Transport Layer Protocols

Introduction
Transmission Control Protocol (TCP)

Introduction
Transport Layer

• Provides reliable, cost effective data transport from source to destination. (Transport Entity).
• Generic services provided by transport layer:
  – Connection-Oriented transport service. (Similar to network layer)
  – Connectionless transport service. (Similar to network layer).

Key Question:
• If transport layer similar to network layer service, why design two distinct layers?

Quality of Service (QoS)

• What is QoS?
• QoS parameters provided by Transport layer.
• Connection establishment delay.
• Connection establishment failure probability.
  – Throughput
  – Transit delay.
  – Residual error ratio.
  – Protection.
  – Priority
  – Resilience
Transport Service Primitives

- Allow application programs to access transport service.
- How is it different from network layer primitives?
- Five common primitives:
  - LISTEN.
  - CONNECT.
  - SEND.
  - RECEIVE.
  - DISCONNECT.
- Transport layer protocols used in Internet:
  - TCP: Connection-oriented.
  - UDP: Connectionless.

Transmission Control Protocol

- **TCP**
  - Application-Network Layer interface.
  - Connection oriented, reliable.
- **TCP at sender**:
  - Application generates streams of data.
  - TCP establishes communication.
  - Chops streams into transportable units (Numbering).
  - Send units one by One.

Transmission Control Protocol

- **TCP at Receiver**:
  - Wait till all units arrive.
  - Check for errors.
  - Error free units delivered to application.
  - Close the connection.
Host to Host Versus Process to Process Communication

• **Host to Host:**
  – Done via IP.
  – Message delivered to destination machine.
  – Still deliver incomplete.

• **Process to Process:**
  – Message handed to application.
  – Achieved via Client-Server paradigm.
  – TCP responsible for this delivery.

Port Addresses

• To define communication:
  – Local host (IP)
  – Local Client program (Port)
  – Remote host (IP)
  – Remote server program (Port)

• Client defines its own port address.
  – Ephemeral.

• Server Port number: Well known.
• Example: TELNET

Socket Addresses/Reserved Port Numbers

**Socket Addresses/Reserved Port Numbers**

• Combination of IP address and port number.
• Communication pipe: Pair of sockets at each end.
• Four pieces of information part of IP and TCP header.
• Reserved port numbers up to 256.
• Same port numbers for any service used in TCP and UDP.
TCP Services

Stream Data Service (Stream Transport Layer Service).
- Accept streams of characters.
- Create packets.
- Send packets across network.
- Receive packets, extract data.
- Order inner information.
- Deliver as stream to receiving application.

TCP Services

Stream Data Service
- Use of buffers.
- Sender writes to buffer.
- No segment to write mapping.
- Application may push, TCP may not follow.
- Segments stored at receiving buffer.
- Application does a read over buffer.
- No segment to read mapping.

TCP Services

- Full Duplex Service
  - Data flow in both directions.
  - One connection simultaneously carries data from A to B and B to A.
  - Acknowledgement Piggybacking.
- Reliable Service
  - Use of acknowledgment.
  - Use of ordering of units.
  - Use of error correction code.
IANA Ranges: Port Numbers

- Well Known Ports: 0-1023.
- Registered Ports: 1024-49151.
  - Not assigned.
  - But must be registered.
  - Not assigned/controlled.
  - Not registered.
  - Ephemeral in nature.
- Note: OS may use ranges other than IANA ranges.

Figure 1: TCP Segment

<table>
<thead>
<tr>
<th>Source Port Address (16)</th>
<th>Destination Port Address (16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Number (32)</td>
<td>Acknowledgment Number (32)</td>
</tr>
<tr>
<td>HLEN (4) Res (6) Flag</td>
<td>Window Size (16)</td>
</tr>
<tr>
<td>Checksum (16)</td>
<td>Urgent Pointer (16)</td>
</tr>
<tr>
<td>Options and Padding</td>
<td>Data</td>
</tr>
</tbody>
</table>

TCP Segment

- **Source Port Address**: Defines port number of sending application program.
- **Destination Port Number**: Defines port number of destination application program.
- **Sequence Number**: Number assigned to first byte of data in segment.
- **Acknowledgment Number**: Sequence number of byte the local host expects to receive next.
- **Header Length**: Specifies length of header.
- **Reserved**: Six bit value reserved for future use.
TCP Segment

- **Control**: Six control bits or flags.
  - URG: Urgent pointer is valid.
  - ACK: Acknowledgement is valid.
  - PSH: Request for Push.
  - RST: Reset the connection.
  - SYN: Synchronize sequence numbers.
  - FIN: Terminate connection.
- **Window Size**: Size of window in bytes, the other party must maintain.

