

Lecture 6:

Transport Layer Protocols

CSE 123: Computer Networks

Aaron Schulman

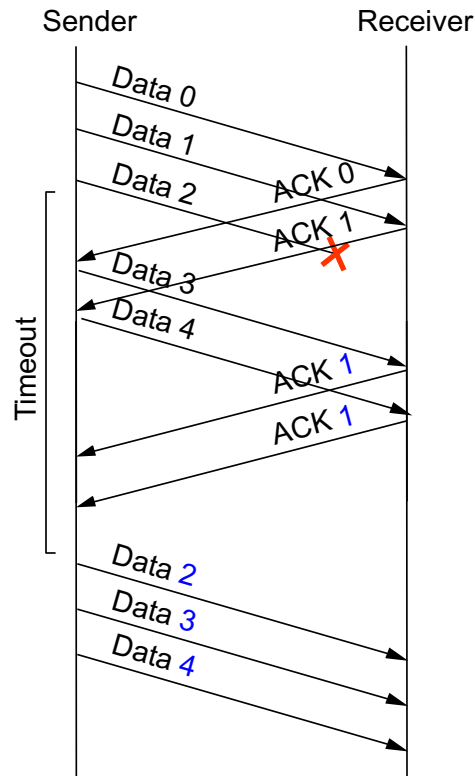


Overview



- Finish Reliable Transmission
 - ◆ Sliding window
- Process naming/demultiplexing
- User Datagram Protocol (UDP)
- Transport Control Protocol (TCP)
 - ◆ Three-way handshake
 - ◆ Flow control

Send Window (sender buffer)



**Go-Back-N Example
with window size 3**

- Bound on number of outstanding packets
 - ◆ Window “opens” upon receipt of *new* ACK
 - ◆ Window resets entirely upon a timeout
- Limits amount of memory waste
 - ◆ Don’t need to remember all packets ever sent for retrans.
 - ◆ Go-Back-N might still lead to sending lots of duplicates
 - » We can do better with **selective retransmission**:
only retransmit missed packets

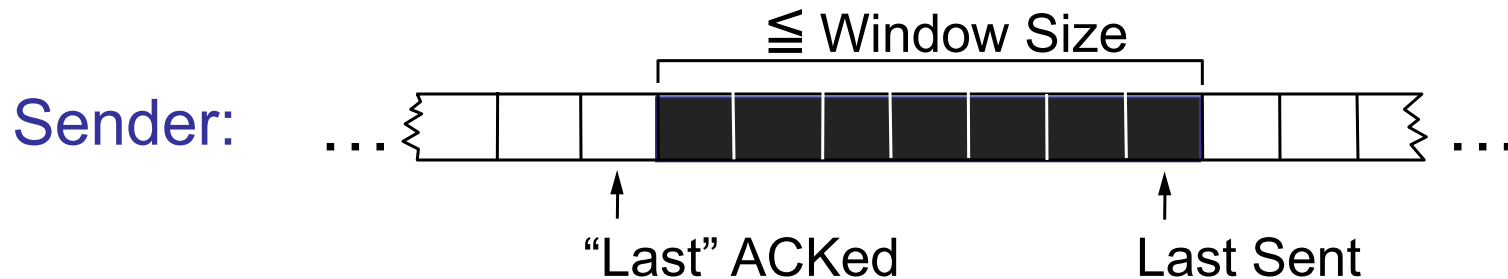


Sliding Window

- Single mechanism that supports:
 - ◆ Multiple outstanding packets
 - ◆ Reliable delivery
 - ◆ In-order delivery
 - ◆ Flow control: Don't send more than receiver can handle
- Sender and receiver each maintain “window” abstractions to track outstanding packets
 - ◆ At the core of all modern ARQ protocols
- Go-Back-N is a special case
 - ◆ Receive window size of one

