CEG 7450: Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocation in High Speed Networks

- Reading
- **[SSZ98] Ion Stoica, Scott Shenker, Hui Zhang, "Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High Speed Networks", ACM *SIGCOMM'98.***

Outline

- Introduction
- CSFQ Architecture
  - Flow Arrival Rate
  - Link Fair Share Rate Estimation
- Simulations
- Conclusions

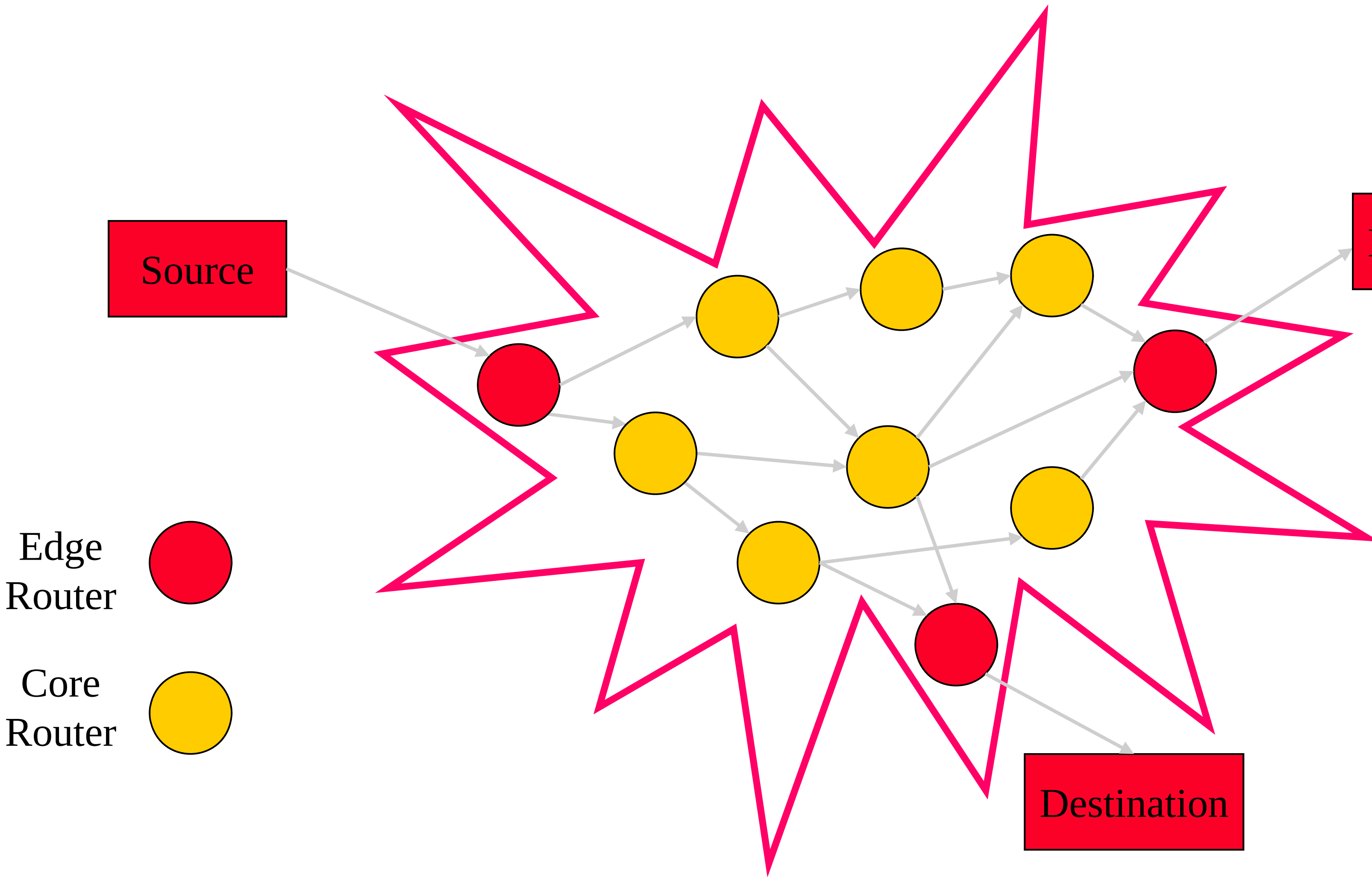
Introduction

- Addresses fair bandwidth allocation in a network
- Fair allocation inherently requires routers to maintain state and perform operations on a per flow basis
- Presents an architecture and a set of algorithms that is "approximately fair" while using FIFO queuing at internal (core) routers

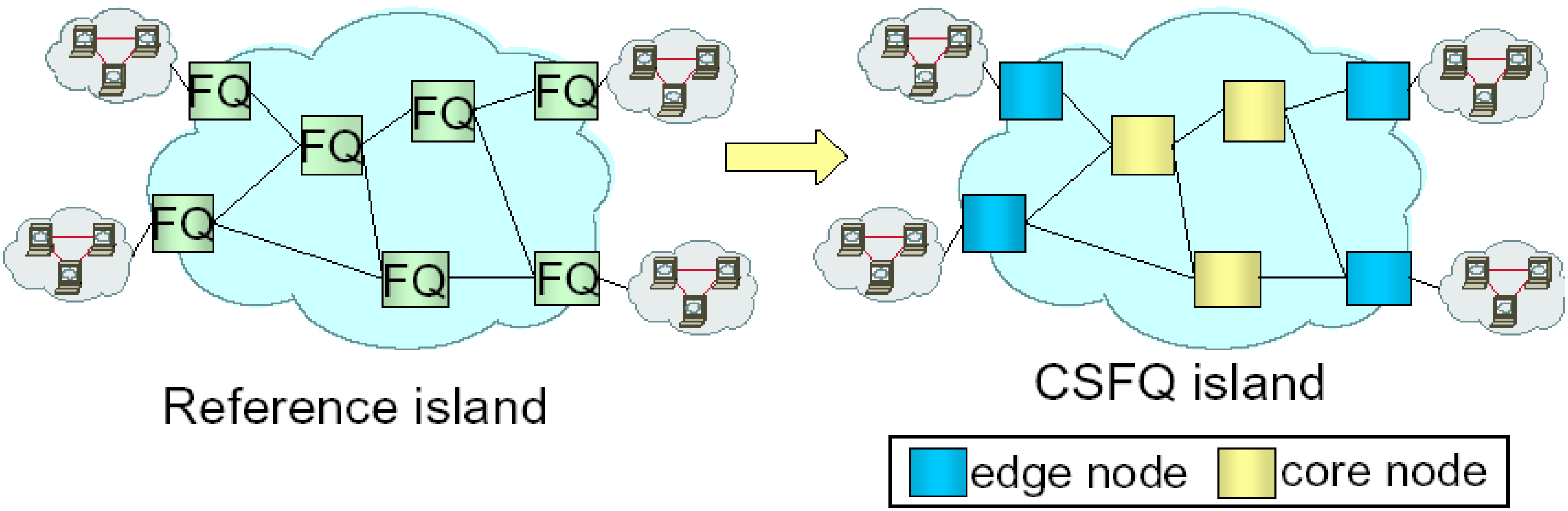
Fair Queueing

- Rigorous fair queueing requires **per flow state**: too costly in high speed core routers
- Yet, some form of **FQ is essential** for efficient, fair congestion control in the backbone network

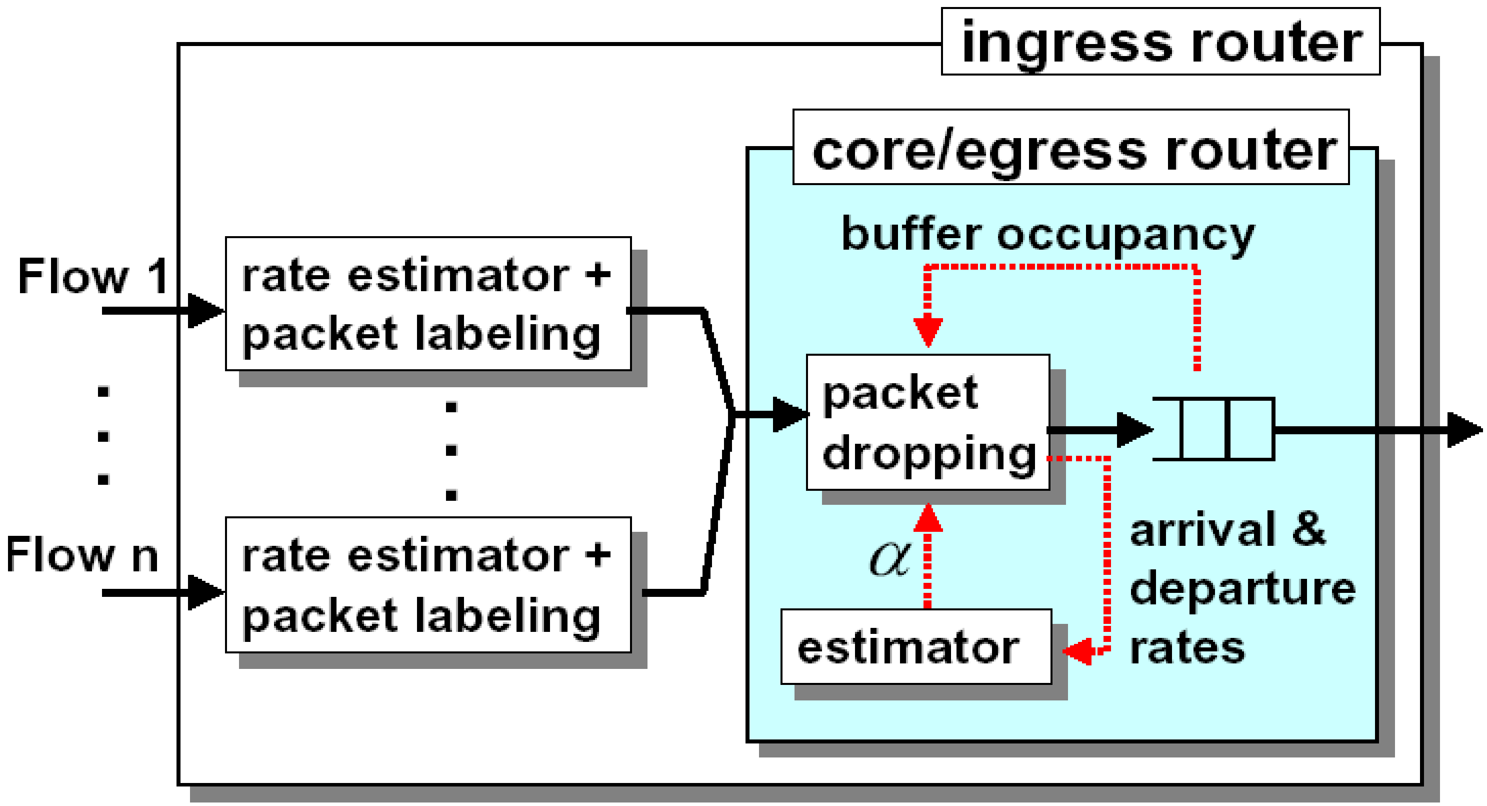
An “Island” of Routers



CSFQ


FQ
FQ
FQ
FQ
FQ
FQ
FQ
Reference island
CSFQ island
edge node
core node

Edge – Core Router Architecture



Core Stateless Fair Queueing

- An architecture and a set of algorithms that is "approximately fair" while using FIFO queueing at internal routers
- **Proposed solution**:

(a) **per flow** accounting and **rate labeling** at **edge routers**

(b) **packet state**: packets carry **rate labels** (e.g., in TOS field)

(c) **stateless FQ** at core routers:

- no per flow state kept
- packet drop probability computed directly from packet label and
- fair rate calculated based on aggregated information

Key Elements of CSFQ

- Edge router estimates **current rate *r(i)*** of each flow and stamps it in IP header (e.g., TOS field)
- Flow **rate value adjusted** as packet travels through various bottlenecks in the backbone
- Core router **estimates max/min fair share** on its links based on aggregate traffic measurements
- Core router **probabilistically drops** packets in a flow which exceeds fair share

Edge router

- Ingress edge routers keep per flow state
- Ingress edge routers compute per-flow rate estimates and insert these estimates as ***labels*** into each packet header (e.g., TOS field)

Flow Arrival Rate Estimation

- 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

- The estimated rate of flow *i*, r_i is updated every time a new packet is received:

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

Where

  - $T = T_i^k = t_i^k - t_i^{k-1}$
  - $L = l_i^k$
  - K is a constant

Core router

- No per flow state
- FIFO queueing with probabilistic dropping of packets on input is employed at core routers
- Labels are updated at each router based only on aggregate information

Dropping Probability in Fluid Flow Model

- Router with output link capacity C
- $r_i(t)$: i^{th} flows arrival rate
- $\alpha(t)$: fair share rate
- In the fluid flow model, incoming bits of flow *i* at a core router are dropped with probability

$$\mathbf{Max}\ (0, 1 - \alpha(t) / r_i(t))$$

Probabilistic Dropping at Router

- FIFO queueing with probabilistic dropping of packets on input is employed at core routers

- If aggregate arrival rate $A < C$, no packet is dropped
- If $A > C$ (i.e., congested link):

  (a) bottlenecked flow (i.e., $r(i,t) > \alpha(t)$)): drop the fraction of "bits" above the fair share, i.e. $(r(i,t) - \alpha(t))/r(i,t)$

  (b) non-bottlenecked flow: no dropping

  Equivalently:

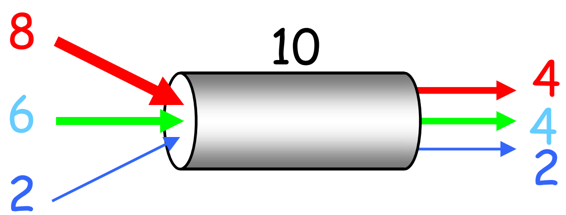
  packet drop probability = $\max(0, 1 - \alpha(t)/r(i,t))$

- Need to estimate A and $\alpha(t)$

Fair Rate Computation

- If link congested, compute f such that

$$\sum_i \min(r_i, f) = C$$

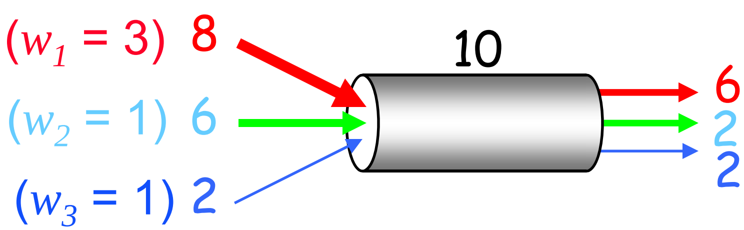


f = 4:
min(8, 4) = 4
min(6, 4) = 4
min(2, 4) = 2

Weighted Fair Rate Computation

- Associate a weight w_i with each flow i
- If link congested, compute f such that

$$\sum_i \min(r_i, f \times w_i) = C$$



f = 2:
min(8, 2*3) = 6
min(6, 2*1) = 2
min(2, 2*1) = 2

Fair share computation with per flow state

- Assume N flows arrive at core router
- Each flow rate ***r(i)*** is stamped in header
- Max-Min fair operation:

  (a) all **bottlenecked** flows get “**fair share**” rate “*α(t)***”** (the excess rate packets are dropped)

  (b) **non-bottlenecked** flows are granted their **full rate**

Thus, at full link utilization:

Sum (over i = 1..N) of min{ r(i,t), *α(t)* } = **C**

where **C** = link capacity

Fair Share Computation with per flow state (cont)

If all r(i) are known at the router, fair share **a** can be easily computed:

(a) try an arbitrary fair share threshold $\alpha(0)$

(b) from "fair share" formula compute the resulting link throughput R

(c) compute new value $\alpha(1) = \alpha(0)$C/R

(d) go back to (b) and iterate until $\alpha(n)$ converges to fixed point

Fair Rate Estimation without Per Flow State

Heuristic algorithm with aggregate state variables:

- α': fair share estimate

- F': aggregate acceptance rate estimate

Fair Rate Estimation

- If the packet is dropped, F' remains the same
- If the packet is not dropped, F' is updated using exponential averaging
- $F'(\alpha) = C$
- When the link state is congested (A'>C) during K_c, update α':

$$\alpha'_{new} = \alpha'_{old} * C / F'$$

- When the link state is not congested during K_c, update α':

α'_{new} = the largest rate of any active flow during K_c

Dropping probability calculation

- α' now feeds into next calculation of drop probability (p) as α:

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

Implementation details (cont)

(a) **flow arrival rate** at edge router computed with exponential averaging

(b) **fair share computation at core router**:

measure aggregate arrival rate A(t) using exponential averaging

If router is congested (i.e., A(t) >C), then:

measure (exp avg) the fraction F of bits currently accepted

i.e., F(t) = current acceptance rate

Assume F is a linear function of **a** (in reality concave function). Then:

New fair share value: α**(new) =** α**(old) C/F(t)**

Label Rewriting

- Labels are updated at each router based on aggregate information
- 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

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

More details…

- Occasionally, router buffer overflows:
  - then, decrease $\alpha(t)$ by 1%
  - Never increase $\alpha(t)$ by more than 25%

- Weighted CSFQ option:

  if $w(i)$ is the weight of flow i, then:

  label: $r(i)/w(i)$

CSFQ Pseudo -Code

on receiving packet p
  if (edge router)
    $i =$ **classify**(p);
    $p.label =$ **estimate_rate**(r_i, p); *// use Eq. (3)*
  $prob = \max(0, 1 - \alpha/p.label)$;
  if $(prob >$ **unif_rand**$(0,1))$
    $\alpha =$ **estimate_**$\alpha(p, 1)$;
    drop(p);
  else
    $\alpha =$ **estimate_**$\alpha(p, 0)$;
    enqueue(p);
    if $(prob > 0)$
      $p.label = \alpha$; *// relabel p*

CSFQ Pseudo -Code

estimate_α $(p, dropped)$

// $\widehat{\alpha}$ 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

$\widehat{A} =$ **estimate_rate**$(\widehat{A}, p)$; // *est. arrival rate (use Eq. (5))*

if $(dropped == FALSE)$

$\widehat{F} =$ **estimate_rate**$(\widehat{F}, p)$; // *est. accepted traffic rate*

if $(\widehat{A} \geq C)$

if $(congested == FALSE)$

$congested = TRUE$;

$start\_time = crt\_time$;

if $(\widehat{\alpha} == 0)$

// $\widehat{\alpha}$ *can be set to* 0 *if no packet is received*

// *during a widow of size* K_c

$\widehat{\alpha} = \max(p.label, \alpha\_K_c)$;

CSFQ Pseudo -Code

```
    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;
```

Simulation Experiments

- **FIFO**
- **RED** (FIFO + Random Early Detection)
- **FRED** (Flow Random Early Drop, SIGCOMM 97): extension of RED to improve fairness; it keeps state of flows which have one or more packets in queue; it preferentially drops packets from flows with large queues
- **DRR** (Deficit Round Robin): **per flow queueing**; drops packets from **largest** queue

FRED (Flow Random Early Drop)

- Maintains per flow state in router
- FRED preferentially drops a packet 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 amount of buffer

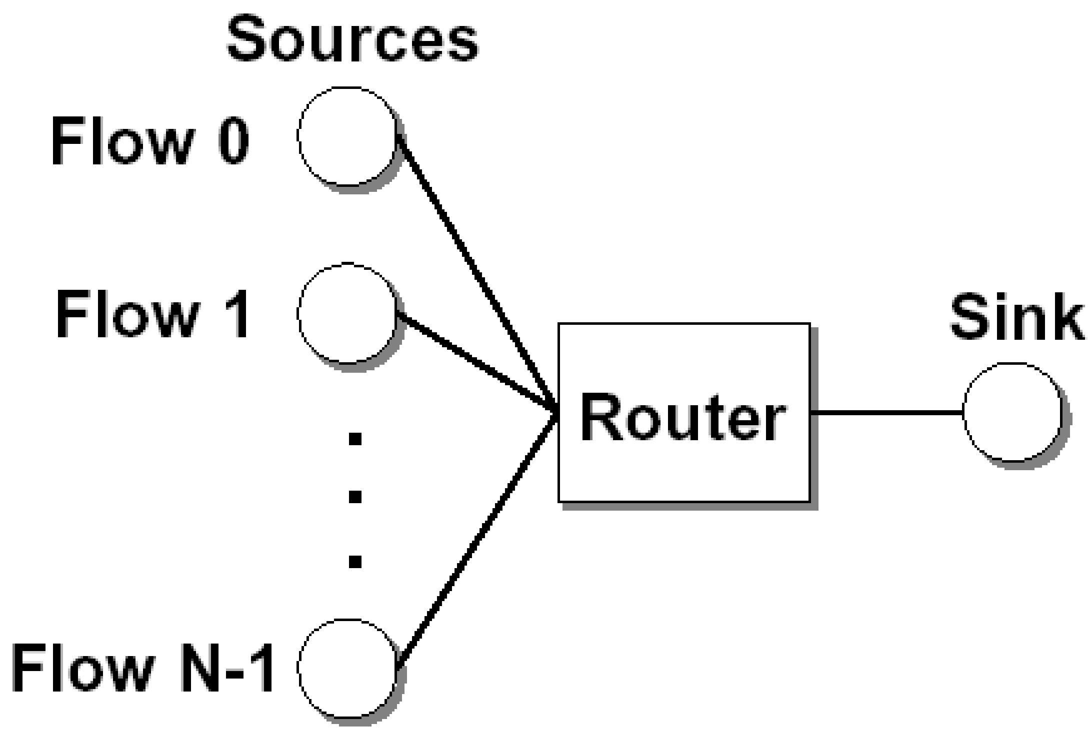
DRR (Deficit Round Robin)