TCP Segment

- **Checksum**: Verifies that the header is not corrupted.
- **Urgent Pointer**: Valid if URG flag set.
  - Points to end of urgent data in the segment.
- **Options**: 40 bytes of optional information in TCP header.
  - End of option.
  - No operation.
  - Maximum segment size.
  - Window Scale Factor
  - Timestamp

Flow Control in TCP

- **Flow Control**: Amount of data source can send before receiving acknowledgment from receiver.
- One Extreme: Acknowledgment for every byte sent.
- Other Extreme: Send all data without worrying about ack.
  - -- Overwhelms the receiver.
- Solution of TCP: Stands somewhere in between.
- Solution implemented using Sliding Window.
- **Sliding Window**: Mechanism allowing sender to transmit more than one packet of data before receiving an acknowledgment.
Sliding Window

- Window covers a portion of buffer that sender can send without ack.
- Window slides over the buffer as soon as ack. is received. (Figure 2).
- Acknowledgment received for first three bytes.
- Window slides three bytes to right.
- Pointer: used by source to know the status of bytes.

**Figure 2: Simple Sliding Window**

Before Sliding

After Sliding

Sliding Window

- Previous example: Fixed sliding window.
- Size of window in TCP is variable.
- Destination in each ack. Segment can define size of window. (Figure)
- Advertised size relative to ack. number.
- Example:
  - Receiver acknowledges receipt of byte 3000.
  - Defines size of window: 200 bytes in ack.
  - Window now expands from byte 3,001 to byte 3,200.
Sliding Window

Points worth noting
• Sender does not have to send a full window's worth of data.
• Size of window can be increased/decreased by destination.
• Destination can send acknowledgment at any time.

Window Management

Two buffers and one window control flow of data.
• Sending TCP Buffer: Stores data coming from sending application.
• Application creates data, writes to buffer.
• Receiving TCP buffer: receives, checks data, stores them in buffer.
• Receiving application consumes data from buffer.
• Size of window advertised usually is space left over at receiver buffer. (Figure 3).

Figure 3: Window Management

Sender

Receiver

Segment 1
Seq 1001, 4000 bytes
Ack 5001 Win 0

Segment 2
Seq 5001, 1000 bytes
Ack 5001 Win 1000
Silly Window Syndrome

Problem: A situation in which small window size is advertised by receiver and small segment sent by sender.

Reasons
- Sender creates data slowly.
- Receiver consumes data slowly.

Result: Data sent in small segments.
- Reduces efficiency of operations.

Example: Segment size 41 bytes: 20 bytes of TCP header, 20 bytes of IP header, 1 byte of data.

Silly Window Syndrome

Syndrome created by Sender
- Sending application creates data slowly.
- Sending TCP creates silly window syndrome.
- Application writes 1 byte into the buffer.
- Sending TCP may create segments of 1 byte.
- Result: 41 byte segments travelling through the Internet.
- Solution: Prevent sending TCP from sending data byte by byte.
- TCP waits for larger block
- How long to wait?—Nagle's Algorithm.

Nagle's Algorithm: (Algorithm of sending TCP)
- Self clocking heuristic that clumps outgoing data to improve throughput, avoid silly window syndrome.

Steps
1. Send First piece of data from application (Even if it is only 1 byte).
2. Accumulate data in the buffer.
   - Wait until receiver sends ack.
   OR
   - Until enough data accumulated to fill max. size segment.
Silly Window Syndrome

3. Step 2 repeated for rest of transmission.
   Note: Segment 3 sent if:
   • Ack. is received for segment 2 OR
   • Enough data accumulated to fill max size segment.
   • What do you observe??

Silly Window Syndrome

Syndrome created by receiver.
• Receiver consumes data slowly. (E.g.: one byte at a time).
• Example:
  • Sender generates data 1k at a time.
  • Receiver consumes data 1 byte at a time.
  • Sending TCP buffer size: 4k: 1k of data sent.
  • Receiving TCP receives data. Buffer full.
  • Receiving TCP advertises window size=0.
  • Receiver reads 1 byte, Receiving TCP advertises 1 byte.
  • Sending TCP sends 1 byte and so on.

