Probabilistic NetKAT

Nate Foster[1] Dexter Kozen[1] Mark Reitblatt[2]
Kostas Mamouras[3] Alexandra Silva[4]

[1]Cornell [2]Facebook [3]Penn [4]UCL

ESOP 2016
Eindhoven
4 April 2016

Motivation

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 . . .

Motivation

But these systems are all deterministic

- network elements modeled as deterministic packet-processing functions
- applicability limited to simple connectivity or routing behavior

We would like to model more complicated situations that often arise in practice

- 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

This Paper

Probabilistic NetKAT (ProbNetKAT)

- a probabilistic extension of NetKAT [Foster & al., POPL 15] [Anderson & al., POPL 14] [Smolka & al., ICFP 15], 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

Contributions

- The design of ProbNetKAT, the first language-based framework for specifying and verifying probabilistic network behavior
- Formal operational and denotational semantics based on Markov kernels
- ProbNetKAT extends NetKAT conservatively
- An appropriate notion of approximation—every program is arbitrarily closely approximated by a loop-free one
- Three illustrative case studies drawn from real-world networks

Design Challenges

A number of important questions do not have obvious answers:

- discrete or continuous distributions?
- lossless or lossy?
- independence of random choices in processing sets of packet histories?
- most importantly: semantics of iteration?

NetKAT

[Anderson & al., POPL 14] [Foster & al., POPL 15] [Smolka & al., ICFP 15]

NetKAT is . . .

- a programming/specification language/logic for specifying, programming, and reasoning about packet switching networks
- based on Kleene algebra with tests (KAT)
- decidable and deductively complete
- implemented and deployed as part of the Frenetic suite of network management tools (compiler and decision procedure)

NetKAT

[Anderson & al., POPL 14] [Foster & al., POPL 15] [Smolka & al., ICFP 15]

NetKAT

=

Kleene algebra with tests (KAT)

+

additional specialized constructs particular to network topology and packet switching

Kleene Algebra (KA)

Idempotent Semiring Axioms

$$
\begin{array}{ll}
p + (q + r) = (p + q) + r & p(qr) = (pq)r \\
p + q = q + p & 1p = p1 = p \\
p + 0 = p & p0 = 0p = 0 \\
p + p = p & \\
p(q + r) = pq + pr & p \leq q \overset{\triangle}{\iff} p + q = q \\
(p + q)r = pr + qr &
\end{array}
$$

Axioms for *

$$
\begin{array}{ll}
1 + pp^* \leq p^* & q + px \leq x \;\Rightarrow\; p^*q \leq x \\
1 + p^*p \leq p^* & q + xp \leq x \;\Rightarrow\; qp^* \leq x
\end{array}
$$

Kleene Algebra with Tests (KAT)

$(K, B, +, \cdot, {}^*, \bar{\ }, 0, 1), \quad B \subseteq K$

- $(K, +, \cdot, {}^*, 0, 1)$ is a Kleene algebra
- $(B, +, \cdot, \bar{\ }, 0, 1)$ is a Boolean algebra
- $(B, +, \cdot, 0, 1)$ is a subalgebra of $(K, +, \cdot, 0, 1)$

Can encode basic imperative language constructs & partial correctness reasoning

- `if` b `then` p `else` $q = bp + \bar{b}q$
- `while` b `do` $p = (bp)^*\bar{b}$
- $\{b\}\ p\ \{c\} = (bp \leq pc) = (bp\bar{c} = 0) = (bp = bpc)$

NetKAT

Can also encode basic networking constructs: packet filtering & forwarding, network topology

- a packet π is an assignment of constant values n to fields x
- a packet history is a nonempty sequence of packets $\pi_1 :: \pi_2 :: \cdots :: \pi_k$
- the head packet is π_1

NetKAT primitives

- assignments $x \leftarrow n$
  assign constant value n to field x in the head packet
- tests $x = n$
  if value of field x in the head packet is n, then pass, else drop
- dup
  duplicate the head packet

NetKAT

Example

$$sw = 6 \;;\; pt = 88 \;;\; dest \leftarrow 10.0.0.1 \;;\; pt \leftarrow 50$$

"For all packets incoming on port 88 of switch 6, set the destination IP address to 10.0.0.1 and send the packet out on port 50."

