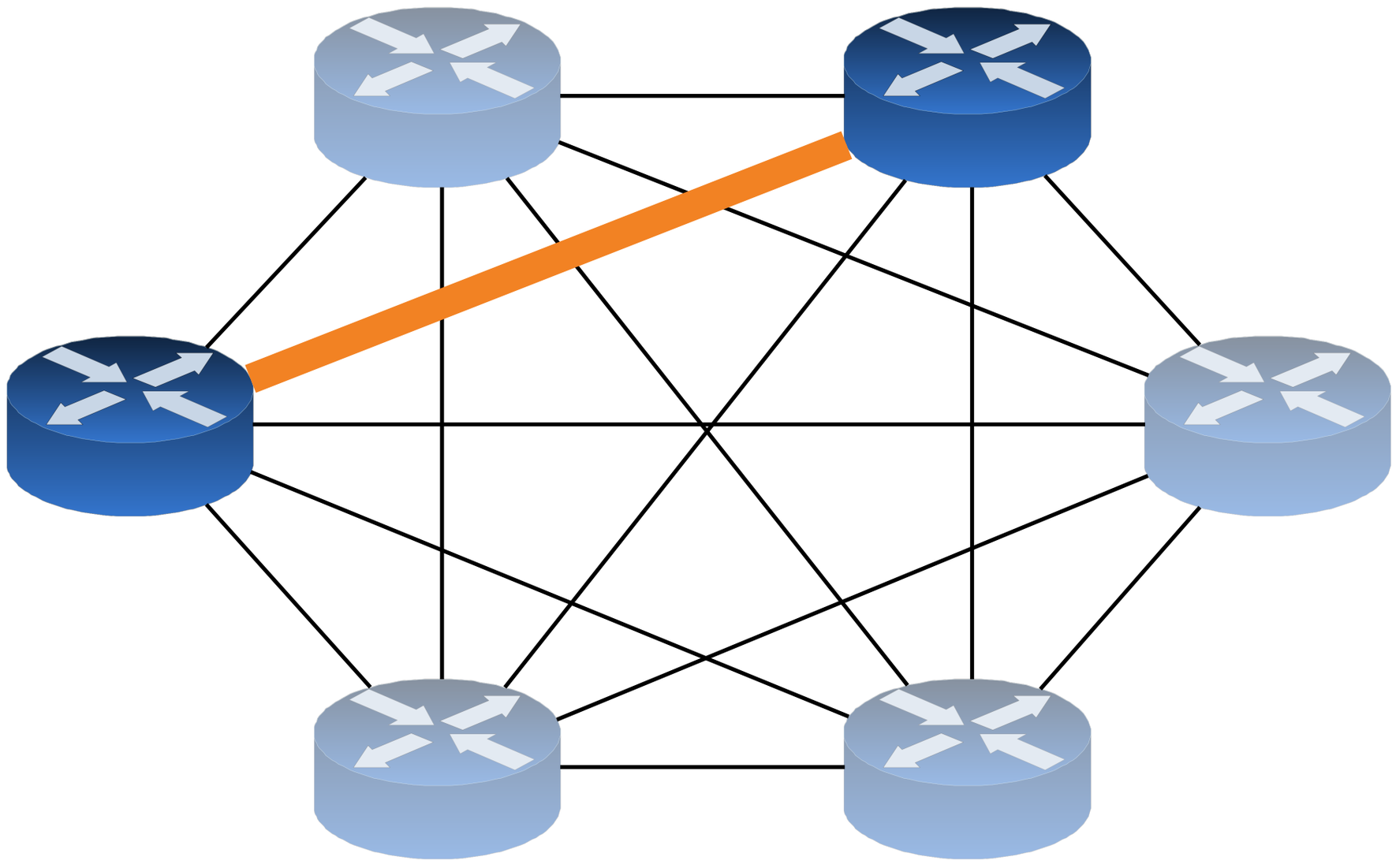
Shortest-path routing cannot route all feasible demands.

Shortest-path Routing



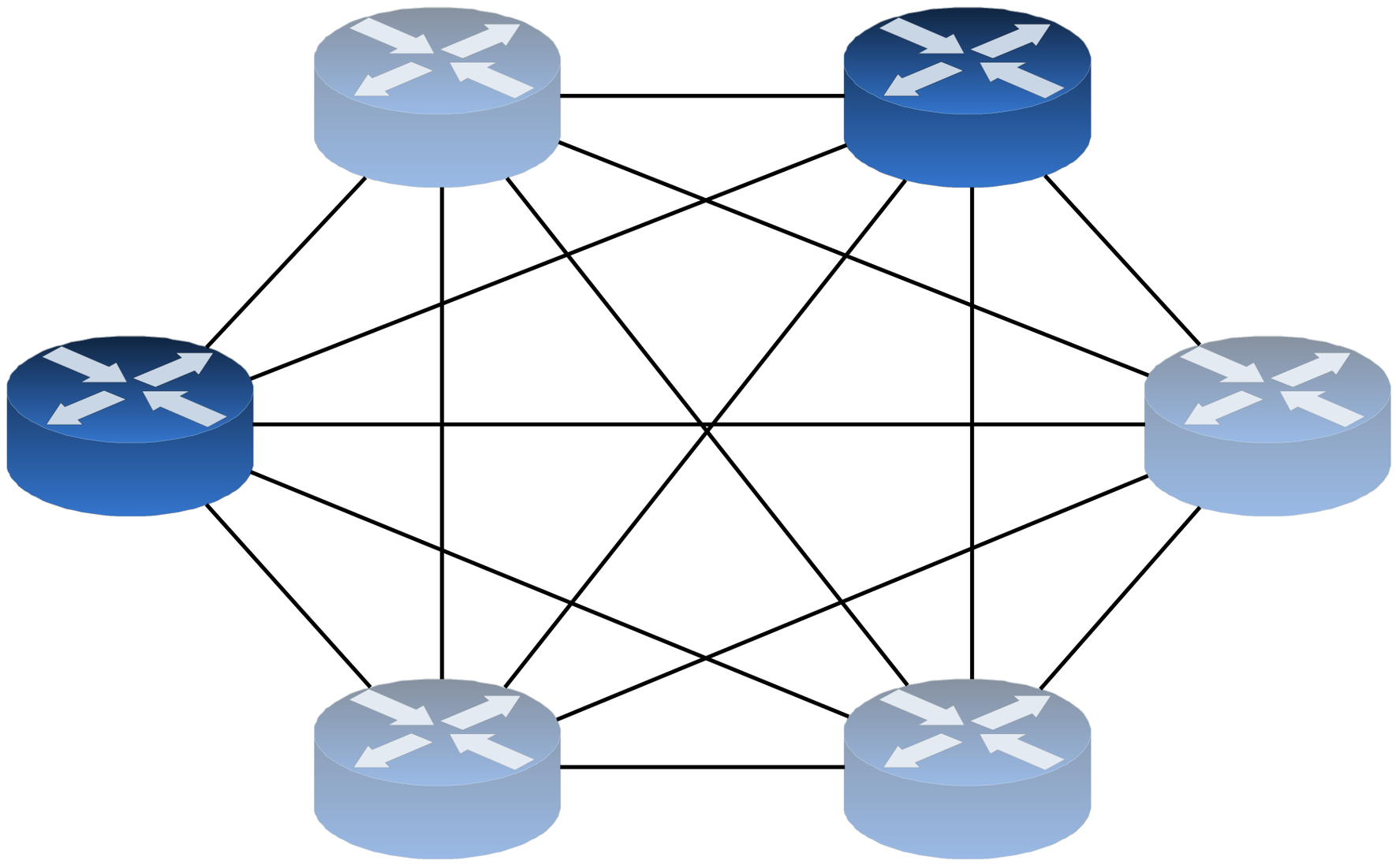
Shortest-path routing cannot route all feasible demands.

Shortest-path Routing



Shortest-path routing cannot route all feasible demands.

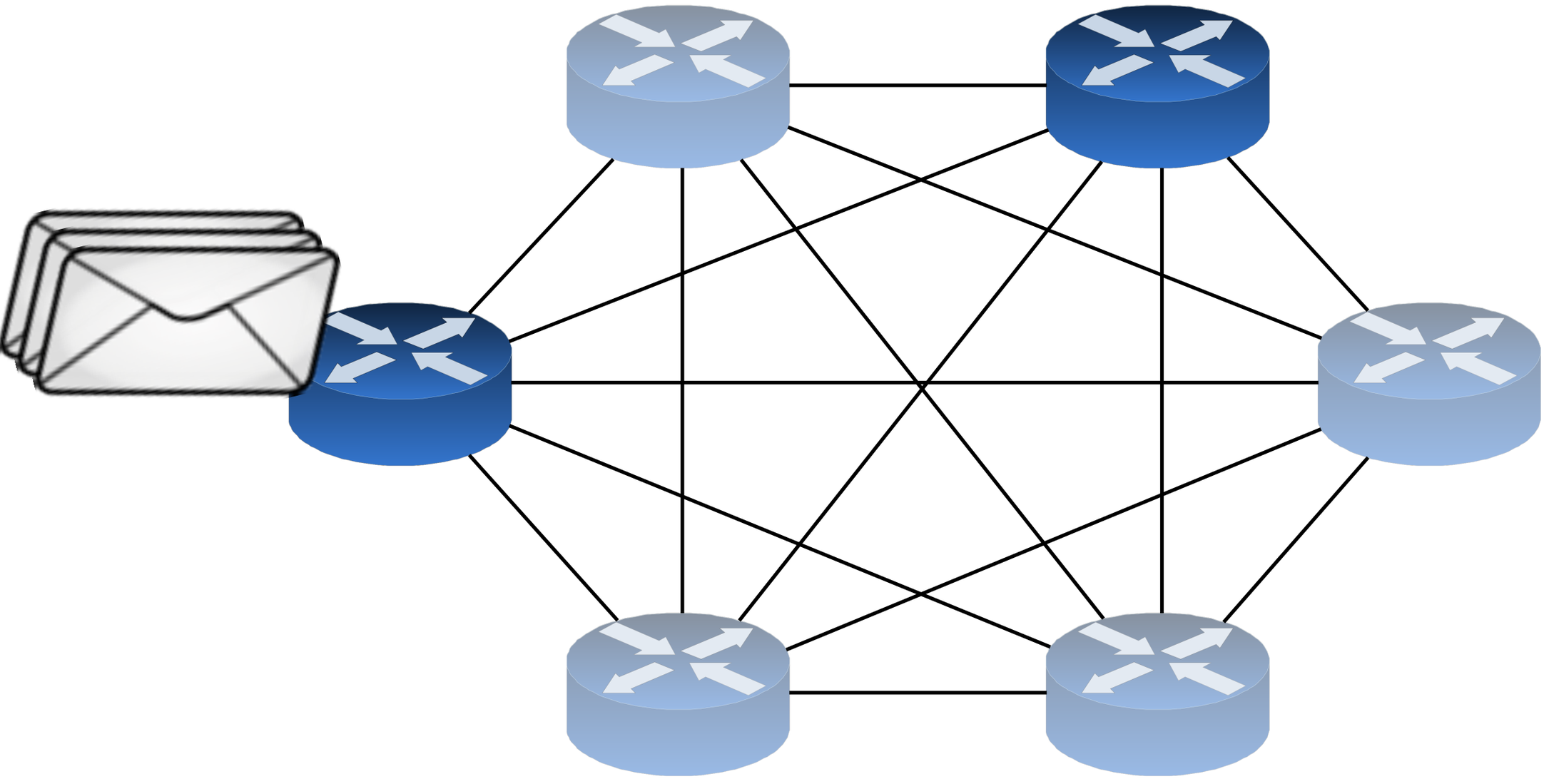
Valiant Load Balancing



Route every packet in two stages:

1. To a *random* intermediary
2. To destination

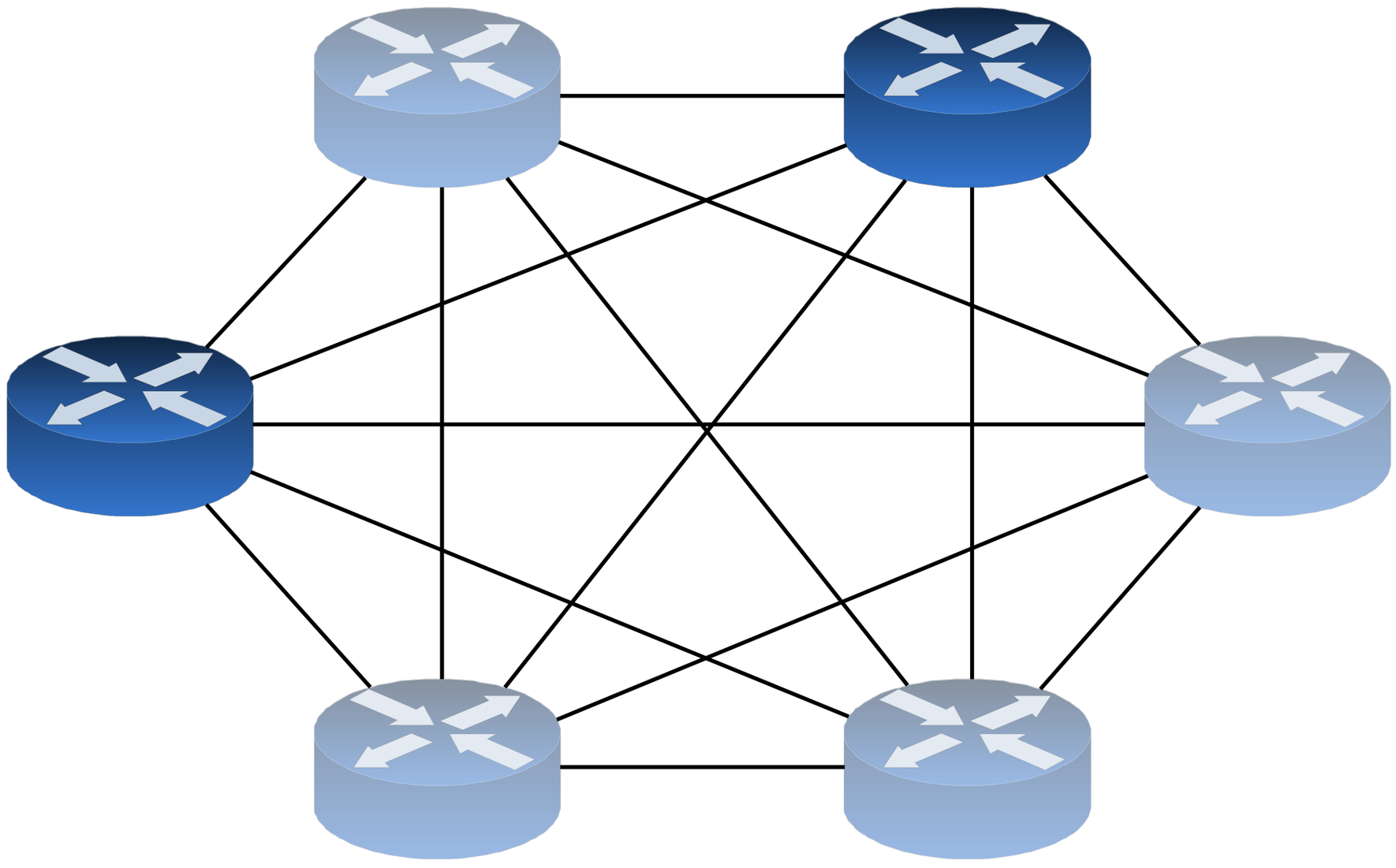
Valiant Load Balancing



Route every packet in two stages:

1. To a *random* intermediary
2. To destination

Valiant Load Balancing



Route every packet in two stages:

1. To a *random* intermediary
2. To destination

Valiant Load Balancing

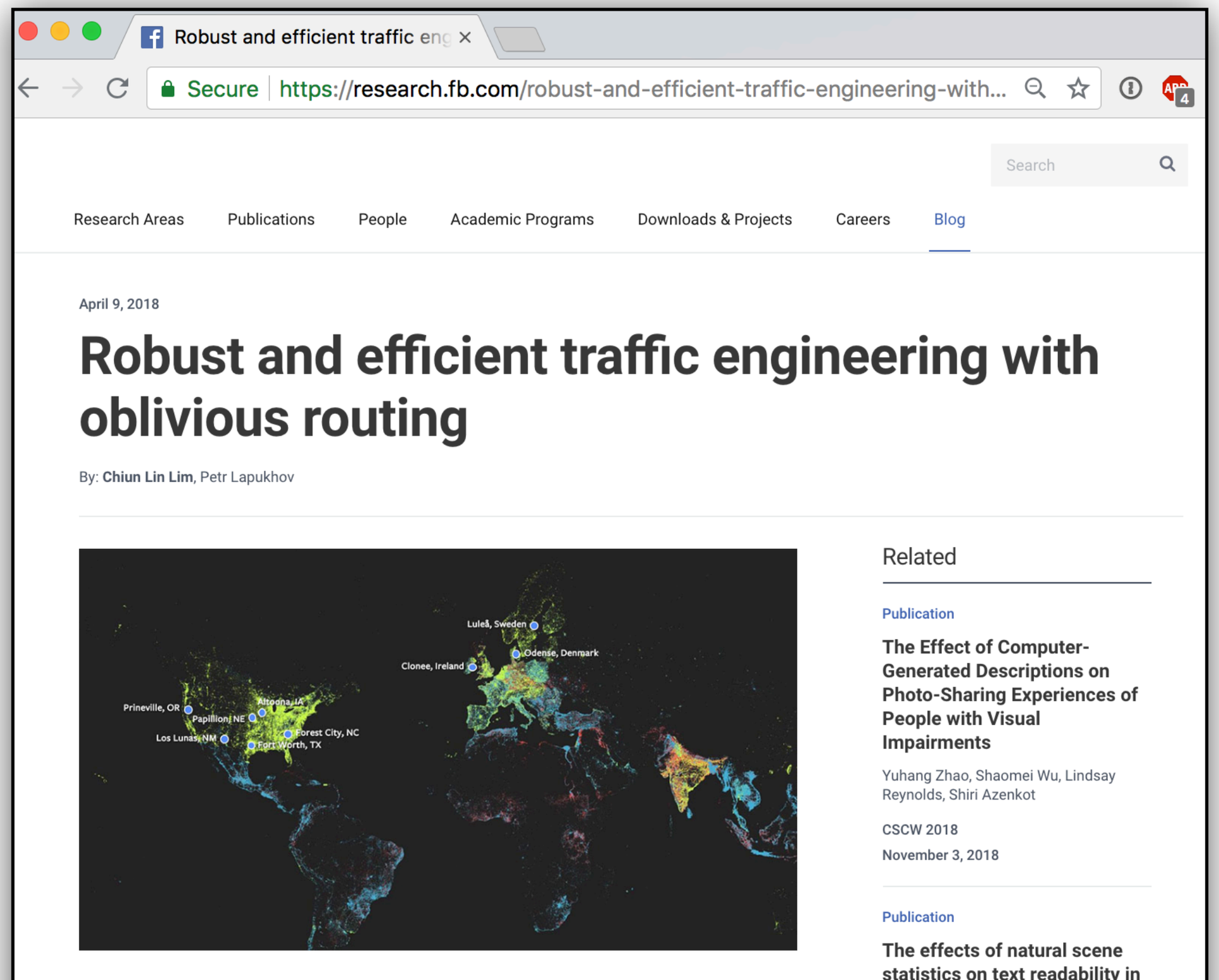
Theorem [Valiant '82]. Two-stage routing can handle all feasible demand matrices.

Intuition: the randomization in the first stage balances the offered load across all nodes (with high probability)

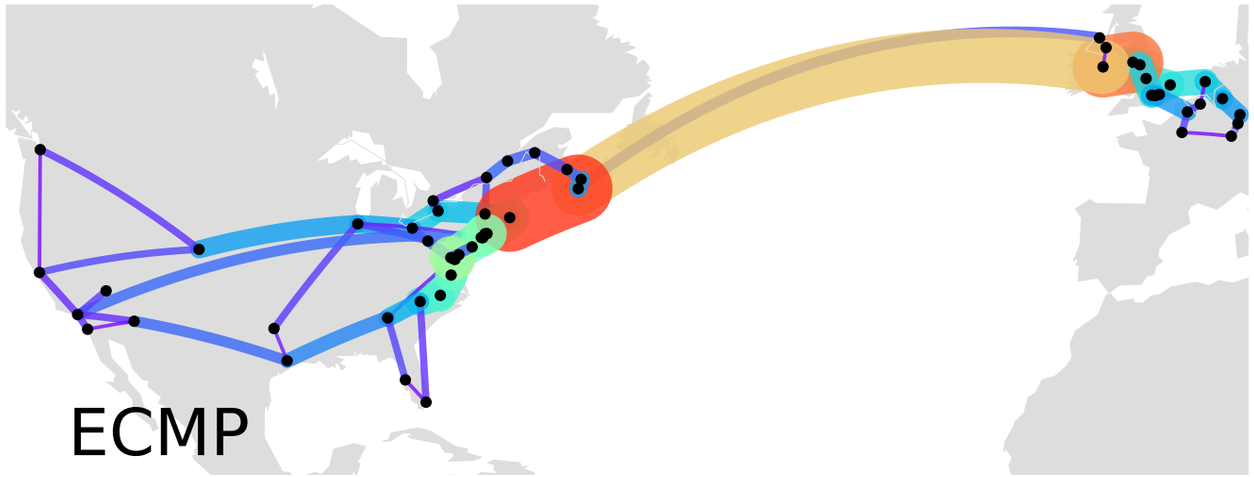
Route every packet in two stages:

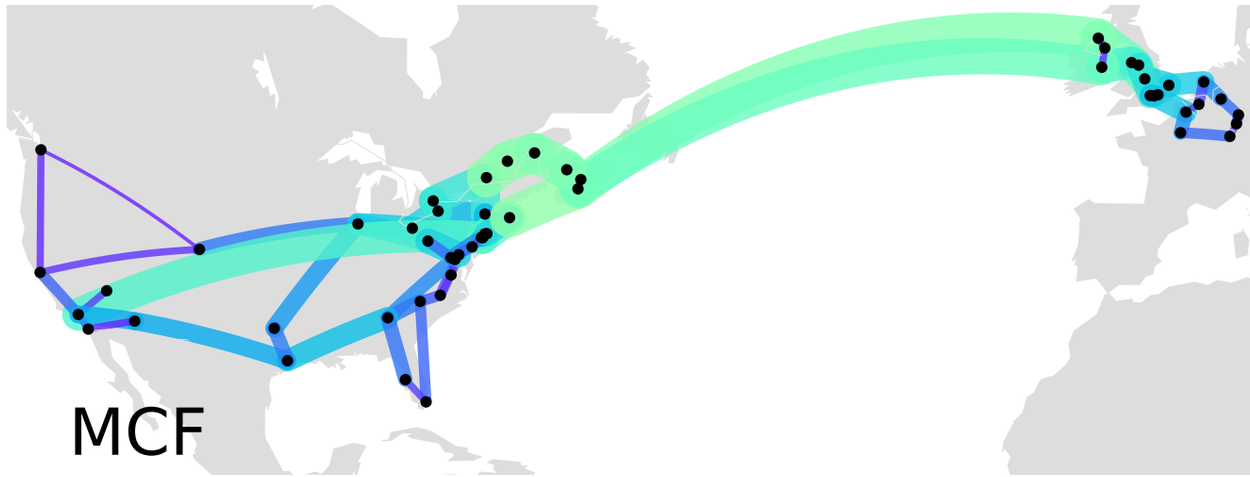
1. To a *random* intermediary
2. To destination

