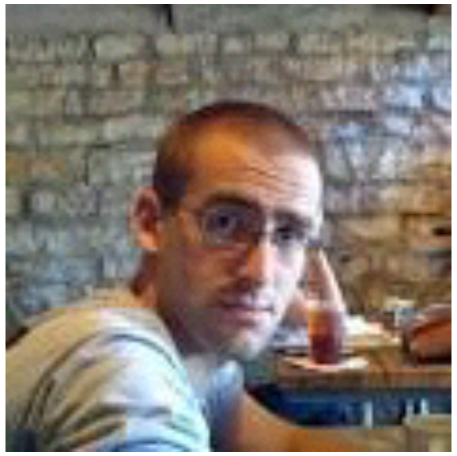
Probabilistic NetKAT

Nate Foster, Dexter Kozen (Cornell U), Konstantinos Mamouras (Penn), Mark Reitblatt (Facebook), **Alexandra Silva (UCL)**




UCL ENGINEERING
Change the world

UCL

Context

Formal specification and verification of networks have recently become a reality

- Frenetic [Foster & al., ICFP 11]
- Pyretic [Monsanto & al., NSDI 13]
- Maple [Voellmy & al., SIGCOMM 13]
- FlowLog [Nelson & al., NSDI 14]
- Header Space Analysis [Kazemian & al., NSDI 12]
- VeriFlow [Khurshid & al., NSDI 13]
- NetKAT [Anderson & al., POPL 14]
- and many others . . .

Context

Formal specification and verification of networks have recently become a reality

Trend in PL&Verification after Software-Defined Networks

- Design *high-level languages* that model essential network features
- Develop *semantics* that enables reasoning precisely about behavior
- Build *tools* to synthesize low-level implementations automatically

✤ NetKAT [Anderson & al., POPL 14]
✤ and many others . . .

Probabilistic NetKAT in a nutshell

[Foster & al., POPL15][Smolka & al., ICFP15][Anderson & al., POPL14]

- ✳ A probabilistic extension of NetKAT, a programming language/logic for specification/verification/programming of packet switching networks
- ✳ Programs denote functions that give **probability distributions** on sets of packet histories
- ✳ Enables reasoning about **probabilistic routing protocols** or behavior of deterministic protocols on **random inputs**
- ✳ Can handle scenarios involving **congestion**, **failure**, and **randomized routing**

ProbNetKAT language

pol ::= drop
| skip
| field = val
| pol1 & pol2
| pol1 ; pol2
| !pol
| pol $+_r$ pol
| pol*
| field := val
| dup

ProbNetKAT language

```
pol ::= drop
      | skip
      | field = val
      | pol1 & pol2
      | pol1 ; pol2
      | !pol
      | pol +r pol
      | pol*
      | field := val
      | dup
```

Boolean Algebra

ProbNetKAT language

pol ::= drop
| skip
| field = val
| pol1 & pol2
| pol1 ; pol2
| !pol
| pol $+_r$ pol
| pol*
| field := val
| dup

Boolean Algebra

+

Kleene Algebra

ProbNetKAT language

```
pol ::= drop
      | skip
      | field = val
      | pol1 & pol2
      | pol1 ; pol2
      | !pol
      | pol +r pol
      | pol*
      | field := val
      | dup
```

Boolean Algebra

+

Kleene Algebra

+

Packet Primitives

ProbNetKAT language

pol ::= drop
| skip
| field = val
| pol1 & pol2
| pol1 ; pol2
| !pol
| pol $+_r$ pol
| pol*
| field := val
| dup

Boolean Algebra

+

Kleene Algebra

+

Packet Primitives

+

Probabilistic choice

ProbNetKAT language

pol ::= drop
| skip
| field = val
| pol1 & pol2
| pol1 ; pol2
| !pol
| pol $+_r$ pol
| pol*
| field := val
| dup

Boolean Algebra

+

Kleene Algebra

KAT (Kozen'96)

+

Packet Primitives

+

Probabilistic choice

ProbNetKAT language

pol ::= drop
| skip
| field = val
| pol1 & pol2
| pol1 ; pol2
| !pol
| pol $+_r$ pol
| pol*
| field := val
| dup

Boolean Algebra

+

Kleene Algebra

+

Packet Primitives

+

Probabilistic choice

ProbNetKAT language

pol ::= drop
| skip
| field = val
| pol1 & pol2
| pol1 ; pol2
| !pol
| pol $+_r$ pol
| pol*
| field := val
| dup

Boolean Algebra

+

Kleene Algebra

+

Packet Primitives

NetKAT
(Anderson et al'14)

+

Probabilistic choice

ProbNetKAT language

pol ::= drop
| skip
| field = v
| pol1 & p
| pol1 ; p
| !pol
| pol $+_r$ p
| pol*
| field := val
| dup

Boolean Algebra

+

NetKAT
derson et al'14)

