Computer Networks X_400487

Lecture 10

Chapter 6: The Transport Layer—Part 2

Lecturer: Jesse Donkervliet

Roadmap: Transport Layer

1. Transport layer responsibilities and challenges
2. Connection establishment and release
3. **Revisiting reliable delivery and flow control**
   1. **Reliable delivery**
   2. Flow control
4. Congestion control and bandwidth allocation
5. TCP and UDP

The End-To-End Argument

The lower layers

If the network is unable to provide a feature by itself, it should be removed from the network and provided by the hosts.

Transport layer or higher

Q: Can you think of an example of a feature provided by the hosts?

Q: Can you think of a *feature provided by the network*?

Error control in the transport layer

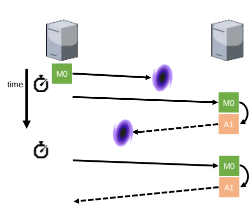
The transport layer is responsible for providing a ***reliable*** data stream over an unreliable network.

Q: Did we not take care of this in the data link layer?

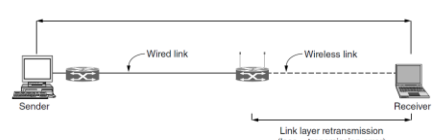
Transport layer checks the end-to-end correctness of data.

Q: Why not do error control only at the transport layer?

Reliable Delivery through Retransmissions

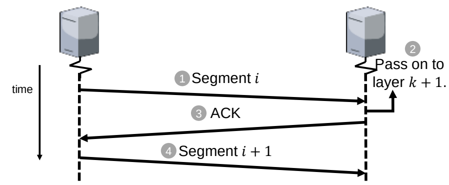


Improving Performance by using Error control on lower layers



Error control and crash recovery

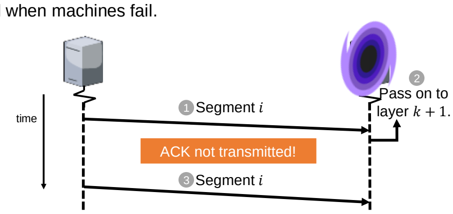
Protocol under normal circumstances.



Error control and crash recovery

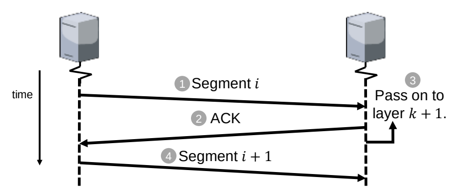
Q: How to solve this?

Protocol when machines fail.



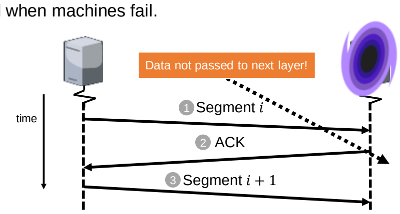
Crash recovery

Protocol under normal circumstances.



Error control and crash recovery

Protocol when machines fail.



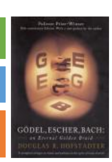
Crash recovery on layer k

We cannot create fool-proof crash recovery in layer k.

Recovery from a layer k crash can only be done by layer $k+1$.

Q: What does this mean in practice?

When a crash occurs, the transport layer leaves it to the application layer to fix it!

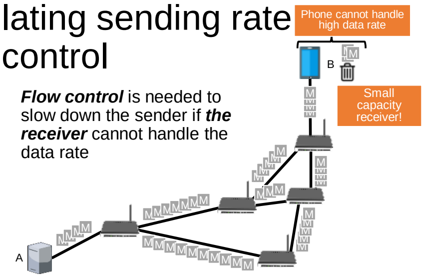


Roadmap: Transport Layer

1. Transport layer responsibilities and challenges
2. Connection establishment and release
3. **Revisiting reliable delivery and flow control**
   1. Reliable delivery
   2. **Flow control**
4. Congestion control and bandwidth allocation
5. TCP and UDP

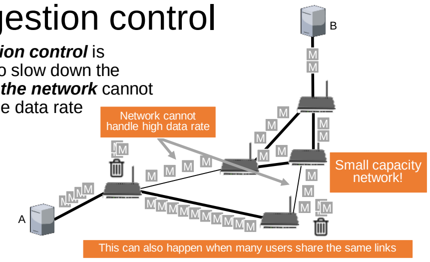
Regulating sending rate
Flow control

Flow control is needed to slow down the sender if ***the receiver*** cannot handle the data rate

