ECE 463
Introduction to Computer Networks

Lecture: Router Mechanisms

Discussion

- TCP Congestion Control
- Fundamental Assumption:
  - End System Based.

- This class:
  - Why adding router support may help/be important?
  - What mechanisms can be added at the router?
    - Seen traction, but not as widely deployed

Queuing Disciplines

- Key decisions router must make:
  - Which packet to serve (transmit) next
  - Which packet to drop next (when required)

- What's used in the Internet today?
  - FIFO (packet that arrives earlier is served earlier)
  - Drop-tail (if packet arrives and router buffer is full, the arriving packet is dropped)

Limitations of purely end-system based mechanisms

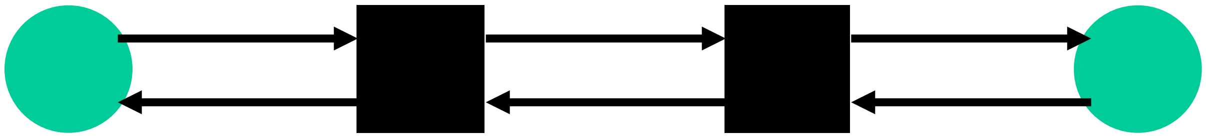
- Not using information available at routers
  - No “early hints” regarding congestion
  - Wait till large queues build up, leading to loss
- No Policing against bad flows.
  - All flows must use TCP and “play the game” correctly
  - No mechanisms to punish a malicious flow
- Synchronization
  - Congestion => All TCP flows slow down
  - As network gets better => All TCP flows ramp up.

Explicit Router Feedback

- Involve routers to aid congestion control
  - Router has unified view of queuing behavior
  - Routers can distinguish between propagation and persistent queuing delays
  - Routers can decide on transient congestion, based on workload

Example 1: DEC bit

- Switches set an explicit congestion bit in the packet header if the queue size is larger than one.
  - Receiver collects the information and forwards it to the sender in ACKs.
- Senders slow down if the bit is set in more than 50% of the packets in a window.
  - multiplicative slow down
  - stepwise increase if bit is not set for certain period of time
- Behavior is very similar to TCP, except that it has explicit feedback.



