컴퓨터 네트워크

강의시간:월 13:30~ 14:45

수 12:00~13:15

교수님:이혁준 교수님

학번:2018202074

이름:김상우

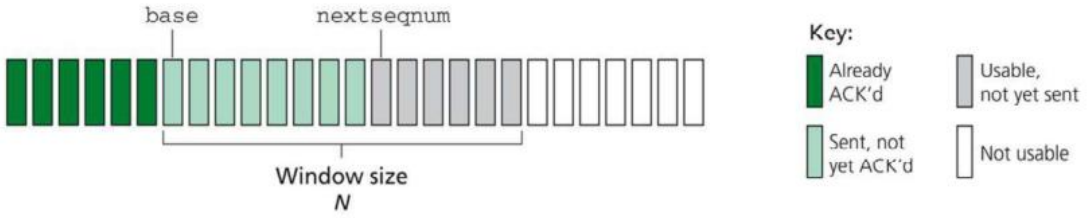
소속:컴퓨터정보공학부

제출일:2022:06:03

서론:

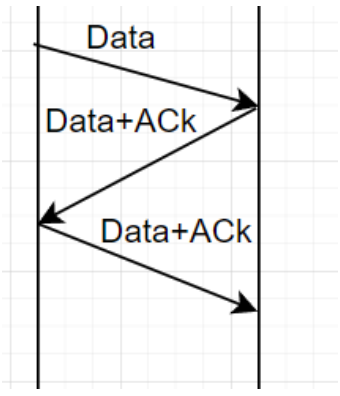
해당 프로젝트는 양방향 GO BACK N을 구현하는 것을 목적으로 하고 있습니다. 해당 프로젝트를 하기에 앞서 GO-BACK-N, PIGGYBACK, 그리고 check sum에 대한 이해가 필요했습니다.

GO-BACK-N은 데이터를 보내고 잘 도착했다면 이에 대한 ACK를 다시 받는 것을 바탕으로 설계되어 있습니다. 여기서 ACK란, sender로부터 reciever가 데이터를 받은 후 확인을 위해 재전송하는 값입니다. 이때 GO-BACK-N의 경우 window를 지정, base와 next sequnum을 이용해 데이터 전송 상태를 표시합니다. 이미 ACK된 경우 window보다 작은 값을 가지게 되고, 보낸 후 ACK가 되지 않은 경우 base이상, nextseqnum 미만의 값을, next seqnum이상이면 보내야 하는 경우 아직 값을 보내지 않은 것으로 분류하여 진행하게 됩니다.



이때, 전송하는 ACK의 값을 receive를 통해 구해지는 expect 값을 기반으로 계속 변경해주게 된다.

해당 프로젝트에선 Piggyback을 기반으로 진행합니다. PiggyBack이란 수신된 데이터에 대한 ACK를 받자마자 보내는 것이 아리나 data를 전송하기 위한 output상황에서 누적 ACK를 포함해 전송 상태의 확인을 병행하는 방식입니다. 단, 실제의 경우에선 한 쪽의 output이 끝나는 시점에서 반대 쪽은 ACK, 즉 전송상태의 여부를 확인할 수 없어 예외처리를 해주어야 합니다.(해당 프로젝트에서는 이를 고려하지 않습니다.)



(처음엔 ACK없이 전송후(해당 프로젝트에선 999로 설정), 이에 대한 ACK가 상대방의 Data와 함께 받아오게 된다.)