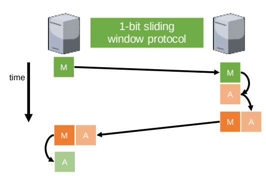


Regulating sending rate
Congestion control

Congestion control is needed to slow down the sender if ***the network*** cannot handle the data rate



Stop-and-Wait:
A 1-Bit Sliding Window Protocol

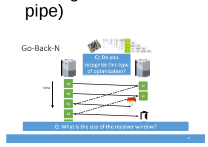


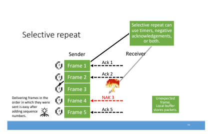
Stop-and-Wait:
A 1-Bit Sliding Window Protocol

Bandwidth inefficient for high-latency channels

Sliding window protocols

Send multiple frames at the same time before waiting for an acknowledgement. (i.e., filling the pipe)





Recap: Link Utilization

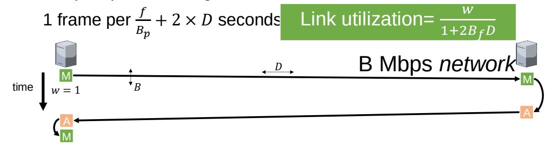
- Frame size (in bits/bytes): f
- Window size (in frames): w
- Bandwidth (max. data rate of physical channel): B_p
- Bandwidth (frames per second): B_f
- Propagation delay (in seconds): D

It takes $\frac{f}{B_p}$ seconds to send frame, $\frac{B_p}{f} = B_f$

It takes D s for the frame to arrive at the receiver, takes D s for the (0-bit) acknowledgment to come back at the sender

1 frame per $\frac{f}{B_p} + 2 \times D$ seconds

Link utilization= $\frac{w}{1+2B_f D}$



Flow control and buffer management

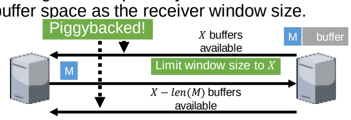
Used by TCP!

Received packets have to be buffered at the receiver.

Q: Why do we need this?

We have to wait for the application to `read` the data

Perform buffer management separately.
Use available buffer space as the receiver window size.



Roadmap: Transport Layer

1. Transport layer responsibilities and challenges
2. Connection establishment and release
3. Revisiting reliable delivery and flow control
4. **Congestion control and bandwidth allocation**
5. TCP and UDP

Today

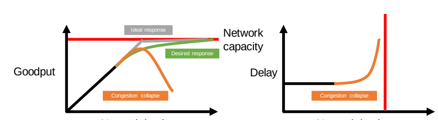
1. Congestion Control in TCP/IP
2. DNS
3. Email
4. Quiz?!

Congestion control

Both packet loss and end-to-end delay can be used to signal congestion!

Both the ***network layer*** and the ***transport layer*** are responsible for congestion control.

The ***transport layer*** controls the workload; implements congestion control and flow control by reducing sending rate.



Congestion control requires resource management

Congestion occurs if the workload is too large for the available network resources.

The workload of all users combined should not be too large for the available network resources.

Coordinate to divide network resources

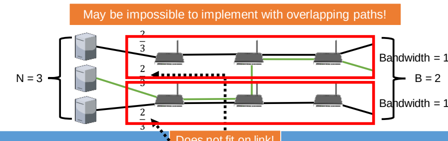
Fair bandwidth allocation

How to divide the available bandwidth over multiple senders?

Assume that we wave a total bandwidth B and N machines.

Q: How much bandwidth does each machine get?

May be impossible to implement with overlapping paths!



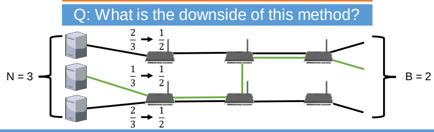
Fair bandwidth allocation
Max-min fairness

Maximizes minimum bandwidth, then uses excess bandwidth where possible.

Q: Is this bandwidth allocation max-min fair?

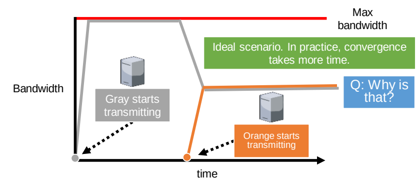
No, because we can increase the minimum bandwidth.

Q: What is the downside of this method?



Fair bandwidth allocation
Convergence

When new connections enter the network, the bandwidth needs to be reallocated.