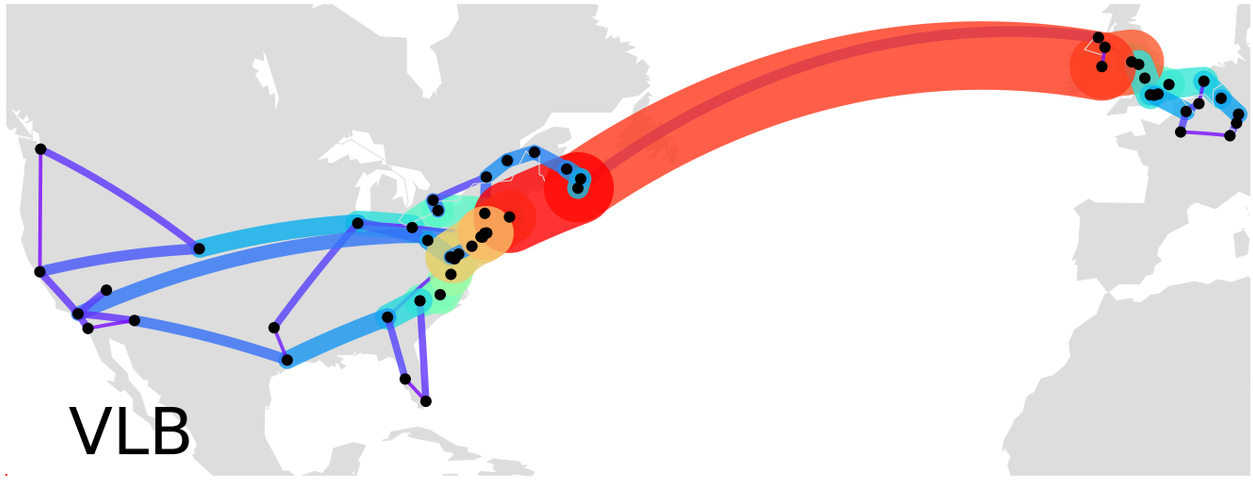
Oblivious Routing

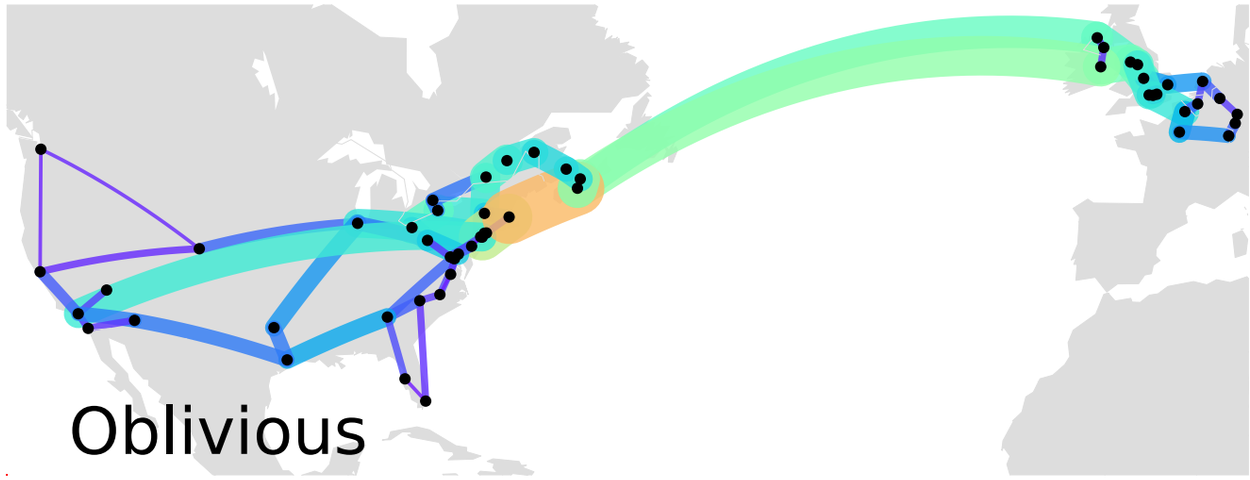


Path Selection





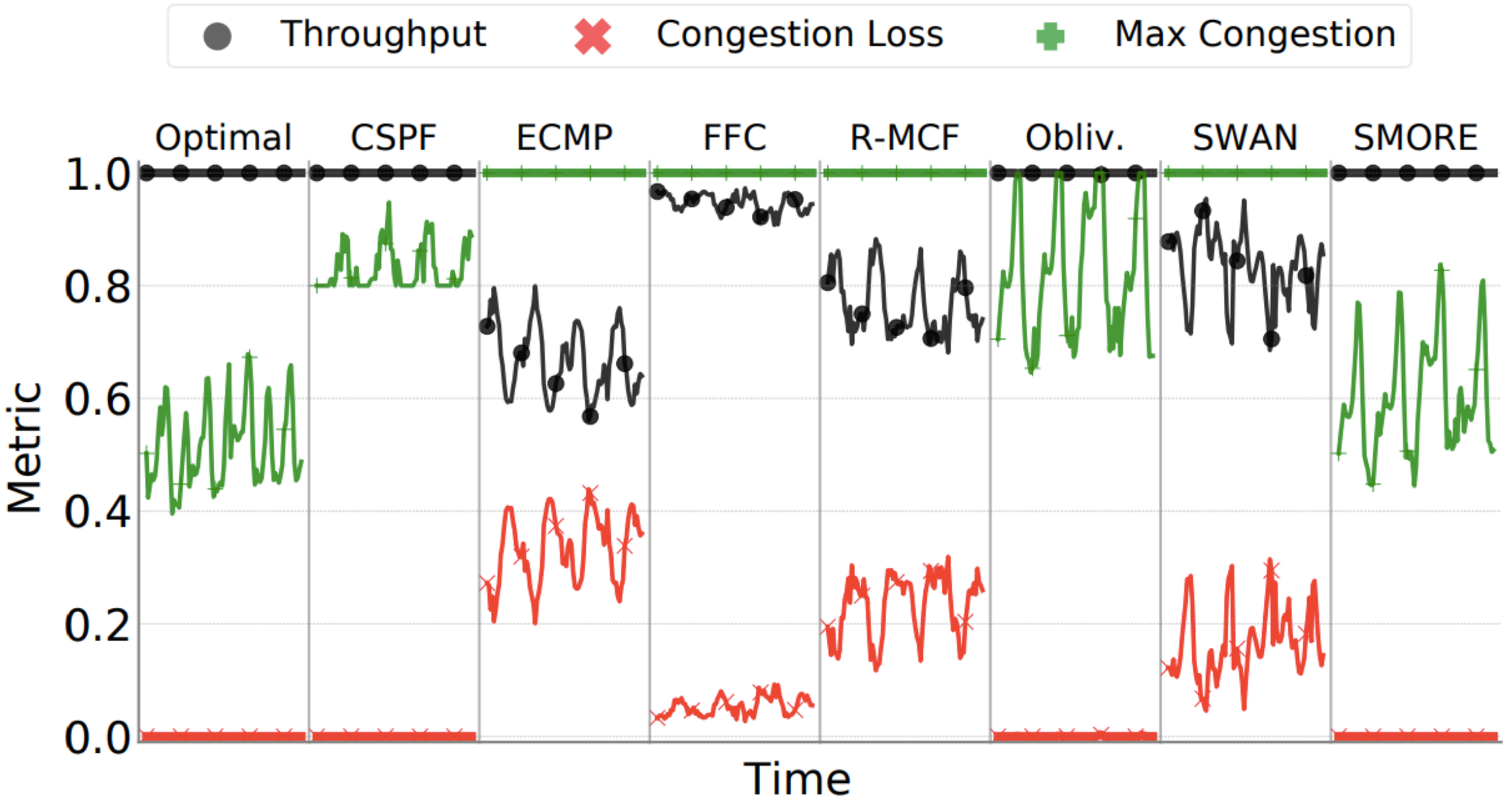




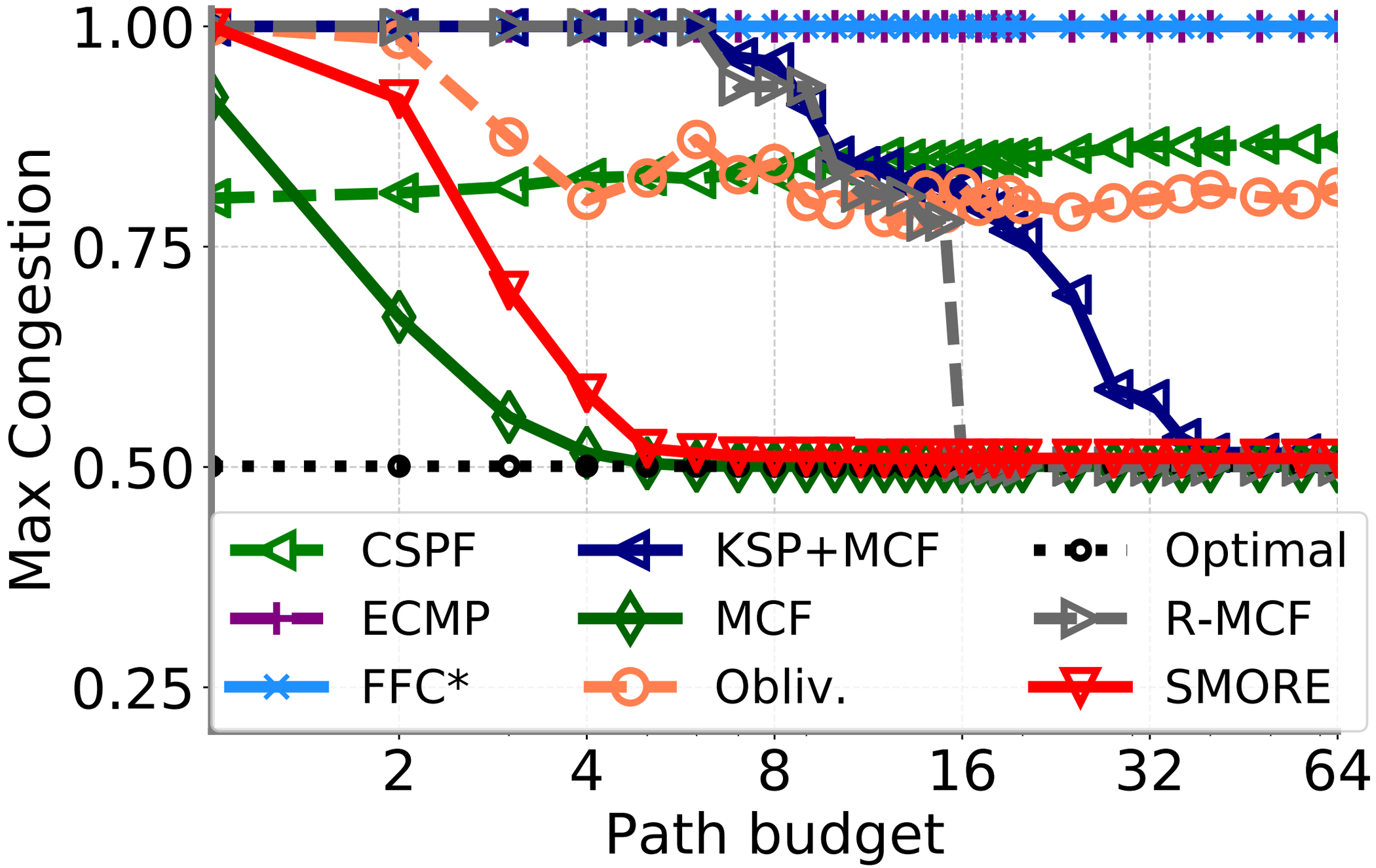
The choice of paths can have a major impact on the congestion induced in wide-area networks

Oblivious Routing

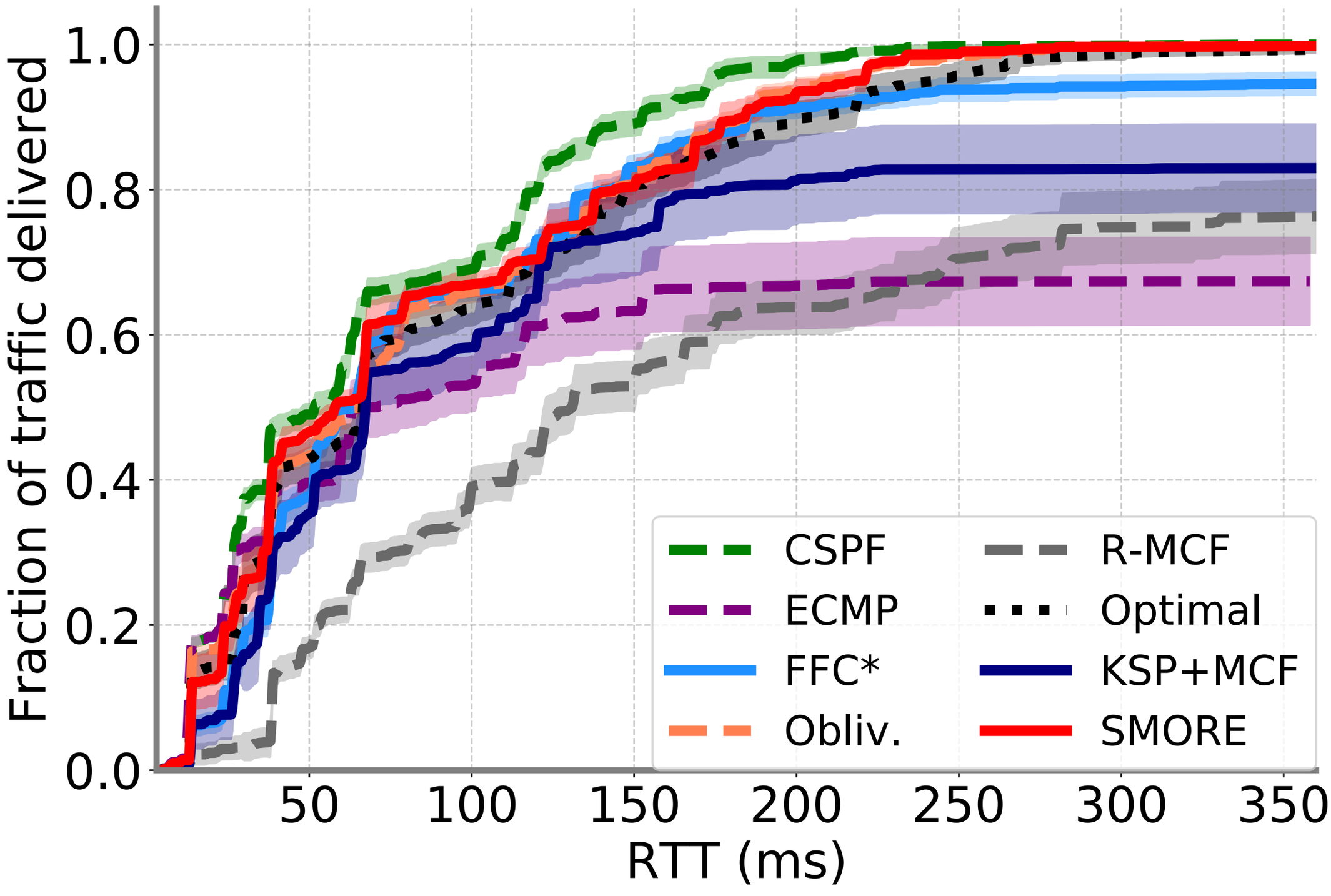
Can generalize VLB to arbitrary topologies by randomly routing traffic using a set of well-chosen tree-structured overlays



Path Budget


1.00
0.75
0.50
0.25
Max Congestion
2
4
8
16
32
64
Path budget
CSPF
ECMP
FFC*
KSP+MCF
MCF
Obliv.
Optimal
R-MCF
SMORE

Latency


Fraction of traffic delivered
1.0
0.8
0.6
0.4
0.2
0.0
50
100
150
200
250
300
350
RTT (ms)
CSPF
ECMP
FFC*
Obliv.
R-MCF
Optimal
KSP+MCF
SMORE

Probablistic NetKAT

A language for **modeling & reasoning** about networks **probabilistically.**

Prob + NetKAT

probabilistic primitive

$p \oplus_r q$

network primitives

f:=n, dup

A language for **modeling & reasoning** about networks **probabilistically.**

ProbNetKAT 2016

NetKAT 2013

KAT 1996

1956 1847

Prob	+	Net	+	KA	+	T
probabilistic primitives		network primitives		regular expressions		boolean tests
$p \oplus_r q$		f:=n, dup		+, ·, *		f=n

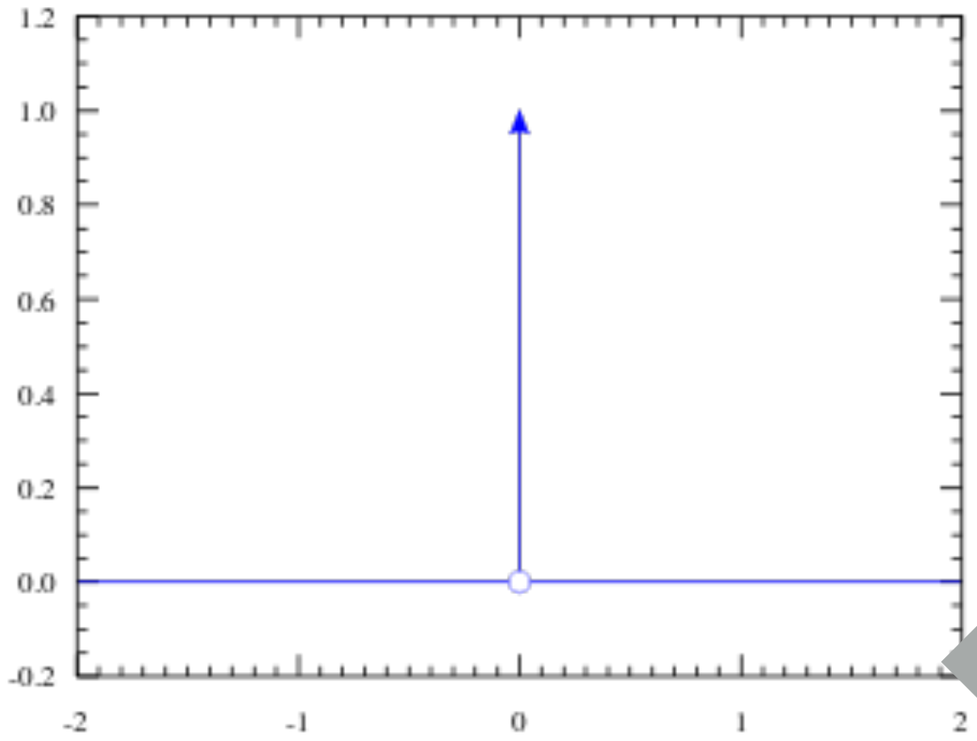
A language for **modeling & reasoning** about networks **probabilistically.**

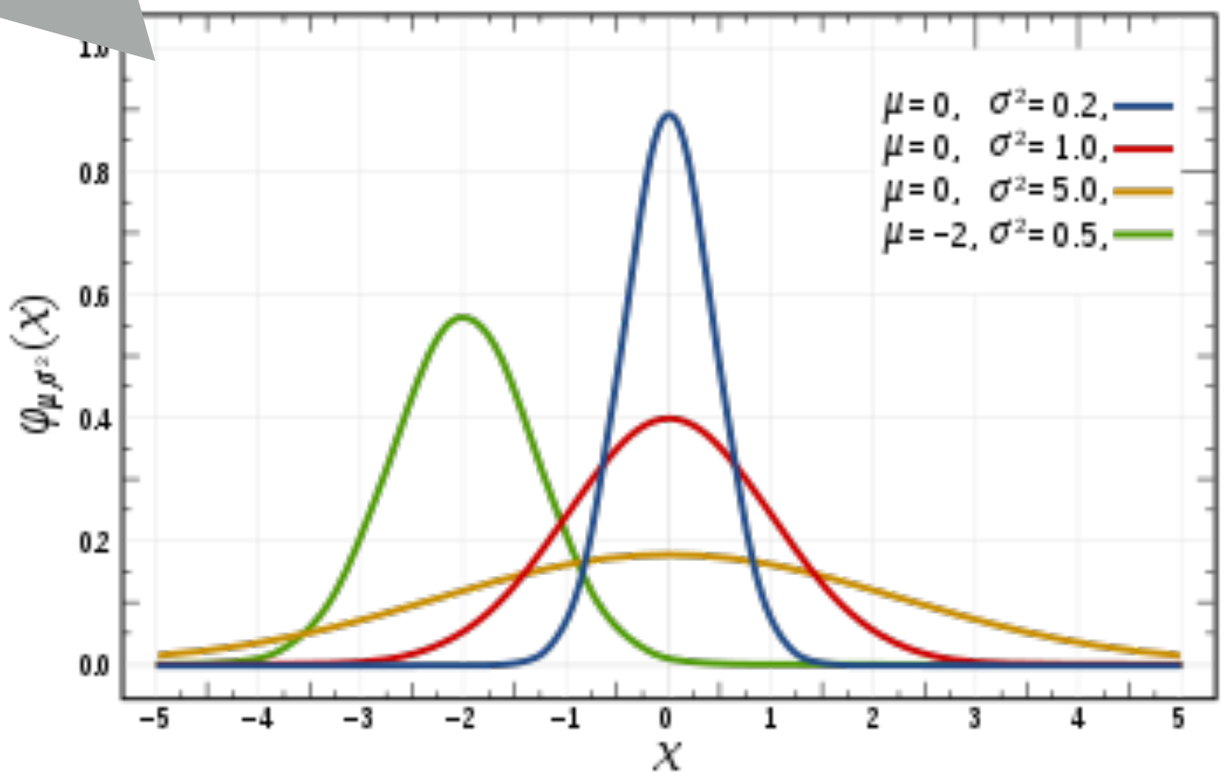
$$[\![p]\!] \in 2^H \to \mathrm{Dist}(2^H)$$

