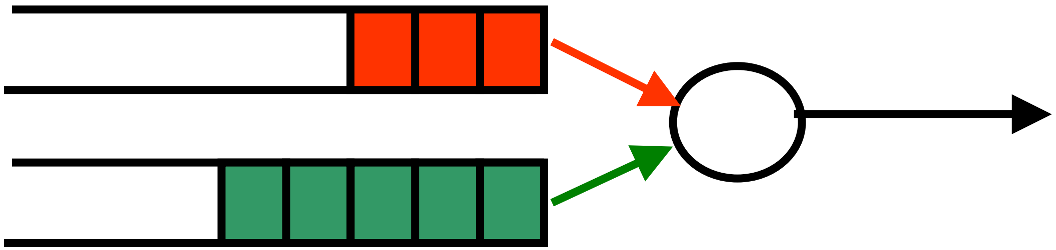


Queuing and Queue Management

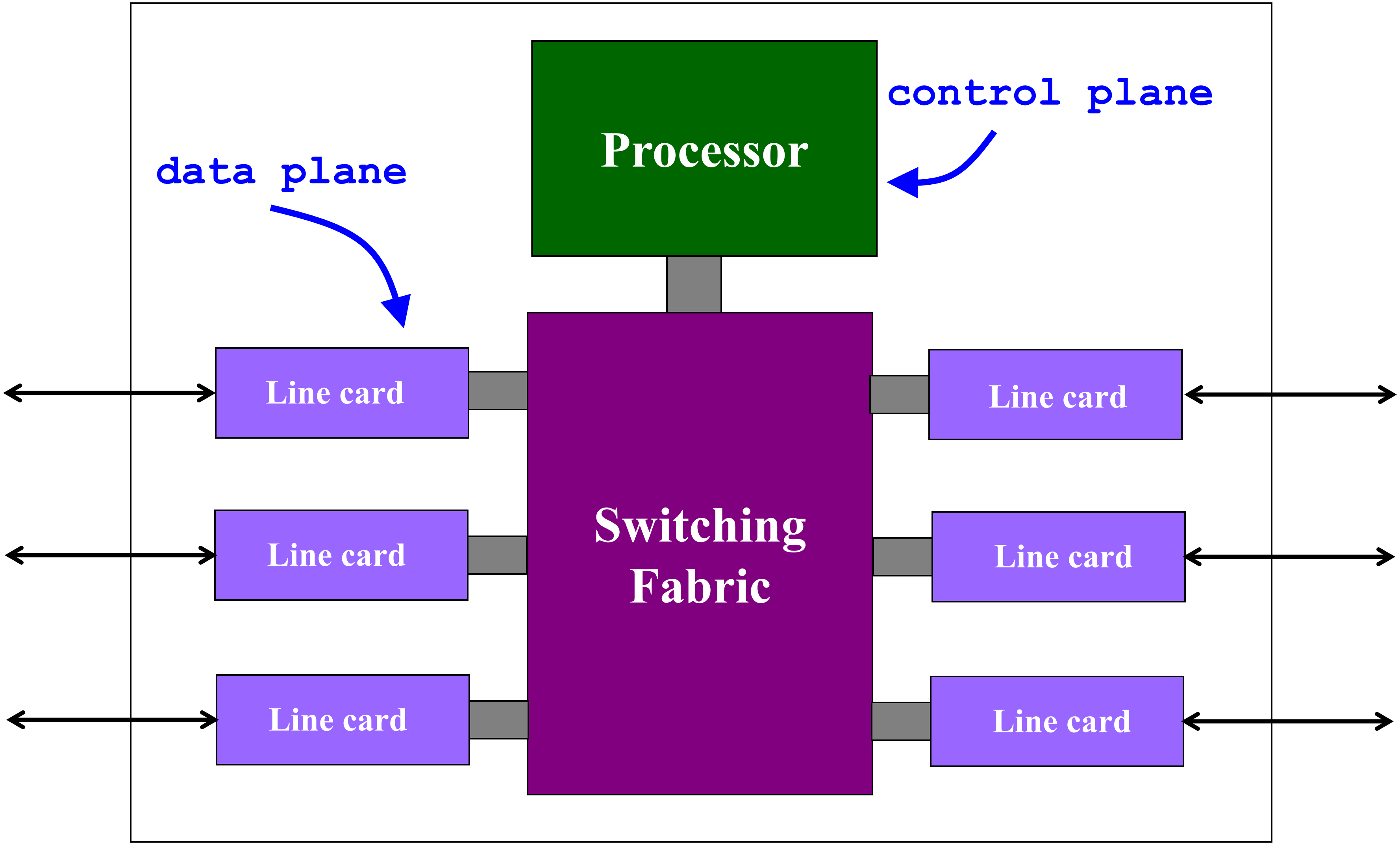
Note: The slides are adapted from the materials from Prof. Richard Han at CU Boulder and Profs. Jennifer Rexford and Mike Freedman at Princeton University, and the networking book (Computer Networking: A Top Down Approach) from Kurose and Ross.

