CSE 589 FALL 2014

PROGRAMMING ASSIGNMENT 2

IMPLEMENTING RELIABLE TRANSPORT PROTOCOLS

ANALYSIS REPORT

PRATIK PRAMOD CHAVAN

UB PERSON #: 5013-4114

E-MAIL ID: [pchavan2@buffalo.edu](mailto:pchavan2@buffalo.edu)

CONTACT NO: (716)-400-4194

INDEX

Name Page

1. Academic Integrity Statement 3
2. Timeout Scheme 3
3. Multiple Timers for SR 4
4. Experiment 1 6
5. Experiment 2 11
6. ACADEMIC INTEGRITY STATEMENT

“I have read and understood the course academic integrity policy located under this link:

[http://www.cse.buffalo.edu/faculty/dimitrio/courses/cse4589\_f14/index.html#integrity”](http://www.cse.buffalo.edu/faculty/dimitrio/courses/cse4589_f14/index.html#integrity).

1. TIMEOUT SCHEME

I have chosen timeouts 15.0, 15.0 and 20.0 for ABT, GBN and SR respectively.

The average time between two messages (lambda) is 50.0 is all the analysis experiments. I wanted my timeouts to be as low as possible. This is because, if timeout is closer to lambda value, in case of loss or corruption, many times before a timer interrupt occurs, new packets will arrive and they will have to be buffered (In case of GBN or SR) and will have to dropped in case of ABT. So, performance will reduce clearly in case of ABT. Also, in GBN and SR, more the number of buffered packets are, more are the retransmissions and so the performance reduces significantly.

I didn't choose value less than 15.0 as it takes 5 logical units to travel from A to B, RTT is 10.0 (to receive ACK). Besides other factors can increase it. So, timeout should not occur when Non-corrupt expected packet is on the way.

There is no specific reason for choosing 20.0 for SR. I found the results to be better with 20.0 when I tried different values within the nearby range and hence chose it.

3. MULTIPLE TIMERS IN SELECTIVE REPEAT USING SINGLE HARDWARE TIMER

PROBLEM:

In Selective Repeat protocol, each packet has its own timer so that unlike Go Back N, on packet loss or corruption event, packets in the entire window need not be sent. Instead SR receiver selectively acknowledges the packets it receives in its window. The sender retransmits packets in the window on timer interrupt for which it has not received the acknowledgement. In starter code of simulator, we are only provided with a single hardware timer. The problem at hand thus is that, how to use this single hardware timer to emulate the task of one individual timer per packet.

SOLUTION:

The approach I have taken to tackle this problem is to basically use a queue data structure.

One important thing to notice before I describe the approach I have taken is that, the only thing of our interest in tackling this problem is that we should be able to correctly tell when a timer interrupt occurs it has occurred for which packet. Also, we should be able to correctly manage manual starting and stopping of timer (while sending packet and when an ACK is received) for respective packets. So, the actual time (even though logical) is not of our interest **if we can correctly predict and manage the order in which timers are starting and stopping.**

Following is in detail the approach I have taken to manage this.

* tm[1000] is an array of int which basically stores the sequence number of the packets whose timers are ON.
* timer\_count is an int variable which stores the current number of active timers
* void tm\_start(int sqn) is a function which takes a sequence number as a parameter. This function then puts this parameter sqn and places it at tm[timer\_count]. It then increments timer\_count. This is like enqueue operation on queue data structure.
* void tm\_stop(int sqn) is a function which takes a sequence number as a parameter. This function then removes it from tm. Push all other elements after the position at which sqn is to one less position. For example: tm[x]=tm[x+1]. x starts with position at which sqn was present. This is like dequeue operation on queue data structure with the modification that we are removing a specific element at a random position and not the one at front of the queue.
* When a packet is added tm\_start() is called and sequence number of that packet is sent as a parameter. It is then pushed on to the tm[] array. This basically indicates that most recent packet whose timer has started is sqn. Similarly, the timer which is started least recently is the first element in tm[].
* When a packet is added tm\_stop() is called and sequence number of that packet is sent as a parameter. It is then removed from the tm[] array.
* When a timer interrupt occurs, the first element from the tm is removed tm\_stop(0) is called. This is because amongst all the running timers, if timer interrupt occurs it will be for the packet whose timer was started the earliest. And as per my structuring of the flow, tm[0] will always be the packet whose timer is started earliest. Now this tm[0] is removed and then put back at tm[timer\_count] using tm\_start(). Then starttimer() is called. This is because ACK has not been received for packet whose timer has expired. It has to be reset. Entry is removed from tm[] only when ACK is received for the packet with that sequence number.

