Transport and Applications Layers

Introduction

1. Focus on the logical end-to-end connection between processes that are running on these two hosts
   - Happens through the transport layer
   - Focus on TCP: Reliability, flow control, and congestion control
     - Also discuss recent developments in TCP protocol

Introduction to Transport Layer and Relationship between Transport and Network Layer

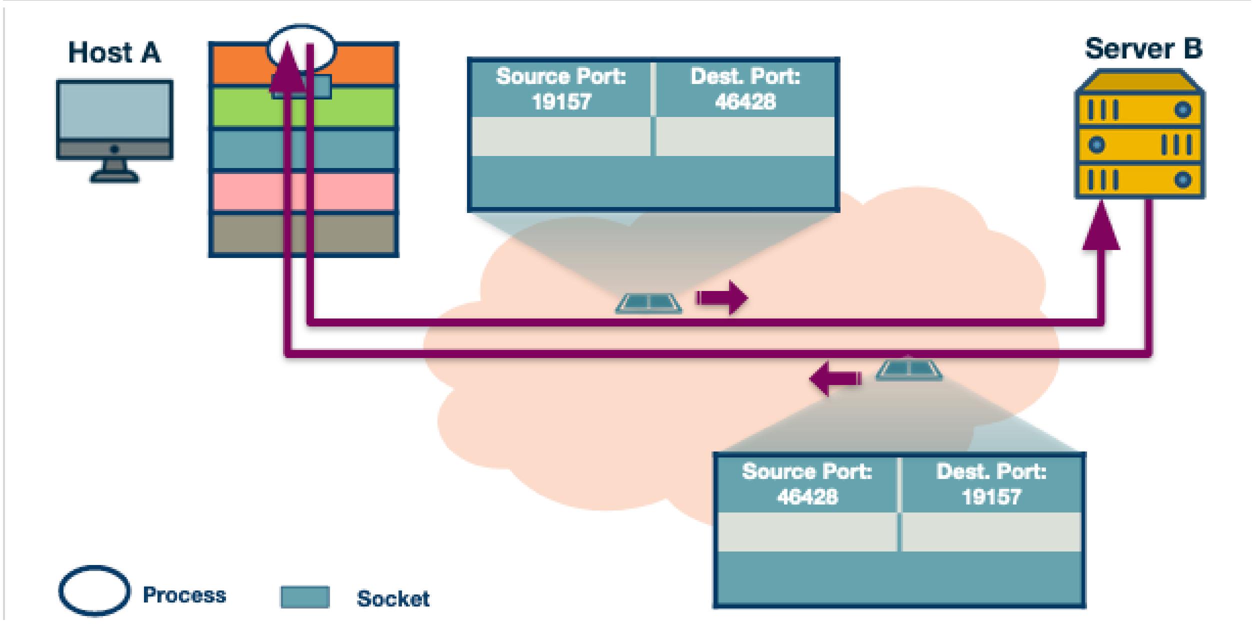
1. Transport Layer
   - Provides an end-to-end connection between two applications running on different hosts
   - How it works:
     - Transport layer on sending hosts receives a message from the application layer and appends its own header (segment)
     - Segment is sent to the network layer which will append (encapsulate) this segment with its own header information
     - Send to the receiving host through routers, bridges, switches, etc.
   - Why do we need a layer between application and network?
     - Network only provides best-effort delivery service model
     - Transport layer can offer guaranteed delivery of packets and integrity
2. Most common protocols:
   - TCP: Transmission Control Protocol
     - Provides strong primitives to make end-to-end communication more reliable and cost-effective
   - UDP: User Datagram Protocol
     - Proivdes basic functionality and relies on the application layer to implement the remaining

Multiplexing: Why Do We Need It?