Goals of Today's Lecture

- Router Queuing Models
  – Limitations of FIFO and Drop Tail
- Scheduling Policies
  – Fair Queuing
- Drop policies
  – Random Early Detection (of congestion)
  – Explicit Congestion Notification (from routers)
- Some additional TCP mechanisms

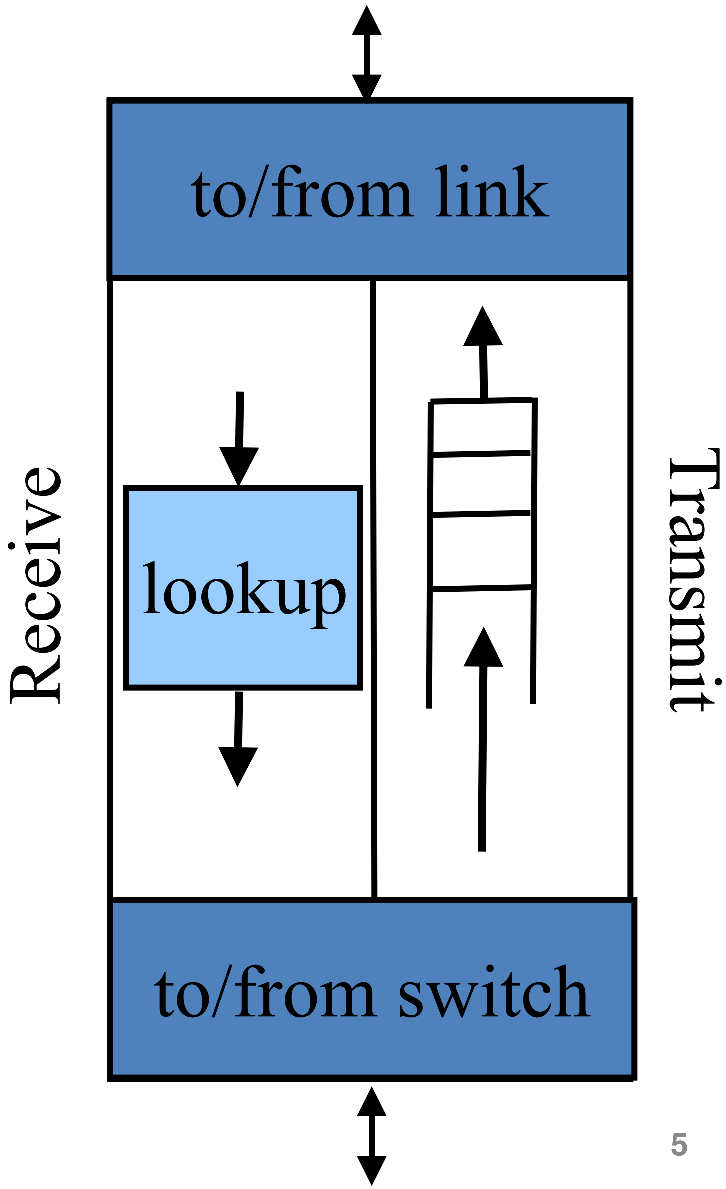
Packet Queues

Router Data and Control Planes

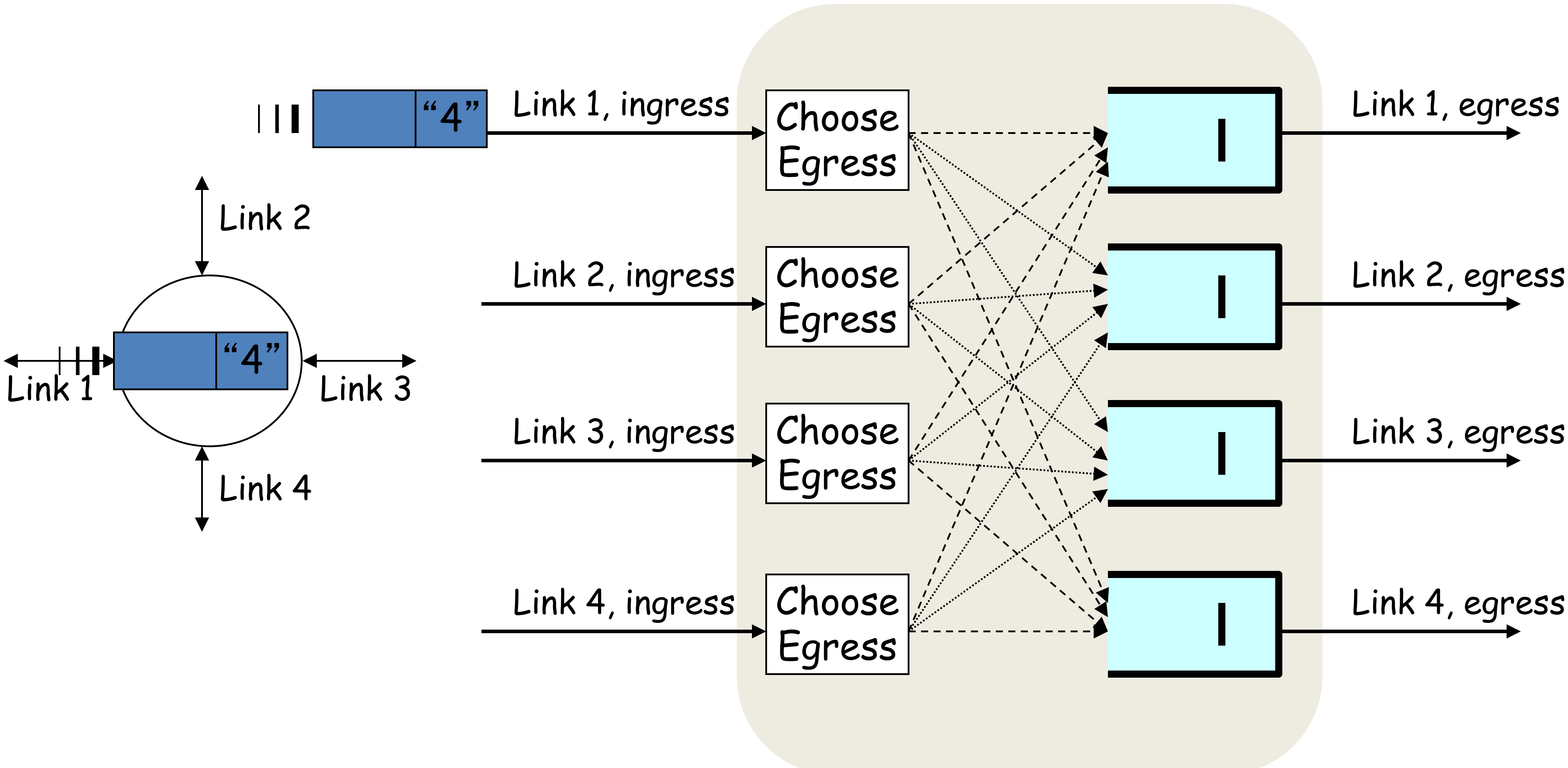


Line Cards (Interface Cards, Adaptors)

- Interfacing
  - Physical link
  - Switching fabric
- Packet handling
  - Packet forwarding
  - Decrement time-to-live
  - Buffer management
  - Link scheduling
  - Packet filtering
  - Rate limiting
  - Packet marking
  - Measurement



Packet Switching and Forwarding

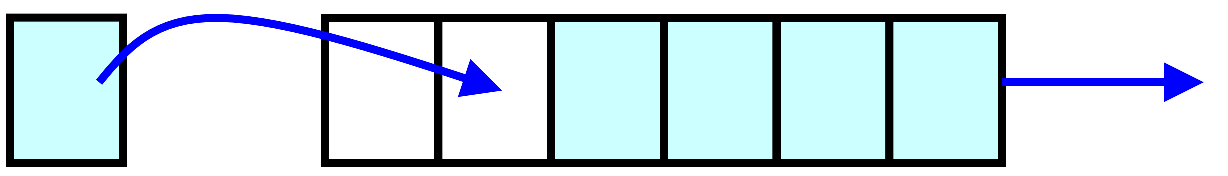


Router Design Issues

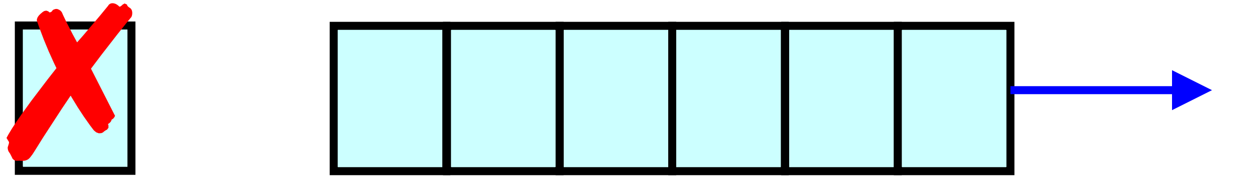
- Scheduling discipline
  - Which packet to send?
