Probabilistic Network Verification

Steffen Smolka

Cornell University

Overview:

Network Verification

(In particular, data plane verification)

Network verification has taken off!

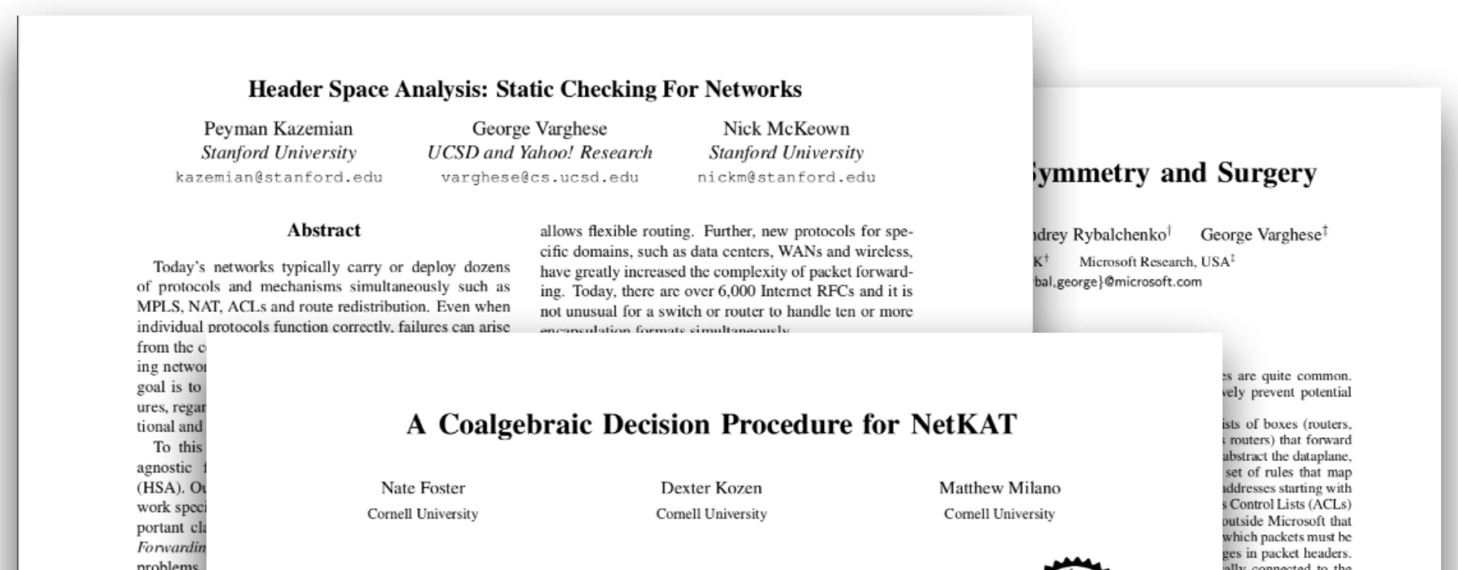
FORWARD NETWORKS

VERIFLOW

Several start-ups

Microsoft Azure

Deployed at big cloud providers


Header Space Analysis: Static Checking For Networks

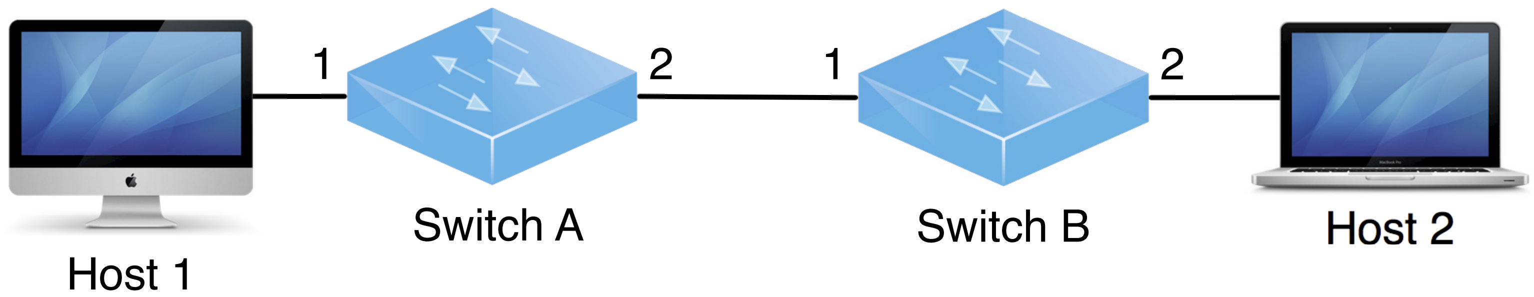
Peyman Kazemian, Stanford University, kazemian@stanford.edu
George Varghese, UCSD and Yahoo! Research, varghese@cs.ucsd.edu
Nick McKeown, Stanford University, nickm@stanford.edu

A Coalgebraic Decision Procedure for NetKAT

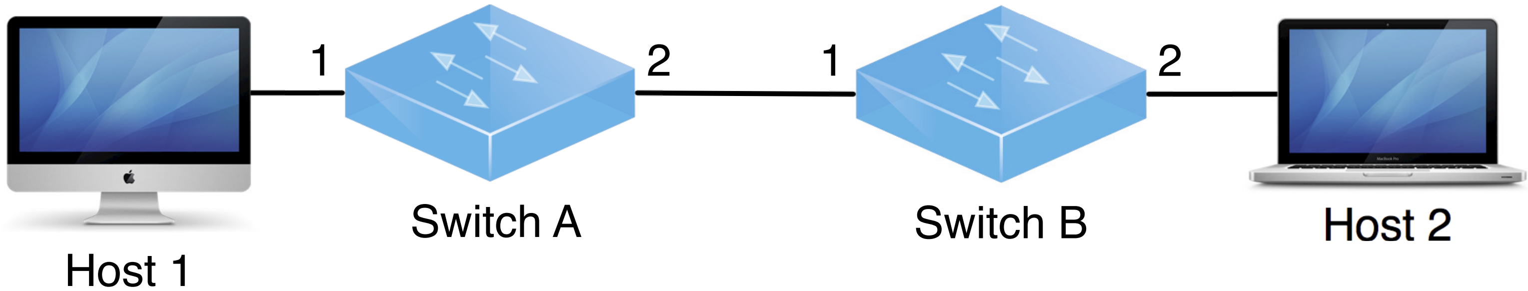
Nate Foster, Cornell University
Dexter Kozen, Cornell University
Matthew Milano, Cornell University

Lots of research

Example: Network Verification



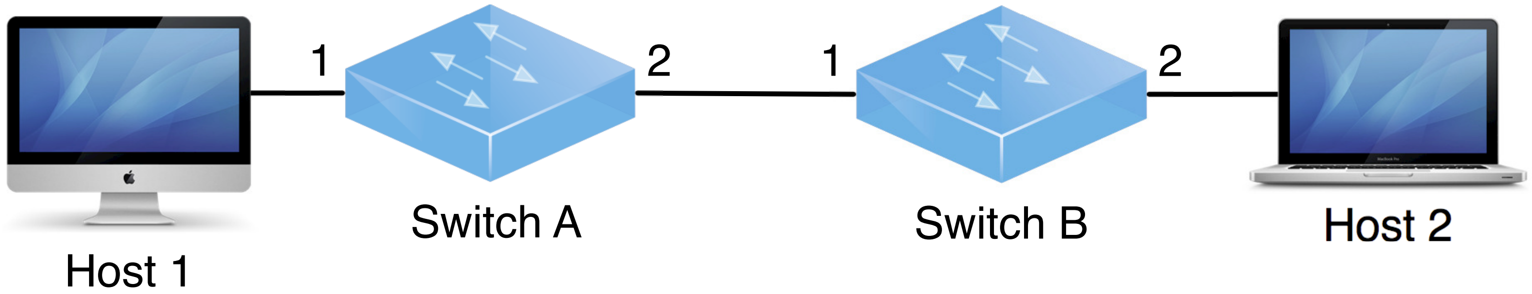
Example: Network Verification



"Are packets routed between hosts?"

"Are ssh packets dropped?"

Example: Network Verification



"Are packets routed between hosts?"

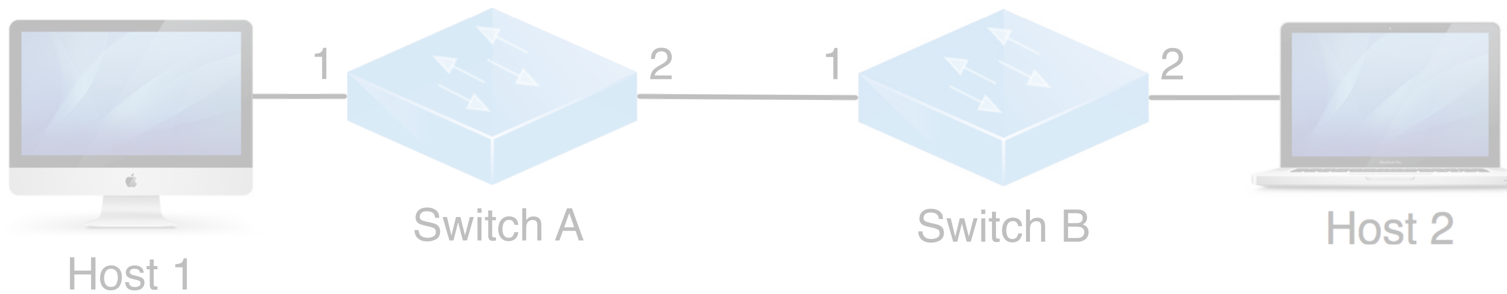
"Are ssh packets dropped?"

Verification Tool

Inputs: Network config & topology + question

Outputs: "Yes" / "No" + counterexample

Example: Network Verification



What's the big deal?

