Flowtune

Flowlet Control for Datacenter Networks

Jonathan Perry, Hari Balakrishnan and Devavrat Shah

Software in the Datacenter

Software in the Datacenter

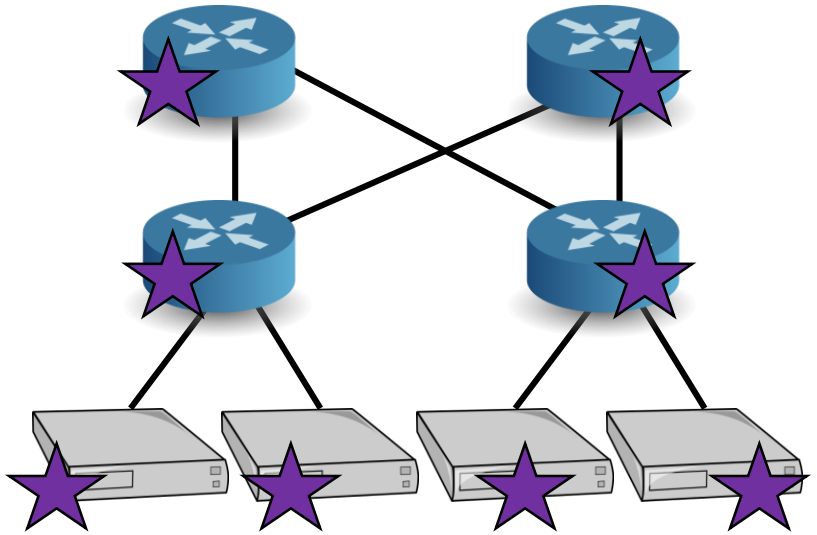
- Response Time: Productivity, Revenue, Reputation

- microservices → develop → network is central
  deploy
  scale

Traditional approach is packet-centric

Switch Mechanisms

Server Mechanisms



Implicit Allocation

Several RTT to converge

Changes many components

Implicit Allocation

Several RTT to converge

Changes many components

Allocate network resources

- Explicitly (maximize utility)
- Quickly, Consitently
- Flexibly (in software)

Flowtune's approach

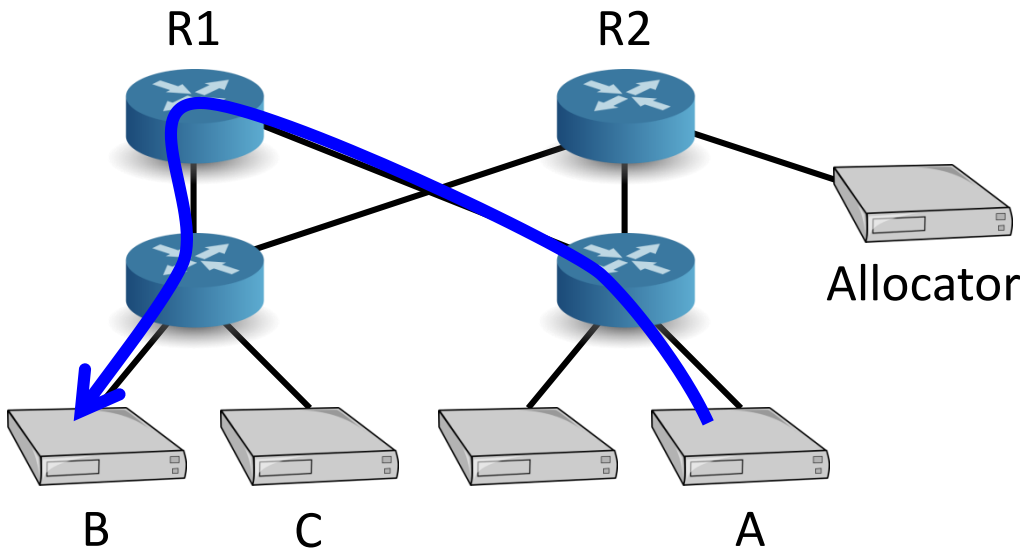
1. Flowlet control

   `send()` $\leftrightarrow$ flowlet

2. Logically centralized
   - Reduce RTT dependence

Example

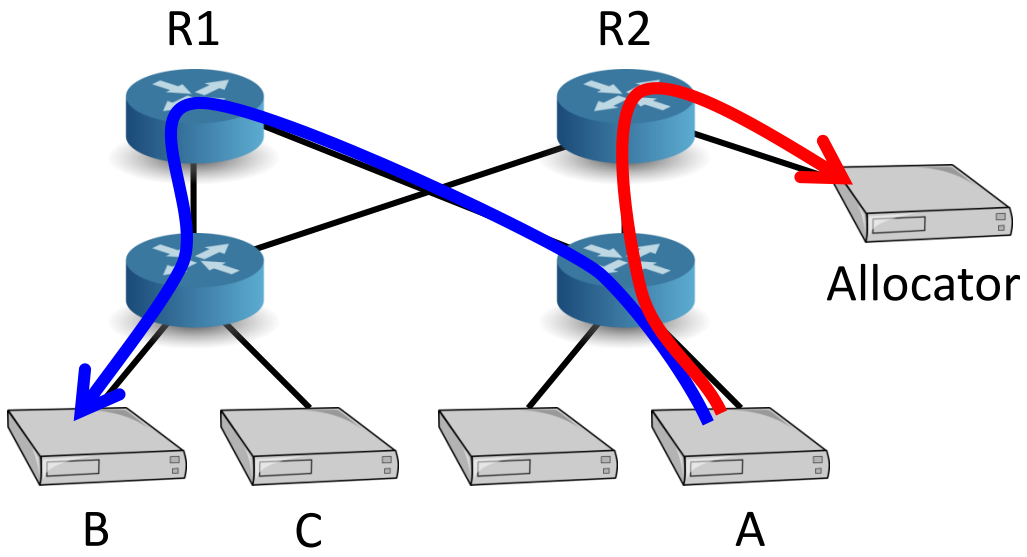
Hadoop on Server A has data for B:



Example

Hadoop on Server A has data for B:

A → Allocator "Hadoop on A has data for B"

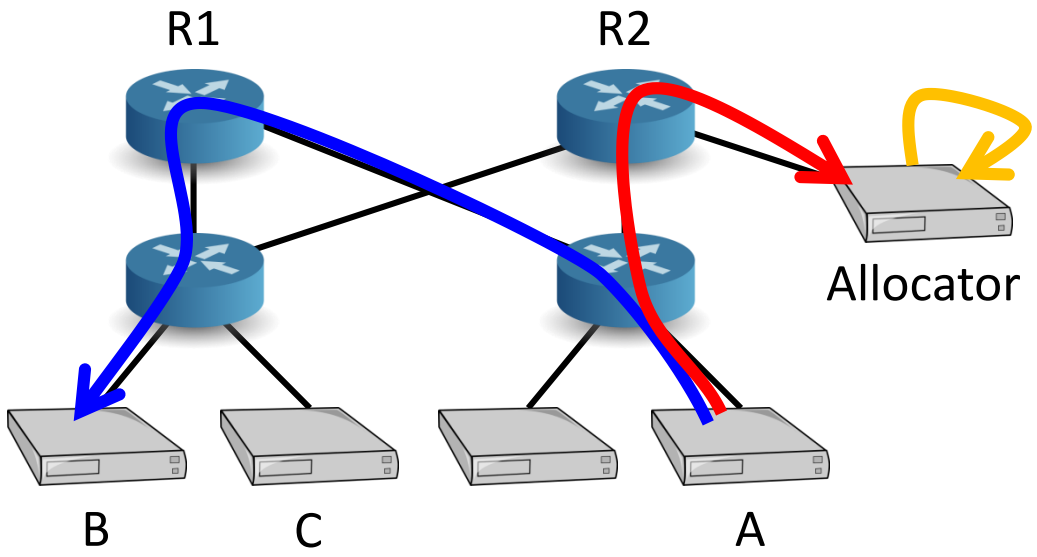


Example

Hadoop on Server A has data for B:

A → Allocator "Hadoop on A has data for B"

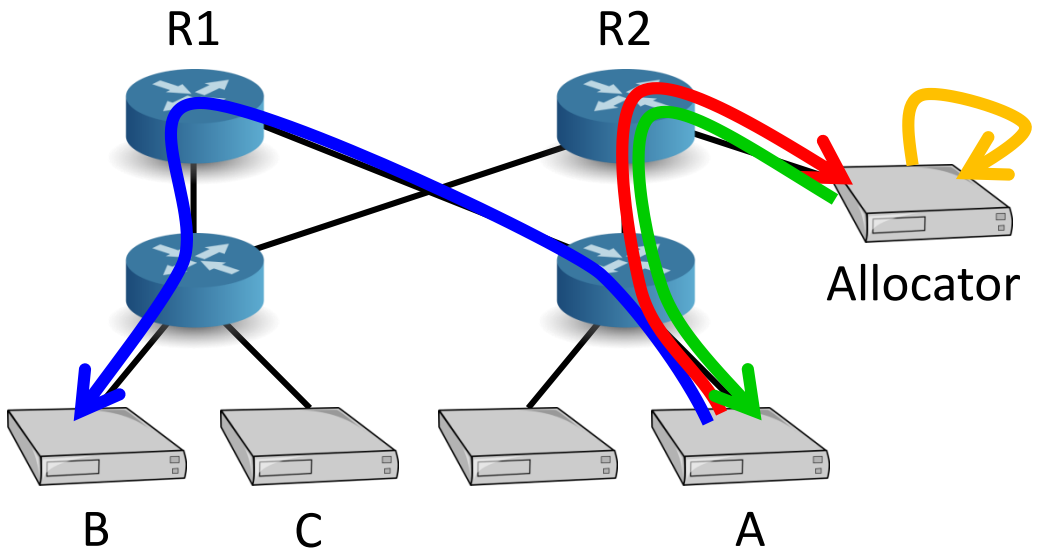
Allocator Assign rates



Example

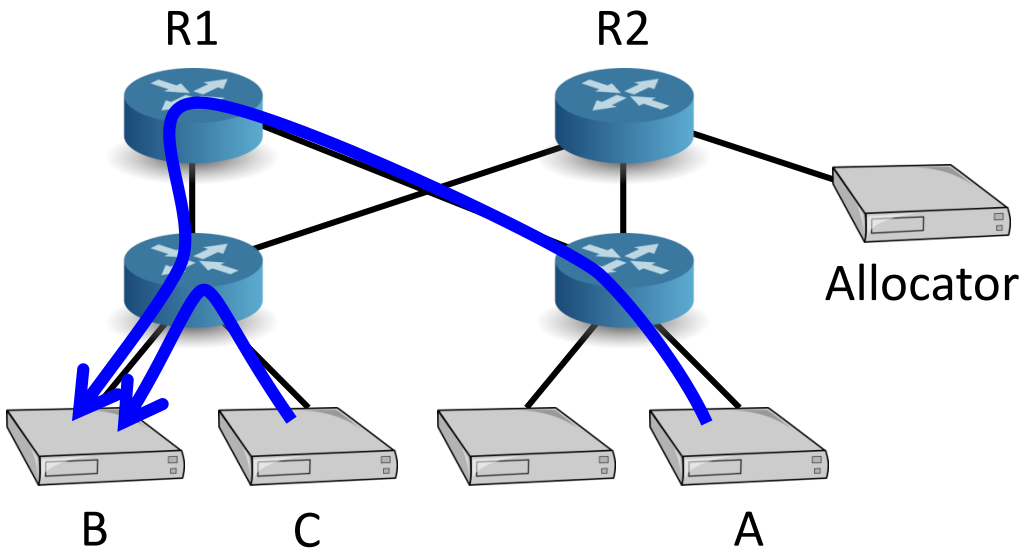
Hadoop on Server A has data for B:

A → Allocator	“Hadoop on A has data for B”
Allocator	Assign rates
Allocator → A	“Send at 40Gbps”



