Randomized Network Algorithms: An Overview and Recent Results

Balaji Prabhakar
Departments of EE and CS
Stanford University

Network algorithms

- Algorithms implemented in networks, e.g. in
  - switches/routers
    - scheduling algorithms
    - routing lookup
    - packet classification
    - security
  - memory/buffer managers
    - maintaining statistics
    - active queue management
    - bandwidth partitioning
  - load balancers
  - web caches
    - eviction schemes
    - placement of caches in a network

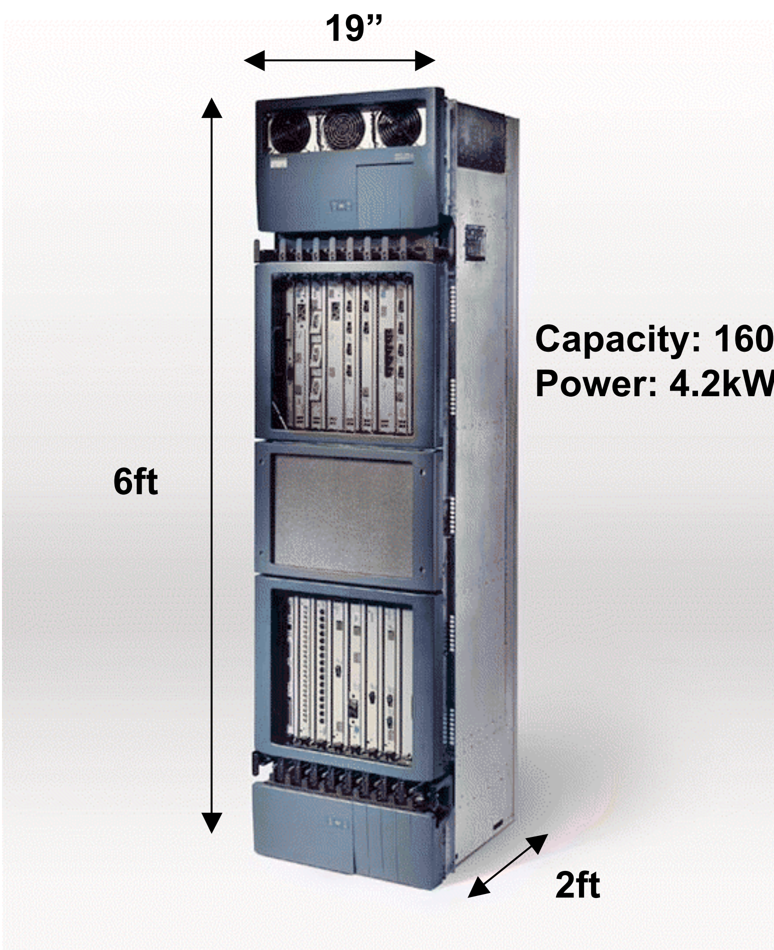
Network algorithms: challenges

- Time constraint: Need to make complicated decisions very quickly
  - line speeds in the Internet core 10Gbps (40Gbps in the near future)
    - i.e. packets arrive roughly every 40ns
  - large number of
    - distinct flows in the Internet core
    - requests arriving per sec at large server farms

- But, there are limited computational resources
  - due to rigid space and heat dissipation constraints

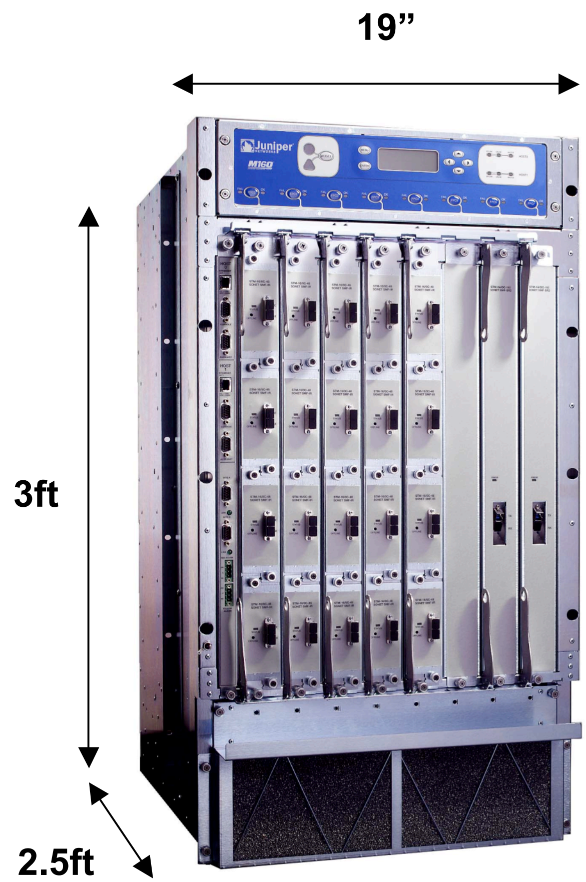
- Algorithms need to be very simple so as to be implementable
  - but simple algorithms may perform poorly, if not well-designed

IP Routers



Capacity: 160Gb/s
Power: 4.2kW

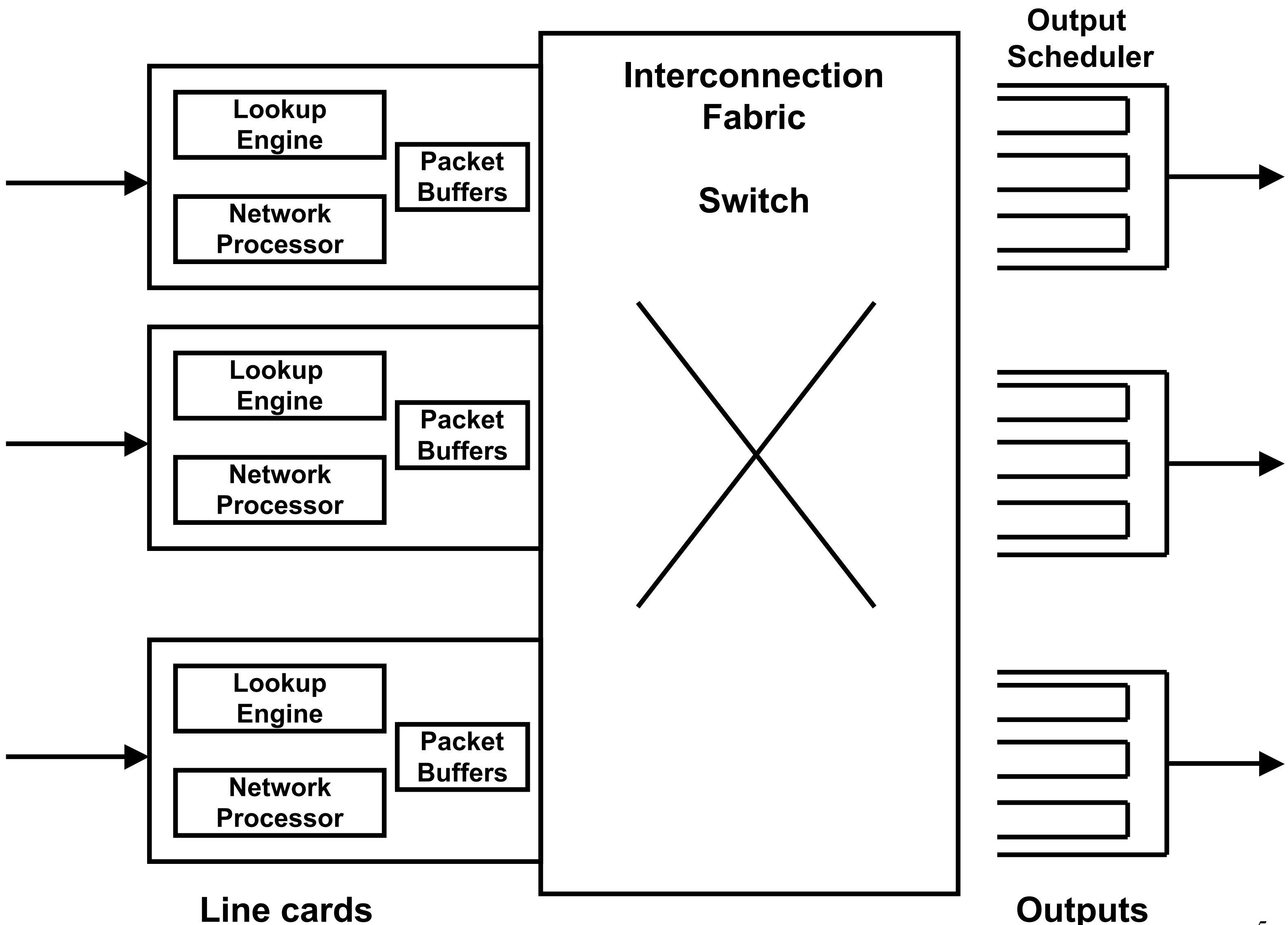
Cisco GSR 12416



Capacity: 80Gb/s
Power: 2.6kW

Juniper M160

A Detailed Sketch



Designing network algorithms

- I will illustrate the use of two ideas for designing efficient network algorithms

  1. Randomization

     - ❑ base decisions upon a small, randomly chosen sample of the state/input, instead of the complete state/input

  2. Power law distributions

     - ❑ Internet packet traces exhibit power law distributions:
       80% of the packets belong to 20% of the flows;
       i.e. most flows are small (mice), most work is brought by a few elephants
     - ❑ identifying the large flows cheaply can significantly simplify the implementation

- Two applications
  - switch scheduling
  - bandwidth partitioning

Randomization: An illustrative example

- Find the youngest person from a population of 1 billion

- Deterministic algorithm: linear search
  - has a complexity of 1 billion

- A randomized version: find the youngest of 30 randomly chosen people
  - has a complexity of 30

- Performance
  - linear search will find the absolute youngest person (rank = 1)
  - if R is the person found by randomized algorithm, we can say

$$P(R \text{ has rank } < 100 \text{ million}) > 1 - \left(\frac{9}{10}\right)^{30} \approx 0.95$$

  - ➢ thus, we can say that the performance of the randomized algorithm is good with a high probability

Randomizing iterative schemes

- Often, we want to perform some operation iteratively
- Example: find the youngest person each year
- Say in 2007 you choose 30 people at random
  - and store the identity of the youngest person in memory
  - in 2008 you choose 29 new people at random
  - let R be the youngest person from these 29 + 1 = 30 people

$$P(R \text{ has rank } < 100 \text{ million}) > 1 - \left(\frac{9}{10}\right)^{58}$$