Verification Tool

Inputs: Network config & topology + question

Outputs: "Yes" / "No" + counterexample

The Power of Verification

Guesswork that network will behave correctly

The Power of Verification

Guesswork that network will behave correctly

Mathematical proof of policy compliance

The Power of Verification

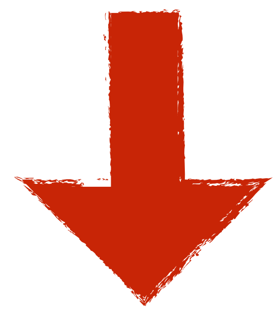
Guesswork that network will behave correctly

Mathematical proof of policy compliance

Consequences:

The Power of Verification

Guesswork that network will behave correctly



Mathematical proof of policy compliance

Consequences:

✦ Bugs can be found before they ever manifest

The Power of Verification

Guesswork that network will behave correctly

Mathematical proof of policy compliance

Consequences:

- Bugs can be found before they ever manifest
- Can change network config with confidence

The Power of Verification

Guesswork that network will behave correctly

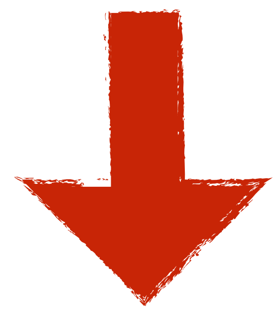
Mathematical proof of policy compliance

Consequences:

- ✦ Bugs can be found before they ever manifest
- ✦ Can change network config with confidence
- ✦ More robust & more efficient network

The Power of Verification

Guesswork that network will behave correctly



Mathematical proof of policy compliance

Consequences:

- ✦ Bugs can be found before they ever manifest
- ✦ Can change network config with confidence
- ✦ More robust & more efficient network
- ✦ Your network operators can sleep better...

Example Network Properties

State of the art tools verify **reachability properties:**

Example Network Properties

State of the art tools verify **reachability properties:**

Waypointing: *every packet traverses a firewall*

Example Network Properties

State of the art tools verify **reachability properties:**

Waypointing: *every packet traverses a firewall*

Isolation: *packets from VLAN 1 cannot enter VLAN2*

Example Network Properties

State of the art tools verify **reachability properties:**

Waypointing: *every packet traverses a firewall*

Isolation: *packets from VLAN 1 cannot enter VLAN2*

Connectivity: *all hosts in the network can communicate*

Example Network Properties

State of the art tools verify **reachability properties:**

Waypointing: *every packet traverses a firewall*

Isolation: *packets from VLAN 1 cannot enter VLAN2*

Connectivity: *all hosts in the network can communicate*

Loop Freedom: *there exist no forwarding loops*

Example Network Properties

State of the art tools verify **reachability properties:**

Waypointing: *every packet traverses a firewall*

Isolation: *packets from VLAN 1 cannot enter VLAN2*

Connectivity: *all hosts in the network can communicate*

Loop Freedom: *there exist no forwarding loops*

Access Control: *internet packets cannot enter the VLAN*

Example Network Properties

State of the art tools verify **reachability properties:**

Waypointing: *every packet traverses a firewall*

Isolation: *packets from VLAN 1 cannot enter VLAN2*

Connectivity: *all hosts in the network can communicate*

Loop Freedom: *there exist no forwarding loops*

Access Control: *internet packets cannot enter the VLAN*

Key Assumption: network behavior is deterministic

Example Network Properties

State of the art tools verify reachability properties:

Waypointing: *every packet traverses a firewall*

Isola

Is it?

Con *cate*

Loop

Access Control: *internet packets cannot enter the VLAN*

Key Assumption: network behavior is deterministic

Probabilistic Network Behavior

Most of the time, networks behave deterministically.

Probabilistic Network Behavior

Most of the time, networks behave deterministically.

But what if...

Probabilistic Network Behavior

Most of the time, networks behave deterministically.

But what if...

- ✦ ... a link or switch fails?

Probabilistic Network Behavior

Most of the time, networks behave deterministically.

But what if...

- ... a link or switch fails?
- ... the network employs resilient routing?

Probabilistic Network Behavior

Most of the time, networks behave deterministically.

But what if...

- ... a link or switch fails?
- ... the network employs resilient routing?
  - *"what's the probability of packet delivery?"*

Probabilistic Network Behavior

Most of the time, networks behave deterministically.

But what if...

- ... a link or switch fails?
- ... the network employs resilient routing?
  - "*what's the probability of packet delivery?*"
  - "*what's the expected path length?*"

Probabilistic Network Behavior

Most of the time, networks behave deterministically.

But what if...

- ... a link or switch fails?
- ... the network employs resilient routing?
  - "*what's the probability of packet delivery?*"
  - "*what's the expected path length?*"
- ... the network employs traffic engineering?

Probabilistic Network Behavior

Most of the time, networks behave deterministically.

But what if...

- ... a link or switch fails?
- ... the network employs resilient routing?
  - *"what's the probability of packet delivery?"*
  - *"what's the expected path length?"*
- ... the network employs traffic engineering?
  - *"what's the expected congestion of this link?"*

Probabilistic Network Verification

Probabilistic NetKAT

Probabilistic NetKAT

A DSL for **programming, modeling & reasoning** about **probabilistic networks**

Probabilistic NetKAT

Nate Foster[1(✉)], Dexter Kozen[1], Konstantinos Mamouras[2], Mark Reitblatt[1], and Alexandra Silva[3]

[1] Cornell University, New York, USA
jnfoster@cs.cornell.edu
[2] University of Pennsylvania, Philadelphia, USA
[3] University College London, London, UK

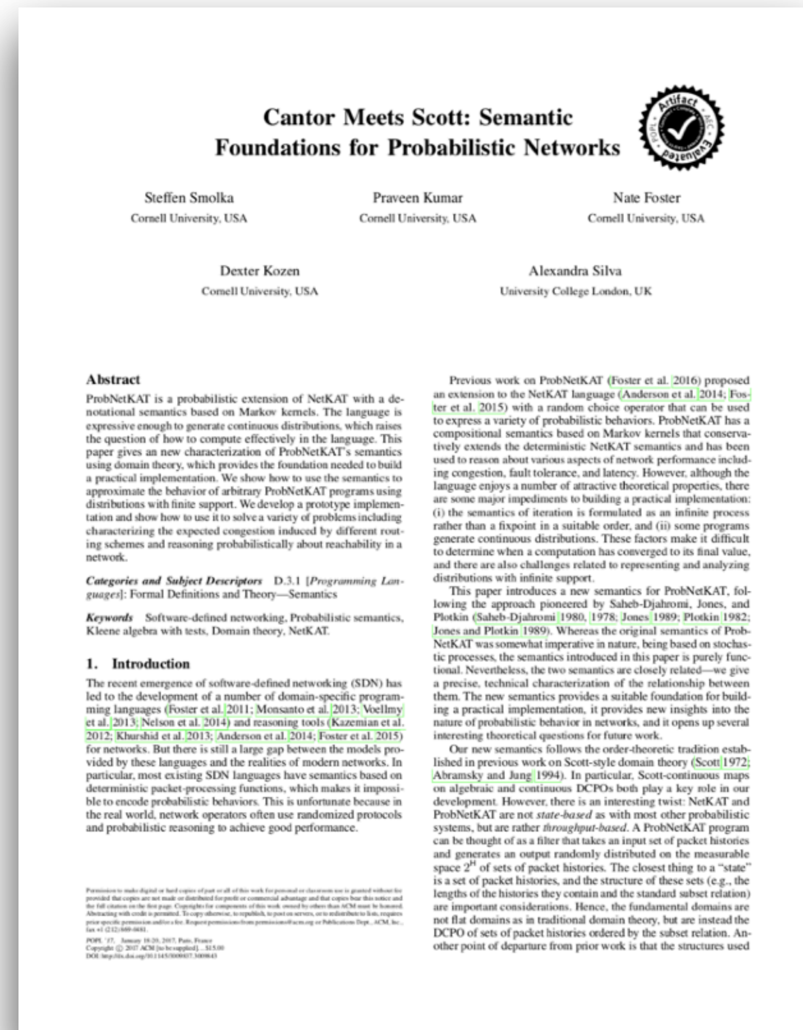
Abstract. This paper presents a new language for network programming based on a probabilistic semantics. We extend the NetKATlanguage with new primitives for expressing probabilistic behaviors and enrich the semantics from one based on deterministic functions to one based on measurable functions on sets of packet histories. We establish fundamental properties of the semantics, prove that it is a conservative extension of the deterministic semantics, show that it satisfies a number of natural equations, and develop a notion of approximation. We present case studies that show how the language can be used to model a diverse collection of scenarios drawn from real-world networks.

1 Introduction

Formal specification and verification of networks has become a reality in recent years with the emergence of network-specific programming languages and property-checking tools. Programming languages like Frenetic [11], Pyretic [35], Maple [51], FlowLog [37], and others are enabling programmers to specify the intended behavior of a network in terms of high-level constructs such as Boolean predicates and functions on packets. Verification tools like Header Space Analysis [21], VeriFlow [22], and NetKAT [12] are making it possible to check properties such as connectivity, loop freedom, and traffic isolation automatically.

However, despite many notable advances, these frameworks all have a fundamental limitation: they model network behavior in terms of deterministic packet-processing functions. This approach works well enough in settings where the network functionality is simple, or where the properties of interest only concern the forwarding paths used to carry traffic. But it does not provide satisfactory accounts of more complicated situations that often arise in practice:

- **Congestion:** the network operator wishes to calculate the expected degree of congestion on each link given a model of the demands for traffic.
- **Failure:** the network operator wishes to calculate the probability that packets will be delivered to their destination, given that devices and links fail with a certain probability.



