# 컴퓨터네트워크

# Bidirectional Go-Back-N

**강의 시간: 화 5, 목 6**

**교수님: 이혁준 교수님**

**학번: 2022202065**

**이름: 박나림**

**학과: 컴퓨터정보공학부**

**제출 일자: 2024 년 05 월 17 일**

# Introduction

이번 프로젝트는 Reliable Transport Protocol 을 구현하는 것이 목표이다. 그 중에서도 Go-Back-N 의 단방향 방식을 구현한 다음, 최종적으로 양방향을 구현하도록 하는 것이다. 먼저 단방향에서는 A-side 와 B-side 로 나누어져서 각각의 함수들이 작동하는데 구현은 하나의 프로그램에서 이루어지도록 통합시켜야 한다. 이때 A 가 packet 을 전송하는 sender, B 가 packet 을 수신하는 receiver 가 된다. 양방향에서는 A, B 모두 sender 와 receiver 의 기능을 가질 수 있어야 한다.

Upper layer 인 Layer5 에서 data 가 내려온다. 이 data 는 메시지 형태로 되어있으며, 해당 메시지를 받아 packet 에 저장하게 된다. 이러한 packet 은 구현할 함수들을 거쳐 tolayer3(); 함수를 이용하여 네트워크로 전송된다. 이때 네트워크에서는 랜덤 확률로 loss 나 corrupt 가 발생하도록 설정되어 있다. 전송된 packet 은 B-side 로 넘어가서 구현할 함수들을 거친 뒤 최종적으로 B 쪽의 layer5 로 올려 보내지게 된다.

구현할 함수들은 다음과 같다.

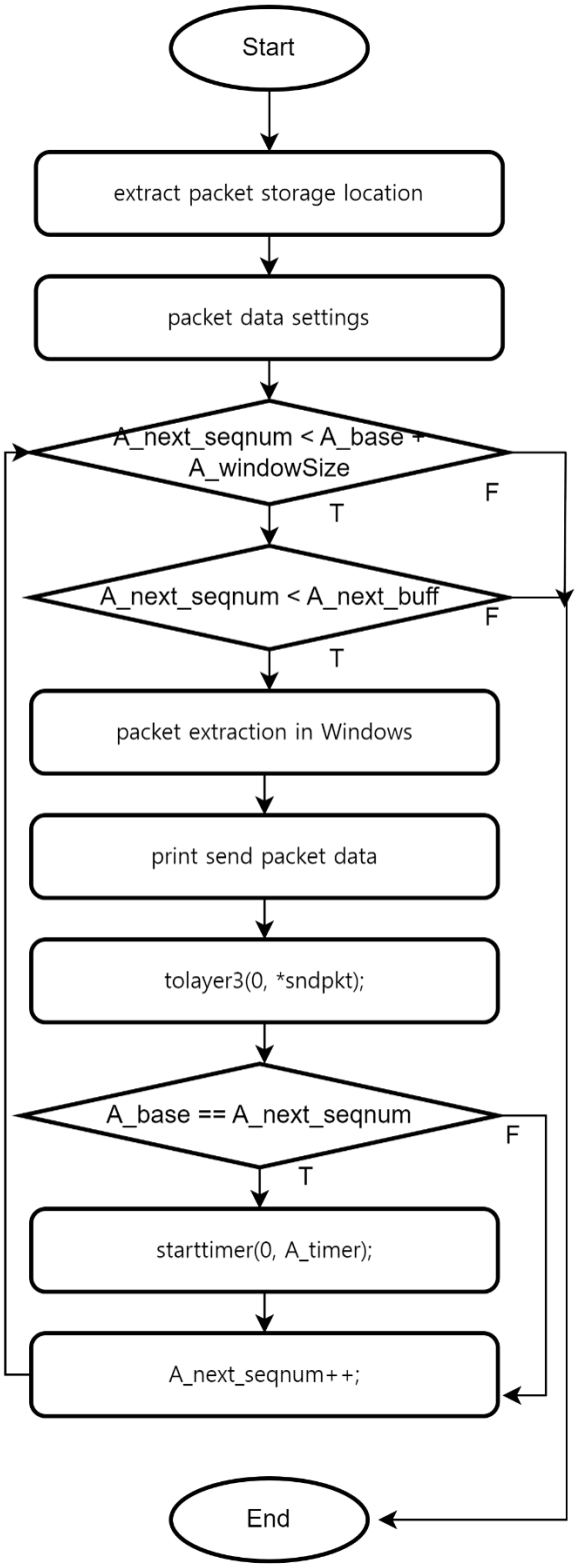
1. A_output(): 상위 레이어에서 메시지를 보낼 때 호출되는 함수로, 데이터를 순서대로 네트워크로 보내도록 구현한다.
2. A_input(): 수신 측에서 보낸 패킷이 도착했을 때 호출되며, 오류에 대한 처리를 한다.
3. A_timerinterrupt(): A 의 타이머가 만료될 시 호출되며, 패킷을 재전송 시킨다.
4. A_init(), B_init(): 각각의 변수들을 초기화 시킨다.
5. B_input(): A 에서 보낸 패킷이 도착했을 때 호출되는 함수로, 오류 처리를 한다. 정상적일 경우 ACK 를 보낸다.