NetKAT Axioms

$$x \leftarrow n \,;\, y \leftarrow m \;\equiv\; y \leftarrow m \,;\, x \leftarrow n \quad (x \neq y)$$

assignments to distinct fields may be done in either order

$$x \leftarrow n \,;\, y = m \;\equiv\; y = m \,;\, x \leftarrow n \quad (x \neq y)$$

an assignment to a field does not affect a different field

$$x = n \,;\, \mathsf{dup} \;\equiv\; \mathsf{dup} \,;\, x = n$$

field values are preserved in a duplicated packet

$$x \leftarrow n \;\equiv\; x \leftarrow n \,;\, x = n$$

an assignment causes the field to have that value

$$x = n \,;\, x \leftarrow n \;\equiv\; x = n$$

an assignment of a value that the field already has is redundant

---

$$x \leftarrow n \,;\, x \leftarrow m \;\equiv\; x \leftarrow m$$

a second assignment to the same field overrides the first

$$x = n \,;\, x = m \;\equiv\; 0 \quad (n \neq m) \qquad\qquad \left(\textstyle\sum_n x = n\right) \;\equiv\; 1$$

every field has exactly one value

NetKAT Semantics

Standard model: packet filtering and forwarding functions

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

where $H = \{\text{packet histories}\}$

NetKAT Semantics

Standard model: packet filtering and forwarding functions

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

where $H = \{\text{packet histories}\}$

$$[\![x \leftarrow n]\!](\pi :: \sigma) \triangleq \{\pi[n/x] :: \sigma\} \qquad [\![1]\!](\sigma) \triangleq [\![\text{skip}]\!](\sigma) = \{\sigma\}$$

$$[\![\phi]\!](\pi :: \sigma) \triangleq \begin{cases} \{\pi :: \sigma\} & \text{if } \pi \vDash \phi \\ \emptyset & \text{if } \pi \nvDash \phi \end{cases} \qquad [\![0]\!](\sigma) \triangleq [\![\text{drop}]\!](\sigma) = \emptyset$$

$$[\![\text{dup}]\!](\pi :: \sigma) \triangleq \{\pi :: \pi :: \sigma\} \qquad [\![p \,;\, q]\!](\sigma) \triangleq \bigcup_{\tau \in [\![p]\!](\sigma)} [\![q]\!](\tau)$$

$$[\![p + q]\!](\sigma) \triangleq [\![p]\!](\sigma) \cup [\![q]\!](\sigma) \qquad [\![p^*]\!](\sigma) \triangleq \bigcup_n [\![p^n]\!](\sigma)$$

NetKAT Examples

[Anderson & al 2014]

Reachability

- Can host A communicate with host B? Can every host communicate with every other host?

Security

- Does all untrusted traffic pass through the intrusion detection system located at C?

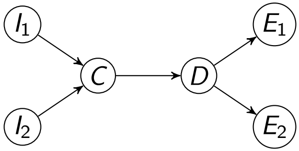
Loop detection

- Is it possible for a packet to be forwarded around a cycle in the network?

Probabilistic NetKAT (ProbNetKAT)

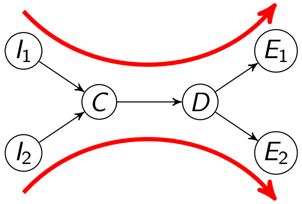
- Extend NetKAT with probabilistic primitives
- Enriched semantics based on Markov kernels
- Extends NetKAT conservatively
- Lose deductive completeness but can still reason semantically
- Applications:
  - expected connectivity given probability of link failures
  - expected load given ingress traffic
  - expected delivery time for probabilistic broadcast protocols (gossip)

Example



Suppose the network operator wants to configure the switches to forward traffic on the two left-to-right paths from I_1 to E_1 and I_2 to E_2.

Example



Suppose the network operator wants to configure the switches to forward traffic on the two left-to-right paths from I_1 to E_1 and I_2 to E_2.

We can specify this in ProbNetKAT as follows:

$$\begin{aligned} p \triangleq\, & (sw = I_1 \,;\, \mathrm{dup} \,;\, sw \leftarrow C \,;\, \mathrm{dup} \,;\, sw \leftarrow D \,;\, \mathrm{dup} \,;\, sw \leftarrow E_1)\ \& \\ & (sw = I_2 \,;\, \mathrm{dup} \,;\, sw \leftarrow C \,;\, \mathrm{dup} \,;\, sw \leftarrow D \,;\, \mathrm{dup} \,;\, sw \leftarrow E_2) \end{aligned}$$

Example (cont'd)

Now suppose now we wish to represent the following traffic model:

> *In each time period, the number of packets originating at I_1 is either 0, 1 or 2, with equal probability, and likewise for I_2.*

Write $\pi_{ij}!$ for a sequence of assignments that produces the packet π_j at switch I_i. We can encode the input distributions at I_1 and I_2 as follows.

$$d_1 \triangleq \mathtt{drop} \oplus \pi_{1,1}! \oplus (\pi_{1,1}! \,\&\, \pi_{1,2}!)$$
$$d_2 \triangleq \mathtt{drop} \oplus \pi_{2,3}! \oplus (\pi_{2,3}! \,\&\, \pi_{2,4}!)$$

d_1 and d_2 are functions giving distributions on sets of histories.