Example

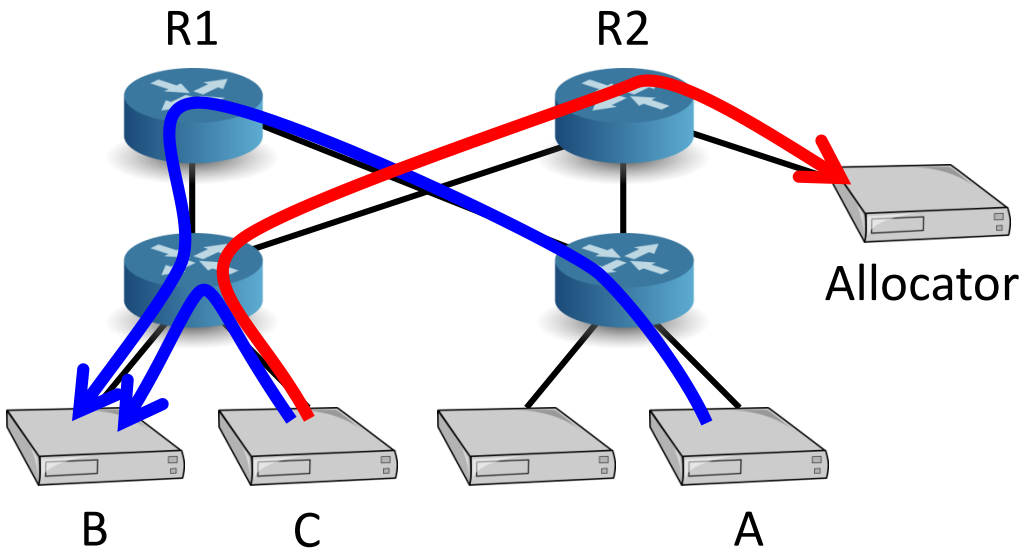
Now say ad_server on Server C has data for B:



Example

Now say ad_server on Server C has data for B:

C → Allocator “ad_server on C has data for B”

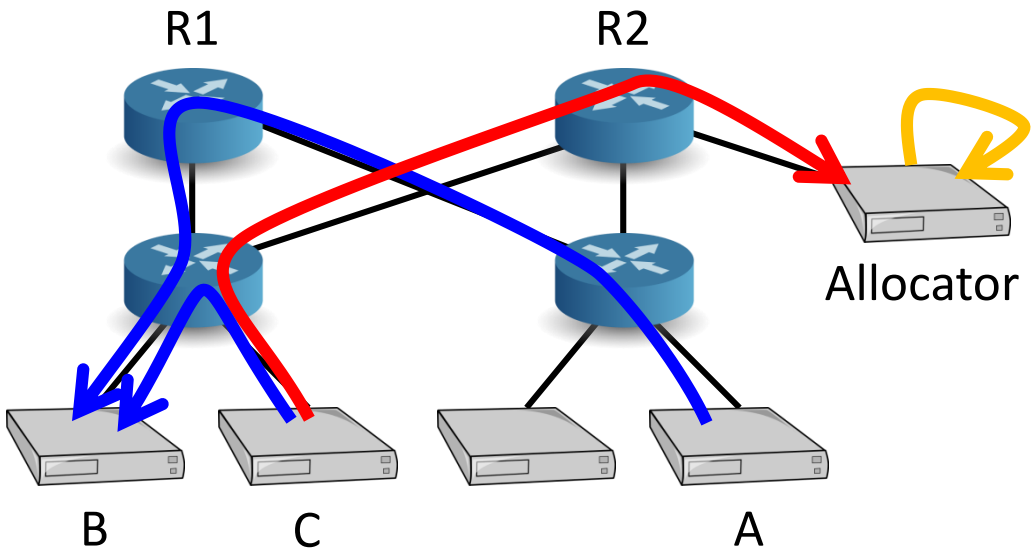


Example

Now say ad_server on Server C has data for B:

C → Allocator “ad_server on C has data for B”

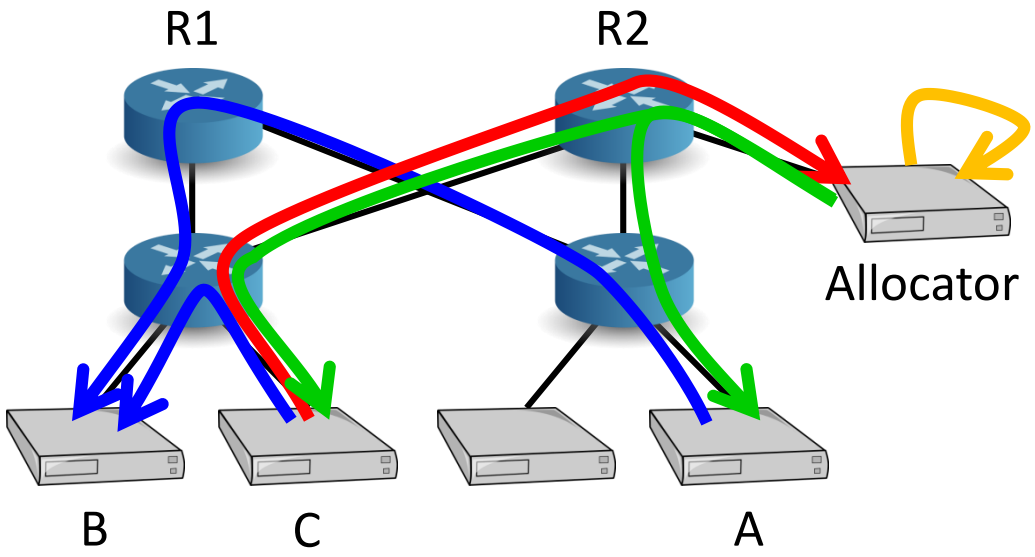
Allocator Assign rates



Example

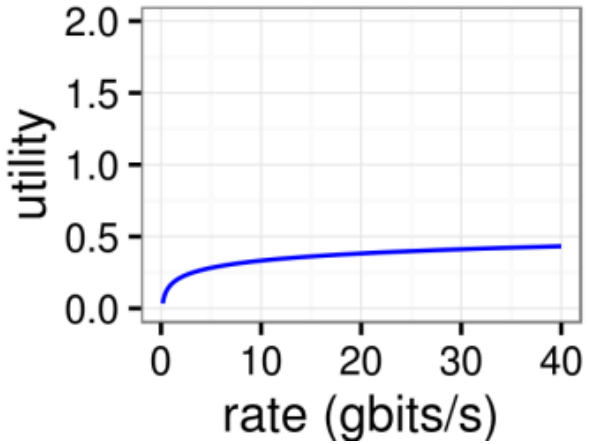
Now say ad_server on Server C has data for B:

C → Allocator	“ad_server on C has data for B”
Allocator	Assign rates
Allocator → A	“Send at 5Gbps”
Allocator → C	“Send at 35Gbps”

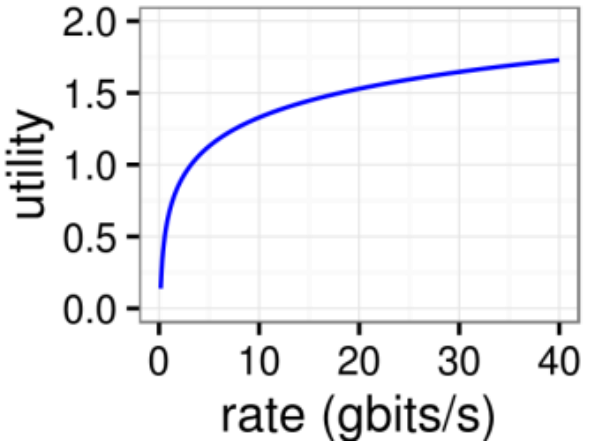


Network Utility Maximization (NUM)

Hadoop flowlet:



Ads flowlet:



Hadoop flowlet rate 2x → $0.05

Ads flowlet rate 2x → $0.20

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

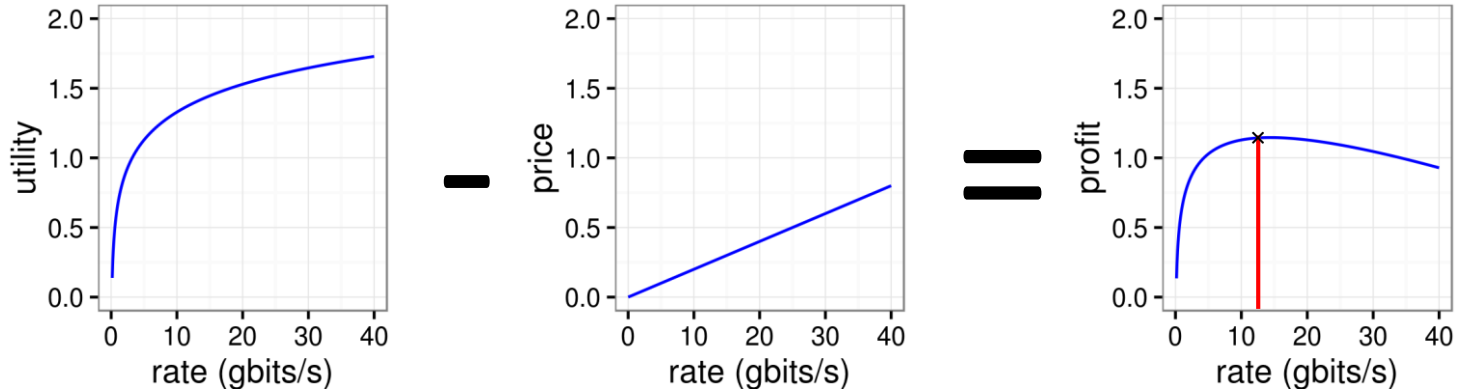
NUM Iterative Optimizer

1. Each link ℓ chooses price p_ℓ using $\sum$ flow rates on ℓ $-$ link capacity

Demand

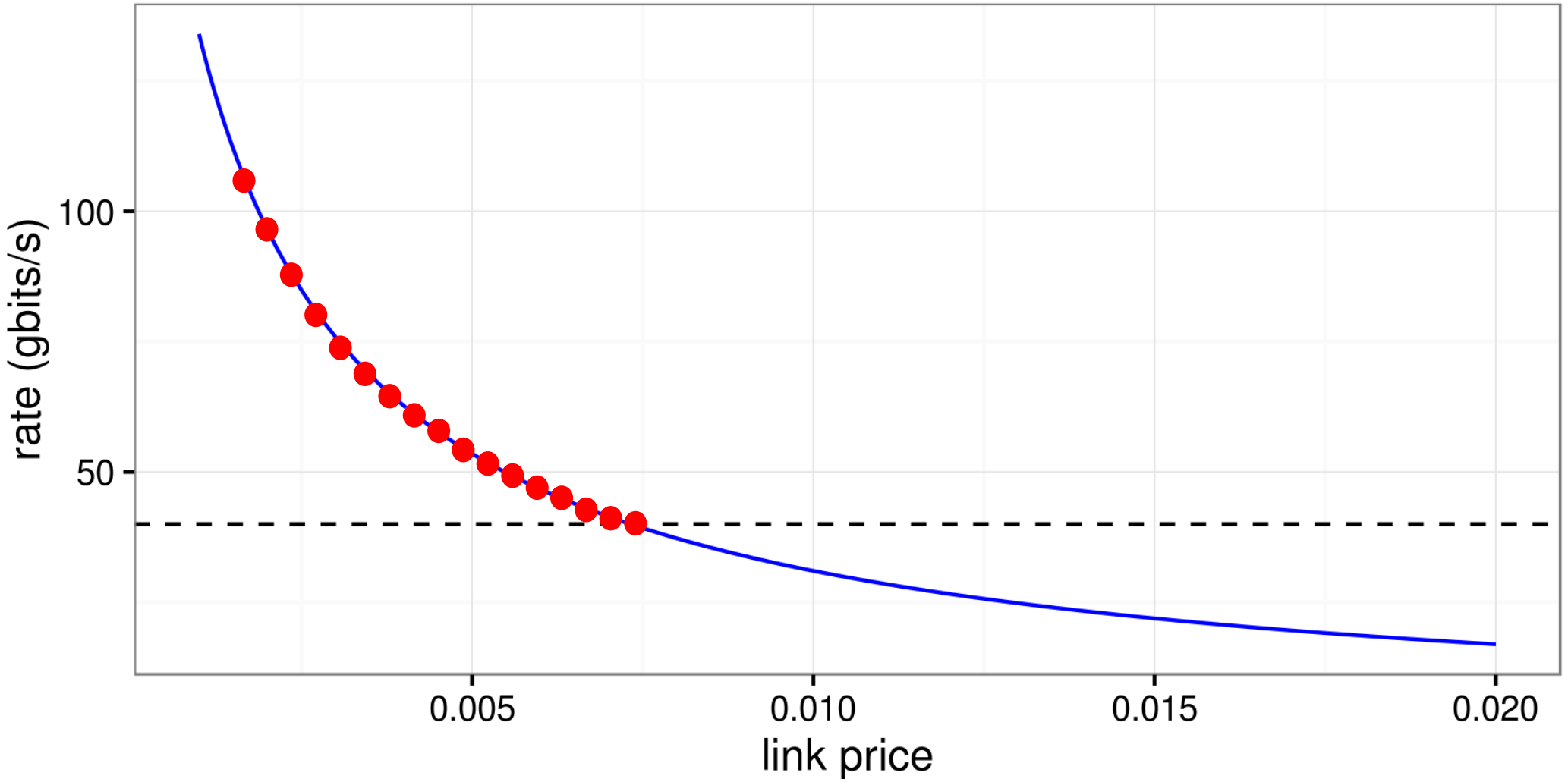
Supply

2. Each flow s chooses rate x_s

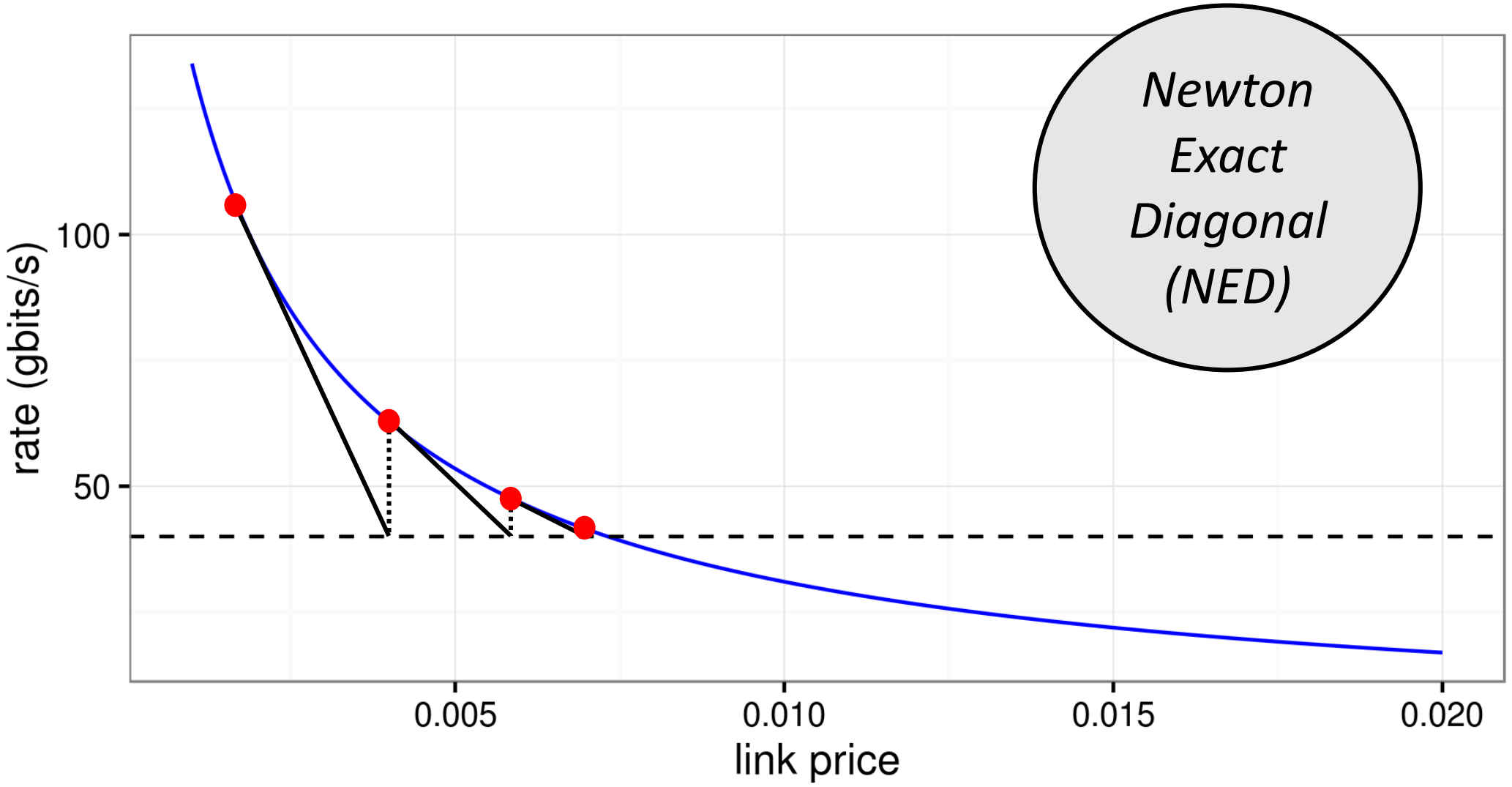


3. Goto 1

Adjusting prices



Adjusting prices



Increasing responsiveness

Solution 1:

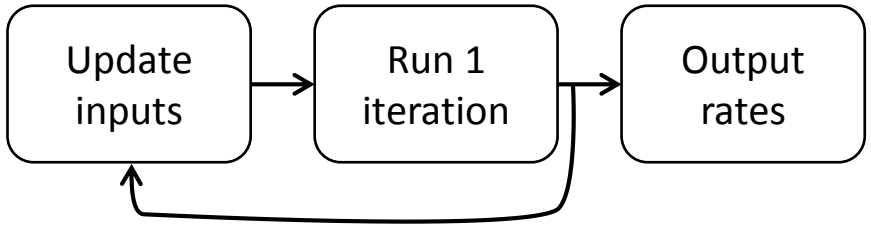
But: too slow!

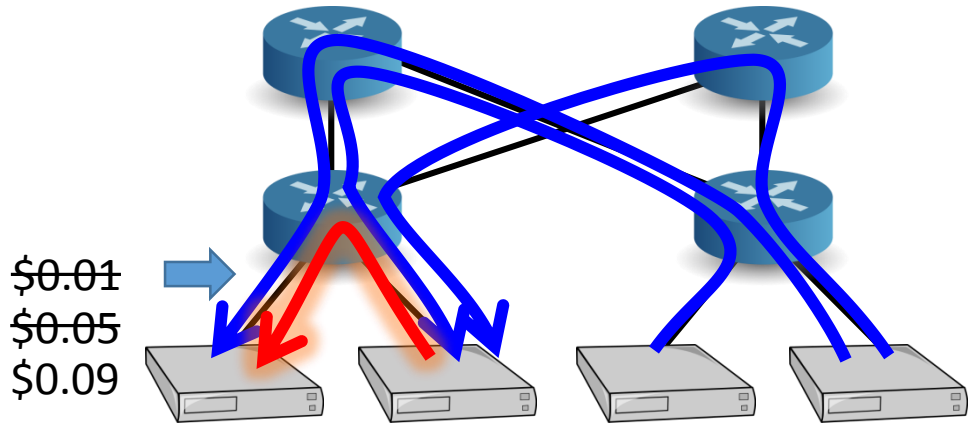
Increasing responsiveness

Solution 1:

But: too slow!

Solution 2:



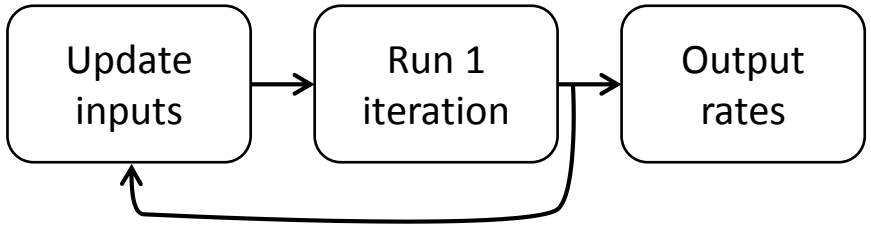


Increasing responsiveness

Solution 1:

But: too slow!

Solution 2:



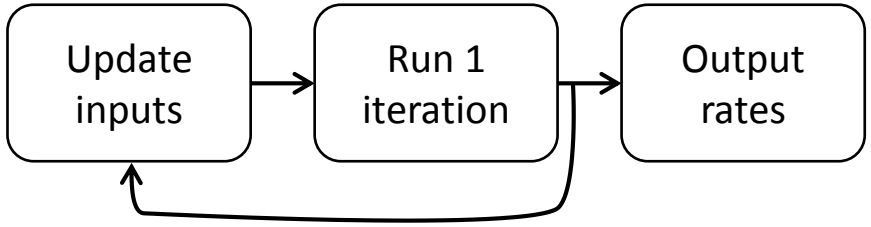
But: queueing, packet drops!

Increasing responsiveness

Solution 1:

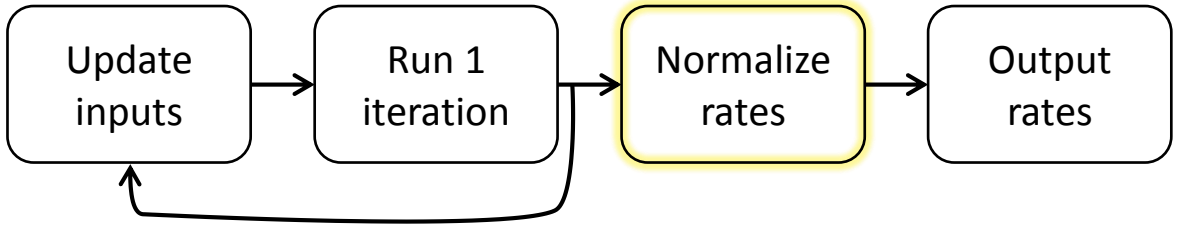
But: too slow!

Solution 2:

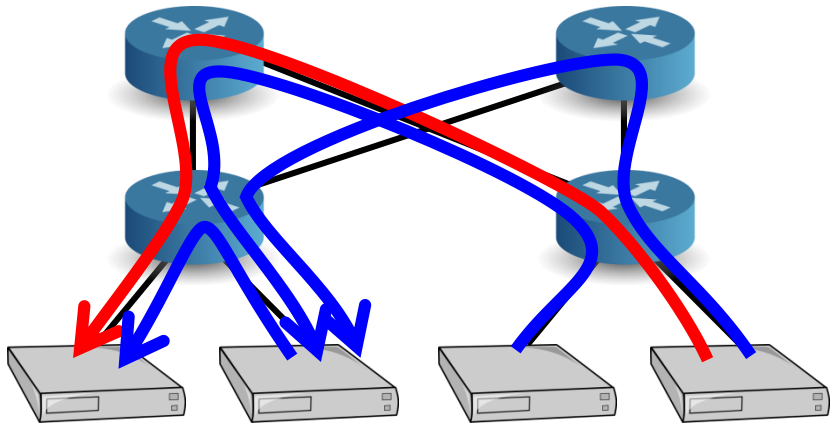


But: queueing, packet drops!

