TRANSPORT LAYER

Protocols:

- TCP (connection-oriented)
- UDP (connectionless)

TRANSPORT LAYER BASICS

Functions of Transport layer:

- Applications communicate simultaneously (email/social media)
- Data is received reliably && in order by correct applications
- Error-handling mechanisms

Roles	▪ Data is sent from application source to application destination but disregards: Destination type/media type/path/congestion/size of network ○ Delivery method ensures it's rebuilt when received ○ Data segmentation & reassembly

Segmentation/Reassembly process is achieved with 2 protocols:
1. **TCP:** Transmission Control Protocol
2. **UDP:** User Datagram Protocol

Primary responsibilities of protocols:
* Tracks communication between applications source/destination
* Data segmentation || Reassembling segments into streams of application data at destination

Conversation: Each set of data flowing between source application/destination application
Hosts can have multiple applications communicating simultaneously
Transport tracks/maintains these conversations

Data Segmentation/Reassembly:
Data must be ready for sending in pieces bc networks have limitations on data amts included in a single packet
Transport protocols segment application data into blocks of appropriate data size

Header: Used for reassembly & added to each data block (tracks data stream)
At destination: Transport reconstructs data into complete stream useful to application layer
Protocols at transport describe how header information is used to reassemble data

Identifying Applications:
Port number: Transport assigns each application an identifier
Each program process that needs access to network is assigned a port number
Transport uses ports to identify the application/service

Multiplexing (interleaving)	Segmenting data from many different users ○ Some data (video) can suck up bw/prevent simultaneous communication ○ ^Makes error recovery/retransmission of damaged data hard ○ Transport protocols provides ways to both send/receive data
To identify segments:	○ Transport adds a header to the segment ○ Header contains fields of bits ○ Values in these fields enable L4 protocols to do different functions in data communication

Reliability: Transport manages reliability requirements in a conversation
– Applications have different reliability requirements

IP	○ Only concerned with structure | addressing |routing of packets ○ Doesn't specify how the delivery/transportation of packets take place
Transport Protocols	○ Specifies how to transfer messages between hosts ○ TCP/UDP

TCP: Transmission Control Protocol

TCP	Reliable, full-featured protocol Ensures all data arrives at destination
UDP	Unreliable, simple protocol

Three basic operations of reliability in TCP:
1. Tracking transmitted data segments
2. Acknowledging received data
3. Retransmitting unacknowledged data

TCP..	▪ Breaks up message into segments ▪ Segments are numbered in sequence & passed to IP process for assembly into packets ▪ TCP tracks the number of segments sent to specific hosts from specific applications ▪ **If sender doesn't receive an acknowledgement in a given time:** – It assumes segments were lost & retransmits only lost segments ○ On receiving host, TCP reassembles message segments and passes them to the application ○ **FTP** and **HTTP** are examples of applications that use TCP to ensure data delivery ○ This process places additional overhead on network resources ○ To support reliability, more control data is exchanged between sending/receiving hosts ○ This control information is contained in a TCP header

TCP is good for:
– Applications where missing data corrupts communication
Example: Databases/Web browsers/Email
– Additional overhead is considered required

UDP: User Datagram Protocol

UDP	○ Better when additional overhead may cause delays in transmission ○ Best-effort delivery protocol (unreliable) ○ There is no acknowledgement that data is received at destination ○ No Transport layer processes that inform sender of success

UDP is good for:
– Applications that can tolerate some data loss during transmission
Example: VoIP/Video/Audio/
– Delays in transmission are unacceptable
– Acknowledgments would slow delivery
– Retransmissions are undesirable

TCP & UDP

TCP	UDP
Reliable—monitors message transmission, tracks data transfer to ensure receipt of all packets	Unreliable—no concept of acknowledgment, retransmission, or timeout –
Ordered—buffering provisions to ensure correct order of data packets	Not ordered—data arrives in order of receipt
Heavyweight—dedicated connection, provisions for speed and congestion control	Lightweight—no dedicated end-to-end connection, no congestion control
Streaming	Datagram oriented
Heavy overhead	Light overhead
Lower speed	Higher speed

