Active Queue Management

Rong Pan
Cisco System

EE384y
Spring Quarter 2006

Outline

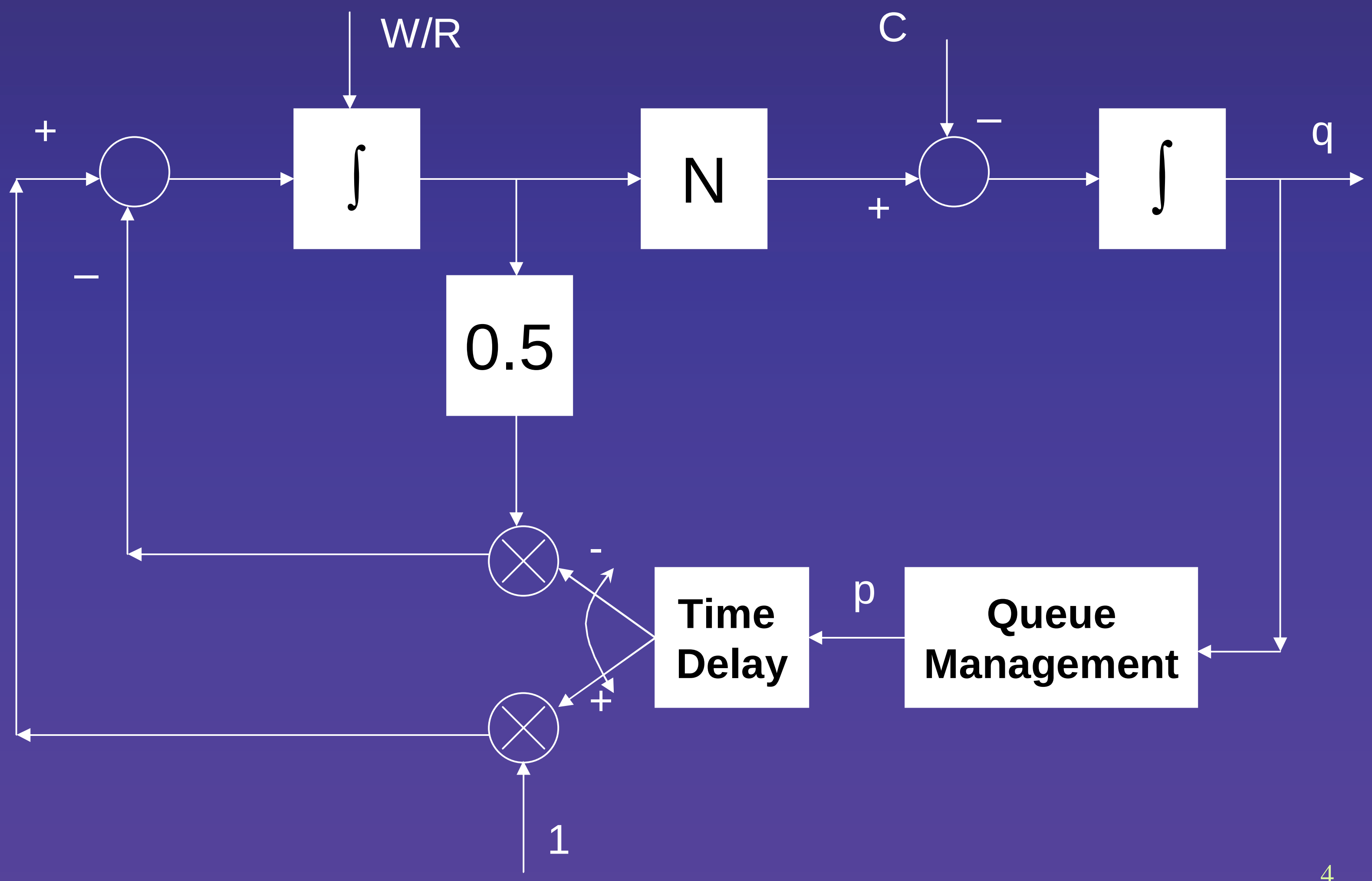
- Queue Management
  - *Drop as a way to feedback to TCP sources*
  - *Part of a closed-loop*
- Traditional Queue Management
  - *Drop Tail*
  - *Problems*
- Active Queue Management
  - *RED*
  - *CHOKe*
  - *AFD*

Queue Management: Drops/Marks

- A Feedback Mechanism To Regulate End TCP Hosts

- End hosts send TCP traffic -> Queue size
- Network elements, switches/routers, generate drops/marks based on their queue sizes
- Drops/Marks: regulation messages to end hosts
- TCP sources respond to drops/marks by cutting down their windows, i.e. sending rate

TCP+Queue Management
- A closed-loop control system

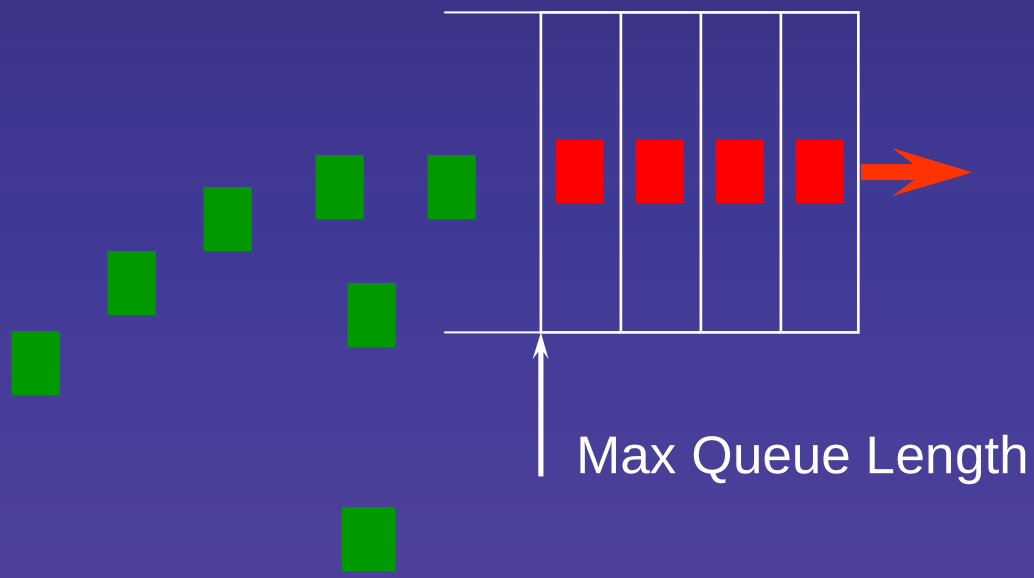


Drop Tail
- problems

- Lock out
- Full queue
- Bias against bursty traffic
- Global synchronization

Tail Drop Queue Management

Lock-Out

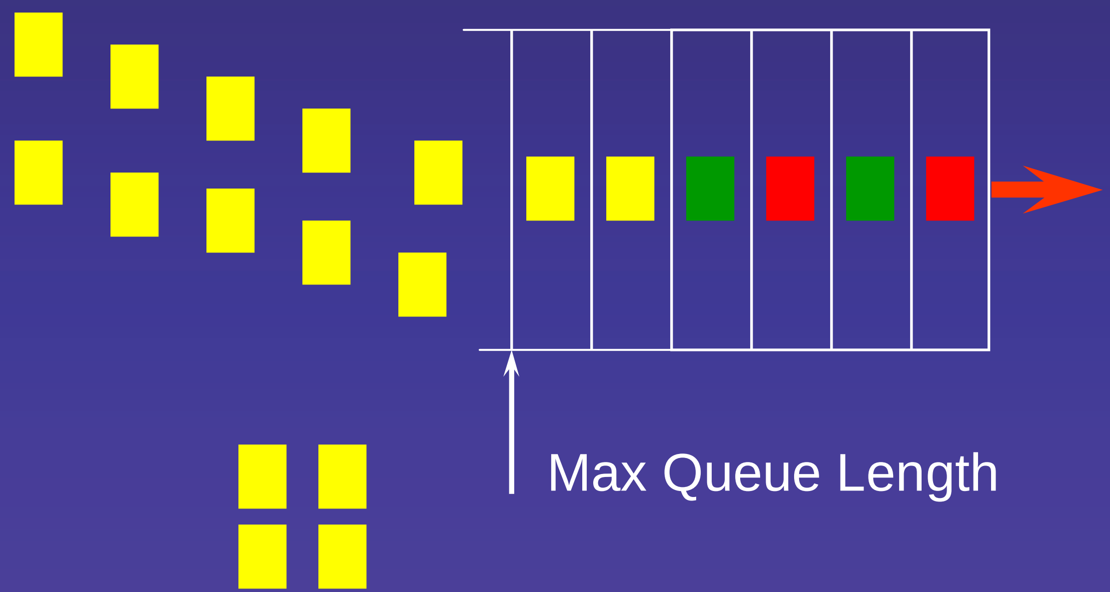


Tail Drop Queue Management

Full-Queue

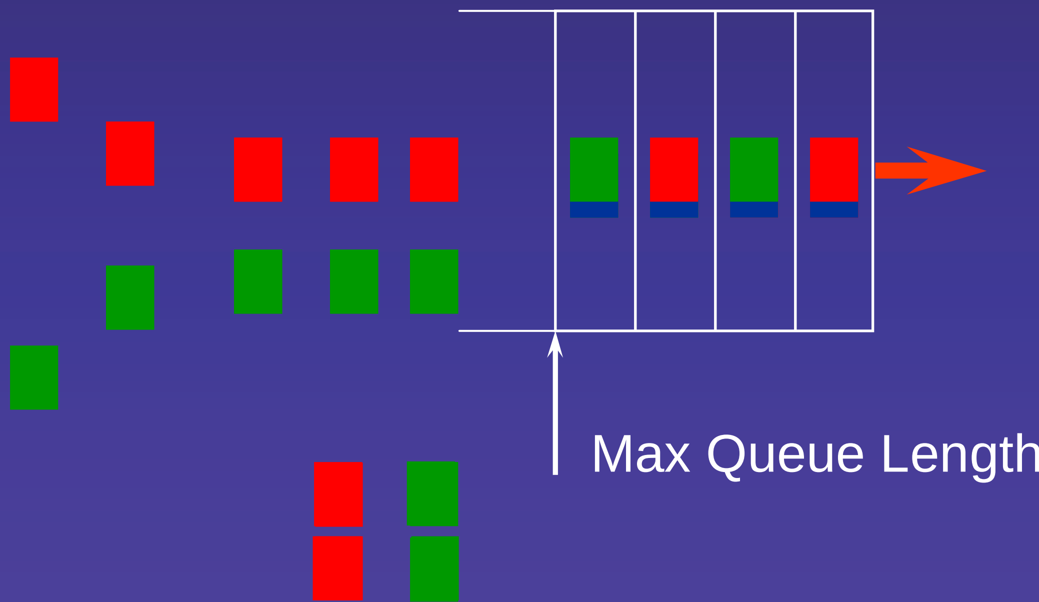
- Only drop packets when queue is full
  - *long steady-state delay*

