

Lecture 22:

Buffering & Scheduling

CSE 123: Computer Networks

Aaron Schulman





Lecture 22 Overview

- Router traffic Policing/Scheduling

Scheduling

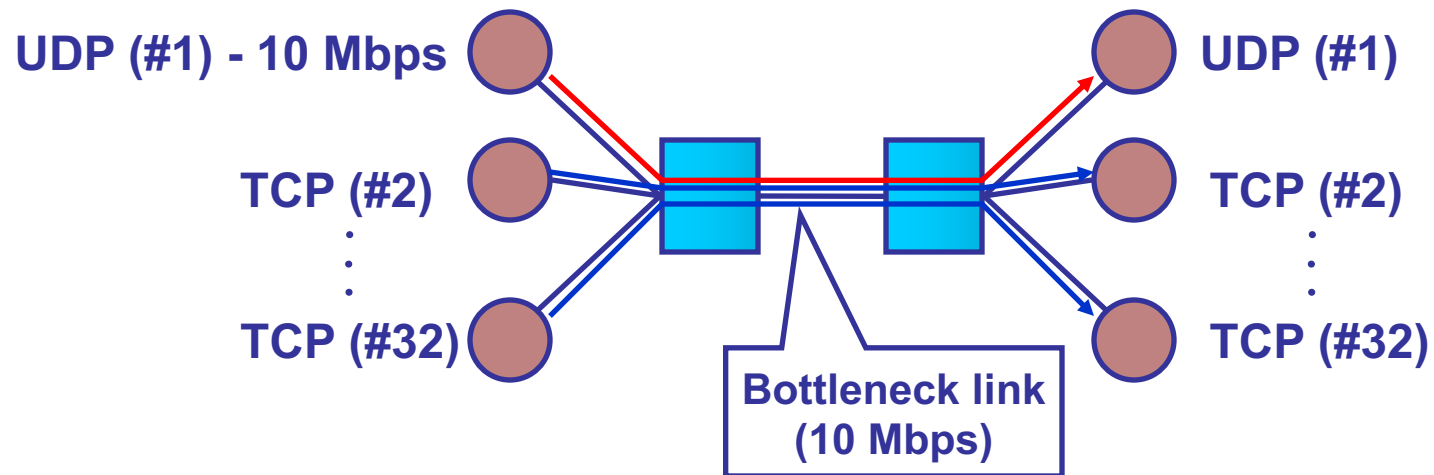


- So far we've done **traffic policing**
 - ◆ Limit the rate of flows regardless of the load in the network
- Routers also need **scheduling**
 - ◆ Dynamically allocate resources when multiple flows compete
 - ◆ Give each “flow” (or src/destination pair) own queue (at least theoretically)
- Weighted fair queuing
 - ◆ Proportional-share scheduling
 - ◆ Schedule round-robins among queues in proportion to some weight parameter

