Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High Speed Networks

Ion Stoica,Scott Shenker, and Hui Zhang

SIGCOMM'98, Vancouver, August 1998

subsequently

IEEE/ACM Transactions on Networking 11(1), 2003, pp. 33-46.

Presented by Bob Kinicki

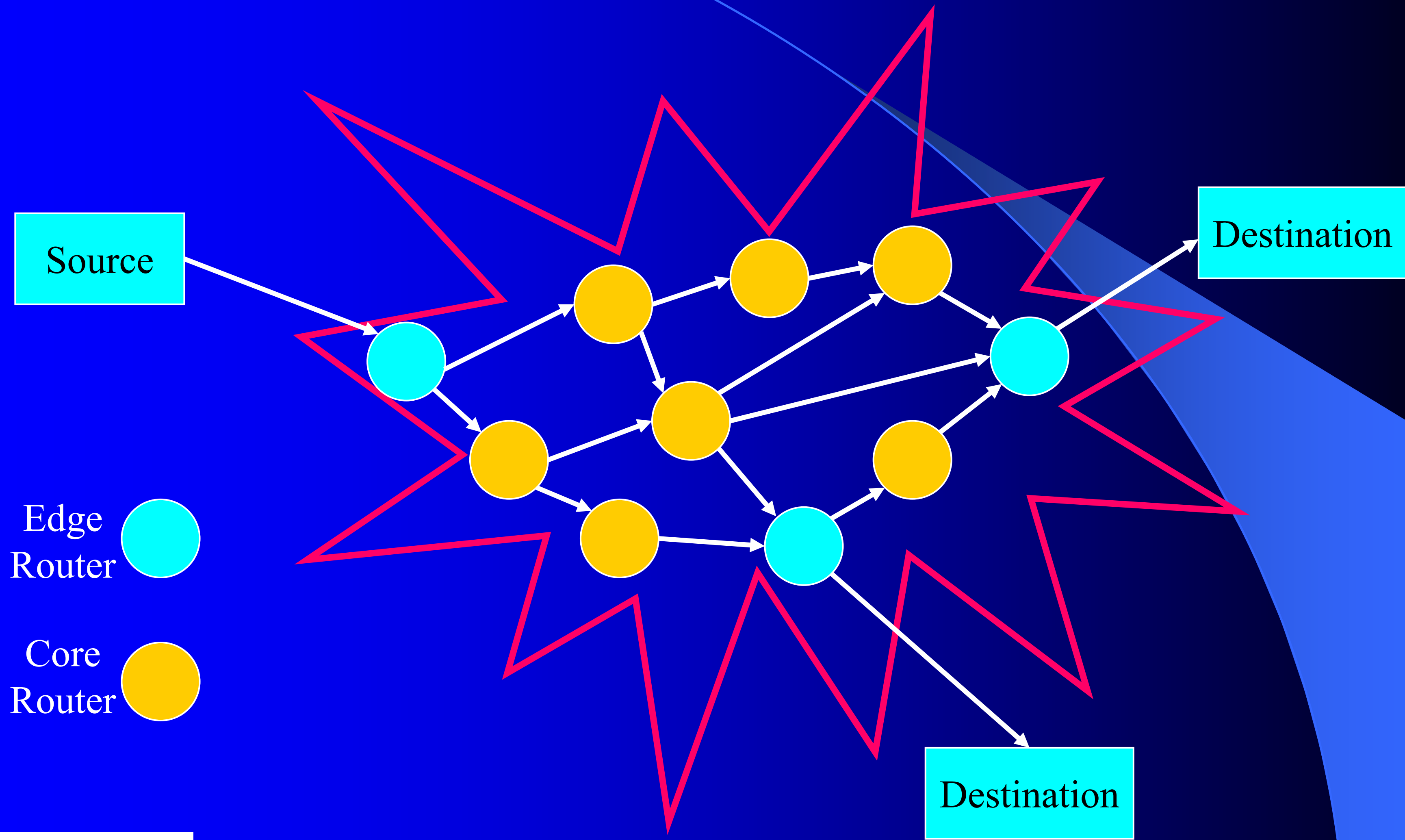
Outline

- **Introduction**
- Core-Stateless Fair Queueing (CSFQ)
  - Fluid Model Algorithm
  - Packet Algorithm
  - Flow Arrival Rate
  - Link Fair Share Rate Estimation
- NS Simulations
- Conclusions

Introduction

- This paper brings forward the concept of “fair” allocation.
- The claim is that fair allocation inherently requires routers to maintain *state* and perform operations on a per flow basis.
- The authors present an architecture and a set of algorithms that is “approximately” fair while using FIFO queueing at internal routers.

An "Island" of Routers



Outline

- Introduction
- **Core-Stateless Fair Queueing (CSFQ)**
  - Fluid Model Algorithm
  - Packet Algorithm
  - Flow Arrival Rate
  - Link Fair Share Rate Estimation
- NS Simulations
- Conclusions

Core-Stateless Fair Queueing

- Ingress edge routers compute per-flow rate estimates and insert these estimates as **labels** into each packet header.
- Only edge routers maintain per flow state.
- Labels are updated at each router based only on aggregate information.
- FIFO queuing with probabilistic dropping of packets on input is employed at the core routers.

Edge – Core Router Architecture

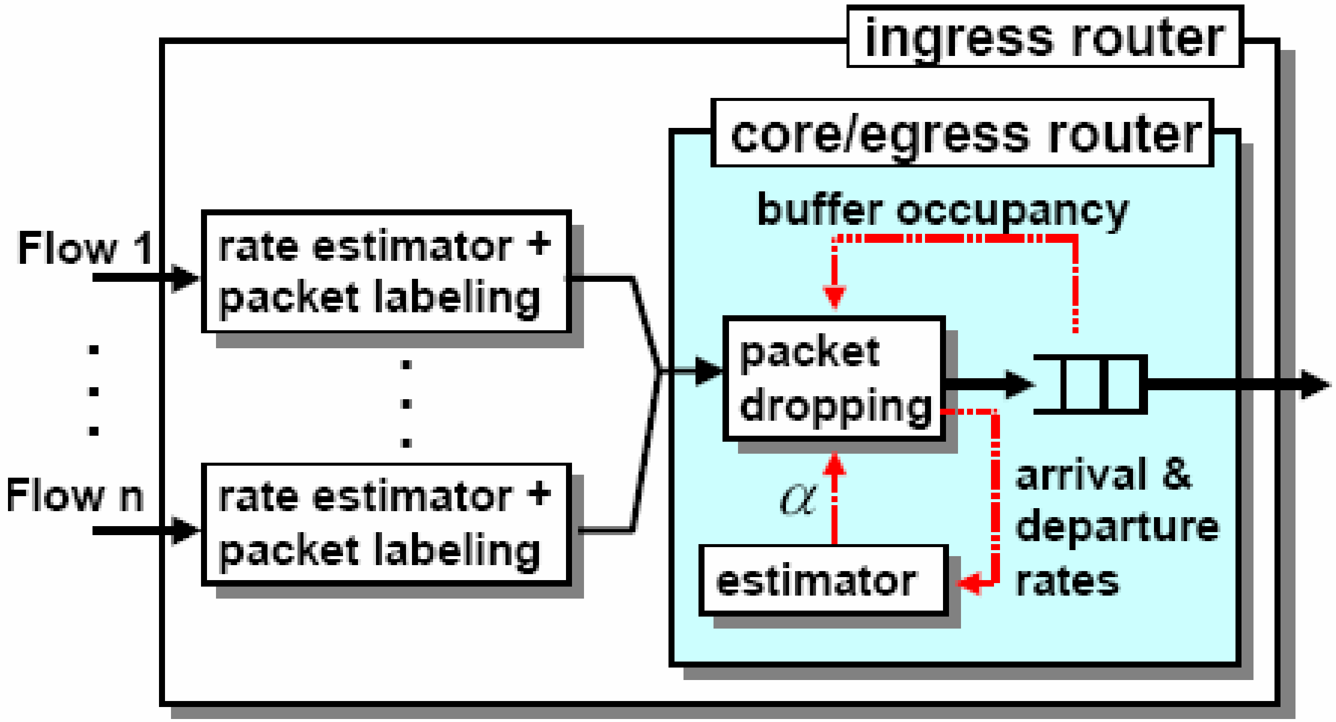


Fig. 2. The architecture of the output port of an edge router, and a core router, respectively.

Fluid Model Algorithm

- Assume the bottleneck router has an output link with capacity *C*.
- Assume each flow's arrival rate, $r_i(t)$, *is known precisely.*

The main idea is that max-min fair bandwidth allocations are characterized such that all flows that are bottlenecked by a router have the same output rate.

- This rate is called the *fair share rate* of the link.
- Let $\alpha(t)$ be the fair share rate at time t.

Fluid Model Algorithm

If max-min bandwidth allocations are achieved, each flow receives service at a rate given by

$$\min (r_i(t), \alpha(t))$$

Let $A(t)$ denote the total arrival rate: $A(t) = \sum_{i=1}^{n} r_i(t)$

If $A(t) > C$ then the fair share is the unique solution to

$$C = \sum_{i=1}^{n} \min(r_i(t), \alpha(t)),$$

Fluid Model Algorithm

Thus, the probabilistic fluid forwarding algorithm that achieves fair bandwidth allocation is:

Each incoming bit of flow **i** is dropped with probability

$$\max(0, 1 - \alpha(t)/r_i(t)) \qquad (2)$$

These dropping probabilities yield fair share arrival rates at the next hop.

