Transport Layer in Internet

• Purpose 1: (De)multiplexing of data streams to different application processes
• Purpose 2: Provide value-added services that many applications want
  – Recall network layer in Internet provides a “Best-effort” service only, transport layer can add value to that
    • Application may want reliability, etc.
  – No need to reinvent the wheel each time you write a new application

Transport Protocols Concern only End Hosts, not Routers

• Lowest level end-to-end protocol.
  – Header generated by sender is interpreted only by the destination
  – Routers view transport header as part of the payload
• Adds functionality to the best-effort packet delivery IP service.
  – Make up for the shortcomings of the core network
(Possible) Transport Protocol Functions

- Multiplexing/demultiplexing for multiple applications.
  - Port abstraction
- Connection establishment.
  - Logical end-to-end connection
  - Connection state to optimize performance
- Error control.
  - Hide unreliability of the network layer from applications
  - Many types of errors: corruption, loss, duplication, reordering.
- End-to-end flow control.
  - Avoid flooding the receiver
- Congestion control.
  - Avoid flooding the network
- More….

User Datagram Protocol (UDP)

- Connectionless datagram
  - Socket: SOCK_DGRAM
- Port number used for (de)multiplexing
  - Port numbers = connection/application endpoint
- Adds end-to-end reliability through optional checksum
  - Protects against data corruption errors between source and destination (links, switches/routers, bus)
  - Does not protect against packet loss, duplication or reordering

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Dest. Port</th>
<th>Length</th>
<th>Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

Using UDP

- Custom protocols/applications can be implemented on top of UDP
  - Use the port addressing provided by UDP
  - Implement own reliability, flow control, ordering, congestion control as it sees fit
- Examples:
  - Remote procedure call
  - Multimedia streaming (real time protocol)
  - Distributed computing communication libraries
Transmission Control Protocol (TCP)

- Reliable bidirectional in-order byte stream
  - Socket: SOCK_STREAM
- Connections established & torn down
- Multiplexing/ demultiplexing
  - Ports at both ends
- Error control
  - Users see correct, ordered byte sequences
- End–end flow control
  - Avoid overwhelming machines at each end
- Congestion avoidance
  - Avoid creating traffic jams within network

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Dest. Port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Connection Setup

- Why need connection setup?
  - Mainly to agree on starting sequence numbers
    - Starting sequence number is randomly chosen
    - Reason, to reduce the chance that sequence numbers of old and new connections from overlapping

High Level TCP Features

- Sliding window protocol
  - Use sequence numbers
- Bi–directional
  - Each host can be a receiver and a sender simultaneously
    - For clarity, we will usually discuss only one direction
Important TCP Flags

- **SYN**: Synchronize
  - Used when setting up connection
- **FIN**: Finish
  - Used when tearing down connection
- **ACK**: Acknowledging received data

Establishing Connection

- **Three-Way Handshake**
  - Each side notifies other of starting sequence number it will use for sending
  - Each side acknowledges other's sequence number
  - SYN-ACK: Acknowledge sequence number + 1
  - Can combine second SYN with first ACK

TCP State Diagram: Connection Setup
**Tearing Down Connection**

- Either Side Can Initiate Tear Down
  - Send FIN signal
  - "I'm not going to send any more data"
- Other Side Can Continue Sending Data
  - Half open connection
  - Must continue to acknowledge
- Acknowledging FIN
  - Acknowledge last sequence number + 1

**State Diagram: Connection Tear-down**

**Sequence Number Space**

- Each byte in byte stream is numbered.
  - 32 bit value
  - Wraps around
  - Initial values selected at startup time
- TCP breaks up the byte stream in packets ("segments")
  - Packet size is limited to the Maximum Segment Size
  - Set to prevent packet fragmentation
- Each segment has a sequence number.
  - Indicates where it fits in the byte stream
Sequence Numbers

- 32 Bits, Unsigned
- Why So Big?
  - For sliding window, must have
    \[ |\text{Sequence Space}| > 2^{|\text{Sending Window}|} \]
  - 2^32 > 2^16: No problem
  - Also, want to guard against stray packets
    - With IP, assume packets have maximum segment lifetime (MSL) of 120s
      - i.e. can linger in network for up to 120s
    - Sequence number would wrap around in this time at 286Mbps

Error Control

- Checksum (mostly) guarantees end–end data integrity.
- Sequence numbers detect packet sequencing problems:
  - Duplicate: ignore
  - Reordered: reorder or drop
  - Lost: retransmit
- Lost segments detected by sender.
  - Use time out to detect lack of acknowledgment
  - Need estimate of the roundtrip time to set timeout
- Retransmission requires that sender keep copy of the data.

