Transport Protocols

End-to-End Protocols

- Underlying network is *best-effort*
  - drop messages
  - re-orders messages
  - delivers duplicate copies of a given message
  - limits messages to some finite size
  - delivers messages after an arbitrarily long delay
- Common end-to-end services
  - guarantee message delivery
  - deliver messages in the same order they are sent
  - deliver at most one copy of each message
  - support arbitrarily large messages
  - support synchronization
  - allow the receiver to flow control the sender
  - support multiple application processes on each host

Basic function of transport layer

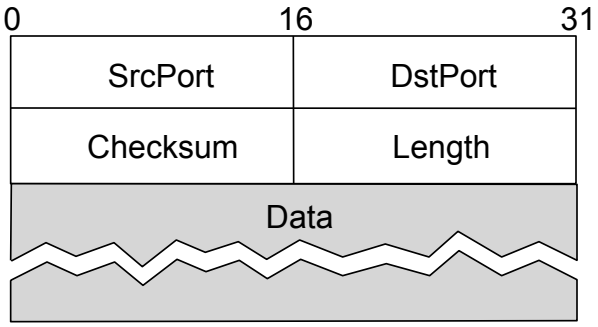
- How can processes on different systems get the right messages?
- *Ports* are numeric locators which enable messages to be demultiplexed to proper process.
  - Ports are addresses on individual hosts, not across the Internet.
- Ports are established using *well-know* values first
  - Port 80 = http, port 53 = DNS
- Ports are typically implemented as message queues
- Simplest function of the transport layer: multiplexing/ demultiplexing of messages
  - Enables processes on different systems to communicate
  - End-to-end since only processes on end hosts invoke this protocol

Other transport layer functions

- Connection control
  - Setting up and tearing down communication between processes
- Error detection within packets – our first focus
  - Checksums
- Reliable, in order delivery of packets – our second focus
  - Acknowledgement schemes
- Flow control
  - Matching sending and receiving rates between end hosts
- Congestion control
  - Managing congestion in the network

User Datagram Protocol (UDP)

- Unreliable and unordered *datagram* service
- Adds multiplexing/demultiplexing
- Adds reliability through optional checksum
- No flow or congestion control
- Endpoints identified by ports
  - servers have *well-known* ports
  - see **`/etc/services`** on Unix
- Header format

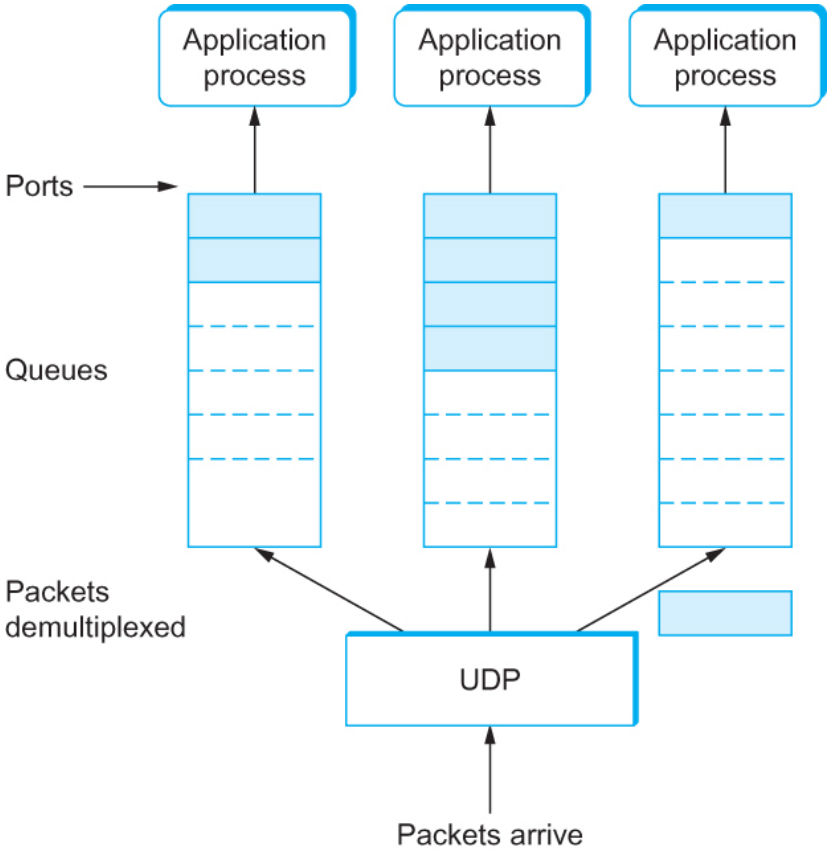


- Optional checksum
  - Computed over psuedo header + UDP header + data

Simple Demultiplexer (UDP)

