Principles of congestion control

Congestion:

- Informally: “too many sources sending too much data too fast for network to handle”
- Different from flow control!
- Manifestations:
  - Lost packets (buffer overflow at routers)
  - Long delays (queueing in router buffers)
- A top-10 problem!
Congestion

- Different sources compete for resources inside network

- Why is it a problem?
  - Sources are unaware of current state of resource
  - Sources are unaware of each other
  - In many situations will result in $< 8$ Mbps of throughput (congestion collapse)
Causes/costs of congestion: Scenario 1

- Two senders, two receivers
- One router, infinite buffers
- No retransmission

- Maximum achievable throughput
- Large delays when congested
Causes/costs of congestion: Scenario 2

- One router, *finite* buffers
- Sender retransmission of lost packet
Causes/costs of congestion: Scenario 2

- Always: $\lambda_{in} = \lambda_{out}$ (goodput)
- "Perfect" retransmission only when loss: $\lambda'_{in} > \lambda_{out}$
- Retransmission of delayed (not lost) packet makes $\lambda'_{in}$ larger (than perfect case) for same $\lambda_{out}$

```
\begin{align*}
\lambda_{out} & \quad \lambda_{in} \quad C/2 \\
\text{a} & \quad \text{C/2} \\
\lambda_{out} & \quad \lambda_{in} \quad C/3 \\
\text{b} & \quad \text{C/2} \\
\lambda_{out} & \quad \lambda_{in} \quad C/4 \\
\text{c} & \quad \text{C/2}
\end{align*}
```

"Costs" of congestion:

- More work (retransmissions) for given "goodput"
- Unneeded retransmissions: Link carries multiple copies of pkt
Causes/costs of congestion: Scenario 3

- Four senders
- Multihop paths
- Timeout/retransmit

**Q:** What happens as $\lambda_{in}$ and $\lambda'_{in}$ increase?
Causes/costs of congestion: Scenario 3

Another “cost” of congestion:

- When packet dropped, any “upstream” transmission capacity used for that packet was wasted!
Congestion collapse

Definition: *Increase in network load results in decrease of useful work done*

Many possible causes

- Spurious retransmissions of packets still in flight
  - Classical congestion collapse
  - How can this happen with packet conservation
  - Solution: Better timers and TCP congestion control

- Undelivered packets
  - Packets consume resources and are dropped elsewhere in network
  - Solution: Congestion control for ALL traffic
Other congestion collapse causes

- Fragments
  - Mismatch of transmission and retransmission units
  - Solutions
    - Make network drop all fragments of a packet
    - Do path MTU discovery

- Control traffic
  - Large percentage of traffic is for control
    - Headers, routing messages, DNS, etc.

- Stale or unwanted packets
  - Packets that are delayed on long queues
  - “Push” data that is never used
Where to prevent collapse?

- Can end hosts prevent problem?
  - Yes, but must trust end hosts to do right thing
  - E.g., sending host must adjust amount of data it puts in the network based on detected congestion

- Can routers prevent collapse?
  - No, not all forms of collapse
  - Doesn’t mean they can’t help
  - Sending accurate congestion signals
  - Isolating well-behaved from ill-behaved sources
Congestion control and avoidance

- A mechanism which
  - Uses network resources efficiently
  - Preserves fair network resource allocation
  - Prevents or avoids collapse

- Congestion collapse is not just a theory
  - Has been frequently observed in many networks
Congestion collapse

Congestion collapse was first observed on the early Internet in October 1986, when the NSFnet phase-I backbone dropped three orders of magnitude from its capacity of 32 kbit/s to 40 bit/s, and continued to occur until end nodes started implementing Van Jacobson's congestion control between 1987 and 1988.
Congestion control vs. avoidance

- Avoidance keeps the system performing at the knee
- Control kicks in once the system has reached a congested state
Two broad approaches towards congestion control:

**End-end congestion control:**
- No explicit feedback from network
- Congestion inferred from end-system observed loss, delay
- Approach taken by TCP

**Network-assisted congestion control:**
- Routers provide feedback to end systems
  - Choke packet from router to sender
  - Single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - Explicit rate sender should send at
End-to-end congestion control - objectives

- Simple router behavior
- Distributedness
- Efficiency: $X_{\text{knee}} = \sum x_i(t)$
- Fairness: $(\sum x_i)^2/n(\sum x_i^2)$
- Power: (throughput$^\alpha$/delay)
- Convergence: control system must be stable
Basic control model

- Let’s assume window-based control
- Reduce window when congestion is perceived
  - How is congestion signaled?
    - Either mark or drop packets
  - When is a router congested?
    - Drop tail queues – when queue is full
    - Average queue length – at some threshold
- Increase window otherwise
  - Probe for available bandwidth – how?
Linear control

- Many different possibilities for reaction to congestion and probing
  - Examine simple linear controls
  - Window(t + 1) = a + b Window(t)
  - Different $a_i/b_i$ for increase and $a_d/b_d$ for decrease

- Supports various reaction to signals
  - Increase/decrease additively
  - Increased/decrease multiplicatively
  - Which of the four combinations is optimal?
Phase plots

- Simple way to visualize behavior of competing connections over time
Phase plots

- What are desirable properties?
- What if flows are not equal?
Additive increase/decrease

- $X_1$ and $X_2$ in-/decrease by same amount over time
Multiplicative increase/decrease

- $X_1$ and $X_2$ in-/decrease by the same factor
  - Extension from origin
Convergence to efficiency

- Want to converge quickly to intersection of fairness and efficiency lines
Distributed convergence to efficiency
Convergence to fairness
Convergence to efficiency & fairness
Increase

User 2’s Allocation $x_2$

User 1’s Allocation $x_1$

Fairness Line

Efficiency Line

$x^L$
What is the right choice?

- Constraints limit us to AIMD
  - Can have multiplicative term in increase
  - AIMD moves towards optimal point
TCP congestion control

- Motivated by ARPANET congestion collapse
- Underlying design principle: Packet conservation
  - At equilibrium, inject packet into network only when one is removed
  - Basis for stability of physical systems
- Why was this not working?
  - Connection doesn’t reach equilibrium
  - Spurious retransmissions
  - Resource limitations prevent equilibrium
TCP congestion control - solutions

- Reaching equilibrium
  - Slow start

- Eliminates spurious retransmissions
  - Accurate RTO estimation
  - Fast retransmit

- Adapting to resource availability
  - Congestion avoidance
TCP congestion control basics

- Keep a congestion window, cwnd
  - Denotes how much network is able to absorb

- Sender’s maximum window:
  - Min (advertised receiver window, cwnd)

- Sender’s actual window:
  - Max window - unacknowledged segments

- If we have large actual window, should we send data in one shot?
  - No, use acks to clock sending new data
Self-clocking

Sender

Receiver

$A_s$

$A_b$

$A_r$

$P_b$

$P_r$
TCP congestion control: Additive increase, multiplicative decrease (AIMD)

- **Approach:** Increase transmission rate (window size), probing for usable bandwidth, until loss occurs
  - **Additive increase:** Increase \( cwnd \) by 1 MSS every RTT until loss detected
  - **Multiplicative decrease:** Cut \( cwnd \) in half after loss

Saw tooth behavior: probing for bandwidth
TCP Fairness

Fairness goal: if N TCP sessions share same bottleneck link, each should get 1/N of link capacity
Why is TCP fair? (Ideal case!)

Two competing sessions:
- Additive increase gives slope of 1, as throughout increases
- Multiplicative decrease decreases throughput proportionally

![Diagram showing equal bandwidth share, congestion avoidance, and loss response.](image)
Assumption for TCPs fairness

- Window under consideration is large enough
- Same RTT
- Similar TCP parameters
- Enough data to send
- ....