

Transport Layer

CSC 790

WAKE FOREST
UNIVERSITY

Department of Computer Science

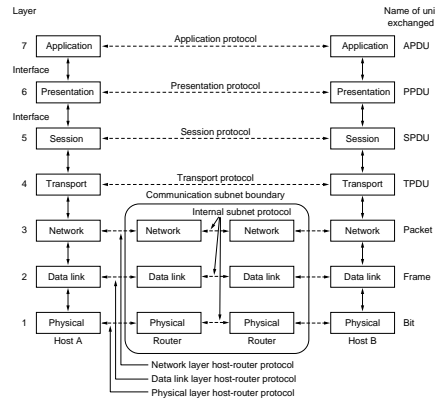
Fall 2009

Transport Layer

- Reliable, cost-effective data transport from source to destination
 - End-to-end protocol only implemented at **hosts**
- *Transport entity* provides services to upper layers
 - Type of service, QoS, data transfer, connection management, and flow control
- Two types of service available
 - Connection-oriented - Establishment, maintenance, and termination of connection
 - Connectionless - Unreliable service (delivery not guaranteed), reduces the overhead associated with transport layer

Is this redundant with layer 2 or 3?

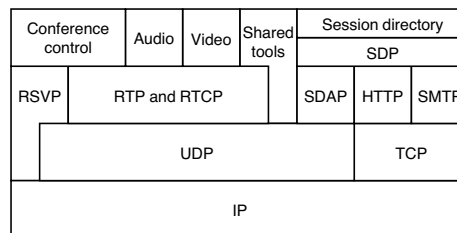
Network and Transport Layers



- The network layer is part of the communication subnet
 - What if the network-layer is connection-oriented but unreliable?
 - What happens if a router crashes?
 - Transport layer *provides reliable service over unreliable network*

Internet Transport Layer

- There are two distinct Internet transport-layer protocols
 - User Datagram Protocol (UDP)
 - Transmission Control Protocol (TCP)



- UDP provides unreliable connectionless service
- TCP provides reliable connection-oriented service

Which service is better for multimedia traffic?

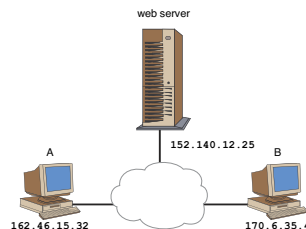
Port Numbers

- Transport addresses are used to identify processes
 - For Internet transport layers, this called a **port number**
 - **Both** source and destination processes have a port number
- Source and destination ports *together* uniquely identify a process

Do we really need source and destination port numbers to identify a process (single process at one end)?
- Port numbers are 16 bits, ranging from 0 to 65535
 - Numbers from 0 to 1023 are **well-known port numbers**
 - For example `http` is 80, `ftp` is 21, complete list at RFC 1700

Web Server Example

- Consider a web server and two stations connected via the Internet



- Web server runs `http` over port 80
 - Any station wishing to connect will use the IP address 152.140.12.25 and port 80
- Let station A connect to the web server
 - A free port is used for source (> 1023), for example 1123
 - Destination port is 80

- Web server creates a new process for each request (Unix fork)
 - Allows server to connect to multiple users simultaneously

What happens if station B connects to the web server? Station B will use the same destination port 80, how does the web server differentiate between station A and station B requests?

What happens if station B connects to the web server and happens to select the same source port number as A? How does the web server differentiate between station A and station B requests?

UDP

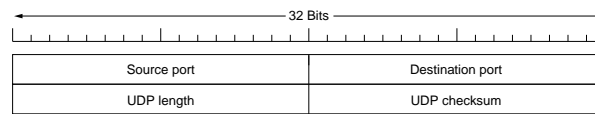
- User Datagram Protocol (UDP) is a lightweight protocol with minimalist service model [RFC 768]
 - Connectionless, so **no** handshaking before sending
 - Unreliable, there is no guarantee of delivery
 - Datagrams may arrive out-of-order
- In addition, there is **no** flow or congestion control
 - Send as much and as quickly as desired

What is flow and congestion control?

- In summary, an UDP application almost talks directly to IP

UDP Datagram Structure

- UDP datagram header has the following structure



- Source and destination port numbers 16 bits each
- UDP length field (16 bits) length of header and data in bytes
- Checksum field (16 bits) error checking for the UDP datagram
 - One's complement sum of all the 16 bit words in the segment

If UDP is unreliable and does not provide acknowledgements, why error check?