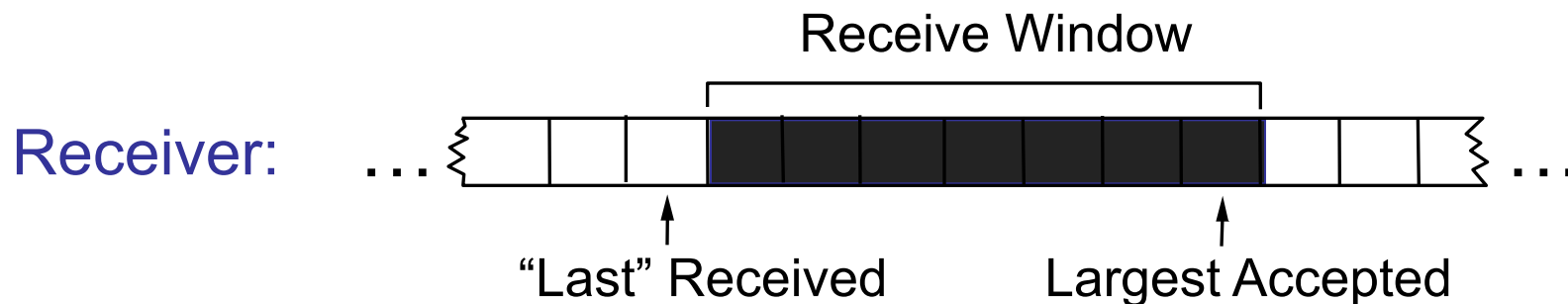


Sliding Window – Sender



- Window bounds outstanding unACKed data
 - ◆ Implies need for buffering at sender
- “Last” ACKed applies to **in-order** data
- What to do on a timeout?
 - ◆ Go-Back-N: resend all unacknowledged data on timeout
 - ◆ Selective Repeat: timer per packet, resend as needed

Sliding Window – Receiver



- Receiver buffers too:
 - ◆ data may arrive out-of-order
 - ◆ or faster than can be consumed by higher layers
 - » **Flow control:** tell sender how much receive window is left
 - Sender will only send enough packets to fill the receive window

Deciding When to Retransmit



- How does sender know when a packet has been lost?
 - ◆ Ultimately sender uses timeouts to decide when to retransmit
- But how long should the timeout be?
 - ◆ Too long: inefficient (large delays, poor use of bandwidth)
 - ◆ Too short: may retransmit unnecessarily (causing extra traffic)
- Right timeout is based on link latency: **round-trip time** (RTT)
 - ◆ Which can vary greatly for reasons well see later

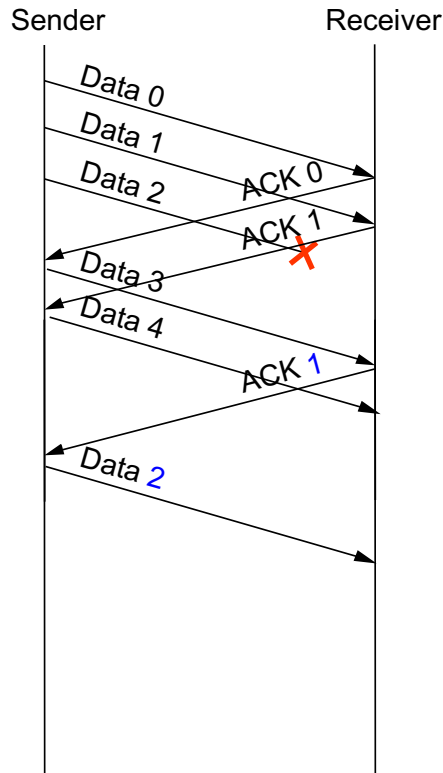
Can we shortcut the timeout?



- Timeout is long in practice
 - ◆ Lots of variation in RTT and timeout must be conservative
- If packets are usually *in order* then *out-of-order* ACKs from receiver imply that a packet was lost
- Knowing something is lost, we can proactively rxmit!
 - ◆ Negative ACK
 - » Receiver requests missing packet
 - ◆ Fast retransmit
 - » When sender receives multiple duplicate acknowledgements resends missing packet

