What We Know

We know:
- How to process packets in a switch
- How to route packets in the network
- How to send packets reliably

We don’t know:
- How fast to send
Implications

- Send too slow: link is not fully utilized
  - Wastes time

- Send too fast: link is fully utilized but....
  - Queue builds up in router buffer (delay)
  - Overflow buffers in routers
  - Overflow buffers in receiving host (ignore)

- Why are buffer overflows a problem?
  - Packet drops (mine and others)
  - Interesting history....(Van Jacobson rides to the rescue)

Abstract View

• Ignore internal structure of router and model it as having a single queue for a particular input-output pair
Three Congestion Control Problems

- Adjusting to bottleneck bandwidth
- Adjusting to variations in bandwidth
- Sharing bandwidth between flows

Single Flow, Fixed Bandwidth

- Adjust rate to match bottleneck bandwidth
  - Without any \textit{a priori} knowledge
  - Could be gigabit link, could be a modem
Single Flow, Varying Bandwidth

- Adjust rate to match instantaneous bandwidth
  - Assuming you have rough idea of bandwidth

Multiple Flows

Two Issues:
- Adjust total sending rate to match bandwidth
- Allocation of bandwidth between flows
Congestion control is a resource allocation problem involving many flows, many links, and complicated global dynamics.

What’s Really Happening?
View from a Single Flow

- **Knee** – point after which
  - Throughput increases very slow
  - Delay increases fast
- **Cliff** – point after which
  - Throughput starts to decrease very fast to zero (congestion collapse)
  - Delay approaches infinity
General Approaches

- Send without care
  - Many packet drops

- Reservations
  - Pre-arrange bandwidth allocations
  - Requires negotiation before sending packets
  - Low utilization

- Pricing
  - Don’t drop packets for the high-bidders
  - Requires payment model

General Approaches (cont’d)

- Dynamic Adjustment
  - Probe network to test level of congestion
  - Speed up when no congestion
  - Slow down when congestion
  - Suboptimal, messy dynamics, simple to implement

- All three techniques have their place
  - But for generic Internet usage, dynamic adjustment is the most appropriate
  - Due to pricing structure, traffic characteristics, and good citizenship
TCP Congestion Control

- TCP connection has window
  - Controls number of unacknowledged packets

- Sending rate: ~Window/RTT

- Vary window size to control sending rate

Sizing the Windows

- cwnd (Congestion Windows)
  - How many bytes can be sent without overflowing routers
  - Computed by congestion control algorithm

- AdvertisedWindow
  - How many bytes can be sent without overflowing the sender
  - Determined by the receiver
EffectiveWindow

- Limits how much data can be in transit
- Implemented as # of bytes
- Described as # packets (segments) in this lecture

MaxWindow = min(cwnd, AdvertisedWindow)

EffectiveWindow = MaxWindow – (LastByteSent – LastByteAcked)

Two Basic Components

- Detecting congestion
- Rate adjustment algorithm
  - Depends on congestion or not
  - Three subproblems within adjustment problem
    - Finding fixed bandwidth
    - Adjusting to bandwidth variations
    - Sharing bandwidth
Detecting Congestion

- Packet dropping is best sign of congestion
  - Delay-based methods are hard and risky

- How do you detect packet drops? ACKs
  - TCP uses ACKs to signal receipt of data
  - ACK denotes last contiguous byte received
    • Actually, ACKs indicate next segment expected

- Two signs of packet drops
  - No ACK after certain time interval: time-out
  - Several duplicate ACKs (ignore for now)

Rate Adjustment

- Basic structure:
  - Upon receipt of ACK (of new data): increase rate
  - Upon detection of loss: decrease rate

- But what increase/decrease functions should we use?
  - Depends on what problem we are solving
Problem #1: Single Flow, Fixed BW

- Want to get a first-order estimate of the available bandwidth
  - Assume bandwidth is fixed
  - Ignore presence of other flows