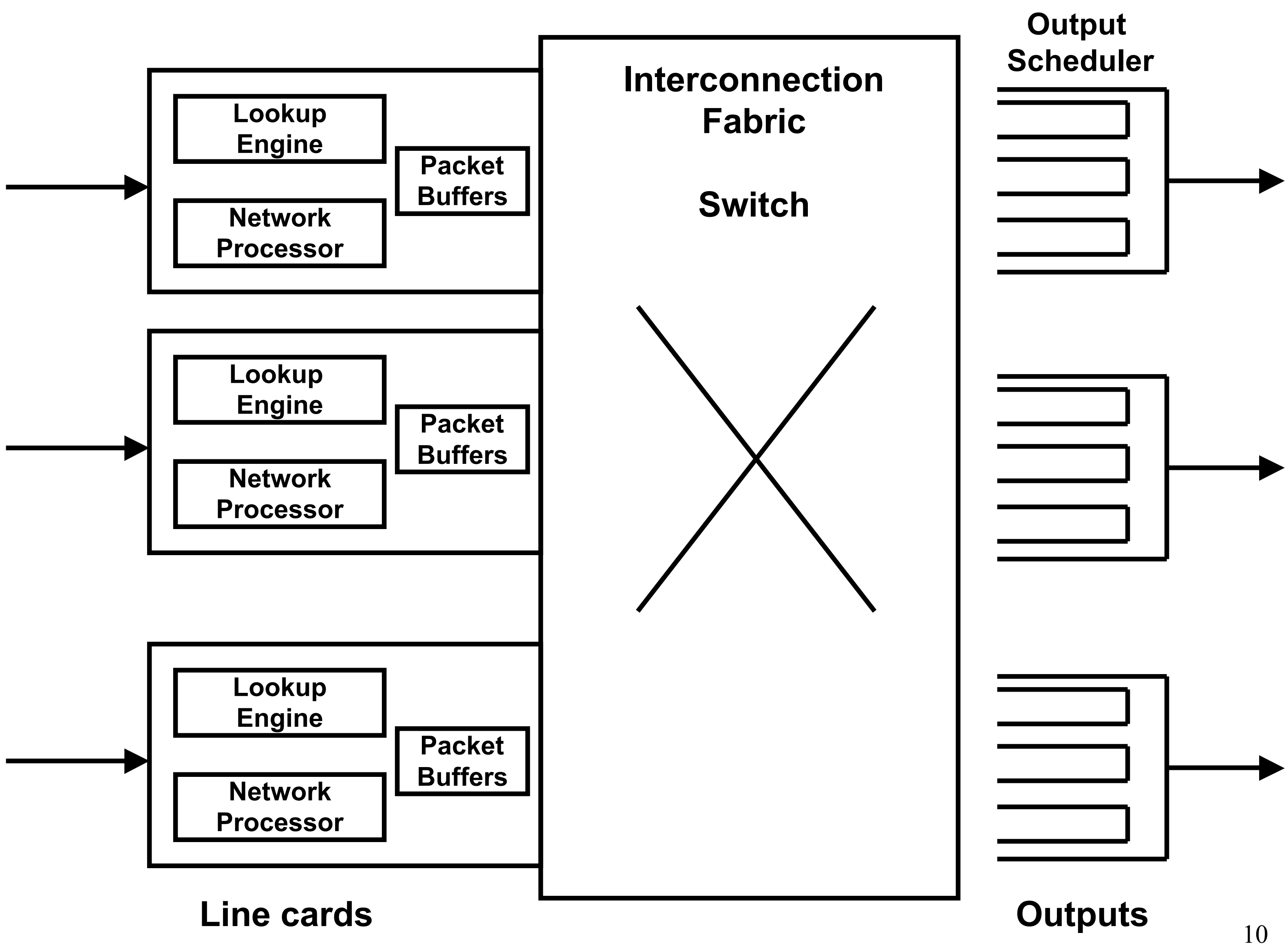
  - or

$$P(R \text{ has rank } < 50 \text{ million}) > 1 - \left(\frac{9}{10}\right)^{30}$$

Randomized switch scheduling algorithms

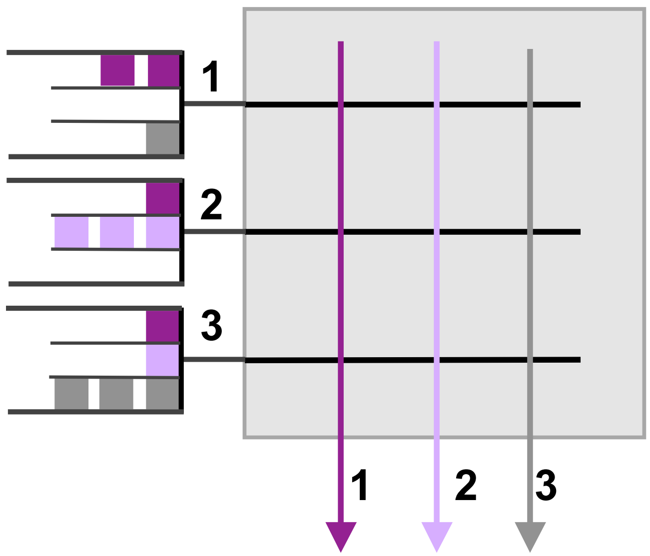
joint work with Paolo Giaccone and Devavrat Shah

A Detailed Sketch



Input queued switch

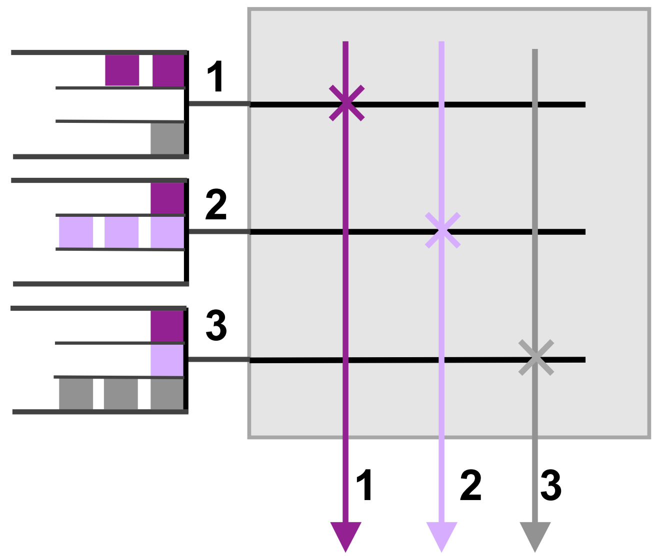
Crossbar fabric



- Crossbar constraints
  - each input can connect to at most one output
  - each output can connect to at most one input

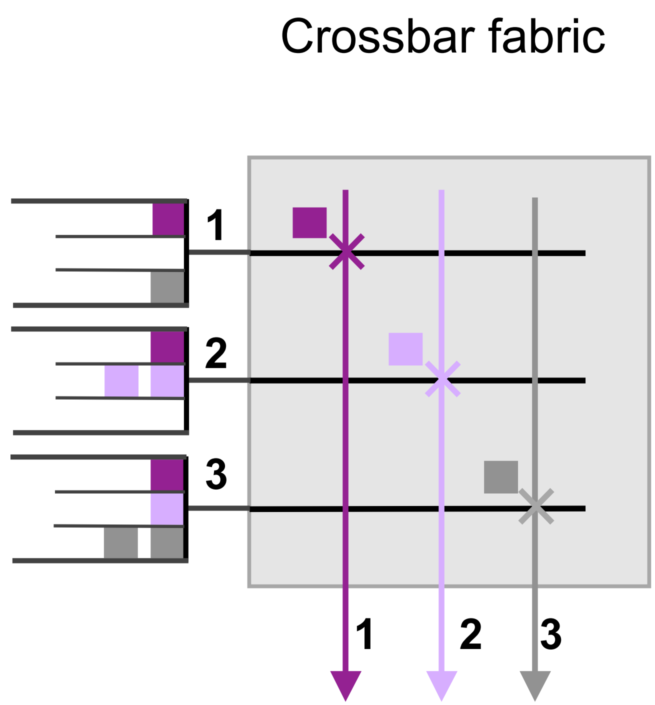
Switch scheduling

Crossbar fabric



- Crossbar constraints
  - each input can connect to at most one output
  - each output can connect to at most one input

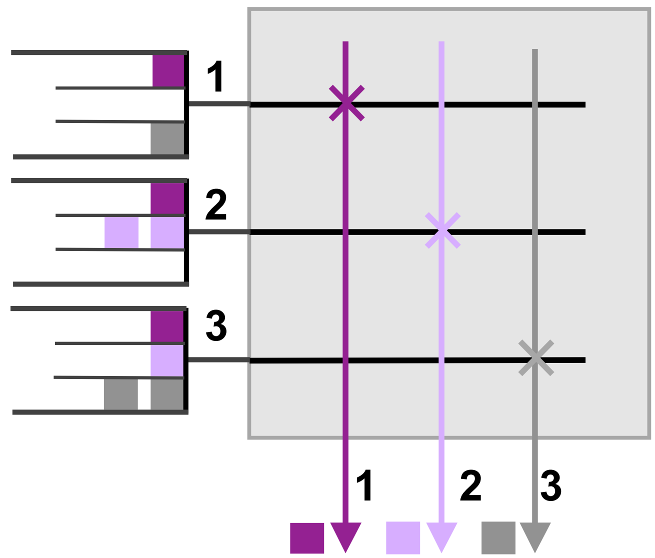
Switch scheduling



- Crossbar constraints
  - each input can connect to at most one output
  - each output can connect to at most one input

Switch scheduling

Crossbar fabric



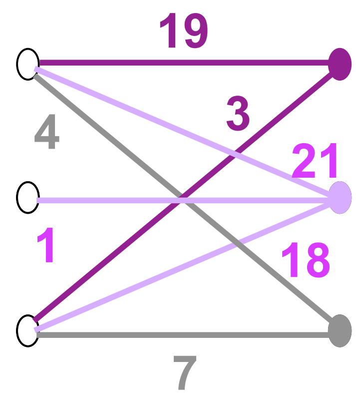
- Crossbar constraints
  - each input can connect to at most one output
  - each output can connect to at most one input

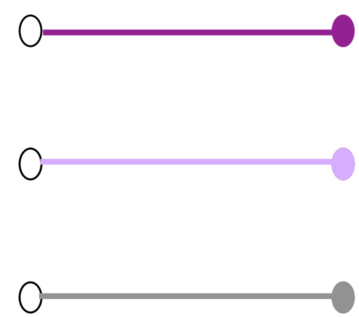
Performance measures

- Throughput
  - an algorithm is *stable* (or delivers 100% throughput) if for any admissible arrival, the average backlog is bounded.

