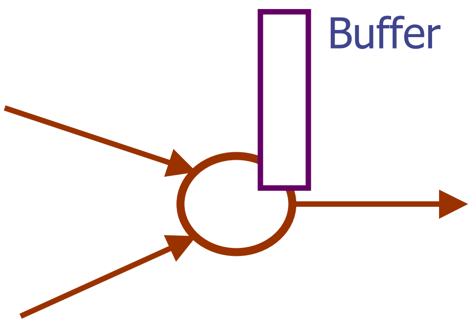
RED-PD: RED with Preferential Dropping

http://www.aciri.org/red-pd

Ratul Mahajan
Sally Floyd
David Wetherall

Background

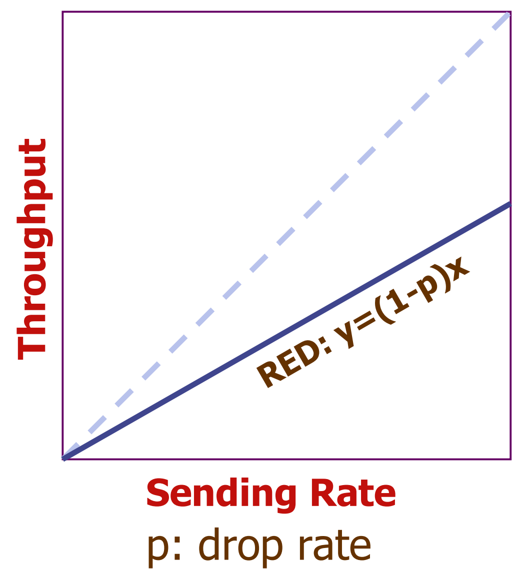
- FIFO – first in first out
  - drop tail



- RED – Random Early Detection
  - detect incipient congestion
  - distribute drops fairly

Problem

- High bandwidth flows increase the drop rate at the router
- Without flow based differentiation all flows see the same drop rate
  - flows get more by sending more



Solution: router mechanisms to protect rest of the traffic from high bandwidth flows

Relevance

- Where is congestion?
  - Backbone (?)
  - Edge routers
  - Exchange points
  - Intercontinental links

- Starting point for aggregate based congestion

A Possible Approach – Per-flow state

Examples: FQ, DRR, CSFQ, FRED

Sequester all flows from each other

+ provide full max-min fairness
  - required for best-effort traffic?

- state for ALL the flows passing through the router, most of which are small "Web mice"

RED-PD's Approach – Partial Flow State

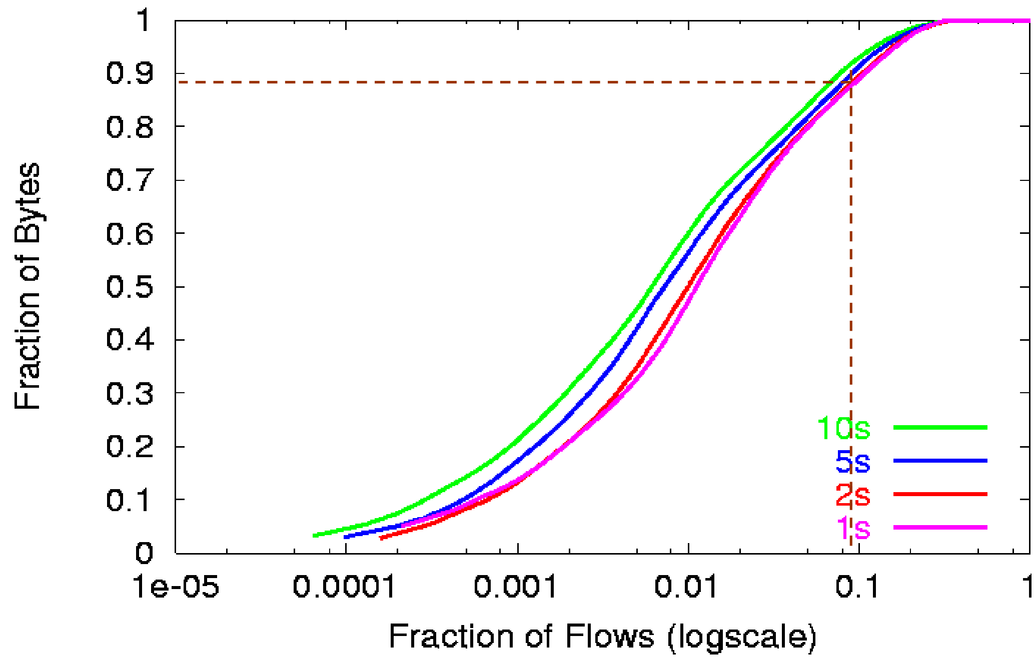
State for high bandwidth flows only

- Identify high bandwidth flows during times of congestion. These are called monitored flows.
- Preferentially drop from monitored flows to limit their throughput.