1. EXPERIMENT 1
2. Window Size : 10

Loss: 0.1

Average Throughputs:

ABT: 0.0162865

GBN: 0.0139405

SR: 0.0159839

1. Window Size : 10

Loss: 0.2

Average Throughputs:

ABT: 0.0151223

GBN: 0.0158502

SR: 0.0110106

1. Window Size : 10

Loss: 0.4

Average Throughputs:

ABT: 0.0117378

GBN: 0.0140513

SR: 0.0004245

1. Window Size : 10

Loss: 0.6

Average Throughputs:

ABT: 0.0070407

GBN: 0.013586

SR: 0.0001314

1. Window Size : 10

Loss: 0.8

Average Throughputs:

ABT: 0.0022823

GBN: 0.0089616

SR: 0.0000402

1. Window Size : 50

Loss: 0.1

Average Throughputs:

ABT: 0.0162865

GBN: 0.0139405

SR: 0.017717

1. Window Size : 50

Loss: 0.2

Average Throughputs:

ABT: 0.0151223

GBN: 0.0158041

SR: 0.0166963

1. Window Size : 50

Loss: 0.4

Average Throughputs:

ABT: 0.0117378

GBN: 0.0124644

SR: 0.0154764

1. Window Size : 50

Loss: 0.6

Average Throughputs:

ABT: 0.0070407

GBN: 0.0040311

SR: 0.0023838

1. Window Size : 50

Loss: 0.8

Average Throughputs:

ABT: 0.0022823

GBN: 0.0021662

SR: 0.000097

COMMENTS:

Performance of ABT reduces which is expected as it drops packets and is stop and wait protocol so throughput is less.

GBN and SR perform quite well up to loss probability 0.4. But performance reduces significantly beyond that.

Average throughput for SR and GBN in some of the cases is lower than expected because of failure of not being able to handle some of the specific scenarios generated by some specific seed values.

Performance is shown via graphs for SR and GBN. For ABT no graph is included as it doesn’t have window size and only parameter that affects performance is increasing loss probability.

GRAPHS FOR EXPERIMENT 1:

GBN – Window size 10 (blue) vs 50 (red) – As per increase in loss

SR – Window size 10 (blue) vs 50 (red)– As per increase in loss

1. EXPERIMENT 2
2. Window Size : 10

Loss: 0.2

Average Throughputs:

ABT: 0.0151223

GBN: 0.0158052

SR: 0.0110106

1. Window Size : 50

Loss: 0.2

Average Throughputs:

ABT: 0.0151223

GBN: 0.0158041

SR: 0.0166963

1. Window Size : 100

Loss: 0.2

Average Throughputs:

ABT: 0.0151223

GBN: 0.0158041

SR: 0.0163096

1. Window Size : 200

Loss: 0.2

Average Throughputs:

ABT: 0.0151223

GBN: 0.0158041

SR: 0.0150995

1. Window Size : 500

Loss: 0.2

Average Throughputs:

ABT: 0.0151223

GBN: 0.0158041

SR: 0.0119902

1. Window Size : 10

Loss: 0.5

Average Throughputs:

ABT: 0.0093954

GBN: 0.011883

SR: 0.0001847

1. Window Size : 50

Loss: 0.5

Average Throughputs:

ABT: 0.0093954

GBN: 0.0070047

SR: 0.0086801

1. Window Size : 100

Loss: 0.5

Average Throughputs:

ABT: 0.0093954

GBN: 0.0072209

SR: 0.0083837

1. Window Size : 200

Loss: 0.5

Average Throughputs:

ABT: 0.0093954

GBN: 0.0072609

SR: 0.0079744

1. Window Size : 500

Loss: 0.5

Average Throughputs:

ABT: 0.0093954

GBN: 0.0072348

SR: 0.006657

1. Window Size : 10

Loss: 0.8

Average Throughputs:

ABT: 0.0022823

GBN: 0.089616

SR: 0.0000402

1. Window Size : 50

Loss: 0.8

Average Throughputs:

ABT: 0.0022823

GBN: 0.0021662

SR: 0.000097

1. Window Size : 100

Loss: 0.8

Average Throughputs:

ABT: 0.0022823

GBN: 0.002182

SR: 0.005512

1. Window Size : 200

Loss: 0.8

Average Throughputs:

ABT: 0.0022823

GBN: 0.0018296

SR: 0.0017385

1. Window Size : 500

Loss: 0.8

Average Throughputs:

ABT: 0.0022823

GBN: 0.0018964

SR: 0.0031594

COMMENTS:

Performance of SR and GBN remains almost same when window sizes are different but loss is the same.

But as loss goes on increasing the performance decreases significantly.

As for ABT, since it is a stop and wait protocol and does not have a window its performance is observed to be deteriorating with increase in loss probability as expected.

Performance is shown via graphs for SR and GBN. For ABT no graph is included as it doesn’t have window size and only parameter that affects performance is increasing loss probability.

GRAPHS FOR EXPERIMENT 2:

GBN: For Loss probabilities 0.2 (blue) vs 0.5 (red) vs 0.8 (green) for different window sizes in increasing order

SR: For Loss probabilities 0.2 (blue) vs 0.5 (red) vs 0.8 (green) for different window sizes in increasing order