Bidirectional Communication

- Each Side of Connection can Send and Receive
- What this Means
  - Maintain different sequence numbers for each direction
  - Single segment can contain new data for one direction, plus acknowledgement for other
    - But some contain only data & others only acknowledgement

\[
\begin{align*}
\text{Send seq 2000} & \quad \text{Ack seq 2001} \\
\text{Send seq 42000} & \quad \text{Ack seq 42001}
\end{align*}
\]
TCP Flow Control

- Sliding window protocol
  - For window size $n$, can send up to $n$ bytes without receiving an acknowledgement
  - When the data are acknowledged then the window slides forward

- Window size determines
  - How much unacknowledged data can the sender sends

- But there is more detail

Complication!

- TCP receiver can delete acknowledged data only after the data has been delivered to the application
- So, depending on how fast the application is reading the data, the receiver’s window size may change!!!

Solution

- Receiver tells sender what is the current window size in every packet it transmits to the sender
- Sender uses this current window size instead of a fixed value

- Window size (also called Advertised window) is continuously changing
- Can go to zero!
  - Sender not allowed to send anything!
Window Flow Control: Receive Side

Receive buffer

Acked but not delivered to user  Not yet acked

window

Window Flow Control: Send Side

window

Sent and acked  Sent but not acked  Not yet sent

Next to be sent
Must retain for possible retransmission

Packet Sent

Packet Received

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Dest. Port</th>
<th>Sequence Number</th>
<th>Acknowledgment</th>
<th>HL/Flags</th>
<th>Window</th>
<th>D. Checksum</th>
<th>Urgent Pointer</th>
<th>Options..</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Dest. Port</th>
<th>Sequence Number</th>
<th>Acknowledgment</th>
<th>HL/Flags</th>
<th>Window</th>
<th>D. Checksum</th>
<th>Urgent Pointer</th>
<th>Options..</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

App write

acknowledged  sent  to be sent  outside window
Ongoing Communication

- Bidirectional Communication
  - Each side acts as sender & receiver
  - Every message contains acknowledgement of received sequence
    - Even if no new data have been received
  - Every message advertises window size
    - Size of its receiving window
  - Every message contains sent sequence number
    - Even if no new data being sent
- When Does Sender Actually Send Message?
  - When sending buffer contains at least max. segment size (header sizes) bytes
  - When application tells it
    - Set PUSH flag for last segment sent
  - When timer expires

What is Congestion?

- The load placed on the network is higher than the capacity of the network
  - Not surprising: independent senders place load on network
- Results in packet loss: routers have no choice
  - Can only buffer finite amount of data
  - End-to-end protocol will typically react, e.g. TCP

Why is Congestion Bad?

- Wasted bandwidth: retransmission of dropped packets
- Poor user service: unpredictable delay, low user goodput
- Increased load can even result in lower network goodput
  - Switched nets: packet losses create lots of retransmissions
  - Broadcast Ethernet: high demand -> many collisions
Sending Rate of Sliding Window Protocol

- Suppose A uses a sliding window protocol to transmit a large data file to B
- Window size = 64KB
- Network round-trip delay is 1 second

- What's the expected sending rate?
  - 64KB/second

- What if a network link is only 64KB/second but there are 1000 people who are transferring files over that link using the sliding window protocol?
  - Packet losses, timeouts, retransmissions, more packet losses... nothing useful gets through, congestion collapse!

TCP Window Flow Control

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Dest. Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Number</td>
<td>Acknowledgment</td>
</tr>
<tr>
<td>HL/Flags</td>
<td>Window</td>
</tr>
<tr>
<td>D. Checksum</td>
<td>Urgent Pointer</td>
</tr>
<tr>
<td>Options...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Dest. Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Number</td>
<td>Acknowledgment</td>
</tr>
<tr>
<td>HL/Flags</td>
<td>Window</td>
</tr>
<tr>
<td>D. Checksum</td>
<td>Urgent Pointer</td>
</tr>
<tr>
<td>Options...</td>
<td></td>
</tr>
</tbody>
</table>

TCP Flow Control Alone Is Not Enough

- We have talked about how TCP's advertised window is used for flow control
  - To keep sender sending faster than the receiver can handle
- If the receiver is sufficiently fast, then the advertised window will be maximized at all time
- But clearly, this will lead to congestion collapse as the previous example if there are too many senders or network is too slow

- Key 1: Window size determines sending rate
- Key 2: Window size must be dynamically adjusted to prevent congestion collapse
How Fast to Send? What’s at Stake?

• Send too slow: link sits idle
  – wastes time

• Send too fast: link is kept busy but....
  – queue builds up in router buffer (delay)
  – overflow buffers in routers (loss)
  – Many retransmissions, many losses
  – Network goodput goes down

[Graph showing Goodput vs. Load with a safe operating point]

Load

Goodput

safe operating point