- 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

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
- Constants (K, K_α, K_c) : all 100 ms

A Single Congested Link



A Single Congested Link

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

Single Congested Link Experiment

10 Mbps congested link shared by N flows

(a) 32 UDP flows with linearly increasing rates

(b) single “ill behaved” UDP flow; 31 TCP flows

Figure 3a: 32 UDP Flows

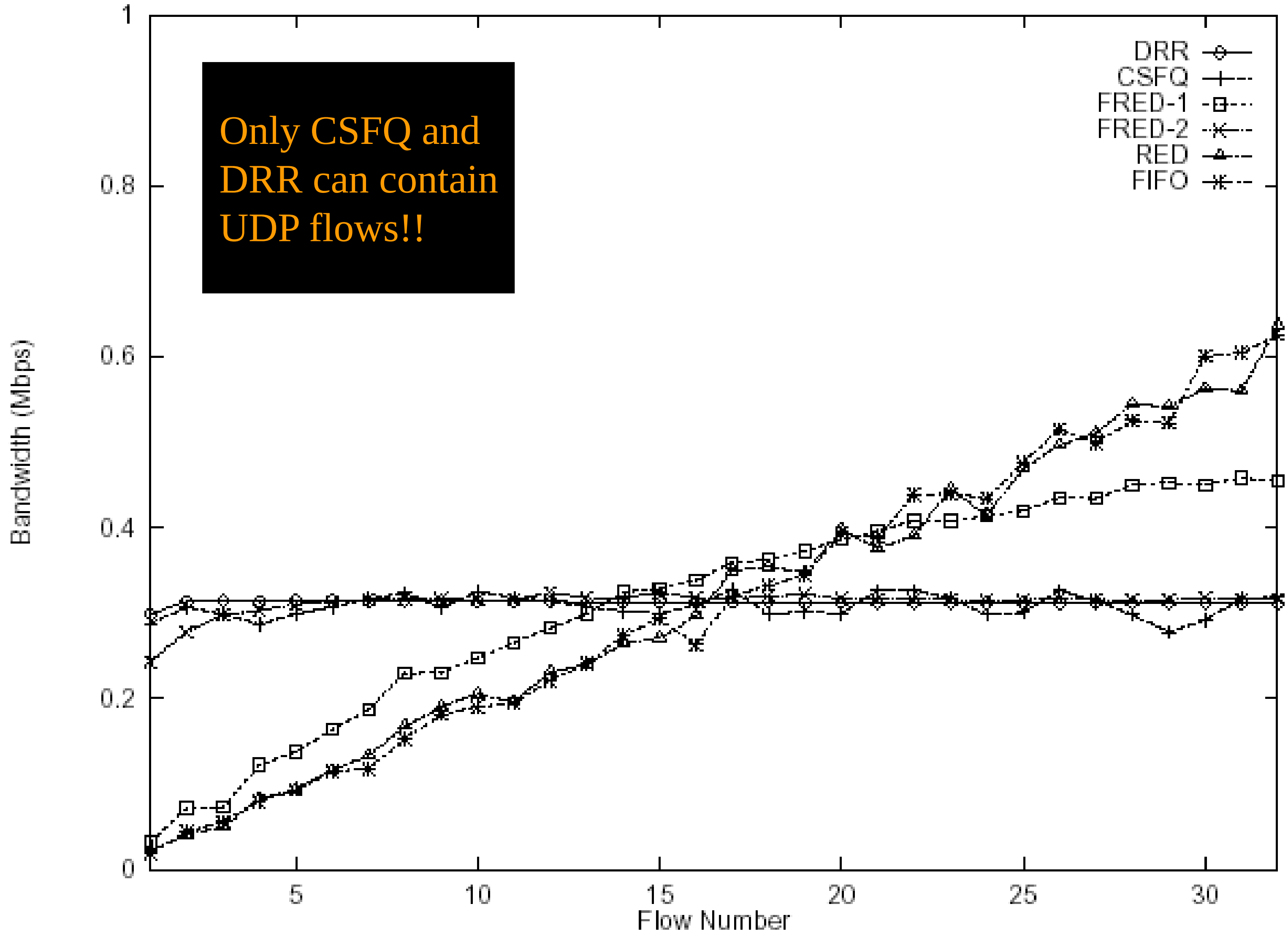
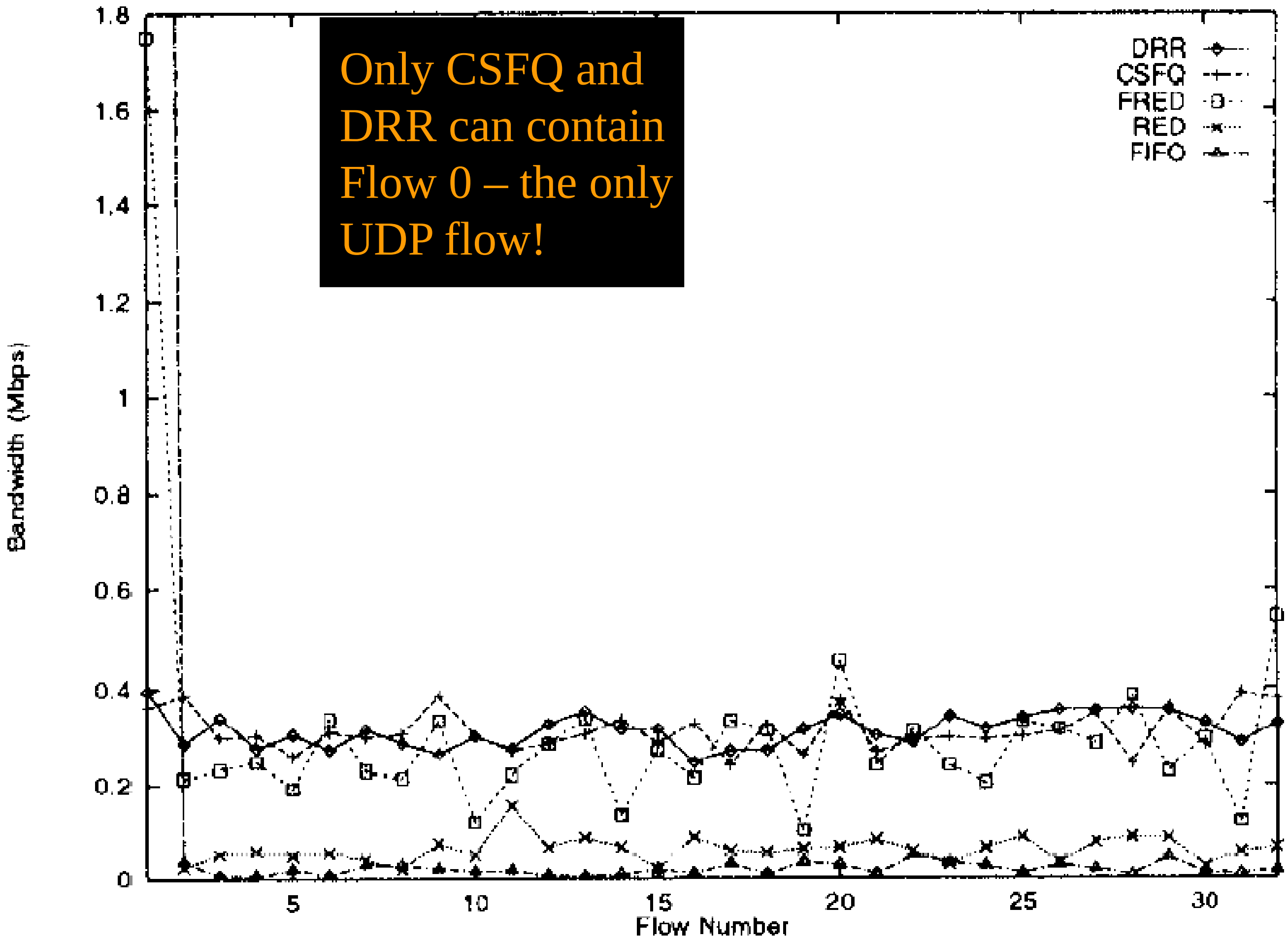
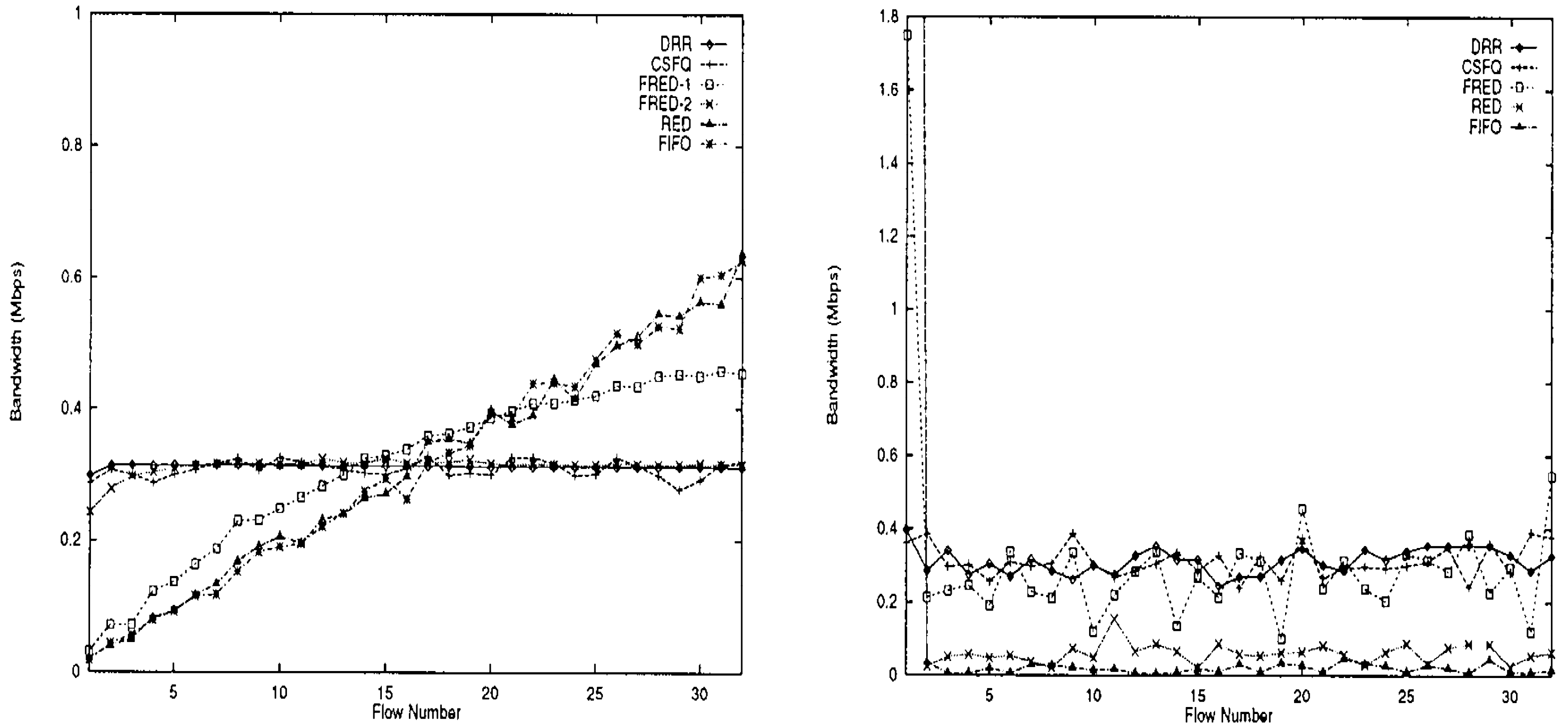


