Transmission Control Protocol

- A TCP entity accepts user data *streams* from local processes, breaks them up into pieces not exceeding 64Kbytes, and sends each piece as a separate IP datagram.
- The IP layer gives no guarantee that datagrams will be delivered properly, so it is up to TCP to *timeout* and *retransmit* them as need be. It is also up to TCP to *reassemble* out-of-order datagrams into messages in proper sequence.
TCP Service Model

- TCP service is obtained by having both sender and receiver create end points, called sockets.
- Socket number (address) = IP address + 16bit port number
- A socket may be used for multiple connections at the same time. Connections are identified by the socket identifiers at both ends. (socket 1, socket 2).
  \[(1,2), (3,2)\]
- Port numbers below 256 are called well-known ports.
- All TCP connections are full-duplex and point-to-point.
- A TCP connection is a byte stream, not a message stream. Message boundaries are not preserved end to end.
TCP Service Model

- To force data out, use the PUSH flag. (e.g., for carriage return)
- For urgent data (e.g., Del, CTRL-C), use URGENT flag that causes TCP to stop accumulating data and transmit immediately. At the destination, the receiving application is interrupted, so it can stop whatever it was doing and read the data stream to find the urgent data.
TCP Protocol

- TCP PDU = segment: 20-byte header + option + data
- Two limits on segment size:
  - fit in the 65,535 byte IP payload
  - fit in the network MTU (Maximum Transfer Unit).
- TCP segment header
  - **source port**: 16 bits.
  - **destination port**: 16 bits.
  - **sequence number**: 32 bits; every byte of data is numbered.
  - **acknowledgement number**: 32 bits; the next byte expected.
  - **TCP header length**: 4 bits; in words (32 bits).
  - **unused**: 6 bits.
  - **URG**: 1 bit; urgent pointer in use.
  - **ACK**: 1 bit; ACK number is valid.
  - **PSH**: 1 bit; PUSH flag; receiver is requested to deliver data to application upon arrival.
TCP Protocol

- **RST**: 1 bit; reset a connection
- **SYN**: 1 bit; connect.
  
  SYN = 1, ACK = 0  Connect Request
  SYN = 1, ACK = 1  Connect Accepted
  ACK = 1  Connect (confirmed)
- **FIN**: 1 bit; release.
- **window size**: 16 bits; receiver buffer space; how many bytes may be sent starting at the byte acknowledged.
- **checksum**: 16 bits; checksums header, data, and pseudo header (source IP + destination IP + 0006h + TCP segment length (16 bits)). Add up all 16-bit words in 1’s complement, and then take the 1’s complement of the sum.
- **urgent pointer**: 16 bits; a byte offset from the current sequence number at which urgent data are to be found.
TCP Option Fields

• Allow each host to specify the maximum TCP payload it is willing to accept (during connection set up).
  - Default maximum segment size (MSS) is 536 byte
  - All Internet hosts are required to accept TCP segment of $536 + 20 = 556$ bytes.

• For high bandwidth-delay product network, the window size needs to be larger than 64K bytes (16 bits). Window scale option allows both side to shift the window size field up to 16 bits to the left (up to $2^{\wedge\wedge}32$ bytes).

• Allow selective repeat instead of go back n protocol. (introduce NAKs).
TCP Connection management

- 3-way handshake

Connection Request

SYN(seq=x)

Connection Accepted

SYN(seq=y, ack=x+1)

Connection confirmed

SYN(seq=x+1, ack=y+1)
TCP Transmission Policy

- Window management in TCP is not directly tied to ACK as it is in most data link protocol.
- As shown in Fig. 6-29, ACK does not allow sender to transmit more. The Window field explicitly tells the sender how much it can transmit.
- Receiver are not required to send ACK as soon as possible.

Ex. Telnet connection, interactive editor that react to every key stroke.
TCP Transmission Policy

- Optimization Approach
  - delay ACK and window updates for 500 msec.
    - if the editor can echo within 500
      - still not efficient by sending 1 byte in 41-byte packet.
  - Nagle’s algorithm: when data come into the sender one byte at a time, just send the first byte and buffer all the rest until the outstanding byte is ACKed. Then send all the buffered bytes in one TCP segment and start buffering again until they are ACKed.
    - widely used
    - not good for X-window application.
Silly Window Syndrome

- Data are passed to the sending TCP entity in large blocks.
- An interactive application on the receiving side reads data 1 byte at a time.