+

Probabilistic choice

KAT = simple imperative language

If b **then** p **else** q = b;p + !b;q

While b **do** p = (bp)*!b

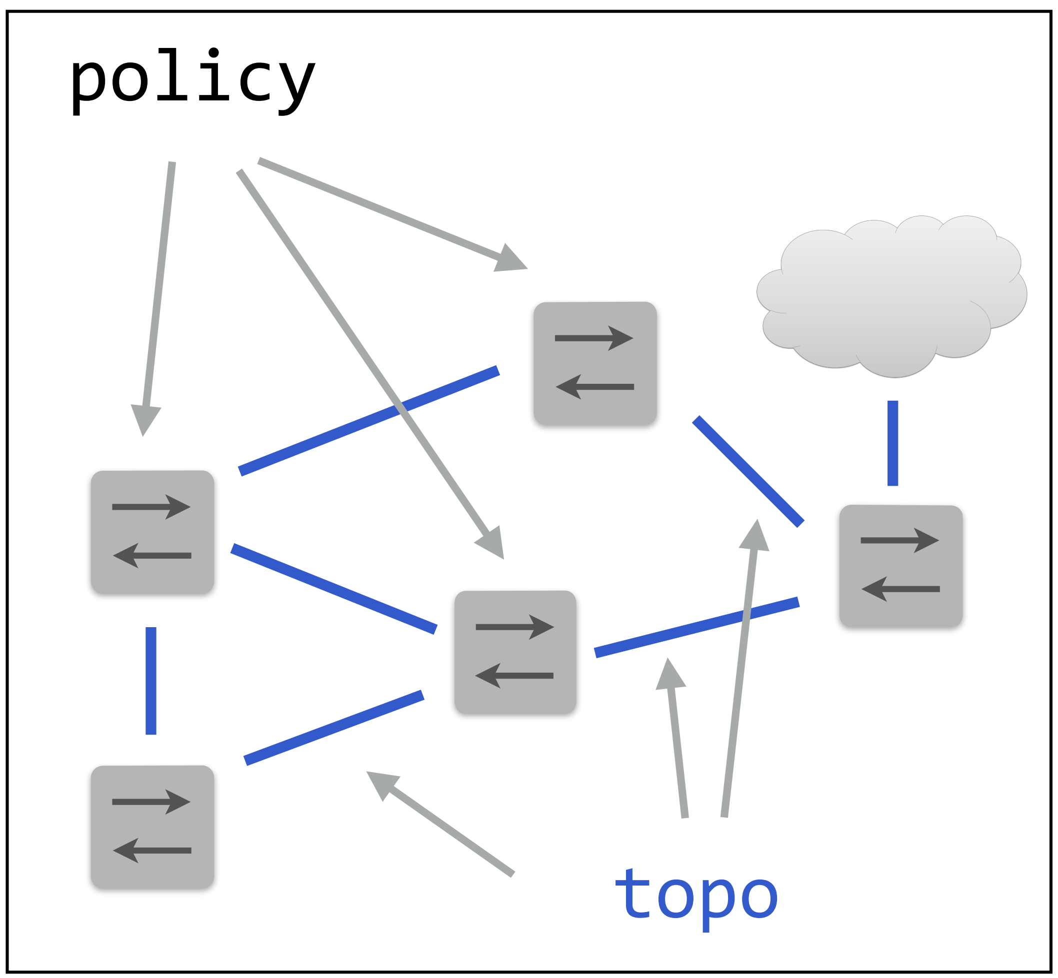
Networks in NetKAT

sw=6;pt=8;dst := 10.0.1.5;pt:=5

For all packets located at port 8 of switch 6, set the destination address to 10.0.1.5 and forward it out on port 5.

Networks in NetKAT

The behavior of an entire network can be encoded in NetKAT by interleaving steps of processions by switches and topology



→

policy

+

(policy; topo); policy

+

(policy; topo; policy; topo); policy

⋮

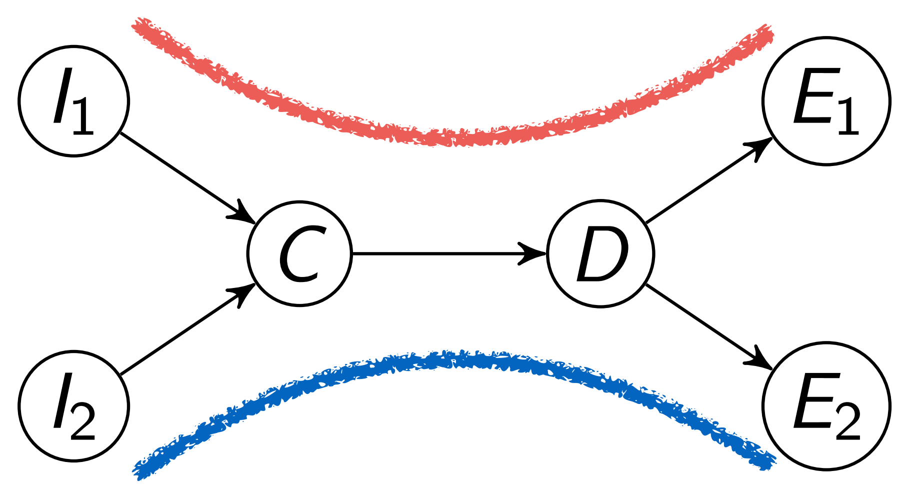
(policy; topo)*; policy

Semantics

$$[\![e]\!]: H \to 2^H$$

- ✴Packet-processing **function**
- ✴Applicability limited to simple connectivity or routing behavior

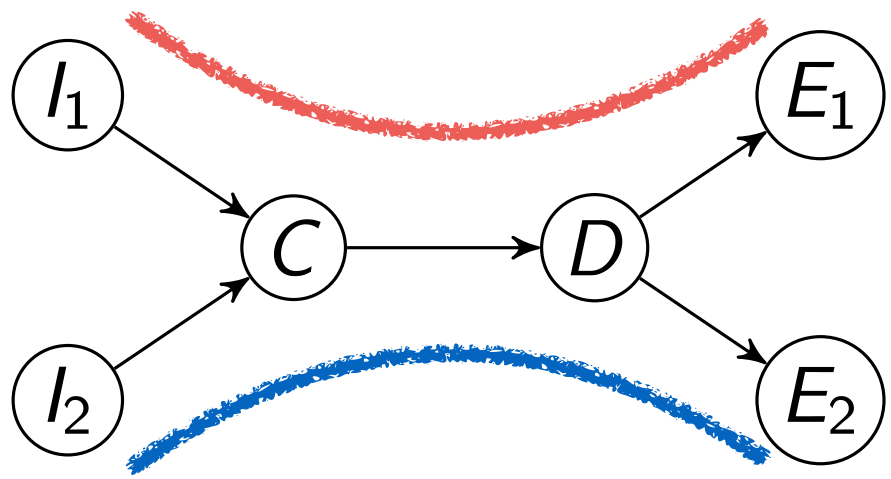
Example



Configure the switches to forward traffic on the two left-to-right paths from I1 to E1 and I2 to E2 .

p = (sw = I1 ; dup ; sw := C ; dup ; sw := D ; dup ; sw := E1) &
(sw = I2 ; dup ; sw := C ; dup ; sw := D ; dup ; sw := E2)

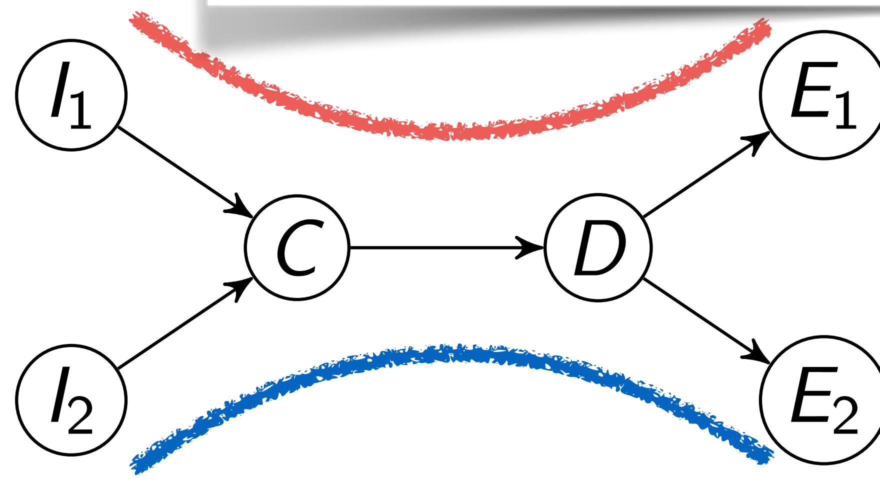
Example



Calculate not just ***where*** traffic is routed but also ***how much*** traffic is sent across each link.

