Router mechanisms for Congestion Control

ECE 50863 – Computer Network Systems

Discussion

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

- Next topic:
  - Why adding router support may help/be important?
  - What mechanisms can be added at the router?
    - Some deployment in wide-area networks, but more recently, seeing extensive usage in data-center networks

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 misbehaving flows.
  - All flows must use TCP and “play the game” correctly
  - No mechanisms to punish a misbehaving/greedy flow
- Synchronization
  - Congestion => All TCP flows slow down
  - As network gets better => All TCP flows ramp up.

Summary of router-based schemes

1. Explicit Congestion Notification (ECN), also known as DEC-BIT scheme.
2. Random Early Detection (RED)
3. Weighted Fair Queuing (WFQ)

- First two schemes (ECN and RED):
  - Relatively simple changes to routers
  - But do not protect against misbehaving flows
- Last scheme (WFQ):
  - More involved router changes
  - Do protect against misbehaving flows.

Explicit Congestion Notification

- Routers provide explicit feedback regarding congestion
  - Router has unified view of queuing behavior
  - Routers can observe persistent queuing delays
  - Routers can decide on transient congestion

ECN, or DEC-Bit scheme

- Routers set an explicit congestion bit in the packet header if the queue size is larger than a threshold.
  - Receiver collects the information and forwards it to the sender in ACKs.
- Senders slow down if the bit is set in more than a fraction 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.

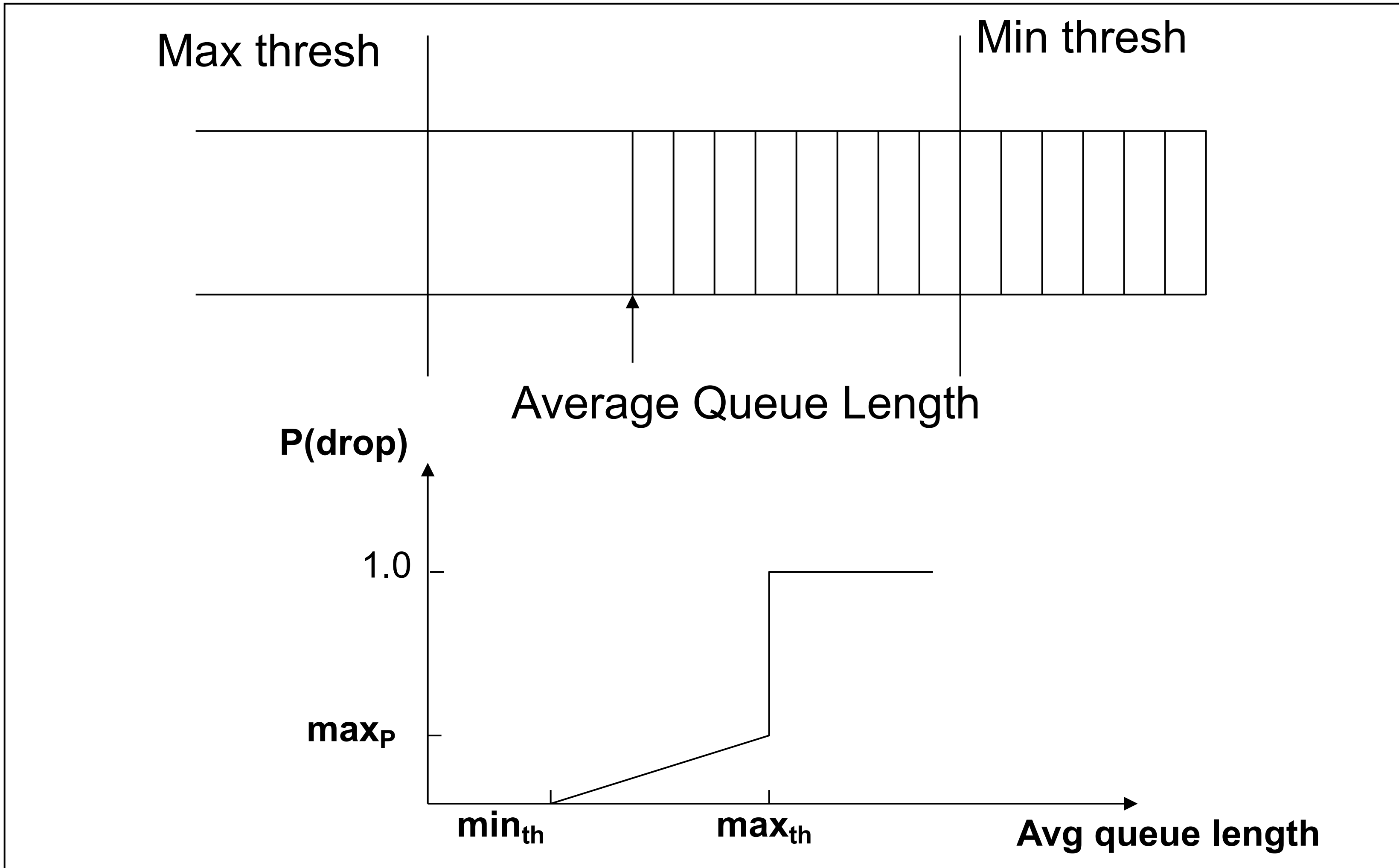
Random Early Detection (RED)