- Extends host-to-host delivery service of the underlying network into a process-to-process communication service
- Adds a level of demultiplexing which allows multiple application processes on each host to share the network

Simple Demultiplexer (UDP)



UDP Message Queue

UDP Checksums

- Optional in current Internet
  - Computed over UDP header + body (data) + pseudo header
- Psuedoheader consists of 3 fields from IP header: protocol number (TCP or UDP), IP src, IP dst and UDP length field
  - Psuedoheader enables verification that message was delivered between correct source and destination.
  - IP dest address was changed during delivery, checksum would reflect this
- UDP uses the same checksum algorithm as IP
  - Internet checksum

UDP in practice

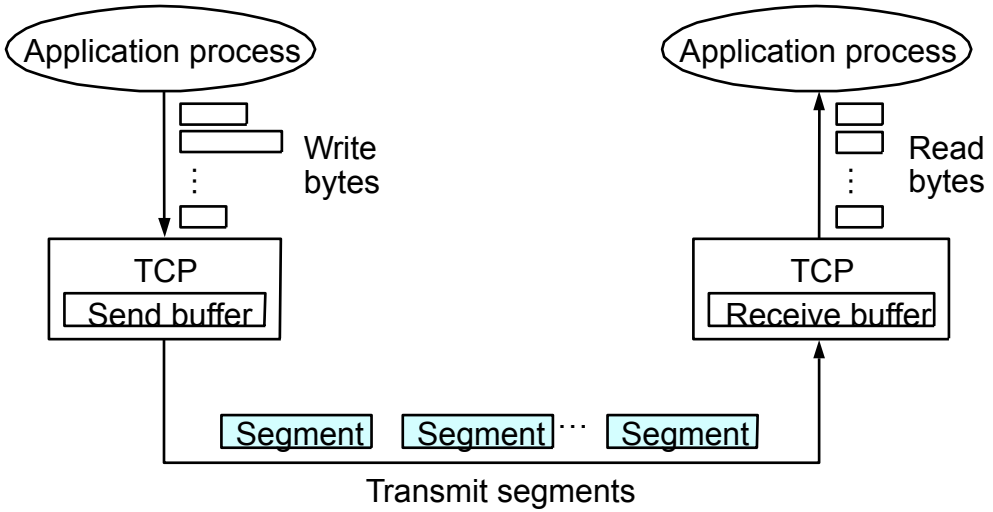
- Minimal specification makes UDP very flexible
  - Any kind of end-to-end protocol can be implemented
    - TCP can be implemented using UDP
- Examples
  - Most commonly used in multimedia applications
    - These are frequently more robust to loss
  - RPC's
  - Many others…

TCP Overview

- TCP is the most widely used Internet protocol
  - Web, Peer-to-peer, FTP, telnet, …
- A two way, reliable, byte stream oriented end-to-end protocol
  - Includes flow and congestion control
- Closely tied to the Internet Protocol (IP)
- A focus of intense study for many years
  - Our goal is to understand the RENO version of TCP
    - RENO is most widely used TCP today
    - RFC 2001 (now expired)
    - RENO mainly specifies mechanisms for dealing with congestion