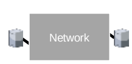
Available bandwidth is unknown

Q: Why is this the case?

The transport layer is not aware of the network topology, or who else is using the network.

Q: How to solve this problem?

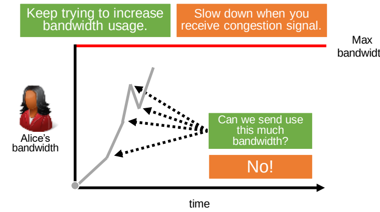
Because there is no centralized control, we need to dynamically adjust bandwidth usage using trial and error.



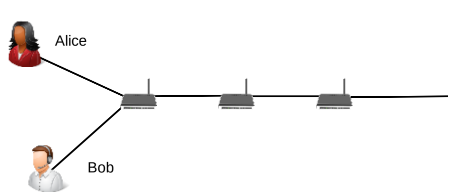
Dynamically adjust bandwidth using trial and error

Keep trying to increase bandwidth usage.

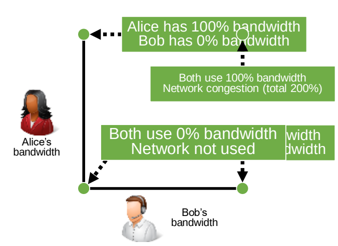
Slow down when you receive congestion signal.



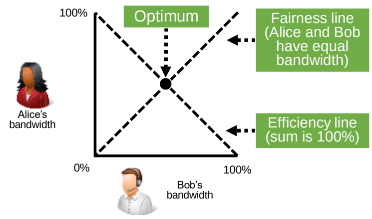
Sharing bandwidth example



Sharing bandwidth example



Sharing bandwidth
Efficiency and fairness



Regulating sending rate
Approaches

Q: Which one should we use?

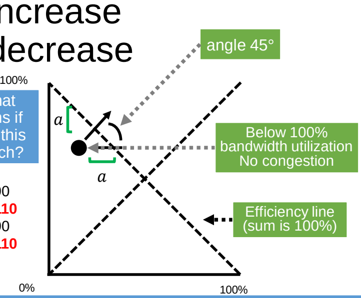
Multiple approaches to increase/decrease sending rate:

1. Additive (rate $+x$, rate $-x$).
2. Multiplicative (rate $\times x$, rate $\times \frac{1}{x}$).
3. Combination of both:
   1. Additive increase, additive decrease.
   2. Additive increase, multiplicative decrease.
   3. Multiplicative increase, additive decrease.
   4. Multiplicative increase, multiplicative decrease.

Additive increase
Additive decrease

Q: What happens if we use this approach?

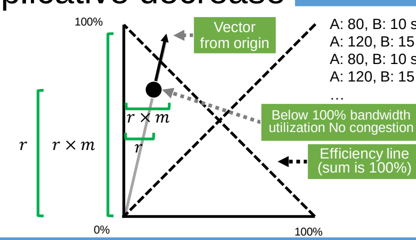
A: 80, B: 10 sum: 90
A: 90, B: 20 sum: **110**
A: 80, B: 10 sum: 90
A: 90, B: 20 sum: **110**
…



Multiplicative increase
Multiplicative decrease

Q: What happens if we use this approach?

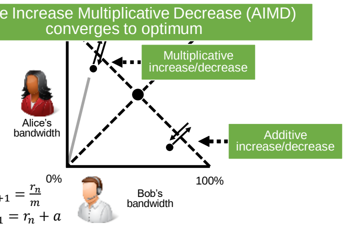
A: 80, B: 10 sum: 90
A: 120, B: 15 sum: **135**
A: 80, B: 10 sum: 90
A: 120, B: 15 sum: **135**
…



Regulating sending rate
Efficiency and fairness

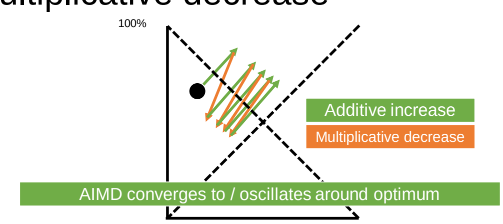
Additive Increase Multiplicative Decrease (AIMD) converges to optimum

A: 80, B: 10 sum: 90
A: 90, B: 20 sum: **110**
A: 45, B: 10 sum: 55
A: 55, B: 20 sum: 75
A: 65, B: 30 sum: 95
A: 75, B: 40 sum: **115**
A: 37, B: 20 sum: 57
Converges



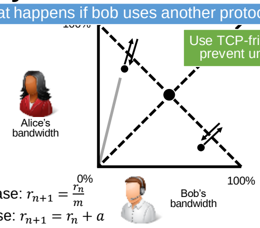
Decrease: $r_{n+1} = \frac{r_n}{m}$
Increase: $r_{n+1} = r_n + a$

Additive increase
Multiplicative decrease



Regulating sending rate Efficiency and fairness

Q: What happens if bob uses another protocol such as UDP?