Bias Against Bursty Traffic



Tail Drop Queue Management

Global Synchronization



Alternative Queue Management Schemes

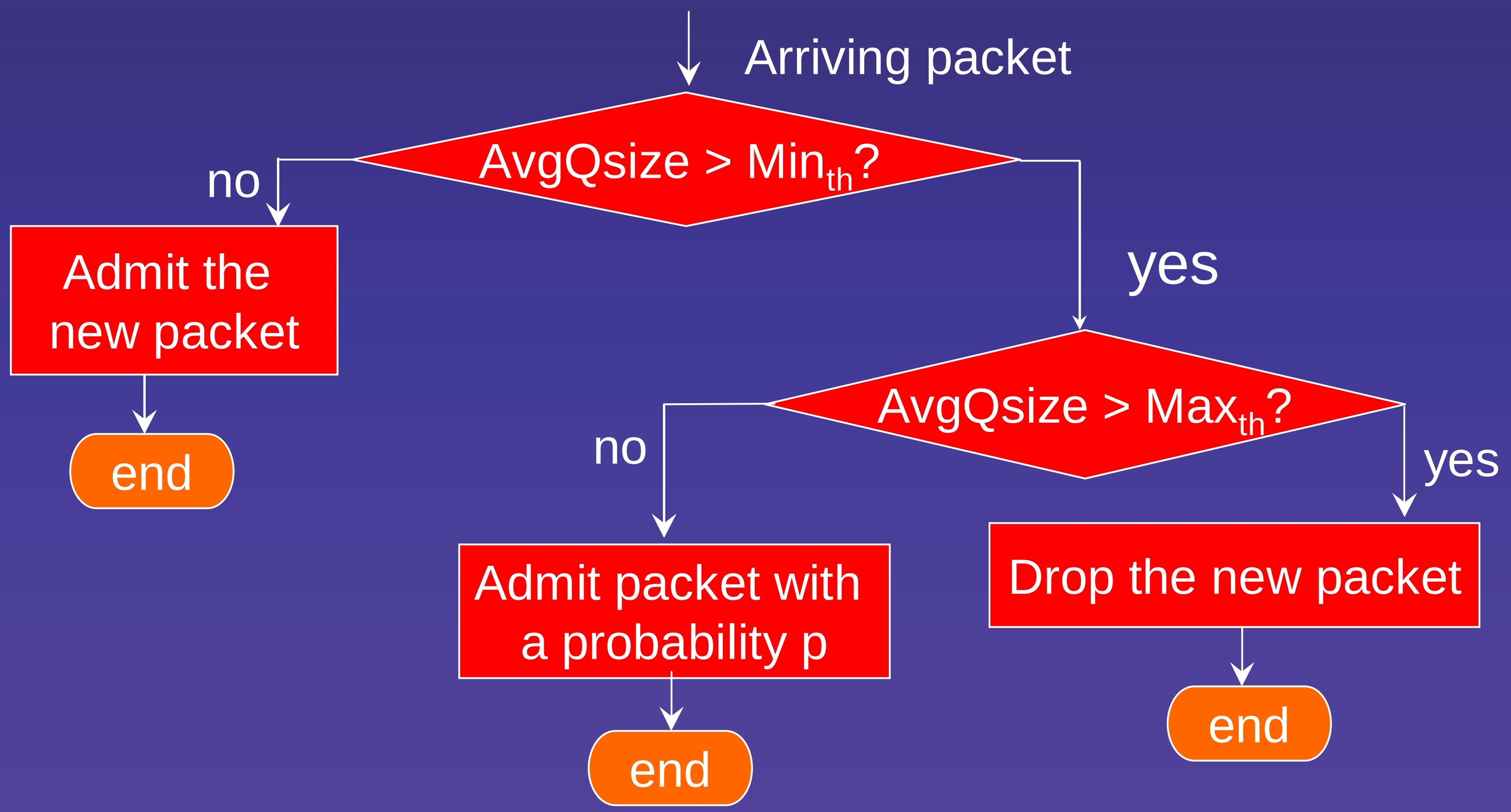
- Drop from front on full queue
- Drop at random on full queue

  ☞ *both solve the lock-out problem*
  ☞ *both have the full-queues problem*

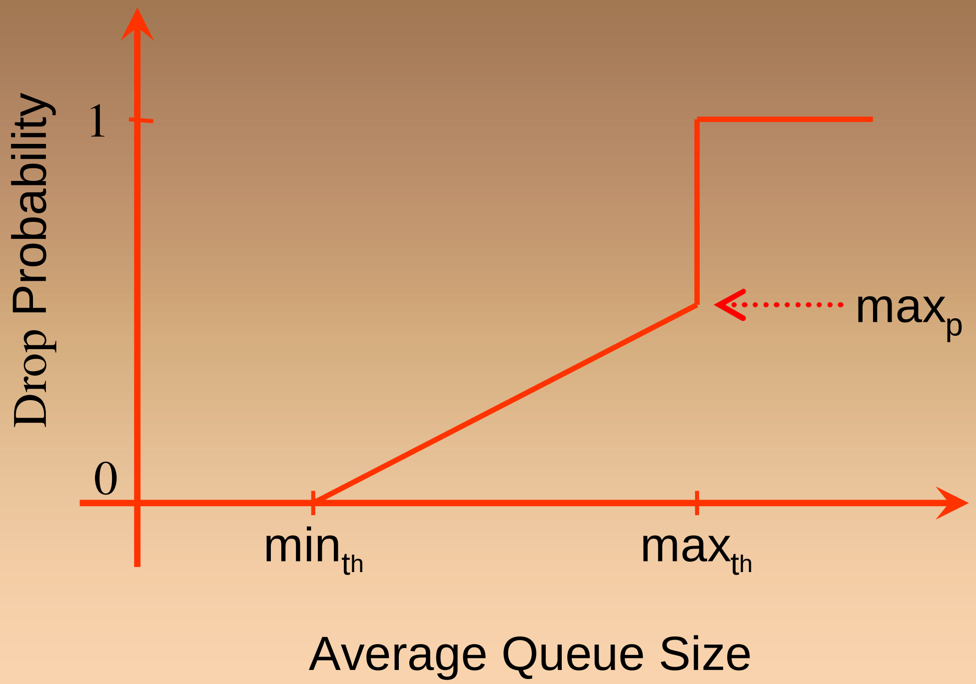
Active Queue Management
Goals

- Solve tail-drop problems
  - *no lock-out behavior*
  - *no global synchronization*
  - *no bias against bursty flow*
- Provide better QoS at a router
  - *low steady-state delay*
  - *lower packet dropping*

Random Early Detection (RED)



RED Dropping Curve



Effectiveness of RED
- Lock-Out & Global Synchronization

- Packets are randomly dropped
- Each flow has the same probability of being discarded

Effectiveness of RED

- Full-Queue & Bias against bursty traffic

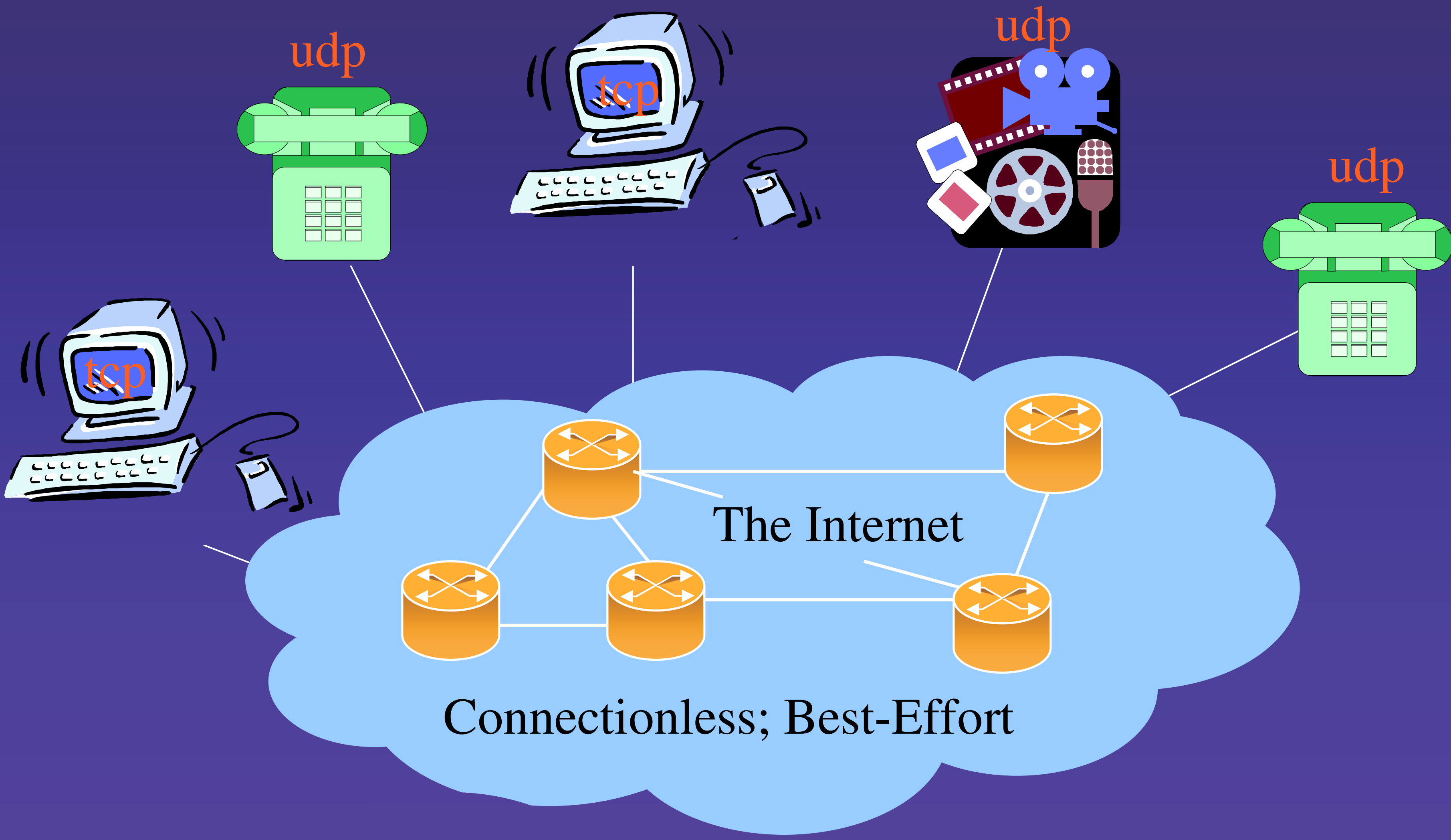
- Drop packets probabilistically in anticipation of congestion
  - *not when queue is full*

- Use q_{avg} to decide packet dropping probability: allow instantaneous bursts

What QoS does RED Provide?

- Lower buffer delay: good interactive service
  - *q_{avg} is controlled to be small*
- Given responsive flows: packet dropping is reduced
  - *early congestion indication allows traffic to throttle back before congestion*
- Given responsive flows: fair bandwidth allocation

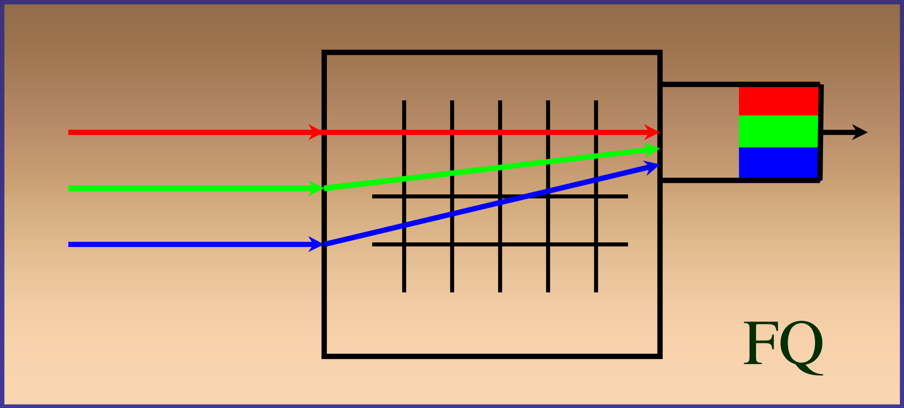
Bad News - unresponsive end hosts



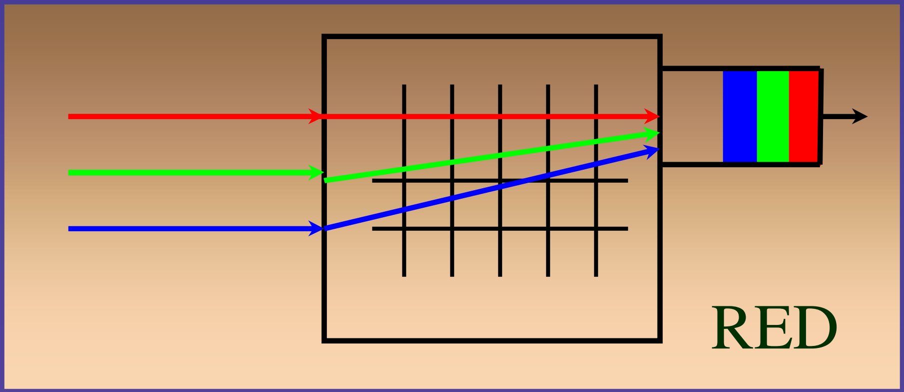
Scheduling & Queue Management

- What routers want to do?
  - *isolate unresponsive flows (e.g. UDP)*
  - *provide Quality of Service to all users*
- Two ways to do it
  - *scheduling algorithms:*
    e.g. FQ, CSFQ, SFQ
  - *queue management algorithms:*
    e.g. RED, FRED, SRED