Example

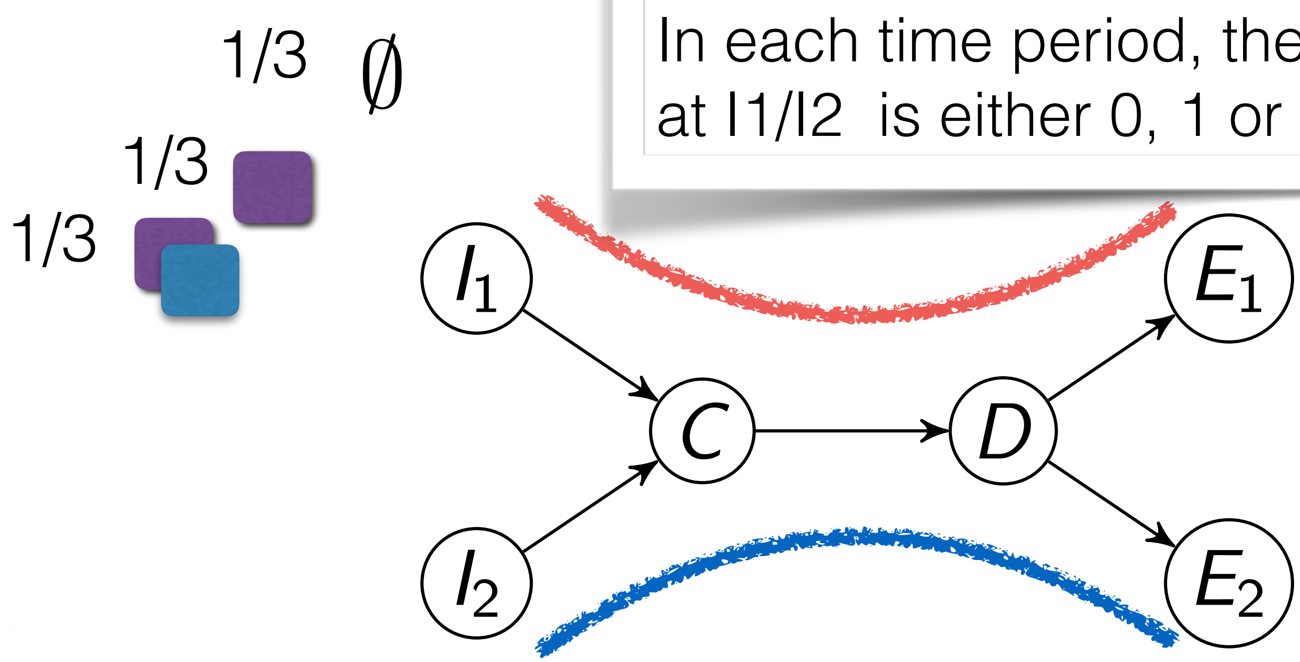
In each time period, the number of packets originating at I1/I2 is either 0, 1 or 2, with equal probability.



Calculate not just ***where*** traffic is routed but also ***how much*** traffic is sent across each link.

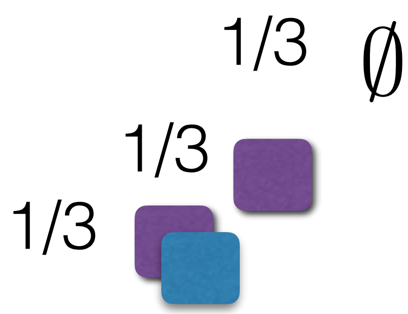
Example

In each time period, the number of packets originating at I1/I2 is either 0, 1 or 2, with equal probability.

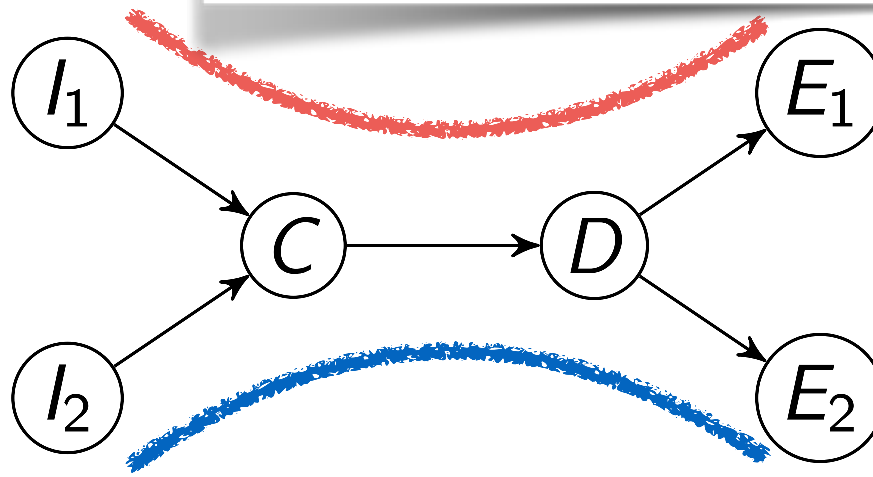


Calculate not just ***where*** traffic is routed but also ***how much*** traffic is sent across each link.

Example



In each time period, the number of packets originating at I1/I2 is either 0, 1 or 2, with equal probability.

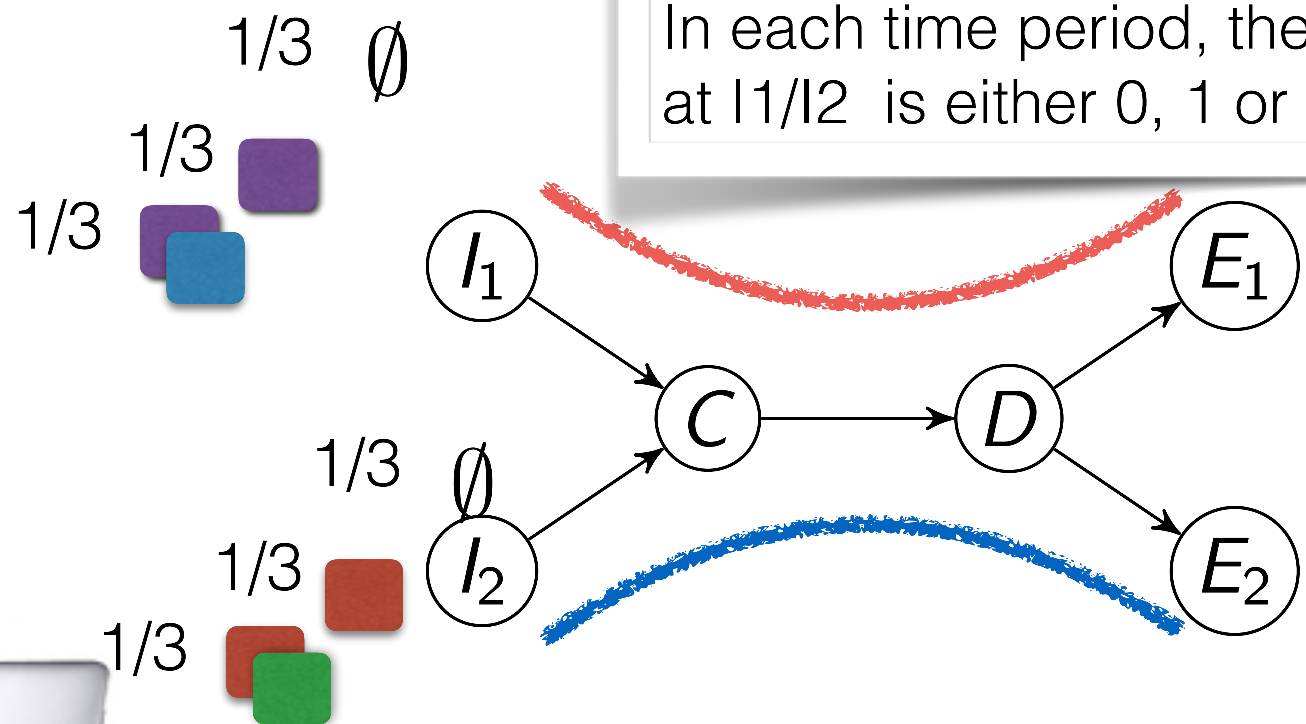


Calculate not just ***where*** traffic is routed but also ***how much*** traffic is sent across each link.

