Transport Layer

Transport layer protocols which take data from application layer services and make sure they reach the service or process the destination end.

Example

One thing that most of us are familiar with is an IP address. An IP address is associated with a machine or host and it's a way to make your data reach that specific host. They can be multiple processes running on that machine, such as Outlook mail, Firefox browser, FTP server, Secure Shell, and Remote Desktop. Now, the host got the data but how do you make sure that each process receives the data that was intended for it? This is where transport layer protocols come in. They receive information from lower layers and pass it up to the right user application or service. In this way, you can have multiple services and applications running on your machine, although you have only one IP address and network interface.

More Properties of Transport Layer

→ Data Division

→ Congestion Control - Analyze Network

→ Error Detection & Correction

→ Reliablity

So a short round up of these services that a transport layer protocol can provide. They can establish, manage, and terminate the connections. Handle the flow of data in case the receiving end is slower or has smaller buffers. Make sure all the packets are sent and are arrow free and avoid too much traffic on the network.

Transmission Control Protocol

A connection oriented transport layer protocol, it provides multiple services.

→ Numbered Packets – rx packets == tx packets

→ Reliabilty

→ Bi-directional communication possible

→ Flow Control – Congestion Control

For reliability TCP uses acknowledgements, each transmission is acknowledged by the receiver. If the packet is not acknowledged in a certain time frame, it is retransmitted. The receiver can also inform the sender of the specific packets it has not received. TCP delays can sometimes be very high for the upper layer application. As TCP waits for all the data to be received, puts the data in order and then forwards it to the upper layer. Reliability is the reason why some application layer protocols do not offer their

own reliable communication structures but rely on TCP instead.

Bi-directional Communication

TCP offers bidirectional communication, both parties can send and receive packets and

they can do it simultaneously.

Flow Control

TCP offers flow control, if the receiver buffers are small, TCP can reduce the rate of transmission in order to avoid overflowing the end system. If the receiver buffers are full, the receiver will start dropping packets, meaning the sender will not get any acknowledgements. This lack of acknowledgments acts as an indicator for the sender to either slow down or stop.

Slow Start

Network congestion is the reduction of throughput because of too much traffic on the network. TCP offers a mechanism called slow start,it starts transmitting lower number of packets and gradually increases until congestion occurs. An indication of congestion is the lack of acknowledgements. That means either the sender has reached the network's transmit capacity or the receiver can't handle anymore packets. In both cases it scales back, this is a method to prevent the host from causing immediate congestion.

Finally, it's up to TCP how it organizes and sends the data. TCP sends data in streams of bytes. It takes data from the application layer, let's say 1000 bytes, chops it into pieces. For example, ten pieces of 100 bytes, and sends the pieces down to the network orInternet layer for transmission. On the other hand it can take data in multiple bytes from the application layer. For example, ten pieces, each of 20 bytes, and send them as 100, 200 bytes packet. TCP is slow due to its reliability controls. Another reason for TCP being slow is the three-way handshake it uses, the initiator sends a synchronized packet. The receiver, upon reception of the synchronized packet, sends a synchronized acknowledge packet. And lastly, the connection initiator sends an acknowledge packet. This is why we call TCP a connection oriented protocol, it first establishes a proper connection. When we want to close a connection, then there is even more comprehensive four way handshake.

Transport Layer Security

TLS, is another transport layer protocol that secures communication between two parties. It's not a part of TCP but it relies on TCP, which is why we are mentioning it here. TCP might not be the best option for very time dependent applications. HTTP doesn't offer any built-in reliability mechanisms but, as it runs on top of TCP, TCP handles that for it. The file transfer protocol, FTP, and the chatting protocol, XMPP, also use TCP. XMPP is a chat protocol, so if you send a good night message or message with some expression of love to someone via XMPP. You don't want your message to get lost and never reach its destination. TCP ensures that the receiver gets your emotions, well maybe a little late, but she or he will get it. TCP can handle that networks or congestions. In unreliable network situations

its better to use TCP.

User Datagram Protocol

UDP is another transport layer protocol. Unlike TCP, it doesn't offer congestion control, it's not reliable and it doesn't send data in packet streams. It also doesn't assure in-order delivery or care if you receive duplicates. Also, there is no three way handshake which is so far the only good thing we have said about UDP. So why does it exist and why would anyone use it?

First of all, because of all the things it doesn't do, it's very simple. This makes it light and as it doesn't care about connections or handshakes, it's fast. It is stateless, meaning it doesn't maintain session or connection state. Each request is treated as a separate unique entity. This also makes it suitable for simple request response queries. TCP sends numbered packet streams and then reassembles them at the other end. UDP treats each packet as an individual complete message. It sends one complete message and receives one complete message with every sent receive request. To show you the difference between the **TCP and UDP formats, this is a 20-byte TCP header. And this is an 8-byte UDP header.** This shows the relative size differences between the two and the simplicity of UDP. UDP has its own use cases, it is used for real time streaming, where timing is more important than in order delivery. And where reliability is not that important because one or two lost frames do not affect the end user experience. Therefore, UDP is suitable for time sensitive applications such as Voice Over IP and multiplayer online games.

Some protocols that employ and benefit from UDP are Domain Name Service, DNS. Dynamic Host Configuration Protocol, DHCP, and Network Time Protocol, NTP. In each of these, you send a query and receive a response. One message is sent, one message is received. And there is DTLS or

DTLS

Datagram Transport Layer Security. It is not part of UDP but it's used in conjunction with UDP.