1. Multiplexing: Allows host to run multiple applications that use the network simultaneously
   - Transport layer uses ports to determine which application a packet is destined for
     - Each application binds itself to a unique port number by opening sockets and listening for any data from a remote application
   - Two ways to use multiplexing
     - Connectionless
     - Connection-oriented

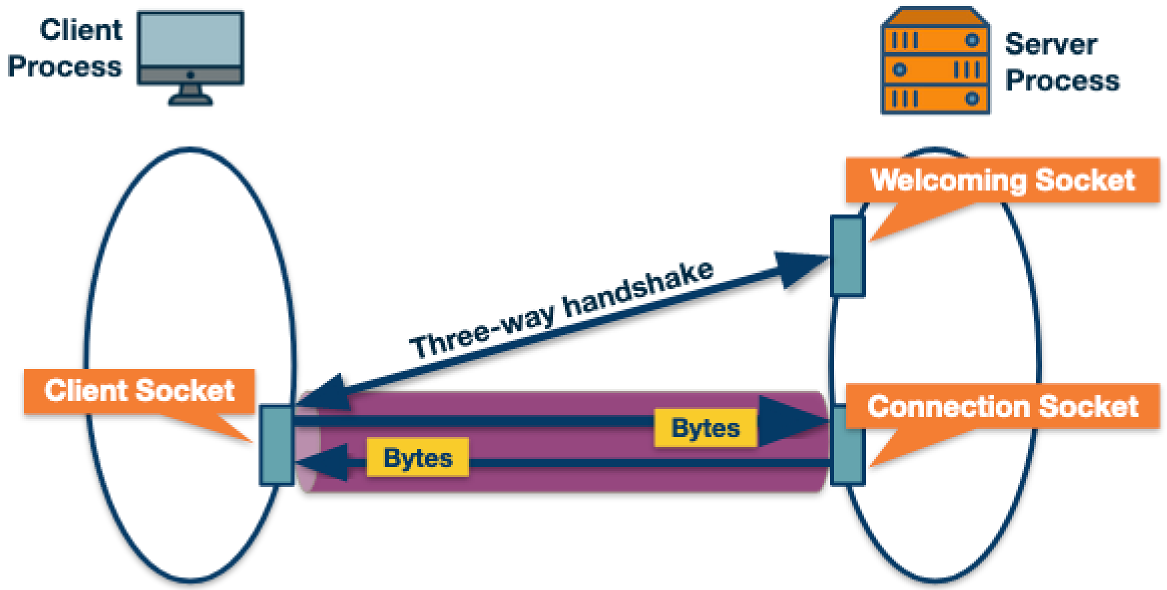
Connection Oriented and Connectionless Multiplexing and Demultiplexing