$$[\![p]\!] \in 2^H \to 2^H$$

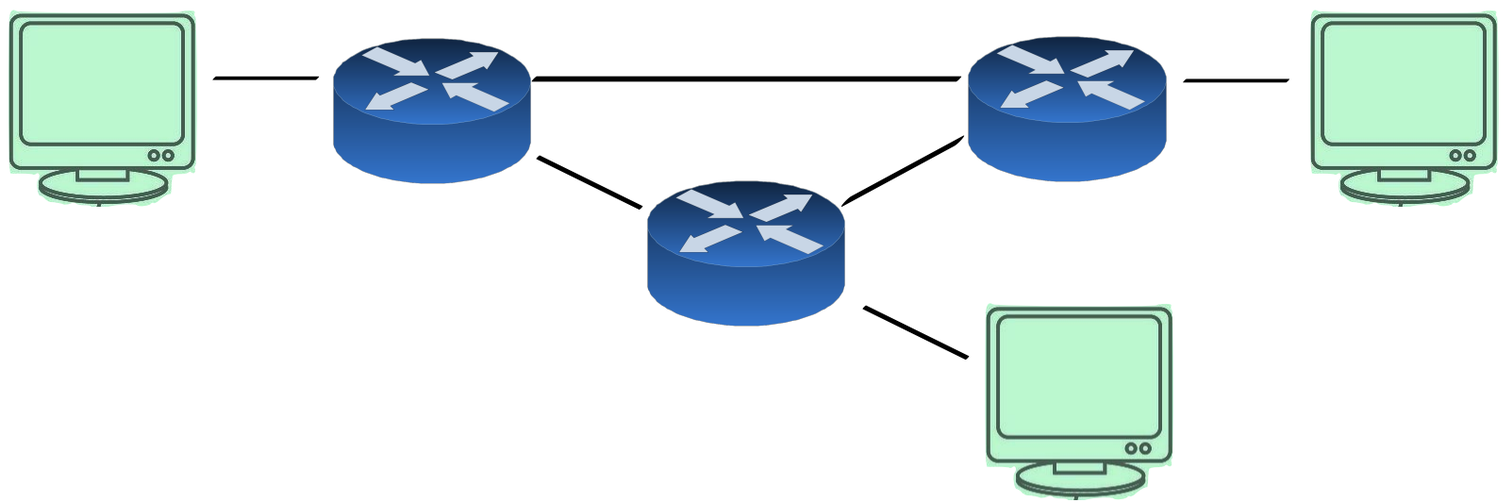
Prob	+	Net	+	KA	+	T
probabilistic primitives		network primitives		regular expressions		boolean tests
$p \oplus_r q$		f:=n, dup		+, ·, *		f=n

Probabilistic Semantics



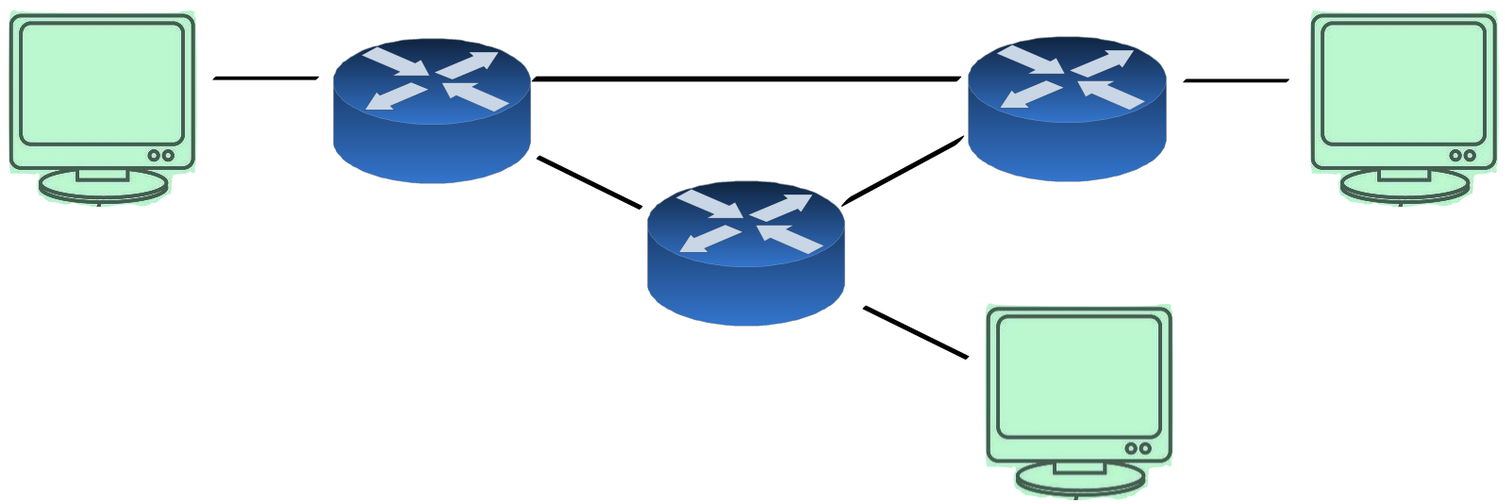


Probabilistic Reasoning



ProbNetKAT model **p**,
input distribution **μ**

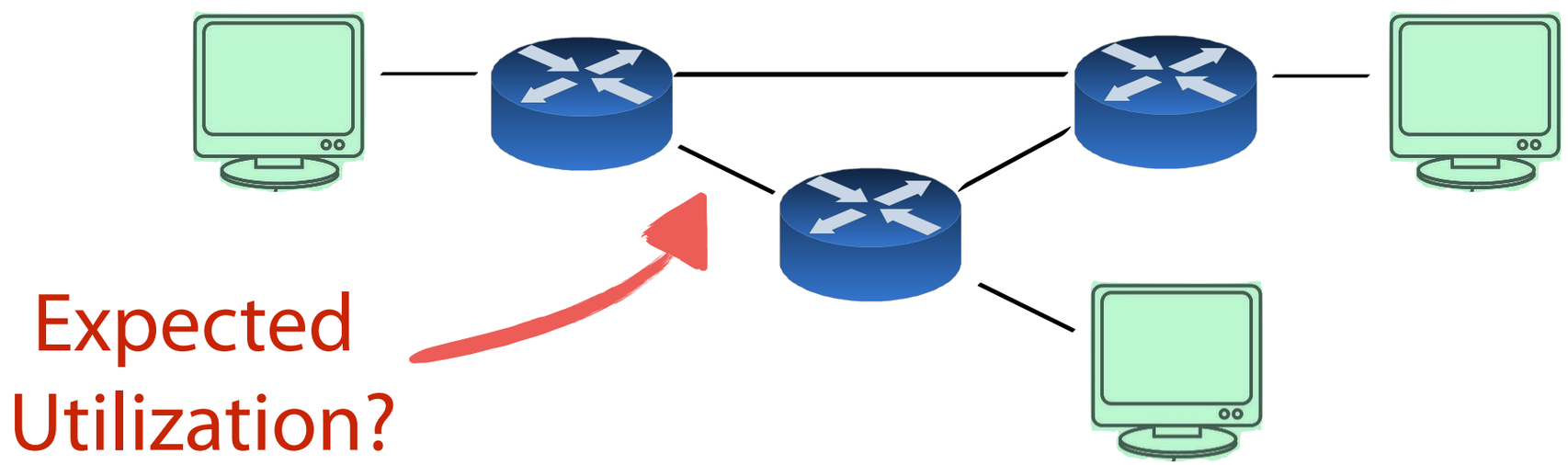
Probabilistic Reasoning



ProbNetKAT model **p**,
input distribution **μ**

→ output distribution **ν** = ⟦p⟧(μ) ∈ Dist(2^H)

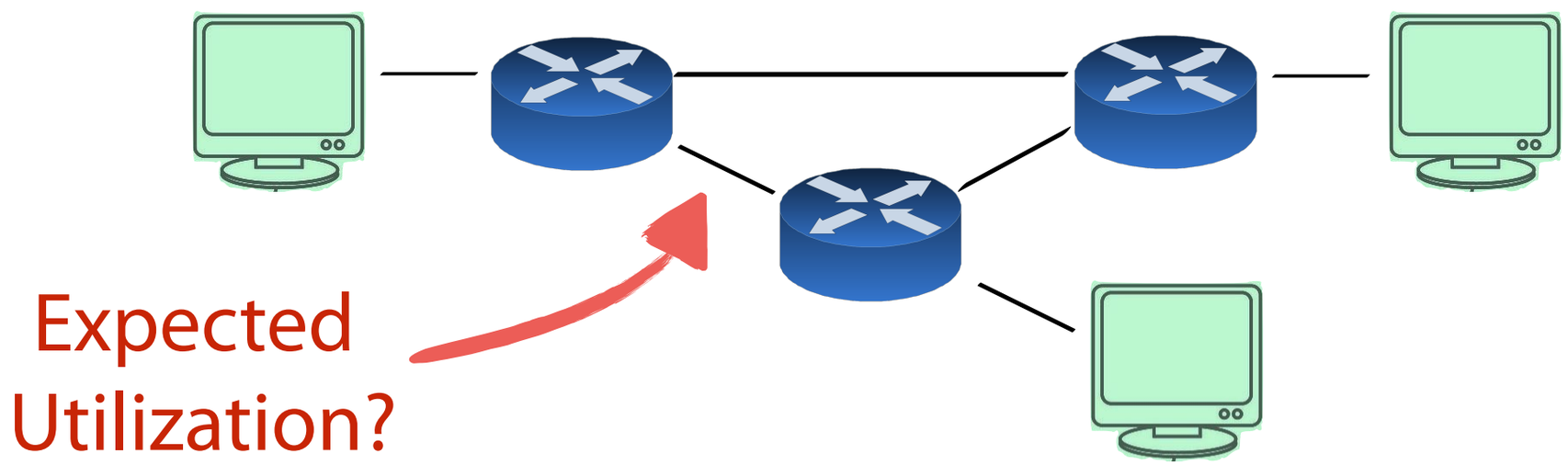
Probabilistic Reasoning



ProbNetKAT model **p**,
input distribution **μ**

→ output distribution **ν** = ⟦p⟧(μ) ∈ Dist(2^H)

Probabilistic Reasoning



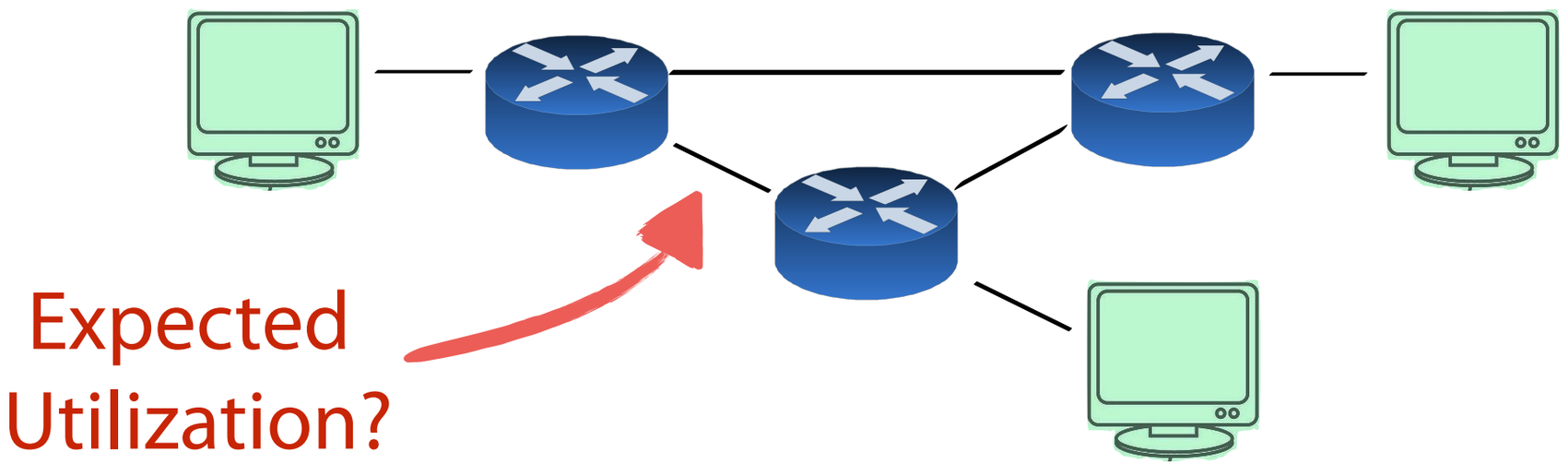
ProbNetKAT model **p**,
input distribution **μ**

→ output distribution **ν** = ⟦p⟧(μ) ∈ Dist(2^H)

utilization query: **Q** : 2^H → [0,∞]

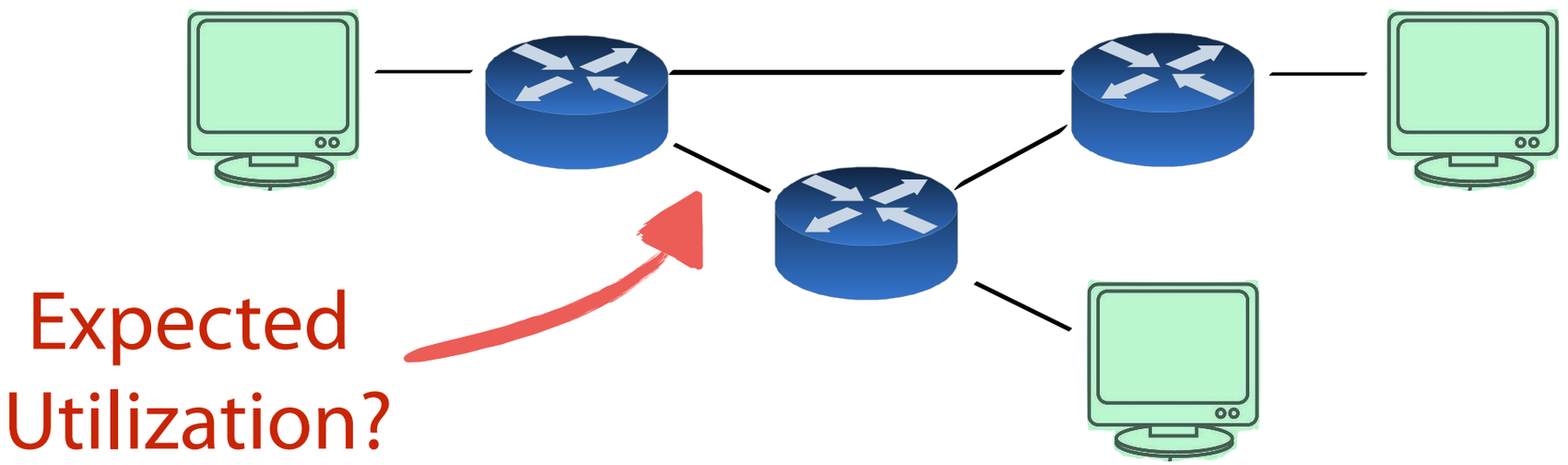
expected utilization: $E_{\nu}[\mathbf{Q}]$