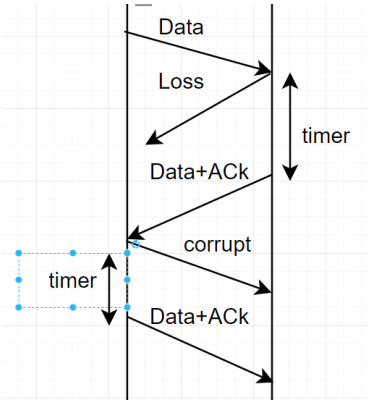
CheckSum의 경우 packet의 elements, 즉 seqnum, acknum, checksum, 그리고 payload를 더하고 이를 1의 보수를 취한 후 wrap around carry까지 더해 0인지 아닌지, 즉 corrupt, 손상이 일어났는지 확인하기 위해 존재합니다. 이 때, payload의 경우 16bit로의 계산을 위해 0을 포함한 짝수 번째의 char에대해 256(2^8)을 곱하고 홀수 번째의 char에 대해서는 그대로 더해주었습니다. 이후 checksum을 65536(2^16)으로 나눠 wrap around carry를 구하고 이를 기존값과 더한 후 ~를 통해 1의 보수를 취하였습니다.

위 프로젝트를 진행함에 있어 다음과 같은 상황에 주의해야 한다.

Loss: packet이 전송되다 소실되는 것으로 전송은 했으나 packet이 reciever에 도착하지 못하는 경우이다.

Corrupt: packet의 담긴 내용이 변질되는 경우로, 원래 보내려던 값과 다른 값이 되는 경우이다.

이들은 timer를 통해 sender에서 해당 packet들을 재전송하게 해주어 해결이 가능하다.



-전체 시스템에 대한 설명 & 데이터 구조 도식화 및 설명

서론의 내용을 바탕으로 필요한 기능을 정리하여 전체시스템을 표현하면 다음과 같습니다.

전송할 data를 제공하고 전송받은 data를 저장할 layer5

data와 ack 등이 담긴 packet이 전송되기 위해 양 side와 연결된 layer3

함수들을 처리하여 layer5와 layer3의 중간에서 이들의 소통을 도와줄 layer4.

서론의 내용을 바탕으로 필요한 변수를 제시하면 다음과 같습니다.

Base:전송 후 ACK를 받아 네트워크 소통이 다 진행된 packet표현을 위한 변수

Exp_seqnum: nak 확인을 위해 다음에 올 것이라 예상되는 seq의 값을 저장하기 위한 변수

Next_seqnum: 전송 후 아직 ack가 오지 않은 경우를 확인하기 위한 변수

Buffend:packet에 대한 buffer가 가득 찼는지 확인하기 위한 변수

ACKstate:output의 경우 ACK가 포함되어야 하는지 확인하기 위한 변수

ACKnum:누적 ACK를 저장하기 위한 변수

이들은 초기에 ACK를 전송하지 않으며 누적 ACK가 없기에 ACK state와 ACKnum은 0으로, 이 base와 Next_seqnum, exqnum, buffend는 첫 packet output에 대해 buffer의 첫번째를 사용하고 ack1을 기대한다는 점에서 1로 초기화할 수 있다.

추가적으로 재전송 시 Packet을 저장할 Buffer가 필요할 것입니다.

이때 layer 4는 이들을 조율하기 위한 함수들을 지녀야 하며 구현을 위해 필요한 기능들은 다음과 같을 것입니다.

기능을 위해 side를 초기화 해줄 함수(A_init)

Layer5로부터 data를 받아 packet을 생성(A_output) 후 layer를 통해 전송(tolayer3)

(위 경우 piggyback이므로 packet생성시 ack상태 추가)

layer 3로부터 data를 받아(A_input) layer5에 전송(tolayer5)

