$P\lambda\omega NK$

Functional Probabilistic NetKAT

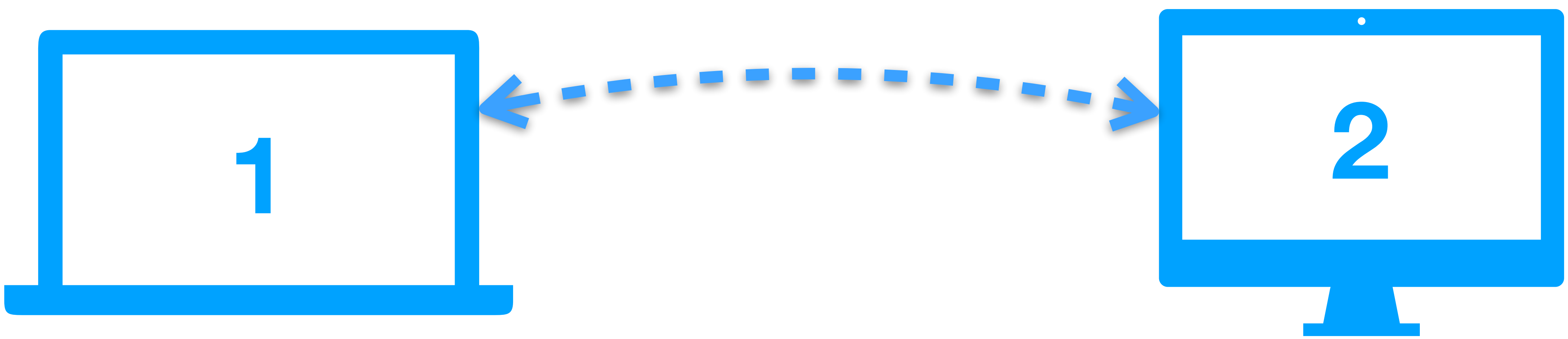
Alexander Vandenbroucke & Tom Schrijvers

KU LEUVEN

NetKAT

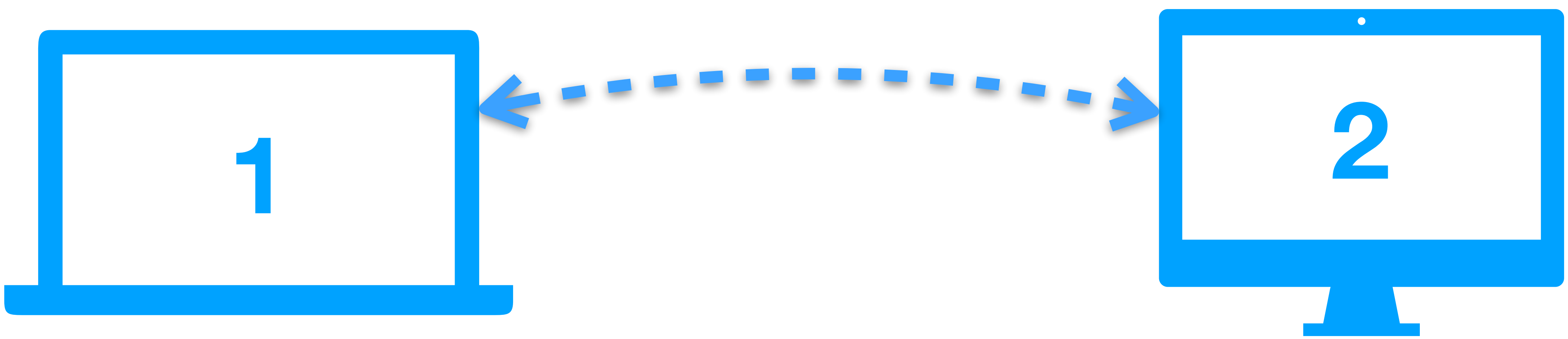
Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, David Walker:
NetKAT: semantic foundations for networks. POPL 2014: 113-126

NetKAT



Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, David Walker:
NetKAT: semantic foundations for networks. POPL 2014: 113-126

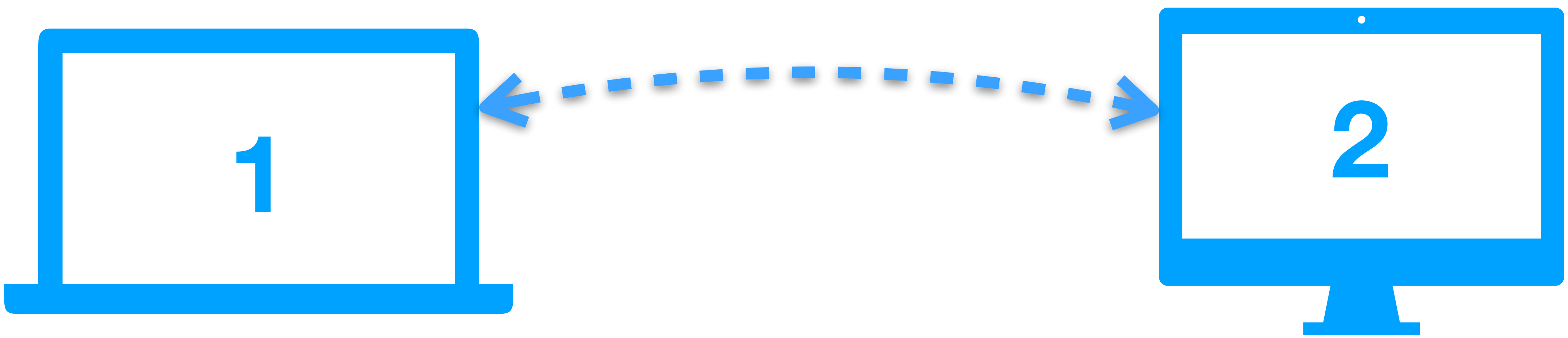
NetKAT



`(sw = 1; sw ← 2) & (sw = 2; sw ← 1)`

Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, David Walker:
NetKAT: semantic foundations for networks. POPL 2014: 113-126

NetKAT

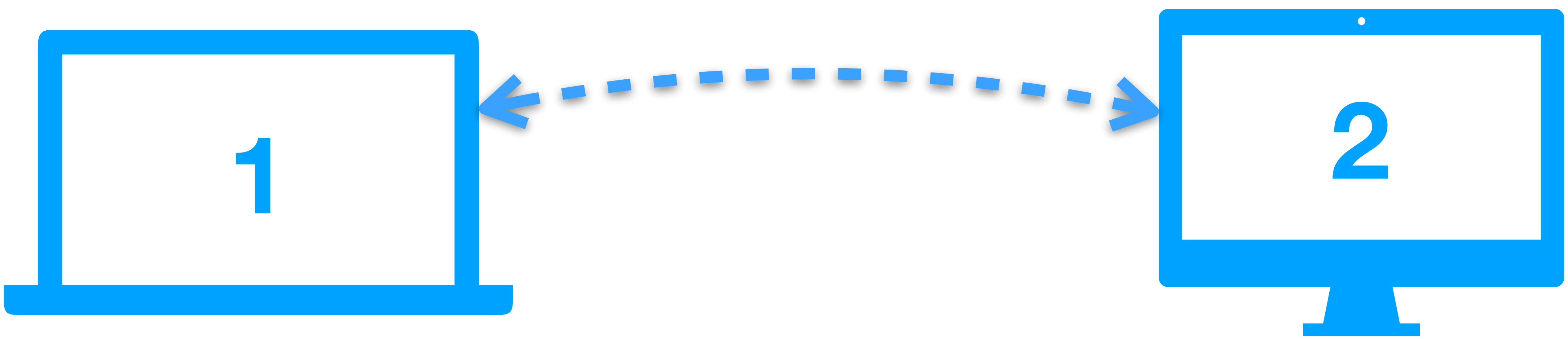


(sw = 1; sw ← 2) & (sw = 2; sw ← 1)

if node 1

Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, David Walker:
NetKAT: semantic foundations for networks. POPL 2014: 113-126

