GBN Question

Assume you have a Go-Back-N (GBN) sender and receiver. The receiver acknowledges each data segment with a cumulative ACK which indicates the next expected data segment. Furthermore, it saves out-of-order segments in a buffer. The sender and receiver buffer can contain four segments each. The time-out period is much larger than the time required for the sender to transmit four segments in a row.

- The sender wants to transmit 10 data segments (0,. . . ,9) to the receiver. Assume that exactly one segment is lost. How many segments has the sender to transmit in the best and worst case? For each case, indicate which segment was lost

GBN Question

Assume you have a Go-Back-N (GBN) sender and receiver. The receiver acknowledges each data segment with a cumulative ACK which indicates the next expected data segment. Furthermore, it saves out-of-order segments in a buffer. The sender and receiver buffer can contain four segments each. The time-out period is much larger than the time required for the sender to transmit four segments in a row.

- The sender wants to transmit 10 data segments (0,. . . ,9) to the receiver. Assume that exactly one segment is lost. How many segments has the sender to transmit in the best and worst case? For each case, indicate which segment was lost
  - Best case: 11 segments, the last segment is dropped.
  - Worst case: 14 segments, e.g., if the second segment is dropped. GBN will retransmit all packets in the current window

GBN Question

Assume you have a Go-Back-N (GBN) sender and receiver. The receiver acknowledges each data segment with a cumulative ACK which indicates the next expected data segment. Furthermore, it saves out-of-order segments in a buffer. The sender and receiver buffer can contain four segments each. The time-out period is much larger than the time required for the sender to transmit four segments in a row.

- Once again, the sender wants to transmit 10 data segments (0,..., 9) to the receiver. This time, assume that exactly one ACK is lost. How many segments does the sender have to transmit in the best and worst case and which ACK was lost?

GBN Question

Assume you have a Go-Back-N (GBN) sender and receiver. The receiver acknowledges each data segment with a cumulative ACK which indicates the next expected data segment. Furthermore, it saves out-of-order segments in a buffer. The sender and receiver buffer can contain four segments each. The time-out period is much larger than the time required for the sender to transmit four segments in a row.

- Once again, the sender wants to transmit 10 data segments (0,..., 9) to the receiver. This time, assume that exactly one ACK is lost. How many segments does the sender have to transmit in the best and worst case and which ACK was lost?
  - Best case: 10 segments, e.g., the ACK for segment 5 is lost. Since GBN uses cumulative ACKs, the ACK for segment 6 implicitly also acknowledges segment 5.
  - Worst case: 11 segments, the ACK for the very last segment is lost.

UDP and TCP

What do we need in the transport layer?

- Application layer
  - Communication for specific applications
  - e.g., HyperText Transfer Protocol (HTTP), File Transfer Protocol (FTP)

- Network layer
  - Global communication between hosts
  - Hides details of the link technology
  - e.g., Internet Protocol (IP)

What Problems Should Be Solved Here?

- Data delivering, to the correct application
    - IP just points towards next protocol
    - Transport needs to demultiplex incoming data (ports)
- Files or bytestreams abstractions for the applications
    - Network deals with packets
    - Transport layer needs to translate between them
- Reliable transfer (if needed)
- Not overloading the receiver
- Not overloading the network

What Is Needed to Address These?

- Demultiplexing: identifier for application process
    - Going from host-to-host (IP) to process-to-process
- Translating between bytestreams and packets:
    - Do segmentation and reassembly
- Reliability: ACKs and all that stuff
- Corruption: Checksum
- Not overloading receiver: “Flow Control”
    - Limit data in receiver’s buffer
- Not overloading network: “Congestion Control”

UDP: Datagram messaging service

UDP provides a connectionless, unreliable transport service

- No-frills extension of “best-effort” IP

- UDP provides only two services to the App layer
  - Multiplexing/Demultiplexing among processes
  - Discarding corrupted packets (optional)

TCP: Reliable, in-order delivery

TCP provides a connection-oriented, reliable, bytestream transport service

- What UDP provides, plus:
    - Retransmission of lost and corrupted packets
    - Flow control (to not overflow receiver)
    - Congestion control (to not overload network)
    - “Connection” set-up & tear-down

Connections (or sessions)

Reliability requires keeping state

- Sender: packets sent but not ACKed, and related timers
- Receiver: noncontiguous packets

Each bytestream is called a connection or session

- Each with their own connection state
- State is in hosts, not network!