d1 = drop $+_{1/3}$ @I1 $+_{1/3}$ (@I1 & @I1)

Example

In each time period, the number of packets originating at I1/I2 is either 0, 1 or 2, with equal probability.

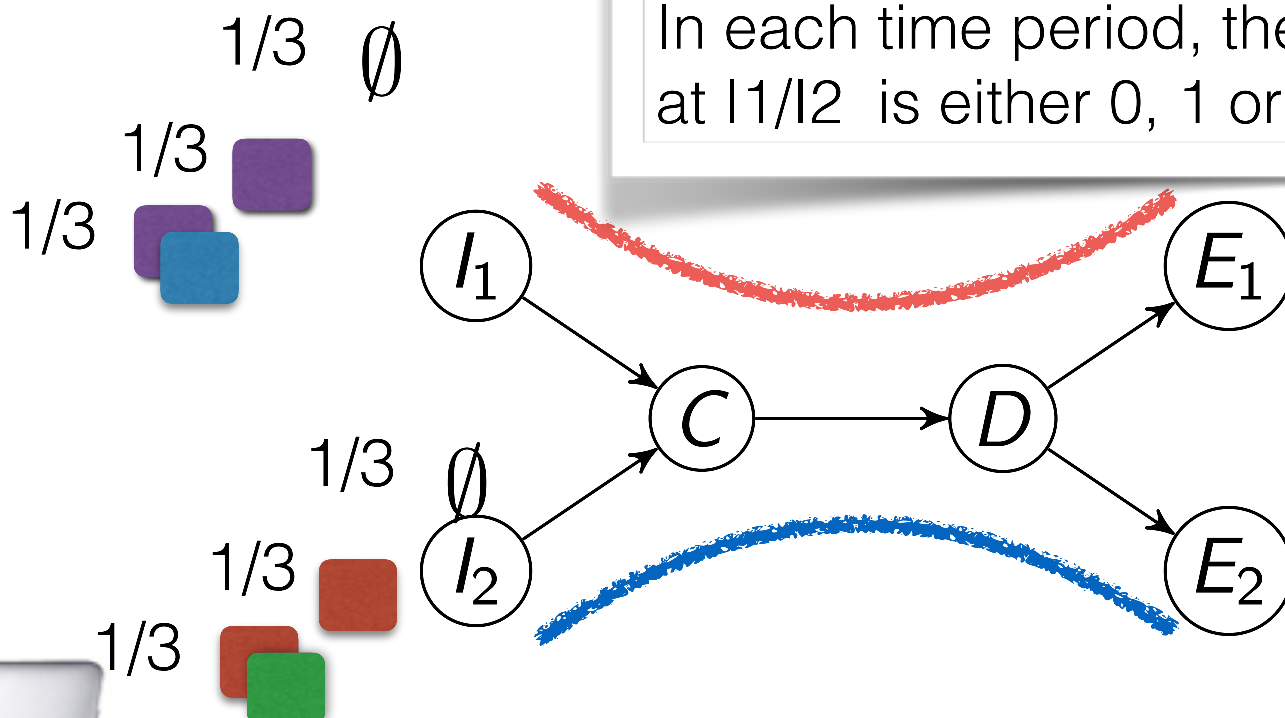


Calculate not just ***where*** traffic is routed but also ***how much*** traffic is sent across each link.

d1 = drop +1/3  @I1 +1/3 ( @I1 &  @I1)

Example

In each time period, the number of packets originating at I1/I2 is either 0, 1 or 2, with equal probability.



Calculate not just ***where*** traffic is routed but also ***how much*** traffic is sent across each link.

d1 = drop +1/3 @I1 +1/3 (@I1& @I1)

d2 = drop +1/3 @I2 +1/3 (@I2& @I2)

Probabilistic semantics

d1 = drop $+_{1/3}$ @l1 $+_{1/3}$ (@l1 & @l1)

d2 = drop $+_{1/3}$ @l2 $+_{1/3}$ (@l2 & @l2)

Distributions on set of histories

Full input distribution to the network : **d1 & d2**

Probabilistic semantics

d1 = drop $+_{1/3}$ @l1 $+_{1/3}$ (@l1 & @l1)

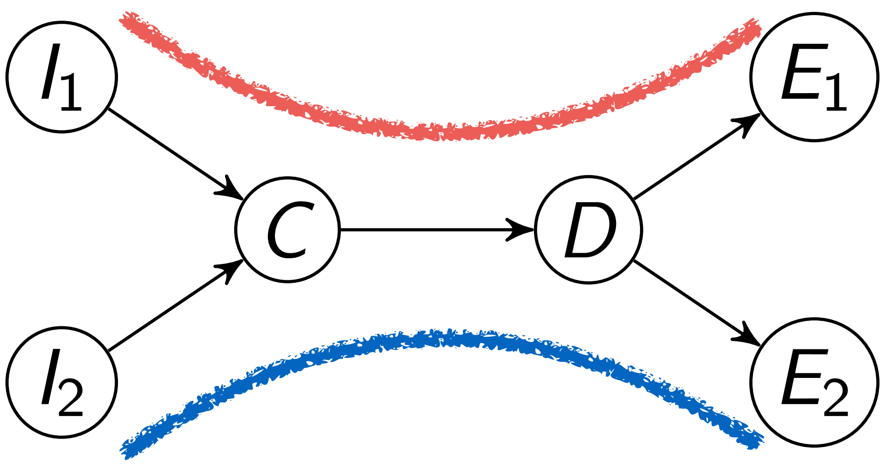
d2 = drop $+_{1/3}$ @l2 $+_{1/3}$ (@l2 & @l2)

Distributions on set of histories

Full input distribution to the network : **d1 & d2**

Compositionality

Probabilistic semantics



d1 = drop $+_{1/3}$ @I1 $+_{1/3}$ (@I1 & @I1)

d2 = drop $+_{1/3}$ @I2 $+_{1/3}$ (@I2 & @I2)

p = (sw = I1 ; dup ; sw := C ; dup ; sw := D ; dup ; sw := E1) &
(sw = I2 ; dup ; sw := C ; dup ; sw := D ; dup ; sw := E2)

Amount of congestion on links in the network?