- Average delay or average backlog (queue-size)

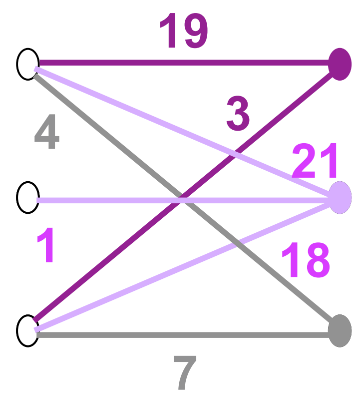
Scheduling: Bipartite graph matching

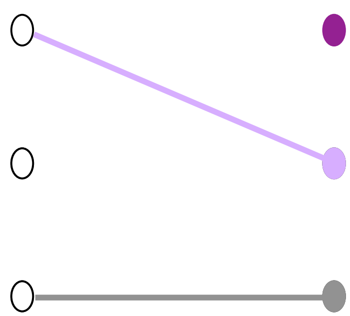




Schedule or Matching

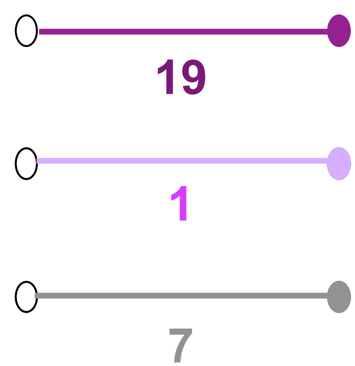
Scheduling algorithms





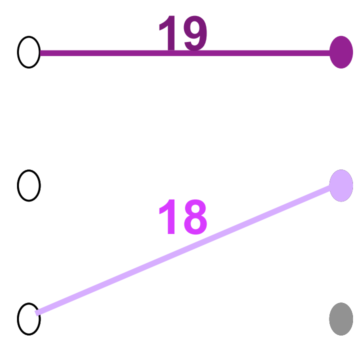
Practical
Maximal Matchings

→ Not stable



Max Size Matching

→ Not stable
(McKeown-Ananthram-Walrand 96)



Max Wt Matching

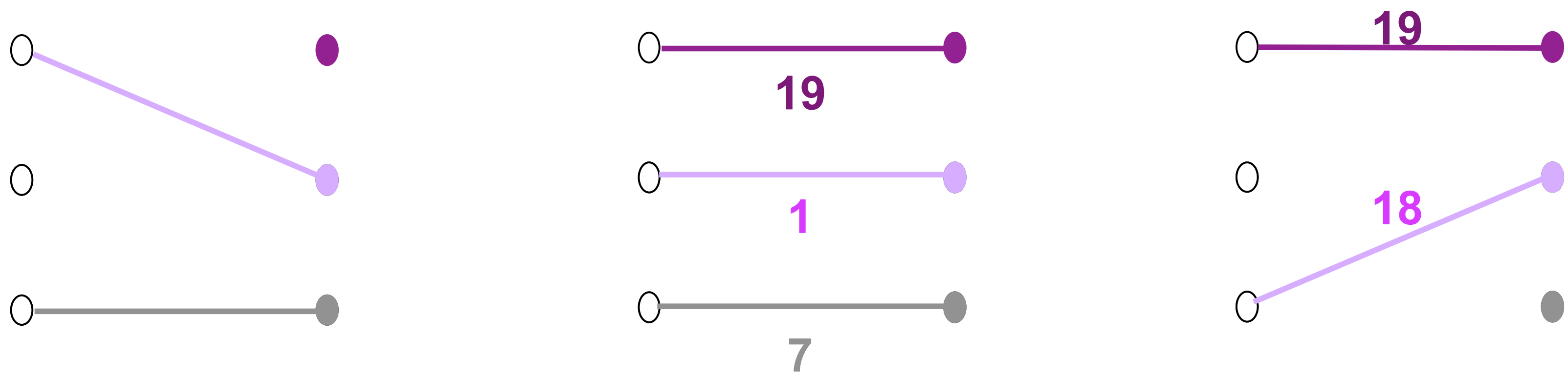
→ Stable
(Tassiulas-Ephremides 92,
McKeown et. al. 96,
Dai-Prabhakar 00)

The Maximum Weight Matching Algorithm

- MWM: performance
  - throughput: stable (Tassiulas-Ephremides 92; McKeown et al 96; Dai-Prabhakar 00)
  - backlogs: very low on average (Leonardi et al 01; Shah-Kopikare 02)

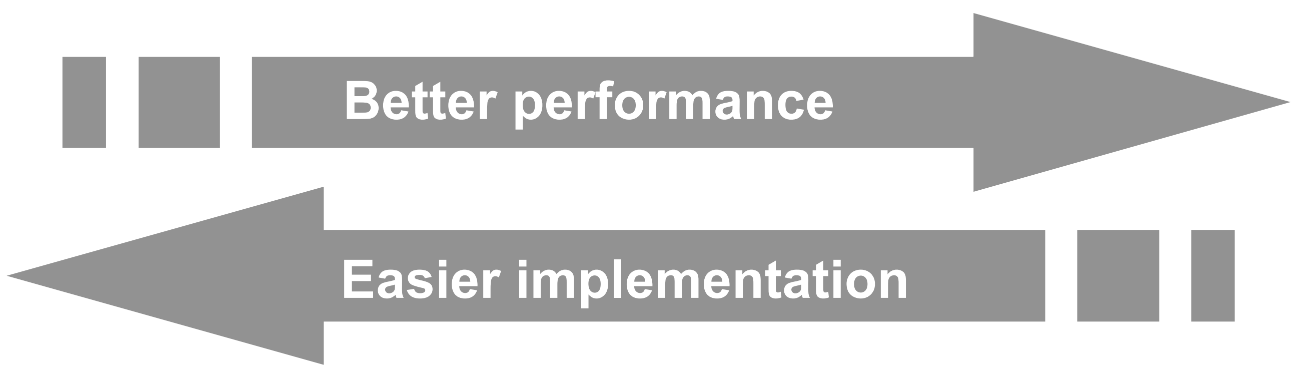
- MWM: implementation
  - has cubic worst-case complexity
    (approx. 27,000 iterations for a 30-port switch)
  - MWM algorithms involve backtracking:
    i.e. edges laid down in one iteration may be removed in a subsequent iteration
  - ➢ algorithm not amenable to pipelining

Switch algorithms



Maximal matching | **Max Size Matching** | **Max Wt Matching**

Not stable | Not stable | Stable and low backlogs



Randomized approximation to MWM

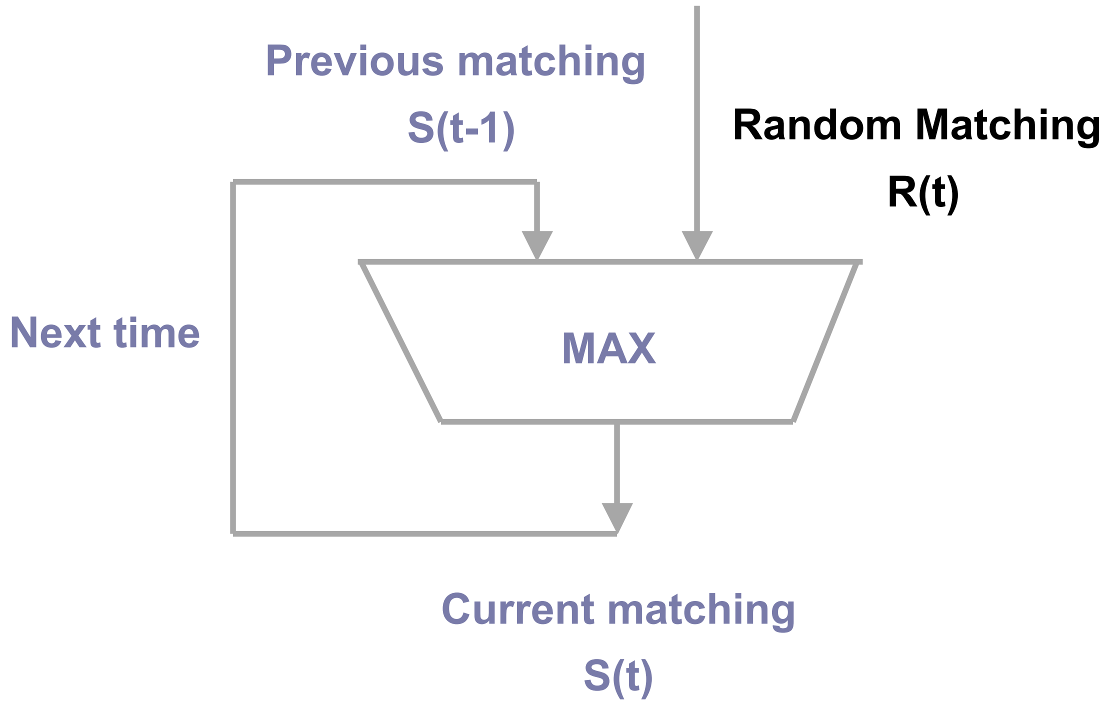
- Consider the following randomized approximation:
  At every time
  - sample d matchings independently and uniformly
  - use the heaviest of these d matchings to schedule packets

