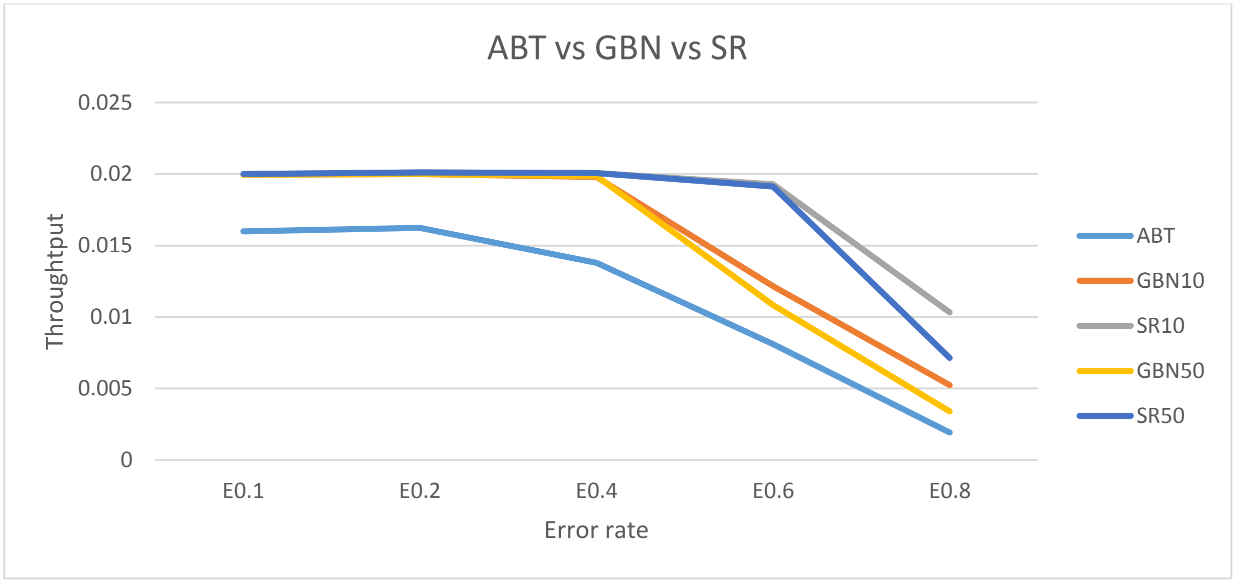
# CSE 589 – PA2

Deepak Madappa

Readings taken from euston.cse.buffalo.edu

## Experiment 1

The experiment was conducted with corruption rate of 0.2 for all protocols. The number next to protocol size is the window size. For example GBN10 means GBN with a window size of 10. Error rate is the probability of the underlying layers losing the packets. The experiments were run with 10 different seeds and average of the 10 runs are plotted below. The tabular result of each experiment can be found in the appendix.



**Analysis:**

The numbers above appears to be as I expected. ABT has clearly the lowest throughput which is expected because ABT makes the minimal use of the medium without keeping it idle, i.e., it keeps one packet it the medium at any time, the channel will have to have idle time in order to perform any worse. And the performance of ABT almost linearly with increase in error rate.

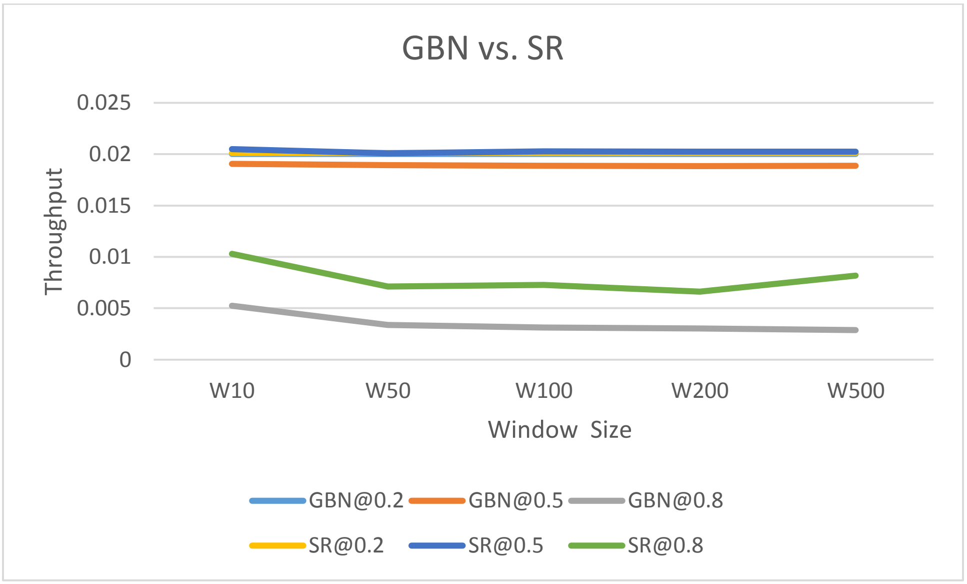
Also in both the window sizes GBN performs worse than SR, the likely reason for this is that the behavior of GBN forces the sender to resend all the packets even for a single loss, this is hogging the bandwidth resending unnecessary packets thus lowering throughput. We can also notice that GBN starts performing worse (after 0.4) vs. SR (after 0.6). This is because there is a point in the network medium after which sending more packets aggressively starts effecting throughput adversely and I've noticed that GBN hits this point earlier than SR does, likely because GBN retransmits unnecessarily.

