**Contents**

Objective:...................................................................................................................................... 3

Design ............................................................................................................................................3

Functions:.......................................................................................................................................3

Ns file………………………………………………………………………………………………………………………………………..6

Observations ..................................................................................................................................7

Conclusion......................................................................................................................................9

References .....................................................................................................................................9

**Objective:**

Part 1 .Build a client-server application.Server sends a header less packet to client in segments

of 50 bytes delimited by a predetermined symbol. Client will subsequently write packet

data to a file.

Part 2.Build a network of nodes. Send packets randomly to router who will then read the

packet and send the packet to a certain destination based upon its content.

**Design:**

Our implementation was based on following components:

Python socket library to bind to network IPs on specific ports and transmit information

Python struct library to format headers along w/ relevant information that will enable

successful routing

User defined retransmission functions used to handle loss of packets

**Functions:**

Part1

1) Following functions in our Python code relate to the **networking**process:

Make\_Frame():-This function packs the payload together to our header. Our header consists of destination mac address and packet sequence number for retransmissions.

Extract\_Frame():-This function is used to extract payload and sequence number out of the received packet.

Resend\_List():-This function takes care of the retransmission part. It sends the packets which are lost to the receiver when the receiver sends a list of lost packets.

The comments showing phase II is where the retransmission starts.

Part2

Following are the functions in our code:

Run\_Router():- Receives packets to be processed from nodes sendingthrough raw

sockets

Run\_Server():- Constructs the packet and sends it on a specific interface using mac

Address

ByteToHex/HexToByte():- Use to encode and decode packet header sequence numbers

for faster throughput

Content\_Based\_Route():- Router function that parses packet content and determines

where to send the packet tobased on color information.

2) Following deals with the outlay of the program

**Pseudo code of the each of the functions with the program flow follows:**

Part1

Algorithm for Sending/Receiving data through raw sockets over Network

---------------------------------------------------------------------

- The server reads 50 delimited bytes at a time from a file.

- The data in file for server is delimited by newline character.

- The server sends 50 bytes packets to the client, who will then write it to a file

- The server-client interaction works as follows:

(a) Server will read the file contents and get its information

(b) Server will tell the receiver how many frames to expect in the first packet

(c) Server continuously frames packets and sends them to client

(d) Client stores the number of packets it is supposed to end up with while processing packets it is continuously receiving.

(e) In the off-chance that client is not receiving packets it keeps a list that is retransmitted to the server via the resend\_list function for the server to resend.

(f) The client reads the packet it is given. Client extracts the payload from the frame and writes the data to a file resembling the original.

Part2

Algorithm for Content-based router

----------------------------------

- The client and server are similar to what we implemented in part 1.

- The data in file for server is delimited by newline character.

- Each 50 byte in data file contains either strings RED or YELLOW or GREEN or BLUE anywhere.

- The server sends 50 bytes packets to the router, which is running our custom content-based routing program.

- The router works as follows:

• We created a configuration file for router (also used by clients), which consists of following data:

Format of each line is:

<node\_name> :<mac\_address> , <rtr\_interface> , <node\_interface>

(i) Node name, could be either RED, or GREEN or YELLOW or BLUE or RTR (router)

(ii) MAC Address of the node

(iii) Interface on router that will be used to send the data

(iv) Interface for clients to receive packets from

(b) The router will fill its custom routing table with the information from the configuration file.

(c) We have used a multi level dictionary of dictionary (dict<dict>) and represents information as follows:

(i) The first dictionary contains the node name, i.e. either color code or RTR

(ii) The second dictionary is a <key, value> pair for each node and keys could be one of the following:

'listen\_if': value is the listening node interface

'dest\_mac': value is MAC address for a color node

'local\_if': value is one of the RTR interfaces connected to a color node

(d) The router binds a listening port to each node and starts receiving 50 bytes packets.

(e) The router runs a Regular Expression search on received 50 bytes packet to identify either color code.

(f) The router gets the MAC Address and Node interface from its routing table(configuration file) based on the color code it received in a 50 bytes packet and forwards the received packet to that node.

- The client receives a packet from router and writes it to a file.

**Configuration File for Part 2:**

**Router.co**

red:00 15 17 1c d7 64,eth2,eth2

blue:00 15 17 1c d9 72,eth4,eth2

green:00 15 17 1e 05 8e,eth3,eth2

yellow:00 15 17 1e 03 02,eth5,eth2

rtr:00 15 17 1c d8 66,eth2,eth2

Above is the Configuration file for the router acting as a routing table. The program will process the code, read colors, attach the corresponding MAC addresses along with the device interface.