1. Connectionless Multiplexing/Demultiplexing



Connectionless Multiplexing/Demultiplexing

2. Connection-oriented Multiplexing/Demultiplexing



Connection-oriented Multiplexing/Demultiplexing

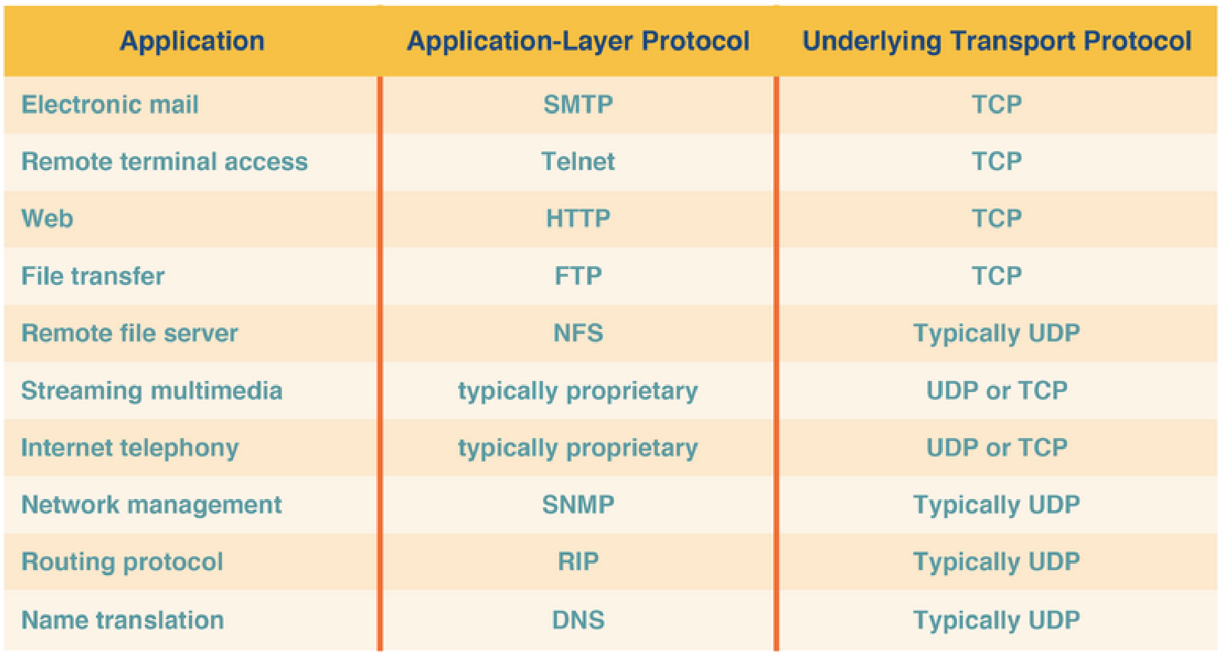
Quiz 1

1. As we have seen, UDP and TCP use port numbers to identify the sending application and destination application. Why don't UDP and TCP just use process IDs rather than define port numbers?
   - Process IDs are specific to operating systems and therefore using process IDs rather than a specially defined port would make the protocol operating system dependent. Also, a single process can set

up multiple channels of communications and so using the process ID as the destination identifier wouldn't be able to properly demultiplex, Finally, having processes listen on well-known ports (like 80 for http) is an important convention.

A Word About the UDP Protocol

1. UDP is an unreliable protocol that lacks the mechanisms that TCP has
    - Connectionless protocol that does not require the three-way handshake before sending packets
    - Offers fewer delays and better control over sending data because:
        - No congestion control or similar mechanisms
        - No connection management overhead
    - With real-time applications that are sensitive to delays, UDP is a better option, despite possibly higher packet loss



Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Remote file server	NFS	Typically UDP
Streaming multimedia	typically proprietary	UDP or TCP
Internet telephony	typically proprietary	UDP or TCP
Network management	SNMP	Typically UDP
Routing protocol	RIP	Typically UDP
Name translation	DNS	Typically UDP

TCP vs UDP

2. UDP packet structure
    - Source port number
    - Destination port number
    - Length of the UDP segment (header and data)
    - Checksum (an error checking mechanism)
        - Provides basic error checking since there is no guarantee for link-by-link reliability
        - Ones complement on the sum of the source port, destination port, and packet length

