Lecture #11: Reliable Transport
adding reliability while also keeping things efficient and fair
today: moving up to the transport layer to discuss reliable transport

application
the things that actually generate traffic

transport
sharing the network, reliability (or not)
examples: TCP, UDP

network
naming, addressing, routing
examples: IP

link
communication between two directly-connected nodes
examples: ethernet, bluetooth, 802.11 (wifi)

CAIDA's IPv4 AS Core, January 2020
(https://www.caida.org/projects/cartography/as-core/2020/)

1970s:
ARPAnet
layering

hosts.txt
distance-vector routing

TCP, UDP

1978:
flexibility and

early 80s:
growth → change

congestion collapse (which led to congestion control)

OSPF, EGP, DNS

1980s:
growth → change

late 80s:
growth → problems

policy routing

CIDR

1990s:
commercialization

1978:
ARPAnet
layering

hosts.txt
distance-vector routing

TCP, UDP

1970s:
flexibility and
layering

early 80s:
growth → change

congestion collapse (which led to congestion control)

OSPF, EGP, DNS

1990s:
commercialization

TCP, UDP
our (first) goal today is to create a **reliable transport protocol**, which delivers each byte of data **exactly once**, **in-order**, to the receiving application.

- **application**
  - the things that actually generate traffic

- **transport**
  - sharing the network, reliability (or not)
  - examples: TCP, UDP

- **network**
  - naming, addressing, routing
  - examples: IP

- **link**
  - communication between two directly-connected nodes
  - examples: ethernet, bluetooth, 802.11 (wifi)
reliable transport protocols deliver each byte of data exactly once, in-order, to the receiving application.

The sender is allowed to have $W$ outstanding packets at once, but no more.

**sequence numbers:** used to order the packets.

**acknowledgments ("ACKs"):** used to confirm that a packet has been received. An ACK with sequence number $k$ indicates that the receiver has received all packets up to and including $k$.

**timeouts:** used to retransmit packets.

This is known as a **sliding-window protocol**. The window of outstanding (un-ACKed) packets slides along the sequence number space.
reliable transport protocols deliver each byte of data exactly once, in-order, to the receiving application.

**sequence numbers:** used to order the packets

**acknowledgments ("ACKs"):** used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received all packets up to and including k

**timeouts:** used to retransmit packets

**spurious retransmission:** the sender retransmitted a packet that the receiver had already ACKed
reliable transport protocols deliver each byte of data exactly once, in-order, to the receiving application.

**sequence numbers**: used to order the packets

**acknowledgments** ("ACKs"): used to confirm that a packet has been received

an ACK with sequence number k indicates that the receiver has received all packets up to and including k

**timeouts**: used to retransmit packets

**spurious retransmission**: the sender retransmitted a packet that the receiver had already ACKed
reliable transport protocols deliver each byte of data exactly once, in-order, to the receiving application.

**question:** what should $W$ be?

how can a single reliable sender, using a sliding-window protocol, set its window size to maximize utilization — but prevent congestion and unfairness — given that there are many other end points using the network, all with different, changing demands?
**congestion control**: controlling the source rates to achieve **efficiency** and **fairness**

**efficiency**: minimize drops, minimize delay, maximize bottleneck utilization

**fairness**: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

**AIMD**: every RTT, if there is no loss, \( W = W + 1 \); else, \( W = W/2 \)
**congestion control**: controlling the source rates to achieve **efficiency** and **fairness**

- **efficiency**: minimize drops, minimize delay, maximize bottleneck utilization
- **fairness**: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

**AIMD**: every RTT, if there is no loss, $W = W + 1$; else, $W = W/2$
**congestion control**: controlling the source rates to achieve **efficiency** and **fairness**

- **efficiency**:
  - (utilization)
  - the network is fully utilized when the bottleneck link is “full”
  - minimize drops, minimize delay, maximize bottleneck utilization

- **fairness**:
  - under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck
  - the network is fair when $S_1$ and $S_2$ are sending at the same rate
  - $S_1$ is sending more than $S_2$
  - $S_2$ is sending more than $S_1$

