210017213

CS3102 P1 – Charactering a Network Path

Introduction

For this practical we were tasked with making measurements by sending probe packets on through a given emulated path slurpe-5. This required creating two separate programs, the first that handled the sending and receiving of the probe packets producing output data that the second program could analyse and make measurements from. The measurements that are required to be analysed are the loss, delay and the end-to-end data rate that are produced by the emulated path.

Unix user ID: 23220

Design

When designing the structure of the probe packet initially I decided to have the sequence number that increments for each packet, as well as the timestamp of when the packet was sent. The sequence number is necessary within the packet as it acts as the ID of the packet, allowing the ability to know what packets have been received and what packets have not been received or 'dropped'. The timestamp is stored within the packet as well and is used to calculate the Round-Trip Time (RTT) when/if the packet is received.

However, after some consideration I decided to also add the size of the payload to the header of the packet as it wouldn't impact on the overall size of the packet being sent due to the padding that would be added if this wasn't included. Having access to the size of the payload for this practical provides no use as the aim is to take measurements about the emulated network path, however in practice it would be useful to know how much data is being sent or received to be able to handle varying packet size with more ease. The diagram design of my packet is as follows:

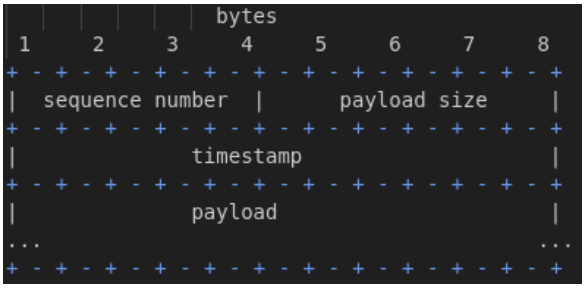

Figure 1 Packet design

Within the probe program I also added the ability to vary the size of the payload of the packets that are sent, the size can be selected through a command line argument. Having this ability for varying the packet payload size made it very easy to produce the data required for taking measurements on different sizes of packets. I decided to only allow one size of packet to be sent for each

experiment because this would not skew the results and would provide a more consistent means for measuring the data.

After implementing my solution, and too late to make any alterations I thought about using signals for when receiving the packet. Which would take the place of the non-blocking pselect() which I have used in my probe program. This is much more inefficient compared to using signals as many iterations are completed before receiving the packet or timing out. The probe program utilises signals when sending each packet, so that the packet was sent every second regardless of if a packet is not received and is dropped by the emulated network path.

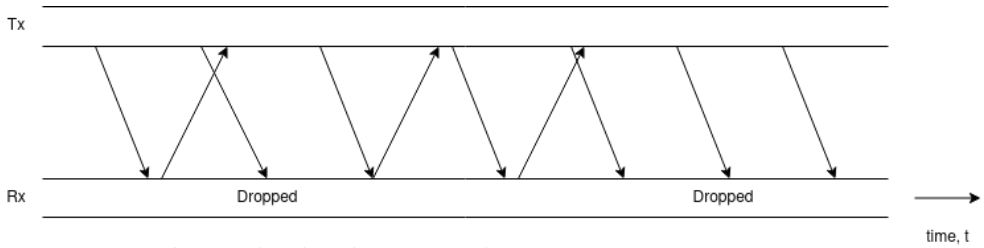

Figure 2 Timeline diagram showing process of probe program

Above, the probe program process is shown over time. Packets are sent every second no matter whether the packet is received or not. Additionally, the packets could be received after the next packet is sent but, in this case in practise this does not occur as the RTT value is never anywhere close to a second. However, I illustrated this in the diagram above because in theory it is possible to happen.

Methodology

To conduct the experiments to be able to make the measurements, I followed these steps:

- Started the slurpe-5 program with fullpe and the address of where the probe program is running. For example: ./slurpe-5 fullpe <probe_program_addr> <probe_program_addr>
- Use the 'script <path>' command on the terminal where the probe program is being executed. This records everything printed on the terminal, the path of the output can also be specified (all my output files can be seen within the /data directory).
- Run the probe program from the <probe_program_addr>, while also specifying the address where slurpe-5 is being run from and the payload size for that experiment. For example: ./slurpe-probe <target> <payload_size>
- The received packets information on is being printed on the terminal and recorded as the probe programs output.
- After the probe program has finished (sent all packets), then use 'exit' so the data outputted by the probe program is recorded.
- To view all the graphs and calculations made, run the Juypter notebook plot_data.ipynb which uses the outputs from the probe program.

This the process to be able to conduct the experiments. This can also be found in the README.md file provided.