Combines the simplicity of FIFO with the protection of full max-min fair techniques that keep per-flow state.

Why Partial Flow State Approach Works

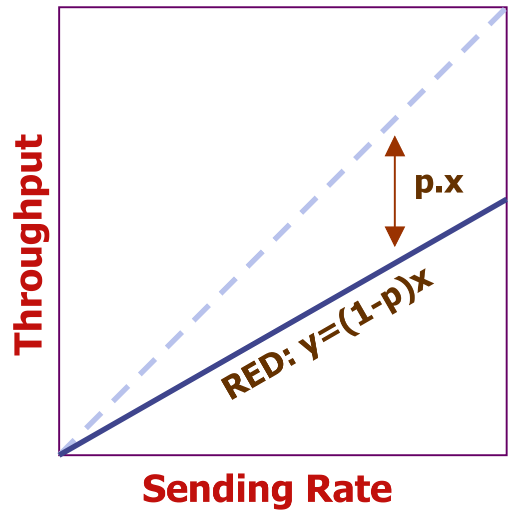
What fraction of flows get what fraction of bytes over different time windows.



Bandwidth distribution is very skewed - a small fraction of flows accounts for most of the bandwidth.

Identification Approach – Drop History

Flows that send more suffer more drops



p: drop rate

- Cheap means for identification
- Drop history contains flows that have been sent congestion signal

Defining "High Bandwidth"

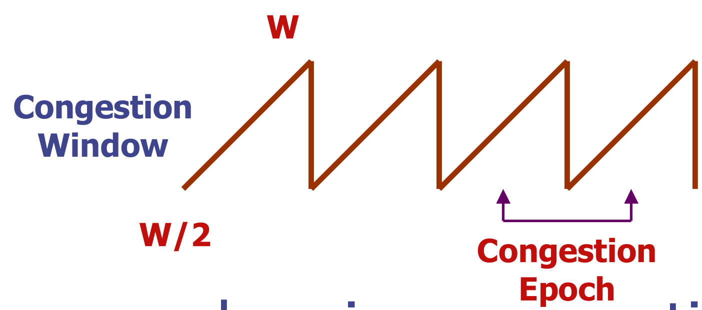
- Pick a round trip time (RTT) *R*
  - call a TCP with RTT *R* as reference TCP

- High bandwidth flows:
  - All flows sending more than the *reference* TCP flow

- Target bandwidth *Target(R,p)* given by the TCP response function
  - $Target(R,p) =\sim \frac{\sqrt{1.5}}{R\sqrt{p}}$ packets/second

  where *p* is the drop rate at the output queue

Identifying High Bandwidth Flows



- TCP suffers one drop in a congestion epoch
- Length of congestion epoch can be computed using drop rate
- *CELength(R,p)* = $\frac{R}{\sqrt{1.5}\sqrt{p}}$ congestion epoch length of the reference TCP
  - identify flows with one or more drops in *CELength(R,p)* seconds

Tuning Identification

1. Issue: all flows suffer a drop once in a while
   - keep the drop history of *K.CELength(R,p)* seconds and identify flows with K or more drops (K>1)