Solution 3:

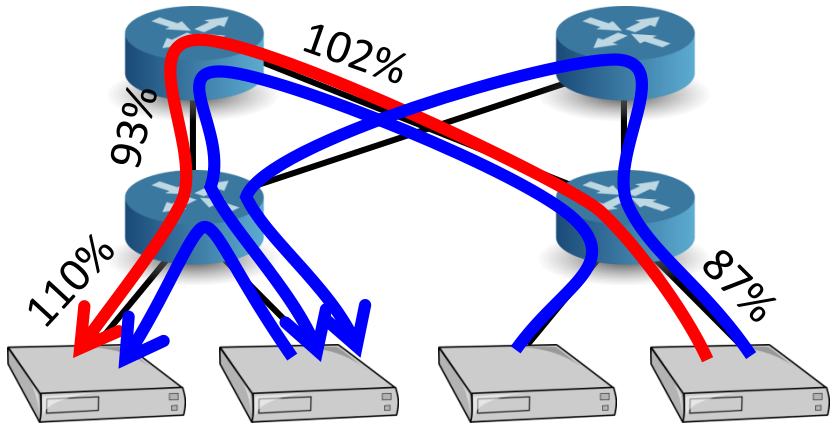


Flowtune normalizes rates



Flowtune normalizes rates

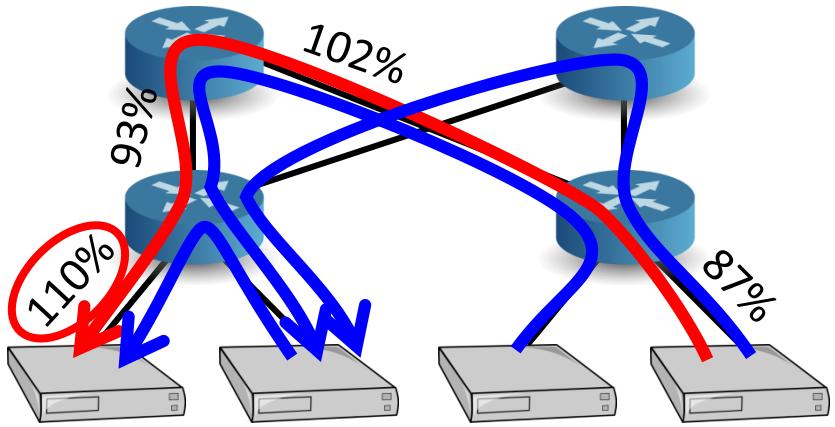
$$r_\ell = \frac{\text{allocation}}{\text{capacity}}$$



Flowtune normalizes rates

$$r_\ell = \frac{\text{allocation}}{\text{capacity}}$$

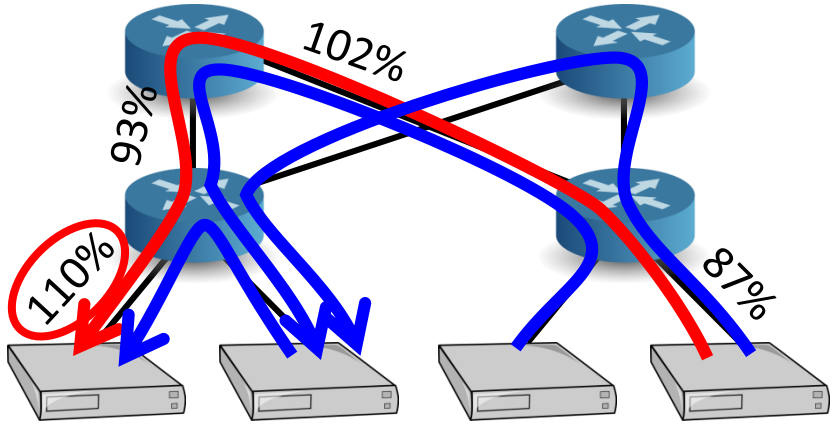
$$\widehat{x_s} = \frac{x_s}{\max(r_\ell)}$$

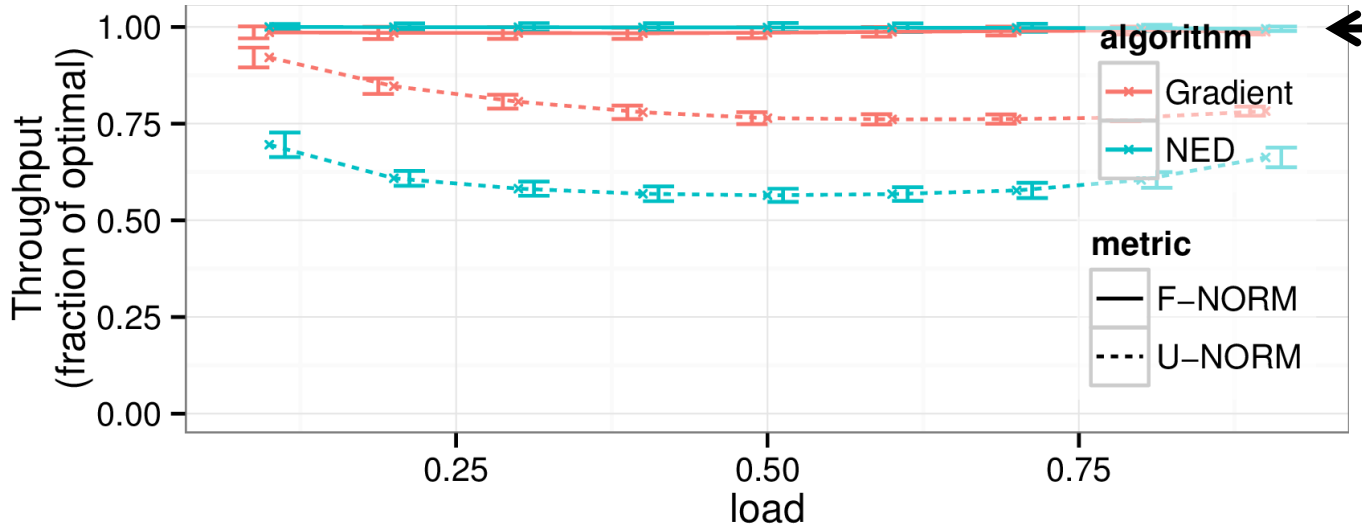


Flowtune normalizes rates

$$r_\ell = \frac{\text{allocation}}{\text{capacity}}$$

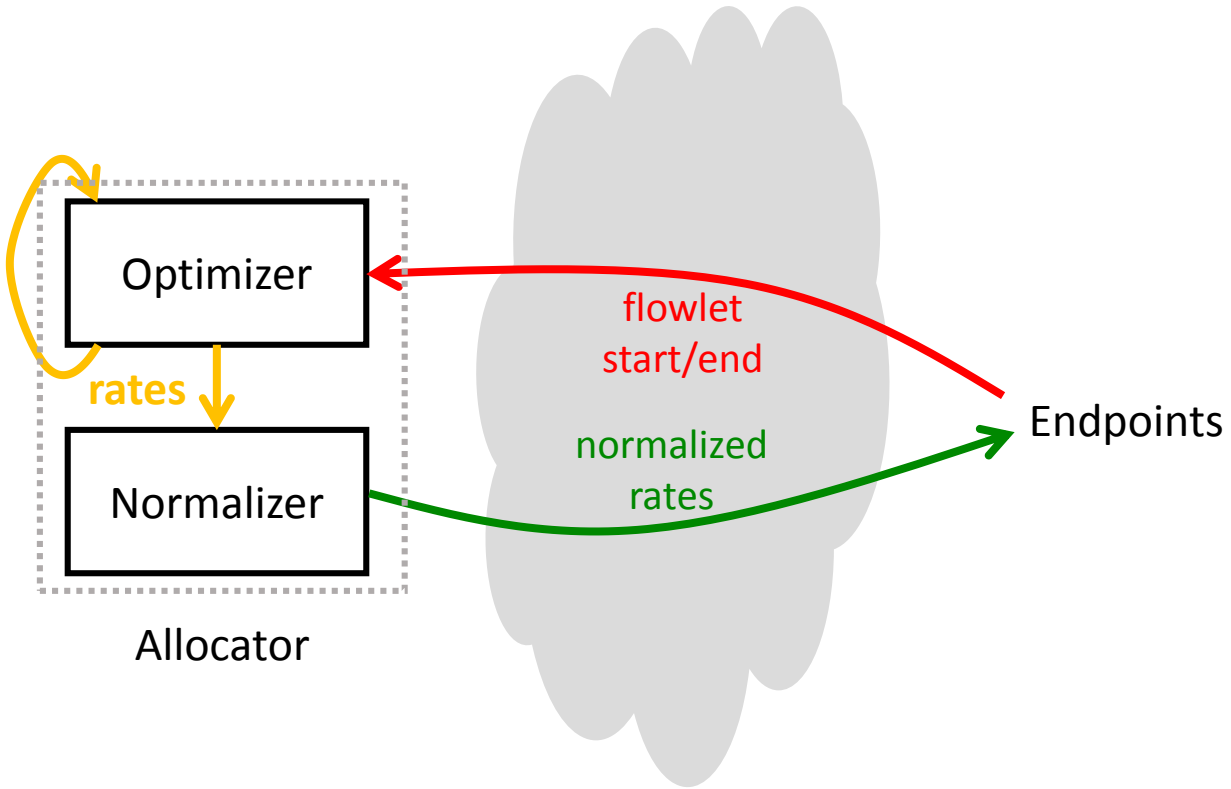
$$\widehat{x_s} = \frac{x_s}{\max(r_\ell)}$$