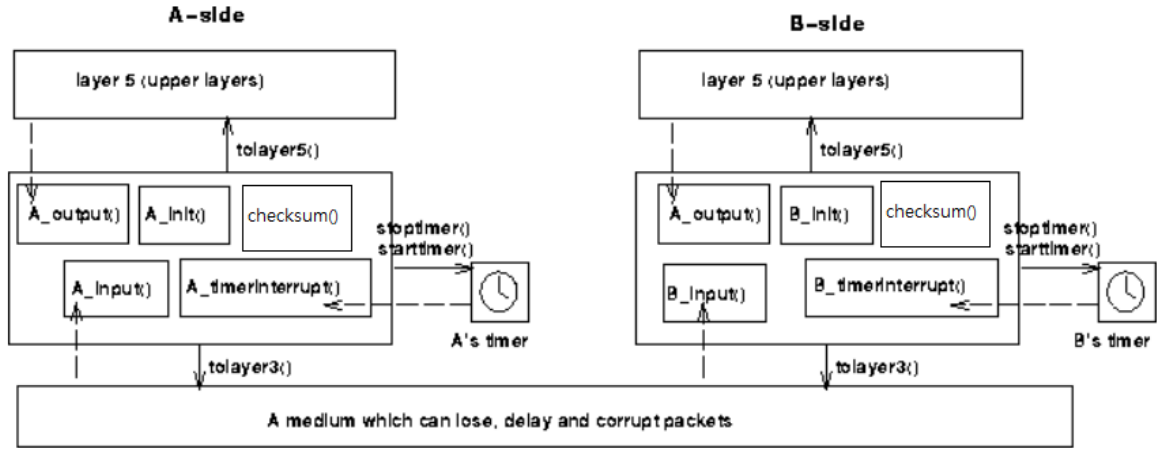
layer3로부터 ack를 받아(A_input) 전송 상태 확인

재전송을 위해 timer의 사용여부를 확인할 함수(stoptimer/starttimer)

Timer로 인해 작동할 함수(A_timerinterrupt)

corrupt여부를 확인하기 위해 checksum을 진행할 함수(checksum)

위의 함수들을 바탕으로 도식화를 진행할 시 다음과 같이 그려지게 될 것입니다.



(제안서에 제시된 단방향 형태의 Go-BACK-N을 수정한 모습이다.)

저희가 구현해야 할 전체 시스템은 위와 같습니다. 2개의 side로 구분되며 이 둘은 layer3를 통해 서로 데이터와 ACK를 주고 받게 됩니다. 이때 주는 것은 함수 tolayer3 사용하게 되며 받는 것은 input함수를 통해 받게 됩니다. input를 통해 얻어지게 된 data는 tolayer5 함수를 통해 layer5로 옮겨져 저장됩니다. Tolayer3의 경우 보내는 data는 output에 의해 layer5로부터 받게 되며 ACK는 input함수를 통해 계속해 변경되는 누적 ACK를 기반으로 구하게 됩니다. 이 과정에서 loss 및 corrupt등의 이유로 인해 재전송이 필요한 경우를 대비, timer를 이용하게 됩니다. 이를 위해 output시에는 새로운 packet에 대해 tolayer3사용후 timer를 시작하게 되며 input에서는 맞는 값을 받을 시 timer정지, 않을 경우 timer를 재개하는 방식을 사용해 timerinterrupt를 제어하며 정확한 네트워크 소통을 지원합니다. Timerinterrupt의 경우 base부터 next seqnum 미만까지 데이터는 전송했지만 ACK를 받지 못한 packet들이 포함된 구간임을 이용, 이 구간의 packet을 tolayer3를 통해 재전송하는 함수입니다.

-스켈레톤 코드로 제공되는 영역 중 구현에 영향을 주는 부분

위 경우, side를 오가게 되는 packet의 변수들은 다음과 같은 목적을 위해 사용될 것입니다.

Seqnum: recieve이후 예측 값 일치 여부 확인

Acknum: sender에 전송 여부 확인을 위해 전송

Payload: layer5로부터 받게 되는 data저장

Checksum: packet element값들을 바탕으로 생성하여 corrupt여부 확인

-동작원리 서술 및 각각 함수에 대한 설명

위에서 설명한 내용을 바탕으로 구현 시 발생하는 동작은 다음과 같이 설명할 수 있다.

0. 변수를 초기화
1. 처음 ACK를 포함하지 않고(ACKnum=999) data만을 전송 (sender)
2. ACK와 data를 전송 (sender)
3. data만이 포함된 packet을 받는 경우 (reciever)
4. ACK와 data가 포함된 packet을 받는 경우 (both)
5. 일정시간동안 보낸 packet에 대해 ack가 오지 않는 경우의 재전송 (sender)

위를 각각 동작0,1,2,3,4,5라고 지칭하며 다음이 어떻게 동작되어야 하는지 확인한다.