Honestly, I expected GBN to perform lot worse, but there is a saving grace with GBN, that is the cumulative acknowledgement, whose advantage is not obvious immediately. Consider the case of SR vs. GBN, in a medium with loss rate of 0.8 the probability of one successful packet delivery and acknowledgement for SR is 0.2 * 0.2 = 0.04. The important thing to notice here is the loss acknowledgement is equally likely to affect delivery as the data packet itself. However, in case of GBN acknowledgement is more likely to be delivered than the data packet. For example, if the sender sends 25 (which may include different packets or same packet retransmitted, doesn't matter, trying to make a point about cumulative ACKs) packets using GBN then probably 5 will make it to receiver (0.8 loss rate). When receiver sends the ACKs for these five packets 1 will make it (again 0.8 loss). So for 25 packets sent the receiver got 1 ACK, which leads to outcome 1/25 = 0.04. Which looks the same as SR but because of cumulative ACK, the one ACK that made it back will likely acknowledge more than 1 packet. So in conclusion ACKs in GBN are more powerful than in SR, this will become more evident below in GBN section.

Here I've used different timeout techniques for GBN and SR but the relative positions of SR and GBN were still the same when I did use the same technique. Also we can notice that different widow sizes have different throughput, we'll discuss more about this in the next experiment.

## Experiment 2

This experiment was conducted under same conditions as experiment 1. The notation GBN@0.2 means GBN protocol with 0.2 loss probability. I've omitted include ABT below since it does not have a window and the effect of error rate on ABT has already been discussed in the above experiment.



