03. Transport Layer

Skipped 3.8 Evolution of Transport-Layer Functionality

3.1 Introduction and Transport-Layer Services

The transport layer is responsible for creating a logical communication layer between two end systems. This means that whatever network, link and physcal layer may be used, yet two machines should be able to smoothly communicate without worrying about the underlying infrastructure.

In the internet era, we primarily use two protocols at the transport layer, TCP or UDP. TCP provides an established connection between two end systems, allowing systems to communicate over a single connection. UDP simply allows you to send packets to a target, but there is no guarantee for ordering or arrival of said packets. Therefore the application should pick protocol accordingly. The primary job of the transport layer is to extend the IP protocol's functionality. TCP does this by making sure that all packets arrive. TCP also provides congestion control, which controls how many packets a single system may send across the links, preventing swamping of the links.

3.2 Multiplexing and Demultiplexing

Multiplexing and demultiplexing is the strategy used to determine which application on a system a transport layer segment is intended for. If a user is running 5 applications on their system, it's not clear which of the five a segment is intended for. To solve this problem, the sender multiplexes, and the receiver demultiplexes.

This is the process of attaching source IP and port, and destination IP and port to the transport layer segment header. The receiver is now able to determine which application the segment is intended for, and it is able to provide an address where the receiver may send segments back to the sender. It's worth noting that TCP and UDP headers also have other properties in their headers, but the destination. The TPC protocol doesn't transfer source and destination ports because it acts over an established connection, and UDP doesn't keep the source and destination IPs.

3.3 Connectionless Transport: UDP

UDP is a rather barebones protocol which sits atop of the IP protocol in the link layer, and follows IPs rule of giving its best attempt to deliver the segment with no guarantees. The only steps the UDP protocol adds are the multiplexing features, containing source IP, source port, destination IP and destination port in its header along with its length and a checksum to validate integrity.

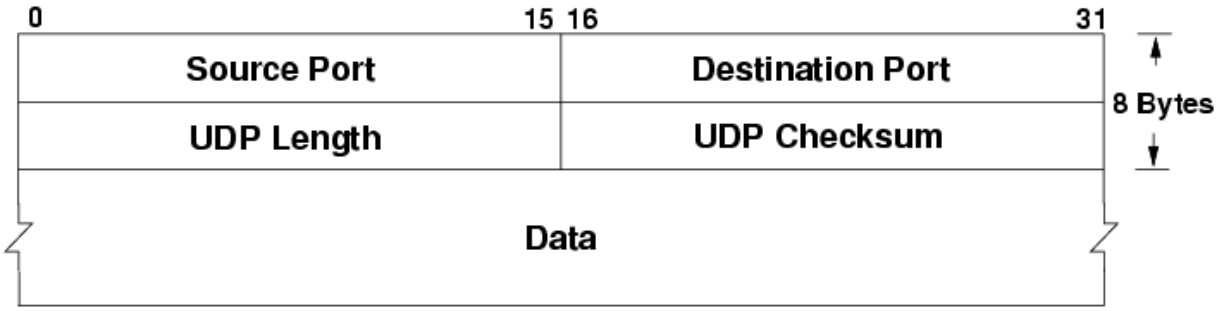
UDP is therefore favored when:

1. You want fine-grained control over what data is sent, and when it is being sent. TCP has congestion control which may delay the data which isn't suitable for realtime applications.
2. You don't need to establish a connection. DNS uses UDP because it doesn't care if the packet doesn't reach the destination, it'll just retry with another server. You also dodge the delay of the TCP handshake.
3. You don't want a connection state. There is also no need to keep a consistent connection, possibly allowing for more communication with more clients at once.
4. You want as little overhead as possible. The UDP header is only 8 bytes compared to TCPs 20 bytes.

Below is a table of common transport layer protocols for common applications.

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Secure remote terminal access	SSH	TCP
Web	HTTP, HTTP/3	TCP (for HTTP), UDP (for HTTP/3)
File transfer	FTP	TCP
Remote file server	NFS	Typically UDP
Streaming multimedia	DASH	TCP
Internet telephony	typically proprietary	UDP or TCP
Network management	SNMP	Typically UDP
Name translation	DNS	Typically UDP

The UDP header has four fields, each being two bytes each. Source port, destination port, length and a checksum. The length contains the byte size of the data + header in the UDP packet, and the checksum is used to verify the integrity of the data.



The checksum is constructed by performing 1s complement on all the 16-bit words in the packet. The receiver verifies by performing 1s complement on the data and the checksum. If the integrety of the data is kept, the result should be an all-ones 16-bit word.