FQ vs. RED

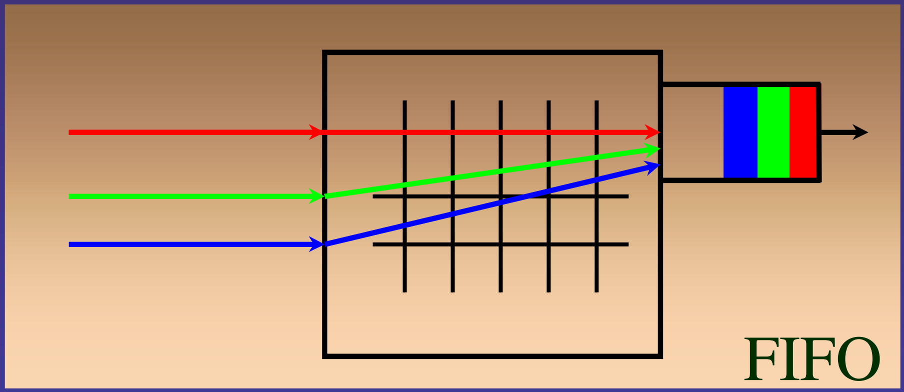


- Isolation from non-adaptive flows
- Hard/Expensive to implement



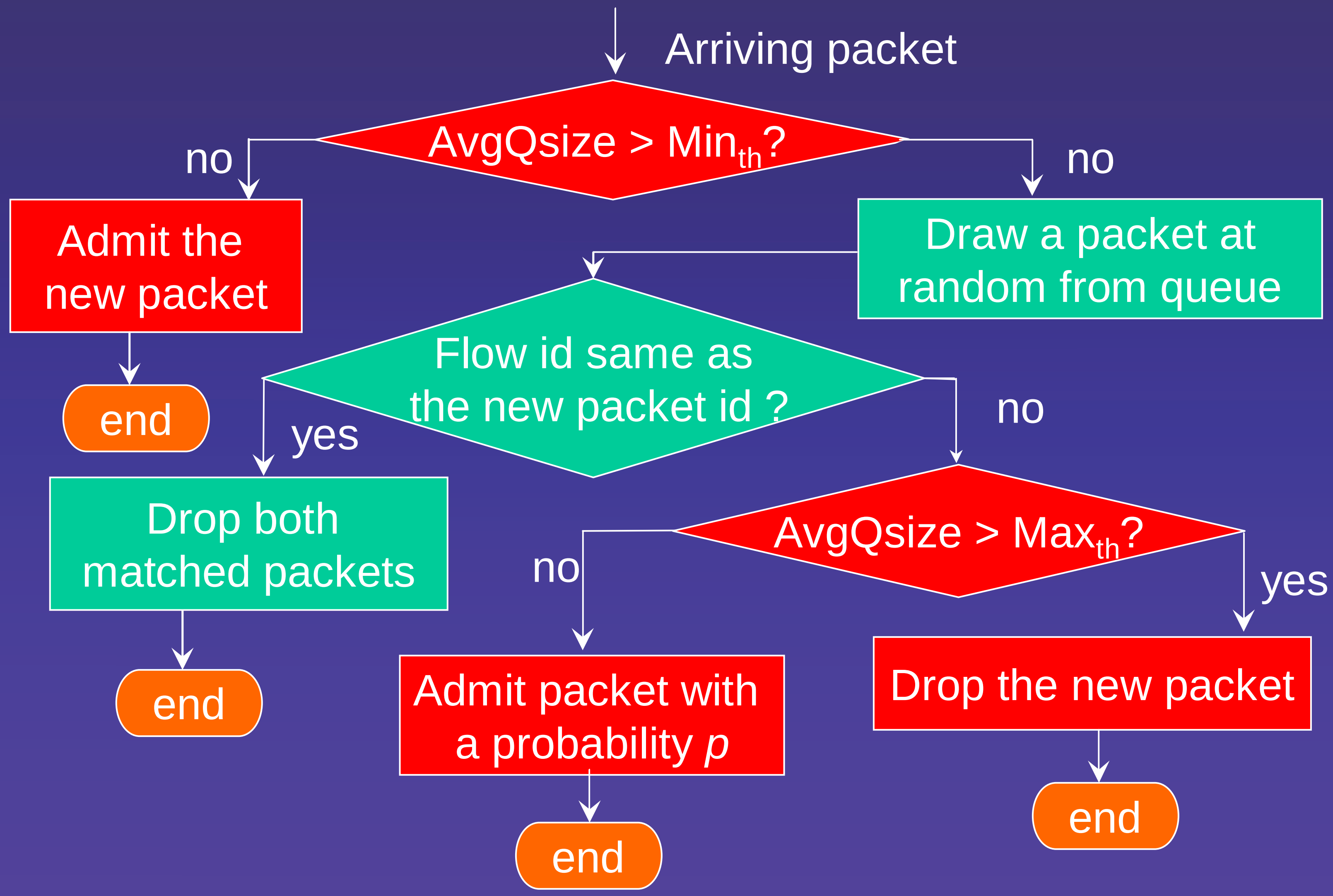
- No isolation from non-adaptive flows
- Easy to implement

Active Queue Manament With Enhancement to Fairness

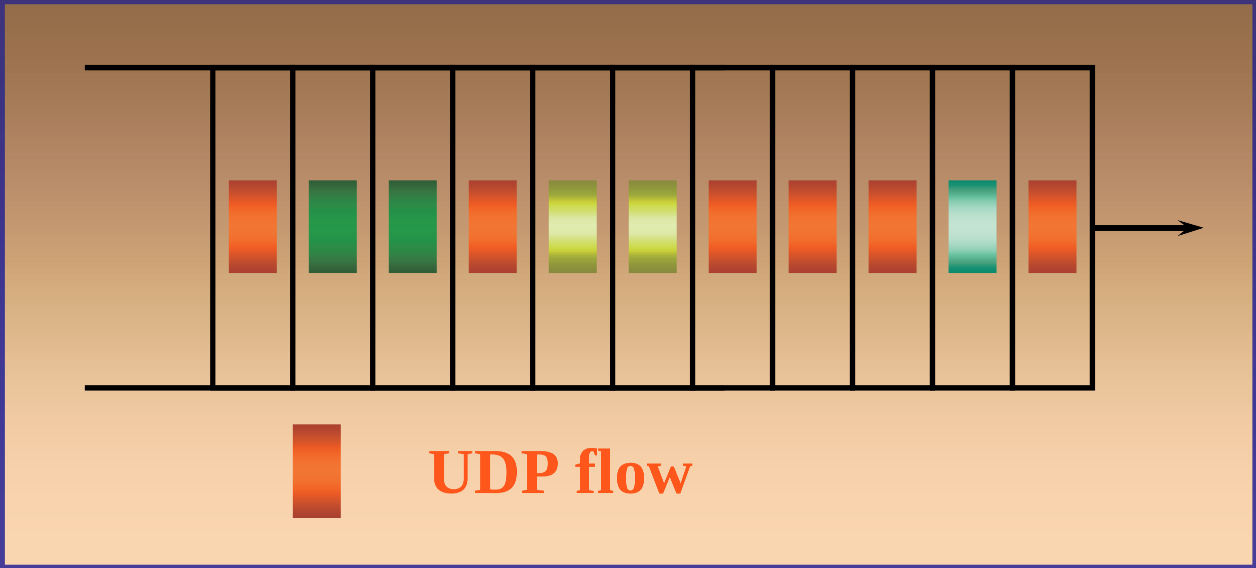


- Provide isolation from unresponsive flows
- Be as simple as RED

CHOKe


Arriving packet
AvgQsize > Min$_{th}$?
no
no
Admit the new packet
Draw a packet at random from queue
end
Flow id same as the new packet id ?
yes
no
Drop both matched packets
AvgQsize > Max$_{th}$?
no
yes
end
Admit packet with a probability *p*
Drop the new packet
end
end

Random Sampling from Queue



- A randomly chosen packet more likely from the unresponsive flow
- Adversary can't fool the system

Comparison of Flow ID

- Compare the flow id with the incoming packet
  - *more acurate*
  - *Reduce the chance of dropping packets from a TCP-friendly flows.*

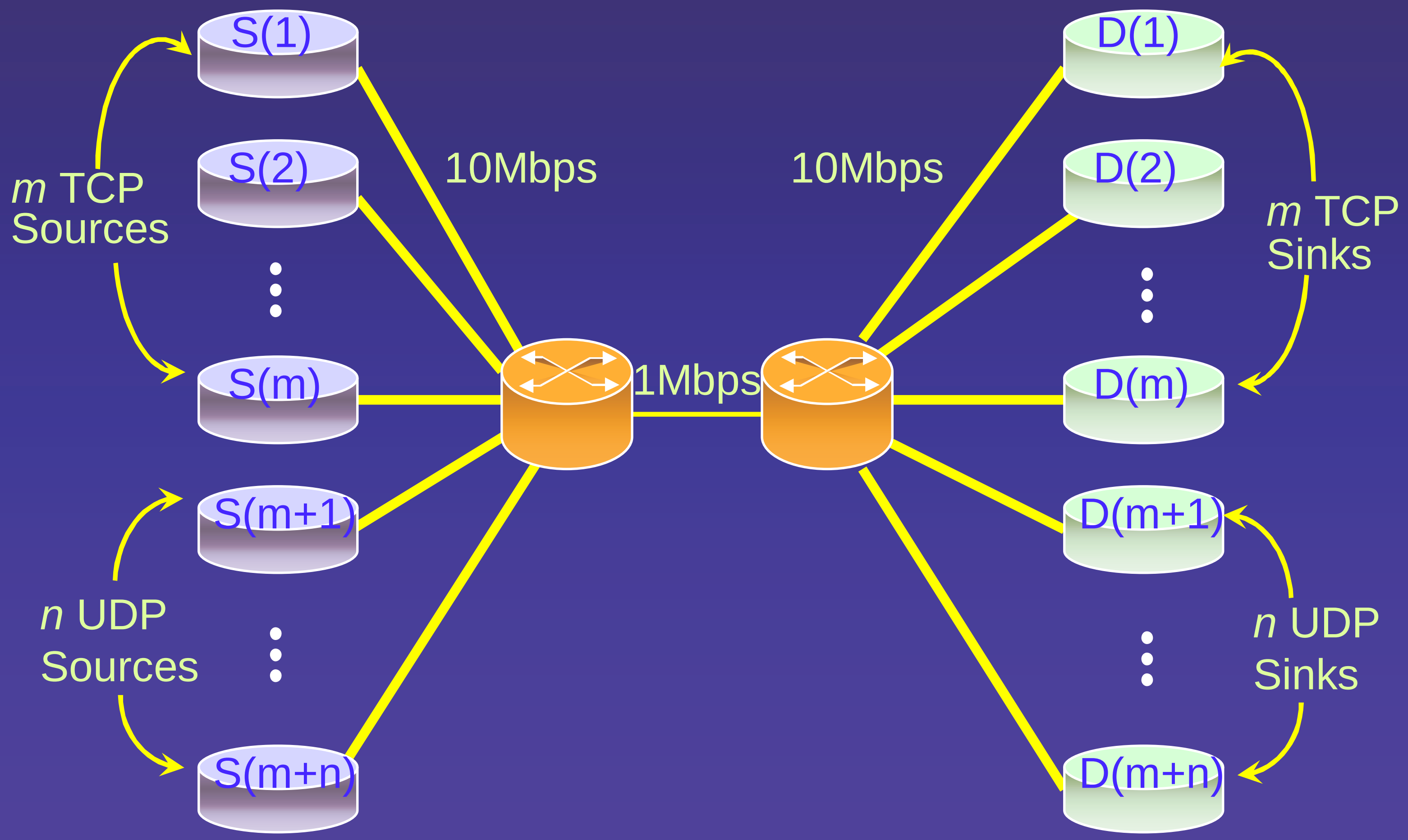
Dropping Mechanism

- Drop packets (both incoming and matching samples)

