

# **CSE3231 Final Exam Study Guide**

**Grant Butler, Tyler Zars**

[gbutler2020@my.fit.edu](mailto:gbutler2020@my.fit.edu) | [tzars2019@my.fit.edu](mailto:tzars2019@my.fit.edu)

# Table of Contents

Sections

Transport Layer

# Transport Layer

## UDP and TCP

Differences:

### TCP (Transport Control Protocol)

Advantages

- Connection oriented transport
  - Sender must establish connection before transmission
  - Sender notified of delivery or of error
- **Byte-stream** service
  - Data transmission and reception are similar to file I/O
- Reliable delivery
  - Guarantees packets are assembled **in order**

Connection Management

- Connection
  - Three way connection increases probability that both endpoints know that connection was accepted.
- Termination
  - Four way handshake requires two FIN and two ACK to complete. Ensures proper termination of connection occurs.

### UDP (User Datagram Protocol)

- Connectionless
  - Destination address and port number are added to transport segment's header and the segment is sent to the destination.
  - No confirmation of error or delivery (ACK) is returned.
    - Unreliable because of no ACK
- Advantages
  - Applications pass **directly** to transport layer.
  - Data is transmitted **immediately**. Will either reach the receiver or not at all.
  - Less to manage:
    - No **congestion-control** or retransmission mechanisms.
    - No Connection Establishment
    - No connection state
    - Results in small packet header
  - Messages can be sent in **broadcast** or **multicast** mode.
    - one to multiple receivers (multicast) or all nodes (broadcast)

## TCP Header Fields

Field	Size (bits)	Description
Source Port	16	Identifies source port number (sender's TCP port)
Destination Port	16	Identifies destination port number (receiving port )
Sequence Number	32	Used to number TCP segments. If SYN = 0, each byte is assigned a sequence number. *
Acknowledgement Number	32	Indicates next sequence number that sending device is expecting.
Offset (Header Length)	4	Shows number of 32 bit words in header. Minimum size of 5 words (0101 in binary).
Reserved	4 (6)	Always set to 0.
TCP Flags	8	Flags are: URG, ACK, PSH, RST, SYN, FIN
Window	16	The size of the receive window, which is the number of bytes beyond sequence number in Acknowledgement field that the receiver is willing to receive.
Checksum	16	Used for error checking of header and data.
Urgent Pointer	16	Shows the end of urgent data so interrupted data streams can be continued. +
TCP Options	variable	0 → End of Options List, 1 → No Operations (NOP, Pad), 2 → Maximum segment size, 3 → Window Scale, 4 → Selective ACK ok, 8 → Timestamp

---

**Size:** 4 bits → nibble, 8 bits → byte, 32 bits → word

**Sequence Number:** If SYN = 1, then this is the initial sequence number. The sequence number of the first byte of data will then be this number + 1. i.e.: Let first byte of data have this be 300. Then if a packet has 10 bytes, then the next packet sent will have a sequence number of  $300 + 10 + 1 = 311$ .

**Urgent Pointer:** When URG is set, the data is given priority.

## TCP Flags Explained

Flag	Description
URG	Urgent Pointer.
ACK	Acknowledgement
PSH	Push function. TCP allows an application to specify that data is to be pushed immediately.
RST	Reset connection. Receiver must respond immediately terminating the connection. Transfer of data ceases, so data in transit is lost. Used for abnormal close of TCP connection, unlike FIN.
SYN	Indicates synchronized sequence numbers. Source is beginning a new sequence.
FIN	Set when no more data is to come from sender. Used for good closing of TCP connection, unlike RST.

# TCP Flow Control

TCP needs to control amount of data a sender transmits to avoid overwhelming the receiver.

## Congestion Control vs Flow Control

### Congestion Control

- focuses on preventing too much data in **network**.
- uses Retransmission Timeout (RTO) and Round Trip Time (RTT)
  - RTT is different for each path a packet takes.
- A router might only be able to handle 100 Mb/s total, but two senders could send more than that.

### Flow Control

- Tries to prevent senders from overrunning **capacity of receivers**.
  - Can't prevent congestion at routers.
- Uses sliding window to control traffic in transit.
  - Uses **AdvertisedWindow** to indicate how much data it can handle.
  - Measures in bytes, not packets.
  - Limits how many unacknowledged bytes can be in transit at a time.
  - TCP vs Data-Link Sliding Windows
    - Data-Link layer controls transmission of frames over links between adjacent nodes.
      - one sender at a time
      - always arrive in order sent (unless frames are lost)
    - TCP deals with end-to-end flow
      - each receiver can have multiple senders
      - each packet can follow a different path
  - Header uses these fields to manage flow control:
    - **SequenceNum**
    - **Acknowledgement**
    - **AdvertisedWindow**

# TCP Congestion Control: Additive Increase and Multiplicative Decrease (AIMD)

TCP Source sets the *CongestionWindow* based on level of congestion it *perceives* in the network.

