Question 1

SIMULATOR:
In the ABP implementation, the process starts `do_gbn()` starts the simulator. It runs a while loop until a success counter reaches the desired number of successful packets. Once the global parameters of tau, delta timeout and BER have been set in the **main()** function. In the main while loop, which counts up the number of successful packets transmitted, first, a frame is initialized, and is given a size and sequence number, assigned using the modulo operator. This frame is packaged as a packet, and the **do_send()** function is called. This function returns a **success** object which is either positive or negative (depending on if the packet was successfully transmitted or no). This success packet's positivity is checked and if it is positive, the success count is incremented, which then goes to the top of the while loop and transmits a new frame. If the success is negative, then the event scheduler is read again, and the current time is updated based on that. At the end of the while loop, the throughput is computed, using the current time as the divisor.

SENDER:
In the **do_send()** function, the current time is incremented by appending a transmission delay, each time a new frame is sent over to the channel. In this function, the **send_abp()** is called. This function returns a received packet from the reverse channel. This packet that is received is then checked if it was null, that means, it was lost in the transmission. In this case, the event is not added to the event scheduler, otherwise, it is added in the event scheduler as an acknowledgement. Then, the event scheduler is read, with the top event in it returned using the read_es() function, and stored in next_event. This next_event is then checked for an acknowledgement or timeout event, which is discussed later.

SEND:
The send_abp() function, constitutes the forward channel, receiver and the reverse channel. When send_abp() is first called, a new timeout is registered (at current time + delta timeout), and any existing timeouts are purged. Then, channel_abp() is called with the current frames sequence number and length passed in as parameters. This function computes if any errors were present in the transmission through the channel. it is done by performing a repeated loop generating 0 or 1 based on the given BER. Then the number of 0s are counted and the packet's transmission is decided to be lost if zero count is greater than 5; it is a good transmission if zero count is = 0; and if it is between 1 and 4, then it is has an error in transmission. This is then sent to the receiver, which verifies if the sequence number of the incoming frame is the expected frame, and if so, the next_expected_frame is incremented by 1 mod 2. If the incoming packet had errors, then the next_expected_frame is not incremented and then the return packet is sent to the reverse channel, but this time the frame length is simply the length of the header. At each time, the current is incremented. Passing through the channel adds a propagation delay of tau, then passing through the receiver adds a time of H/C, then again the reverse channel propagation delay of tau, and finally the initial transmission delay L/C. This packet is what is read as packet_received in the do_send() function.

CHECKER:
After the received packet is added to the event scheduler as discussed earlier, and the event scheduler is read for the top most event (the earliest in time), that event is removed from the event scheduler and is checked for its type. If the event type is an ACK, then its error flag and RN are checked. If all of those were correct, it is concluded a *good* packet has been received and the

sequence number is incremented mod 2, and the next_expected_ack is also incremented mod 2. This is then added to an success object which is returned to the do_send() function. The current time is also incremented to the time recorded in the event that was read. If the packet received had errors, then the success object returns a value of 0, which the do_send() function returns to do_abp() and do_abp() is instructed update the current time based on the time of the top event in the event scheduler then re-run the while loop, which would send the same frame, which was in the buffer, again. If the top event in the event scheduler was a timeout, then the frame in the buffer is retransmitted by recursively calling do_send() until a success if found.

ABP SUMMARY OF RESULTS:

Δ/τ	2τ = 10ms			2τ = 500ms		
	BER = 0.0	BER = 1e-5	BER = 1e-4	BER = 0.0	BER = 1e-5	BER = 1e-4
2.5	954,441.33	820,880.82	242,231.88	23,877.14	20,463.67	5,869.07
5.0	954,441.33	740,443.55	150,001.93	23,877.14	17,789.92	3,386.43
7.5	954,441.33	672,833.93	109,577.53	23,877.14	15,963.49	2,367.21
10.0	954,441.33	604,772.67	86,159.63	23,877.14	14,291.54	1,843.74
12.5	954,441.33	565,437.90	70,763.20	23,877.14	12,675.93	1,490.70