Decrease: $r_{n+1} = \frac{r_n}{m}$

Increase: $r_{n+1} = r_n + a$

Roadmap: Transport Layer

1. Transport layer responsibilities and challenges
2. Connection establishment and release
3. Revisiting reliable delivery and flow control
4. Congestion control and bandwidth allocation
5. **TCP and UDP**

Internet protocols

The protocols that make the internet work.

Most popular on the transport layer:
1. UDP
2. TCP

But others exist!

All or Nothing

May not meet your application's requirements!

Insufficient separation between *mechanism* and *policy*

You can create your own!

Comparing complexity by number of RFCs

Request For Comment (RFC) are published by the Internet Engineering Task Force (IETF).

UDP: RFC 768

Overview of RFCs in RFC 4614.

TCP: RFC 793, RFC 1122, RFC 1323, RFC 2018, RFC 2581, RFC 2873, RFC 2988, RFC 3168, RFC …, RFC …, …

User Datagram Protocol (UDP)

RFC 768

Very thin layer on top of IP. Header provides ***ports*** needed to connect to remote applications.

32 bits

Source port	Destination port
UDP length	UDP checksum

The UDP header

User Datagram Protocol (UDP)

RFC 768

Very thin layer on top of IP. Header provides ***ports*** needed to connect to remote applications.

32 bits

Source port	Destination port
UDP length	UDP checksum

The UDP header

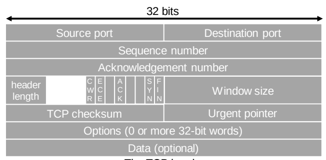
Includes fields from the IP header!

UDP does ***not*** do:
1. Flow control
2. Congestion control
3. Retransmissions

Q: Can you name a service that works well with UDP?

Transmission Control Protocol (TCP)

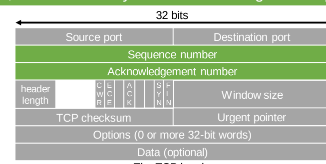
One of the most important protocols on the internet.
Provides a ***reliable end-to-end byte stream*** over an unreliable network.



The TCP header

Transmission Control Protocol (TCP)

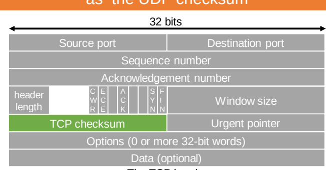
Sequence numbers and acknowledgements allow reliable, in-order delivery and enable sliding window protocols



The TCP header

Transmission Control Protocol (TCP)

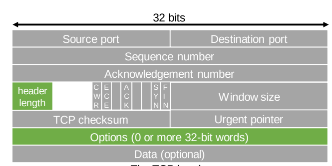
TCP checksum uses same IP-header fields as the UDP checksum



The TCP header

Transmission Control Protocol (TCP)

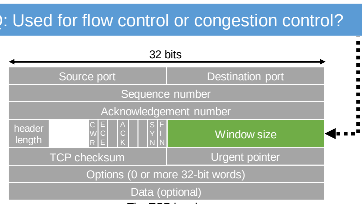
Q: How do we know how long the TCP segment is?



The TCP header

Transmission Control Protocol (TCP)

Q: Used for flow control or congestion control?

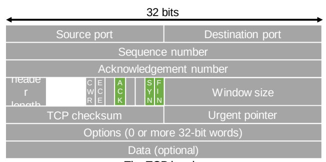


The TCP header

Connections in TCP

Transmission Control Protocol (TCP)

Used to establish/release connections

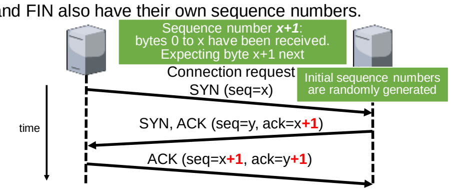


The TCP header

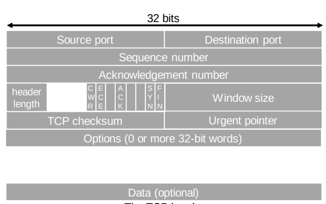
TCP connection establishment Three-way handshake

Uses timestamp option to improve performance on high-bandwidth networks

Every ***data byte*** has its own sequence number.*
*SYN and FIN also have their own sequence numbers.