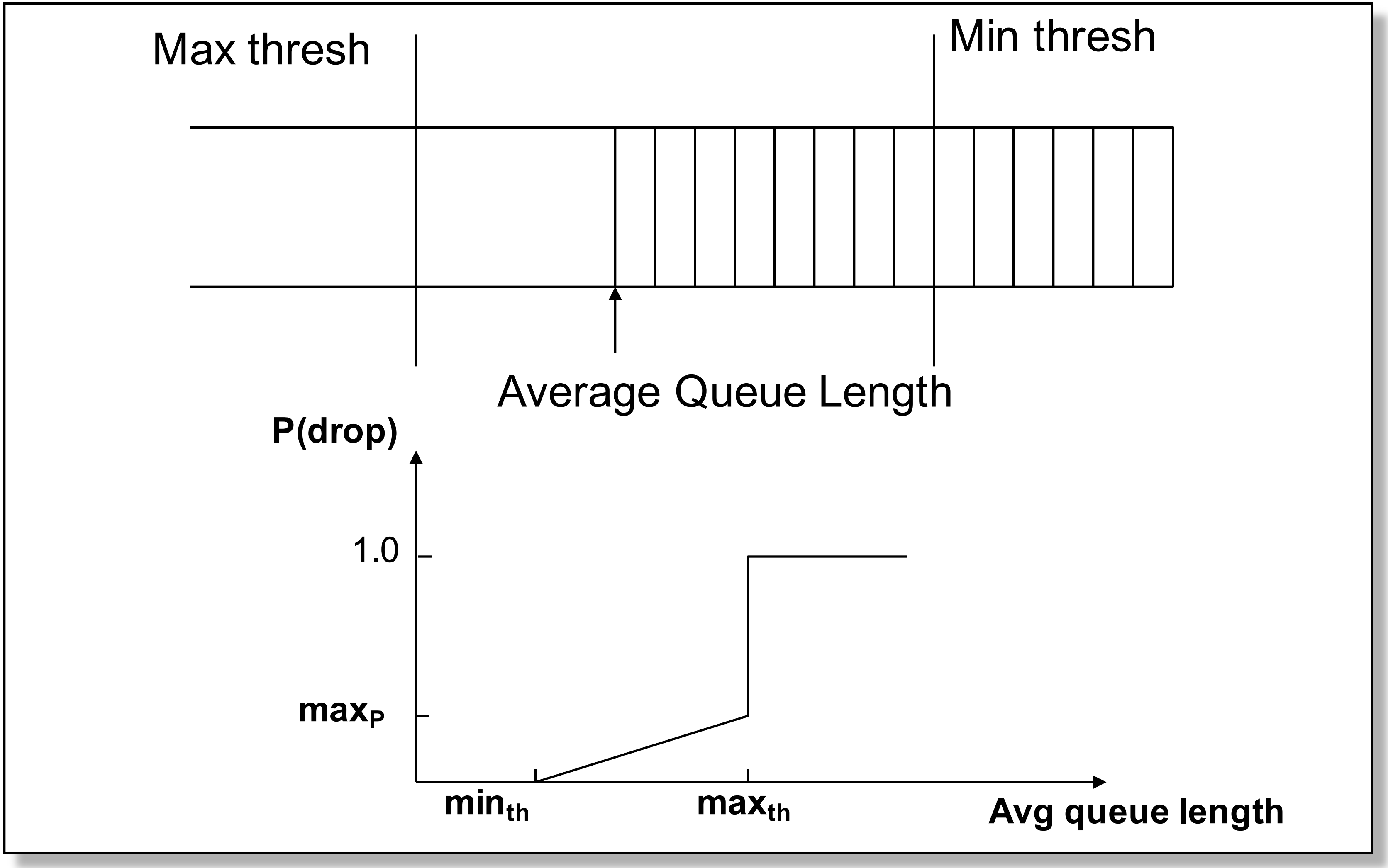
Example 2: RED

- Random Early Detection
- Drop packets before queue becomes full (early drop)
- Intuition: notify senders of incipient congestion
  - Example: early random drop (ERD):
    - If qlen > drop level, drop each new packet with fixed probability *p*
    - Does not control misbehaving users

RED Algorithm

- Maintain running average of queue length
- If avg < min_{th} do nothing
  - Low queuing, send packets through
- If avg > max_{th}, drop packet
  - Gentle approach not working, more drastic measures needed
  - Some researchers have argued for a more continuous transition to complete dropping
- Else drop packet with probability proportional to queue length
  - Notify sources of incipient congestion

RED Operation



More details

- AvgLen:
  - Weighted running average, like TCP timeout
  - AvgLen= (1-Weight) * AvgLen + Weight * SampleLen
    - SampleLen: queue length when measurement made
    - Weight: between 0 and 1.
- Prob of dropping:
  - Not only depends on queue thresholds
  - But also on how long since last packet dropped.
  - See Textbook, page 491.

Discussion: Schemes so far…

- No fundamental change to state in routers
  - Improvements over TCP?
  - Short-comings?

