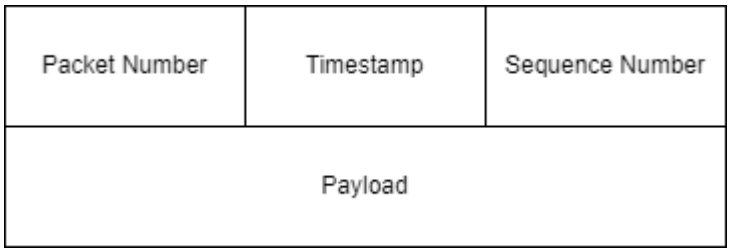
CS3102 Practical 1 Report

Overview

The objective of this practical was to write a C program that establishes a UDP socket connection to slurpe-4 and to take measurements of the path emulated by this binary file. It is split into two parts: R1, which involves writing the probe program to make measurements, and R2, which involves analyzing these measurements. I have managed to create a program which secures a UDP connection, and therefore I'm able to send and receive UDP packets which I have designed. However, I haven't been able to make any measurements, and as a result R2 hasn't been attempted. This report will solely focus on my implementation for R1. The port number used for my implementation is **22694.**

Design

The UDP packets take the form of a struct which has four fields: packet number, timestamp, sequence number and payload. The first three are part of the header where timestamp is the time (in milliseconds) for when the packet was created and sequence number is a unique identification number for each packet. The payload of the packet simply has the payload field.



Initially, my design involved a separate struct just for the header fields which is then referenced in the packet struct. This is a better design since it separates the header and the payload of the packet. However, due to there being several segmentation issues that occurred during development, I have decided to use a much simpler implementation with only one struct and no pointers stored in it.

Implementation

The program is split into two files: slurpe-probe.c and UdpSocket.c. The first program file is for creating, sending, and receiving the UDP packets. I took the code from my_protocol.c, timer.c and udp-chat.c and modified it for this practical. I also worked with ao85 where we

discussed approaches to the practical. The second program file is for establishing the socket connection between slurpe-probe.c and slurpe-4.

The main function contains the main event driven loop because the requirement was to have a packet being sent every second and using an event driven loop for this made sense to only send UDP packets only when necessary. Here the SIGIO and SIGALARM signals are set up through their respective functions before entering the main loop so that their handler functions can be initialized. Then the event driven loop is entered which waits for a signal. The handler function for SIGALARM involves sending a UDP packet. This is because SIGALARM generates signals for timed intervals, so it made sense to attach it to that handler. On the other hand, the handler function for SIGIO involves receiving the packets relayed back due to SIGIO being the signal used when a file descriptor is ready to perform input or output. The packets are created in a separate function which makes sense because packets are supposed to be created every second. This becomes easier by simply having a function call to create the packets in the SIGALARM handler function. For the same reason a separate function for generating the timestamp was used which is called from the same handler function.

To send the data, the struct must be converted to a byte stream. This was done using memcpy() since this was a really convenient and easy method to use. Before this, it was important to use the function htons() on the struct fields so that the data stream that is generated is canonical. The same process is repeated when receiving the byte stream, except the variables are switched around since the conversion is the other way.

Testing

When it comes to testing, there wasn't any procedure that I used to conduct my tests. I simply tested my code as I was coding. Any errors that occurred was dealt with by manually going through the relevant parts of the code and trying to figure out what the issue is and why it occurred. From the tests that I have done, the code seems to run fine without any major issues.