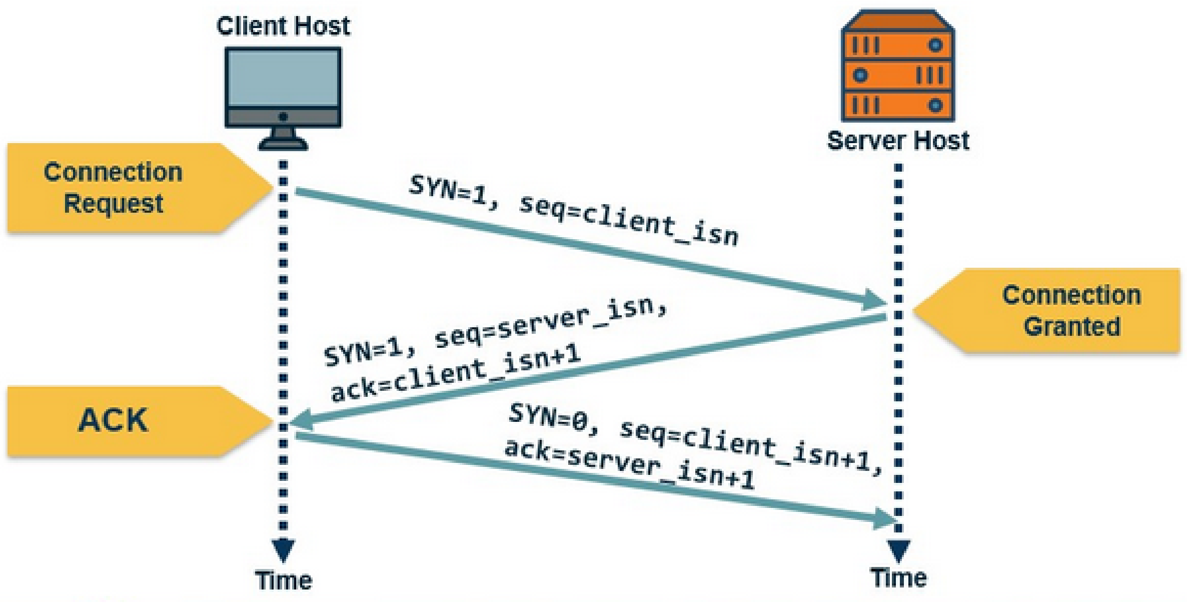
Quiz 2

1. UDP and TCP use 1's complement for their checksums. But why is it that UDP takes the 1's complement of the sum – why not just use the sum? Exploring this further, using 1's complement, how does the receiver compute and detect errors? Using 1's complement, is it possible that a 1-bit error will go undetected? What about a 2-bit error?
    - To detect errors, the receiver adds the four words (the three original words and the checksum). If the sum contains a zero, the receiver knows there has been an error. While all one-bit errors

will be detected, but two-bit errors can be undetected (e.g., if the last digit of the first word is converted to a 0 and the last digit of the second word is converted to a 1).

The TCP Three-Way Handshake

1. Three-way Handshake
    - TCP client sends a special segment (containing no data) with the SYN bit set to 1. The client also generates an initial sequence number (client isn) and includes it in this special TCP SYN segment
    - The server, upon receiving this packet, allocates the required resources for the connection and sends back the special "connection-granted" segment which we call SYNACK segment. This packet has the SYN bit set to 1, the acknowledgement field of the TCP segment header set to (client_isn+1), and a randomly chosen initial sequence number (server_isn) for the server
    - When the client receives the SYNACK segment it also allocates buffer and resources for the connection and sends an acknowledgement with SYN bit set to 0
2. Connection teardown
    - When the client wants to end the connection, it sends a segment with FIN bit set to 1 to the server
    - The server acknowledges that it has received the connection closing request and is now working on closing the connection
    - The server then sends a segment with FIN bit set to 1, indicating that connection is closed
    - The client sends an ACK for it to the server. It also waits for sometime to resend this acknowledgement in case the first ACK segment is lost



TCP Handshake

Reliable Transmission

1. Many applications want the guarantee that all packets have arrived in order
    - Use Automatic Repeat Request (ARQ) to achieve this
        - Send an ACK when a segment is received; if the sender doesn't receive the ACK in some duration, it can simply resend
    - Stop and Wait ARQ: Sender waits for acknowledgement from the receiver
        - How to set timeout value? Based on esimated RTT
        - Bad performance

- Send at most N unacknowledged packets at once
  - N is the window size

Transmission Control

1. Why control the transmission rate?
   - Sender doesn't know link capacity, other traffic on the same link
2. Where should transmission control function reside in the network stack?
   - Could leave it up to application developers (UDP), but transmission control is a fundamental function for most applications, so implementing it in the transport layer is easier

Flow Control

1. Flow control: Controlling the transmission rate to protect the receiver buffer
   - Don't want to overflow the receiver's packet buffer
   - Match sender's rate against receiver's rate of reading the data
     - Receive window (rwnd) provides the sender an idea of how much data the receiver can handle at the moment
     - Sends rwnd in every segment/ACK it sends back to the sender
   - Receiver makes sure the maximum number of unacknowledged bytes is no more than rwnd
     - Sender sends segments of size 1 even after rwnd == 0 so it can learn when there is more space available

Congestion Control Introduction

1. Don't want combined transmission rate to be higher than the link's capacity
   - Can cause long queues, packet drops, etc.
   - Known as congestion control
   - Networks are quite dynamic: Users join and leave, initiate data transmission, and terminating existing flows
     - Congestion control needs to be dynamic enough to adapt to these changes

What are the goals of congestion control?