TCP was initially described in RFC 793

TCP	**1. Connection oriented conversations by establishing sessions** **2. Reliable delivery**

	3. Ordered data reconstruction **4. Flow control**

1. **Connection oriented:** Negotiates/Establishes connection (session) of source/destination PRIOR to forwarding traffic
   1. Session establishment prepares devices to communicate
   2. Devices negotiate the amount of traffic forwarded at a time/Data between the two is closely managed
   3. Sessions are only terminated after all communication is complete
2. **Reliable Delivery: Ensuring each piece of data a source sends arrives at the destination**
   1. It's possible for data segments to get lost/corrupted during transmission
   2. TCP ensures all pieces reach their destination by having the source retransmit lost/corrupted data
3. **Same-Order Delivery:**
   1. By numbering and sequencing segments, TCP ensures these segments are reassembled in proper order
4. **Flow Control:** Prevents segmentation loss on the network and avoids retransmission needs
   1. Hosts have limited resources (memory/bandwidth)
   2. When TCP becomes aware resources are overtaxed, it can request the sending application reduce the data flow rate
   3. Done by TCP regulating the amount of data the source transmits

Role of TCP: Stateful protocol: The ability to keep track of conversations within a session

1. After an established connection, TCP is able to keep track of the conversation within that session Example: Sender expects destination to acknowledge it received data
2. TCP tracks which information has been sent/has been acknowledged
3. If data isn't acknowledged: Sender assumes data didn't arrive: It resends the data

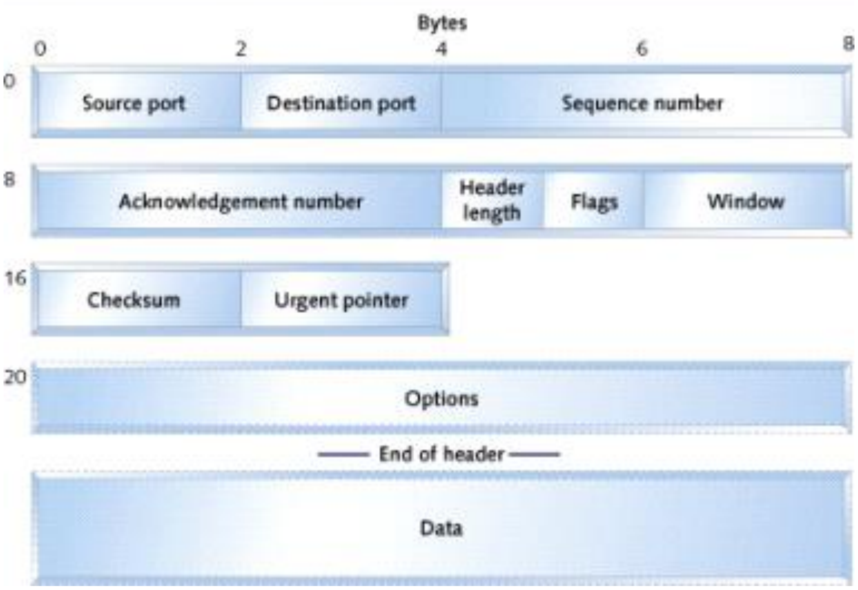
The stateful session:	○ Begins with session establishment ○ Ends when session is closed with session termination ○ Maintaining information requires unnecessary resources for a stateless protocol like UDP ○ TCP gets additional overhead to gain these functions

TCP segment has 20 bytes of overhead in the header encapsulating application layer data
UDP segments only have 8 bytes of overhead

TCP Extra Overhead:

Sequence number	○ **32 bits** ○ Used for data reassembly purposes
Acknowledgement number	○ **32 bits** ○ Indicates that the data has been received
Header length	○ **4 bits** ○ Known as "data offset" ○ Indicates the length of the TCP segment header
Reserved	○ **6 bits** ○ This field is reserved for the future
Control bits	○ **6 bits** ○ Includes bit codes (flags) ○ Flags indicate the purpose and function of the TCP segment
Window Size	○ **16 bits** ○ Indicates the number of segments that can be accepted at one time
Checksum	○ **16 bits** ○ Used for error checking of the segment header and data
Urgent	○ **16 bits** ○ Indicates whether data is urgent