The reason UDP has error-checking is because it's designed to run atop of IP which should function on about any layer-2 infrastructure. It's a backup to ensure that no error

has occured during transit, regardless of what other error checking the lower layers perform.

3.4 Principles of Reliable Data Transfer

Reliable data transfer is all about ensuring that all packets arrive at the target destination, and in the correct order. To ensure this, we need an Automatic Repeat Request protcol (ARQ). It has the following components:

1. Error detection: a way to detect bit errors,
2. Receiver feedback: a way to have receiver send back acknowledgement or not-ackowledged packets to determine further action,
3. Retransmission: a way to re-transmit packets.

To keep track of what packets make it and which don't, we introduce a sequence number which is incremented for each packet. In a stop-and-wait protocol we don't need to reply with which sequence number was acknowledged, because there's only one packet in transmission at a time. In these protocols, the sequence number is either 1 or 0.

The sender may also have a timer, and if there is no acknowledgement by the end of the timer, it'll re-send the packet. In typical protocols, one re-sends the packet after a reasonable time, which may lead to duplicate packets. Duplicate packets are not a concern, because we'll simply ackowledge the first, and discard the second.

It's extremely inefficient to operate on a single packet at once, which is why pipelining protocols are the standard. Pipelining allows for sending multiple packets without waiting for acknowledgement. This strategy comes with the following consequences:

1. The number range of sequence numbers must be bigger (since we're not operating on 1 and 0 anymore)
2. The sender and receiver have to buffer more packets (because there are multiple packets in transmission at once)
3. Both of the above factors are determined by the recovery strategy. Common ones are Go-Back-N and Selective-Repeat

The Go-Back-N strategy allows N unacknowledged packets in the pipeline at once, meaning if $N = 8$, then packet 11 cannot be sent before $0..3$ are ackowledged. N is often referred to as the window size, and Go-Back-N is the window sliding protocol. The protocol sender has to respond to the following events from the receiver:

1. Invocation from above: when the application layer sends packets, we have to determine if the window is full. If it's full, we'd theoretically send the packets back to the application, but in reality we buffer the packets or lock it with a semaphore/mutex.
2. Receipt of an ACK: when an acknowledgement of sequence number n is received, we assume all packets up until and including n are received.
3. Timeout event: if a timeout event is received, we'll try to re-send all N unacknowledged packets in the window.

Selective-Repeat differs from Go-Back-N in the way that it may send and/or receive packets out of order which are buffered and pieced together in the right order at the receiver. Only the errored packets are being requested to be retransmitted instead of the entire window in Go-Back-N.

3.5 Connection-Oriented Transport: TCP

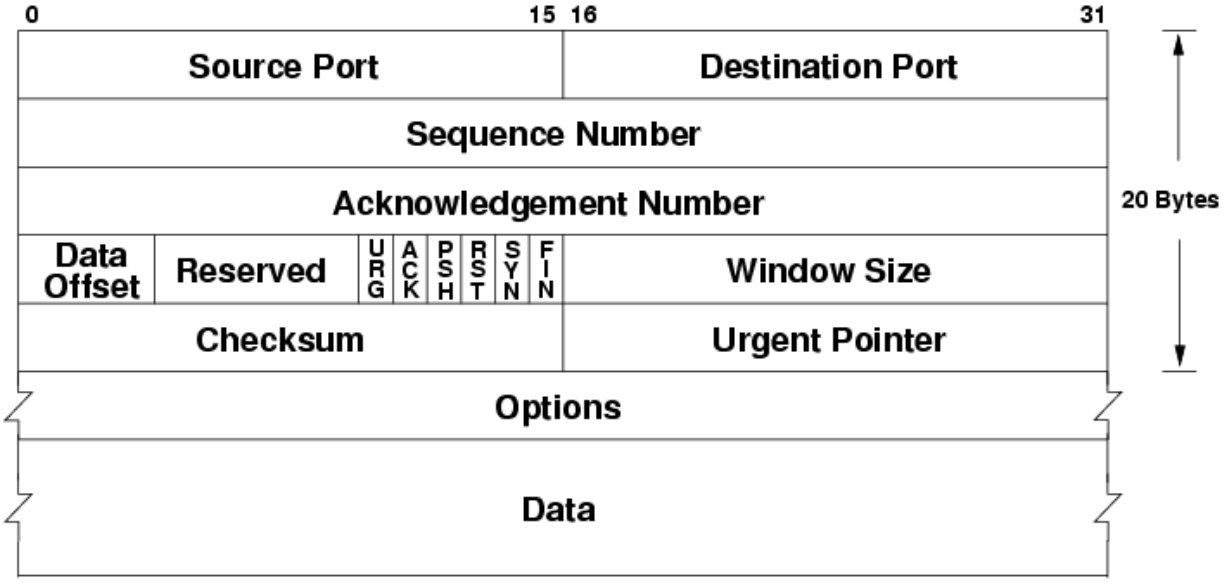
TCP is a transport layer protocol which contains ensures reliable data transfer. It is also:

- A full-duplex service, meaning communication can go both ways
- A point-to-point service, meaning communication only occurs between two end systems
- Connected by a three-way handshake. Client sends a SYN, expecting a SYN-ACK and confirming connection with an ACK.
- Buffers data in a send buffer. When either system wants to send data, it's sent to the TCP send buffer which will, depending on the data size partition it into multiple segments and send them to the other end. The maximum segment size (MSS) is typically determined by the length of the largest link-layer frame minus the size of the TCP header (40 bytes).

The TCP header is a 20-to-60 bytes long header appended to each TCP segment. It contains the following special fields:

1. Sequence number: if the SYN flag is set, this is the first sequence number, otherwise accumulated sequence number for the first byte of data
2. Acknowledgement number: if the ACK flag is set, this is the next expected sequence number the sender expects.

3. Data offset: byte offset into the header where the data can be expected (options may be 0-40 bytes large)
4. Reserved: unused data, reserved for future use
5. Flags (URG, ACK, PSH, RST, SYN, FIN): different flags for different TCP message types
6. Window size: amount of window size units the sender is currently willing to receive
7. Checksum: 16-bit checksum to error-check the TCP header
8. Urget pointer: if the URG flag is set, then this is a sequence number offset from the sequence number with the last urgent byte.
9. Options: variable length TCP options (see https://en.wikipedia.org/wiki/Transmission_Control_Protocol#TCP_segment_structure)



The reliable transmission over TCP is done with a single timer. The timer tracks the last packet which has not received an ACK yet. If the timer runs out, the packet is resent. The timer restarts every time an ACK of higher sequence number has been received.

TCP uses a modified Go-Back-N and Selective-Repeat strategy called Selective-Acknowledgement which allows TCP to acknowledge out-of-order segments selectively allowing us to only re-transmit what has not been acknowledged.

3.6 Principles of Congestion Control

Congestion control is a solution to the problem where the hosts transmit enough data quickly enough to overflow the network layer and its buffers, resulting in dropped packets (because network layer buffers are full). This means the transport layer might re-send packets that'll get dropped regardless and it'll cause huge delays.

We have two types of congestion control; end-to-end and network-assisted. In end-to-end congestion control, the network layer does not give any feedback to the transport layer about congestion control, so it's up to the transport layer to reduce load. In network-assisted congestion control the network layer might provide bits or flags to hint at congestion at a link which would allow the transport layer to reduce load.

3.7 TCP Congestion Control

TCP adjusts its transmission rate based on how congested it percieves the connection to be. It starts with a maximum transmission rate it will lower as necessary. If TCP notices a timeout or a triple ACK then the rate should be reduced. The rate of increase is based on how quick it receives ACKs from the other end. TCP follows these principles to determine when to change rate:

1. Lost segments imply congestion, therefore rate should be decreased
2. Acknowledged segments imply segments arrive, therefore rate may be increased
3. Triple acknowledgement is often caused by congestion or a timeout, therefore rate should be decreased.

TCP also uses the following algorithms to determine how much the rate should be changed:

1. Slow-start: start by transmitting at $rate = 1MSS/RTT$, doubling rate for each acknowledged segment. This leads to exponential growth. If there is a loss event, it re-starts at $rate = 1MSS/RTT$, while keeping track of the rate at the time of the loss event $lrate = \frac{rate}{2}$. Once the new rate reaches the old rate divided by two ($lrate$), TCP enters congestion avoidance mode.
2. Congestion avoidance: preserve the current rate $irate = rate$ and keep adding $rate+ = 1MSS/RTT$ to the rate until a new loss event occurs. Upon loss event, the rate is reduced to halv of the congestion avoidance mode added rate. In case of triple ACK, 3MSS is also removed to account for the triple ACK segments. TCP enters fast-recovery mode.

3. Fast recovery: in fast recovery, rate is increased by 1MSS for each duplicate ACK received after the missing segment which caused fast-recovery. Once the ACK for the missing segment is received, TCP enters congestion-avoidance mode again. If a timeout occurs, it jumps to slow-start again.

TCP is decently fair if all traffic on the link was TCP, but the algorithm isn’t necessarily fair if there’s other traffic such as UDP which doesn’t care for other transports.