Modern Congestion Control

Brighten Godfrey

CS 538 February 13 2017

Including slides by Mo Dong

Isn't Congestion Control a done deal?

are

~~can be~~ not happy with TCP ?

TCP

Est. 1988

High BDP	Wireless	Satellite	Inter-DC
BIC H-TCP Compound CUBIC FAST TCP	Westwood Vegas Veno	Hybla STAR	Illinois SABUL
10X	10X	17X	4X

Unstable， RTT Unfair, Bufferbloat, Crash on Changing Networks, ……..

Point Solutions

+

Performance

Far from Optimal

Why is it so hard?

Two directions today

Help from inside the network

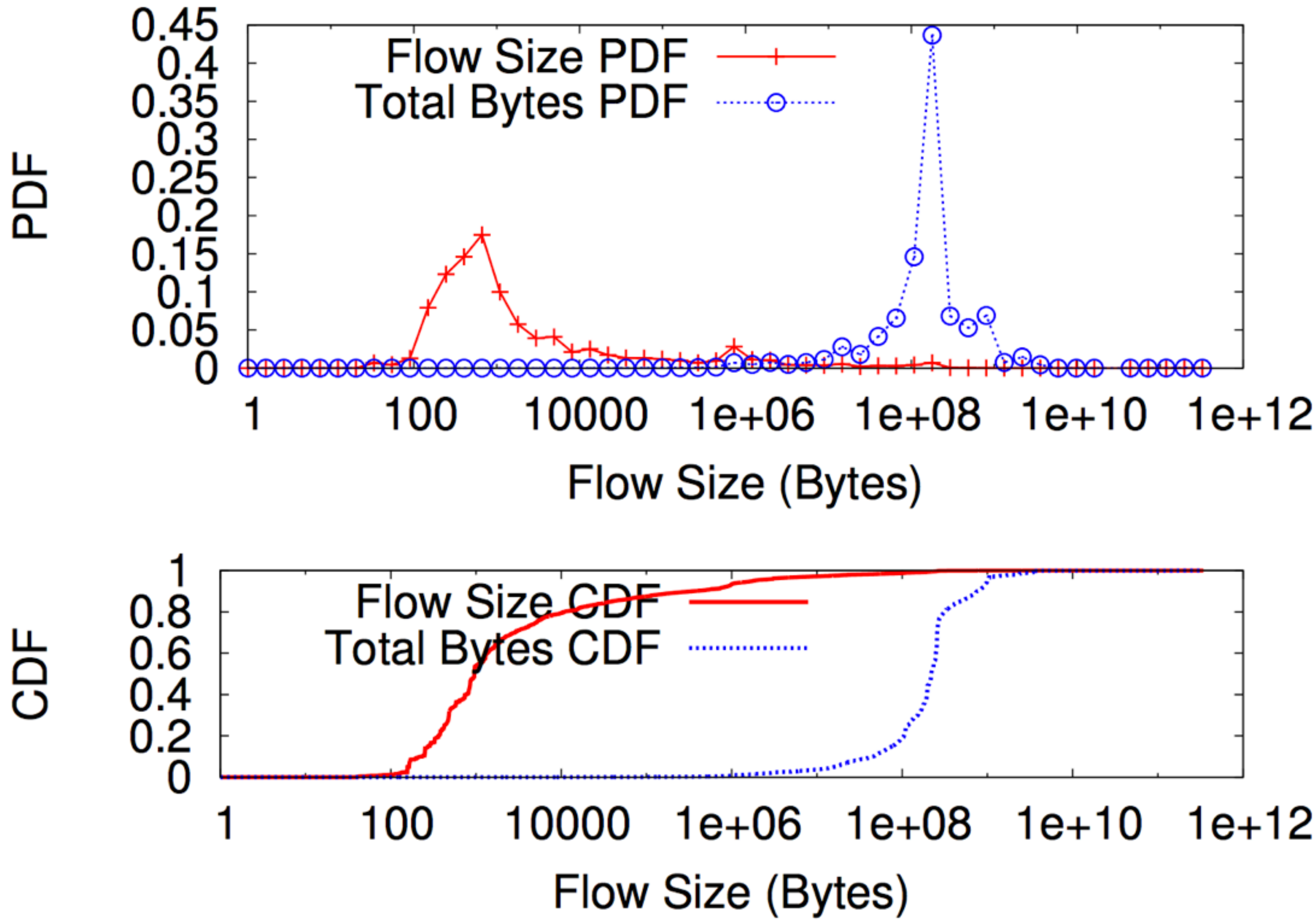
Better end-to-end algorithms

DCTCP

[Alizadeh et al., SIGCOMM'10]

(adapted from Alizadeh's slides)

Data center traffic characteristics



[VL2, SIGCOMM'09]

What do we want?

Short flows

complete flows before their deadlines

Long flows

no deadline, but still preferable to finish earlier

Low latency is the key

YAHOO!

400 ms slowdown resulted in a traffic decrease of 9%

[Yslow 2.0; Stoyan Stefanov]

Google

100 ms slowdown reduces # searches by 0.2-0.4%

[Speed matters for Google Web Search; Jake Brutlag]

AOL

Users with lowest 10% latency viewed 50% more pages than those with highest 10% latency

[The secret weapons of the AOL optimization team; Dave Artz]

mozilla Firefox

2.2 sec faster web response increases 60 million more Firefox install package downloads per year

[Firefox and Page Load Speed; Blake Cutler]

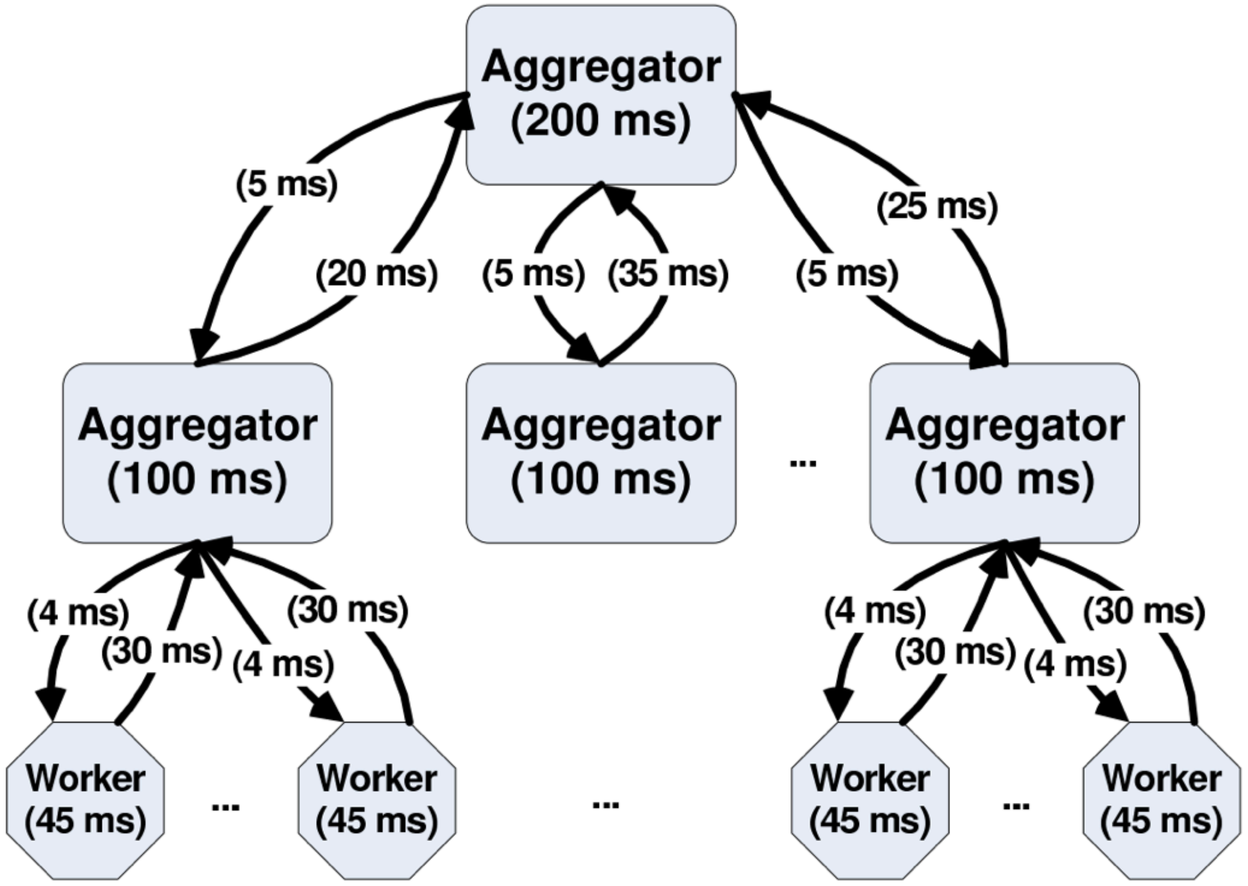
Walmart

Users with 0-1 sec load time have 2x conversion rate of 1-2 sec

[Is page performance a factor of site conversion? And how big is it; Walmart Labs]