How to implement this?



$$E_{\nu}[\mathbf{Q}] = \int Q \, d\nu$$

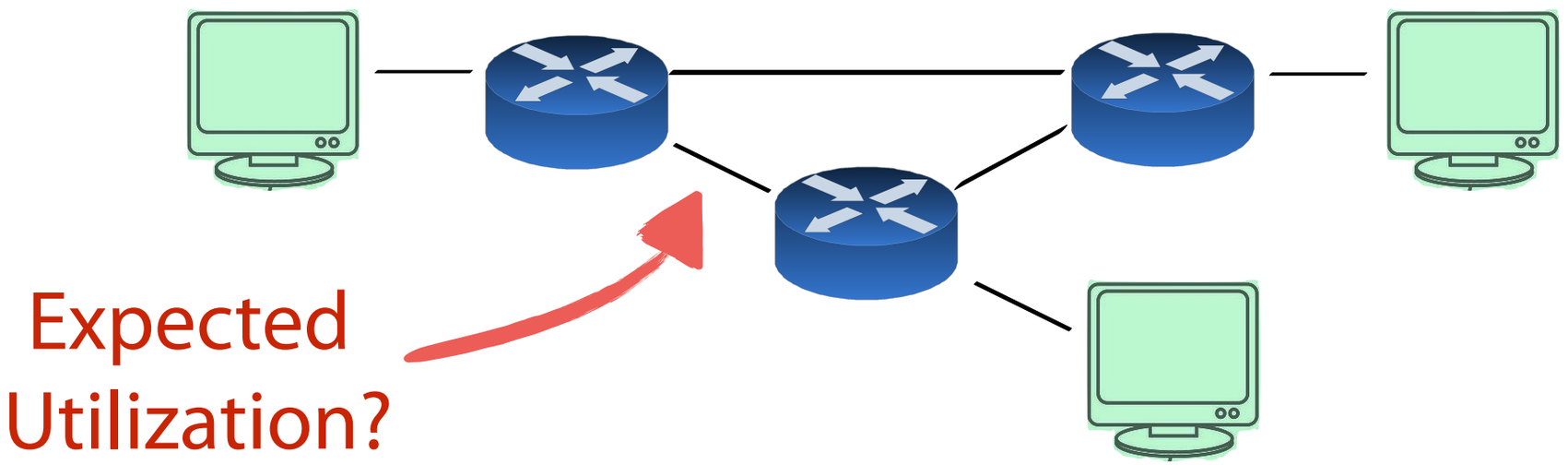
How to implement this?



$$E_{\nu}[Q] = \int Q \, d\nu$$

Lebesgue
Integral

How to implement this?



Expected Utilization?

$$E_{\nu}[\mathbf{Q}] = \int Q \, d\nu$$

Lebesgue Integral

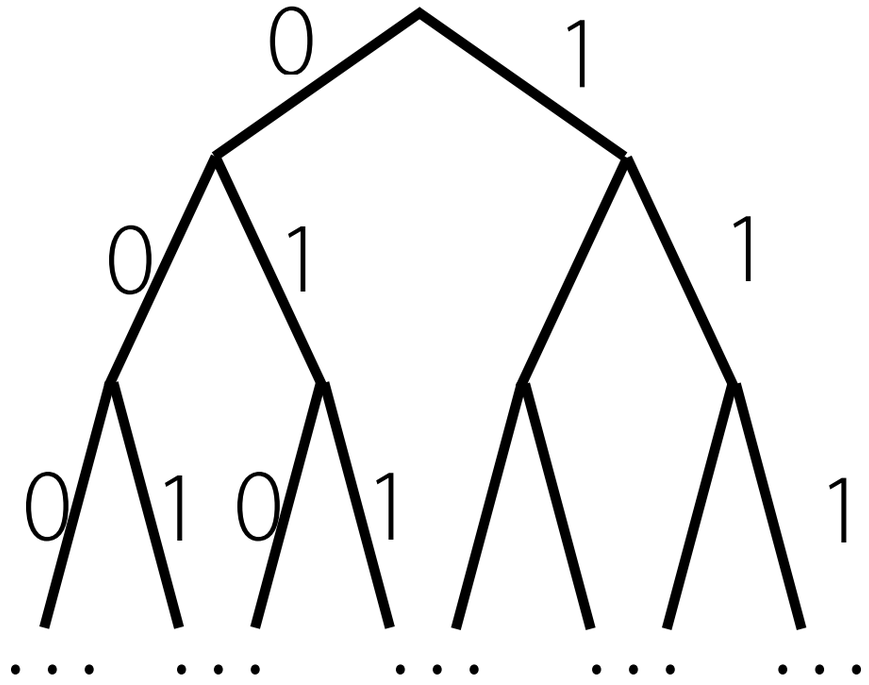
may be a continuous distribution!

Continuous Distribution

$$((f{:=}0 \oplus f{:=}1) \bullet \mathbf{dup})^*$$

Continuous Distribution

$$((f{:=}0 \oplus f{:=}1) \bullet \mathbf{dup})^*$$



execution

≅

infinite path

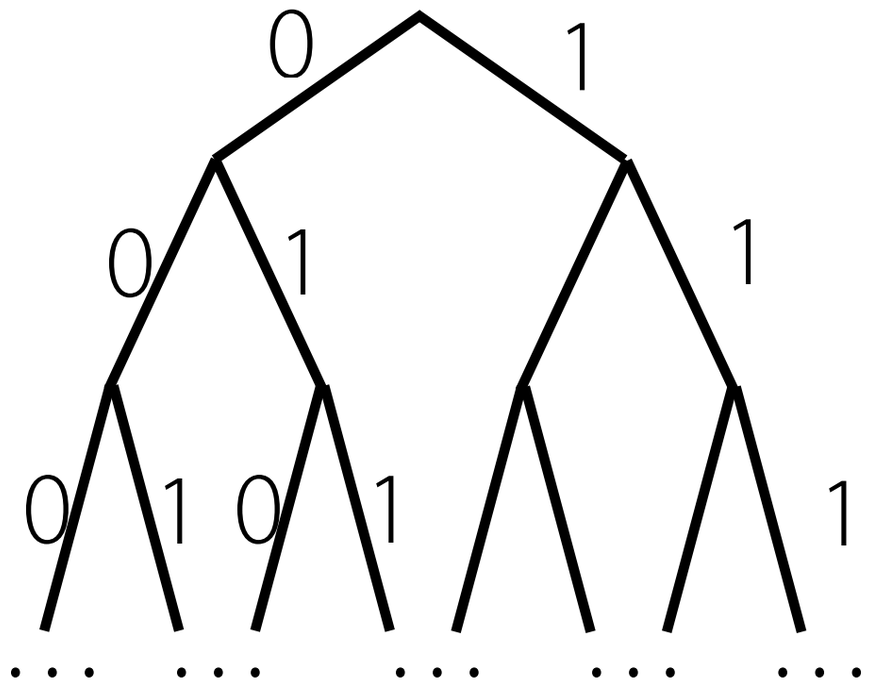
≅

random output

$\in 2^{\mathbb{H}}$

Continuous Distribution

$$((f{:=}0 \oplus f{:=}1) \bullet \mathbf{dup})^*$$



execution

≅

infinite path

≅

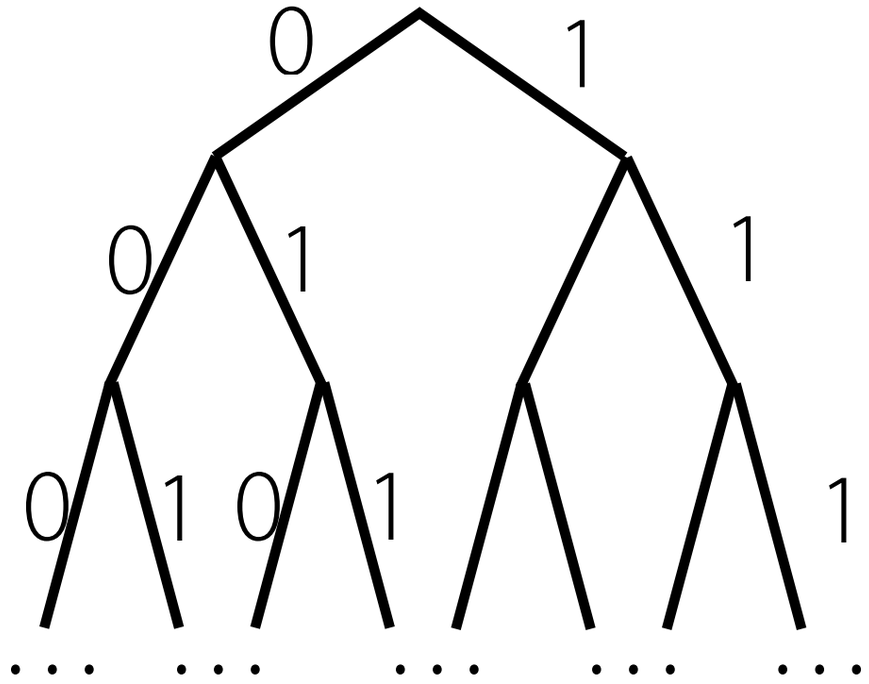
random output

$\in 2^H$

How many paths are there? → one for every $r \in [0,1]$

Continuous Distribution

$$((f{:=}0 \oplus f{:=}1) \bullet \mathbf{dup})^*$$



execution

≅

infinite path

≅

random output

$\in 2^H$

How many paths are there? → one for every $r \in [0,1]$

What's the probability of any particular path? → 0

Key Idea

limits + continuity → approximation

Key Idea

limits + continuity → approximation

$\mu_1, \mu_2, \mu_3, \ldots$ —**converges**→ $\mu \in \mathrm{Dist}(2^H)$

Key Idea

limits + **continuity** → approximation

$\mu_1, \mu_2, \mu_3, \ldots$ —**converges**→ $\mu \in \text{Dist}(2^H)$

continuity (↓)

$f(\mu_1), f(\mu_2), f(\mu_3), \ldots$ —**converges**→ $f(\mu) \in R$

Main Results

Main Results

1) Iteration-free programs generate only finite distributions

Main Results

1) Iteration-free programs generate only finite distributions
2) Iteration may introduce continuous distributions …
   … but can be approximated by bounded iteration

Main Results

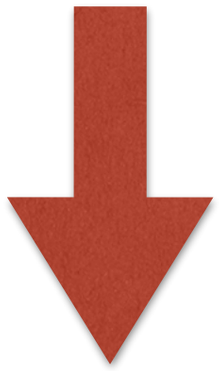
1) Iteration-free programs generate only finite distributions
2) Iteration may introduce continuous distributions …
   … but can be approximated by bounded iteration

3) All programs can be approximated

Main Results

1) Iteration-free programs generate only finite distributions
2) Iteration may introduce continuous distributions …
   … but can be approximated by bounded iteration

compositionality of approximation



continuity of $+, \cdot, ^*, \oplus$

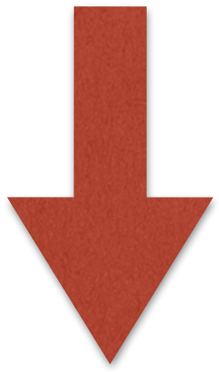
3) All programs can be approximated

Main Results

1) Iteration-free programs generate only finite distributions

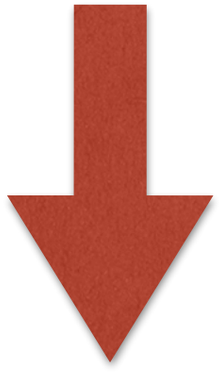
2) Iteration may introduce continuous distributions …
… but can be approximated by bounded iteration