  – *More arrival -> More Drop*

  – *Give users a disincentive to send more*

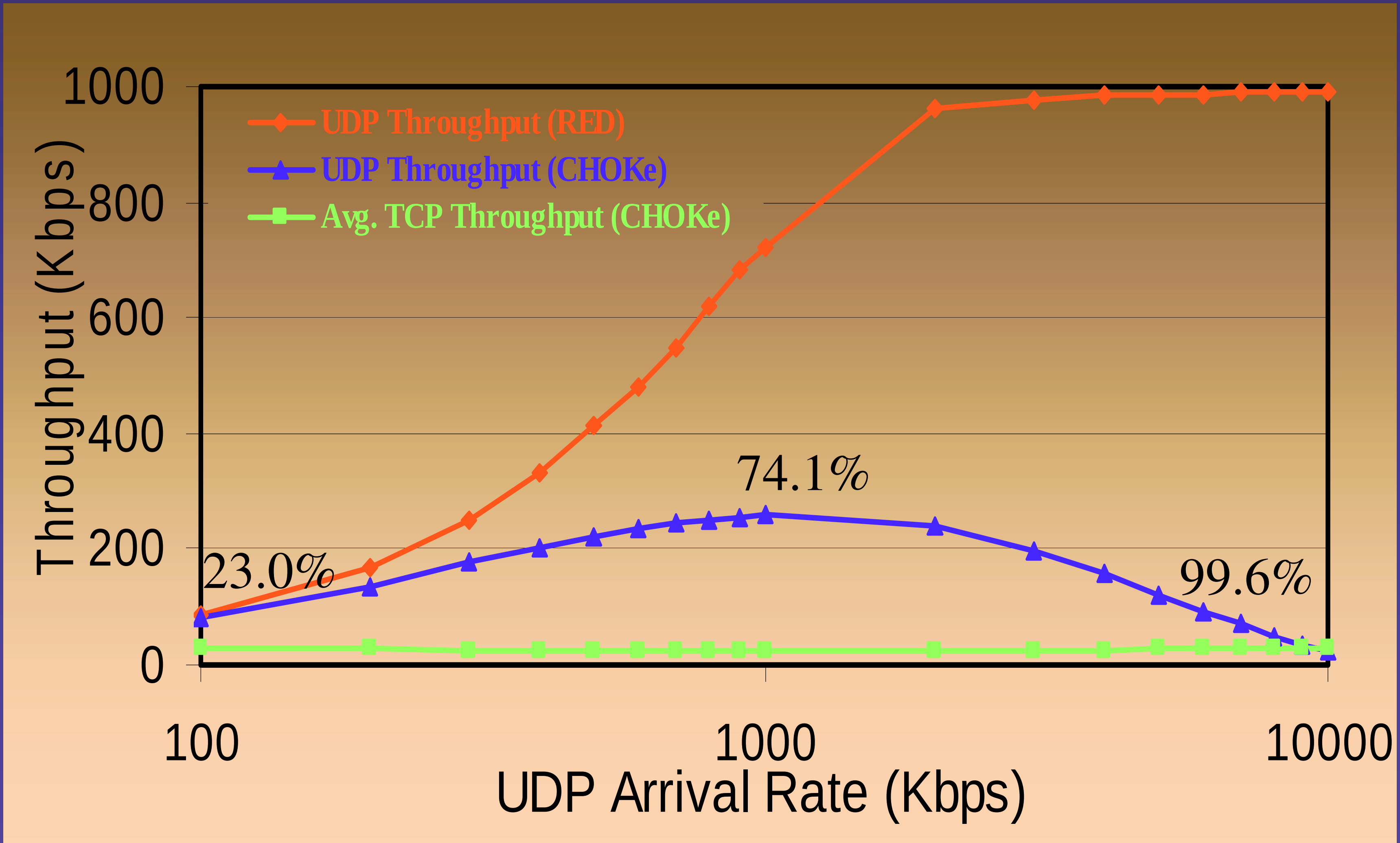
Simulation Setup



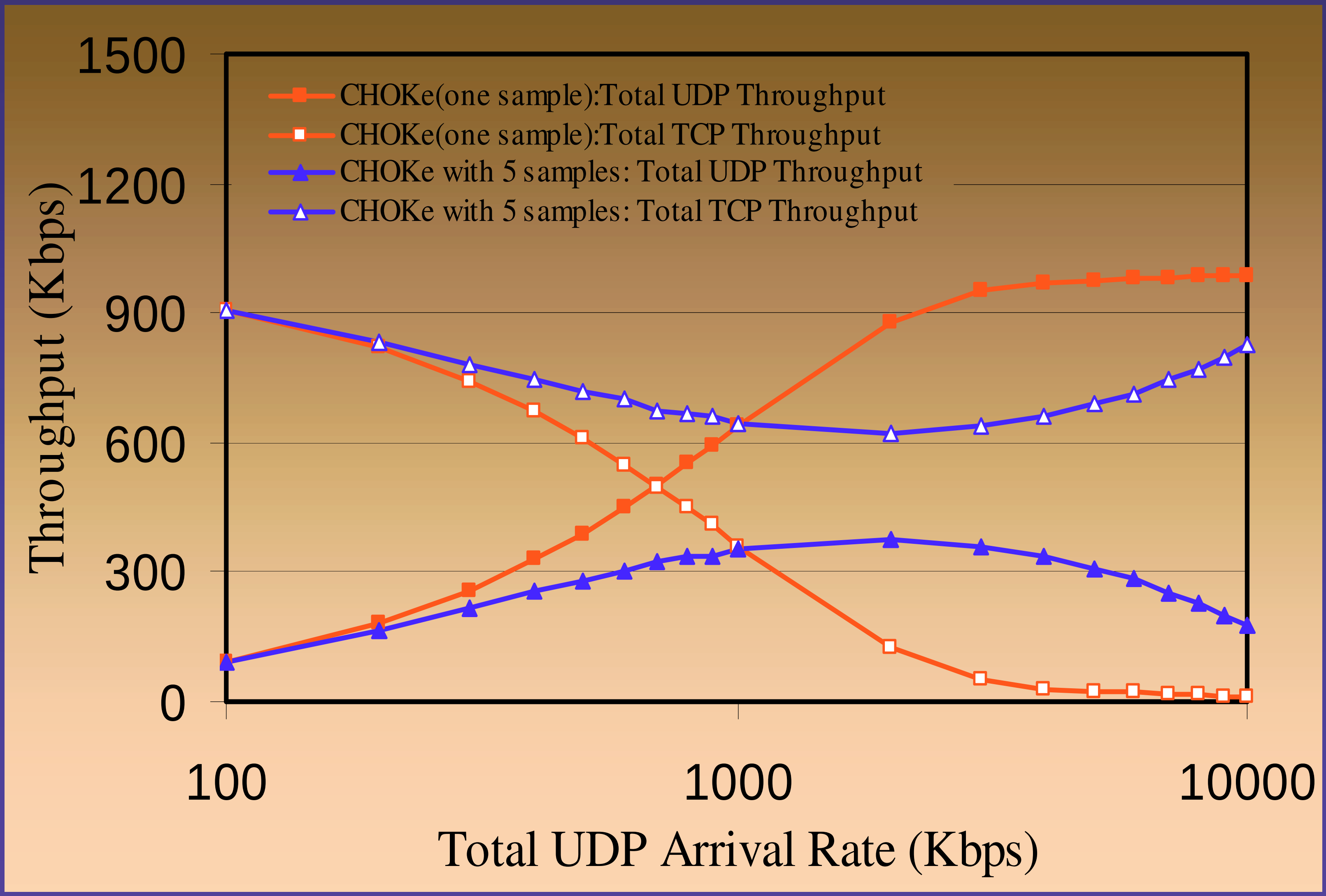
Network Setup Parameters

- 32 TCP flows, 1 UDP flow
- All TCP's maximum window size = 300
- All links have a propagation delay of 1ms
- FIFO buffer size = 300 packets
- All packets sizes = 1 KByte
- RED: $(min_{th}, max_{th}) = (100,200)$ packets

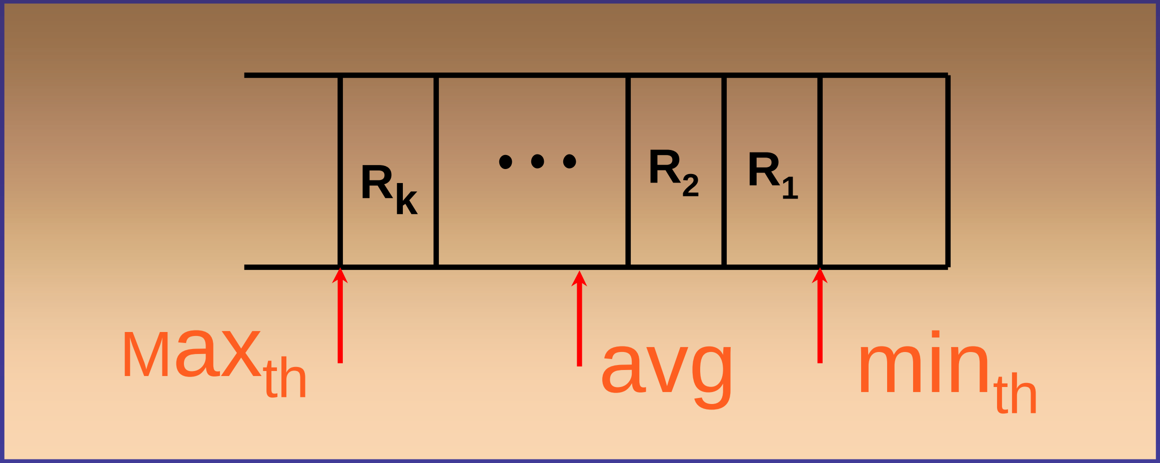
32 TCP, 1 UDP (one sample)



32 TCP, 5 UDP (5 samples)

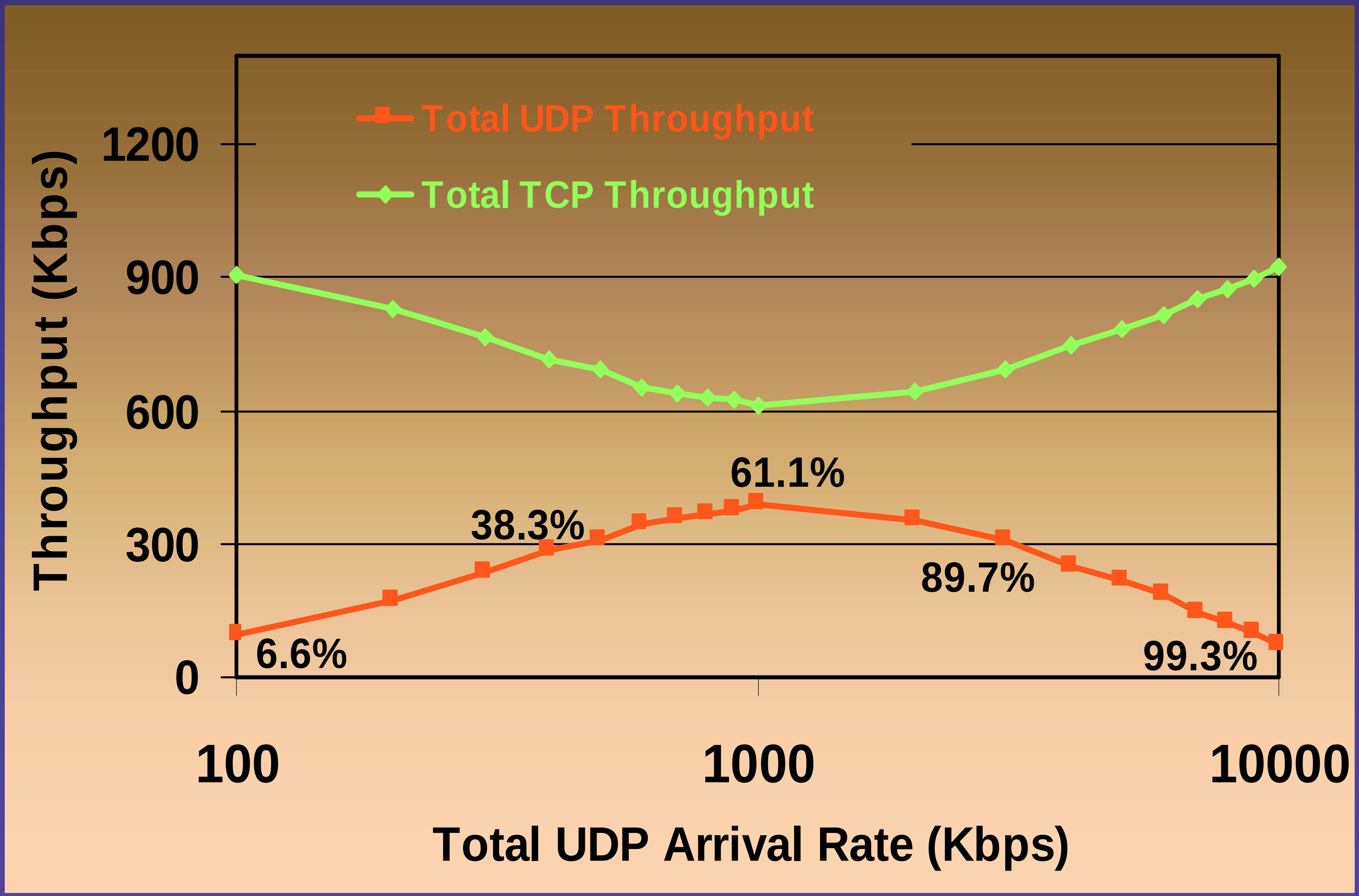


How Many Samples to Take?