동작 0:각 side(A,B)에 대해 위에 언급한 대로 각각을 ACKstate와 ACKnum을 0으로 초기화하고, 그 외 A와 B 변수에 대해 1로 초기화 한다. =>init함수

동작 1: data전송, 즉 layer5로부터 값을 받게 될 때 output함수를 호출하고 이를 통해 이가 시작될 것이다. 이때 baseACK 포함여부(동작 2와의 구분)를 확인하기 위해 ACK State를 확인, 해당 side의 ACK state가 0, 즉 ACK를 보낼 필요가 없는 경우 보낼 packet의 acknum을 999로 설정한다. 이후 output의 인자로 들어오는 message를 packet의 payload로서 저장하고 packet의 seqnum을 side의 next seqnum으로 한다. 이후 checksum을 0으로 초기화한 후, 해당 값들을 checksum하여 packet의 checksum을 정해준다. 이후 동작 5를 위해 buffer에 해당 packet의 정보를 저장한다. 이후, tolayer3를 이용, layer3를 통해 다른 layer로 해당 packet을 전송한다. 전송을 하고 ack를 받지 못했으니 next_seqnum에 1을 증가시켜주고 누적 ACK를 전송하였으니 재전송을 제외한 경우 다음 input이 들어올 때 까지 해당 ACK에 대한 전송이 필요가 없으므로 state를 0으로 바꾸어 준다. 이때 base와 nextsequm을 비교해 지금까지 ACK가 모두 들어왔는지 확인하고 두 값이 같다면 timer를 시작한다.

=>output 내부에서 ackstate구분, packet생성, tolayer3, side 변수 값 변경이 요구될 것이다.

동작 2: data전송, 즉 layer5로부터 값을 받게 될 때 output함수를 호출하고 이를 통해 이가 시작될 것이다. 이때 baseACK 포함여부(동작 1와의 구분)를 확인하기 위해 ACK State를 확인, 해당 side의 ACK state가 1, 즉 ACK를 보낼 필요가 있는 경우 보낼 packet의 acknum을 해당 side의 누적 ACKnum(ACKnum변수 이용)로 설정한다. 이후 output의 인자로 들어오는 message를 packet의 payload로서 저장하고 packet의 seqnum을 side의 next seqnum으로 한다. 이후 checksum을 0으로 초기화한 후, 해당 값들을 checksum하여 packet의 checksum을 정해준다. 이후 동작 5를 위해 buffer에 해당 packet의 정보를 저장한다. 이후, tolayer3를 이용, layer3를 통해 다른 layer로 해당 packet을 전송한다. 전송을 하고 ack를 받지 못했으니 next_seqnum에 1을 증가시켜주고 누적

ACK를 전송하였으니 재전송을 제외한 경우 다음 input이 들어올 때 까지 해당 ACK에 대한 전송이 필요가 없으므로 state를 0으로 바꾸어 준다. 이때 base와 nextsequm을 비교해 지금까지 ACK가 모두 들어왔는지 확인하고 두 값이 같다면 timer를 시작한다.

=>output 내부에서 ackstate구분, packet생성, tolayer3, side 변수 값 변경이 요구될 것이다.

동작 3:해당 side의 ackstate를 1로 변경한다.(ack전송 필요) checksum을 확인해 corrupt여부를 판단한다. Corrupt인 경우 해당 message를 무시한다. Corrupt가 아닌 경우, buffer(ack대기 신호가 window size보다 큰지 확인)가 full인지 확인하고 full인 경우 해당 input을 무시한다. Buffer가 여유 공간이 있는 경우, expect seqnum인지 확인한다. 만약 예측했던 경우라면 받은 packet의 payload 값을 layer5로 전송해준다. 이후 누적 ACK를 예상 seqnum으로 바꾸고 예상 seqnum에 1을 더해 준다.

=>input내부에서 ackstate변경 및 checksum확인, ack수신 대기 packet수 확인을 위한 조건문 필요. 이후 예측된 seqnum과의 비교가 필요하며 이 결과에 따라서 tolayer5함수 호출과 side의 acknum, expect seqnum의 변경이 필요할 것이다.