compositionality of approximation



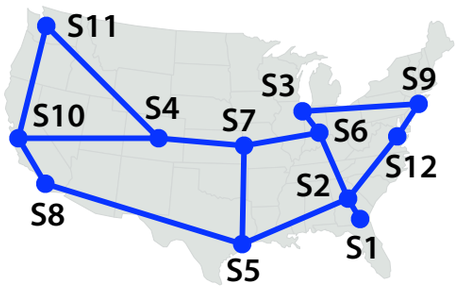
continuity of $+, \cdot, *, \oplus$

3) All programs can be approximated

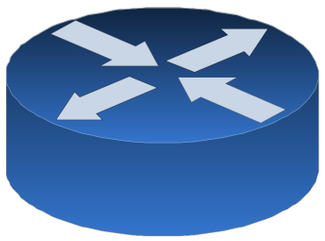


continuity of $\int$, $\mathbb{E}[-]$

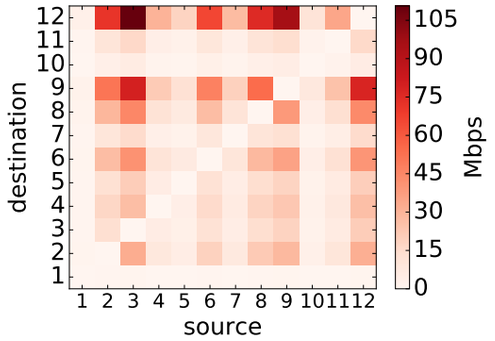
4) Queries can be approximated


S11
S3
S9
S10
S4
S7
S6
S12
S2
S8
S1
S5

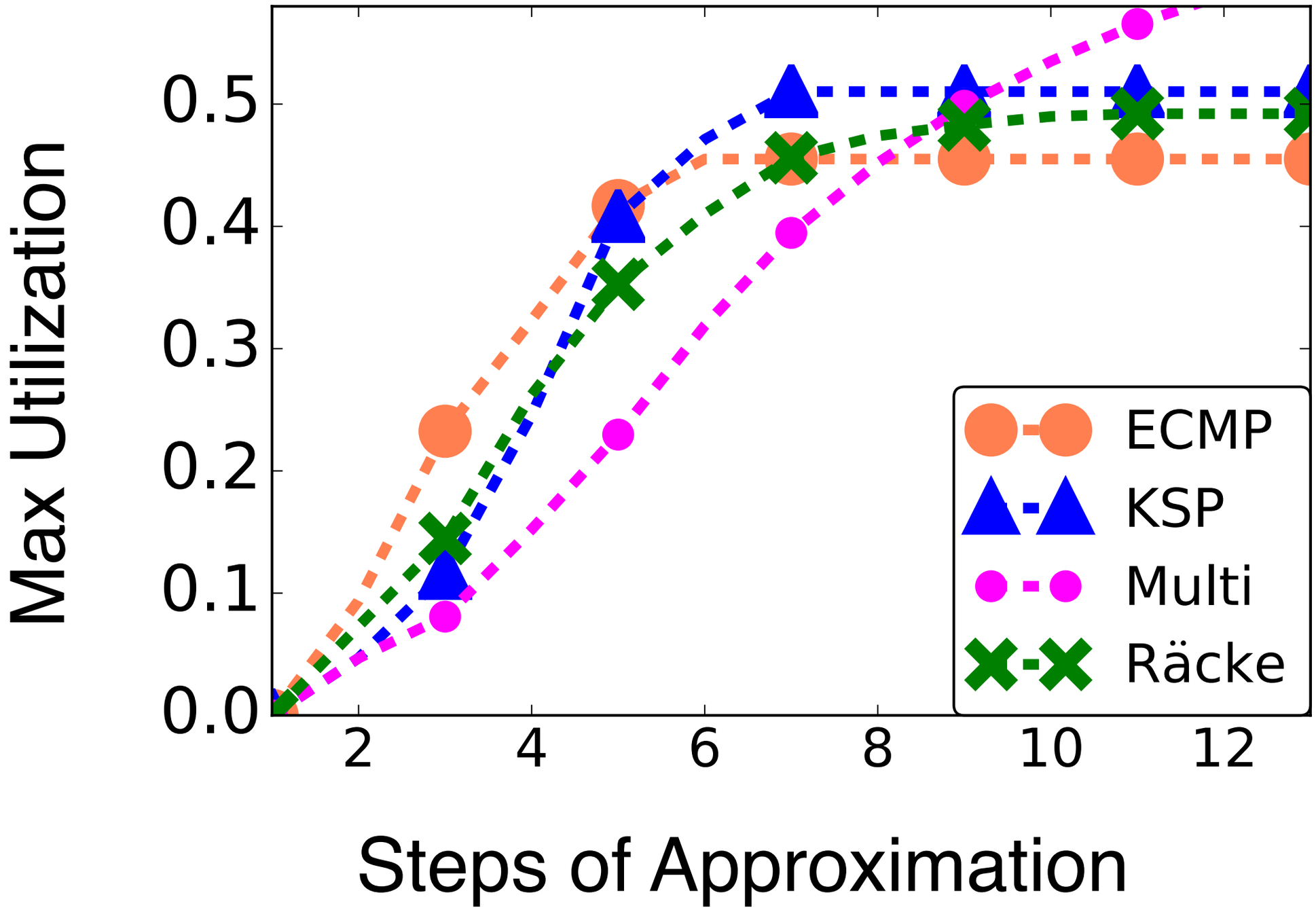
t



p


destination
source
Mbps
1 2 3 4 5 6 7 8 9 10 11 12
0 15 30 45 60 75 90 105

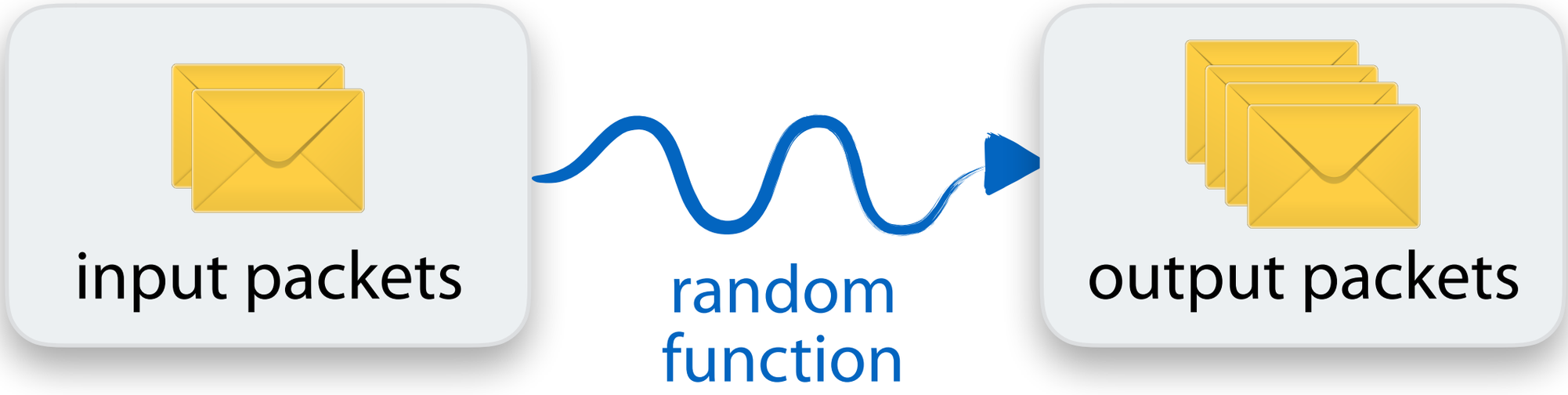
μ


Max Utilization
0.0
0.1
0.2
0.3
0.4
0.5
2
4
6
8
10
12
Steps of Approximation
ECMP
KSP
Multi
Räcke

Quantitative Verification

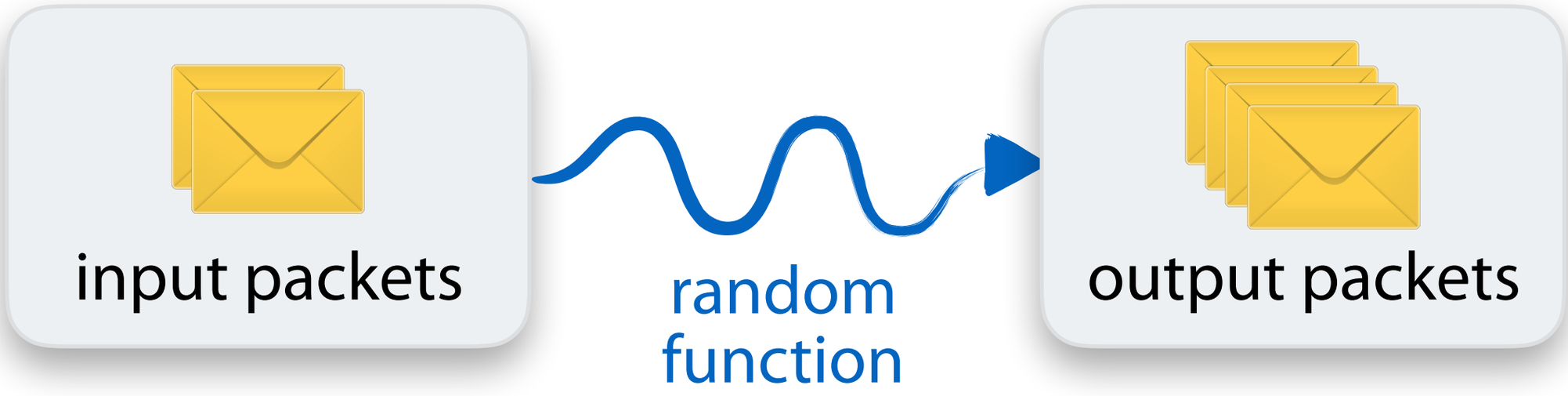
Decidable Semantics

Restricted to the local fragment, a NetKAT program can be seen as a **random** function on packets:



Decidable Semantics

Restricted to the local fragment, a NetKAT program can be seen as a **random** function on packets:



Such functions can be modeled as **Markov chains**:

- **states** are given by **sets of packet**: $S = 2^{Pk}$
- Chain is given by transition matrix $\mathbf{B} \in [0,1]^{SxS}$
- **B(a, b):** probability of producing **output b** on **input a**
- state space large, but **finite!**

Example: Sequential Composition

ProbNetKAT primitives can be mapped to matrices and program operators map to matrix operations.

$$\mathbf{B[\![p \cdot q]\!]} := \begin{array}{c} \\ a \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccc} b_1 & \cdots & b_n \end{array} \\ \begin{bmatrix} & \cdots & \\ 1/3 & 1/3 & 1/3 \\ & \cdots & \end{bmatrix} \end{array} \cdot \begin{array}{c} \begin{array}{ccc} \cdots & c & \cdots \end{array} \\ \begin{bmatrix} & 1/6 & \\ \vdots & 2/6 & \vdots \\ & 3/6 & \end{bmatrix} \end{array} \begin{array}{c} \\ b_1 \\ \vdots \\ b_n \end{array}$$

$\mathbf{B[\![p]\!]}$ $\mathbf{B[\![q]\!]}$

Similarly for p **&** q, p $\oplus_{\mathbf{r}}$ q.

Problem: Kleene Star

Question

Let π_1**!** be the program that generates packet π_1

Consider p = (true $\oplus$ π_1**!**). What is $\mathbf{B}[\![p^*]\!](\{\pi_0\}, \{\pi_0\})$?

Answer

$$\mathbf{B}[\![p^{(n)}]\!](\{\pi_0\}, \{\pi_0\}) \;=\; (1/2)^n$$

Thus,

$$\begin{aligned} & \mathbf{B}[\![p^*]\!](\{\pi_0\}, \{\pi_0\}) \\ = & \lim_{n\to\infty} \mathbf{B}[\![p^{(n)}]\!](\{\pi_0\}, \{\pi_0\}) \\ = & \lim_{n\to\infty} (1/2)^n \;=\; 0 \end{aligned}$$

But how to compute these limits in general?

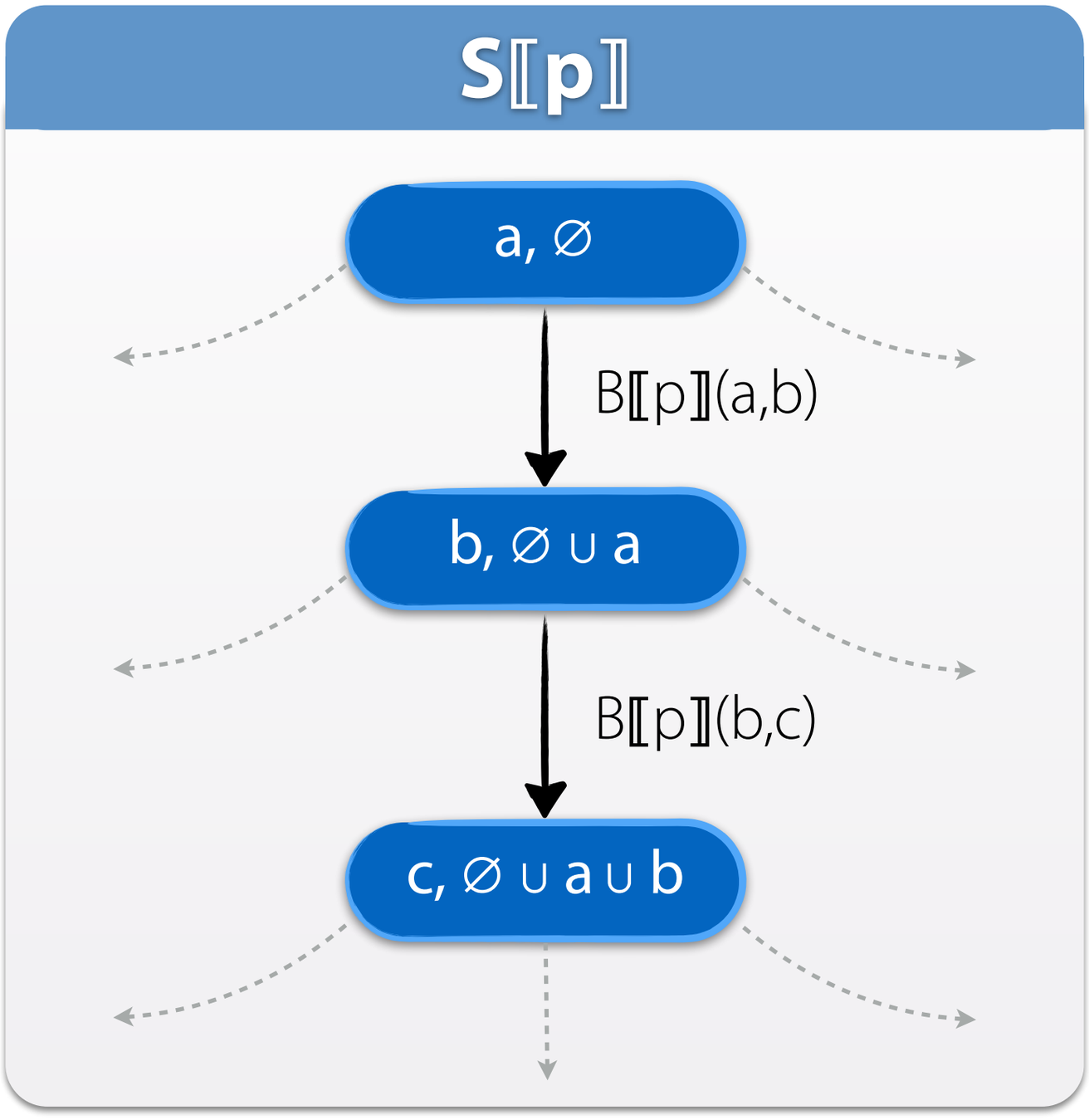
Small Step Semantics

1 step in **S⟦p⟧** ≈ 1 iteration of **p***

In one iteration, p*:

- **executes p** to get new set of packets
- **emits** previous set of **packets**

$$B[\![p^*]\!] := \lim_{n \to \infty} S[\![p]\!]^n$$



Small Step Semantics

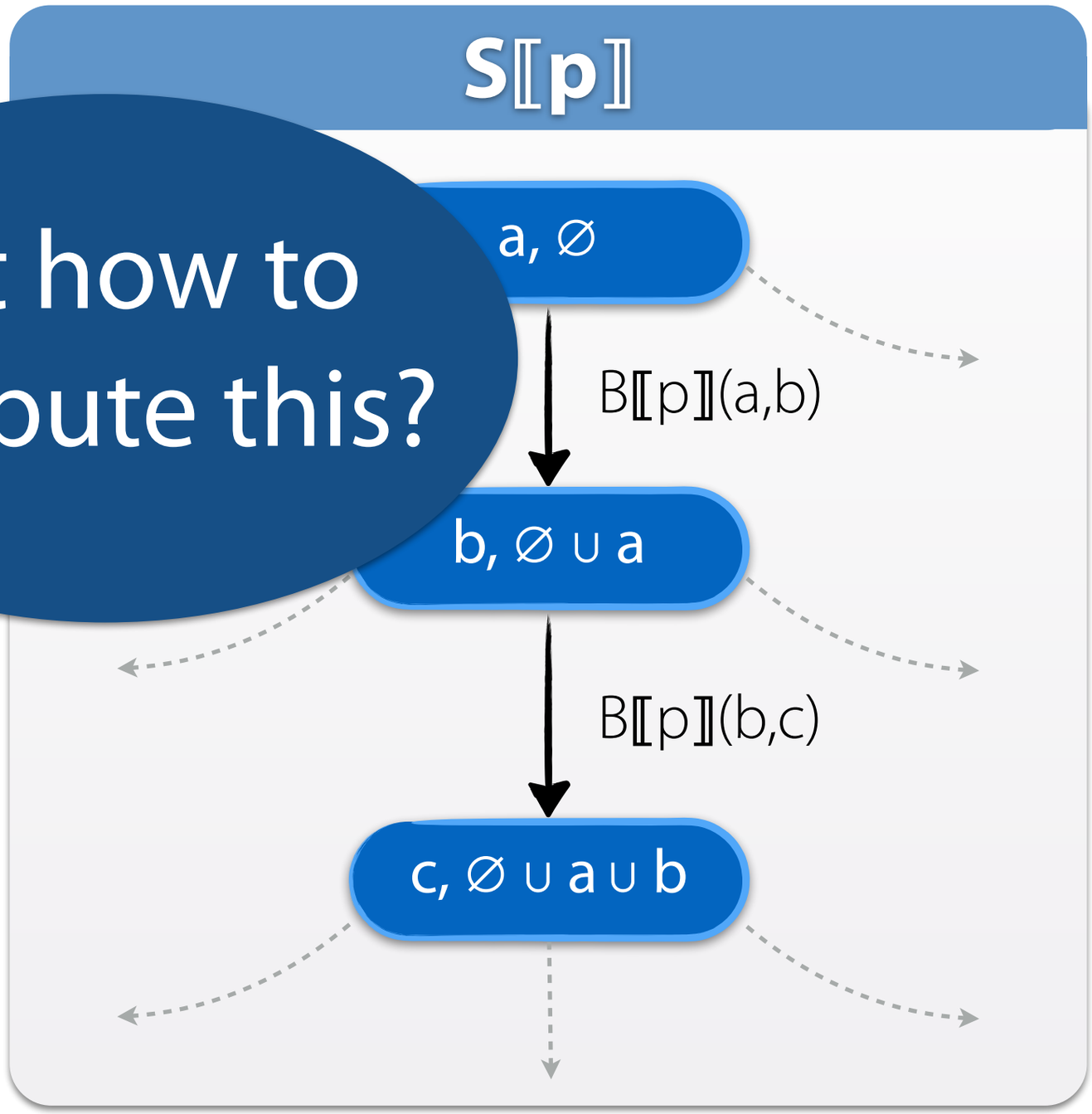
1 step in **S⟦p⟧** ≈ 1 iteration of **p***