- Random Early Detection
- Drop packets before queue becomes full (early drop)
- Intuition: notify senders of incipient congestion

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
- Else drop packet with probability proportional to queue length
  - Notify sources of incipient congestion

RED Operation



Summary of router-based schemes

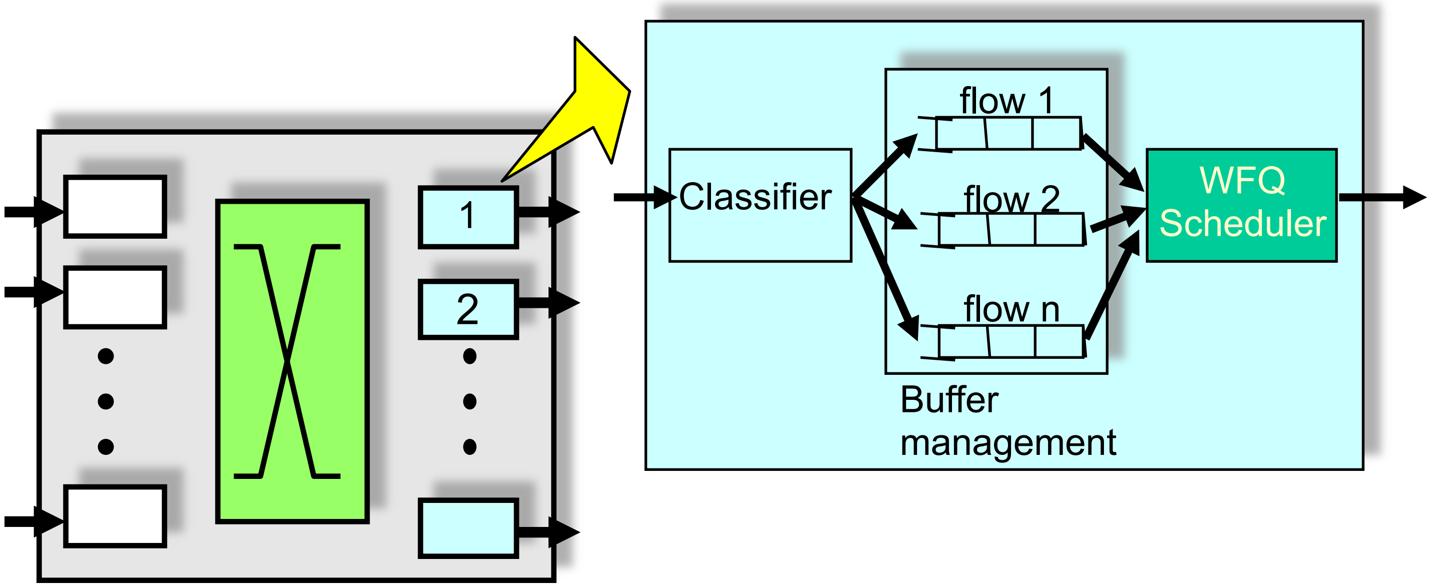
- So far discussed two schemes (ECN and RED):
  - Relatively simple changes to routers
  - But do not protect against misbehaving flows
  - Provide early warning of congestion through more direct router feedback
  - Avoid synchronization issues.

- Next topic: WFQ:
  - More involved router changes
  - Do protect against misbehaving flows.