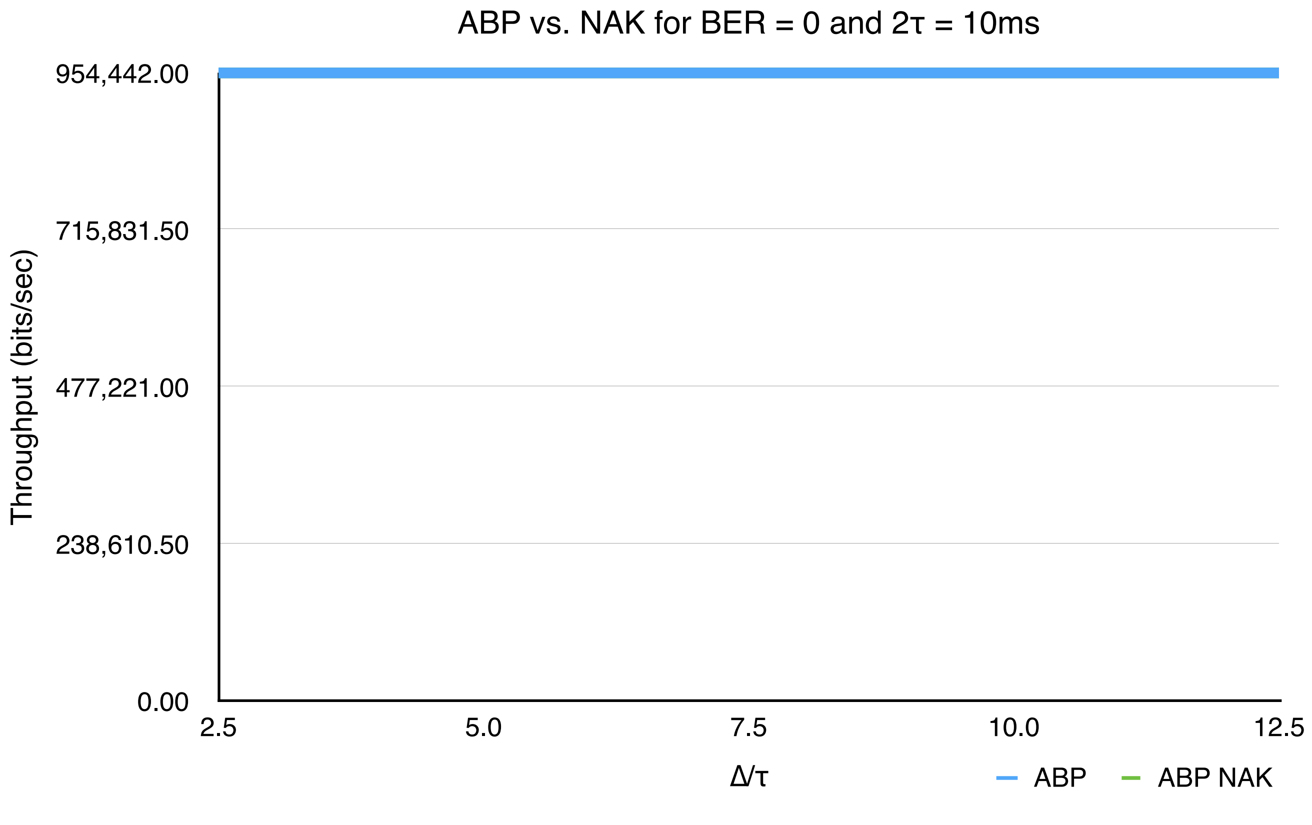
Question 2

In contrast to ABP, ABP_NAK works by acting upon receiving ACK in error or with RN not equal to NEXT_EXPECTED_ACK. The sender takes such events as negative acknowledgements (NAK) and act on them by resending the same packet as soon as such events occur. In comparison to results found for ABP in question1, the general throughput of ABP NAK is significantly higher, for all cases of BER and propagation delay, *tau*. However, for the BER = 0 case for both values of *tau*, the throughput of ABP and ABP NAK is the same because in the BER = 0 case, there are no errors that would require the simulator to wait for an acknowledgement. For BER != 0 cases, for both values of *tau*, 1e-5 and 1e-4, the throughput is higher for ABP NAK and generally remains constant (with little variability) with increasing delta value. The reason the throughput is higher for ABP NAK compared to ABP is because in ABP, upon receiving an ACK with error, the system waits for a timeout before resending the packet, by reading the top event in the event scheduler and updating the value of current time to that, then resending it. However, for the ABP NAK case, as soon as the system detects the arrival of a ACK with error, it immediately goes and retransmits the frame, instead of waiting for a timeout (which could be a lot later than the current time) to occur before retransmitting the frame. This would greatly reduce the the total time for sending a certain number of packets, thereby increasing the overall throughput of the system. NAK, or negative acknowledgements occur as a result of the protocol interpreting ACKs with error as NAKs, and then instead of waiting for a certain amount of time before it is able to resend the same frame to the other node, the receiver, through the channel, it simply looks at the buffer, which would hold the frame that was sent, and its acknowledgement was received in error. Therefore, in real applications of ABP, the ABP NAK mechanism is often used, as it increases throughput of packets transmitted while keeping the throughput constant over increasing timeout period. The throughput remains constant with increasing delta is because the ABP NAK protocol doesn't adhere to the timeout, and immediately resends it, instead of waiting on the timeout amount and thereby increasing time.

ABP NAK SUMMARY OF RESULTS

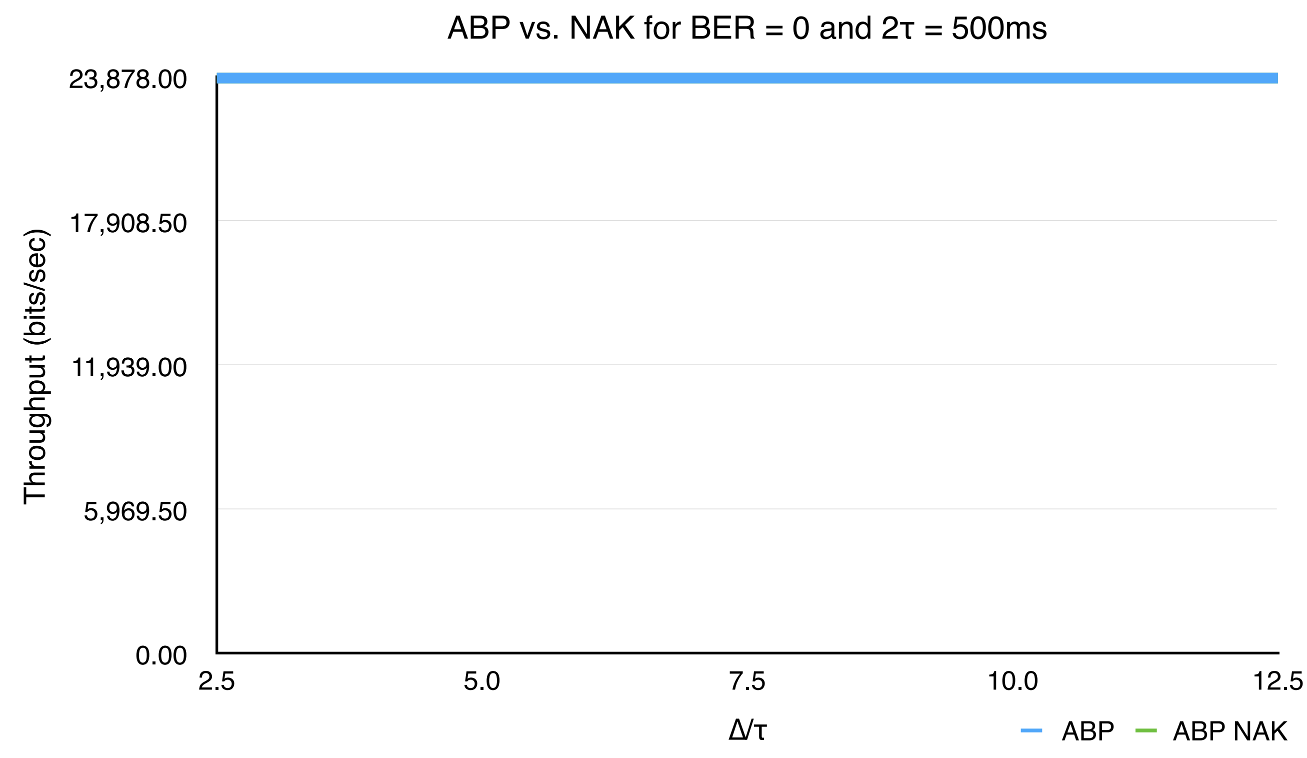
Δ/τ	2τ = 10ms			2τ = 500ms		
	BER = 0.0	BER = 1e-5	BER = 1e-4	BER = 0.0	BER = 1e-5	BER = 1e-4
2.5	954,441.33	837,229.24	274,550.68	23,877.14	21,046.40	6,847.15
5.0	954,441.33	838,921.80	276,023.07	23,877.14	20,926.50	6,826.23
7.5	954,441.33	841,659.02	271,022.65	23,877.14	21,009.36	6,833.28
10.0	954,441.33	837,082.38	273,473.13	23,877.14	21,115.26	6,833.84
12.5	954,441.33	842,401.88	272,298.02	23,877.14	20,979.82	6,830.74

ABP vs. ABP NAK PLOTS:

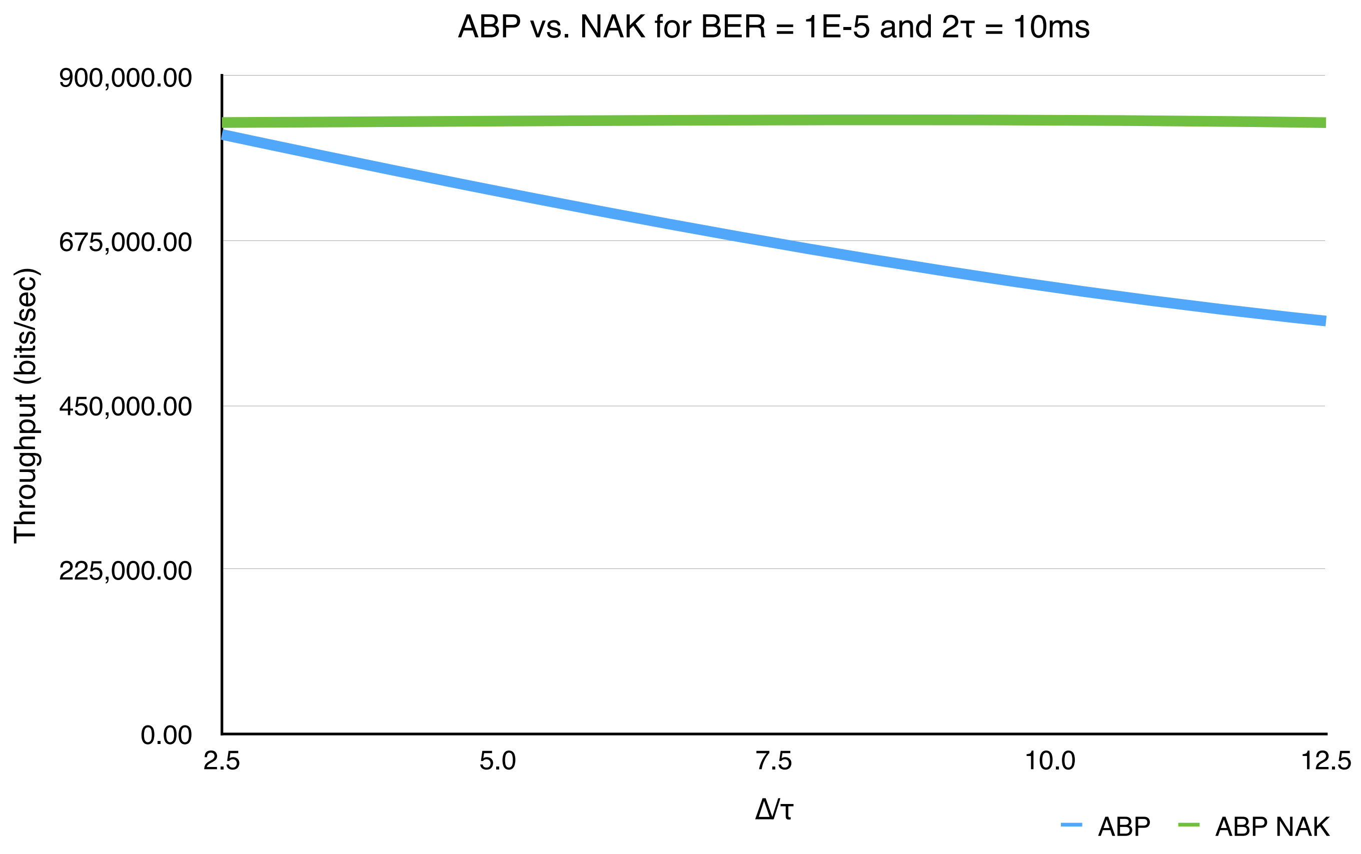


In the above plot, it can be seen that the throughput for the ABP NAK case for BER = 0 remains constants, similar to the throughput for ABP. This throughput is capped at around 954,441 bits/second, as expected and explained in Question 2.

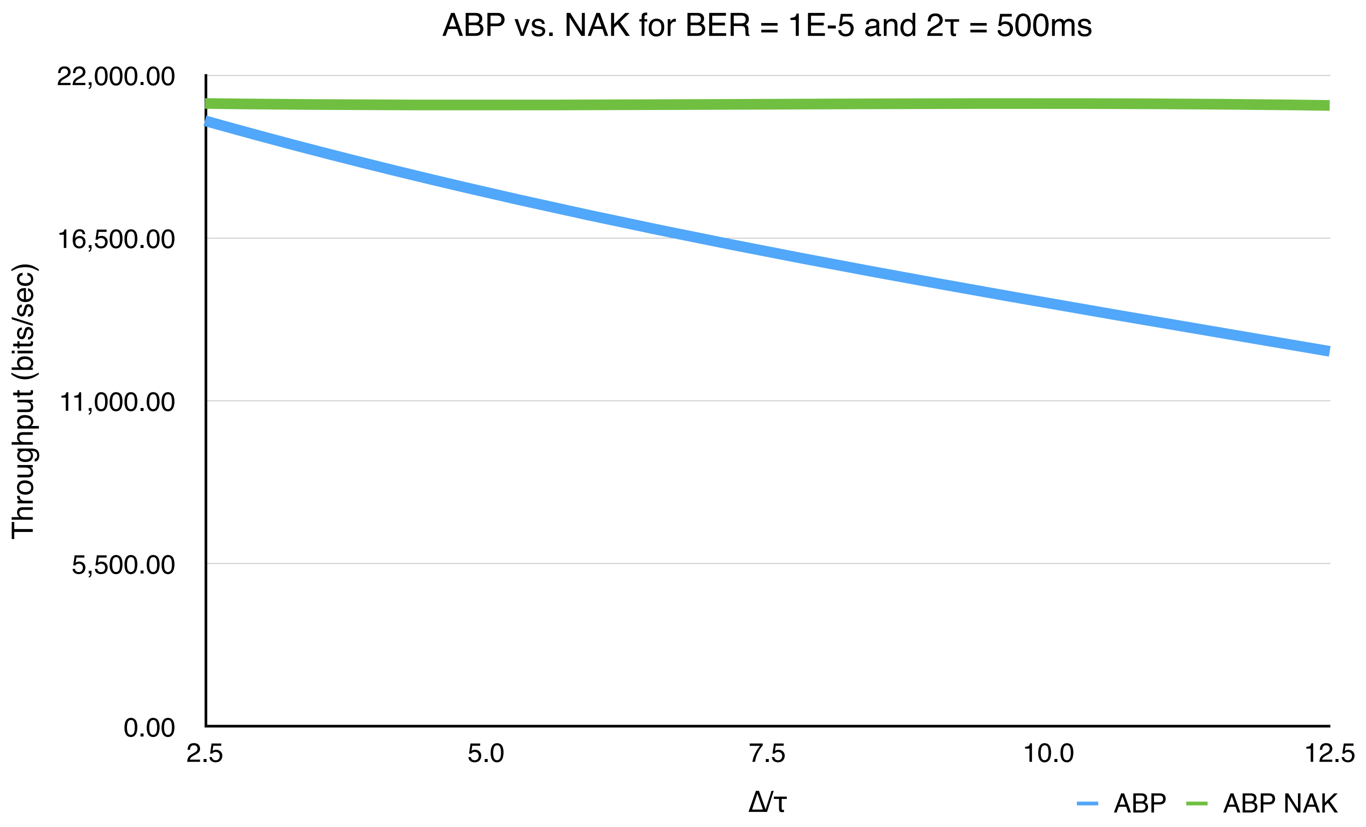
NOTE: The above and below plots for BER = 0 show lines for both ABP and ABP NAK, however, since the values for both are exactly the same.