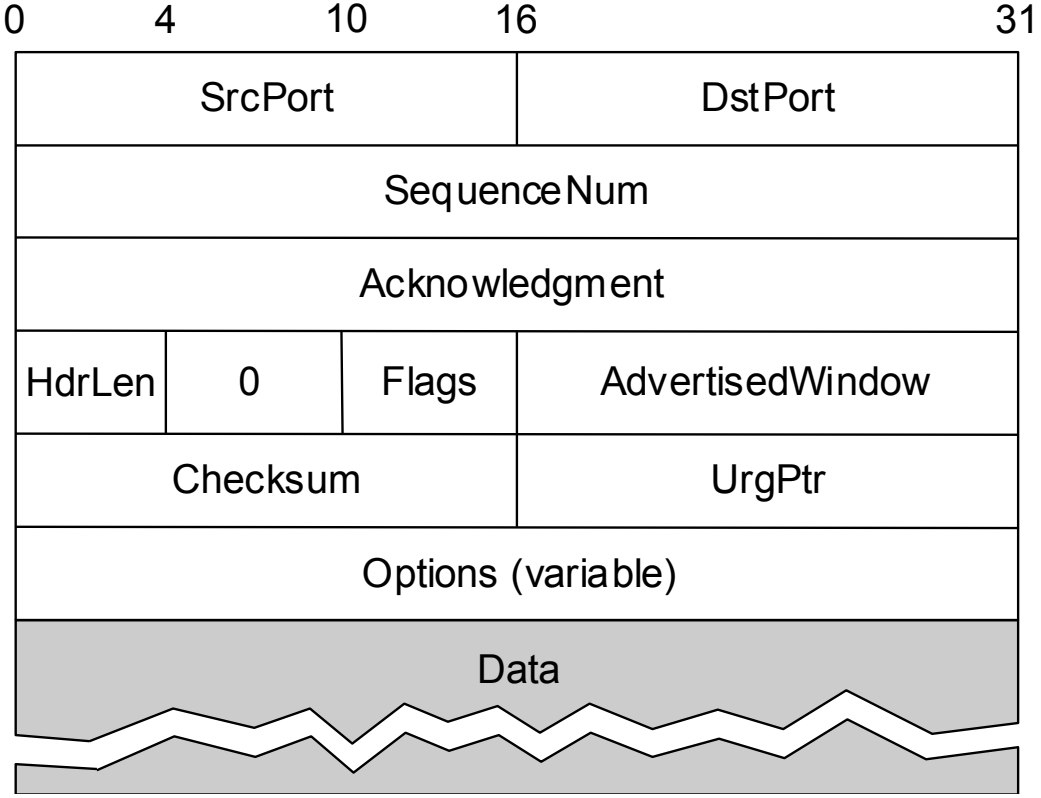
TCP Features

- Connection-oriented
- Byte-stream
  - app writes bytes
  - TCP sends *segments*
  - app reads bytes
- Reliable data transfer
- Full duplex
- Flow control: keep sender from overrunning receiver
- Congestion control: keep sender from overrunning network



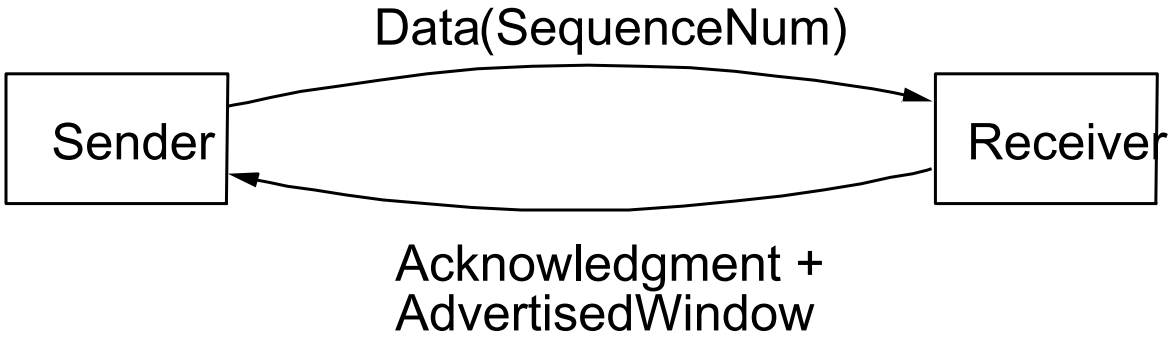
Segment Format

- SequenceNum: First byte in data
- Acknowledgement: Next expected byte in reverse direction
- Flags: SYN, FIN, RESET, PSH, URG, ACK
- HdrLen: Length of header in 32-bit words
- AdvertisedWindow: Amount of room to receive data