1. Desirable properties of congestion control
   - Efficiency: High throughput
   - Fairness: Each user should have their fair share of bandwidth
   - Low delay: Real-time applications require responsiveness
   - Fast convergence: Flow should converge quickly
     - Network typically has few short flows and many long flows
     - If convergence is slow, it's unfair to short flows

Congestion control flavors: E2E vs Network-assisted

1. Network-assisted
   - Rely on network layer to provide explicit feedback to the sender about congestion in the network
     - Packets could be lost under severe congestion
2. End-to-end
   - Hosts infer congestion from the network behavior and adapt transmission rate
   - TCP ended up using the end-to-end approach
   - Congestion control is a primitive of the transport layer, routers operate in the network layer
     - Modern newtorks can provide feedback using ECN or QCN

How a host infers congestion? Signs of congestion

1. Packet delay
   - Queues in the router buffers back up, leading to increased packet delays
   - Increase in round trip time, which can be estimated based on ACKs, can indicate congestion in the network
2. Packet loss
   - Routers start dropping packets as the network gets congested
   - Can also occur due to routing errors, hardware failure, time-to-live (TTL) expiry, error in the links, or flow control problems

How Does a TCP Sender Limit the Sending Rate?

1. TCP congestion control was designed to:
   - Determine the network's available capacity
   - Choose how many packets to send without adding to the network's congestion level
   - TCP uses a congestion window similar to the receive window used for flow control (cwnd)
     - Maximum number of unacknowledged data that a sending hosts can have in transit
   - TCP uses a probe-and-adapt approach in adapting the congestion window
     - When congestion is detected, the window is decreased
   - TCP sender cannot send faster than the slowest component, which is either the network or receiving host

Congestion Control at TCP - AIMD

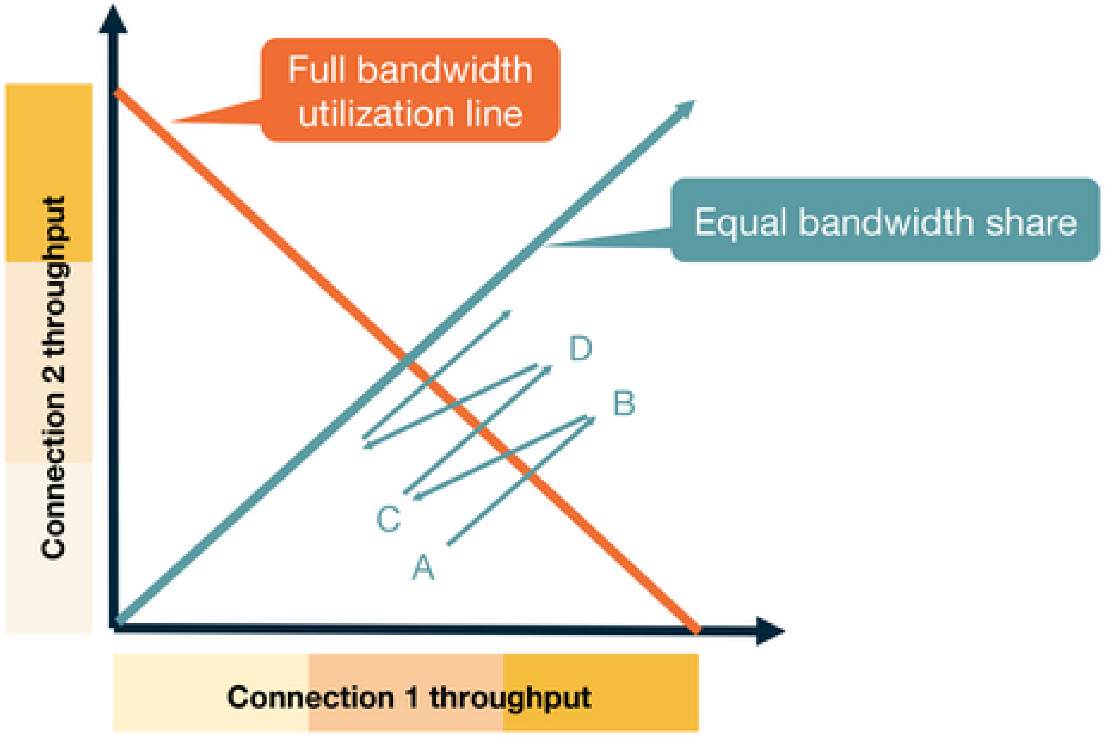
1. Additive Increase
   - Connection starts with constant initial window (typically 2)
   - Increase the window by 1 packet every RTT (round trip time)
   - Happens incrementally, do not wait for all ACKs from previous RTT
2. Multiplicative Decrease
   - When TCP detects that a loss event has occurred, it halves cwnd to a minimum value of 1
   - TCP Reno
     - Triple duplicate ACKs mean halve cwnd
     - A timeout event means set cwnd back to its starting value