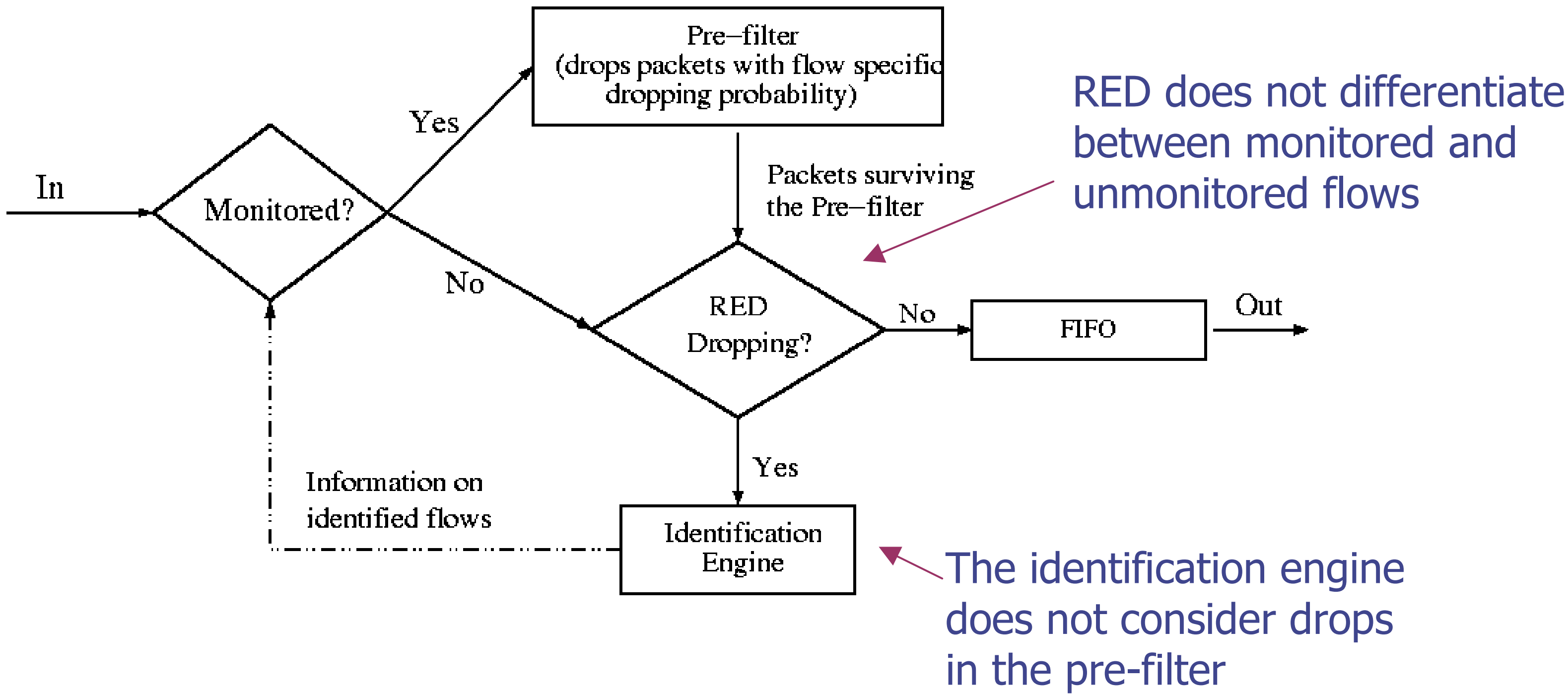
2. Issue: multiple losses in a window of data
   - maintain the drop history as M separate lists instead of a single list
     - length of each list is *(K/M) .CELength(R,p)* seconds
   - identify flows with drops in *K* or more of the *M* lists
     - lets go flows with more than K drops in a short period

Preferential Dropping

- Probabilistically drop packets from monitored flows before they enter the output queue

- The dropping probability is computed to bring down rate of a monitored flow into the output queue to *Target(R,p)*
  - probability = 1 – Target(R,p)/arrival rate

Architecture



- If a monitored flow is identified again, the pre-filter dropping probability is too small
- If a monitored flow is suffering too few drops, the pre-filter dropping probability is too large

Probability Computation

- Decrease dropping probability of a monitored flow with no drops in M lists by halving it
  - upper bound on reduction in one round
- Increase dropping probability of an identified flow
  - an identified flow could be either a monitored flow or a newly identified flow.
- Don't change dropping probability too fast