Packet Algorithm

- Moving from a bit-level, buffer-less fluid model to a packet-based, buffer model leaves two challenges:
  - Estimate the flow arrival rates $r_i(t)$
  - Estimate the fair share $\alpha(t)$
- This is possible because the rate estimator incorporates the packet size.

Flow Arrival Rate

At each edge router, use exponential averaging to estimate the rate of a flow. For flow i, let

l_i^k be the length of the k^{th} packet.

t_i^k be the arrival time of the k^{th} packet.

Then the estimated rate of flow i, r_i is updated every time a new packet is received:

$$r_i^{new} = (1-e^{-T/K}) * L / T + e^{-T/K} * r_i^{old} \qquad (3)$$

where

$$T = T_i^k = t_i^k - t_i^{k-1}$$

$L = l_i^k$ and K is a constant

Link Fair Rate Estimation

- If we denote the estimate of the fair share by $\widehat{\alpha}(t)$ and the acceptance rate by $F(\widehat{\alpha}(t))$, we have

$$F(\widehat{\alpha}(t)) = \sum_{i=1}^{n} \min\left(r_i(t), \widehat{\alpha}(t)\right)$$

- Note – if we know $r_i(t)$ then $\widehat{\alpha}(t)$ can be determined by finding the unique solution to $F(x) = C$.
- However, this requires per-flow state !

Heuristic Algorithm

- The heuristic algorithm needs three aggregate state variables: $\widehat{\alpha}(t)$, $\widehat{A}$, $\widehat{F}$ where $\widehat{A}$ is the estimated aggregate arrival rate.
- When a packet arrives, the router computes:

$$\widehat{A}_{new} = (1 - e^{-T/K_\alpha})\frac{l}{T} + e^{-T/K_\alpha}\widehat{A}_{old} \qquad (5)$$

- and similarly computes $\widehat{F}$.

CSFQ Algorithm

- When a packet arrives, $\widehat{A}$ is updated using exponential averaging (equation 5).
- If the packet is dropped, $\widehat{F}$ remains the same.
- If the packet is not dropped, $\widehat{F}$ is updated using exponential averaging.
- At the end of an epoch (defined by K_c), if the link is congested during the whole epoch, update $\widehat{\alpha}(t)$:

$$\widehat{\alpha}_{new} = \widehat{\alpha}_{old} \frac{C}{\widehat{F}}$$

CSFQ Algorithm (cont.)

- If the link is not congested, $\widehat{\alpha}_{new}$ is set to the largest rate of any active flow.
- $\widehat{\alpha}_{new}$ feeds into the calculation of drop probability, p, for the next arriving packet as α in

$$p = \max(0, 1 - \alpha / label)$$

CSFQ Algorithm (cont.)

- Estimation inaccuracies may cause $\widehat{F}$ to exceed link capacity.
- Thus, to limit the effect of Drop Tail buffer overflows, every time buffer overflows $\widehat{\alpha}_{new}$ is decreased by 1% in simulations.

CSFQ Pseudo Code

```
on receiving packet p
  if (edge router)
    i = classify(p);
    p.label = estimate_rate(r_i, p); // use Eq. (3)
  prob = max(0, 1 − α/p.label);
  if (prob > unif_rand(0, 1))
    α = estimate_α(p, 1);
    drop(p);
  else
    α = estimate_α(p, 0);
    enqueue(p);
    if (prob > 0)
      p.label = α; // relabel p
```

Figure 3

CSFQ Pseudo Code

```
estimate_α (p, dropped)
  // α̂ and α_K_c are initialized to 0;
  // α_K_c is used to compute the largest packet label seen
  // during a window of size K_c
  Â = estimate_rate(Â, p); // est. arrival rate (use Eq. (5))
  if (dropped == FALSE)
    F̂ = estimate_rate(F̂, p); // est. accepted traffic rate
  if (Â ≥ C)
    if (congested == FALSE)
      congested = TRUE;
      start_time = crt_time;
      if (α̂ == 0)
        // α̂ can be set to 0 if no packet is received
        // during a widow of size K_c
        α̂ = max(p.label, α_K_c);
    else
      if (crt_time > start_time + K_c)
        α̂ = α̂ × C/F̂;
        start_time = crt_time;
  else // Â < C
    if (congested == TRUE)
      congested = FALSE;
      start_time = crt_time;
      α_K_c = 0;
    else
      if (crt_time < start_time + K_c)
        α_K_c = max(α_K_c, p.label);
      else
        α̂ = α_K_c;
        start_time = crt_time;
        α_K_c = 0;
  return α̂;
```

Label Rewriting

- At core routers, outgoing rate is merely the minimum between the incoming rate and the fair rate, α .
- Hence, the packet label L can be rewritten by

$$L_{new} = \min(L_{old}, \alpha)$$

Outline

- Introduction
- Core-Stateless Fair Queueing (CSFQ)
  - Fluid Model Algorithm
  - Packet Algorithm
  - Flow Arrival Rate
  - Link Fair Share Rate Estimation
- **NS Simulations**
- Conclusions

Simulations

- A major effort of the paper is to compare CSFQ to four algorithms via ns-2 simulations.
- FIFO
- RED
- FRED (Flow Random Early Drop)
- DRR (Deficit Round Robin)

FRED (Flow Random Early Drop)

- Maintains per flow state in router.
- FRED preferentially drops a packet of a flow that has either:
  - Had many packets dropped in the past
  - A queue larger than the average queue size
- Main goal : Fairness
- FRED-2 guarantees to each flow a minimum number of buffers.

DRR (Deficit Round Robin)

- Represents an efficient implementation of WFQ.
- A sophisticated per-flow queueing algorithm.
- Scheme assumes that when router buffer is full the packet from the longest queue is dropped.
- Can be viewed as “best case” algorithm with respect to fairness.

ns-2 Simulation Details

- Use TCP, UDP, RLM and On-Off traffic sources in separate simulations.
- Bottleneck link: 10 Mbps, 1ms latency, 64KB buffer
- RED, FRED (min, max) thresholds: (16KB, 32KB)
- K and K_c = 100 ms. K_α = 200ms.

A Single Congested Link

- First Experiment : 32 UDP CBR flows
  - Each UDP flow is indexed from 0 to 31 with flow 0 sending at 0.3125 Mbps and each of the *i* subsequent flows sending *(i+ 1)* times its fair share of 0.3125 Mbps.
- Second Experiment : 1 UDP CBR flow, 31 TCP flows
  - UDP flow sends at 10 Mbps
  - 31 TCP flows share a <u>single</u> 10 Mbps link.