NetKAT



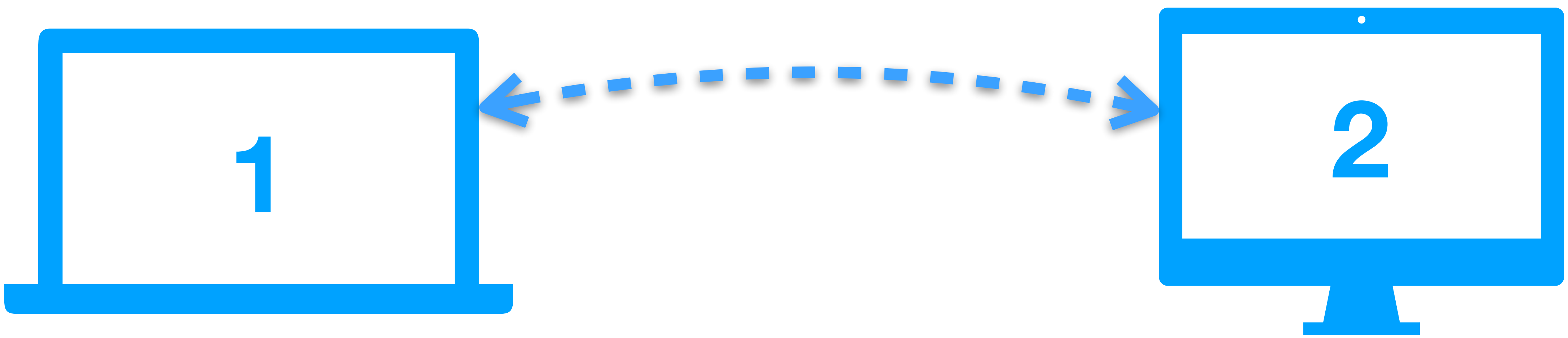

```
(sw = 1; sw ← 2) & (sw = 2; sw ← 1)
```

if node 1

send to node 2

Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, David Walker:
NetKAT: semantic foundations for networks. POPL 2014: 113-126

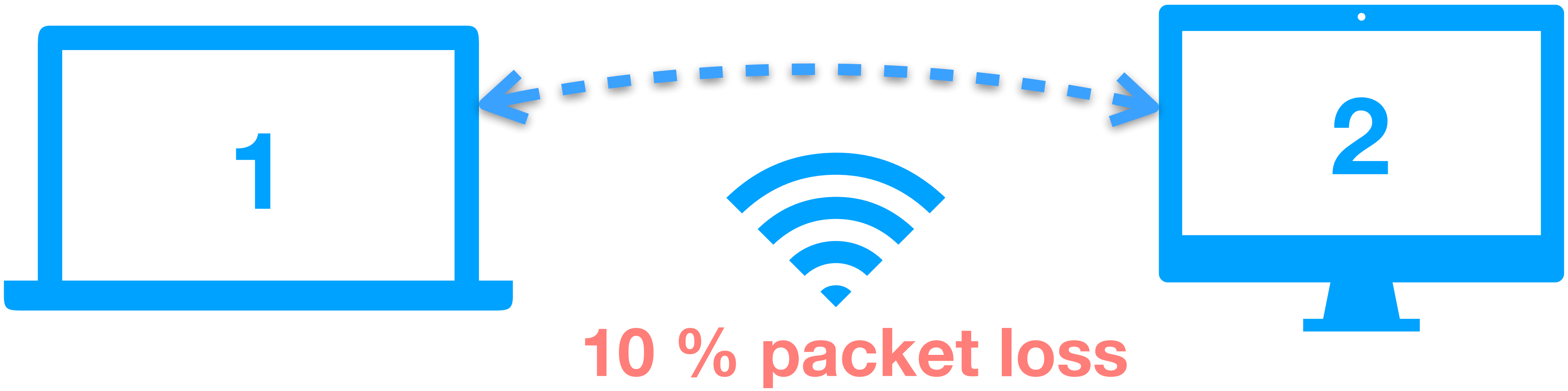
NetKAT



$$\underbrace{(\mathtt{sw} = 1;}_{\text{if node 1}}\ \underbrace{\mathtt{sw} \leftarrow 2)}_{\text{send to node 2}}\ \&\ \underbrace{(\mathtt{sw} = 2;}_{\text{if node 2}}\ \underbrace{\mathtt{sw} \leftarrow 1)}_{\text{send to node 1}}$$

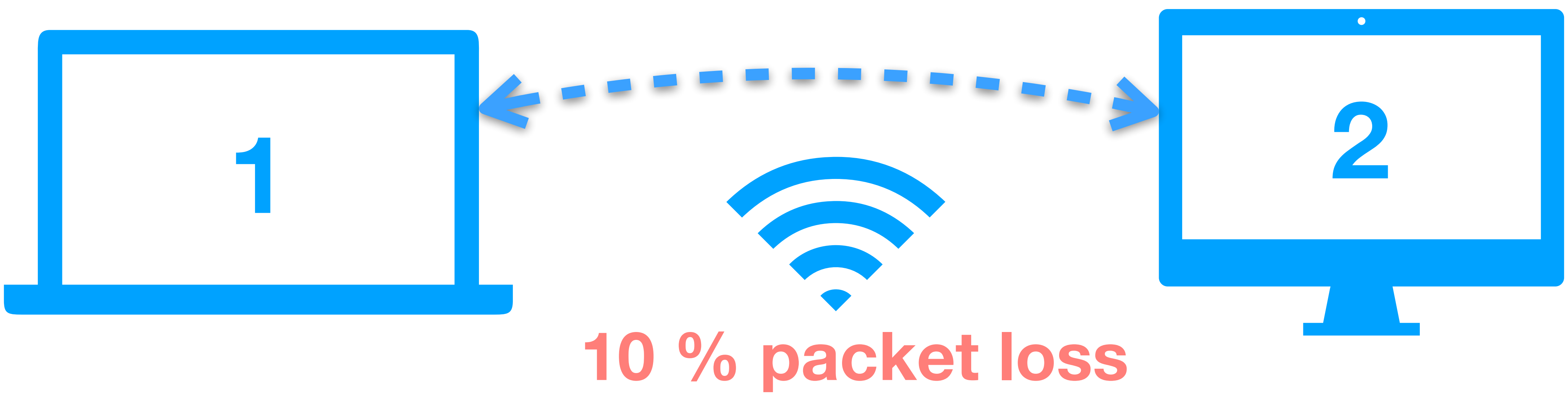
Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, David Walker:
NetKAT: semantic foundations for networks. POPL 2014: 113-126

Probabilistic NetKAT



Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, Alexandra Silva: Probabilistic NetKAT. ESOP 2016: 282-309

Probabilistic NetKAT



`(sw = 1; sw ← 2` $\oplus_{0.9}$ `drop) & (sw = 2; sw ← 1` $\oplus_{0.9}$ `drop)`

Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, Alexandra Silva: Probabilistic NetKAT. ESOP 2016: 282-309