Important Context: Sockets and Ports

- Sockets: an operating system abstraction

- Ports: a networking abstraction
    - This is not a port on a switch (which is an interface)
    - Think of it as a logical interface on a host

Sockets

- A socket is a software abstraction by which an application process exchanges network messages with the (transport layer in the) operating system
    - socketID = socket(..., socket.TYPE)
    - socketID.sendto(message, ...)
    - socketID.recvfrom(...)

- Two important types of sockets
    - UDP socket: TYPE is SOCK_DGRAM
    - TCP socket: TYPE is SOCK_STREAM

Ports

- Problem: which app (socket) gets which packets
- Solution: port as transport layer identifier (16 bits)
  - Packet carries source/destination port numbers in transport header
- OS stores mapping between sockets and ports
  - Port: in packets
  - Socket: in OS

More on Ports

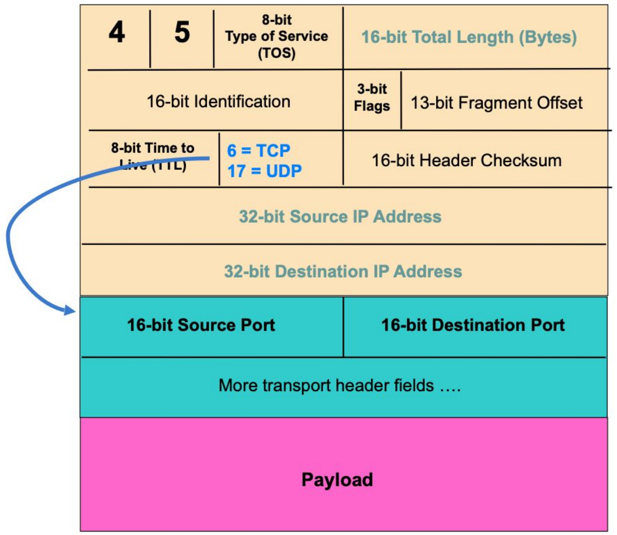
- Separate 16-bit port address space for UDP, TCP

- “Well known” ports (0-1023)
  - Agreement on which services run on these ports
  - e.g., ssh:22, http:80
  - Client (app) knows appropriate port on server
  - Services can listen on well-known port

- Ephemeral ports (most 1024-65535):
  - Given to clients (at random)

Multiplexing and Demultiplexing

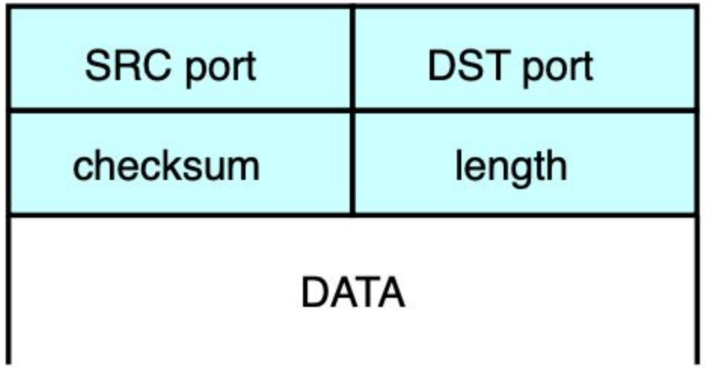
- Host receives IP datagrams
  - Each datagram has source and destination IP address,
  - Each segment has source and destination port number
- Host uses IP addresses and port numbers to direct the segment to appropriate socket



UDP: User Datagram Protocol

- Lightweight communication between processes
  - Avoid overhead and delays of ordered, reliable delivery
  - Send messages to and receive them from a socket

- UDP described in RFC 768 – (1980!)
  - IP plus port numbers to support (de)multiplexing
  - Optional error checking on the packet contents
    - (checksum field = 0 means “don’t verify checksum”)



Why Would Anyone Use UDP?