The full input distribution is $d \triangleq d_1 \,\&\, d_2$.

Example (cont'd)

To calculate a distribution that encodes congestion on links, push the input distribution d through the forwarding policy p using sequential composition: $d\ ;\ p$.

This produces a distribution on sets of histories. In this example, there are nine such sets, and the output distribution is uniform:

$$\begin{array}{l}
\emptyset \\
\{E_{1,1}{:}D_1{:}C_1{:}I_{1,1}\} \\
\{E_{2,3}{:}D_3{:}C_3{:}I_{2,3}\} \\
\{E_{1,1}{:}D_1{:}C_1{:}I_{1,1},\ E_{1,2}{:}D_2{:}C_2{:}I_{1,2}\} \\
\{E_{2,3}{:}D_3{:}C_3{:}I_{2,3},\ E_{2,4}{:}D_4{:}C_4{:}I_{2,4}\} \\
\{E_{1,1}{:}D_1{:}C_1{:}I_{1,1},\ E_{2,3}{:}D_3{:}C_3{:}I_{2,3}\} \\
\{E_{1,1}{:}D_1{:}C_1{:}I_{1,1},\ E_{1,2}{:}D_2{:}C_2{:}I_{1,2},\ E_{2,3}{:}D_3{:}C_3{:}I_{2,3}\} \\
\{E_{1,1}{:}D_1{:}C_1{:}I_{1,1},\ E_{2,3}{:}D_3{:}C_3{:}I_{2,3},\ E_{2,4}{:}D_4{:}C_4{:}I_{2,4}\} \\
\{E_{1,1}{:}D_1{:}C_1{:}I_{1,1},\ E_{1,2}{:}D_2{:}C_2{:}I_{1,2},\ E_{2,3}{:}D_3{:}C_3{:}I_{2,3},\ E_{2,4}{:}D_4{:}C_4{:}I_{2,4}\}
\end{array}$$

Example (cont'd)

To calculate the expected number of packets traversing the link ℓ from C_1 to C_2, filter on the set $b = \cdots{:}D_i{:}C_i{:}\cdots$ and ask for the expected size of the result. (In this example, all histories traverse ℓ, so b actually has no effect.)

Example (cont'd)

To calculate the expected number of packets traversing the link ℓ from C_1 to C_2, filter on the set $b = \cdots{:}D_i{:}C_i{:}\cdots$ and ask for the expected size of the result. (In this example, all histories traverse ℓ, so b actually has no effect.)

The expected number of packets traversing ℓ is given by integration:

$$\int_{a\in 2^H} |a| \cdot [\![d \mathbin{;} p \mathbin{;} b]\!](da) = 2$$

Semantic Considerations

When formulating the semantics of ProbNetKAT, several novel considerations arose:

- NetKAT and ProbNetKAT are not a state-based, but rather flow-based
- computation is lossless; "halting" is not a relevant notion (all programs "halt" and produce an output w.p. 1 (which may be the empty set)
- it is no longer the case that the meaning of a program on an input set of packet histories is uniquely determined by its action on individual histories; probabilistic decisions may be correlated
- parallel composition (&) is not idempotent, except for deterministic programs
- distributivity no longer holds in general, except for deterministic programs
- in the presence of both duplication (dup) and iteration (*), discrete distributions do not suffice

Markov Kernels

Let $(S, \mathcal{B}_S)$ and $(T, \mathcal{B}_T)$ be measurable spaces. A function $P : S \times \mathcal{B}_T \to \mathbb{R}$ is called a Markov kernel if

- for fixed $A \in \mathcal{B}_T$, the map $P(-, A) : S \to \mathbb{R}$ is a measurable function on $(S, \mathcal{B}_S)$
- for fixed $s \in S$, the map $P(s, -) : \mathcal{B}_T \to \mathbb{R}$ is a probability measure on $(T, \mathcal{B}_T)$

The measurable spaces and Markov kernels form a category, the Kleisli category of the Giry monad. Composition is given by Lebesgue integration: for $P : S \to T$ and $Q : T \to U$,

$$(P \,;\, Q)(s, A) = \int_{t \in T} P(s, dt) \cdot Q(t, A).$$

Associativity is essentially Fubini's theorem.

