Reliable Transport Protocols

Jose Luis Ayerdis Espinoza
joseluis@buffalo.edu / #50247563

Contents

Academic Integrity

I have read and understood the course academic integrity policy.

Timeout scheme

For both Alternative bit protocol and Go-back-end when running the classic experiments 1 and experiment 2 the selected time interval was fixed on `constant.h` as the `AVG_TRAVEL_TIME=20` this number was drawn by considering an avg time of 5 units to travel from one end to the other, so on avg should take around 10 time unit, the extra offset comes from considering a 10 unit time of processing on each end side, and truth to be told multiple values were tested from ranges of 12 to 30 before setting the constant value of 20.

Further explanation on the different scenarios tested can be found in a section below *time experiment* with an extensive comparison on the effect of time-related factors: fix time delays, adaptative timer and tweak in the average rate at which packets are sent in each protocol.

Experiments

An overview of the what each experiment parameters is describe below:

Experiment	Message	Corruption	Loss	Window	Total
Loss probability on window sizes	1000	0.2	0.1-0.8	10, 50	30
Window sizes on loss probability	1000	0.2	0.2-0.8	10-500	45
Corruption and packet loss relationship	500	0-0.8	0-0.8	50	363
Time experiments[4]	500	0	0.2	10, 50	24

Incremental loss probability on window sizes (Experiment #1)

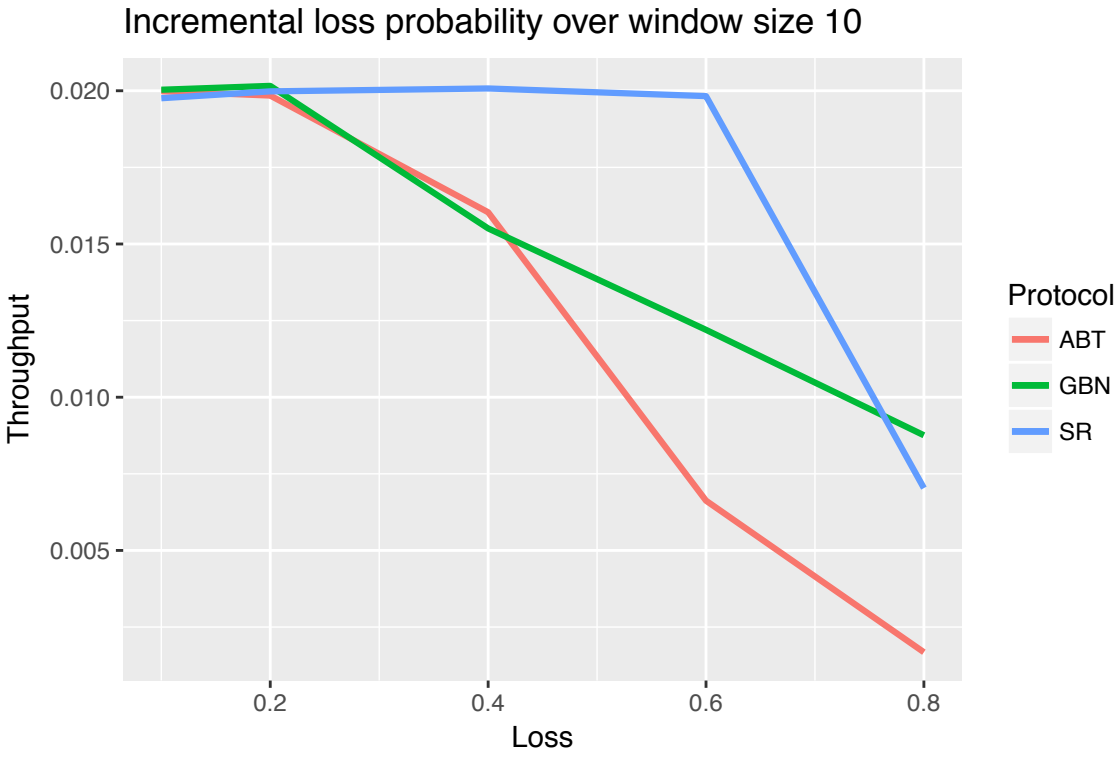
As suggested the test conducted with the following parameters

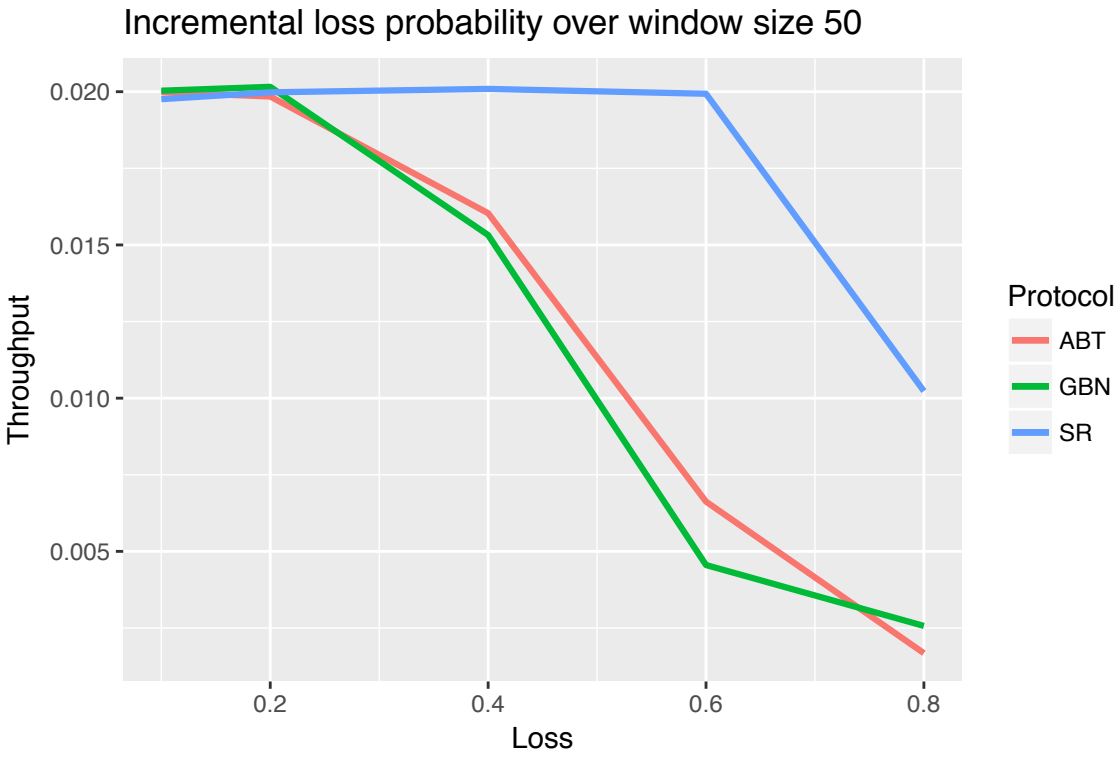
Protocol	Message	Corruption	Loss	Window
Alternative Bit protocol	1000	0.2	0.1-0.8	10, 50
Go back end	1000	0.2	0.1-0.8	10, 50
Selective repeat	1000	0.2	0.1-0.8	10, 50