Key Ideas behind WFQ

- Classify traffic into different flows
  - Each flow's traffic goes into a different queue
- A "flow" is a sequence of packets that are related
- Most fine-grained:
  - Same source/destination IP address and port numbers
- More coarse-grained:
  - Traffic to same destination in one queue
  - Separate audio vs. video vs. data traffic in different queues

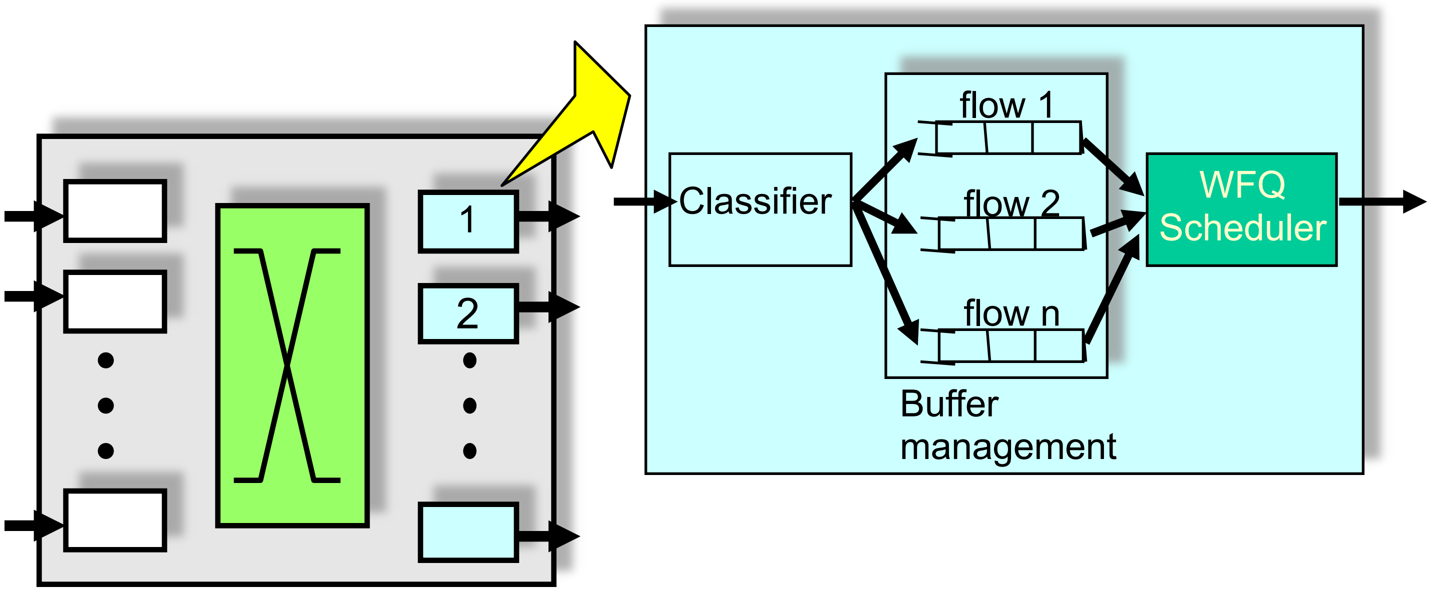
WFQ Architecture