- Finer control over what data is sent and when
    - As soon as an application process writes into the socket
    - … UDP will package the data and send the packet
- No delay for connection establishment
    - UDP just blasts away without any formal preliminaries
    - … which avoids introducing any unnecessary delays
- No connection state
    - No allocation of buffers, sequence #s, timers …
    - … making it easier to handle many active clients at once
- Small packet header overhead
    - UDP header is only 8 bytes

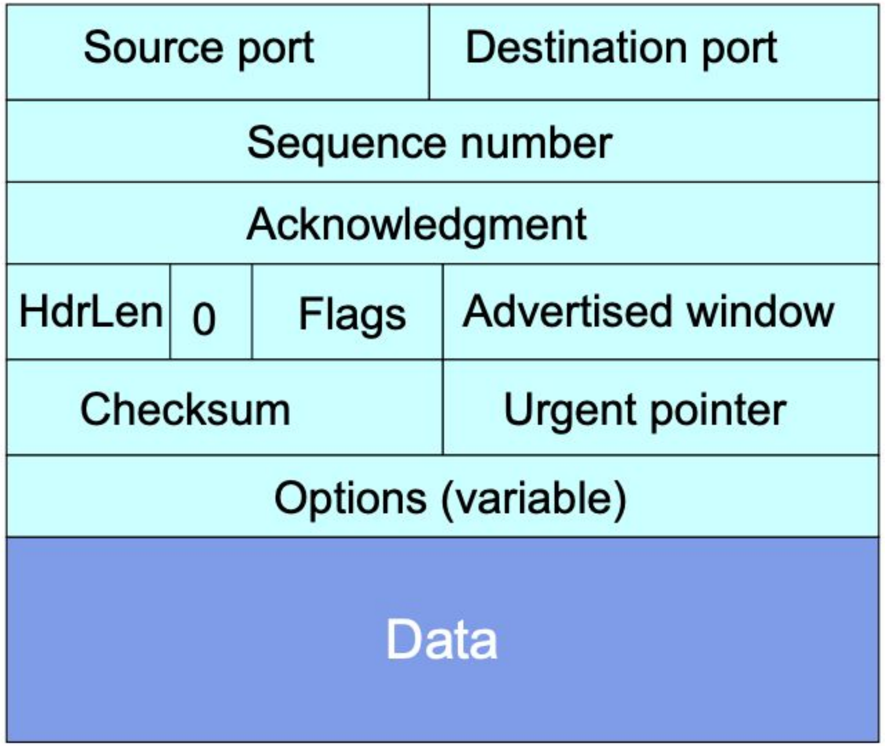
Basic Components of Reliability

- ACKs
  - Can’t be reliable without knowing whether data has arrived
  - TCP uses byte sequence numbers to identify payloads
- Checksums
  - Can’t be reliable without knowing whether data is corrupted
  - TCP does checksum over TCP and pseudoheader
- Timeouts and retransmissions
  - Can’t be reliable without retransmitting lost/corrupted data
  - TCP retransmits based on timeouts and duplicate ACKs
  - Timeout based on estimate of RTT

Other TCP Design Decisions

- Sliding window flow control
  - Allow W contiguous bytes to be in flight
- Cumulative acknowledgements
  - Selective ACKs (full information) also supported
- Single timer set after each payload is ACKed
  - Timer is effectively for the “next expected payload”
  - When timer goes off, resend that payload and wait
    - And double timeout period
- Various tricks related to “fast retransmit”
  - Using duplicate ACKs to trigger retransmission

TCP Header



Source port
Destination port
Sequence number
Acknowledgment
HdrLen
0
Flags
Advertised window
Checksum
Urgent pointer
Options (variable)
Data