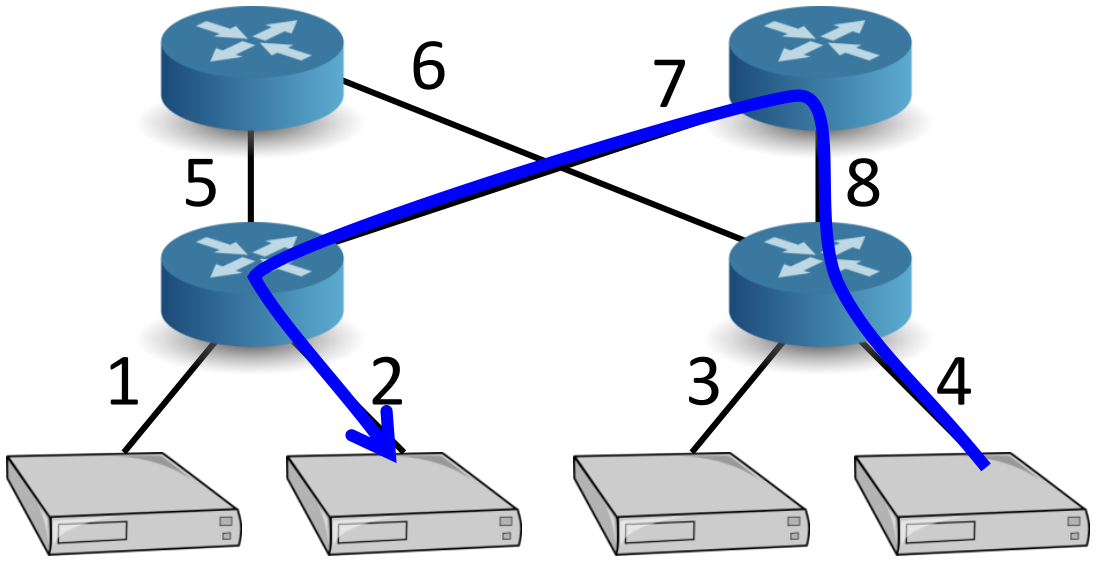


99.7% of optimal throughput

Architecture



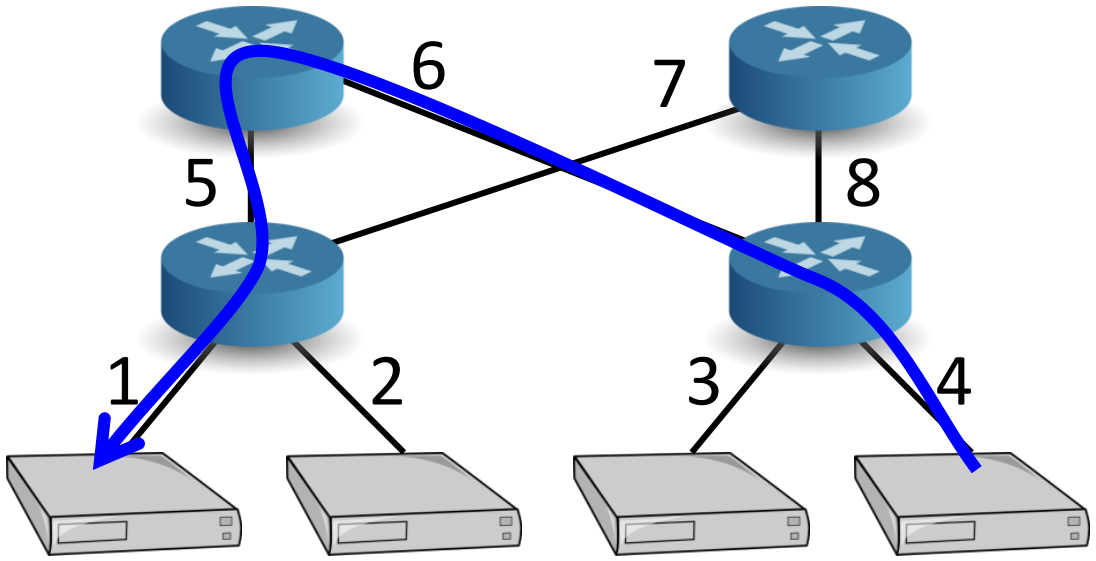
Multicore



Core 1: p_1 p_2 p_3 p_4

Core 2: p_5 p_6 p_7 p_8

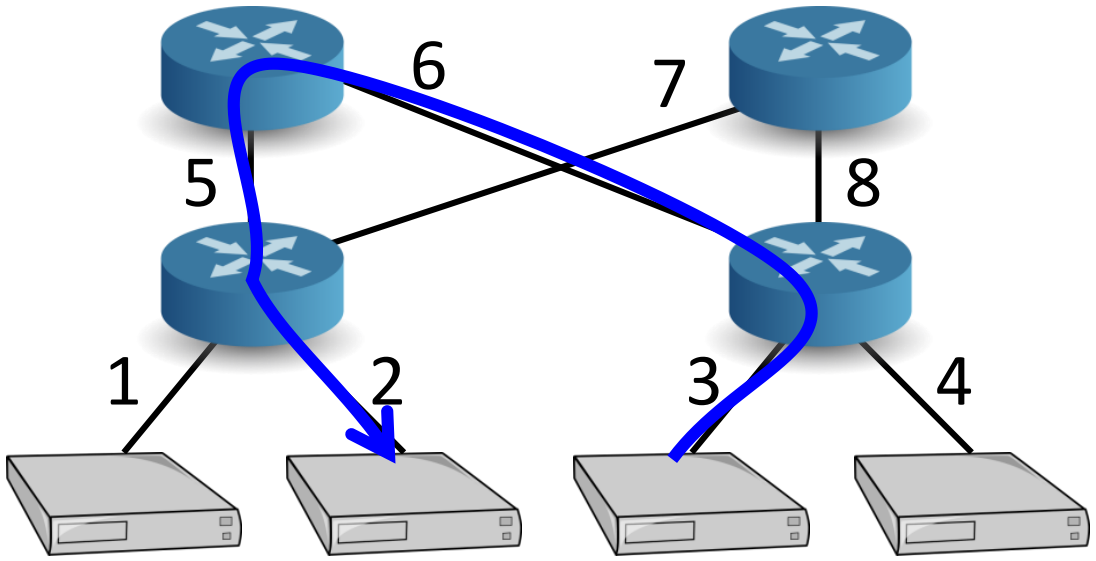
Multicore



Core 1: p_1 p_2 p_3 p_4

Core 2: p_5 p_6 p_7 p_8

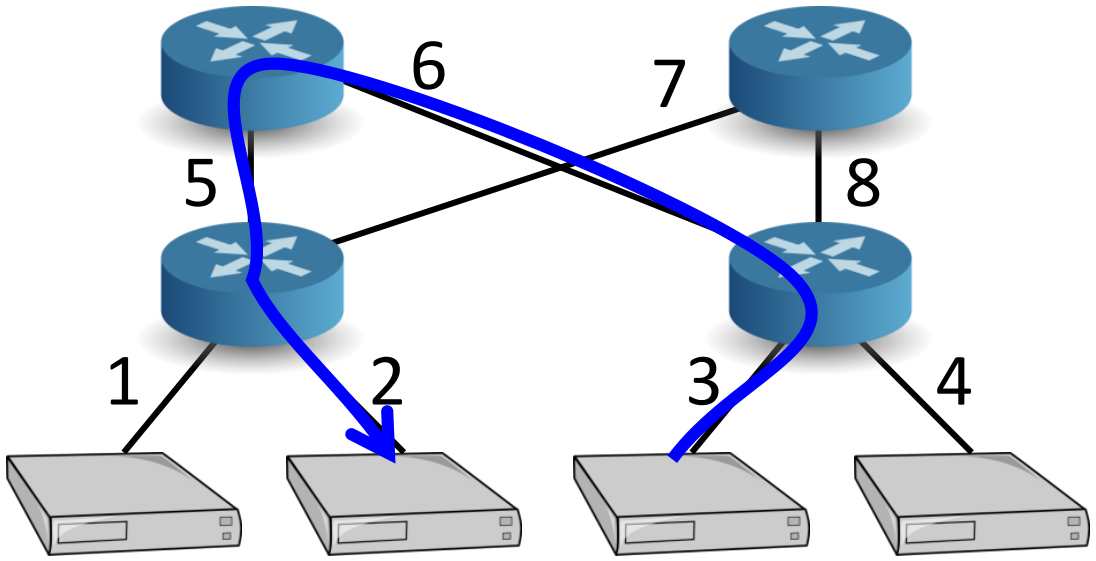
Multicore



Core 1: p_1 p_2 p_3 p_4

Core 2: p_5 p_6 p_7 p_8

Multicore



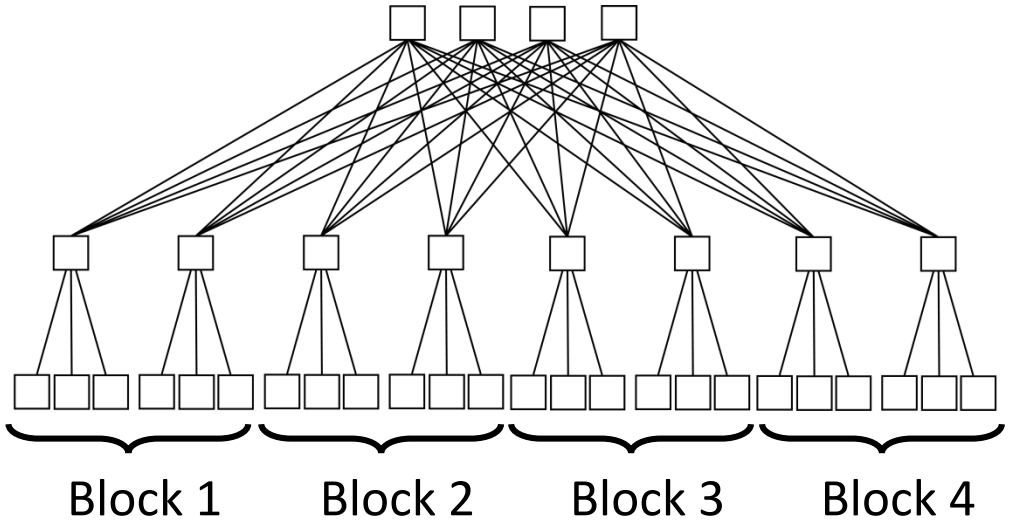
Core 1: p_1 p_2 p_3 p_4

Core 2: p_5 p_6 p_7 p_8

For each link ℓ compute $f(\text{flows on } \ell)$

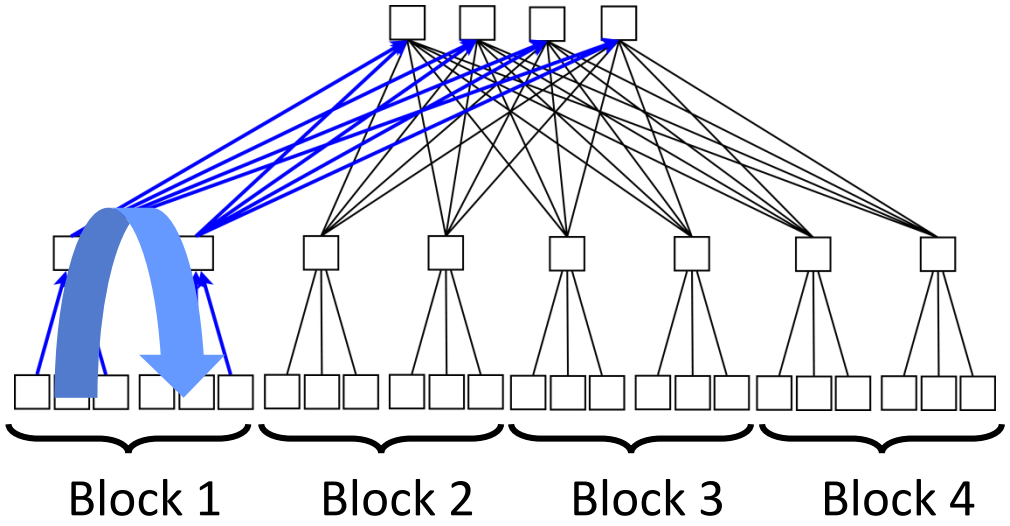
For each flow x compute $g(\text{links } x \text{ traverses})$

Multicore



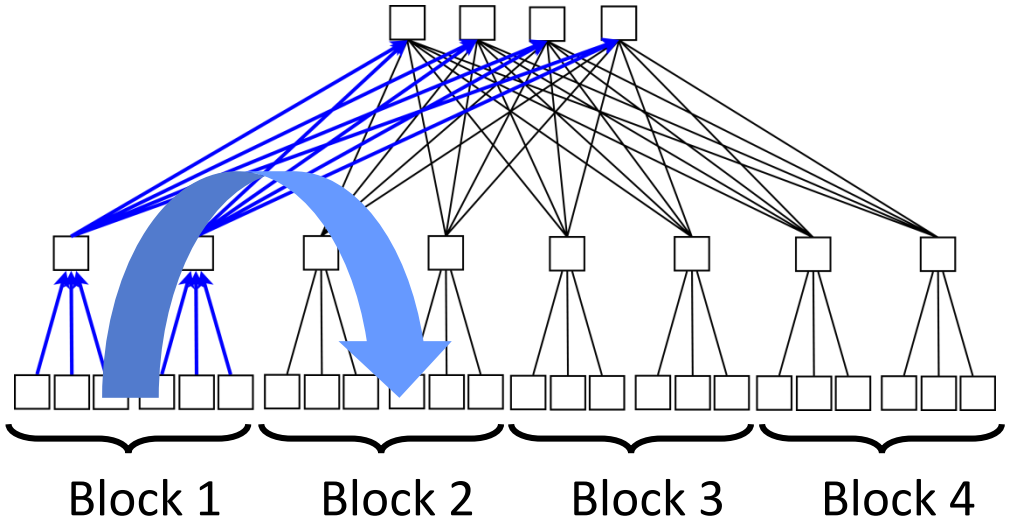
core 0	core 1	core 2	core 3
core 4	core 5	core 6	core 7
core 8	core 9	core 10	core 11
core 12	core 13	core 14	core 15