Figure 4: TCP segment format



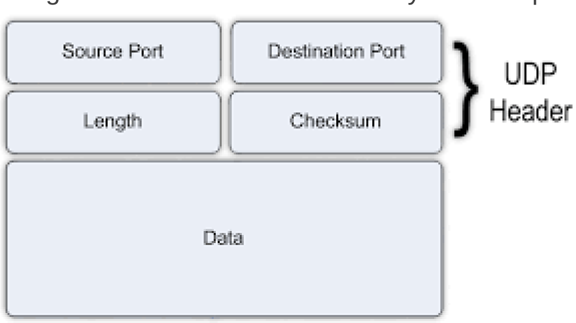
UDP: User Datagram Protocol

Best effort transport protocol: Described in RFC 768 it's a simple protocol without reliability and flow control

UDP	**1. Connectionless** **2. Unreliable delivery** **3. No ordered data reconstruction** **4. No flow control**

1. **Connectionless:** UDP doesn't establish a connection between hosts before data is sent/received
   2. Unreliable delivery: UDP doesn't provide services to ensure data is delivered reliably
1. No processes to have the sender re-transmit data that is lost/corrupted
3. **No order data reconstruction:**
1. Occasionally data is received in a different order than it was sent
2. UDP doesn't provide mechanisms for reassembling data in original sequence
3. Data is delivered to the application in the order it arrives
4. **No flow control:**
1. No mechanisms to control the data amount transmitted by source to avoid overwhelming destination
2. The source sends data
3. If resources on destination become overtaxed, most likely data drops until resources become available
4. Unlike TCP: No mechanism for automatic re transmission

Role of UDP: Connectionless protocol: Doesn't include reliability/flow control mechanisms of TCP
1. Suited for applications that can tolerate data loss (VoIP, video/audio streams, DNS, DHCP)
2. **Datagrams**: Pieces of communication in UDP
3. Datagrams are sent as best effort by the transport layer

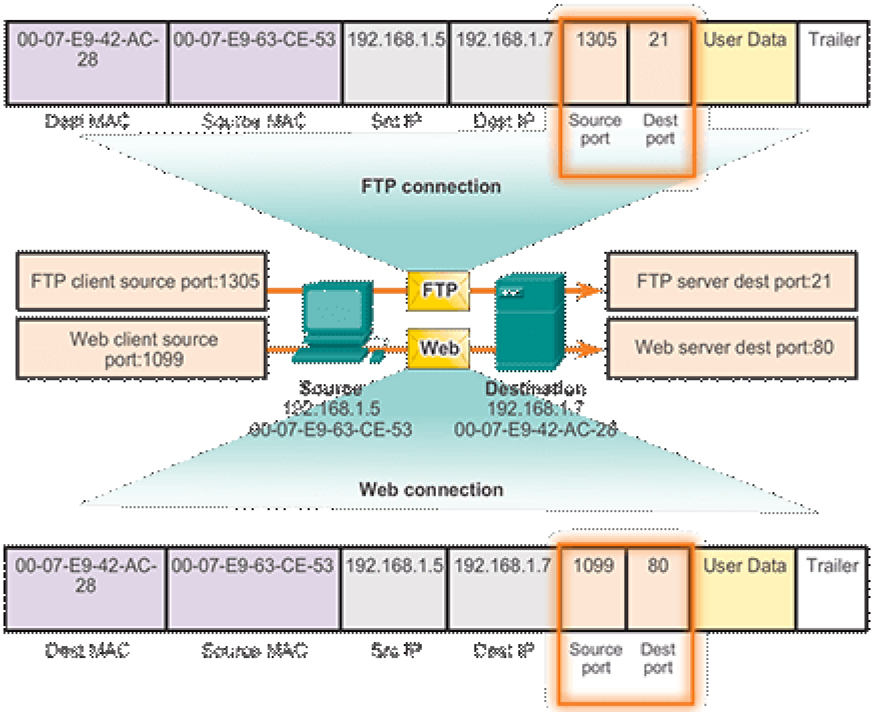


TCP/UDP Port Addressing

The header of each segment/datagram has a source/destination port

Source port number: The number associated with the originating application on the local host