Example with contending hosts

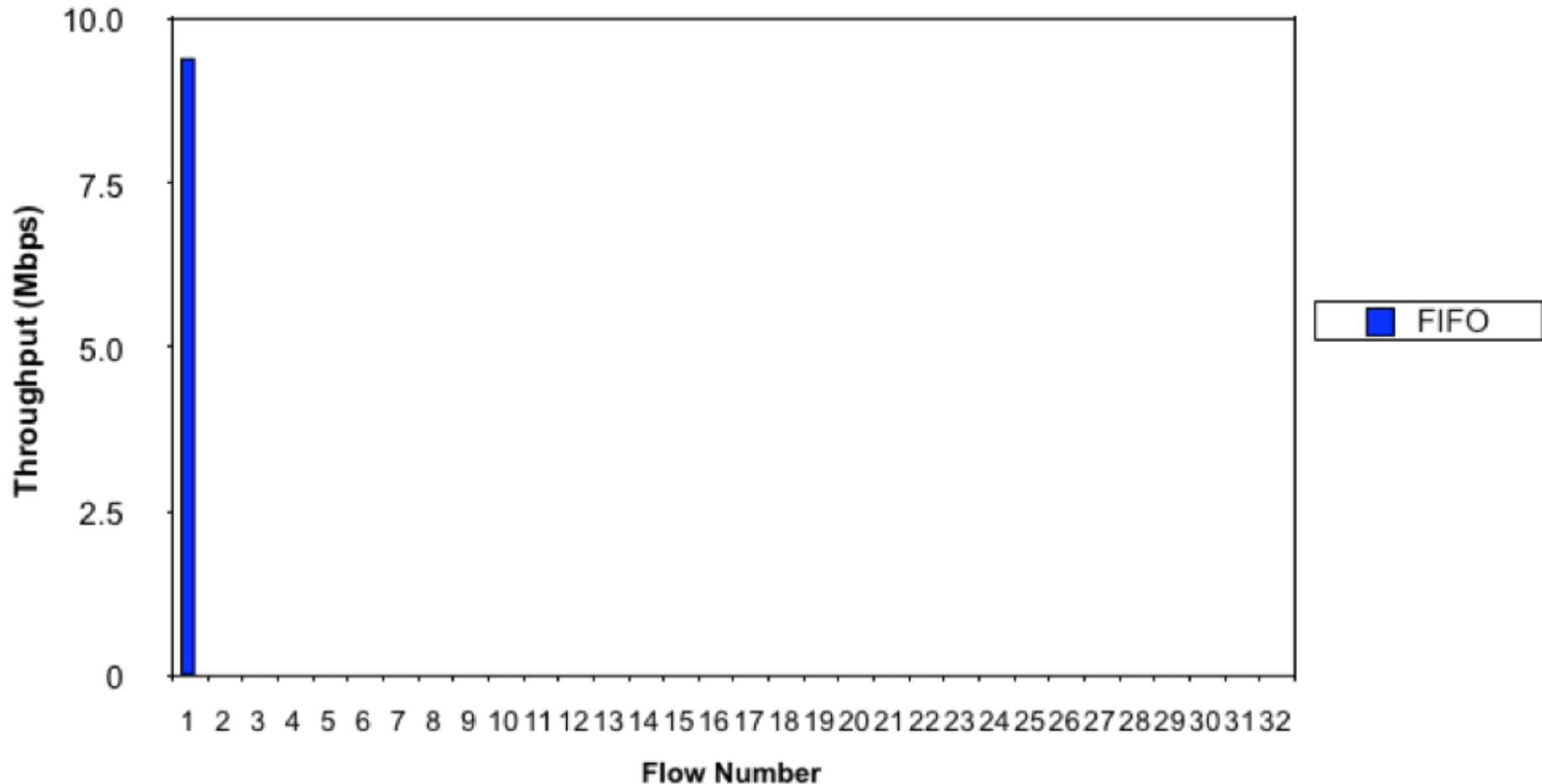
1 UDP (10 Mbps) and 31 TCPs sharing 10 Mbps
(Note: *TCP will throttle due to congestion control*)