Weighted Fair Queuing

ECE 50863 – Computer Network Systems

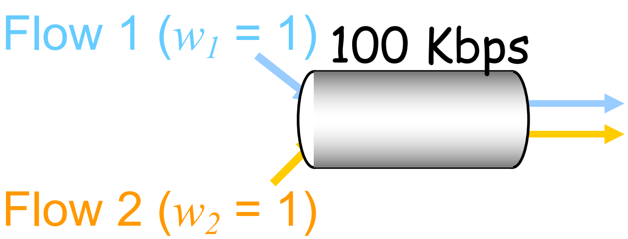
WFQ Architecture



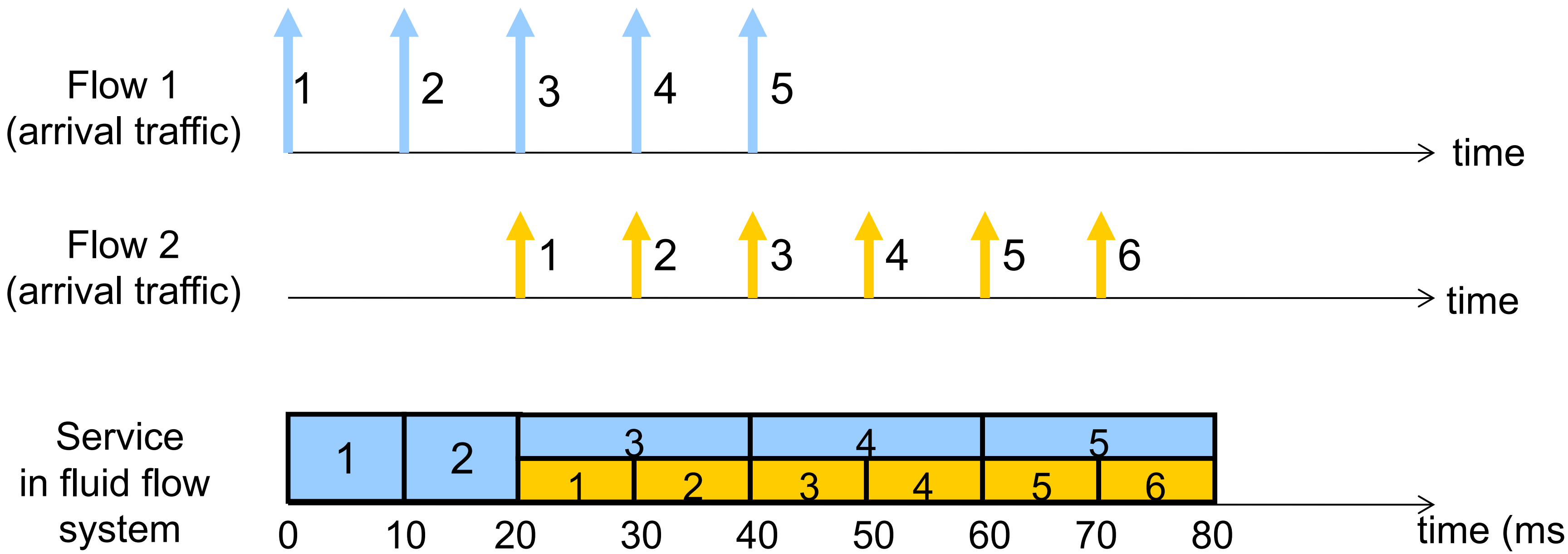
Ideal Implementation

- 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

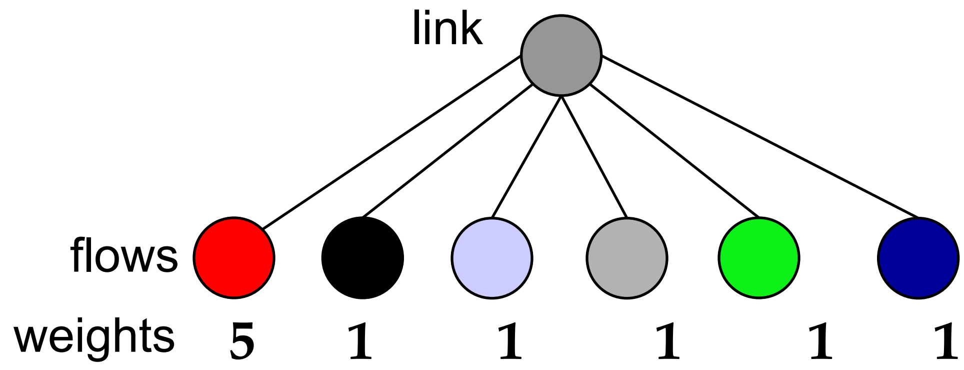


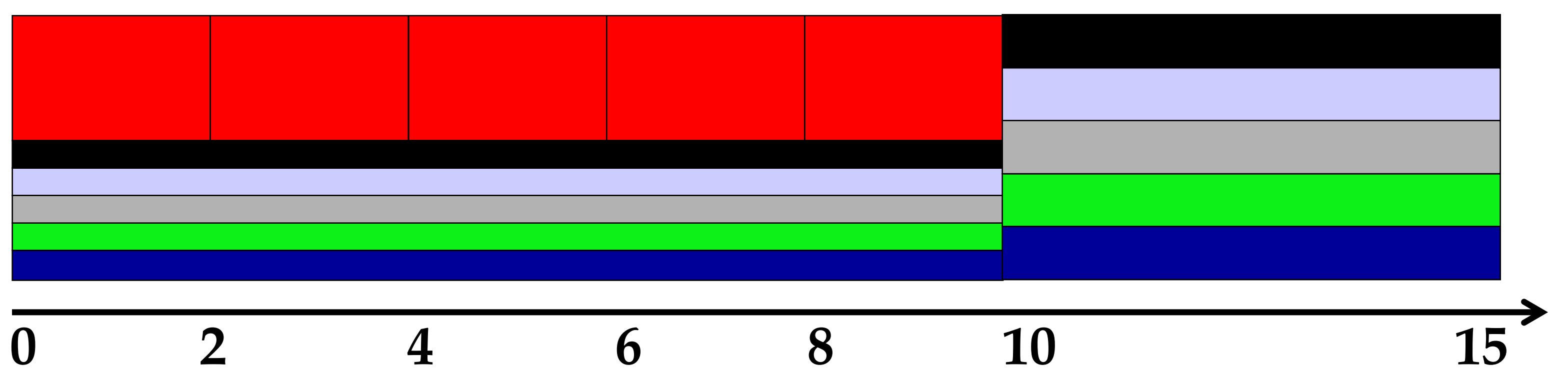
	Packet Size (bits)	Packet inter-arrival time (ms)	Arrival Rate (Kbps)
Flow 1	**1000**	**10**	**100**
Flow 2	**500**	**10**	**50**