Figure 3b : One UDP Flow, 31 TCP Flows



(a) linear rate UDPs; (b) single UDP + 31 TCPs



(a) (b)

A Single Congested Link

- Third Experiment Set : 31 simulations
  - Each simulation has a different N

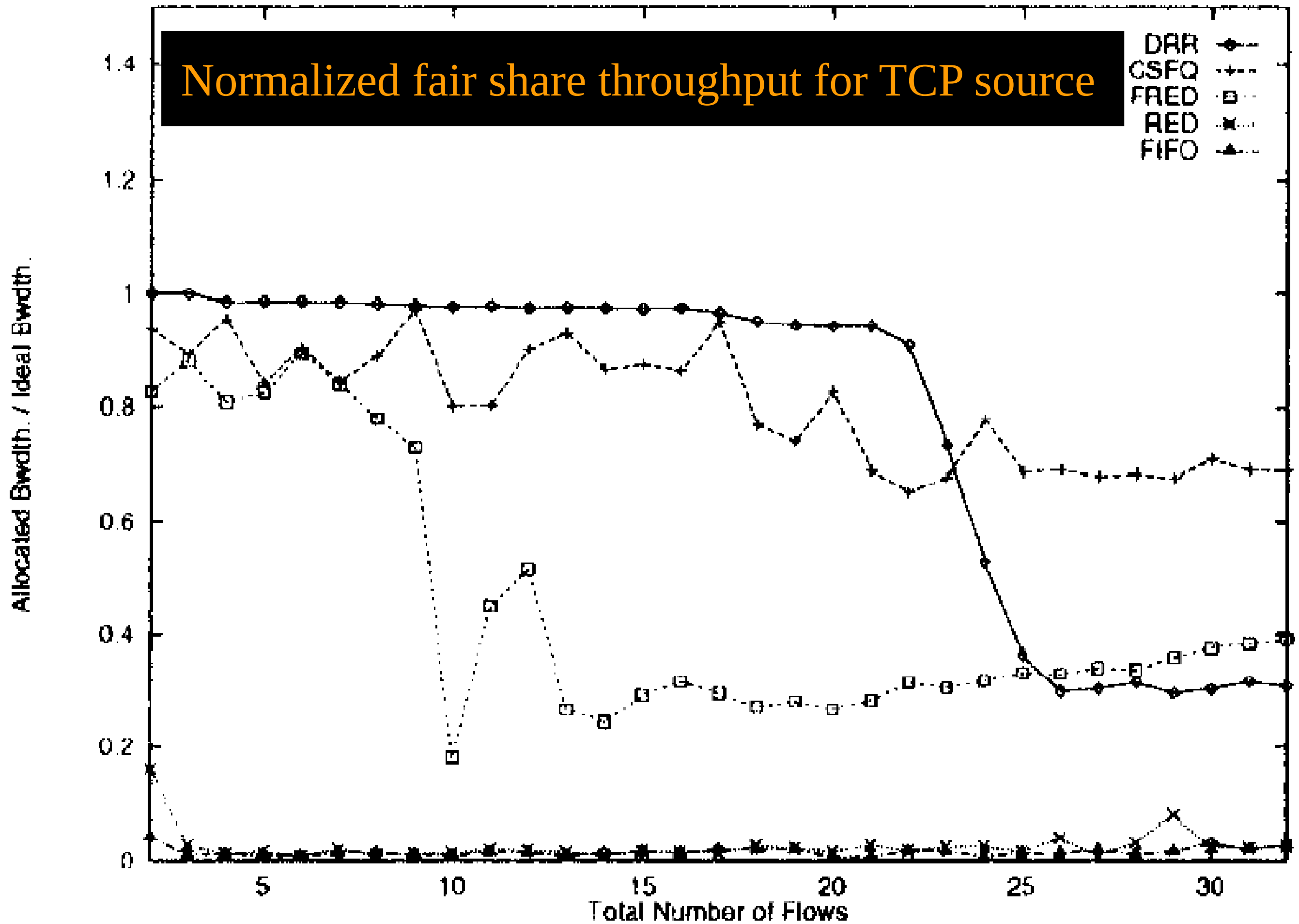
    N = 2 … 32.
  - One TCP and N-1 UDP flows with each UDP flow sending at <u>twice</u> fair share rate of 10/N Mbps

Figure 4 : One TCP Flow, N-1 UDP Flows

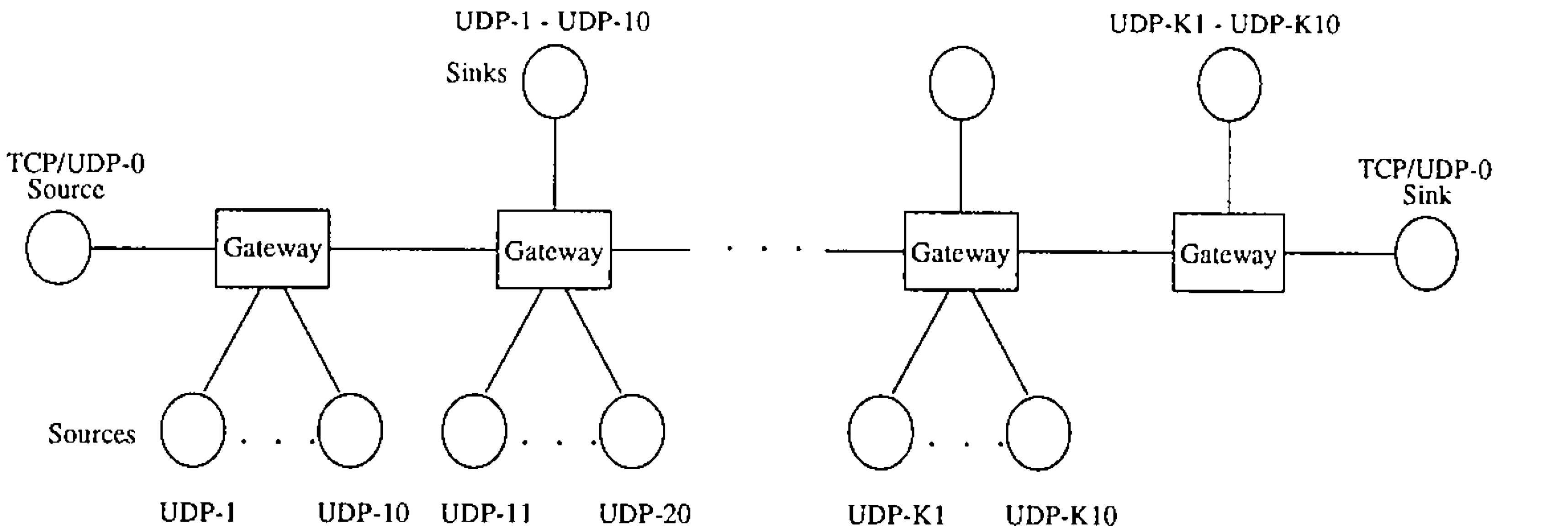
DRR good for less than 22 flows.

CSFQ better than DRR when a large number of flows.

CSFQ beats FRED.