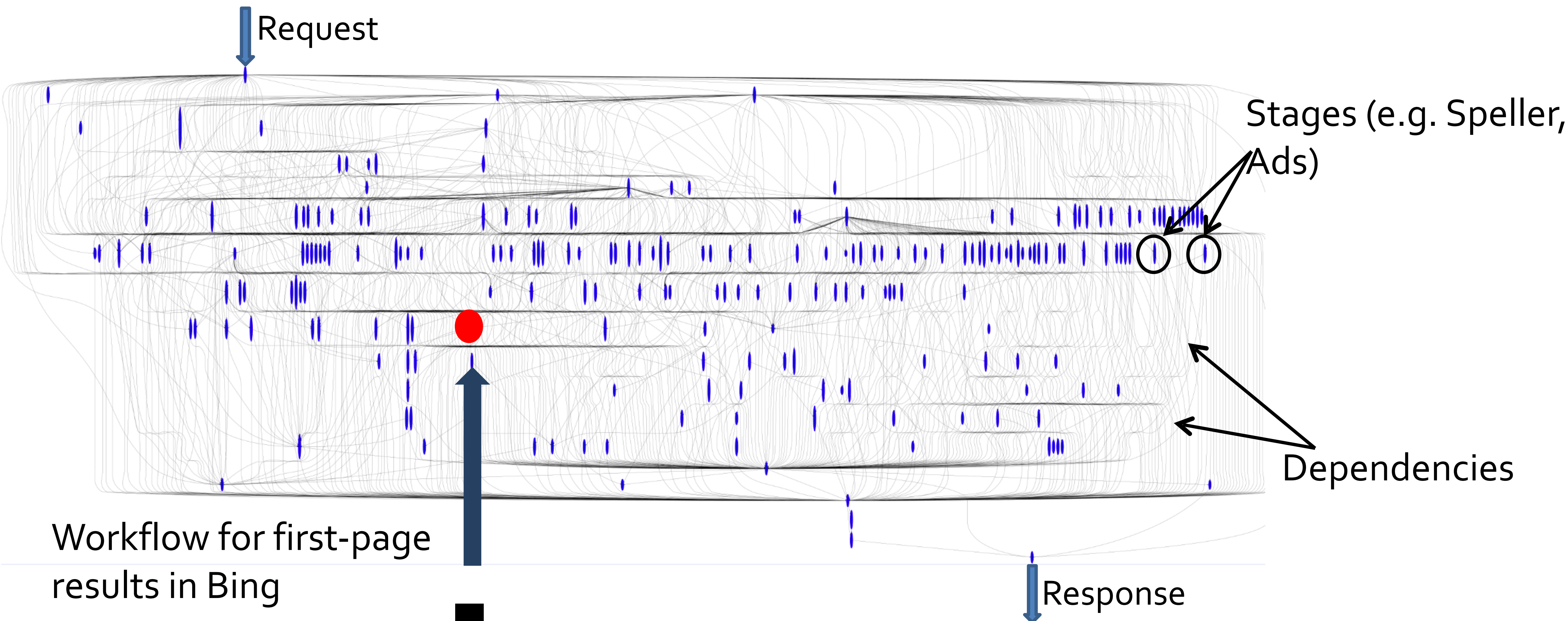
Improving latency in data centers

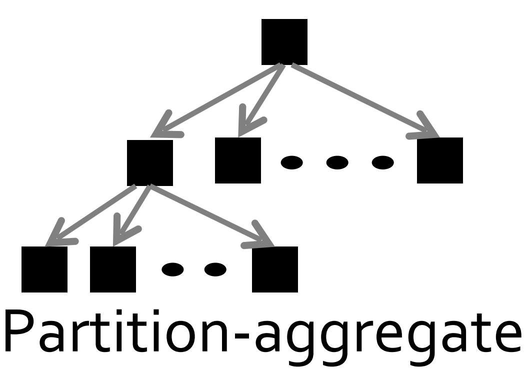
Server side optimization: Parallel computation



partition aggregate model

Web services have complex workflows





– Up to 150 stages, degree of 40, path lengths of 10 or more
– Each stage can be a complex workflow

Stochastic delays accumulate across stages

From: Speeding up Distributed Request-Response Workflows, Jalaparti, Bodik, Kandula, Menache, Rybalkin, and Yan, SIGCOMM 2013

Three problems [DCTCP]

Incast

Queue buildup

Buffer pressure

Incast

Basic problem

- Synchronized flows overflow the switch buffer

Causes

- (Barrier) synchronized many-to-one traffic pattern
- Short flows (10s KB to 100s KB)
- Small queue buffer (4 to 8 MB shared memory)
- Large default RTO (300 ms)

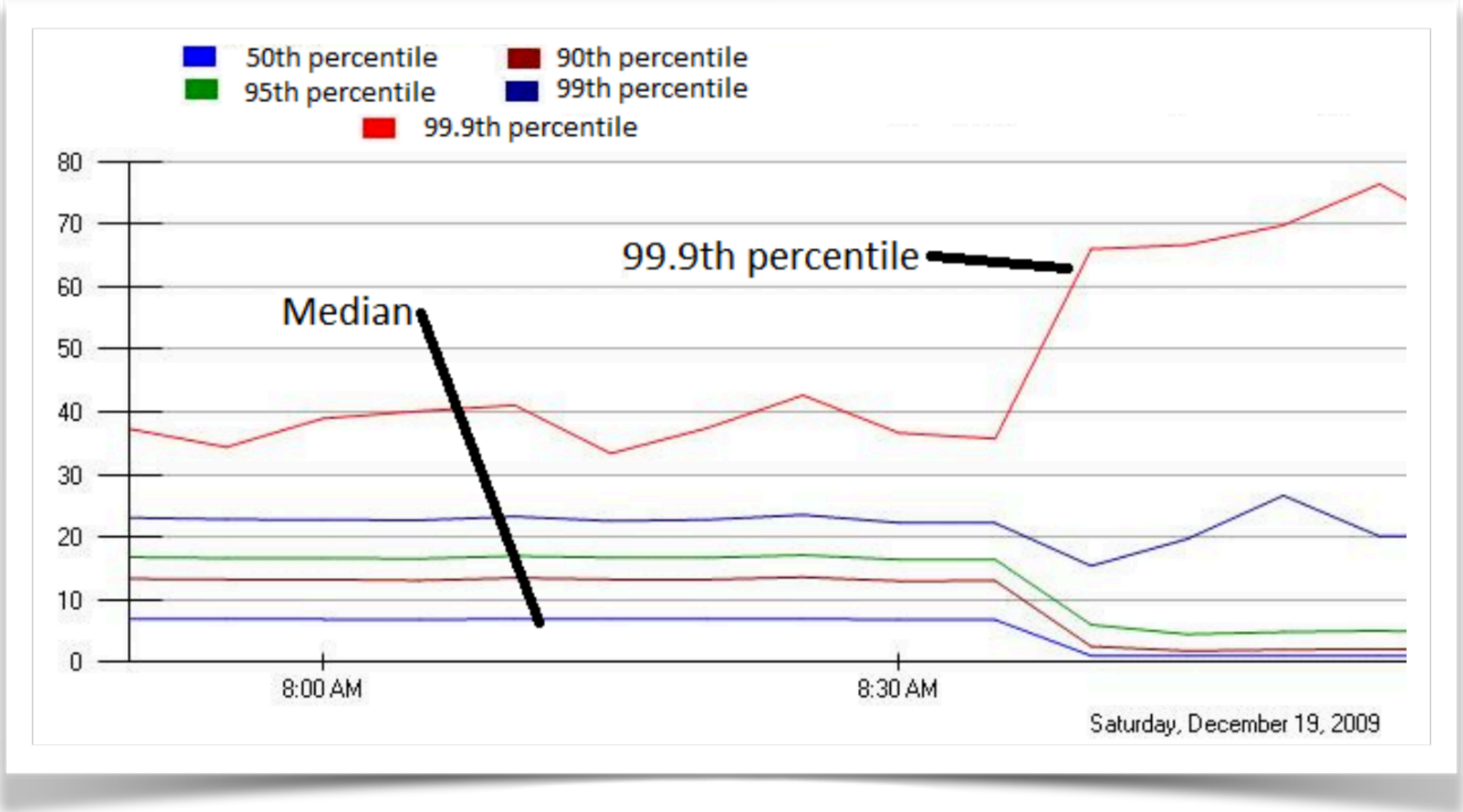
Fixing TCP Incasts

Use larger switch buffers

Decrease RTOmin

Desynchronize flows (random delay ~10ms)

Query completion time [ms]



Queue buildup and buffer pressure

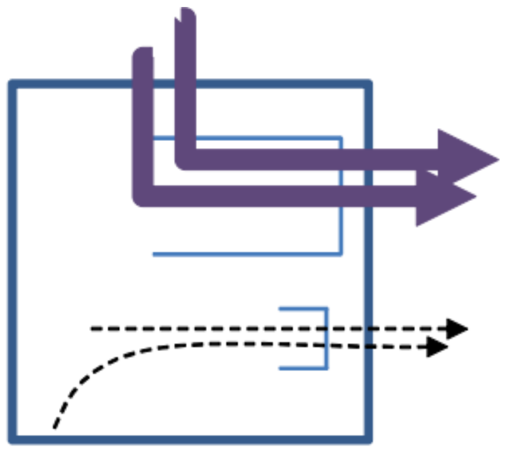
Causes: Long TCP flows occupy switch buffer

Queue buildup: short flow experiences increased delay

90%: RTT < 1ms (Bing's DC)

10%: 1 ms < RTT < 15 ms

Buffer pressure: 4 MB shared memory, i.e., how much buffer per port is not a constant



Many solutions to Incast do not apply here...