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

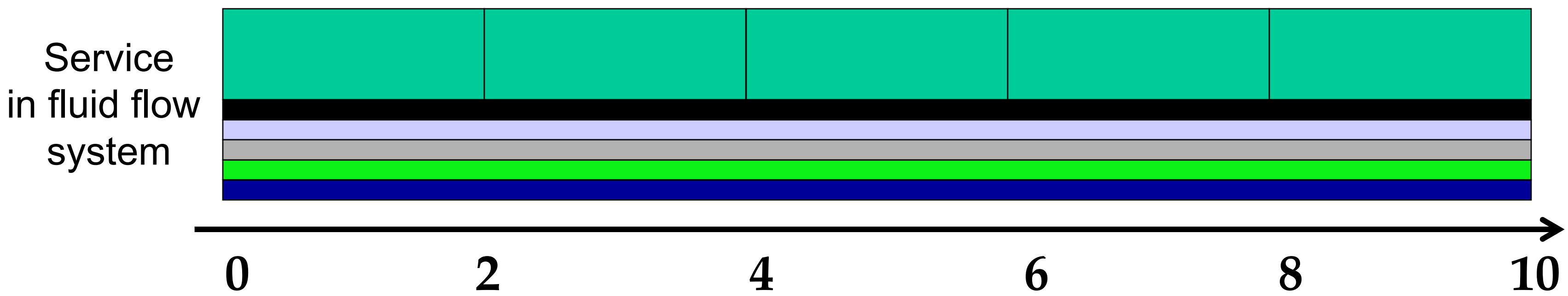




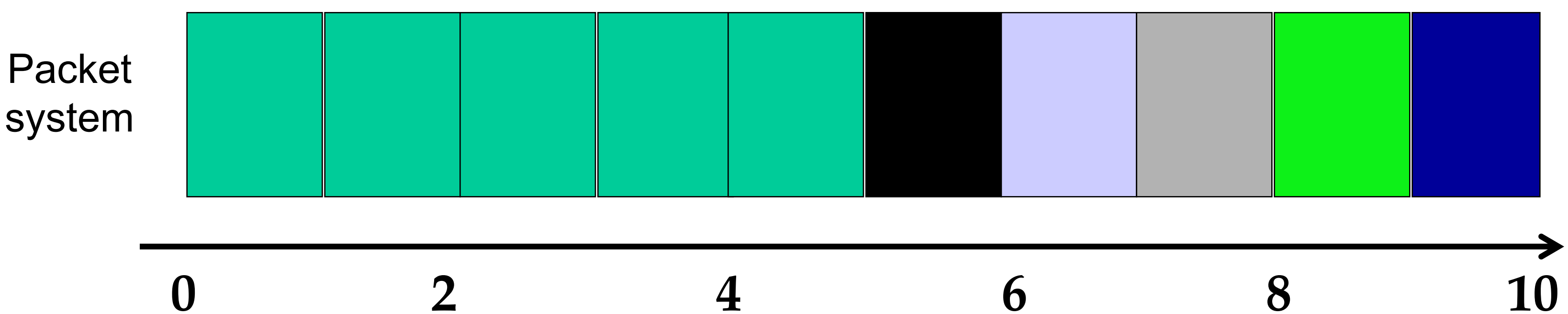
Real Implementation

- 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

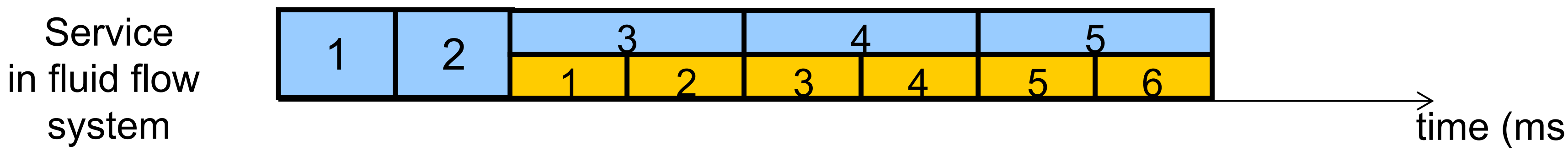
Packet System: Example 1



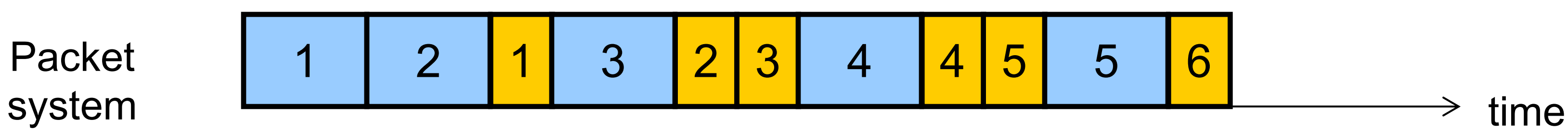
- Select the first packet that finishes in the fluid flow system



