Transport Layer

Dr. G. A. Marin

This material is provided for educational purposes only. No further reproduction is permitted.

OSI Reference Model
Two Types of Transport Service

- **Connection-oriented**
  - Sets up connection, sends data, releases connection
  - Reliable, in-sequence delivery (recovers lost or damaged data)

- **Connectionless**
  - Just sends/receives...
  - No guarantees
  - Developers of Internet Applications (email, Web, File Transfer, phone) choose one of the above.

Connection-Oriented vs Connectionless at Layers 2,3,4

- **Layer 2 CO** may be especially useful on links with high error rates. CO at DLC means a reliable service that retransmits errored or lost frames at layer 2.
- **Layer 3 CO** or CL service is offered by the network provider. Quality may differ across Internet, for example. CO at NL means setting up connections before sending data. All data follows same route, etc.
- **Layer 4 CO** or CL service is offered to applications by the transport entities that operate in the end points (hosts). Allows end-stations to deal with poor service, congestion discards, etc.
Transport Service Model

Construction of a Frame
Server/Clients Basic Primitives

- Server executes LISTEN (and blocks)
- Any ready client executes a CONNECT
  - blocks caller process and sends Conn Req to server transport process
- Server transport entity checks that server is on LISTEN, unblocks server, sends Conn Accepted back to client.
- When Conn Accepted TPDU received at client, client unblocked and connection is established.
- Data exchanged using SEND and RECEIVE.
- Transport user sees only primitives LISTEN, CONNECT, SEND, RECEIVE, DISCONNECT.

Basic Connection Steps

- Client application issues a CONNECT
- Client transport entity sends Connection Request (CR) in TPDU
- Server transport entity checks to see Server is blocked on LISTEN
  - then it unblocks server
  - then it sends Connection Accepted (CA) to Client transport entity
- Use SEND and RECEIVE primitives to exchange data.
- Use DISCONNECT to end connection.
**Disconnecting**

- **Asymmetric Disconnect**
  - Either transport user may issue the `DISCONNECT` primitive which results in a `DISCONNECT TPDU` being sent to the remote transport entity.
  - When TPDU arrives, connection released.
- **Symmetric Disconnect**
  - One side issues `DISCONNECT` to indicate no more data to send.
  - Connection not released until both sides issue `DISCONNECT` primitive.

**Transport Service Addresses**

- When process issues `CONNECT` primitive, it must specify "to what?"
- Answer is the access point of the service: Transport Service Access Point or TSAP.
  - In Internet these are IP address, port.
- These addresses either "well-known" or are generally available from a name server whose address is well-known.
  - Example: FTP port 21; TELNET port 23
Establishing a Connection

- Complicated because network can lose, store or duplicate packets.
  - Nightmare: packets pop out of network twice - each time requesting transfer of a large sum of money to an account.
- Dealing with delayed duplicates:
  - Change transport address with each request.
  - Number connections with an ID so you know if one is being recreated. (But machines crash...)
  - Better: Kill off aged packets inside the network.
- With bounded packet lifetimes, possible to establish connections safely.

Three-way Handshake

[Diagram showing the three-way handshake process]
Difficult to Tell if Connect/Disconnect REALLY Happened

Flow Control at Transport Layer

- Why is it needed?
  - network may be unreliable (connectionless)
  - Data Link Flow Control NOT end-to-end (only to receiving network layer)
  - Receiving transport layer may be out of buffers
- Sender buffers: when receiving transport layer cannot guarantee buffer availability. Receiver free to use shared (dynamic) buffering schemes.
- Receiver buffers: when it can guarantee buffer available. Usually dedicated space per connection (max TPDU x window size).
- May be extremely wasteful (single char min).
Buffer Management:

- May vary by traffic type
  - Low bandwidth/bursty traffic best handled by dynamic buffer allocation with sender buffering.
  - High-bandwidth traffic may best be handled by dedicated buffers at receiver.
- Sending host generally requests buffers at receiver (collectively or per connection)
- Receiver grants what it can afford and sender keeps track of number of unacknowledged TPDUs vs number of granted buffers.