In the above plot, it can be seen that the throughput for the ABP NAK case for BER = 0 remains constants, similar to the throughput for ABP. This throughput is capped at around 23,877 bits/second, as expected and explained in Question 2.

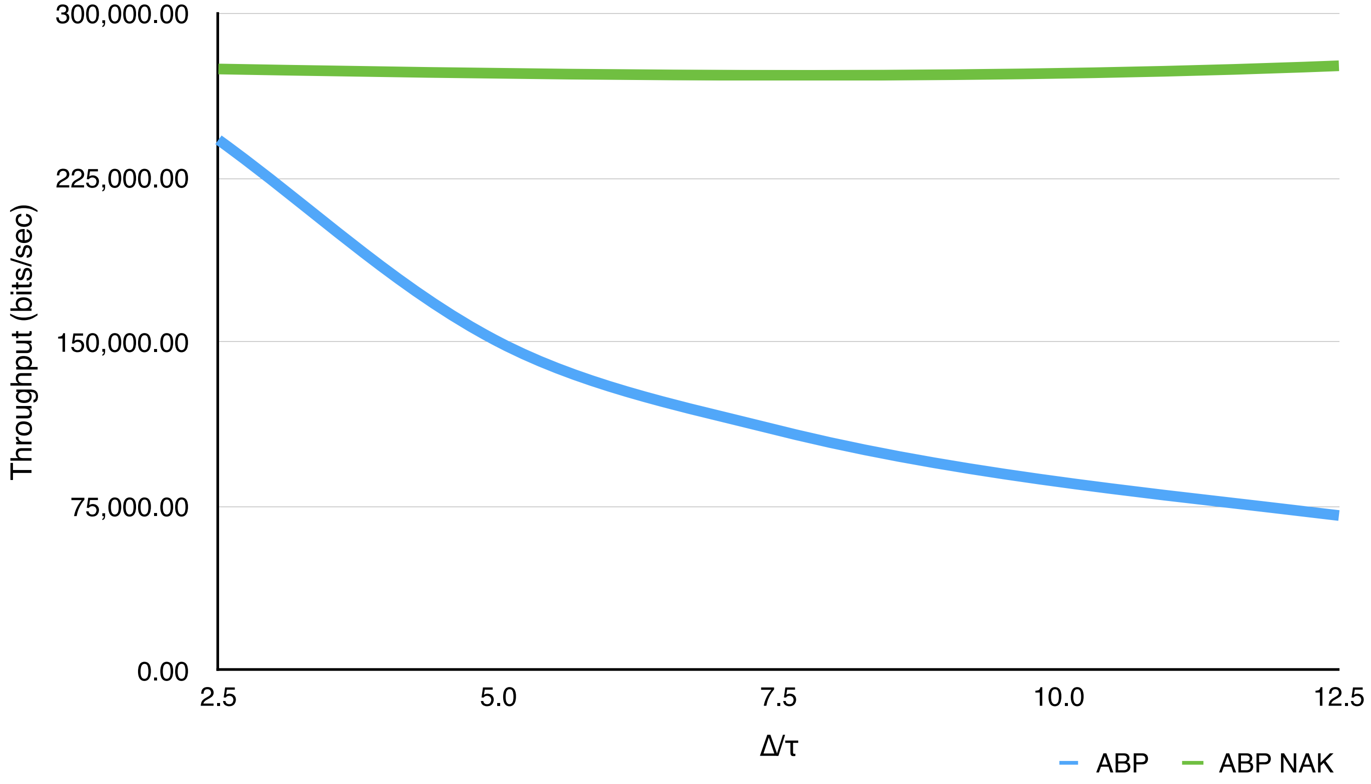


In the above plot, it can be seen that the throughput for the ABP NAK case for BER = 10ms remains constant at around 800,000 bits/sec, however the throughput for ABP decreases almost linearly to a value of around 500,000 bits/sec for delta = 12.5, as expected and explained in Question 2.



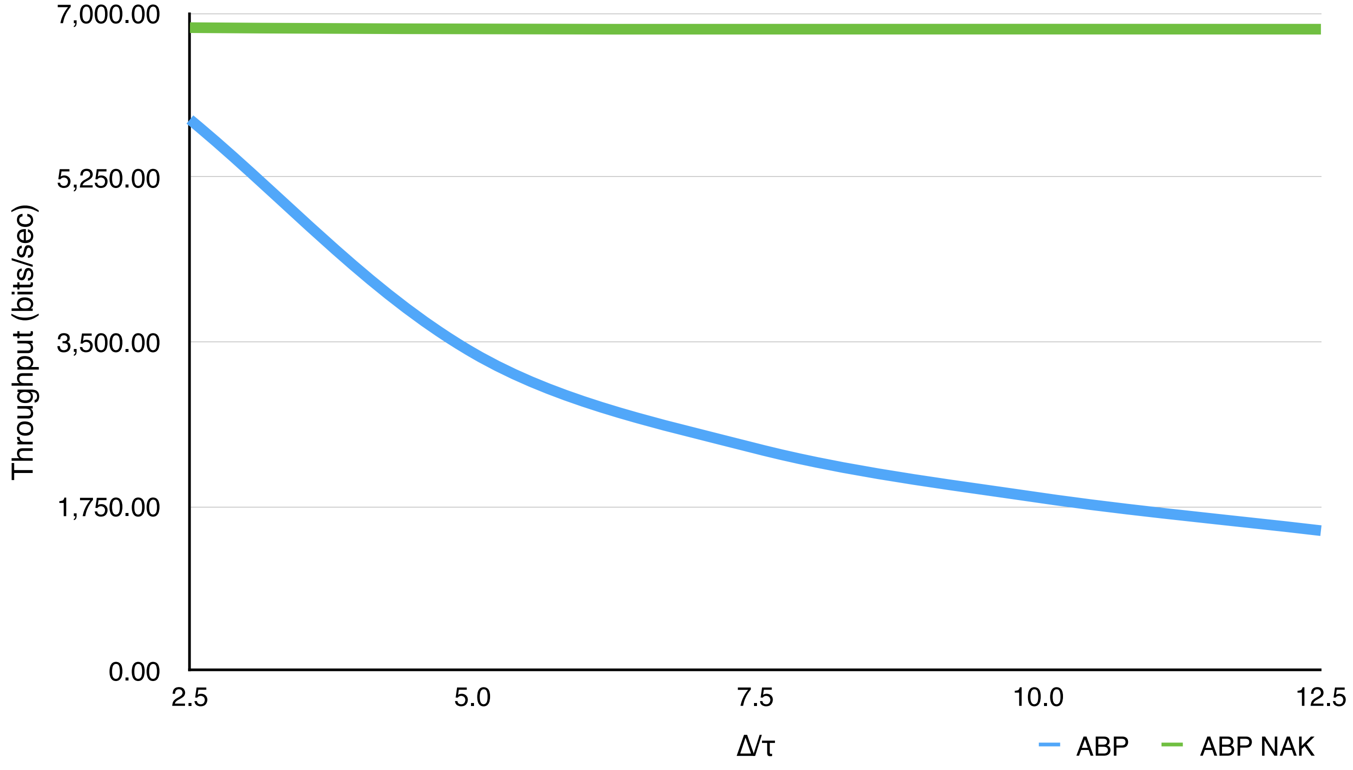
In the above plot, it can be seen that the throughput for the ABP NAK case for BER = 500ms remains constant at around 20,000 bits/sec, however the throughput for ABP decreases almost linearly to a value of around 10,000 bits/sec for delta = 12.5, as expected and explained in Question 2.