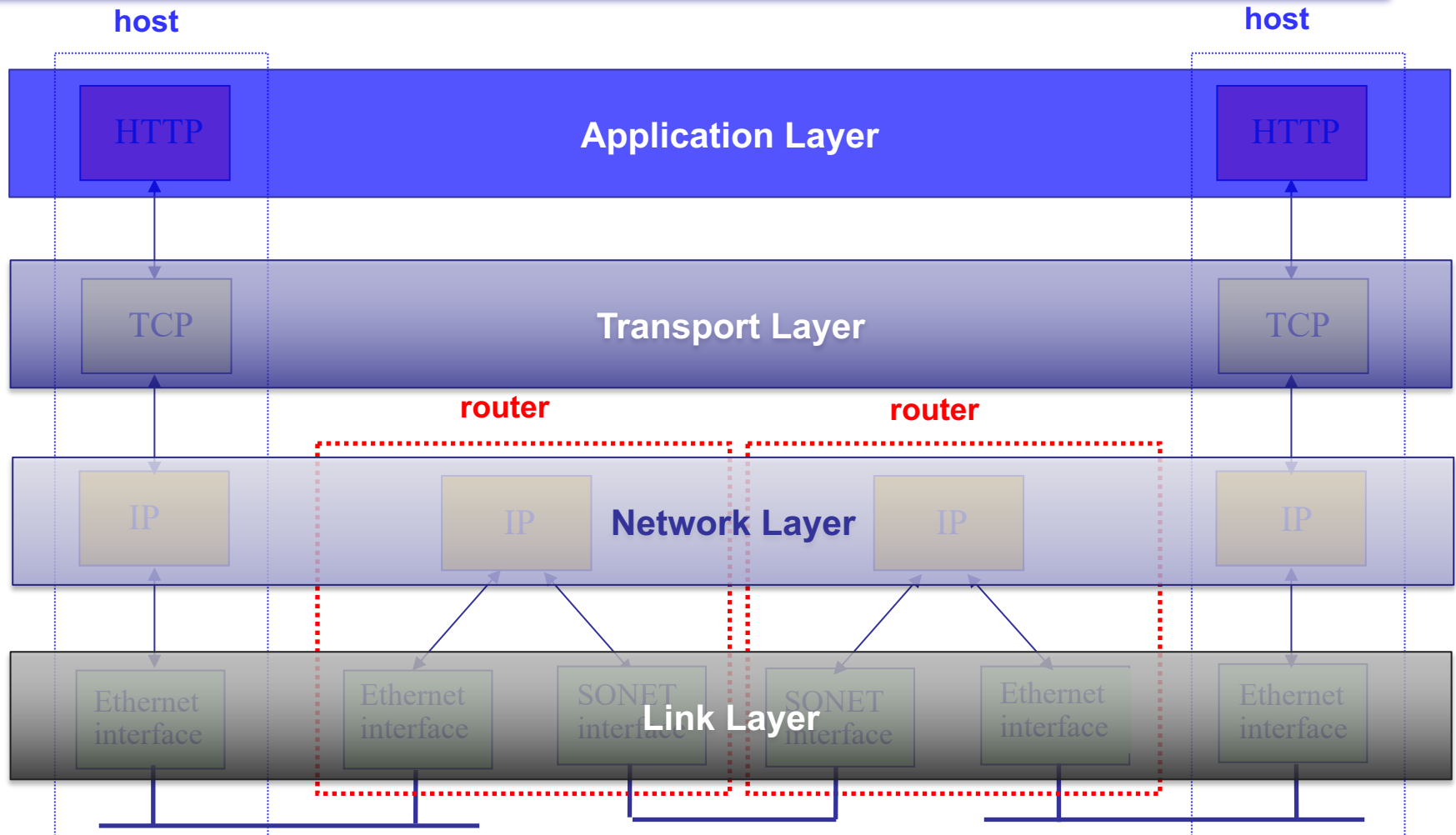


Fast retransmit



- Don't bother waiting for timeout
 - ◆ Receipt of duplicate acknowledgement (**dupACK**) indicates loss
 - ◆ Retransmit immediately
- Used in TCP
 - ◆ Need to be careful if frames can be reordered
 - ◆ Today's TCP identifies a loss if there are **three** duplicate ACKs in a row

The Transport Layer provides more than just Reliable Transmission





Transport Layer Tasks

- Define and provide specific delivery semantics
 - ◆ To which end point (which application/process)?
 - ◆ When?
 - ◆ How?
 - ◆ If?
- Multiplexing different processes on the same links
- ... and Reliability (if the application needs it)

Naming Processes/Services



- Process here is an abstract term for your Web browser (HTTP), Email servers (SMTP), hostname translation (DNS)
- How do we identify for remote communication?
 - ◆ Process id or memory address are OS-specific and transient
- So TCP and UDP use **ports**
 - ◆ 16-bit integers representing mailboxes that processes “rent”
 - ◆ Identify process uniquely as (IP address, protocol, port)