Will discuss this later

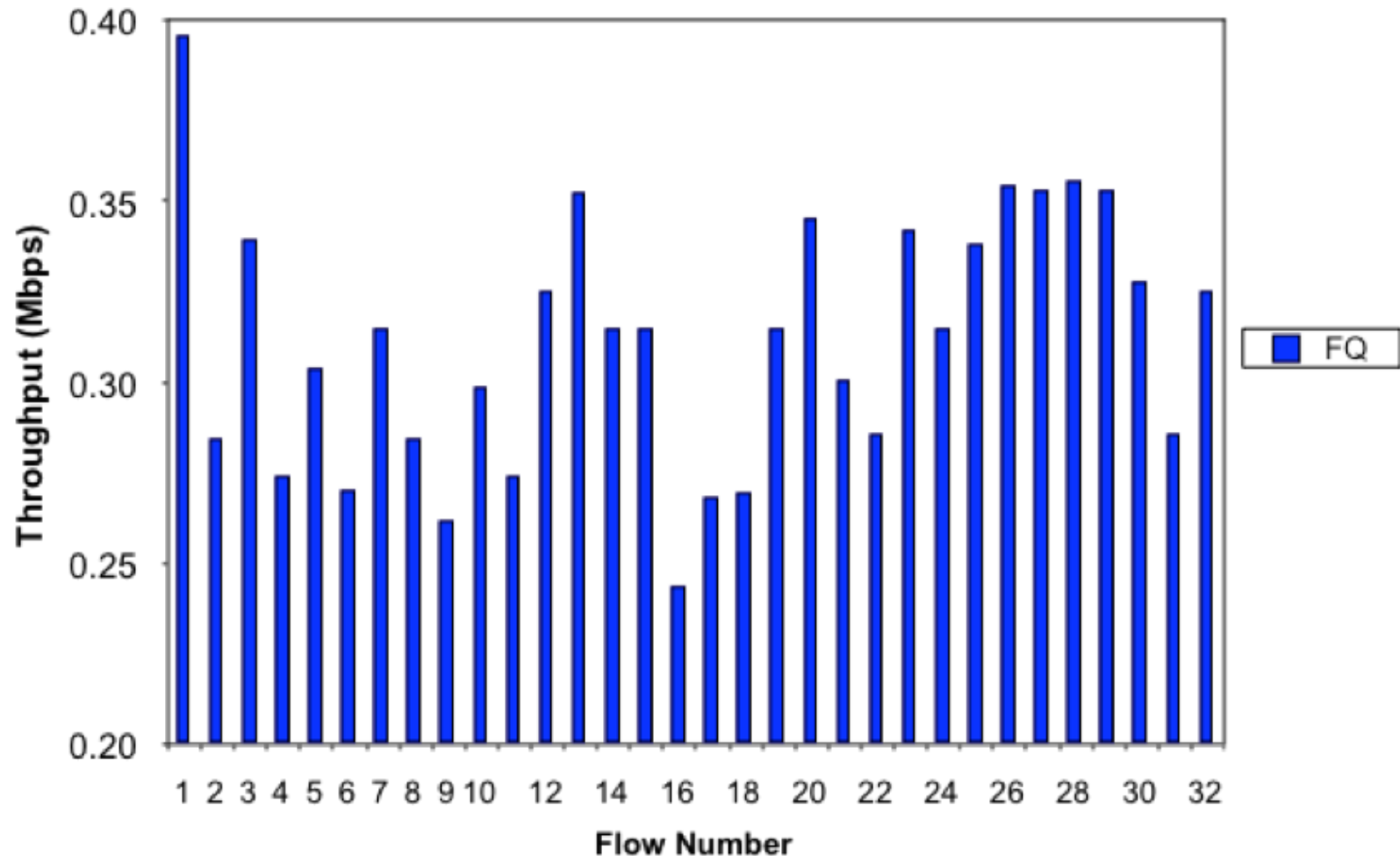




UDP vs. TCP w/FIFO

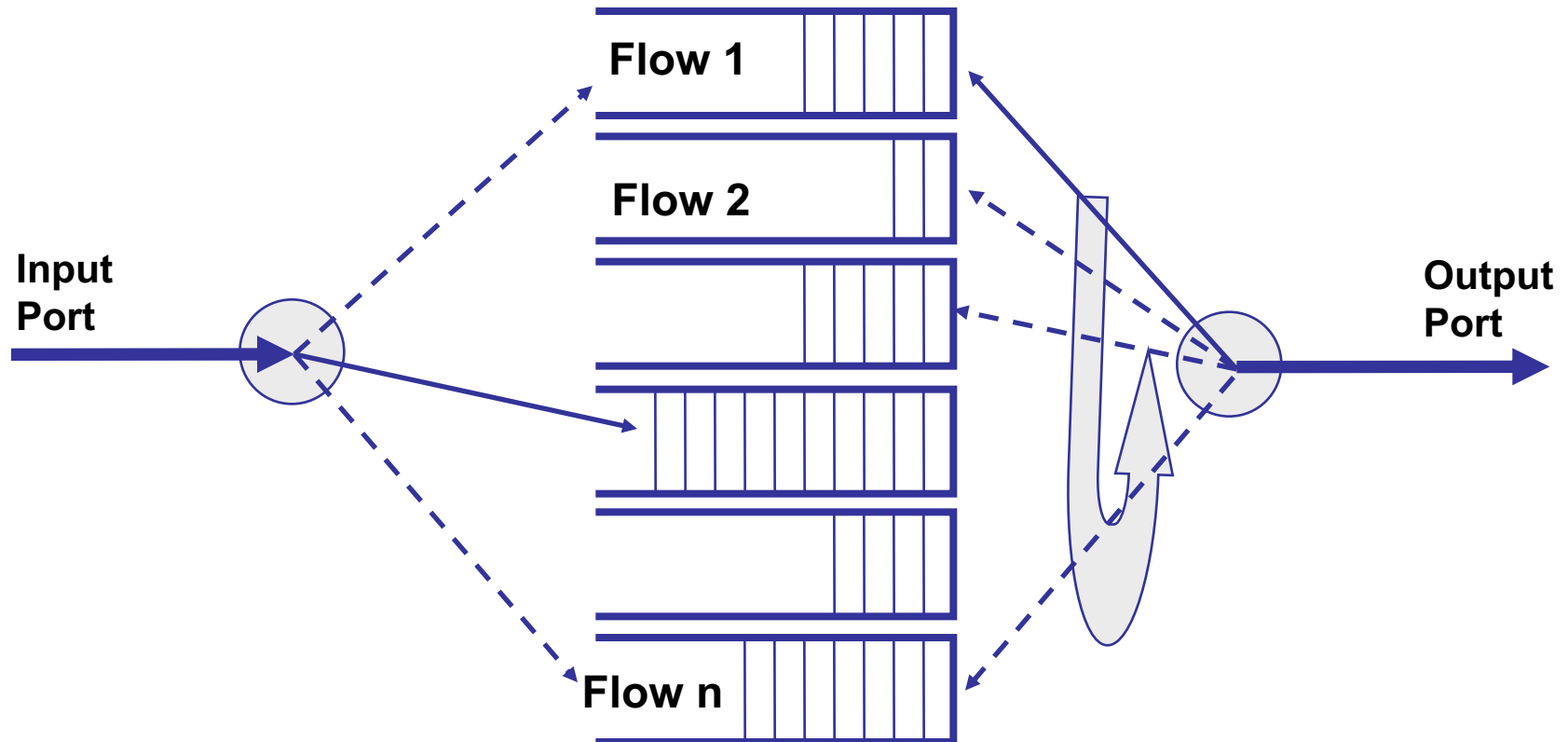


TCP vs. UDP w/Fair Queuing





(Weighted) Fair Queuing





Fair Queuing

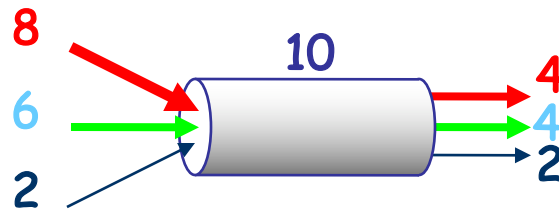
- Maintain a queue for each flow
 - ◆ What is a flow?
 - » We approximate this with the **5 tuple**
 - » (IP src, IP dst, port src, port dst, protocol)
- Implements **max-min fairness**: each flow receives $\min(r_i, f)$, where
 - ◆ r_i – flow arrival rate
 - ◆ f – link fair rate (see next slide)
- **Weighted Fair Queuing** (WFQ) – associate a weight with each flow to divvy bandwidth up non-equally