Figure 5b: 32 UDP Flows

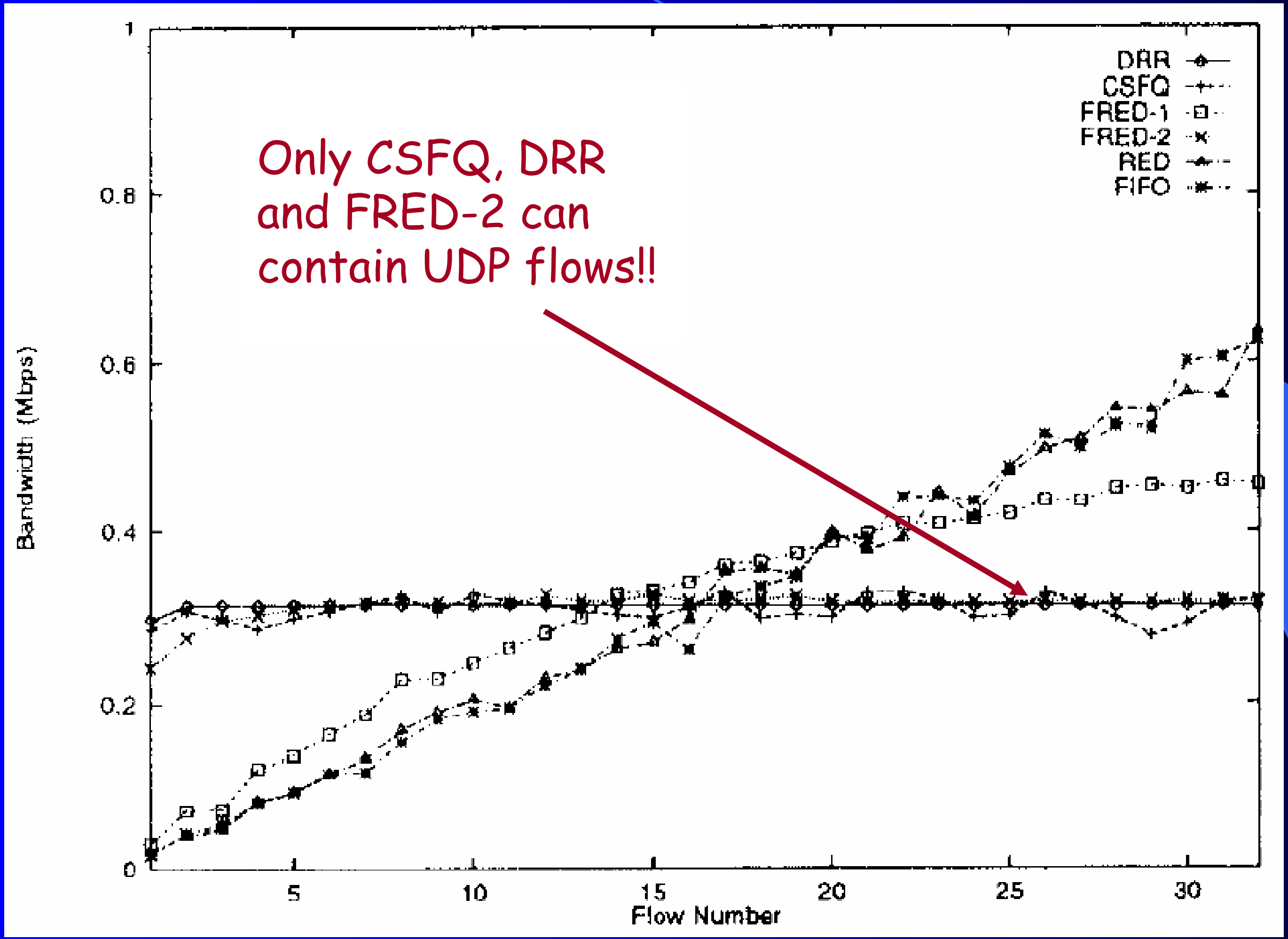
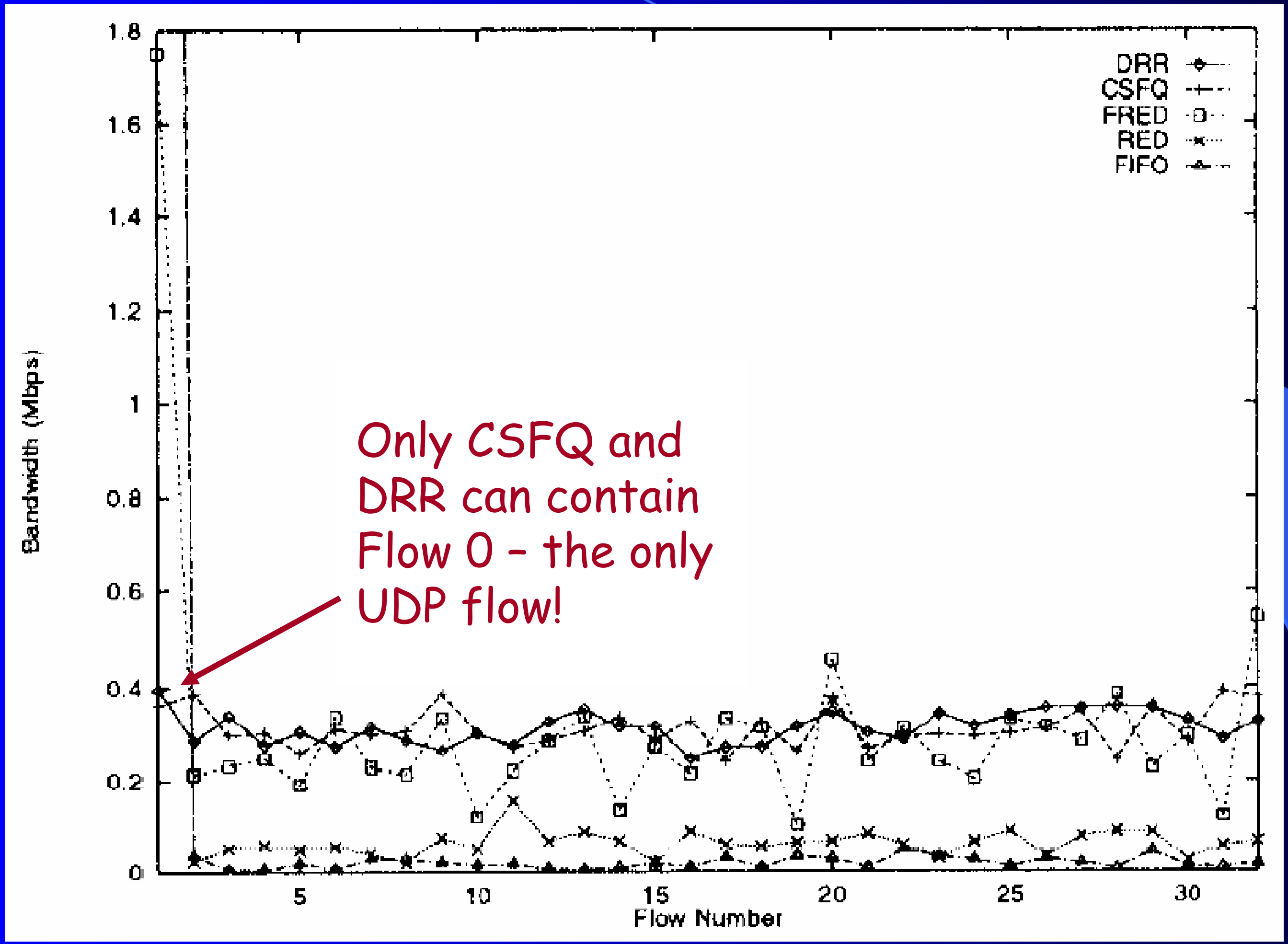


Figure 6b : One UDP Flow, 31 TCP Flows



A Single Congested Link

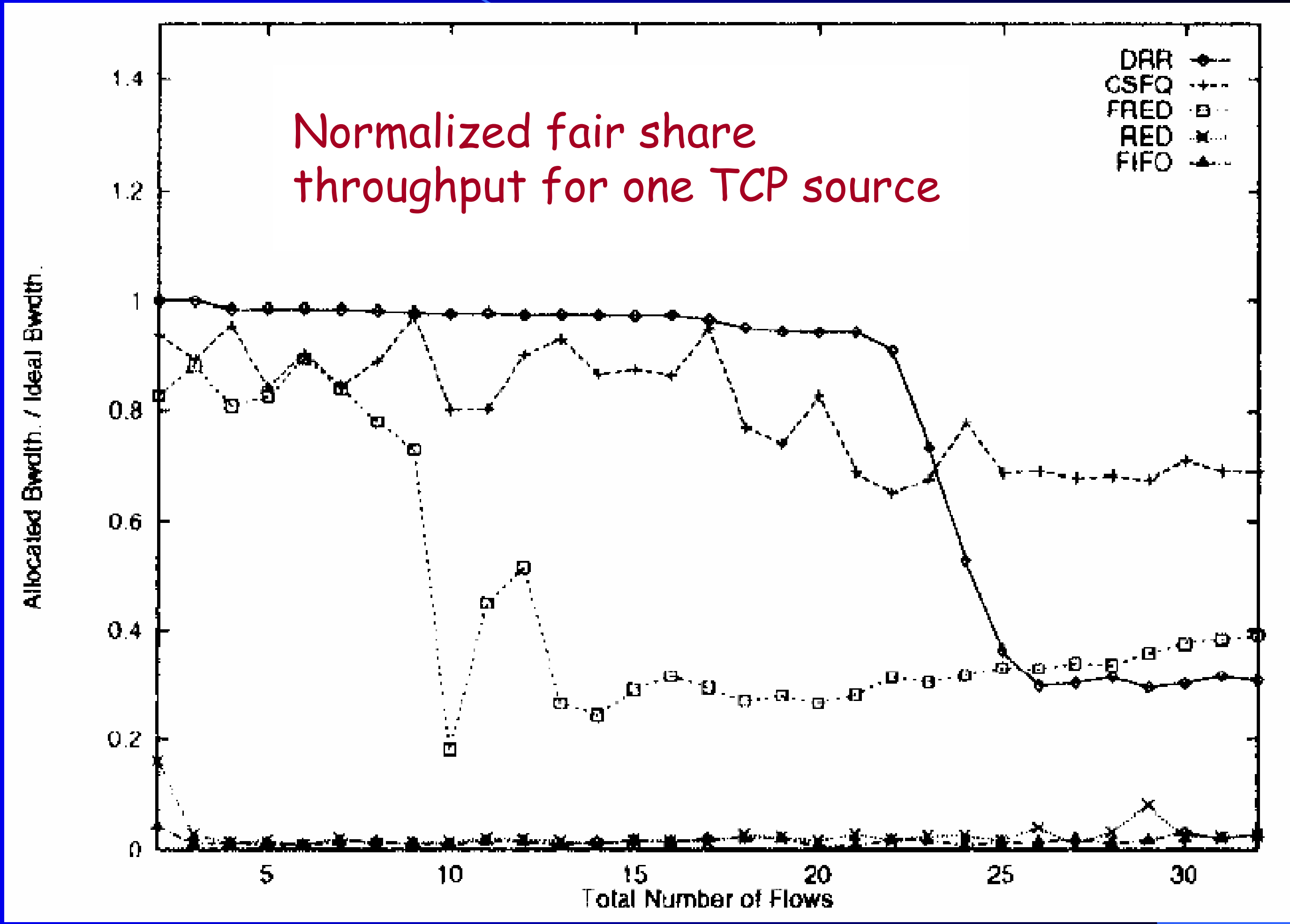
- Third Experiment Set : 31 simulations
  – Each simulation has a different N,
    N = 2 … 32.
  – One TCP and N-1 UDP flows with each UDP flow sending at <u>twice</u> fair share rate of 10/N Mbps.