Picking Port Numbers

- We still have the problem of allocating port numbers
 - ◆ What port should a Web server use on host *X*?
 - ◆ To what port should you send to contact that Web server?
- Servers typically bind to **well-known** port numbers
 - ◆ e.g., HTTP 80, SMTP 25, DNS 53, ... look in /etc/services
 - ◆ Ports below 1024 traditionally reserved for well-known services
- Clients use OS-assigned temporary (**ephemeral**) ports
 - ◆ Above 1024, recycled by OS when client finished

Establishing well-known ports



Network Working Group
Request for Comments #322
NIC #9609

V. Cerf
J. Postel
UCLA-NMC
26 March 72

Well Known Socket Numbers

The Network Measurement Group would like to establish a network standard socket number for a Process Discard service (not all HOSTs need cooperate.) To do this, it is necessary to know which sockets at each HOST have already been allocated to standard network functions.

At all HOSTs which permit login for services, socket 1 is the Network Logger on which ICP may be performed.

We would like to catalog other sockets which are supposed to be well-known, so we would appreciate having a note or phone call from each HOST Technical Liaison describing the function and socket numbers of network service programs at each HOST. The catalog will be published and we would recommend that it be maintained at NIC.

Dr. Vint Cerf or Jon Postel
3804 Boelter Hall
UCLA Computer Science Department
Los Angeles, California 90024