Internet Classifier

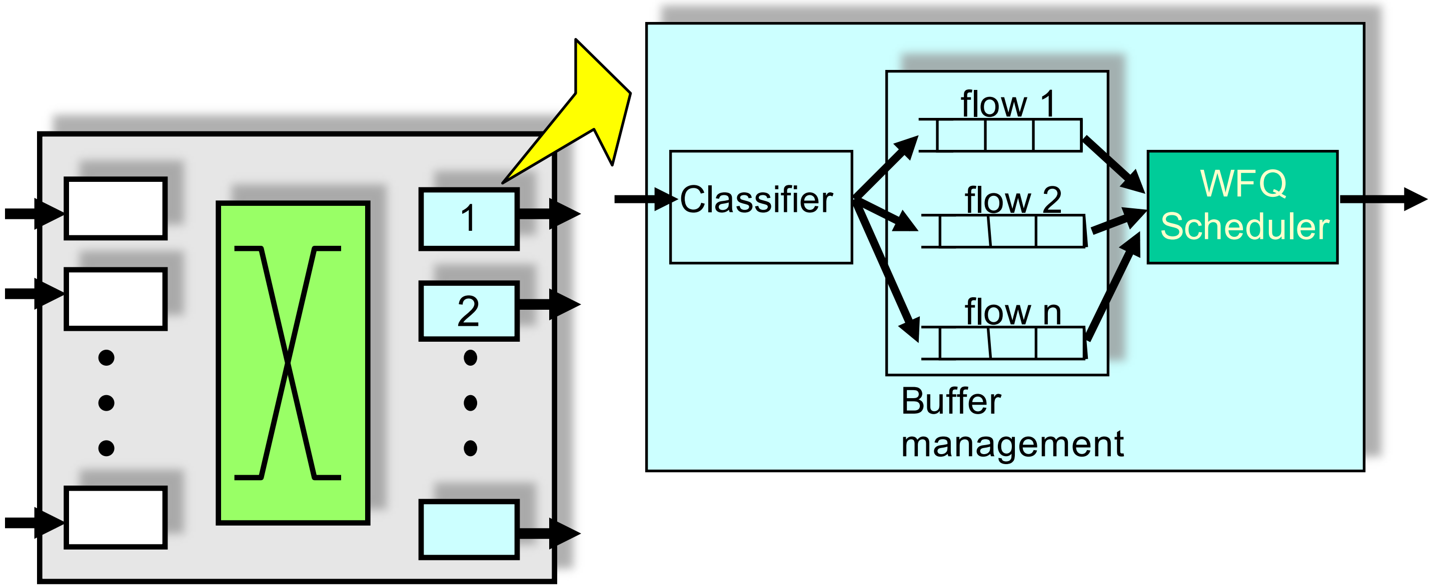
- A “flow” is a sequence of packets that are related (e.g. from the same application)
  - source/destination IP address (32 bits)
  - source/destination port number (16 bits)
  - protocol type (8 bits)
  - type of service (4 bits)

Key Idea

- Employ at routers:
  - Per-flow classification
  - Per-flow queuing

- Used today more predominant at the "edge" of the network
  - E.g. At campus routers
  - Use classifier to pick out Kazaa packets
  - Use scheduler to limit bandwidth consumed by Kazaa traffic