Figure 6b : One TCP Flow, N-1 UDP Flows

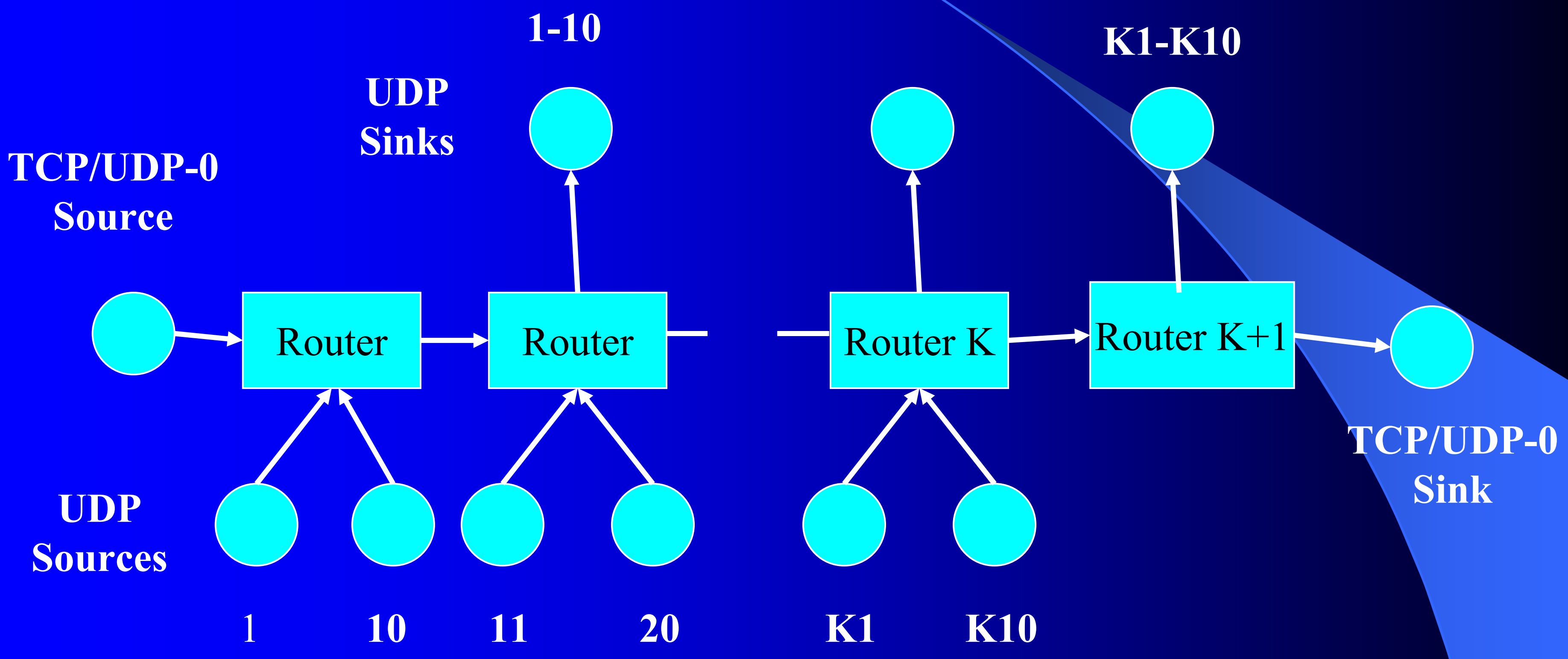
DRR good for less than 22 flows.

CSFQ better than DRR when a large number of flows.

CSFQ beats FRED.



Multiple Congested Links



Multiple Congested Links

- First experiment : UDP flow 0 sends at its fair share rate, 0.909 Mbps while the other ten "crossing" UDP flows send at 2 Mbps.
- Second experiment: Replace the UDP flow with one TCP flow and leave the ten crossing UDP flows.

Figure 8a : UDP source

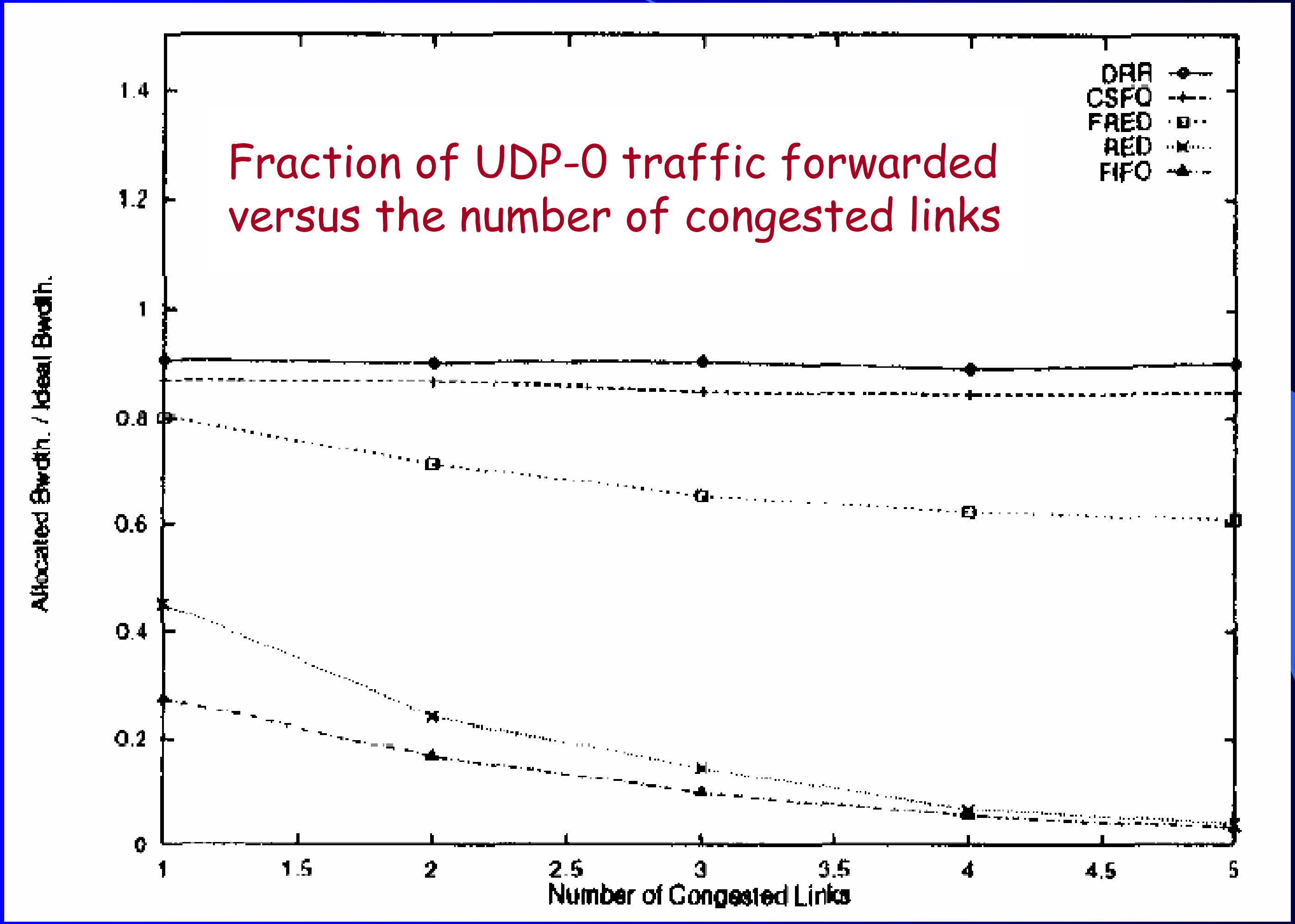
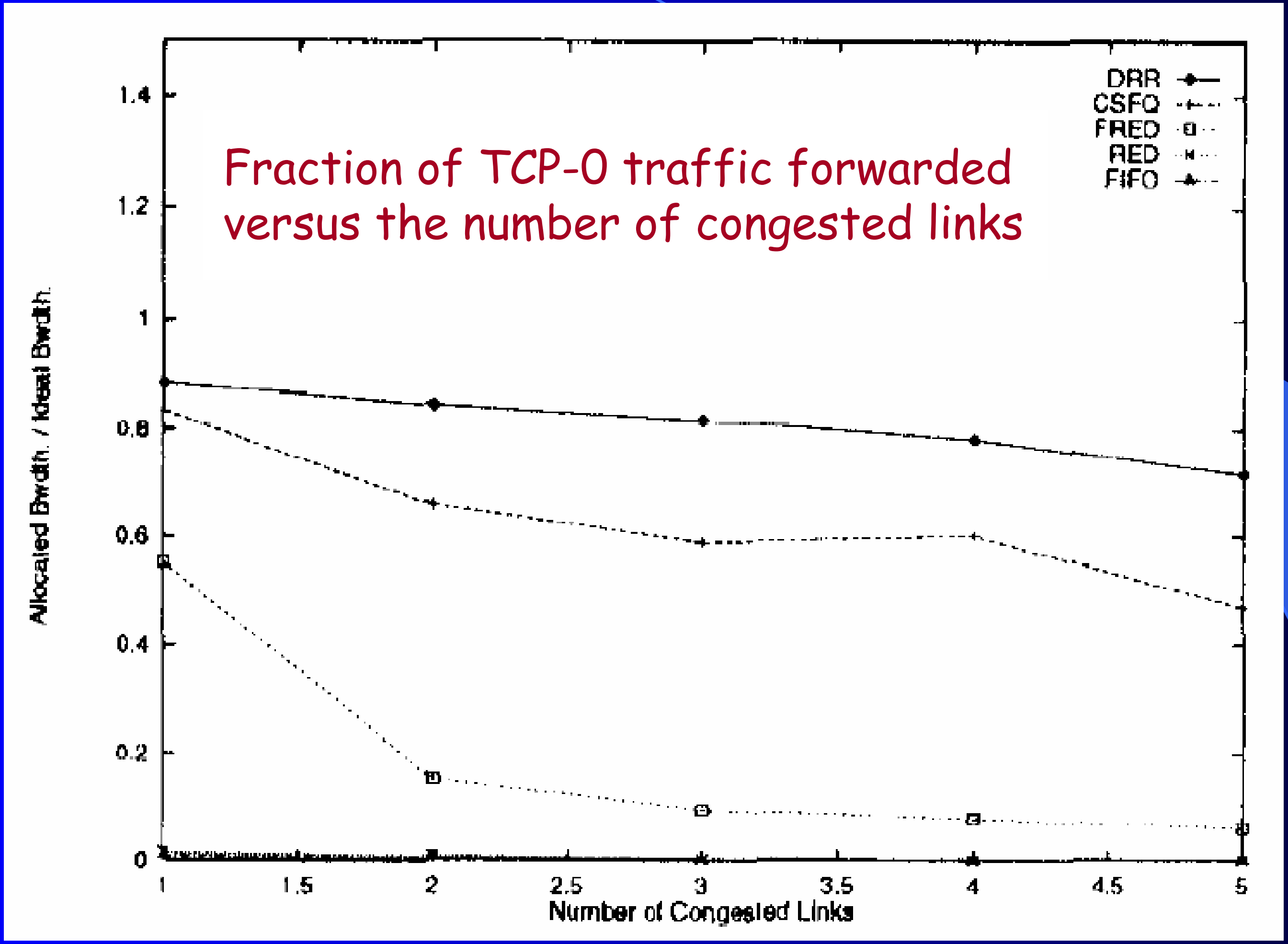