DCTCP: Two goals

Goal #1: Low latency and high burst tolerance

- Ensuring low queue occupancy

Goal #2: Still having high throughput for long flows

- Using most of the network bandwidth

Achieving either goal is not hard; what's hard is to achieve both

Explicit Congestion Notification

Switches mark packet's ECN bit *before* buffer overflows

TCP sender treats ECN signals as if a single packet is dropped — but packets are not actually dropped

More useful for short flows — avoid packet drop, therefore avoid RTO timeout.

Well-supported by today's commodity switches and end-hosts

DCTCP: Two Key ideas

1. React in proportion to the extent of congestion, not its presence

ECN Marks	TCP	DCTCP
1011110111	cut window by **50%**	cut window by **40%**
0000000001	cut window by **50%**	cut window by **5%**

2. Mark based on instantaneous queue length
   - Fast feedback to better deal with bursts

DCTCP Algorithm

Switch side:

- mark packet iff queue length > *K*

Sender side:

- maintain running avg of fraction of marked pkts

In each RTT:

$$F = \frac{\#\ of\ marked\ ACKs}{Total\ \#\ of\ ACKs} \qquad \alpha \leftarrow (1-g)\alpha + gF$$

- adaptive window decreases: $Cwnd \leftarrow (1 - \frac{\alpha}{2})Cwnd$

Why does it work?

Small buffer occupancies

→ bursts fit

→ low queueing delay

Aggressive marking when queue buffer builds up

→ fast reaction before packet drops

Adaptive window reduction

→ high throughput

Discussion

- Can we leverage more capability and information in date center environment? Switch features? Traffic patterns? etc..
- Can we use DCTCP in wide area networks?
- Can we use other switch features to improve the performance?
- Short flow performance in general settings

Discussion

How does RCP compare?

Does this solve all our congestion problems in DCs?

Could we apply DCTCP to wide-area environments?

Dealing with complex environments without information from the network


TCP
ex machina
TCP Ex Machina: Computer-Generated Congestion Control
Keith Winstein and Hari Balakrishnan
SIGCOMM 2013
Figures in following slides from Remy project
http://exmachina-movie.com/

Machine learning based CC

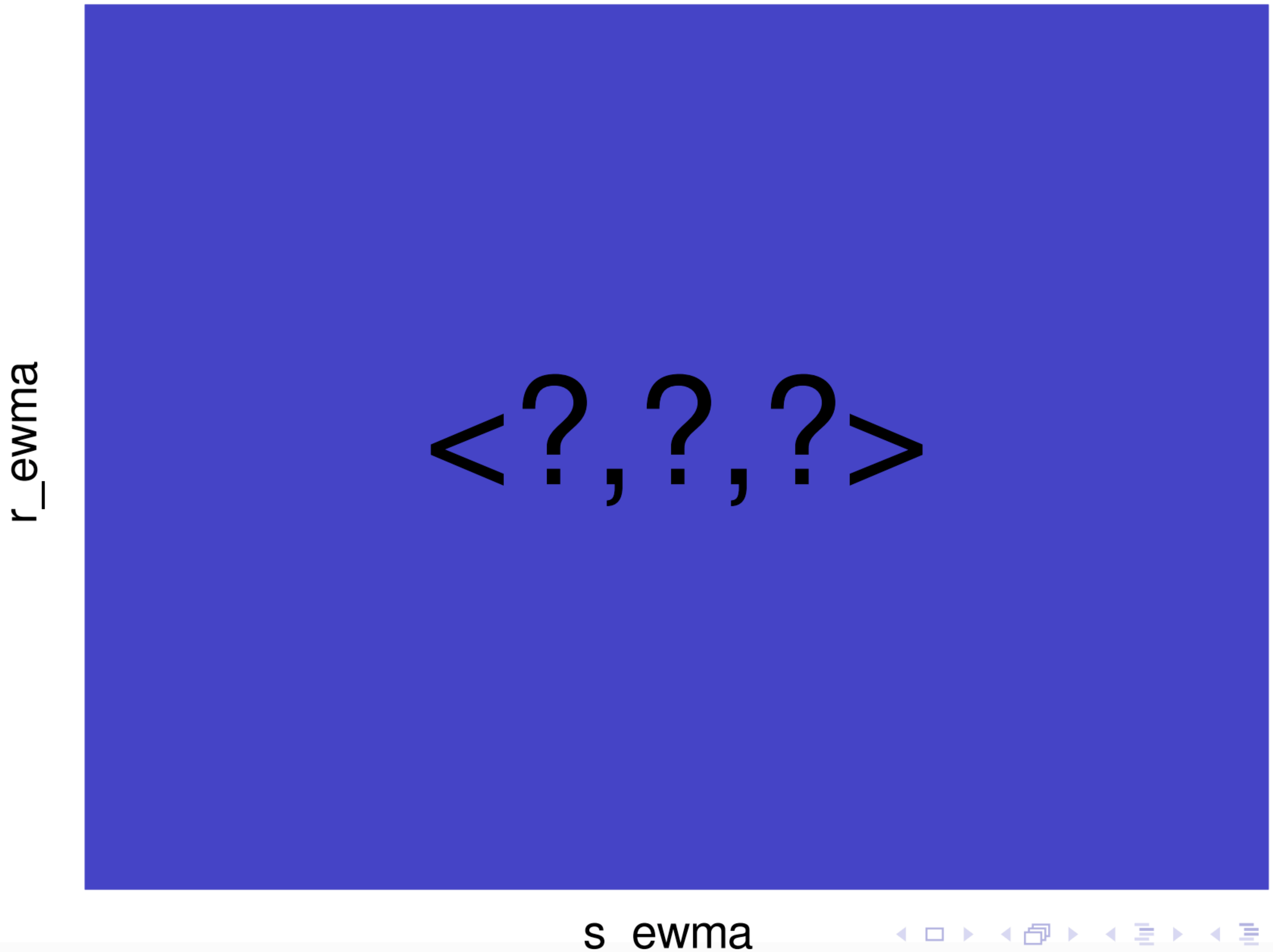
- Given a range of possible network conditions
  - Bandwidth, RTT, number of senders
- Using a set of congestion control signal
  - r_ewma, s_ewma, rtt_ratio

Machine learning based CC

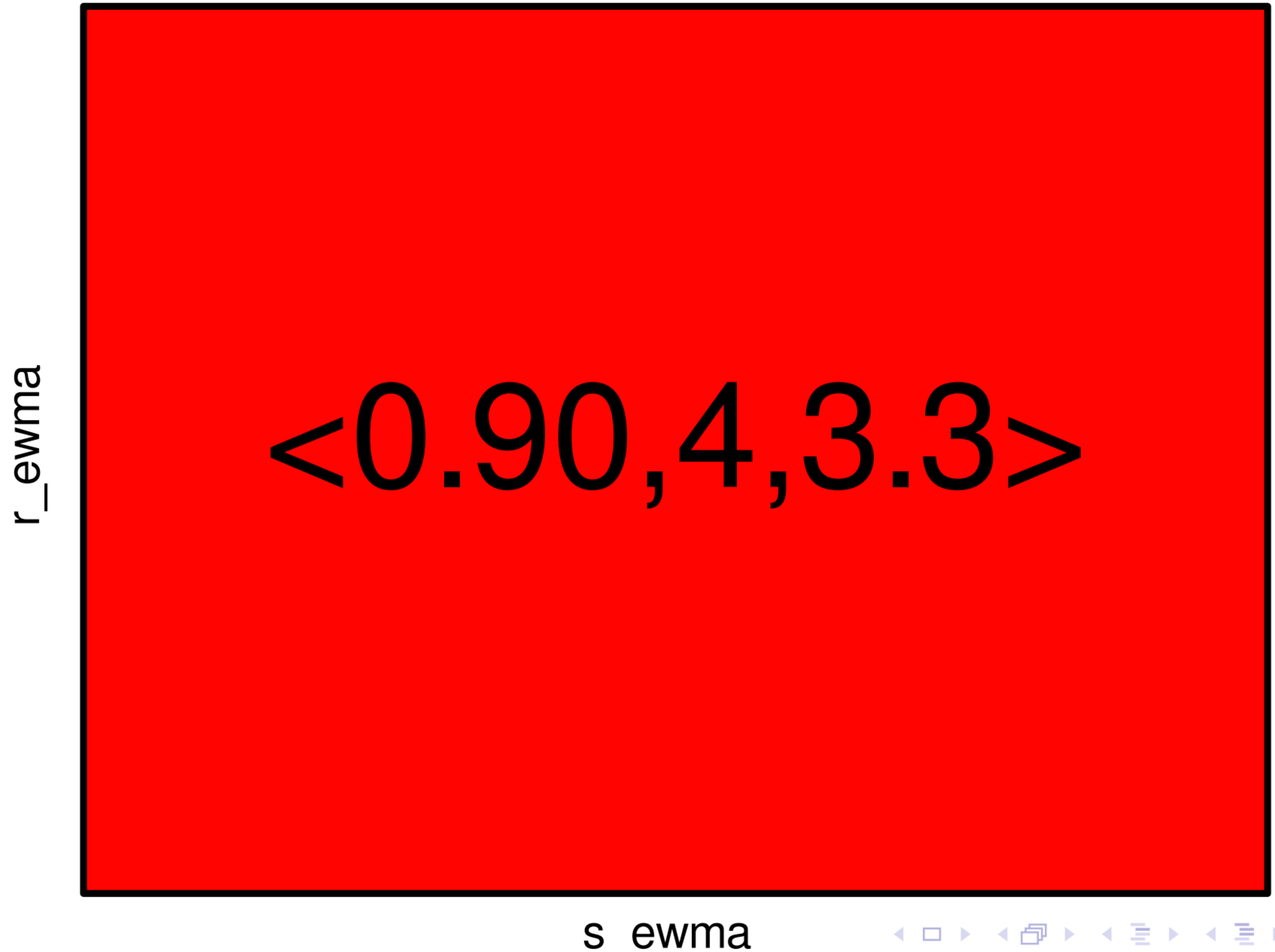
- Use offline machine learning to train a map
  - Rule(r_ewma, s_ewma, rtt_ratio) → <m, b, τ>

m	Multiple to congestion window
b	Increment to congestion window
τ	Minimum interval between two outgoing packets