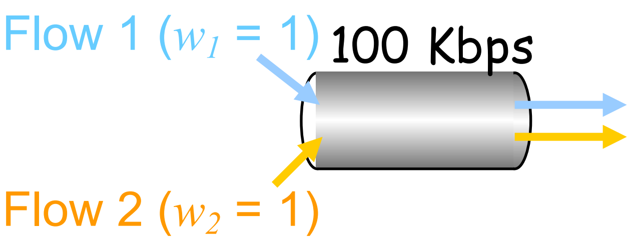
WFQ Architecture



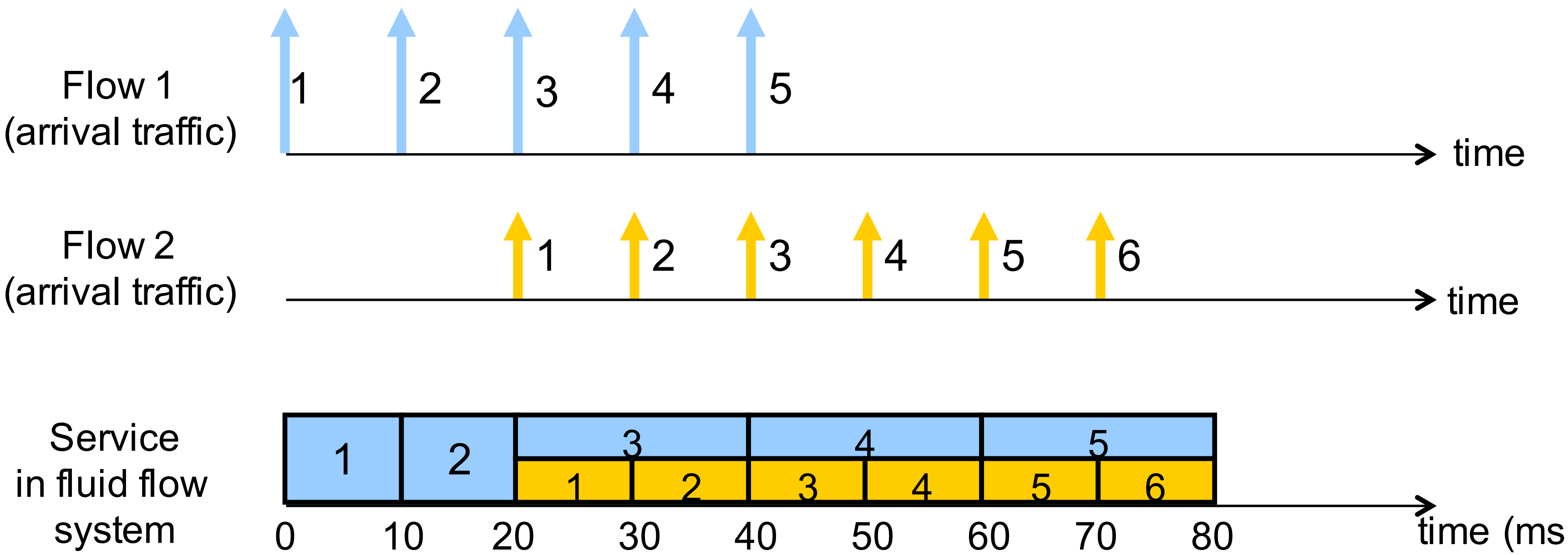
Ideal Implementation

- Bit-by-bit weighted round robin
  - During each round from each flow that has data to send, send a number of bits equal to the flow's weight

Fluid Flow System: Example 1

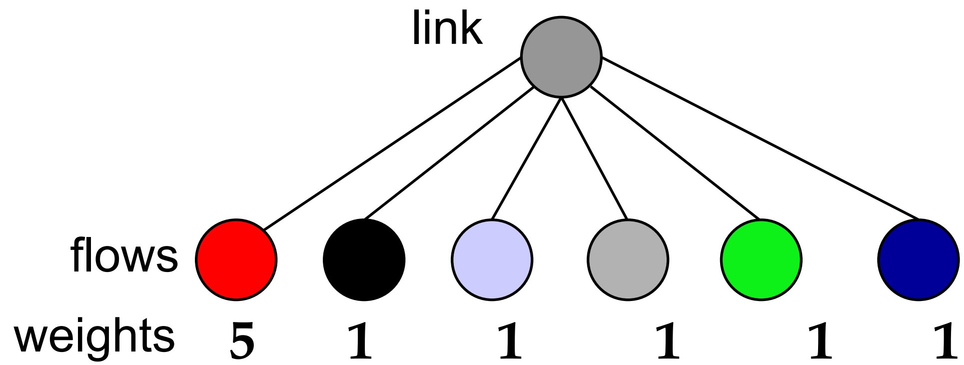


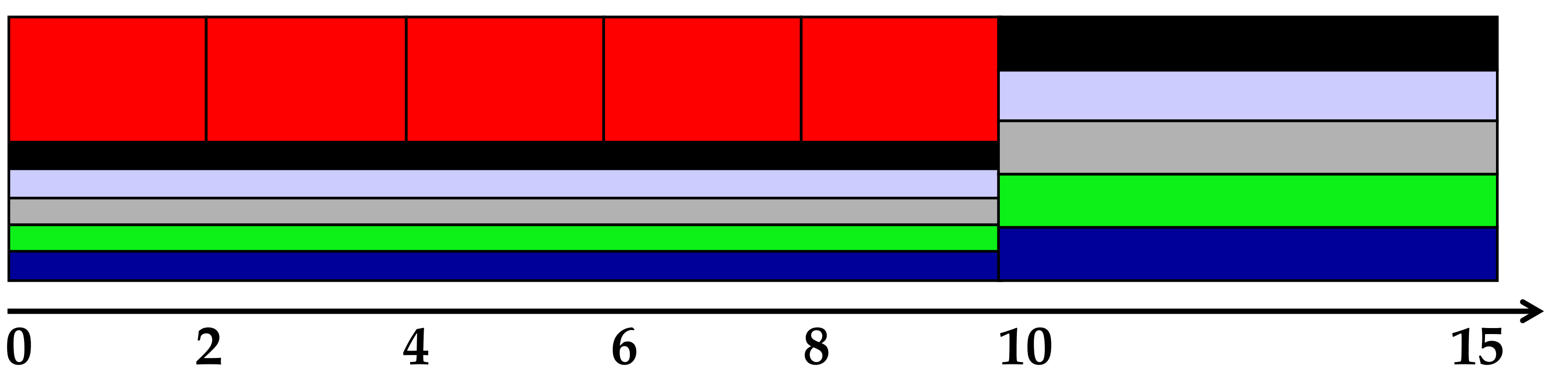
	Packet Size (bits)	Packet inter-arrival time (ms)	Arrival Rate (Kbps)
Flow 1	**1000**	**10**	**100**
Flow 2	**500**	**10**	**50**



