Transmission Control Protocol

Reliable Transport & Congestion Control

Introduction to Computer Networks

CSC358

Dr. Michael A. Galle

January 24, 2025

Inspired by the lectures of Prof. Peter Marbach

Objective

By the end of this class, you will understand:

- The basic concept of Reliable Transport and how it is implemented in TCP
- The need for Congestion Control and its implementation in TCP
- Enhancements and special overhead reduction algorithms in TCP

All materials for this lecture are available in the course repo

- git clone https://github.com/michaelgalle/ComputerNetworks.git

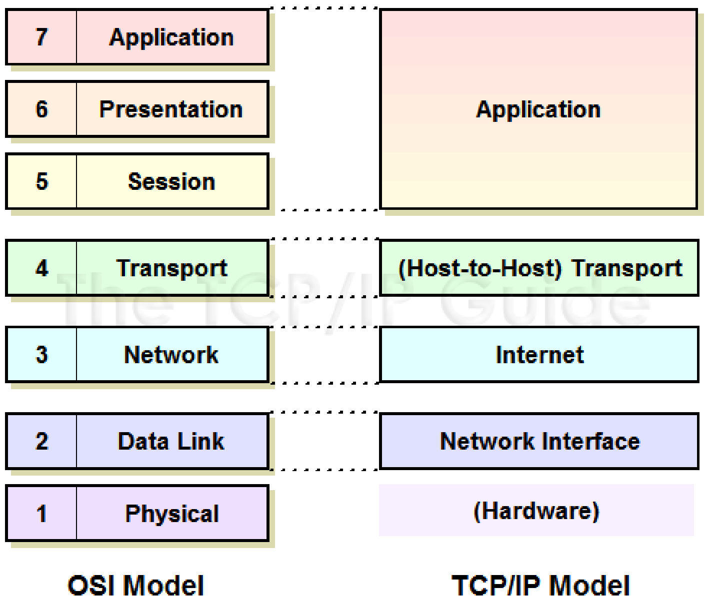
Overview

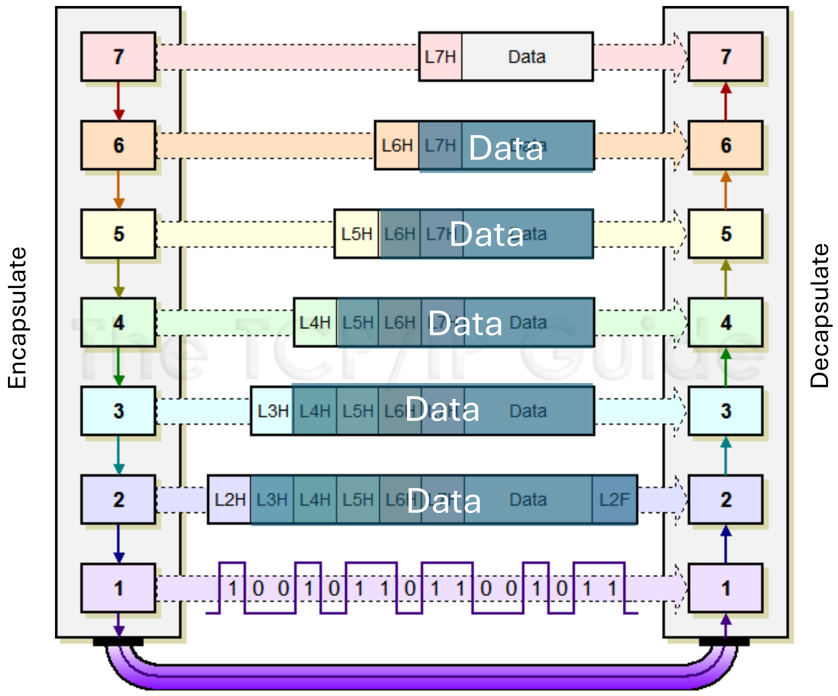
1. Bridge-in
2. Reliable Transport
3. Congestion Control
4. Flipped-classroom Exercise
   - TCP overhead reduction
   - TCP congestion control enhancements

1. Bridge-in

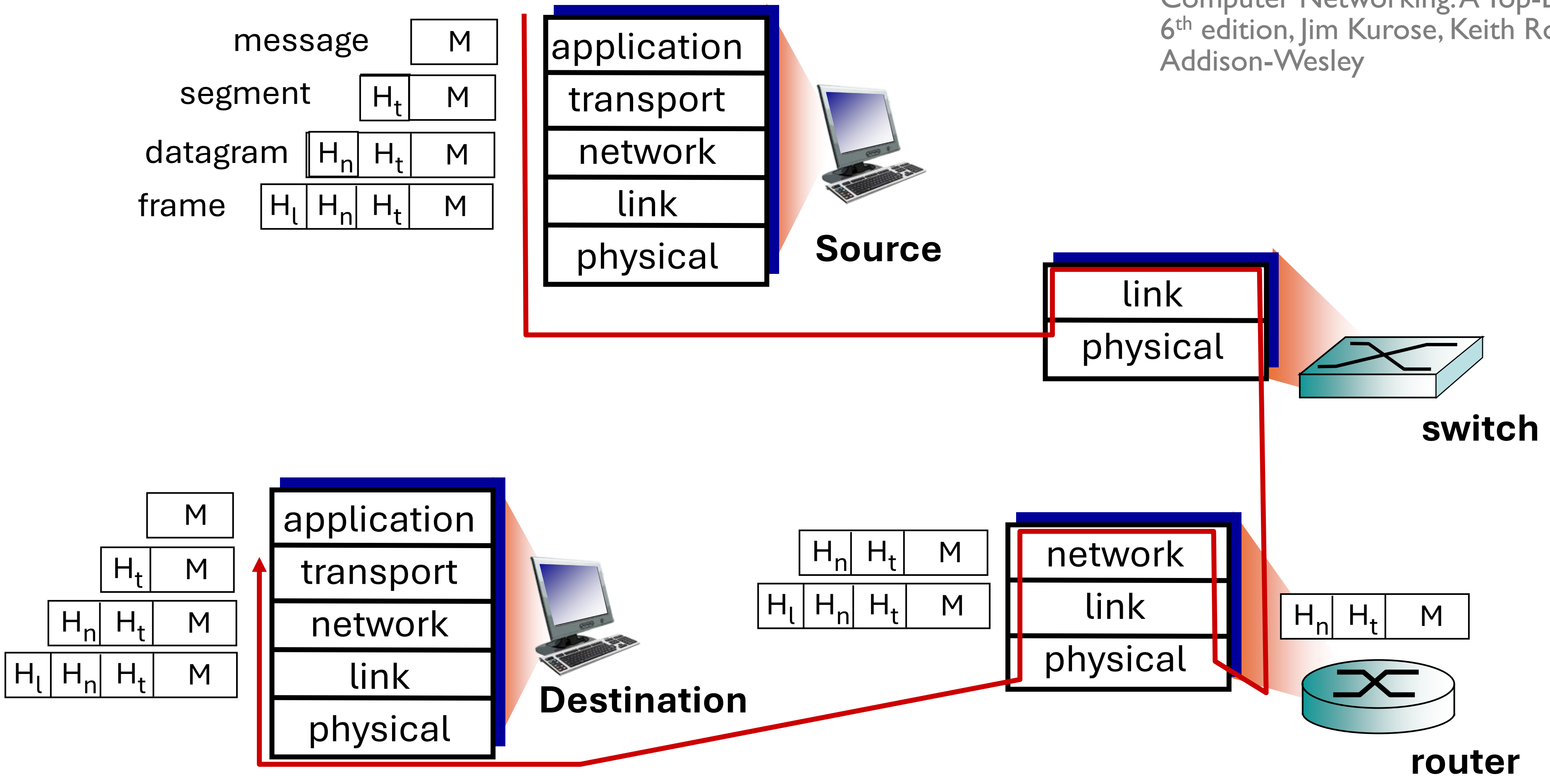
- ✓Layered communication – OSI, TCP/IP
- ✓ Routing
- ✓ Addressing (and subnet design)
- ✓ Transport layer protocols
- ❑ Reliable transport
- ❑ Congestion control