Multiple congested links



Multiple Congested Links

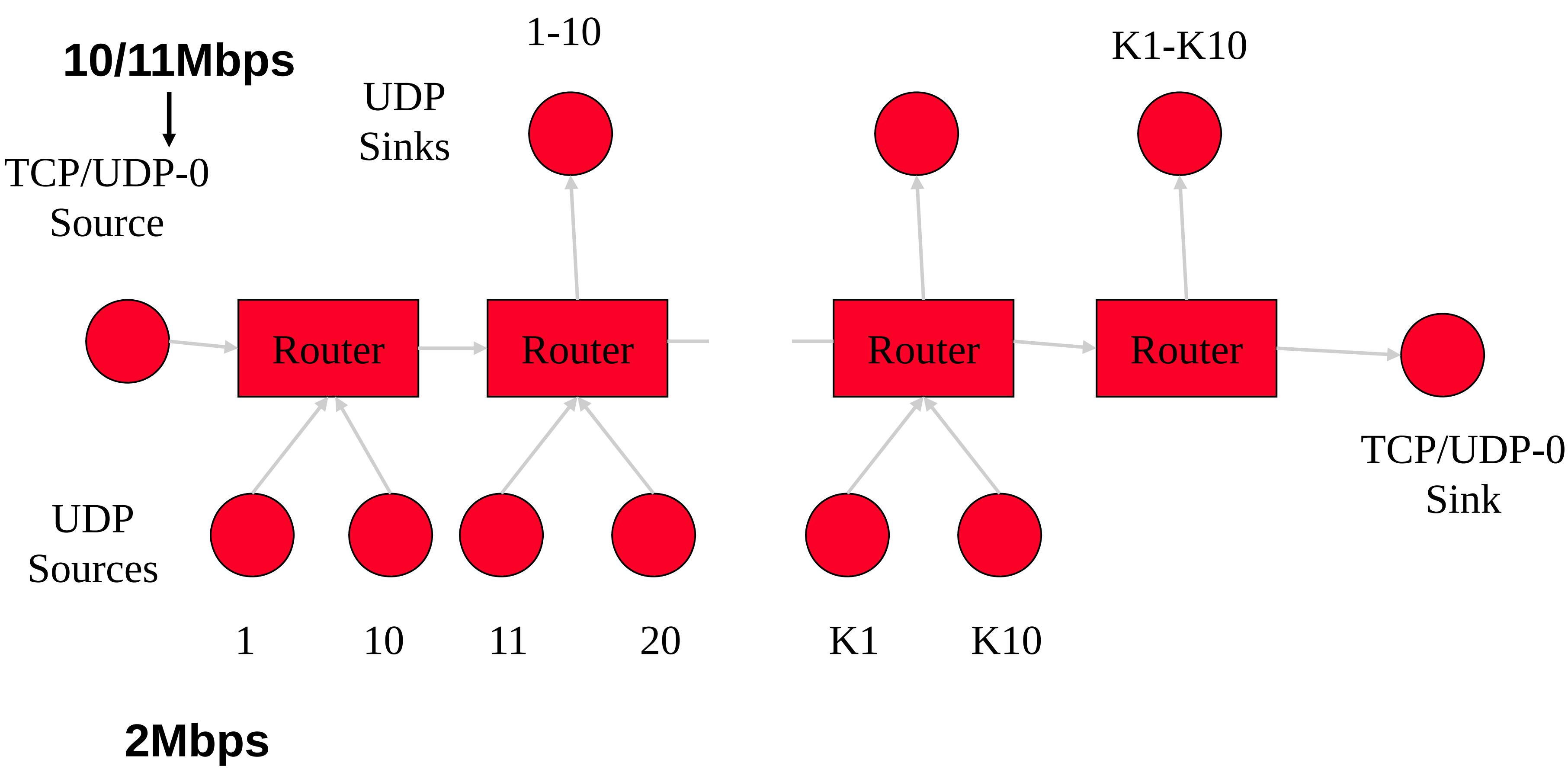


Figure 6a : UDP source

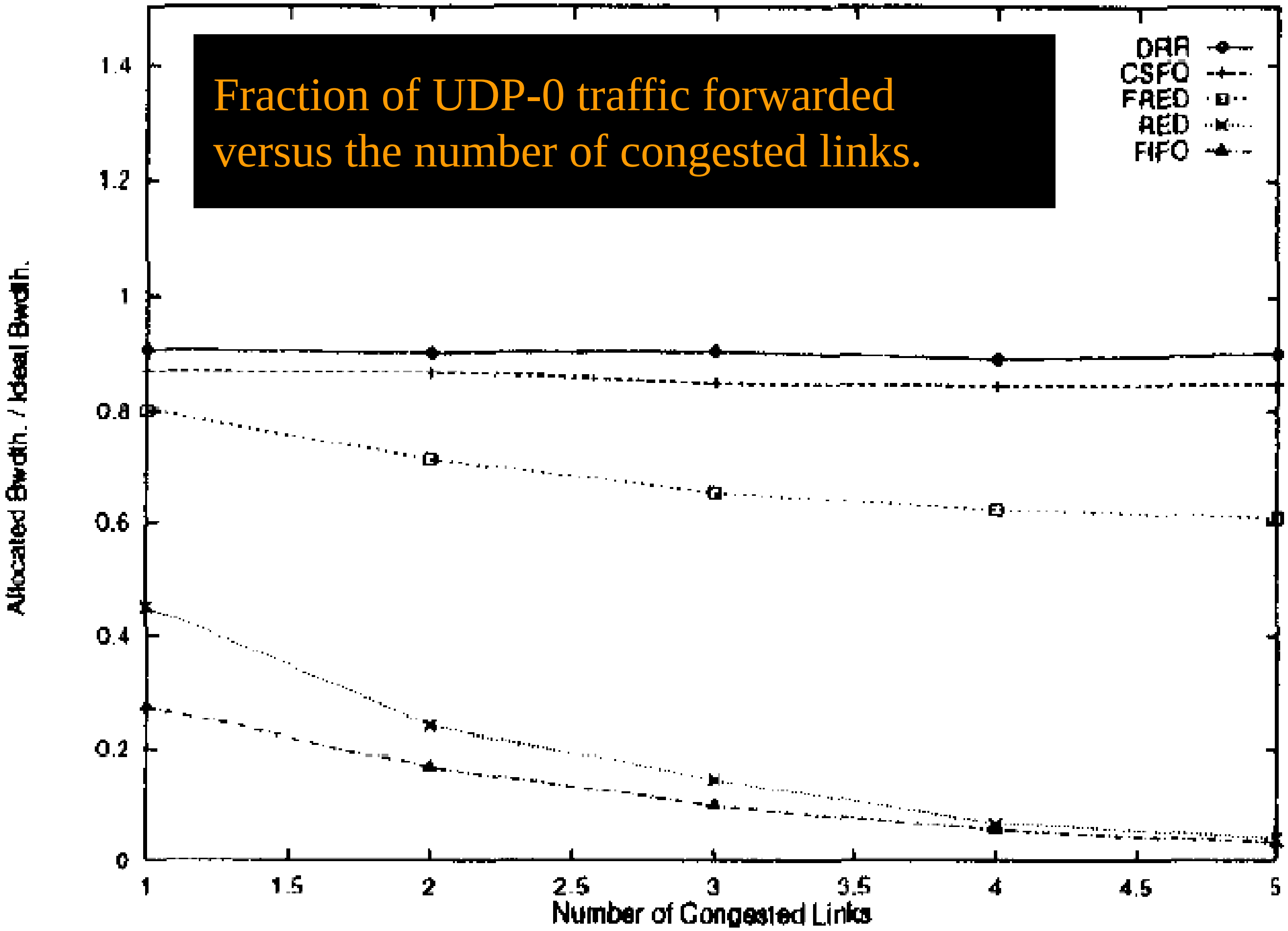
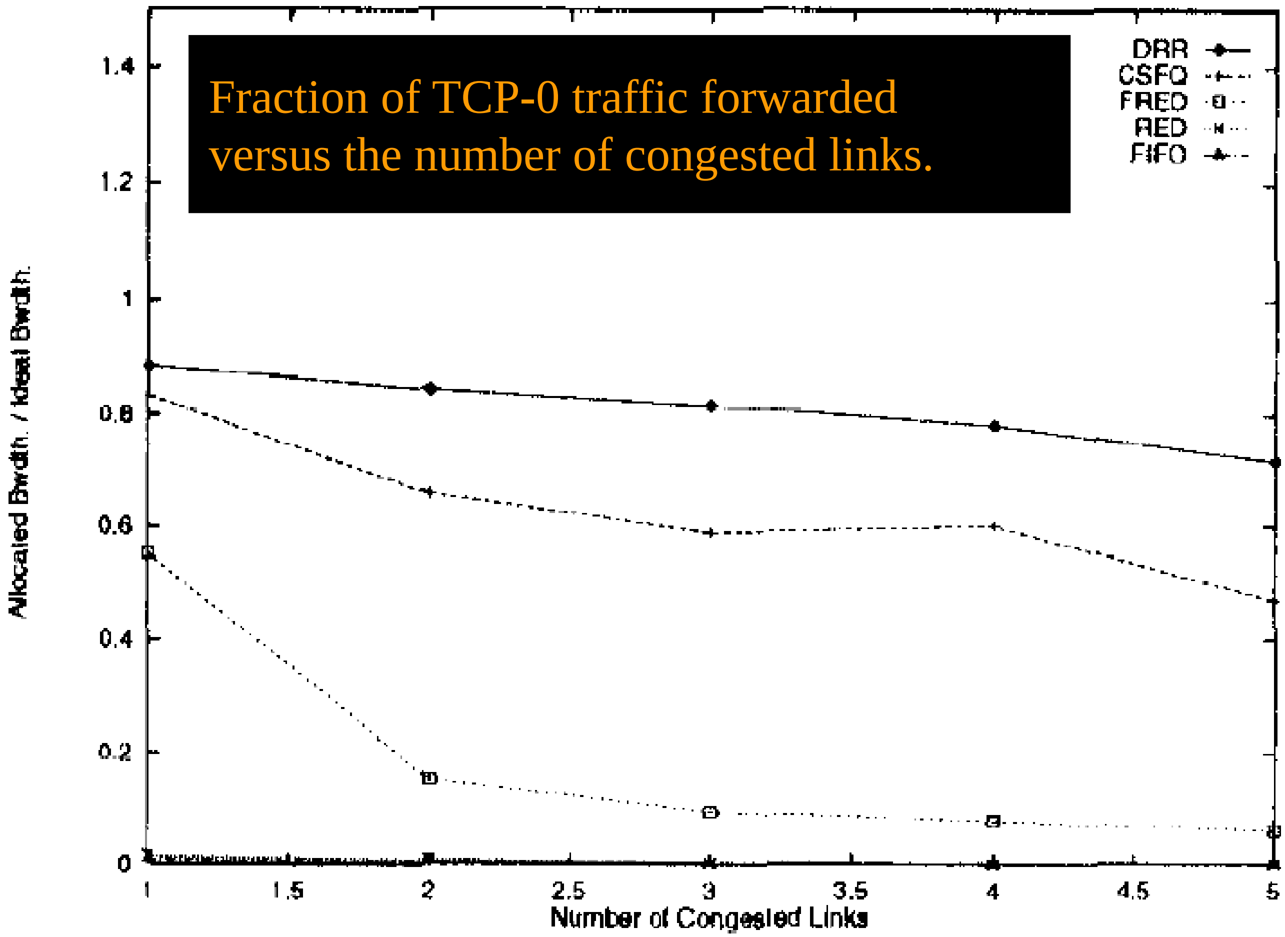


Figure 6b : TCP Source



TCP competes with UDP

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
- Each source uses 7-layer encoding where layer i sends at 2^{i+4} Kbps; 1Mbps = subscribe to first 5 layers

Receiver-driven Layered Multicast

- Simulation of three RLM flows and one TCP flow over 4Mbps link
- Fair share for each is 1 Mbps
- Router buffer set to 64 KB, K, K_c, and K_α are set to 250 ms

