# Congestion Control

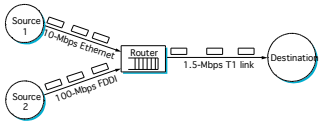
Outline

- Resource Allocation
- Queuing
- TCP Congestion Control



## Issues

- Two sides of the same coin
  - pre-allocate resources so at to avoid congestion
  - control congestion if (and when) is occurs

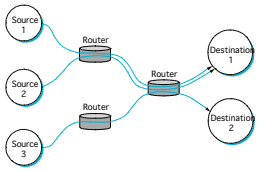



- Two points of implementation
  - hosts at the edges of the network (transport protocol)
  - routers inside the network (queuing discipline)
- Underlying service model
  - best-effort (assume for now)
  - multiple *qualities of service* (later)



## Framework

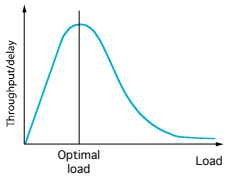
- Connectionless flows
  - sequence of packets sent between source/destination pair
  - maintain *soft state* at the routers




- Taxonomy
  - router-centric versus host-centric
  - reservation-based versus feedback-based
  - window-based versus rate-based

## Evaluation

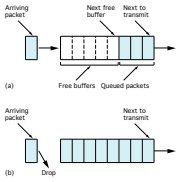
- Fairness
- Power (ratio of throughput to delay)

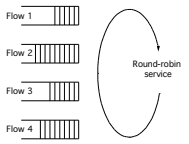





## Queuing Disciplines

- First-In-First-Out (FIFO)
  - does not discriminate between traffic sources
- Fair Queuing (FQ)
  - explicitly segregates traffic based on flows
  - ensures no flow captures more than its share of capacity
  - variation: weighted fair queuing (WFQ)
- Problem?



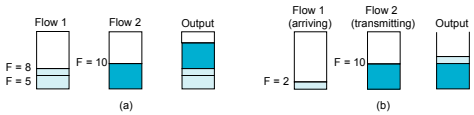



## FQ Algorithm

- Suppose clock ticks each time a bit is transmitted
- Let $P_i$ denote the length of packet $i$
- Let $S_i$ denote the time when start to transmit packet $i$
- Let $F_i$ denote the time when finish transmitting packet $i$
- $F_i = S_i + P_i$
- When does router start transmitting packet $i$?
  - if before router finished packet $i$ - 1 from this flow, then immediately after last bit of packet $i$ - 1 ($F_{i-1}$)
  - if no current packets for this flow, then start transmitting when arrives (call this $A_i$)
- Thus: $F_i = MAX\,(F_{i-1}, A_i) + P_i$

## FQ Algorithm (cont)

- For multiple flows
  - calculate $F_i$ for each packet that arrives on each flow
  - treat all $F_i$'s as timestamps
  - next packet to transmit is one with lowest timestamp
- Not perfect: can't preempt current packet
- Example






## TCP Congestion Control

- Idea
  - assumes best-effort network (FIFO or FQ routers) each source determines network capacity for itself
  - uses implicit feedback
  - ACKs pace transmission (*self-clocking*)
- Challenge
  - determining the available capacity in the first place
  - adjusting to changes in the available capacity



## Additive Increase/Multiplicative Decrease

- Objective: adjust to changes in the available capacity
- New state variable per connection: `CongestionWindow`
  - limits how much data source has in transit

```
MaxWin = MIN(CongestionWindow,
                    AdvertisedWindow)
EffWin = MaxWin - (LastByteSent -
                   LastByteAcked)
```

- Idea:
  - increase `CongestionWindow` when congestion goes down
  - decrease `CongestionWindow` when congestion goes up

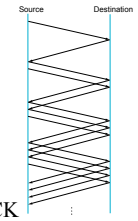
## AIMD (cont)

- Question: how does the source determine whether or not the network is congested?
- Answer: a timeout occurs
  - timeout signals that a packet was lost
  - packets are seldom lost due to transmission error
  - lost packet implies congestion



## AIMD (cont)

- Algorithm
  - increment **`CongestionWindow`** by one packet per RTT (*linear increase*)
  - divide **`CongestionWindow`** by two whenever a timeout occurs (*multiplicative decrease*)



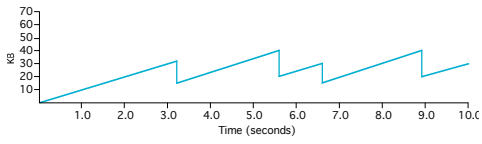

- In practice: increment a little for each ACK

```
Increment = MSS * (MSS/CongestionWindow)
CongestionWindow += Increment
```



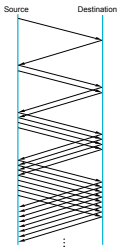
## AIMD (cont)

- Trace: sawtooth behavior



## Slow Start

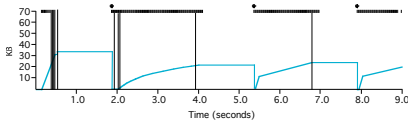
- Objective: determine the available capacity in the beginning
- Idea:
  - begin with `CongestionWindow` = 1 packet
  - double `CongestionWindow` each RTT (increment by 1 packet for each ACK)






## Slow Start (cont)

- Exponential growth, but slower than all at once
- Used…
  - when first starting connection
  - when connection goes dead waiting for timeout
- Trace

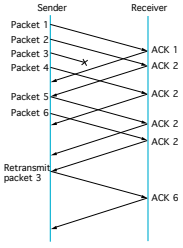



- Problem: lose up to half a `CongestionWindow`'s worth of data

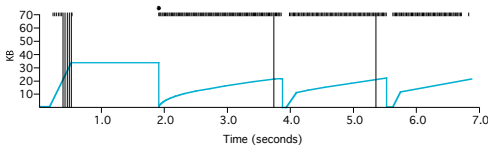


## Fast Retransmit and Fast Recovery

- Problem: coarse-grain TCP timeouts lead to idle periods
- Fast retransmit: use duplicate ACKs to trigger retransmission



## Results




- Fast recovery
  - skip the slow start phase
  - go directly to half the last successful `CongestionWindow (ssthresh)`