Fair Rate Computation

- If link congested, compute f such that

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



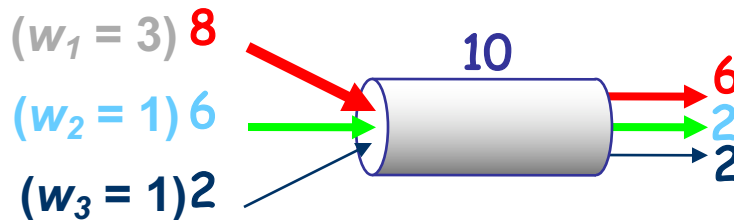
$f = 4$:
 $\min(8, 4) = 4$
 $\min(6, 4) = 4$
 $\min(2, 4) = 2$



Weighted Fair Queuing

- Associate a weight w_i (*priority*) with each flow I
- If link congested, compute f such that

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



$f = 2$:

$$\begin{aligned} \min(8, 2 \times 3) &= 6 \\ \min(6, 2 \times 1) &= 2 \\ \min(2, 2 \times 1) &= 2 \end{aligned}$$

Flow i is guaranteed to be allocated a rate $\geq w_i \cdot C / (\sum_k w_k)$



If $\sum_k w_k \leq C$, flow i is guaranteed to be allocated a rate $\geq w_i$

WFQ Implementation: Fluid Flow

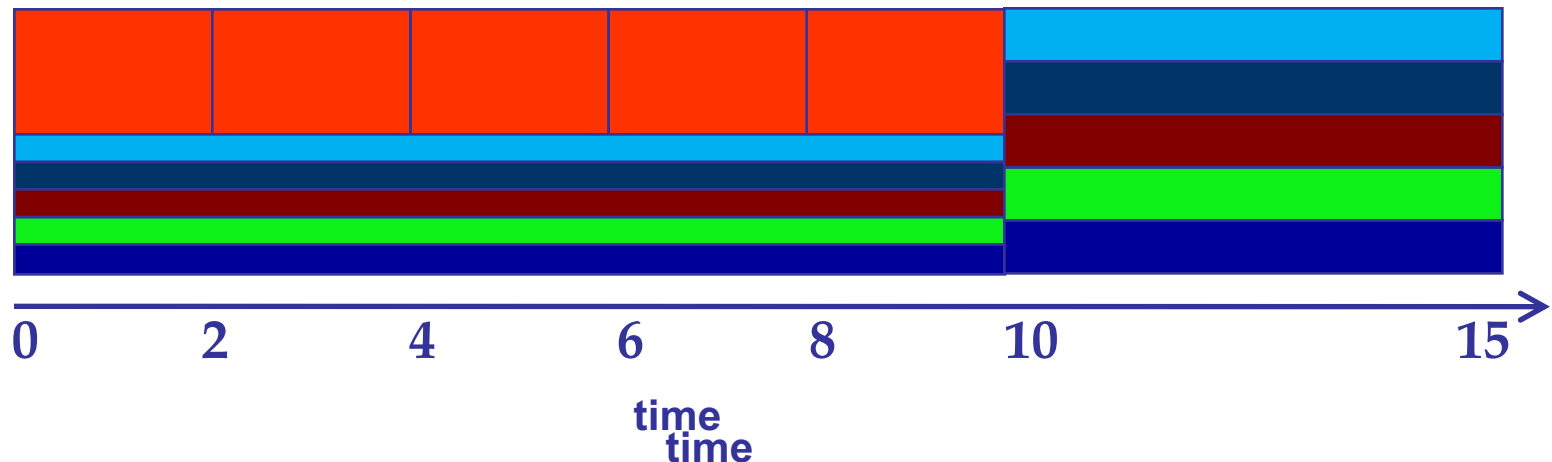
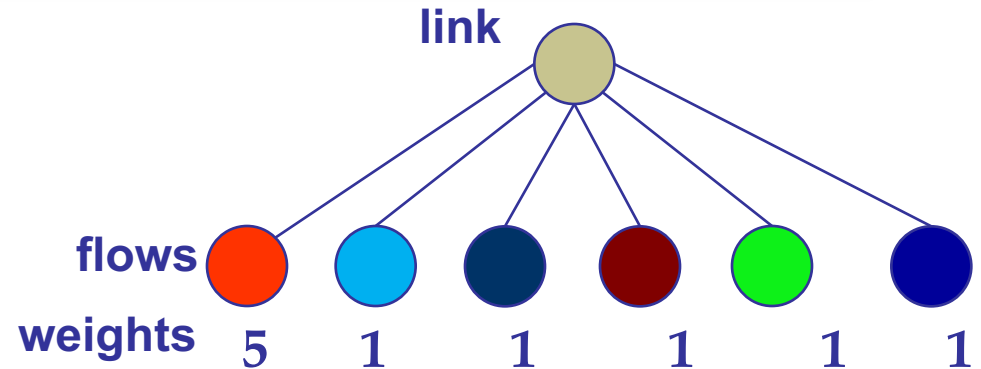