TCP Timestamp Option

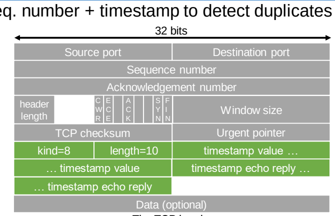


The TCP header

TCP Timestamp Option

Q: How does this improve performance?

Use seq. number + timestamp to detect duplicates



The TCP header

TCP PAWS

More specifically, if the maximum effective bandwidth at which TCP is able to transmit over a particular path is B bytes per second, then the following constraint must be satisfied for error-free operation:

2**31 / B > MSL (secs) [1]

The following table shows the value for Twrap = 2**31/B in seconds, for some important values of the bandwidth B:

Jacobson, Braden, & Borman [Page 5]

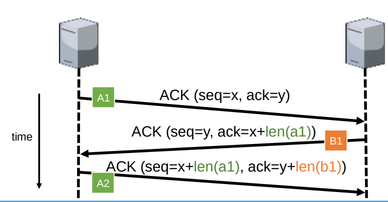
RFC 1323 TCP Extensions for High Performance May 1992

Network	B*8 bits/sec	B bytes/sec	Twrap secs
ARPANET	56kbps	7KBps	3*10**5 (~3.6 days)
DS1	1.5Mbps	190KBps	10**4 (~3 hours)
Ethernet	10Mbps	1.25MBps	1700 (~30 mins)
DS3	45Mbps	5.6MBps	380
FDDI	100Mbps	12.5MBps	170
Gigabit	1Gbps	125MBps	17

TCP sequence numbers

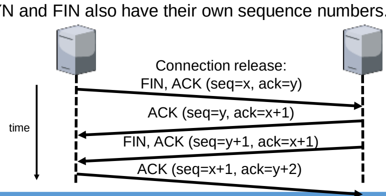
Every ***data byte*** has its own sequence number

Initial sequence numbers are randomly generated



TCP connection release
Two simplex channels

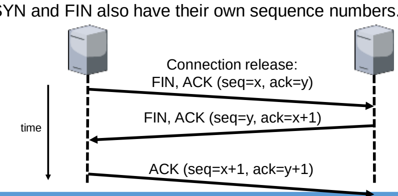
Every ***data byte*** has its own sequence number.*
*SYN and FIN also have their own sequence numbers.



TCP connection release
Two simplex channels

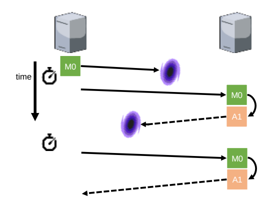
Q: How to solve the two army problem?

Every ***data byte*** has its own sequence number.*
*SYN and FIN also have their own sequence numbers.



Error Control in TCP

Reliable Delivery through Retransmissions



Setting Retransmission Timers

How long should we wait before retransmitting a frame?

Q: What are the bounds?

- Timer must be longer than round-trip time.

Congestion makes round-trip time variable!

- If we set timer too high, bandwidth efficiency goes down

Dynamic Timeouts in TCP

Use a *weighted moving average* to smooth round trip time (R):

$$\text{SRTT} = \alpha \times \text{SRTT} + (1 - \alpha) \times \text{R}$$

Do the same for the round trip time variance (RTTVAR):

$$\text{RTTVAR} = \beta \times \text{RTTVAR} + (1 - \beta) \times |\text{SRTT} - \text{R}|$$

Calculate new retransmission timeout (RTO) based on these values:

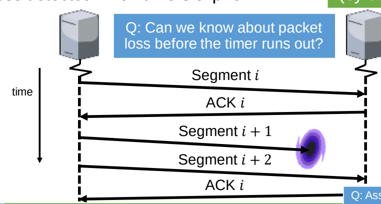
$$\text{RTO} = \text{SRTT} + 4 \times \text{RTTVAR}$$

Performance improvement Fast ***retransmission***

Takes time (by design)!

Packet loss detected when timers expire.

Q: Can we know about packet loss before the timer runs out?