  - Some notion of fairness? Priority?
- Drop policy
  - When should you discard a packet?
  - Which packet to discard?
- Need to balance throughput and delay
  - Huge buffers minimize drops, but add to queuing delay (thus higher RTT, longer slow start, ...)

FIFO Scheduling and Drop-Tail

- Access to the bandwidth: first-in first-out queue
  – Packets only differentiated when they arrive

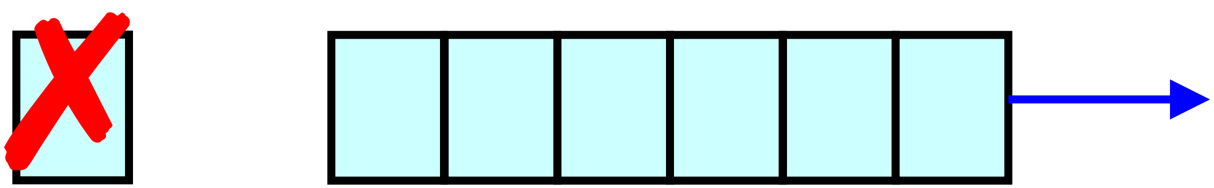


- Access to the buffer space: drop-tail queuing
  – If the queue is full, drop the incoming packet



Problems with tail drop

- Under stable conditions, queue almost always full
  - Leads to high latency for all traffic
- Possibly unfair for flows with small windows
  - Larger flows may fast retransmit (detecting loss through Trip Dup ACKs), small flows may have to wait for timeout
- Window synchronization
  - More on this later…