위 내용은 단방향 기준이며, 양방향일 시 각 함수마다 sender 와 receiver 의 기능을 모두 할 수 있도록 통합시킨다.

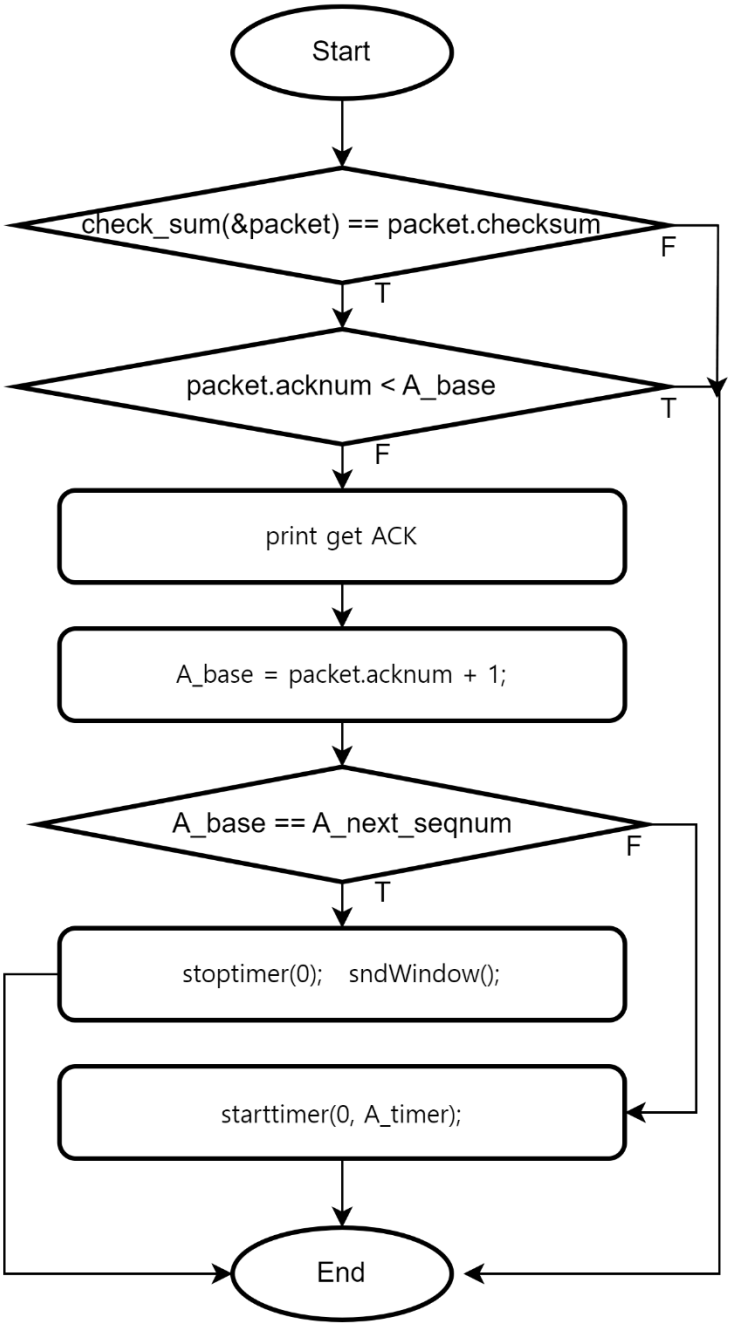
추가적으로, 패킷의 오류를 검사하는 checksum 함수를 구현할 때 8-bit 또는 16-bit 1's complement 로 계산되도록 만들어야 한다. 또한 Piggyback 방식으로서 packet 에 ACK 를 같이 실어 보내도록 해야 한다.