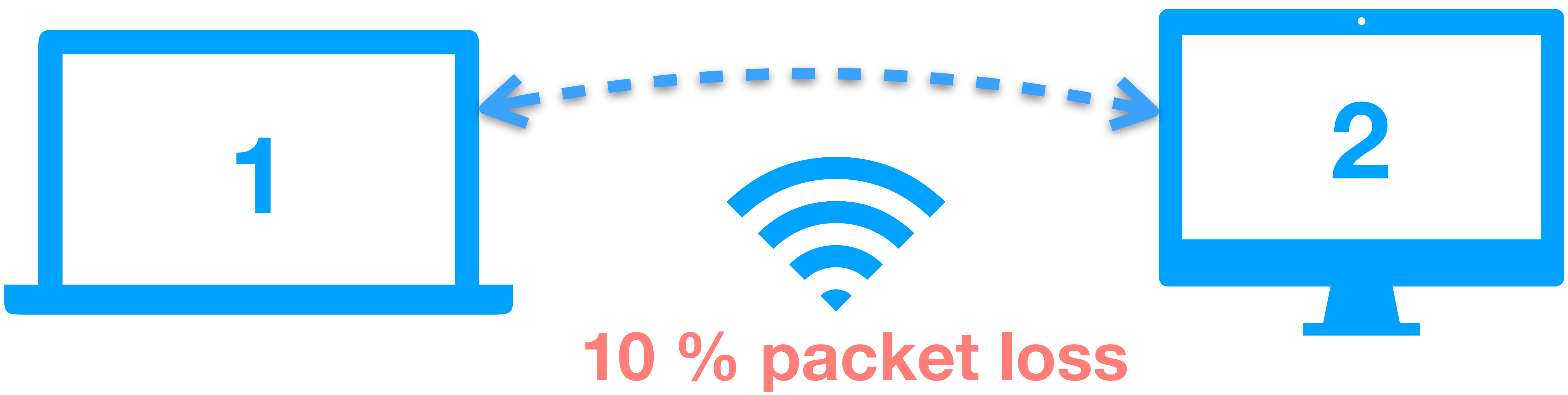
Probabilistic NetKAT

(sw = 1; sw ← 2 $\oplus_{0.9}$ drop) & (sw = 2; sw ← 1 $\oplus_{0.9}$ drop)

if node 1

Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, Alexandra Silva: Probabilistic NetKAT. ESOP 2016: 282-309

Probabilistic NetKAT



$$(\texttt{sw} = 1;\ \texttt{sw} \leftarrow 2 \oplus_{0.9} \texttt{drop})\ \&\ (\texttt{sw} = 2;\ \texttt{sw} \leftarrow 1 \oplus_{0.9} \texttt{drop})$$

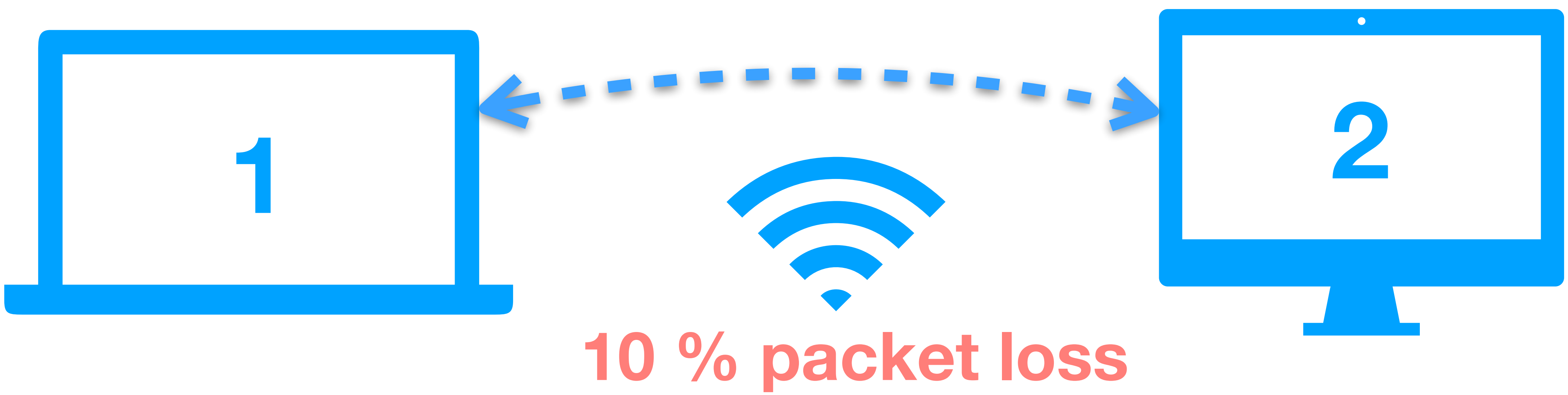
if node 1

send to node 2

90%

Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, Alexandra Silva: Probabilistic NetKAT. ESOP 2016: 282-309

Probabilistic NetKAT



$$(\mathtt{sw = 1;\ sw} \leftarrow \mathtt{2} \oplus_{0.9} \mathtt{drop}) \mathbin{\&} (\mathtt{sw = 2;\ sw} \leftarrow \mathtt{1} \oplus_{0.9} \mathtt{drop})$$

if node 1

send to node 2

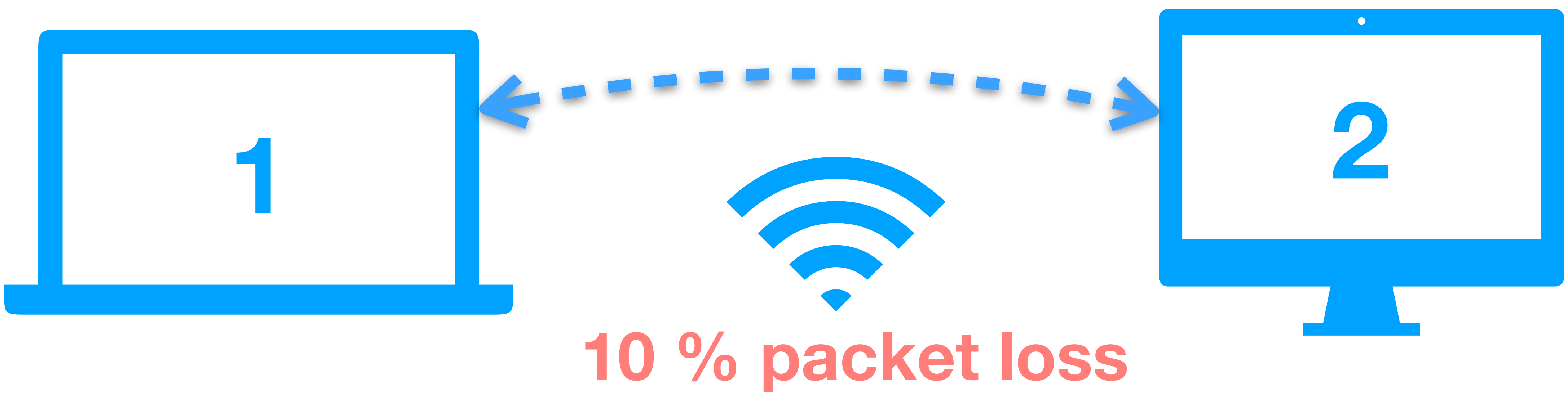
or drop packet

90%

10%

Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, Alexandra Silva: Probabilistic NetKAT. ESOP 2016: 282-309

Probabilistic NetKAT



$$(\texttt{sw = 1; sw} \leftarrow \texttt{2} \oplus_{0.9} \texttt{drop}) \;\&\; (\texttt{sw = 2; sw} \leftarrow \texttt{1} \oplus_{0.9} \texttt{drop})$$

if node 1	send to node 2	or drop packet	if node 2	send to node 1	or drop packet
	90%	10%		90%	10%

Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, Alexandra Silva: Probabilistic NetKAT. ESOP 2016: 282-309

Queries

10 % packet loss

$(\texttt{sw = 1; sw} \leftarrow \texttt{2} \oplus_{0.9} \texttt{drop}) \;\&\; (\texttt{sw = 2; sw} \leftarrow \texttt{1} \oplus_{0.9} \texttt{drop})$