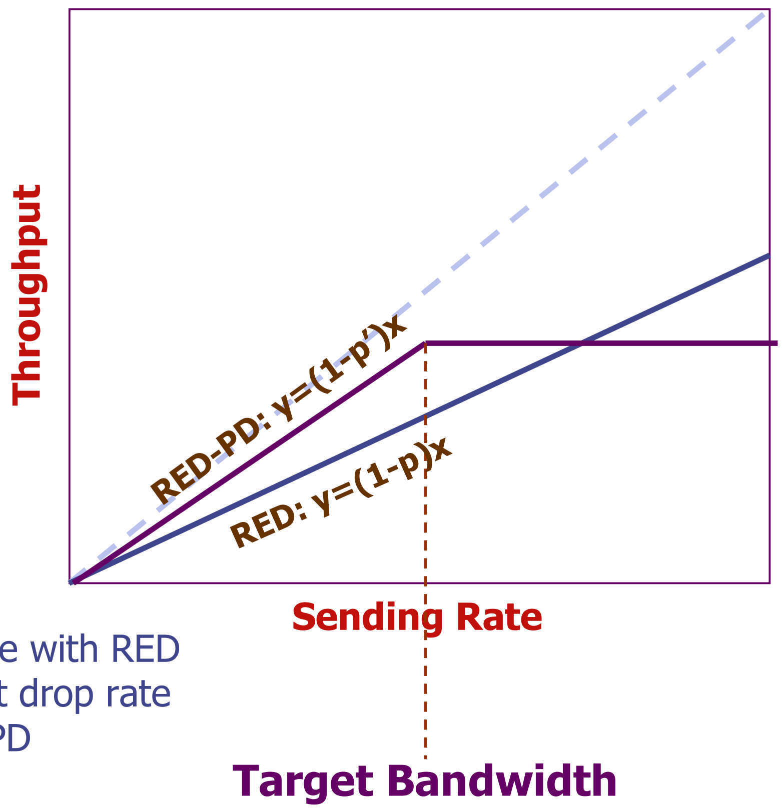
Probability Increase

- Add an increase quantum to existing probability

- The increase quantum should depend on
  - *ambient* drop rate (drop rate at the output queue)
  - sending rate of a flow

- increase quantum = (number_drops/avg_drops).p
  - upper bound on increase in one step

Effect of RED-PD



p: drop rate with RED
p': ambient drop rate with RED-PD

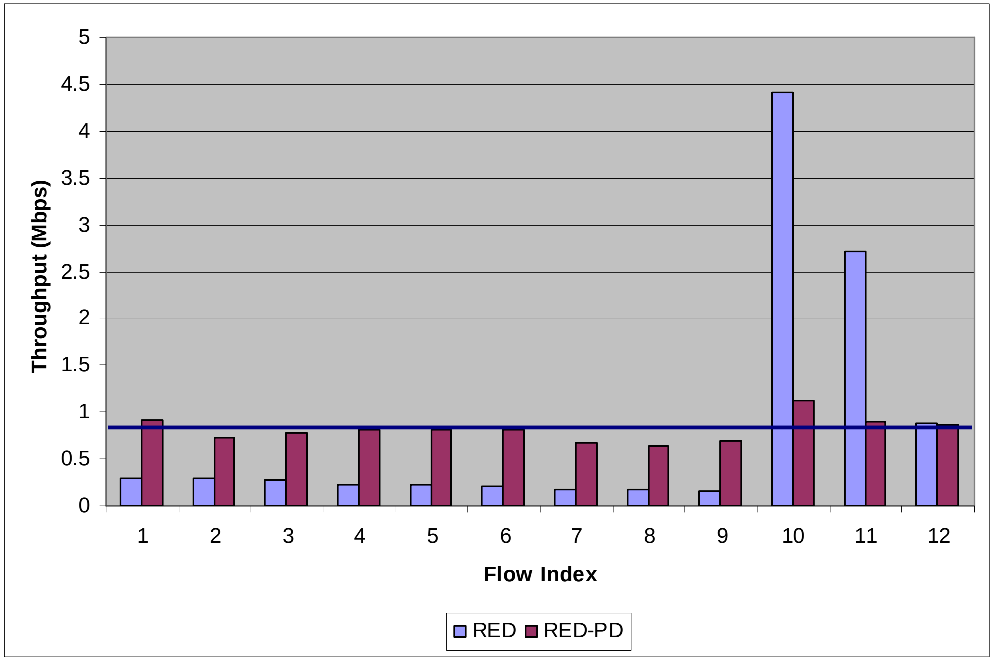
- Reduction in ambient drop rate (p' < p)
- Full max-min fair in the extreme case (p' = 0)

Evaluation

- Fairness
- Response time
- Effect of target RTT *R*

- Probability of identification
- Persistent congestion throughput
- Web traffic
- Multiple congested links
- TFRC
- Byte mode operation