Destination number: The number associated with the destination application on the remote host



Messages delivered using TCP/UDP protocols or services are identified by a port number
Port number: Numeric identifier in each segment used to keep track of specific conversations/destination services requested
Every message a host sends contains both a source/destination port
Destination Port:

1. Client places a destination port in the segment to tell the destination server what service is being requested

Example: A client specifies port 80 in destination: A server receives the message & knows web services are requested

1. A server can offer more than one service simultaneously

Source port number:

- Randomly generated by sender to identify a conversation between 2 devices
- Allows for multiple conversations to occur simultaneously

Example: A device can send multiple HTTP service requests to a web server at the same time

1. Separate conversations are tracked by source ports
2. The source/destinations ports are placed within a segment
3. Segments are encapsulated within an IP packet
4. IP packet contains IP address of the source/destination

Socket: The combination of the source/destination IP addresses and the source/destination port numbers

- Sockets are used to identify the server/service being requested by the client

The combination of L4 port # & L3 IP address of host || Identifies application process running on individual host
Socket pair: Consists of the source/destination IP addresses/port numbers and identifies the specific conversation

1. Client socket may look like this (1099 represents source port): 192.168.1.5:1099
2. Socket on a web server might be 192.168.1.7:80
3. Together these two sockets combine to form a socket pair

   192.168.1.5:1099 & 192.168.1.7:80 = Socket pair

Sockets:

- Endpoints are known so data can move from an application on 1 host to application on another
- Multiple processes running on a client distinguishes themselves
- Multiple connections to a server process to be distinguished

The source port of a client request is randomly generated

1. The port acts like a return address for the requesting application
2. L4 keeps track of this port/application who initiated request
3. So when a response is returned: It can be forwarded to the correct application
4. The requesting application port is used as the destination port in the response from the server

IANA: Internet Assigned Numbers Authority: Assigns port numbers

Different port number types:	**1. Well-Known ports** **2. Registered ports** **3. Dynamic or private ports**

Well-known	○ **0 – 1023** ○ Numbers reserved for services/applications ○ Commonly used for: – HTTP, IMAP (Internet Message Access Protocol), SMTP (Simple Mail Transfer Protocol: Email) and Telnet ○ Defining these ports for server applications, allow client applications to be coded to request connection to that specific port/associated service
Registered	○ **1024 – 49151** ○ Assigned to user processes/applications ○ Individual applications a user chose to install, rather than common ○ When not used for a server resource, they can be dynamically selected by a client as its source port
Dynamic or Private	○ **49152 – 65535** ○ **AKA ephemeral ports** ○ Assigned dynamically (usually) to client applications when it initiates a connection to a service ○ Often used to identify client application during communication ○ Client uses a well-known port to identify/connect to the service being requested on the server ○ Uncommon for a client to connect to a service using private port (p2p programs do use these)

Well-Known/Registered TCP Ports:

Registered TCP Ports		Well-Known TCP Ports:	
MSN Messenger	1863	**FTP**	21
Cisco SCCP (VoIP)	2000	**Telnet**	23
Alternate HTTP	8008	**SMTP**	25
Alternate HTTP	8080	**HTTP**	80
		IMAP	143
		IRC	194
		HTTPS	443

Well-Known/Registered UDP Ports:

Registered UDP Ports:		Well-Known UDP Ports:	
RADIUS Authentication Protocol	1812	**TFTP**	69
RTP (Voice/Video Transport Protocol)	5004	**RIP**	520
SIP (VoIP)	5040		

Well-Known/Registered Common TCP/UDP Ports:

Registered TCP/UDP Common Ports:		Well-Known TCP/UDP Common Ports:	
MS SQL	1433	**DNS**	53
WAP (MMS)	2948	**SNMP**	161
		AOL IM, IRC	531

It's good to know which active TCP connections are open/running on a networked host
Netstat: An important network utility that can be used to verify connections
Lists: Protocol in use || Local address/port number || Foreign address/port number || Connection state

- Used to examine open connections when performance appears compromised

- Unexplained TCP connections pose a threat bc they can indicate something/someone is connected to local host
- Unnecessary TCP connections consume system resources, slowing host's performance

