Software in the Datacenter
Software in the Datacenter

Response Time: Productivity, Conversion, Retention

microservices/containers → bigger problem
Software in the Datacenter

Response Time: Productivity, Conversion, Retention

microservices/containers → bigger problem
Software in the Datacenter

Response Time: Productivity, Conversion, Retention

microservices/containers $\rightarrow$ bigger problem

Allocate network resources
  - Explicitly (maximize utility)
  - Quickly, Consistently
  - Flexibly (in software)
Traditional approach is packet-centric
Traditional approach is packet-centric

Switch Algorithms
Traditional approach is packet-centric

Switch Algorithms

Server Algorithms
Traditional approach is packet-centric

Switch Algorithms

Server Algorithms

Implicit Allocation

Several RTT to converge

Changes many components
Flowtune’s approach

1. *Flowlet control*

   Allocation changes *only* when:
   - Flowlets arrive
   - Flowlets terminate
Flowtune’s approach

1. *Flowlet control*

   Allocation changes *only* when:
   - Flowlets arrive
   - Flowlets terminate
Flowtune’s approach

1. *Flowlet control*

   Allocation changes *only* when:
   - Flowlets arrive
   - Flowlets terminate
Flowtune’s approach

1. *Flowlet control*

   Allocation changes *only* when:
   - Flowlets arrive
   - Flowlets terminate
Flowtune’s approach

1. *Flowlet control*

Allocation changes *only* when:
- Flowlets arrive
- Flowlets terminate
Flowtune’s approach

1. *Flowlet control*

   Allocation changes *only* when:
   - Flowlets arrive
   - Flowlets terminate

2. Logically centralized
   - Reduce RTT dependence
Example

Hadoop on Server A has data for B:
Example

Hadoop on Server A has data for B:

A → Allocator

“Hadoop on A has data for B”
Example

Hadoop on Server A has data for B:

A → Allocator

“Hadoop on A has data for B”

Allocator

Assign rates

R1

R2

Allocators

B C A
Example

Hadoop on Server A has data for B:

A \rightarrow \text{Allocator}

\text{Allocator} \rightarrow A

“Hadoop on A has data for B”

Assign rates

“Send at 40Gbps”
Example

Now say ad_server on Server C has data for B:
Example

Now say ad_server on Server C has data for B:
Example

Now say ad_server on Server C has data for B:

C → Allocator  "ad_server on C has data for B"
Example

Now say ad_server on Server C has data for B:

C → Allocator

"ad_server on C has data for B"

Allocator

Assign rates
Example

Now say ad_server on Server C has data for B:

C → Allocator

"ad_server on C has data for B"

Allocator → A

"Send at 5Gbps"

Allocator → C

"Send at 35Gbps"
Why is this hard?

Need to choose rates given active flowlets

1. Updates cascade!
Why is this hard?

Need to choose rates given active flowlets

1. Updates cascade!
2. What is the goal? To act like TCP?
Why is this hard?

Need to choose rates given active flowlets

1. Updates cascade!
2. What is the goal? To act like TCP?
Why is this hard?

Need to choose rates given active flowlets

1. Updates cascade!
2. What is the goal? To act like TCP?
Why is this hard?

Need to choose rates given active flowlets

1. Updates cascade!
2. What is the goal? To act like TCP?
Why is this hard?

Need to choose rates given active flowlets

1. Updates cascade!
2. What is the goal? To act like TCP?
Performance Policy

Hadoop flow rate 2x \Rightarrow$0.05
Ad_server flow rate 2x \Rightarrow$0.20
NUM Iterative Optimizer

1. Each link $\ell$ chooses price $p_{\ell}$ using

2. Each flow $s$ chooses rate $x_s$

3. Goto 1
NUM Iterative Optimizer

1. Each link $\ell$ chooses price $p_\ell$ using $\sum_{s \in S(\ell)} x_s - c_\ell$

2. Each flow $s$ chooses rate $x_s$

3. Goto 1
NUM Iterative Optimizer

1. Each link $\ell$ chooses price $p_\ell$ using

$$\sum_{s \in S(\ell)} x_s - c_\ell$$

Supply

2. Each flow $s$ chooses rate $x_s$

3. Goto 1

_Kelly et al., Journal of the Operational Research Society, 1998_
NUM Iterative Optimizer

1. Each link $\ell$ chooses price $p_\ell$ using

\[
\sum_{s \in S(\ell)} x_s - c_\ell
\]

2. Each flow $s$ chooses rate $x_s$

3. Goto 1
NUM Iterative Optimizer

1. Each link $\ell$ chooses price $p_\ell$ using
   \[ \sum_{s \in S(\ell)} x_s - c_\ell \]

2. Each flow $s$ chooses rate $x_s$

3. Goto 1

*Kelly et al., Journal of the Operational Research Society, 1998*
NUM Iterative Optimizer

1. Each link $\ell$ chooses price $p_\ell$ using
   \[ \sum_{s \in S(\ell)} x_s - c_\ell \]
   Demand \quad Supply

2. Each flow $s$ chooses rate $x_s$

3. Goto 1

Kelly et al., Journal of the Operational Research Society, 1998
NUM Iterative Optimizer

