Formal Abstractions for Packet Scheduling

A. Mohan, Y. Liu, N. Foster, T. Kappé, D. Kozen

Has let me use his slides!
Software-defined networking made networks programmable.
Software-defined networking made networks programmable.

We want control over packet scheduling.
Software-defined networking made networks programmable.

We want control over packet scheduling.
Software-defined networking made networks programmable.

We want control over packet scheduling.

Basic tools work fine...
Software-defined networking made networks programmable.

We want control over packet scheduling.

Basic tools work fine…
Software-defined networking made networks programmable.

We want control over *packet scheduling*.

Basic tools work fine...
…but modern scheduling requires more.
...but modern scheduling requires more.

R traffic goes to either Pittsburgh or Toronto.
Interleave \( R \) and \( B \); interleave \( P \) and \( T \).

\( R \) traffic goes to either Pittsburgh or Toronto.

Goal:

...but modern scheduling requires more.
...but modern scheduling requires more.

Traffic goes to either Pittsburgh or Toronto.

Goal:
...but modern scheduling requires more.

R traffic goes to either Pittsburgh or Toronto.

Goal:
New plan!

Interleave small, medium, and large packets.
New plan!

Interleave small, medium, and large packets.
No general way to deploy our gadget.
No general way to deploy our gadget.

A human needs a range of trees.
No general way to deploy our gadget.

A human needs a range of trees. The hardware wants to support one tree.
No general way to deploy our gadget.

A human needs a *range* of trees.

The hardware wants to support *one* tree.
No general way to deploy our gadget.

A human needs a *range* of trees.  

The hardware wants to support *one* tree.
No general way to deploy our gadget.

A human needs a range of trees.

The hardware wants to support one tree.

this work
No general way to deploy our gadget.

A human needs a range of trees.

The hardware wants to support one tree.

this work
Aside: PIFO Trees

Sivaraman et al. at SIGCOMM ’16
Review: FIFO

Just an ordered collection.
Review: FIFO

Just an ordered collection.

Two ways of interacting with the collection:
Review: FIFO

Just an ordered collection.

Two ways of interacting with the collection:

push
Review: FIFO

Just an ordered collection.

Two ways of interacting with the collection:

push

🍎
Review: FIFO

Just an ordered collection.

Two ways of interacting with the collection:

push
Review: FIFO

Just an ordered collection.

Two ways of interacting with the collection:

push          pop
Review: FIFO

Just an ordered collection.

Two ways of interacting with the collection:

push                  pop
Review: FIFO

Just an ordered collection.

Two ways of interacting with the collection:

push  pop
Review: FIFO

Just an ordered collection.

Two ways of interacting with the collection:

push  pop
Review: priority queue

Everything from before holds, but we have a little more control.
Review: priority queue

Everything from before holds, but we have a little more control.

Say we have a queue prioritized by pH.
Review: priority queue

Everything from before holds, but we have a little more control.

Say we have a queue prioritized by pH.
Review: priority queue

Everything from before holds, but we have a little more control.

Say we have a queue prioritized by pH.
Review: priority queue

Everything from before holds, but we have a little more control.

Say we have a queue *prioritized by pH*. 
Review: priority queue

Everything from before holds, but we have a little more control.

Say we have a queue prioritized by pH.
Review: priority queue

Everything from before holds, but we have a little more control.

Say we have a queue \textit{prioritized by pH}. 
Review: priority queue

The priority need not be inherent to the item!
Review: priority queue

The priority need not be inherent to the item!
We can have a ranking function:
Review: priority queue

The priority need not be inherent to the item!
We can have a ranking function:
Review: priority queue

The priority need not be inherent to the item!

We can have a *ranking function*:
Review: priority queue

The priority need not be inherent to the item!
We can have a *ranking function*: 

![Diagram of a priority queue with an apple and a lemon]
Review: priority queue

The priority need not be inherent to the item!
We can have a *ranking function*:
Review: priority queue

The priority need not be inherent to the item!

