Transmission Control Protocol (TCP)

Practical 7

UCD School of Computer Science

November 2018

Transmission Control Protocol (TCP) - Introduction

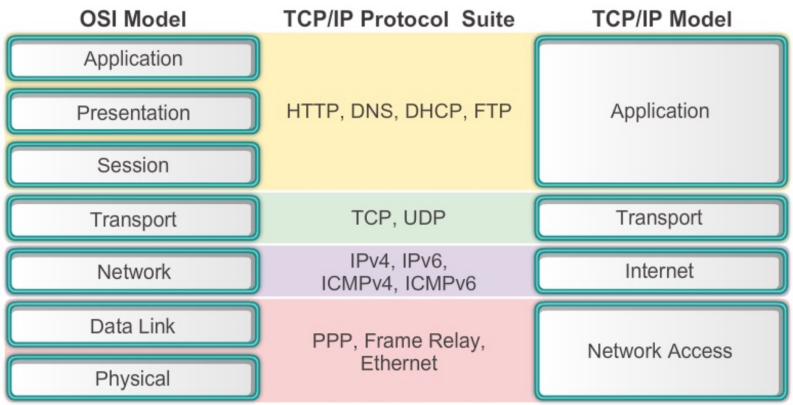

Figure: Position of TCP in the TCP/IP protocol stack

- TCP resides at Transport layer, and is one of the core protocols of the Internet Protocol (IP) suite.

- TCP is much more complex transport protocol in comparison to UDP:
  - **Connection oriented** → provide mechanisms to set-up and tear-down a full duplex connection between two end points.
  - **Reliability** → guarantee error free and ordered delivery of information.
  - **Flow and Congestion control** → have mechanisms to control traffic ingested to the network.
- Some applications use TCP as transport protocol:
  - World Wide Web (WWW)
  - File Transfer Protocol (FTP)
  - Simple Mail Transfer Protocol (SMTP)
  - etc.

TCP Segment

- A TCP packet is called a **segment**.
- Each segment consists of a header of 20 (with no options) to 60 (with options) bytes, followed by data from the application program.
- TCP accepts data from a data stream, divides it into chunks, and adds a TCP header to create a TCP segment $\rightarrow$ TCP segment is then encapsulated into an Internet Protocol (IP) datagram and exchanged with peers.

TCP Header

Offsets Octet		0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																Urgent pointer (if URG set)															
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

Figure: TCP Header

TCP Header description

1. **Source port (16 bits)** Identifies the sending port
2. **Destination port (16 bits)** Identifies the receiving port
3. **Sequence number (32 bits)**
   - If the `SYN` flag is set to 1 $\rightarrow$ This is the initial sequence number. The sequence number of the actual first data byte and the acknowledged number in the corresponding ACKs are then equal to this sequence number plus 1.
   - If the `SYN` flag is set to 0 $\rightarrow$ This is the accumulated sequence number of the first data byte of this segment for the current session.
4. **Acknowledged number (32 bits)**
   - If the `ACK` flag is set, then the value of this field is the next sequence number that the receiver is expecting. This acknowledges receipt of all prior bytes (if any). The first `ACK` sent by each end point acknowledges the other end's initial sequence number itself but no data.
5. **Data offset (or header length) 4 bits**
   - Specifies the size of TCP header in 32-bit words. The minimum size header is 5 words and the maximum is 15 words, thus giving the minimum and maximum size of 20 and 60 bytes, respectively (so allowing for up to 40 bytes of options in the header). This field gets its name from the fact that it is also the offset from the start of TCP segment to the actual data.