- Want to start slow, but rapidly increase rate until packet drop occurs (“slow-start”)

- Adjustment:
  - cwnd initially set to 1
  - cwnd++ upon receipt of ACK

Slow-Start

- cwnd increases exponentially: cwnd doubles every time a full cwnd of packets has been sent
  - Each ACK releases two packets
  - Slow-start is called “slow” because of starting point
### Problems with Slow-Start

- Slow-start can result in many losses
  - Roughly the size of cwnd $\sim$ BW\*RTT

- Example:
  - At some point, cwnd is enough to fill "pipe"
  - After another RTT, cwnd is double its previous value
  - All the excess packets are dropped!

- Need a more gentle adjustment algorithm once have rough estimate of bandwidth

### Problem #2: Single Flow, Varying BW

- Want to be able to track available bandwidth, oscillating around its current value

- Possible variations: (in terms of RTTs)
  - Multiplicative increase or decrease: $cwnd \rightarrow a*cwnd$
  - Additive increase or decrease: $cwnd \rightarrow cwnd + b$

- Four alternatives:
  - AIAD: gentle increase, gentle decrease
  - AIMD: gentle increase, drastic decrease
  - MIAD: drastic increase, gentle decrease (too many losses)
  - MIMD: drastic increase and decrease
Problem #3: Multiple Flows

- Want steady state to be “fair”

- Many notions of fairness, but here all we require is that two identical flows end up with the same bandwidth

- This eliminates MIMD and AIAD

- AIMD is the only remaining solution!

Buffer and Window Dynamics

- No congestion $\rightarrow$ $x$ increases by one packet/RTT every RTT
- Congestion $\rightarrow$ decrease $x$ by factor 2
AIMD Sharing Dynamics

- No congestion → rate increases by one packet/RTT every RTT
- Congestion → decrease rate by factor 2

AIAD Sharing Dynamics

- No congestion → x increases by one packet/RTT every RTT
- Congestion → decrease x by 1
Efficient Allocation: Challenges of Congestion Control

- Too slow
  - Fail to take advantage of available bandwidth → underload
- Too fast
  - Overshoot knee → overload, high delay, loss
- Everyone’s doing it
  - May all under/over shoot → large oscillations
- Optimal:
  - $\sum x_i = X_{\text{goal}}$
  - Efficiency = 1 - distance from efficiency line

2 user example

User 1: $x_1$
User 2: $x_2$

Efficiency line

User 1: $x_1$

MIAD

- Does not converge to fairness
  - Not stable at all
- Does not converge to efficiency
  - Stable iff

$$x_{1h} = x_{2h} = \frac{b_1 a_D}{1 - b_1}$$
AIAD

- Does not converge to fairness
  - Stable
- Does not converge to efficiency
  - Stable iff $a_D = a_I$

MIMD

- Does not converge to fairness
  - Stable
- Converges to efficiency iff
  \[ b_I \geq 1 \]
  \[ 0 \leq b_D < 1 \]
AIMD

- Converges to fairness
- Converges to efficiency
- Increments smaller as fairness increases
  - effect on metrics?

Implementing AIMD

- After each ACK
  - Increment \( cwnd \) by \( 1/cwnd \) \( (cwnd += 1/cwnd) \)
  - As a result, \( cwnd \) is increased by one only if all segments in a \( cwnd \) have been acknowledged

- But need to decide when to leave slow-start and enter AIMD
  - Use ssthresh variable
Slow Start/AIMD Pseudocode

Initially:
  cwnd = 1;
  ssthresh = infinite;

New ack received:
  if (cwnd < ssthresh)
      /* Slow Start*/
      cwnd = cwnd + 1;
  else
      /* Congestion Avoidance */
      cwnd = cwnd + 1/cwnd;

Timeout:
  /* Multiplicative decrease */
  ssthresh = cwnd/2;
  cwnd = 1;

The big picture (with timeouts)

![Diagram showing the progression from Slow Start to AIMD with timeouts and ssthresh](image-url)