Figure 7a : DRR

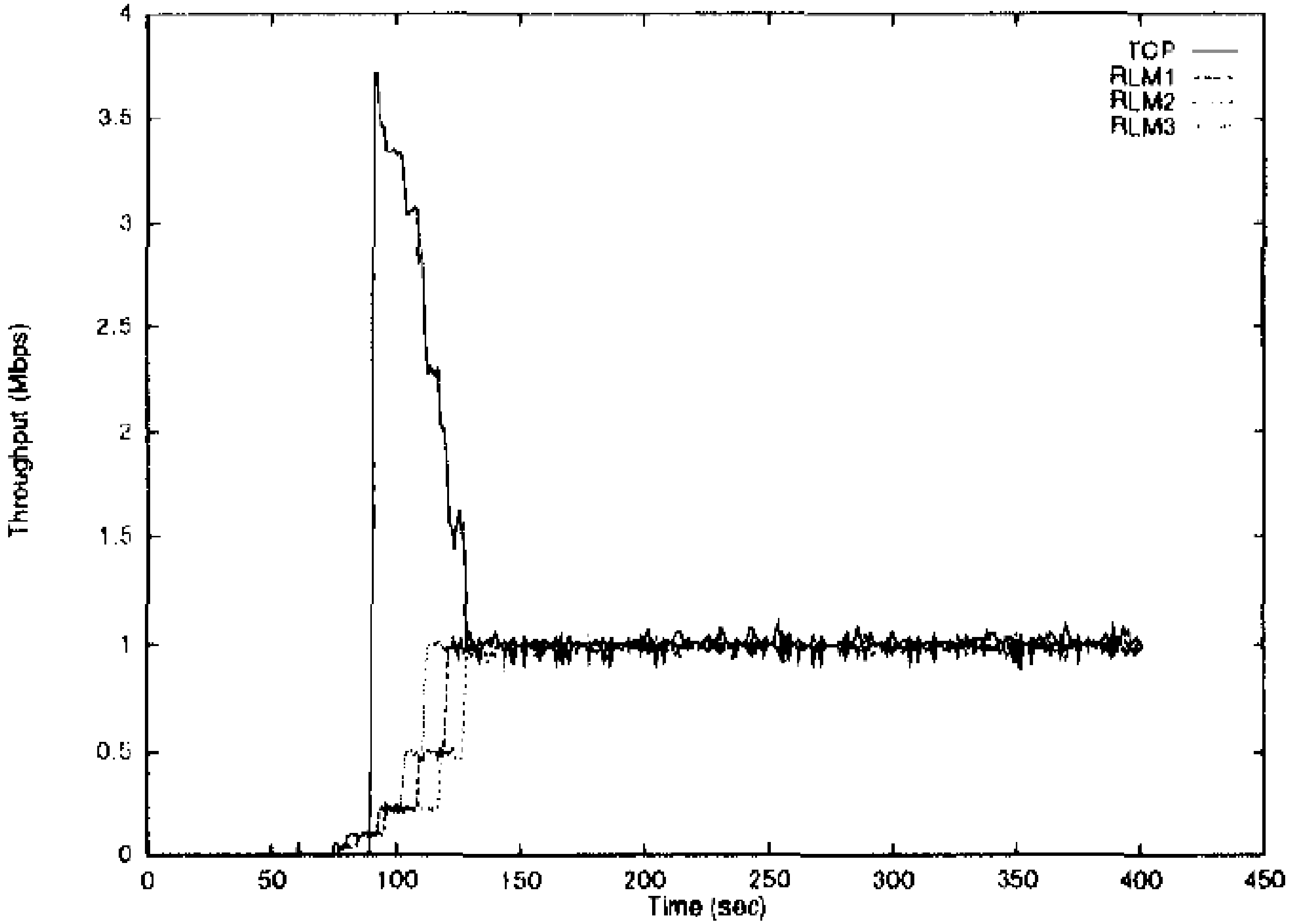


Figure 7b : CSFQ

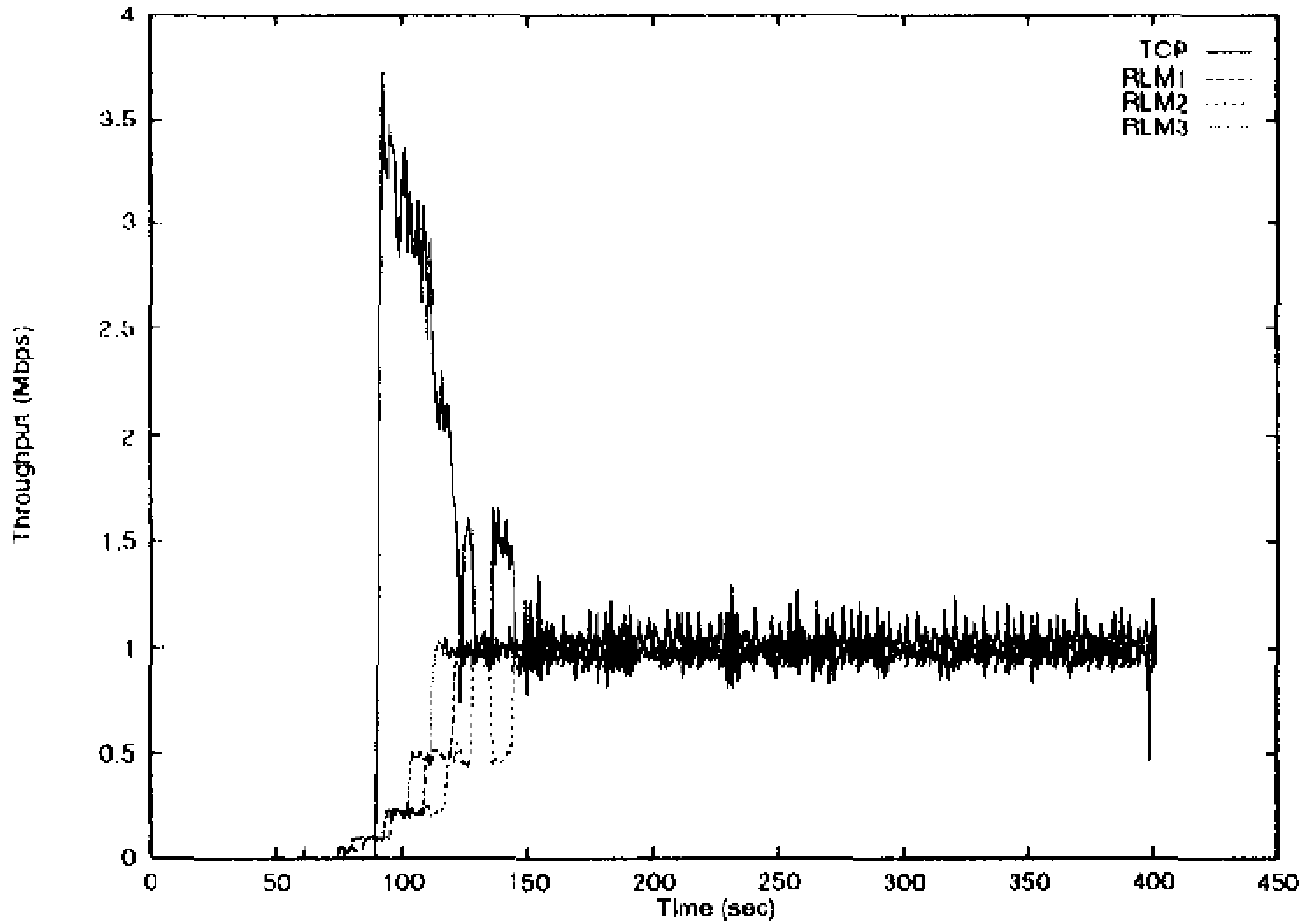


Figure 7c : FRED

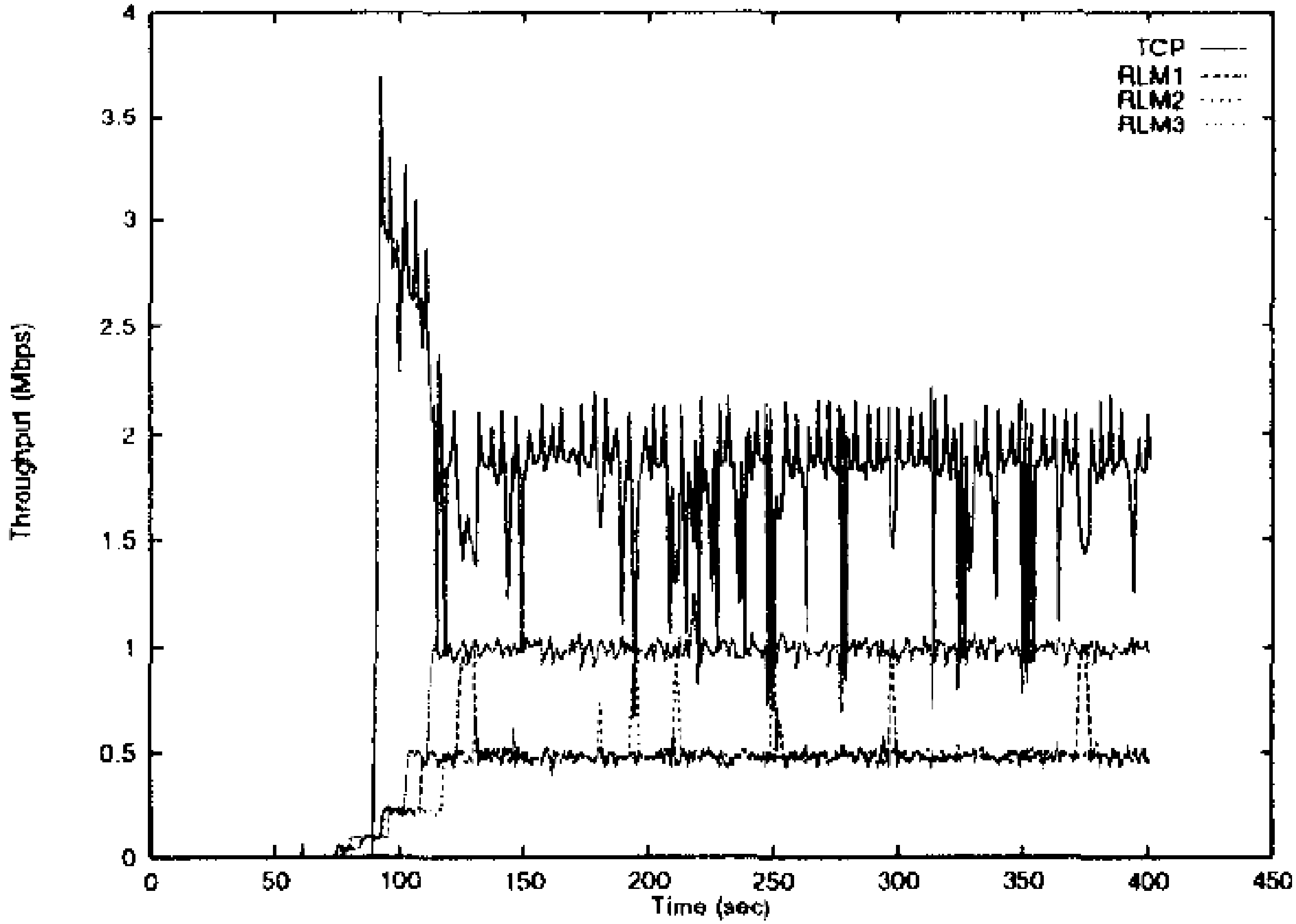


Figure 7d : RED

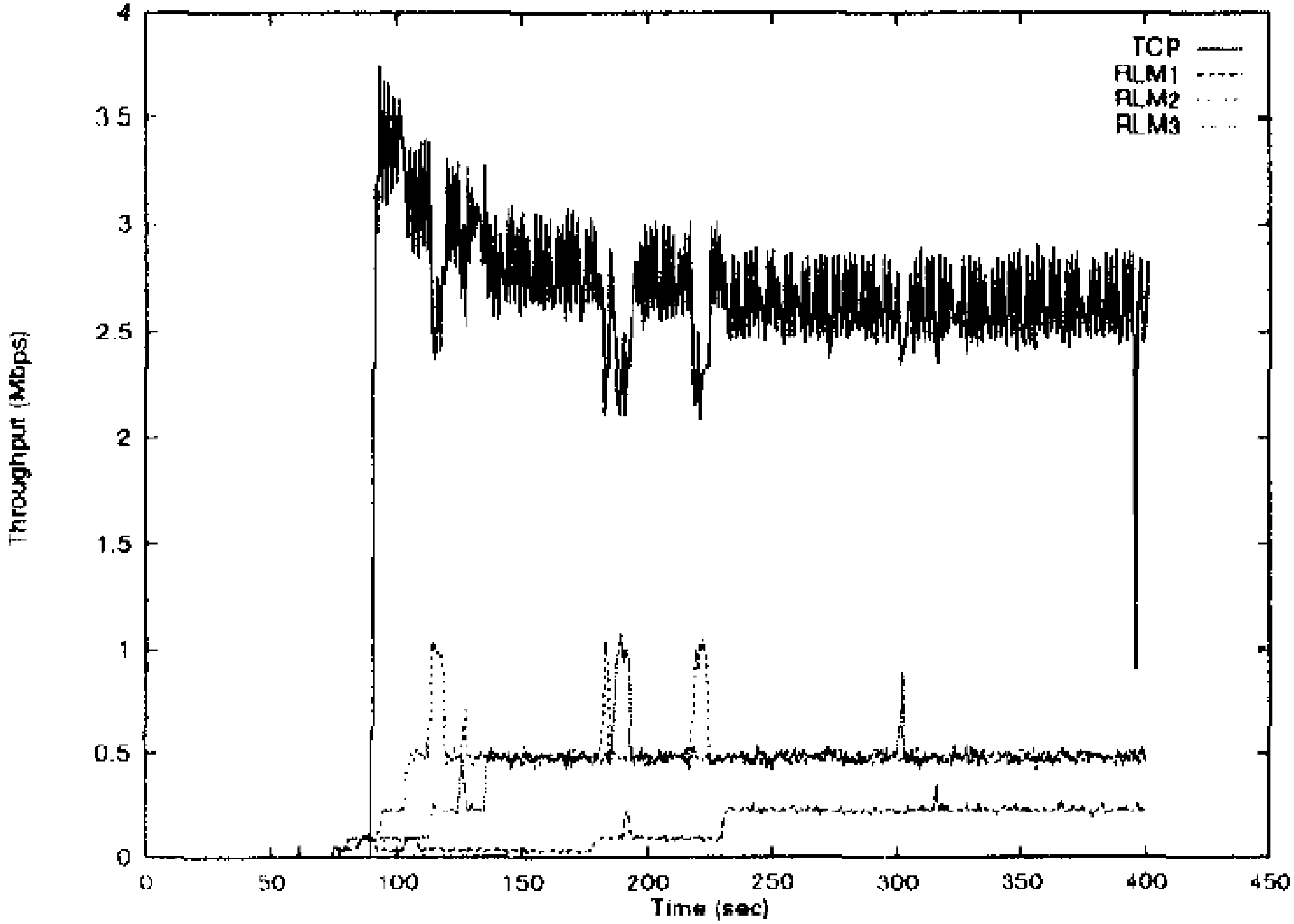
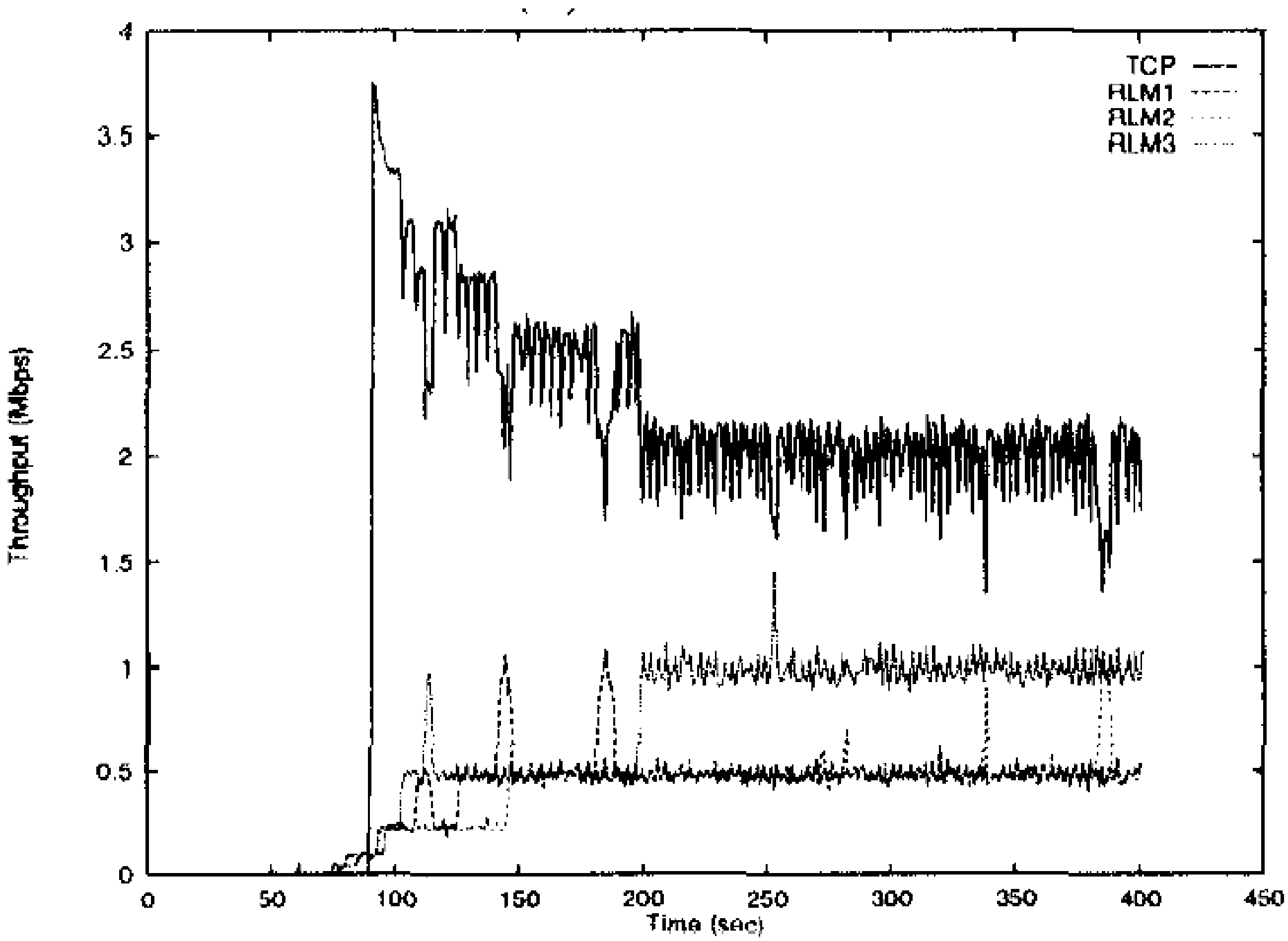


Figure 7e : FIFO



On-Off Flow Model

- One approach to modeling interactive, Web traffic: OFF represents “think time”
- 10Mbps link shared by one ON-OFF source and 19 CBR sources: 0.5Mbps each
- ON and OFF drawn from exponential distribution with means of 200 ms and 19*200 ms respectively
- During ON period source sends at 10 Mbps

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

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