Cantor Meets Scott: Semantic Foundations for Probabilistic Networks

Steffen Smolka, Cornell University, USA
Praveen Kumar, Cornell University, USA
Nate Foster, Cornell University, USA
Dexter Kozen, Cornell University, USA
Alexandra Silva, University College London, UK

Abstract

ProbNetKAT is a probabilistic extension of NetKAT with a denotational semantics based on Markov kernels. The language is expressive enough to generate continuous distributions, which raises the question of how to compute effectively in the language. This paper gives an new characterization of ProbNetKAT's semantics using domain theory, which provides the foundation needed to build a practical implementation. We show how to use the semantics to approximate the behavior of arbitrary ProbNetKAT programs using distributions with finite support. We develop a prototype implementation and show how to use it to solve a variety of problems including characterizing the expected congestion induced by different routing schemes and reasoning probabilistically about reachability in a network.

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal Definitions and Theory—Semantics

Keywords Software-defined networking, Probabilistic semantics, Kleene algebra with tests, Domain theory, NetKAT.

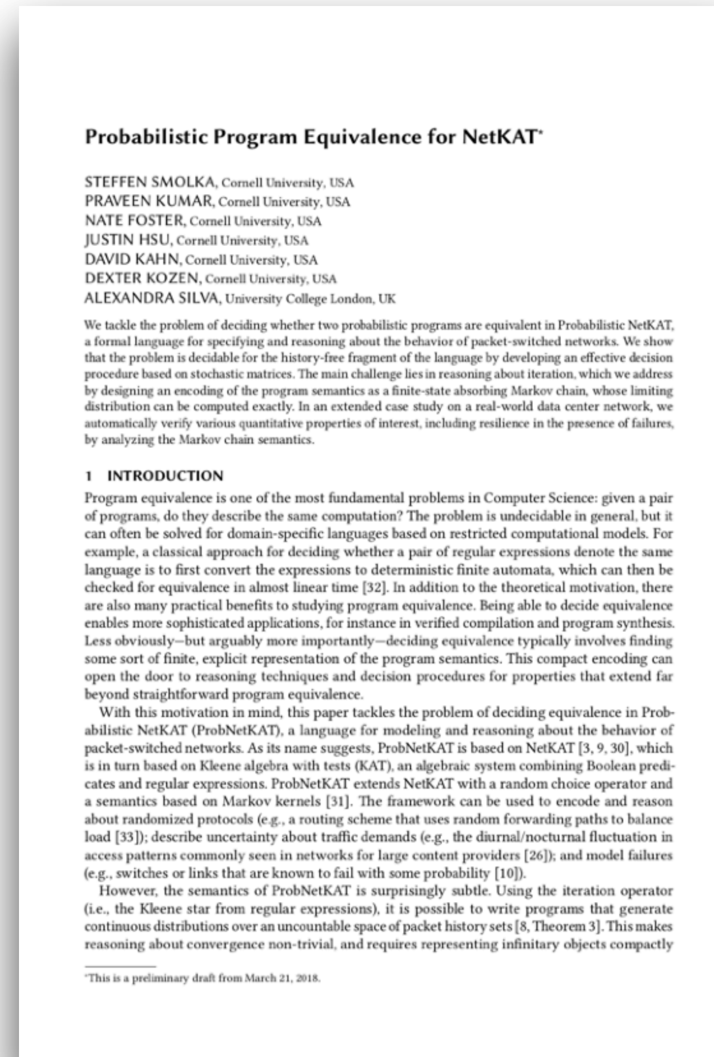
1. Introduction

The recent emergence of software-defined networking (SDN) has led to the development of a number of domain-specific programming languages (Foster et al. 2011; Monsanto et al. 2013; Voellmy et al. 2013; Nelson et al. 2014) and reasoning tools (Kazemian et al. 2012; Khurshid et al. 2013; Anderson et al. 2014; Foster et al. 2015) for networks. But there is still a large gap between the models provided by these languages and the realities of modern networks. In particular, most existing SDN languages have semantics based on deterministic packet-processing functions, which makes it impossible to encode probabilistic behaviors. This is unfortunate because in the real world, network operators often use randomized protocols and probabilistic reasoning to achieve good performance.

Previous work on ProbNetKAT (Foster et al. 2016) proposed an extension to the NetKAT language (Anderson et al. 2014; Foster et al. 2015) with a random choice operator that can be used to express a variety of probabilistic behaviors. ProbNetKAT has a compositional semantics based on Markov kernels that conservatively extends the deterministic NetKAT semantics and has been used to reason about various aspects of network performance including congestion, fault tolerance, and latency. However, although the language enjoys a number of attractive theoretical properties, there are some major impediments to building a practical implementation: (i) the semantics of iteration is formulated as an infinite process rather than a fixpoint in a suitable order, and (ii) some programs generate continuous distributions. These factors make it difficult to determine when a computation has converged to its final value, and there are also challenges related to representing and analyzing distributions with infinite support.

This paper introduces a new semantics for ProbNetKAT, following the approach pioneered by Saheb-Djahromi, Jones, and Plotkin (Saheb-Djahromi 1980, 1978; Jones 1989; Plotkin 1982; Jones and Plotkin 1989). Whereas the original semantics of ProbNetKAT was somewhat imperative in nature, being based on stochastic processes, the semantics introduced in this paper is purely functional. Nevertheless, the two semantics are closely related—we give a precise, technical characterization of the relationship between them. The new semantics provides a suitable foundation for building a practical implementation, it provides new insights into the nature of probabilistic behavior in networks, and it opens up several interesting theoretical questions for future work.

Our new semantics follows the order-theoretic tradition established in previous work on Scott-style domain theory (Scott 1972; Abramsky and Jung 1994). In particular, Scott-continuous maps on algebraic and continuous DCPOs both play a key role in our development. However, there is an interesting twist: NetKAT and ProbNetKAT are not *state-based* as with most other probabilistic systems, but are rather *throughput-based*. A ProbNetKAT program can be thought of as a filter that takes an input set of packet histories and generates an output randomly distributed on the measurable space 2^H of sets of packet histories. The closest thing to a "state" is a set of packet histories, and the structure of these sets (e.g., the lengths of the histories they contain and the standard subset relation) are important considerations. Hence, the fundamental domains are not flat domains as in traditional domain theory, but are instead the DCPO of sets of packet histories ordered by the subset relation. Another point of departure from prior work is that the structures used



Probabilistic Program Equivalence for NetKAT*

STEFFEN SMOLKA, Cornell University, USA
PRAVEEN KUMAR, Cornell University, USA
NATE FOSTER, Cornell University, USA
JUSTIN HSU, Cornell University, USA
DAVID KAHN, Cornell University, USA
DEXTER KOZEN, Cornell University, USA
ALEXANDRA SILVA, University College London, UK

We tackle the problem of deciding whether two probabilistic programs are equivalent in Probabilistic NetKAT, a formal language for specifying and reasoning about the behavior of packet-switched networks. We show that the problem is decidable for the history-free fragment of the language by developing an effective decision procedure based on stochastic matrices. The main challenge lies in reasoning about iteration, which we address by designing an encoding of the program semantics as a finite-state absorbing Markov chain, whose limiting distribution can be computed exactly. In an extended case study on a real-world data center network, we automatically verify various quantitative properties of interest, including resilience in the presence of failures, by analyzing the Markov chain semantics.

1 INTRODUCTION

Program equivalence is one of the most fundamental problems in Computer Science: given a pair of programs, do they describe the same computation? The problem is undecidable in general, but it can often be solved for domain-specific languages based on restricted computational models. For example, a classical approach for deciding whether a pair of regular expressions denote the same language is to first convert the expressions to deterministic finite automata, which can then be checked for equivalence in almost linear time [32]. In addition to the theoretical motivation, there are also many practical benefits to studying program equivalence. Being able to decide equivalence enables more sophisticated applications, for instance in verified compilation and program synthesis. Less obviously—but arguably more importantly—deciding equivalence typically involves finding some sort of finite, explicit representation of the program semantics. This compact encoding can open the door to reasoning techniques and decision procedures for properties that extend far beyond straightforward program equivalence.

With this motivation in mind, this paper tackles the problem of deciding equivalence in Probabilistic NetKAT (ProbNetKAT), a language for modeling and reasoning about the behavior of packet-switched networks. As its name suggests, ProbNetKAT is based on NetKAT [3, 9, 30], which is in turn based on Kleene algebra with tests (KAT), an algebraic system combining Boolean predicates and regular expressions. ProbNetKAT extends NetKAT with a random choice operator and a semantics based on Markov kernels [31]. The framework can be used to encode and reason about randomized protocols (e.g., a routing scheme that uses random forwarding paths to balance load [33]); describe uncertainty about traffic demands (e.g., the diurnal/nocturnal fluctuation in access patterns commonly seen in networks for large content providers [26]); and model failures (e.g., switches or links that are known to fail with some probability [10]).

However, the semantics of ProbNetKAT is surprisingly subtle. Using the iteration operator (i.e., the Kleene star from regular expressions), it is possible to write programs that generate continuous distributions over an uncountable space of packet history sets [8, Theorem 3]. This makes reasoning about convergence non-trivial, and requires representing infinitary objects compactly

*This is a preliminary draft from March 21, 2018.