ProbNetKAT Syntax

- random choice: $p \oplus_r q$, where $r \in [0, 1]$
  Flip an r-biased coin (Pr(heads)=r, Pr(tails)=$1 - r$), do p on heads and q on tails
- parallel composition: $p \,\&\, q$
  Perform both p and q, making any probabilistic choices in p and q independently, and take the union of the resulting sets
- extended tests: Formally just an element $b \in 2^H$.
  Filter the input set by b

A Measurable Space

2^H = powerset of packet histories H forms a topological space generated by basic clopen sets

$$B_\tau = \{a \in 2^H \mid \tau \in a\}, \ \tau \in H$$

Homeomorphic to the Cantor space, the topological product of countably many copies of the discrete two-element space

$\mathcal{B} \subseteq 2^{2^H}$ = the Borel sets of this topology (smallest σ-algebra containing the sets B_τ)

The measurable space $(2^H, \mathcal{B})$ with outcomes 2^H and events $\mathcal{B}$.

ProbNetKAT programs p are interpreted as Markov kernels $[\![p]\!] : 2^H \to 2^H$.

Semantics of Atomic Operations

- $[\![x \leftarrow n]\!](a, -) \triangleq \delta_{\{\pi[n/x]:\sigma \mid \pi:\sigma \in a\}}$
- $[\![x = n]\!](a, -) \triangleq \delta_{\{\pi:\sigma \mid \pi:\sigma \in a,\ \pi(x)=n\}}$
- $[\![b]\!](a, -) \triangleq \delta_{a \cap b}$
- $[\![\mathtt{skip}]\!](a, -) \triangleq \delta_{a}$
- $[\![\mathtt{drop}]\!](a, -) \triangleq \delta_{\emptyset}$
- $[\![\mathsf{dup}]\!](a, -) \triangleq \delta_{\{\pi:\pi:\sigma \mid \pi:\sigma \in a\}}$

These are all deterministic kernels: δ_a = Dirac (point mass) measure on a

Parallel Composition &

Operational semantics

$$\begin{aligned}[\![p \mathbin{\&} q]\!](s,-) \triangleq\ & \texttt{let } a = \texttt{sample}([\![p]\!](s,-)) \texttt{ in} \\ & \texttt{let } b = \texttt{sample}([\![q]\!](s,-)) \texttt{ in} \\ & a \cup b\end{aligned}$$

Denotational semantics

$$[\![p \mathbin{\&} q]\!](s,-) \triangleq ([\![p]\!](s,-) \times [\![q]\!](s,-)) \mathbin{;} \cup^{-1}$$

The union operation $\bigcup : 2^H \times 2^H \to 2^H$ is continuous, thus measurable

$[\![p \mathbin{\&} q]\!](s,A) =$ probability that the union of two independent samples taken with respect to $[\![p]\!](s,-)$ and $[\![q]\!](s,-)$ lies in A

Properties of &

Lemma

- *& is associative and commutative*
- *& is linear in both arguments*
- $(\delta_a \mathbin{\&} \mu)(A) = \mu(\{b \mid a \cup b \in A\})$
- $\delta_a \mathbin{\&} \delta_b = \delta_{a \cup b}$
- $\delta_\emptyset$ *is a two-sided identity for &*
- $\mu \mathbin{\&} \mu = \mu$ *iff* $\mu = \delta_a$ *for some* $a \in 2^H$

Sequential Composition $p\ ;\ q$

Operational semantics

$$\begin{aligned}[\![p\ ;\ q]\!](a,-) \triangleq\ & \texttt{let } b = \texttt{sample}([\![p]\!](a,-)) \texttt{ in} \\ & \texttt{sample}([\![q]\!](b,-))\end{aligned}$$

Denotational semantics

$$[\![p\ ;\ q]\!](a,A) = \int_{b \in 2^H} [\![p]\!](a,db) \cdot [\![q]\!](b,A)$$

Composition in NetKAT and ProbNetKAT take place in the Kleisli category of the powerset and Giry monads, respectively

Random Choice $p \oplus_r q$

Operational semantics

$$
\begin{aligned}
[\![p \oplus_r q]\!](a, -) \triangleq\ & \texttt{let } b = \texttt{flip}(r) \texttt{ in} \\
& \texttt{if } b \texttt{ then sample}([\![p]\!](a, -)) \\
& \texttt{else sample}([\![q]\!](a, -))
\end{aligned}
$$

Denotational semantics

$$[\![p \oplus_r q]\!](a, A) = r[\![p]\!](a, A) + (1 - r)[\![q]\!](a, A)$$

Do p with probability r and q with probability $1 - r$

Iteration p^*

The usual definition of p^* as a sum of powers does not work!