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 inter-arrivals are exponentially distributed with mean 0.05 ms and Pareto distribution that yields a mean connection length of 20,1 KB packets
- 1 UDP flow: 10 Mbps

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

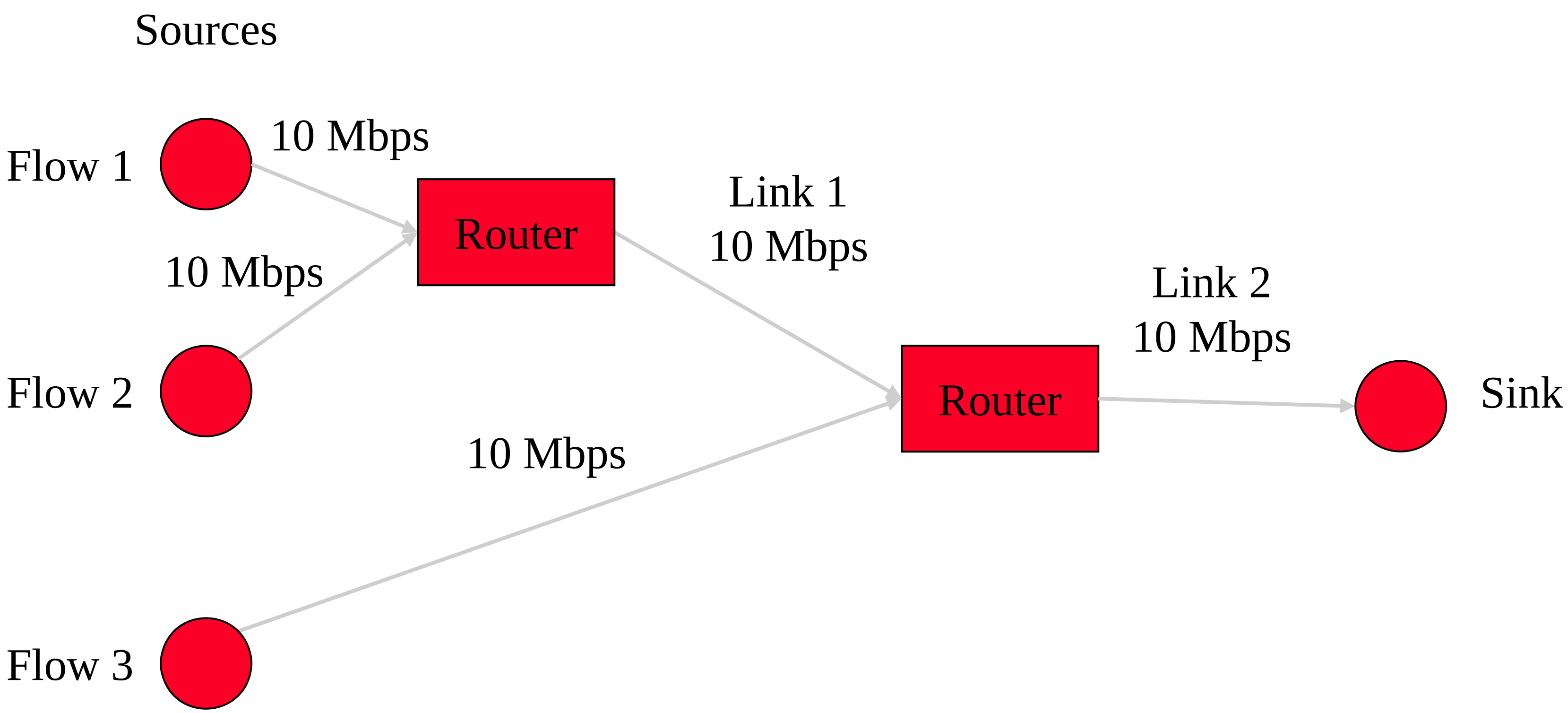


Table 4 : UDP and TCP with Packet Relabeling

Traffic	Flow 1	Flow 2	Flow 3
UDP	3.36	3.32	3.28
TCP	3.43	3.13	3.43

Unfriendly Flows

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

Conclusions

- This paper presents Core Stateless Fair Queueing and offers many simulations to show how CSFQ provides better fairness than RED or FIFO
- **CSFQ ‘clobbers’ UDP flows!**