- Ideally we would like to use a small value of d. However,…

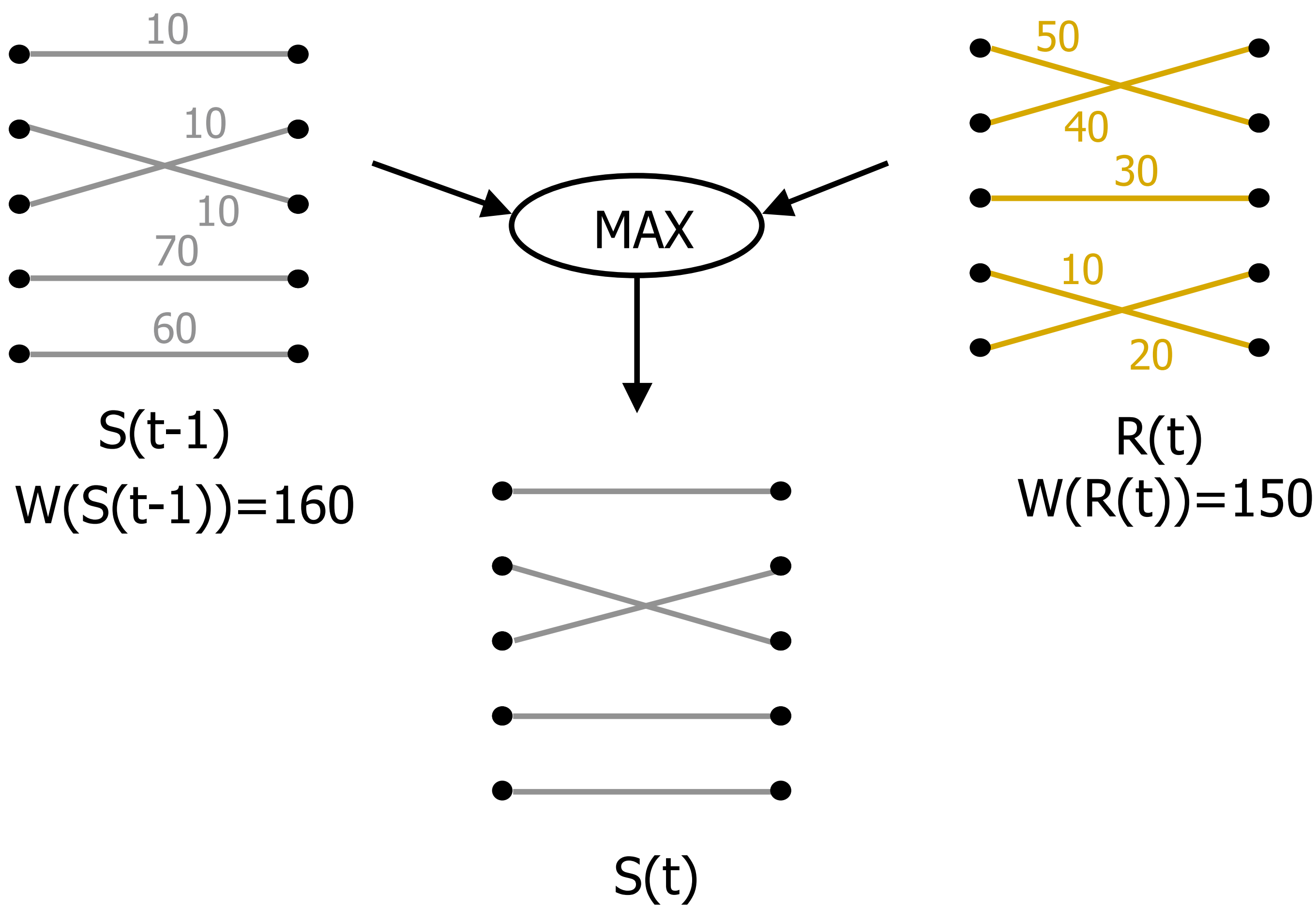
Theorem. This algorithm is not stable even when d = N. In fact, when d = N, the throughput is at most $1 - \frac{1}{e} \approx 63\%$

(Giaccone-Prabhakar-Shah 02)

Tassiulas' algorithm



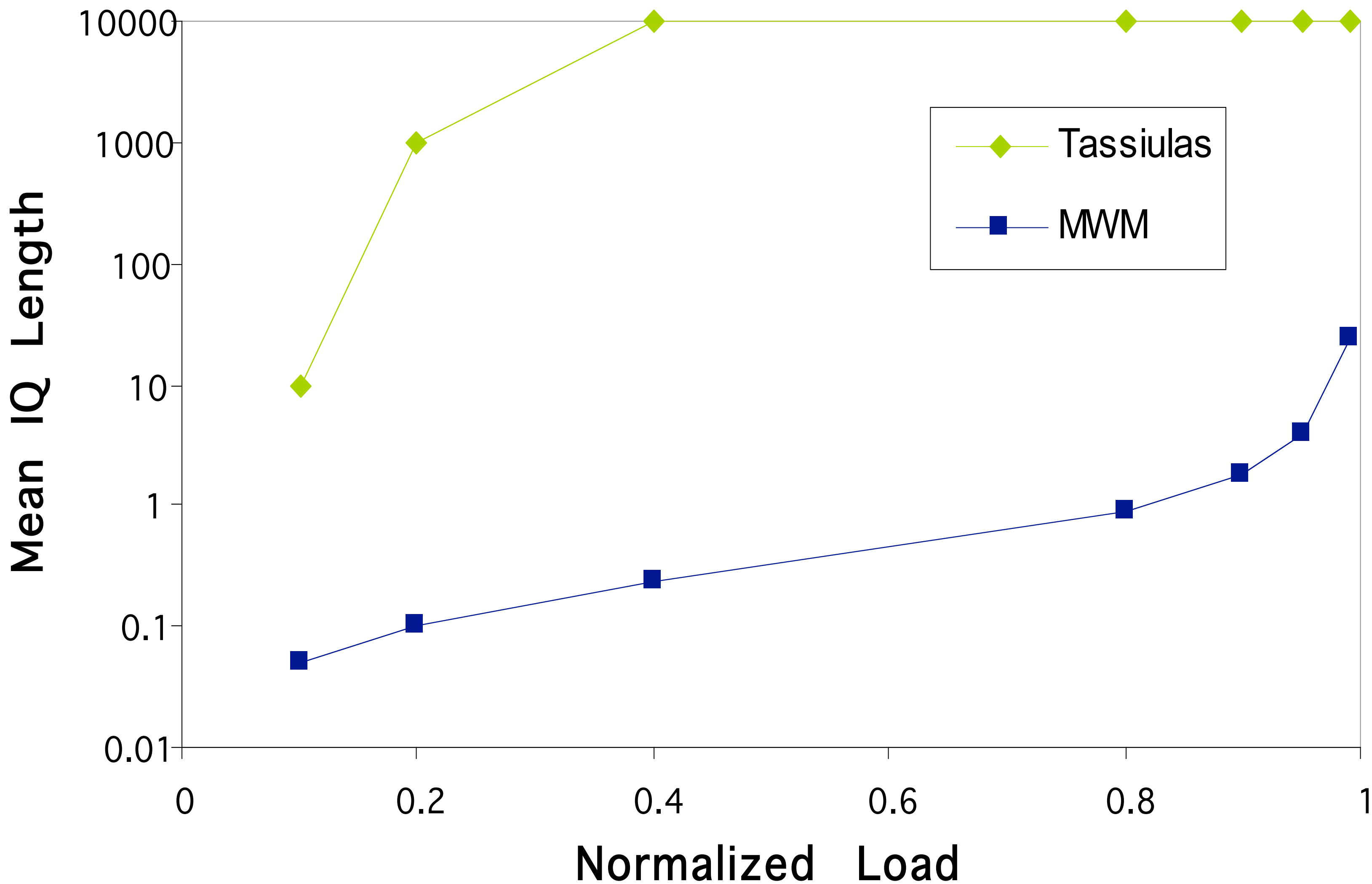
Tassiulas' algorithm: Use past sample



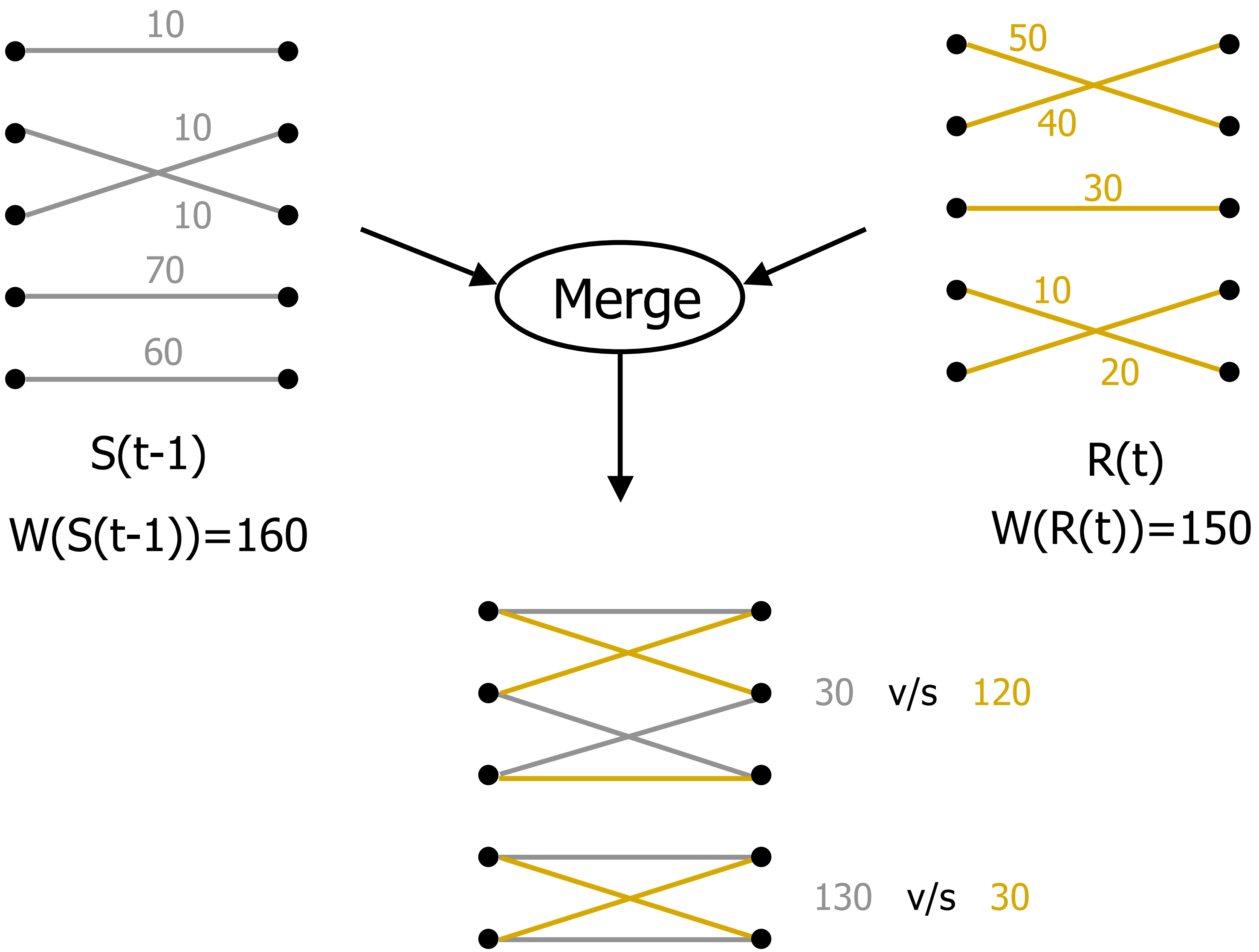
Performance of Tassiulas' algorithm

Theorem (Tassiulas 98): The above scheme is stable under any admissible Bernoulli IID inputs.