1. Bridge-in


7 Application
6 Presentation
5 Session
4 Transport
3 Network
2 Data Link
1 Physical
Application
(Host-to-Host) Transport
Internet
Network Interface
(Hardware)
OSI Model
TCP/IP Model

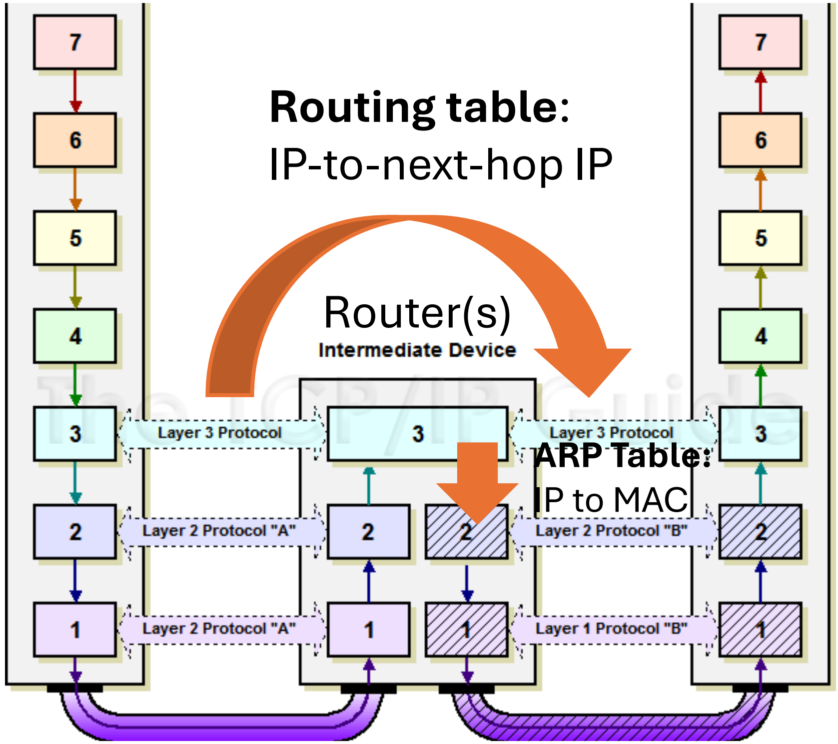

Encapsulate
Decapsulate
7
6
5
4
3
2
1
L7H Data
L6H L7H Data
L5H L6H L7H Data
L4H L5H L6H Data
L3H L4H L5H Data
L2H L3H L4H L5H Data L2F
1 0 0 1 0 1 1 0 1 1 0 0 1 0 1 1

1. Bridge-in



Computer Networking: A Top-Down Approach
6th edition, Jim Kurose, Keith Ross
Addison-Wesley

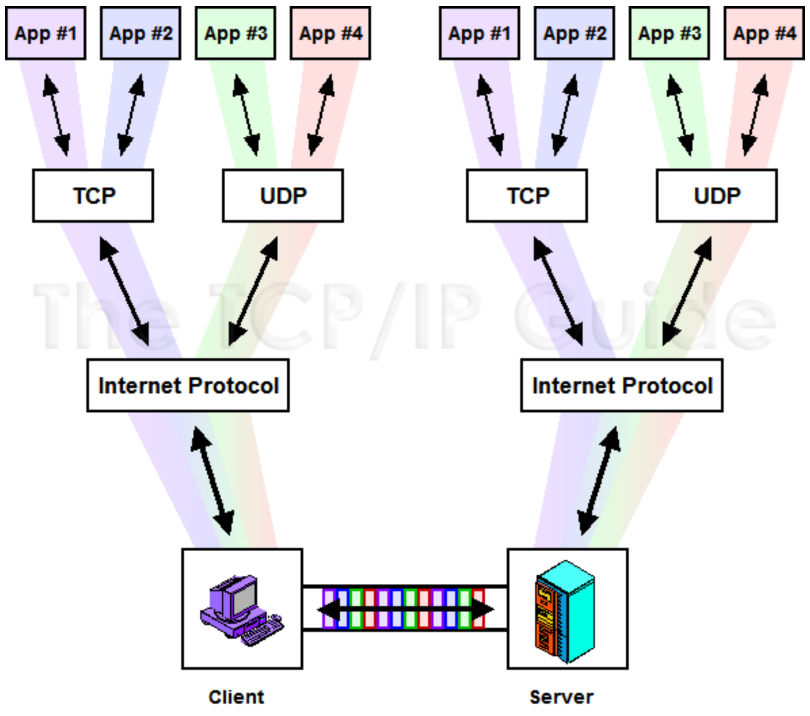
1. Bridge-in




```
C:\Users\micha>route print

IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway
          0.0.0.0          0.0.0.0     192.168.50.1
        127.0.0.0        255.0.0.0         On-link
        127.0.0.1  255.255.255.255         On-link
  127.255.255.255  255.255.255.255         On-link
     192.168.50.0    255.255.255.0         On-link
```

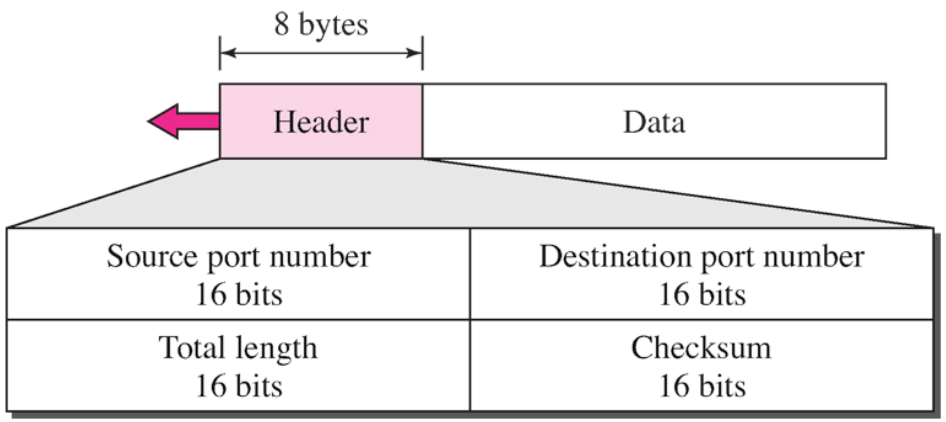
```
C:\Users\micha>arp -a

Interface: 192.168.52.1 --- 0x3
  Internet Address      Physical Address      Type
  192.168.52.255        ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
```