Multicore



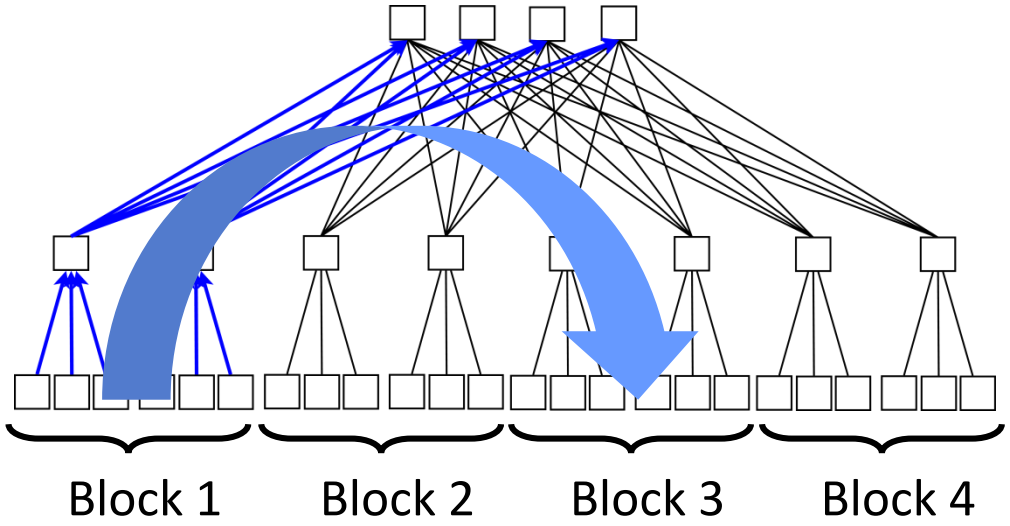
Source \ Destination	Block 1	Block 2	Block 3	Block 4
Block 1	core 0	core 1	core 2	core 3
Block 2	core 4	core 5	core 6	core 7
Block 3	core 8	core 9	core 10	core 11
Block 4	core 12	core 13	core 14	core 15

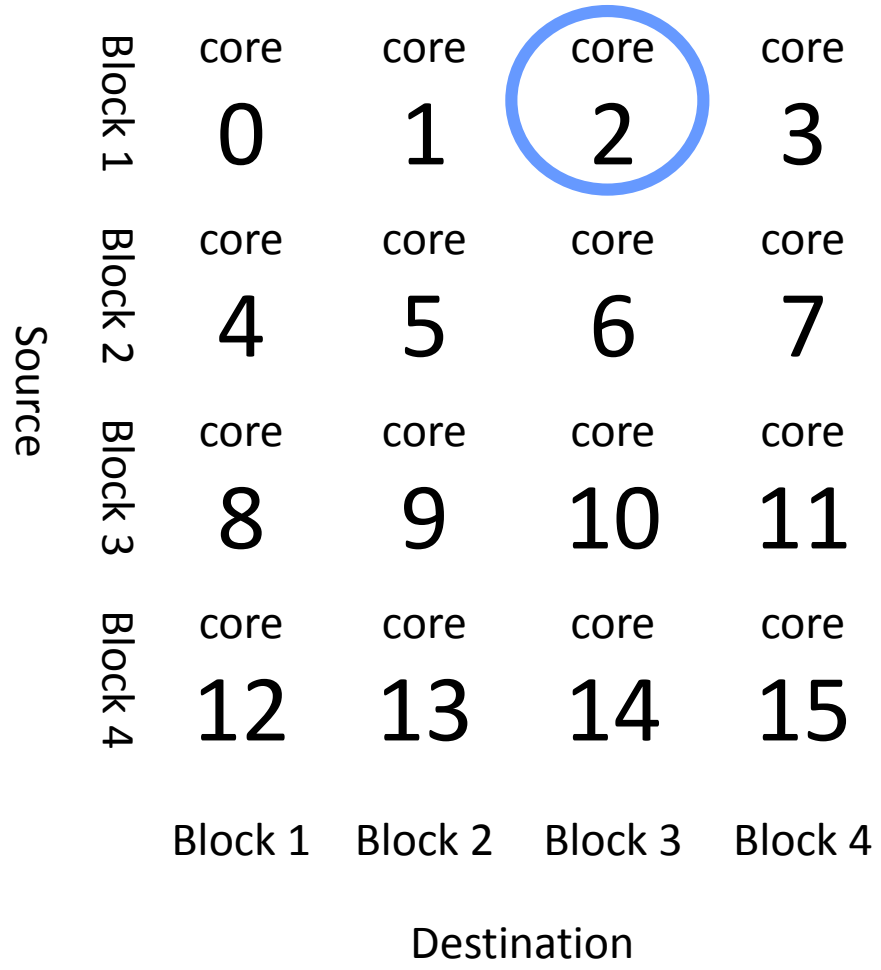
Multicore



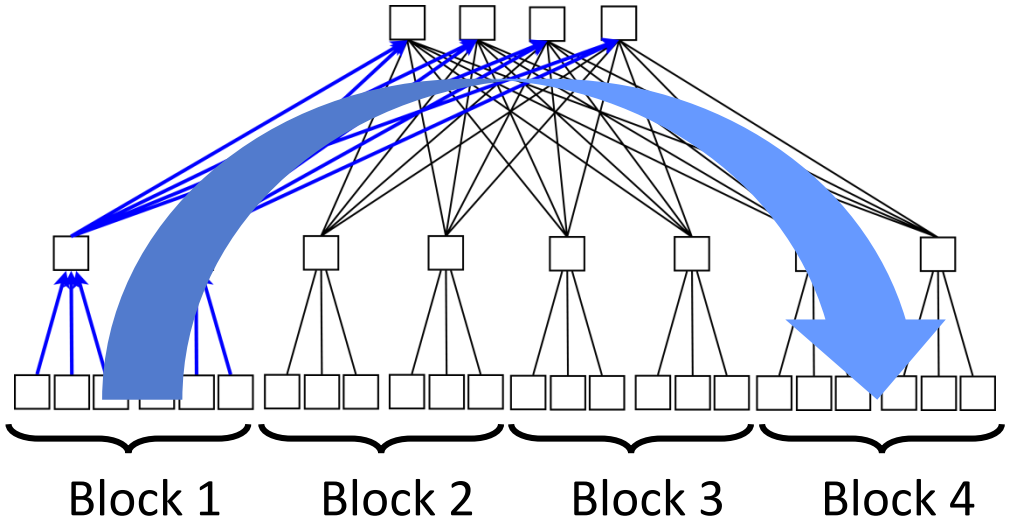
Source \ Destination	Block 1	Block 2	Block 3	Block 4
Block 1	core 0	core 1	core 2	core 3
Block 2	core 4	core 5	core 6	core 7
Block 3	core 8	core 9	core 10	core 11
Block 4	core 12	core 13	core 14	core 15

Multicore


Block 1
Block 2
Block 3
Block 4

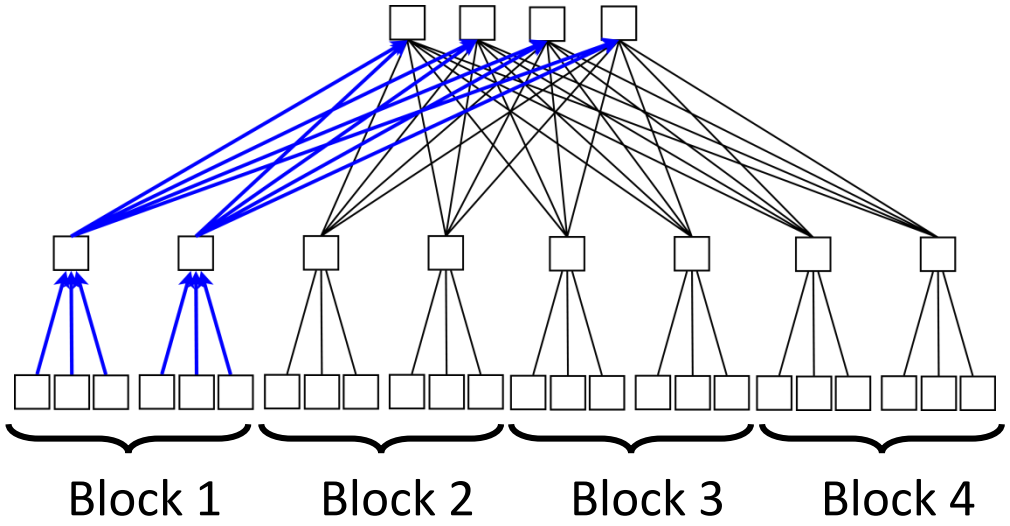

Source
Block 1
Block 2
Block 3
Block 4
core 0
core 1
core 2
core 3
core 4
core 5
core 6
core 7
core 8
core 9
core 10
core 11
core 12
core 13
core 14
core 15
Block 1
Block 2
Block 3
Block 4
Destination

Multicore



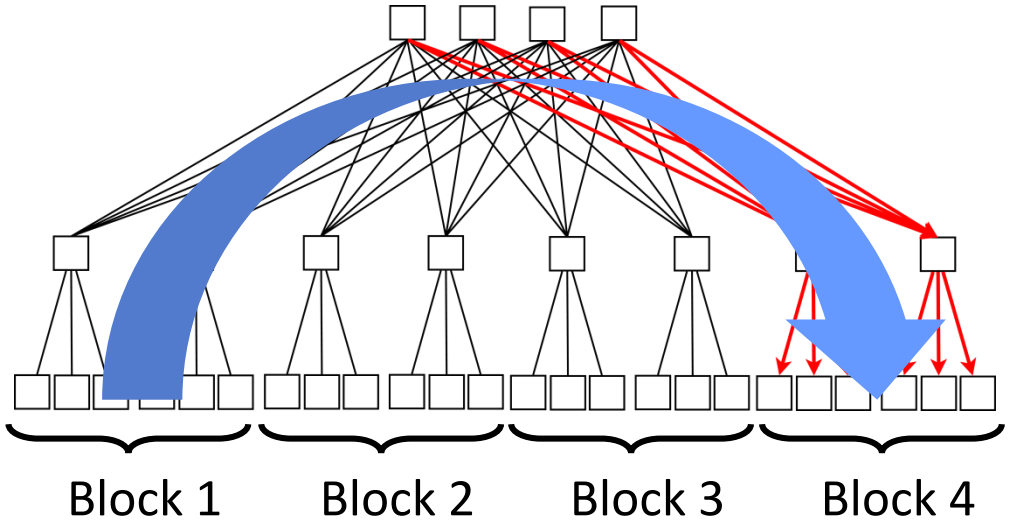
Source \ Destination	Block 1	Block 2	Block 3	Block 4
Block 1	core 0	core 1	core 2	core 3
Block 2	core 4	core 5	core 6	core 7
Block 3	core 8	core 9	core 10	core 11
Block 4	core 12	core 13	core 14	core 15

