AJAYKRISHNAN JAYAGOPAL
CS14B033

Go-Back-N Protocol

7th March 2017

OBSERVATIONS

The Go-Back-N protocol implementation was tested using 2 machines: one at the DCF (with IP 10.6.15.144) and my local machine. The Sender program was run on my machine while the Receiver was run on the DCF machine.

Due to a lot of variability in network performance, there were several points of time when the network performance suddenly became very poor, and hence the RTTs (Round Trip Time) of some packets was deviating from the average RTT by almost 6x-7x times. The average RTT was normally in the range 3-4 milliseconds but occasionally, some packets had an RTT of 16-25 ms even without any packets being dropped. Since the timeout for packet acknowledgement was set to only **2*Avg.RTT**, this resulted in the timeout firing for some of the packets. Since the maximum number of packet retransmissions was capped at 5 beyond which the Sender was terminated, this resulted in frequent terminations of the Sender since the ACK for certain packets was invariably arriving after **2*Avg.RTT**. On setting the timeout to a fixed value of 100 ms, this problem was solved. Also, changing the constant factor from 2 to around 8-9 also solved this issue.

Since the RANDOM_DROP_PROB was very small $(10^{-7} - 10^{-4})$, and MAX_PACKETS was relatively small (500-5000), packets were being dropped very rarely. One way to solve this would have been to increase MAX_PACKETS to a very large value, but then the total time of execution was very high.

It was also observed that the RTT depended on whether debugging mode was enabled on the Receiver. The overhead of printing to the terminal is high and hence when debug mode was ON, the Receiver was taking more time to send back the ACK then when debug mode was OFF.

I instead tested with slightly higher values of RANDOM_DROP_PROB $(10^{-3} - 10^{-1})$ and observed that as the RANDOM_DROP_PROB was increased, the Retransmission ratio and the average RTT also increased. I also tested with different packet lengths, from 128 to 1024 and observed that the Retransmission ratio was unaffected whereas the average RTT increased.

The results are summarized as follows.

RESULTS

For the following tables, **MAX_PACKETS = 5000, BUFFER_SIZE = 10, PACKET_GEN_RATE = 10, WINDOW_SIZE = 3 and debugging mode enabled.**

For **RANDOM_DROP_PROB** = 10^{-7}

Sr.No	PACKET_LENGTH	Retransmission Ratio	Avg. RTT (in ms)
1.	128	1.0	3.7013
2.	1024	1.0	4.47123

For **RANDOM_DROP_PROB** = 10^{-4}

Sr.No	PACKET_LENGTH	Retransmission Ratio	Avg. RTT (in ms)
1.	128	1.0925	3.8069
2.	1024	1.004	4.47128

The following are some additional measurements made for **PACKET_LENGTH** = **512** and all the other parameters being the same as before:

Sr.No	RANDOM_DROP_PROB	Retransmission Ratio	Avg. RTT (in ms)
1.	0.005	1.168	3.7705
2.	0.01	1.224	3.9812
3.	0.05	1.294	4.2893
4.	0.1	1.312	4.3465

The average RTT is not only dependent on RANDOM_DROP_PROB but also on the network performance.

README.txt

Instructions for compilation:

```
make
```

Instructions for running:

1. Sender:

```
java Sender -s localhost -p 12345 -l 128 -r 10 -n 500 -w 3 -b 10 -d
```

2. Receiver:

```
java Receiver -p 12345 -l 128 -n 500 -e 0.01 -d
```

Instructions for running typescript (In the typescripts folder):

1. Sender:

```
scriptreplay --timing=sender_timing.log sender_script.log
```

2. Receiver:

```
scriptreplay --timing=receiver_timing.log
receiver_script.log
```

Comments

The assignment was very interesting and was helpful in revising concepts regarding multithreading, synchronization and locking.