[ESOP '16]

[POPL '17]

What ProbNetKAT can do

What ProbNetKAT can do

Verify reachability properties

What ProbNetKAT can do

Verify reachability properties

- ✦ but for probabilistic networks

What ProbNetKAT can do

Verify reachability properties

- ✦ but for probabilistic networks

Verify fault tolerance

What ProbNetKAT can do

Verify reachability properties

- ✦ but for probabilistic networks

Verify fault tolerance

- ✦ k-resilience

What ProbNetKAT can do

Verify reachability properties

- ✦ but for probabilistic networks

Verify fault tolerance

- ✦ k-resilience
- ✦ "*is scheme A is more resilient than scheme B?*"

What ProbNetKAT can do

Verify reachability properties

- ✦ but for probabilistic networks

Verify fault tolerance

- ✦ k-resilience
- ✦ "*is scheme A is more resilient than scheme B?*"
- ✦ probability of packet delivery

What ProbNetKAT can do

Verify reachability properties

- ✦ but for probabilistic networks

Verify fault tolerance

- ✦ k-resilience
- ✦ "*is scheme A is more resilient than scheme B?*"
- ✦ probability of packet delivery

Compute quantitative network metrics

What ProbNetKAT can do

Verify reachability properties

- ✦ but for probabilistic networks

Verify fault tolerance

- ✦ k-resilience
- ✦ "*is scheme A is more resilient than scheme B?*"
- ✦ probability of packet delivery

Compute quantitative network metrics

- ✦ "*expected number of hops?*"

What ProbNetKAT can do

Verify reachability properties

- ✦ but for probabilistic networks

Verify fault tolerance

- ✦ k-resilience
- ✦ "*is scheme A is more resilient than scheme B?*"
- ✦ probability of packet delivery

Compute quantitative network metrics

- ✦ "*expected number of hops?*"
- ✦ "*expected link congestion?*"

What ProbNetKAT can do

Verify reachability properties

- but for probabilistic networks

Verify fault tolerance

- k-resilience
- *"is scheme A is more resilient than scheme B?"*
- probability of packet delivery

Compute quantitative network metrics

- *"expected number of hops?"*
- *"expected link congestion?"*
- computes analytical solution, not approximation

Case Study

F10: A Fault-Tolerant Engineered Network

Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, Thomas Anderson
University of Washington

[NSDI'13]

Case Study

F10: A Fault-Tolerant Engineered Network

Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, Thomas Anderson
University of Washington

[NSDI'13]

Motivation

- ✦ short-term failures in data centers are common

Case Study

F10: A Fault-Tolerant Engineered Network

Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, Thomas Anderson
University of Washington

[NSDI'13]

Motivation

- ✦ short-term failures in data centers are common
- ✦ application performance suffers

Case Study

F10: A Fault-Tolerant Engineered Network
Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, Thomas Anderson
University of Washington

[NSDI'13]

Motivation

- ✦ short-term failures in data centers are common
- ✦ application performance suffers
- ✦ despite 1:1 redundancy!

Case Study

F10: A Fault-Tolerant Engineered Network

Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, Thomas Anderson
University of Washington

[NSDI'13]

Motivation

- short-term failures in data centers are common
- application performance suffers
- despite 1:1 redundancy!

Case Study

F10: A Fault-Tolerant Engineered Network

Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, Thomas Anderson

University of Washington

[NSDI'13]

Motivation

- ✦ short-term failures in data centers are common
- ✦ application performance suffers
- ✦ despite 1:1 redundancy!

Solution

Case Study

F10: A Fault-Tolerant Engineered Network

Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, Thomas Anderson

University of Washington

[NSDI'13]

Motivation

- ✦ short-term failures in data centers are common
- ✦ application performance suffers
- ✦ despite 1:1 redundancy!

Solution

- ✦ detect failures of neighboring links & switches...

Case Study

F10: A Fault-Tolerant Engineered Network

Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, Thomas Anderson

University of Washington

[NSDI'13]

Motivation

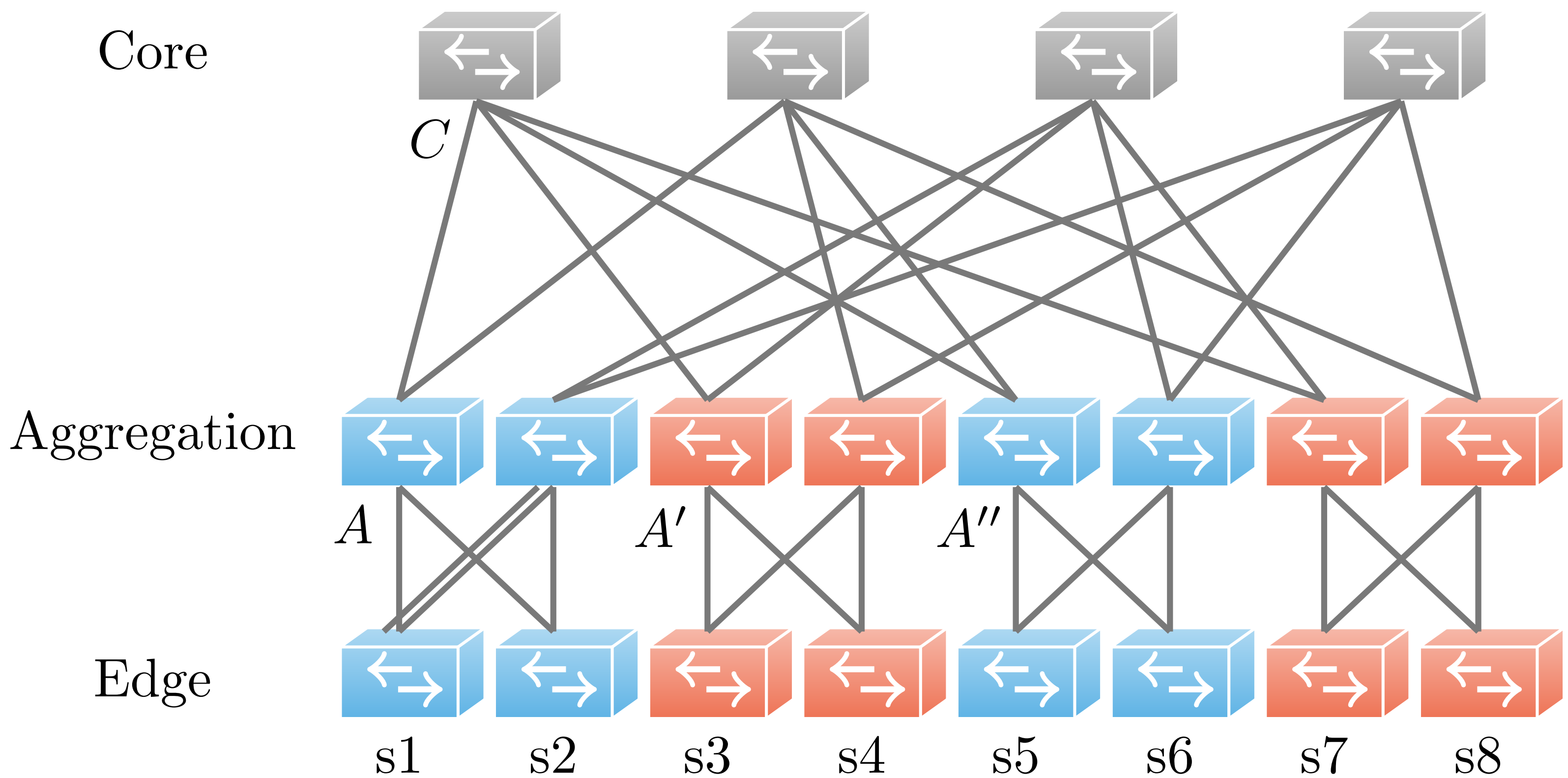
- ✦ short-term failures in data centers are common
- ✦ application performance suffers
- ✦ despite 1:1 redundancy!

Solution

- ✦ detect failures of neighboring links & switches...
- ✦ ...and route around them