# Flow chart

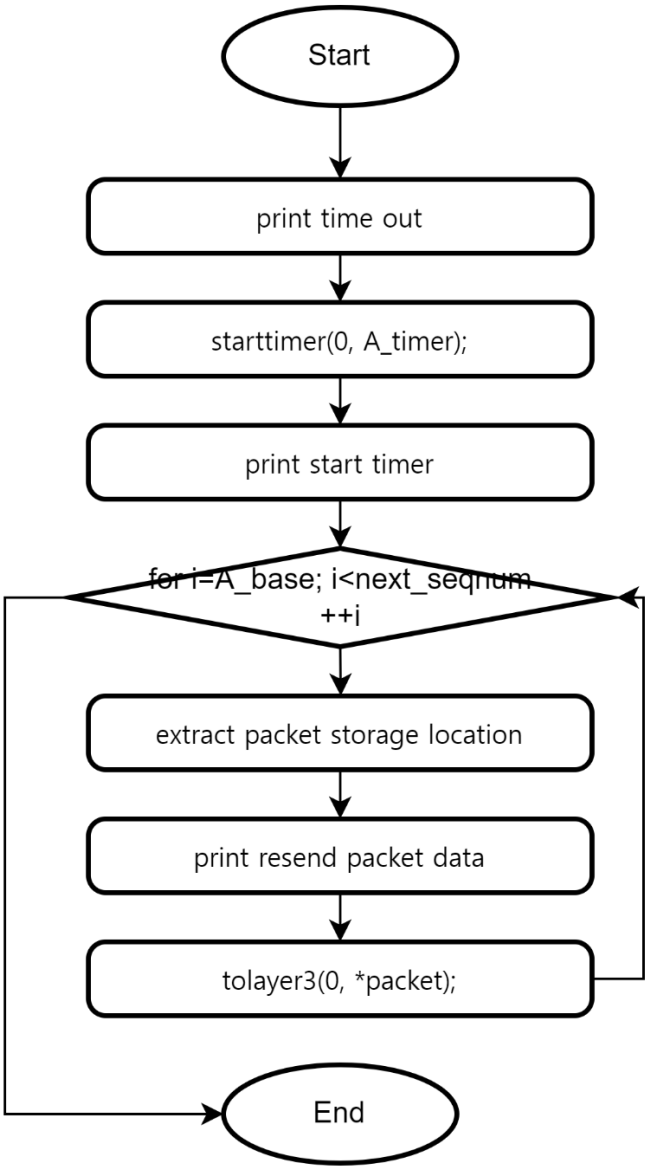
**A_output()**



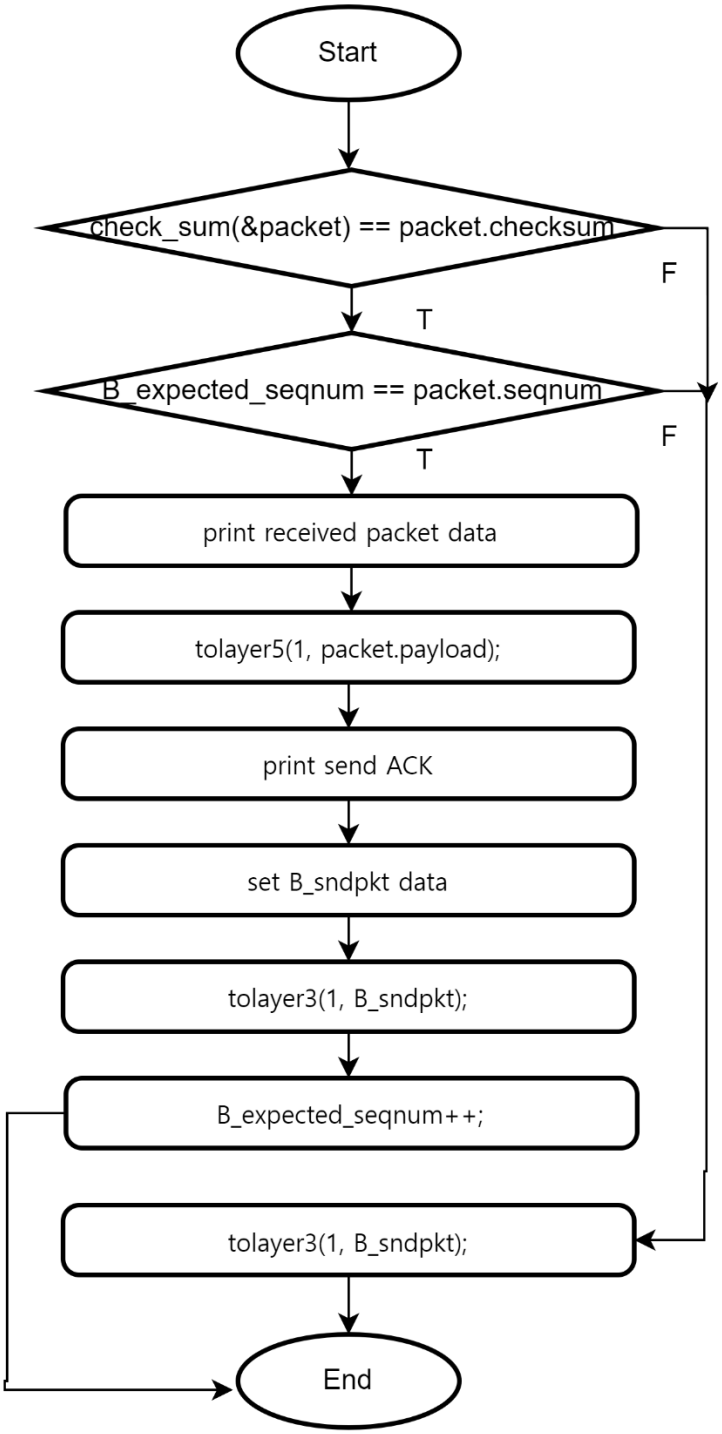
**A_input()**


Start
check_sum(&packet) == packet.checksum
F
T
packet.acknum < A_base
T
F
print get ACK
A_base = packet.acknum + 1;
A_base == A_next_seqnum
F
T
stoptimer(0); sndWindow();
starttimer(0, A_timer);
End

**A_timerinterrupt()**



**B_input()**


Start
check_sum(&packet) == packet.checksum
F
T
B_expected_seqnum == packet.seqnum
F
T
print received packet data
tolayer5(1, packet.payload);
print send ACK
set B_sndpkt data
tolayer3(1, B_sndpkt);
B_expected_seqnum++;
tolayer3(1, B_sndpkt);
End

# Code Description

단방향 GBN 으로 구현한 코드는 다음과 같은 과정을 수행한다.