I conducted two experiments, the first with the 'small' packets which only contain the packet header (no payload) so each packet contains 16 bytes, and the second with the 'large' packets where each packet is in total 1400 bytes.

I decided to use Python matplotlib and numpy (these can be seen in /data/requirements.txt) to process the probe program output and then produce graphs from that data. I simply read in the output file, processed the data and made the measurements from that data. I chose these tools as they provide an easy way to produce graphs and perform calculations.

The measurements I was focused on were the delay, loss, and the end-to-end data rate.

The delay I measured by using the RTT value of each packet, this was calculated when the packet was received to assure the most accurate RTT value is gathered.

The loss I measured by comparing the number of packets I received with the number of packets that were sent, this allowed me to determine how many packets weren't received or 'dropped'.

The end-to-end data rate I calculated by performing the following:

$$b = (large\ packet\ size - small\ packet\ size) * 8$$

$$T_{d\ path} = \frac{\left(median\left(T_{d\ large\ packets}\right) - median\left(T_{d\ small\ packets}\right)\right)}{2}$$

$$T_x = \frac{b}{10^9}$$

$$T_p = T_{d\ path} - T_x$$

$$r = \frac{b}{T_p}$$

Figure 3 Calculation for End-to-end data rate

Where r is the end-to-end data rate.

Results

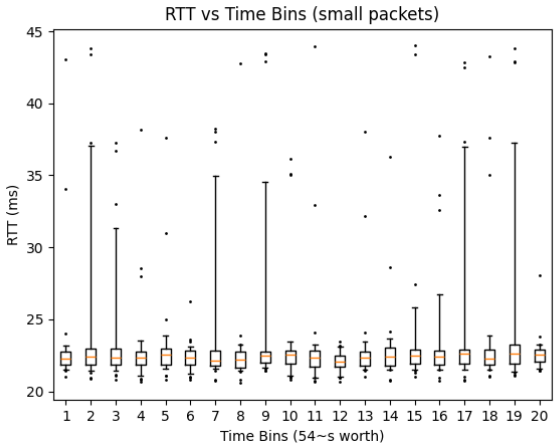

Figure 4 RTT vs Time Bins of small packets

In the graph above the RTT is shown in milliseconds against time bins. Each of the time bins above equate to 1 minute of time containing approximately 54 ~ packets due to the average loss time of around 11%. The spread of the box plots is between the 5th and 95th percentile which shows the most relevant spread of values. The values were produced when sending small packets.

From Figure 3 a basic observation would be that on average the RTT value is very consistent over time with some outliers.

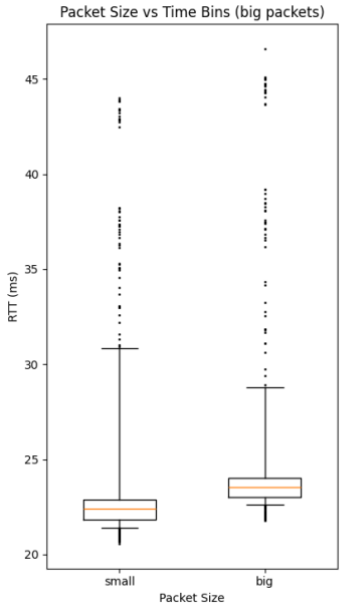

Figure 5 Spread of RTT values with different packet sizes

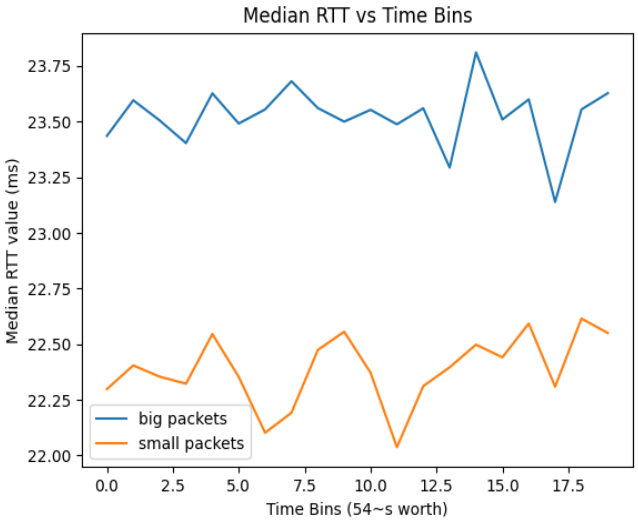

Figure 6 Median RTT values for different packet sizes

Both graphs above show similar things but in slightly different ways. Figure 5 looks at the overall spread of all the RTT values when sending packets of different sizes. Whereas Figure 6 looks at the median of the RTT values over the time bins values for small and big packets.