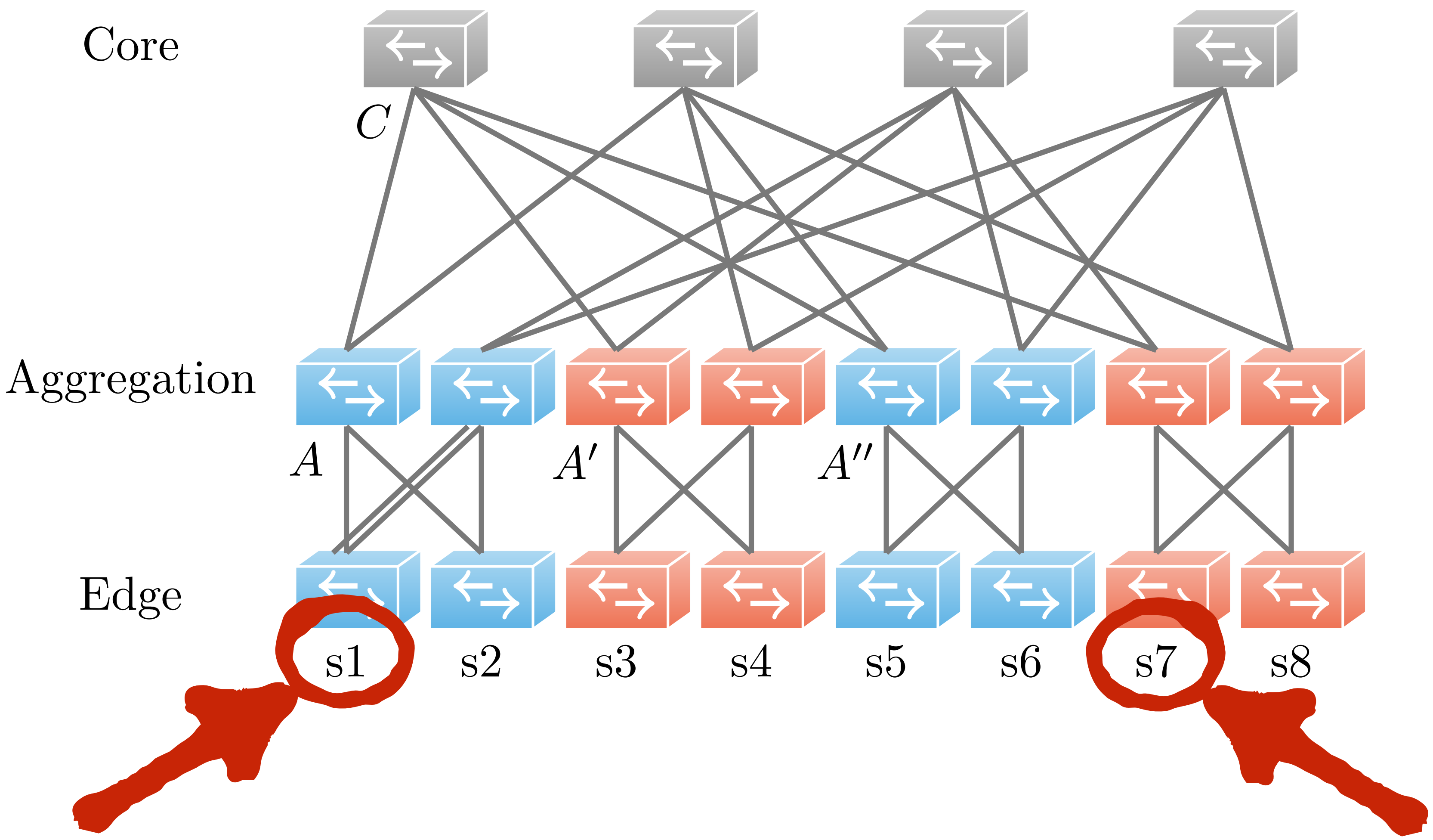
Case Study: Topology

An ABFatTree is much like a regular FatTree



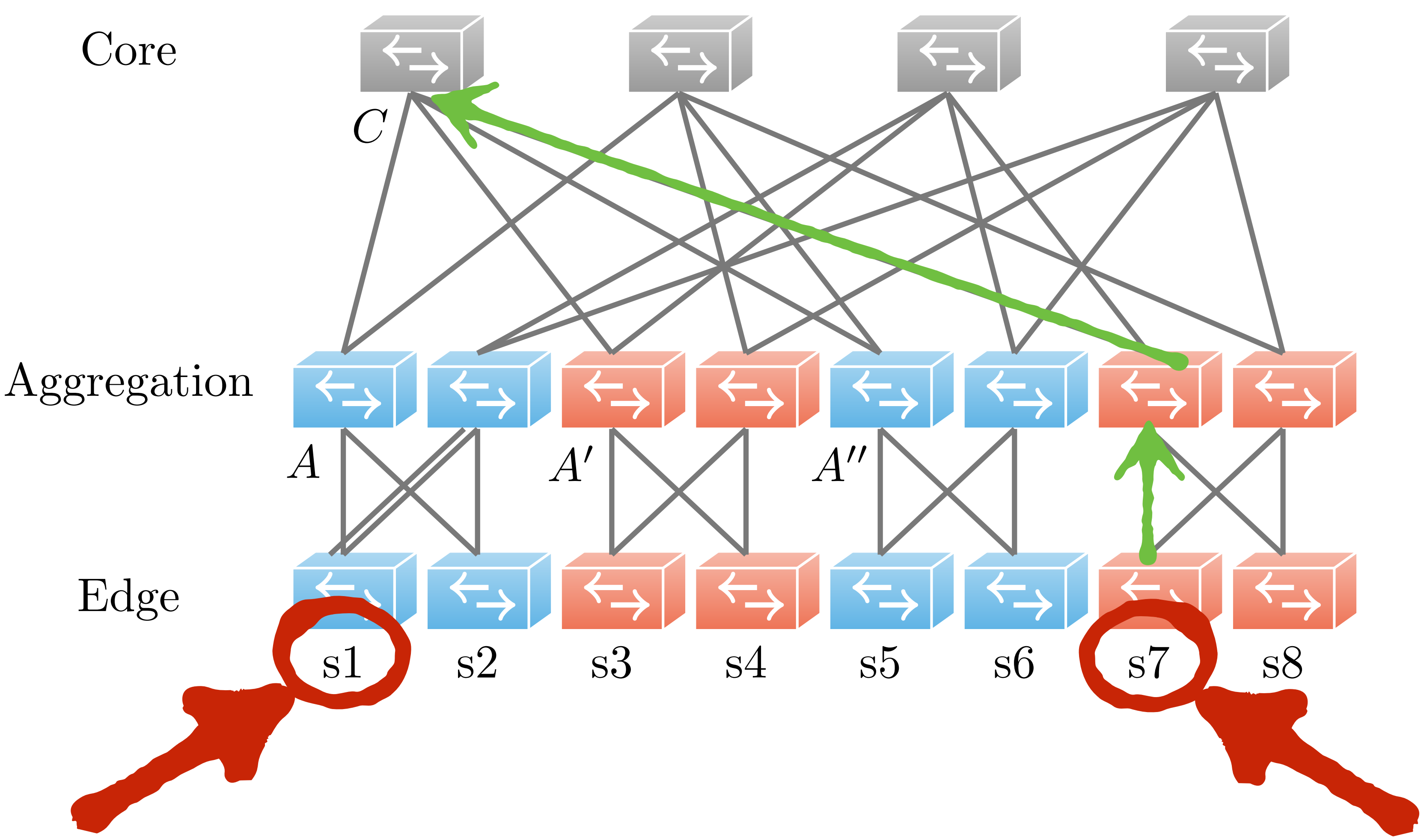
Case Study: Topology

An ABFatTree is much like a regular FatTree



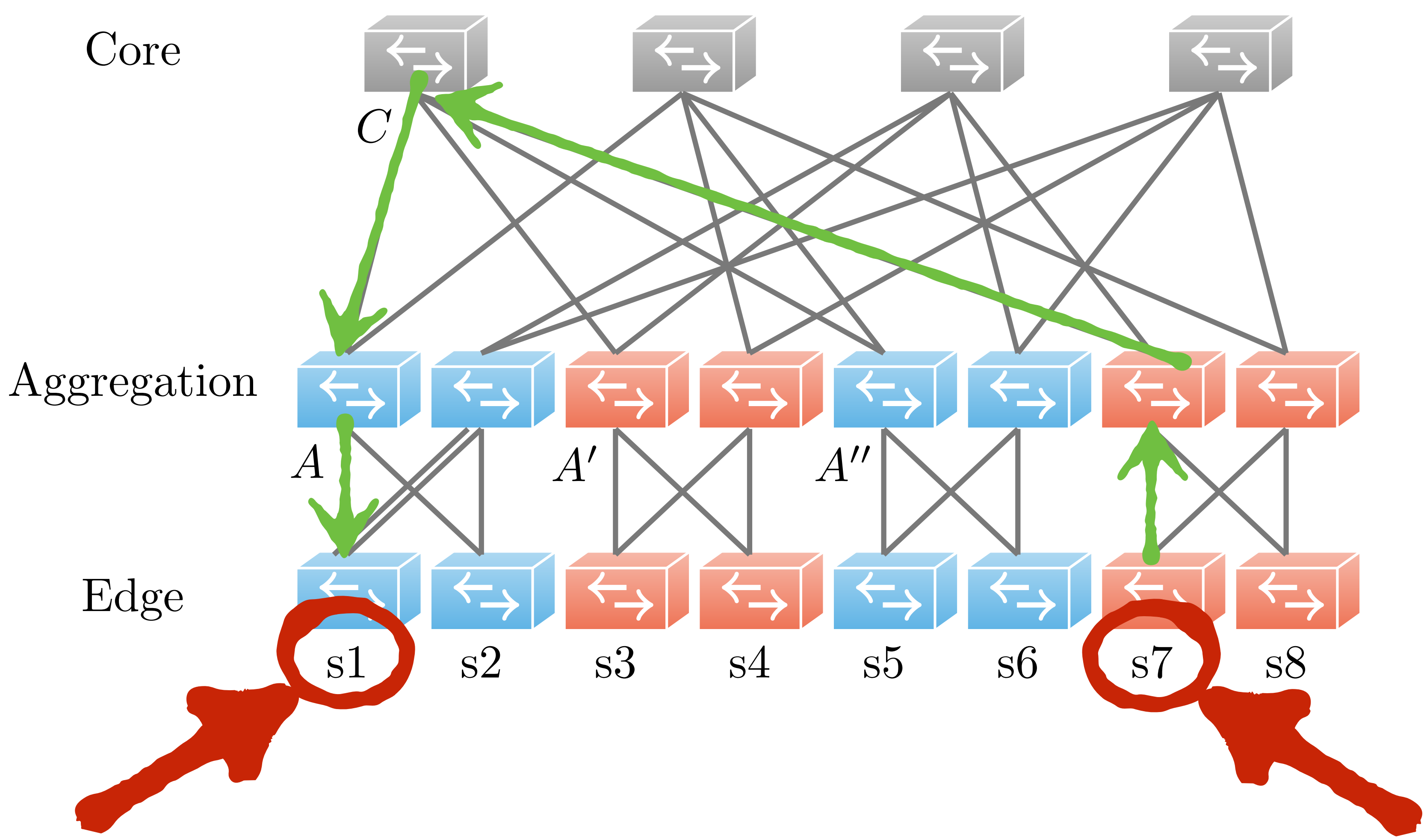
Case Study: Topology

An ABFatTree is much like a regular FatTree



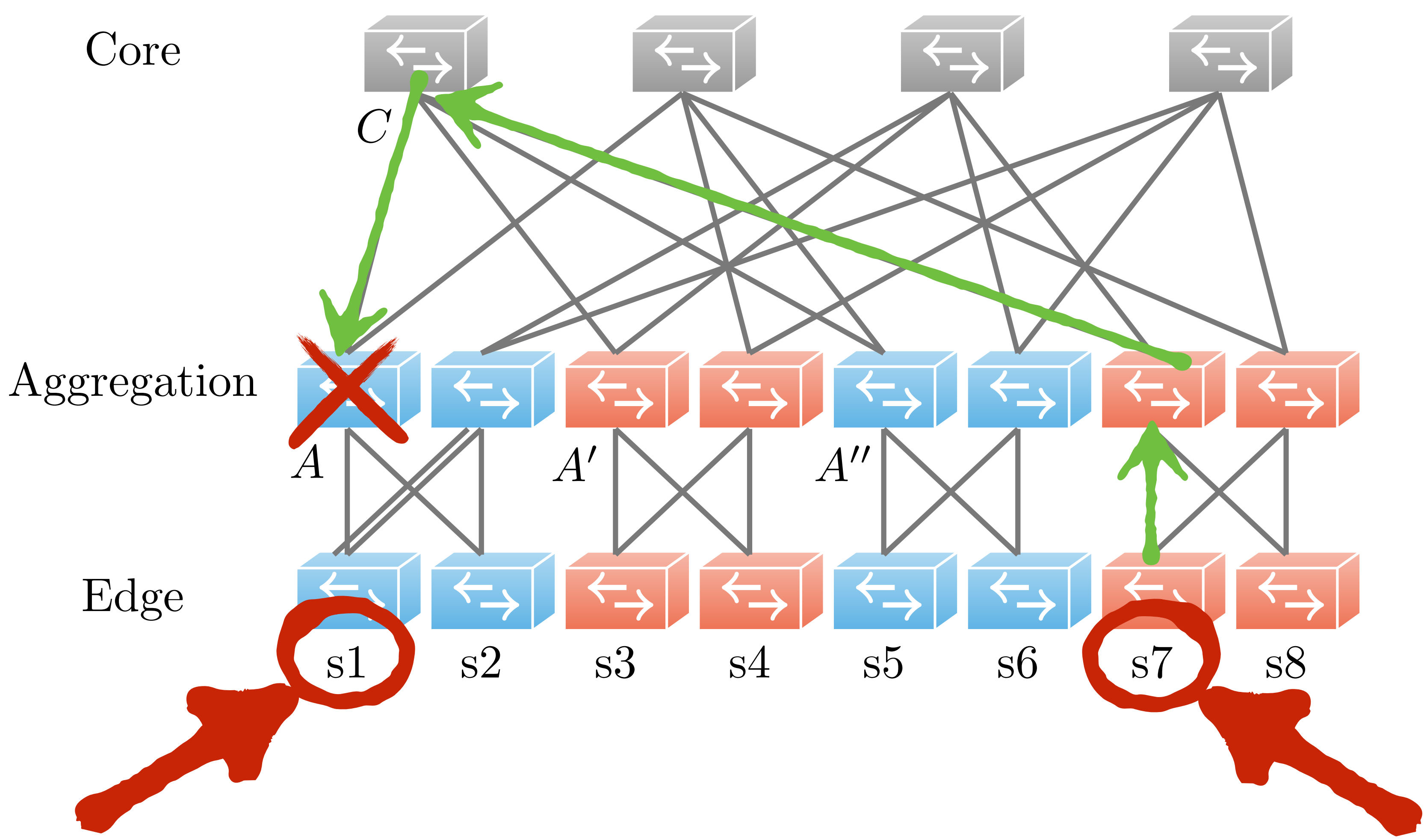
