30. TCP and UDP

30. Transport Layer: TCP, UDP

30.1. Cheat sheet

- Layer: 4 (transport)
- Purpose: Establish connections between individual processes on machines (TCP and UDP). Guarantee that packets are delivered successfully and in the correct order (TCP only).
- Vulnerability: On-path and MITM attackers can inject data or RST packets. Off-path attackers must guess the 32-bit sequence number to inject packets.
- Defense: Rely on cryptography at a higher layer (TLS). Use randomly generated sequence numbers to stop off-path attackers.

30.2. Networking background: Ports

Recall that IP, the layer 3 (inter-network) protocol, is a best-effort protocol, meaning that packets can be corrupted, reordered, or dropped entirely. Also, IP addresses uniquely identify machines, but do not support multiple processes on one machine using the network (e.g. multiple browser tabs, multiple applications).

The transport layer solves the problem of multiple processes by introducing **port numbers**. Each process on a machine that wants to communicate over the network uses a unique 16-bit port number. Recall that port numbers are unique per machine, but cannot be used for global addressing–two machines can have processes with the same port number. However, an IP address and a port number together uniquely identify one process on one machine.

On client machines, such as your laptop, port numbers can be arbitrarily assigned. As long as each application uses a different port number, incoming packets can be sorted by port number and directed to the correct application. However, server machines offering services over the network need to use constant, well-known port numbers so client machines can send requests to those port numbers. For example, web servers always receive HTTP requests at port 80, and HTTPS (secure) requests at port 443. Ports below 1024 are "reserved" ports: only a program running as root can receive packets at those ports, but anyone can send packets to those ports.

The transport layer has 2 main protocols to choose from: TCP guarantees reliable, in-order packet delivery, while UDP does not. Both protocols use port numbers to support communication between processes. The choice of protocol depends on the context of the application.

30.3. Protocol: UDP

UDP (user datagram protocol) is a best-effort transport layer protocol. With UDP, applications send and receive discrete packets, and packets are not guaranteed to arrive, just like in IP. It is possible for datagrams to be larger than the underlying network's packet size, but this can sometimes introduce problems.

The UDP header contains 16-bit source and destination port numbers to support communication between processes. The header also contains a checksum (non-cryptographic) to detect corrupted packets.

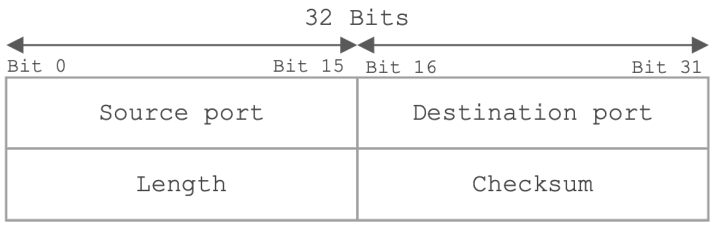

Figure 1: UDP header

30.4. Protocol: TCP

TCP (Transmission Control Protocol) is a reliable, in-order, connection-based stream protocol. In TCP, a client first establishes a connection to the server by performing a handshake. Once established, the connection is reliable and in order: TCP handles resending dropped packets until they are received on the other side and rearranging any packets received out of order. TCP also handles breaking up long messages into individual packets, which lets programmers think in terms of high-level, arbitrary-length bytestream connections and abstract away low-level, fixed-size packets.

Like UDP, the TCP header contains 16-bit source and destination port numbers to support communication between processes, and a checksum to detect corrupted packets. Additionally, a 32-bit **sequence number** and a 32-bit **acknowledgment (ACK) number** are used for keeping track of missing or out-of-order packets. Flags such as SYN, ACK, and FIN can be set in the header to indicate that the packet has some special meaning in the TCP protocol.

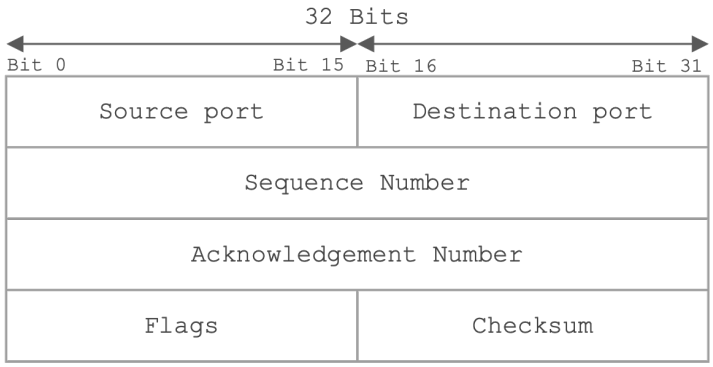

Figure 2: TCP header

A unique TCP connection is identified by a 5-tuple of (Client IP Address, Client Port, Server IP Address, Server Port, Protocol), where protocol is always TCP. In other words, a TCP connection is a sequence of back-and-forth communications between one port on one IP address, and another port on another IP address.

