Network 1 Final Course Project

Head

Dr. Peyman Pahlevani

Teaching Assistant

Kiana Ghassabi

Written By

Solmaz Mohamadi

Sepide Bahrami

IASBS – winter 2019

Institute for Advanced Studies
in Basic Sciences, IASBS

First Phase:
Implementation (Point-to-Point Connection)

In this project, we have used C++ to implement the desired protocols. For each of them, we have considered a specific structure in which we define the needed variables and functions there easily. The main part of the code consists of switch between the protocols that the user can choose in the console.

- Case 1 - Stop and Wait
- Case 2 - Go Back N
- Case 3 - Selective Repeat

By choosing each of them, you are entering into a loop where you can give the input data to send them to the receiver site. Therefore, because of using the fixed-size framing, we divide your entered data-word into frames with five bits in each of them. Here we assume that you are the network layer in sender site, which sends data-word to the data-link layer in sender site to send them to the same level in receiver site.
When each frame is sent, they are immediately delivered to the network layer at the receiver site individually and the layer will be waited for the new frames to be reached out. The sender and receiver site algorithms are written respectively in the mentioned loop you are entering. Right after you do an operation in the sender site algorithm, the while(true) breaks and the loop enters into the receiver site algorithm to handle the operation occurred previously in the sender site.
We did not use files or threads for implementation in both sites, but just two simple and understandable loops, which made our job easier. This explained part is same in each of three protocols. There are just some differences in implementation of each of their algorithms as will be described in the following.

Stop & Wait

In this protocol, we generally must send frames to the receiver and wait for the acknowledgement of the frame to be received at the sender site. If there is a packet loss, the receiver does not get the packet, so there is no acknowledgement for that frame. Then, the time-out happens and the mentioned packet will be sent again. In the receiver site, if an acknowledgement loss occurs, the time-out happens again and sender assumes that receiver did not get the packet. So it sends the desired packet again. However, this time the receiver discard the sent packet because it already exists. In this protocol, there is a long wait before sending the next frame to receiver site. Therefore, it is kind of wasting our bandwidth waiting for just an acknowledgement, which could even be lost, and even the long usual wait gets worse then.

So what happens in our code?

- ❖ Before everything, we choose a random number in which our simulated ack loss will happen in that number (packet loss simulation will be explained further). By calling get_data() function, the data-word you enter will be saved in the sent vector and by calling send_frame(), we send frame to the received vector. By sending, we mean pushing back elements into just a new vector in receiver site. So if a packet loss happens, then we call send_frame_again() and this time we send the correct form of the frame from our stored vector which we saved it right before sending out the frame.
  When the time for the loss of ack comes, the sender site assumes that it did not get the ack and then the time-out happens, the while for receiver breaks and the last sent frame will be sent over again to the receiver site.

Go Back N – Selective Repeat

Both of these protocols are alike each other. The problem with stop and wait was the long wait in sending each frame and so the less utilization in our network. To overcome this problem, there is this idea that while sender is waiting for the ack, it also does something useful and regardless of the ack of sent frame, starts to send other frames continuously. Then, if a packet loss happens and receiver does not get the packet properly here are two approaches:

- ❖ Go Back N protocol says that if there is a packet loss, then I will send the whole window again to the receiver, which is somewhat redundant!
- ❖ Selective Repeat protocol says that if there is a packet loss, then I will just send the corrupted packet again and save the bandwidth.

So what happens in our code?

- ❖ Like the first case, we get the data-word to be sent from user and then cut it into frames and send them one by one to the receiver. Assume that the network is ideal and there is no packet loss. In this case, you go through the sender algorithm and send the first frame to the receiver. After breaking the sender while, you go right into the receiver algorithm and here the receiver sends the acknowledgement. Then again, you go through the sender algorithm. If it was stop and wait, you would be received the ack by now. Nevertheless, here before you receive the ack for the new requested frame, you send that frame already. Because anytime you are in sender algorithm, you can send frames no matter what is happening in the network. Therefore the request_to_send Boolean is always true in these protocols unlike stop and wait. Window size is **4** in both protocols.