Case Study: Topology

An ABFatTree is much like a regular FatTree



Case Study: Topology

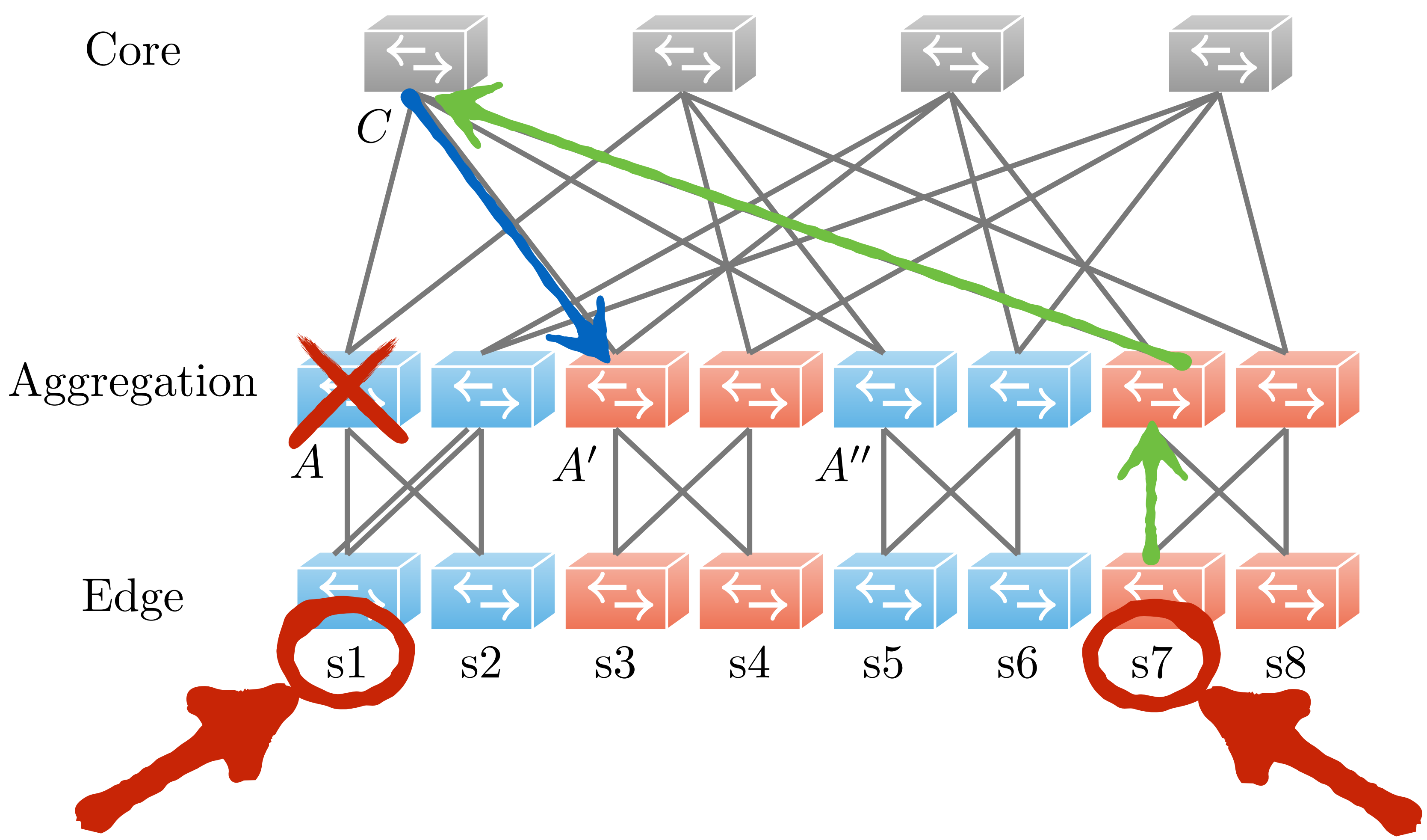
An ABFatTree is much like a regular FatTree



But it provides shorter detours around failures

Case Study: Topology

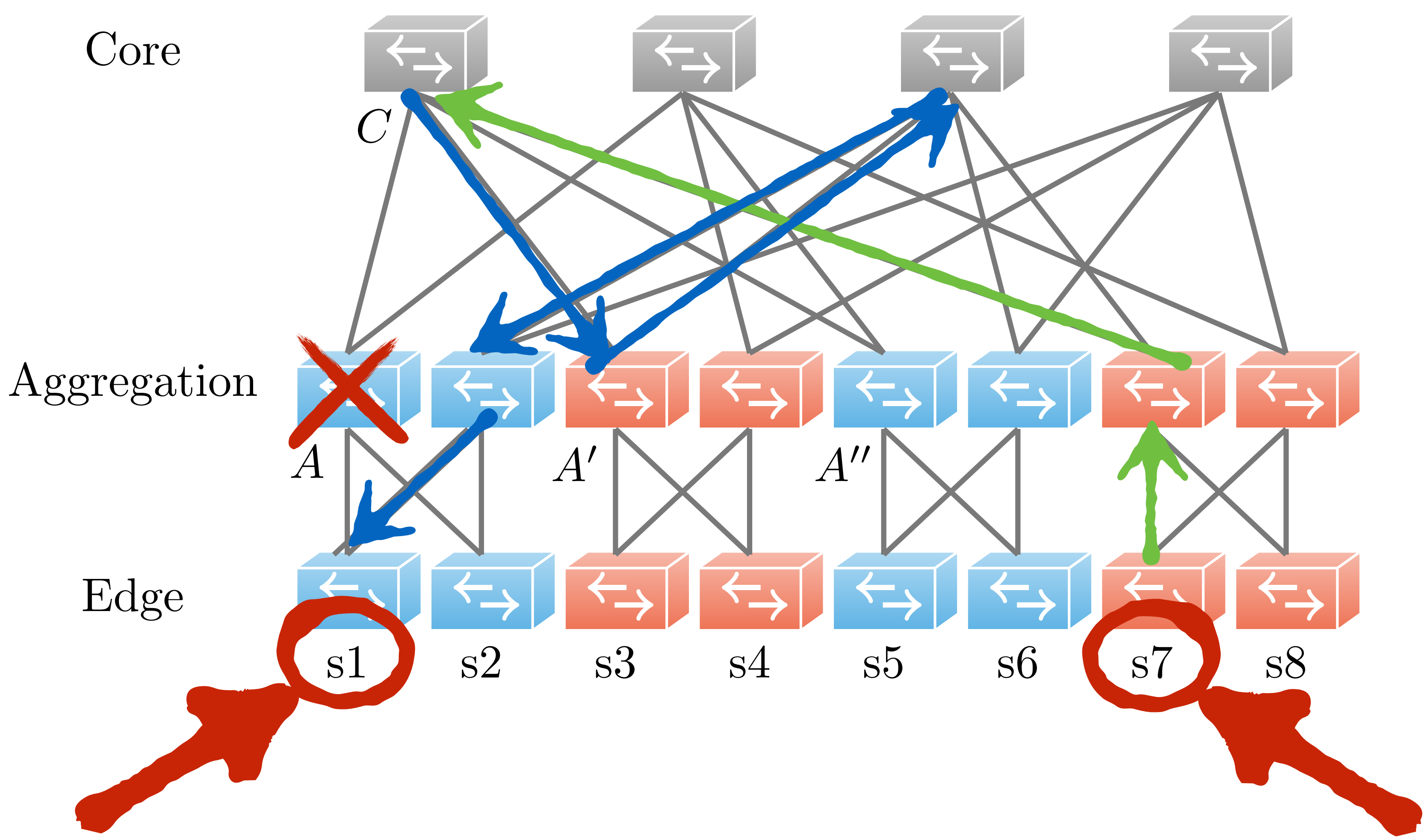
An ABFatTree is much like a regular FatTree