One action for all state

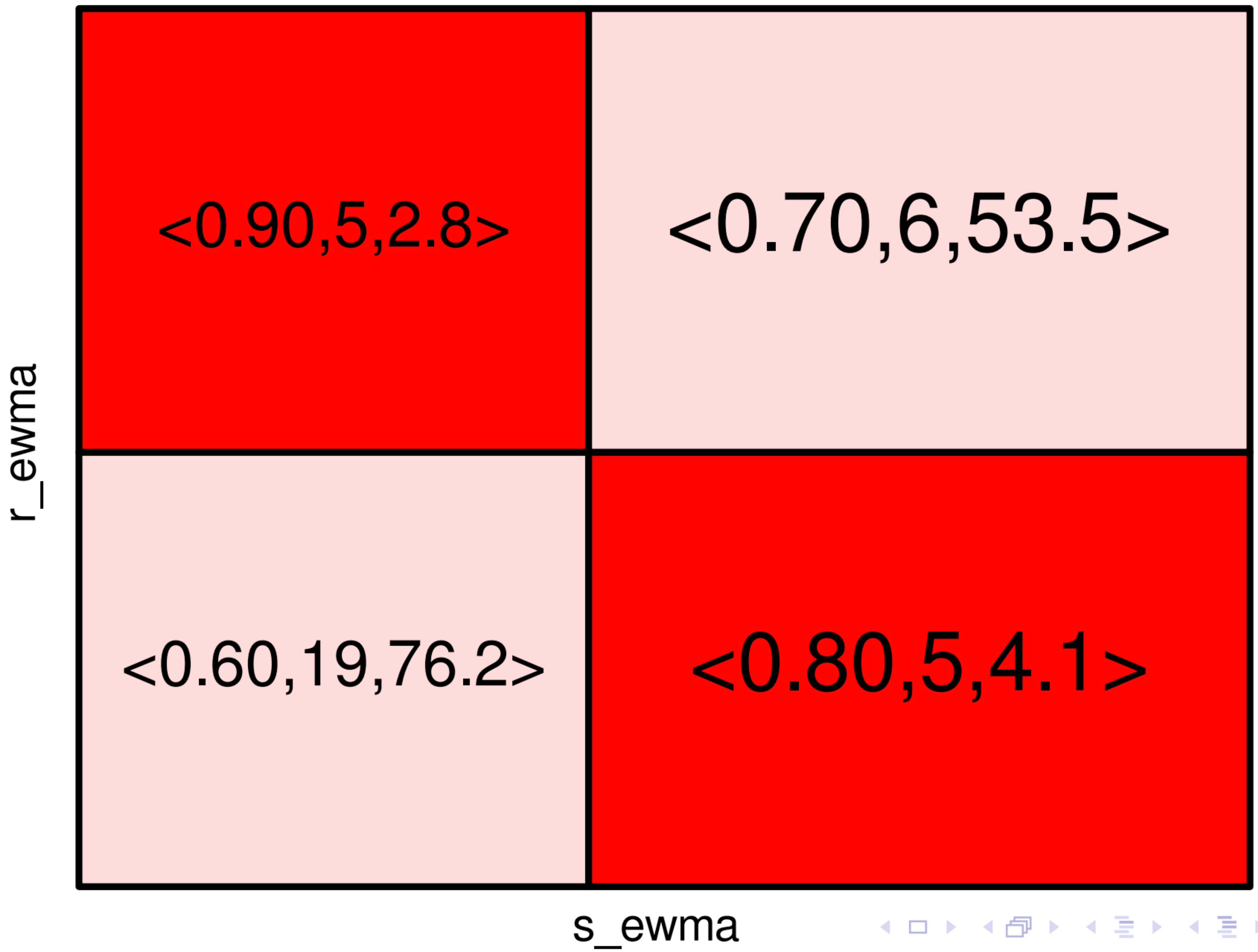


The best single action, split on median

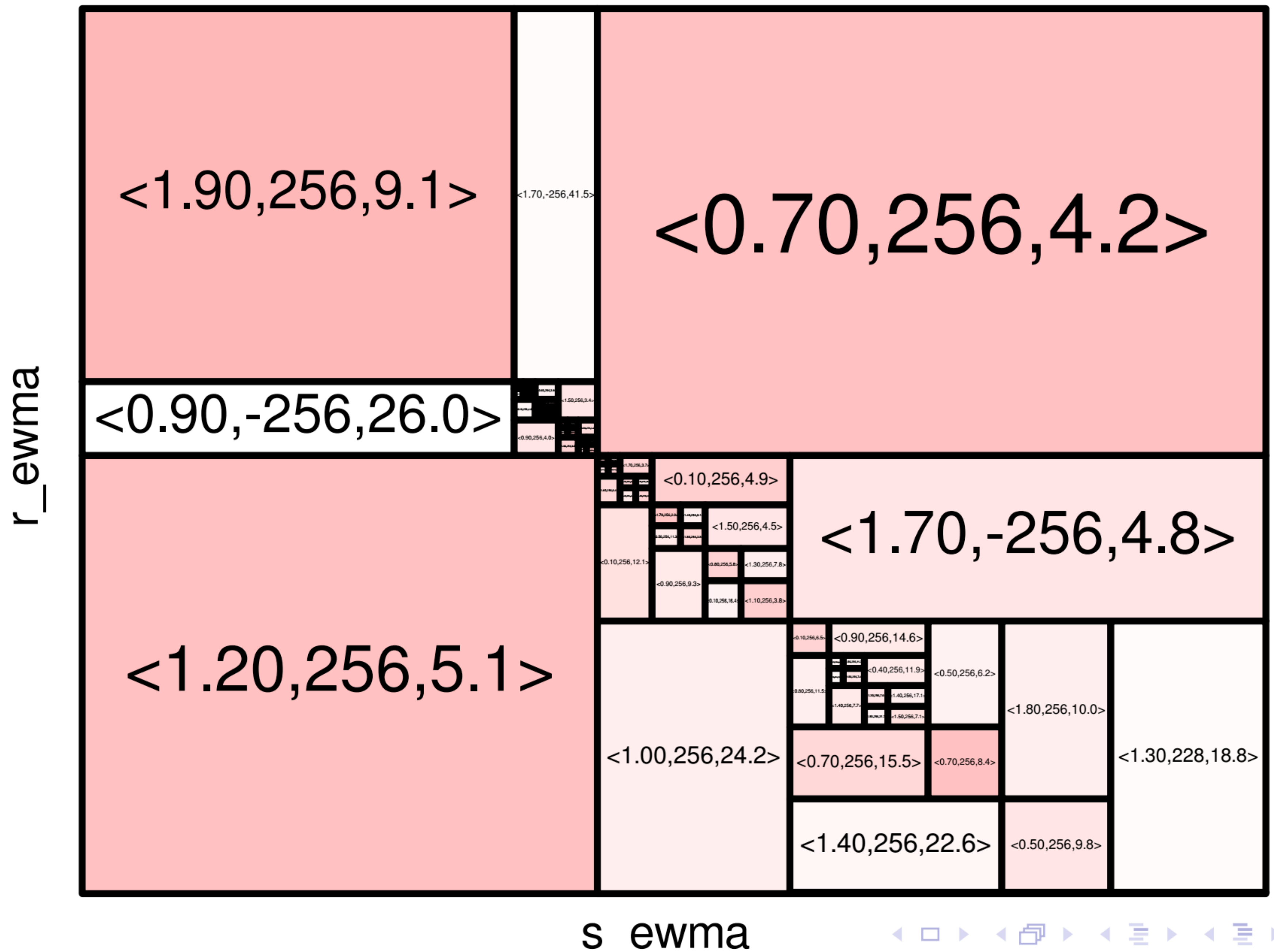


Optimize for each sub actions

Split the most used rule

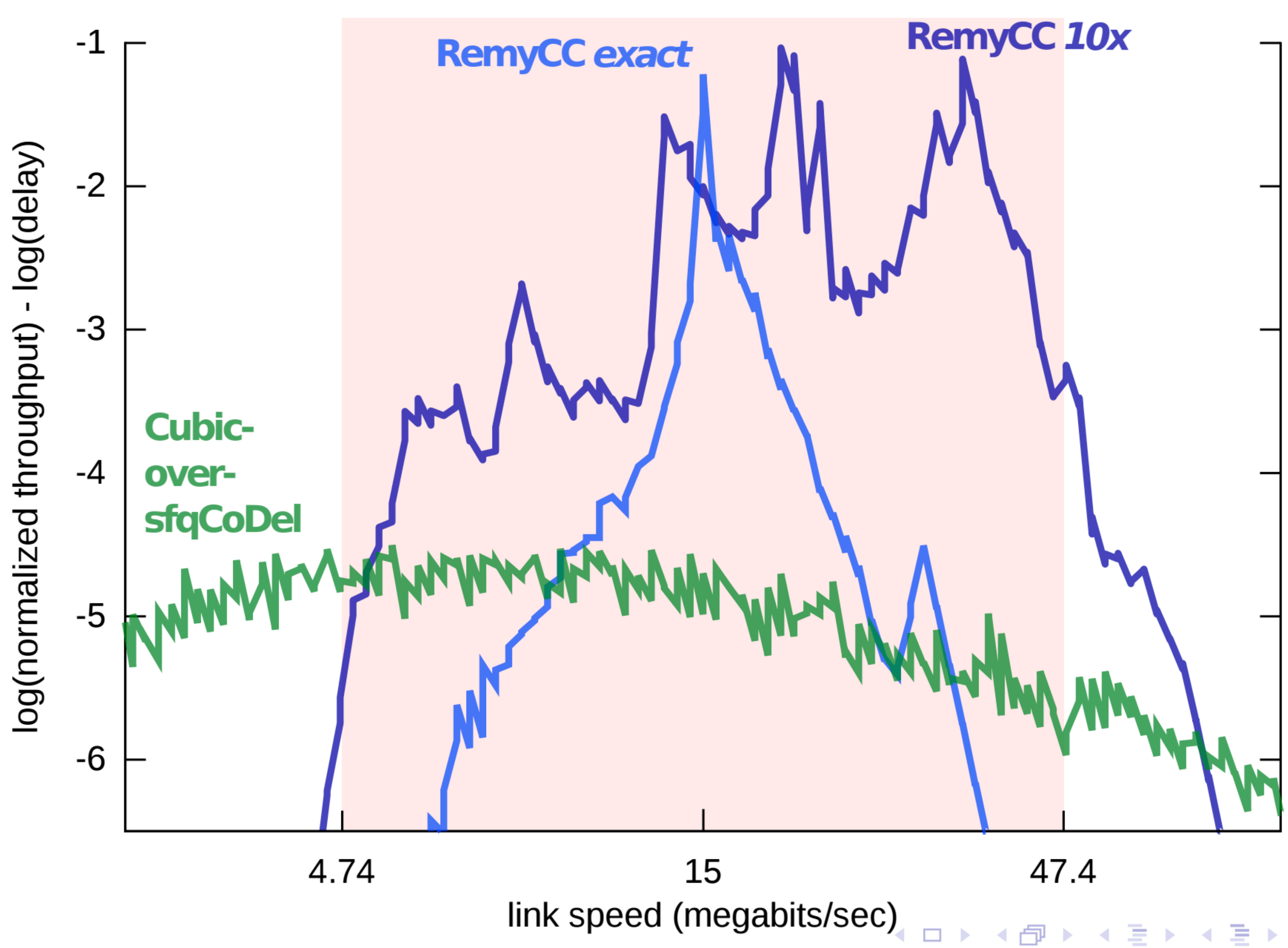


Iterate



<1.90,256,9.1>
<1.70,-256,41.5>
<0.70,256,4.2>
<0.90,-256,26.0>
<1.50,256,3.4>
<0.90,256,4.0>
<0.10,256,4.9>
<1.70,-256,4.8>
<1.50,256,4.5>
<0.10,256,12.1>
<1.30,256,7.8>
<0.90,256,9.3>
<1.10,256,3.8>
<1.20,256,5.1>
<0.90,256,14.6>
<0.40,256,11.9>