동작 4:해당 side의 ackstate를 1로 변경한다.(ack전송 필요) checksum을 확인해 corrupt여부를 판단한다. Corrupt인 경우 해당 message를 무시한다. Corrupt가 아닌 경우, buffer(ack대기 신호가 window size보다 큰지 확인)가 full인지 확인하고 full인 경우 해당 input을 무시한다. Buffer가 여유 공간이 있는 경우, expect seqnum인지 확인한다. 만약 예측했던 경우라면 받은 packet의 payload 값을 layer5로 전송해준다. 이후 누적 ACK를 예상 seqnum으로 바꾸고 예상 seqnum에 1을 더해 준다. 이때 동작 3와의 구분을 위해 acknum이 999가 아닌지 확인하고 이가 참이라면 base를 확인, acknum이 base보다 작을 경우 이를 무시하고(Go-Back-N의 특성), 그렇지 않을 경우 base를 acknum+1로 변경하여 다음 ack를 기다리는 packet으로 변경할 수 있도록 한다. 이후 해당 값이 들어왔으므로 timer를 정지한다. 이때 아직 보낸 packet전부에 대해 ack가 들어온 게 아니라면 다시 timer를 시작한다.

=>input내부에서 ackstate변경 및 checksum확인, ack수신 대기 packet수 확인을 위한 조건문 필요. 이후 예측된 seqnum과의 비교가 필요하며 이 결과에 따라서 tolayer5함수 호출과 side의 acknum, expect seqnum의 변경이 필요할 것이다. acknum확인을 통해 ack가 포함 여부 및 base이후 값인지 확인하고 여부를 확인 후 이를 바탕으로 base를 이동하며 ack를 기다리는 packet이 남아있을 경우 timer를 다시금 작동시키고 이러한 packet이 없을 경우 재전송이 필요없으므로 timer를 멈춘다.

동작5:위 동작의 경우 조건이 일정시간이 지났음에도 ack가 도달하지 않은 경우이므로 timer를 통해 작동할 것이며 이때 함수인 timerinterrupt를 호출하는 형식으로 이가 실행될 것이다. 이때 재전송의 범위는 ack를 받지 않는 범위, 즉, base부터 nextseqnum 직전까지 이다. 우리는 해당 동작

을 위해 동작 1,2에서 해당 packet들을 저장하였다. 이를 바탕으로 반복문을 통해 해당 범위의 packet들을 buffer에서 꺼내 tolayer3를 통해 전송하면 된다. 이를 수행하기 위해 timer가 정지되었으므로 해당 함수를 호출하며 timer를 다시 시작해준다.

=>timerinterrupt에서 timer를 재시작한 후 반복문 내 tolayer3를 통해 전송

위 동작들을 바탕으로 작성한 각 함수들은 다음과 같이 구성될 것이다.(A,B동일)

```
init{       //side에 대한 초기화를 위한 함수
            initialize Base, exp_seqnum, next_seqnum, buffend to 1
            //buffer 및 ACK확인을 위해 1로 초기화
            initialize ACKstate, ACKnum to 0   //ACK전송여부와 누적 ACK는 0으로 초기화
}

Output{ //another side로 data전달을 위한 함수
            if(over window size || buffer is full) end this function
            Make packet(next_seqnum, 999, 0, message's data)   //send to another side
            If(Ackstate==1) packet's acknum is current side's ACKnum      //ACK 추가
            Packet's checksum = checksum of packet                  //corrupt 확인을 위해
            Save packet to buffer and buffend ++
            Printf format
            Send packet to another side using Tolayer3()
            If(No need ack yet) start timer       //to need resend
            Next_sequm++; ACKstate=0;
            //move to next packet and this don't need ack now
}
```

```
Input{    //packet을 전송 받았을 경우
        ACKstate=1        //we need to ack send when output(piggyback)
        If(packet is corrupt(check using checksum)){drop message}
        If(buffer is over){end function}
        Else if(packet have no except seq){print format}        //nak check
        Else if(packet have except seq){                //correct ack is taken
                Print format
                Save data in layer5(tolayer5)
                ACKnum=exp_seqnum, exp_num++
                //expect seqnum is taken, save that to acknum and make next expect
        }
        If(packet have ack(check acknum !=999)){    //have ack
                If(base>acknum){drop message}    //that ACK is taken
                Else{
                        Base move to packet.acknum+1 //acknum-packet's ack is taken
                        Stoptimer                //if base = next seqnum, don't need resend
                        If(base !=next seqnum){restart timer}
                }
        }
}
Timerinterrupt{ //timer call function to resend
        Starttimer        //to next resend
        For(base~(next seqnum-1)){                //resend packet that is waiting ack
                Take packet from buffer according to loop and
                Send packet to another side using tolayer3 //resend packet to another side
        }
}
```