- Fig. 6-30

Receiver’s buffer is full

Application reads one byte

Room for one more byte

Window update segment sent

New byte arrives

Receiver’s buffer is full

- Clark’s solution: receiver should not send a window update until it can handle the maximum segment size it advertised (when the connection was established) or its buffer is half empty, whichever is smaller.

- The sender can also help by not sending tiny segments.

- Nagle’s algorithm (w.r.t. sender) and Clark’s solution (w.r.t. receiver) are complementary.
TCP Congestion Control

- Solution to congestion is to slow down the data rate.
- Detecting congestion
  - Time out is due to lost/erred packets on lines, lost/delayed packets at congestion
- Causes of congestion
  1. due to insufficient network capacity
  2. due to insufficient receiver capacity
- Each sender maintains two windows
  1. The window the receiver has granted
  2. Congestion window (maintained by the sender)
Minimum of the two is the number of bytes that can be sent.
TCP Congestion Control

• Initially, congestion window = MSS.
• Sender sends one max. segment. If it is ACKed before the timer goes off, congestion window is doubled. It then sends two segments.
• As each of these segments is ACKed, the congestion window is increased by one MSS.
• Congestion window keeps growing exponentially until either a timeout occurs or the receiver’s window is reached. -- **Slow Start** algorithm.
• Internet uses a third parameter, threshold.
  - initially, threshold is at 64Kbytes
  - when a time out occurs, threshold is set to half of the current congestion window, and the congestion window is reset to one MSS.
TCP Congestion Control

- Slow start is then used to determine what the network can handle, except the exponential growth stops when the threshold is hit. From that point on, successful transmissions grow the congestion window linearly (by one max. segment for each burst, instead of one per segment), up to the size of the receiver’s window.
TCP Timer Management

- For each connection, TCP maintains a variable, RTT, that is the best current estimate of the round-trip time to the destination.

- When a segment is sent, a (retransmission) timer is started, both to see how long (M) the ACK takes and to trigger a retransmission if it takes too long.
  \[ \text{RTT} = a \times \text{RTT} + (1 - a) \times M, \quad (a=7/8) \]
  \[ \text{Time-out} = b \times \text{RTT} \quad (b=?) \]

Jacobson: \( D = \text{mean deviation of RTT} \)
\[ D = a \times D + (1 - a) \times |\text{RTT} - M| \]
\[ \text{Time-out} = \text{RTT} + b \times D, \quad (b = 4) \]
TCP Timer Management

- What to do when a segment times out and is sent again?
- When the ACK comes in, it is unclear whether the ACK refers to the first transmission or the later one.
- **Karn’s algorithm**: Do not update RTT an any segments that have been retransmitted. Instead, the timeout is doubled on each failure until the segments get through the first time.
Wireless TCP

- In theory, transport protocols should be independent of the technology of the underlying network layer.
- In practice, it does matter because most TCP implementation have been optimized based on assumptions that are true for wired networks but which fail for wireless networks.
- The principal problem is the congestion control algorithm.
- Most TCP implementations assume that timeouts are caused by congestion, not by lost packets.
- Consequently, when a timer goes off, TCP slows down and sends less vigorously.
Wireless TCP

- Wireless transmission links are highly unreliable.
- The proper approach to dealing with lost packets is to send them again, and as quickly as possible. Slowing down just makes matters worse.
- When the sender does not know what the network is, it is difficult to make the correct decision.
- For example, consider an inhomogeneous end-to-end path.
Wireless TCP

- Solution 1: **Indirect TCP**
- Split the TCP connection into two separate connections.
- Timeouts on the first connection slow the sender down, whereas timeouts on the second one speed it up.
- Violates the semantics of TCP (not end-to-end; base station performs transport layer functions).
Wireless TCP

- Solution 2: **Snooping**
- Add a snooping device agent at the base station.
- The agent observes and caches TCP segments going out to the mobile host, and ACKs coming back from it.
- If ACK does not come back within the (shorter) timeout interval, the agent retransmits the segment.
- If the agent sees duplicate ACKs from the mobile hosts (meaning the mobile host missed something), it generates a retransmission. Duplicate ACKs are discarded.
- If the wireless link is very lossy, the source may time out waiting for an ACK and invoke the congestion control algorithm.
- Selective repeat for missing bytes on reverse traffic.