Operational semantics

An infinite stochastic process:

$$
\begin{aligned}
[\![p^*]\!](c_0, -) = {} & \texttt{forall } n \geq 0 \\
& \texttt{let } c_{n+1} = \texttt{sample}([\![p]\!](c_n, -)) \texttt{ in} \\
& \bigcup_n c_n
\end{aligned}
$$

Denotational semantics

Quite technical—requires the Kolmogorov extension theorem

A Fixpoint Equation

Theorem $[\![p^*]\!] = [\![\texttt{skip} \mathbin{\&} pp^*]\!]$.

- We can encode conditionals and while loops in the standard way:

$$\texttt{if } b \texttt{ then } p \texttt{ else } q = bp \mathbin{\&} \bar{b}q \qquad \texttt{while } b \texttt{ do } p = (bp)^* \bar{b}$$

- the fixpoint theorem ensures that the while loop works as desired

$$\texttt{while } b \texttt{ do } p = \texttt{if } b \texttt{ then } (p \,;\, \texttt{while } b \texttt{ do } p) \texttt{ else skip}$$

- does not determine $[\![p^*]\!]$ uniquely; e.g., it can be shown that a probability measure μ is a solution of

$$[\![\texttt{skip}^*]\!](\pi, -) = [\![\texttt{skip} \mathbin{\&} \texttt{skip} \,;\, \texttt{skip}^*]\!](\pi, -)$$

  iff $\mu(B_\pi) = 1$.

Conservativity of the Extension

Lemma

All syntactically deterministic ProbNetKAT programs p (those without an occurrence of $\oplus_r$) are (semantically) deterministic. That is, for any $a \in 2^H$, the distribution $[\![p]\!](a, -)$ is a point mass.

Let $[\![\cdot]\!]_{\mathrm{N}}$ and $[\![\cdot]\!]_{\mathrm{P}}$ denote the semantic maps for NetKAT and ProbNetKAT respectively.

Theorem

For deterministic programs, ProbNetKAT semantics and NetKAT semantics agree in the following sense. For $a \in 2^H$, define $[\![p]\!]_{\mathrm{N}}(a) = \bigcup_{\tau \in a} [\![p]\!]_{\mathrm{N}}(\tau)$. Then for any $a, b \in 2^H$,

$$[\![p]\!]_{\mathrm{N}}(a) = b \iff [\![p]\!]_{\mathrm{P}}(a) = \delta_b.$$

Conservativity of the Extension

Corollary

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

Some Useful Properties

- $[\![p \mathbin{\&} \texttt{drop}]\!] = [\![p]\!] = [\![\texttt{drop} \mathbin{\&} p]\!]$
- $[\![p \oplus_r p]\!] = [\![p]\!]$
- $[\![(p \mathbin{\&} q) \mathbin{\&} s]\!] = [\![p \mathbin{\&} (q \mathbin{\&} s)]\!]$
- $[\![p \mathbin{\&} q]\!] = [\![q \mathbin{\&} p]\!]$
- $[\![p \oplus_r q]\!] = [\![q \oplus_{1-r} p]\!]$
- $[\![\left(p \oplus_{\frac{a}{a+b}} q \right) \oplus_{\frac{a+b}{a+b+c}} s]\!] = [\![p \oplus_{\frac{a}{a+b+c}} \left(q \oplus_{\frac{b}{b+c}} s \right)]\!]$

Properties of Deterministic Programs

Parallel composition & is not idempotent except in the deterministic case, neither does sequential composition distribute over & in general. However, if the term being distributed is deterministic, then the property holds.

Lemma

If p is deterministic, then

$$[\![p(q \mathbin{\&} r)]\!] = [\![pq \mathbin{\&} pr]\!] \qquad [\![(q \mathbin{\&} r)p]\!] = [\![qp \mathbin{\&} rp]\!].$$

Neither equation holds unconditionally.

A Continuous Measure

Omitting * or dup, ProbNetKAT programs can generate only discrete measures. This raises the question of whether it is possible to generate a continuous measure at all.

Theorem

Let π_0 and π_1 be distinct packets and let p be the program that changes the current packet to either π_0 or π_1 with equal probability. The measure $\mu = [\![p \,;\, (\mathsf{dup} \,;\, p)^*]\!](\pi_0, -)$ is supported by the subspace of 2^H consisting of all sets containing exactly one history of each length and linearly ordered by the suffix relation. This subspace is homeomorphic to the Cantor space and μ is the uniform (Lebesgue) measure on this space.

Thus discrete measures are not adequate.

