Justin Ciocoi

Oct. 4, 2023

CSCI 379

Computer Networking

Transport Layer

Transport vs. Network Layer

- The network layer performs logical communication between hosts whereas the transport layer performs logical communication between processes which are reliant on network layer services
- **Internet Transport Layer Protocols**
  - TCP
    - Reliable
    - In order
    - Congestion controlled
    - Flow controlled
    - Connection handshake
  - UDP
    - No-frills
    - Extension of "best-effort" IP
    - UDP provides no delay or bandwidth guarantees

Multiplexing and Demultiplexing

- When data is sent from a sender, additional data is added on in the form of a header
  - This is referred to as *multiplexing*
- When the data is received by the receiver, the information present in the header will be used to *demultiplex* the data, or ensure it reaches the correct destination

User Datagram Protocol (UDP)

- UDP is often described as *bare-bones, no-frills, or best effort*, which indicates the lesser quality of the services it provides when compared to TCP

- When using UDP, segments could be
  - Lost
  - Delivered out of order to the recipient
- UDP is a **connectionless protocol**
  - There is no handshake between sender and receiver as there is with TCP
  - Each UDP segment is handled independently of all the other UDP segments
- *When* is UDP most commonly used?
  - Streaming multimedia applications, since they are loss tolerant (can afford to lose some segments) and rate sensitive (are heavily impacted by TCPs rate-limiting congestion control)
  - DNS
  - SNMP (Simple Network Management Protocol, which is a protocol used to monitor and manage network connected devices)
- Reliable transfer over UDP
  - Reliable transfer can be achieved over UDP by implementation at the application level
- UDP segment header
  - Source port number (16-bit)
  - Destination port number (16-bit)
  - Length of UDP segment, including header (16-bit)
  - Checksum (16-bit)
  - The UDP header has a fixed size of *64 bits* (8 bytes)

UDP Checksum

- Treat the segment contents as a string of 16-bit integers
- The checksum will be the ones complement sum of segment contents
- Sender puts the checksum in header

- Receiver calculates its own checksum and compares it to the sent checksum
  - If they match, no error is detected (there could still be errors)
  - If they do not, errors are detected
- Example for two 16-bit integers
  - $$1110011001100110 + 1101010101010101 = \\ (1)1011101110111011 = 1011101110111100$$
  - The above is the sum, and the checksum is the ones complement of the sum, or 0100010001000011

Reliable Data Transfer (TCP)

- If the channel is error/loss free
  - Sender sends
  - Receiver receives
  - No feedback is needed
- If the channel is unreliable in the sender to receiver direction
  - Sender sends
  - Receiver sends acknowledgement (ACK)
- If the channel is unreliable in both directions
  - Sender sends
  - Receiver sends ACK
  - Both or one *can* be lost
  - If data or ACK is lost is lost, sender is stuck forever
    - Thus, you should use timers to make the sender re-transmit data
- Sequence numbers can also be used in order for the receiver to be able to differentiate duplicate packets
- Take a 1Gbps link, 15 ms prop. delay, and an 8000-bit packet:

- $D_{trans} = \frac{L}{R} = \frac{8000 bits}{10^9 bits/sec} = 8$
- U_{sender}: utilization, or the fraction of time that the sender is busy sending
  - $U_{sender} = \frac{L/R}{RTT + L/R} = \frac{0.008}{30.008} = 0.00027 \approx 0.03\%$

Go-Back-N

- The sender can have up to N unacknowledged packets in the pipeline
- The receiver will only send a *cumulative acknowledgement* and won't acknowledge if there is a gap
- The sender has a **timer** for the oldest unacknowledged packer
  - When the timer expires, all unacknowledged packets will be retransmitted

Selective Repeat

- The sender can have up to N unacknowledged packets in the pipeline
- The receiver will send *individual acknowledgments* for each packet
- Similar timer system as the Go-Back-N protocol, but it operates on a packet-by-packet basis

TCP Overview

- Connection-oriented
- Reliable
- Point-to-point
- Pipelined
- Full duplex
  - Bidirectional simultaneous data transfer
- Flow Controlled
- TCP **segment structure**
  - Source and Destination Ports
    - 16-bits each

- Sequence number
  - 32-bits
- Acknowledgement number
  - 32-bits
- Other fields like checksum, receive window (`rwnd`) and Urgent data pointer are used
- Options field and application data field follow the above and are of variable length
- **TCP Header** is **20 bytes** long

- The TCP specifications do not specify how to deal with handling out of order segments, and this is thus up to the implementor

TCP Timeouts

- How does TCP set the **timeout value**, or the amount of time that should pass before packet retransmission
  - It should be *longer* than the **RTT**
  - If its too *short* premature timeouts will occur and cause unnecessary retransmissions
  - If its too *long* the connection will react slowly to segment loss
- How does TCP estimate RTT?
  - A variable `SampleRTT` is the time between segment transmission and acknowledgement receipt
    - Retransmissions are ignored
  - The `SampleRTT` will vary, and a smoother Estimated RTT is desired
    - Therefore, TCP will average *several recent measurements* to Estimate the value of RTT
  - $EstimatedRTT = (1 - \alpha) * EstimatedRTT + \alpha * SampleRTT$
    - A typical value for α is 0.125
  - A safety margin is added as a buffer to the `EstimatedRTT`
    - $DevRTT = (1 - \beta) * DevRTT + \beta * |SampleRTT - EstimatedRTT|$

            - Typically, $\beta = 0.25$
    - So, $TimeoutInterval = EstimatedRTT + 4 * DevRTT$

TCP Sender Events

- When data is received from the app
    - Creates a segment with a sequence number
    - Starts the timer if it is not already running
- Upon timeout
    - Retransmit segment that caused timeout
    - Restart timer
- Upon reception of acknowledgement
    - Update what is known to be acknowledged
    - Start timer for any still unacknowledged segments
- Retransmission Scenarios
    - When the acknowledgement is lost
        - packet is retransmitted after timeout is reached
    - When a premature timeout occurs
        - Receiver receives duplicate packets and sends duplicate acknowledgements