Probability Distribution

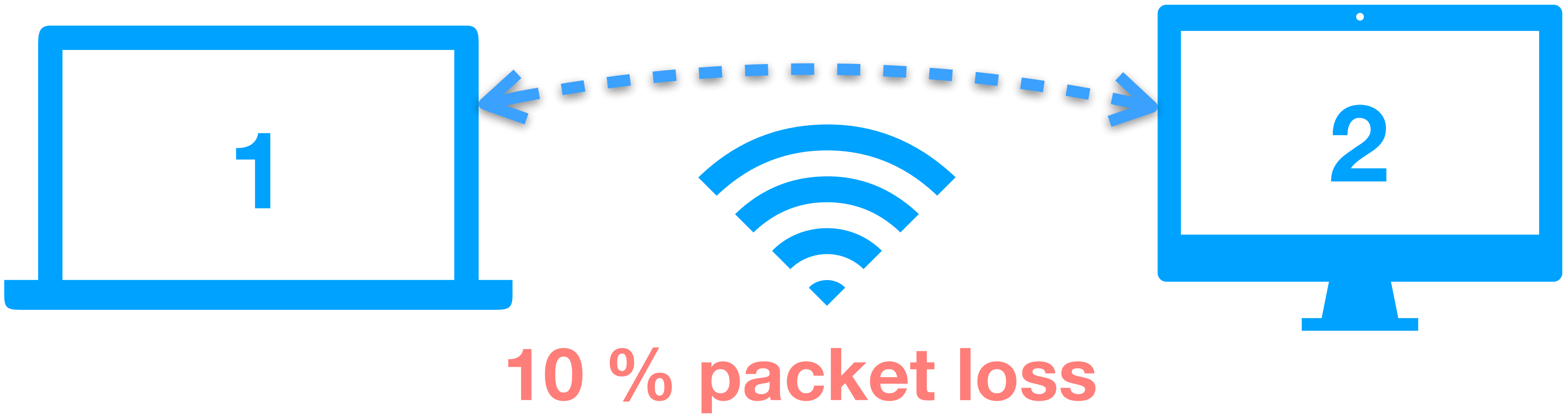
Approximate Inference

- expected latency
- fault tolerance

Exact Inference

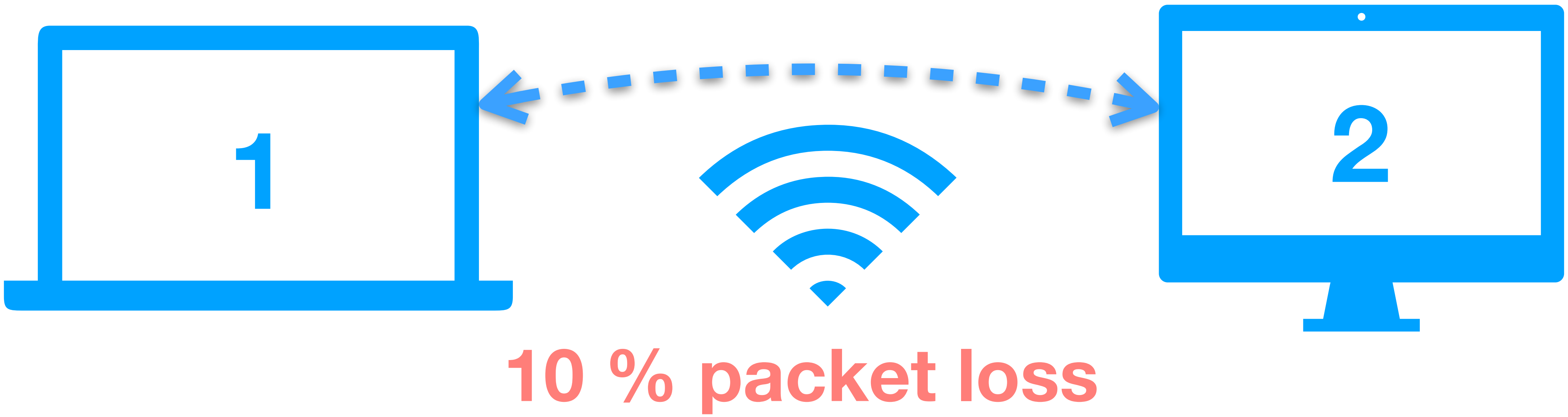
- absence of routing loops
- fault tolerance
- (re-)routing correctness

Functions



$$(\texttt{sw = 1; sw} \leftarrow \texttt{2} \oplus_{0.9} \texttt{drop}) \;\&\; (\texttt{sw = 2; sw} \leftarrow \texttt{1} \oplus_{0.9} \texttt{drop})$$

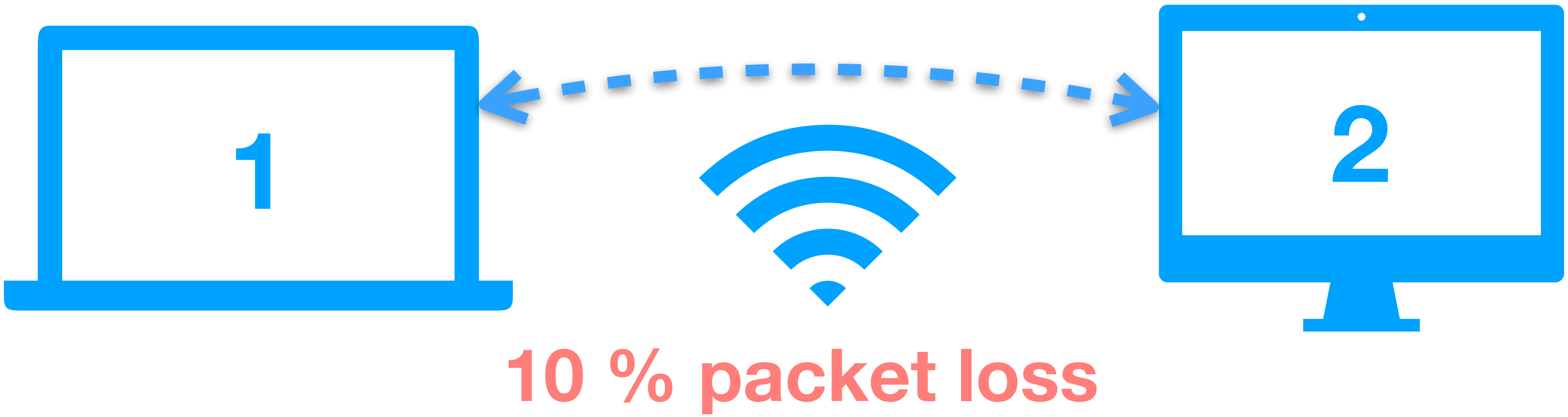
Functions



forward = λsrc.λdst.sw = src; sw $\leftarrow$ dst $\oplus_{0.9}$ drop

(sw = 1; sw $\leftarrow$ 2 $\oplus_{0.9}$ drop) & (sw = 2; sw $\leftarrow$ 1 $\oplus_{0.9}$ drop)

