Telecommunications Assignment 1

REPORT

Niall Hunt

16319138
10/11/2017

Contents

Introduction

Assignment

We were tasked with implementing a basic network structure. We had a client and a server given to us that would send packets to each other. Our implementation had to add a gateway that would act as a stepping stone between the client and the server. We also had to adjust the program so that we could have multiple clients talk to the server. Acknowledgements are sent back from the server to the client if the correct packet is received. This type if simple network could be an example of an IOT (Internet of Things) network where the clients are sensors sending information to the server. The server may do some analysis or calculations on the received data. This type if IOT network is likely to become more and more prevalent over the years as “smart homes” and “smart buildings” become more and more popular.

My Implementation

This is simplistic overview of how a network would be implemented in real life. Because of this some of the implementations won’t work if a lot of additional complexity is added to the network. My implementation is more of demonstration of how the network would work. In my program the clients can only send their packets to one server. They are also only capable of sending one packet per run as this is what is asked in the assignment. I extended the given classes and added one of my own to meet the given criteria.

Part 1: Multiple Clients

Explanation

The first problem I tackled was multiple clients. I thought this would be a straightforward way to learn more precisely how the client and server operated, as I worked. I first examined the given code to make sure I had a good understanding of inner workings of the program.

The problem with multiple clients is that each client must have its own port number. Port numbers are used to identify where to send the packets. My server has a port number of 60001 and my gateway has a port number of 60000. The clients must be able to send packets to the server (through the gateway) and receive the correct acknowledgement. The port number is used for this.

My Implementation

For my implementation I have the client numbers starting at 50000 (like the given code). I decided to change the ports for the server and gateway (explained below in part 2) so that the clients would have a band of ports that would be for client instances only. I decided that for this simple network (and most small smart home networks) that 10,000 clients would be more than adequate. Hence, I changed the default server port and gateway port to be 60000 and 60001 respectively.

When a client is created (ie the client program is run) the client constructor is called with the default port number. The constructor will attempt to create a socket with the given port number. If the number is already in use a

Part 1: Multiple Clients

java.net.BindException will be thrown. In this case one is added to the port number and the socket creation is attempted again. This repeats until a valid port number is found or until the maximum number of clients is reached.

The way I did this was by surrounding my try-catch statement, that creates the socket, with a while loop. The while loop finishes once the socket is created successfully. If the socket cannot be created the exception is caught and one is added to the port number.

A screenshot of the client constructor can be seen below:

```
Client(Terminal terminal, String dstHost, int dstPort, int srcPort)
{
    Boolean validPort = false;
    while(!validPort)
    {
        try {
            this.recieved = false;
            this.terminal= terminal;
            dstAddress= new InetSocketAddress(dstHost, DEFAULT_GATEWAY_PORT);
            socket= new DatagramSocket(srcPort);

            validPort = true;
            System.out.println("Source port: " + srcPort);
            this.dstPort = dstPort;
            this.srcPort = srcPort;
            this.sequenceNumber = 0;

            listener.go();
        }
        catch(java.net.BindException e)
        {
            srcPort++;
            if(srcPort >= dstPort)
            {
                System.out.println("Maximum number of clients reached");
                System.exit(1);
            }
        }
        catch(java.lang.Exception e) {e.printStackTrace();}
    }
}
```

Part 2: Gateway

Explanation

The second problem I took on was the gateway. Armed now with a deeper understanding of the workings of the client and server, I was able to quite quickly create a working gateway. However, a new problem was soon encountered. I was unable to send the acknowledgement back to the client from the gateway. I had to find a way around this.

From the course material I was aware that I was to use the header to send information about the packet along with the packet itself. I had to learn of ways to change integers to byte arrays and vice versa but this can quite easily be done with ByteBuffer.

Implementation

In my header I send along the final destination (ie the server port if sent from the client), the source port and the sequence number (explained below in part 3). The header is then used in the gateway to send the packet on to its destination.

I use ByteBuffer to change the destination port (an integer) into a byte array which I then can copy into the header byte array using System.arraycopy. in my gateway I extract the destination port from the header using System.arraycopy and ByteBuffer. I use the DatagramPacket method setPort()passed through the gateway and send the packet on to its final destination.