Dynamic Buffer Allocation

<table>
<thead>
<tr>
<th>A</th>
<th>Message</th>
<th>B</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt; request 8 buffers&gt;</td>
<td></td>
<td>A wants 8 buffers</td>
</tr>
<tr>
<td>2</td>
<td>&lt;ack = 15, buf = 4&gt;</td>
<td></td>
<td>B grants messages 0-3 only</td>
</tr>
<tr>
<td>3</td>
<td>&lt;seq = 0, data = m0&gt;</td>
<td></td>
<td>A has 3 buffers left now</td>
</tr>
<tr>
<td>4</td>
<td>&lt;seq = 1, data = m1&gt;</td>
<td></td>
<td>A has 2 buffers left now</td>
</tr>
<tr>
<td>5</td>
<td>&lt;seq = 2, data = m2&gt;</td>
<td></td>
<td>Message lost but A thinks it has 1 left</td>
</tr>
<tr>
<td>6</td>
<td>&lt;ack = 1, buf = 3&gt;</td>
<td></td>
<td>B acknowledges 0 and 1, permits 2-4</td>
</tr>
<tr>
<td>7</td>
<td>&lt;seq = 3, data = m3&gt;</td>
<td></td>
<td>A has buffer left</td>
</tr>
<tr>
<td>8</td>
<td>&lt;seq = 4, data = m4&gt;</td>
<td></td>
<td>A has 0 buffers left, and must stop</td>
</tr>
<tr>
<td>9</td>
<td>&lt;seq = 2, data = m2&gt;</td>
<td></td>
<td>A times out and retransmits</td>
</tr>
<tr>
<td>10</td>
<td>&lt;ack = 4, buf = 0&gt;</td>
<td></td>
<td>Everything acknowledged, but A still blocked</td>
</tr>
<tr>
<td>11</td>
<td>&lt;ack = 4, buf = 1&gt;</td>
<td></td>
<td>A may now send 5</td>
</tr>
<tr>
<td>12</td>
<td>&lt;ack = 4, buf = 2&gt;</td>
<td></td>
<td>B found a new buffer somewhere</td>
</tr>
<tr>
<td>13</td>
<td>&lt;seq = 5, data = m5&gt;</td>
<td></td>
<td>A has 1 buffer left</td>
</tr>
<tr>
<td>14</td>
<td>&lt;seq = 6, data = m6&gt;</td>
<td></td>
<td>A is now blocked again</td>
</tr>
<tr>
<td>15</td>
<td>&lt;ack = 6, buf = 0&gt;</td>
<td></td>
<td>A is still blocked</td>
</tr>
<tr>
<td>16</td>
<td>&lt;ack = 6, buf = 4&gt;</td>
<td></td>
<td>Potential deadlock</td>
</tr>
</tbody>
</table>
**Transport Layer Multiplexing**

**Upward**

**Downward**

- If transport entity is within host (usual), it can easily recover from network/router crashes.
- If host crashes, counters will be reinitialized and host will not know where to begin.
- Suppose host asks client: "What state are you in?" Client says: "Waiting for the ack to TPDU 6." Host thinks it must have received TPDU 5 ok (because it ACKed 5) and asks for 6 again.
- BUT host may have already received TPDU 6 and passed it up to application (written to ap) before sending ACK 6. Just after writing and before ack, it crashed. In this case host would get a DUP of 6.
**TCP Service**

- Sender and receiver create end points (sockets).
  - Socket numbers consist of host IP address plus 16-bit port number.
  - To obtain TCP service, connection must be established between sockets on each end.
- Port numbers below 256 are called "well-known ports." (RFC 1700)
- All TCP connections are full-duplex, pt-to-pt.
- TCP connection is a byte stream (does not preserve application-level boundaries).

**TCP Segments**

- Sending and receiving TCP entities exchange data in segments.
  - Segment has a fixed 20-byte header (plus optional data) followed by data bytes.
  - Each segment must fit into the 65,535 byte IP payload max.
  - Each network also supports a maximum transfer unit (MTU).
  - If segment too large for a network, router may divide it into multiple segments (repeats the IP and segment header overhead).
Transport Layer May Support QoS Parameters

- Connection establishment delay
- Connection establishment failure probability
- Throughput
- Transit delay
- Residual error ratio
- Protection
- Priority
- Resilience

Problems

- Chapter 1: 5,7,14,16,18,26,27
- Chapter 3: 1,3,6,12,22,24,28
- Chapter 4: 3,4,19,20,28,40
- Chapter 5: 8,16,19,20,26,28,34,38
- Chapter 6: 1,2,3,6,7,14,22,23,31 through 37 (due Monday, 23rd)