1. Bridge-in

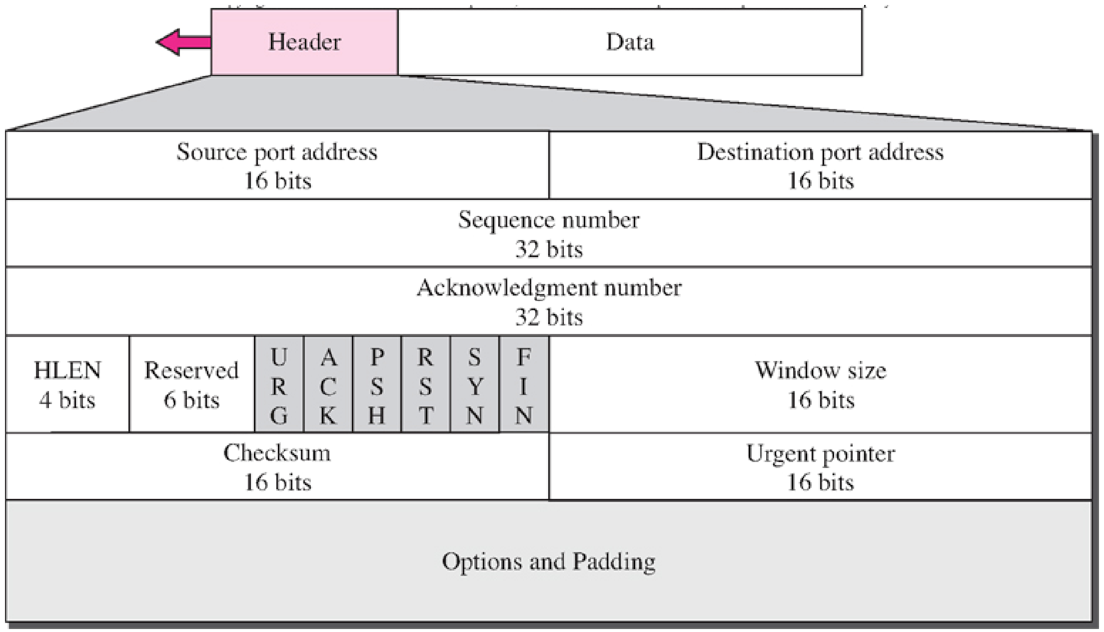


$$\underbrace{\underbrace{\textbf{192.168.1.2}}_{\text{IP address}}\ \textbf{at}\ \underbrace{\textbf{port 53}}_{\text{Port \#}}\ \textbf{using}\ \underbrace{\textbf{TCP}}}_{\text{Socket}}$$

1. Bridge-in



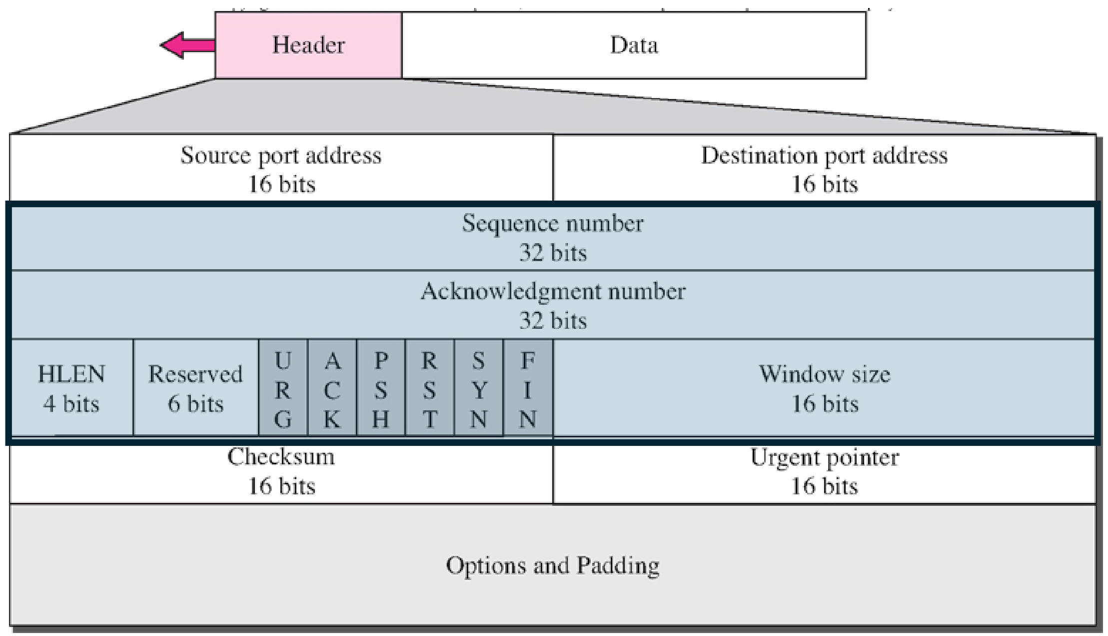
UDP packet structure

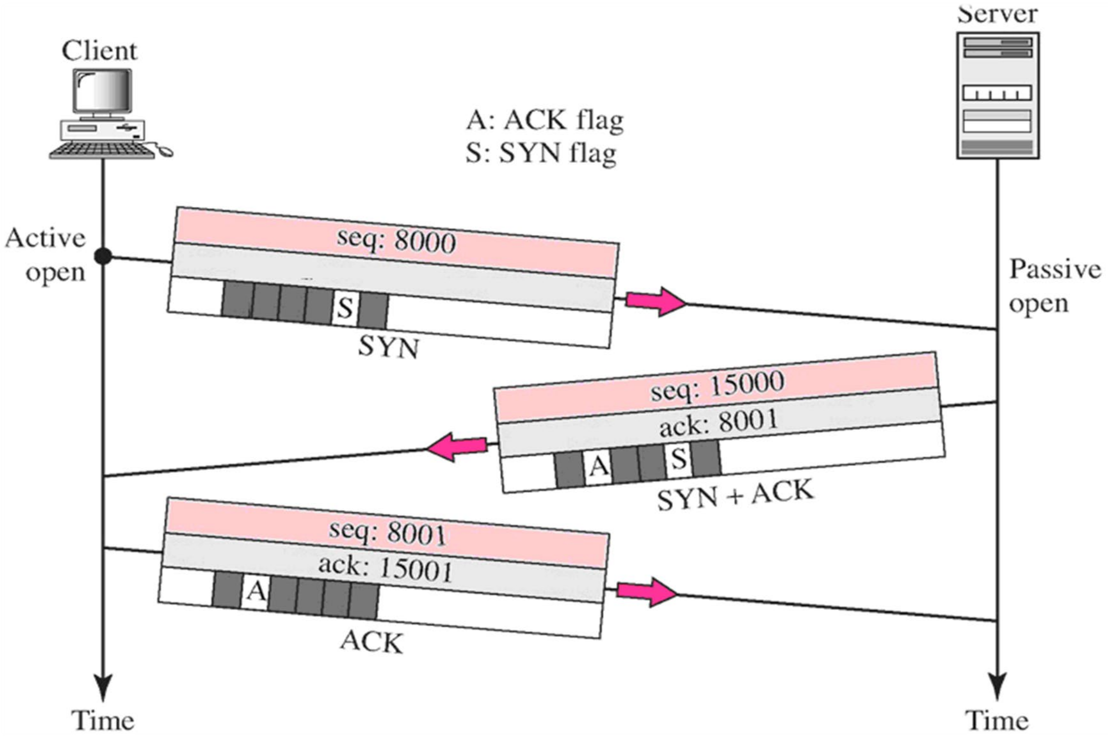


TCP packet structure

Data Communications and Networking, By Forouzan, McGraw-Hill

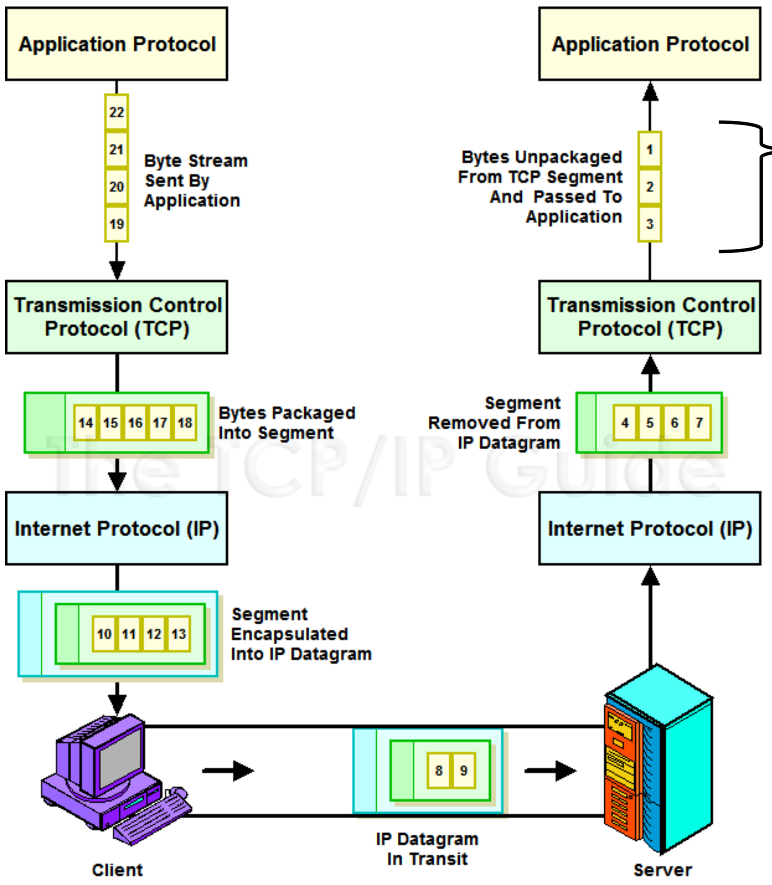
1. Bridge-in





Connection-oriented transport

1. Bridge-in



Reordered Based on Sequence number in TCP header

tcpipguide.com

In-sequence delivery

1. Bridge-in

If your network is not reliable, what can happen to your packets?