Silly Window Syndrome

Solutions
• Clark’s Solution: Send ack. as soon as data arrives.
  • Announce window size of zero till enough buffer space available. (max size segment or half buffer).
• Delayed Acknowledgment
  • Segment not acknowledged immediately.
  • Decent space available in incoming buffer.
  • Ack. sent back to sender.
  • Prevents sender from sliding its window.
  • Reduces traffic.
  • Retransmissions possible.
Error Control

TCP a reliable transport layer protocol.
- Application relies on TCP to deliver data to receiver:
  - In order
  - Without error.
  - Without any part lost/duplicated.
- Reliability provided using error control.
- Error control includes mechanisms to detect errors.
- Also includes mechanisms to correct errors upon detection.

31

Error Control

- 3 tools for Error detection:
  - Checksum
  - Acknowledgment
  - Time-out.
- Note: No negative ack. used in TCP.
- Error Correction mechanism very simple.
  - Achieved using Time out counter for each segment.
    - Each counter checked periodically.

32

Figure 4: Corrupted Segment

Sender

Seq: 1201, 200 bytes

ACK: 1601

Seq: 1401, 200 bytes

Seq: 1601, 200 bytes

Timeout

Segment 1

Segment 2

Segment 3

Segment 3 transmitted

Segment 3 corrupted

Segment 3

Receiver

ACK: 1601

ACK: 1801

33
Duplicate and Out of Order Segment

- Duplicate segments identified on the basis of sequence numbers.
- Out of order segments not acknowledged TCP receives all datagrams that precede it.
TCP Timers

- To perform its operations smoothly TCP maintains four timers:
  - Retransmission.
  - Persistence
  - Keepalive.
  - Time-waited.
- We shall examine them one by one.

Retransmission Timer

Controls lost or discarded datagrams.
- Retransmission time: waiting time for ack. of segment.
  Two situations:
  - If ack. received for segment before timer goes off, timer destroyed.
  - If timer goes off before ack. arrives, segment retransmitted, timer reset.

Calculation of Retransmission Time

Calculation of Retransmission Time
- No two connections same.
- TCP cannot use same retransmission time for both the connections.
- Fixed retransmission time results in serious consequences.
  - Long retransmission timer problems.
  - Short retransmission timer problems.
- Timer not fixed for single connection too.
- Keep track of Non traffic and Congestion Periods
Calculation of Retransmission Time

TCP uses dynamic retransmission timeout.
• Based on Round Trip Time (RTT).
• Several formulas used. Most common:
  Retransmission Time = 2 * RTT
• Calculation of RTT done dynamically: Two methods:
  – Option 1: Timestamp. (Described in header section)
  – Option 2: Start timer and wait.

Calculation of Retransmission Time

Option 2:
• TCP sends segment, starts timer, waits for ack.
• Measures time between sending of segment, receiving of ack.
• Each segment has round trip value.
• Value of RTT used in calculation of retransmission time of next segment = Updated value of RTT using:
  • RTT = $\alpha$ * Previous RTT + (1 - $\alpha$) current RTT
  • Usually $\alpha$ = 90%.

Karn's Algorithm

Problem
• Segment not acknowledged during retransmission period.
• Segment retransmitted.
• Sender receives ack.
  – Ack for original one
  – OR
  – Ack for retransmitted one.
• RTT calculation based on which segment??
• Karn's Solution: Ignore RTT of retransmitted segment in New RTT.
• RTT not updated till ack. received without retransmission.
Karn's Algorithm
- Merely ignoring retransmitted segments leads to failure.
- Example: Sending segment after sharp increase in delay.
- Solution: Timer Backoff Strategy.
  - Transmission expires causing retransmission, TCP increases timeout.
- Each retransmission increases timeout.
- Upper limit on timeout value.
- New timeout = $\beta^* \text{timeout}$.
  - $\beta$ normally 2.

TCP Deadlock Problem
- Deals with zero Window Size Advertisement.

Scenario
- Receiver announces window size=0.
- Sender waits for ack from receiver.
- This ack can be lost.
- Receiver is expecting sender to sends segments.
- Sender still waits for ack. (Both TCPs in waiting state).
- Solution: Persistence Timer.