Packet System: Example 2

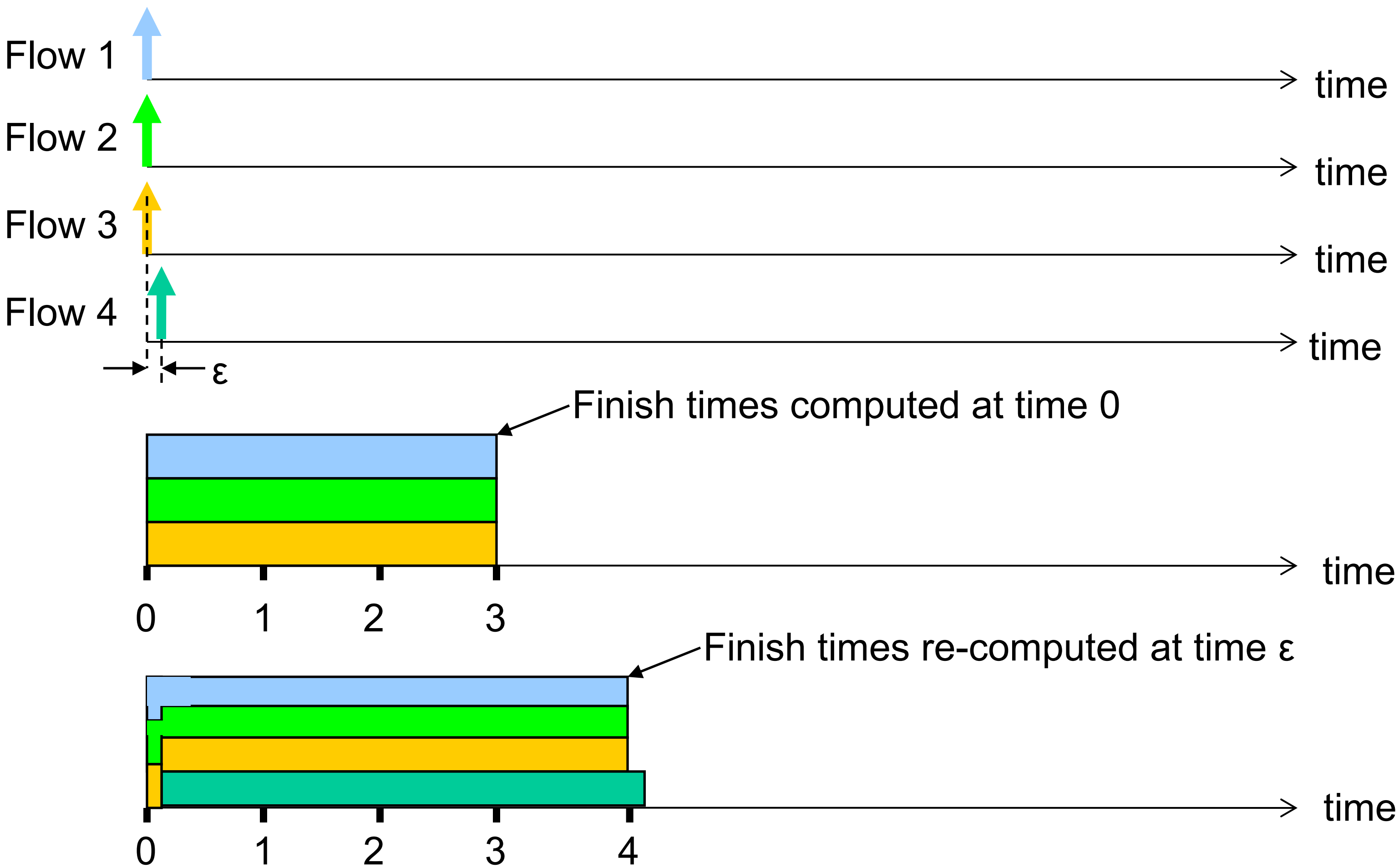


- Select the first packet that finishes in the fluid flow system



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
  - Need to recompute for all packets of all flows everytime a single new packet comes in