We can have a *ranking function*: 

![Diagram of priority queue with apples and a lemon]
Review: priority queue

The priority need not be inherent to the item!

We can have a ranking function:
Introducing: PIFO

Just a PQ, with a ranking function, but with rank-ties broken in FIFO order.
Baby goal: interleave R and B.
Baby goal: interleave R and B.

A PIFO will suffice.
Baby goal: interleaver R and B.

A PIFO will suffice.

$B_n, \ldots, B_1, (R,B)^*$
Baby goal: interleave $R$ and $B$.

A PIFO will suffice.

$R \rightarrow B_n, \ldots, B_1, (R,B)^*$
Baby goal: 
interleave $R$ and $B$.

A PIFO will suffice.

$B_n, \ldots, B_1, (R,B)^*$
Baby goal: interleave $R$ and $B$.

A PIFO will suffice.

$B_n, \ldots, B_1, (R,B)^*$
A PIFO will suffice.

Baby goal: interleave $R$ and $B$.

$$B_n, \ldots, B_1, (R,B)^*$$

$$R_n, \ldots, R_1, (R,B)^*$$
Baby goal: interleave $R$ and $B$.

A PIFO will suffice.

$B_n, \ldots, B_1, (R,B)^*$

$R_n, \ldots, R_1, (R,B)^*$

$(R,B)^*$
interleave R and B; interleave P and T.
Goal: interleave $R$ and $B$; interleave $P$ and $T$. 
Goal:
interleave R and B;
interleave P and T.
Goal:
interleave $R$ and $B$;
interleave $P$ and $T$.

$B_3, B_2, P_2, B_1, P_1$
Goal:
interleave $R$ and $B$;
interleave $P$ and $T$.

$T_1 \rightarrow B_3, B_2, P_2, B_1, P_1$
Goal:
interleave R and B;
interleave P and T.
Goal:
interleave $R$ and $B$;
interleave $P$ and $T$.

$T_1 \rightarrow B_3, B_2, P_2, B_1, P_1$

$B_3, B_2, P_2, T_1, B_1, P_1$

$B_3, B_2, P_2, B_1, T_1, P_1$
Goal:
interleave $R$ and $B$;
interleave $P$ and $T$.

$T_1 \rightarrow B_3, B_2, P_2, B_1, P_1$
Goal:
interleave $R$ and $B$; 
interleave $P$ and $T$. 

$T_1 \rightarrow B_3, B_2, P_2, B_1, P_1$ 

$B_3, B_2, P_2, T_1, B_1, P_1$ 

$B_3, B_2, P_2, B_1, T_1, P_1$
Goal:
interleave $R$ and $B$;
interleave $P$ and $T$. 

$T_1 \rightarrow B_3, B_2, P_2, B_1, P_1$
Goal:
interleave $R$ and $B$;
interleave $P$ and $T$.

$T_1 \rightarrow B_3, B_2, P_2, B_1, P_1$

$B_3, P_2, B_2, T_1, B_1, P_1$
Goal:
interleave $R$ and $B$;
interleave $P$ and $T$.
Goal:
interleave \( R \) and \( B \);
interleave \( P \) and \( T \).
Goal: interleave R and B; interleave P and T.
Enqueueing a packet can require the reordering of buffered packets.

No PIFO can do this.
Introducing: PIFO trees

Interleave R and B; interleave P and T.
Introducing: PIFO trees

Interleave R and B; interchange P and T.
Introducing: PIFO trees

Interleave $R$ and $B$; interleave $P$ and $T$. 

![Diagram of PIFO trees with nodes labeled $P_2, P_1, B_3, B_2, B_1$.]
Introducing: PIFO trees

2, 2, 1, 2, 1

1

2

P₂, P₁

B₃, B₂, B₁

This behaves like a queue!

Interleave R and B; interleave P and T.
Introducing: PIFO trees

This behaves like a queue!
How do we pop it?

Interleave R and B; 
interleave P and T.
Introducing: PIFO trees

Interleave R and B; interleave P and T.

This behaves like a queue!