The results of the experiments show as expected **a decreasing trend in throughput as the loss level increases**. Each protocol support it differently, selective repeat does significantly better job at keeping a steady throughput on increment loss levels compared to ABT and GBN, its double buffering and transmission of packet selectively at both the receiver and sender prevent it from overflowing the network with unnecessary packet like go back end and make the transmission more fluid unlike ABT that needs to wait for each packet to arrived.

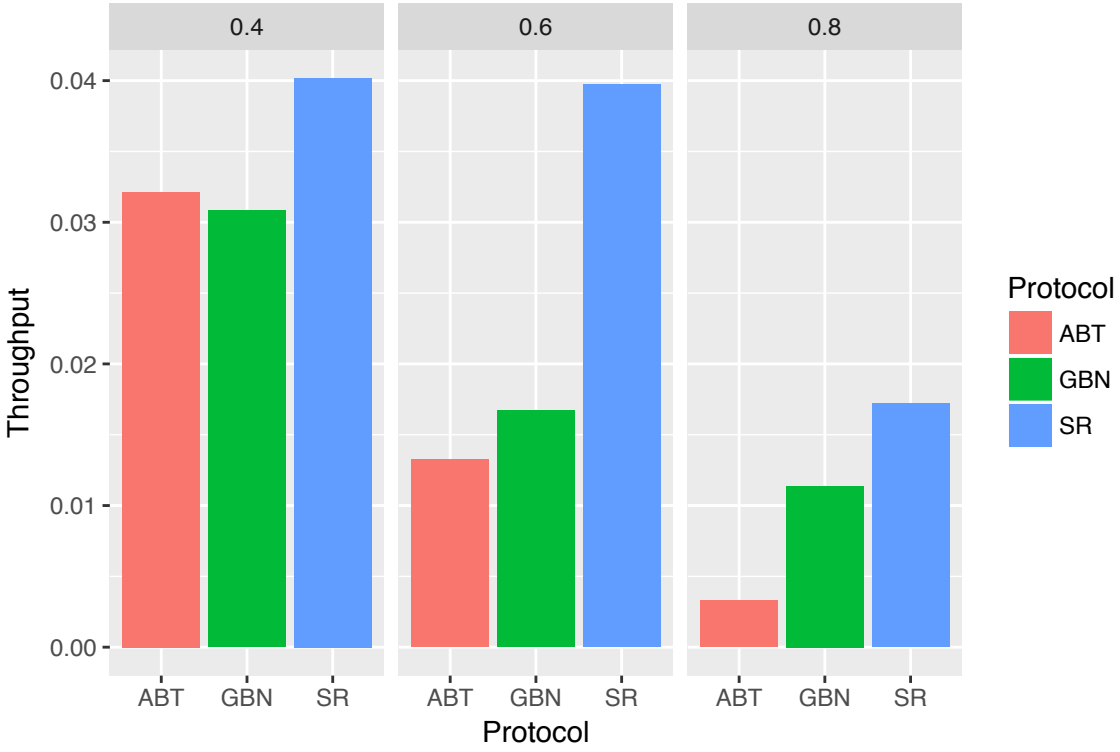
On the other hand, both GBN and ABT show a significant decrease when loss level rises above 20% threshold, as an unexpected result GBN is not able to endure it as well as SR but it is more reliable than ABT as the loss level increase.

[4]Explore adaptative timeout effect on protocol performance.


Incremental loss probability over window size 10
Throughput
0.020
0.015
0.010
0.005
0.2
0.4
0.6
0.8
Loss
Protocol
ABT
GBN
SR


Incremental loss probability over window size 50
Throughput
0.020
0.015
0.010
0.005
0.2
0.4
0.6
0.8
Loss
Protocol
ABT
GBN
SR

Bar chart also facilitates the visualization on how superior is selective repeat compared to the rest of the protocols by outperforming GBN and AR in loss levels of `0.4 0.6 0.8`. Both window 10 and 50 losses were aggregated into a single plot.