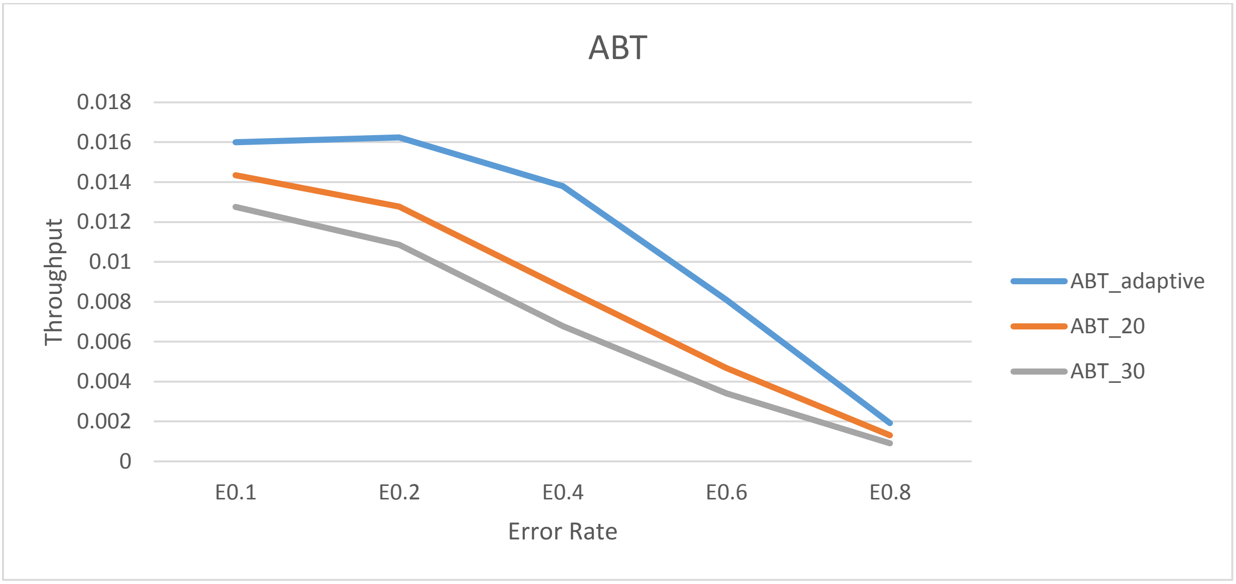
**Analysis:**

Both algorithms perform well enough till error rate of 0.5 so there is nothing much to discuss there. For the error rate of 0.8 it is straight forward why GBN has dropped with increase in window size. Longer window size in GBN implies more packets to be resent when there is a loss. But I expected throughput to increase with the increase of window size in SR which actually was true when I tried with error rate 0.8 and corruption 0. I think SR hit some kind of threshold after error rate 0.8 which made window size less consequential. My intuition was that since the window size and error are both large, the protocol is stuck resending later packets in window and not resending earlier packets in the window which means later packets can't be delivered to layer5 by receiver even though they were received successfully (the probability of a successful delivery is pretty low, 0.16 * 0.16). I tried implementing a timeout scheme where earlier packets in widow get more weightage to retransmit (lower timeout) than later packets. I believe I could not tune the weights well enough for this to work as I did not see any improvement in performance.

**Properties of medium:** Before I start the analysis of individual protocols I'd like to point out certain observations I made about the medium. Firstly the loss rate is fixed and the actions of my protocol has no effect on the loss rate, this behavior is sort of a deviation from real network where our actions have a small effect on loss rate. Second thing is that backing off isn't really helpful. It does matter which packet you send, but not sending any packet doesn't really help.

**ABT:**

The following experiments were done with same settings as previous experiments. ABT_20 implies ABT with a constant timeout of 20, and same for 30. ABT_adaptive uses my adaptive timeout algorithm.



Analysis:

The best performance I got from ABT is when I used an adaptive timer, I used the following formula.

EstimatedRTT = (1 – ALPAH) • EstimatedRTT + ALPHA • SampleRTT

DevRTT = (1 – BETA) • DevRTT + BETA • | SampleRTT – EstimatedRTT |

TimeoutInterval = EstimatedRTT + MULTIPLIER • DevRTT

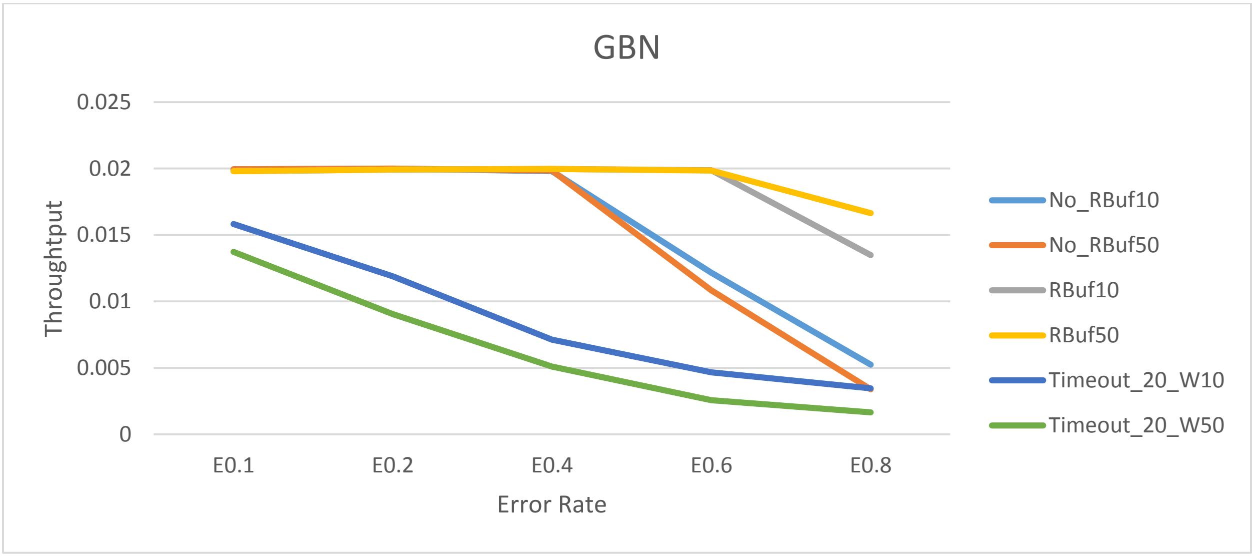
I tried with different values for ALPHA, BETA and MULTIPLIER and optimized to,