Fluid Flow System: Example 2

- Red flow has packets backlogged between time 0 and 10
  - Backlogged flow → flow's queue not empty
- Other flows have packets continuously backlogged
- All packets have the same size

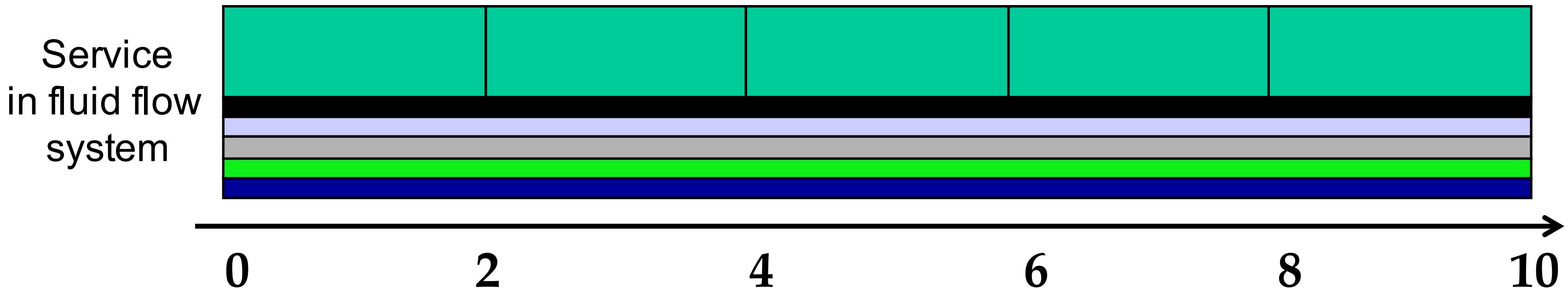




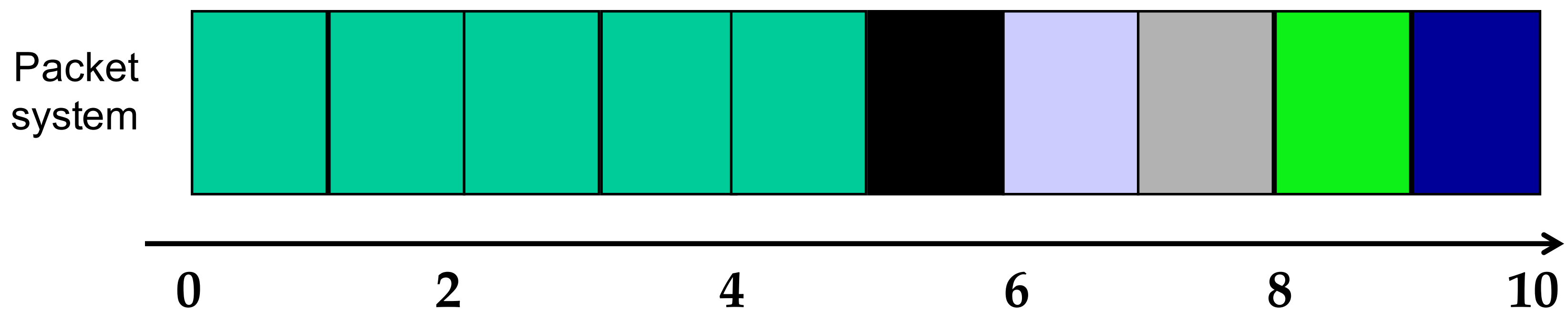
Real Implementation

- Bit-by-bit RR not feasible in practice.
- Packet-by-packet RR?
  – No. Key issue: different flows – different packet sizes
- Solution:
  – “Emulate” Bit-by-Bit RR.
  – Serve packets in order of finish time in ideal model