**NS File for Part 1:**

sourcetb\_compat.tcl

set ns [new Simulator]

set n1 [$ns node]

set n2 [$ns node]

setlinkA [$ns duplex-link $n1 $n2 1000Mb 0ms DropTail]

$ns rtproto Static

$ns run

**NS File for Part 2:**

sourcetb\_compat.tcl

set ns [new Simulator]

set r1 [$ns node]

setnred [$ns node]

setngreen [$ns node]

setnblue [$ns node]

setnyellow [$ns node]

setlinkA [$ns duplex-link $nred $r1 1000Mb 0ms DropTail]

setlinkB [$ns duplex-link $ngreen $r1 1000Mb 0ms DropTail]

setlinkC [$ns duplex-link $nblue $r1 1000Mb 0ms DropTail]

setlinkD [$ns duplex-link $nyellow $r1 1000Mb 0ms DropTail]

$ns rtproto Static

$ns run

**Observations**

Part1

D) Use iperf to measure and compute the effective throughput for UDP packets with 50 byte payload – turn in the description and the transcripts of your experiments to measure the throughput of only the data portion of the packet

Ans) To utilize IPerf for this experiment, we used following switches:

At Server side:

• –s Running server

• –u Use UDP protocol

• –l <byte> Specified Datagram size, we used 78 bytes (28 bytes header + 50 bytes payload)

At Client side:

• –c <server> Running client connected to specified server

• –u Use UDP protocol

• –l <byte> Specified Datagram size, we used 78 byes (28 bytes header + 50 bytes payload)

• –b <bits/sec> Bandwidth to use, defaults to approximately 1Mbits/sec

We increased bandwidth from default till we started to see packet loss at some higher bandwidth. So, starting from 1Mbits/sec and recording at regular intervals, we got the following statistics:

(all experiments were run for 10 seconds and were run atleast 3 times to minimize errors in collecting statistics)

**Bandwidth set at client**

**Throughput at server**

Default

1.05Mbits/sec

1 Mbits/sec

1.00Mbits/sec

10 Mbits/sec

10.1Mbits/sec

25Mbits/sec

26.0Mbits/sec

50Mbits/sec

52.0Mbits/sec

75Mbits/sec

78.0Mbits/sec

100Mbits/sec

104.0Mbits/sec

110Mbits/sec

104.0Mbits/sec

125Mbits/sec

104.0Mbits/sec

150Mbits/sec

104.0Mbits/sec

175Mbits/sec

105.0Mbits/sec

200Mbits/sec

105.0Mbits/sec

205Mbits/sec

105.0Mbits/sec

208Mbits/sec

105.0Mbits/sec

But the UDP throughput is considering even the Headers in the packet. So we need to remove the header length first and then calculate the throughput.

We observed, UDP throughput is 105Mbps. For a 50byte payload with header length=54Bytes,total packet length is going to be 104 bytes. So we can calculate accordingly the UDP throughput as 51Mbps for a packet length of 50 bytes.

For our code, we are getting throughput as approximately **104Mbps.**

We initially observed that there was a high loss of packets. So we decided to add a microsecond delay at the sender side so as to slow down the transmission mechanism. This would have affected the throughput but eventually we removed this microsecond delay because we started doing retransmissions to make sure the file was sent over reliably. We observed that the printf statements at the receiver helped as a delay between the packets. We have kept a start timer at the receiver side from when the first packet is received and then the stop timer after the last packet is received. We have used a multi-level dictionary to store the packet sequence number and the payload. We found that the files of size smaller than 500 KB could be sent without any losses and gave a higher throughput. Files larger than that entered the phase-II for retransmission.

The approach we took to decrease the bytes in header was simple. Since the sequence numbers got huger we decided to send the sequence numbers in ‘hex’ format which reduced the header length by 2 bytes and gave an increased throughput.

**Conclusion**

Part1

Overall, our code performs better than iperf. The MD5 for both the files at the sender and receiver were the same. The throughput obtained was approx **104Mbps.**

Part2

Our router was able to identify the packet it received from a sender and based on the color code, was able to identify the destination receiver. There werenumerous ways we could show it. The one we utilized though was to verify that the file we wrote to had all the relevant lines including the words that were sent inside the packet. We were also able to successfully maximize up our throughput using hex numbers for packet sequencing and retransmission for file fidelity.

We have submitted the code –Part 1 files: rc\_part1.py (client code),rs\_part1.py(server code)

Part2 files: rc\_part2.py (client code),rs\_part2.py(server code),router.py(router code),config.txt(configuration file)

**References**

http://docs.python.org/library/socket.html

http://docs.python.org/howto/regex.html

http://docs.python.org/tutorial/inputoutput.html