- Now assume that there would be a packet loss. In go-back-n protocol we considered a time-out Boolean for the whole window. So if frame 0 is lost, other frames will not be received and each time you are going through the if(corrupted(frame)). When the sequence number is out of window size, the sender will not send any frames and it will change the time_out Boolean to true. Continuing the while, sender algorithm gets through the if(time_out) and here sends the whole window again.

  However, in selective repeat we have four time_out Booleans, as our window size is four. Each time a frame is lost, the receiver extracts exactly which frame it is and sets the equal time_out for that frame to true. When the sequence number is out of window size, the sender will not send any frames and it will change the general time_out Boolean to true. Continuing the while, sender gets through the if(time_out) and according to which time_out has been true, sends that special frame number again to the receiver.

Second Phase: Simulating Packet Loss

To simulate this, we have considered while you are sending data (pushing elements into receiver vector), if the frame number is the one you considered to be corrupted, then we will push some '-' characters instead of the original data-word and in the receiver site there is a function which checks for '-' characters in the received vector. If the function returns true, then there is a packet loss and by sending frame again we correct those '-'s to their original ones and then everything in received vector will look proper. We have specified some special numbers in which the packet loss will happen there.

Third Phase:
Throughput Charts and Analysis

To analyze the throughput we considered the 3%, 10%, 23%…, and 100% in packet loss. Then we calculated the runtime of sender and receiver site algorithms with each percentage to send **30** frames over the network each one carrying **5** bits.

Stop and Wait Throughput Chart

Loss Percentage	Time	Frames per Second	Throughput
3% packet loss	30 frames / 4.61s	6.50 frames / 1s	32.5 bps
10% packet loss	30 frames / 4.97s	6.03 frames / 1s	30.1 bps
23% packet loss	30 frames / 5.57s	5.38 frames / 1s	26.9 bps
33% packet loss	30 frames / 5.82s	5.15 frames / 1s	25.7 bps
43% packet loss	30 frames / 6.06s	4.95 frames / 1s	24.7 bps
53% packet loss	30 frames / 6.52s	4.60 frames / 1s	23.0 bps
63% packet loss	30 frames / 6.73s	4.45 frames / 1s	22.2 bps
76% packet loss	30 frames / 7.36s	4.07 frames / 1s	20.3 bps
93% packet loss	30 frames / 7.97s	3.76 frames / 1s	18.8 bps
100% packet loss	30 frames / 8.19s	3.66 frames / 1s	18.3 bps

As it is shown in the above chart, it is obvious that when there is increase in packet loss, there is a significant decrease in the throughput of our implemented network.

Go-Back-N Throughput Chart

Loss Percentage	Time	Frames per Second	Throughput
3% packet loss	30 frames / 0.60s	50.0 frames / 1s	250.0 bps
10% packet loss	30 frames / 0.60s	50.0 frames / 1s	250.0 bps
23% packet loss	30 frames / 0.80s	37.5 frames / 1s	187.5 bps
33% packet loss	30 frames / 0.80s	37.5 frames / 1s	187.5 bps
43% packet loss	30 frames / 1.37s	21.8 frames / 1s	109.0 bps
53% packet loss	30 frames / 1.42s	21.1 frames / 1s	105.5 bps
63% packet loss	30 frames / 1.50s	20.0 frames / 1s	100.0 bps
76% packet loss	30 frames / 1.50s	20.0 frames / 1s	100.0 bps
93% packet loss	30 frames / 1.70s	17.6 frames / 1s	88.0 bps
100% packet loss	30 frames / 1.93s	15.5 frames / 1s	77.5 bps

We can say that stop and wait is not good enough, and we should be considering some other protocols. Here, we can see that go-back-n protocol is a much better choice than stop and wait as it increased the total throughput of our channel.

- The question is how have this happened?

In stop and wait, sender always waits a specific time to get the acknowledgement from the receiver site. So while waiting, it has no choice and permition of sending more packets and the bandwidth remains useless and unoccupied. However, in go-back-n the sender sends other packets while waiting for the acknowledgement of the last packet and this is the reason for significant increase in throughput and utilization of our implemented network. Just to notice, we have used 100ms wait in stop and wait after sending each frame (consider this as a wait for receiving ack) and 10ms wait for sending a whole window again to the receiver in go-back-n. These waits have affected our numbers in charts.