추가적으로 check sum을 위한 함수는 다음과 같다.(동작 설명은 위에서 서술)

```
Checksum(packet){
        Make Addvalue//return 을 위한 변수
        Add packet's seqnum, acknum, checksum to addvalue to addvalue
        For(I = 1~10){add packet.payload[2*i]*2^8+ paketpayload[2*i-1] to addvalue}
        //add payload by 16bit (2words)
        Add (addvalue)/(2^16) to addvalue//wrap around carry
        Return ~((unsigned short)(addvalue))//1's complement
}
```

-결과화면 캡쳐 및 화면에 대한 설명

```
-----  Stop and Wait Network Simulator Version 1.1 --------

Enter the number of messages to simulate: 20
Enter  packet loss probability [enter 0.0 for no loss]:0.2
Enter packet corruption probability [0.0 for no corruption]:0.2
Enter average time between messages from sender's layer5 [ > 0.0]:10
Enter TRACE:2
```

위 값은 제안서에 적혀 있는 GBN 테스트를 위한 조건 값이다 이를 바탕으로 결과화면을 출력하였다. 위에서 볼 수 있듯이 message는 20개가 출력되며 loss와 corrupt는 0.2, 즉 5개당 1개 꼴로 생성된다. Message는 10의 시간당 1번꼴로 나오게 된다.

결과는 다음과 같다.

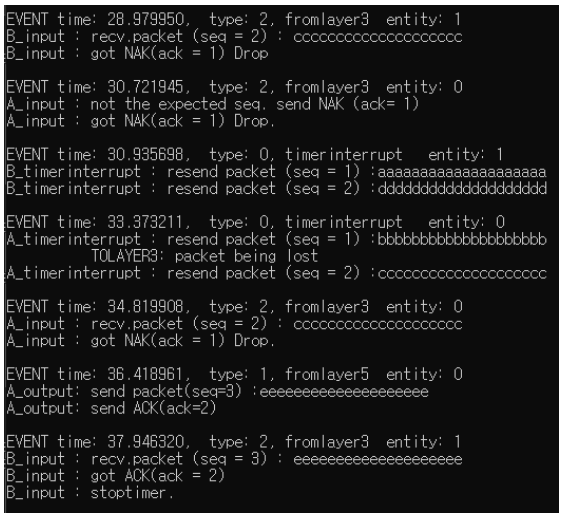
```
EVENT time: 0.935697,  type: 1, fromlayer5  entity: 1
B_output: send packet(seq=1) :aaaaaaaaaaaaaaaaaaaa
          TOLAYER3: packet being corrupted
B_output : start timer

EVENT time: 5.026246,  type: 2, fromlayer3  entity: 0
A_input : Packet corrupted. Drop.

EVENT time: 10.935698,  type: 0, timerinterrupt   entity: 1
B_timerinterrupt : resend packet (seq = 1) :aaaaaaaaaaaaaaaaaaaa
          TOLAYER3: packet being corrupted

EVENT time: 13.373211,  type: 1, fromlayer5  entity: 0
A_output: send packet(seq=1) :bbbbbbbbbbbbbbbbbbbb
A_output: send ACK(ack=0)
A_output : start timer

EVENT time: 17.024719,  type: 2, fromlayer3  entity: 0
A_input : Packet corrupted. Drop.
```