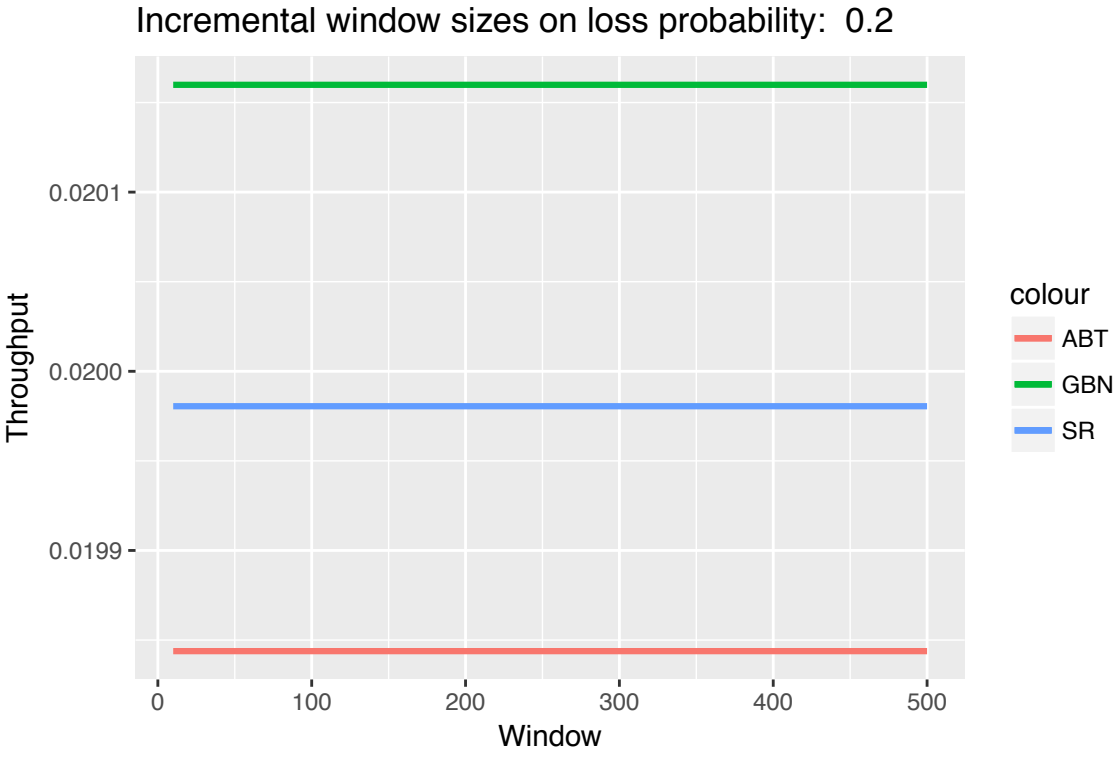


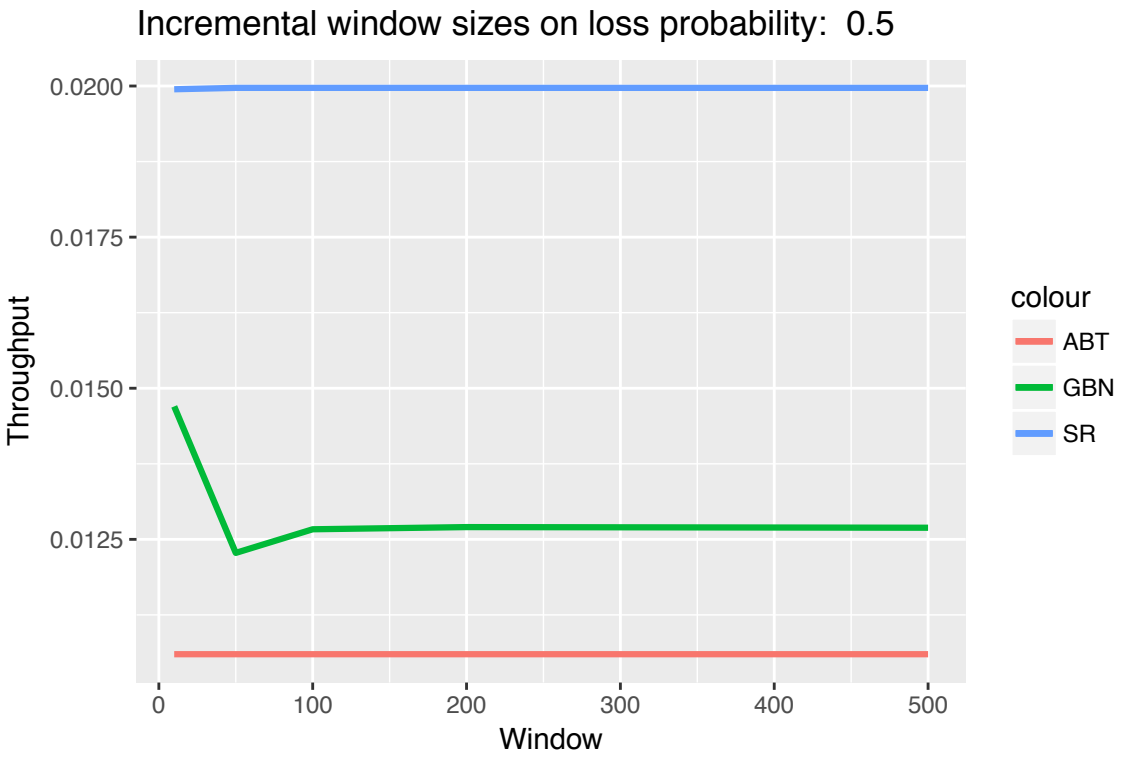
Incremental window sizes on loss probability (Experiment #2)