When is UDP Preferred

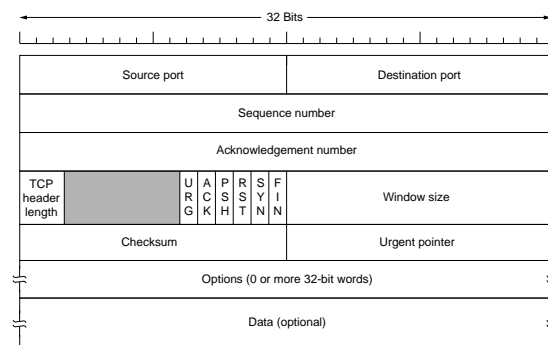
UDP has the following benefits, making it better for some applications

- No connection establishment
 - TCP requires a three-way handshake before sending data
 - UDP sends data immediately, no initial connection delay
- No connection state
 - TCP requires state information about send/receive buffers, congestion-control, sequence numbers... *per connection*
- Small packet header overhead
- **Unregulated send rate**
 - Send as quickly as desired
 - Has introduced the idea of *TCP friendly* applications

TCP

- Transport Control Protocol (TCP) provides connection-oriented, reliable service [RFC 793, 1122, 1323, 2018, 2581]
- All connections TCP connections are full-duplex, point-to-point
 - Requires three-way handshake to establish connection
- Reliable transport service
 - Deliver all data without error and proper order
- Congestion control mechanism
 - Attempts to limit each connection to *fair share* of bandwidth
 - **No** minimum bandwidth guaranteed
- TCP connection is a byte stream, not a message stream
 - Data is buffered before sent (additional delay)

TCP Header

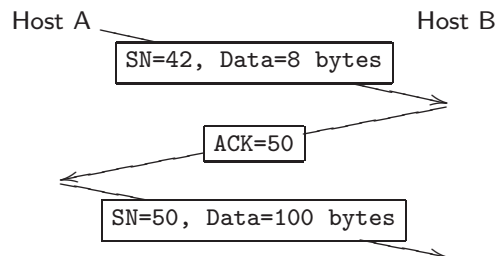


- Source and destination port fields (16 bits each)
- Sequence and acknowledgement number fields (32 bits each)
- Window field - Number of bytes receiver can accept from sender

Is this congestion control?

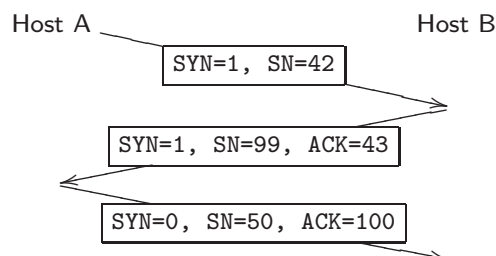
Sequence and Acknowledgement Numbers

- TCP views data as a *unstructured* ordered stream of bytes
- Sequence/acknowledgements numbers are for bytes **not** segments
- If a host wants to send stream of data
 - TCP will number each byte of information
 - SN will represent the first byte number
- Acknowledgement numbers are for the **next** byte expected
 - In addition acknowledgements can be cumulative



TCP Connection Management

- Events to establish a connection (*3-way handshake*)
 1. Client side sends SYN segment to server
 2. Server, check if process *listening* on port
 3. Client side receives SYNACK



- For termination, consider connection as two simplex connections
Any problems with TCP handshake with respect to multimedia?

TCP Reliable Data Transfer

- TCP ensures that data placed in the receive buffer is
 - Not corrupted, has no gaps or duplicates, and is in sequence
 - *So what happens if segments are lost or out-of-order?*
Why would this occur?
- Two options are possible when segments are out-of-order
 1. Receiver discards any out-of-order segments, *easy to implement but not very efficient*
 2. Receiver keeps out-of-order and attempts to reorder, *more efficient but difficult to implement*

Which option is specified by the RFC?

TCP Flow Control

- TCP creates a receive buffer for data received
 - Rate at which data removed, depends on application
 - Therefore, receive buffer can become full
 - TCP provides *flow control* to prevent the sender from overflowing the receiver's buffer
- TCP requiring sender to maintain a **received window**
 - Window indicates the amount of free space at the receiver
 - *Can change size over time*
 - Sending segments reduces the window, while receiver acknowledgements increase the window

