Transmission Control Protocol (TCP)

Outline

• Transmission Control Protocol

Transport Layer 1/2

Transport Layer Addressing

Port Numbers

Addresses

• Data link layer → MAC address
• Network layer → IP address
• Transport layer → Port number (choose among multiple processes running on destination host)

Port numbers are 16-bit integers (0 → 65,535)

• Servers use well known ports, 0-1023 are privileged
• Clients use ephemeral (short-lived) ports

Internet Assigned Numbers Authority (IANA) maintains a list of port number assignment

• Well-known ports (0-1023) → controlled and assigned by IANA
• Registered ports (1024-49151) → IANA registers and lists use of ports as a convenience (49151 is ⅓ of 65536)
• Dynamic ports (49152-65535) → ephemeral ports

For well-known port numbers, see /etc/services on a UNIX or Linux machine
Socket Addressing

- Process-to-process delivery needs two identifiers
  - IP address and Port number
  - Combination of IP address and port number is called a socket address (a socket is a communication endpoint)
  - Client socket address uniquely identifies client process
  - Server socket address uniquely identifies server process
- Transport-layer protocol needs a pair of socket addresses
  - Client socket address
  - Server socket address
  - For example, socket pair for a TCP connection is a 4-tuple:
    - Local IP address, local port, and
    - Foreign IP address, foreign port

Multiplexing and Demultiplexing

- Sender side may have several processes that need to send packets (albeit only 1 transport-layer protocol)
- At receiver side, after error checking and header dropping, transport-layer delivers each message to appropriate process

Transmission Control Protocol 1/10

- TCP must perform typical transport layer functions:
  - Segmentation breaks message into packets
  - End-to-end error control since IP is an unreliable Service
  - End-to-end flow control to avoid buffer overflow
  - Multiplexing and demultiplexing sessions
- TCP is [originally described in RFC 793, 1981]
  - Reliable
  - Connection-oriented virtual circuit
  - Stream-oriented users exchange streams of data
  - Full duplex concurrent transfers can take place in both directions
  - Buffered TCP accepts data and transmits when appropriate (can be overridden with “push”)

Transmission Control Protocol 2/10

- Reliable
  - Requires ACK and performs retransmission
  - If ACK not received, retransmit and wait a longer time for ACK. After a number of retransmissions, will give up
  - How long to wait for ACK? (dynamically compute RTT for estimating how long to wait for ACKs, might be ms for LANs or seconds for WANs)
    \[
    RTT = \alpha \times \text{old RTT} + (1 - \alpha) \times \text{new RTT}
    \]
    where \( \alpha \) usually 90%
  - Most common, Retransmission time = 2* RTT
  - Acknowledgments can be “piggy-backed” on reverse direction data packets or sent as separate packets

Transmission Control Protocol 3/10

- Sequence Numbers
  - Associated with every byte that it sends
  - To detect packet loss, reordering and duplicate removal
  - Two fields are used sequence number and acknowledgment number. Both refer to byte number and not segment number
  - Sequence number for each segment is the number of the first byte carried in that segment
  - The ACK number denotes the number of the next byte that this party expects to receive (cumulative)
    - If an ACK number is 5643 received all bytes from beginning up to 5642
    - This acknowledges all previous bytes as received error-free

Transmission Control Protocol 4/10

- Sending and Receiving buffers
  - Senders and receivers may not produce and consume data at same speed
  - 2 buffers for each direction (sending and receiving buffer)
Transmission Control Protocol

• TCP uses a sliding window mechanism for flow control
• Sender maintains 3 pointers for each connection
  ➢ Pointer to bytes sent and acknowledged
  ➢ Pointer to bytes sent, but not yet acknowledged
    ✓ Sender window includes bytes sent but not acknowledged
  ➢ Pointer to bytes that cannot yet be sent

Transmission Control Protocol

• Flow Control
  ➢ Tell peer exactly how many bytes it is willing to accept
    (advertised window ➔ sender can not overflow receiver buffer)
    ✓ Sender window includes bytes sent but not acknowledged
    ✓ Receiver window (number of empty locations in receiver buffer)
    ✓ Receiver advertises window size in ACKs
  ➢ Sender window <= receiver window (flow control)
    ✓ Sliding sender window (without a change in receiver’s advertised window)
      Expanding sender window (receiving process consumes data faster than it receives ➔ receiver window size increases)
      Shrinking sender window (receiving process consumes data more slowly than it receives ➔ receiver window size reduces)
    ➢ Closing sender window (receiver advertises a window of zero)

Transmission Control Protocol

• Error Control
  ➢ Mechanisms for detecting corrupted segments, lost segments, out-of-order segments, and duplicated segments
  ➢ Tools: checksum (corruption), ACK, and time-out (one time-out counter per segment)
    ✓ Lost segment or corrupted segment are the same situation: segment will be retransmitted after time-out (no NACK in TCP)
    ✓ Duplicate segment (destination discards)
    ✓ Out-of-order segment (destination does not acknowledge, until it receives all segments that precede it)
    ✓ Lost ACK (loss of an ACK is irrelevant, since ACK mechanism is cumulative)