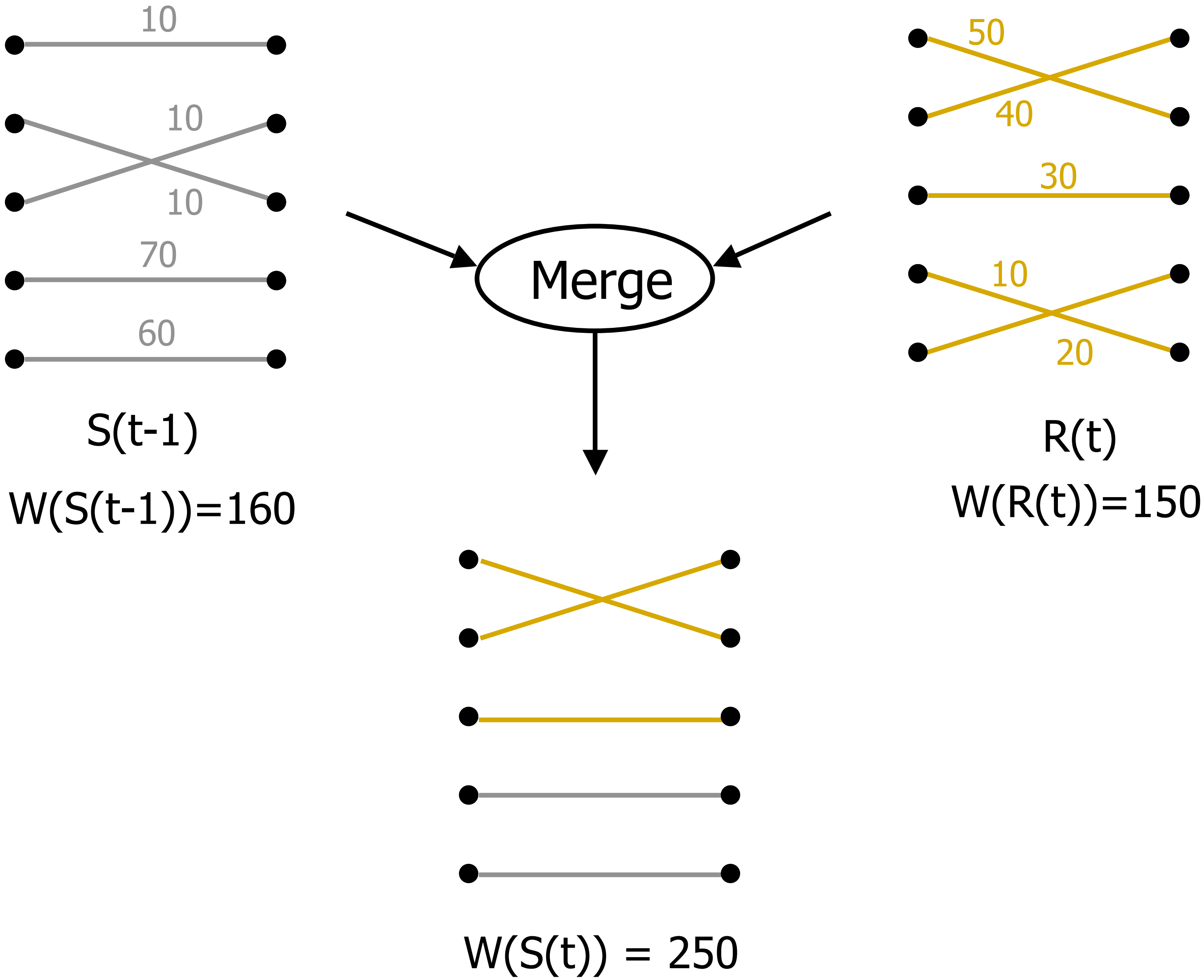
Backlogs under Tassiulas' algorithm



Reducing backlogs: the Merge operation



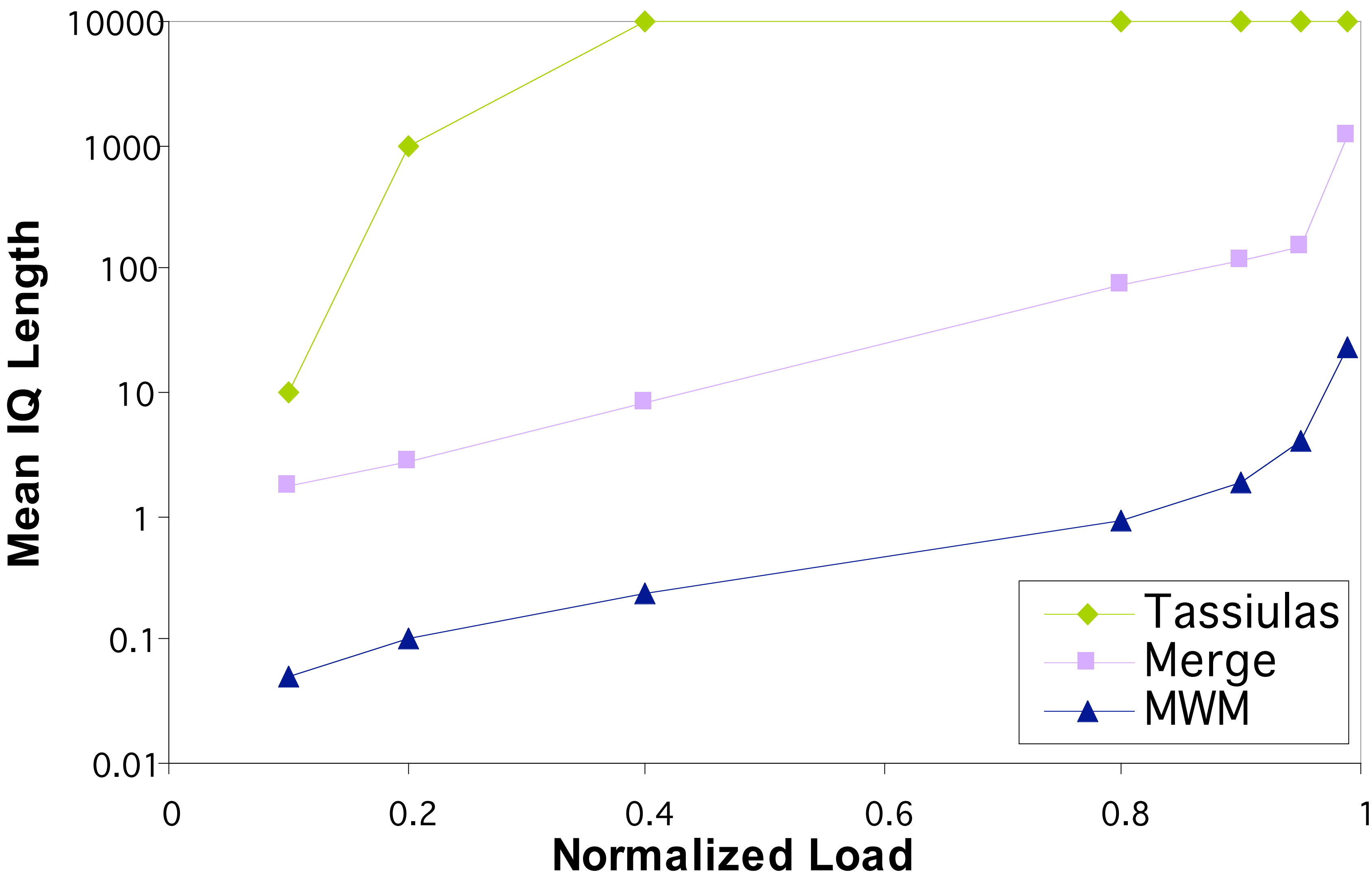
Reducing backlogs: the Merge operation



Performance of Merge algorithm

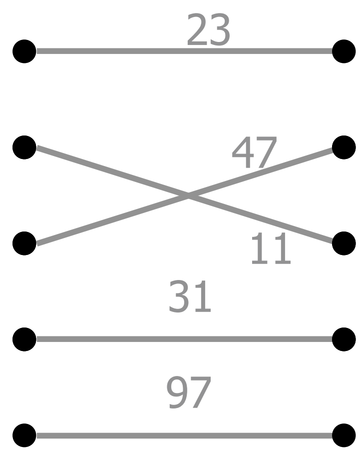
Theorem (GPS): The Merge scheme is stable under any admissible Bernoulli IID inputs.

Merge v/s Max



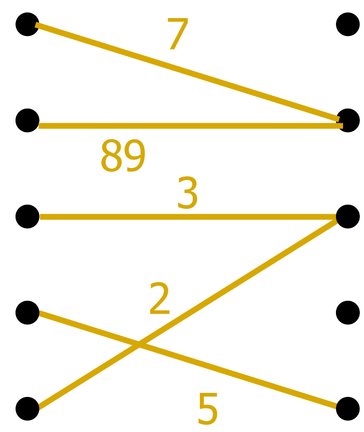
Mean IQ Length
10000
1000
100
10
1
0.1
0.01
0
0.2
0.4
0.6
0.8
1
Normalized Load
Tassiulas
Merge
MWM

Use arrival information: Serena



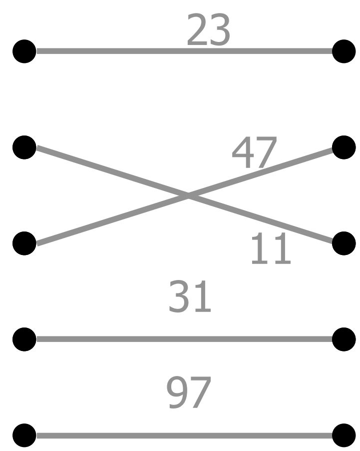
S(t-1)