Loss

Corruption (bit error)

What can you do when this happens?

Retransmit them

1. Reliable Transport

- Reliable data transport is the ability to recover from packet
  - Loss
  - Corruption (bit error)

- TCP achieves this by **retransmitting** these packets
- This is a type of **Automatic Repeat Request (ARQ)** scheme

1. Reliable Transport

- But how can we keep track of the packets that got lost?

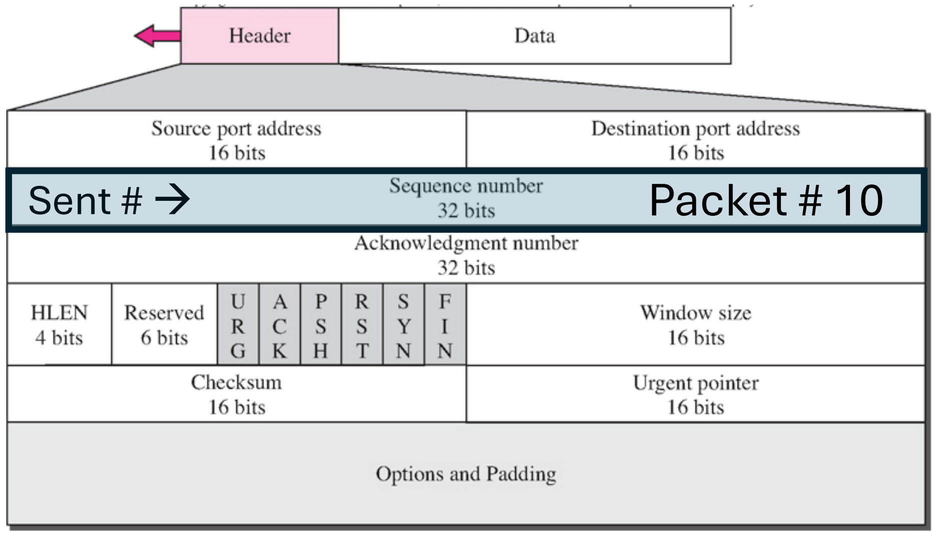
Number them

- But how will we know they got lost?

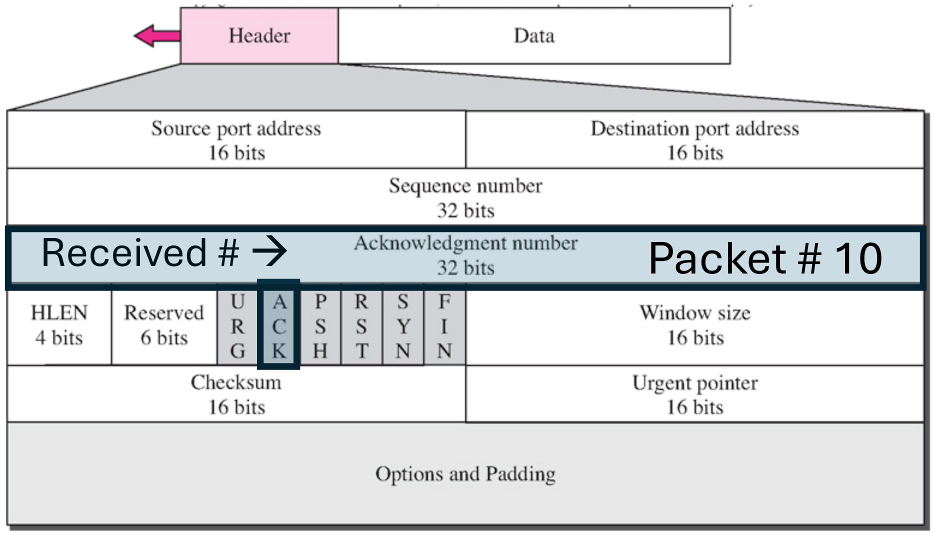
Ask the receiver

1. Reliable Transport

What numbers can I use?



Transmitters sent packet



Receiver's response (to transmitter)

Data Communications and Networking, By Forouzan, McGraw-Hill

1. Reliable Transport

OK, let's see one way this could work …

1. Send message to recipient
2. **Stop-and-wait** for acknowledgement (ACK) from receiver
3. Send next packet on successful ACK

What is wrong with this?

Waiting period slows throughput

1. Reliable Transport

How could we improve throughput?

Send a group of packets at a time

So, let's allow up to N packets to be sent before requiring an ACK

1. Reliable Transport

What can we do when some of the N packets are lost?

Resend all N packets

(Go-back-to-N retransmission scheme)

What's wrong with this?

Resends even the successful packets → wastes bandwidth

1. Reliable Transport

Can we do better?

Resend *only* the lost packets

Can we actually do this?

Yes, but this is a TCP enhancement

Selective Acknowledgement (SACK), will be discussed later in the lecture

1. Reliable Transport

So, the transmitter gets an ACK for successful packets, but how does it know that a packet was lost or corrupted in the first place?

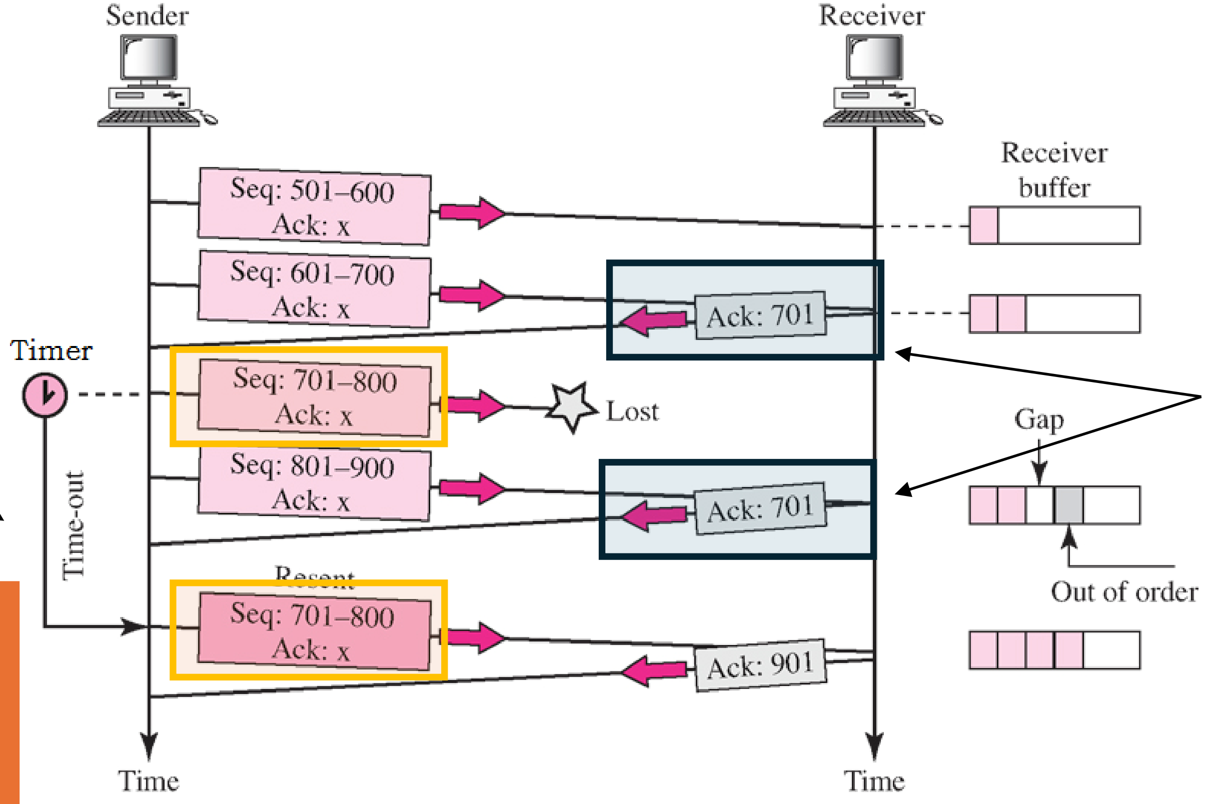
1. Reliable Transport

Retransmission Type 1:

Time-out

Counter reaches 0 with no ACK

On transmission the sender starts count-down timer, if it reaches zero it *assumes* the packet was lost and retransmits



Data Communications and Networking, By Forouzan, McGraw-Hill

Retransmission type 2:

Fast Retransmission

after duplicate ACK number

If transmitter receives the same ACK number more than once it *assumes* the most recent segment was lost and retransmits

2. Congestion Control

Flow control

1. Receiver alerts sender about the **capacity in its buffer**
2. Sender adjusts Tx rate accordingly

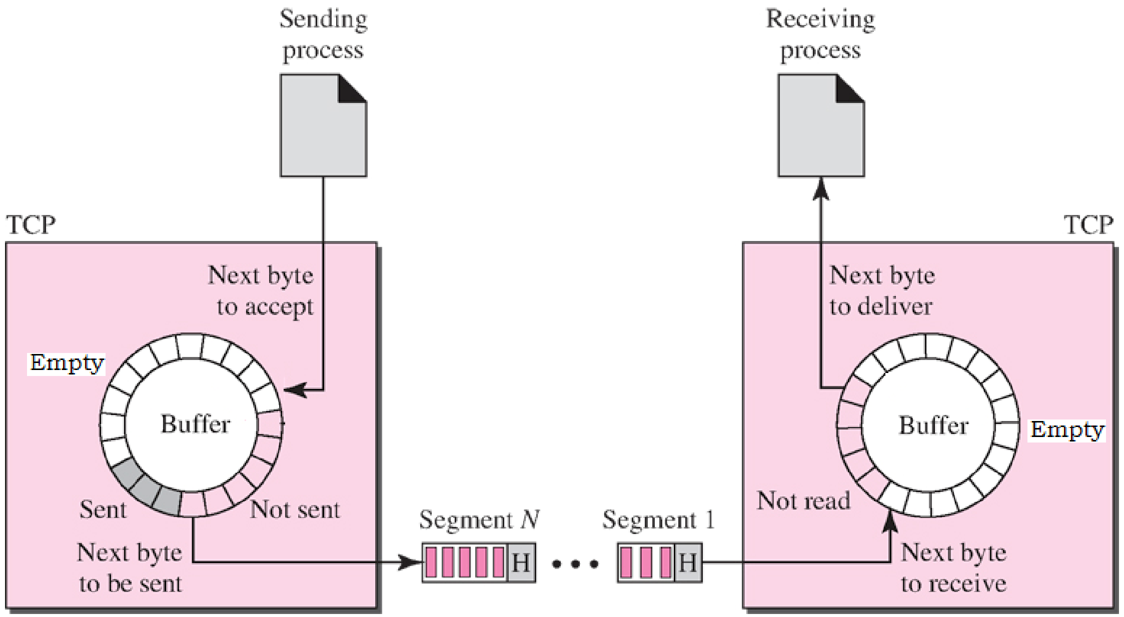
Congestion control

1. Intermediate routers to **drop packets** on congestion
2. Sender monitors for lost packets and adjusts Tx rate

2. Congestion Control

- Why do we need flow control?
- If the inflow rate is higher than the outflow rate the buffer will *overflow*.
- The capacity of the buffer is called the **send window**

Sender and receiver TCP ring buffers

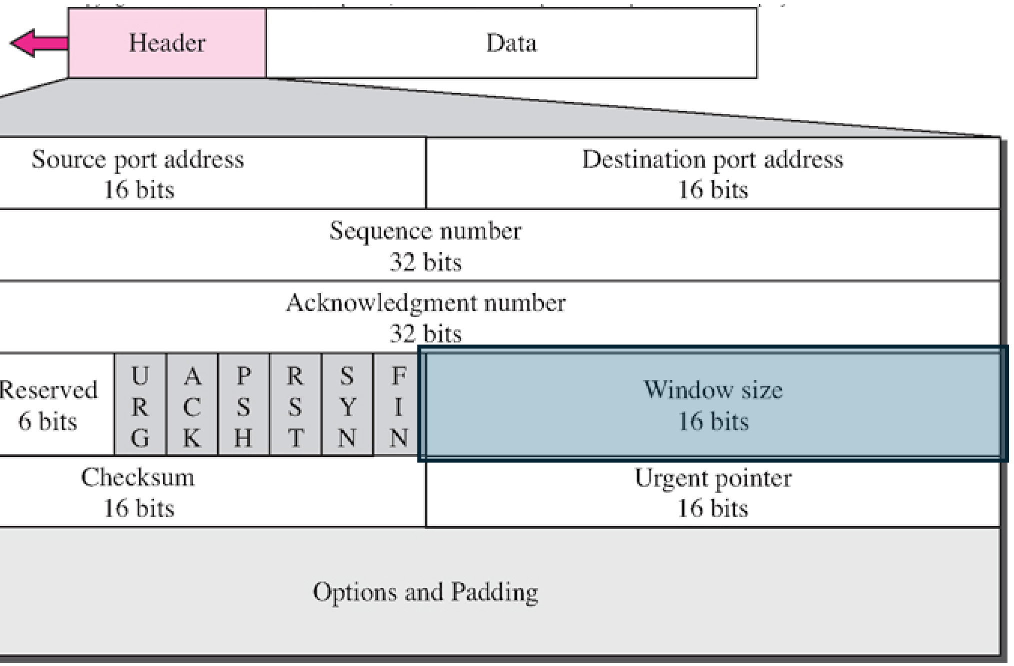


Data Communications and Networking, By Forouzan, McGraw-Hill

2. Congestion Control

How does the receiver tell the sender how big its buffer (send window) is?

TCP ACK packet to sender

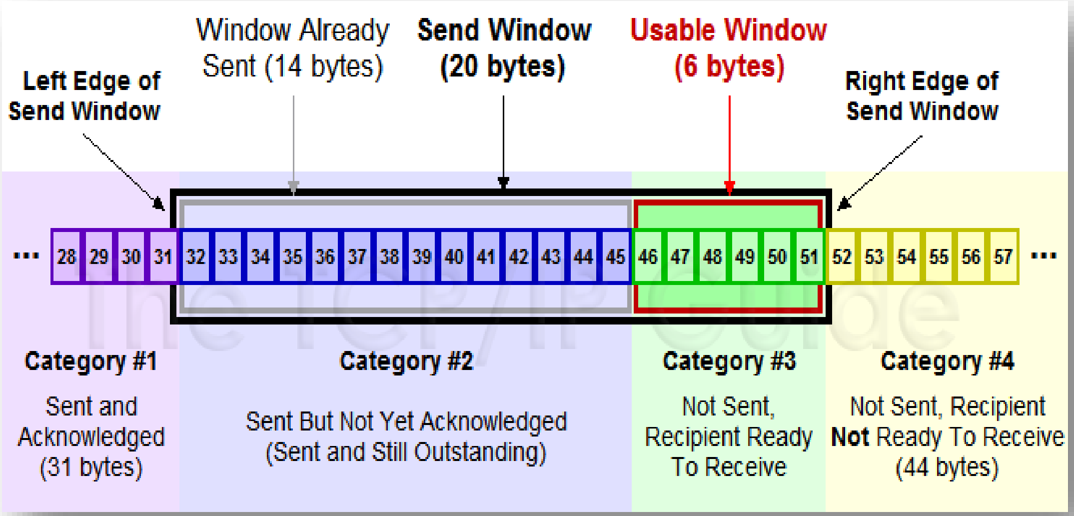


Data Communications and Networking, By Forouzan, McGraw-Hill

2. Congestion Control

But if N packets are sent at once, how does the transmitter know how much usable window room it still has left?