Q: Assumption?

Duplicate acknowledgments can indicate packet loss

Flow Control in TCP

Flow control and buffer management

Used by TCP!

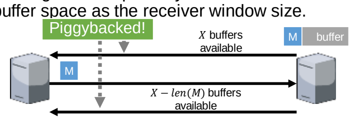
Received packets have to be buffered at the receiver.

Q: Why do we need this?

We have to wait for the application to `read` the data

Perform buffer management separately.
Use available buffer space as the receiver window size.

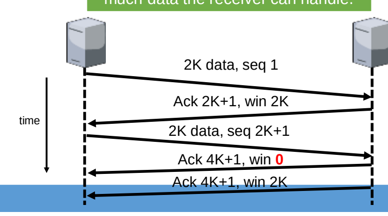
Piggybacked!



TCP window size Flow control

Q: Can you think of a potential problem?

The ***window size*** tells sender how much data the receiver can handle.



TCP window size Nagle's algorithm

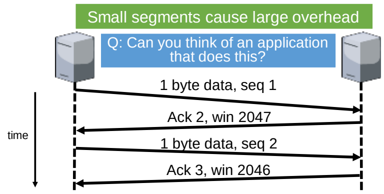
Do not send more than one small packet at a time: wait for ack

Q: For which applications may this not work well?

A sender that produces data in small amounts.

Small segments cause large overhead

Q: Can you think of an application that does this?

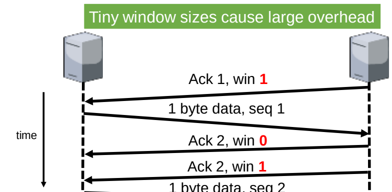


TCP window size Silly-window syndrome

Do not send window updates if available space is too small

A receiver that consumes data in small amounts.

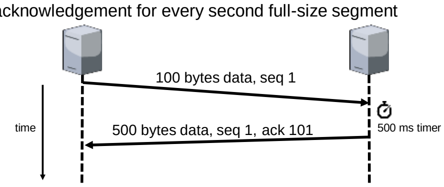
Tiny window sizes cause large overhead



TCP Delayed Acknowledgements

Try to improve bandwidth efficiency (e.g., through piggy-backing)

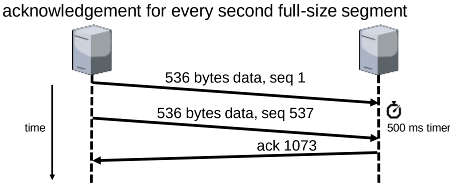
- Wait up to 500 ms to send acknowledgement
- Send acknowledgement for every second full-size segment



TCP Delayed Acknowledgements

Try to improve bandwidth efficiency (e.g., through piggy-backing)

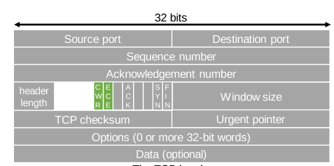
- Wait up to 500 ms to send acknowledgement
- Send acknowledgement for every second full-size segment



Congestion Control in TCP

Transmission Control Protocol (TCP)

Used for Explicit Congestion Notification



The TCP header

Additive increase multiplicative decrease in TCP

AIMD used to prevent network congestion. Converges to fair and efficient bandwidth allocation.

TCP implements this using its ***congestion window***.

Congestion window is tracked on the sender.
Specifies how many segments can be transmitted.

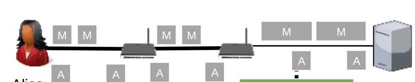
Not the same as the 'window size' field in the TCP segment header!

Q: How does TCP combine the two windows?

AIMD in TCP What value to start with?

We want ***fast convergence***, but sending a large burst can occupy low-bandwidth links for a long time.

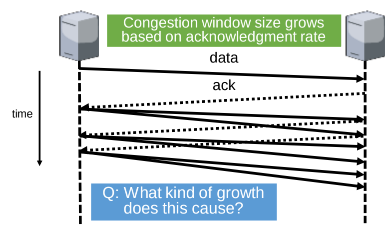
Increase congestion window whenever acknowledgements arrive.



Acknowledgement rate tells us data rate of slowest link.