From Figure 5 it can be seen the big packets on average have a higher RTT values than the small packets do. This is consistent with Figure 6 where it can be seen the median RTT values from each bin is higher in the big packets compared to the small packets.

I decided to use the median RTT value instead of the mean as using the mean value as the mean would consider the outliers so therefore wouldn't be a true representation of the 'average' RTT value that is produced, so median is a better option to find the average values.

From Figure 5 the spread of the box plots is between the 5th and 95th percentile which shows the most relevant spread of values.

	Small Packets	Big Packets
Delay	11.21 ms	11.76 ms
Loss	11.5%	11.08%
End-to-end data rate	20.38 Mbps	

Figure 7 Table showing Delay, Loss and End-to-end data rate

In Figure 7 it can be seen the median delay for bigger packets is larger than the median delay for smaller packets, as to be expected. However, the loss rate of packets is larger when sending smaller packets compared to when big packets were sent, which is not intuitive to what might be expected. The end-to-end data rate is around the value that was to be expected.

Analysis and Conclusion

From Figure 4 it can be seen the RTT is very consistent and on average sits the same value, with some outliers included as well. This shows that the path emulated by slurpe-5 overall produces a very consistent outlet for sending and receiving packets within a similar amount of time. There are more outliers seen at certain points over the course of the experiment running, however from the results no conclusive information can be drawn from this about when and why there are these spikes. This is very similar to a network path outside of the emulated path slurpe-5, where random spikes can be experienced in delay which can be due to many external factors.

When it comes to the differences experienced over the emulated network path when sending small and big packets, from both Figure 5 and Figure 6 it can be seen the RTT values overall are higher when sending the bigger packets compared to when sending the smaller packets. This illustrates the amount of time it takes to transmit the packet increases as the size of the packet increases, as to be expected as the packet contains more data. This can be clearly seen in Figure 6 and Figure 7 as the median RTT value, and delay value is consistently higher when sending larger packets compared to the smaller packets.

To calculate delay:

$$delay = \frac{median(rtt\ values)}{2}$$

Figure 8 Delay calculation

To calculate the delay, the RTT value is halved because the RTT time represents the amount of time it takes for the packet to be sent and received, whereas the delay is the amount of time for one of these processes to be performed.

To produce the RTT boxplots, I created time bins which hold the RTT values for a set time-period of 60 seconds. This allowed me to use this range of values to produce the boxplots in matplotlib. The time bins have varying sizes as packets are dropped, I decided that using the time bins for a set time-period lead to a more relevant graph being produced as it more accurately represents the networks behaviour through that time, rather than having time bins of s set size as this would inaccurately represent different scales of time.

In Figure 7 it can be observed the percentage loss when sending the small packets is higher compared to when sending the big packets. This goes against what would be expected. The expected result would be for the small packets to have a higher (or the same) loss rate comparatively to the big packets. The reasoning why this occurred could be due to a sensitivity to smaller packets from the emulated network path, but this is highly unlikely. The reason for this result could be a coincidence, as the data is taken from 20-minute experiments and the values are relatively close being only 0.42~% of a difference. If the experiments were to run for much longer, the loss rate would be likely to be lower for the smaller packets and higher for the larger packets.

To calculate loss:

$$loss(\%) = \left(\frac{packets\ sent - packets\ received}{packets\ sent}\right) * 100$$

Figure 9 Loss calculation

From Figure 7 it would suggest there is a bottleneck within the emulated network path because the data rate is only 20~Mbps and should be roughly 900~Mbps because it is travelling over ethernet. This is not allowing the full potential of the ethernet connection to be harnessed, leading to a relatively low end-to-end data rate being produced.

My process for calculating the end-to-end date rate can be seen in Figure 3.

A critique of my experiments conducted is that I should have run the experiments for a longer period of time, as this could have eradicated getting results such as that the loss rate is higher for smaller packets which is out with the expectation. Another way to make the results produced more consistent, would be to re-run the experiment multiple times as I have only conducted the experiment for big and small packets once.

One conclusion that can be drawn is that the more data a packet carries, the longer amount of time it will take to be sent and received by the emulated network path.

Data file list

In the /data directory the following things can be found:

- /images directory which includes all the graphs produced from the Jupyter Notebook ploy_data.ipynb
- output_big.txt : this is the file which includes the whole output when running the experiment with big packets
- output_small.txt : this is the file which includes the whole output when running the experiment with small packets
- plot_data.ipynb : the notebook which is used to produce the graphs and calculate some of the values e.g. loss
- process_data.py : this is a python file which performs some data processing, for example it handles reading in the output file from the probe program.