Approximation

Every program can be approximated arbitrarily closely by a loop-free program, using a suitable notion of approximation for the iterates of a loop. This means that in many real-world applications, finite or discrete distributions suffice.

The appropriate notion of approximation is weak convergence. A sequence of measures μ_n converge weakly to μ if for all bounded continuous real-valued functions f,

$$\lim_n \int_a f(a) \cdot \mu_n(da) = \int_a f(a) \cdot \mu(da).$$

That is, the expected values of f with respect to the μ_n converge to the expected value of f with respect to μ.

Weak Convergence of $p^{(m)}$ to p^*

Define

$$p^{(0)} = \mathtt{skip} \qquad\qquad p^{(n+1)} = \mathtt{skip} \,\&\, p \,;\, p^{(n)}.$$

This is the outcome of the first n steps of the process defining p^*.

Note that $p^{(n)}$ is not p^n, nor is it $p^0 \,\&\, \cdots \,\&\, p^n$.

Theorem

The measures $[\![p^{(m)}]\!](c, -)$ *converge weakly to* $[\![p^*]\!](c, -)$.

Approximation by *-Free Programs

Theorem

All ProbNetKAT program operators are continuous with respect to weak convergence.

Corollary

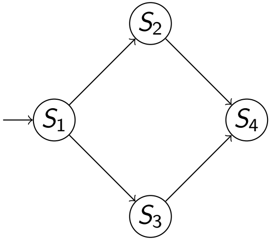
For every ProbNetKAT program p, there is a sequence of *-free programs that converge weakly to p.

Applications

Fault Tolerance

- Failures are a fact of life in real-world networks
- A recent empirical study of data center networks [Gill & al 11] found that failures occur frequently, often cause degraded performance and service disruptions
- We can model failures in ProbNetKAT: $p \oplus_d \texttt{drop}$ succeeds and executes p with probability d and fails with probability $1 - d$

Fault Tolerance



Consider the network topology pictured. We wish to forward traffic from S_1 to S_4. We know that the link $S_1 \rightarrow S_2$ fails with 10% probability and all other links are reliable. What is the probability that a packet that originates at S_1 will be successfully delivered to S_4?

The topology and failure probability are encoded as follows:

$$\begin{aligned} t \triangleq & (sw = S_1; pt = 2; ((sw \leftarrow S_2; pt \leftarrow 1) \oplus_{.9} \texttt{drop})) \\ & \& \ (sw = S_1; pt = 3; sw \leftarrow S_3; pt \leftarrow 1) \\ & \& \ (sw = S_2; pt = 4; sw \leftarrow S_4; pt \leftarrow 2) \\ & \& \ (sw = S_3; pt = 4; sw \leftarrow S_4; pt \leftarrow 3) \end{aligned}$$

(adopting the convention that each port is named according to the identifier of the switch it connects to—e.g., port 1 on switch S_2 connects to switch S_1)

Fault Tolerance

Next, encode the behavior on switches. Try two policies:

1. Send all traffic via S_2:

$$p \triangleq (sw = S_1; pt \leftarrow 2) \,\&\, (sw = S_2; pt \leftarrow 4)$$

2. Divide the traffic evenly between S_2 and S_3:

$$\begin{aligned} p' \triangleq\ & (sw = S_1; (pt \leftarrow 2 \oplus pt \leftarrow 3)) \\ & \,\&\, (sw = S_2; pt \leftarrow 4) \,\&\, (sw = S_3; pt \leftarrow 4) \end{aligned}$$

The egress predicate is $e \triangleq sw = S_4$. The complete network program is $(p\ ;\ t)^*\ ;\ e$. That is, the network alternates between forwarding on switches and topology, iterating these steps until the packet is either dropped or exits the network.

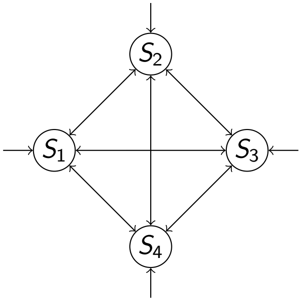
Calculations using our semantics yield a 90% chance of delivery in case 1 and a 95% chance in case 2. The positive effect with respect to failures has also been observed in previous work on randomized routing [Zhang-Shen and McKeown, IWQoS 05].

Applications

Load Balancing

- Operators must often balance traffic to avoid excessive congestion on any link
- an attractive approach to this problem is randomized routing: traffic is spread randomly over a diverse set of paths
- Valiant load balancing (VLB) [Valiant 82] is a classic randomized routing scheme that provides low expected congestion. VLB forwards packets using a simple two-phase strategy:
  1. the ingress switch forwards the packet to a randomly selected neighbor
  2. the neighbor forwards the packet to the final destination.