A	Protocol used
B	Source Port
C	Address of name of remote host
D	Destination Port
E	Connection State

C:\> netstat
Active Connections

Proto	Local Address	Foreign Address	State
TCP	Kenpc:3126	192.168.0.2:netbios-ssn	ESTABLISHED
TCP	Kenpc:3158	207.138.126.152:http	ESTABLISHED
TCP	Kenpc:3159	207.138.126.169:http	ESTABLISHED
TCP	Kenpc:**3166**	www.cisco.com:**http**	**ESTABLISHED**
A	**B**	**C D**	**E**

TCP/UDP Segmentation
PDU: Protocol Data Units
Dividing application data into segments:

- Ensures data is transmitted within limits of media
- Ensures data from different applications can be multiplexed on to the media
- TCP/UDP handle segmentation differently

TCP	○ Each segment header contains a sequence number ○ This allows Transport layer functions on the destination host to reassemble segments in order transmitted ○ Ensures destination application has data in the exact form that the sender intended
UDP	○ **Services using UDP also track conversations between applications BUT:** ○ Not concerned with order information was transmitted, or maintaining a connection ○ **No sequence number in header** ○ Simpler design/less overhead than TCP (faster data xfer) ○ Information might arrive in a different order than transmitted ○ Why? ○ Packets can take different paths: An application using UDP should know data may not arrive in order

Both protocols support communication between source/destination, but the way communication occurs is different
A key distinction between TCP/UDP is reliability:

- Reliability of TCP communication is obtained through using connection-oriented sessions
- Before a host using TCP sends data to another: TCP initiates a process to create a connection with destination
- Stateful connection enables tracking a session between hosts