Output from slurpe-probe.c:

```
sb409@pc7-097-l:/cs/home/sb409/Documents/Third_Year/CS3102/P1/code $ make packets
gcc -c slurpe-probe.c
gcc -c UdpSocket.c
gcc -o packets slurpe-probe.o UdpSocket.o
sb409@pc7-097-l:/cs/home/sb409/Documents/Third_Year/CS3102/P1/code $ ./packets pc7-011-l
29100
Packet Number: 1
Timestamp: 29100
Sequence Number: 10001
Payload: 012345678910111213141516171819202122232425262728293031
<-013917113172012345678910111213141516171819202122232425262728293031 (data stream to the network)
Packet Number: 1
Timestamp: 29100
Sequence Number: 10001
Payload:0123456789101112131415161718192021222324252627282930310097000
->013917113172012345678910111213141516171819202122232425262728293031 (data stream from the network)
30100
Packet Number: 2
Timestamp: 30100
Sequence Number: 10002
Payload: 012345678910111213141516171819202122232425262728293031
<-023918117148012345678910111213141516171819202122232425262728293031 (data stream to the network)
31100
Packet Number: 3
Timestamp: 31100
Sequence Number: 10003
Payload: 012345678910111213141516171819202122232425262728293031
<-033919121124012345678910111213141516171819202122232425262728293031 (data stream to the network)
32100
Packet Number: 4
Timestamp: 32100
Sequence Number: 10004
Payload: 012345678910111213141516171819202122232425262728293031
<-043920125100012345678910111213141516171819202122232425262728293031 (data stream to the network)
Packet Number: 4
Timestamp: 32100
Sequence Number: 10004
Payload:0123456789101112131415161718192021222324252627282930311071158111100
->043920125100012345678910111213141516171819202122232425262728293031 (data stream from the network)
33100
Packet Number: 5
Timestamp: 33100
Sequence Number: 10005
Payload: 012345678910111213141516171819202122232425262728293031
<-053921129760123456789101112131415161718192021222324252627282930313211211100
Packet Number: 5
Timestamp: 33100
Sequence Number: 10005
Payload:0123456789101112131415161718192021222324252627282930313211211100
->053921129760123456789101112131415161718192021222324252627282930313211211100
34100
Packet Number: 6
Timestamp: 34100
Sequence Number: 10006
Payload: 012345678910111213141516171819202122232425262728293031
<-063922133520123456789101112131415161718192021222324252627282930313211211100
Packet Number: 6
Timestamp: 34100
```

Output from slurpe-4:

```
1676649132.461052 20230217-15:52:12.461052 138.251.29.107 : 38 bytes
1676649133.461005 20230217-15:52:13.461005 138.251.29.107 : 38 bytes -- packet dropped
1676649134.461010 20230217-15:52:14.461010 138.251.29.107 : 38 bytes -- packet dropped
1676649135.460987 20230217-15:52:15.460987 138.251.29.107 : 38 bytes
1676649136.461000 20230217-15:52:16.461000 138.251.29.107 : 38 bytes
1676649137.461010 20230217-15:52:17.461010 138.251.29.107 : 38 bytes
1676649138.460904 20230217-15:52:18.460904 138.251.29.107 : 38 bytes
1676649139.461022 20230217-15:52:19.461022 138.251.29.107 : 38 bytes
1676649140.461023 20230217-15:52:20.461023 138.251.29.107 : 38 bytes
1676649141.460982 20230217-15:52:21.460982 138.251.29.107 : 38 bytes
```

Evaluation

The code seems to work well when it comes to communicating between the two end-systems, however the way the packets were handled could have been better. In particular, the formatting of the packeting could have been much better. For example, I could have had some form of overlay for the packet struct. Also, the struct fields could have been cleared out first before adding any values to eliminate any unwanted and unexpected values. Initially, my program had all of this implemented, but I had to remove the code for all of that because I was getting a few errors which I couldn't debug. Another thing I found hard to successfully implement was making sure that the data stream and the struct fields match each other. There were issues with this where they weren't matching due to the data stream being jumbled up. However, I was able to resolve that. Additionally, my implementation doesn't allow for proper measurements of the emulated path to be taken. The design of the struct packet could have been improved to allow for this. One example is by adding flexibility in terms of the amount of bytes that can be sent as part of the payload. This would allow measurements for the data rate to be made as mentioned in the lectures by sending a set of smaller packets followed by larger packets.

Conclusion

Overall, my implementation provides a start for measurements to be made as it is able to establish a connection and communicate properly with the other end-system by sending and receiving packets in the form of a byte stream. However, it is limited in terms of the design of the packets, making it hard to make proper measurements to assess the emulated path.