6.033 Spring 2022
Lecture #11: Reliable Transport
adding reliability while also keeping things efficient and fair
1970s: ARPAnet  
1978: flexibility and layering  
early 80s: growth → change  
late 80s: growth → problems  
1993: commercialization

hosts.txt  
distance-vector routing

TCP, UDP  
OSPF, EGP, DNS

congestion collapse  
policy routing  
CIDR

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

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)
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

| link | communication between two directly-connected nodes  
|      | examples: ethernet, bluetooth, 802.11 (wifi) |
| network | naming, addressing, routing  
|         | examples: IP |
| transport | sharing the network, reliability (or not)  
|          | examples: TCP, UDP |
| application | the things that actually generate traffic |
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.
reliable transport protocols deliver each byte of data exactly once, in-order, to the receiving application.
reliable transport protocols deliver each byte of data exactly once, in-order, to the receiving application.
reliable transport protocols deliver each byte of data exactly once, in-order, to the receiving application.

sequence numbers: used to order the packets
reliable transport protocols deliver each byte of data exactly once, in-order, to the receiving application.

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

![Diagram showing sequence numbers from sender to receiver](image-url)
reliable transport protocols deliver each byte of data **exactly once, in-order**, to the receiving application.

**sequence numbers:** used to order the packets
reliable transport protocols deliver each byte of data exactly once, in-order, to the receiving application.

Sequence numbers: used to order the packets.
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 $k$. 
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 $k$
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 $k$.
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 $k$.

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.

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 \( k \).

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.

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.

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.

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.

the sender is allowed to have \( W \) outstanding packets at once, but no more.

\[
\begin{array}{c|c}
\text{sender} & \text{receiver} \\
\hline
1 & 1 \\
2 & 2 \\
3 & 3 \\
4 & 4 \\
5 & 5 \\
\vdots & \text{X} \\
10 & 6 \\
\end{array}
\]

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 \( k \).

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.

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 \( k \).

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. 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 \( k \)

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.

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.

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

- **sequence numbers**: used to order the packets
  - an ACK with sequence number \( k \) indicates that the receiver has received all packets up to and including \( k \)

- **acknowledgments (“ACKs”)**: used to confirm that a packet has been received

- **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.

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
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
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
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
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.
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
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
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.

Notice that (in this example) the timeout expired before the sender got an ACK indicating that 7 had been received.
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

notice that (in this example) the timeout expired before the sender got an ACK indicating that 7 had been received.
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

notice that (in this example) the timeout expired before the sender got an ACK indicating that 7 had been received.
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.

Notice that (in this example) the timeout expired before the sender got an ACK indicating that 7 had been received.
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.

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.

**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.

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 $k$

timeout: 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.
reliable transport protocols deliver each byte of data \textbf{exactly once, in-order}, to the receiving application.

\textbf{question:} what should $W$ be?
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
congestion control: controlling the source rates to achieve efficiency and fairness

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

---

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

---

**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**

- **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”

**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**:
- 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**: 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

\[ R_1 + R_2 = B \]

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”

**fairness**: the network is fair when $S_1$ and $S_2$ are sending at the same rate

**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: 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”

efficiency: minimize drops, minimize delay, maximize bottleneck utilization

**fairness**: the network is fair when $S_1$ and $S_2$ are sending at the same rate

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”
  
- **fairness**: the network is fair when $S_1$ and $S_2$ are sending at the same rate

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

**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**

**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**: 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

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

\[ R_1 + R_2 = B \]

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

\[ R_1 = R_2 \]

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**: 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, \( W = W/2 \)

\[
R_1 + R_2 = B
\]

- \( R_1 \) (\( S_1 \)'s sending rate)
- \( R_2 \) (\( S_2 \)'s sending rate)
- \( B \)

- **efficiency** (utilization)
  - the network is fully utilized when the bottleneck link is “full”

- **fairness**
  - the network is fair when \( S_1 \) and \( S_2 \) are sending at the same rate
**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, 
\[ W = W + 1; \text{ else, } W = W/2 \]
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

\[ R_1 + R_2 = B \]

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

\[ R_1 = R_2 \]

\( W \) = \( \frac{W}{2} \) if there is no loss, otherwise, \( W = W + 1 \)

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, \( \text{W} \) = \( \text{W} + 1 \); else, \( \text{W} \) = \( \text{W}/2 \)
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**:
- 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, $W = W/2$

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

**Graph**:
- $R_1$ (S₁'s sending rate)
- $R_2$ (S₂'s sending rate)
- $B$ (Bottleneck bandwidth)

$R_1 + R_2 = B$
**congestion control**: controlling the source rates to achieve **efficiency** and **fairness**

- **efficiency** (utilization): 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: 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, \( W = W/2 \)

\[
R_1 + R_2 = B
\]

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 sending rate)

\[
R_2
\]

(\( S_2 \)'s sending rate)
**congestion control**: controlling the source rates to achieve **efficiency** and **fairness**

The network is fully utilized when the bottleneck link is "full"

**efficiency** (utilization)

the network is fully utilized when the bottleneck link is "full"

**fairness**

the network is fair when $S_1$ and $S_2$ are sending at the same rate

$R_1 + R_2 = B$

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

\[ R_1 + R_2 = B \]

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

\[ R_1 = R_2 \]

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

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

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:** 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
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.
**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
**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
**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

\( \text{four} = \text{original ACK} + 3 \text{ “dup” ACKs} \)
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**

**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

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

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

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

in practice, fairness is tough to define and assess

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**

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

\( \text{(four = original ACK + 3 "dup" ACKs)} \)
1970s: ARPAnet
1978: flexibility and layering
early 80s: growth → change
late 80s: growth → problems
1993: commercialization

hosts.txt
distance-vector
TCP, UDP
OSPF, EGP, DNS
congestion collapse
policy routing
CIDR

1978:
link
network
application

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

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?

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)

Katrina LaCurts | lacurts@mit.edu | 6.033 2022
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.”