In one iteration, p*:

- **executes p** to c
  new set of pac
- **emits** previous
  of **packets**

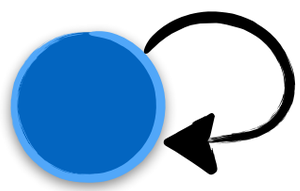
But how to compute this?

$$B[\![p^*]\!] := \lim_{n \to \infty} S[\![p]\!]^n$$



Absorbing Markov Chains

S⟦p⟧ can be "massaged" into an **absorbing Markov chain**

Absorbing state: 

Absorbing chain: any state can reach absorbing state

Crucial property: for #steps → ∞, will reach absorbing state with probability 1 (no matter the start state)

$$T = \begin{bmatrix} I & 0 \\ R & Q \end{bmatrix}$$

$$\lim_{n \to \infty} T^n = \begin{bmatrix} I & 0 \\ (I - Q)^{-1} R & 0 \end{bmatrix}$$

Absorbing Markov Chains

S⟦p⟧ can be "massaged" into an **absorbing Markov chain**

Absorbing sta

Absorbing tate

Crucial pro orbing state with prob e)

So $B⟦p^*⟧ = \lim_{n\to\infty} S⟦p⟧^n$ can be computed explicitly!

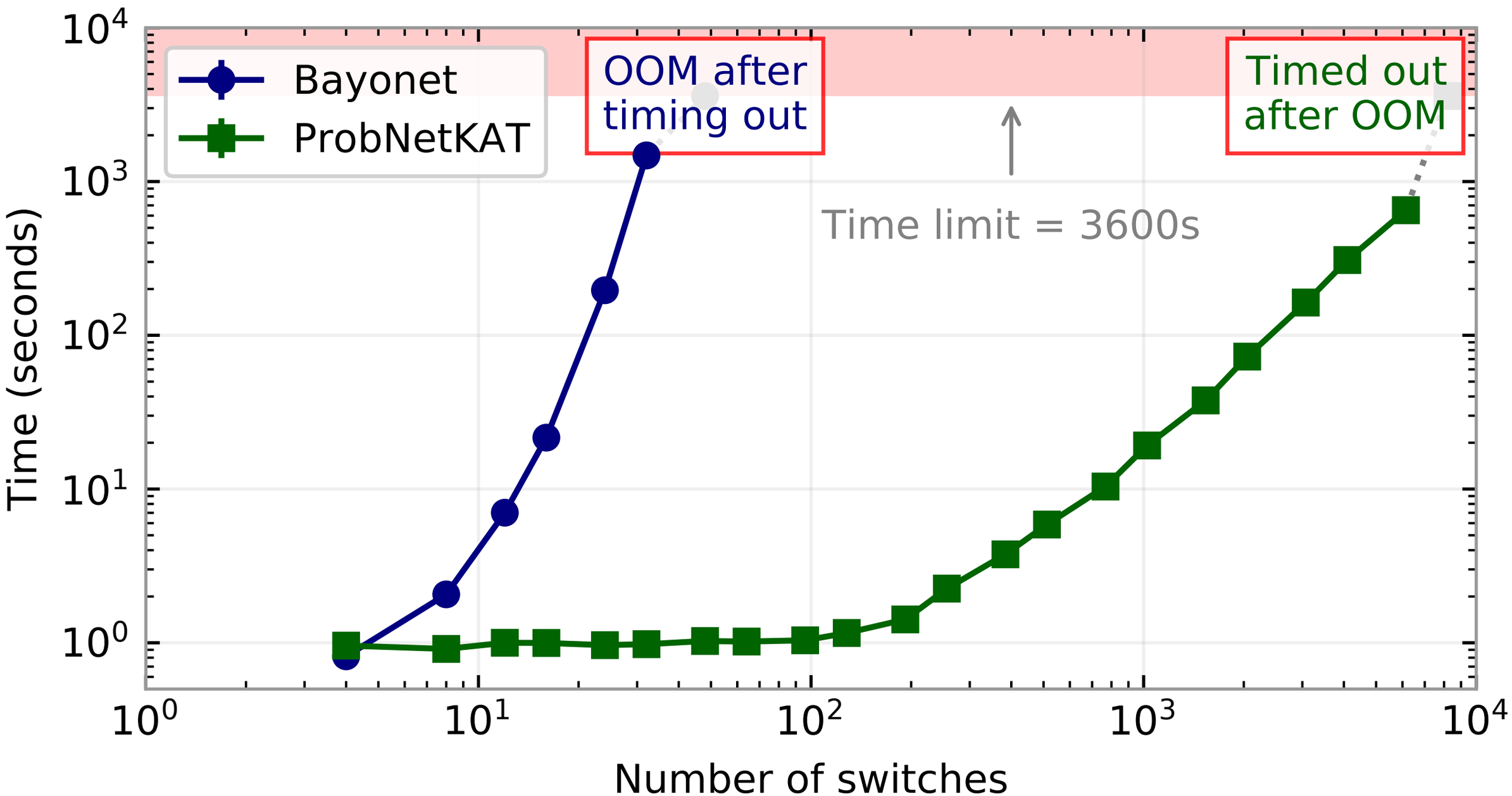
$$T = \begin{bmatrix} I & 0 \\ R & Q \end{bmatrix}$$

$$\lim_{n\to\infty} T^n = \begin{bmatrix} I & 0 \\ (I-Q)^{-1}R & 0 \end{bmatrix}$$

Evaluation

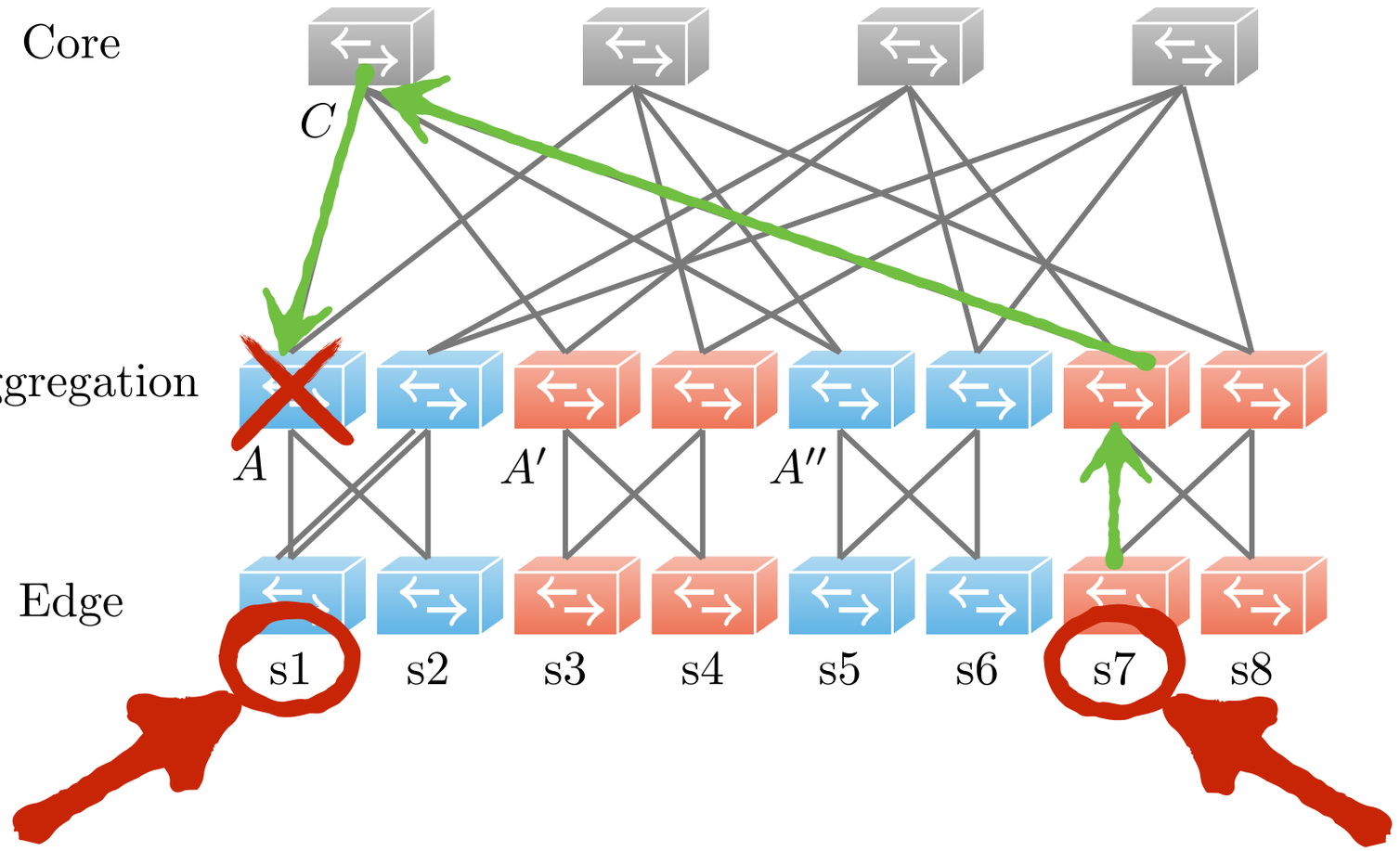
Scalability

Because semantics is formulated in terms of (sparse) matrices, can engineer an efficient implementation



Case Study: Topology

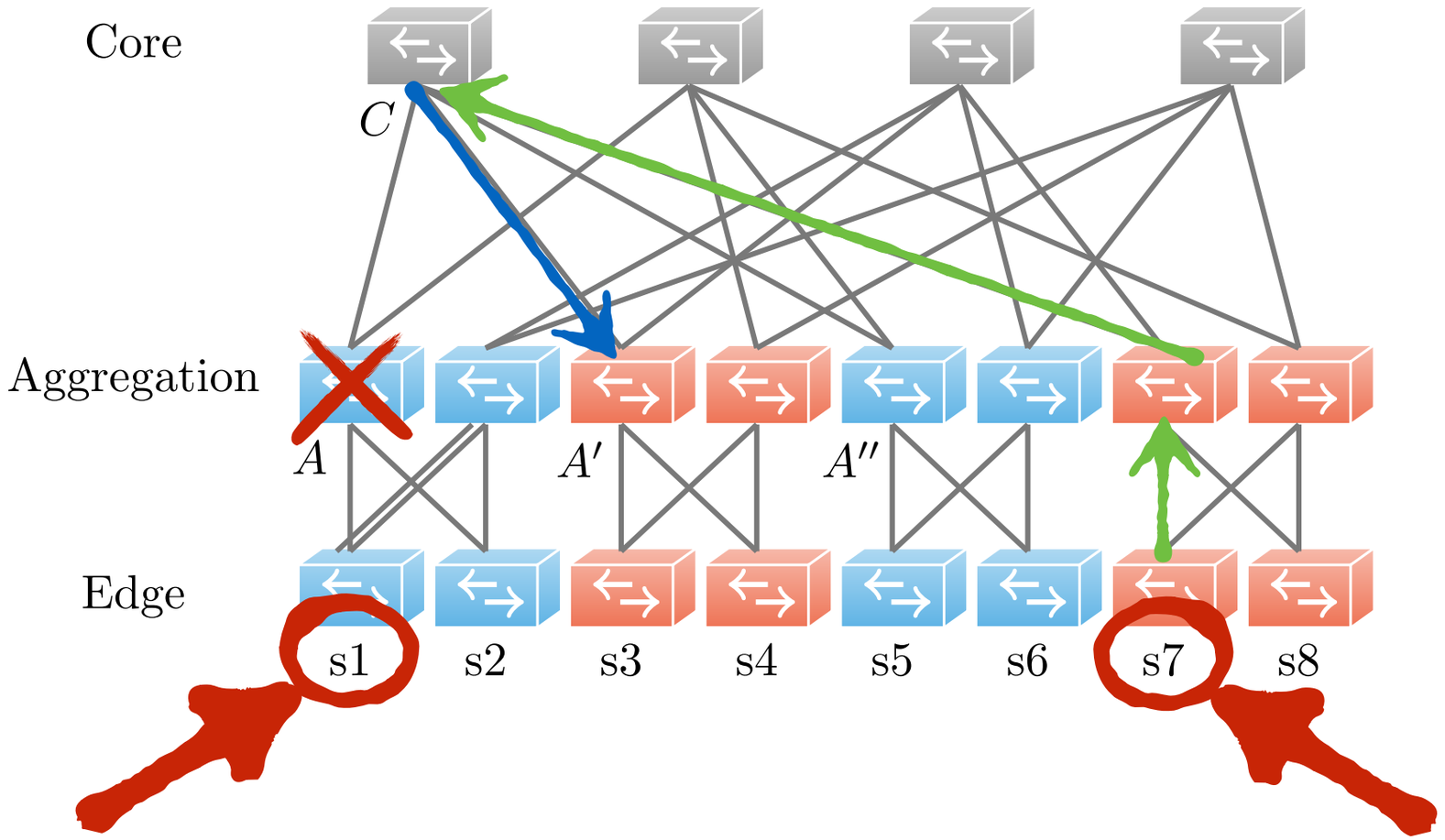
An ABFatTree is much like a regular FatTree