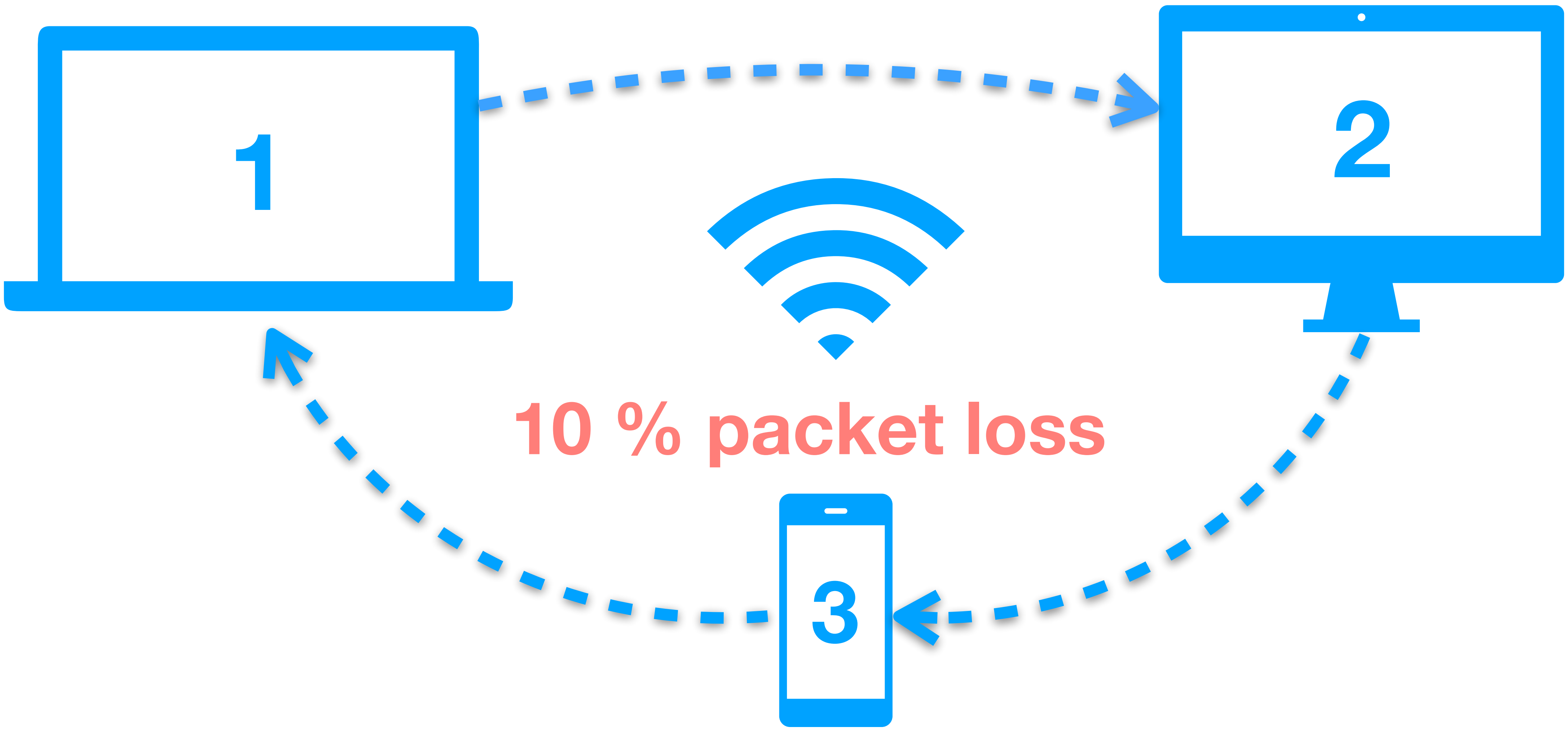
Functions



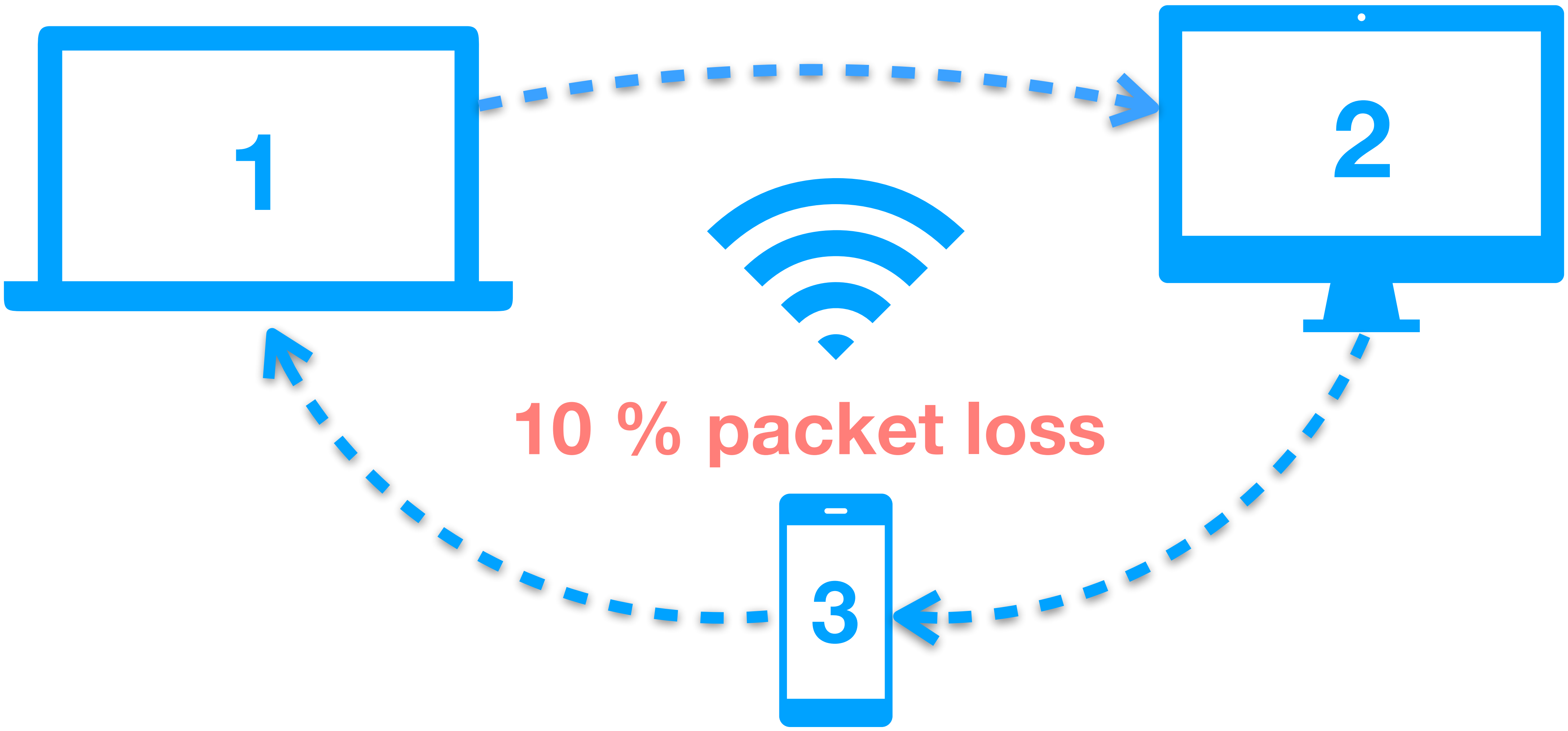
$\texttt{forward} = \lambda\texttt{src}.\lambda\texttt{dst}.\texttt{sw} = \texttt{src};\ \texttt{sw} \leftarrow \texttt{dst} \oplus_{0.9} \texttt{drop}$