Selective Repeat Throughput Chart

Loss Percentage	Time	Frames per Second	Throughput
3% packet loss	30 frames / 0.50s	60.0 frames / 1s	300.0 bps
10% packet loss	30 frames / 0.60s	50.0 frames / 1s	250.0 bps
23% packet loss	30 frames / 0.66s	45.4 frames / 1s	227.0 bps
33% packet loss	30 frames / 0.72s	41.6 frames / 1s	208.0 bps
43% packet loss	30 frames / 0.78s	38.4 frames / 1s	192.0 bps
53% packet loss	30 frames / 0.83s	36.1 frames / 1s	180.5 bps
63% packet loss	30 frames / 0.88s	34.0 frames / 1s	170.0 bps
76% packet loss	30 frames / 0.96s	31.2 frames / 1s	156.0 bps
93% packet loss	30 frames / 1.09s	27.5 frames / 1s	137.5 bps
100% packet loss	30 frames / 1.19s	25.2 frames / 1s	126.0 bps

Finally, we can again think of improving our utilization while having many packet losses. The policy in go-back-n was to send the whole window again if there was packet loss. However, with selective repeat, we can decrease the delay in sending the whole window again and just send the required frame in the window. Here we can see that from 3% to 33% loss in packets, both go-back-n and selective repeat act alike each other and there is not that much difference between both of them. But when there is increase in packet loss, selective repeat acts much better because it does not have the delay for sending whole window. There is a lot of difference in throughput with 43% loss between two protocols.

Conclusion of this part

In conclusion, we can say that stop and wait is not a wise choice for even a simple network because you are wasting time and that is not acceptable in today's world pace. If we have a good estimation of the packet loss percentage, if it is less than 33%, it is better to use go-back-n rather than selective repeat. Because the software implementation is easier and its complexity is better. Nevertheless, if we assume that there will be many losses of packets, the better choice is to use selective repeat protocol.

Fourth Phase: Modifications
Multipoint Connection and Change in Frame Size

Multipoint Connection:

In the first phase, we implemented all three protocols with the assumption of being just one receiver at destination. Here we thought about having at least two receivers in our network. Generally, we can say that when we are adding more receivers, we are adding complexity and also redundancy to our network. In our code, we doubled every variables we wrote before in first phase. Like Sn, Rn, Booleans, etc. because if we had one from each of them, inconsistency would rise in network. For example, receiver 1 gets the frame 0 and changes Rn to 1. Receiver 2 does not get the frame 0 and changes Rn to 0. So now, here the sent frame would be sent again to the receiver 1. To avoid this happening for each of our receivers we must have specific variables to use which is redundancy. In addition, there would be complexity in both sender and receiver site algorithms. What we have considered in code is that for example, sender sends frame 0 to receiver 1, on the following it goes through a same if() condition to send frame 0 to receiver 2. In the receiver site, it first delivers the frame to network layer 1 and then network layer 2. Because our main part consist of a loop for algorithms, so we should have multiple if() conditions if we are supporting multi-receivers. In addition, there is another assumption we have made and it is that we considered the bandwidth high enough that we can send both frames to both receivers once in a time, somewhat parallel, and there is no delay for a frame to be sent after another frame has been sent already. We have the same consideration for sending acknowledgements through the channel.

Change in Frame Size:

In the first phase, our frame size was 5 and here we changed it to 10. It was such an easy job and we just changed some variables in our iterations in used vectors.

Fifth Phase:
Throughput Charts and Analysis After Modifications

To analyze the throughput, again we considered the 3%, 10%, 23%..., and 100% in packet loss. Then we calculated the runtime of sender and receiver site algorithms with each percentage to send **30** frames to each receiver (total number of **60** frames per each sending) over the network each one carrying **10** bits. The loss percentage, which is mentioned, happens respectively for both receiver 1 and receiver 2 in network.