$[\![d; p]\!] =$

1/9 . { } + ***no packet***
1/9 . {E1;1:C2;1:C1;1:I1;1} + ***one packet from I1-E1***
1/9 . {E1;1:C2;1:C1;1:I1;1; E1;2:C2;2:C1;2:I1;2} + ***two packets from I1-E1***
….

Probabilities are needed

✳ expected congestion: the network operator wishes to calculate the expected congestion on each link, given a model of incoming traffic

✳ reliability: the network operator wishes to calculate the probability of successful packet delivery given probability of failure of some network components

✳ randomized routing: the network operator wishes to use randomized routing schemes such as equal-cost multi-path routing (ECMP) or Valiant load balancing (VLB) to balance load across multiple paths

Yet another (Net)KAT extension?

The obvious extension does not work…

$$[\![e]\!] \colon H \to 2^H$$

$$[\![e]\!] \colon H \to \mathcal{D}(2^H)$$

$$[\![e]\!](h) = \delta([\![e]\!](h)) \Big\}$$ for the deterministic fragment

$$[\![p +_r q]\!](h) = r[\![p]\!](h) + (1-r)[\![p]\!](h)$$

Yet another (Net)KAT extension?

The obvious extension does not work…

$$[\![e]\!] : H \to 2^H$$

$$[\![e]\!] : H \to \mathcal{D}(2^H)$$

$$[\![e]\!](h) = \delta([\![e]\!](h)) \Big\}$$ for the deterministic fragment

$$[\![p +_r q]\!](h) = r[\![p]\!](h) + (1-r)[\![p]\!](h)$$

The meaning of a program on an input set of packet histories is **not** uniquely determined by its action on individual histories

Yet another (Net)KAT extension?

The obvious extension does not work…

$$[\![e]\!] : H \to 2^H$$

$$[\![e]\!] : H \to \mathcal{D}(2^H)$$

$$[\![e]\!](h) = \delta([\![e]\!](h)) \Big\}$$ for the deterministic fragment

$$[\![p +_r q]\!](h) = r[\![p]\!](h) + (1-r)[\![p]\!](h)$$

The meaning of a progra… istories is **not** uniquely determine… istories

Not compositional!!

Example bad semantics

$$[\![\pi_0! +_{.5} \pi_1!]\!](\pi_1) = [\![(\pi_0!\&\pi_1!) +_{.5} drop]\!](\pi_1) = 0.5$$

Problem 1 Different from desired meaning

$$[\![\pi_0! +_{.5} \pi_1!]\!] = 0.5\delta_{\{\pi_0\}} + 0.5\delta_{\{\pi_1\}}$$

$$[\![(\pi_0!\&\pi_1!) +_{.5} drop]\!] = 0.5\delta_{\{\pi_0,\pi_1\}} + 0.5\delta_{\emptyset}$$

Example bad semantics

$$[\![\pi_0! +_{.5} \pi_1!]\!](\pi_1) = [\![(\pi_0!\&\pi_1!) +_{.5} drop]\!](\pi_1) = 0.5$$

Problem 1 Different from desired meaning

$$[\![\pi_0! +_{.5} \pi_1!]\!] = 0.5\delta_{\{\pi_0\}} + 0.5\delta_{\{\pi_1\}}$$

$$[\![(\pi_0!\&\pi_1!) +_{.5} drop]\!] = 0.5\delta_{\{\pi_0,\pi_1\}} + 0.5\delta_{\emptyset}$$

Problem 2

$$[\![(\pi_0! +_{.5} \pi_1!); \pi_0!]\!] = \delta_{\{\pi_0!\}}$$

$$[\![((\pi_0!\&\pi_1!) +_{.5} drop); \pi_0!]\!] = 0.5\delta_{\{\pi_0!\}} + 0.5\delta_{\emptyset}$$

Example bad semantics

$$[\![\pi_0! +_{.5} \pi_1!]\!](\pi_1) = [\![(\pi_0!\&\pi_1!) +_{.5} drop]\!](\pi_1) = 0.5$$

Problem 1 Different from desired meaning

$$[\![\pi_0! +_{.5} \pi_1!]\!] = 0.5\delta_{\{\pi_0\}} + 0.5\delta_{\{\pi_1\}}$$

$$[\![(\pi_0!\&\pi_1!) +_{.5} drop]\!] = 0.5\delta_{\{\pi_0,\pi_1\}} + 0.5\delta_{\emptyset}$$

Problem 2

Not compositional!!

$$[\![(\pi_0! +_{.5} \pi_1!);\pi_0!]\!] = \delta_{\{\pi_0!\}}$$

$$[\![((\pi_0!\&\pi_1!) +_{.5} drop);\pi_0!]\!] = 0.5\delta_{\{\pi_0!\}} + 0.5\delta_{\emptyset}$$

It gets worse….

Discrete measures are not enough

$$p = \pi_0! +_{.5} \pi_1!$$

$$p; (dup; p)^*$$

It gets worse….

Discrete measures are not enough

$$p = \pi_0! +_{.5} \pi_1!$$

$$p; (dup; p)^*$$

Sets the input packet to either 0 or 1 with equal probability,
then repeat:
(i) output the current packet,
(ii) duplicate the current packet, and
(iii) set the new current packet to 0 or 1 with equal probability.

It gets worse….

Discrete measures are not enough

$$p = \pi_0! +_{.5} \pi_1!$$

$$p; (dup; p)^*$$

$[\![p; (dup; p)^*]\!](\pi_0)$ is a continuous measure

Our solution

Markov Kernels

$$[\![e]\!]\colon 2^H \times \mathcal{B} \to \mathbb{R}$$

measurable on the first argument
probability measure on the second argument

$$\mathcal{B} \subseteq 2^{2^H}$$

smallest sigma-algebra containing

$$\mathcal{B}_\tau = \{a \in 2^H \mid \tau \in a\}$$

Our solution

Markov Kernels

$$[\![e]\!] \colon 2^H \times \mathcal{B} \to \mathbb{R}$$

$$\mathcal{B} \subseteq 2^{2^H}$$

smallest sigma-algebra containing

$$\mathcal{B}_\tau = \{a \in 2^H \mid \tau \in a\}$$

$$[\![x \leftarrow n]\!](a) = \delta_{\{\pi[n/x]:\sigma \mid \pi:\sigma \in a\}}$$

$$[\![x = n]\!](a) = \delta_{\{\pi:\sigma \mid \pi:\sigma \in a,\ \pi(x)=n\}}$$

$$[\![\mathtt{dup}]\!](a) = \delta_{\{\pi:\pi:\sigma \mid \pi:\sigma \in a\}}$$

$$[\![\mathtt{skip}]\!](a) = \delta_a$$

$$[\![\mathtt{drop}]\!](a) = \delta_\varnothing$$

Our solution

Markov Kernels

$$[\![e]\!]\colon 2^H \times \mathcal{B} \to \mathbb{R}$$

$$\mathcal{B} \subseteq 2^{2^H}$$

smallest sigma-algebra containing

$$\mathcal{B}_\tau = \{a \in 2^H \mid \tau \in a\}$$

$$[\![x \leftarrow n]\!](a) = \delta_{\{\pi[n/x]:\sigma \mid \pi:\sigma \in a\}}$$

$$[\![x = n]\!](a) = \delta_{\{\pi:\sigma \mid \pi:\sigma \in a,\ \pi(x)=n\}}$$

$$[\![\mathtt{dup}]\!](a) = \delta_{\{\pi:\pi:\sigma \mid \pi:\sigma \in a\}}$$

$$[\![\mathtt{skip}]\!](a) = \delta_a$$

$$[\![\mathtt{drop}]\!](a) = \delta_\varnothing$$

$$[\![p \,\&\, q]\!](a) = [\![p]\!](a) \,\&\, [\![q]\!](a)$$

$$(\mu \,\&\, \nu)(A) \triangleq (\mu \times \nu)(\{(a,b) \mid a \cup b \in A\}).$$