As suggested the test conducted with the following parameters

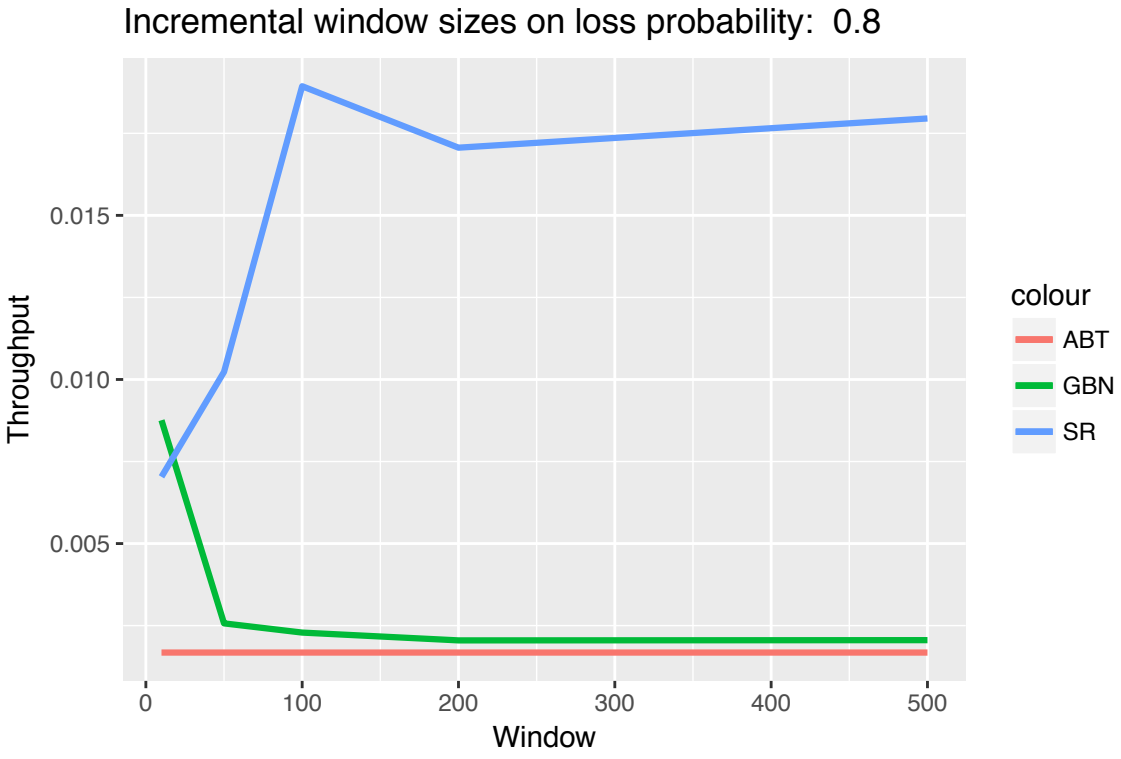
Protocol	Message	Corruption	Loss	Window
Alternative Bit protocol	1000	0.2	0.2 0.5 0.8	10-500
Go back end	1000	0.2	0.2 0.5 0.8	10-500
Selective repeat	1000	0.2	0.2 0.5 0.8	10-500

For both 0.2 and 0.5 loss over different window sizes the protocols render a similar behavior, as expected ABT is immune to any window size changes across all the three plot presented below it had a constant rate of throughput. But even SR and GBN had a steady throughput with this values.





The most interest results are shown when a loss rate is set to up to 80% in the simulation. Notice how both start with very low throughput in around 0.006, and as the window sizes increase GBN loss performance, due to the increased amount of packet that is required to sent each time a packet is lost, in contrast, selective repeat benefit by performing significantly better, a bigger window and the buffering of packets in the receiver allows it to endured a bigger loss rate. Similar to the first batch of experiment its double buffer and selective packet resent are a reflective advantage.



Corruption vs packet loss experiment

So far the experiments have isolated the protocol for simultaneous variance in corruption and loss. The goal is to explore in detailed the effect of the combination of these two components for each protocol since they both have similar effects on a packet on the network, which of these will be the most detrimental for each protocol.

Experiment run with the following set of conditions:

- An increased rate of packet loss from a network that ensures no loss during transmission (loss at 0) to a full loss rate that guarantees all packet will be loss (loss level at 1)
- An increased rate of packet corruption from a network that ensures no corruption will ever occur during transmission (corruption at 0) to a full corrupted network incapable to transmit packet without corruption.

A total of **363 corruption experiments** that took around 15 hours to finished.

Protocol	Message	Corruption	Loss	Window
Alternative Bit protocol	500	0 - 1	0 - 1	50
Go back end	500	0 - 1	0 - 1	50
Selective repeat	500	0 - 1	0 - 1	50

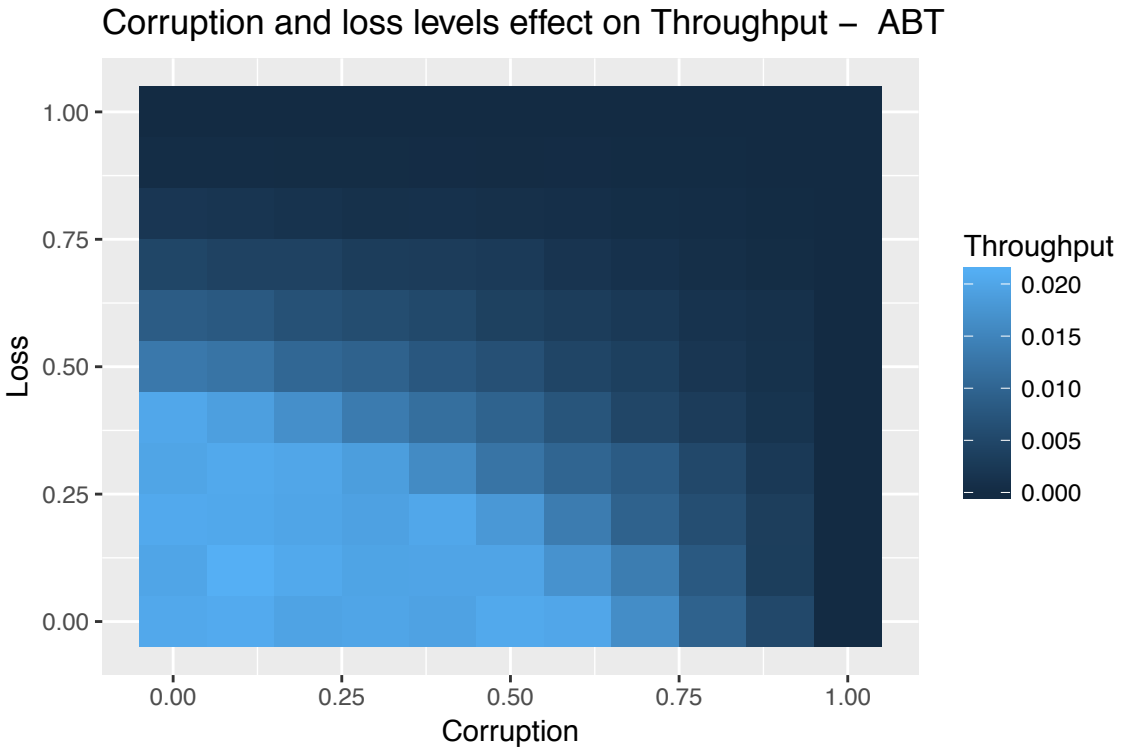
Each protocol relationship between throughput, loss and corruption vary, in order to ease the visualization of two variables over one two statistical tools are used:

- Heatmap that will plot Corruption and Loss, with intensity in colors for throughput.
- Linear regression model output the statistical correlation between the variables and the expected effect each has per protocol.