Persistence Timer
- Persistence timer used for each connection.
- Sender receives ack with zero window size.
- Sender starts persistence timer.
- Persistence timer goes off.
- Sender sends special segment: Probe.
  - Contains 1 byte of data.
- Alerts receiver of current situation.
- Value of persistence time = Value of retransmission time.
Persistence Timer

- If no response from receiver:
  - Send another probe
  - Persistence timer doubled.
  - Process repeated till threshold = 60 seconds.
  - Continue to send probe every 60 seconds till window is reopened.

Keepalive Timer

- Prevents long idle connection between two TCPs.
  
  **Scenario:**
  - Client opens TCP connection to server.
  - Transfers some data to server.
  - Becomes silent
    - May have crashed.
  - Result: Connection remains open forever.
  - Solution: Equip server will keepalive timer.

Keepalive Timer

- Time out usually 2 hours.
  - Server sends probe after 2 hours.
  - Each probe spaced 75 secs apart.
  - No response, connection is terminated.
Time-Waited Timer

- Used during connection termination.
- TCP closes connection.
- But connection really not closed.
- Connection held at limbo for time waited period. Why??
- Value of timer: Twice the expected lifetime of segment.

Connection

- TCP connection oriented.
- Virtual path between sender and destination.
- All segments of one msg. sent over virtual path.
- Facilitates retransmission of lost/damaged frames.
- Connection oriented transmission achieved through:
  - Connection establishment.
  - Connection termination.

Connection Establishment

- Before data transfer
  - -- Each party initialises communication
  - -- Gets approval from other party.
Four actions taken:
- Sender (A) announces its wish for connection
  - -- Includes initial initialisation information.
- Receiver (B) sends ack. To confirm A's request.
- B sends segment: Its initialisation info about traffic from B to A.
- A sends ACK. to confirm B's request.
Connection Establishment

• No interval between steps 3 and 4.
• Hence combined into one step.
• Connection establishment is a three way handshaking process.
• Three way handshaking: Client makes connection with server using TCP.
• Starts with server.
• Server program requests TCP for Passive Open.
• Client program requests TCP for Active Open.

Three Way Handshaking

TCP now starts three way handshaking process.
(Figure 7)
1. Client sends first segment: SYN Segment.
   • Source, destination ports.
   • Initialisation Sequence Number (ISN).
   • ISN: numbering bytes.
   • MSS, window scale factor defined.
   • Window size not defined here, no acks.

   • Dual purpose segment: acknowledges client, conveys its ISN to client.
   • Acknowledgment: Flag set.
   • Ack. Number = Client ISN +1.
   • Defines window size.(If needed).
   • Contains window scale factor if needed with MSS of server.
   • This is two segments combined into one.
Three Way Handshaking

   - Acknowledges receipt of second segment.
   - Acknowledgment number: Server ISN + 1.
   - Client also defines server window size.
   - Note: Data can be sent with third packet.
   - Rare Situation: Both processes issue an active open.
   - Both transmit SYN+ACK segment to each other.
   - Single connection established.

Connection Termination

Any party can close connection.
- Connection in one direction is then terminated.
- Other party can continue to send data in other direction.
- This is a four way process.
- Procedure starts with client: It issues active close to TCP.
- Client TCP closes connection in client-server direction.
- Server later finishes sending data to client.
- Issues a close to TCP in server-client direction: Passive close.
Connection Termination: Four Way Handshaking

1. Client sends first segment: FIN segment. (Figure 8)
2. Server sends second segment: ACK segment to confirm receipt of FIN.
   -- Ack Number = 1 + Seq No. of FIN segment.
   -- When finished, server sends third segment: FIN segment.
4. Client sends fourth segment: ACK segment to confirm receipt of FIN segment.
   -- Ack number: 1+seq no. of FIN.