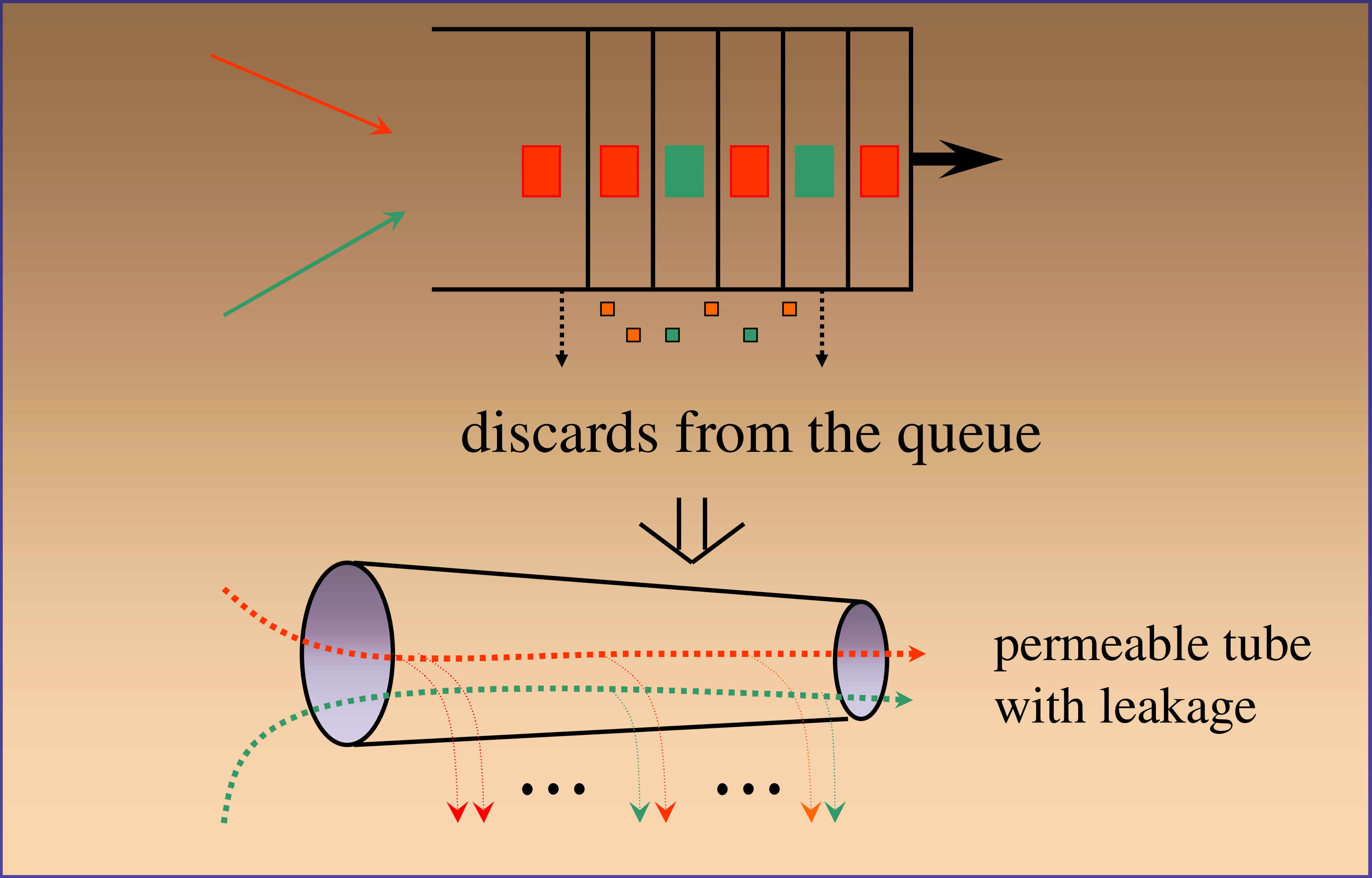


- Different samples for different $Qlen_{avg}$
  - *# samples ↓ when* $Qlen_{avg}$ *close to* min_{th}
  - *# samples ↑ when* $Qlen_{avg}$ *close to* max_{th}

32 TCP, 5 UDP (self-adjusting)



Analytical Model



Fluid Analysis

- *N*: the total number of packets in the buffer
- $L_i(t)$: the survial rate for flow i packets

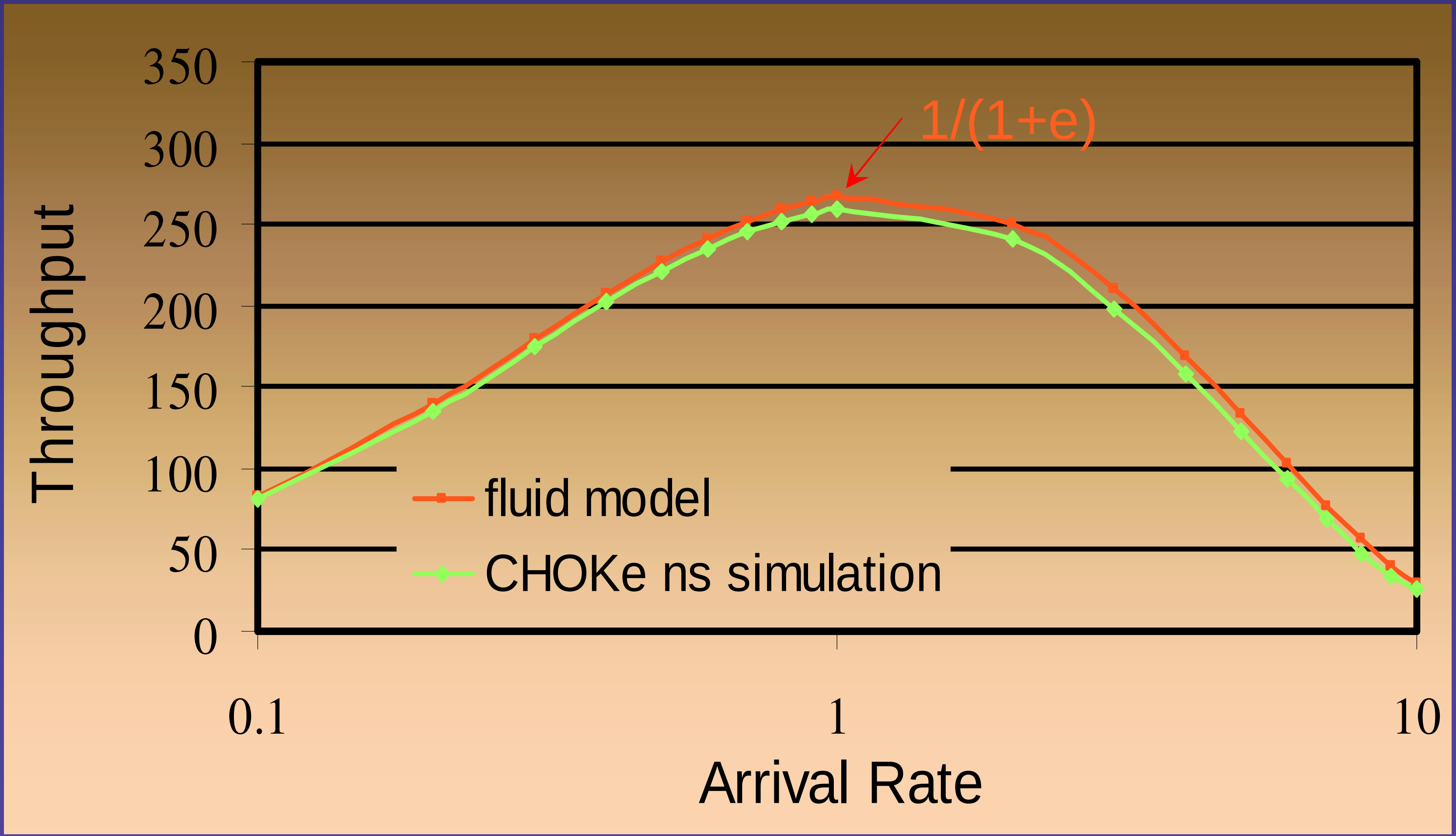
$$L_i(t)\delta t - L_i(t+\delta t)\delta t = \lambda_i \delta t\, L_i(t)\delta t / N$$

$$- dL_i(t)/dt = \lambda_i L_i(t)\, N$$

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

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

Model vs Simulation
- multiple TCPs and one UDP



Fluid Model
- Multiple samples

- Multiple samples are chosen

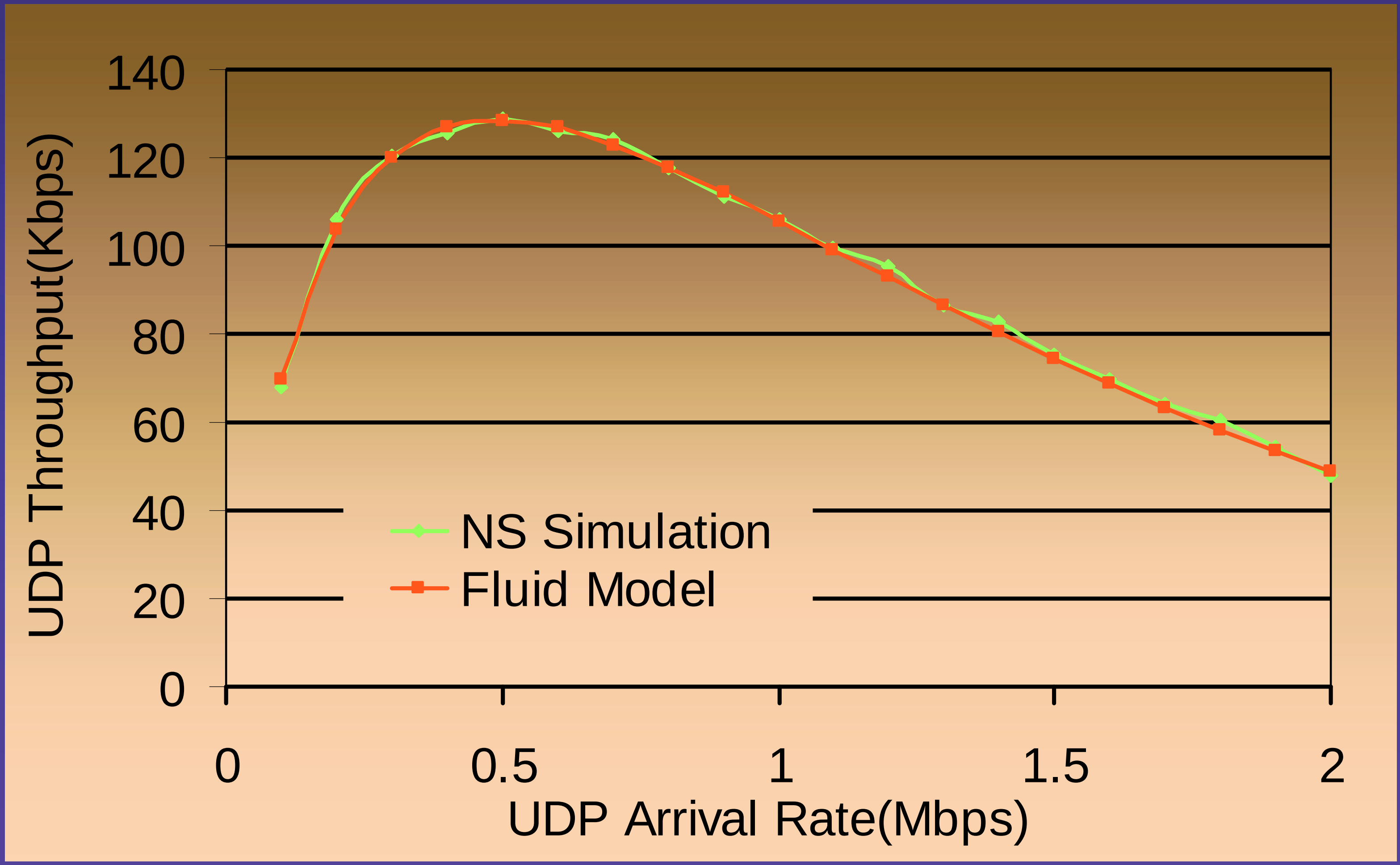
$$L_i(t)\delta t - L_i(t+\delta t)\delta t = M\lambda_i\, \delta t\, L_i(t)\delta t\, /N$$