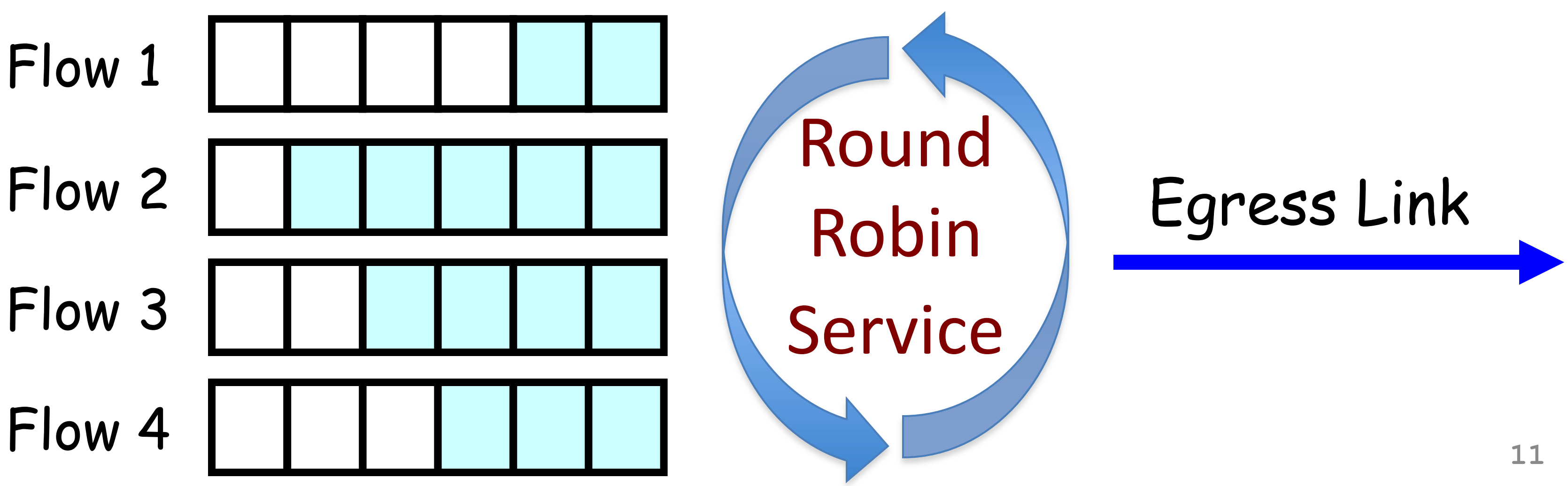
Scheduling Policies

(Weighted) Fair Queuing

(and Class-based Quality of Service)

Fair Queuing (FQ)

- Maintains separate queue per flow
- Ensures no flow consumes more than its 1/n share
  - Variation: weighted fair queuing (WFQ)
- If all packets were same length, would be easy
- If ***non-work-conserving*** (resources can go idle), also would be easy, yet lower utilization



Fair Queuing Basics

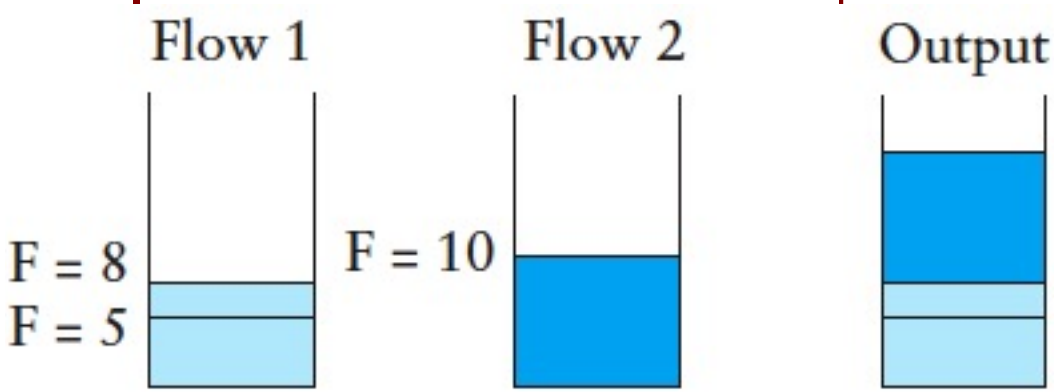
- Track how much time each flow has used link
  - Compute time used if it transmits next packet
- Send packet from flow that will have lowest use if it transmits
  - Why not flow with smallest use so far?
  - Because next packet may be huge!