  - Expensive
- 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

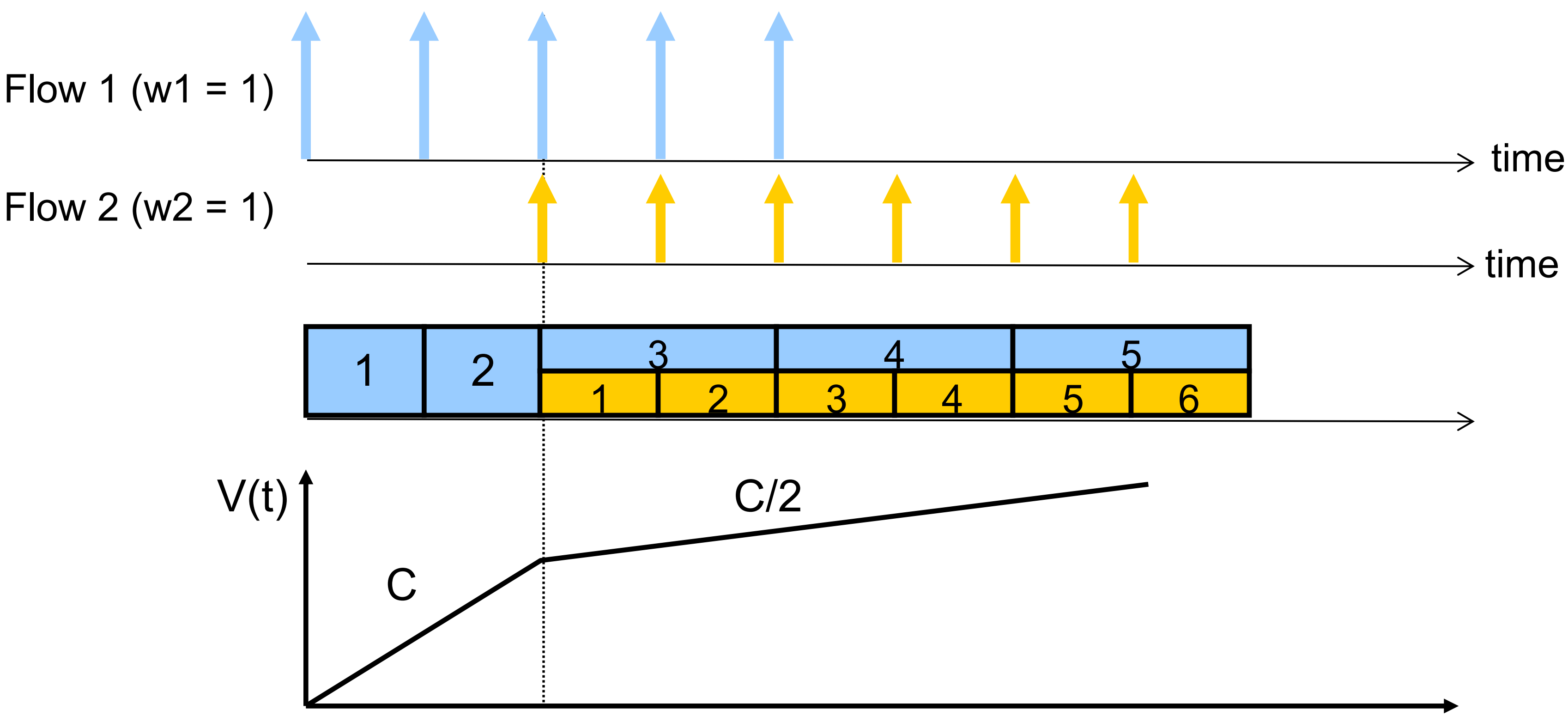
- When a new packet arrives:
  - “Virtual finish time” doesn’t change
  - Order in which 2 packets already in system finish does not change