W(S(t-1))=209



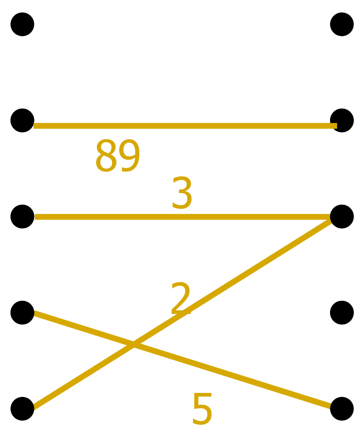
The arrival graph

Use arrival information: Serena



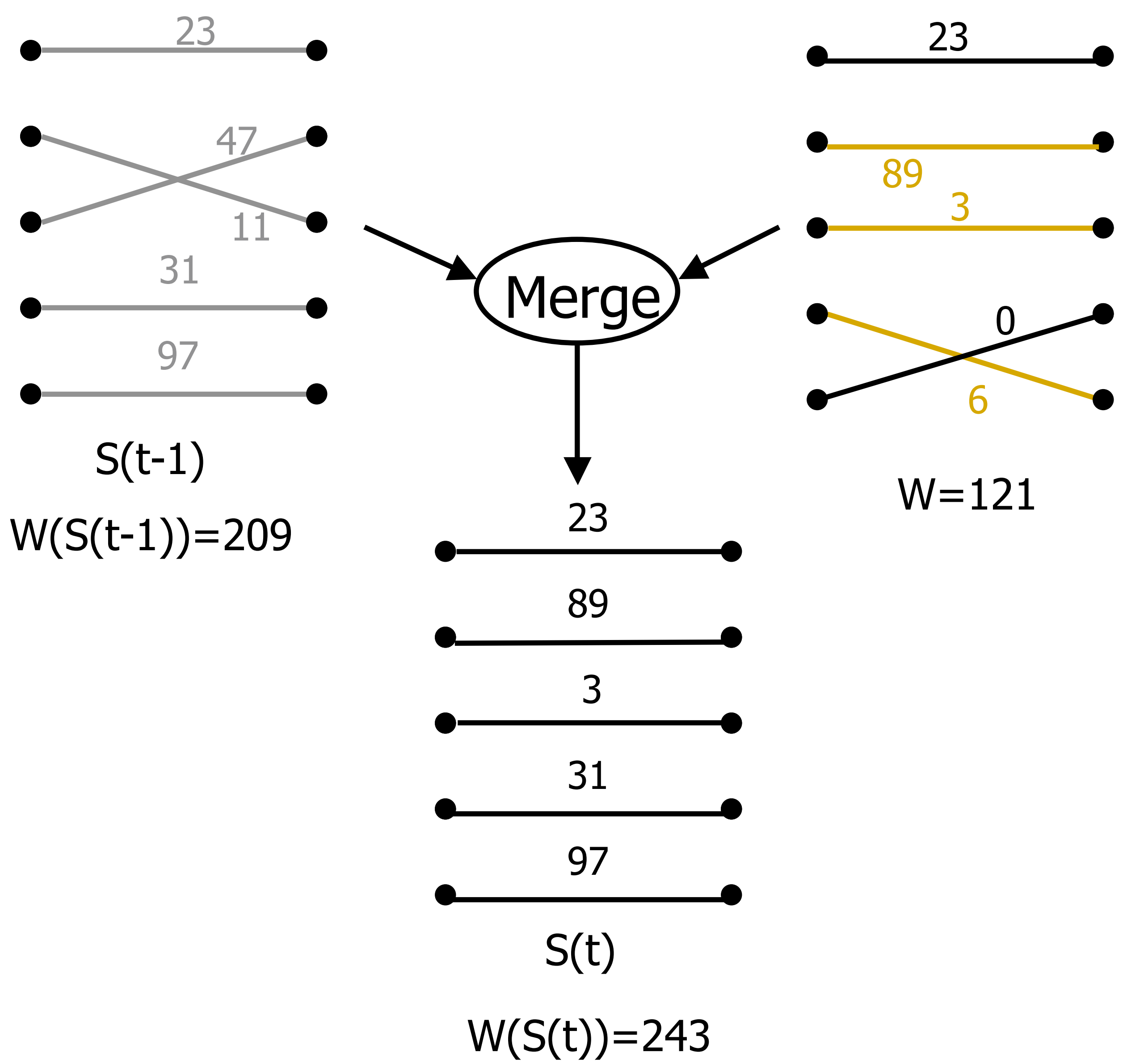
S(t-1)

W(S(t-1))=209



The arrival graph

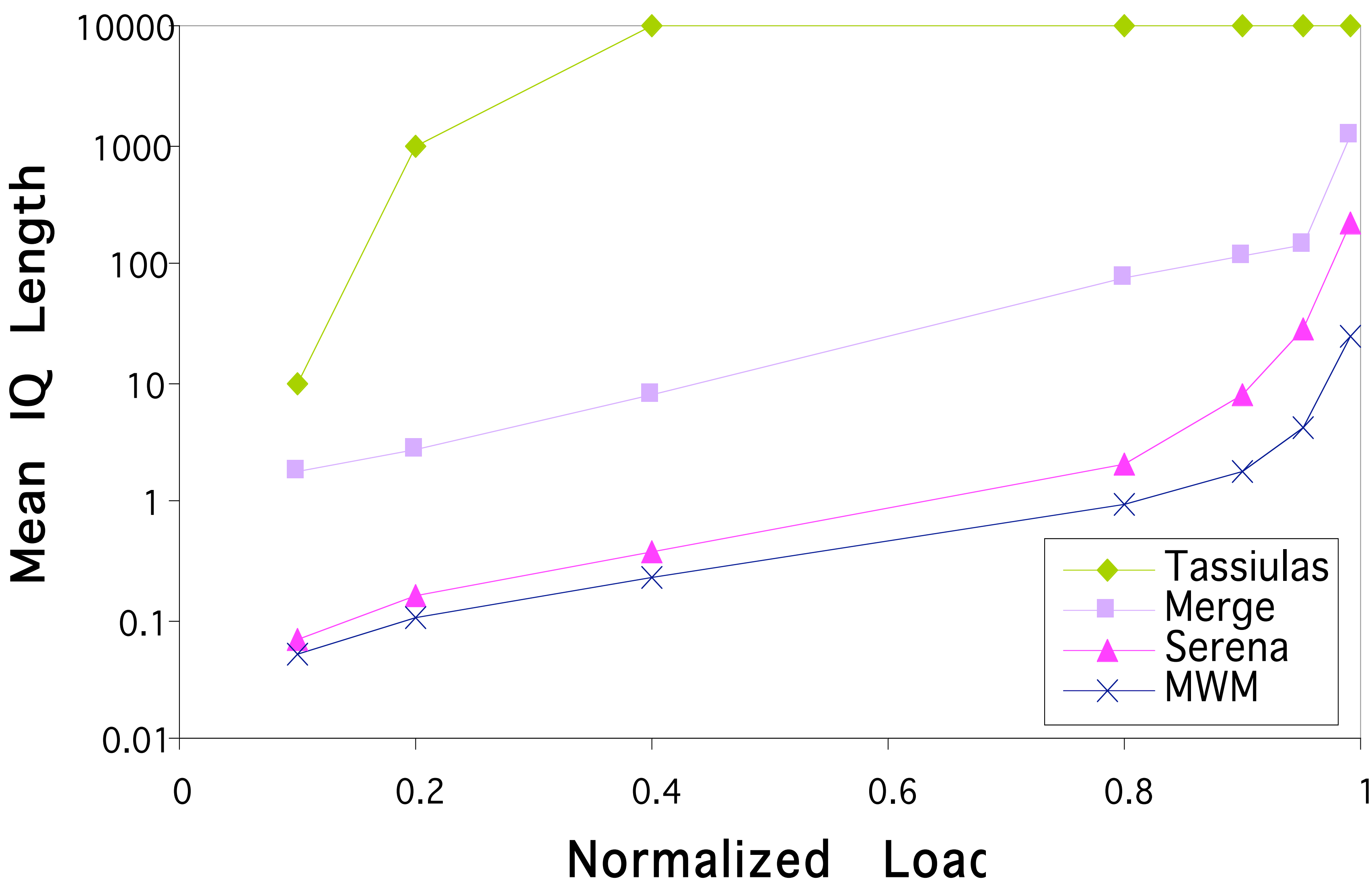
Use arrival information: Serena



Performance of Serena algorithm

Theorem (GPS): The Serena algorithm is stable under any admissible Bernoulli IID inputs.

Backlogs under Serena



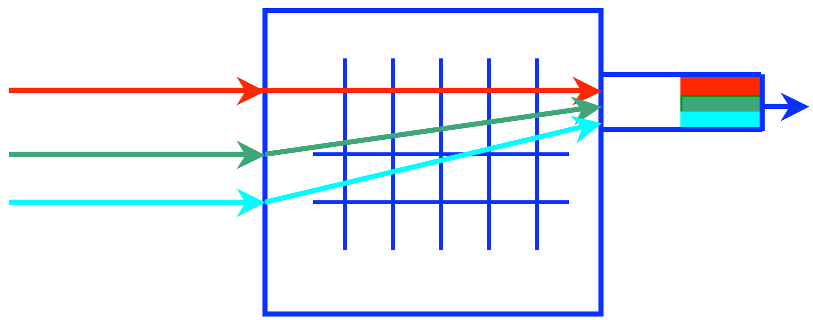
Bandwidth partitioning

(jointly with R. Pan, C. Psounis, C. Nair, B. Yang)

The Setup

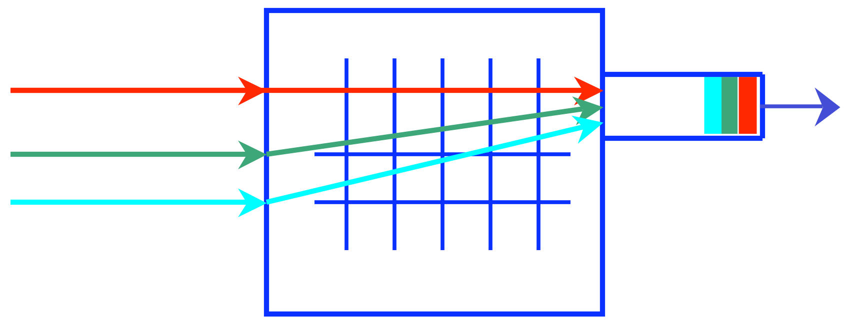
- A congested network with many users
- Problems:
  - allocate bandwidth fairly
  - control queue size and hence delay

Approach 1: Network-centric



- Network node: fair queueing
- User traffic: any type
  - ➢ problem: complex implementation

Approach 2: User-centric

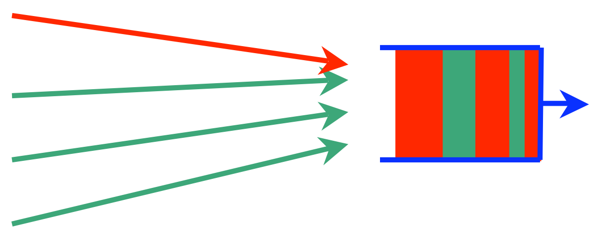


- Network node: simple FIFO
- User traffic: responsive to congestion (e.g. TCP)
  - ➢ problem: requires user cooperation

- For example, if the red source blasts away, it will get all of the link's bandwidth
- Question: Can we prevent a single source (or a small number of sources) from hogging up all the bandwidth, without explicitly identifying the rogue source?
- We will deal with full-scale bandwidth partitioning later

A Randomized Algorithm: First Cut

- Consider a single link shared by 1 unresponsive (red) flow and k *distinct* responsive (green) flows
- Suppose the buffer gets congested



- Observe: It is likely there are more packets from the red (unresponsive) source
- So if a randomly chosen packet is evicted, it will likely be a red packet
- Therefore, one algorithm could be:

When buffer is congested evict a randomly chosen packet

Comments

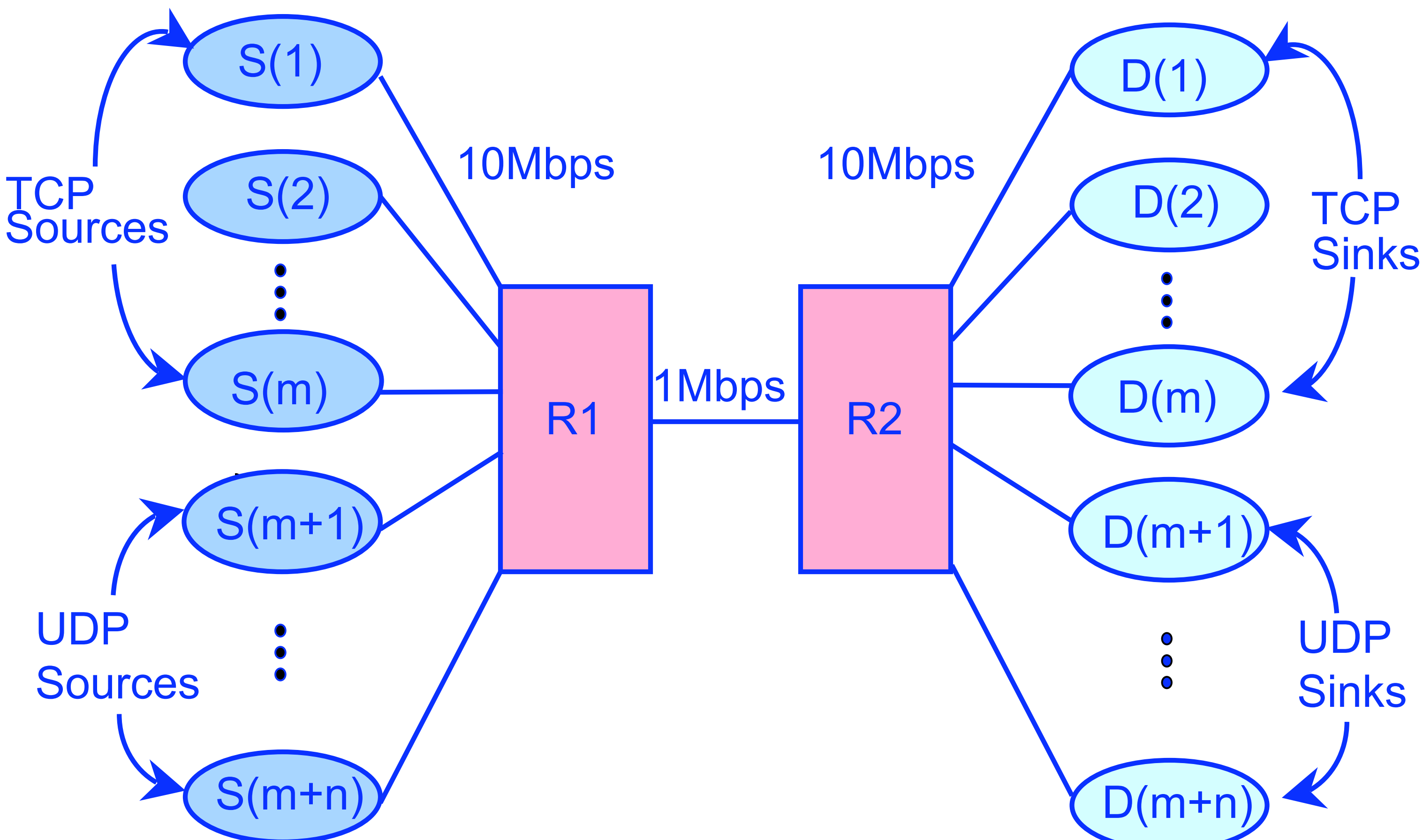
- Unfortunately, this doesn’t work because there is a small non-zero chance of evicting a green packet
- Since green sources are responsive, they interpret the packet drop as a congestion signal and back-off
- This only frees up more room for red packets

Randomized algorithm: Second attempt

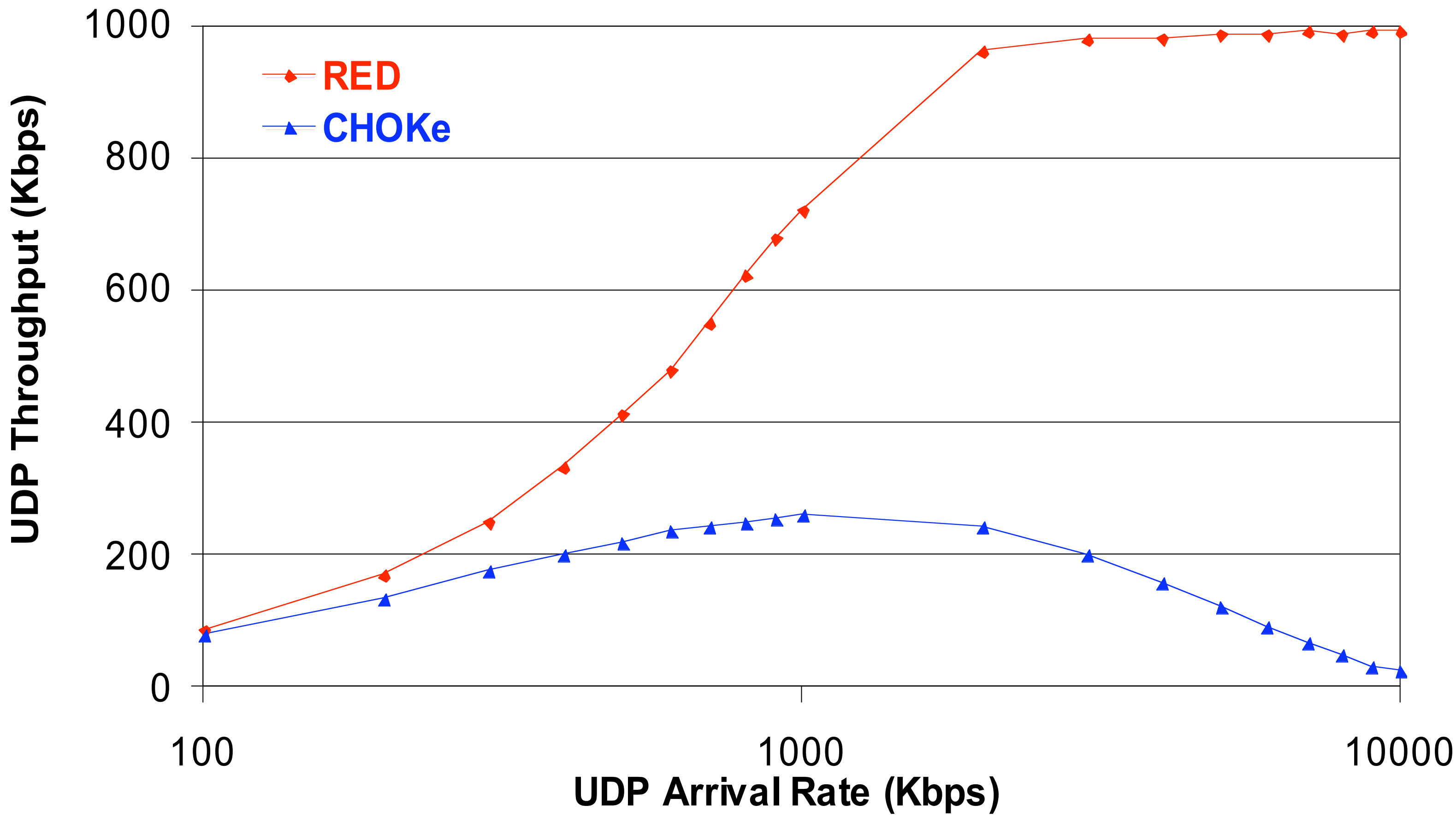
- Suppose we choose two packets at random from the queue and compare their ids, then it is quite unlikely that both will be green
- This suggests another algorithm:

  Choose two packets at random and drop them both if their ids agree
- This works: That is, it limits the maximum bandwidth the red source can consume