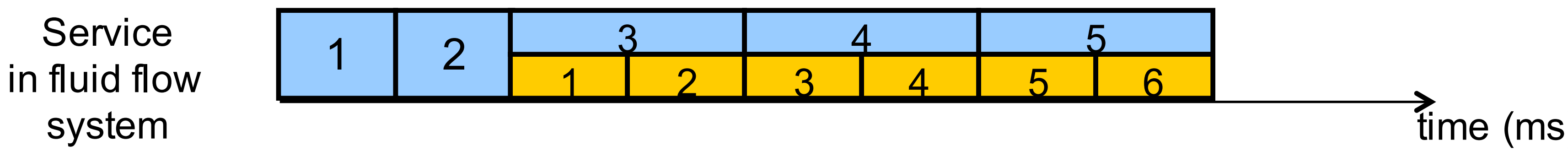
Packet System: Example 1



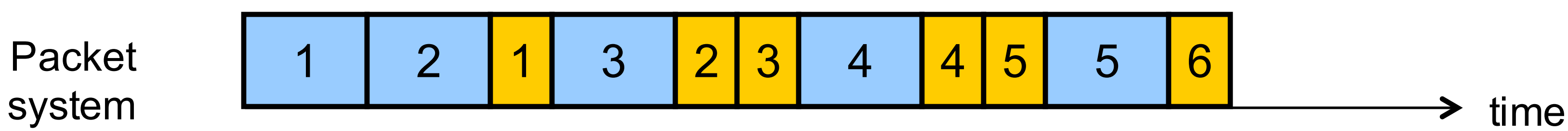
- Select the first packet that finishes in the fluid flow system