Slow start in TCP

1. Want to be able to rapidly increase congestion window from a cold start
   - TCP Reno has a slow start phase where the congestion window is increased exponentially instead of linearly, as is the case in AIMD
     - Switches to AIMD after meeting a threshold
   - Slow start is also applied when a connection dies while waiting for a timeout to occur
     - Use previous knowledge about the connection

TCP Fairness

1. Consider a simple scenario where two TCP connections share a single link with bandwidth R
   - AIMD will lead to fairness in bandwidth sharing due to how it combats congestion



TCP Fairness

Quiz 3

1. TCP utilizes the Additive Increase Multiplicative Decrease (AIMD) policy for fairness. Other possible policies for fairness in congestion control would be Additive Increase Additive Decrease (AIAD), Multiplicative Increase Additive Decrease (MIAD), and Multiplicative Increase Multiplicative Decrease (MIMD). Would these other policies converge? If so, how would their convergence behavior differ from AIMD?
   - In AIAD and MIMD, the plotted throughput line will oscillate over the full bandwidth utilization line but will not converge as was shown for AIMD. On the other hand, MIAD will converge.
   - None of the alternative policies are as stable. The decrease policy in AIAD and MIAD is not as aggressive as AIMD, so those will not effectively address congestion control. In contrast, the increase policy in MIAD and MIMD is too aggressive.

Caution About Fairness

1. There can be cases where TCP is not fair
   - Difference in RTT of different TCP connections
     - Smaller RTT values would increase their congestion window faster than longer RTT values leading to an unequal sharing of bandwidth
   - Single application using multiple parallel TCP connections

Congestion Control in Modern Network Environments: TCP CUBIC

1. Increases in link speeds have led to a desire for changes in the TCP congestion control mechanisms mainly with a desire to improve link utilization

- TCP Reno has low network utilization, especially when the network bandwidth is high or the delay is large
  - High bandwidth delay product networks
- TCP CUBIC uses a cubic polynomial as a growth function
  - W(t) = C(t-K)^3 + Wmax
  - Wmax is the window when the packet loss was detected
  - C is a scaling constant
  - K is the time period that the above function takes to increase W to Wmax when there is no further loss even
    - K = (Wmax * B / C) ^ (1/3)

Quiz 4

1. Explain how in TCP Cubic the congestion window growth becomes independent of RTTs.
   - The key feature of CUBIC is that its window growth depends only on the time between two consecutive congestion events. One congestion event is the time when TCP undergoes fast recovery. This feature allows CUBIC flows competing in the same bottleneck to have approximately the same window size independent of their RTTs, achieving good RTT-fairness.

The TCP Protocol: TCP Throughput

1. Desire a simple model that predicts the throughput for a TCP connection
   - BW < MSS/RTT * 1/sqrt(p)
     - BW: Bandwidth
     - MSS: Maximum Segment Size
     - RTT: Round Trip Time
     - p = Probability loss

Datacenter TCP

1. Datacenter networks have proposed and implemented new TCP congestion control algorithms
   - Flow characteristics of DC networks are different from the public Internet
     - Many short flows that are sensitive to delay
   - Private entity often owns DC networks
     - Makes changing the transport layer easier since new algorithms do not need to coexist with old ones
   - DCTCP and TIMELY are two popular examples of TCP designed for DCs