Multicore



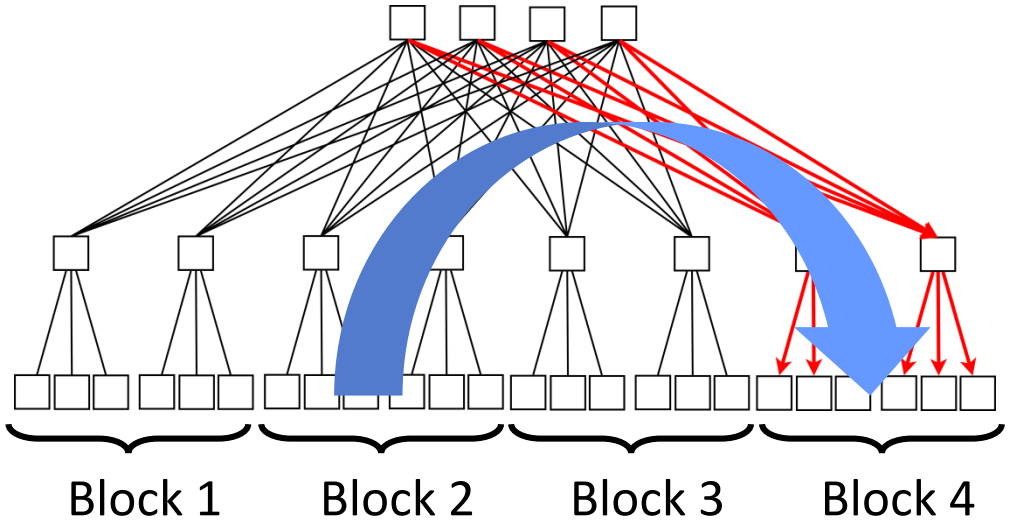
Source \ Destination	Block 1	Block 2	Block 3	Block 4
Block 1	core 0	core 1	core 2	core 3
Block 2	core 4	core 5	core 6	core 7
Block 3	core 8	core 9	core 10	core 11
Block 4	core 12	core 13	core 14	core 15

Multicore



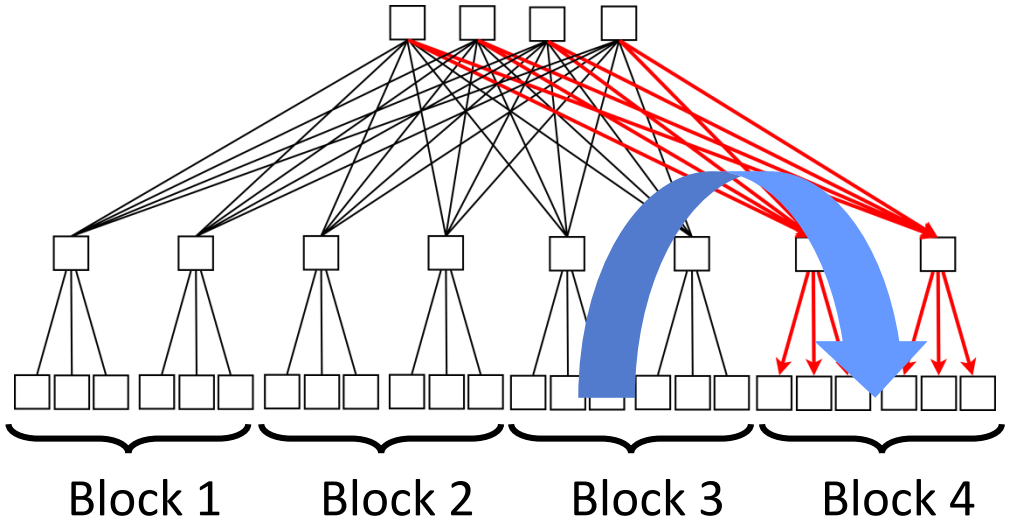
Source \ Destination	Block 1	Block 2	Block 3	Block 4
Block 1	core 0	core 1	core 2	core 3
Block 2	core 4	core 5	core 6	core 7
Block 3	core 8	core 9	core 10	core 11
Block 4	core 12	core 13	core 14	core 15

Multicore



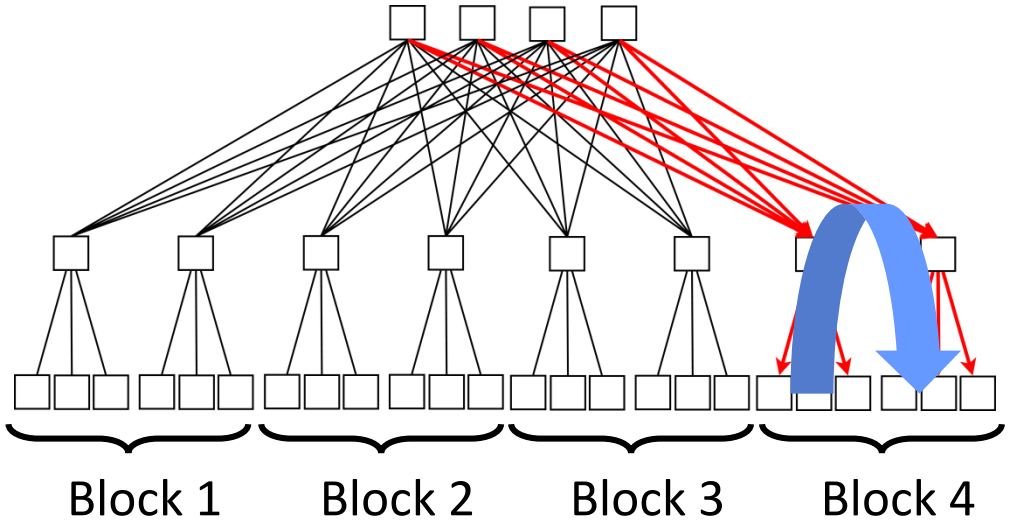
Source \ Destination	Block 1	Block 2	Block 3	Block 4
Block 1	core 0	core 1	core 2	core 3
Block 2	core 4	core 5	core 6	core 7
Block 3	core 8	core 9	core 10	core 11
Block 4	core 12	core 13	core 14	core 15

Multicore



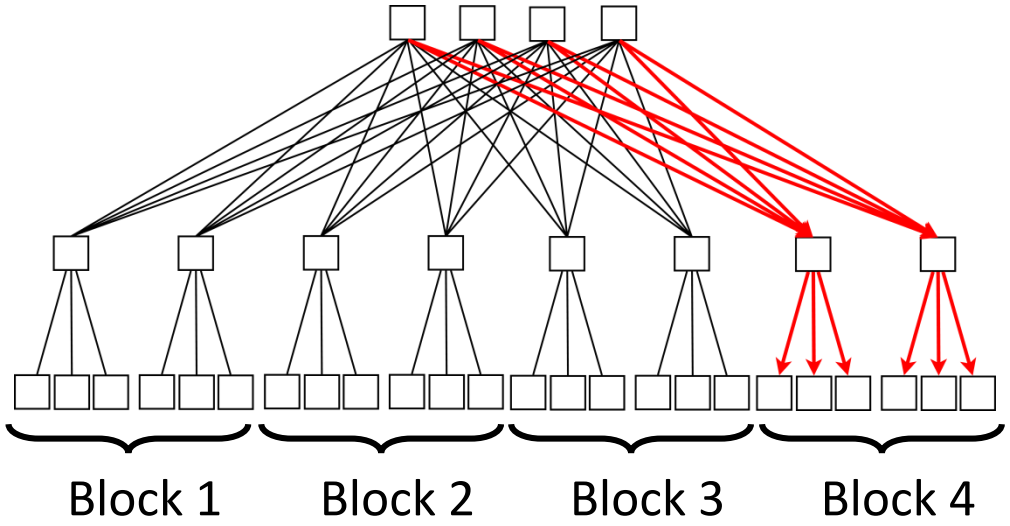
Source \ Destination	Block 1	Block 2	Block 3	Block 4
Block 1	core 0	core 1	core 2	core 3
Block 2	core 4	core 5	core 6	core 7
Block 3	core 8	core 9	core 10	core 11
Block 4	core 12	core 13	core 14	core 15

Multicore



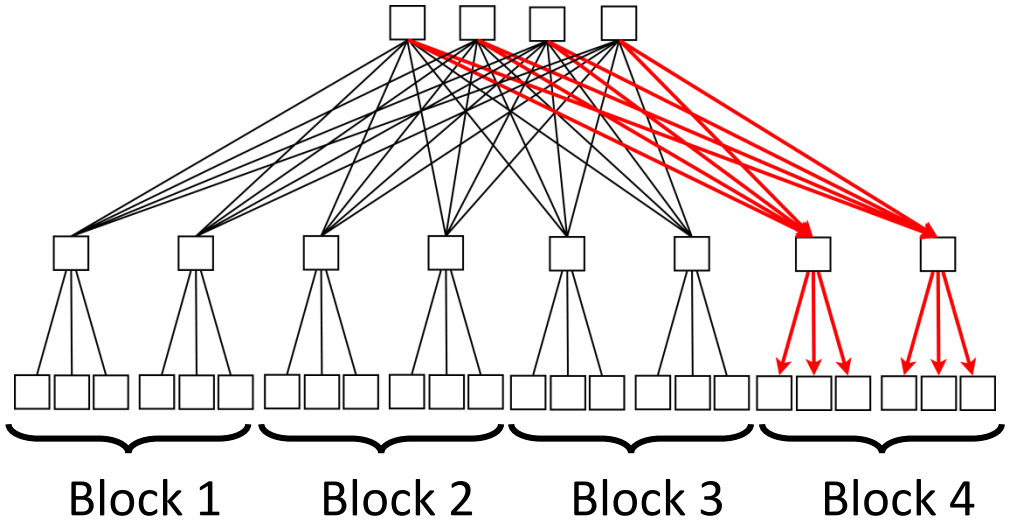
Source \ Destination	Block 1	Block 2	Block 3	Block 4
Block 1	core 0	core 1	core 2	core 3
Block 2	core 4	core 5	core 6	core 7
Block 3	core 8	core 9	core 10	core 11
Block 4	core 12	core 13	core 14	core 15

Multicore



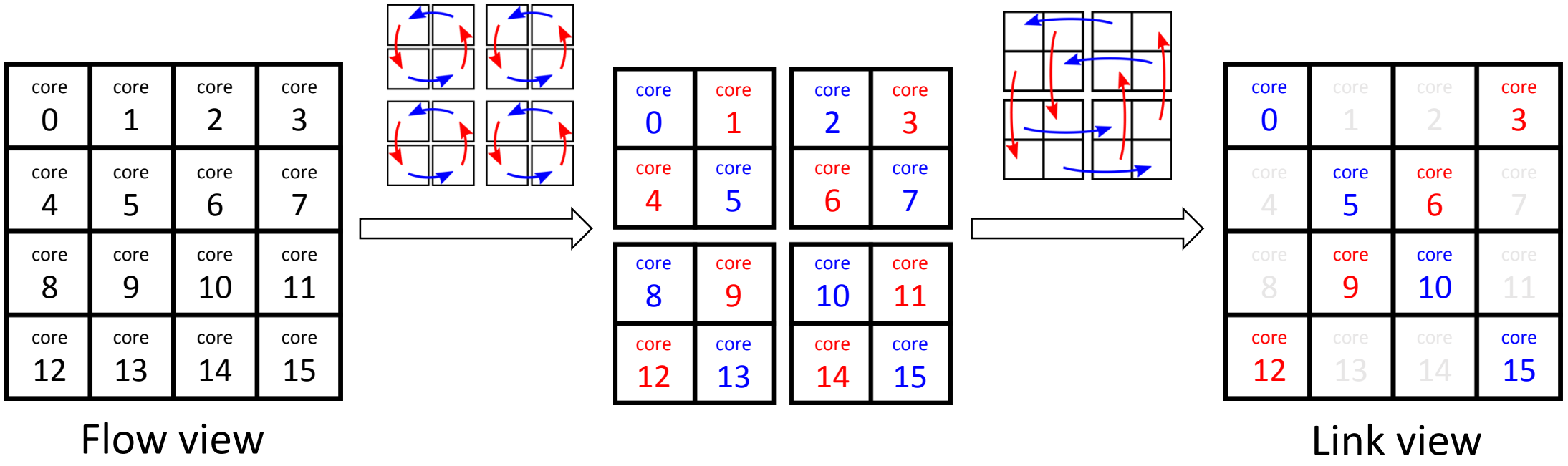
Source \ Destination	Block 1	Block 2	Block 3	Block 4
Block 1	core 0	core 1	core 2	core 3
Block 2	core 4	core 5	core 6	core 7
Block 3	core 8	core 9	core 10	core 11
Block 4	core 12	core 13	core 14	core 15