ABT Heatmap and regression

The three protocol exhibit the same trend but with keys difference in how they support the combination of corruption and loss. As seen in all of them an increased level of corruption or loss can have an impact on the message delivery.

Although ABT is more susceptible to lose level than corruption level, this can be attributed to the timeout interval set to 20, which is a bit more than the average RTT, the nature of ABT make it wait in order to send the next packet, while when it detects a corruption it just resent the packet immediatly.

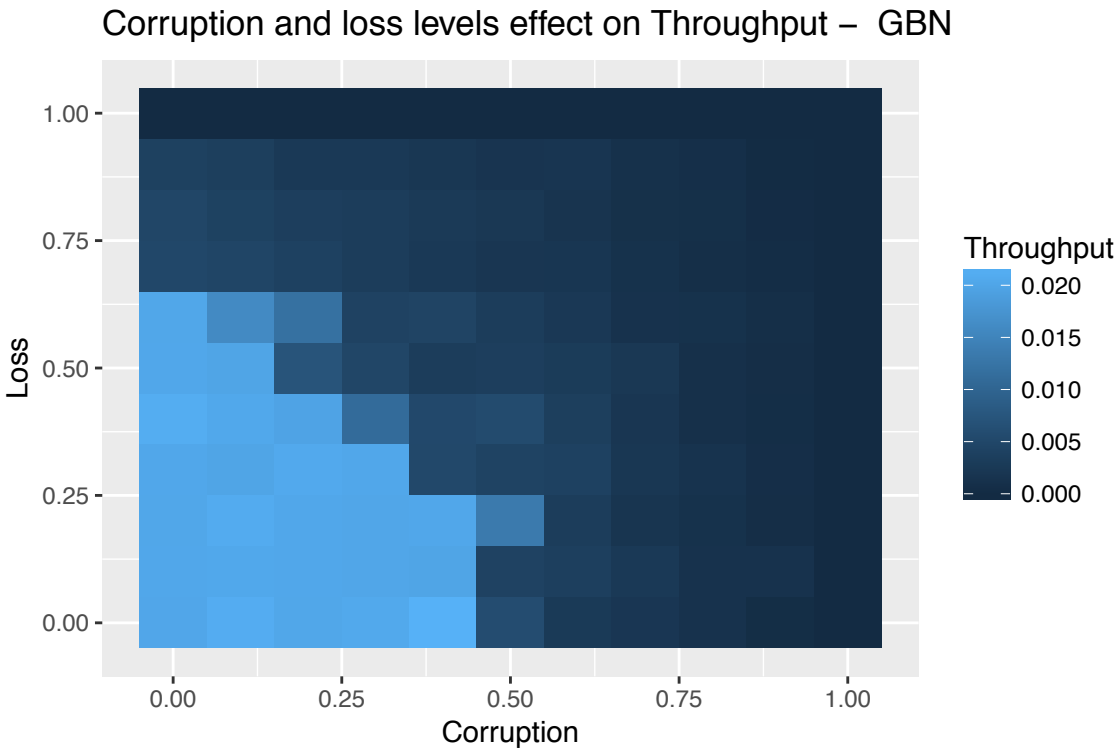


A linear regression shows the relationship between corruption and loss levels, as seen in the formula loss levels have a higher impact on the throughput in ABT protocol, its expected throughput over perfect condition would be around `0.0221014`.

$Throughput = 0.0221014 - 0.01199 Corruption - 0.0178136 Loss$

GBN Heatmap and regression

Go back end in the other hand can endure a higher level of loss while preserving throughput performance, pipelining take an important role allowing the protocol to send more packet at the same time and dequeuing faster than a stop and wait protocol. Unfortunately, it falls short at higher corruption levels, and when both corruption levels and loss levels reach around 0.5 the throughput is diminished significantly.