먼저, layer5 에서 주어지는 data 를 A_output()에서 받는다. 구조체로 되어있는 패킷의 포인터를 생성하여 window(buffer)의 인덱스를 주소를 저장한다. 이때 인덱스는 Max Buffer Size 로 나눈 나머지로 계산된다. 그 후 패킷의 정보들을 저장한 뒤 인덱스를 증가시키고 sndWindow()를 통해 패킷을 네트워크로 전송시킨다. 이때 윈도우에 있는 패킷들을 보내며, 출력을 위해 다시 패킷의 위치를 계산하게 된다. 또한 모든 패킷을 보냈다면 starttimer()로 타이머를 재시작시킨다. 만약 next_seqnum 이 next_buff 인덱스보다 크거나 같다면 전송을 종료한다.

네트워크로 보내진 패킷은 B_input()으로 받는다. 우선적으로 받은 패킷의 오류를 검사하기 위해 Check_sum()의 결과가 패킷의 checksum 과 동일할 때만 수행한다. 만약 같지 않다면 corrupted 된 상황이기 때문이다. 두번째로 expected_seqnum 이 패킷의 seqnum 과 같은지 비교한다. 다르다면 마찬가지로 예상 seqnum 이 다르기 때문에 종료된다. 제대로 데이터를 받은 경우에는 해당 결과를 출력한 뒤 layer5 로 패킷의 데이터만 올려보낸다. 그 후 결과를 전달하기 위해 B 패킷의 acknum 에 expected_seqnum 을 저장한다. checksum 도 설정한 뒤 이 패킷을 다시 layer3 인 네트워크로 내려보낸다.