- It's tricky to implement WFQ when there are different packet sizes in different queues.
 - ◆ Can only serve one packet at a time.
- Simplification: Flows can be served one bit at a time
- WFQ can be implemented using bit-by-bit weighted round robin
 - ◆ During each round from each flow that has data to send, send a number of bits equal to the flow's weight



Fluid Flow Example

- **Red flow** has packets backlogged between time 0 and 10
- Other flows have packets continuously backlogged
- All packets have the same size



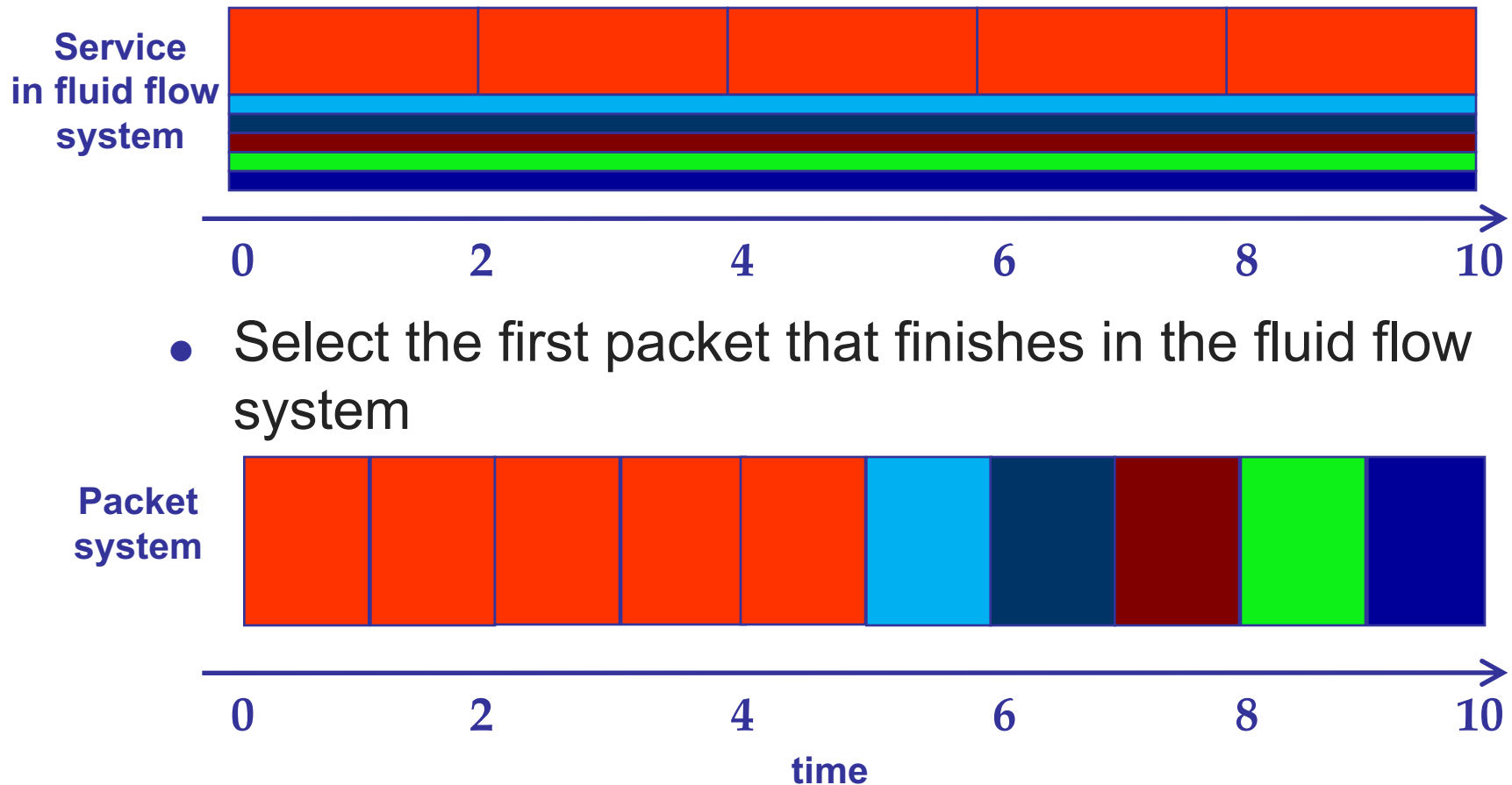
Packet-Based Implementation



- Packet (real) system can't do bit-by-bit splitting: packet transmission cannot be preempted. Why?
- Issue: flows can get more bandwidth by sending bigger packets
- Solution: serve packets in the order in which they would have finished being transmitted in the fluid flow system



Packet-Based Example

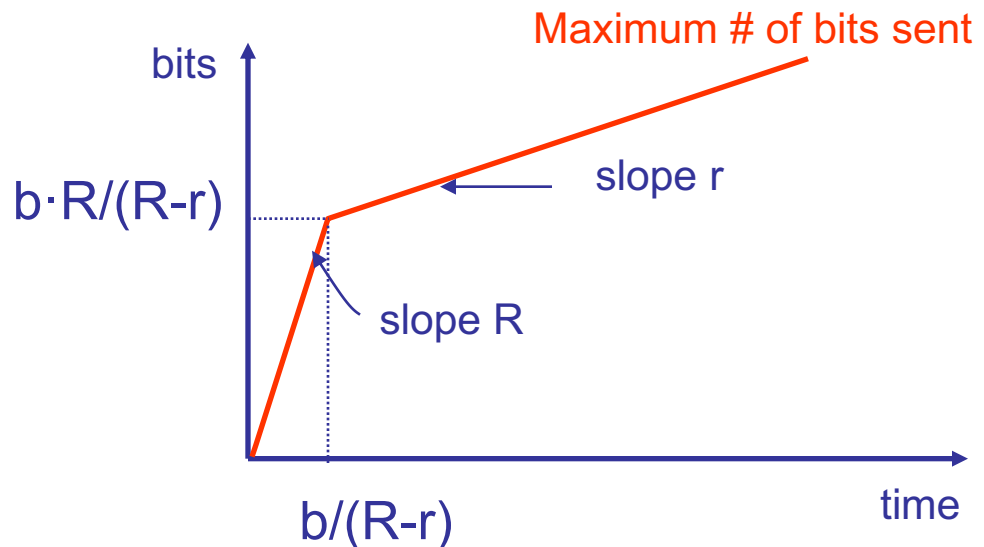
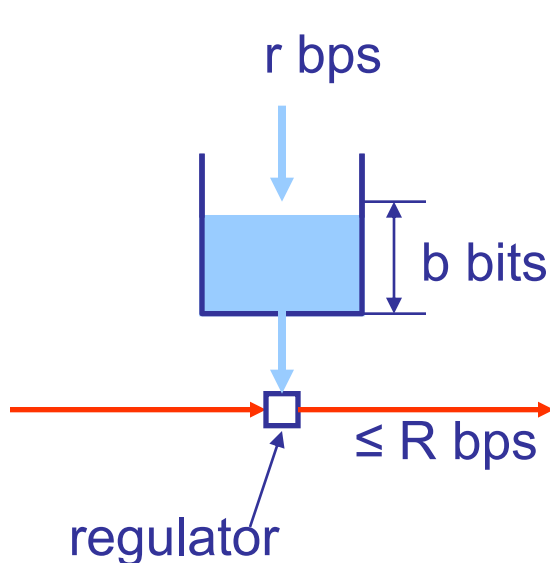