**AIMD**:
- every RTT, if there is no loss, $W = W + 1$; else, $W = W/2$
congestion control: controlling the source rates to achieve efficiency and fairness

\[ R_1 + R_2 = B \]

the network is fully utilized when the bottleneck link is “full”

the network is fair when \( S_1 \) and \( S_2 \) are sending at the same rate

\[ R_1 = R_2 (S_1's \text{ sending rate}) \]

\[ R_2 = R_2 (S_2's \text{ sending rate}) \]

eventually, \( R_1 \) and \( R_2 \) will come to oscillate around the fixed point

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, \( W = W + 1 \); else, \( W = W/2 \)

Katrina LaCurts | lacurts@mit.edu | 6.033 2022
congestion control: controlling the source rates to achieve efficiency and fairness

**efficiency**: minimize drops, minimize delay, maximize bottleneck utilization

**fairness**: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

**AIMD**: every RTT, if there is no loss, 
\[ W = W + 1; \text{ else, } W = W/2 \]

**slow-start**: at the start of the connection, double \( W \) every RTT
**congestion control**: controlling the source rates to achieve **efficiency** and **fairness**

### Efficiency

Minimize drops, minimize delay, maximize bottleneck utilization

### Fairness

Under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

### AIMD

Every RTT, if there is no loss, \( W = W + 1 \); else, \( W = W/2 \)

### Slow-Start

At the start of the connection, double \( W \) every RTT

### Fast Retransmit/Fast Recovery

Retransmit packet \( k+1 \) as soon as four ACKs with sequence number \( k \) are received

\( \text{(four = original ACK + 3 “dup” ACKs)} \)
congestion control: controlling the source rates to achieve efficiency and fairness

in practice, if a single packet is lost, the three “dup” ACKs will be received before the timeout for that packet expires

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, \( W = W + 1 \); else, \( W = W/2 \)

slow-start: at the start of the connection, double \( W \) every RTT

fast retransmit/fast recovery: retransmit packet \( k+1 \) as soon as four ACKs with sequence number \( k \) are received

(four = original ACK + 3 “dup” ACKs)
**congestion control**: controlling the source rates to achieve **efficiency** and **fairness**

**efficiency**: minimize drops, minimize delay, maximize bottleneck utilization

**fairness**: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

**AIMD**: every RTT, if there is no loss, $W = W + 1$; else, $W = W/2$

**slow-start**: at the start of the connection, double $W$ every RTT

**fast retransmit/fast recovery**: retransmit packet $k+1$ as soon as four ACKs with sequence number $k$ are received

(four = original ACK + 3 “dup” ACKs)
**congestion control**: controlling the source rates to achieve **efficiency** and **fairness**

**AIMD**: every RTT, if there is no loss, \( W = W + 1 \); else, \( W = W/2 \)

**slow-start**: at the start of the connection, double \( W \) every RTT

**fast retransmit/fast recovery**: retransmit packet \( k+1 \) as soon as four ACKs with sequence number \( k \) are received

(four = original ACK + 3 “dup” ACKs)

**efficiency**: minimize drops, minimize delay, maximize bottleneck utilization

**fairness**: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck
congestion control: controlling the source rates to achieve efficiency and fairness

in certain types of networks, this style of congestion control can make these problems worse

in practice, fairness is tough to define and assess

AIMD is not the final word in congestion avoidance; modern versions (e.g. CUBIC TCP) use different rules to set the window size

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

fairness: under infinite offered load, split bandwidth evenly among all sources sharing a bottleneck

AIMD: every RTT, if there is no loss, \( W = W + 1 \); else, \( W = W/2 \)

slow-start: at the start of the connection, double \( W \) every RTT

fast retransmit/fast recovery: retransmit packet \( k+1 \) as soon as four ACKs with sequence number \( k \) are received
(four = original ACK + 3 “dup” ACKs)
next time: TCP congestion control doesn’t react to congestion until after it’s a problem; could we get senders to react before queues are full?
**Sally Floyd, Who Helped Things Run Smoothly Online, Dies at 69**

In the early 1990s, Dr. Floyd was one of the inventors of Random Early Detection, which continues to play a vital role in the stability of the internet.

One byproduct of Dr. Floyd’s work reflected her passion for keeping things fair to all internet users. “Her work on congestion control was about keeping it working for everyone,” Dr. Kohler said. “For people with fast connections, and for people with slow connections.”