위 결과 패킷은 A_input()애서 받는다. 마찬가지로 패킷의 checksum 을 검사하여 corrupted 여부를 확인한 뒤 패킷의 acknum 이 base 보다 큰지 확인한다. 만약 작다면 위에서 에러가 발생하여 NAK 가 보내진 상황이므로 종료시킨다. 제대로 ACK 를 받은 경우에는 base 를 acknum+1 로 업데이트 시킨다. 이때 base 가 next_seqnum 과 같다면 window 내의 패킷을 전부 보냈다는 뜻이므로 타이머를 멈추고 sndWindow()를 통해 그 다음 패킷들을 전송시킨다. base 위치가 같지 않다면 아직 window 내에 패킷들이 남아있는 상태이므로 starttimer()로 타이머를 재시작한다.

만약 어떠한 패킷을 전송한 뒤 타이머가 끝날 때까지 ACK 를 받지 못한다면, A_timerinterrupt()가 호출된다. 그러면 먼저 starttimer()로 타이머를 재시작 시킨 뒤 for 문으로 base 부터 next_seqnum 전까지 window 에 남아있는 패킷을 전부 네트워크로 재전송 시킨다.

# Result & Analysis

5 개의 메시지 개수로 시간을 충분히 주었을 때 나온 결과이다. 패킷을 보내면 B_input 에서 패킷을 받고 ACK 를 보낸다. ACK 는 seqnum 과 동일한 번호이다. 시간이 충분하므로 A_input 에서 바로 ACK 를 받을 수 있다. 이러한 과정이 5 번 반복된다.

```
Enter the number of messages to simulate: 5
Enter  packet loss probability [enter 0.0 for no loss]:0.00001
Enter packet corruption probability [0.0 for no corruption]:0.00001
Enter average time between messages from sender's layer5 [ > 0.0]:10
Enter TRACE:1
===send packet (seq=1): aaaaaaaaaaaaaaaaaaa
B_input: received packet (seq=1): aaaaaaaaaaaaaaaaaaa
B_input: send ACK (1)
A_input: get ACK (1)
===send packet (seq=2): bbbbbbbbbbbbbbbbbbb
B_input: received packet (seq=2): bbbbbbbbbbbbbbbbbbb
B_input: send ACK (2)
A_input: get ACK (2)
===send packet (seq=3): ccccccccccccccccccc
B_input: received packet (seq=3): ccccccccccccccccccc
B_input: send ACK (3)
A_input: get ACK (3)
===send packet (seq=4): ddddddddddddddddddd
B_input: received packet (seq=4): ddddddddddddddddddd
B_input: send ACK (4)
A_input: get ACK (4)
===send packet (seq=5): eeeeeeeeeeeeeeeeeee
 Simulator terminated at time 56.140324
 after sending 5 msgs from layer5
```