Token Bucket

(similar to WFQ, but doesn't allocate spare capacity)



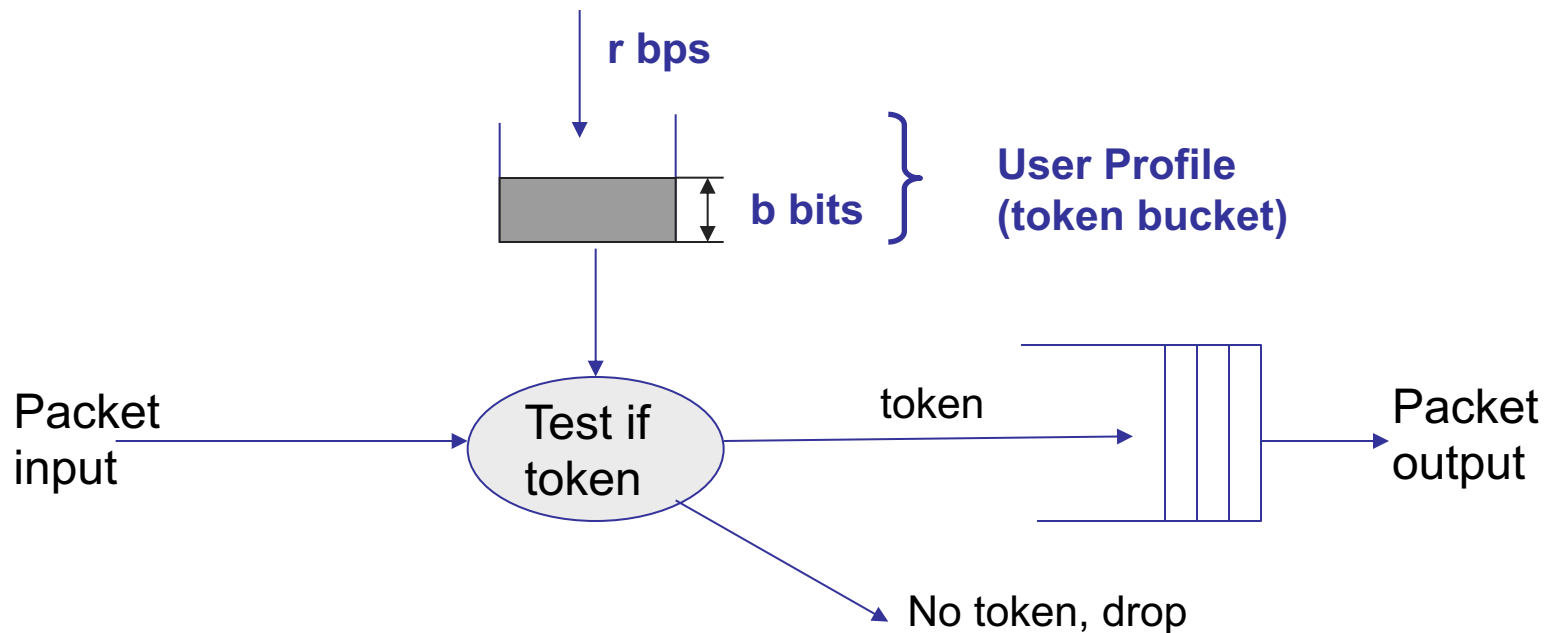
- Parameters
 - ◆ r – average rate, i.e., rate at which tokens fill the bucket
 - ◆ b – bucket depth (limits size of burst)
 - ◆ R – maximum link capacity or peak rate (optional parameter)
- A bit can be transmitted only when a token is available





Traffic Policing

- Drop packets that don't meet **flow policy**
- Output limited to average of r bps and bursts of b





Traffic Shaping

- Shape (queue) packets according to flow policy
- Output limited to average of r bps and bursts of b