FQ Algorithm

- Imagine clock tick per bit, then tx time ~ length

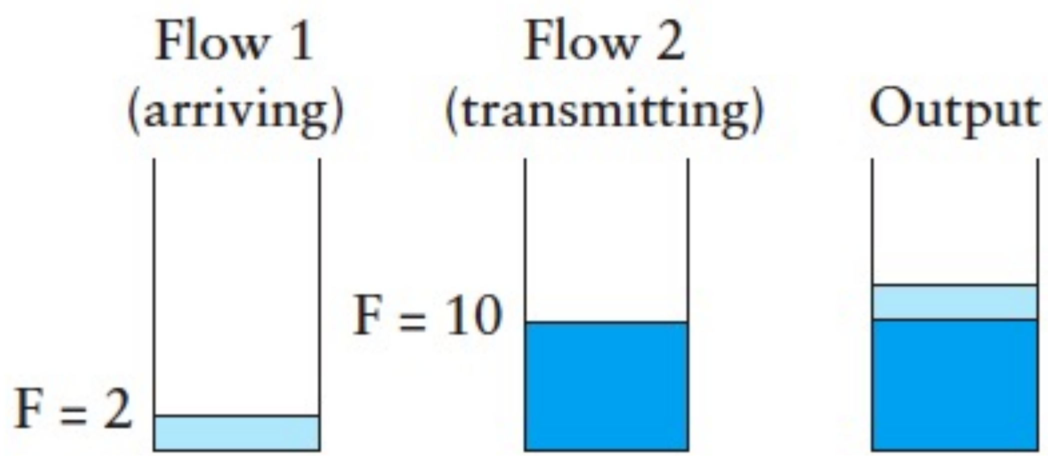
Finish time F_i = max (F_{i-1}, Arrive time A_i) + Length P_i

- Calculate estimated F_i for all queued packets
- Transmit packet with lowest F_i next



FQ Algorithm (2)

- Problem: Can't preempt current tx packet
- Result: Inactive flows ($A_i > F_{i-1}$) are penalized
  - Standard algorithm considers no history
  - Each flow gets fair share only when packets queued



FQ Algorithm (3)

- Approach: give more *promptness* to flows utilizing less bandwidth historically
- Bid $B_i = \max(F_{i-1}, A_i - \delta) + P_i$
  - Intuition: with larger δ, scheduling decisions calculated by last tx time F_{i-1} more frequently, thus preferring slower flows
- FQ achieves max-min fairness
  - First priority: maximize the *minimum* rate of any active flows
  - Second priority: maximize the second min rate, etc.

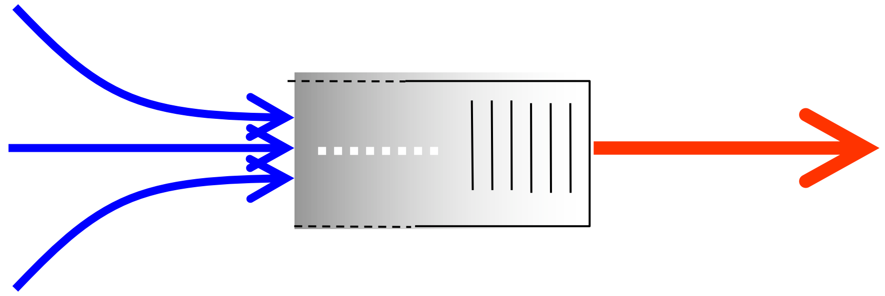
Uses of (W)FQ

- Scalability
  - # queues must be equal to # flows
  - But, can be used on edge routers, low speed links, or shared end hosts
- (W)FQ can be for classes of traffic, not just flows
  - Use IP TOS bits to mark "importance"
  - Part of "Differentiated Services" architecture for "Quality-of-Service" (QoS)

Early Detection of Congestion

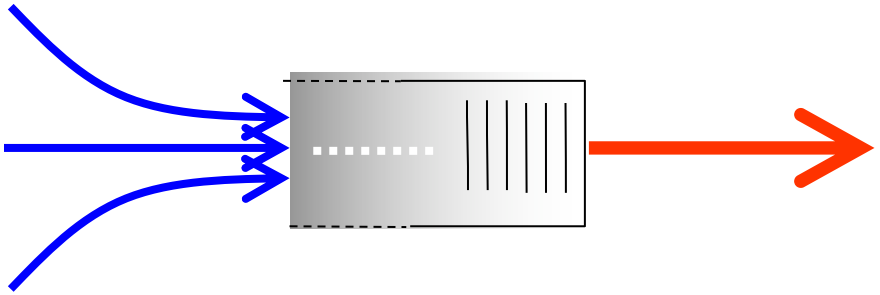
Bursty Loss From Drop-Tail Queuing

- TCP depends on packet loss
  - Packet loss is indication of congestion
  - And TCP *drives* network into loss by additive rate increase
- Drop-tail queuing leads to *bursty* loss
  - Congested link: many packets encounter full queue
  - Synchronization: many connections lose packets at once