(213) 825-4864 Vint

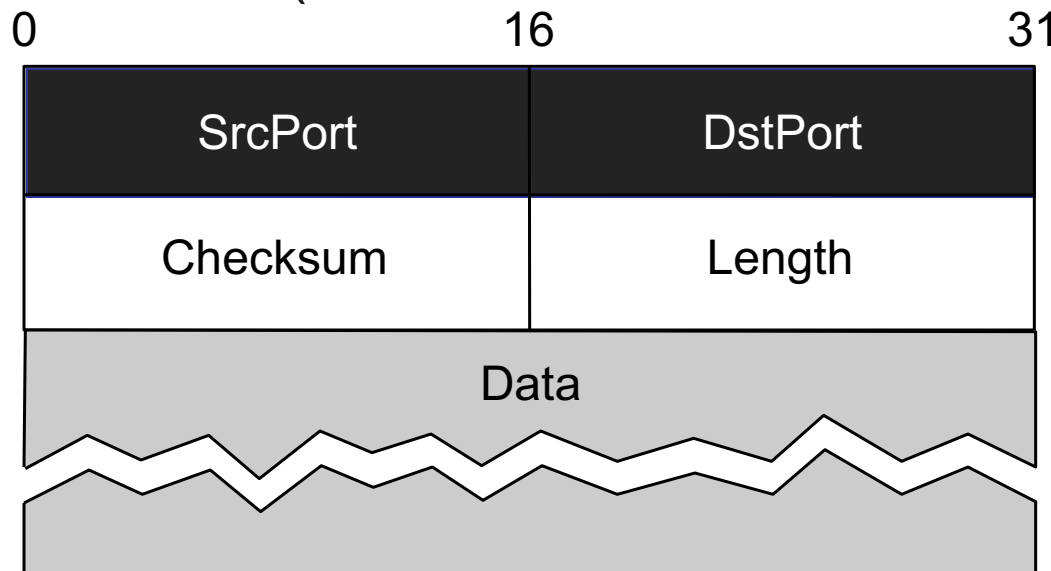
(213) 825-4733 Jon

(213) 825-2368 Secretary

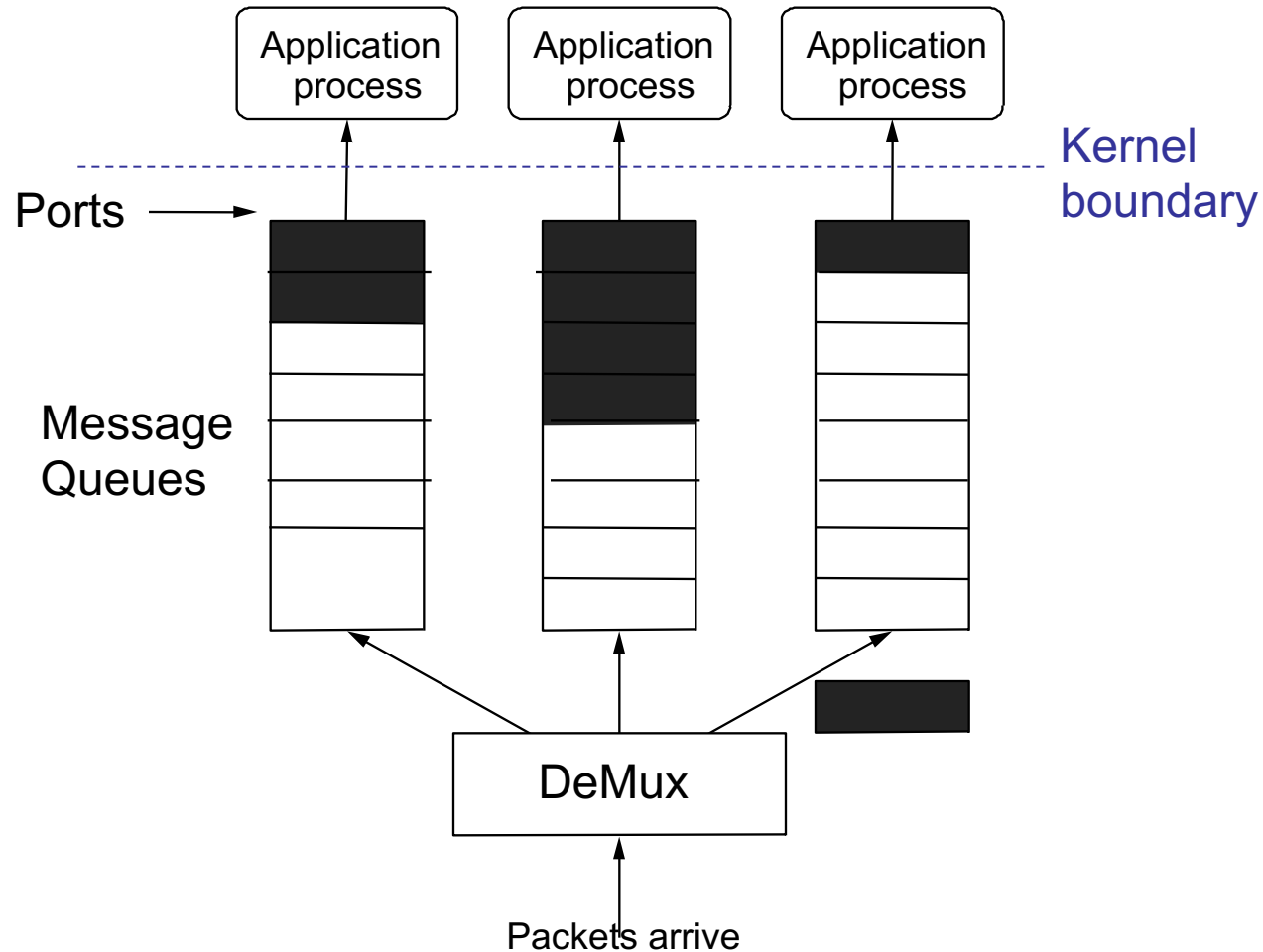
User Datagram Protocol (UDP)



- Provides *unreliable message delivery* between processes
 - ◆ Source port filled in by OS as message is sent
 - ◆ Destination port identifies UDP delivery queue at endpoint
- Connectionless (no state about who talks to whom)