$$-\, dL_i(t)/dt = M\lambda_i\, L_i(t)\, N$$

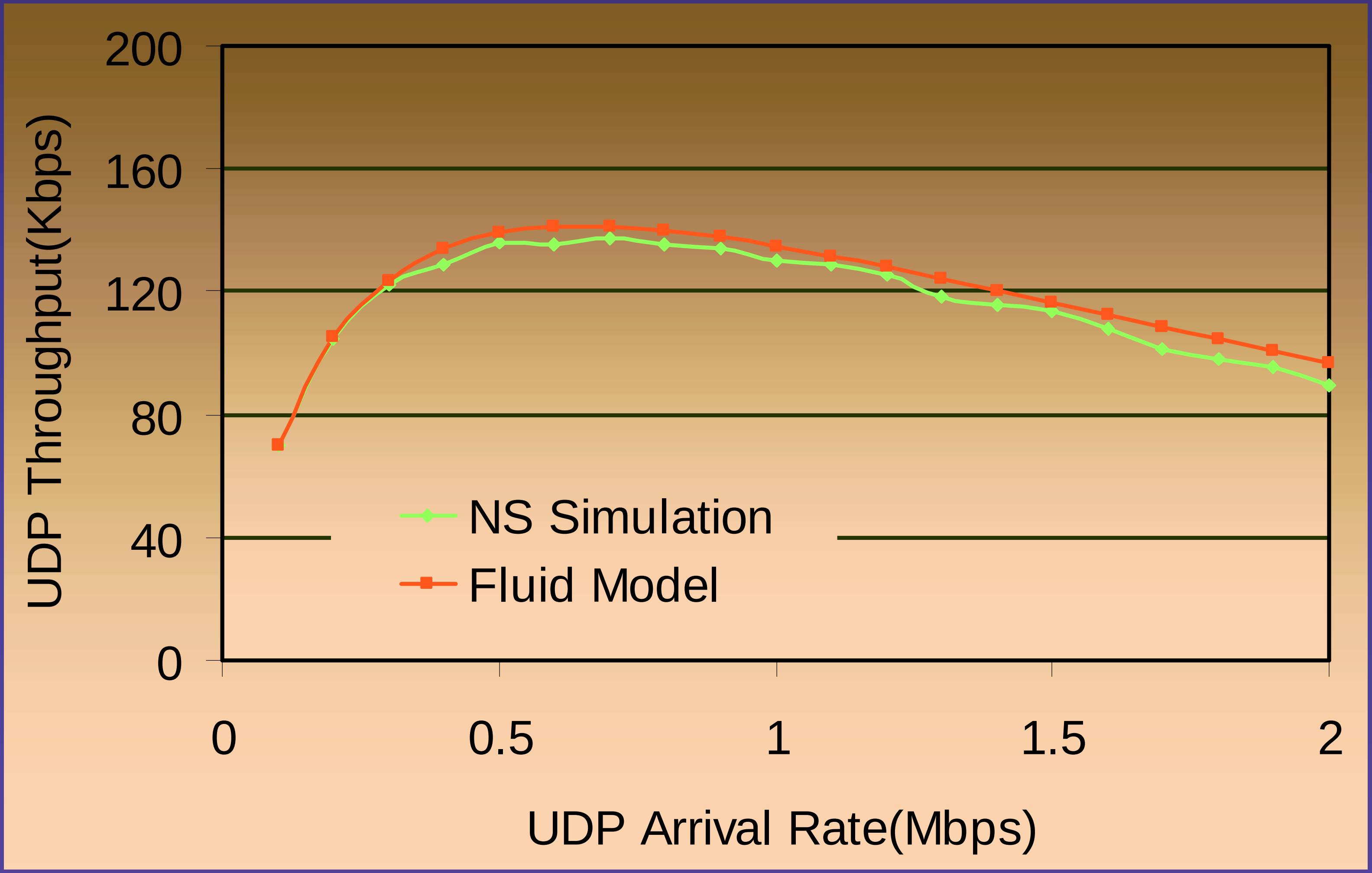
$$L_i(0) = \lambda_i\,(1-p_i)^M$$

$$L_i(D) = \lambda_i\,(1-p_i)^M - M\lambda_i\, p_i$$

Two Samples
- multiple TCPs and one UDP



Two Samples
- multiple TCPs and two UDP



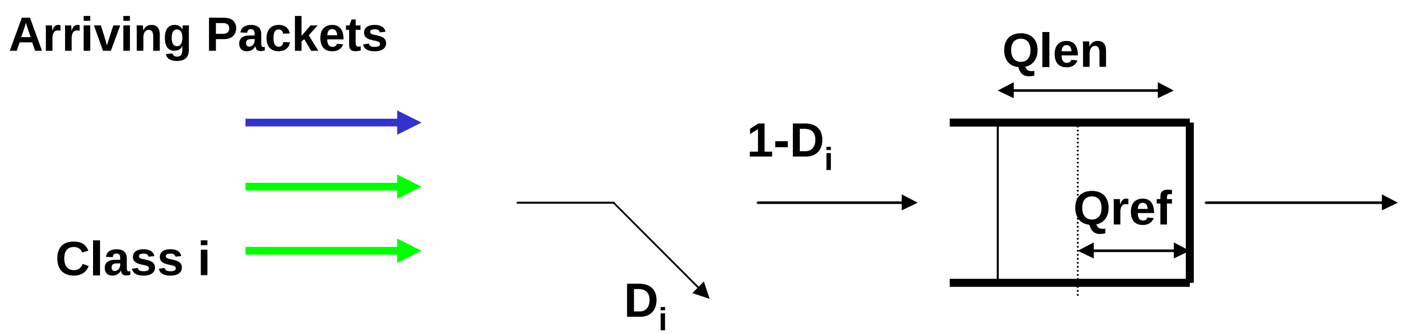
What If We Use a Small Amount of State?

AFD: Goal

- Approximate equal bandwidth allocation
  - *Not only AQM, approximate DRR scheduling*
  - *Provide soft queues in addition to physical queues*
- Keep the state requirement small
- Be simple to implement

AFD Algorithm: Details (Basic Case: Equal Share)

Di = Drop Probability for Class i



M_i = Arrival estimate for Class i (Bytes over interval T_s)

Mfair = Mfair - a (Qlen - Qref) + b (Qlen_old - Qref)

Fair Share

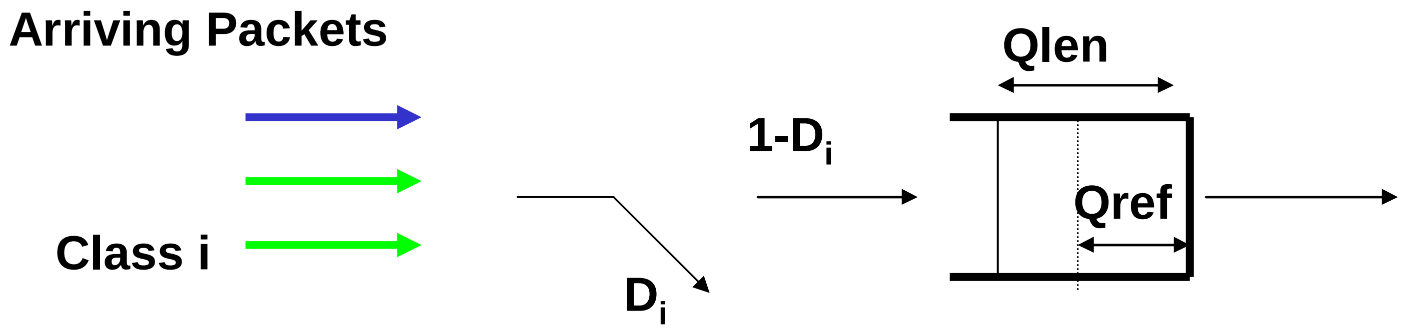
If $M_i \leq$ Mfair : No Drop ($D_i = 0$)

If $M_i >$ Mfair : $D_i > 0$ such that

$M_i (1-D_i) =$ Mfair

AFD Algorithm: Details (General Case)

Di = Drop Probability for Class i



M_i = Arrival estimate for Class i (Bytes over interval T_s)

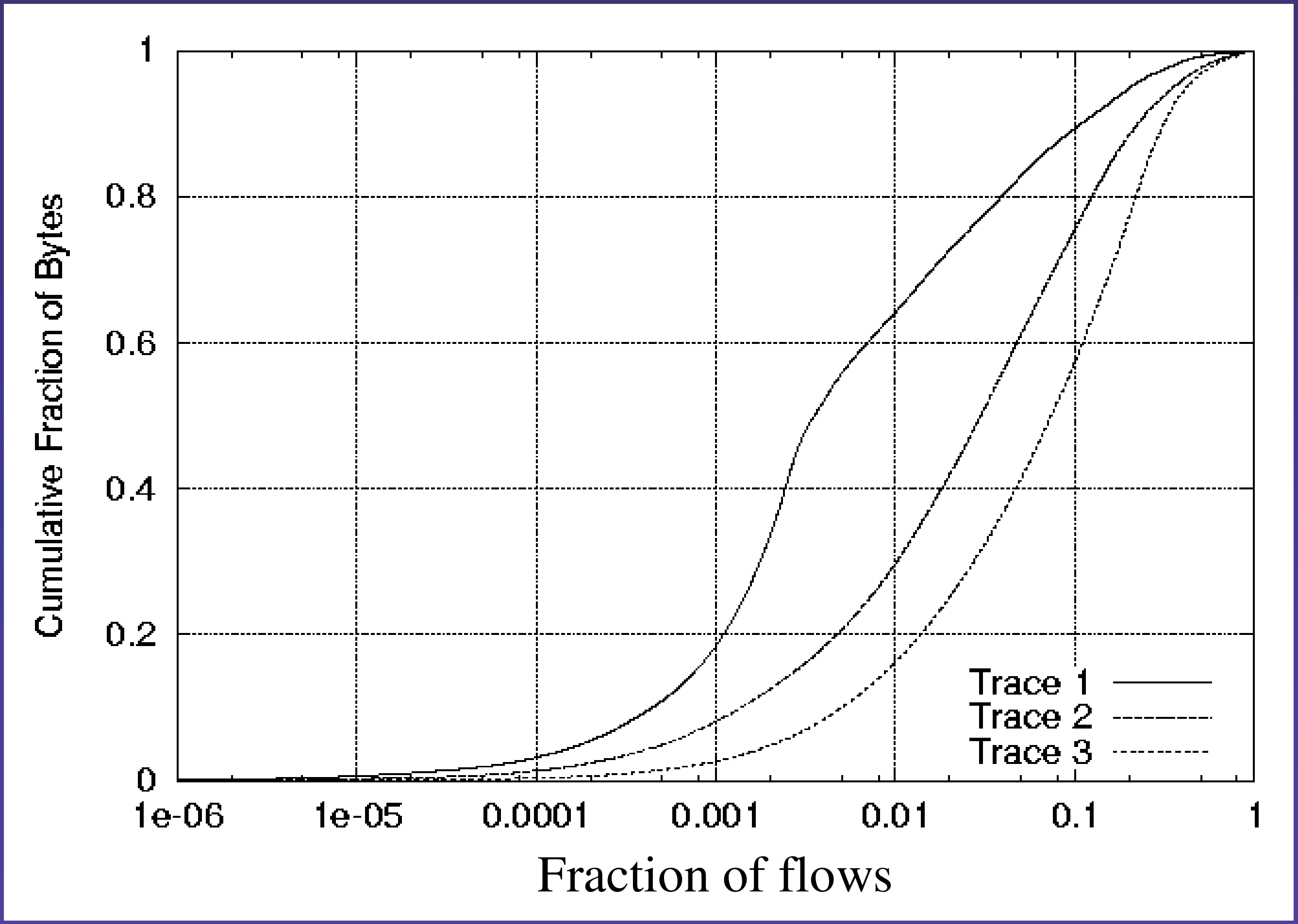
Mfair = Mfair - a (Qlen - Qref) + b (Qlen_old - Qref)

Fair Share

If $M_i \leq F(\text{Mfair}, \text{Min}_i, \text{Max}_i, W_i, \ldots)$: No Drop ($D_i = 0$)

If $M_i > \text{Mfair}$: $D_i > 0$ such that $M_i (1\text{-}D_i) = F(\text{Mfair}, \text{Min}_i, \text{Max}_i, W_i, \ldots)$

