CSE3231 Final Exam Study Guide

Grant Butler, Tyler Zars

gbutler2020@my.fit.edu | tzars2019@my.fit.edu

Table of Contents

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
  - Garuntees packets are assembled in order

Connection Management

- Connection
  - Three way connection increases probability that both endpoints know that connection was acccepted.
- 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 packed was lost, and halves *slow start threshold* and goes into slow start.

Fast Recovery

- Lost packed 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 calulates 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:

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

++(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:

$$
\begin{array}{c}
1000\ 0110\ 0101\ 1110 \\
1010\ 1100\ 0110\ 0000 \\
0111\ 0001\ 0010\ 1010 \\
1000\ 0001\ 1011\ 0101
\end{array}
$$

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

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

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

Applications

Application layer interfaces with transport layer, isolating applications from the details of packet delivery. Applications can use either UDP or TCP. Some use UDP but add features like acknowledgements on their own to get reliable delivery without TCP's overhead.

Protocols

Usually, an application that supports interaction and data transfer have a *protocol* for communication between nodes. One may be a *server*, collecting and delivering data, while another may be a client, requesting and providing data. Application Protocols describe how endpoints will interact to accomplish tasks.

Internet RFC (Request For Comment)

A set of application protocols that standardize actions across vendors. Examples include:

- Simple Mail Transfer Protocol (SMTP) *RFC 5321*
- Hypertext Transfer Protocol (HTTP) *RFC 2616/7540*
- File Transfer Protocol (FTP) *RFC 959*

Clear application protocols allow developers to create servers and clients that interact in established and predictable ways.

- Some inconsistencies happen when features are added that are not part of the protocol.

Port Numbers

Servers typically listen on well known ports for connections. A range of UDP and TCP ports are assigned to specific protocols, while others are not reserved.

- Ports 0-1023 are reserved by the Internet Assigned Numbers Authority (IANA).

Common ports and their services:

Service	Port
SSH	22
HTTP	80
NTP (Network Time Protocol)	123
IMAP (email)	143 or 220
LDAP (authentication protocol)	289
HTTPS (using TLS/SSL)	443

Request/Reply Protocols

Messages are transmitted by client and server to manage and exchange data. Often consist of text commands.

- *Stateful* protocols require client and server to keep track of current state of exchange.
- *Stateless* protocols might have a server not keep a record of exchanges and close connection after each message.

Publish-Subscribe Protocol (PubSub)

- **Publishers** make data available for *subscribers* who register to receive types of messages.
  - *Loosely coupled* to subscribers and produce whether or not it is used.
  - Has better scalability because publishers don't manage connections.

Examples:

- Apache Kafka
- Google Cloud Pub/Sub
- Data Distribution Service (DDS)

Message Queueing

Similar to PubSub, these protocols don't require servers to wait for a client to request. Important aspects include:

- A *queue manager* is implemented and announced to users.
- Applications *register* to be notified when messages arrive, and then can download those when ready.
- Applications can add messages to a queue.
- Queueing *decouples* senders and receivers, so senders don't wait before they add to the queue.

Examples:

- Apache ActiveMQ
- Microsoft Message Queueing
- Java Message Service

Peer to Peer

Each node can exchange messages with any other node.

- a node can be a client **and** a server
- data is *decentralized* and passed between peers
- not all nodes have same capabilities, so some may perform special tasks to help other nodes
- peer networks can be *structured* with set topology or *unstructured* and allow rapid changes to adapt

Examples:

- Bitcoin and other Cryptocurrency
- BitTorrent
- Gnutella

HyperText Transfer Protocol (HTTP)

Used for connections between clients (browsers) and servers in the World Wide Web.
Provides request/response interaction between server and multiple clients.

- Web browser acts as *User Agent*
- Communications are based on TCP
- *Stateless* protocol
  - keep-alive feature was added in v1.1
  - retaining state information was solved with variables in messages or web cookies on client's host

Messages:

```
START_LINE <CRLF>
MESSAGE_HEADER <CRLF>
<CRLF>
MESSAGE_BODY <CRLF>
```

`START_LINE` indicates request or response message. Each section ends with CRLF.

Cookies

Coockies are used to support *stateful* client/server interactions

- server sends cookies (state) with response
- client stores them locally
  - client sends cookie with a new request to the server
- server extracts state information from cookie

Examples:

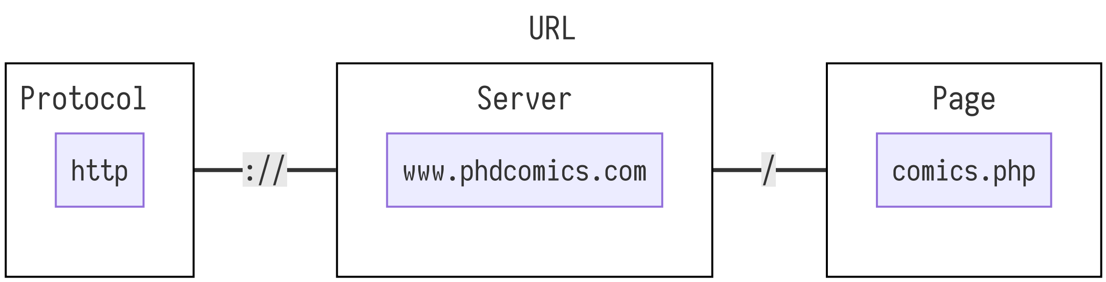
Domain	Path	Content	Expires	Secure
toms-casino.com	/	CustomerID=297793521	15-10-10 17:00	Yes
jills-store.com	/	Cart=1-00501;1-07031;2-13721	11-1-11 14:22	No
aportal.com	/	Prefs=Stk:CSCO+ORCL;Spt:Jets	31-13-20 23:59	No
sneaky.com	/	UserID=4627239101	31-12-19 23:59	No

URLs

Identification of servers and hyperlinks is based on *Uniform Resource Locators* (URLs), which contain information to access a target server and document on that server.

- HTTP headers are text based and use standard header fields to manage connections and data exchange
- HTTP connections are not encrypted, but *HTTPS* creates an encrypted connection before data is exchanged
    - protects data, but does not authenticate users

Example:



- begins with protocol it will connect to
- specifies *domain name* (server) and *specific file* on the server
    - also can specify email accounts, local files, links to FTP servers, and other sources

History:

- links information in documents, presented by **Vannevar Bush** in 1945
- **Ted Nelson** coined hypertext in 1963 and helped create a system with hyperlinks
- **Douglas Engelbart** demonstrated a user interface with different tools and documents in 1968
- **Tim Berners-Lee** created a hypertext sharing system called the *World Wide Web* in 1990

Headers