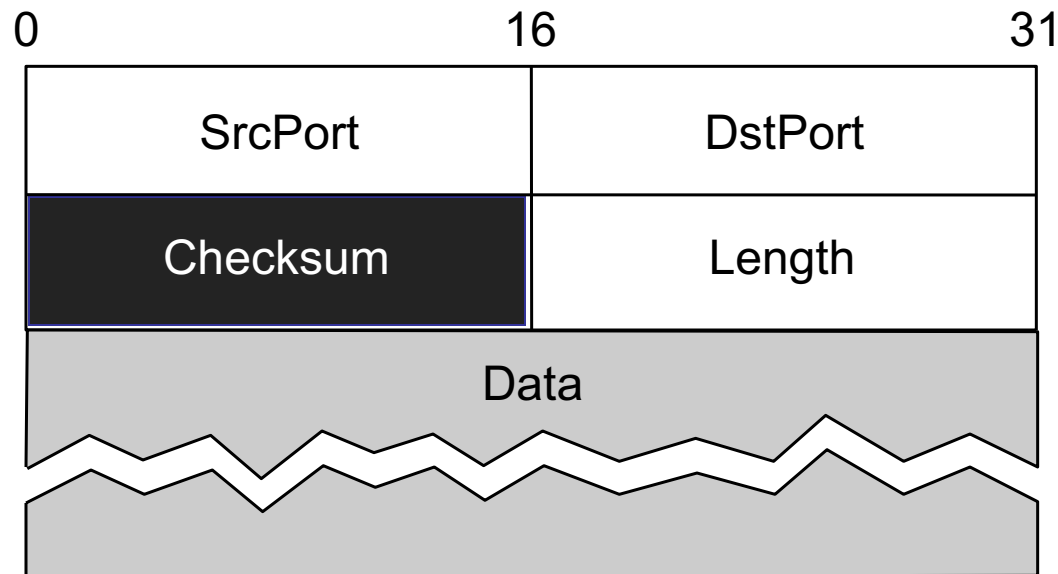
UDP Delivery





UDP Checksum

- UDP includes optional protection against errors
 - ◆ Checksum intended as an end-to-end check on delivery
 - ◆ So it covers data, UDP header, and **IP pseudoheader (history)**





Applications for UDP

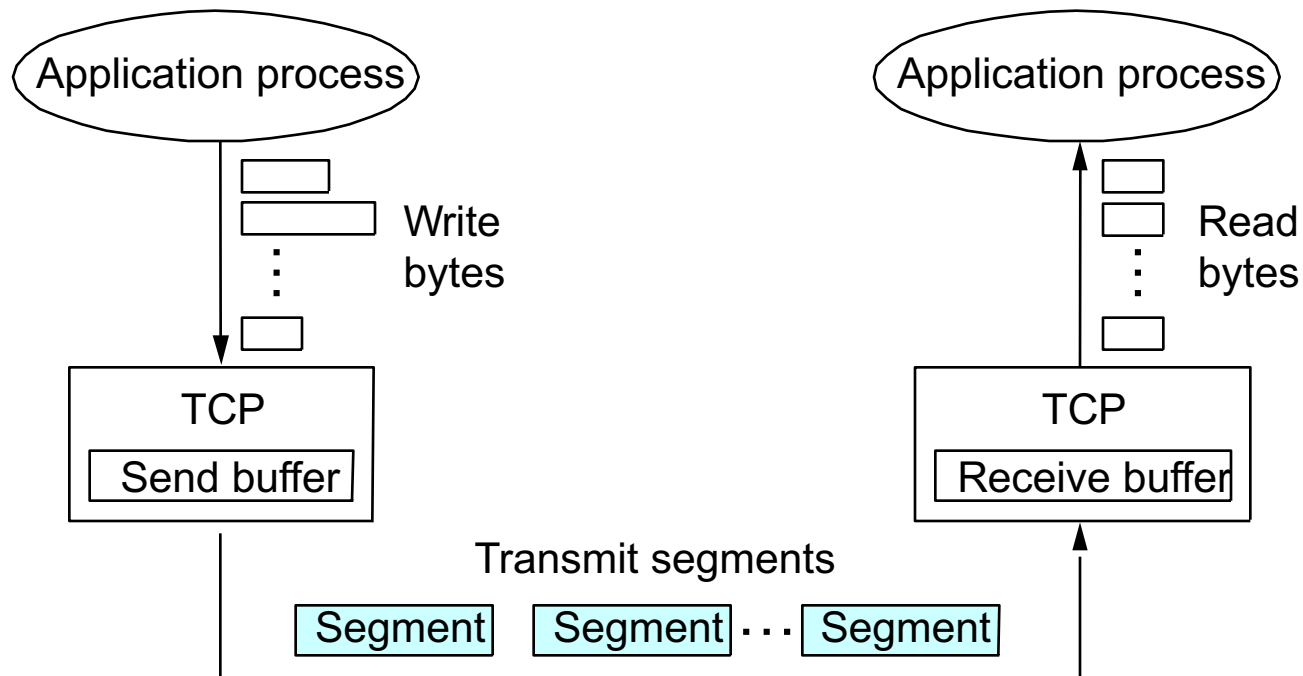
- Streaming media (e.g., live video)
- DNS (Domain Name Service)
- NTP (Network Time Protocol) (synchronizing clocks)
- FPS multi-player video games (e.g., Call of Duty)
- Why might UDP be appropriate for these?