Segment Format (cont)

- Each connection identified with 4-tuple:
  - **`(SrcPort, SrcIPAddr, DsrPort, DstIPAddr)`**
  - Used as the demuxing key
  - Port can be reused after connection tear down – "another incarnation"
- Sliding window + flow control
  - **`acknowledgment, SequenceNum, AdvertisedWinow`**



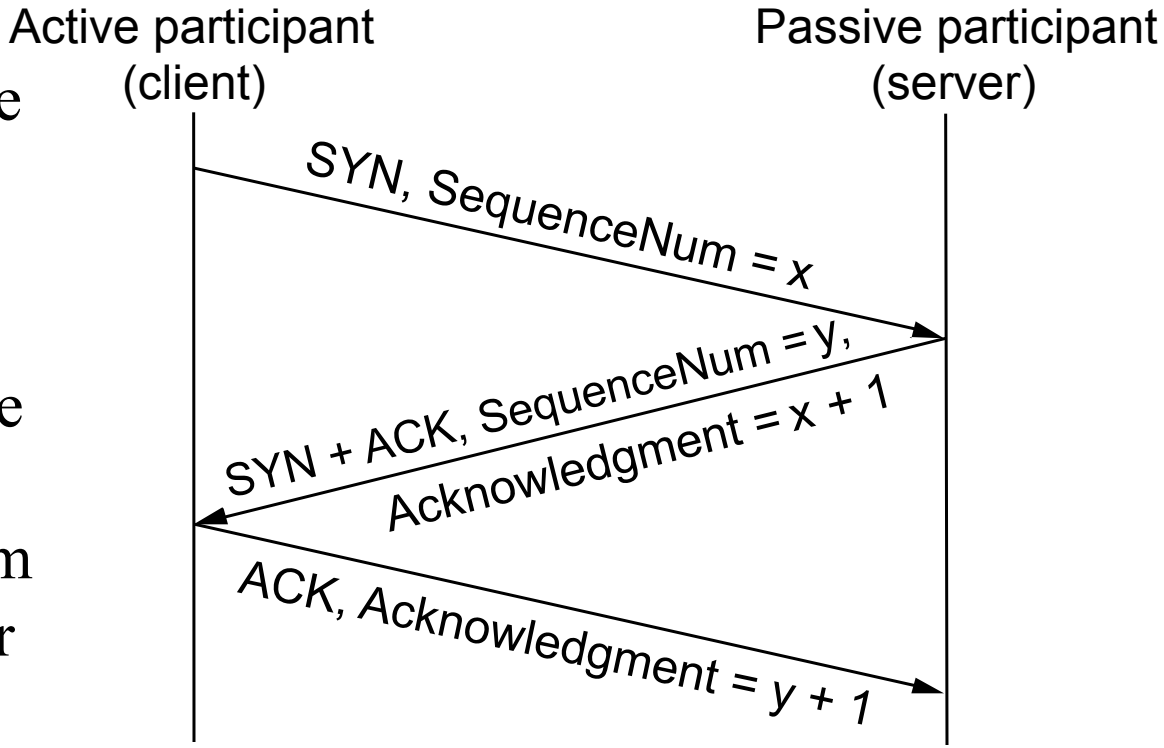
- Flags
  - **`SYN, FIN, RESET, PUSH, URG, ACK`**
- Checksum is the same as UDP
  - pseudo header + TCP header + data

Sequence Numbers

- 32 bit sequence numbers
  - Wrap around supported
- TCP breaks byte stream from application into packets (limited by Max. Segment Size)
- Each byte in the data stream is considered
- Each packet has a sequence number
  - Initial number selected at connection time
  - Subsequent numbers indicate first data byte number in packet
- ACK's indicate ***next byte expected***