같은 조건에서 시간만 줄이게 되면 아래와 같은 결과가 나타난다. 첫번째로 패킷을 보내면, B_input 에서 받은 다음 ACK 를 보낸다. 그리고 ACK 가 도착하기 전에 A 에서 두번째 패킷을 보낸다. 그 뒤 첫번째 패킷에 대한 ACK 를 받게 된다. B_input 에서는 두번째 패킷을 받은 뒤 ACK 를 보낸다. 마찬가지로 ACK 가 도착하기 전에 A 에서 세번째 패킷을 전송한 뒤, 그리고나서 두번째 ACK 를 받게 된다. 그 후 B_input 에서 세번째 패킷을 받은 뒤 ACK 를 전송한다. 이때 타이머가 만료되어 A_timerinterrupt 가 호출된 것을 볼 수 있다. 타이머를 재시작 시키면서 ACK 를 받지 못한 세번째 패킷부터 재전송 시킨다. 그 후 다시 네번째 패킷을 보내고나서 세번째 ACK 를 받는다. 이 과정이 반복되며 끝이난다.

```
Enter the number of messages to simulate: 5
Enter  packet loss probability [enter 0.0 for no loss]:0.00001
Enter packet corruption probability [0.0 for no corruption]:0.00001
Enter average time between messages from sender's layer5 [ > 0.0]:5
Enter TRACE:1
===send packet (seq=1): aaaaaaaaaaaaaaaaaaa
B_input: received packet (seq=1): aaaaaaaaaaaaaaaaaaa
B_input: send ACK (1)
===send packet (seq=2): bbbbbbbbbbbbbbbbbbb
A_input: get ACK (1)
Warning: attempt to start a timer that is already started
B_input: received packet (seq=2): bbbbbbbbbbbbbbbbbbb
B_input: send ACK (2)
===send packet (seq=3): ccccccccccccccccccc
A_input: get ACK (2)
Warning: attempt to start a timer that is already started
B_input: received packet (seq=3): ccccccccccccccccccc
B_input: send ACK (3)
A_timerinterrupt: time out!
A_timerinterrupt: start timer
A_timerinterrupt: resend packet (seq=3): ccccccccccccccccccc
===send packet (seq=4): ddddddddddddddddddd
A_input: get ACK (3)
Warning: attempt to start a timer that is already started
===send packet (seq=5): eeeeeeeeeeeeeeeeeee
 Simulator terminated at time 22.354229
 after sending 5 msgs from layer5
```

10 개의 패킷으로 중간에 lost 나 corrupted 가 발생한 상황일 때의 결과이다. 초기에는 정상적으로 패킷을 주고받다가, 5 번째 패킷에 대한 ACK 가 도착하기 전에 lost 가 발생하였다. 그리고 여섯번째 패킷을 보내고 나서 timeout 이 발생한다. ACK 를 못 받은 5 번째 패킷부터 다시 재전송하여 5, 6 이 전송되는데 5 번째에서 이번엔 corrupted 가 발생하게 된다. 그렇게 되면 B_input 에서는 6 번째 패킷을 받아서 ACK 를 전송하게 된다. 여기서 다시 corrupted 가 발생하면 A 에서는 corrupted packet 을 받았다는 안내문이 출력된다. 그 외의 ACK 에 대해서는 수신 받음을 출력하며 또다시 타이머가 만료되면 ACK 를 받지 못한 패킷들에 대해 다시 재전송 시킨다. 이 과정을 반복한다.

```
Enter the number of messages to simulate: 10
Enter  packet loss probability [enter 0.0 for no loss]:0.1
Enter packet corruption probability [0.0 for no corruption]:0.1
Enter average time between messages from sender's layer5 [ > 0.0]:10
Enter TRACE:1
===send packet (seq=1): aaaaaaaaaaaaaaaaaaa
B_input: received packet (seq=1): aaaaaaaaaaaaaaaaaaa
B_input: send ACK (1)
A_input: get ACK (1)
===send packet (seq=2): bbbbbbbbbbbbbbbbbbb
B_input: received packet (seq=2): bbbbbbbbbbbbbbbbbbb
B_input: send ACK (2)
A_input: get ACK (2)
===send packet (seq=3): ccccccccccccccccccc
B_input: received packet (seq=3): ccccccccccccccccccc
B_input: send ACK (3)
A_input: get ACK (3)
===send packet (seq=4): ddddddddddddddddddd
B_input: received packet (seq=4): ddddddddddddddddddd
B_input: send ACK (4)
A_input: get ACK (4)
===send packet (seq=5): eeeeeeeeeeeeeeeeeee
B_input: received packet (seq=5): eeeeeeeeeeeeeeeeeee
B_input: send ACK (5)
          TOLAYER3: packet being lost
```

```
===send packet (seq=6): ffffffffffffffffff
A_timerinterrupt: time out!
A_timerinterrupt: start timer
A_timerinterrupt: resend packet (seq=5): eeeeeeeeeeeeeeeeee
          TOLAYER3: packet being corrupted
A_timerinterrupt: resend packet (seq=6): ffffffffffffffffff
B_input: received packet (seq=6): ffffffffffffffffff
B_input: send ACK (6)
          TOLAYER3: packet being corrupted
===send packet (seq=7): gggggggggggggggggg
A_input: Corrupted packet.
A_input: get ACK (6)
Warning: attempt to start a timer that is already started
A_timerinterrupt: time out!
A_timerinterrupt: start timer
A_timerinterrupt: resend packet (seq=7): gggggggggggggggggg
A_input: get NAK (6)
===send packet (seq=8): hhhhhhhhhhhhhhhhhh
          TOLAYER3: packet being corrupted
B_input: received packet (seq=7): gggggggggggggggggg
B_input: send ACK (7)
===send packet (seq=9): iiiiiiiiiiiiiiiiii
A_input: get ACK (7)
Warning: attempt to start a timer that is already started
A_timerinterrupt: time out!
A_timerinterrupt: start timer
A_timerinterrupt: resend packet (seq=8): hhhhhhhhhhhhhhhhhh
```

```
A_timerinterrupt: resend packet (seq=9): iiiiiiiiiiiiiiiiii
          TOLAYER3: packet being corrupted
A_input: get NAK (7)
          TOLAYER3: packet being lost
B_input: received packet (seq=8): hhhhhhhhhhhhhhhhhh
B_input: send ACK (8)
          TOLAYER3: packet being lost
A_input: get NAK (7)
===send packet (seq=10): jjjjjjjjjjjjjjjjjj
          TOLAYER3: packet being lost
 Simulator terminated at time 109.668350
 after sending 10 msgs from layer5
```