Not Per-Flow State

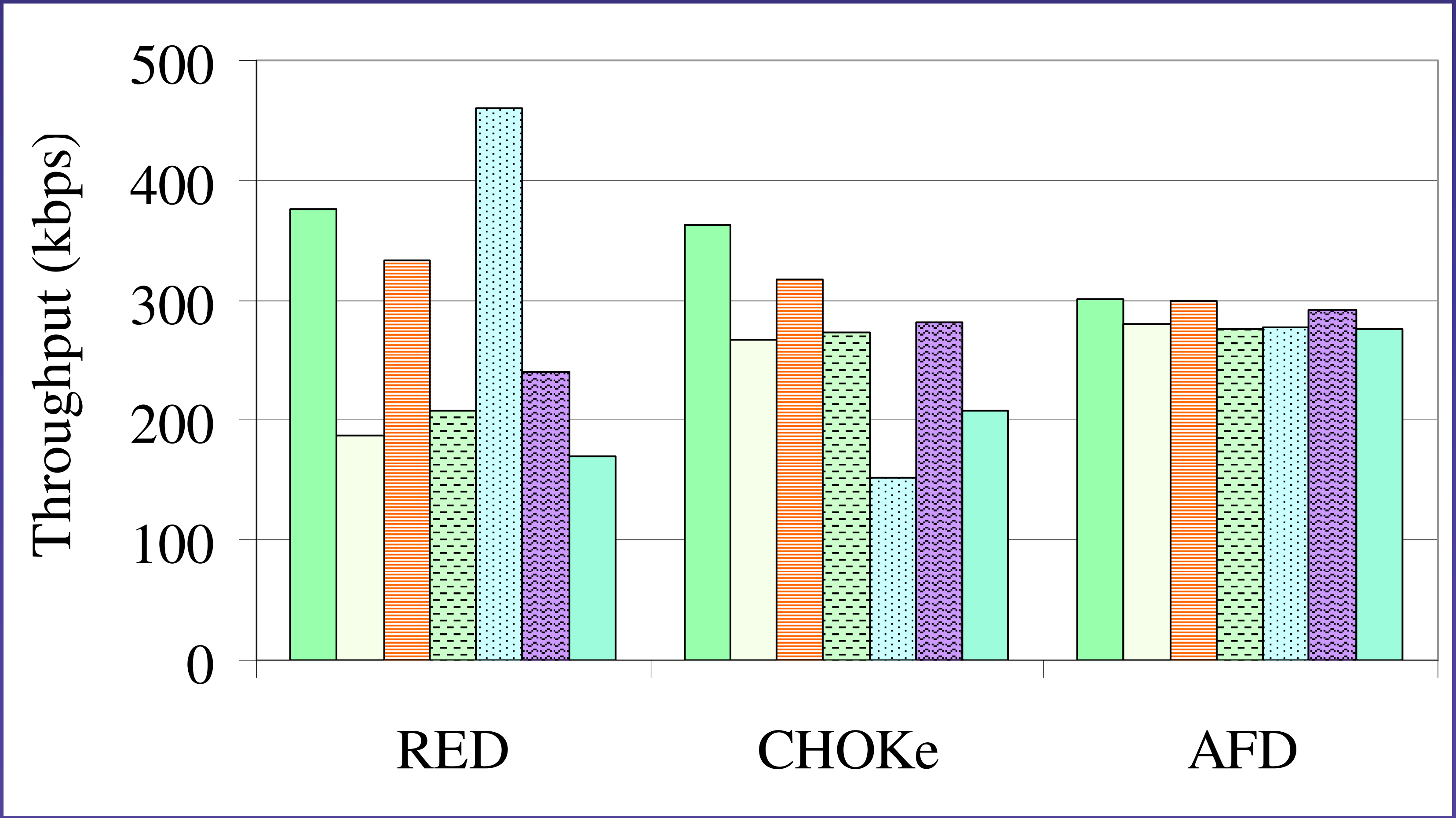


- State requirement on the order of # of unresponsive flows

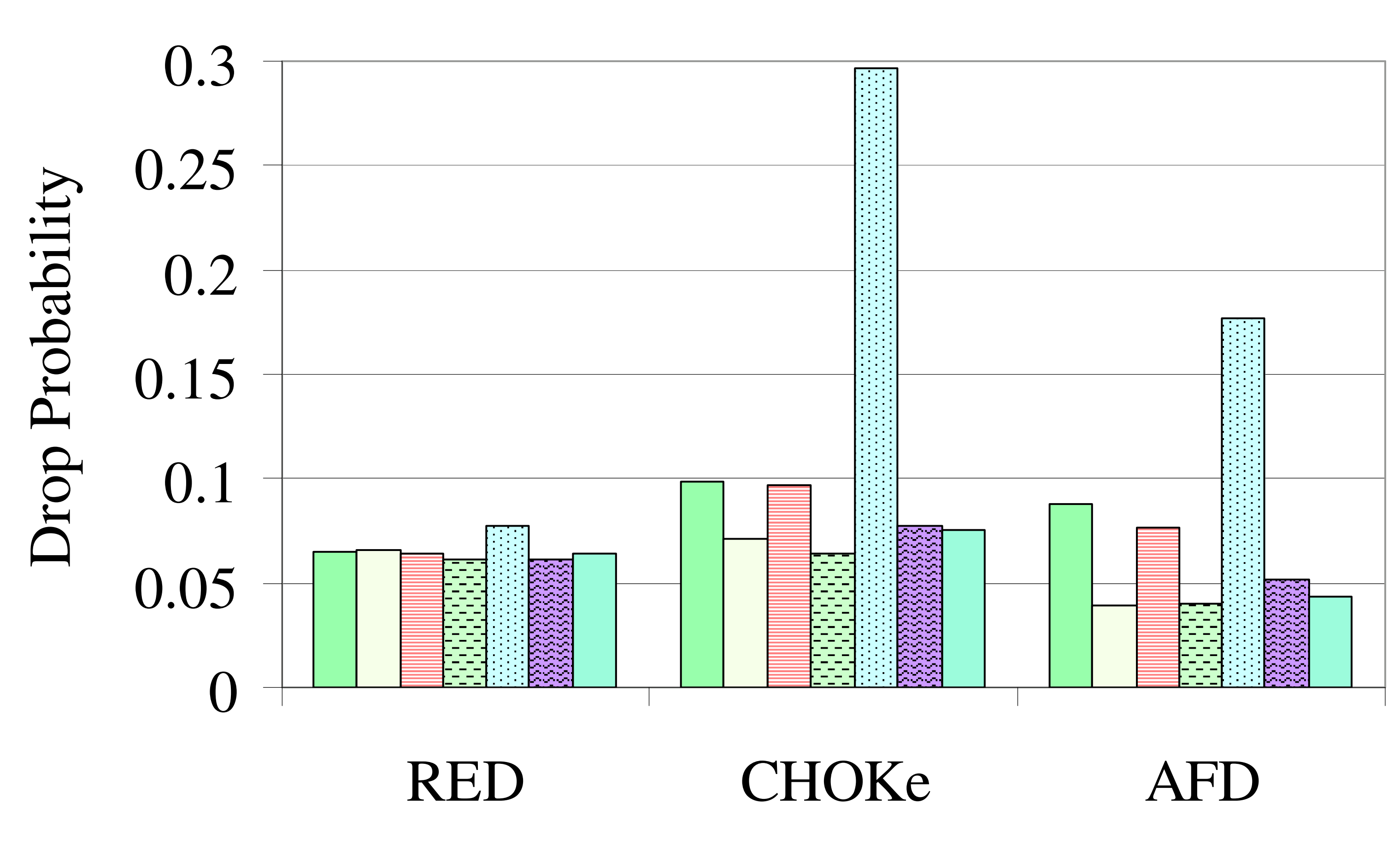
AFD Solution: Details

- Based on 3 simple mechanisms
  - *estimate per "class" arrival rate*
    - *counting per "class" bytes over fixed intervals (T_s)*
    - *potential averaging over multiple intervals*
  - *estimate deserved departure rate (so as to achieve the proper bandwidth allocation for the class)*
    - *Observation and averaging of queue length as measure of congestion*
    - *Functional definition of "fair share" based on fairness criterion*
  - *perform probabilistic dropping (pre-enqueue) to drive arrival rate to equal desired departure rate*

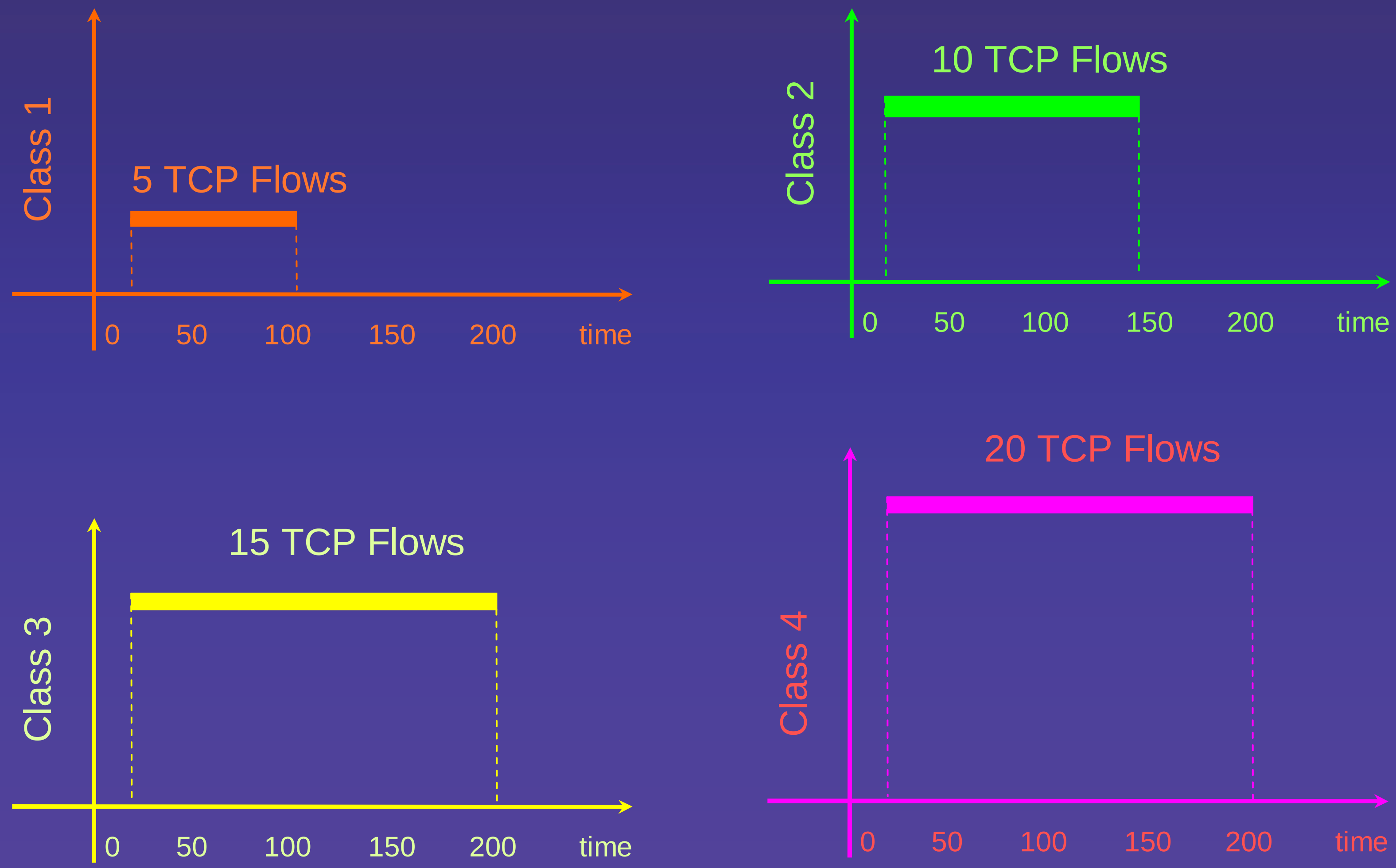
Mixed Traffic with Different Levels of Unresponsiveness



Drop Probabilities
(note differential dropping)