How do we pop it?
Introducing: PIFO trees

```
2, 2, 1, 2
```

```
1
```

```
2
```

```
P₂, P₁
```

```
B₃, B₂, B₁
```

Interleave R and B; interleave P and T.

This behaves like a queue!

How do we pop it?
Introducing: PIFO trees

Interleave R and B; interleave P and T.

This behaves like a queue!
How do we pop it?

\[ 2, 2, 1, 2 \]

\[ 1 \quad 2 \]

\[ P_2, P_1 \]
\[ B_3, B_2, B_1 \]
Introducing: PIFO trees

This behaves like a queue!
How do we pop it?

Interleave R and B; interleave P and T.

\[ \begin{array}{c}
2, 2, 1, 2 \\
1 \quad 2 \\
P_2 \quad B_3, B_2, B_1 \\
P_1
\end{array} \]
Introducing: PIFO trees

Interleave R and B; interleave P and T.

This behaves like a queue!
How do we pop it?
Introducing: PIFO trees

Interleave R and B; interleave P and T.

This behaves like a queue!
How do we pop it?
Introducing: PIFO trees

This behaves like a queue!
How do we pop it?

Interleave R and B; interleave P and T.
Introducing: PIFO trees

This behaves like a queue!

How do we pop it?

Interleave R and B; 
interleave P and T.
Introducing: PIFO trees

This behaves like a queue!
How do we pop it?

Interleave R and B; interleave P and T.

B₃, B₂, P₂, B₁, P₁
Introducing: PIFO trees

Interleave R and B; interleave P and T.

This behaves like a queue!

How do we pop it?
Introducing: PIFO trees

This behaves like a queue!
How do we pop it?

Interleave R and B; interleave P and T.
Introducing: PIFO trees

Interleave R and B; interleave P and T.

This behaves like a queue!
How do we pop it?
How do we push into it?

2, 2, 1, 2, 1
1
2
P₂, P₁
B₃, B₂, B₁
B₃, B₂, P₂, B₁, P₁
Introducing: PIFO trees

Interleave R and B; interleave P and T.

push $T_1$
Introducing: PIFO trees

Interleave R and B; interleave P and T.

Push $T_1$
Introducing: PIFO trees

Interleave R and B; interleave P and T.

push $T_1$
Introducing: PIFO trees

Interleave R and B; interleave P and T.

push $T_1$

2, 2, 1, 2, 1

B_3, B_2, B_1

$B_3, B_2, \overline{P_2}, B_1, \overline{P_1}$
Introducing: PIFO trees

Interleave R and B; interleave P and T.

Push $T_1$
Introducing: PIFO trees

Interleave R and B; interchange P and T.

push $T_1$
Introducing: PIFO trees

Interleave $R$ and $B$; interleave $P$ and $T$. 

push $T_1$
Introducing: PIFO trees

Interleave R and B; interleave P and T.

push $T_1$
Introducing: PIFO trees

Interleave R and B; interleave P and T.

push $T_1$
Introducing: PIFO trees

Interleave $R$ and $B$; interleave $P$ and $T$.

push $T_1$
Goal: interleave R and B; interleave P and T.
Goal:
interleave R and B;
interleave P and T.
Aside: PIFO Trees

Sivaraman et al. at SIGCOMM ’16
Key Insight
Key Insight

A PIFO tree manifests a programming language.
Key Insight

A PIFO tree manifests a programming language.

A program is precisely a scheduling algorithm.

1

2

1

2

1

2
Key Insight

A PIFO tree manifests a *programming language*.

A program is precisely a *scheduling algorithm*.
Key Insight

A PIFO tree manifests a programming language.

A program is precisely a scheduling algorithm.

Path: $[\langle 2, r_1 \rangle, \langle B_1, r_2 \rangle]$
Key Insight

A PIFO tree manifests a **programming language**.

A program is precisely a **scheduling algorithm**.

Path: \([(2, r_1), (B_1, r_2)]\)
A PIFO tree manifests a programming language.

A program is precisely a scheduling algorithm.
Key Insight

A PIFO tree manifests a *programming language*.