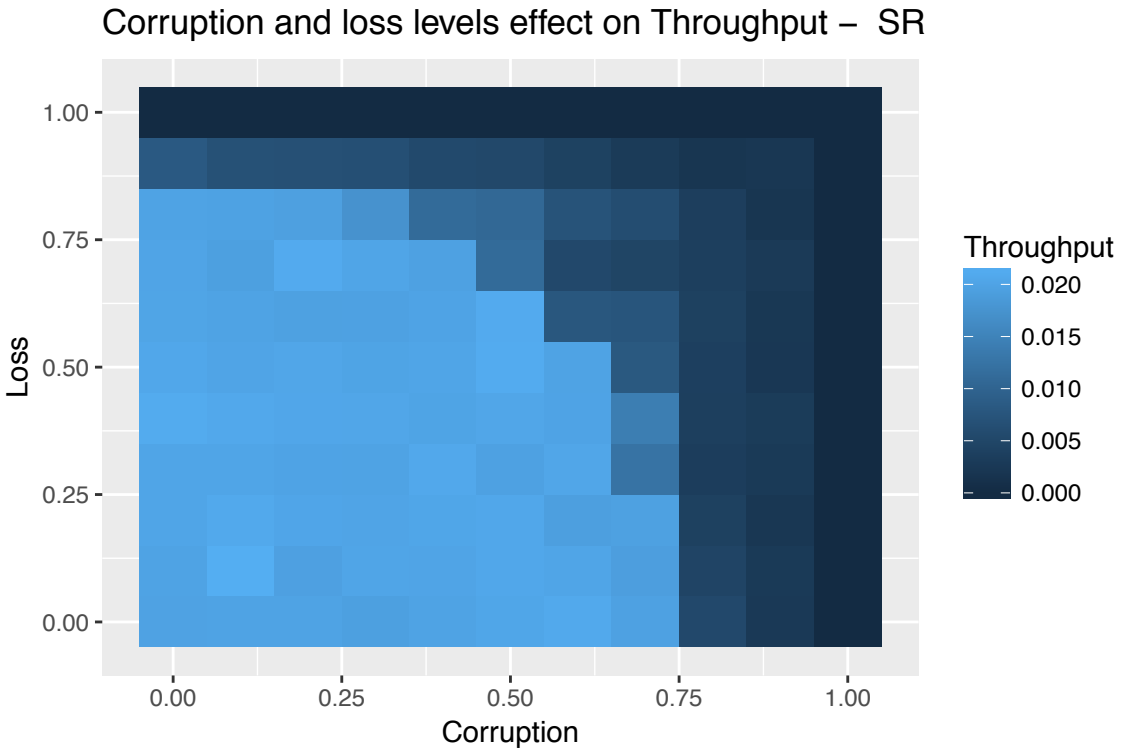


In case for Go back end the linear regression present an oposite behaviour than ABT, as the corruption levels affect the throughput more as it can be seen in the graph.

$Throughput = 0.0201085 - 0.0160385 Corruption - 0.0116855 Loss$

SR Heatmap and regression

Selective repeat has the lighter colors area, supporting levels of both corruption and loss up to 60% without significant loss of throughput, in fact, it takes levels of 80% of any of these components to affect negatively throughput to the protocol.



Finally, the regression shows, just as the graph above on how the Selective repeat protocol has similar corruption and loss coefficients, but a biasing variable of $\beta_0 = 0.02669$, allows it higher throughput on different condition of loss and corruption. The bias variable act as a compensation factor to allow a proper prediction of throughput, it can be attributed to the nature of the algorithm and not to any network factor.

$Throughput$ = 0.0266916 -0.0181862$Corruption$ -0.0126868$Loss$

Time experiments

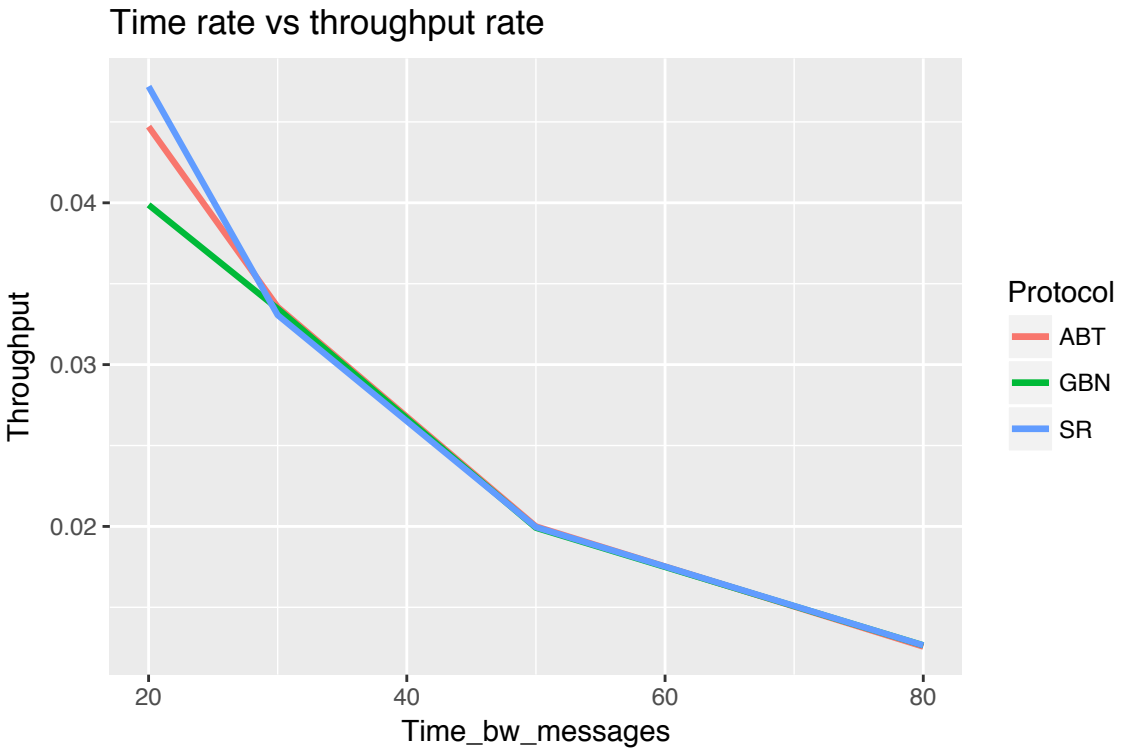
Time experiment goal is to determine the significance of time rate and time delay for each protocol.

Time rate over throughput

The first set of experiment is aimed to observed how time rate changes can make protocol behave differently with the following parameters:

Protocol	Message	Corruption	Loss	Window	Time rate
Alternative Bit protocol	1000	0	0.2	10,50	20 30 50 80
Go back end	1000	0	0.2	10,50	20 30 50 80
Selective repeat	1000	0	0.2	10,50	20 30 50 80

Results showed a trend identical for all of the protocol when time rate is increase significantly throughput gets lower, which can be explained by the relationship that throughput has over the time. If the time rate is high then packets will have to be delivered in longer periods of time, what is surprising is how much the sent packet rate affect all protocols equally, as shown in the graph below.



Adaptative timeout vs Fix timeout

So far, a fixed timeout delayed between packets have been considered, but an interesting aspect of TCP as a protocol is the ability to adapt its transmission timer depending on the performance of the network, the fastest it detects a network can transmit the smaller the interval gets. This is the only experiment that **required recompilation in of the source code** by a single constant in `constant.h`.

