The Transport Layer

Michael Brodskiy

Professor: E. Bernal Mor

October 11, 2023

- Transport Services and Protocols
  - Provide logical communication between application processes running on different hosts
  - Transport protocols actions in end systems:
    * Sender: breaks application messages into segments, passes to Network layer
    * Receiver: reassembles segments into messages, passes to Application layer
  - Two transport protocols available to internet applications
    1. TCP
    2. UDP
- Transport vs. Network Layer
  - Network layer: logical communication between two hosts
  - Transport layer: logical communication between processes
    * Relies on, enhances, network layer services
- Two Internet Transport Protocols
  - TCP: Transmission Control Protocol
    * Reliable, in-order delivery
    * Congestion Control
    * Flow Control
    * Connection set-up
  - UDP: User Datagram Protocol
    * Unreliable, unordered delivery
    * No-frills extension of "best-effort" IP

- Services not available:
  - Delay guarantees
  - Throughput guarantees

- Multiplexing/Demultiplexing
  - Multiplexing at sender: Handle data from multiple sockets, add transport header (later used for demultiplexing)
  - How demultiplexing works
    - Host receives IP packets
      - Each packet has source IP address, destination IP address
      - Each packet carries one transport-layer segment
      - Each segment has source, destination port number
    - Host uses IP address and port numbers to direct segment to appropriate socket
  - Connectionless Demultiplexing
    - Create a socket in the client, the Transport layer automatically assigns a host-local port number to the socket
    - When data is sent into UDP socket, must specify
      - Destination IP address
      - Destination port number
    - When a host receives UDP segment, the Transport layer:
      - Checks destination port number in segment
      - Directs UDP segment to socket with that port number
    - IP datagrams with same destination port number but different source IP addresses and/or source port numbers will be directed to same socket at destination
  - Connection-Oriented Demultiplexing
    - TCP socket identified by 4-tuple:
      - Source IP address
      - Source port number
      - Destination IP address
      - Destination port number
    - Demultiplexing receiver users all four values to direct segment to appropriate socket
    - A server may support simultaneous TCP sockets:
      - Each socket identified by its own 4-tuple
      - Each socket associated with a different connecting client

        * Note: the TCP server has a welcoming socket
            · Each time a client initiates a TCP connection to the server, a new socket is created for this connection
            · To support n simultaneous connections, the server would need $n + 1$ sockets

- Connectionless Transport: UDP
    - "No frills", "bare bones" Internet transport protocol
    - "Best effort" service, UDP segments may be:
        * Lost
        * Delivered out-of-order to application
    - Connectionless:
        * No handshaking between UDP sender, receiver
        * Each UDP segment handled independently of others
    - Why is there a UDP?
        * No connection establishment (which can add RTT delay)
        * Simple: no connection state at sender, receiver
        * Small header size
        * No congestion control
            · UDP can blast away as fast as desired
            · It can function in the face of congestion
    - UDP used in:
        * Streaming multimedia apps (loss tolerant, rate sensitive)
        * DNS
        * HTTP/3
    - If reliable transfer or other services needed over UDP (like in HTTP/3)
        * Add needed reliability at application layer
        * Add congestion control at application layer

- Internet Checksum
    - Goal: detect "errors" (*e.g.* flipped bits) in received segment — optional if UDP segment is encapsulated in IPv4 packet: carries all-zeros if unused
    - Sender:
        * Treat segment contents, including UDP header fields and IP addresses, as sequence of 16-bit words
        * Checksum: ones complement of the ones complement sum of all 16-bit words

    - Sender puts checksum value into UDP checksum field
  - Receiver — error detected?
    - All 16-bit words are added, including checksum
    - Result = 1111111111111111 $\rightarrow$ no error
      - But maybe errors, nonetheless?
    - Result $\neq$ 1111111111111111 $\rightarrow$ Error
      - Do not recover from the error: discard segment or pass damaged data with a warning

- Principles of Reliable Data Transfer
  - Consider only unidirectional data transfer
  - But control information will flow in both directions
  - Incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)

- Components of Error Recovery
  - Checksum to detect bit errors
  - Acknowledgments (ACKs): receiver explicitly tells sender the packet received is OK
  - Negative Acknowledgments (NAKs): receiver explicitly tells sender that pack had errors
  - Sender retransmits packet on receipt of NAK
  - What happens if ACK/NAK is corrupted?
    - Sender doesn't know what happened at receiver
    - Can't just retransmit, possible duplicate
  - Sender:
    - Sequence number added to packet
    - Two sequence numbers will suffice
    - Must check if received ACK/NAK corrupted
    - Sender must "remember" whether last sent packet had sequence number of 0 or 1
  - Receiver:
    - Must check if received packet is duplicate
      - Receiver must remember whether 0 or 1 is expected packet sequence number
    - If duplicate, send ACK to force sender to move on
    - Note: receiver can not know if its last NAK/ACK received ok at sender

- A similar protocol without NAKs may be developed:
  * Instead of NAK, receiver sends ACK for last packet received OK
    · Receiver must explicitly include in the ACK the sequence number of packet being ACKed
  * Duplicate ACK at sender results in same action as NAK: retransmit current packet
  * TCP uses this approach to be NAK-free
- This protocol may be further developed:
  * Approach: sender waits "reasonable" amount of time for ACK
  * Only retransmit if no expected ACK received in this time
  * If packet (or ACK) just delayed (not lost):
    · Retransmission will be duplicate, but sequence number already handles this
    · Receiver must specify sequence number of packet being ACKed
  * Requires countdown timer to interrupt after "reasonable" amount of time (timeout)

• Utility
  - For a "Stop and Wait" approach, the utility can be defined as:

$$U_{sender} = \frac{L/R}{RTT + (L/R)}$$

  - Thus, the performance of stop and wait would not be food if the RTT is high
  - Stop and wait limits performance of underlying infrastructure

• Pipelined Protocols
  - Pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged packets
    * Range of sequence numbers must be increaseL k-bit sequence number in packet header
    * Buffering at sender and/or receiver
  - Two generic forms of pipelined protocols: Go-Back-N (GBN) and selective repeat