TCP Header description - Cont

6. **Reversed (3 bits)** for future use and should be set to 0
7. **Flags (9 bits)** a.k.a **Control bits**, contains 9 **1-bit** flags as follows:
   - `NS` → ECN-nonce concealment protection (added to header by RFC 3540)
   - `CWR` → Congestion Window Reduced (CWR) flag is set by the sending host to indicate that it received a TCP segment with the ECE flag set and had responded in congestion algorithm mechanism (added to header by RFC 3168)
   - `ECE` → ECN-Echo indicates
     - If the `SYN` flag is set to 1, that the TCP peer is ECN capable
     - If the `SYN` flag is clear 0, that a packet with Congestion Experienced flag in IP header set is received during normal transmission (added to header by RFC 3168)
   - `URG` → indicates that the Urgent pointer field is significant.
   - `ACK` → indicates that the Acknowledgement field is significant. All packets after the initial `SYN` packet sent by the client should have this flag set.
   - `PSH` → Push function. Asks to push the buffered data to the receiving application.
   - `RST` → Reset the connection.
   - `SYN` → Synchronize sequence number. Only the first packet sent from each end point should have this flag set.
   - `FIN` → No more data from the sender.

TCP Header description - Cont

8. **Window Size (16 bits)** the size of receive window, which specifies the number of window size units (by default, bytes) (beyond the sequence number in the acknowledgement field) that the sender of this segment is currently willing to receive (see *Flow Control* and *Window Scaling*)
9. **Checksum (16 bits)** is used for error-checking of the header and data.
10. **Urgent pointer (16 bits)** If the `URG` flag is set, then this 16-bit field is an offset from the sequence number indicating the last urgent data byte.
11. **Options (Variable 0-320 bits, divisible by 32)** The length of this field is determined by the data offset field. Options have up to three fields: Option-Kind (1 byte), Option-Length (1 byte), Option-Data (variable). The Option-Kind field indicates the type of option, and is the only field that is not optional.
12. **Padding** The TCP Header padding is used to ensure that the TCP header ends and data begins on a 32-bit boundary. The padding is composed of zeros.