Keep track of sent (but not yet acknowledged) packets



$$\begin{bmatrix} Usable \\ Window \end{bmatrix} = \begin{bmatrix} Send \\ Window \end{bmatrix} - \begin{bmatrix} Window\ Already \\ Sent, no\ ACK \end{bmatrix}$$

2. Congestion Control

Flow control ✓

Congestion control (next)

2. Congestion Control

Why do we need congestion control?

Congestion Collapse

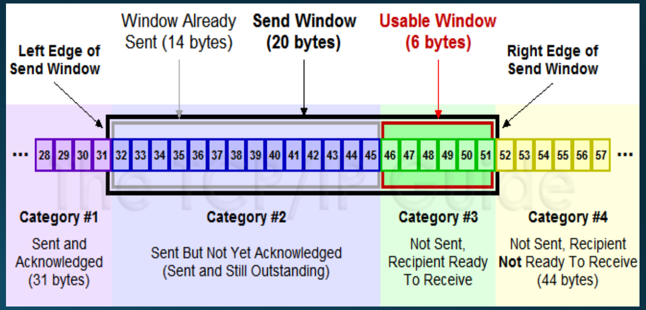
Want to prevent this

- On congestion routers drop packets
- Tx timeout expires or duplicate ACK
- Tx retransmits dropped packet → More congestion

Congestion → Drop → Retrans. ⊕ → Collapse

Not good, positive feedback loop

2. Congestion Control



How can we avoid congestion collapse?

Negative feedback

Need to control retransmission on packet drop

How?

Further restrict the Send Window on packet drop

$$Send\ Window = RcvWin$$

Currently the Send Window is only determined by the receiver's window

2. Congestion Control – TCP Tahoe

How can we restrict the Send Window?

Define a Congestion Window ($CongWin$) so that the Send Window is:

$$Send\ Window = min\ (RcvWin, CongWin)$$

If we assume that most of the time $RcvWin \gg CongWin$ *then:*

$$Send\ Window = CongWin$$

Send Window is mainly determined by the congestion window

2. Congestion Control – TCP Tahoe

- $CongWin$ is an integer multiple (w) of the max segment size (MSS)

$$CongWin = w \times MSS$$

536 bytes
by default

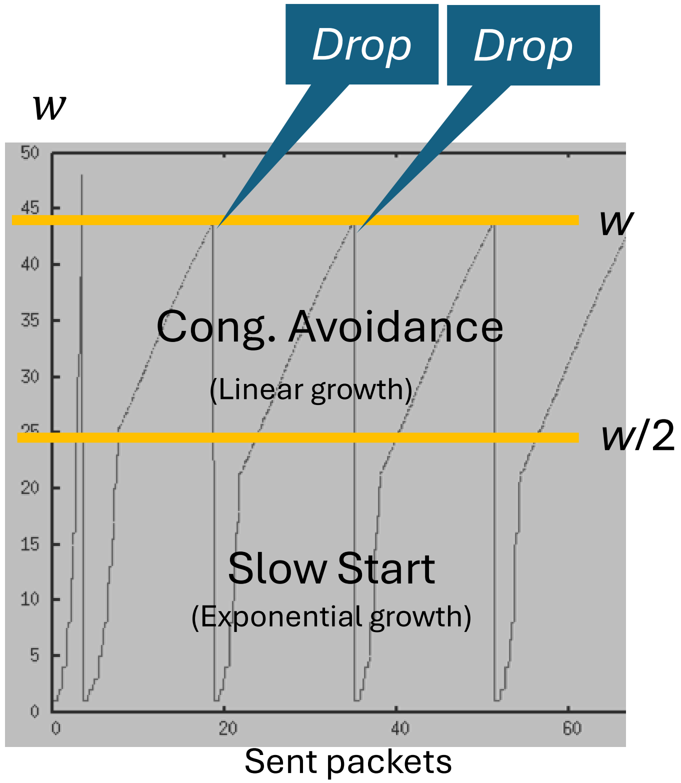
- The main variable we will change is w
- This congestion control method is known as TCP Tahoe (Slow Start) Algorithm

2. Congestion Control

$$CongWin = w \times MSS$$

Phase	*w*	threshold
Slow start (*w* < threshold)	*w = 1, then* *w* = 2 × *w* (exponential growth)	threshold initialized to some value on first run, then reset in loss phase
Congestion Avoidance (*w* >= threshold)	*w* = *w* + 1 (linear growth)	
Loss (Timeout expires, no ACK – packet **drop**)	*w* = 1 (return to slow start)	threshold = *w* / 2

w increments with every sent packet



http://www.cs.emory.edu/~cheung/Courses/455/Syllabus/A1-congestion/tcp2.html

2. Congestion Control

Great, so we can prevent congestion collapse, but how does this affect our transmission rate?

$$Send\ Window = CongWin = w \times MSS \quad [bytes]$$

Unit analysis: Transmission rate has units of [bytes / s], so we need something for the denominator in seconds

2. Congestion Control

If the transmitter sends all *w* segments in **one burst**, how long must it wait before it can send more?

- It needs to wait for an ACK
- The burst must reach the receiver, and the ACK must travel back
- This is called the Round-Trip Time (RTT)
- So, we have $w \times MSS$ *bytes* sent every RTT *seconds* for an instantaneous bit rate of

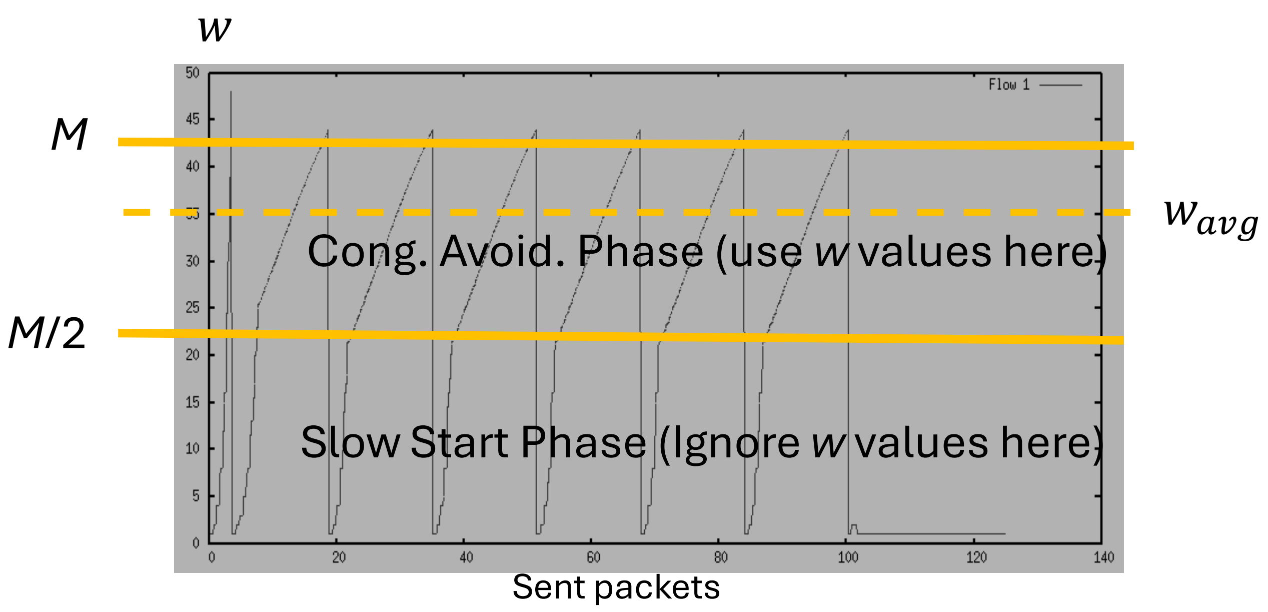
$$R(w) = \frac{w \times MSS}{RTT} \quad [bytes/s]$$