Figure 8: Four Way Handshaking

Sender
Segment 1: FIN
seq: 2500, ack: ___________

Receiver
Segment 2: ACK
seq: 7000, ack: 2501
Segment 3: FIN
seq: 7001, ack: 2501
Segment 4: ACK
seq: 2501, ack: 7002

Connection Resetting

Resetting: Current connection is destroyed.
- Connection requested to non-existent port.
  -- Receiver sends segment with RST bit set to annul request
- One TCP aborts connection due to abnormal situation.
  -- Sends RST segment to close connection.
- TCP on one side discovers TCP on other side idle for long time.
  -- RST segment sent to destroy connection.
States for TCP

- CLOSED: No connection.
- LISTEN: Server waiting for class from client.
- SYN-SENT: Connection req. sent, waiting for ack.
- SYN-RCVD: Connection req. received.
- ESTABLISHED: Connection established.
- FIN-WAIT-2: Receiver accepts closing of connection.
- CLOSING: Both sides decide to close simultaneously.
- TIME-WAIT: Waiting for transmitted segments to die.
- CLOSE-WAIT: Server waiting for applic. To close.
- LAST-ACK: Server waiting for last ack.

State Transition Diagram

Client States:
- CLOSED
- SYN-SENT
- ESTABLISHED
- FIN-WAIT-1
- FIN-WAIT-2
- TIME-WAIT

Server States:
- CLOSED
- LISTEN
- SYN-RCVD
- ESTABLISHED
- CLOSE-WAIT
- LAST-ACK

Refer to Figure 9
Congestion Control in TCP

- **Congestion**: Condition of severe delay caused by overload of datagrams at one or more routing points.
- Congestion goes beyond threshold, datagrams discarded.
- TCP must avoid this situation.
- Flow control problem: Receiver dictated window size.
- One important entity ignored: Network.
- Today TCP protocols include this feature: Congestion Window Size.
- Allowed Window = min (receiver advertisement, congestion window)

Steady State:
- Congestion Window = Receiver Window.
- Assumption of TCP: Datagram loss comes from congestion.

**Strategy**: Multiplicative Decrease Congestion Avoidance.
- Segment loss detected.
- Congestion window size reduced by half.
- Congestion window size increased exponentially

How to recover when congestion ends?
- Possibility: Reverse multiplicative decrease.
- Problem: Unstable system oscillating wildly between 'no traffic' and 'congestion'.

**Solution**: Slow Start to scale up transmission.
- Initialize the window size to 1.
- Send segment 1.
- Receive ack and increase window size to 2.
- Send 2 segments.
- Receive 2 acks.
- Increase window size to 4 and so on.
Congestion Control in TCP

- Additional Restriction: When status
- Congestion window= half of its original size before congestion:
- Enter into Congestion Avoidance phase.
- Increase congestion window by only 1.

TCP Operation

Uses concepts common to transport layer.
- Encapsulation/decapsulation: Protocol capable of encapsulating/decapsulating messages.
- Queuing: Uses Queues.
- Multiplexing/Demultiplexing: Several applications use services of single TCP.

TCP Operation

Pushing Data:
- Stream of data captured in buffer.
- Data buffered, segments formed.
- For some programs not acceptable.
- Solution: Sending side requests Push operation.
- Sending TCP sets PSH bit.
- PSH tells receiving TCP to deliver data to sender ASAP.
- Note: Choice of using option left to TCP.
TCP Operation

Urgent Data
- Sender wants data to be read out of order by receiver.
  - Example: Aborting a process.
- Solution: Send segment using URG bit set.
  - Inserted at beginning of segment.
  - Urgent pointer field: defines end of urgent data/beginning of normal data.
- Receiving TCP extracts urgent data, delivers it out of order to Receiving application.

TCP Design

Complex Protocol.
- Uses tens of thousands of lines of code.
- We look at simplified, bare bones TCP.
  - Error handling and checking, packet validation ignored.
  - For actual code: Consult TCP/IP Illustrated, Volume 2 by Richard Stevens.
- Design: Involves a table, 3 modules and a set of timers.
- Refer to Figure

Figure 10: TCP Design
TCP Components

Transmission Control Blocks (TCBs)
• Structure holding information about connection.
• Controls the connection
• An array of TCBs for several connections.
• Fields: Some fields listed below:
  – State
  – Process
  – Local/remote IP address, local/remote window
  – Local/remote port number, sending/receiving sequence number

TCP Components

Input Processing Module
• Details to process data/acknowledgment in
• ESTABLISHED state.
• Functions:
  – Sending ack.,
  – Window size announcement, error checking and so on.

Output processing module
• Details to send data in ESTABLISHED state.

TCP Components

Main Module: Invoked by:
– Arrived TCP segment
– Time out event.
– Message from application.
• Action taken depends on current state.
Timer: TCP records the information of timers here.