Example

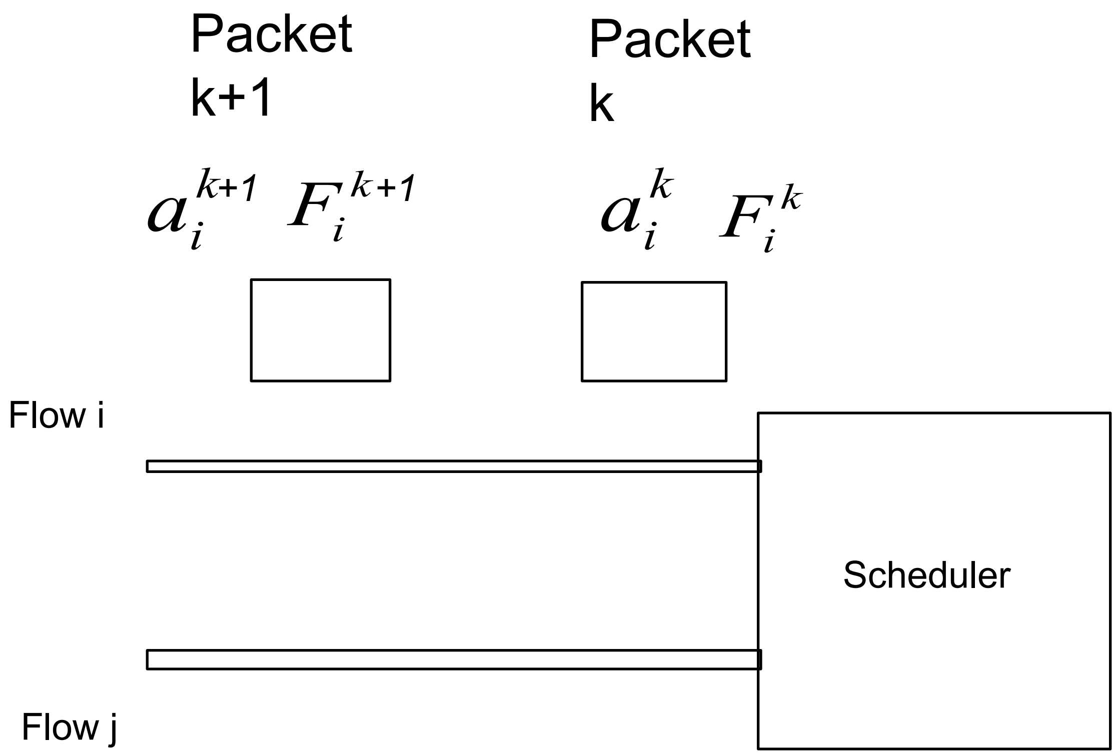
- Suppose each packet is 1000 bits, so takes 1000 rounds to finish
- So, packets of F1, F2, F3 finishes at virtual time 1000
- When packet F4 arrives at virtual time 1 (after one round), the virtual finish time of packet F4 is 1001
- But the virtual finish time of packet F1,2,3 remains 1000
- Finishing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



Scheduler



Fair Queueing Implementation

- Define
  - F_i^k - virtual finishing time of packet k of flow i
  - a_i^k - virtual 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

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

Scheduling algorithm

- Smallest virtual finishing time first scheduling policy
  - Packets sorted in order of virtual finish time
  - Compute virtual finish time for newly arriving packet
    - Virtual finish times of other packets unaffected
  - Insert in sorted order
  - Serve the next packet with smallest virtual finishing time.

Approximation vs. Ideal

- WFQ policy “emulates” ideal fluid flow model.
- When is there a discrepancy between the two?

	Arrival Time	Finish Time
Packet P1 (Flow 1)	200	1000
Packet P2 (Flow 2)	250	300

Ideal Model: P2 finishes first
Real Model: P2 arrives later. If the router already started servicing P1, then, P1 cannot be preempted and it would finish first.

FQ: Pros

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

- Can be used to provide guaranteed services
  - If all routers run WFQ, and
  - Traffic regulated using a mechanism called “token bucket”
  - Feasible to bound end-to-end delay experienced by packets.

Fair Queuing: Cons

- Complex state
  - Must keep a queue per flow
    - Hard in routers with many flows (e.g., backbone routers)
    - Flow aggregation is a possibility (e.g. based on destination network)
- Complex computation
  - Classification into flows may be hard
  - Must keep queues sorted by finish times
- Ideas seeing a resurgence in Data Center Networks