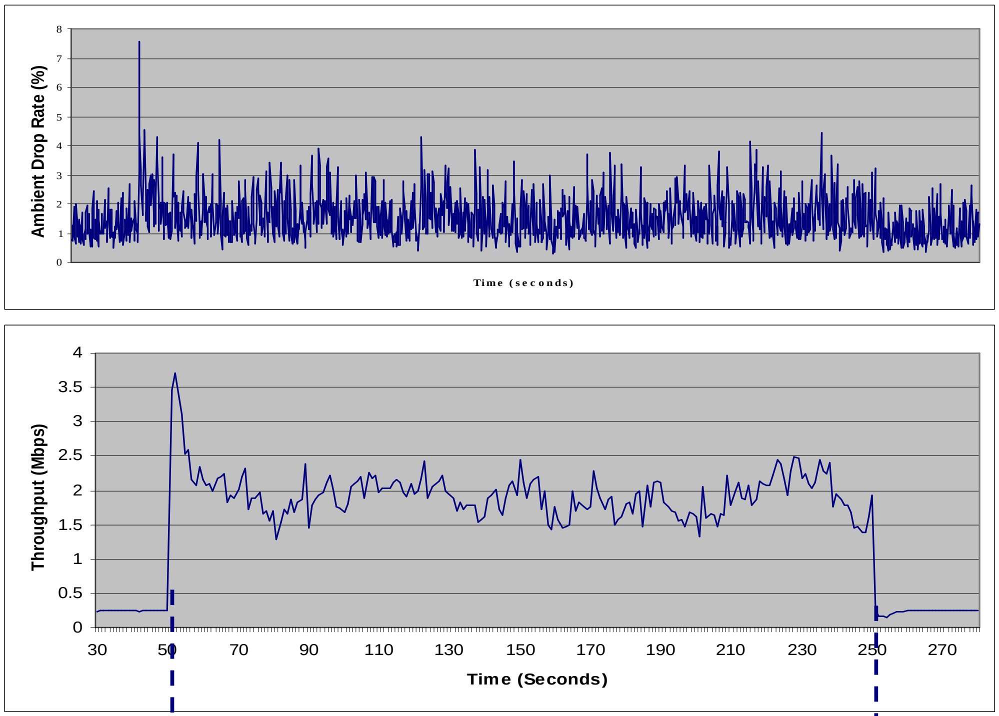
Fairness



10 Mbps link
R= 40ms
Flow Index
1-3 30ms TCP
4-6 50ms TCP
7-9 70ms TCP
10 5 Mbps CBR
11 3 Mbps CBR
12 1 Mbps CBR

Iterative probability changes successfully approximate fairness

Response Time



0.25Mbps 4Mbps 0.25Mbps

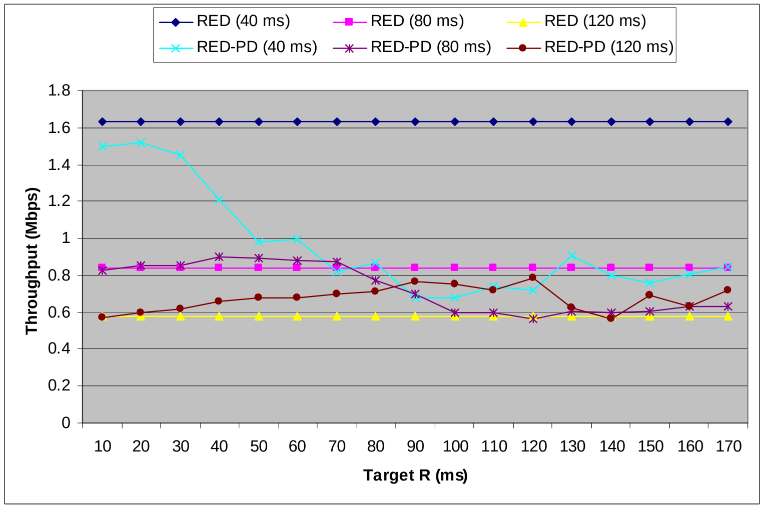
Sending rate of CBR changes with time

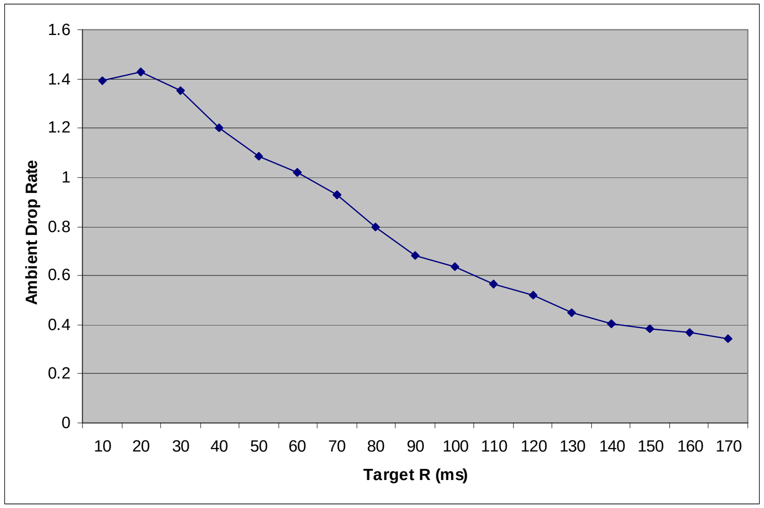
10 Mbps link
1 CBR flow
9 TCP flows
R = 40 ms