Multicore



Source \ Destination	Block 1	Block 2	Block 3	Block 4
Block 1	core 0	core 1	core 2	core 3
Block 2	core 4	core 5	core 6	core 7
Block 3	core 8	core 9	core 10	core 11
Block 4	core 12	core 13	core 14	core 15

In the paper…

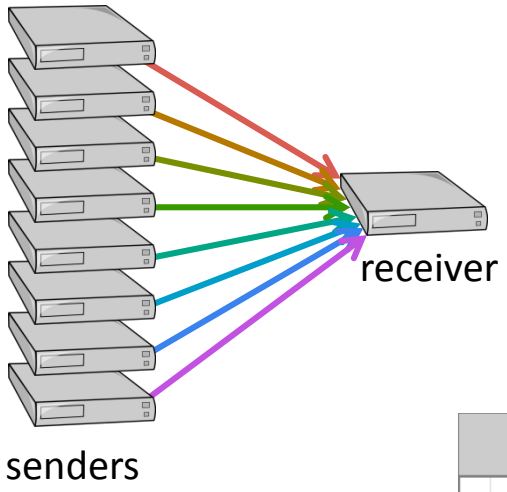

core 0
core 1
core 2
core 3
core 4
core 5
core 6
core 7
core 8
core 9
core 10
core 11
core 12
core 13
core 14
core 15
Flow view
Link view

4608 servers in < $31\mu s$

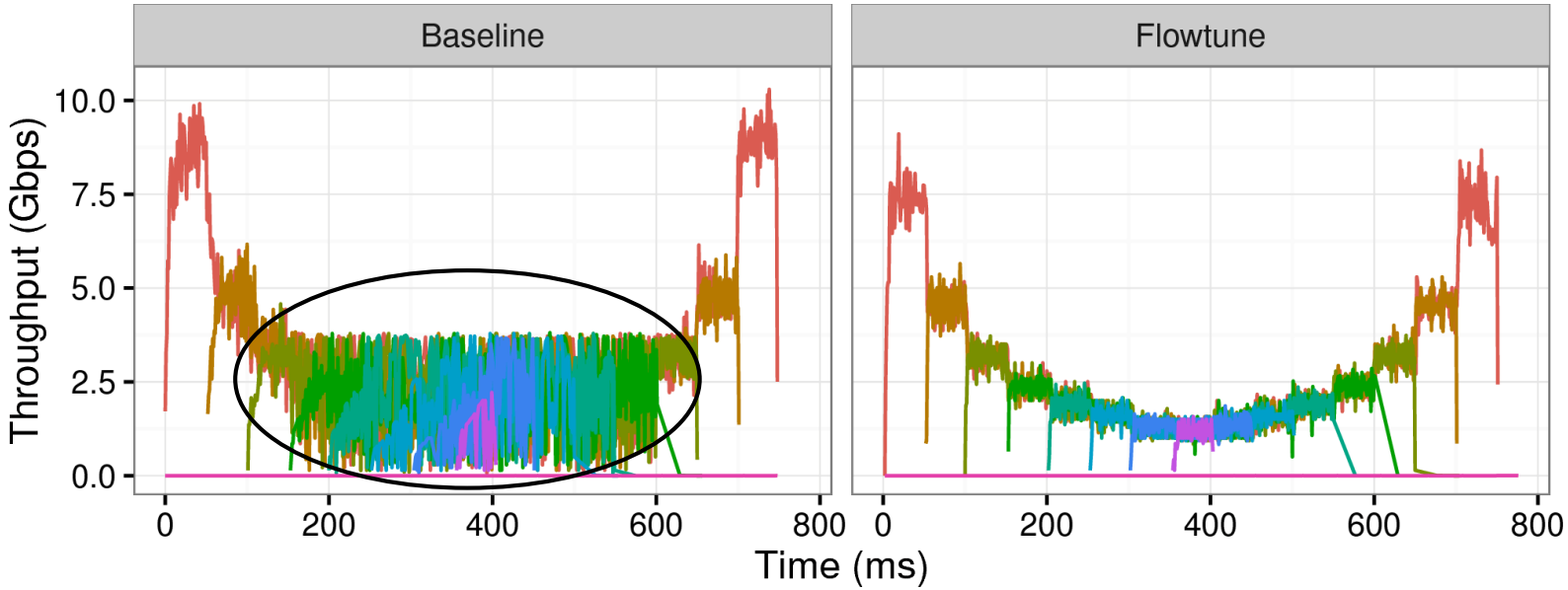
Cores	Nodes	Flows	Cycles	Time
4	384	3072	19896.6	8.29 µs
16	768	6144	21267.8	8.86 µs
64	1536	12288	30317.6	12.63 µs
64	1536	24576	33576.2	13.99 µs
64	1536	49152	40628.5	16.93 µs
64	3072	49152	57035.9	23.76 µs
64	4608	49152	73703.2	30.71 µs

Communication $> \frac{1}{2}$ of time

EC2: Resource Allocation

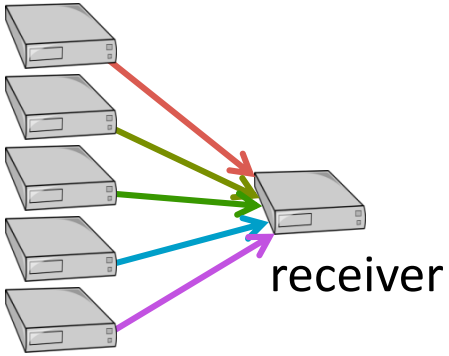


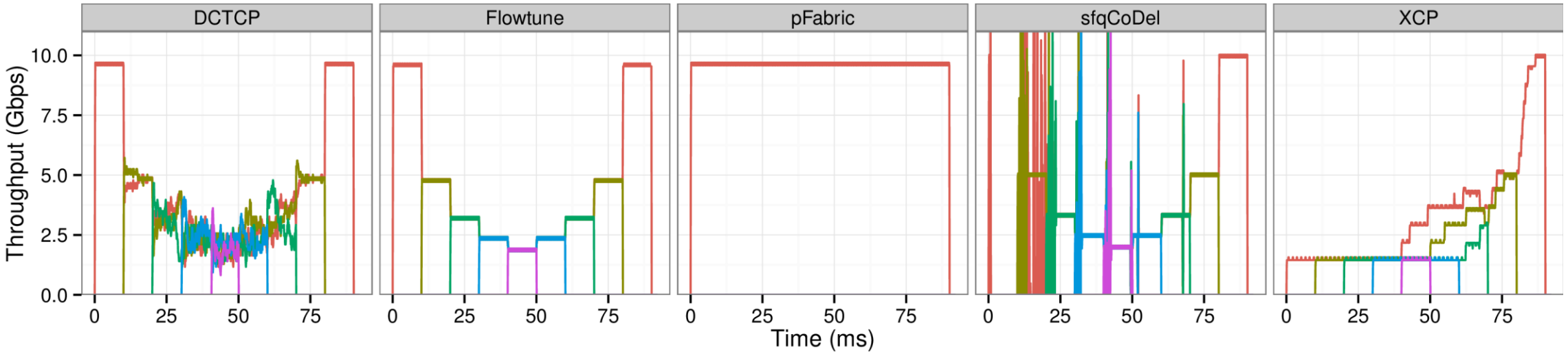
8 senders, every 50 milliseconds



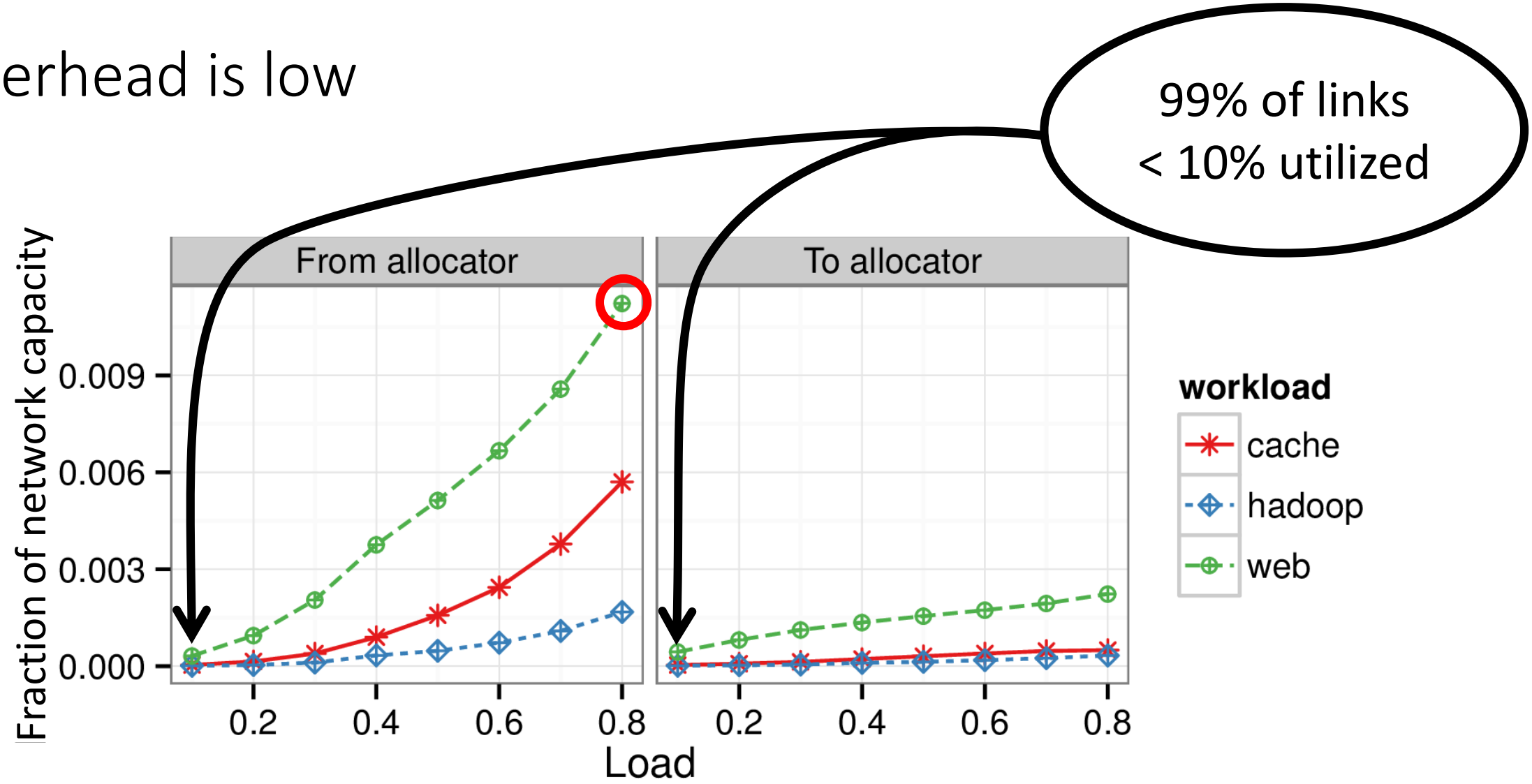
Ns-2: Flowtune converges quickly to a fair allocation

Every 10 milliseconds:





Overhead is low



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

Open Questions

- Handling mice
  - Bypass the allocator? Fastpass?
- External traffic
  - Measure & react?
- Deadlines, Co-flow
  - Market?
- Multicore: 3-tier Clos, WAN

Flowtune

Application Developers ⟺ Explicit Policy ⟺ Network Engineers

Give application developers
control over network transport