But it provides shorter detours around failures

Case Study: Topology

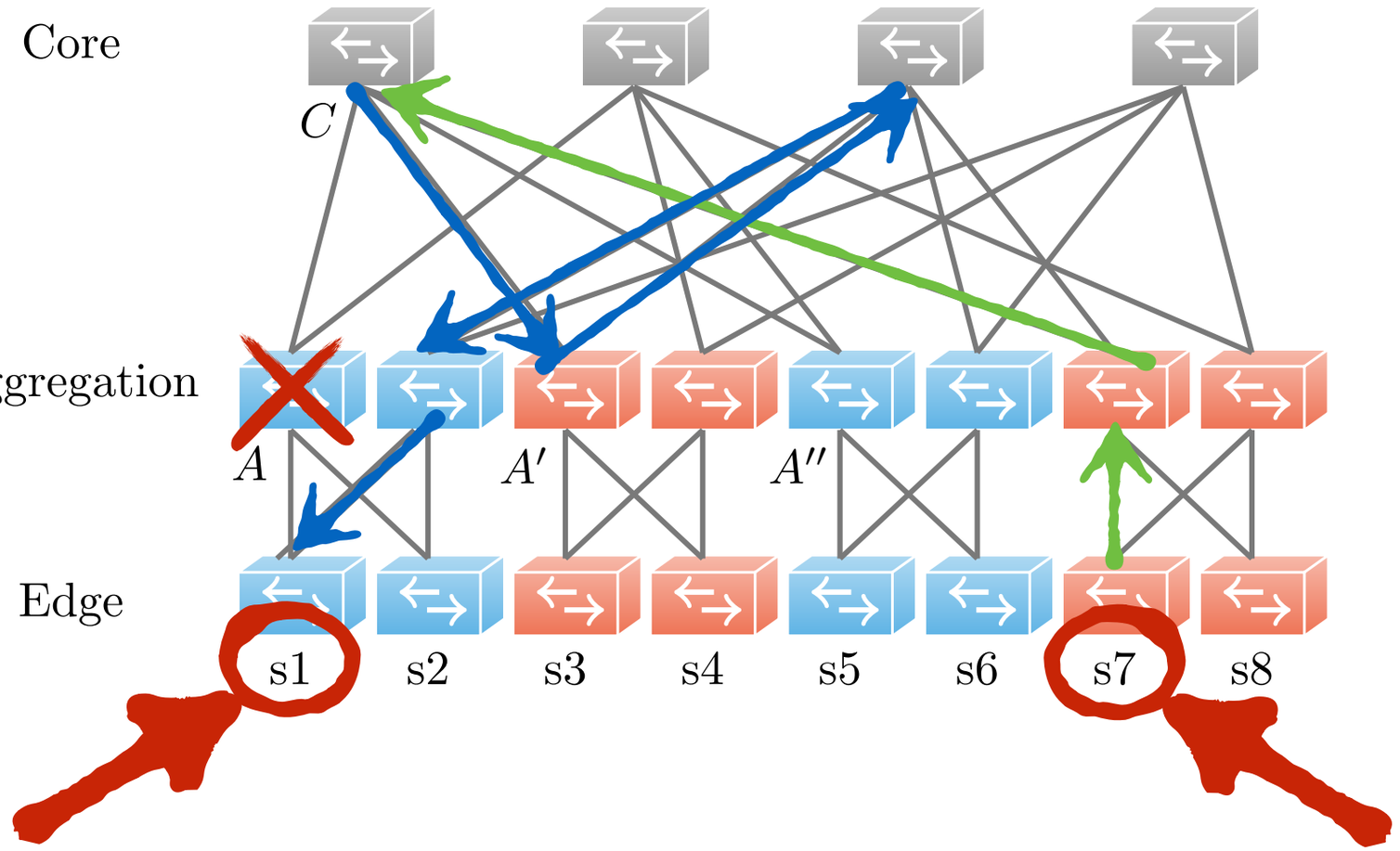
An ABFatTree is much like a regular FatTree



But it provides shorter detours around failures

Case Study: Topology

An ABFatTree is much like a regular FatTree



But it provides shorter detours around failures

Case Study: Routing Schemes

We implemented F10 as a series of 3 refinements

Case Study: Routing Schemes

We implemented F10 as a series of 3 refinements

Complexity

$F10_0$

shortest path routing

$F10_3$

$F10_{3,5}$

Resilience

Case Study: Routing Schemes

We implemented F10 as a series of 3 refinements

Complexity

$F10_0$

shortest path routing

$F10_3$

$F10_0$ + 3-hop rerouting

$F10_{3,5}$

Resilience

Case Study: Routing Schemes

We implemented F10 as a series of 3 refinements

Complexity

$F10_0$

shortest path routing

$F10_3$

$F10_0$ + 3-hop rerouting

$F10_{3,5}$

$F10_3$ + 5-hop rerouting

Resilience

Case Study: Routing Schemes

We implemented F10 as a series of 3 refinements

Complexity

$F10_0$

shortest path routing

$F10_3$

$F10_0$ + 3-hop rerouting

$F10_{3,5}$

$F10_3$ + 5-hop rerouting

Resilience

Requires Extra Packet State

Case Study: k-Resilience

We verified k-resilience using ProbNetKAT

Case Study: k-Resilience

We verified k-resilience using ProbNetKAT

Sophistication of Routing Scheme →

k	$\mathrm{F10}_0$	$\mathrm{F10}_3$	$\mathrm{F10}_{3,5}$
0	✓	✓	✗
1	✗	✓	✗
2	✗	✓	✗
3	✗	✗	✗
4	✗	✗	✗
∞	✗	✗	✗

k = number of failures ✔ = 100% packet delivery

Case Study: k-Resilience

We verified k-resilience using ProbNetKAT

Sophistication of Routing Scheme

k	$\mathrm{F10}_0$	$\mathrm{F10}_3$	$\mathrm{F10}_{3,5}$
0	✓	✓	✗
1	✗	✓	✗
2	✗	✓	✗
3	✗	✗	✗
4	✗	✗	✗
∞	✗	✗	✗

k = number of failures ✔ = 100% packet delivery

Case Study: k-Resilience

We verified k-resilience using ProbNetKAT

Sophistication of Routing Scheme

k	$\mathrm{F10}_0$	$\mathrm{F10}_3$	$\mathrm{F10}_{3,5}$
0	✓	✓	✗
1	✗	✓	✗
2	✗	✗	✗
3	✗	✗	✗
4	✗	✗	✗
5	✗	✗	✗
6	✗	✗	✗

An uninitialized flag caused all packets to be dropped

k = number of failures ✔ = 100% packet delivery

Case Study: k-resilience

After fixing the bug...

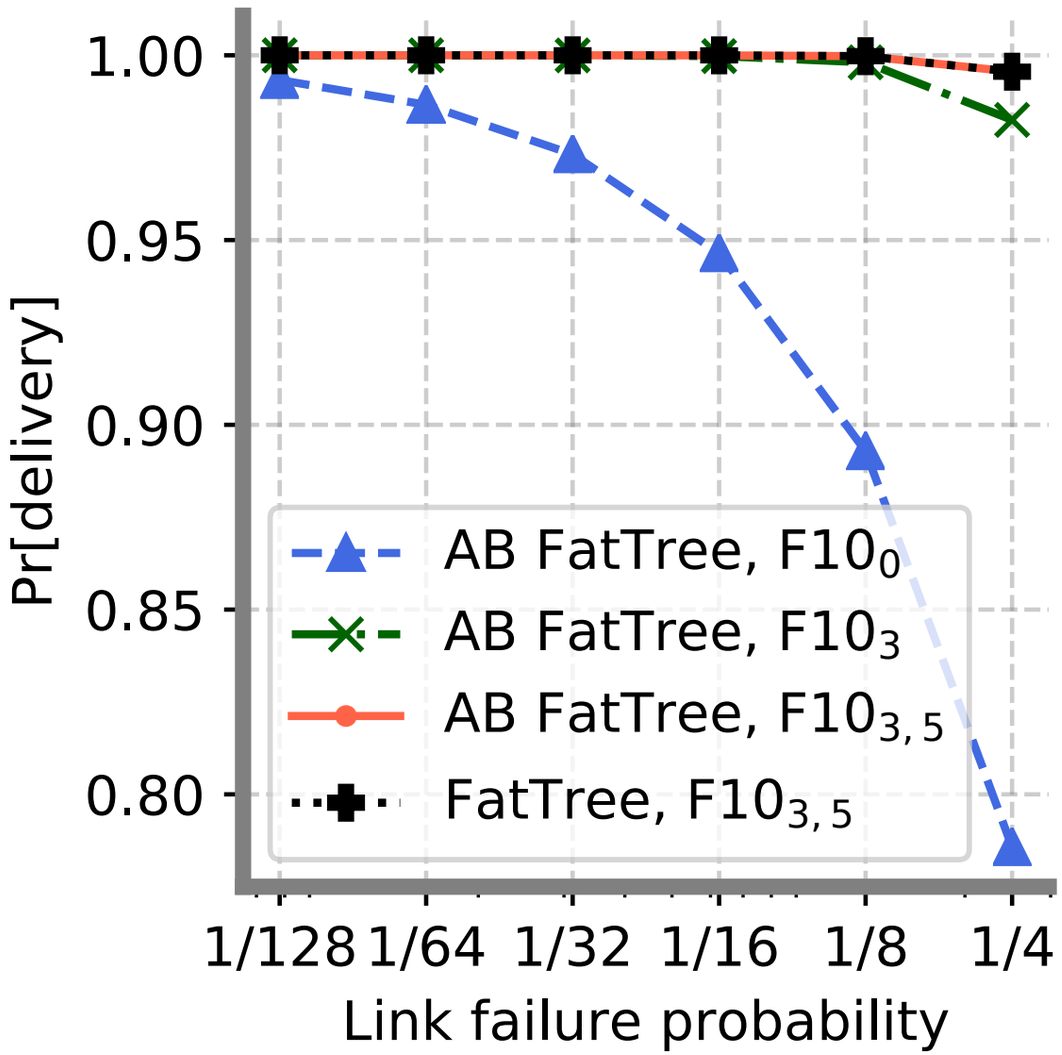
Sophistication of Routing Scheme

k	$\mathrm{F10}_0$	$\mathrm{F10}_3$	$\mathrm{F10}_{3,5}$
0	✓	✓	✓
1	✗	✓	✓
2	✗	✓	✓
3	✗	✗	✓
4	✗	✗	✗
∞	✗	✗	✗

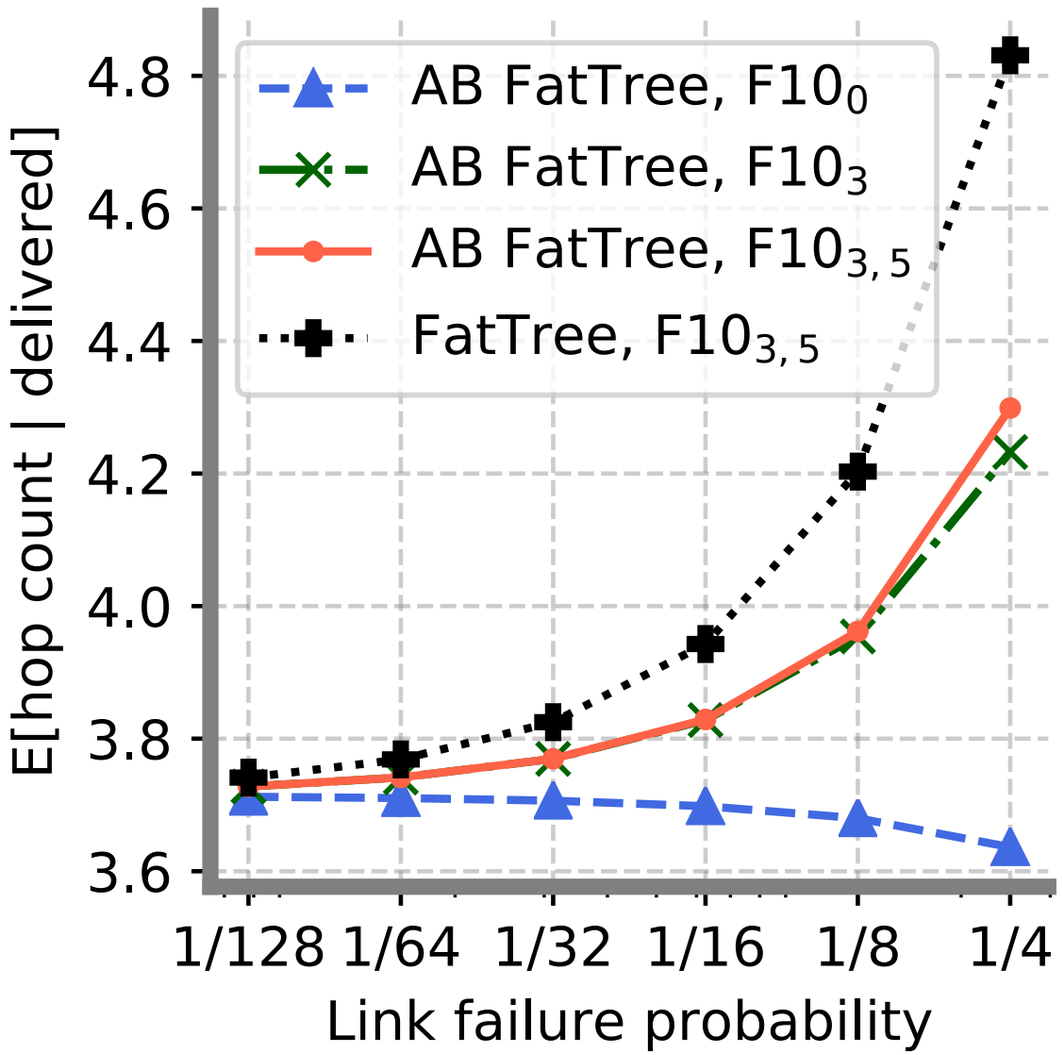
k = number of failures ✔ = 100% packet delivery

Case Study: Results

Resilience



Latency



Wrapping Up...

Conclusion

- Randomized algorithms are a powerful tool for solving problems in many domains, including networking
- Programming languages based on probabilistic semantics are needed to express and reason about these algorithms
- Can build practical tools that analyze quantitative network properties such as peak congestion, resilience to failures, and latency automatically

Reading

- Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. **Probabilistic NetKAT**. In *European Symposium on Programming (ESOP)*, April 2016.

  Steffen Smolka, Praveen Kumar, Nate Foster, Dexter Kozen, and Alexandra Silva. **Cantor Meets Scott: Semantic Foundations for Probabilistic Networks**. In ACM SIGPLAN—SIGACT Symposium on Principles of Programming Languages (POPL), Paris, France, January 2017.

Acknowlegments

- Dexter Kozen (Cornell)
- David Kahn (CMU)
- Konstantinos Mamouras (Rice)
- Mark Reitblatt (Facebook)
- Alexandra Silva (UCL)
- Steffen Smolka (Cornell)
- Justin Hsu (Wisconsin)