Our solution

Markov Kernels

$$[\![e]\!]: 2^H \times \mathcal{B} \to \mathbb{R}$$

$$\mathcal{B} \subseteq 2^{2^H}$$

smallest sigma-algebra containing

$$\mathcal{B}_\tau = \{a \in 2^H \mid \tau \in a\}$$

$$[\![x \leftarrow n]\!](a) = \delta_{\{\pi[n/x]:\sigma \mid \pi:\sigma \in a\}}$$

$$[\![x = n]\!](a) = \delta_{\{\pi:\sigma \mid \pi:\sigma \in a,\ \pi(x)=n\}}$$

$$[\![\texttt{dup}]\!](a) = \delta_{\{\pi:\pi:\sigma \mid \pi:\sigma \in a\}}$$

$$[\![\texttt{skip}]\!](a) = \delta_a$$

$$[\![\texttt{drop}]\!](a) = \delta_\varnothing$$

$$[\![p \,\&\, q]\!](a) = [\![p]\!](a) \,\&\, [\![q]\!](a)$$

$$(\mu \,\&\, \nu)(A) \triangleq (\mu \times \nu)(\{(a,b) \mid a \cup b \in A\}).$$

$$[\![p +_r q]\!](a) = r[\![p]\!](a) + (1-r)[\![p]\!](a)$$

set of histories!

Properties

Conservative Extension

For deterministic programs, ProbNetKAT semantics and NetKAT semantics agree

⇓

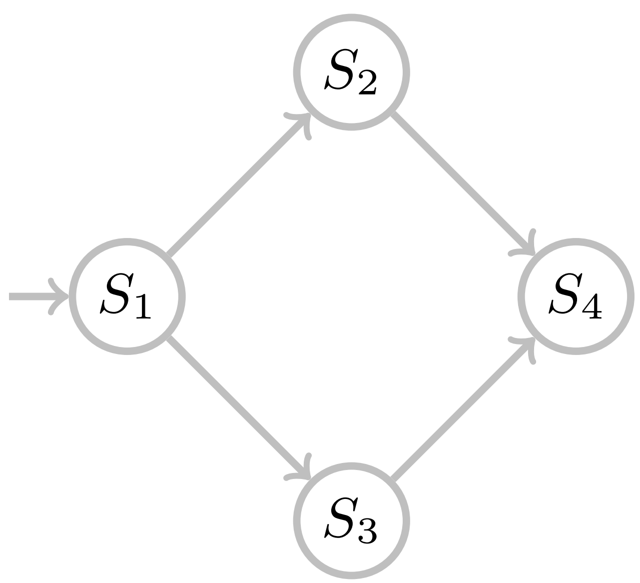
The NetKAT axioms are sound and complete for deterministic ProbNetKAT programs.

Some more properties

$[[p \& drop]] = [[drop \& p]] = [[p]]$
$[[p +_r p]] = [[p]]$
$[[p +_r q]] = [[q +_{1-r} p]]$
$[[(p \& q) \& s]] = [[p \& (q \& s)]]$
…

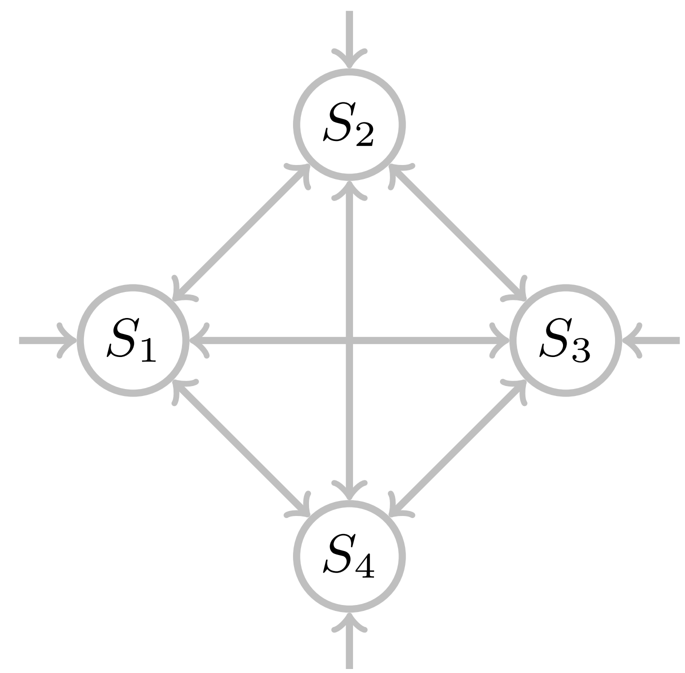
Applications

Fault Tolerance



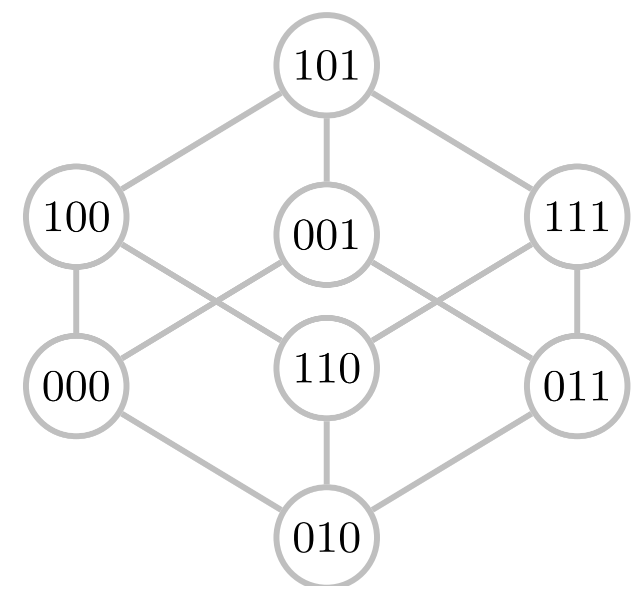
Probability packets are delivered at S4 if S1->S2 fails 10%