Figure 8b : TCP Source



Receiver-driven Layered Multicast

- RLM is an adaptive scheme in which the source sends the information encoded in a number of layers.
- Each layer represents a different multicast group.
- Receivers join and leave multicast groups based on packet drop rates experienced.

Receiver-driven Layered Multicast

- Simulation of three RLM flows and one TCP flow with a 4 Mbps link.
- Fair share for each is 1 Mbps.
- Since router buffer set to 64 KB, K, K_c, and K_α are set to 250 ms.
- Each RLM layer I sends 2^{i+4} Kbps with each receiver subscribing to the first five layers.

Figure 9b : FRED

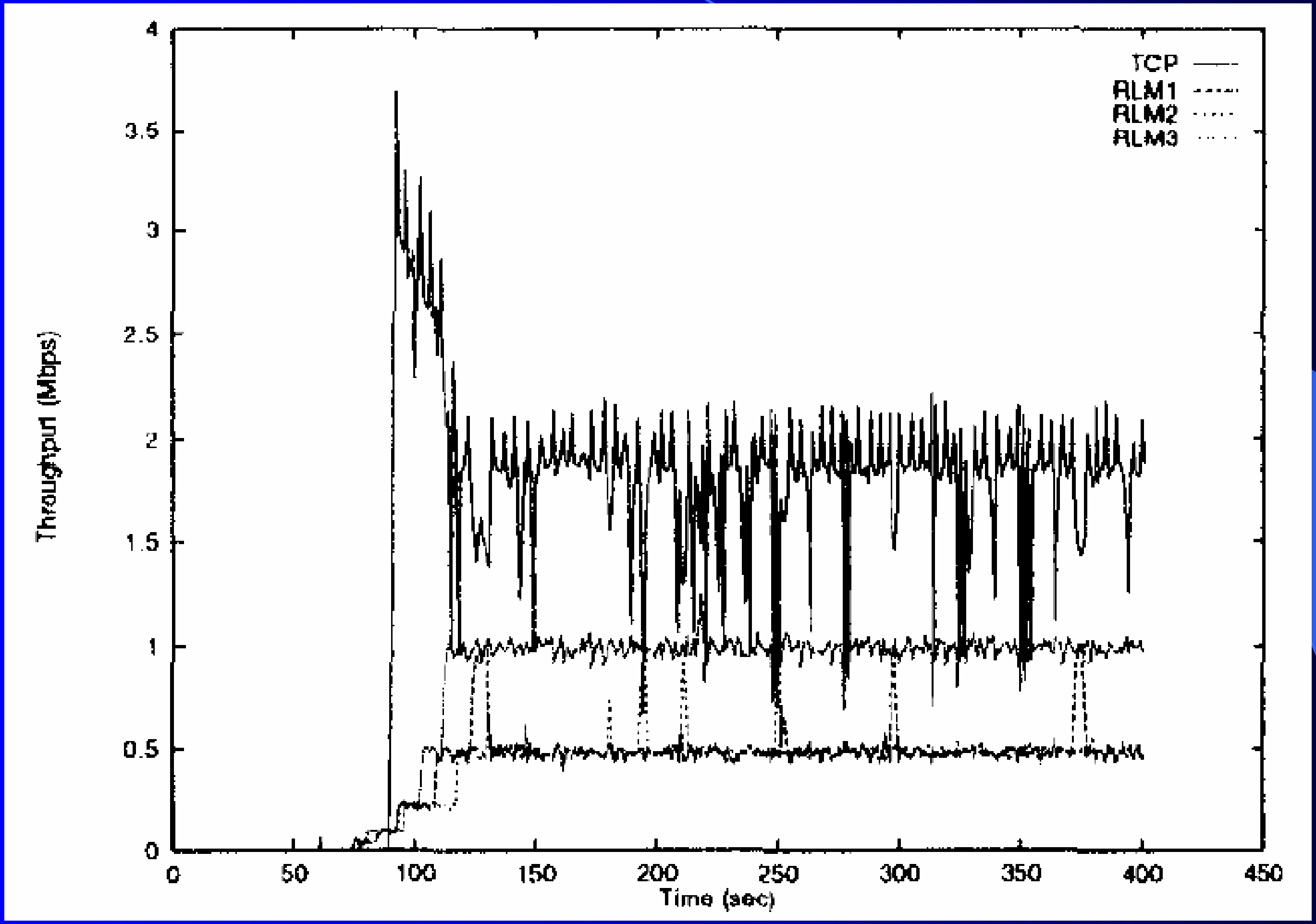


Figure 9e : RED

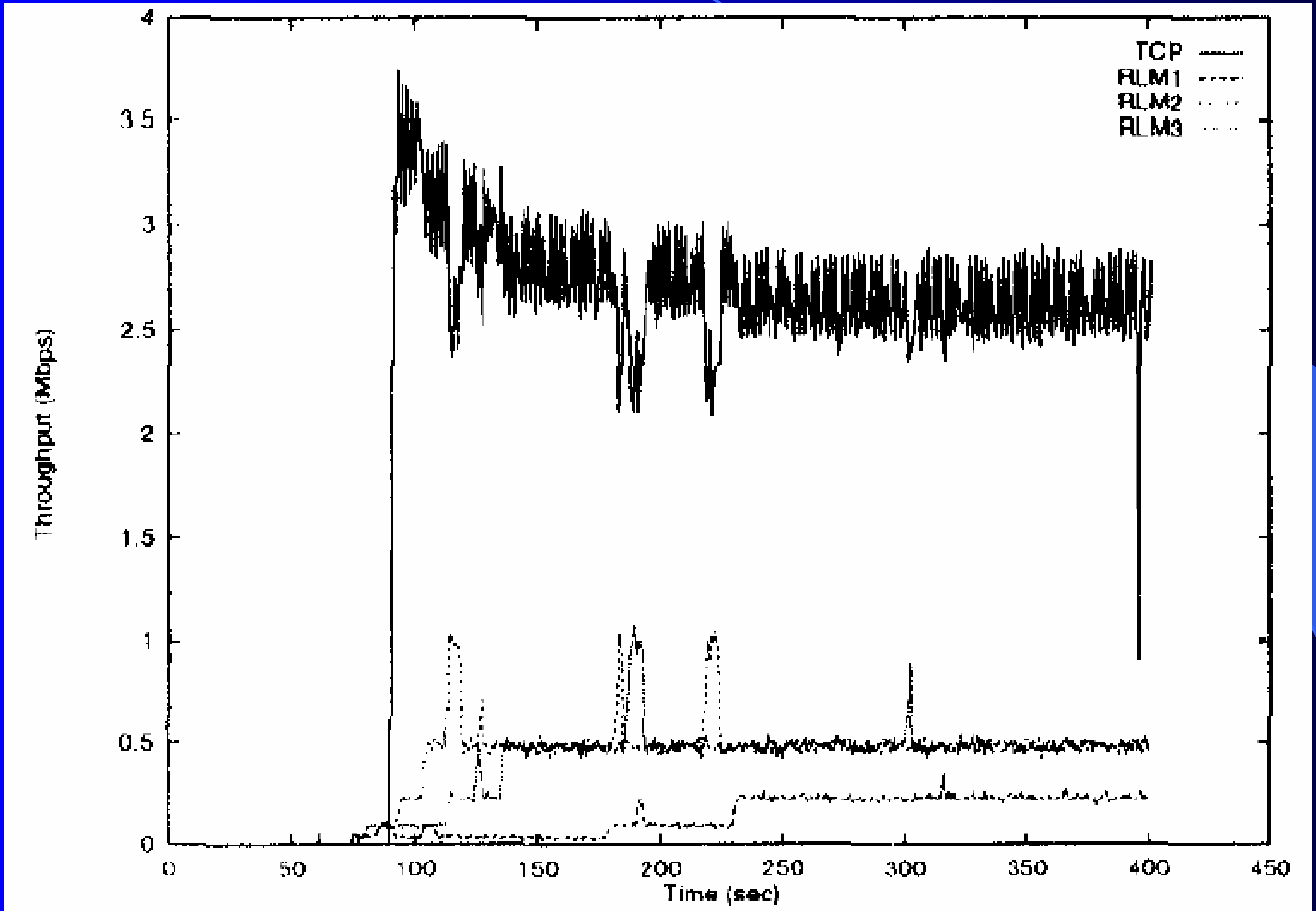


Figure 9f : FIFO

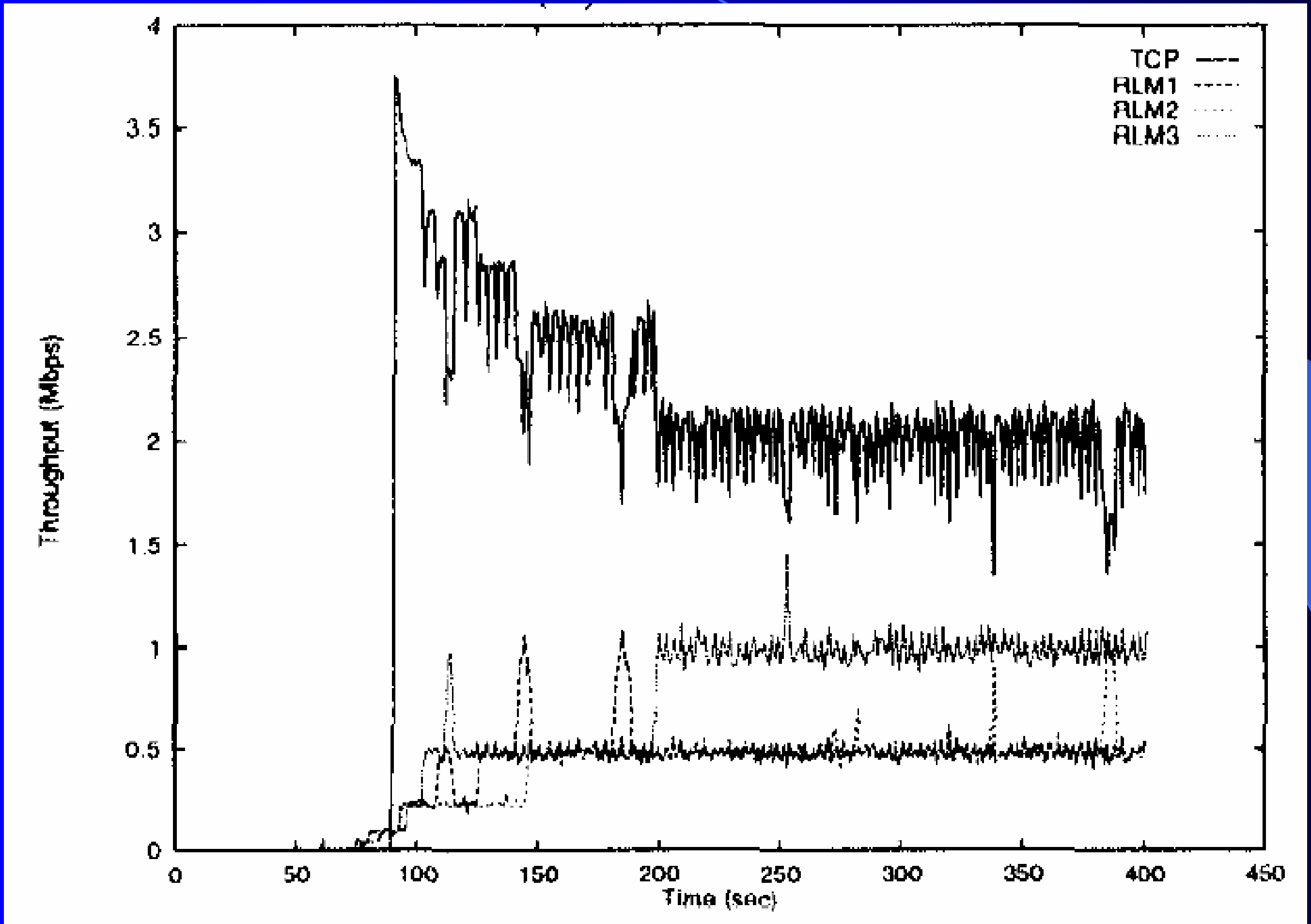
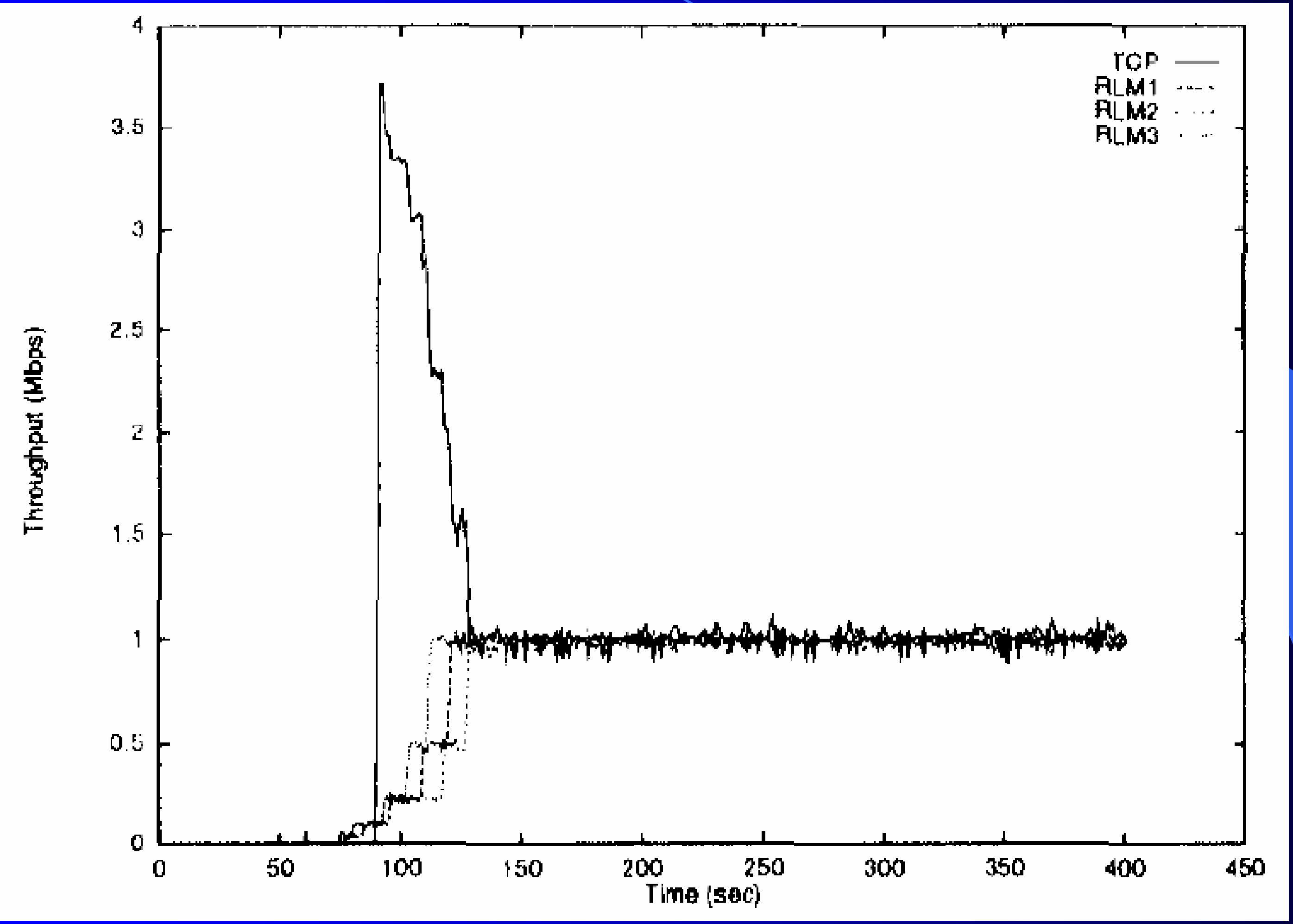