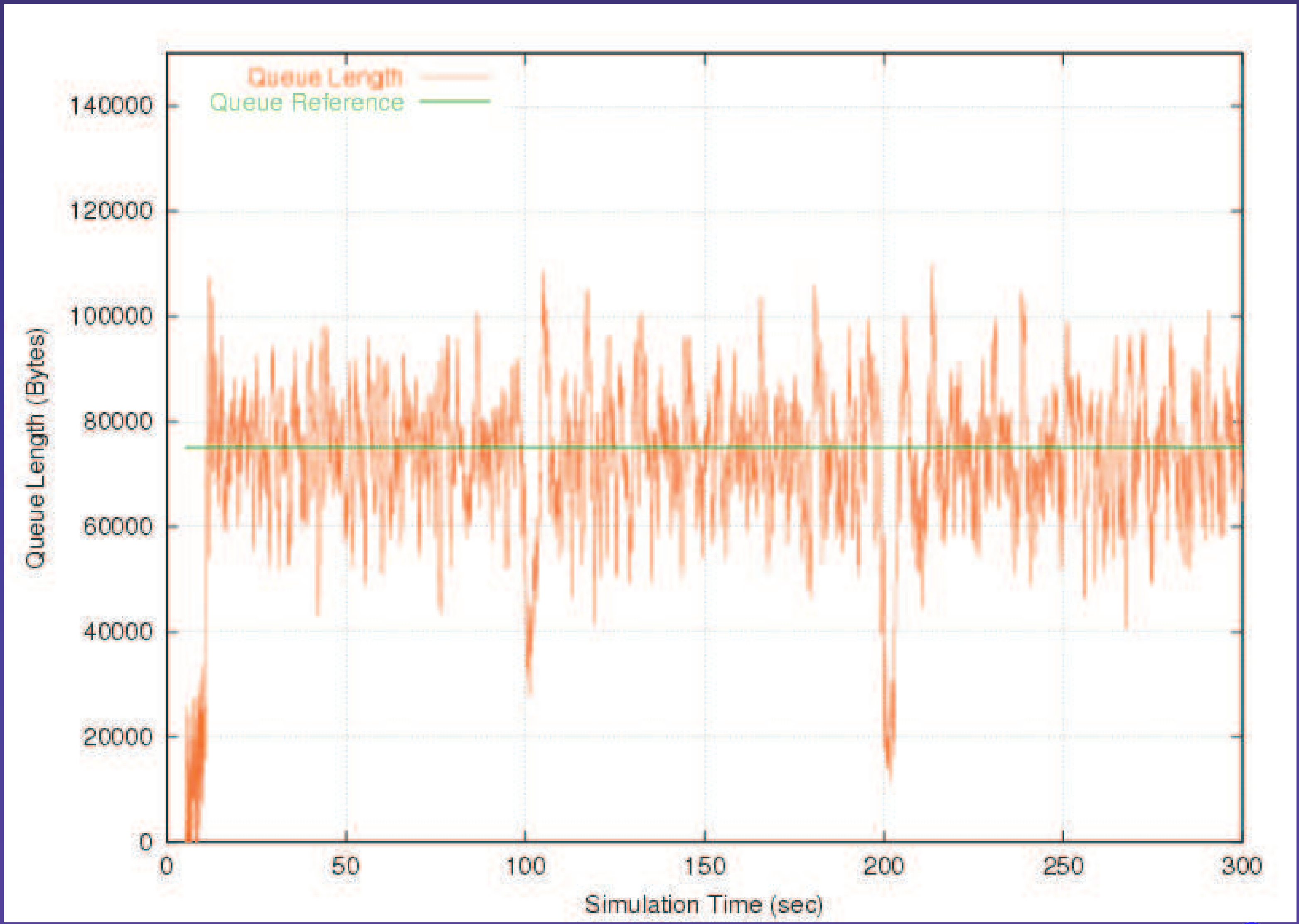


Different Number of TCP Flows in Each Class

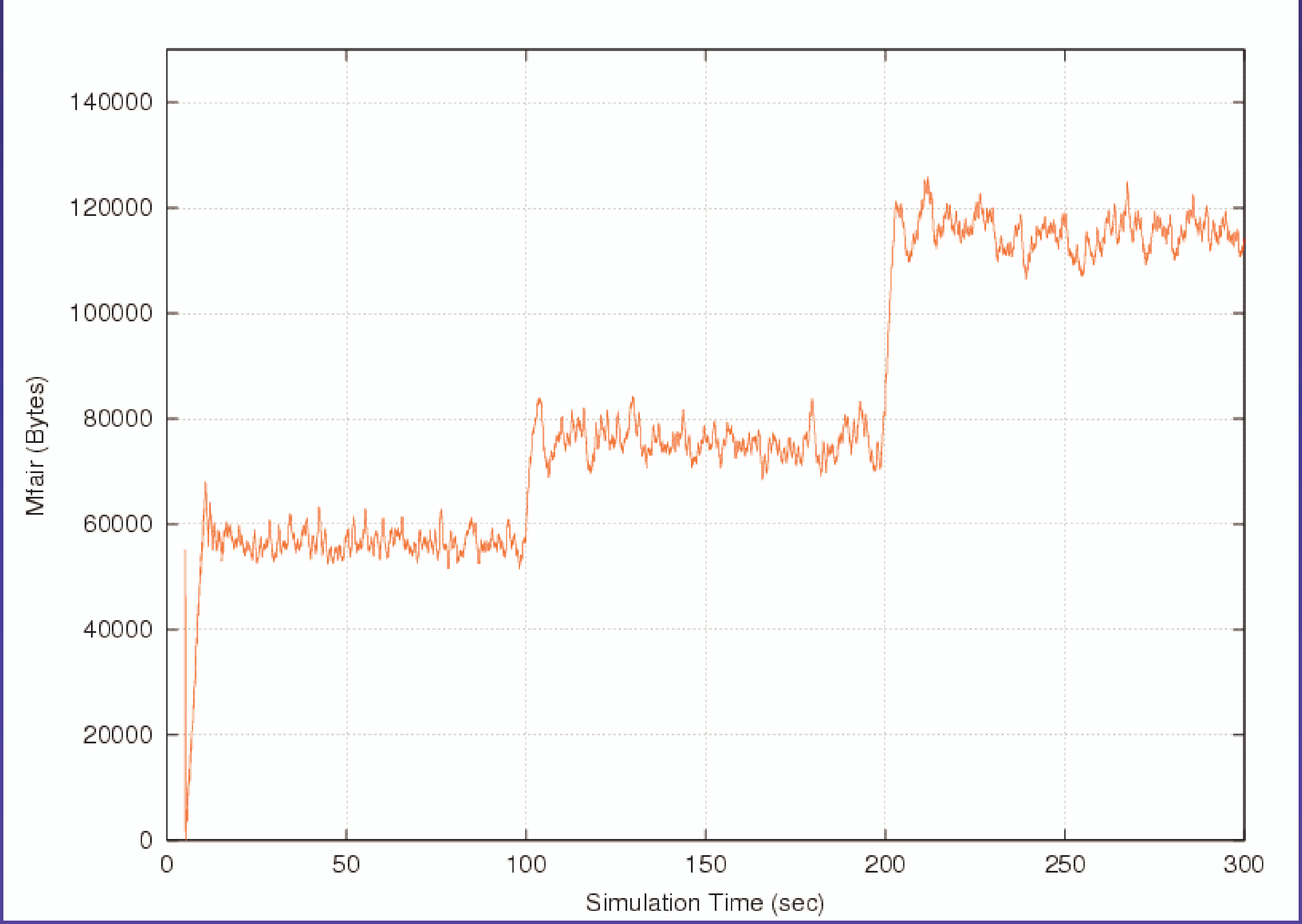


Different Class Throughput Comparison

Queue Length



Mfair



Mfair (Bytes)
140000
120000
100000
80000
60000
40000
20000
0
0
50
100
150
200
250
300
Simulation Time (sec)

AFD Implementation Issues

- Monitor Arrival Rate
- Determine Drop Probability
- Maximize Link Utilization

Arrival Monitoring

- Keep a counter for each class
  - *Count the data arrivals (in bytes) of each class in 10ms interval:* arv_i
- Arrival rate of each class is updated every 10ms
  - $m_i = m_i(1-1/2^c)+arv_i$
  - *c determines the average window*

Implementing the Drop Function

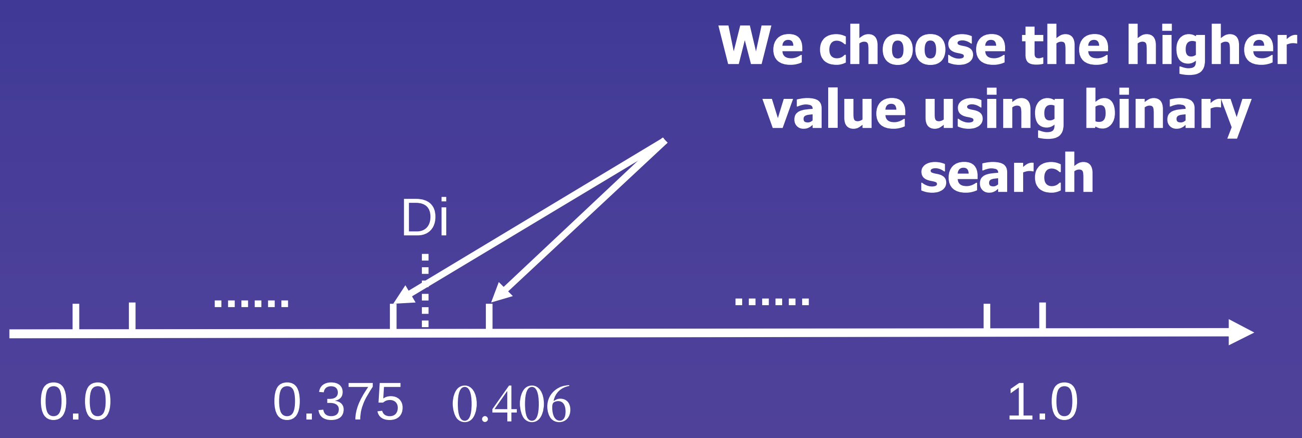
- If $M_i \leq$ Mfair then $D_i = 0$
- Otherwise, rewrite the drop function as

$$D_i = (1 - \frac{m_{fair}}{m_i})$$
$$\Rightarrow m_i(1 - D_i) = m_{fair}$$
$$\Rightarrow m_i D_i = m_i - m_{fair}$$

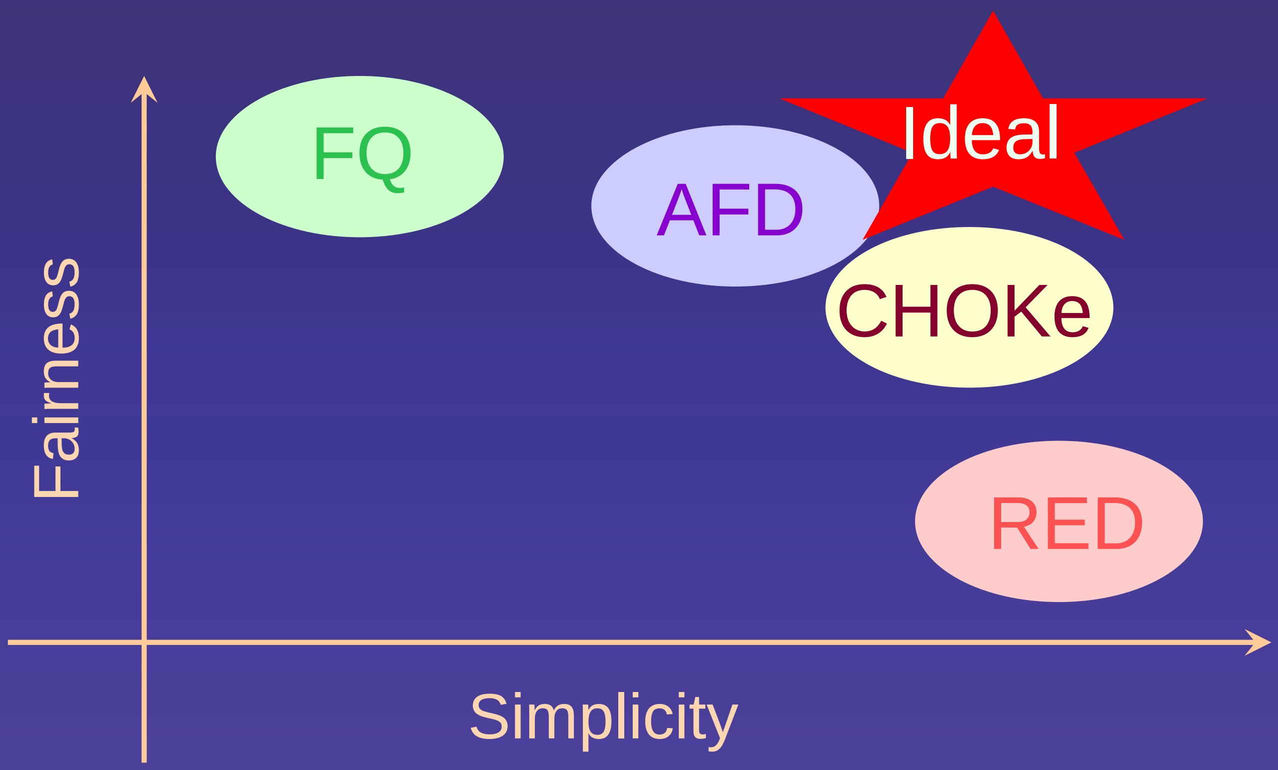
- Suppose we have predetermined drop levels, find the one such that D_i* M_i = (M_i – Mfair)

Implementing the Drop Function

- **Drop levels are: 1/32, 1/16, 3/32…**
- **Suppose m_i = 100, m_{fair} = 62.0 => D_i = 0.380,**



AFD - Summary



- Equal share is approximated in a wide variety of settings
- The state requirement is limited

Summary

- Traditional Queue Management
  - *Drop Tail, Drop Front, Drop Random*
  - *Problems: lock-out, full queue, global synchronization, bias against bursty traffic*

- Active Queue Management
  - *RED: can't handle unresponsive flows*
  - *CHOKe: penalize unresponsive flows*
  - *AFD: provides approximate fairness with limited states*