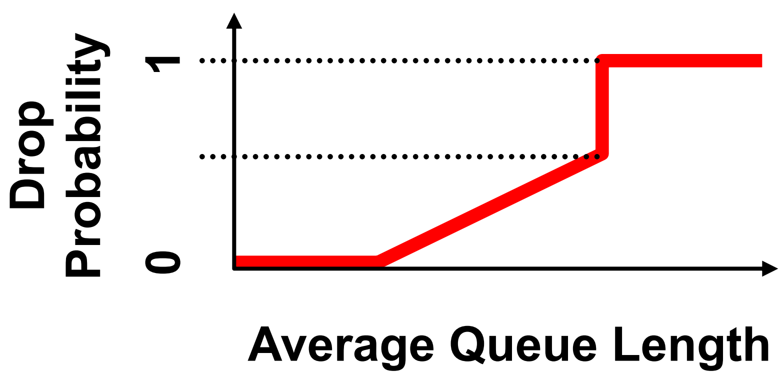
Slow Feedback from Drop Tail

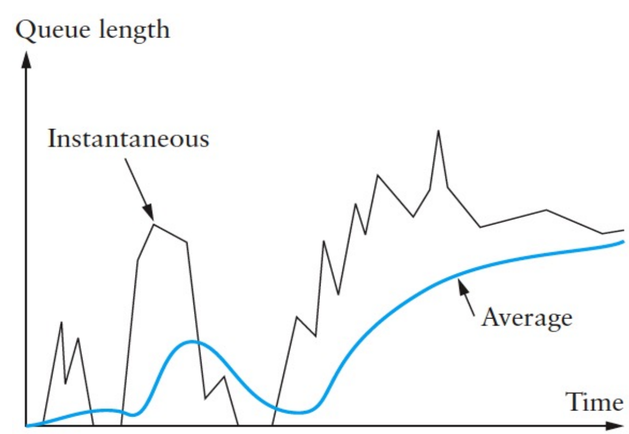
- Feedback comes when buffer is completely full
  - … even though the buffer has been filling for a while
- Plus, the filling buffer is increasing RTT
  - … making detection even slower
- Better to give early feedback
  - Informs sender to slow down before it's too late!



Random Early Detection (RED)

- Basic idea of RED
  - Router notices that queue is getting backlogged
  - … and randomly drops packets to signal congestion
- Packet drop probability
  - Drop probability increases as queue length increases
  - Else, set drop probability as function of avg queue length and time since last drop





Properties of RED

- Drops packets before queue is full
  - In the hope of reducing the rates of some flows
- Tolerant of burstiness in the traffic
  - By basing the decisions on average queue length
- Which of the following are true?

  (A)Drops packet in proportion to each flow's rate

  (B)High-rate flows selected more often

  (C)Helps desynchronize the TCP senders

  (D)All of the above

Problems With RED

- Hard to get tunable parameters just right
  - How early to start dropping packets?
  - What slope for increase in drop probability?
  - What time scale for averaging queue length?
- RED has mixed adoption in practice
  - If parameters aren't set right, RED doesn't help
  - Hard to know how to set the parameters
- Many other variations in research community
  - Names like "Blue" (self-tuning), "FRED"…

From Loss to Notification

Feedback: From loss to notification

- Early dropping of packets
  - Good: gives early feedback
  - Bad: has to drop the packet to give the feedback
- Explicit Congestion Notification
  - Router marks the packet with an ECN bit
  - Sending host interprets as a sign of congestion

Explicit Congestion Notification

- Needs support by router, sender, AND receiver
  - End-hosts check ECN-capable during TCP handshake

- ECN protocol (repurposes 4 header bits)

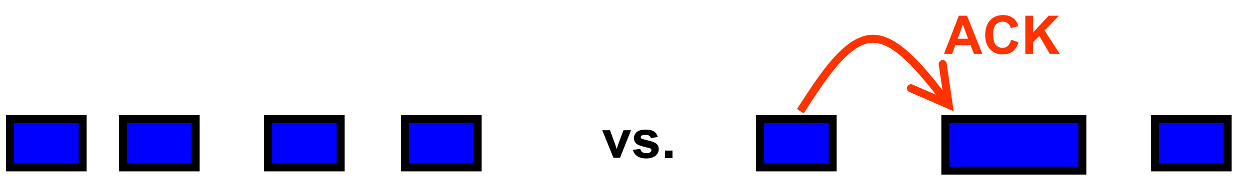
1. Sender marks "ECN-capable" when sending
2. If router sees "ECN-capable" and congested, marks packet as "ECN congestion experienced"
3. If receiver sees "congestion experienced", marks "ECN echo" flag in responses until congestion ACK'd
4. If sender sees "ECN echo", reduces *cwnd* and marks "congestion window reduced" flag in next packet

Other TCP Mechanisms

Nagle's Algorithm and Delayed ACK

Nagle's Algorithm

- Wait if the amount of data is small
  - Smaller than Maximum Segment Size (MSS)
- And some other packet is already in flight
  - I.e., still awaiting the ACKs for previous packets
- That is, send at most one small packet per RTT
  - … by waiting until all outstanding ACKs have arrived



- Influence on performance
  - Interactive applications: enables batching of bytes
  - Bulk transfer: transmits in MSS-sized packets anyway

Nagle’s Algorithm

- Wait if the amount of data is small
  - Smaller than Maximum Segment Size (MSS)
- And some other packet is already in flight