Conclusions

- CSFQ does not require per flow state within the core
- CSFQ performance comparable to DRR (which however requires per flow state)
- superior to FRED (“partial” per flow state)
- much better than RED, FIFO (no per flow state)
- large latency and propagation delay effects (such as on a cross country connection or on a satellite segment) still to be explored
- use of TOS field (ie, packet state) potentially controversial

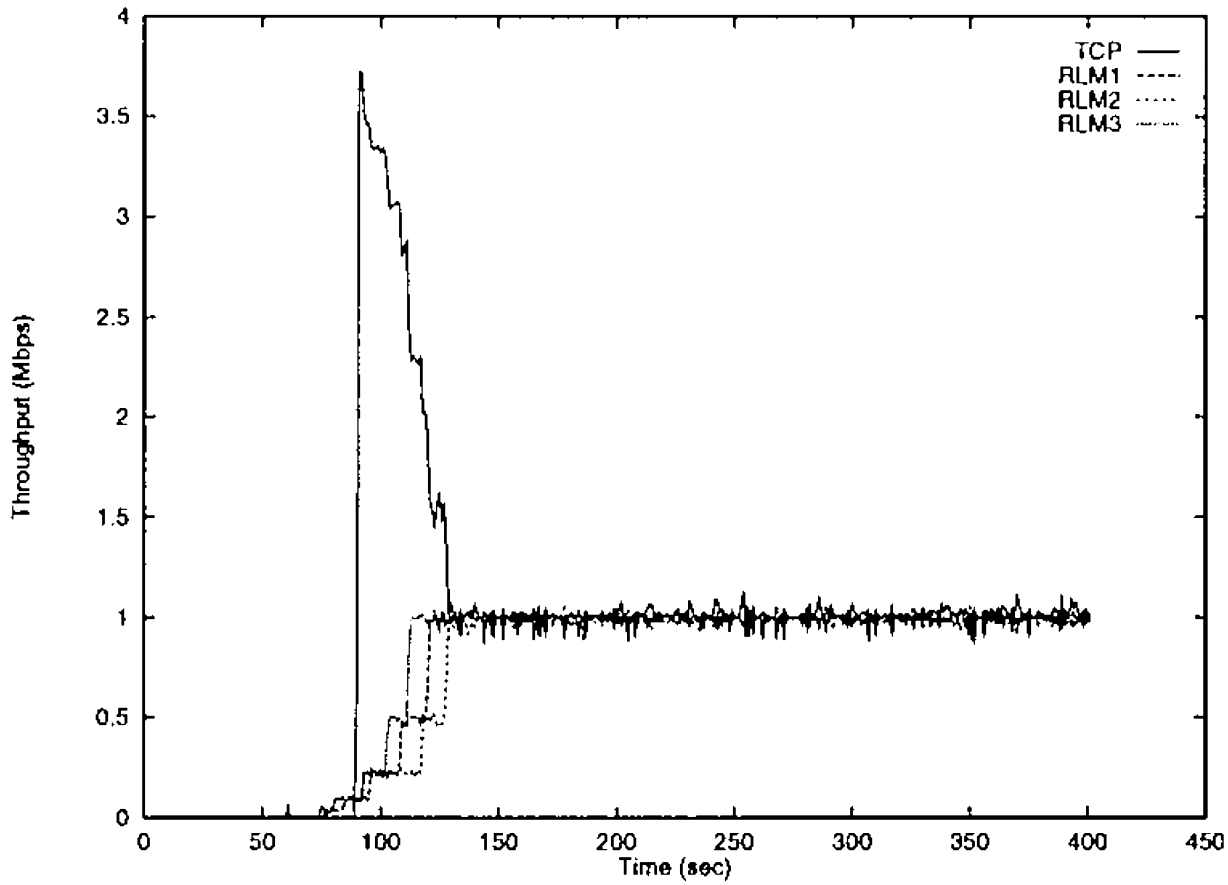
Significance

- First paper to use hints from the edge of the network
- Deals with UDP; Many algorithms do not
- Makes a reasonable attempt to look at a variety of traffic types

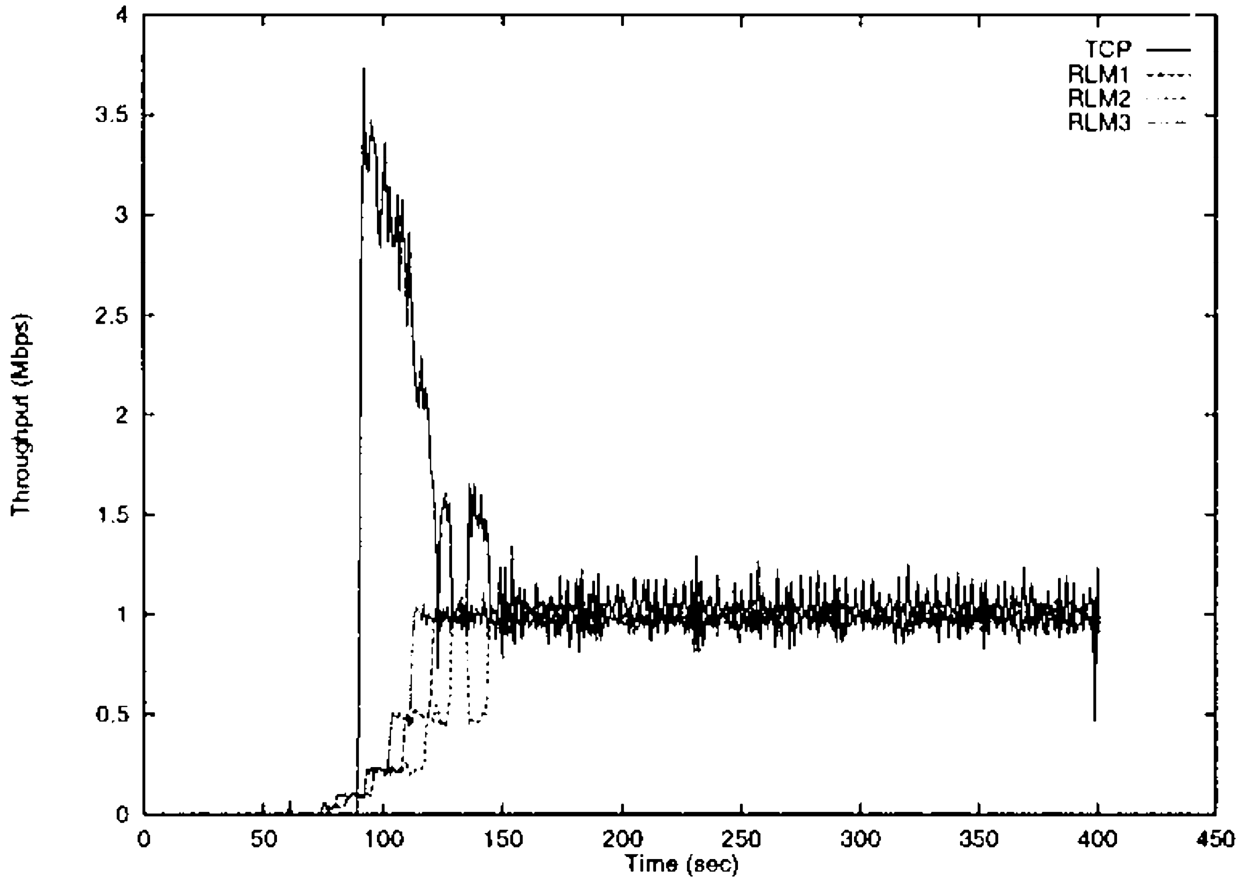
Problems/ Weaknesses

- How does one set K constants for a variety of situations
- No discussion of algorithm “stability”

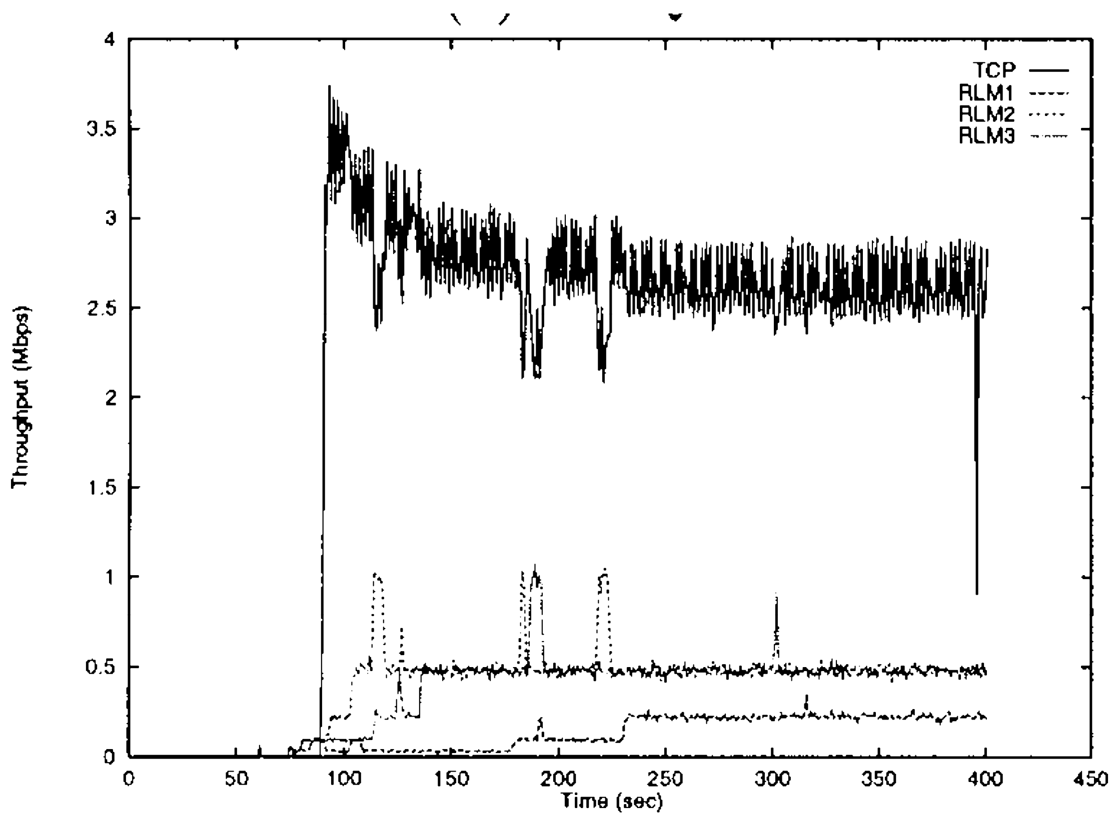
Coexistence of TCP and Receiver Layered Multicast: DRR



Coexistence of TCP and Receiver Layered Multicast: CSFQ



Coexistence of TCP and Receiver Layered Multicast: RED



(d) RED