<0.50,256,6.2>
<1.40,256,17.1>
<1.50,256,7.1>
<1.80,256,10.0>
<1.00,256,24.2>
<0.70,256,15.5>
<0.70,256,8.4>
<1.30,228,18.8>
<1.40,256,22.6>
<0.50,256,9.8>
r_ewma
s_ewma

Discussion



RemyCC *exact*
RemyCC *10x*
Cubic-over-sfqCoDel
log(normalized throughput) - log(delay)
-1
-2
-3
-4
-5
-6
4.74
15
47.4
link speed (megabits/sec)

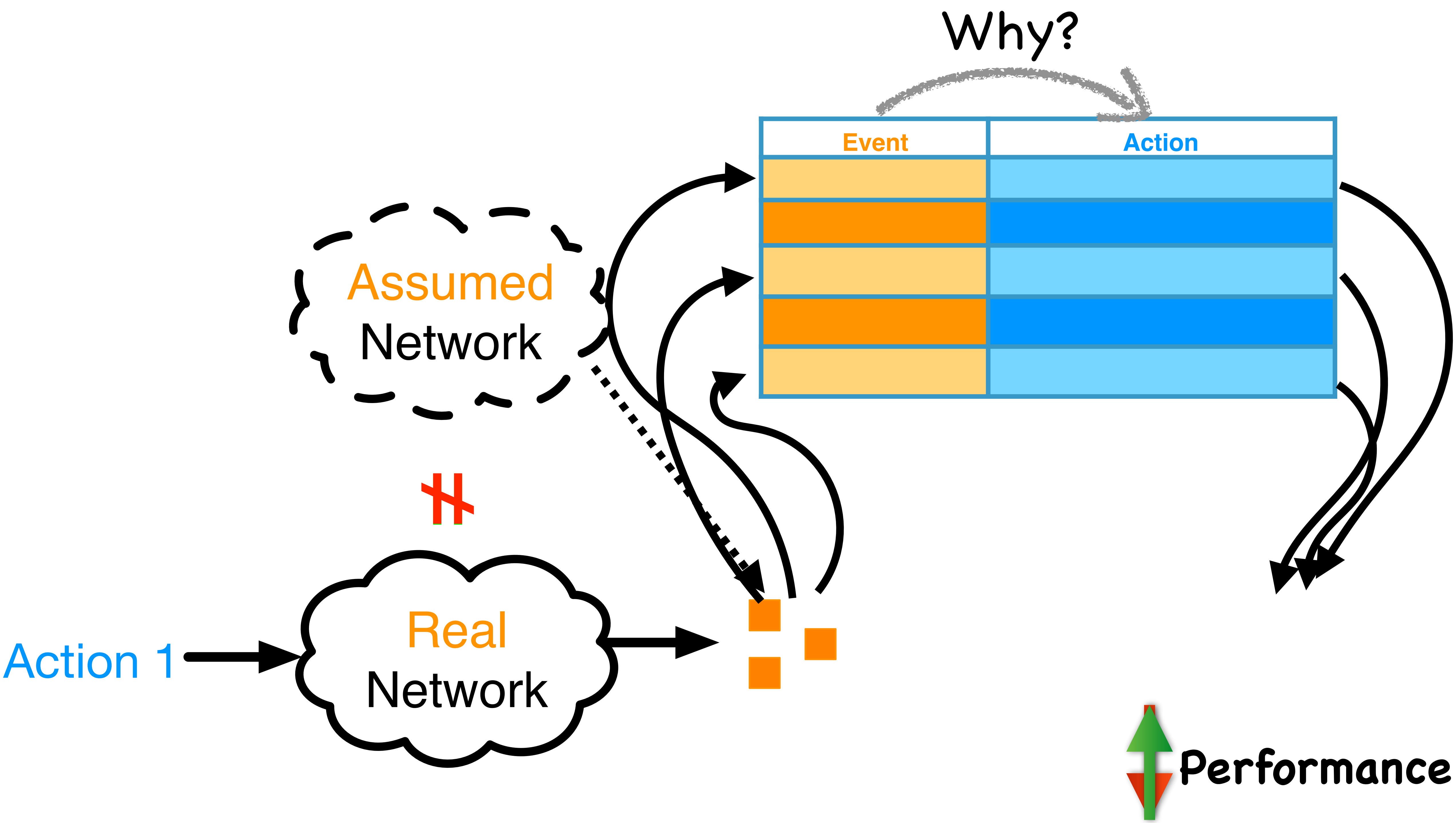
Learning Online

PCC: Re-architecting congestion control for consistent high performance
Dong, Li, Zarchy, Godfrey, Schapira
NSDI 2015