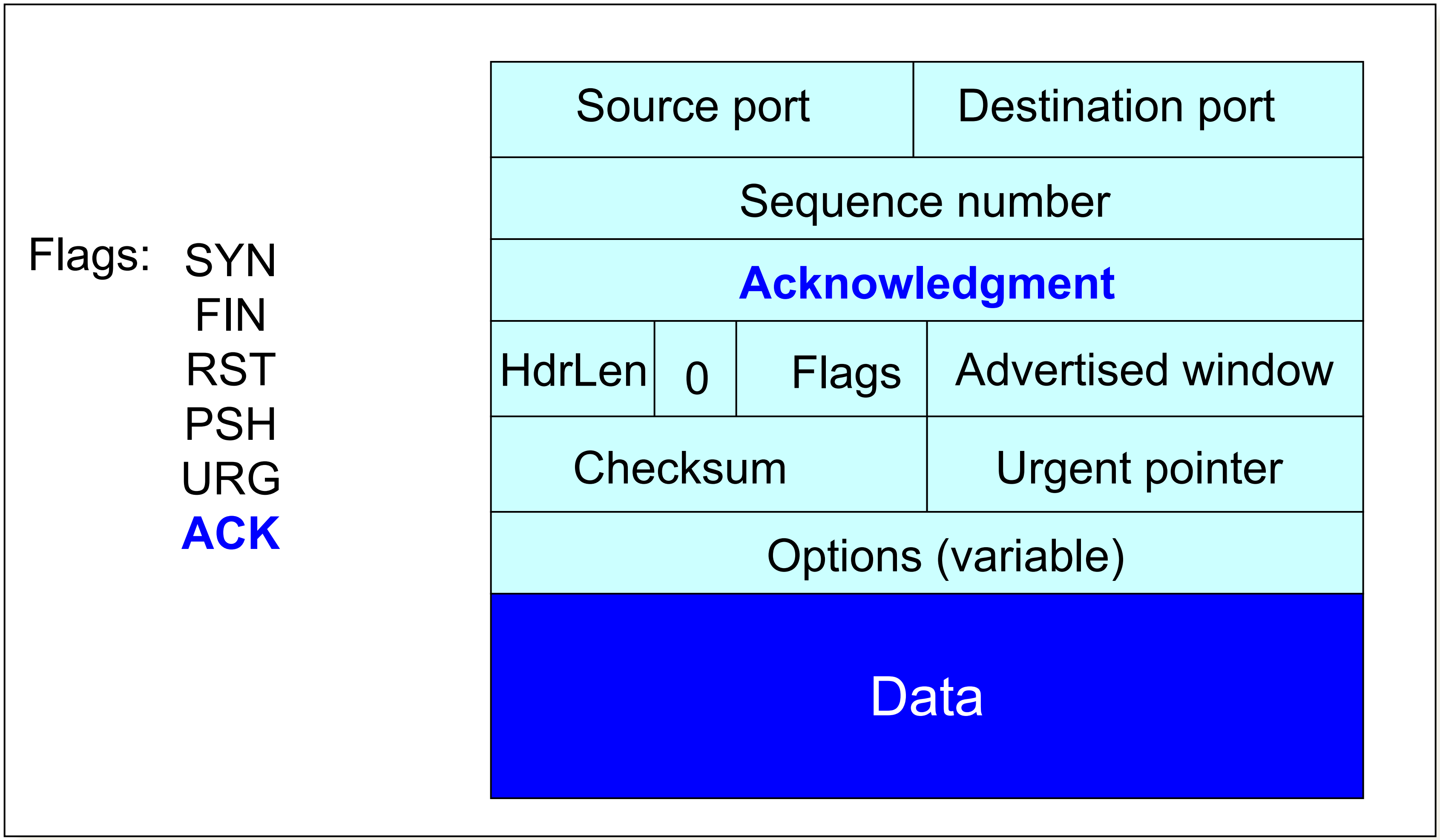
Turning Nagle Off

```
void
tcp_nodelay (int s)
{
    int n = 1;
    if (setsockopt (s, IPPROTO_TCP, TCP_NODELAY,
                    (char *) &n, sizeof (n)) < 0)
        warn ("TCP_NODELAY: %m\n");
}
```