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

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

Requires State

Resilience

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	$F10_0$	$F10_3$	$F10_{3,5}$
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	✗	✓	✗
			✗
			✗
			✗
			✗

An uninitialized flag caused all packets to be dropped

k = number of failures

✔ = 100% packet delivery

Case Study: k-resilience

After fixing the bug...

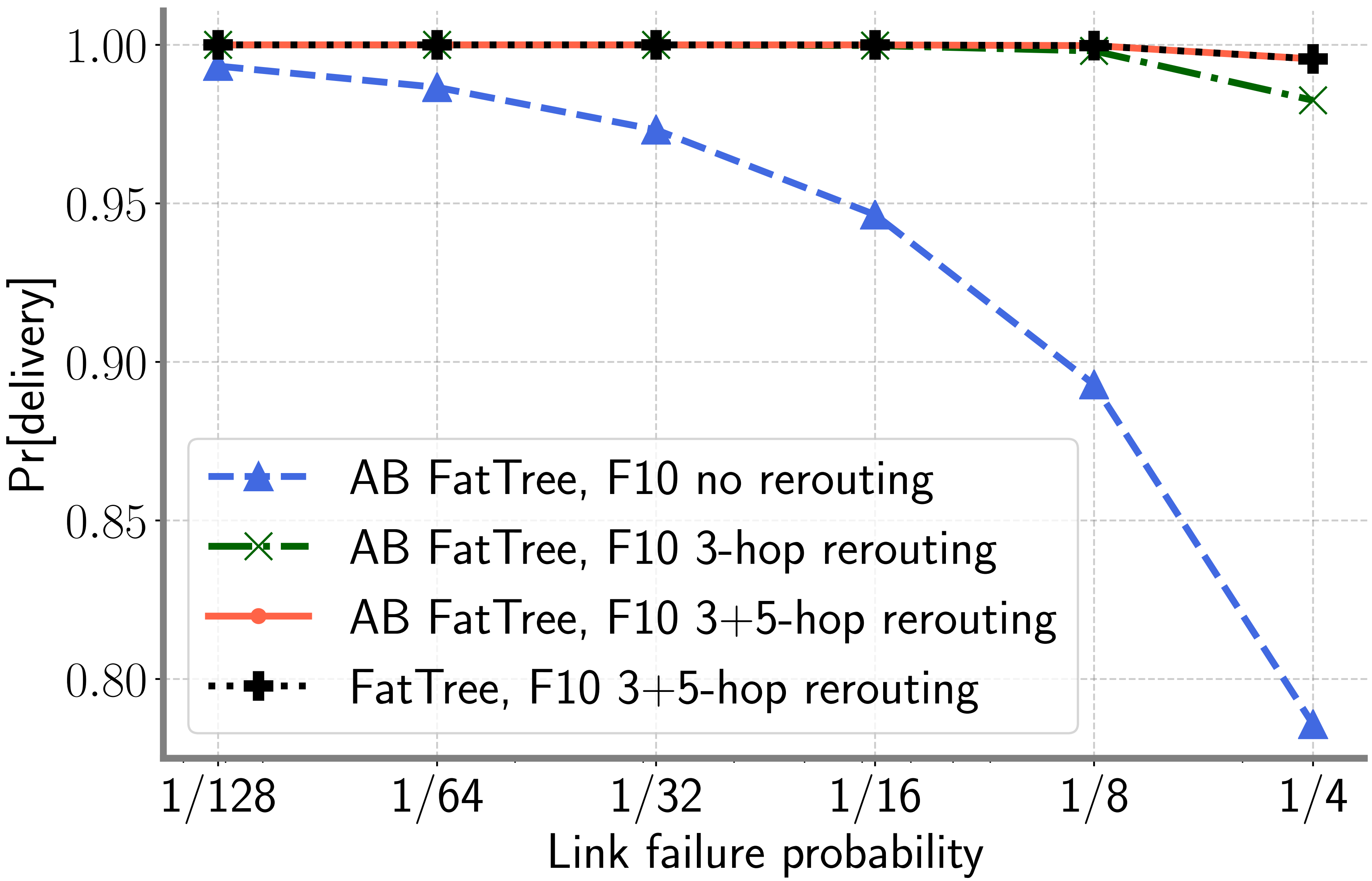
Sophistication of Routing Scheme →

k	$F10_0$	$F10_3$	$F10_{3,5}$
0	✓	✓	✓
1	✗	✓	✓
2	✗	✓	✓
3	✗	✗	✓
4	✗	✗	✗
∞	✗	✗	✗

k = number of failures ✔ = 100% packet delivery

Case Study: probability of delivery

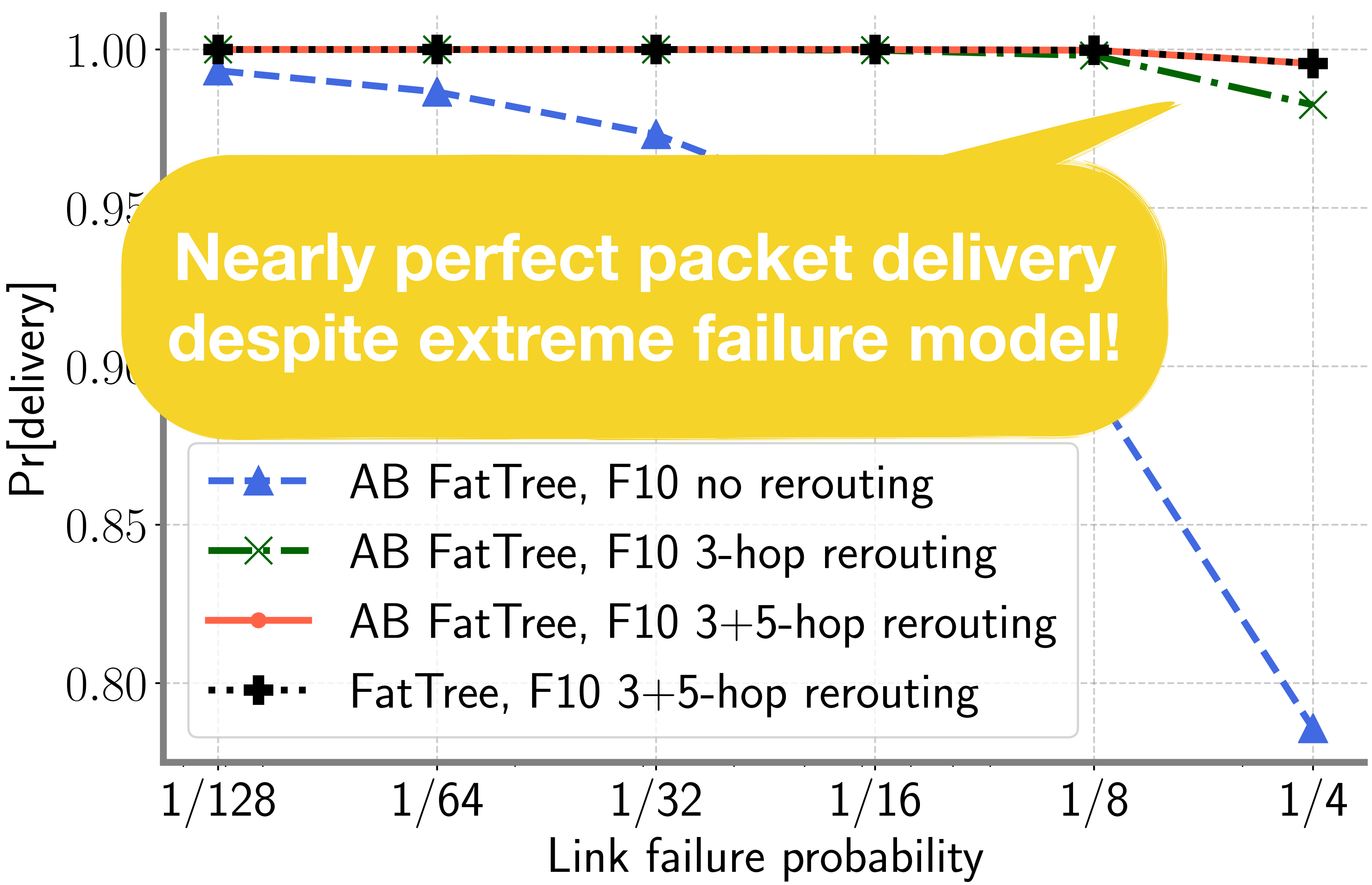
We evaluated packet loss when link failures increase