Packet System: Example 2

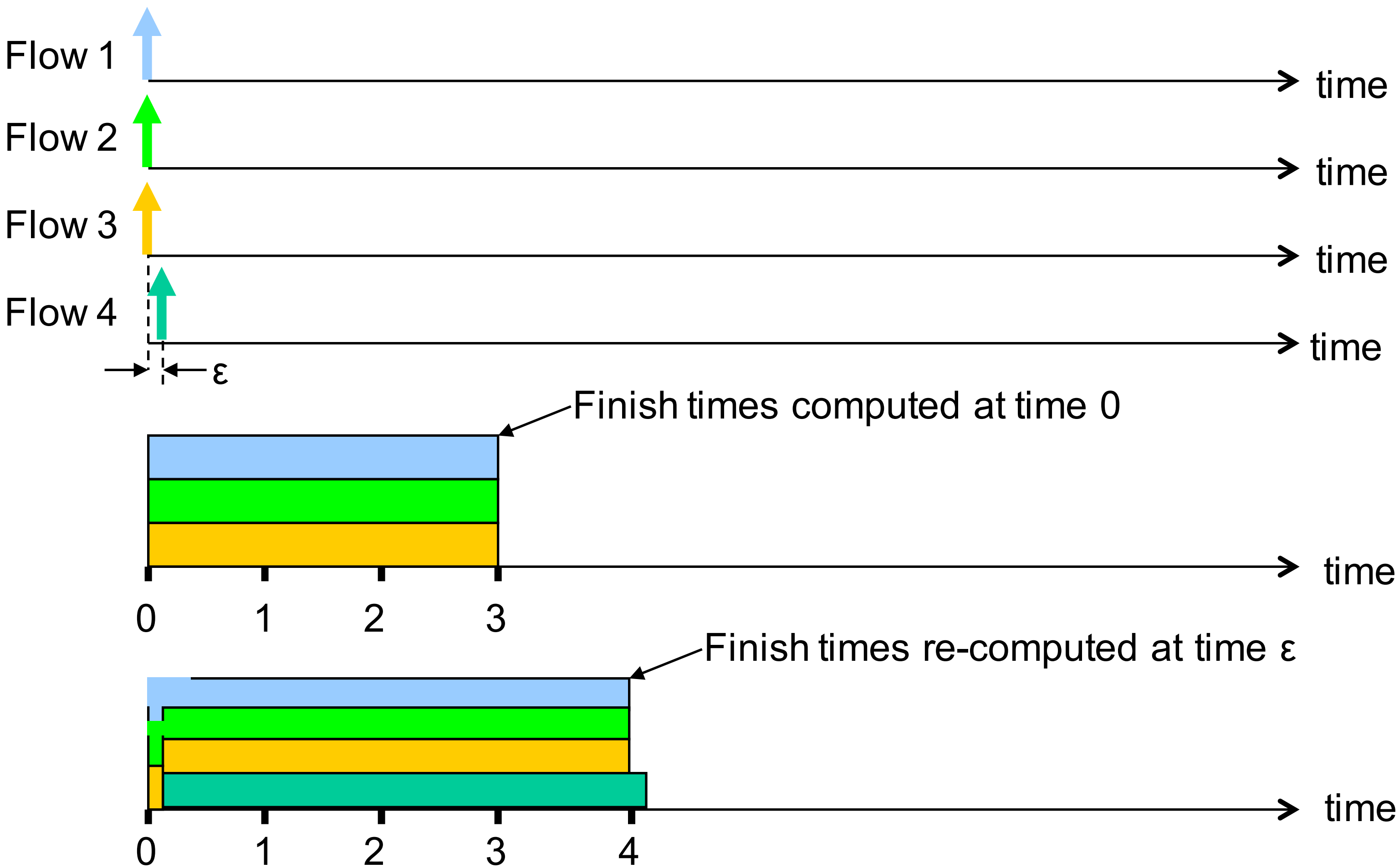


- Select the first packet that finishes in the fluid flow system



Issues with computing finish time

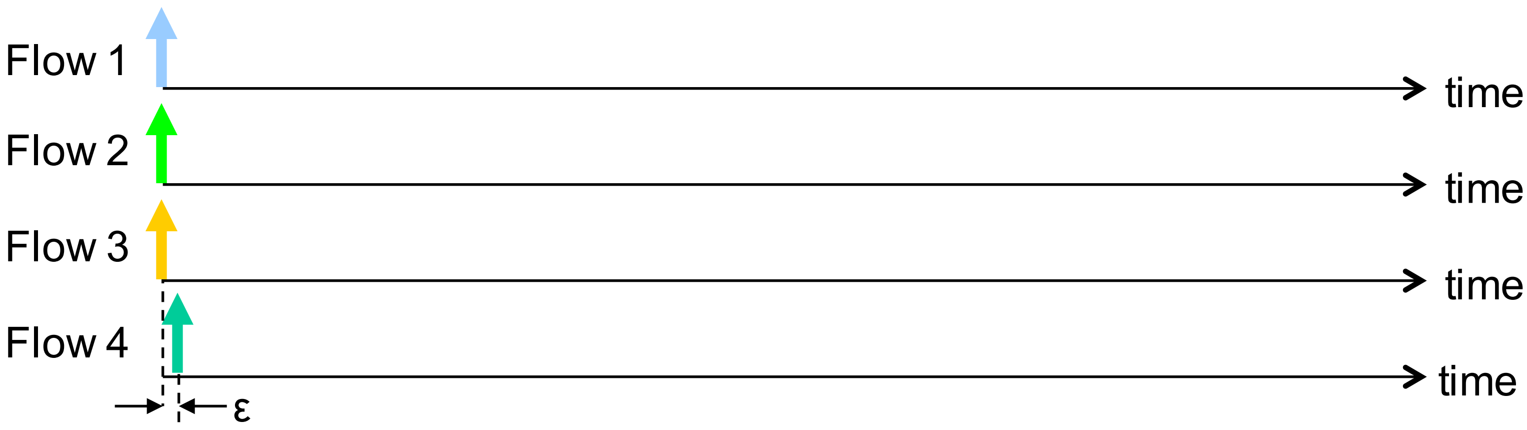
- Four flows, each with weight 1



“Virtual” Finish Time (VFT)

- Wall clock finish time depends on number of active flows
- Solution: Maintain the round # when a packet finishes
- System virtual time V(t) – index of the round in the bit-by-bit round robin scheme
- When a new packet arrives:
  - “Virtual finish time” doesn’t change
  - Order in which 2 packets already in system finish does not change