1. Each link $\ell$ chooses price $p_\ell$ using
   \[ \sum_{s \in S(\ell)} x_s - c_\ell \]

2. Each flow $s$ chooses rate $x_s$

3. Goto 1

---

Kelly et al., Journal of the Operational Research Society, 1998
NUM Iterative Optimizer

1. Each link $\ell$ chooses price $p_\ell$ using
\[ \sum_{s \in S(\ell)} x_s - c_\ell \]

2. Each flow $s$ chooses rate $x_s$

3. Goto 1

Kelly et al., Journal of the Operational Research Society, 1998
NUM Iterative Optimizer

1. Each link $\ell$ chooses price $p_\ell$ using
   \[ \sum_{s \in S(\ell)} x_s - c_\ell \]

2. Each flow $s$ chooses rate $x_s$

3. Goto 1

---

Kelly et al., Journal of the Operational Research Society, 1998
Going Multicore

1. Each link \( \ell \) chooses price \( p_\ell \) using 
\[
\sum_{s \in S(\ell)} x_s - c_\ell
\]

2. Each flow \( s \) chooses rate \( x_s \) using 
\[
\sum_{\ell \in L(s)} p_\ell
\]
Going Multicore

1. Each link $\ell$ chooses price $p_\ell$ using $\sum_{s \in S(\ell)} x_s - c_\ell$

2. Each flow $s$ chooses rate $x_s$ using $\sum_{\ell \in L(s)} p_\ell$

Core 1: $x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6$
Core 2: $x_7 \ x_8 \ x_9 \ x_{10} \ x_{11} \ x_{12}$
Core 3: $x_{13} \ x_{14} \ x_{15} \ x_{16} \ x_{17} \ x_{18}$
Core 4: $x_{19} \ x_{20} \ x_{21} \ x_{22} \ x_{23} \ x_{24}$
Going Multicore

1. Each link $\ell$ chooses price $p_\ell$ using
   \[ \sum_{s \in S(\ell)} x_s - c_\ell \]

2. Each flow $s$ chooses rate $x_s$ using
   \[ \sum_{\ell \in L(s)} p_\ell \]

Core 1: $x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6$

Core 2: $x_7 \quad x_8 \quad x_9 \quad \boxed{x_{10}} \quad x_{11} \quad x_{12}$

Core 3: $x_{13} \quad x_{14} \quad x_{15} \quad x_{16} \quad x_{17} \quad \boxed{x_{18}}$

Core 4: $x_{19} \quad \boxed{x_{20}} \quad x_{21} \quad x_{22} \quad x_{23} \quad x_{24}$
Going Multicore

1. Each link \( \ell \) chooses price \( p_\ell \) using
\[
\sum_{s \in S(\ell)} x_s - c_\ell
\]

2. Each flow \( s \) chooses rate \( x_s \) using
\[
\sum_{\ell \in L(s)} p_\ell
\]

Core 1: \( x_1 \quad x_2 \quad x_3 \quad x_4 \quad \boxed{x_5} \quad x_6 \)

Core 2: \( x_7 \quad x_8 \quad \boxed{x_9} \quad x_{10} \quad x_{11} \quad x_{12} \)

Core 3: \( \boxed{x_{13}} \quad x_{14} \quad x_{15} \quad \boxed{x_{16}} \quad x_{17} \quad x_{18} \)

Core 4: \( x_{19} \quad x_{20} \quad x_{21} \quad x_{22} \quad x_{23} \quad x_{24} \)
Going Multicore

1. Each link $\ell$ chooses price $p_\ell$ using $\sum_{s \in S(\ell)} x_s - c_\ell$

2. Each flow $s$ chooses rate $x_s$ using $\sum_{\ell \in L(s)} p_\ell$

Core 1: $x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6$
Core 2: $x_7 \ x_8 \ x_9 \ x_{10} \ x_{11} \ x_{12}$
Core 3: $x_{13} \ x_{14} \ x_{15} \ x_{16} \ x_{17} \ x_{18}$
Core 4: $x_{19} \ x_{20} \ x_{21} \ x_{22} \ x_{23} \ x_{24}$
Challenge

How to efficiently switch views: Flow ↔ Link
Challenge

How to efficiently switch views: Flow ↔ Link
Challenge

How to efficiently switch views: Flow ↔ Link
Challenge

How to efficiently switch views: Flow ↔ Link
Challenge

How to efficiently switch views: Flow ⇔ Link
Multicore

<table>
<thead>
<tr>
<th>src</th>
<th>lb</th>
<th>lb</th>
<th>lb</th>
<th>lb</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fb</td>
<td>fb</td>
<td>fb</td>
<td>fb</td>
</tr>
<tr>
<td></td>
<td>fb</td>
<td>fb</td>
<td>fb</td>
<td>fb</td>
</tr>
<tr>
<td></td>
<td>fb</td>
<td>fb</td>
<td>fb</td>
<td>fb</td>
</tr>
<tr>
<td></td>
<td>fb</td>
<td>fb</td>
<td>fb</td>
<td>fb</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dst</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Multicore