Load balancing



Maximum number of packets traversing a link

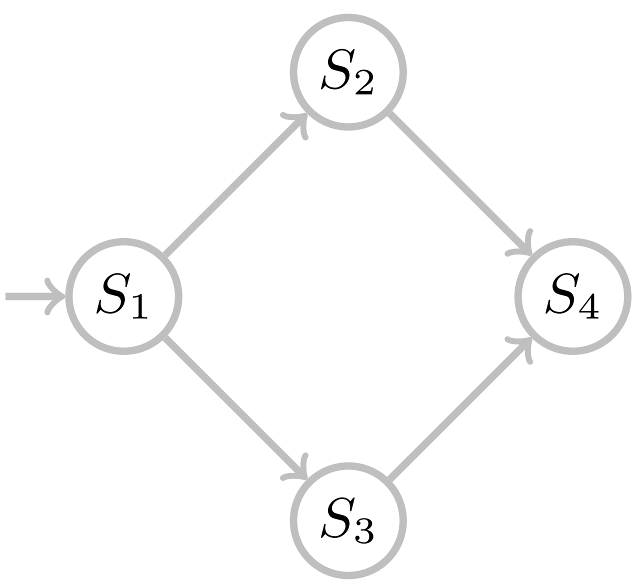
Gossiping protocols



Expected number of infected nodes after n rounds

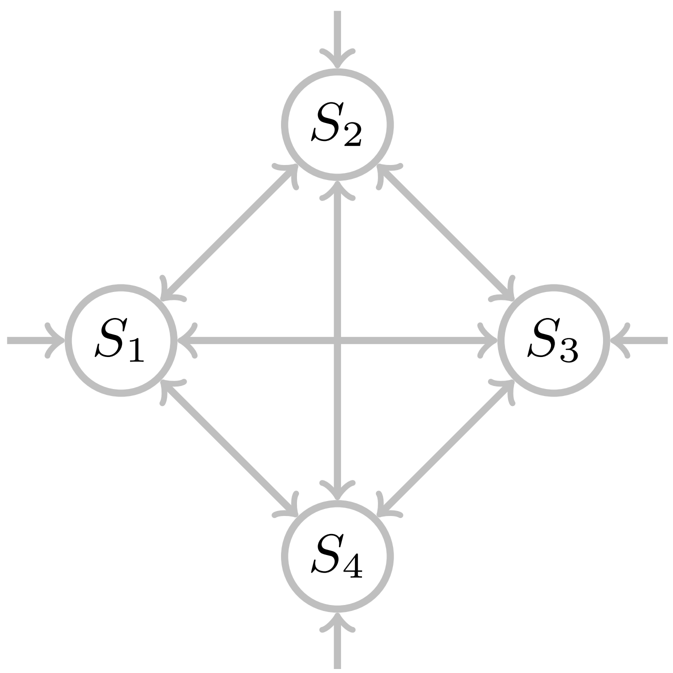
Applications

Fault Tolerance



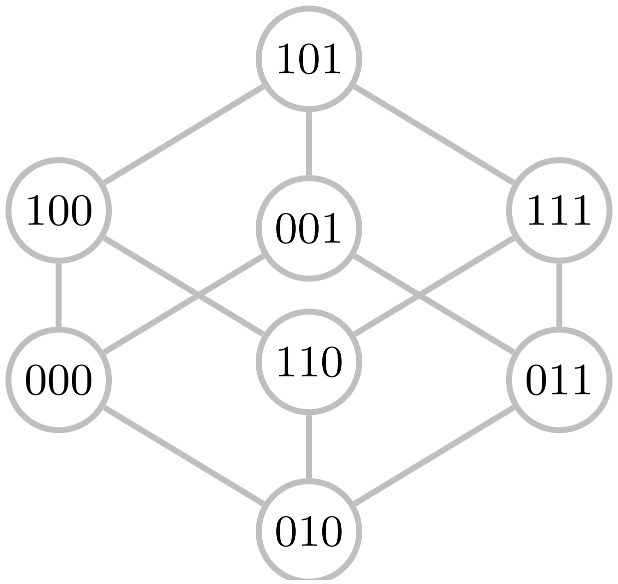
Probability packets are delivered at S4 if S1->S2 fails 10%

Load balancing



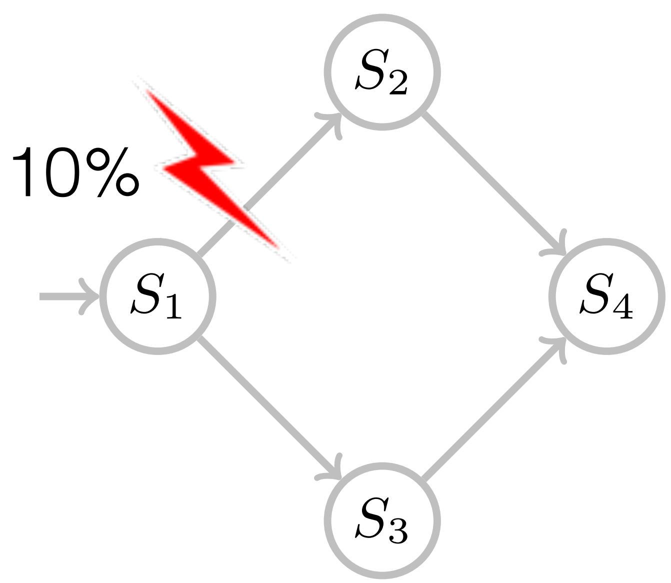
Maximum number of packets traversing a link

Gossiping protocols



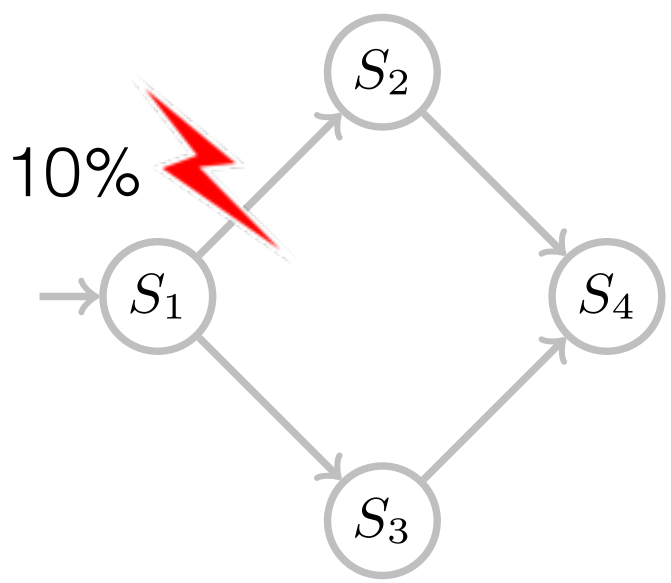
Expected number of infected nodes after n rounds