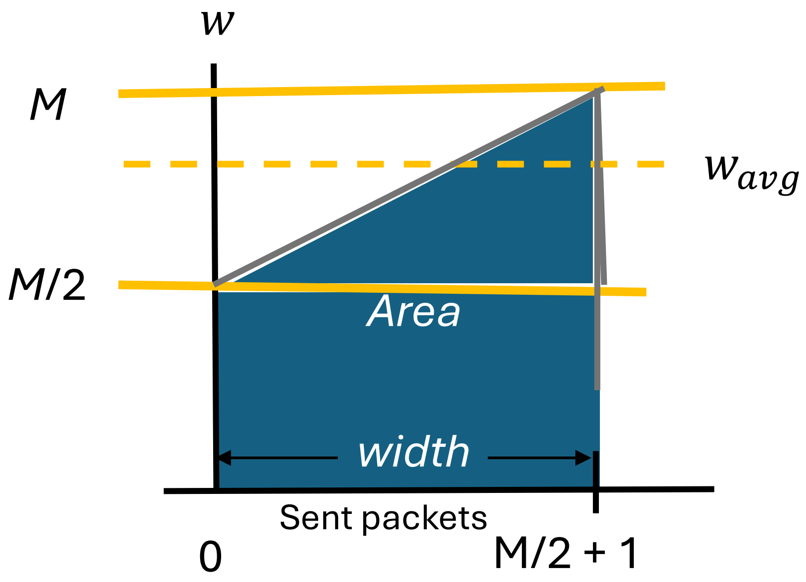
But what we really care about is the *average* bit rate

Need to find the average value of *w* (w_{avg})

2. Congestion Control

- We can *approximate* w_{avg} by ignoring the slow start phase and considering only the linear phase where w grows from M/2 to M and then returns to 1 in M/2 + 1 steps

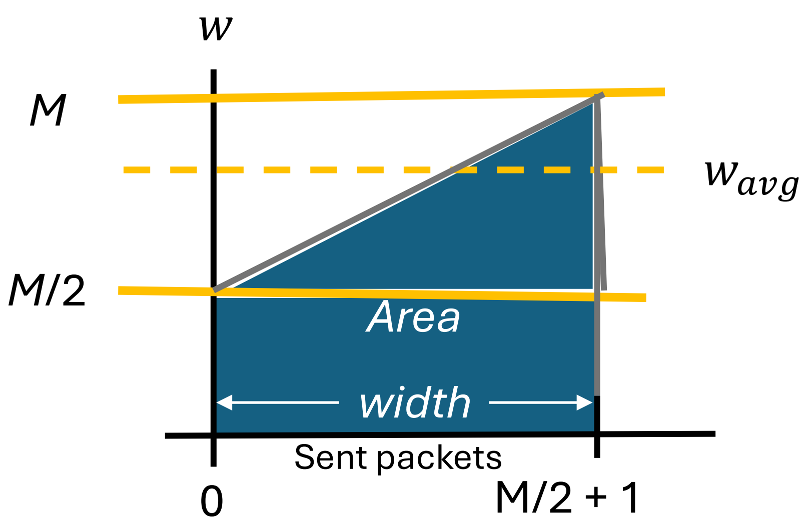




http://www.cs.emory.edu/~cheung/Courses/455/Syllabus/A1-congestion/tcp2.html

2. Congestion Control

- Approximate w_{avg} in terms of M via the **trapezoid rule**



$$Area \approx (last - first) \times \frac{1}{2}(f(first) + f(last))$$

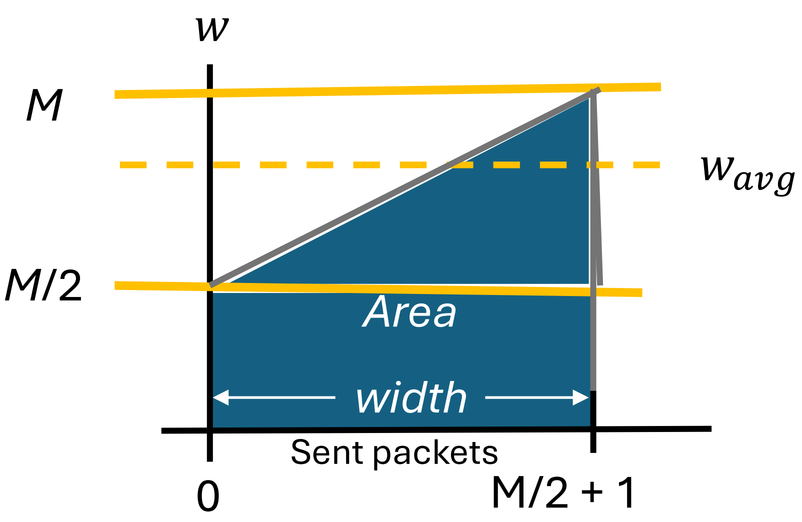
$$w_{avg} \approx \frac{Area}{width} = \frac{\cancel{\left(\frac{M}{2} + 1 - 0\right)} \times \frac{1}{2}\left(\frac{M}{2} + M\right)}{\cancel{\left(\frac{M}{2} + 1 - 0\right)}} = \frac{3M}{4}$$

- And the Average Transmission Rate is:

$$\boxed{R(w_{avg}) \approx \frac{w_{avg} \times MSS}{RTT} = \frac{3M}{4} \times \frac{MSS}{RTT}}$$

2. Congestion Control

- Alt.: Approximate using Gauss's formula for the **sum of consecutive integers**



$$Area \approx \frac{n}{2}(first + last), \qquad n\ is\ \#\ of\ integers\ (i.e., M/2+1)$$

$$w_{avg} \approx \frac{Area}{width} = \frac{\frac{\cancel{(M/2+1)}}{2}(M/2+M)}{\cancel{(M/2+1)}} = \frac{3M}{4}$$

- And the Average Transmission Rate is:

$$\boxed{R(w_{avg}) \approx \frac{w_{avg} \times MSS}{RTT} = \frac{3M}{4} \times \frac{MSS}{RTT}}$$

Summary

- Reliable transport is the ability to recover from packet loss & corruption.
- TCP achieves this by retransmitting these packets in an ARQ scheme that numbers the packets and asks the receiver to acknowledge (ACK) them.
- TCP implements flow and congestion control techniques to prevent overflow of the receiver's buffer and prevent congestion collapse on the network.
- Flow and congestion control techniques regulate the transmission rate

3. Flipped Classroom

In this active learning exercise, the class is divided into 4 groups to present one of the TCP topics (below):

1. Explicit Congestion Notification (ECN)
2. Selective Acknowledgement (SACK)
3. Small Packet Problem (solution – Nagle's Algorithm)
4. Silly Window Syndrome (solution – Clark's Algorithm)

Resources include

- Editable flipped classroom slides (originals follow)
- Section 3 of the notes that accompany this lecture