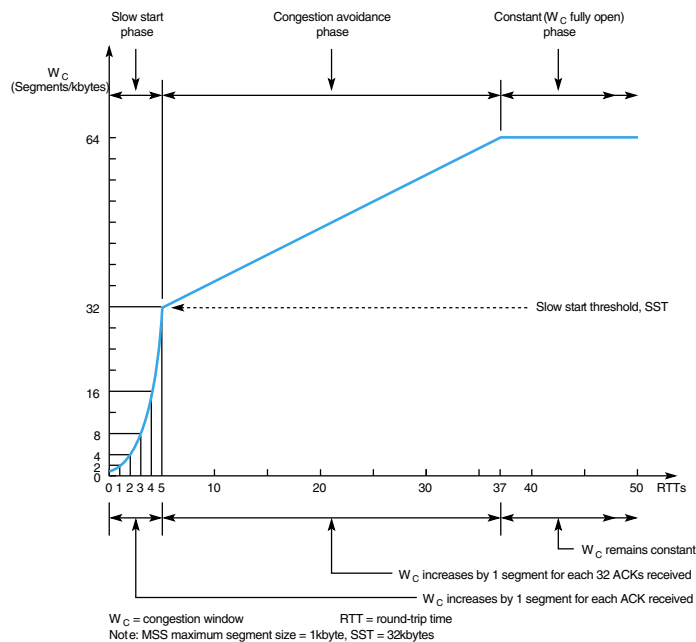
TCP Congestion Control

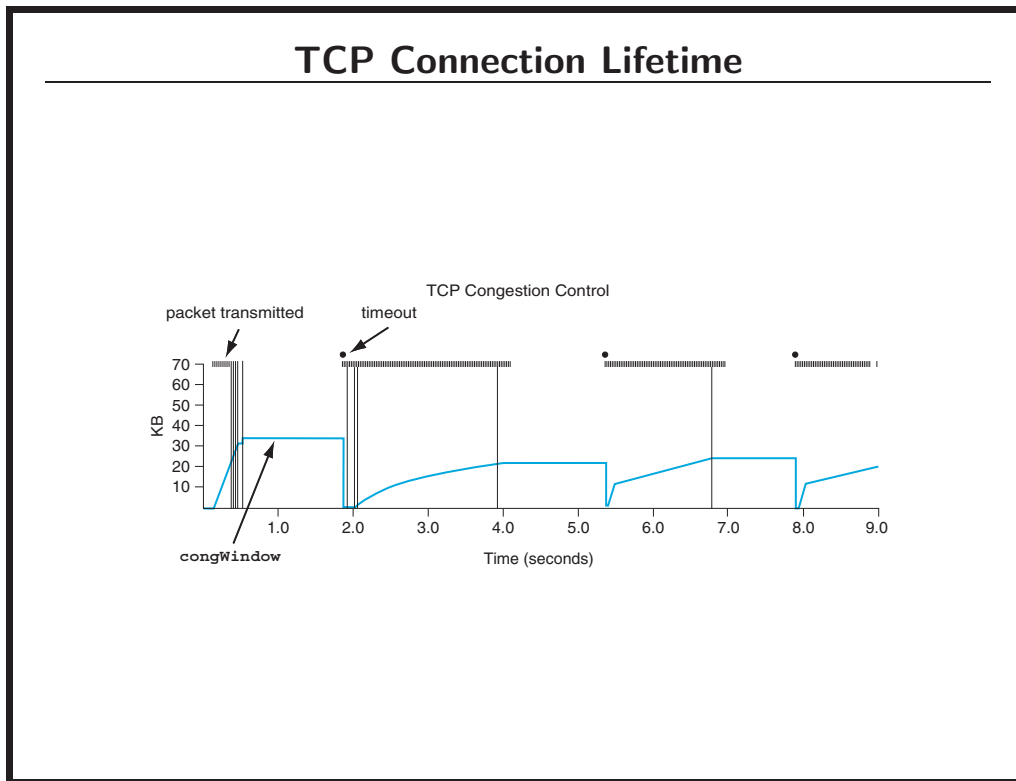
Control TCP connections sharing the bandwidth of a congested link

- TCP congestion control is closed-loop and uses implicit feedback
- Transmission rate limited by the congestion window size, w
 - Number of bytes that can be sent without ACKs
- General operation
 - Transmit as fast as possible as long as no segments are lost
 - TCP starts with a small w , and increases slowly over
 - Once a segment is lost, w is reduced quickly
 - Process is repeated during the connection lifetime

Why is this implicit feedback, not explicit feedback?

TCP Congestion Control





TCP Average Throughput

- An important measure of TCP performance is throughput
 - Rate at which data transmitted from sender to receiver
- Throughput will depend on w *Which window? flow or congestion?*
 - If w segments sent back-to-back, must wait RTT to send again
 - If w bytes every RTT seconds, the throughput (bytes/sec) is

$$\frac{w}{\text{RTT}}$$

So the throughput is constant over time?

UDP or TCP?

Which transport layer is more appropriate for multimedia traffic?

Is reliable, in-order delivery not important for multimedia?