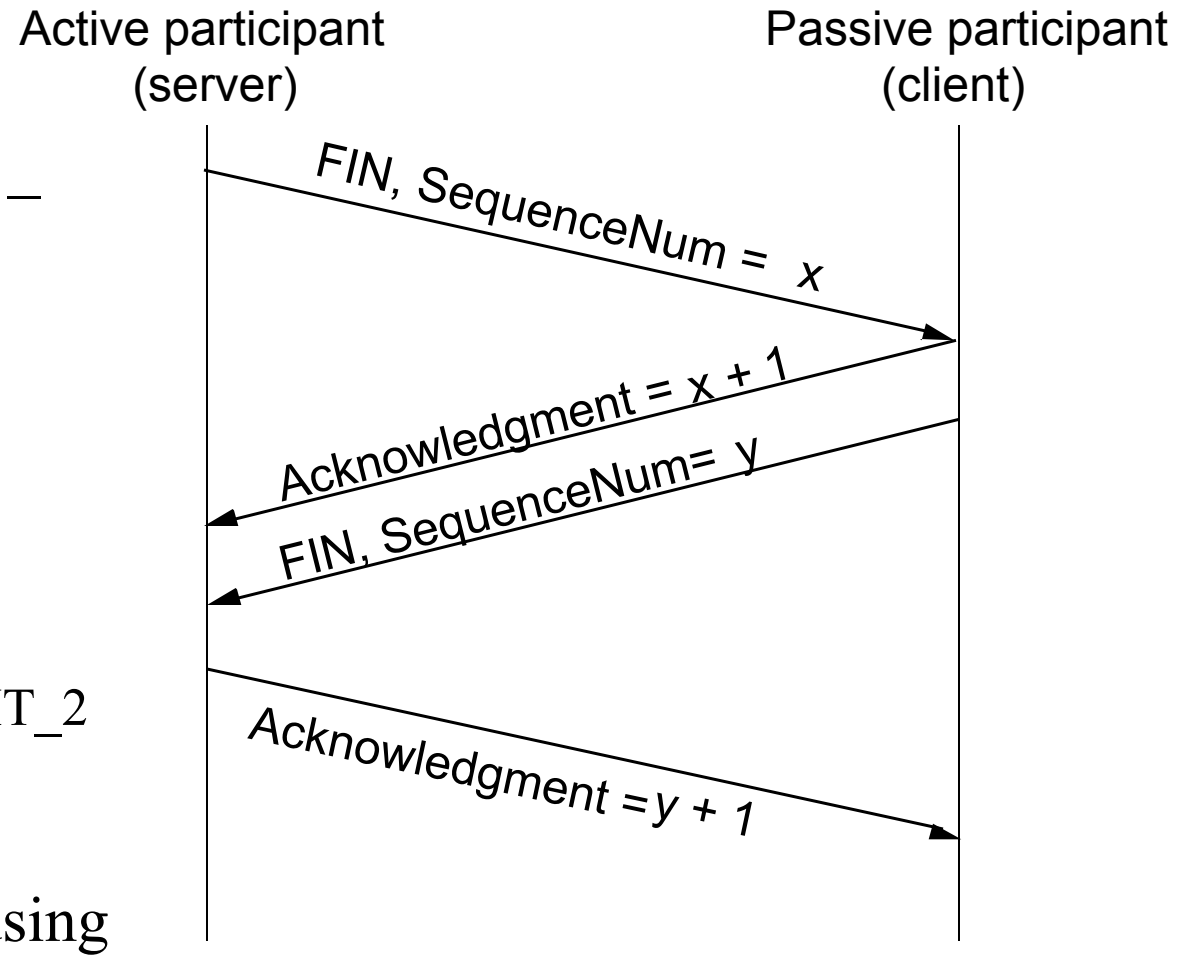
Connection Establishment

- This diagram shows the typical "3-way handshake".
- Each side picks a random initial sequence number.
- This avoids the problem of segment from earlier incarnation interfering with later incarnation.



Connection Termination

- "This side closes first" – i.e. close is invoked by server side of the connection.
- Visualized in the state transition diagram as: ESTABLISHED => FIN_WAIT_1 => FIN_WAIT_2 => TIME_WAIT
- Other possibilities can similarly be observed using the state transition diagram



State Transition Diagram

- Reason for TIME_WAIT.
- Local side's ACK in response to a FIN may not be successfully delivered.
- If the local side had immediately closed and reassigned the port to another incarnation, the older other sides retransmitted FIN segment may wrongly terminate the new connection.