Dramatic improvement when using rerouting

Case Study: probability of delivery

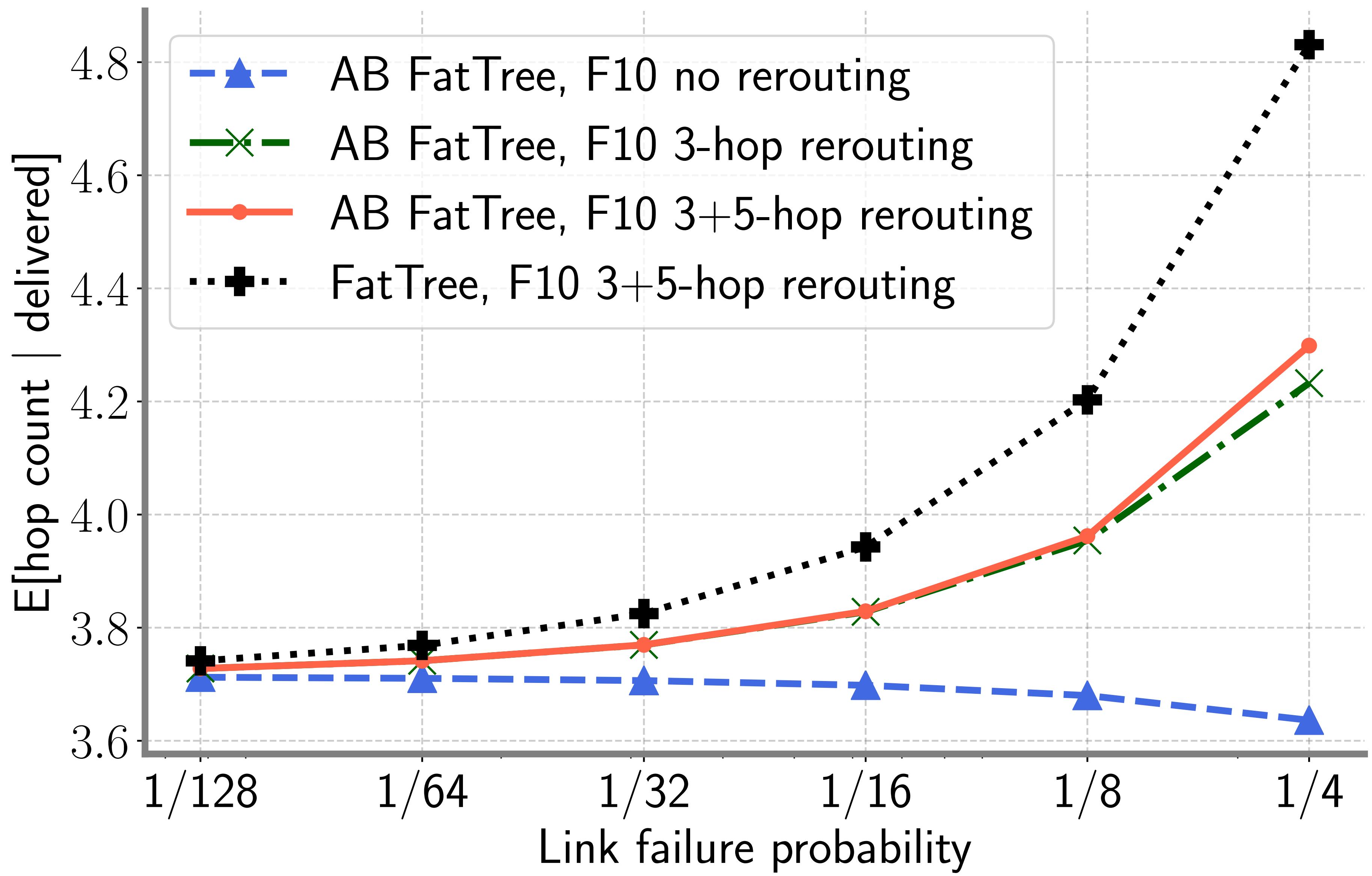
We evaluated packet loss when link failures increase



Dramatic improvement when using rerouting

Case Study: expected hop count

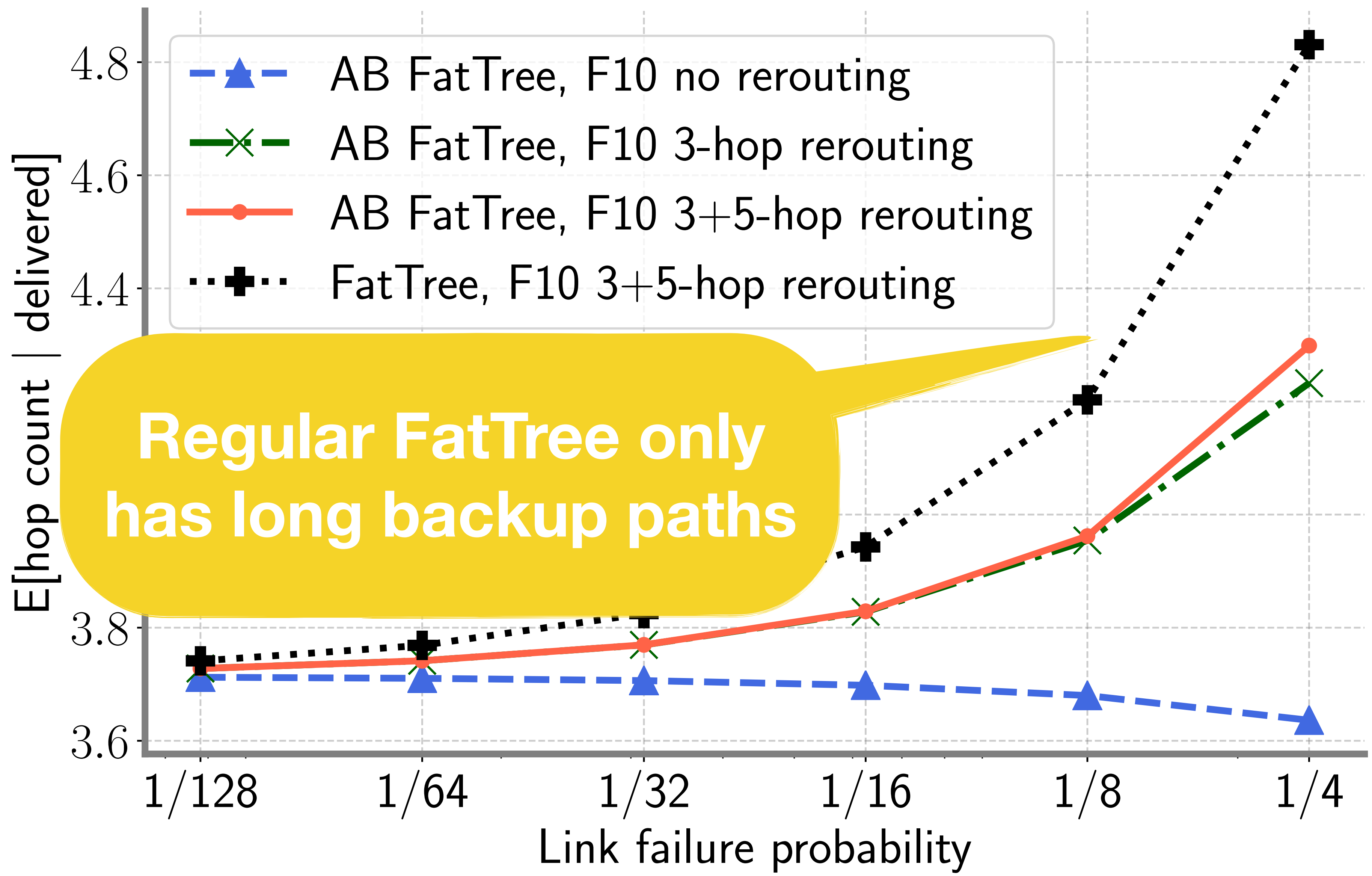
The price of resilience: increased paths lengths



ABFatTree outperforms regular FatTree

Case Study: expected hop count

The price of resilience: increased paths lengths



ABFatTree outperforms regular FatTree

Wrapping Up

Conclusion

ProbNetKAT is the first **probabilistic** network verification tool

Conclusion

ProbNetKAT is the first **probabilistic** network verification tool

Can verify reachability properties
even if network behavior is not deterministic

Conclusion

ProbNetKAT is the first **probabilistic** network verification tool

Can verify reachability properties
even if network behavior is not deterministic

Can reason about resilience
e.g., k-resilience, probability of delivery

Conclusion

ProbNetKAT is the first **probabilistic** network verification tool

Can verify reachability properties
even if network behavior is not deterministic

Can reason about resilience
e.g., k-resilience, probability of delivery

Can reason about quantitative properties
e.g., expected path length under failure model

Future Work

Future Work

Scalable implementation

Current prototype does not scale beyond 100 switches

Future Work

Scalable implementation
Current prototype does not scale beyond 100 switches

Probabilistic Inference
Given observation of packet loss, what link failure has most likely occurred?

Future Work

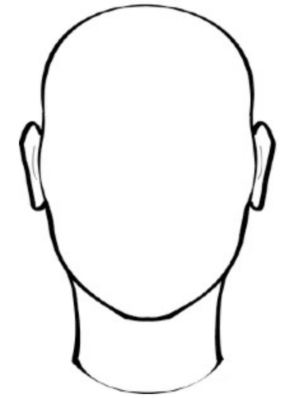
Scalable implementation
Current prototype does not scale beyond 100 switches

Probabilistic Inference
Given observation of packet loss, what link failure has most likely occurred?

More expressive language
ProbNetKAT has no notion of queuing or time

Nate
Foster

Justin
Hsu



David
Kahn

Dexter
Kozen

Praveen
Kumar

Alexandra
Silva

Steffen
Smolka

PROBABILISTIC
NETKAT