`forward 1 2 & forward 2 1`

Modifiability

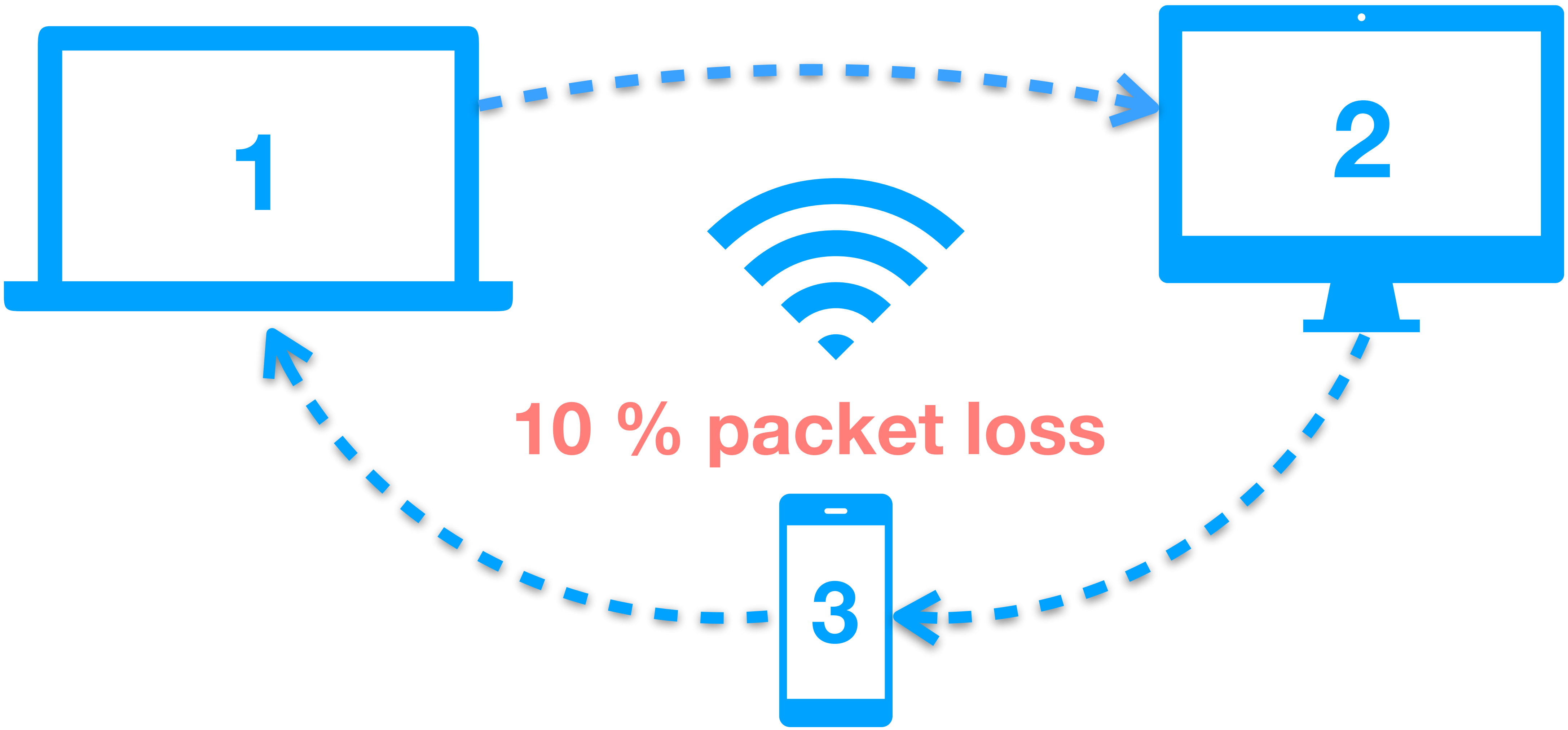


Modifiability



$$
\begin{aligned}
&(\mathtt{sw} = 1;\ \mathtt{sw} \leftarrow 2 \oplus_{0.9} \mathtt{drop})\\
\&\ &(\mathtt{sw} = 2;\ \mathtt{sw} \leftarrow 3 \oplus_{0.9} \mathtt{drop})\\
\&\ &(\mathtt{sw} = 3;\ \mathtt{sw} \leftarrow 1 \oplus_{0.9} \mathtt{drop})
\end{aligned}
$$

Modifiability



`forward 1 2 & forward 2 3 & forward 3 1`

What is P$\lambda\omega$NK?

What is P$\lambda\omega$NK?

NetKAT

Networking Primitives

$\downarrow$

- **Equivalence**
- **Compilation**

What is P$\lambda\omega$NK?

NetKAT + **Prob NetKAT**

Networking Primitives + **Probabilistic Choice**

↓ ↓

- **Equivalence**
- **Compilation**

- **Approximation**
- **Exact inference**

What is P$\lambda\omega$NK?

NetKAT

Networking Primitives

+

Prob NetKAT

Probabilistic Choice

+

P$\lambda\omega$NK

Functional Syntax

- Equivalence
- Compilation

- Approximation
- Exact inference

- Approximation
- Exact inference

What is P$\lambda\omega$NK?

NetKAT

Networking Primitives

- Equivalence
- Compilation

Prob NetKAT

Probabilistic Choice

- Approximation
- Exact inference

P$\lambda\omega$NK

Functional Syntax

- Approximation
- Exact inference

Compilation
by Strong Normalisation
(see later)

Syntax

What is P$\lambda\omega$NK?

NetKAT

Networking Primitives

+

Prob NetKAT

Probabilistic Choice

+

P$\lambda\omega$NK

Functional Syntax

What is P$\lambda\omega$NK?

NetKAT

Networking Primitives

```
skip drop dup
f = n   f ← n
& ; ¬ *
```

Prob NetKAT

Probabilistic Choice

P$\lambda\omega$NK

Functional Syntax

What is P$\lambda\omega$NK?

NetKAT

Networking Primitives

+

Prob NetKAT

Probabilistic Choice

+

P$\lambda\omega$NK

Functional Syntax

```
skip drop dup
f = n   f ← n
& ; ¬ *
```

$\oplus_r$

What is P$\lambda\omega$NK?

NetKAT + **Prob NetKAT** + **P$\lambda\omega$NK**

Networking Primitives

```
skip drop dup
f = n   f ← n
& ; ¬ *
```

Probabilistic Choice

$\oplus_r$

Functional Syntax

```
λx:S.E  E E
thunk(E) force(V)
produce(V) unit
E to x.E
```

What is P$\lambda\omega$NK?

NetKAT

Networking Primitives

```
skip drop dup
f = n   f ← n
& ; ¬ *
```

Prob NetKAT

Probabilistic Choice

$\oplus_r$

P$\lambda\omega$NK

Functional Syntax

```
λx:S.E  E E
thunk(E) force(V)
produce(V) unit
E to x.E
```

Inspired by Call-by-Push-Value (CBPV)

Paul Blain Levy. 2001. Call-by-push-value. Ph.D. Dissertation. Queen Mary University of London, UK.

Semantics Blues

Semantics Blues

NetKAT

Networking Primitives

+

Prob NetKAT

Probabilistic Choice

+

P$\lambda\omega$NK

Functional Syntax

Semantics Blues

NetKAT

Networking Primitives

+

Prob NetKAT

Probabilistic Choice

+

P$\lambda\omega$NK

Functional Syntax

Sets of Packet Histories

Semantics Blues

NetKAT

Networking Primitives

+

Prob NetKAT

Probabilistic Choice

+

P$\lambda\omega$NK

Functional Syntax

↓

Sets of Packet Histories

↓

Continous Measures of sets of Packet Histories

Semantics Blues

NetKAT

Networking Primitives

$\rightarrow$ **Sets of Packet Histories**

+

Prob NetKAT

Probabilistic Choice

$\rightarrow$ **Continous Measures of sets of Packet Histories**

+

P$\lambda\omega$NK

Functional Syntax

$\rightarrow$ **Measures of sets of Packet Histories and Functions**

Semantics Blues

Measure theory does not support Higher Order Functions

NK

onal
ax

Sets of Packet Histories

Continous Measures of sets of Packet Histories

Measures of sets of Packet Histories and Functions

Semantics Blues

Measure theory does not support Higher Order Functions

QBS (Measures + HO Functions)

NK

onal
ax

Sets of Packet Histories

Continous Measures of sets of Packet Histories

Measures of sets of Packet Histories and Functions

Chris Heunen, Ohad Kammar, Sam Staton, and Hangseok Yang. 2017. A convenient category for higher-order probability theory. In LICS. IEE Computer Society, 1-12

Semantics Blues

Measure theory does not support Higher Order Functions

↓

QBS (Measures + HO Functions)

ωQBS (Measures + HO Functions + Iteration)

NK

onal

ax

Sets of Packet Histories

Continous Measures of sets of Packet Histories

Measures of sets of Packet Histories and Functions

Chris Heunen, Ohad Kammar, Sam Staton, and Hangseok Yang. 2017. A convenient category for higher-order probability theory. In LICS. IEE Computer Society, 1-12