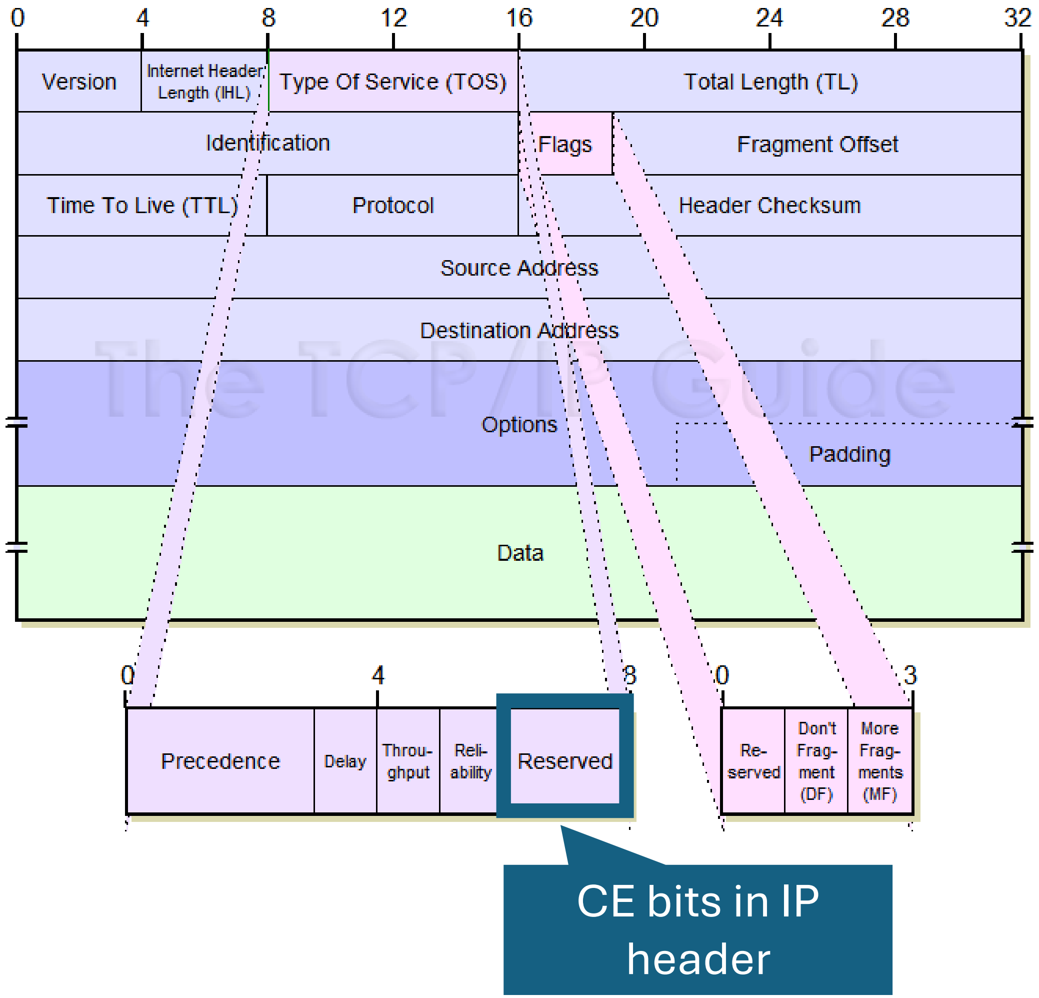
3. Explicit Congestion Notification (ECN)

- A TCP protocol enhancement that enables congestion reporting
- Source, destination and intermediate routers to be ECN aware
- Instead of **dropping** a packet on congestion an ECN aware router will **set a flag** in its **IP** header to notify the **receivers TCP**
- Receivers TCP will signal the senders TCP to slow its Tx rate

3. Explicit Congestion Notification (ECN)

- An ECN aware **router** signals congestion using two **reserved bits** in the **type-of-service** field of the IP header

- Routers operate at the ‘internet’ layer so they can only see and modify the IP header



3. Explicit Congestion Notification (ECN)

- The two CE bits can be set

00 – ECN not implemented
10 – ECN Capable Transport, ECT(0)
01 – ECN Capable Transport, ECT(1)
11 – Congestion Encountered, CE

Endpoints negotiate which bit is used by either side during the TCP handshake

Congestion signaled by setting the other reserved bit to '1' so the field becomes '11'

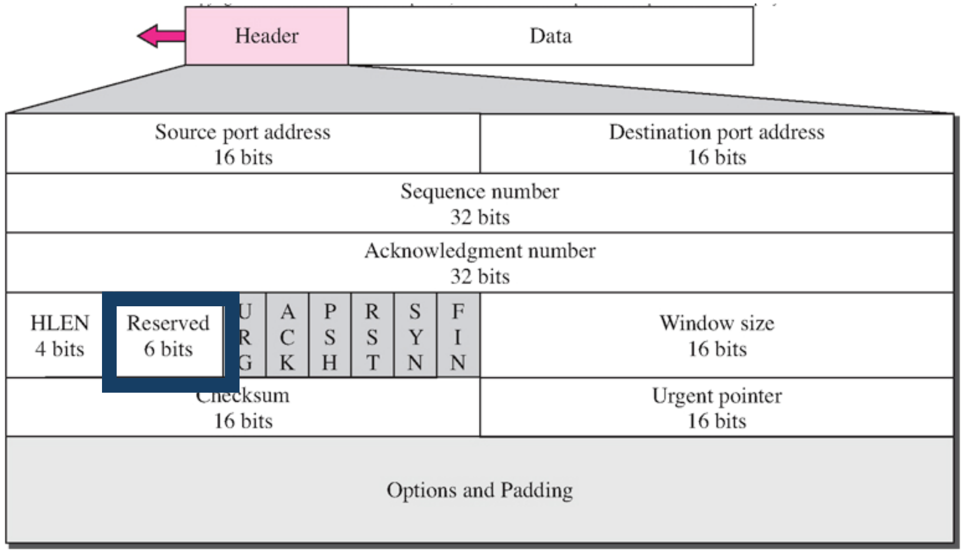
Congestion can only be handled by TCP by reducing the Send (Congestion) Window

3. Explicit Congestion Notification (ECN)

On Congestion Encountered, CE bits ’11’ must be handled by **TCP**

- The receivers IP layer notifies its TCP layer

- Receiver’s TCP signals congestion to sender’s TCP by using bit 5 and 6 of the **reserved bits** of the TCP header
  - **Bit 6** signals congestion
  - **Bit 5** tells sender to reduce its Congestion Window



3. Selective Acknowledgment (SACK)

Problem

- Traditional TCP acknowledges the value of the highest byte received before a lost segment.
- Any lost packet requires retransmissions of all segments up to that point (wastes bandwidth).

Solution:

- Selective Acknowledgement (SACK) allows a receiver to inform the sender about specific segments in the send window that have been successfully received, so only the lost packets are retransmitted.

3. Selective Acknowledgment (SACK)

Example:

Suppose a sender transmits data in the byte range 0–10,000, and packets corresponding to byte ranges 2,000–3,000 and 6,001–7,000 are lost.

Bytes	No SACK	SACK
0 – 1,999	ACK	ACK
2,000 – 3,000 (lost)	Retransmit	Retransmit
3,001 – 6,000	Retransmit	ACK
6,001 – 7,000 (lost)	Retransmit	Retransmit
7001 – 10,000	Retransmit	ACK

3. Small packet – Nagle's Algorithm

Problem:

- Sender transmits TCP segments with large header compared to payload
    - wastes bandwidth (high overhead), creates congestion (many small packets)
    - e.g. Telnet apps that send one TCP segment per keystroke

Solution

- Nagle's Algorithm: Sender side strategy
    1. Send first byte and *accumulate new bytes while waiting for ACK*
    2. On ACK send accumulated bytes
    3. Repeat
- Trade-off: Increased latency for lower overhead and congestion

3. Silly Window Syndrome – Clark's Algorithm

Problem:

- A slow **receiver** sends ACK with a small window size
- Sender can only send small TCP segment → Small Packet Problem (again)

Solution:

- Clark's Algorithm: **Receiver** side strategy

    1. *Wait for a minimum window size to be available in Rx buffer before sending ACK*
    2. *Process data* (freeing up buffer space) while waiting

- Trade-off: Increased *latency* for lower overhead and *congestion*

References

- INFO73180 – Data Communications and Networks, Michael Galle, Lecture slides and handouts
- Computer Networks – A Top-Down Approach, Kurose and Ross
- CSC358 – Introduction to Computer Networks, Peter Marbach lectures, https://www.cs.toronto.edu/~marbach/csc358_F19.html, accessed Dec. 2024.
- http://www.tcpipguide.com/
- Data Communications and Networking, Forouzan, McGraw-Hill