Stop and Wait Throughput Chart

Loss Percentage	Time	Frames per Second	Throughput
3% packet loss	60 frames / 9.59s	6.25 frames / 1s	62.5 bps
10% packet loss	60 frames / 10.3s	5.82 frames / 1s	58.2 bps
23% packet loss	60 frames / 11.0s	5.45 frames / 1s	54.5 bps
33% packet loss	60 frames / 11.8s	5.08 frames / 1s	50.8 bps
43% packet loss	60 frames / 12.4s	4.83 frames / 1s	48.3 bps
53% packet loss	60 frames / 13.0s	4.61 frames / 1s	46.1 bps
63% packet loss	60 frames / 14.3s	4.19 frames / 1s	41.9 bps
76% packet loss	60 frames / 15.0s	4.00 frames / 1s	40.0 bps
93% packet loss	60 frames / 16.9s	3.55 frames / 1s	35.5 bps
100% packet loss	60 frames / 17.2s	3.48 frames / 1s	34.8 bps

Here we see that stop and wait is the worst choice for either point-to-point or multi-point connection. As we increase the size of our frame, it is obvious that we can send less frames per second. With frame size = 5 we could send more frames. Here even with having parallelism and sending twice of each frame, yet frames per second is less than previous chart for stop and wait. The increase in throughput is again because of having two receivers and sending frames concurrently. If it were just one receiver, throughput would be half of its current value. Having big size framing is like a bus, which has occupied a long space in a highway, where there can be smaller cars instead of the bus to have more utilization.

Go-Back-N Throughput Chart

Loss Percentage	Time	Frames per Second	Throughput
3% packet loss	60 frames / 1.39s	43.1 frames / 1s	431 bps
10% packet loss	60 frames / 1.42s	42.2 frames / 1s	422 bps
23% packet loss	60 frames / 1.46s	41.0 frames / 1s	410 bps
33% packet loss	60 frames / 1.48s	40.5 frames / 1s	405 bps
43% packet loss	60 frames / 1.76s	34.0 frames / 1s	340 bps
53% packet loss	60 frames / 1.79s	33.5 frames / 1s	335 bps
63% packet loss	60 frames / 1.80s	33.5 frames / 1s	335 bps
76% packet loss	60 frames / 2.01s	29.5 frames / 1s	295 bps
93% packet loss	60 frames / 2.03s	29.5 frames / 1s	295 bps
100% packet loss	60 frames / 2.10s	28.5 frames / 1s	285 bps

Here we see that again go-back-n is performing much better than stop and wait and the channel throughput is almost seventh time more, which is more utilization and not wasting the bandwidth. Yet frames per second component is less than the one with frame size = 5 and the reason is what we have talked in the previous page.

So let us have a look at the selective repeat chart in the next page and have a final conclusion of what have been done in this project.

Selective Repeat Throughput Chart

Loss Percentage	Time	Frames per Second	Throughput
3% packet loss	60 frames / 1.37s	43.7 frames / 1s	437 bps
10% packet loss	60 frames / 1.40s	42.8 frames / 1s	428 bps
23% packet loss	60 frames / 1.46s	41.0 frames / 1s	410 bps
33% packet loss	60 frames / 1.51s	39.7 frames / 1s	397 bps
43% packet loss	60 frames / 1.59s	37.7 frames / 1s	377 bps
53% packet loss	60 frames / 1.67s	35.9 frames / 1s	359 bps
63% packet loss	60 frames / 1.71s	35.0 frames / 1s	350 bps
76% packet loss	60 frames / 1.81s	33.1 frames / 1s	331 bps
93% packet loss	60 frames / 1.82s	32.9 frames / 1s	329 bps
100% packet loss	60 frames / 1.90s	31.5 frames / 1s	315 bps

Conclusion of this part

Here it is like what happened in point-to-point connection. Until 33% of packet loss, there is not much difference between go-back-n and selective repeat. In 43% suddenly selective repeats performs much better and happens to be our suggested choice for network implementation.

If we try to have less frame size, we can send more frames per second and it is good while there is packet loss, because we are going to send again a smaller frame rather than a big one, which has more delay.

Citation

We get the idea of having the algorithms respectively in main part of the code with the help of this code on the link below.

https://gist.github.com/Zerk123/bbd8283a7cf0442d0fedfecb47272dd4