First, revisit TCP's architecture…

Hardwired Mapping

	Event	Action
Reno		
Scalable		
CUBIC		
FAST		
HTCP		


Why?
Event
Action
Assumed
Network
≠
Action 1
Real
Network
Performance

Flow f sends at R

Event	Action
Pac…	~~Dec R a lot~~
	~~Maintain R~~
	~~Increase R~~

No event-control mapping optimal for all network scenarios

Network

f causes most congestion

…fer

other high rate flow causing congestion

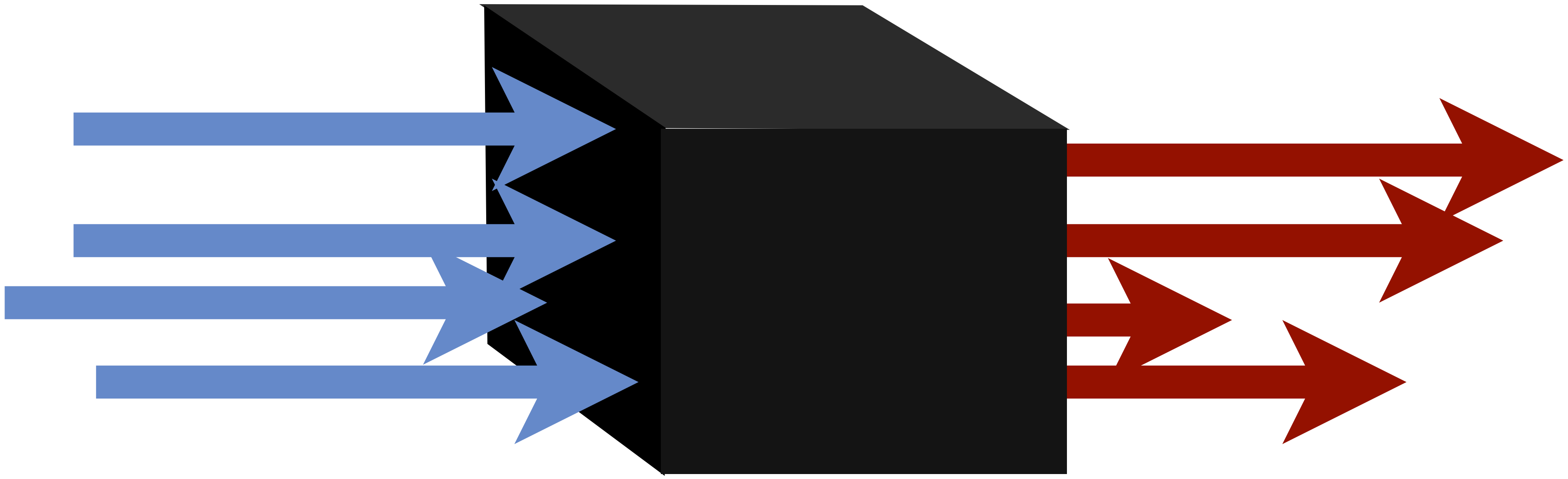
loss is random

PCC

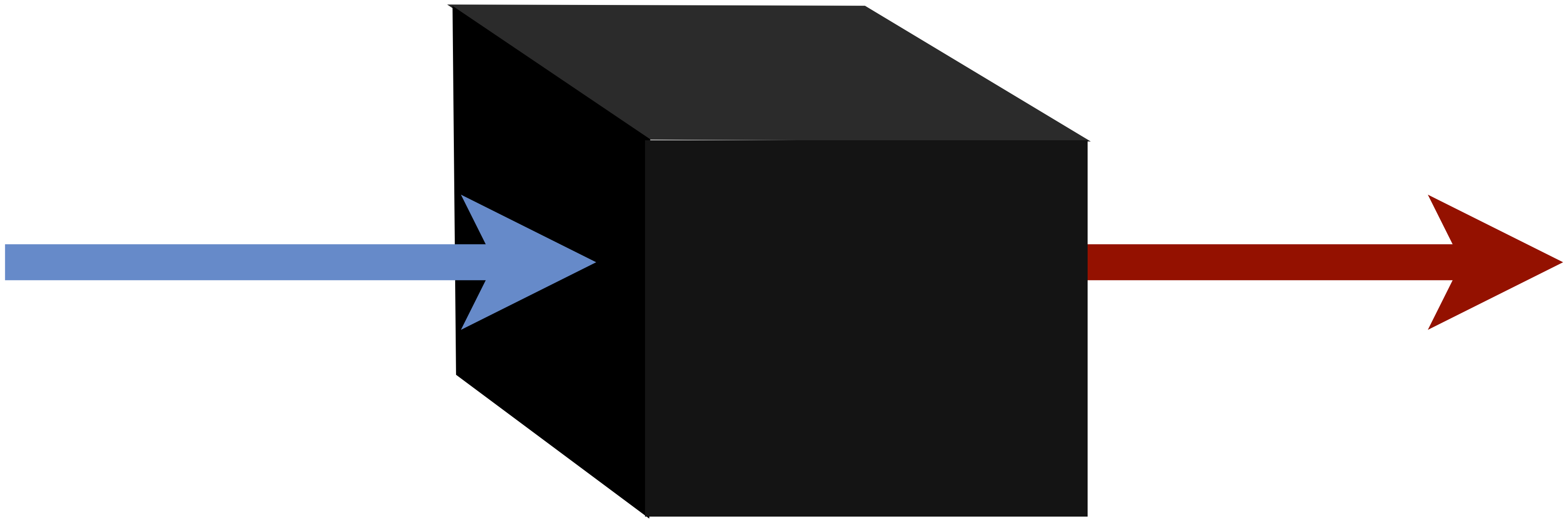
[Dong et al., NSDI'15]

(adapted from Dong's slides)

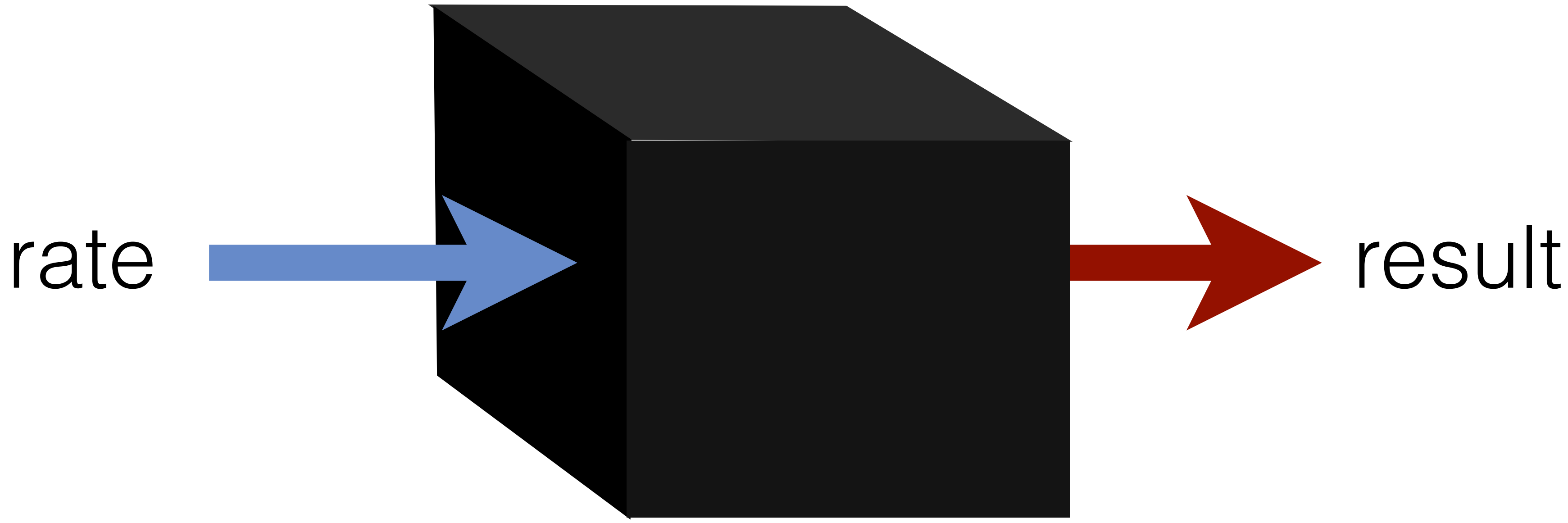
What is the right rate to send?



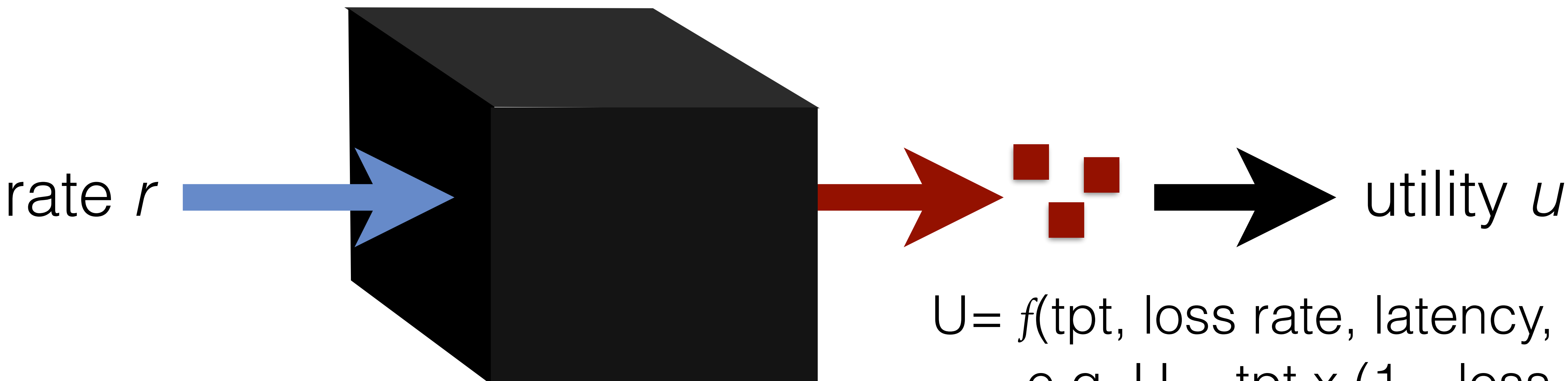
What is the right rate to send?



What is the right rate to send?