Transmission Control Protocol



- Reliable bi-directional **bytestream** between processes
 - ◆ Uses a sliding window protocol for efficient transfer
- Connection-oriented
 - ◆ Conversation between two endpoints with beginning and end
- Flow control (last lecture)
 - ◆ Prevents sender from over-running receiver buffers
 - ◆ (tell sender how much buffer is left at receiver)
- Congestion control (later in term)
 - ◆ Prevents sender from over-running network capacity

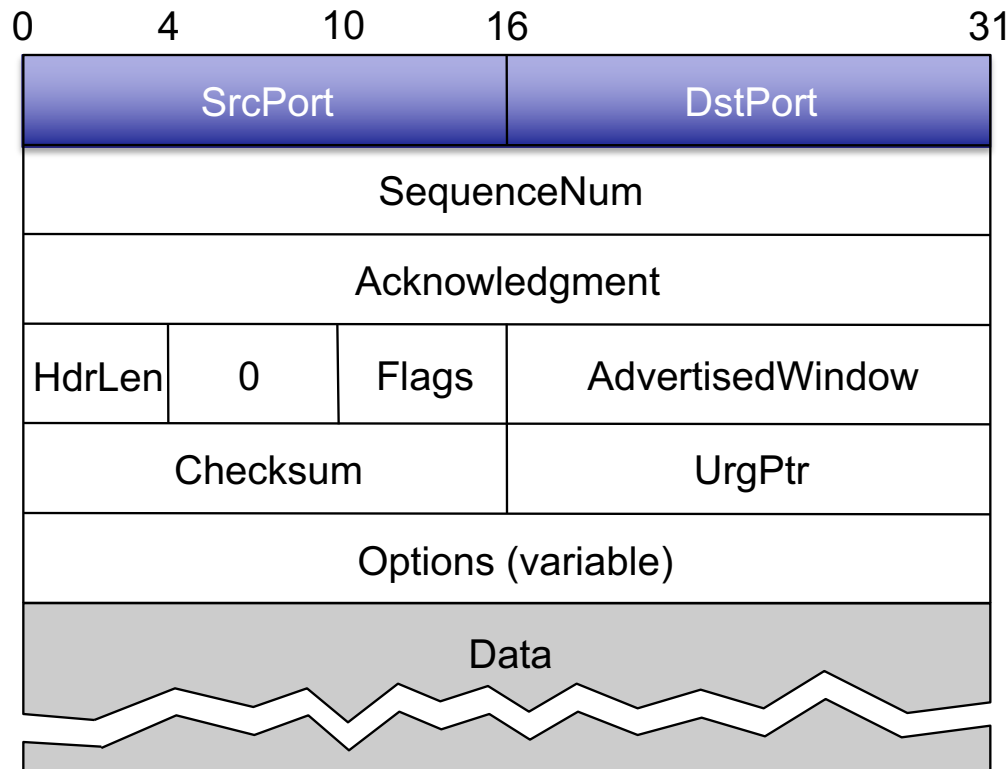