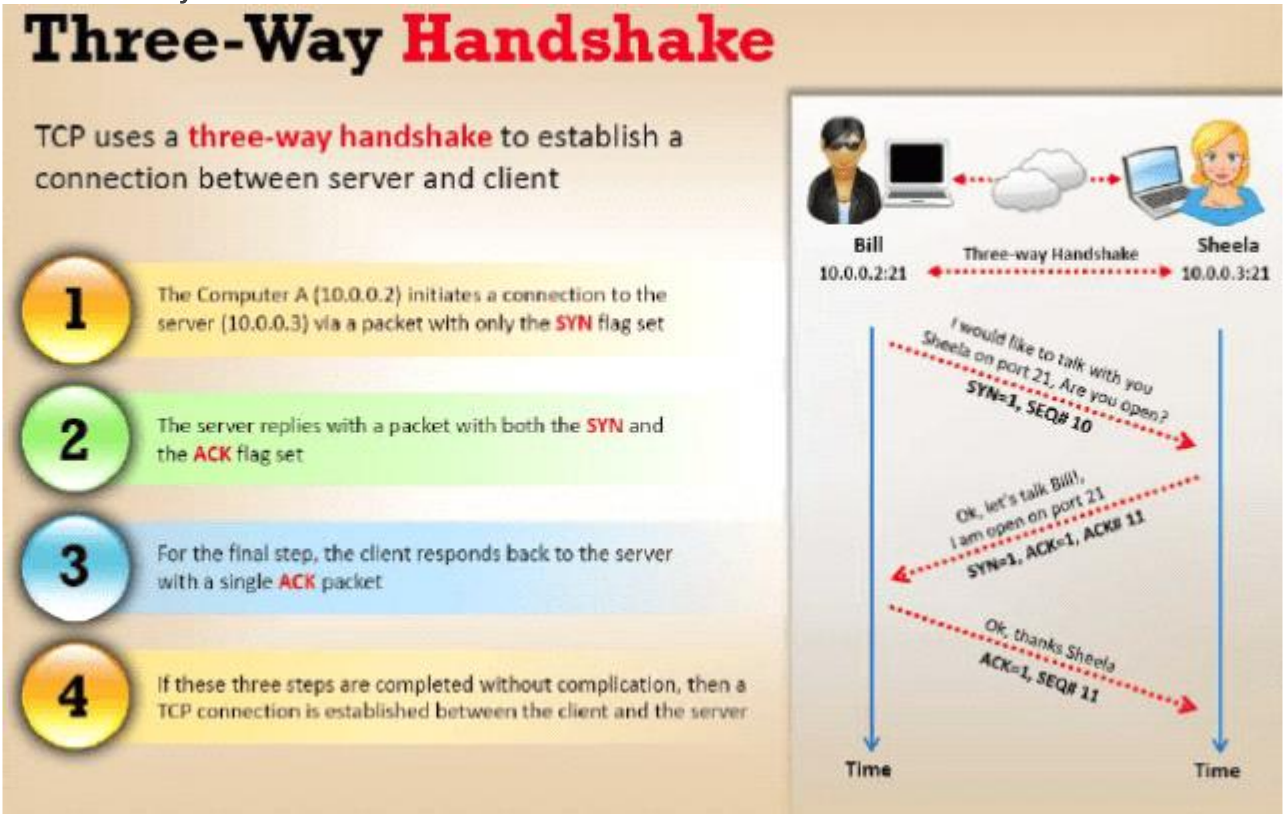
TCP Server Processes

- Servers can run multiple application processes at the same time
- Processes wait until a client initiates communication with a request for information
- Each process is configured to use a port (default/admin)
- Individual servers can't have 2 services on the same port
- It's common to provide more than 1 service at the same time (Web & FTP server)
- Security: Restrict access only to ports associated with services/applications for authorized users

TCP Connection Establishment and Termination
– TCP is a full-duplex protocol, where each connection represents two 1way communication streams/sessions
– To establish the connection, hosts perform a 3-way handshake

– Control bits in TCP headers indicate progress/status of the connection

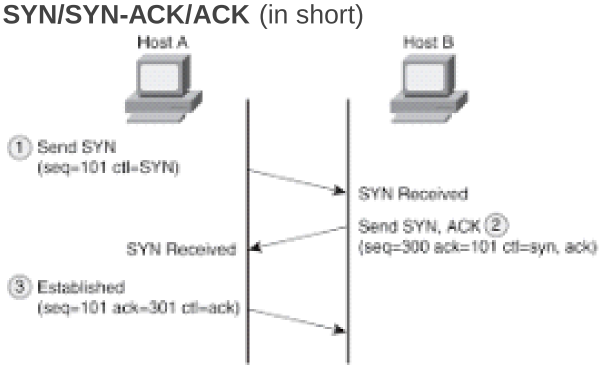
The Three-Way Handshake



3-way Handshake	○ Establishes destination is present on network ○ Verifies destination has an active service ○ Verifies destination is accepting requests on destination port the client will use for the session ○ Informs destination that the source client intends to establish a session on that port

TCP connections: The host establishes the connection with the server

- **Initiating client requests a client-to-server communication session with the server**
- Server acknowledges a client-to-server communication session & requests a server-to-client session
- Initiating client acknowledges server-to-client communication session
- Send **SYN** (SEQ=100 CTL=SYN)
- **SYN** Received (SEQ=300 ACK=101 CTL=SYN,ACK)
- ESTABLISHED (SEQ=101 ACK=301 CTL=ACK)
- **SYN/SYN-ACK/ACK** (in short)



Within the TCP segment header: Six 1bit fields contain control information used to manage TCP processes.

URG	Urgent pointer field significant

ACK	Acknowledgement field significant
PSH	Push function
RST	Reset the connection
SYN	Synchronize sequence numbers
FIN	No more data from sender

Analysis of the 3-Way Handshake:

- **Initiating client requests a client-to-server communication session with the server**

1. TCP client starts handshake by sending a segment with a SYN (synchronized sequence number) control flag. It indicates an initial value in the sequence number field, within the header
2. **ISN: Initial Sequence Number:** This initial value for the SYN, is RANDOMLY chosen & used to track data flow from client-to-server for the session.
3. The ISN in the header of each segment increases by 1 for each byte of data sent from client-to-server as the conversation continues.
4. The SYN control flag is set and the relative sequence number is at 0.
5. The true values are 32-bit binary numbers.