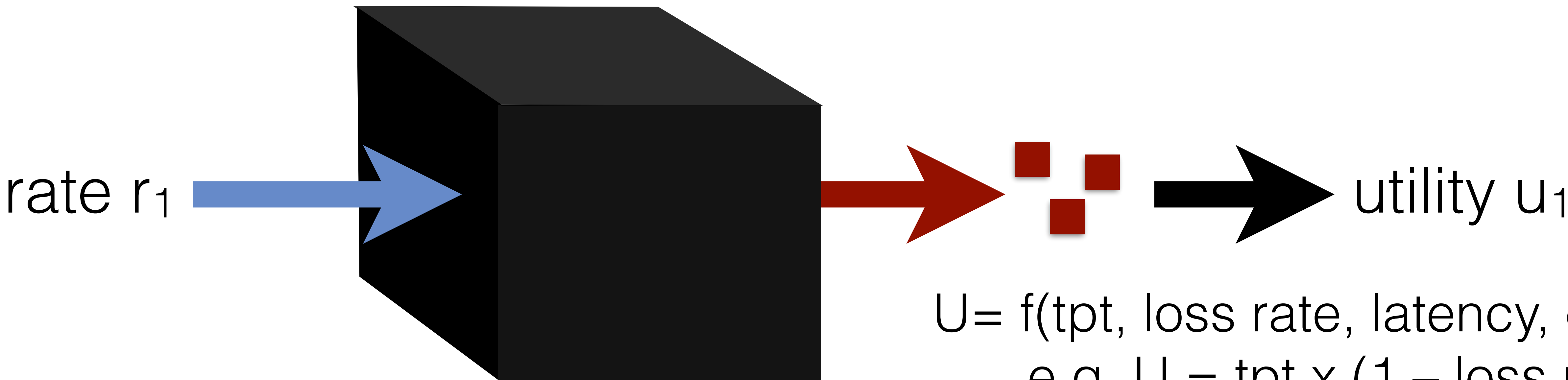
What is the right rate to send?



U= *f*(tpt, loss rate, latency, etc.)

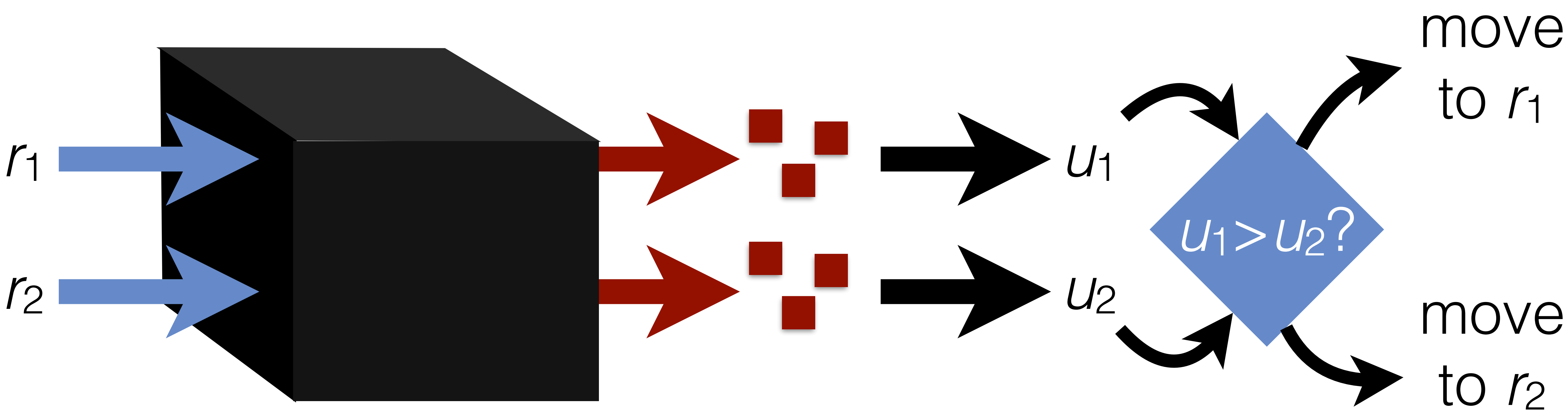
e.g. U = tpt x (1 – loss rate)

What is the right rate to send?



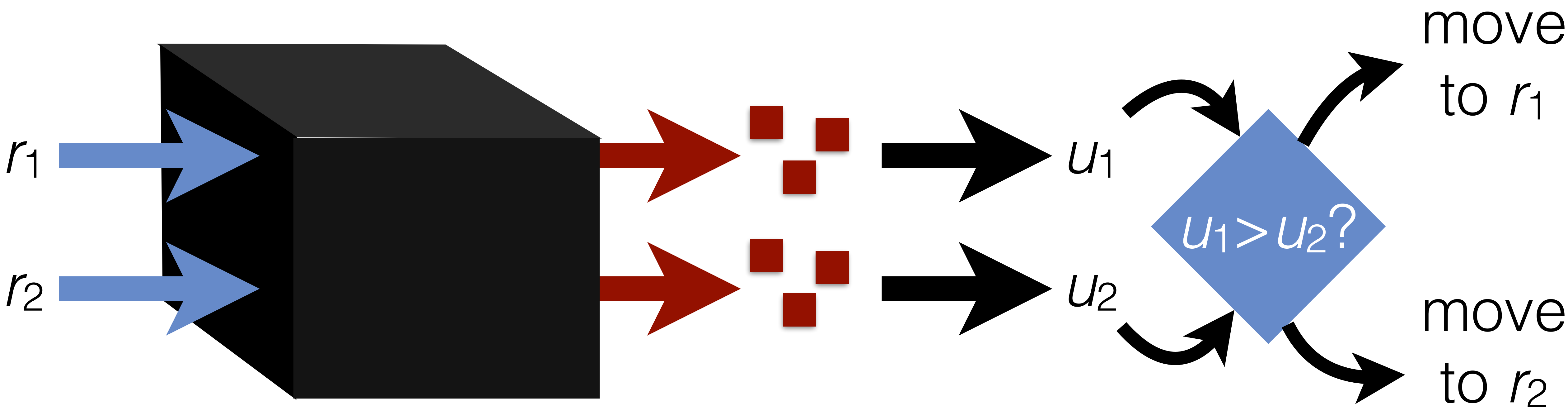
U= f(tpt, loss rate, latency, etc.)
e.g. U = tpt x (1 – loss rate)

No matter how complex the network,
rate r —> utility u



r1
r2
u1
u2
u1>u2?
move to r1
move to r2

Performance-oriented Congestion Control



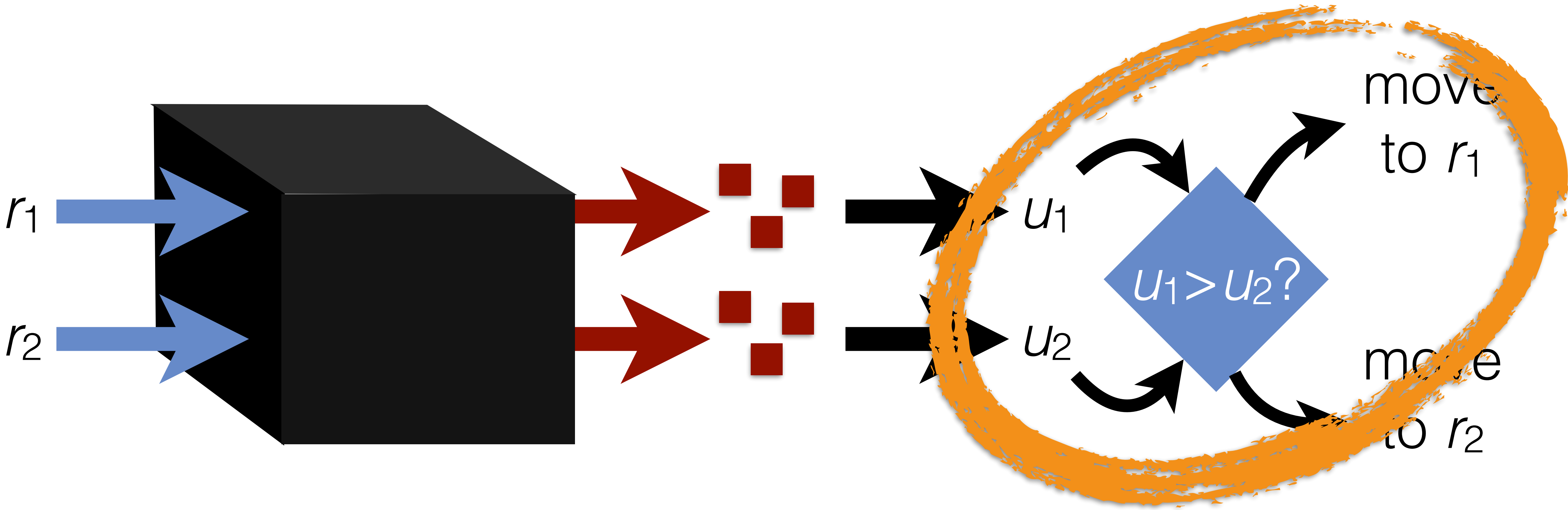
yields

Observe real performance

Control based on empirical evidence

Consistent high performance

Where is Congestion Control?



Selfishly maximizing utility
=> non-cooperative game

Do we converge to a fair Nash equilibrium?