Figure 9a : DRR



Conference Figure : CSFQ

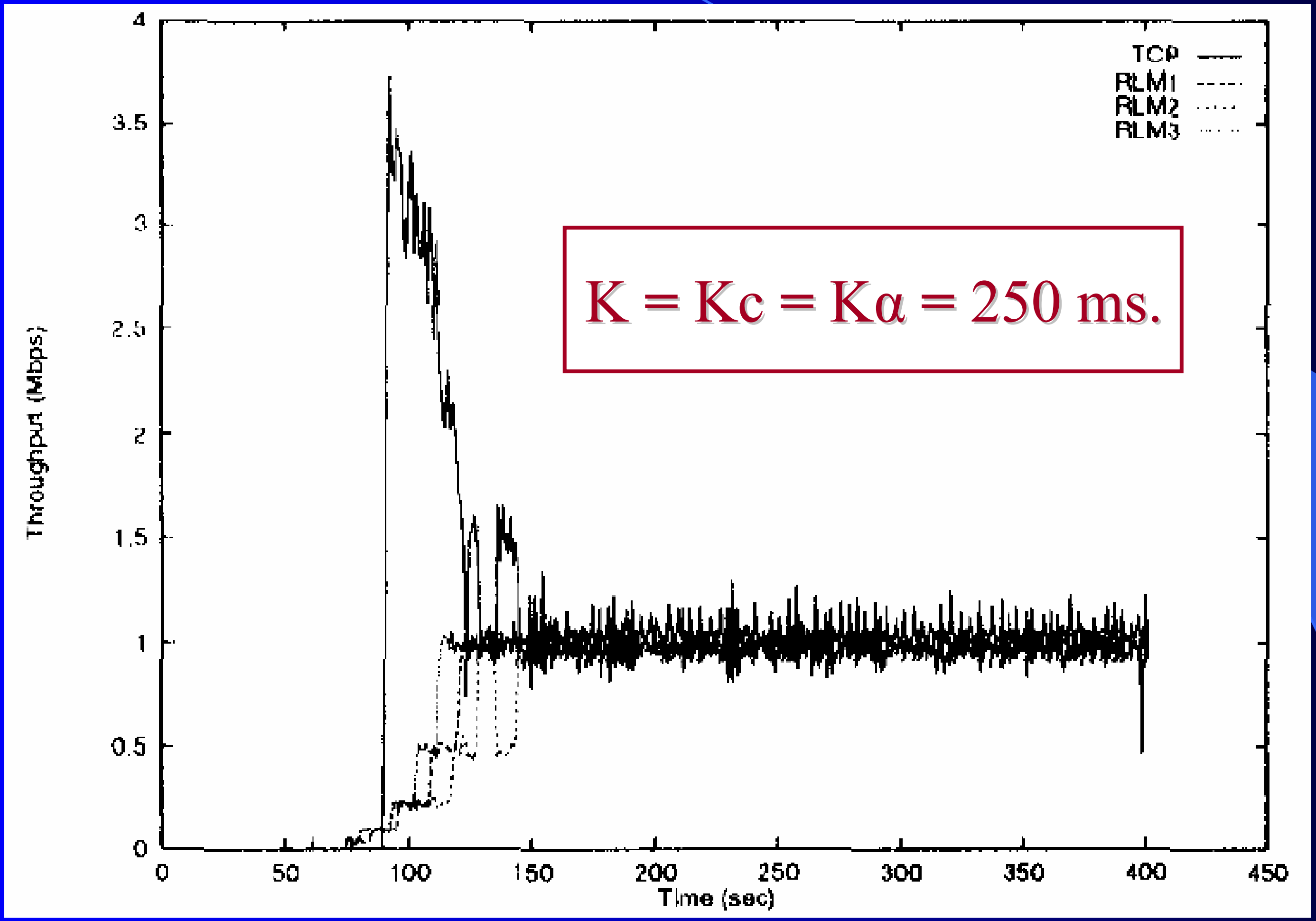
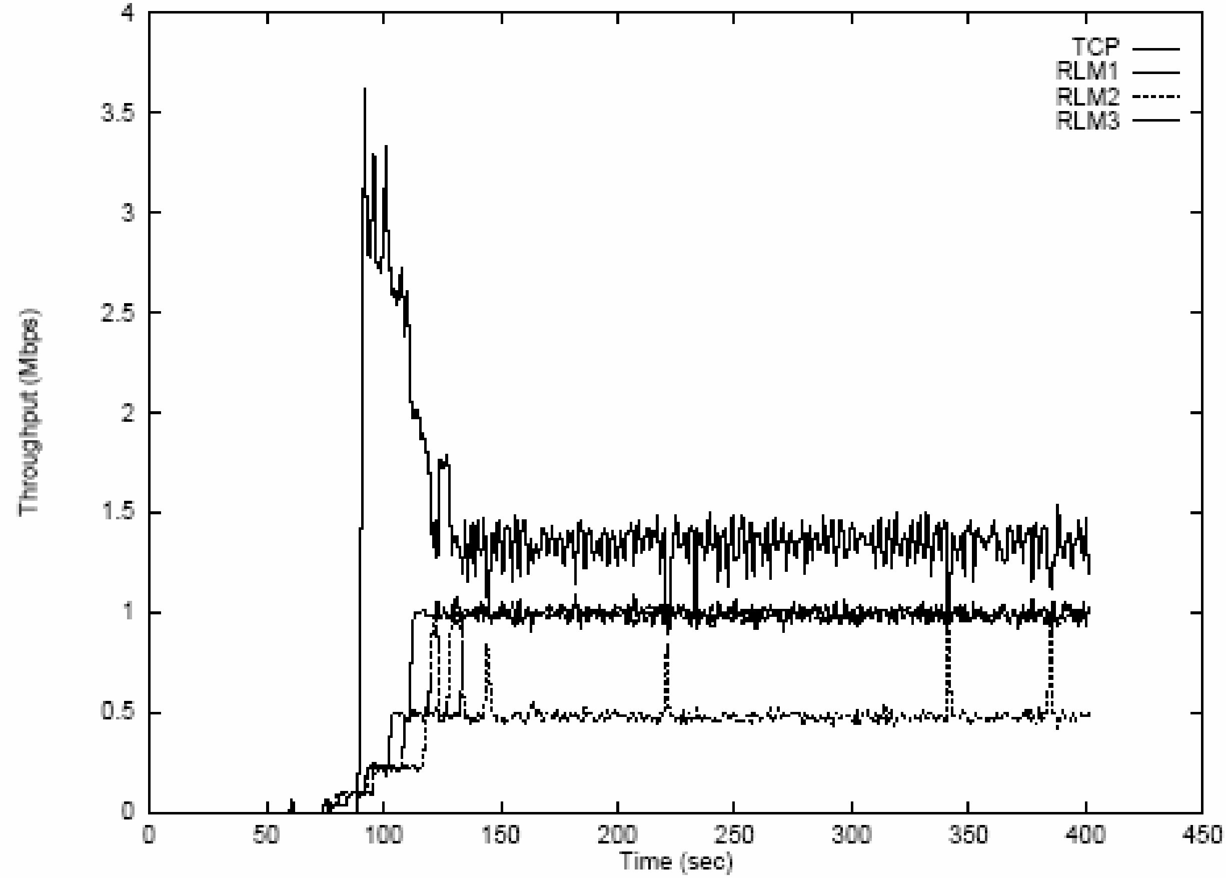
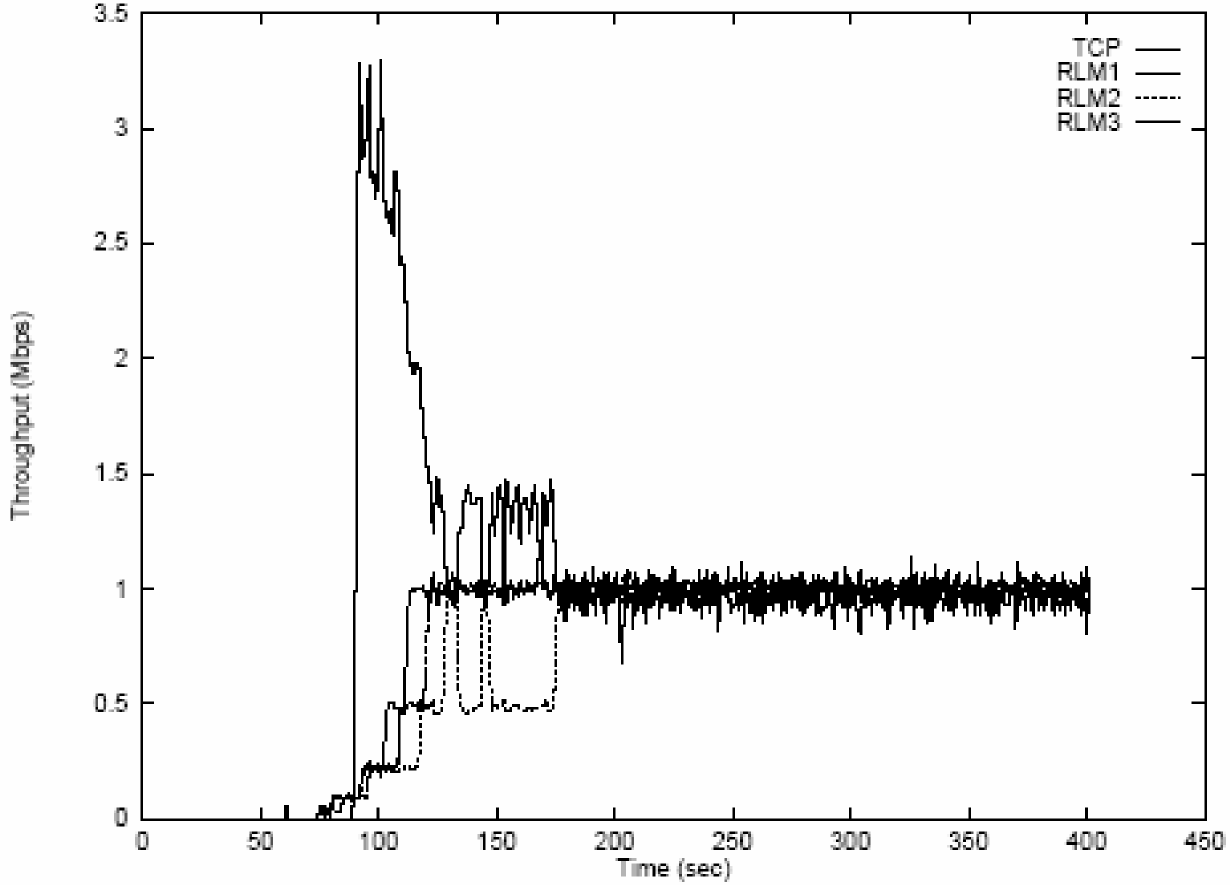