TCP Delivery





TCP Header Format

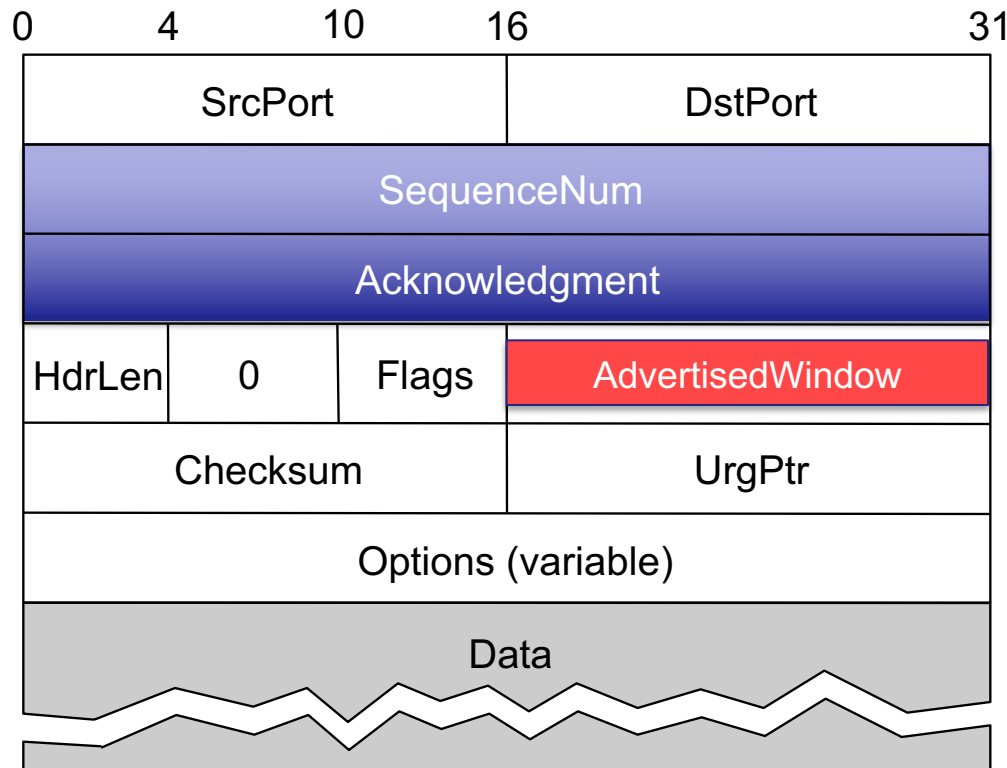
- Ports plus IP addresses identify a connection (4-tuple)





TCP Header Format

- Sequence, Ack numbers used for the sliding window
 - ◆ How big a window? Flow control/congestion control determine

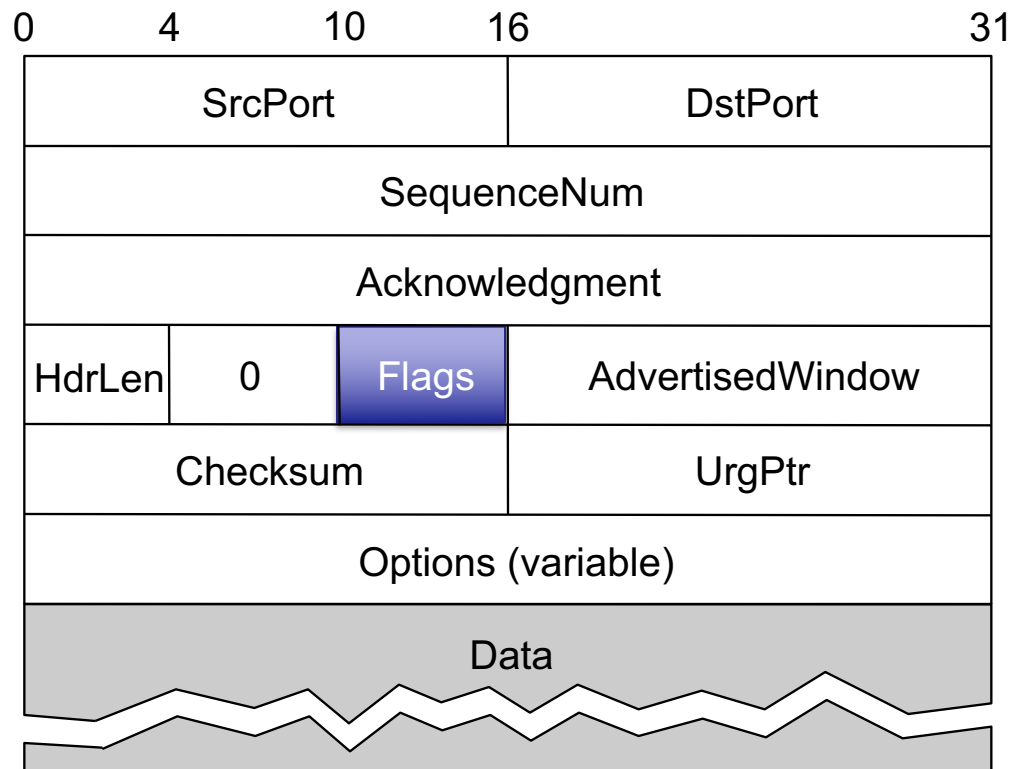


Flow control –
Receiver tells the
transmitter how big its
remaining window is



TCP Header Format

- Flags may be ACK, SYN, FIN, URG, PSH, RST





For next time...

- More TCP details
- Read Ch 5.2 in P&D