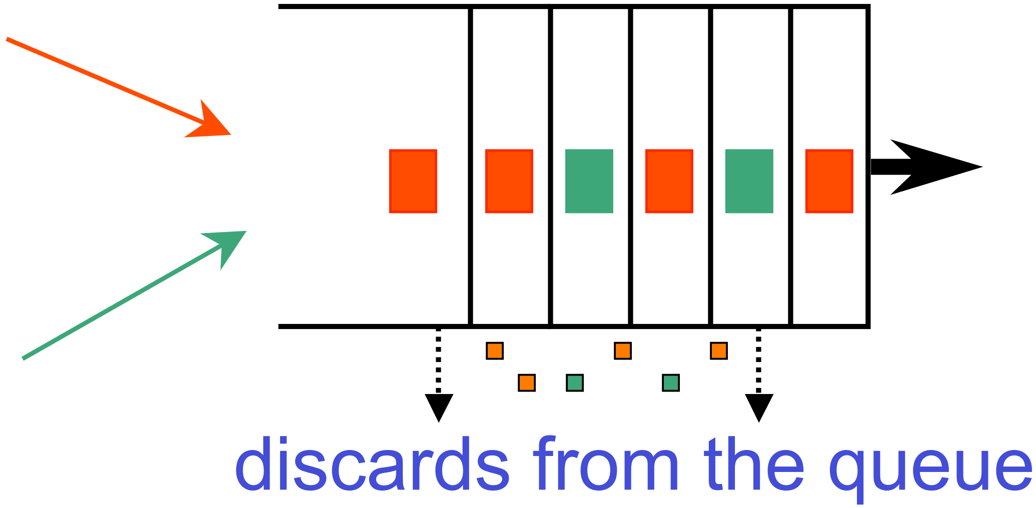
Simulation Comparison: The setup

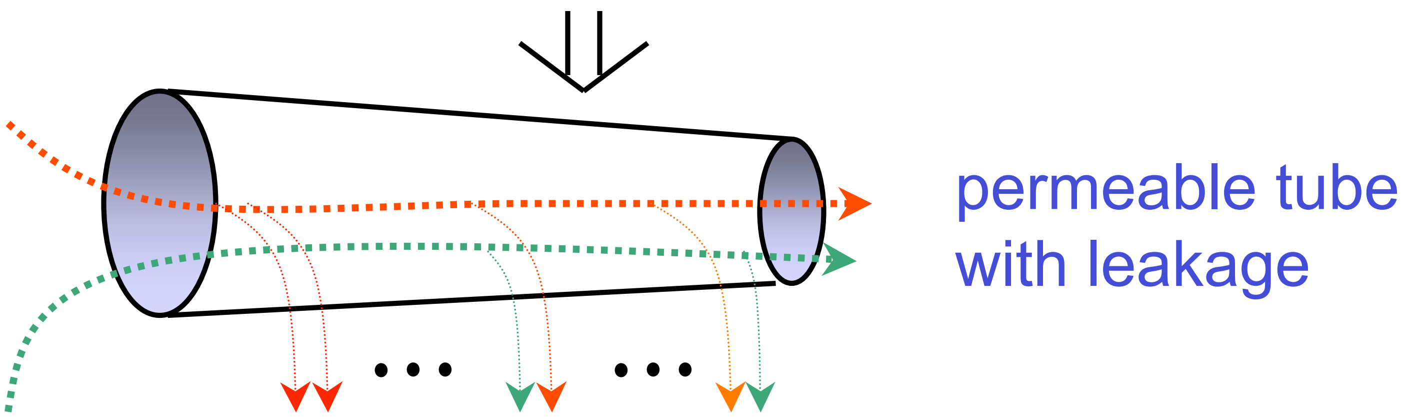


1 UDP source and 32 TCP sources



A Fluid Analysis





The Equation

$$L_i(t) - L_i(t + \delta t) = \lambda_i \delta t \, \frac{L_i(t)}{N}$$

$$=> \frac{dL_i(t)}{dt} = -\lambda_i \, \frac{L_i(t)}{N}$$

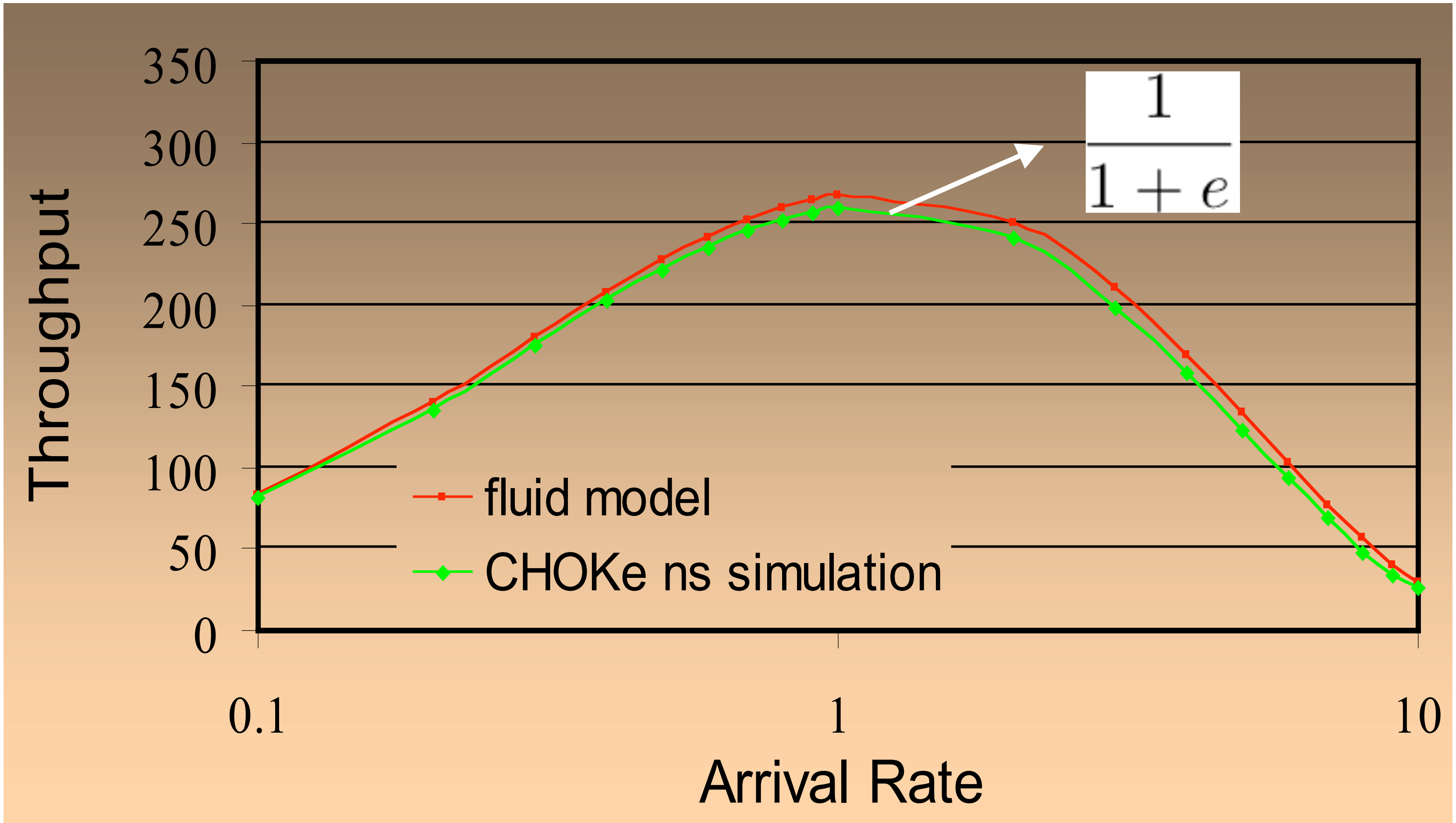
Boundary Conditions

$$L_i(0) = \lambda_i(1 - p_i);$$

$$L_i(D) = \lambda_i(1 - 2p_i)$$

$$p_i = \int_0^D \frac{L_i(t)\delta t}{N}$$

Simulation Comparison: 1UDP, 32 TCPs

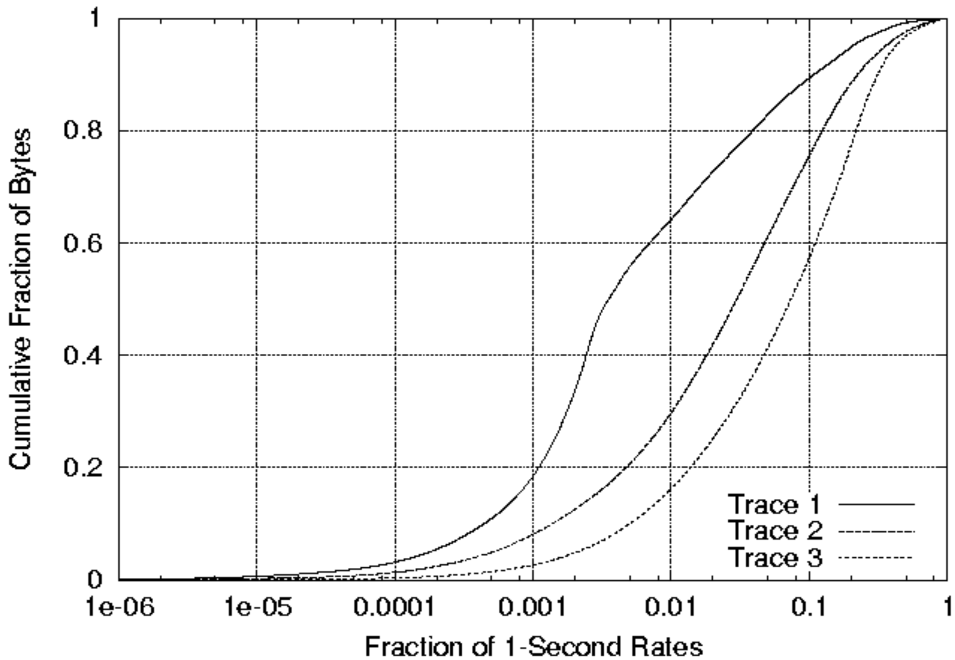


Complete bandwidth partitioning

- We have just seen how to prevent a small number of sources from hogging all the bandwidth
- However, this is far from ideal fairness
  - but, approaching ideal bandwidth partitioning, seems very costly
  - (recall the fair queueing algorithm)

Our approach: Exploit power laws

- Most flows are very small (mice), most bandwidth is consumed by a few large (elephant) flows: simply partition the bandwidth amongst the elephant flows



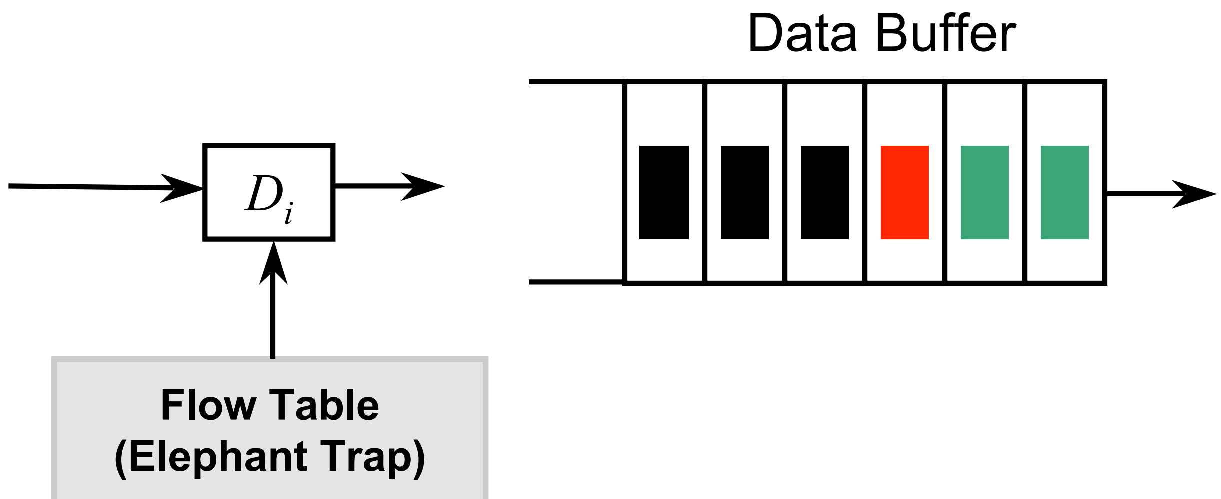
- New problem: Quickly (automatically) identify elephant flows, allocate bandwidth to them

Detecting large (elephant) flows

- Detection:
  - Flip a coin with bias *p* (= 0.1, say) for heads on each arriving packet, independently from packet to packet.
  - A flow is “sampled” if one its packets has a head on it

- A flow of size X has roughly 0.1X chance of being sampled
  - flows with fewer than 5 packets are sampled with prob $\cong$ 0.5
  - flows with more than 10 packets are sampled with prob $\cong$ 1
- Most mice will not be sampled, most elephants will be

The AFD Algorithm



- AFD is a randomized algorithm
  - joint with Rong Pan, Flavio Bonomi, Lee Breslau, Bob Olsen and Scott Shenker
- Current implementation plans at Cisco; 5 platforms
  - Apex-Chopper NPU based SPAs for GSR12000, and 7600
  - Next generation MAC ASICs for 6500, and DC3
  - Cat 3K wireless service cards

Conclusions

- Efficient network hardware design poses a lot of interesting algorithmic problems, mainly because of very tight constraints
- Simple algorithms are needed
- We’ve seen that randomization and power laws can be exploited