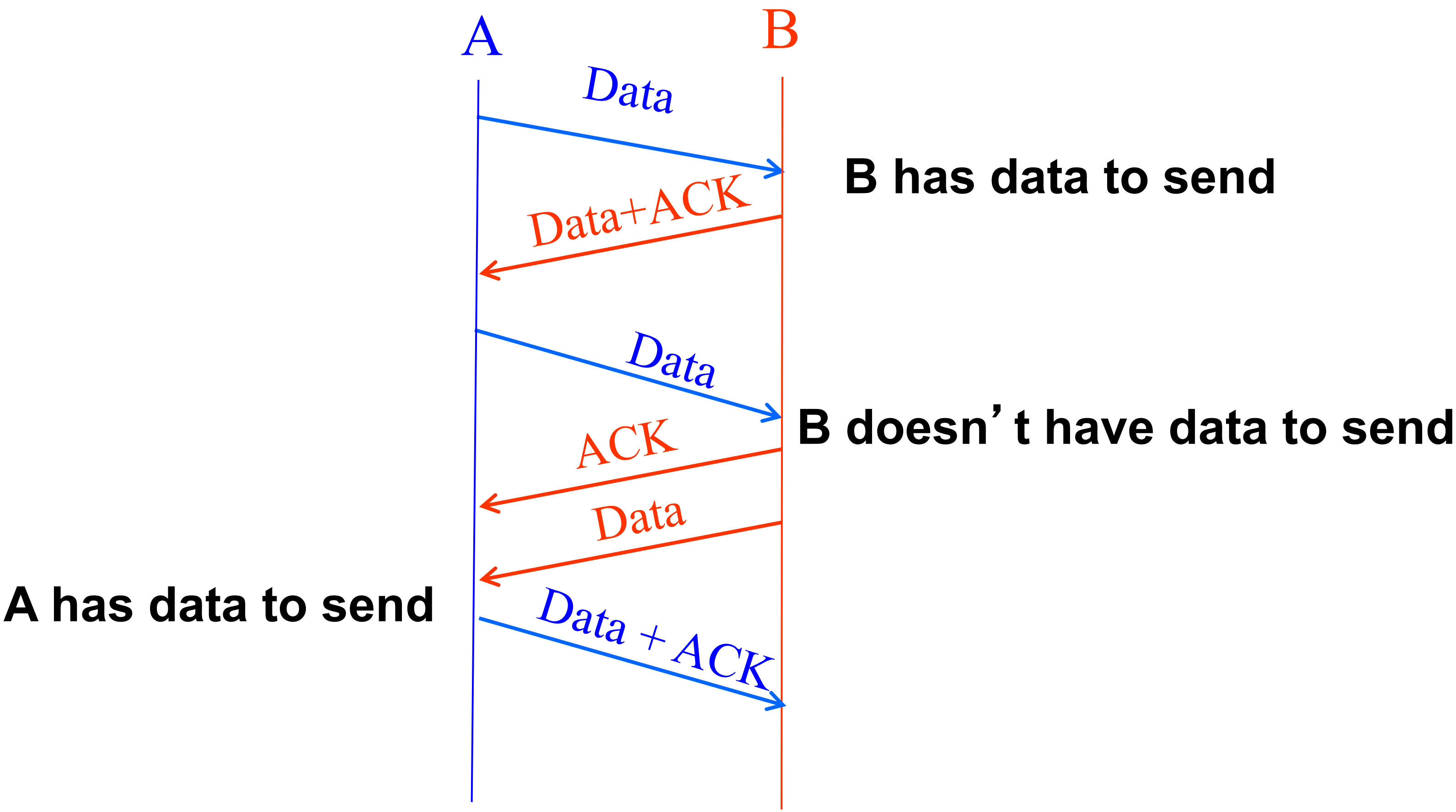
Motivation for Delayed ACK

- TCP traffic is often bidirectional
  - Data traveling in both directions
  - ACKs traveling in both directions
- ACK packets have high overhead
  - 40 bytes for the IP header and TCP header
  - … and zero data traffic
- Piggybacking is appealing
  - Host B can send an ACK to host A
  - … as part of a data packet from B to A

TCP Header Allows Piggybacking

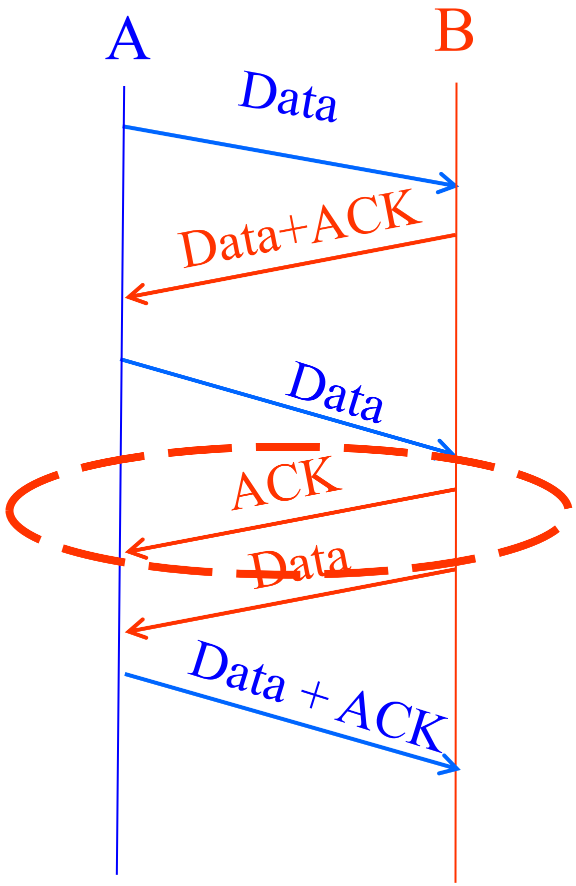


Example of Piggybacking



Increasing Likelihood of Piggybacking

- Example: ssh or even HTTP
  - Host A types command
  - Host B receives and executes the command
  - … and then data are generated
  - Would be nice if B could send the ACK with the new data
- Increase piggybacking
  - TCP allows the receiver to *wait* to send the ACK
  - … in the hope that the host will have data to send



Delayed ACK

- Delay sending an ACK
  - Upon receiving a packet, the host B sets a timer
    - Typically, 200 msec or 500 msec
  - If B's application generates data, go ahead and send
    - And piggyback the ACK bit
  - If the timer expires, send a (non-piggybacked) ACK

- Limiting the wait
  - Timer of 200 msec or 500 msec
  - ACK every other full-sized packet