A program is precisely a *scheduling algorithm*.

\[
\text{Path: } [(2, r_1), (B_1, r_2)]
\]

- tree shape
- language expressivity
Which leads to some very PL-ey questions:

tree ← shape → language
shape ← expressivity → language
Which leads to some very PL-ey questions:

Compare expressivity of languages?
Which leads to some very PL-ey questions:

Compare expressivity of languages?
Compare expressivity of trees?
Which leads to some very PL-ey questions:

- Compare expressivity of languages?
- Compare expressivity of trees?
- Compile a program so it runs against a new tree?
No general way to deploy our gadget.

A human needs a range of trees.

The hardware wants to support one tree.
A human needs a range of trees.

The hardware wants to support one tree.
A human needs a *range* of trees.

The hardware wants to support *one* tree.

No general way to deploy our gadget.
Contributions
Contributions

Formal model of PIFO trees
Contributions

Formal model of PIFO trees
General theorems of expressiveness w.r.t. tree shape
Contributions

Formal model of PIFO trees

General theorems of expressiveness w.r.t. tree shape

Compiler
Contributions

Formal model of PIFO trees

General theorems of expressiveness w.r.t. tree shape

Compiler

Simulator
Expressivity of trees

Trees with more leaves are more expressive. Taller trees are more expressive.
Expressivity of trees

Trees with more leaves are more expressive.
Taller trees are more expressive.

Captured elegantly by:

*Homomorphic embedding.*

Map root to root, leaves to leaves. Respect ancestry.
Expressivity of trees

*Homomorphic embedding.*
Map root to root, leaves to leaves. Respect ancestry.
Expressivity of trees

Homomorphic embedding.
Map root to root, leaves to leaves. Respect ancestry.
Expressivity of trees

Homomorphic embedding.
Map root to root, leaves to leaves. Respect ancestry.
Compiling programs

Diagram:

- Left diagram:
  - Node 1
  - Node 2
  - Node 3

- Right diagram:
  - Node 1
  - Node 2
  - Node 3

Arrows indicate the direction of compilation flow.
Compiling programs
Compiling programs
Compiling programs
Compiling programs
Compiling programs

\[
\begin{align*}
&= 1 \quad 2 \quad 3 \\
1 &\rightarrow \ldots \\
2 &\xrightarrow{\text{x}} \text{x} \\
3 &\rightarrow \ldots
\end{align*}
\]

\[
\begin{align*}
&= 1 \quad 2 \\
\frac{1}{3} &\rightarrow \ldots \\
\frac{2}{3} &\rightarrow \ldots
\end{align*}
\]
Compiling programs
Compiling programs
Compiling programs
Compiling programs
Compiling programs
Compiling programs

1, 3, 1, 2, 3, 1, 2, 3

1, 2, 1, 2, 2, 1, 2, 2
Compiling programs

1, 3, 1, 2, 3

1, 2, 1, 2, 2, 1, 2, 2
Compiling programs

1, 3, 1, 2, 3, 1, 2, 3

1, 2, 1, 2, 2, 1, 2, 2

1, 2, 1, 2, 2, 1, 2, 2
Compiling programs

1, 3, 1, 2, 3, 1, 2, 3

1, 2, 1, 2, 2, 1, 2, 2
Compiling programs

Path: \([2, r_1), \ldots\]
Compiling programs

Path: [(2, r_1), ...]

1, 2, 3, 1, 2, 3, 1, 2, 3

1, 2, 1, 2, 2, 1, 2, 2
Compiling programs

Path: [\((2, r_1), \ldots\)]
Compiling programs

Path: \([ (2, r_1), \ldots ] \)
Compiling programs

Path: [(2, $r_1$), ...]

Path: [(2, ?), (1, ?), ...]

1, 2, 3, 1, 2, 3, 1, 2, 3

1, 2, 1, 2, 2, 1, 2, 2
Compiling programs

Path: \([(2,r_1), \ldots]\)
Compiling programs

Path: [(2, r₁), …]
Compiling programs

Path: [(2, r₁), …]
Compiling programs

Path: [(2, r_1), ...]