ABP vs. NAK for BER = 1E-4 and 2τ = 10ms



In the above plot, it can be seen that the throughput for the ABP NAK case for BER = 10ms remains constant at around 250,000 bits/sec, however the throughput for ABP decreases almost linearly to a value of around 70,000 bits/sec for delta = 12.5, as expected and explained in Question 2.

ABP vs. NAK for BER = 1E-4 and 2τ = 500ms



In the above plot, it can be seen that the throughput for the ABP NAK case for BER = 500ms remains constant at around 7,000 bits/sec, however the throughput for ABP decreases almost linearly to a value of around 1,500 bits/sec for delta = 12.5, as expected and explained in Question 2.

Question 3

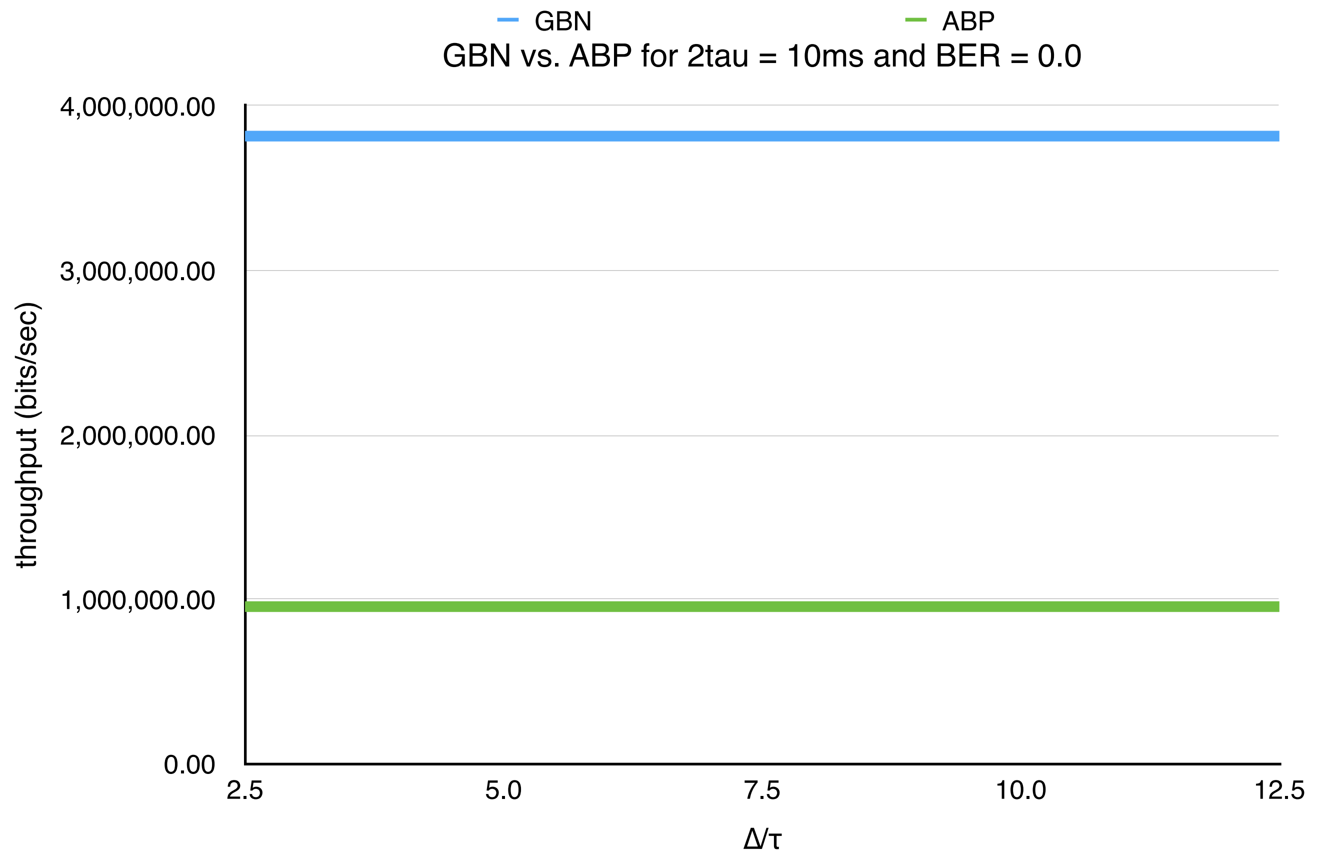
INITIALIZER:
In the GBN implementation, the initialize() function is called first, that sets all values needed for the simulation to zero (or their initial values).

EVENT PROCESSOR:
The Event Processor in the GBN implementation of the simulator runs until there are no more events in the event scheduler, after which it computes the throughput by using the current time variable. While the event scheduler has events (i.e., ACK or TIMEOUT), the event processor checks first, if the desired number of packets (in this simulation, set to 10,000) is reached. If it is, then the loop is broken out of and the throughput is computed. Then, if the desired number of packets is not reached, the first event in the event scheduler is read, and if it's an ACK in ERROR, then it is ignored and the event scheduler is read again. Then the current time is updated with the first event in the event scheduler's time. If the first event is a TIMEOUT event, then the its location counter is set to 0 and the NEXT_EXPECTED_FRAME is set to the sequence number of the first frame in the buffer. Then the existing timeouts are purged and the sender() function is called (this is discussed later). If the event was an ACK without error and the RN was one of the expected RNs, then the the window shifting amount is calculated using (RN-P + N + 1) mod (N + 1), and the buffer is shifted to the left by that amount, and any existing timeouts are purged. Then a new timeout is registered with the time of the first item in the buffer plus delta timeout. Then the sender() function is called to send any outstanding packets in the buffer.