Mathijs Vákár, Ohad Kammar, and Sam Staton. 2019. A domain theory for statistical probabilistic programming PACMPL 3, POPL (2019), 36:1-36:29

Semantics Blues

Measure theory does not support Higher Order Functions

↓

QBS (Measures + HO Functions)

ωQBS (Measures + HO Functions + Iteration)

↓

Conservative semantics

NK

onal

ax

Sets of Packet Histories

Continous Measures of sets of Packet Histories

Measures of sets of Packet Histories and Functions

Chris Heunen, Ohad Kammar, Sam Staton, and Hangseok Yang. 2017. A convenient category for higher-order probability theory. In LICS. IEE Computer Society, 1-12

Mathijs Vákár, Ohad Kammar, and Sam Staton. 2019. A domain theory for statistical probabilistic programming PACMPL 3, POPL (2019), 36:1-36:29

Compilation

Compilation

$(\underline{\lambda \mathtt{x}:\mathbb{N}.\mathtt{produce(x)\ 1}})\ \mathtt{to\ y.sw} \leftarrow \mathtt{y}$

P$\lambda\omega$NK

PNK

Compilation

$$(\underline{\lambda x{:}\mathbb{N}.\texttt{produce}(x)\ 1})\ \texttt{to}\ y.\texttt{sw} \leftarrow y$$

$$\Downarrow \quad \textbf{reduce}$$

$$\underline{\texttt{produce}(1)\ \texttt{to}\ y.\ \texttt{sw} \leftarrow y}$$

P$\lambda\omega$NK

PNK

Compilation

$\texttt{(}\underline{\lambda\texttt{x:}\mathbb{N}\texttt{.produce(x) 1}}\texttt{) to y.sw} \leftarrow \texttt{y}$

$\Downarrow$ reduce

$\underline{\texttt{produce(1) to y. sw} \leftarrow \texttt{y}}$

$\Downarrow$ reduce

$\texttt{sw} \leftarrow \texttt{1}$

P$\lambda\omega$NK

PNK

Compilation

$\texttt{(}\underline{\lambda\texttt{x}:\mathbb{N}.\texttt{produce(x) 1}}\texttt{) to y.sw} \leftarrow \texttt{y}$

$\Downarrow$ reduce

$\underline{\texttt{produce(1) to y. sw} \leftarrow \texttt{y}}$

$\Downarrow$ reduce

$\texttt{sw} \leftarrow \texttt{1}$

P$\lambda\omega$NK

erase

PNK

$\texttt{sw} \leftarrow \texttt{1}$

Compilation

$$(\underline{\lambda x{:}\mathbb{N}.\mathtt{produce}(x)\ 1})\ \mathtt{to}\ y.\mathtt{produce}(y)$$

$$\Downarrow \quad \textsf{reduce}$$

$$\underline{\mathtt{produce}(1)\ \mathtt{to}\ y.\ \mathtt{produce}(y)}$$

$$\Downarrow \quad \textsf{reduce}$$

$$\mathtt{produce}(1)$$

P$\lambda\omega$NK

$$\downarrow \quad \textsf{erase}$$

PNK

$$\mathtt{skip}$$

Compilation

<u>`(skip & skip) to x.p`</u>

$\Downarrow$ reduce?

<u>`skip to x.p`</u> `&` <u>`skip to x.p`</u>

$\Downarrow^*$

`p' & p’`

P$\lambda\omega$NK erase

PNK

`p' & p’`

Compilation

<u>(skip & skip) to x.p</u>   [|(skip & skip) to x.p|]

$\Downarrow$ reduce?

<u>skip to x.p</u> & <u>skip to x.p</u> $\not\cong$

$\Downarrow^{*}$

p' & p'

PλωNK   erase

PNK   p' & p'   [|p' & p'|]

Compilation

<u>(skip & skip) to x.p</u>

$\Downarrow$ reduce?

<u>skip to x.p</u> & <u>skip to x.p</u>

$\Downarrow^{*}$

p' & p'

P$\lambda\omega$NK

$\downarrow$ erase

PNK

p' & p'

[|(skip & skip) to x.p|]

$\cong$ idempotent

[|skip to x.p|]

$\cong$ neutral

[|p'|]

$\ncong$

[|p' & p'|]

Compilation

```
skip & skip to x.p
```

$\Downarrow$ reduce

```
skip & skip; [x → unit]p
```

$\Downarrow^{*}$

```
skip & skip; p'
```

P$\lambda\omega$NK erase

PNK

```
skip & skip; p'
```

Compilation

`skip & skip to x.p`

$\Downarrow$ reduce

`skip & skip; [x → unit]p`

$\Downarrow^*$

`skip & skip; p'`

PλωNK

erase

PNK

`skip & skip; p'`

Solution: Use a type system to restrict parallelism to <u>unit</u> computations

G ⊢ E1 & E2 : P(1)

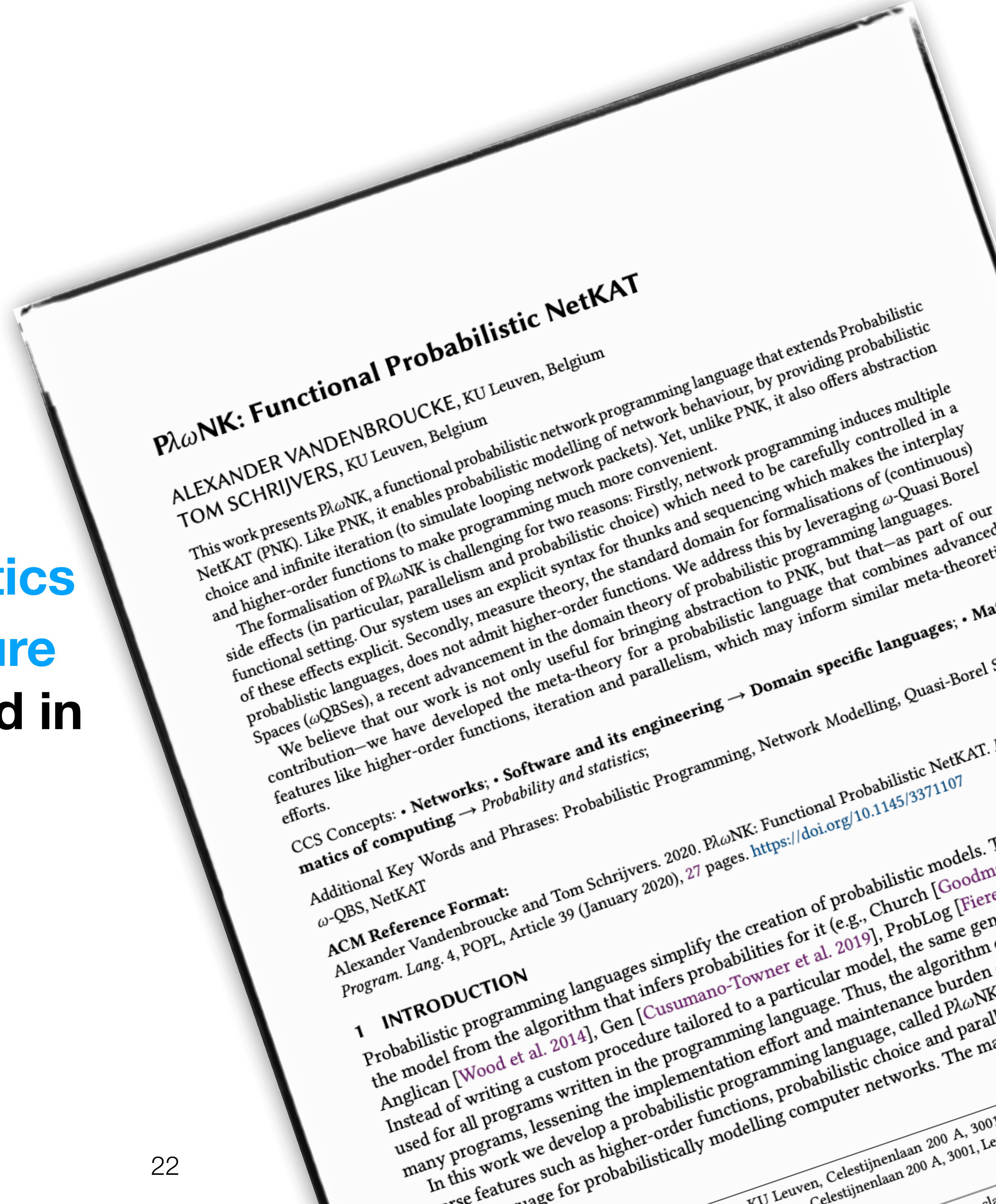
Wrapping Up

Summary

PL$\lambda\omega$NK

- ★ **Functional network modelling**
  - **+packet state**
  - **+parallelism**
  - **+probabilities**
- ★ **Higher-Order Semantics: ωQBS**
- ★ **Compilation to Probabilistic NetKAT**

