Computer Communications A3
Joakim Hjertholm (jhj010)

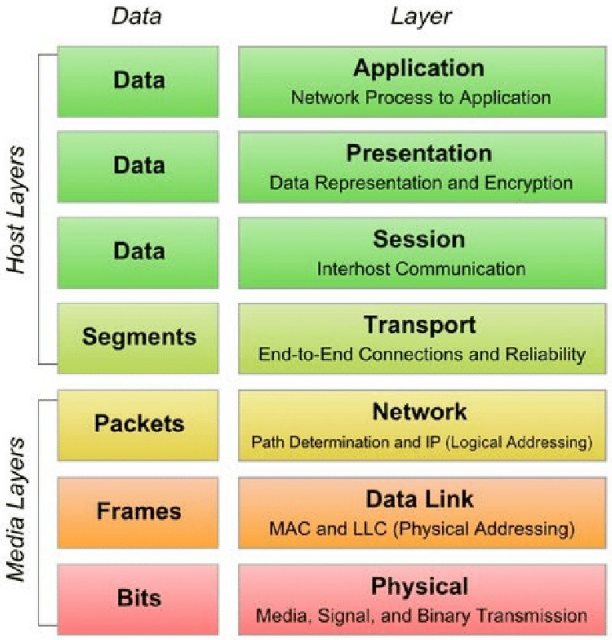
Introduction

In this project we are implementing the transport layer to ensure reliable data transfer between the sender and receiver, without losing data to corruption or packet loss and handling delay. We solve this by using GBN (Go-Back-N) and checking our data before sending and after receiving. We are given a simulation that runs the TCP layers, and we implement the transport layer.

Background

OSI-stack:

Is a reference model for how to conduct data communication. It's divided into 7 layers which make up the full process of communication between to devices.



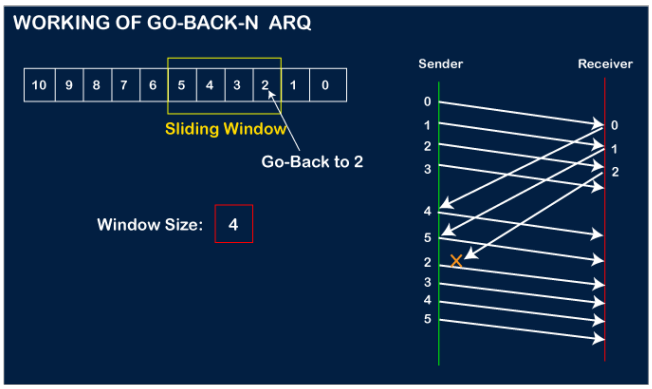
TCP:

Serve the same purpose that the OSI-stack does, but the only difference is that layer 5 and 6 are placed inside the application layer.

Transport layer:

Layer number four in the OSI-stack also known as transport layer, the place where data is verified and made into segments for the network layer to handle.

Design & Implementation



Our mission is to ensure reliable data transfer by implementing the transport layer. To ensure this we use GBN, and we are able to handle delays and packet loss. It's done by first having a window of packets you want to send, in our example it's four. While sending the packet we can encounter packet loss, and as we can see all the packets arrive but only two acknowledgments were received. Because we received acknowledgment for the two first packets our base pointer moves up

by two, and we resend all packets from base up to the end of our window. If we had lost the first acknowledgment but received the rest, then we know that all the packets were received because the receiver would not send an acknowledgment for packet two if it didn't receive packet one. This decision is made in the transport layer, and in our implementation the sender will send packets equal to our window size at the time and wait for an acknowledgment or the clock to timeout. If the clock times out before the packets arrives a callback function will run on another thread, this will resend all the packets already sent starting from the current base value.

When a packet is delayed the sender will keep on sending the current window, the callback function will most likely be called and depending on if the base value has incremented it will send back as many packets not acknowledged in the window.

Corruption only happens on the data in packets and not on acknowledgments in this simulation, and because of that we calculate a check sum on the data string of our packet. This sum will be stored in the packet for the receiver to calculate with the same function and see if the value matches. If the packet data was corrupted we send back a packet with the current sequence number minus one, this ensures that the sender does a callback and resend the packets.

The flow of our simulation starts with the application layer requesting the transport layer to send the next data segment, which the transport layer takes and together with giving it a sequence number and checksum value, it creates a packet with all these parameters stored within it. This packet is then sent to the network layer, here we simulate the chance for corruption, delay and packet loss. After network has sent the data with or without delay and corruption, it is received by the other TCP in their transport layer where the packet is evaluated for corruption and expected sequence number. If everything checks out the data is sent to application layer.

Evaluation

At the same time:

Finishing time (s)	Packet Loss %	Corruption %	Packet Delay %
3.9261	10	10	10
5.5470	20	20	20
9.0475	30	30	30
16.6635	40	40	40
16.4883	50	50	50

As we can see the implementation can run up to 50% on each one (did not test above 50%) without error, we can conclude that the implementation can handle multiple of complications without problems.

Individually

Percentage %	20	50	90
Packet Loss time	3.3728	6.2207	64.4615
Corruption time	2.1500	7.9184	41.6326
Packet Delay time	1.7264	2.9002	3.0648

Individually the implementation can handle up to 90% of each one without any errors, taking some time as the chance of getting a package and acknowledgment through is very low. The delay does not take as long time because the packets will eventually reach their destination and will not get lost or discarded. The corruption takes less time then packet loss because it only affects the data packets.

Conclusion

We have ensured that our transport layer implementation can deliver reliable data transfer between sender and receiver. With corruption, delay and packet loss the implementation doesn’t lose any data and receiver acquired all the expected data. This was solved by using GBN in the implementation.