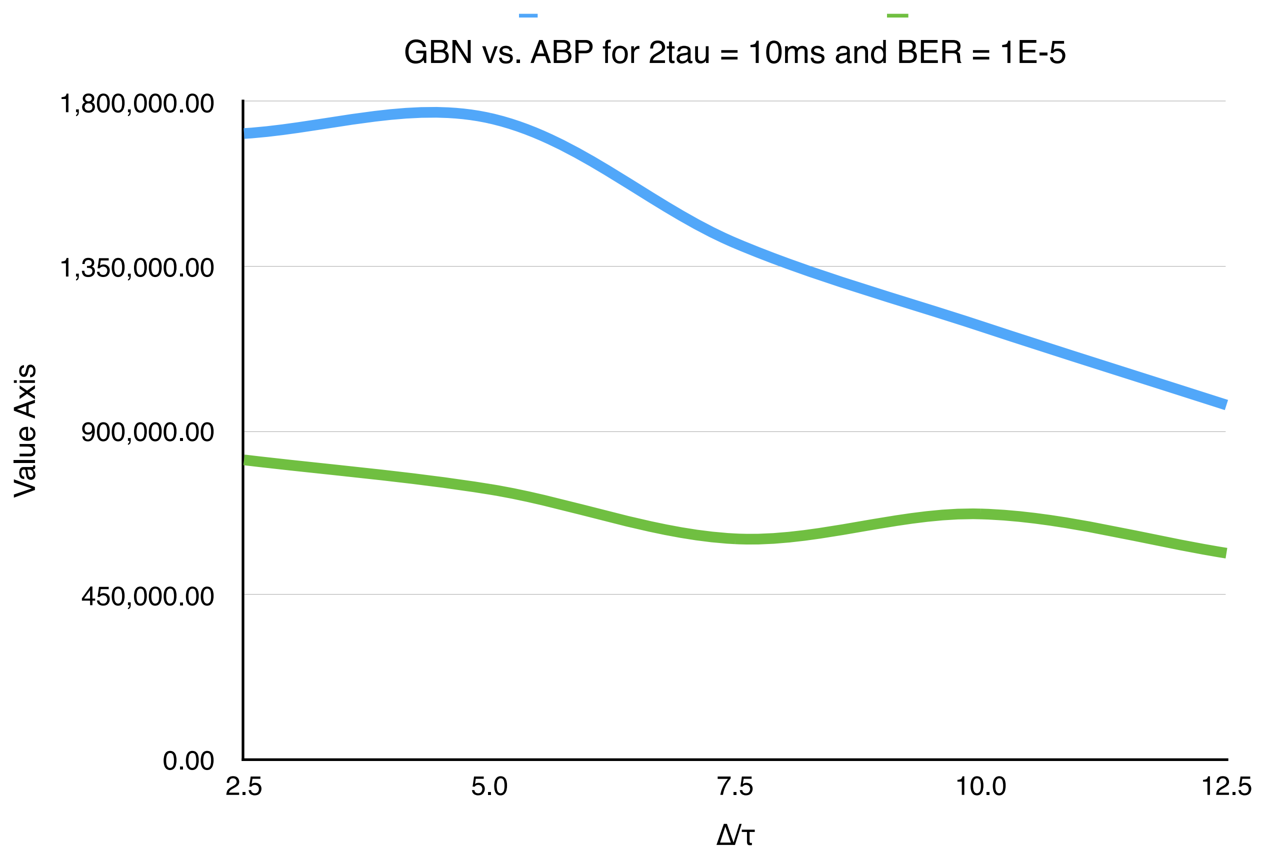
SENDER:
The sender() function loops until there are packets in the buffer, after which it exits the loop. At the beginning of the loop, the function checks if the desired number of packets has been reached, upon which it will exit the loop. If not, then the current time is incremented by the transmission delay amount of the packet about to be sent. The time is then stored in that packets time array, and the NEXT_EXPECTED_ACK for that packet is incremented according to its sequence number + 1 mod (N + 1), where N is the buffer size. Then if the location counter is 0, a new time out event is created at the time of the first item in the buffer + delta timeout. Then the frame is sent to the send() function (same as the ABP case, except for the receiver sequencing the numbers mod (N + 1)). Then after a packet is received, the event scheduler is read for the first event. If it is a timeout, then the timeouts in the event scheduler are purged, counter is reset to 0 and the NEXT_EXPECTED_FRAME is set to the sequence number of the first frame in the buffer, and the process is continued, i.e., the frame is retransmitted. If the next event was an ACK without error, then the window sliding amount is calculated (as discussed earlier), the buffer is shifted left by that amount, the timeouts in the event scheduler are purged, and a new timeout is register at the time of the first item in the buffer plus delta timeout. However, if the RN of the received packet is not in the expected ACKs, but is not an ERROR ACK, then the first item in the event scheduler is removed, the location counter is then incremented along with the sequence number of the the current location counter sequence number.

GBN SUMMARY OF RESULTS:

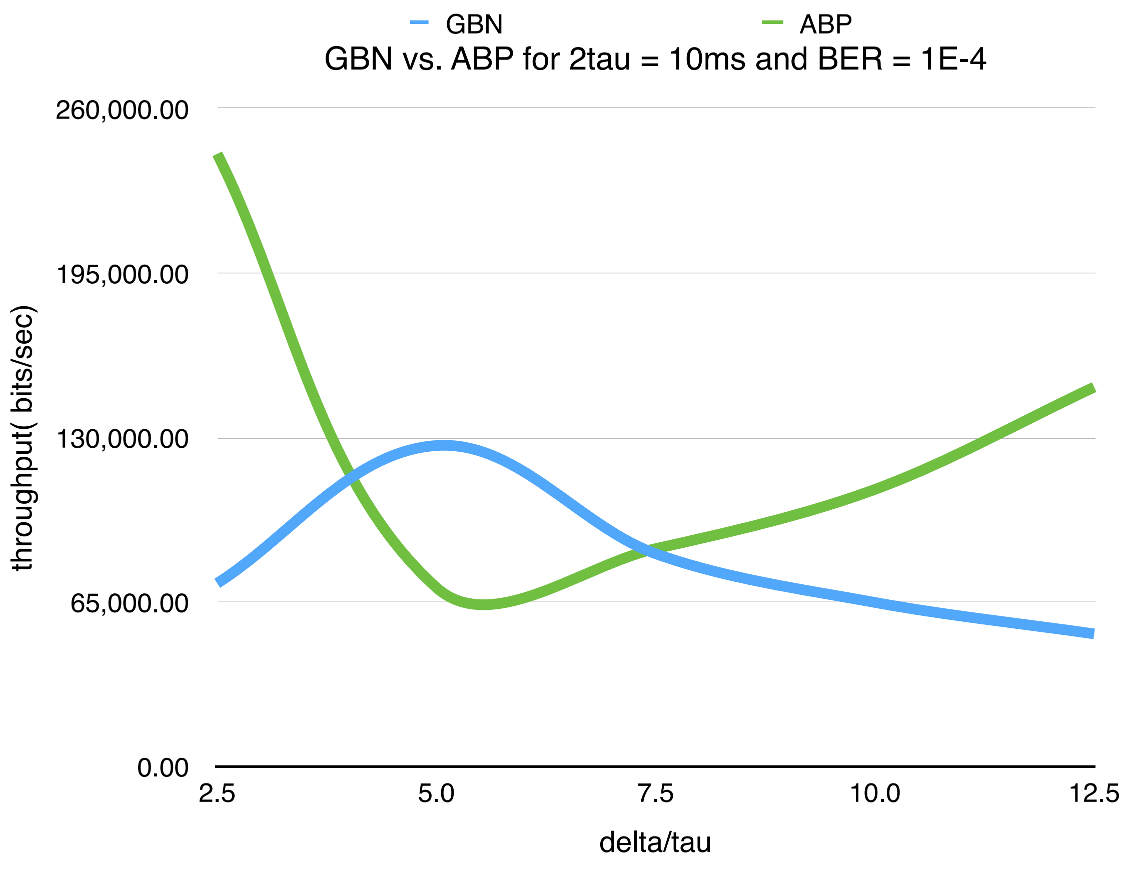
Δ/τ	2τ = 10ms			2τ = 500ms		
	BER = 0.0	BER = 1e-5	BER = 1e-4	BER = 0.0	BER = 1e-5	BER = 1e-4
2.5	3,815,954.19	1,714,596.42	72,477.72	95,507.42	60,912.24	5,395.21
5.0	3,815,954.19	1,756,587.23	126,919.41	95,507.42	42,843.80	2,780.90
7.5	3,815,954.19	1,414,624.68	84,156.27	95,507.42	32,996.95	1,828.12
10.0	3,815,954.19	1,186,919.35	64,892.13	95,507.42	26,890.73	1,442.53
12.5	3,815,954.19	971,189.14	52,467.41	95,507.42	22,519.98	1,098.09



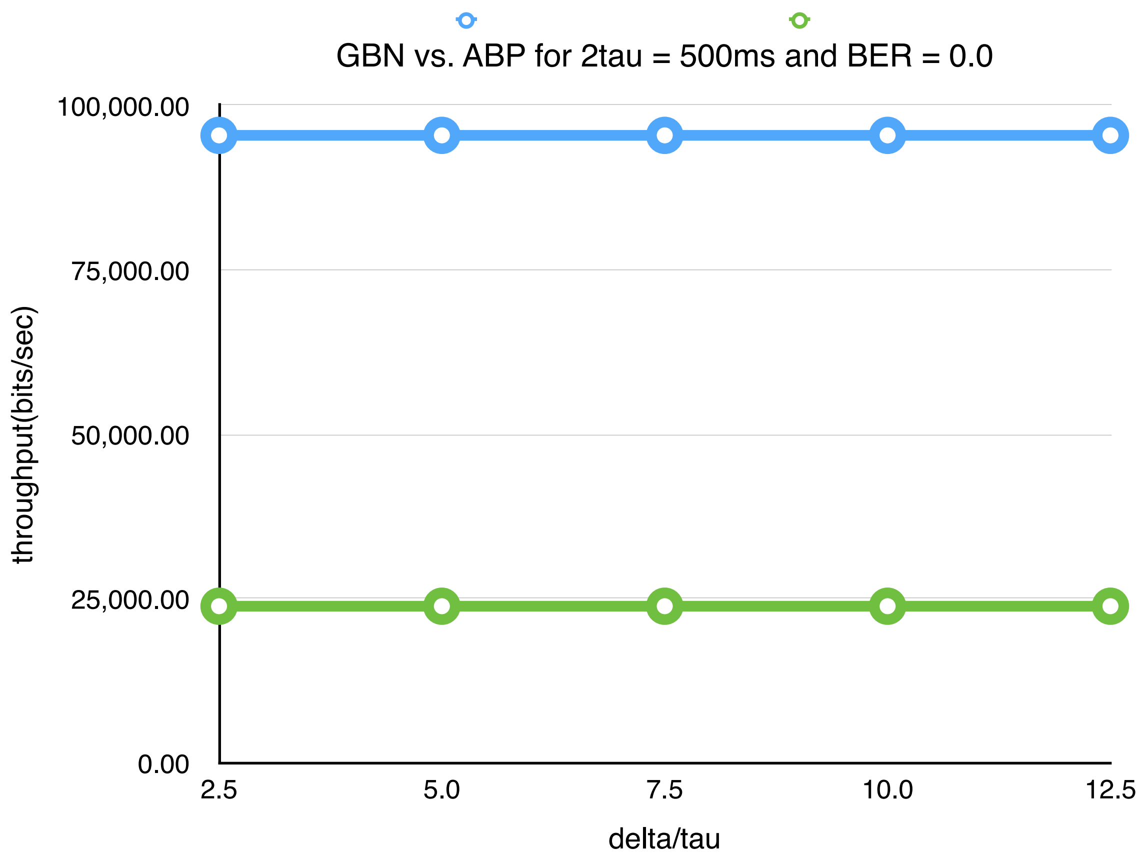
The throughput of GBN is almost 4 times that of ABP, as expected.