ALPHA = 0.3, BETA = 0.125, MULTIPLIER = 1;

I’m sure these are not the optimal values but this is the best I could find, it should be possible to find optimal values for these using regression. As expected adaptive timeout outperforms constant timeout. SampleRTT is the RTT for the packets which got the ACK in first send, will not consider resent packets.

**GBN:**

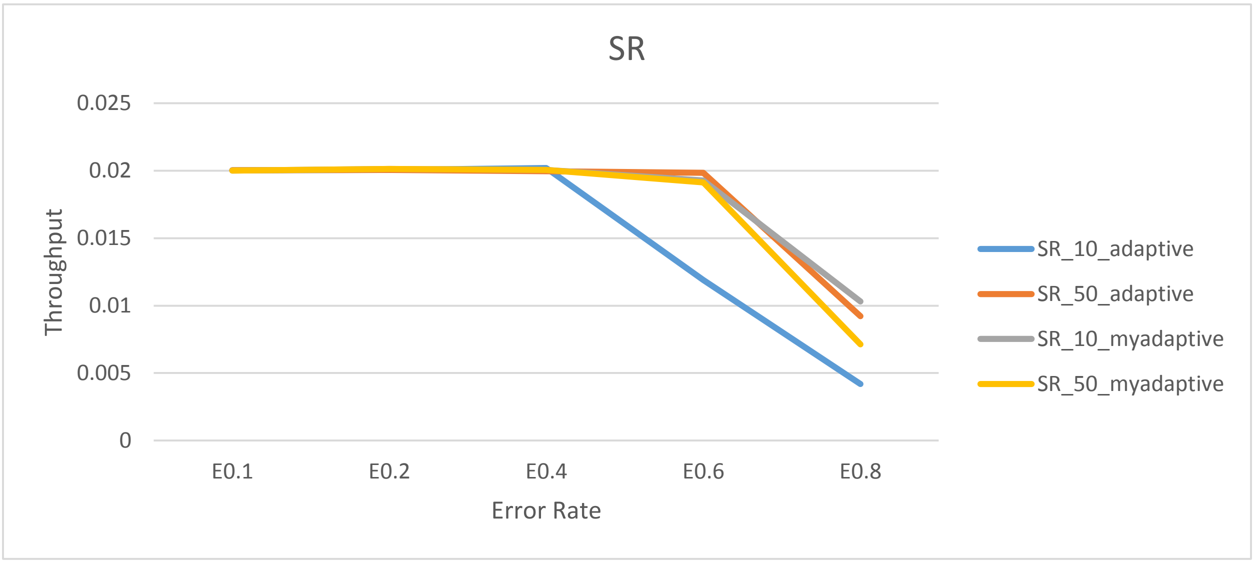
In the following experiment I implemented receiver buffer, No_RBuf means Receiver Buffer has not been used RBuf is when it is used, the numbers next are window sizes. In both cases adaptive timer was used. ALPHA=0.4, MULTIPLIER = 0, so did not add the devRTT part. It helped to be responsive the recent timeouts and keep a high ALPHA value. And TIMEOUT_20_W10 implies constant timeout of 20 and window size of 10.



As evident using a receiver window boosts throughput significantly, GBN with receiver buffer outperformed even SR in my experiments. The reason for this cumulative acknowledgement as I mentioned before. In case of SR both data packet and ACK have to beat the odds, but in case of GBN only data packets have to beat the odds, ACKs get multiple changes to make it because of cumulative ACKs.

**SR:**

In the chart below SR_10_adaptive implies SR with window size 10 and adaptive timeout which used the previously mentioned timeout algorithm with ALPHA = 0.125, BETA=0.25 and MULTIPLIER = 4. Myadaptive implies my own adaptive algorithm was used which I'll be describing below.