- Involves **decreasing** congestion window when congestion goes up and **increasing** the congestion window when level of congestion goes down.
- This is called **Additive Increase / Multiplicative Decrease** (AIMD).

## Additive Increase

- Every successful send from source that is a *CongestionWindow*'s worth of packets adds the equivalent of 1 to *CongestionWindow*.
  - Success is measured as one ACK per RTT.
- Increase is slower than decrease and avoids too rapid an increase in transmission rate.

## Multiplicative Decrease

- Easier to understand in terms of packets, despite **CongestionWindow** being measured in bytes.
  - e.g.:
    - *CongestionWindow* is 16 packets
    - If a loss is detected, *CongestionWindow* is set to 8.
    - Additional losses go → 4, 2, 1.

## Slow Start

1. Source starts *CongestionWindow* at one packet.
2. Sends one packet.
3. ACK arrives → *CongestionWindow* += 1.
4. Two packets are sent.
5. Two ACKs → *CongestionWindow* += 2.

Trend: TCP effectively **doubles** every RTT.

1. **Slow Start** begins by doubling *CongestionWindow* size.
2. When threshold is reached, switches to *additive increase*.
3. When packet is lost, *CongestionWindow* goes to 1 and slow start repeats.

# TCP Timeout and RTT

## Timeout

Timeout period must be long enough to allow longer paths. If a packet is lost, multiple packets can be sent out before timeout expires. Receiver can't ACK because missing packet caused a gap in *SequenceNum*. Sender can reach *CongestionWindow* limit while waiting for timeout.

## Fast Retransmission and *Duplicate Acknowledgements*

Receiver sends ACK for later packets, but with ACK number of last packet before lost packet--i.e. *duplicate acknowledgements*.

- Tells sender that at least one packet hasn't arrived, but later ACK's indicate some later packets arrived.
- Duplicate ACK number tells sender which packet wasn't received.

Sender can *resend* missing packet without waiting for timeout to expire. This is called *fast retransmission* and can trigger transmission of lost packets sooner than regular timeout.

- Not triggered until *three duplicate ACK's* arrive.
- Sender knows packet was lost, and halves *slow start threshold* and goes into slow start.

## Fast Recovery

- Lost packet decreases *CongestionWindow* to one and starts slow start.
- Fast retransmission signals congestion, and instead of lower *CongestionWindow*, fast recovery uses ACKs in transit to trigger sending of new packets.
- Removes slow start phase when fast retransmit detects lost packet.



## Round Trip Time (RTT)

Retransmission Timeout (RTO) is based on Round Trip Time (RTT) for a given connection.

- At connection, sender and receiver determine RTT and sender uses that for RTO.
- Sender calculates an average RTT to deal with delays.

Determining RTT:

- Sender and receiver both need RTT, so they put timestamps in options field to track send and receive times.
- A *Smoothed RTT* (SRTT) is calculated based on the SRTT averaged over time and the most recent RTT.

$$SRTT = \alpha * SRTT + (1 - \alpha)RTT \text{ where } \alpha = 0.9$$

- Smoothed RTT calculation was revised to include *variance* in RTT.
  - **Variance** measures how much RTT changes over time.

$$VarRTT = \beta * VarRTT + (1 - \beta) * |SRTT - RTT| \text{ where } \beta = 0.75$$

- Retransmission Timeout (RTO) is calculated as follows:

$$RTO = SRTT + 4^{++} * VarRTT$$

<sup>++</sup>(multiplying by 4 is based on experimentation)

## IP Checksum

Header Checksum: 16 bits

- A checksum on the header only. Since some header fields change, it is *recomputed* and verified each time the header is processed.
- Algorithm:
  - 16 bit one's complement of the one's complement sum of all 16 bit words in the header. The value of the checksum field is zero.

IP Checksum Example:

1. break sequence into 16-bit words
2. Add 16-bit values. Each carry-out produced is added to the LSb.
3. Invert all bits to get one's complement.

Header to check:

```
1000 0110 0101 1110
1010 1100 0110 0000
0111 0001 0010 1010
1000 0001 1011 0101
```

Add 16-bit values 2 at a time and convert to one's complement:

1000 0110 0101 1110	<i>first val</i>
+ 1010 1100 0110 0000	<i>second val</i>
<hr/>	
1 0011 0010 1011 1110	<i>carry - out</i>
+ 0000 0000 0000 0001	<i>add to LSb</i>
<hr/>	
0011 0010 11011 1111	
+ 0111 0001 0010 1010	<i>third val</i>
<hr/>	
0 1010 0011 1110 1001	<i>no carry - out</i>
+ 1000 0001 1011 0101	<i>fourthval</i>
<hr/>	
1 0010 0101 1001 1110	<i>carry - out</i>
+ 0000 0000 0000 0001	<i>add to LSb</i>
<hr/>	
0010 0101 1001 1111	<i>one's complement sum</i>
	<i>flip bits</i>
1101 1010 0110 0000	<i>one's complement</i>

Thus, the 16 bit checksum is 1101 1010 0110 0000 .

# Applications