Congestion Control is Game Theory

Find a utility function that:

- has an unique and nice NE under FIFO queue
- expresses a generic data transmission objective
- maintains consistent high performance

$$u_i(x) = T_i \qquad - x_i * L_i$$

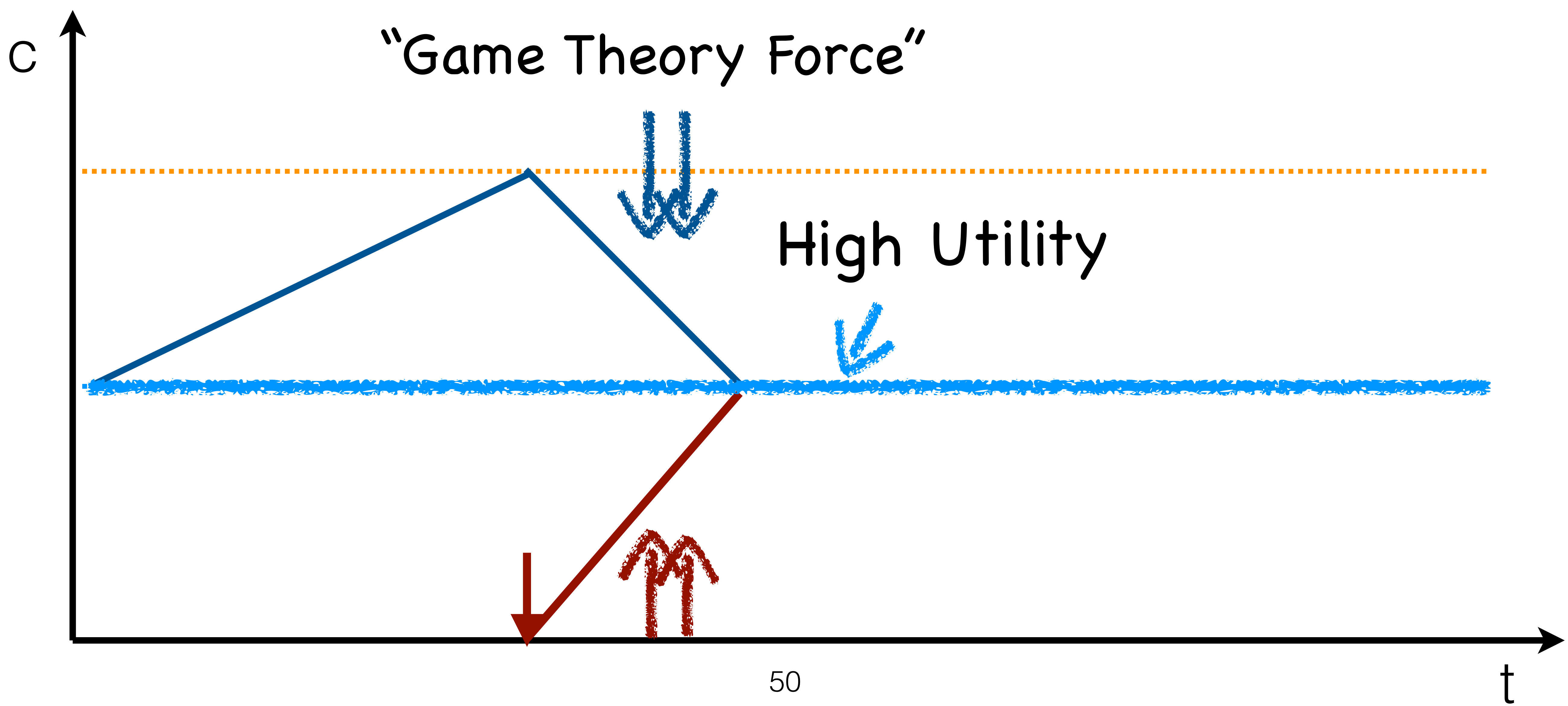
x_i is sending rate

L_i is the observed loss rate

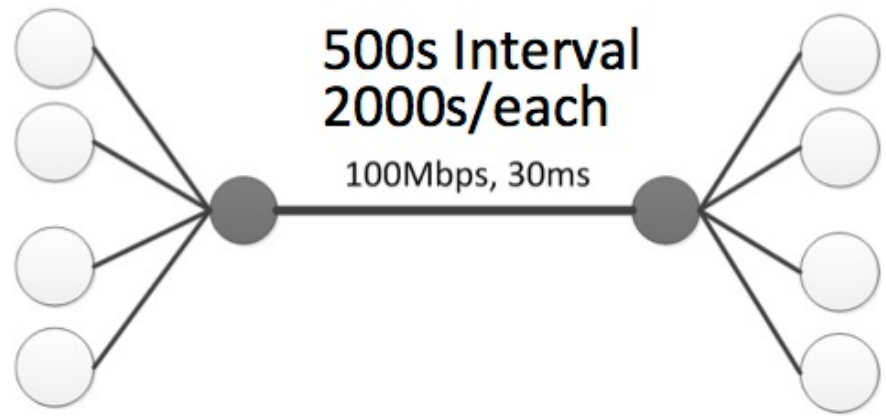
$T_i = x_i * (1 - L_i)$ is throughput

PCC Dynamics

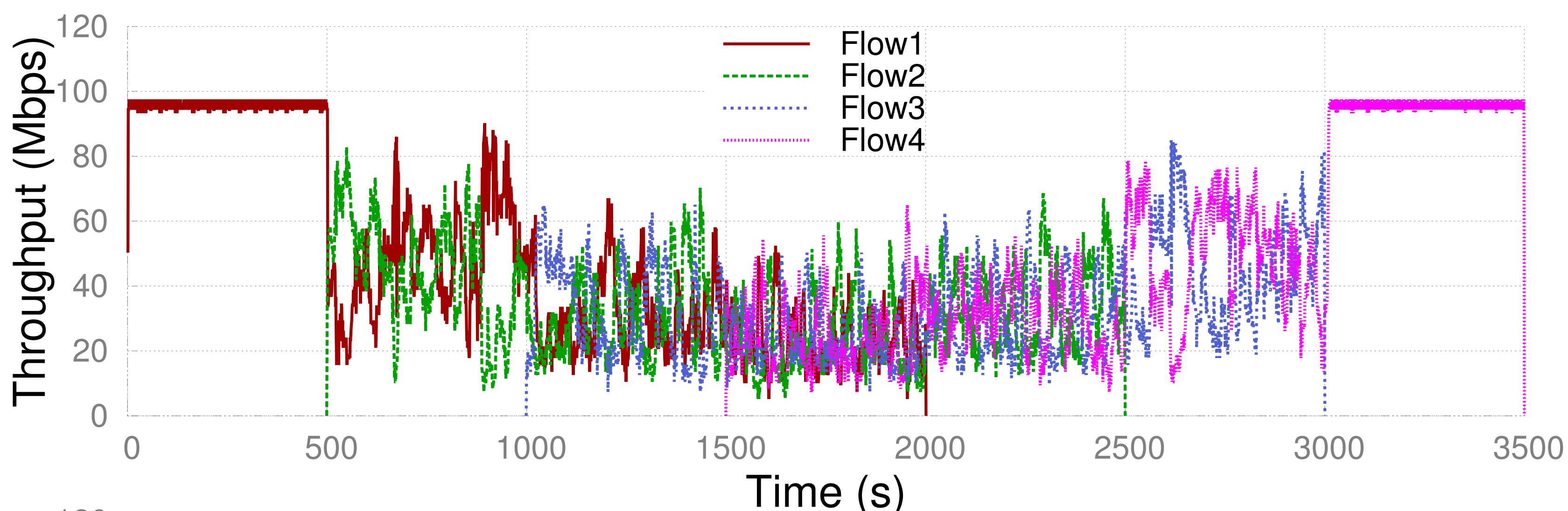
PCC senders' utility functions make them react differently to the same network.



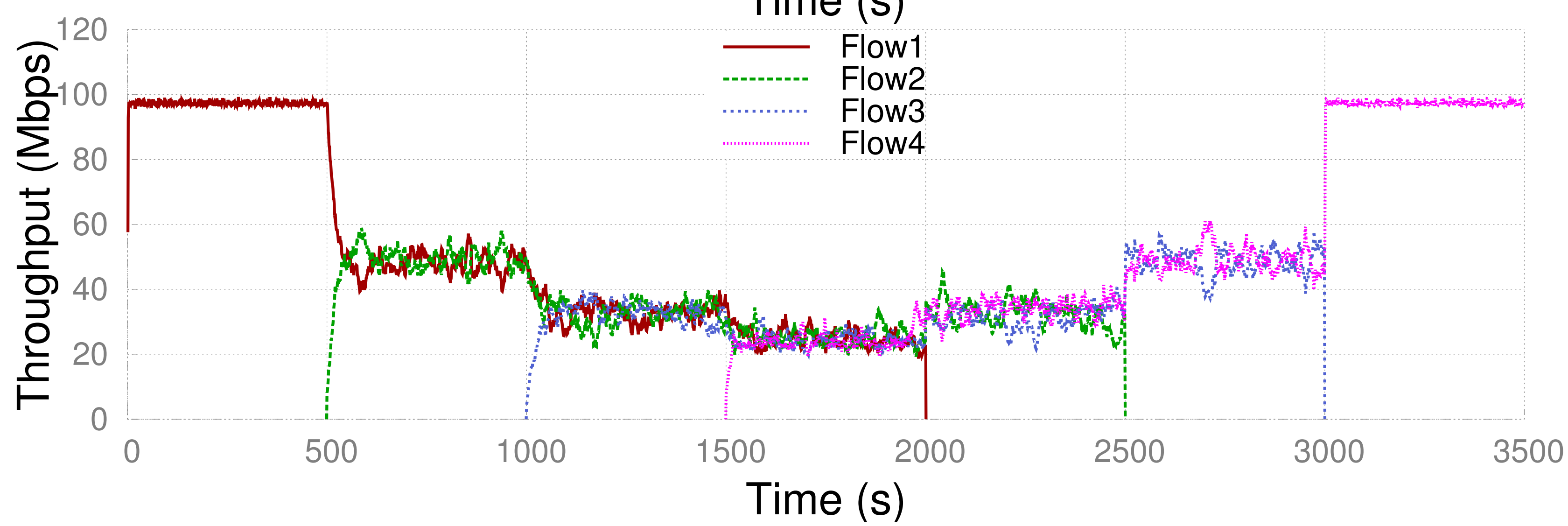
Convergence



TCP



PCC



Announcements

Feedback

- Reviews: sent
- Project: will send comments within a week

Next time: dive into hardware

- How to build a fast router