Second. Server acknowledges client-to-server session & requests server-to-client session

1. TCP server must acknowledge receipt of SYN segment from client to establish a session from client-to-server
2. The server sends a segment back to client with the ACK (acknowledgement) flag. It indicates the ACK # is good
3. With this flag in the segment, client recognizes acknowledgement that the server received from SYN via TCP client
4. The value of the ACK number field is equal to the ISN + 1
5. This establishes a session from the client-to-server
6. ACK flag remains set for the balance of the session

Recall the conversation between the client/server is actually two 1way sessions:
A. One from client to server
B. One from server to client

- In 2nd step, server must initiate a response to the client
- To start the session, a server uses the SYN flag in the same way the client did
- It sets the SYN flag in the header to establish a session from server-to-client
- The SYN flag indicates the initial value of the sequence number field in the header
- This value is used to track data flow in this session from server-back-to-client

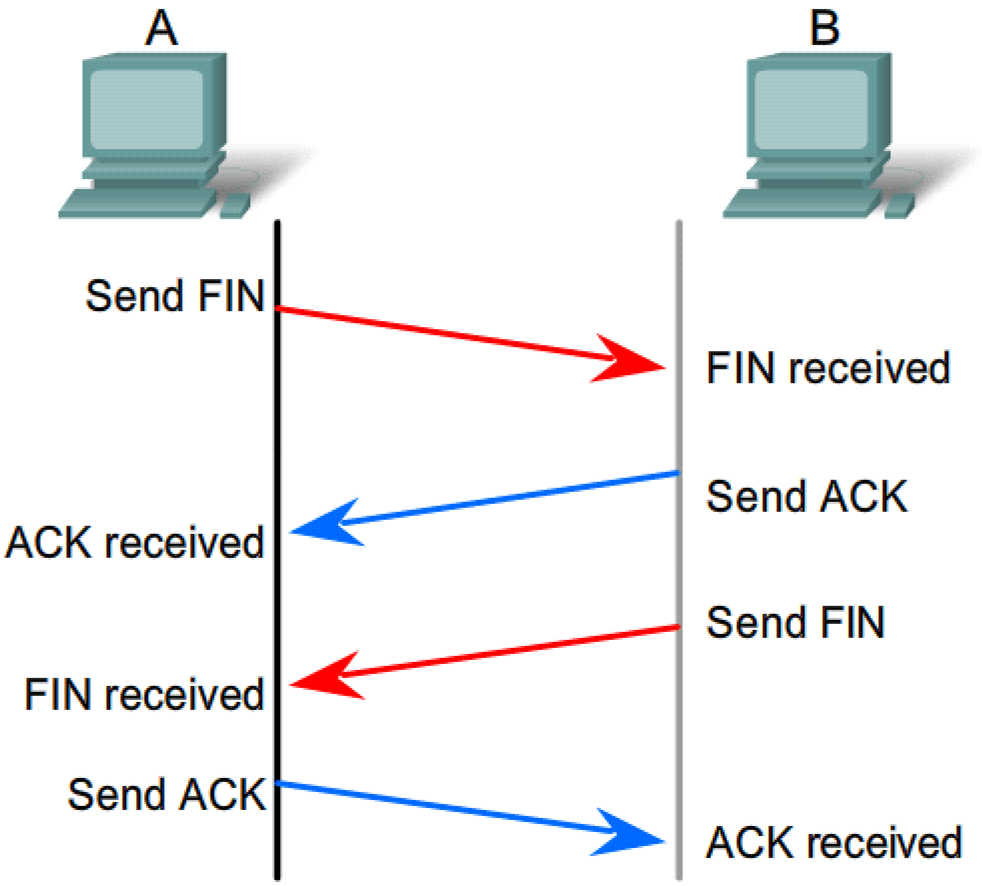
Third. The initiating client acknowledges the server-to-client communication session.