TCP communication works between any two machines, but it is most commonly used between a **client** requesting a service (such as your computer) and a **server** providing the service. To provide a service, the server waits for connection requests (sometimes called listening for requests), usually on a well-known port. To request the service, the client makes a connection request to that server's IP address and well-known port.

A TCP connection consists of two bytestreams of data: one from the client to the server, and one from the server to the client. The data in each stream is indexed using sequence numbers. Since there are two streams, there are two sets of sequence numbers in each TCP connection, one for each bytestream.

In every TCP packet, the sequence number field in the header is set to the index of the first byte sent in that packet. In packets from the client to the server, the sequence number is an index in the client-to-server bytestream, and in packets from the server to the client, the sequence number is an index in the server-to-client

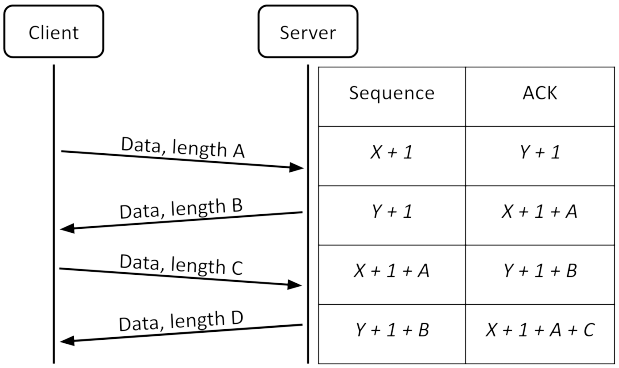

Figure 3: Diagram of TCP communication, with sequence numbers and ACK numbers

bytestream. If packets are reordered, the end hosts can use the sequence numbers to reconstruct the message in the correct order.

To ensure packets are successfully delivered, when one side receives a TCP packet, it must reply with an acknowledgment saying that it received the packet. If the packet was dropped in transit, the recipient will never send an acknowledgment, and after a timeout period, the sender will re-send that packet.

If the packet is delivered, but the acknowledgment is dropped in transit, the sender will notice that it never received an acknowledgment and will re-send the packet. The recipient will see a duplicate packet (since the original packet was delivered), discard the duplicate, and re-send the acknowledgment.

Sending acknowledgment packets is wasteful in a two-way communication, so TCP combines acknowledgment packets with data packets. Each TCP packet can contain both data and an acknowledgment that a previous packet was received.

To support acknowledgments, the acknowledgment (ACK) number in the header is set to the index of the last byte received, plus 1. (This is equivalent to the index of the next byte the sender expects to receive.) In other words, in packets from the client to the server, the ACK number is the next unsent byte in the server-to-client stream, and in packets from the server to the client, the ACK number is the next unsent byte in the client-to-server stream.

Note that in each packet, the sequence number is an index in the sender's bytestream, and the ACK number is an index in the recipient's bytestream.

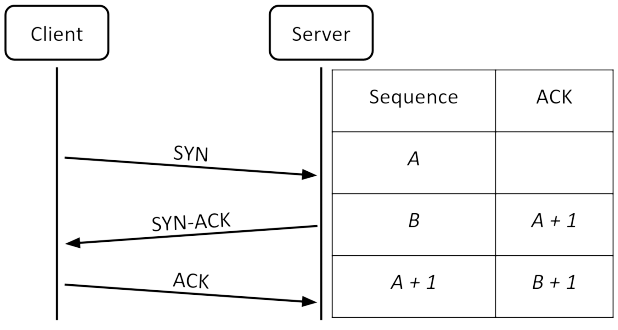

Figure 4: Diagram of the TCP 3-way handshake

Note that the sequence numbers do not start at 0 (for a security reason discussed below). Instead, to initiate a connection, the client and server participate in a three-way **TCP handshake** to exchange random initial sequence numbers.