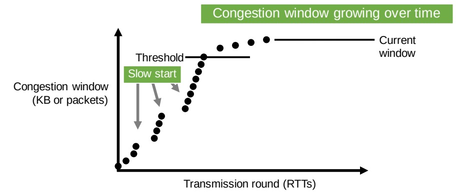
Low bandwidth

AIMD in TCP 'slow' start



TCP 'slow' start

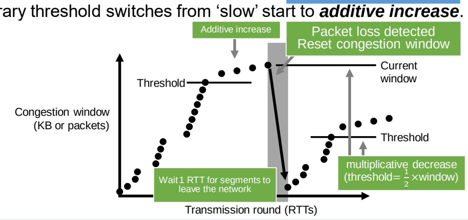
Arbitrary threshold switches from 'slow' start to ***additive increase***.



TCP Tahoe

Q: Can you think of another way to detect packet loss?

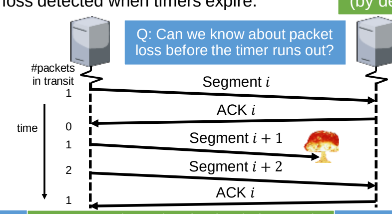
Arbitrary threshold switches from 'slow' start to ***additive increase***.



Performance improvement
Fast ***retransmission***

Takes time (by design)!

Packet loss detected when timers expire.

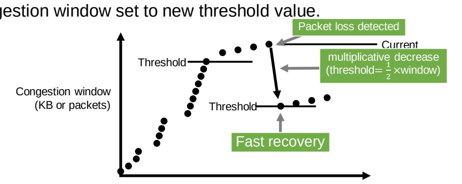


We can count the number of packets in the network!

TCP Reno
(= TCP Tahoe + fast ***recovery***)

Calculates the number of segments in the network by counting the number of duplicate acknowledgements

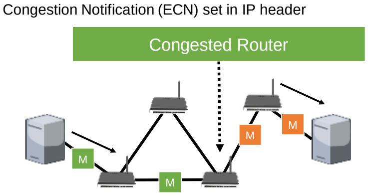
Threshold reduced using ***multiplicative decrease***.
Congestion window set to new threshold value.



What about
Explicit Congestion Notification?

M = regular IP packet with TCP segment

M = Explicit Congestion Notification (ECN) set in IP header

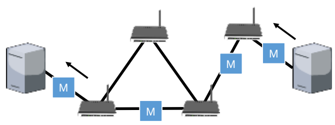


What about Explicit Congestion Notification?

M = regular IP packet with TCP segment

M = Explicit Congestion Notification (ECN) set in IP header

M = ECN-Echo (ECE) set in TCP header



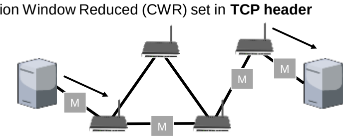
What about Explicit Congestion Notification?

M = regular IP packet with TCP segment

M = Explicit Congestion Notification (ECN) set in **IP header**

M = ECN-Echo (ECE) set in **TCP header**

M = Congestion Window Reduced (CWR) set in **TCP header**



Different Flavors of TCP

TCP versions and congestion signals

1. TCP determines rate based on packet loss.
2. CUBIC TCP determines rate based on packet loss. Used by default in Linux, Windows, MacOS
3. FAST TCP determines rate based on end-to-end delay.
4. Compound TCP determines rate based on end-to-end delay and packet loss.
5. TCP with Explicit Congestion Notification.
6. XCP explicitly tells sender what rate to use.

TCP versions and congestion signals

Implicit congestion signals

1. TCP determines rate based on packet loss.
2. CUBIC TCP determines rate based on packet loss. Used by default in Linux, Windows, MacOS
3. FAST TCP determines rate based on end-to-end delay.
4. Compound TCP determines rate based on end-to-end delay and packet loss.
5. TCP with Explicit Congestion Notification.
6. XCP explicitly tells sender what rate to use.

Roadmap: Transport Layer

1. Transport layer responsibilities and challenges
2. Connection establishment and release
3. Revisiting reliable delivery and flow control
4. Congestion control and bandwidth allocation
5. TCP and UDP

Transport Layer Summary

- Sockets interface
- Connection establishment and release
  - Duplicate detection
  - **Two army problem**
- Seq. num wrap around + duplicate detection → performance limit
- **End-to-end argument**

- Error control
  - Timer management
  - Detection using time-outs or **duplicate acknowledgements**
- Flow control
  - Sending rate limited to smallest **window size**
  - **Nagle's algorithm**
  - **Silly window syndrome**
- Congestion control
  - Sharing available resources
    - **AIMD**
  - Multiple signals: packet loss, latency, etc.