Load Balancing



Consider the four-node mesh topology shown. Assume that each switch has ports named $1, 2, 3, 4$, that port i on switch i connects to the outside world, and that all other ports j connect to switch j.

We can write a ProbNetKAT program for VLB this load balancing scheme by splitting it into two parts, one for each phase of routing. We can use topological information to distinguish the phases. Incoming packets (port i on switch i) are forwarded randomly, and packets on internal ports are forwarded deterministically.

Load Balancing

The initial (random) phase is modeled by

$$p_1 \triangleq \mathop{\&}_{k=1}^{4} (sw = k \; ; \; pt = k \; ; \bigoplus_{j \neq k} pt \leftarrow j)$$

The second (deterministic) phase is modeled by:

$$p_2 \triangleq \left(\mathop{\&}_{k=1}^{4} (sw = k \; ; \; pt \neq k) \right) ; \left(\mathop{\&}_{k=1}^{4} (dst = k \; ; \; pt \leftarrow k) \right)$$

The guards $sw = k \; ; \; pt \neq k$ restrict to second-phase packets. The overall switch term p is $p_1 \;\&\; p_2$.

The topology term t is encoded as before.

Load Balancing

VLB can route $nr/2$ load in a network with n switches and internal links with capacity r. In our example, $n = 4$ and $r = 1$, so we can route two packets with no expected congestion. We model this demand with a term d that generates two packets with random origins and random destinations:

$$d \triangleq (\bigoplus_{k=1}^{4}(\pi_{k,k,0}!) \mathbin{\&} \bigoplus_{k=1}^{4}(\pi_{k,k,1}!)) \mathbin{;} (\bigoplus_{k=1}^{4} dst \leftarrow k)$$

The full network program to analyze is then $d \mathbin{;} (p \mathbin{;} t)^* \mathbin{;} p$.

Defining $X_{\mathrm{max}} \triangleq$ the maximum number of packets traversing a single internal link and using the semantics of ProbNetKAT, we find that the expected value of X_{max} is one packet, as predicted.

Applications

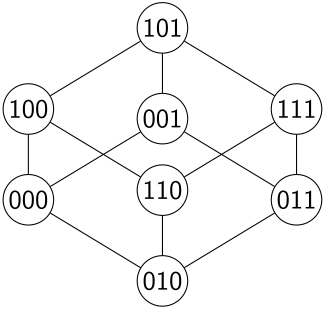
Gossip Protocols

- used to efficiently disseminate information in large-scale distributed systems [Demers 1987]
- tend to converge rapidly to a consistent global state while only requiring bounded worst-case communication
- in each round, every node communicates with a randomly selected peer and the nodes update their state using information shared during the exchange

We can use ProbNetKAT to model the convergence of gossip protocols

- introduce a single packet to model the "rumor" being gossiped by the system
- when a node receives the packet, it randomly selects one of its neighbors to infect

Gossip Protocols



We run this protocol on a hypercube. Gossiping on a hypercube is highly uniform: numbering the switches in binary, we can randomly select a neighbor by flipping a single bit.

The fragment of the switch program p for switch 000 is as follows:

$$sw = 000 \; ; \; ((pt \leftarrow 001 \oplus pt \leftarrow 010 \oplus pt \leftarrow 100) \;\&\; pt \leftarrow 0)$$

The overall forwarding policy is obtained by combining analogous fragments for the other switches using parallel composition (&).

Gossip Protocols

Encoding the hypercube topology as t as before, we can analyze $(p\ ;\ t)^*$ and calculate the expected number of infected nodes after a given number of rounds X_{infected} using the ProbNetKAT semantics. The results for the first few rounds are shown.

This captures the usual behavior of a push-based gossip protocol.

Rounds	$E[X_{\text{infected}}]$
0	1.00
1	2.00
2	3.33
3	4.86
4	6.25
5	7.17
6	7.66

Related Work

Probabilistic Programming

Computational models, semantics, and logics for probabilistic programs have been extensively studied for many years [SahebDjahromi 78; Ramshaw 79; Kozen 79, 83; Morgan & al 96; Kozen & al 13; Larsen & al 12; Gordon & al 14; Gretz & al 15]. Our semantics for ProbNetKAT builds on these foundations and extends it to the new domain of network programming.

Probabilistic programming in AI

has also been extensively studied [Roy 11; Gordon 11]. However, the goals of this work are somewhat different in that it focuses on Bayesian inference.