**Analysis:**

As I noticed previous RTT is not the only indication we can used to set our new timeout. Also as error rate increased I started getting fewer samples which could be used since packets which get ACK the first time are very few. But there are other indicators such as duplicate ACKs. I implemented a timeout scheme where every time an ACK arrives it changes the timeout as follows.

- Every ACK which falls within window and is the first ACK for that sequence decreases the timeout by a value, in my case 0.14.
- Every ACK which falls within window and is duplicate increases the timeout by 0.14.
- Every ACK falling outside of the window will increase the timeout by 1.

So essentially, when right thing happens (i.e., first ACK within window) we try to push the limits and reduce the timeout. When wrong thing happens (duplicate ACK), algorithm becomes more conservative and increases timeout. When an out of window packet arrives, which means that things are worse in that case we increase timeout drastically (by 1). These values again are probably not optimal and can be optimized with regression.

Also, it can also be noted that in cases where I use adaptive algorithm give in text book throughput increased with window size as expected.

APPENDIX

Experiment 1: corruption rate = 0.2

| | E0.1 | E0.2 | E0.4 | E0.6 | E0.8 |
|---|---|---|---|---|---|
| ABT | 0.015993 | 0.016239 | 0.013801 | 0.008093 | 0.001922 |
| GBN10 | 0.019959 | 0.020007 | 0.019803 | 0.012162 | 0.005245 |
| SR10 | 0.020006 | 0.020117 | 0.020062 | 0.019282 | 0.010314 |
| GBN50 | 0.019959 | 0.020007 | 0.019857 | 0.010832 | 0.003391 |
| SR50 | 0.020006 | 0.020117 | 0.020065 | 0.019126 | 0.007134 |

Experiment2: corruption rate = 0.2

| | W10 | W50 | W100 | W200 | W500 |
|---|---|---|---|---|---|
| GBN@0.2 | 0.020007 | 0.020007 | 0.020007 | 0.020007 | 0.020007 |
| GBN@0.5 | 0.019059 | 0.01892 | 0.018873 | 0.018851 | 0.018861 |
| GBN@0.8 | 0.005245 | 0.003391 | 0.003125 | 0.003029 | 0.002884 |
| SR@0.2 | 0.020117 | 0.020117 | 0.020117 | 0.020117 | 0.020117 |
| SR@0.5 | 0.020505 | 0.020079 | 0.020269 | 0.02025 | 0.02025 |
| SR@0.8 | 0.010314 | 0.007134 | 0.007286 | 0.006628 | 0.008165 |

ABT: corruption rate = 0.2

| | E0.1 | E0.2 | E0.4 | E0.6 | E0.8 |
|---|---|---|---|---|---|
| ABT_adaptive | 0.015993 | 0.016239 | 0.013801 | 0.008093 | 0.001922 |
| ABT_20 | 0.014342 | 0.012778 | 0.008687 | 0.004673 | 0.001314 |
| ABT_30 | 0.012757 | 0.010868 | 0.006788 | 0.003414 | 0.000912 |

GBN: corruption rate = 0.2

| | E0.1 | E0.2 | E0.4 | E0.6 | E0.8 |
|---|---|---|---|---|---|
| No_RBuf10 | 0.019959 | 0.020007 | 0.019803 | 0.012162 | 0.005245 |
| No_RBuf50 | 0.019959 | 0.020007 | 0.019857 | 0.010832 | 0.003391 |
| RBuf10 | 0.019812 | 0.01993 | 0.019983 | 0.019875 | 0.013501 |
| RBuf50 | 0.019812 | 0.01993 | 0.019983 | 0.019841 | 0.016654 |
| Timeout_20_W10 | 0.015834 | 0.011891 | 0.007122 | 0.004673 | 0.003479 |
| Timeout_20_W50 | 0.013723 | 0.009049 | 0.005096 | 0.002586 | 0.00165 |

SR: corruption rate = 0.2

| | E0.1 | E0.2 | E0.4 | E0.6 | E0.8 |
|---|---|---|---|---|---|
| SR_10_adaptive | 0.020024 | 0.020048 | 0.020197 | 0.011906 | 0.004189 |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| SR_50_adaptive | 0.020024 | 0.020048 | 0.01997 | 0.019834 | 0.009211 |
| SR_10_myadaptive | 0.020006 | 0.020117 | 0.020062 | 0.019282 | 0.010314 |
| SR_50_myadaptive | 0.020006 | 0.020117 | 0.020065 | 0.019126 | 0.007134 |