`ADAPTATIVE_TIMEOUT`: Enable adaptative timeout for the three protocol by default this flag is set to FALSE. It enable the calculation for an estimatedRTT using formula from:

$EstimatedRTT = \alpha * EstimatedRTT + (1 - \alpha) * SampleRTT$

$DevRTT = (1 - \beta) + \beta|SampleRTT - EstimatedRTT|$

$TimeoutInterval = EstimatedRTT + 4 * DevRTT$

> The above SHOULD be computed using $\alpha = 0.125\beta = 0.25$ (assuggested in [JK88])[2]

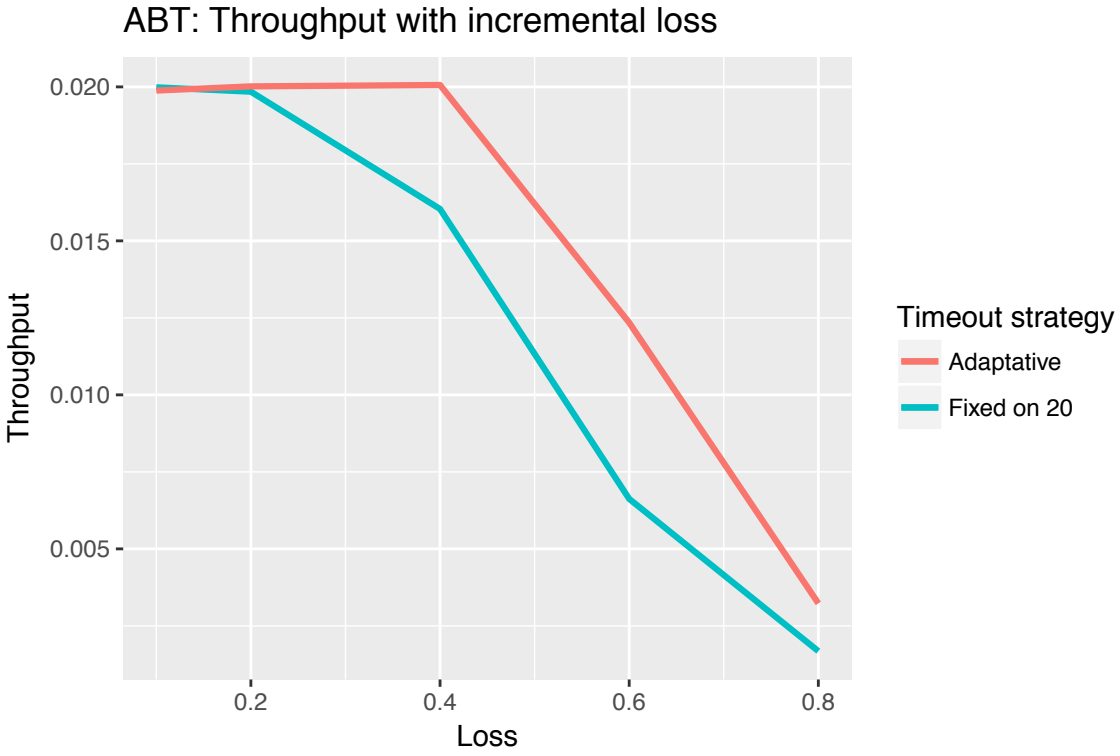
This implementation take several input from the RFC although with the two keys difference:

- Unlike suggested on [2] "Until a round-trip time (RTT) measurement has been made for a segment sent between the sender and receiver, the sender SHOULD set RTO <- 1 second", it is assumed the `AVG_TRAVEL_TIME` as the default timeout until the TimeoutInterval is calculated.
- Other rule that is broken intentionally is "Whenever RTO is computed, if it is less than 1 second" instead the RTO is set again to the last TimeoutInterval.

The results show a significant increase in throughput compared to previous experiments for each protocol, the graph shows a contrast. The reason is rather clear adaptative interval allow each protocol to wait the minimum amount of time per packet transmission, since throughput is $packet/time$, by reducing the time, the throughput gets increased.