Block 1  Block 2  Block 3  Block 4
Multicore
Multicore
Multicore
Multicore
Multicore
Aggregating LinkBlocks
Aggregating LinkBlocks
Aggregating LinkBlocks
Aggregating LinkBlocks
Aggregating LinkBlocks
Aggregating LinkBlocks
Aggregating LinkBlocks
768 servers in < 9µs

<table>
<thead>
<tr>
<th>Cores</th>
<th>Nodes</th>
<th>Flows</th>
<th>Cycles</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>384</td>
<td>3072</td>
<td>19896.6</td>
<td>8.29 µs</td>
</tr>
<tr>
<td>16</td>
<td>768</td>
<td>6144</td>
<td>21267.8</td>
<td>8.86 µs</td>
</tr>
<tr>
<td>64</td>
<td>1536</td>
<td>12288</td>
<td>30317.6</td>
<td>12.63 µs</td>
</tr>
<tr>
<td>64</td>
<td>1536</td>
<td>24576</td>
<td>33576.2</td>
<td>13.99 µs</td>
</tr>
<tr>
<td>64</td>
<td>1536</td>
<td>49152</td>
<td>40628.5</td>
<td>16.93 µs</td>
</tr>
<tr>
<td>64</td>
<td>3072</td>
<td>49152</td>
<td>57035.9</td>
<td>23.76 µs</td>
</tr>
<tr>
<td>64</td>
<td>4608</td>
<td>49152</td>
<td>73703.2</td>
<td>30.71 µs</td>
</tr>
</tbody>
</table>
768 servers in $< 9\mu s$

<table>
<thead>
<tr>
<th>Cores</th>
<th>Nodes</th>
<th>Flows</th>
<th>Cycles</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>384</td>
<td>3072</td>
<td>19896.6</td>
<td>8.29 $\mu s$</td>
</tr>
<tr>
<td>16</td>
<td>768</td>
<td>6144</td>
<td>21267.8</td>
<td>8.86 $\mu s$</td>
</tr>
<tr>
<td>64</td>
<td>1536</td>
<td>12288</td>
<td>30317.6</td>
<td>12.63 $\mu s$</td>
</tr>
<tr>
<td>64</td>
<td>1536</td>
<td>24576</td>
<td>33576.2</td>
<td>13.99 $\mu s$</td>
</tr>
<tr>
<td>64</td>
<td>1536</td>
<td>49152</td>
<td>40628.5</td>
<td>16.93 $\mu s$</td>
</tr>
<tr>
<td>64</td>
<td>3072</td>
<td>49152</td>
<td>57035.9</td>
<td>23.76 $\mu s$</td>
</tr>
<tr>
<td>64</td>
<td>4608</td>
<td>49152</td>
<td>73703.2</td>
<td>30.71 $\mu s$</td>
</tr>
</tbody>
</table>
Reducing iteration latency
Reducing iteration latency

But: too slow!
Reducing iteration latency

Straw-man:

But: too slow!

Solution 2:
Reducing iteration latency

Straw-man:

Solution 2:

Solution 3:

But: too slow!

But: over-allocates links!
Reducing iteration latency

Straw-man:

Update inputs $\rightarrow$ Run 100 iterations $\rightarrow$ Output rates

But: too slow!

Solution 2:

Update inputs $\rightarrow$ Run 1 iteration $\rightarrow$ Output rates

But: over-allocates links!

Solution 3:

Update inputs $\rightarrow$ Run 1 iteration $\rightarrow$ Normalize rates $\rightarrow$ Output rates
Flowtune normalizes rates

• For each flow:
  • Find link $\ell$ on path with largest $r_\ell = \frac{\sum \text{flow rates}}{\text{link capacity}}$
  • Normalize: $x_s \leftarrow \frac{x_s}{r_\ell}$
Flowtune normalizes rates

- For each flow:
  - Find link $\ell$ on path with largest $r_\ell = \frac{\sum \text{flow rates}}{\text{link capacity}}$
  - Normalize: $x_s \leftarrow \frac{x_s}{r_\ell}$
Architecture

Optimizer

Normalizer

Allocator

Endpoints

flowlet start/end
normalized rates
Flowtune converges quickly to a fair allocation

Every 10 milliseconds:

![Diagram showing network traffic over time for different protocols: DCTCP, Flowtune, pFabric, sfqCoDel, XCP. The graph plots throughput (Gbps) against time (ms).]
Setup

Inside the Social Network’s (Datacenter) Network, Roy et al., SIGCOMM’15

4 spines
40 Gbits/s
9 racks
10 Gbits/s
16 servers/rack

pFabric, Alizade et al., SIGCOMM’13
Flowtune reduces p99 FCT
Centralization overhead is low
Overhead is constant with scale
Flowtune achieves low drop rate
Amazon EC2: Resource Allocation
EC2: Response Time is Reduced
Flowtune

Allocate network resources
- Explicitly (maximize utility)
- Quickly, Consistently (centralized)
- Flexibly (in software)

Give application developers control over network transport