1. TCP client responds with a segment containing ACK that is the response to SYN sent by the server
2. No user data in this segment
3. The value in the acknowledgement number field contains 1 more than the ISN received from server
4. After both sessions are established, all additional segments exchanged in this session will have the ACK flag set

S**ecurity can be added to the data network by:**

- Denying the establishment of TCP sessions
- Only allowing sessions to be established for specific services
- Only allowing traffic as a part of already established sessions
- These security measures can be implemented for all TCP sessions or selected sessions

TCP session Termination Analysis



- To close a connection, the Finish (FIN) flag must be set in the segment header
- To end each 1-way session, a 2-way handshake is used, consisting of a FIN segment and ACK segment
- To terminate a single conversation, 4 exchanges are needed to end both sessions

- **When client has no more data to send, it sends a segment with FIN flag set**
- Server sends an ACK to acknowledge receipt of the FIN to terminate the session from client-to-server
- Server sends a FIN to client to terminate the server-to-client session
- Client responds with ACK to acknowledge the FIN from the server

Reliability and Flow Control
- Two main advantages TCP has over UDP
- When services send data using TCP, segments may arrive at destination out of order
- For data to be understood, data in these segments is reassembled
- Sequence numbers are assigned in the header of each packet to achieve this
- During session setup an ISN is set. This ISN represents the starting value for bytes in the session/is transmitted to the receiving application.
- As data is transmitted during the session, the sequence number is incremented by number of bytes transmitted. This byte tracking enables each segment to be uniquely identified/acknowledged.
- Segment sequence numbers enable reliability by indicating how to reassemble/reorder received segments
- Receiving TCP process places data from a segment into a receiving buffer
- Segments are placed in proper sequence number order and passed to application layer when reassembled
- Any segments that arrive with out of order sequence numbers are held for processing
- When segments with missing bytes arrive, they are put in order

SEQ (sequence number) and ACK number are used together to confirm receipt of bytes of data contained in transmitted segments. SEQ number = Relative number of bytes transmitted in session (including bytes in current segment)
Expectational acknowledgement: TCP uses ACK number sent back to source to indicate next byte receiver expects

Source is informed destination has received all bytes in stream, up to but not including the byte indicated by ACK #. The sending host is expected to send a segment that uses a SEQ number equal to the ACK number.

Window size: A field in the TCP header that enables the management of lost data and flow control.

- Reduce overhead: Multiple segments of data can be sent & ACKed with a single TCP message in opposite direction
- This acknowledgement contains an ACK number based on total number of bytes received in session
- Example: Starting with a SEQ # of 2000, if 10 segments of 1000 bytes each were received, an ACK number of 12001 would be returned to the source (1000×10 + 2000 + 1 byte).
- The amount of data a source can transmit before acknowledgement must be received in its window size

Handling Segment Loss

Destination host service using TCP only ACK's data for contiguous SEQ bytes. If 1/more segments are missing, only data in the first SEQ of bytes are ACK'd.

Example:

- Segments with sequence numbers 1500-3000 and 3400-3500 were received.
- The ACK is 3001 (3000 is the last number before loss)
- There are segments with SEQ numbers 3001-3399 that haven't been received.

SACK: Selective Acknowledgements:

- Allows possible destination to ACK bytes in discontinuous segments
- Host would only need to retransmit missing data
- Only if you have 2 hosts that support SACKs (optional)

Congestion Avoidance:

Another way to control data flow is dynamic window sizes. When network resources are constrained, TCP reduces window size to require received segments be acknowledged more frequently. This slows rate of transmission.

- Receiving host sends window size value to sending host to indicate number of bytes it's prepared to receive
- If destination needs to slow communication, it can send a smaller window size value to source as part of an ACK
- Receiving host has congestion: It can respond to sending host with a segment that specifies reduced window size
- After congestion is gone, window size will be increased

UDP: Key application layer protocols that use UDP

DNS	SNMP	DHCP	RIP	TFTP	VoIP	Online games

- Applications that use UDP send small amounts of data that can fit in one segment
- Some applications send larger amounts that must be split into multiple segments
- UDP PDU is referred to as a datagram
- When multiple datagrams are sent to a destination, they can take many paths and arrive in the wrong order
- UDP doesn't track SEQ numbers and has no way to reorder datagrams into order
- UDP reassembles data in order received.

UDP Client Processes:

The UDP client process randomly selects a port number from the range of dynamic port numbers and uses this as the source port for the conversation. The destination port is usually the well-known or registered port number assigned.

Randomized source ports help with security