Transmission Control Protocol

• Congestion Control
  ➢ TCP assumes the cause of a lost segment is due to congestion in the network
  ➢ If the cause of the lost segment is congestion, retransmission of the segment does not remove the problem, it actually aggravates it
  ➢ The network needs to tell the sender to slow down (affects the sender window size in TCP)
    ✓ Actual window size = Min (receiver window size, congestion window size)
      The congestion window is flow control imposed by the sender
      The advertised window is flow control imposed by the receiver

Transmission Control Protocol

• Full-Duplex
  ➢ send and receive data in both directions.
  ➢ Keep sequence numbers and window sizes for each direction of data flow
TCP Connection Establishment

- Passive open
  - SYN: Synchronize
  - ACK: Acknowledge

TCP Options

- MSS Option
  - Maximum segment size is the maximum amount of data it is willing to accept in each TCP segment.
  - Sending TCP uses receiver’s MSS as its MSS.

- Window Scale Option
  - Maximum window is 65,535 bytes (corresponding field in TCP header occupies 16 bits).
  - It can be scaled (left-shifted) by 0-14 bits providing a maximum of $65,535 \times 2^{14}$ bytes (one gigabyte).
  - Option needed for high-speed connections or long delay paths.
  - In this case, the other side must send the option with its SYN.

TCP MSS and output

- MSS is = (interface MTU - fixed sizes of IP and TCP headers (20 bytes)).
- MSS on an Ethernet (IPv4) = 1460 bytes (1500 - 40)
- Successful return from write implies you can reuse application buffer.

TCP Connection Termination

- FIN: Finish
- Step 1 can be sent with data
- Steps 2 and 3 can be combined into 1 segment

State Transition Diagram

- Typical TCP states visited by a TCP client

- Typical TCP states visited by a TCP server
**State Transition Diagram**

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOSED</td>
<td>There is no connection.</td>
</tr>
<tr>
<td>LISTEN</td>
<td>The server is waiting for calls from the client.</td>
</tr>
<tr>
<td>SYN-SENT</td>
<td>A connection request is sent; waiting for acknowledgment.</td>
</tr>
<tr>
<td>SYN-RCVD</td>
<td>A connection request is received.</td>
</tr>
<tr>
<td>ESTABLISHED</td>
<td>Connection is established.</td>
</tr>
<tr>
<td>FIN-WAIT-1</td>
<td>The application has requested the closing of the connection.</td>
</tr>
<tr>
<td>FIN-WAIT-2</td>
<td>The other side has accepted the closing of the connection.</td>
</tr>
<tr>
<td>TIME-WAIT</td>
<td>Waiting for retransmitted segments to die.</td>
</tr>
<tr>
<td>CLOSE-WAIT</td>
<td>The server is waiting for the application to close.</td>
</tr>
<tr>
<td>LAST-ACK</td>
<td>The server is waiting for the last acknowledgment.</td>
</tr>
</tbody>
</table>

Can use `netstat` command to see some TCP states

---

**Packet Exchange**

Send 1-segment request and receive 1-segment reply

---

**TIME_WAIT State**

- The end that performs the active close goes through this state
- Duration spent in this state is twice the maximum segment life (2 MSL)
  - MSL: maximum amount of time any given IP can live in the network
- Every TCP implementation must choose a value for MSL
  - Recommended value is 2 minutes (traditionally used 30 seconds)
- TIME_WAIT state motives
  - Allow old duplicate segments to expire in the network (relate to connection incarnation)
  - TCP will not initiate a new incarnation of a connection that is in TIME_WAIT state
  - Implement TCP's full-duplex connection termination reliably
  - The end that performs the active close might have to resend the final ACK

---

**TCP Segment Format**

- Source Port and Destination Port
  - Identify processes at ends of the connection
- Control bits
  - URG urgent (urgent data present)
  - ACK acknowledgment
  - PSH push request
    - Inform receiver TCP to send data to application ASAP
  - RST reset the connection
  - SYN synchronize sequence numbers
  - FIN sender at end of byte stream
TCP Header Fields 2/2

• **Sequence Number**: position of the data in the sender’s byte stream
• **Acknowledgment Number**: position of the byte that the source expects to receive next (valid if ACK bit set)
• **Header Length**: header size in 32-bit units. Value ranges from [5-15]
• **Window**: advertised window size in bytes
• **Urgent**
  ✓ defines end of urgent data (or “out-of-band”) data and start of normal data
  ✓ Added to sequence number (valid only if URG bit is set)
• **Checksum**: 16-bit CRC (Cyclic Redundancy Check) over header and data
• **Options**: up to 40 bytes of options