Fault Tolerance



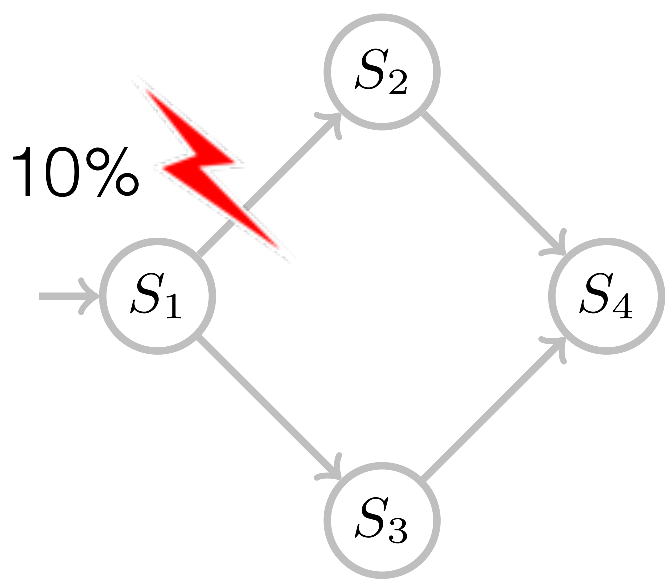
Fault Tolerance

What is the
probability that a packet that originates at S1
will be successfully delivered to S4 ?



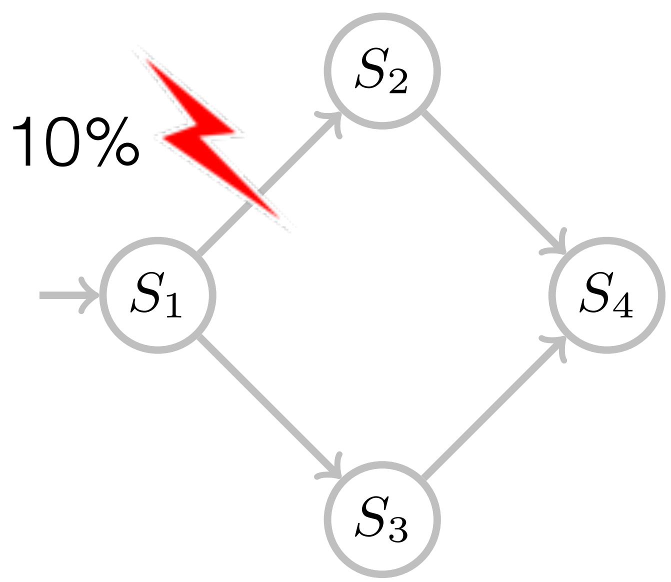
Fault Tolerance

What is the
probability that a packet that originates at S1
will be successfully delivered to S4 ?



t = sw=S1 ; pt =2 ;((sw:=2 ; pt:=1) $+_{.9}$ drop)
& sw=S1 ; pt =2 ;sw:=3 ; pt:=1)
& sw=S2 ; pt =4 ;sw:=4 ; pt:=2)
& sw=S3 ; pt =4 ;sw:=4 ; pt:=3)

Fault Tolerance



Switch behavior

p = (sw=1; pt:=2) & (sw =2 ; pt:=4)

all traffic via S2

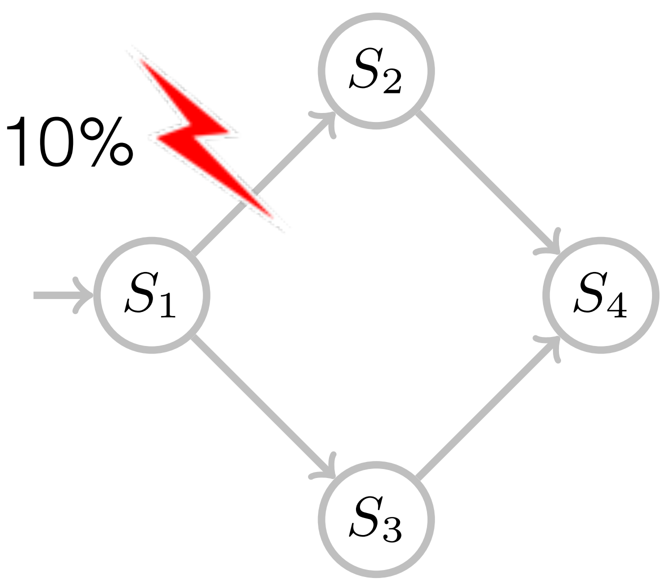
q = (sw=1; (pt:=2 $+_{.5}$ pt:=3))
& (sw =2 ; pt:=4)& (sw=3 ; pt:=4)

traffic split between S2 and S4

e = sw=4

egress predicate

Fault Tolerance



Switch behavior

p = (sw=1; pt:=2) & (sw =2 ; pt:=4)

all traffic via S2

What is the
probability that a packet that originates at S1
will be successfully delivered to S4 ?

traffic split between S2 and S4

e = sw=4

egress predicate

Fault Tolerance

Switch behavior

p = (sw=1; pt:=2) & (sw =2 ; pt:=4)

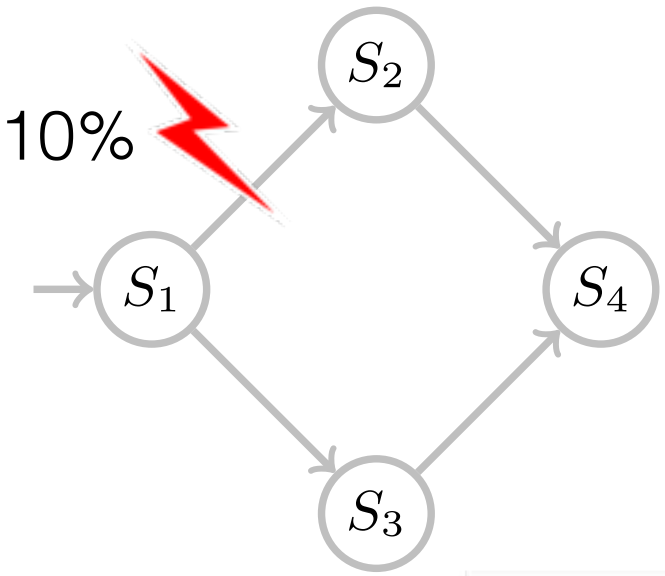
all traffic via S2

What is the
probability that a packet that originates at S1
will be successfully delivered to S4 ?

traffic split between S2 and S4

e = sw=4

egress predicate



(p;t)*;e → 90%

[[-]]

(q;t)*;e → 95%

Also observed in work on randomised routing [Zhang-Shen and McKeown, IWQoS 05]

Conclusions

- ✦ First language-based framework for specifying and verifying **probabilistic network behavior**.
- ✦ Formal semantics for ProbNetKAT based on Markov kernels (conservative over NetKAT).
- ✦ Notion of approximation — every ProbNetKAT program is arbitrarily closely approximated by loop-free programs.
- ✦ Several case studies — fault tolerance, load balancing, and a probabilistic gossip protocol.

Future work

Axiomatizations

Decision procedure

Simulation

Certified Compiler

Future work

Axiomatizations

Decision procedure

Certified Compiler

Simulation

Questions?