This works with the acknowledgements as well if the server creates the packets properly.

Part 2: Gateway

A screenshot of the header parsing and the sending of the packet can be seen below:

```
byte[] dstPortArray = new byte[PacketContent.BYTE_SIZE_OF_PORTS];
System.arraycopy(data, 0, dstPortArray, 0, PacketContent.BYTE_SIZE_OF_PORTS);
int dstPort = ByteBuffer.wrap(dstPortArray).getInt();
System.out.println("Destination port: " + dstPort);

packet.setPort(dstPort);
try
{
    socket.send(packet);
}
catch (IOException e) {e.printStackTrace();}
```

Part 3: Acknowledgement and Timeout

Explanation

In real world applications we need acknowledgements so that we are sure the server has received the correct packet, or even received a packet at all. We do this by sending an acknowledgement packet from the server to the client. We send a positive response if the correct packet was received and a negative one in the case where an incorrect packet is received.

To ensure the right packets are received and in the right order we use sequence numbers. The first packet sent to the server has a sequence number of 0, the second a sequence number of 1 and so on. If the server receives a packet with the wrong sequence number, it sends back a negative response.

We need timeouts as well in real world networks as connections can be interrupted and packets lost. If the client dose not receive the proper acknowledgement within a given timeframe it will resend the packet to the server.

Implementation

The way I implemented acknowledgements was to extract the sequence number from the header on the serer side. In this case as our clients only send one packet, the server checks to see if the sequence number is 0. If it is the incorrect number, it will send the negative response with the sequence number 0 as that is the packet that we are waiting for.

If it receives the correct packet it will send back the positive response with the sequence number 1 as this is the next expected packet. As our demonstration is only small the client accepts the acknowledgement and the program is complete as there are no more packets to transmit.

Part 3: Acknowledgement and Timeout

If the client does not receive the positive response within the given timeframe it will resend the packet and again wait for the response. As our program is simple and runs only over local host I implemented this simply using a received Boolean which is set to true on receipt of a packet and a while loop. I realise no that this naïve way of implementing timeout will not work as it enters an infinite loop of sending packets if the acknowledgement is not received. Unfortunately I had no time to fix it and should have used setSOTimeout() a Datagram socket method or by using Java's ScheduledExecutorService. These would be much better solutions to the timeout problem.

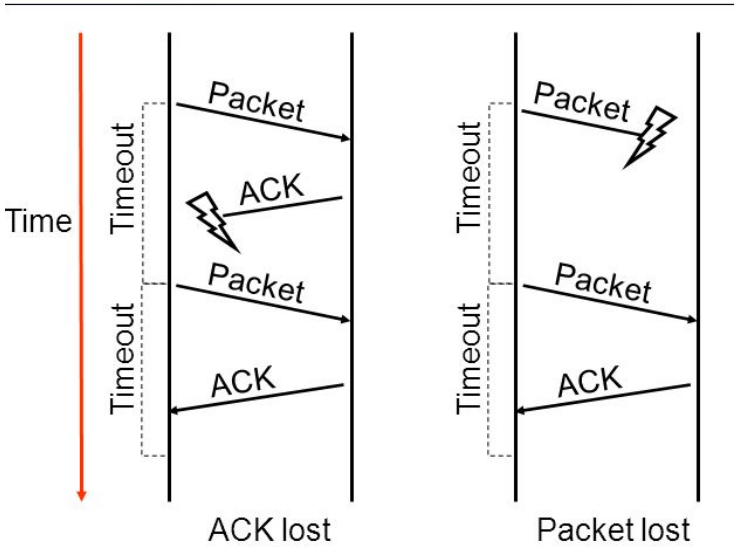

Image taken from a set of slides by Ally Bucknam found at: http://slideplayer.com/slide/4228411/

Conclusion

In doing this assignment I gained a much deeper understanding of how packets are put together and how they are processed by clients and servers. Looking back on my program now there are multiple things I would’ve changed. I am not happy with my rather ignorant and naïve timeout function, and I would like to have made the program better able to handle expansion and added complexity as I rather foolishly stuck too closely to the brief and the small network shown, instead of adding functionality that would work for other networks not just this specific one.