Flow brought down in 6s and released in 5s

The speed of RED-PD's reaction depends on ambient drop rate and flow's sending rate.
Its faster when either is higher

Effect of target RTT *R*



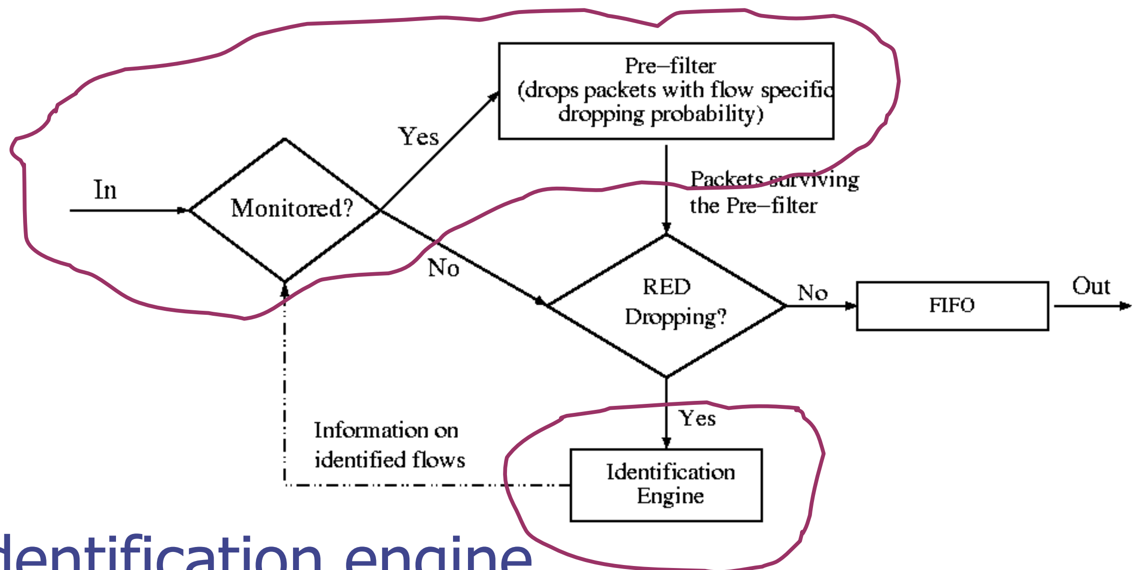


10 Mbps link
14 TCP flows
2 each of RTT 40, 80 & 120 ms
and 8 of RTT 160 ms

Increasing *R*

- increases fairness
- increases state
- decreases ambient drop rate

Implementation Feasibility



- Identification engine
  - state for drop history
  - not in fast forwarding path
- Flow classification and pre-filter
  - hash table of all flows being monitored
  - drop with the computed probability

Comparison

	State for what?	What state?	Fast-path processing?	When processing?
FQ/ DRR	All flows	Queues	Queue management and scheduling	Arrival/ Departure
FRED	All buffered flows	Number of buffered packets	Drop probability computation and coin tossing	Arrival/ Departure
CSFQ	All flows	Arrival rate estimate, time of last packet	Update arrival rate estimate, change header	Arrival
RED-PD	High bandwidth flows	Dropping probability, drop history	Coin tossing	Arrival

Future Work

- State requirements
- Dynamically varying *R*
  - Target ambient drop rate
  - State limitations
- Misbehaving flows

Conclusions

- Increased fairness using much less state
- Tunable degree of fairness
- Reasonable response time
- Lightweight and easy to implement

A continuum of policies

Per-flow state | High bandwidth flow state | Misbehaving flow state | No flow state

Continuum of policies

FQ
DRR SFQ | CBQ Router Mechanisms | FIFO

Scheduling Approaches

← RED-PD →

CSFQ FRED SFB | RED

Preferential Dropping Approaches