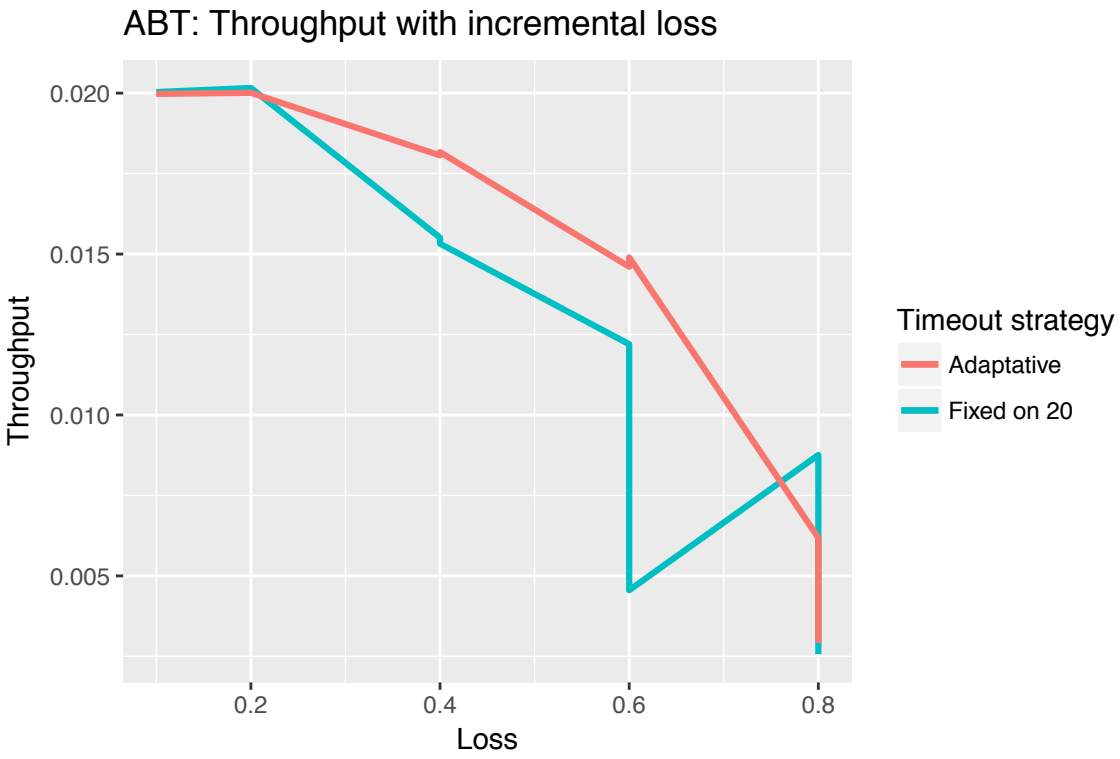
ABT adaptative vs fixed

The adaptative implementation allows ABT to keep a throughput more stable with the same condition as Incremental loss probability on window sizes (Experiment #1) experiments #1, the increase in throughput is significant even with an implementation so simple for the time interval it decrease the waiting time for resending a loss packet and therefor allows it to transmit data at a faster rate.



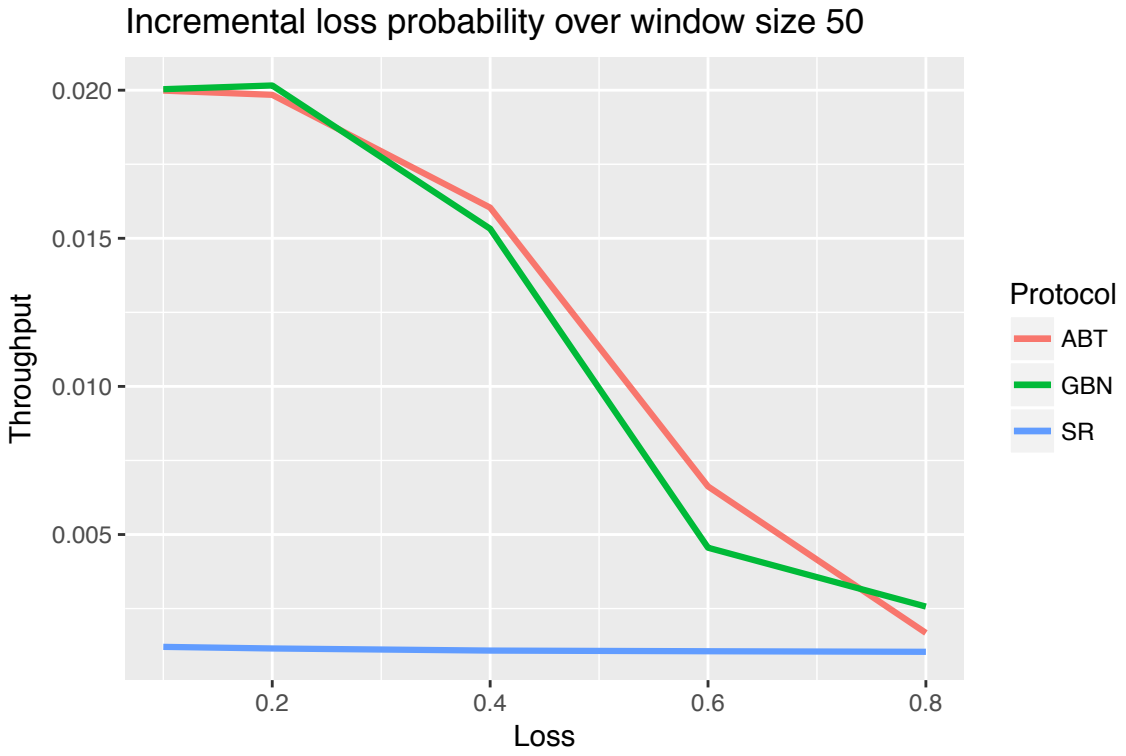
GBN adaptative vs fixed

Go back end protocol also benefits although the graph in the graph the comparison between each strategy become evident as the loss level increase, this can be attribute to the window size that allows GBN to transmit data at a faster rate than ABT, as the loss level increase though the adaptative strategy becomes more efficient at decreasing the time for transmitting data.



Caveat: Selective repeat wrong implementation

The implementation of each protocol although is relatively simple to comprehend but the amount of different variables that can affect the performances (corruption, loss, timeframe, window) makes it challenge to debug. The automatic test for basic, sanity and advance does provide peace of mind however even test pass it is not a solid indicator that the protocol implementation is correct. For instance in the graph bellow one of the earliest attempt to plot data, shows selective repeat performing significantly worse that any of the two procols which directly contradicts the theory behind it, ergo the **selective repeat implementation was indubitable wrong**.



Source code references

- *Queue with BSD libc* which provide a set of macros for automatically working with queue.
  - BSD Library Functions Manual
  - Queue examples by Rogério Carvalho Schneider
- *Checksum* the assigment description referred to an easy way to do checksum which for the most part was followed, except that a logical XOR was preffered over bitwise.
  - Is this the right way to find a checksum? - Stackoverflow by Ian
- R Studio for the analysis and presentation as it provide multiple mechanism for visualizing data. The script for data visualization is provided with the code base as `MNC.Rmd` and the experiments builder `build_experiment.sh`[5].

References

[1] G. Varghese and A. Lauck, "Hashed and hierarchical timing wheels: Efficient data structures for implementing a timer facility," IEEE/ACM Trans. Netw., vol. 5, no. 6, pp. 824–834, 1997.

[2] RFC6298 | V. Paxson., M. Allman,J. Chu. "Computing TCP's Retransmission Timer" - June 2011https://tools.ietf.org/rfc/rfc6298.txt

[5] 135 experiment run sequentially in stones server take around 10 hours