1. The client sends a **SYN packet** (a packet with no data and the SYN flag set) to the server. The client sets the sequence number field to a random 32-bit **initial sequence number (ISN)**.
2. If the server decides to accept the request, it sends back a **SYN-ACK packet** (a packet with no data and both the SYN flag and ACK flag set). The server sets the sequence number field to its own random 32-bit initial sequence number (note that this is different from the client's ISN). The acknowledgment number is set to the client's initial sequence number + 1.
3. The client responds with an **ACK packet** (a packet with no data and the ACK flag set). The sequence number is set to the client's initial sequence number + 1 and the acknowledgement number is set to the server's initial sequence number + 1.

To end a connection, one side sends a FIN (a packet with the FIN flag set), and the other side replies with a FIN-ACK. This indicates that the side that sent the FIN will not send any more data, but can continue accepting data. This leaves the TCP connection in a "half closed" state, where one side stops sending but will receive and acknowledge further information. When the other side is done, it sends its own FIN as well, and it is acknowledged with a FIN-ACK reply.

Connections can also be unilaterally aborted. If one side sends a RST packet with a proper sequence number, this tells the other side that "I won't send any more data on this connection and I won't accept any more data on this connection." Unlike FIN packets, RST packets are not acknowledged. A RST usually indicates something went wrong, such as a program crashing or abruptly terminating a connection.

30.5. Tradeoffs between TCP and UDP

TCP is slower than UDP, because it requires a 3-way handshake at the start of each connection, and it will wait indefinitely for dropped packets to be sent again. However, TCP provides better correctness guarantees than UDP.

UDP is generally used when speed is a concern. For example, DNS requires extremely short response times, so it uses UDP instead of TCP at the transport layer. Video games and voice applications often use UDP because it is better to just miss a request than to stall everything waiting for a retransmission.

30.6. Attack: TCP Packet Injection

The main attack in TCP is **packet injection**. The attacker spoofs a malicious packet, filling in the header so that the packet looks like it came from someone in the TCP connection.

A related attack is **RST injection**. Instead of sending a packet with malicious data, the attacker sends a packet with the RST flag, causing the connection to abruptly terminate. This attack is useful for censorship: for example, Comcast used RST injection to abruptly terminate BitTorrent uploads.

Recall that that there are three types of network attackers. Each one has different capabilities in attacking the TCP protocol.

Off-path Adversary: The off-path adversary cannot read or modify any messages over the connection. Therefore, to attack a TCP communication, an off-path adversary must know or guess the values of the client IP, client port, server IP, and server port. Usually, the server IP address and port are well-known. Whether we know the client IP or port depends on our threat model. The off-path attacker must also guess the sequence number to inject a packet into the communication, because if the sequence number is too far off from what the recipient is expecting, it will reject the spoofed packet. Sanity check: What is the approximate probability of correctly guessing a random sequence number?[1]

On-path Adversary: The on-path adversary can read, but not modify messages. Since they can read the sequence numbers, IP addresses, and ports being used in the connection, an on-path adversary can inject messages into a TCP connection without guessing any values. As a concrete example, assume Alice has just sent a packet to Bob with sequence number X, and Bob responds with a packet of his own with sequence

[1]The sequence number is 32 bits, so guessing a random sequence number succeeds with probability $1/2^{32}$.

number Y and ACK X + 1. An on-path adversary Mallory wants to inject data into this TCP connection. While she cannot stop Alice from responding (because Mallory is not a man-in-the-middle), Mallory can race Alice's next packet with her own, using sequence number X + 1, ACK Y + 1, and Alice's IP and port. Since TCP on its own does not provide integrity, Bob will not be able to distinguish which message actually came from Alice, and which one came from Mallory.

In-path Adversary: The in-path (man-in-the-middle) adversary has all the powers of the on-path adversary and can additionally modify and block messages sent by either party. As a result, the same attack as the on-path adversary outlined above applies, and in addition, the in-path adversary doesn't have to race the party they are spoofing. A man in the middle can just block the message from ever arriving to the other party and send their own.

30.7. Defenses: TLS, random initial sequence numbers

The main problem here is that TCP by itself provides no confidentiality or integrity guarantees. To prevent injections like these, we rely on TLS, which is a higher-layer protocol that secures TCP communication with cryptography.

One important defense against off-path attackers is using random, unpredictable initial sequence numbers. This forces the off-path attacker to guess the correct sequence number with very low probability.