> In the paper:
> > **background**,
> > **denotational semantics**
> > **compilation procedure** **(partially mechanised in Abella)**



PλωNK: Functional Probabilistic NetKAT

ALEXANDER VANDENBROUCKE, KU Leuven, Belgium
TOM SCHRIJVERS, KU Leuven, Belgium

This work presents PλωNK, a functional probabilistic network programming language that extends Probabilistic NetKAT (PNK). Like PNK, it enables probabilistic modelling of network behaviour, by providing probabilistic choice and infinite iteration (to simulate looping network packets). Yet, unlike PNK, it also offers abstraction and higher-order functions to make programming much more convenient.

The formalisation of PλωNK is challenging for two reasons: Firstly, network programming induces multiple side effects (in particular, parallelism and probabilistic choice) which need to be carefully controlled in a functional setting. Our system uses an explicit syntax for thunks and sequencing which makes the interplay of these effects explicit. Secondly, measure theory, the standard domain for formalisations of (continuous) probablistic languages, does not admit higher-order functions. We address this by leveraging ω-Quasi Borel Spaces (ωQBSes), a recent advancement in the domain theory of probabilistic programming languages.

We believe that our work is not only useful for bringing abstraction to PNK, but that—as part of our contribution—we have developed the meta-theory for a probabilistic language that combines advanced features like higher-order functions, iteration and parallelism, which may inform similar meta-theoreti
efforts.

CCS Concepts: • **Networks**; • **Software and its engineering** → **Domain specific languages**; • **Ma**
matics of computing → *Probability and statistics*;

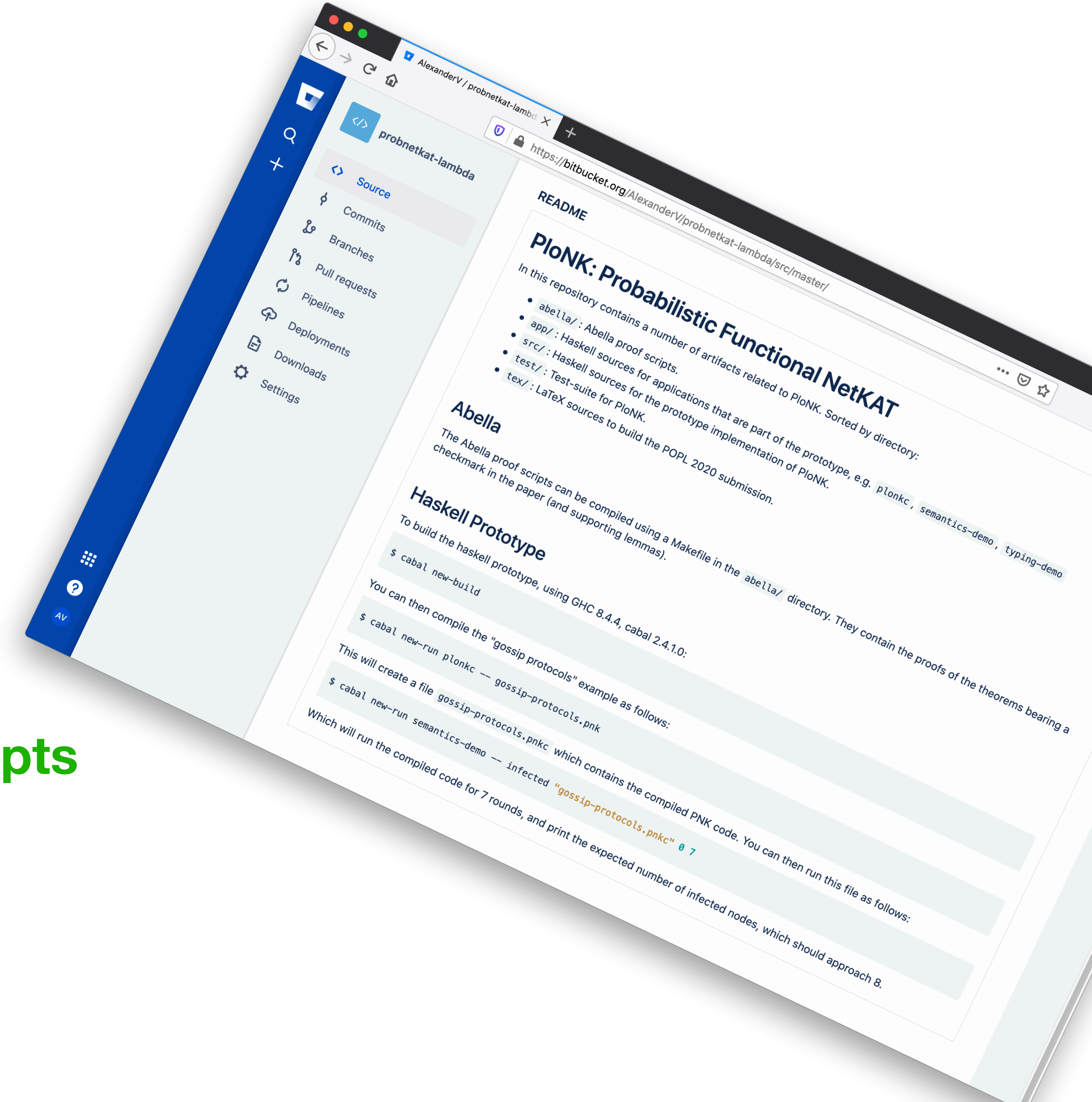
Additional Key Words and Phrases: Probabilistic Programming, Network Modelling, Quasi-Borel S
ω-QBS, NetKAT

ACM Reference Format:
Alexander Vandenbroucke and Tom Schrijvers. 2020. PλωNK: Functional Probabilistic NetKAT.
Program. Lang. 4, POPL, Article 39 (January 2020), 27 pages. https://doi.org/10.1145/3371107

1 INTRODUCTION

Probabilistic programming languages simplify the creation of probabilistic models. T
the model from the algorithm that infers probabilities for it (e.g., Church [Goodm
Anglican [Wood et al. 2014], Gen [Cusumano-Towner et al. 2019], ProbLog [Fiere
Instead of writing a custom procedure tailored to a particular model, the same gen
used for all programs written in the programming language. Thus, the algorithm
many programs, lessening the implementation effort and maintenance burden
In this work we develop a probabilistic programming language, called PλωNK
rse features such as higher-order functions, probabilistic choice and parall
uage for probabilistically modelling computer networks. The ma

- On bitbucket
  - **prototype implementation**,
  - **examples**
  - **Abella proof scripts**



Future Work

- ★ **Investigate alternatives to CBPV: FG-CBV**
- ★ **Links with Logic Programming**
- ★ **Other Semantics**

```
λquestion.
  dst ← answer question ⊕_{0.1} panic
panic = drop*
```