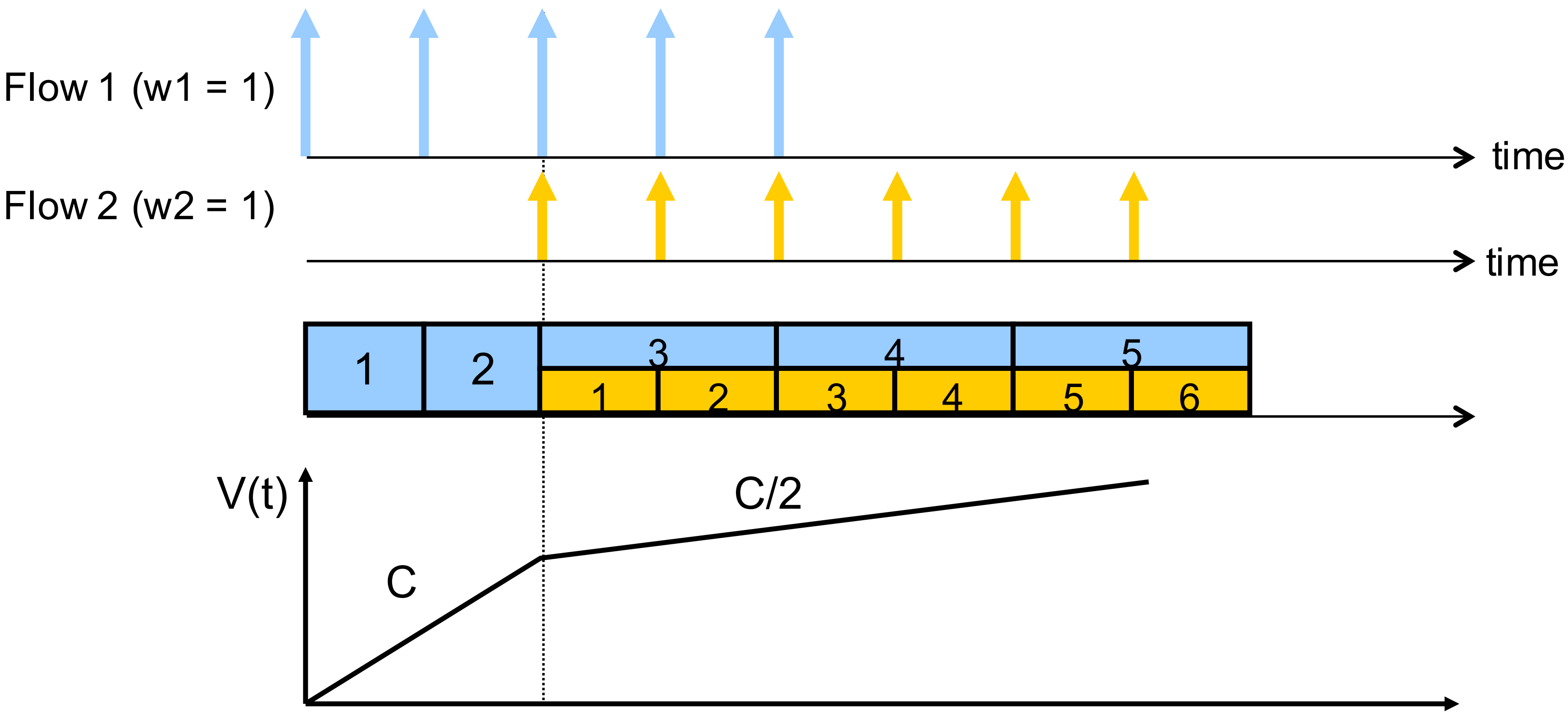
Example



- Suppose each packet is 1000 bits, so takes 1000 rounds to finish
- So, packets of F1, F2, F3 finishes at virtual time 1000
- When packet F4 arrives at virtual time 1 (after one round), the virtual finish time of packet F4 is 1001
- But the virtual finish time of packet F1,2,3 remains 1000
- Finishing order of F1,2,3 is preserved

System Virtual Time (Round #): V(t)

- V(t) increases inversely proportionally to the sum of the weights of the backlogged flows
- Since round # increases slower when there are more flows to visit each round.



Fair Queueing Implementation

- Define
  - F_i^k - virtual finishing time of packet k of flow i
  - a_i^k - arrival time of packet k of flow i
  - L_i^k - length of packet k of flow i
  - w_i – weight of flow i

- The finishing time of packet $k+1$ of flow i is

$$F_i^{k+1} = \max(V(a_i^{k+1}), F_i^k) + L_i^{k+1}/w_i$$

- Smallest virtual finishing time first scheduling policy

FQ: Pros

- Achieve fair allocation
  - Can be used to protect well-behaved flows against malicious flows

- Can be used to provide guaranteed services

Fair Queuing: Cons

- Complex state
  - Must keep queue per flow
    - Hard in routers with many flows (e.g., backbone routers)
    - Flow aggregation is a possibility (e.g. do fairness per domain)
- Complex computation
  - Classification into flows may be hard
  - Must keep queues sorted by finish times

Summary

- Sophistication at routers can help
  - RED → eliminate full-queues problems
  - FQ → heavy-weight but explicitly fair to all
  - Can be useful to enable QoS