Figure 9c: CSFQ



(c) CSFQ ($K = 100$ ms, $K_\alpha = 200$ ms.)

Figure 9d: CSFQ



(d) CSFQ ($K = 20$ ms, $K_{\alpha} = 40$ ms.)

On-Off Flow Model

- One approach to modeling interactive, Web traffic :: OFF represents "think time"
- ON and OFF drawn from exponential distribution with means of 100 ms and 1900 ms respectively.
- During ON period source sends at 10 Mbps.

Table 1 : One On-Off Flow, 19 TCP Flows

Algorithm	Delivered	Dropped
DRR	1080	3819
CSFQ	1000	3889
FRED	1064	3825
RED	2819	2080
FIFO	3771	1128

4899 packets sent!

Web Traffic

- A second approach to modeling Web traffic that uses Pareto Distribution to model the length of a TCP connection.
- In this simulation 60 TCP flows whose interarrivals are exponentially distributed with mean 0.05 ms and Pareto distribution that yields a mean connection length of 20,1 KB packets.

Table 2 : 60 Short TCP Flows, One UDP Flow

Algorithm	Mean Transfer Time for TCP	Standard Deviation
DRR	25	99
CSFQ	62	142
FRED	40	174
RED	592	1274
FIFO	840	1695

Table 3 : 19 TCP Flows, One UDP Flow with propagation delay of 100 ms.

Algorithm	Mean Throughput	Standard Deviation
DRR	6080	64
CSFQ	5761	220
FRED	4974	190
RED	628	80
FIFO	378	69

Packet Relabeling

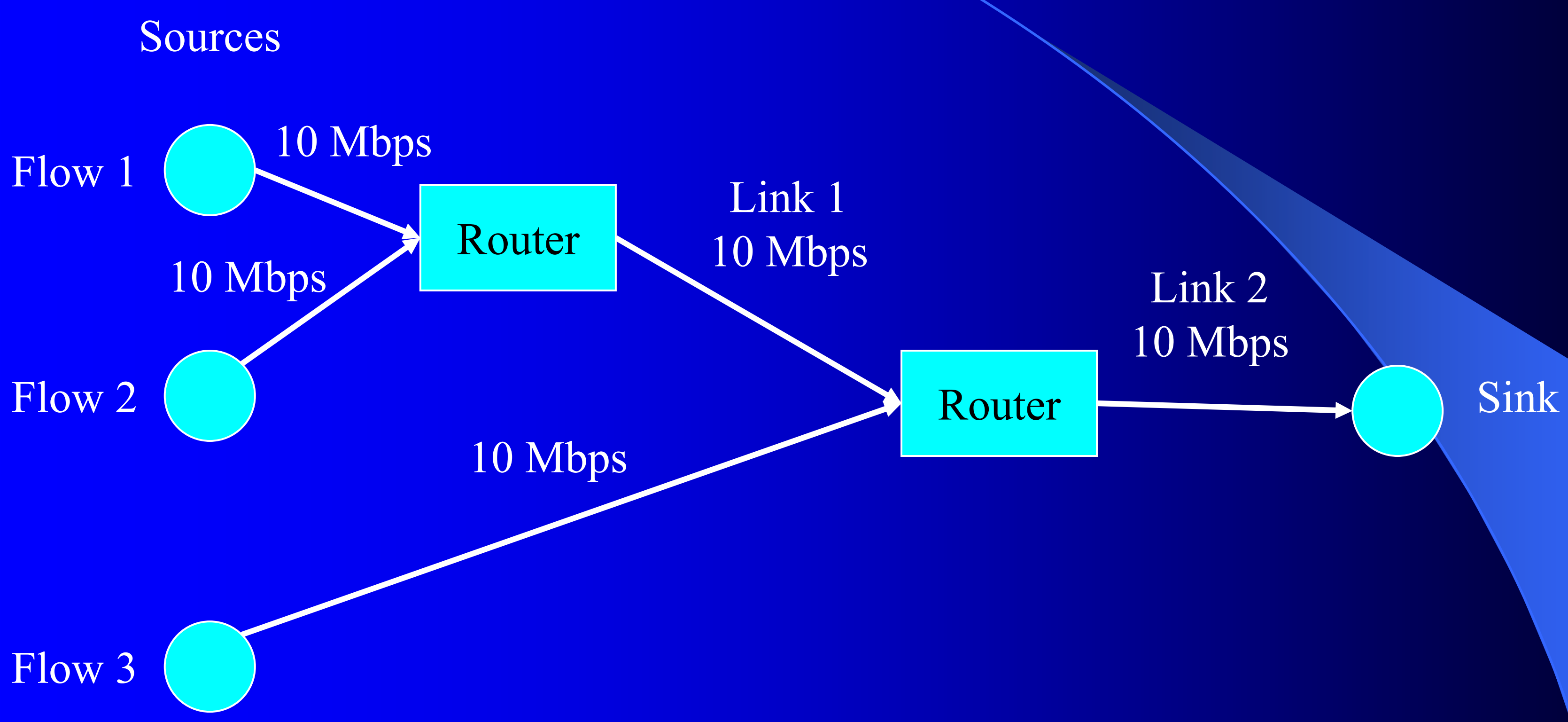


Table 4 : UDP and TCP with Packet Relabeling

Traffic	Flow 1	Flow 2	Flow 3
UDP	3.267	3.262	3.458
TCP	3.232	3.336	3.358

Unfriendly Flows

- Using TCP congestion control requires cooperation from other flows.
- Three types cooperation violators:
  - Unresponsive flows (e.g., Real Audio)
  - Not TCP-friendly flows (e.g., RLM)
  - Flows that lie to cheat.
- ***This paper deals with unfriendly flows!!***

Outline

- Introduction
- Core-Stateless Fair Queueing (CSFQ)
  - Fluid Model Algorithm
  - Packet Algorithm
  - Flow Arrival Rate
  - Link Fair Share Rate Estimation
- NS Simulations
- **Conclusions**

Conclusions

- This paper presents Core Stateless Fair Queueing and offers many simulations to show how CSFQ provides better fairness than RED or FIFO.
- They mention issue of “large latencies”. This is the robust versus fragile flow issue from FRED paper.
- **CSFQ ‘clobbers’ UDP flows!**

Significance

- First paper to use hints from the edge of the subnet.
- Deals with UDP. Many AQM algorithms ignore UDP.
- Makes a reasonable attempt to look at a variety of traffic types.

Problems/ Weaknesses

- “Epoch” is related to three constants in a way that can produce different results.
- How does one set K constants for a variety of situations.
- No discussion of algorithm “stability”

Acknowledgments

- Figures extracted from presentation by Nagaraj Shirali and Choong-Soo Lee in Spring 2002 and modified for annotations.