Project 2 – Transport Layer Services

Trevor Robinson & Chakrya Ros

COP5518

8/6/2023

Background

This project is a simulation of network communication within the transport layer. It uses four separate classes, UDPSender, UDPNetwork, UDPReceiver, and UDPPacket.

The UDPPacket class is what does all the work in the background. This class takes the user input provided by the user and encapsulates the payload into a segment that includes the source IP (16 bytes), source port (6 bytes), destination IP (16 bytes), and destination port (6 bytes) (in that order). At the end of the segment, the last 10 bytes are dedicated to the payload (message) that includes header information and the part of the message that is being sent. Here, 4 bytes of header information (including ….) and 6 bytes of payload are included, bringing it to a total of 54 bytes per packet sent. If a message is 18 bytes long, there will have to be 3 packets sent for the receiver to receive the whole message.

The UDPSender is the client host that accepts user input and sends it as a part of the packet that was created by the UDPPacket class. It is sent to the network (IP and port provided in command-line arguments) and waits for an ack from the receiver before sending the next packet for the message. The sender will listen for specific messages from the receiver that informs it to re-send packets if necessary.

The UDPNetwork is the host that acts as a simulated network that listens for messages being sent back and forth between the sender and receiver. Here, the packet that is processed by UDPPacket and sent by UDPSender is received by the network. The UDPNetwork is also listening for responses sent by UDPReceiver. The network code then extracts the necessary information (destination IP and port) and sends it to the correct destination. The user may also choose to run the program with a percent chance for each of the following: packet loss, packet delay, and packet corruption. If a random number that is calculated by the code is within the percentage provided, it will either simulate loss, delay, or corruption to that specific packet. If delay occurs, the packet will be sent at a slight delay to the receiver. If loss occurs, the network drops the packet and the receiver never sees it, requiring the sender code to re-send that packet because of a timeout. If corruption occurs, the receiver code recognizes the corruption and sends a message back to the user to re-send that packet, where the sender complies and re-sends.

The UDPReceiver is the host that is listening for messages to receive from the sender. This is simulating a server, where it will receive packets, extract the necessary information, and display the original message that the sender sent to the console. Once it receives a packet, it will need to successfully receive it before sending an ack back to the sender, where it will allow the sender to send the next packet.

Experiment

In this experiment, we use four separate classes to simulate network communication. The UDPPacket class is not run but is instead compiled so that the sender and receiver can use it to make the correct packet formats given certain information. To run the program, the user needs to compile it using the Java Development Kit (JDK) and run it using the following arguments (IN THIS ORDER):

**“javac UDPPacket.java”**

**“javac UDPReceiver.java”**

**“javac UDPNetwork.java”**

**“javac UDPSender.java”**

**then**

**“java UDPReceiver <port>”**

**“java UDPNetwork <port> <lossPercent> <delayPercent> <errorPercent>”**

**“java UDPSender <port> <rcvHost> <rcvPort> <networkHost> <networkPort>”**

For testing, we used the port number “60400” for the receiver, “60310” for the network, and “60000” for the sender. We also used a variety of combinations for loss, delay, and corruption arguments for simulation and comparison, as seen in the following images and results section. We used localhost for both IP addresses, as we were able to simulate the communication using a single host. rcvPort would use “60400”, and networkPort would use “60310”. Here is an example of how it would be run after UDPPacket is compiled:

**“Java UDPReceiver 60400”**

**“Java UDPNetwork 60310 35 100 0”**

**“Java UDPSender 60000 localhost 60400 localhost 60310”**

Results

Discussion

This project

Conclusions

Overall, this experiment gave us an idea of how network communication works at the transport layer of a network.