TCP Header - Example

- 00 CD 00 18 00 00 0E F1 00000D5D

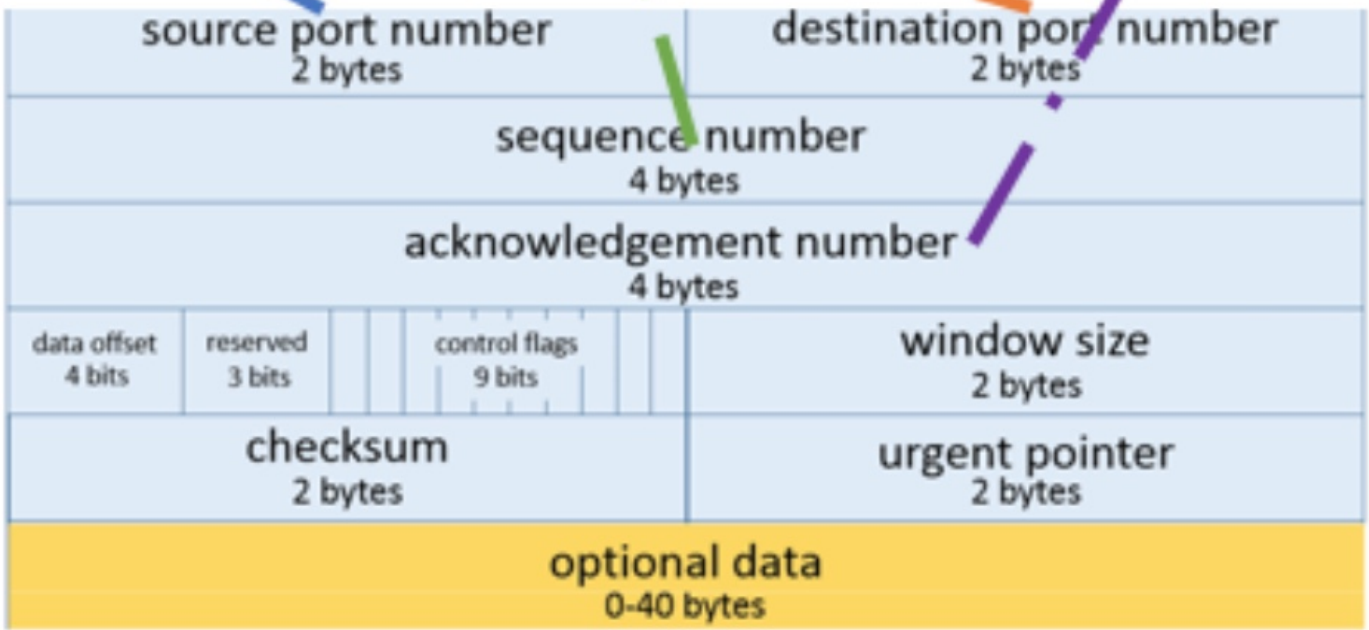

Figure: TCP Header example

TCP Connection Establishment

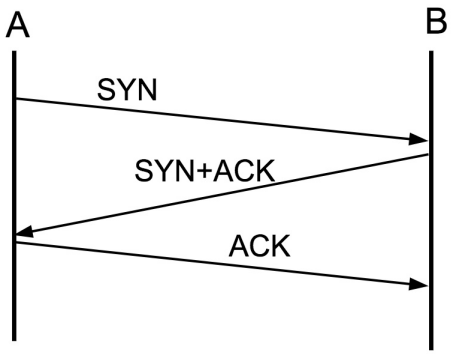

Figure: TCP 3-way handshake

- To establish a connection, TCP uses a three-way handshake.
- Before host A attempts to connect to host B, host B must first bind to and listen at a port to open it up for connections: this is called a passive open. Once the passive open is established, host A may initiate an active open.

- To establish a connection, the three-way handshake (or 3-steps) occurs:
  - `SYN` The active open is performed by endpoint A sending a `SYN` to the server. Endpoint A sets the segment's sequence number to a `random value` a.
  - `SYN + ACK` In response, endpoint B replies with a `SYN + ACK`. The acknowledgement number is set to one more than the received sequence number, i.e., $a + 1$, and the sequence number that B chooses for the packet is another random number, b.
  - `ACK` Finally, A sends and `ACK` back to the server. The sequence number is set to the received acknowledgement value, i.e., $a + 1$ and the acknowledgement number is set to one more than the received sequence number, i.e., $b + 1$
- At this point, both A and B have received an acknowledgement of the connection; and a full-duplex communication is established.

TCP Connection termination

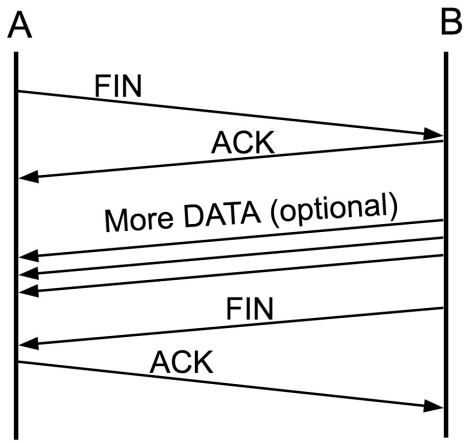

Figure: TCP Connection termination

- The connection termination phase uses a four-way handshake, with each side of the connection terminating independently.

- Endpoint A sends B a packet with the `FIN` flag set (a FIN packet) to announce that A has no more data to send.
- B sends back to A an `ACK` of the `FIN`
- B may continue to send additional data to A (**Optional**)
- When B is also ready to cease sending, B then sends its own `FIN` to A.
- A sends B an `ACK` of the `FIN`.
- After both `FIN/ACK` exchanges are concluded, the side which sent the first `FIN` before receiving one waits for a timeout before finally closing the connection, during which the local port is unavailable for new connections. This prevents confusion due to delayed packets being delivered during subsequent connections.
- It is also possible to terminate the connection by a 3-way handshake, when A sends a `FIN` and B responds with a `FIN + ACK` (merely combines 2 steps into one) and A replies with an ACK. This is perhaps the most common method.

ENJOY !!!