Path: [(2, r_1), (1, r_1), ...]
Given an embedding, we lift it to arrive at a compiler.

Path: [(2, r₁), ...]
Generating embeddings automatically!
Generating embeddings automatically!

*Homomorphic embedding.*
Map root to root, leaves to leaves. Respect ancestry.
Generating embeddings automatically!

*Homomorphic embedding.*
Map root to root, leaves to leaves. Respect ancestry.

Two new algorithms,
both starting with heterogeneous source trees.
Generating embeddings automatically!

*Homomorphic embedding.* Map root to root, leaves to leaves. Respect ancestry.

Two new algorithms, both starting with heterogeneous source trees.

1. If target tree is regular $d$-ary for some $d$. 
Generating embeddings automatically!

*Homomorphic embedding.*
Map root to root, leaves to leaves. Respect ancestry.

Two new algorithms, both starting with heterogeneous source trees.

1. If target tree is regular $d$-ary for some $d$.
2. If target tree is itself heterogeneous.
Workflow

WFQ: 40/40/20

A → B → RR → C → D → WFQ: 10/40/50 → E → F → G
Workflow

WFQ: 40/40/20

A → B → RR → C → D

WFQ: 10/40/50

E → F → G

logical
But the hardware supports a regular-branching binary tree.
But the hardware supports a regular-branching binary tree.

Here’s how I’ll use that tree.
Here’s how I’ll use that tree.
Workflow

logical

actual
Simulation

WFQ: 40/40/20

logical

A

B

RR

C

D

WFQ: 10/40/50

E

F

G

actual

WFQ: 40/40/20

T

A

B

RR

T

C

D

WFQ: 10/40/50

T

G

E

F
Simulation

logical

actual
Simulation

WFQ: 40/40/20

A - B - RR

C - D - WFQ: 10/40/50

E - F - G

WFQ: 40/40/20

T - RR

A - B - T - WFQ: 10/40/50

C - D - T - G

E - F
Underlying formalism

\[
\begin{align*}
\text{Leaf}(p) & \in \text{PIFOTree}(\ast) \\
\text{Internal}(qs, p) & \in \text{PIFOTree}(\text{Node}(ts)) \\
\text{push}(\text{Leaf}(p), pkt, r) & = \text{Leaf}(p') \\
\text{push}(\text{Internal}(qs, p), pkt, (i, r) :: pt) & = \text{Internal}(qs[i/q'], p')
\end{align*}
\]
A general way to deploy PIFO trees.
A general way to deploy PIFO trees.

Let the hardware support some tree.
A general way to deploy PIFO trees.

Let the human program against some tree.

Let the hardware support some tree.
A general way to deploy PIFO trees.

Let the human program against some tree.

Let the hardware support some tree.
A general way to deploy PIFO trees.

Let the human program against some tree.

Compilable?

Let the hardware support some tree.
A general way to deploy PIFO trees.

Let the human program against some tree.

Let the hardware support some tree.
A general way to deploy PIFO trees.

Let the human program against some tree.

Let the hardware support some tree.

Compilable?
A general way to deploy PIFO trees.

Let the human program against some tree.

Let the hardware support some tree.
A general way to deploy PIFO trees.

Let the human program against some tree.

Compilable?

Let the hardware support some tree.
A general way to deploy PIFO trees.

Let the hardware support some tree.

Compilable?

Let the human program against some tree.

Let the hardware support some tree.
A general way to deploy PIFO trees.

Let the human program against some tree.

Let the hardware support some tree.
A general way to deploy PIFO trees.

Let the human program against some tree.

Let the hardware support some tree.
Let the human program against some tree.

Let the hardware support some tree.

A general way to deploy PIFO trees.
A general way to deploy PIFO trees.

Let the human program against some tree.

Compilable?

Let the hardware support some tree.
Formal Abstractions for Packet Scheduling

Mohan, Liu, Foster, Kappé, Kozen

cs.cornell.edu/~amohan