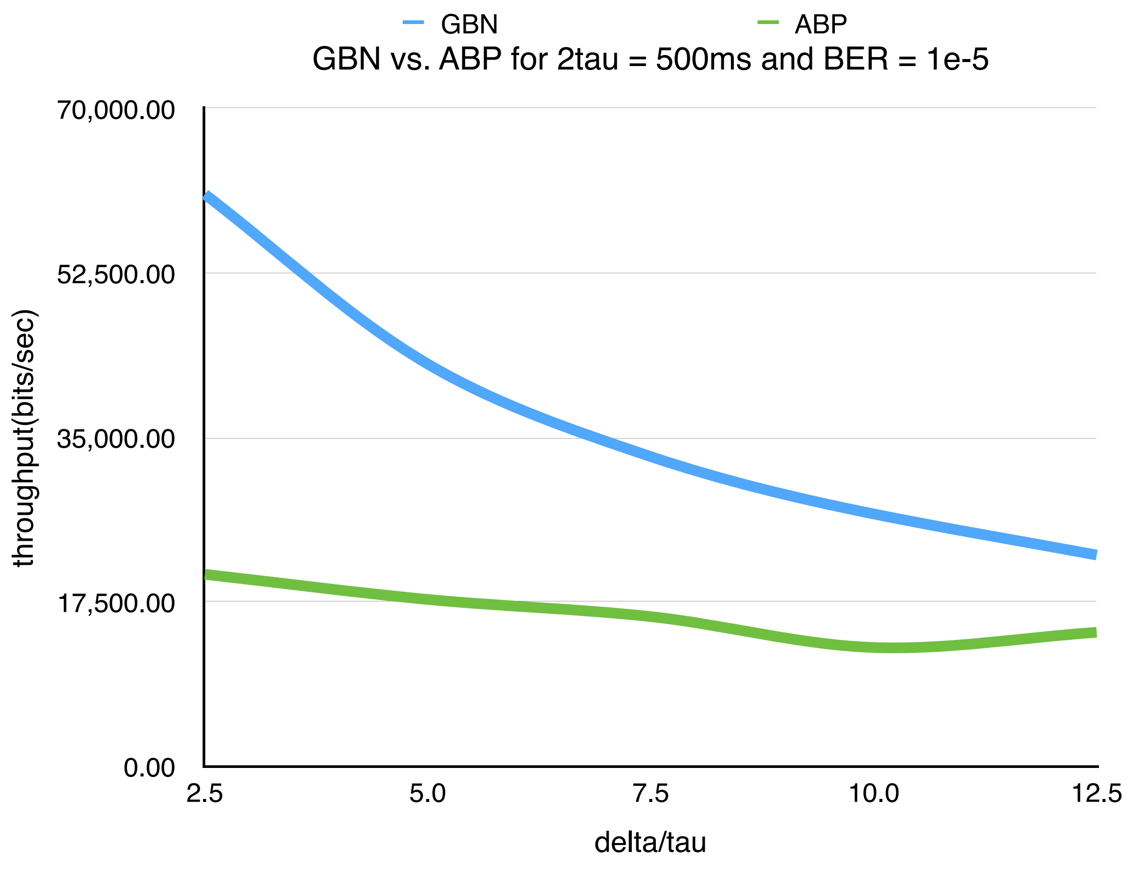
In general, the GBN throughput is greater than ABP, as expected.



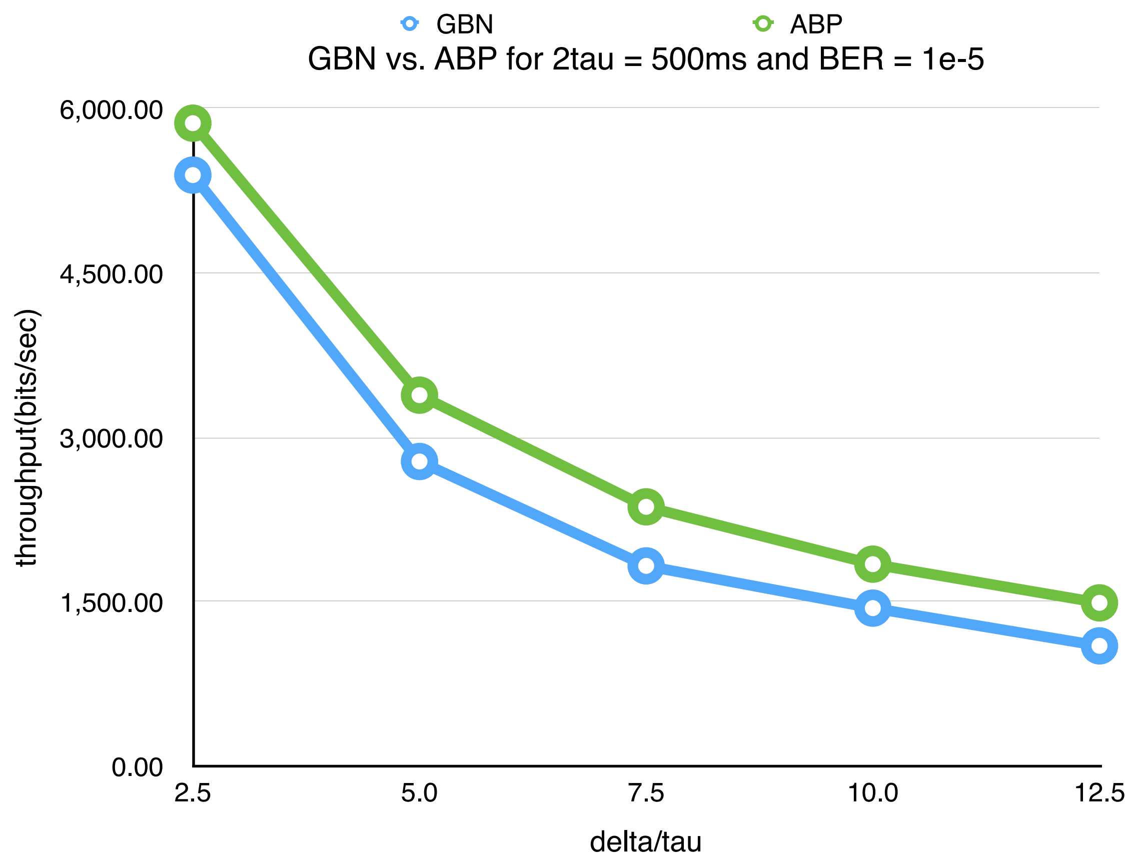
Something went wrong in the simulation, which could not be corrected. However, we should see the GBN throughput to be much greater than ABP.



GBN throughput is 4times the throughput of ABP



GBN has higher throughput than ABP. However the throughput is decreasing as delta increases.



GBN has slightly higher throughput than ABP since the error rate has gone up

GBN COMPARISON WITH ABP

In most GBN cases, the throughput was a lot higher than ABP (almost 4 times, given that the buffer size had increased from 1 to 4). However, unexpectedly, for 2tau = 10ms and BER = 1e-5, the results were corrupted (which could not be fixed after numerous attempts). The throughput of GBN is higher because, in contrast to ABP, GBN can continuously transmit packets without waiting on timeouts to occur. With timeouts, the ABP throughput considerably decreases. Since there is a larger buffer, the GBN system can transmit packets, while receiving error ACKs, without waiting for timeouts to occur. However, as error rate increased (from BER = 0, to BER = 1e-4), the general trend in GBN transmission was that the throughput decreased (almost approaching ABP values for BER = 1e-4 and 2tau = 500ms). This is because with more error, the GBN system would have to retransmit the entire frame (i.e. in some cases more than 1 packet). This is why the throughput can decrease so much for GBN. However, in practical applications, due to the buffer size being greater than ABP and the multitasking ability of GBN, it is a preferred protocol for wireless transmissions, as it makes good use of transmission bandwidth (in most cases), and transmits a larger number of packets (or looked another way, raw bits) per second.