Related Work

Probabilistic Automata

Probabilistic automata in several forms have been a popular model going back to early work of [Paz 71], as well as many other more recent efforts [Segala 06, 95; McIver & al 08; Larsen & Skou 91, 92; Desharnais & al 02; Cattani & Segala 02; Jonsson & Larsen 91; Kwiatkowska & al 02, 07; Baier 08; Abate & al 14]. Probabilistic automata are a suitable operational model for probabilistic programs and play a crucial role in the development of decision procedures for bisimulation equivalence, synthesis of probabilistic programs, and model checking In the present paper, we do not touch upon any of these issues so the connections to probabilistic automata theory are thin. However, we expect they will play an important role in our future work.

Related Work

Probability & Nondeterminism

Denotational models combining probability and nondeterminism have been proposed in papers by several authors [Jones & Plotkin 89; Jones 90; McIver & Morgan 96, 05; Varacca & Winskel 06; Tix & al 09]. Because ProbNetKAT does not have nondeterminism, we have not encountered the extra challenges arising in the combination of nondeterministic and probabilistic behavior.

Labeled Markov Processes

General models for labeled Markov processes, primarily based on Markov kernels, have been studied extensively [Panangaden 98, 09; Doberkat 07]. Our use of Markov processes is influenced heavily by these works.

Related Work

Network Programming

Recent years have seen an incredible growth of languages and systems for programming and reasoning about networks: Frenetic [Foster & al 11]; Pyretic [Monsanto & al 13]; Maple [Voellmy & al 13]; NetKAT [Anderson & al 14]; and FlowLog [Nelson & al 14]. However, as mentioned previously, all of these language are based on deterministic packet-processing functions and do not handle probabilistic traffic models or forwarding policies. Of all these frameworks, NetKAT is the most closely related as ProbNetKAT builds directly on its features.

Network Verification Tools

In addition to programming languages, a number of network verification tools have emerged: Header Space Analysis [Kazemian & al 12]; VeriFlow [Khurshid & al 13]; the NetKAT verifier [Foster & al 15]; and Libra [Zeng & al 14]. Like the network programming languages described above, these tools only model deterministic networks and verify deterministic properties.

Related Work

Network Calculus

Network calculus is a general framework for analyzing network behavior using tools from queuing theory [Cruz 91; Le Boudec & Thiran 01; Jiang 06]. It models the low-level behavior of network devices in significant detail, including features such as traffic arrival rates, switch propagation delays, and the behaviors of components like buffers and queues. This enables reasoning about quantitative properties such as latency, bandwidth, and congestion. Like ProbNetKAT, stochastic network calculus provides tools for reasoning about the probabilistic behavior, especially in the presence of statistical multiplexing. However, network calculus is generally considered difficult to use, since it requires external facts from queuing theory to establish many desired results. In contrast, ProbNetKAT is a self-contained, language-based framework that offers general programming constructs and a complete denotational semantics.

Conclusion

- We have introduced Probabilistic NetKAT (ProbNetKAT), a conservative extension of NetKAT with constructs for reasoning about the probabilistic behavior of networks. To our knowledge, this is the first language-based framework for specifying and verifying probabilistic network behavior.
- We have developed a formal semantics for ProbNetKAT based on Markov kernels and shown that the extension is conservative over NetKAT.
- We have determined the appropriate notion of approximation and have shown that every ProbNetKAT program is arbitrarily closely approximated by loop-free programs.
- We have presented several case studies that illustrate the use of ProbNetKAT on examples involving fault tolerance, load balancing, and a probabilistic gossip protocol.

Conclusion

Future Directions

Our examples have used the semantic definitions directly in the calculation of distributions. Although we have exploited several general properties of our system in these arguments, we have made no attempt to assemble them into a formal deductive system or decision procedure as done previously for NetKAT [Anderson & al 14, Foster & al 15]. These questions remain topics for future investigation.

As a more practical next step, we would like to augment the existing NetKAT compiler [Smolka & al 15] with tools for handling the probabilistic constructs of ProbNetKAT along with a formal proof of correctness. Features such as OpenFlow [McKeown & al 08] "group tables" support simple forms of randomization and emerging platforms such as P4 [Bosshart 14] offer additional flexibility. Hence, there already exist machine platforms that could serve as a compilation target for restricted fragments of ProbNetKAT.

Thanks to . . .

- Cornell PLDG
- DIKU COPLAS
- National Security Agency
- National Science Foundation CNS-1111698, CNS-1413972, CCF-1422046, CCF-1253165, CCF-1535952
- Office of Naval Research N00014-15-1-2177
- Dutch Research Foundation (NWO) 639.021.334, 612.001.113
- gifts from Cisco, Facebook, Google, and Fujitsu

For papers and code, please visit:

http://frenetic-lang.org/

Thanks!