# Consideration

이번 프로젝트를 진행하면서, 최종적으로 양방향을 구현하지 못해 아쉬웠다. 처음에는 양방향 GBN 을 구현하는 것을 목적으로 A 와 B 가 같이 sender 와 receiver 의 기능을 모두 수행하도록 output 과 input, timer 를 서로 동일하게 코드를 작성하였다. 하지만 그렇게 하니 코드를 실행했을 때 sender 의 기능만 수행되고 receiver 의 기능은 수행하지 못하였다. A 와 B 의 input()함수가 문제인 것 같아서 코드를 라인별로 동작시켜 보았을 때, if(expected_num==seqnum)부분이 실행되지가 않아서 ACKstate 가 계속 0 인채로 남아 ACK 가 전달되지 않았다. expected_num 이 제대로 업데이트가 되지 않는 문제라 생각하여 output()의 함수도 다시 재실행을 시켜보았으나, ACKstate 도 업데이트가 되지 않아 전체적으로 코드 작성에 어려움이 있었다. 따라서 처음부터 양방향을 구현하지 말고 단방향부터 구현해야겠다고 생각하여 해당 코드를 작성하였다.

단방향에서는 먼저 A_output 에서 한번에 패킷을 전송하는 방식으로 구현하였는데, 이후 A_input 에서도 ACK 를 수신하였을 때 window 를 업데이트하고 다음 패킷을 보내는 과정에서 해당 과정이 다시 필요하다고 느껴 전송하는 함수를 따로 만들었다. 이 함수에서는 while 문으로 반복하며 패킷을 네트워크로 전송하도록 하였는데 처음에는 단순히 seqnum 이 base+windowSize 보다 작을 때만 반복하도록 만들었더니 코드를 실행시켰을 때 무한반복으로 인해 오류가 발생하였었다. 이에 버퍼 인덱스를 확인하는 과정을 추가하여 다시 실행하였을 땐 정상적으로 실행이 될 수 있었다. 또한 패킷의 정보를 저장하는 과정에서도, 원래는 포인터로 하지 않고 단순히 구조체를 불러와서 정보를 저장하는 방식으로 진행하였는데 그렇게 하니 결과를 출력할 때 버퍼가 이상하게 출력되는 현상이 나타났다. 마찬가지로 seqnum 이 제대로 저장되지 않아서 패킷을 한번에 다 보낼 때까지 ACK 를 받을 수 없었다. 그래서 포인터로 window 의 주소를 저장한 다음 정보를 저장하도록 하였더니 정상적으로 결과가 출력될 수 있었다.

이번 프로젝트에서는 단방향 코드만 작성했지만, 프로젝트를 진행하면서 양방향, 단방향에 대해 둘 다 자세한 공부를 할 수 있었다. Go-Back-N 프로토콜의 특징도 생각하며 구현을 하니 더 잘 이해할 수 있었던 것 같다.