B_output을 통해 ACK가 포함되지 않은 상태에서 corrupt된 packet이 전송되었다. A는 해당 packet이 corrupted임을 감지, packet을 drop한다. 이 때문에 state가 바뀌어 A_output은 ACK 0(첫 패킷이 성공적으로 받아지지 않았을 경우)을 포함하고 있다. 두 번째 corrupt가 A input을 통해 들어와도 마찬가지이다. 이때 B의 timer가 일정시간동안 ack를 받지 못하고 다시금 timeinterrupt함수 호출을 통해 data를 전송하고 있음을 확인할 수 있다.

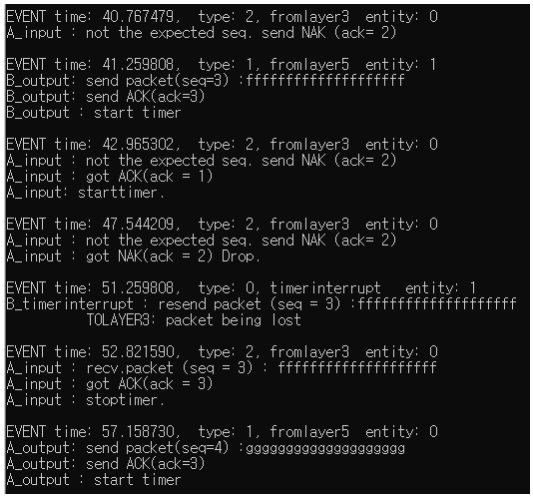
```
EVENT time: 18.921690,  type: 2, fromlayer3  entity: 1
B_input : recv.packet (seq = 1) : bbbbbbbbbbbbbbbbbbbb
B_input : got NAK(ack = 1) Drop

EVENT time: 20.935698,  type: 0, timerinterrupt   entity: 1
B_timerinterrupt : resend packet (seq = 1) :aaaaaaaaaaaaaaaaaaaa

EVENT time: 22.093571,  type: 1, fromlayer5  entity: 0
A_output: send packet(seq=2) :cccccccccccccccccccc
A_output: send ACK(ack=0)

EVENT time: 23.373211,  type: 0, timerinterrupt   entity: 0
A_timerinterrupt : resend packet (seq = 1) :bbbbbbbbbbbbbbbbbbbb
A_timerinterrupt : resend packet (seq = 2) :cccccccccccccccccccc

EVENT time: 25.891445,  type: 2, fromlayer3  entity: 0
A_input : recv.packet (seq = 1) : aaaaaaaaaaaaaaaaaaaa

EVENT time: 26.282541,  type: 1, fromlayer5  entity: 1
B_output: send packet(seq=2) :dddddddddddddddddddd
B_output: send ACK(ack=1)
          TOLAYER3: packet being lost
```

이에 대해 B_input은 A_output의 ACK가 기대값이 아님을 알고 packet을 drop하는 것을 볼 수 있

다. 또한 이를 통해 packet이 잘못 전송됨을 인지, timer를 stop하지 않고 계속해 timeinterrupt를 통해 전송하고 있다. A가 보낸 message가 drop되었으므로 A가 보낸 값에 대한 ACK는 B에서 전송되지 않고 timer작동으로 A또한 계속해 ACK가 넘어오지 않은 값들을 재전송 한다. 이후 A_input으로 seq=1인 결과를 받았고 이번엔 B_output을 통해 전송된 ddd...의 값이 lost되었다.



이후 A_Output이 진행되지 않아 B_input에 대하여 ack가 오지 않았고, 그 때문에 B_input에서 packet은 계속 버려지게 되며 B_timerinterrupt가 계속해 발생한다. 이는 A_input 또한 마찬가지이다. 이후 A_output이 발생, seq=3에 전송과 동시에 ack=2를 전송한다. B_input이 기대하던 ack=2를 받게 되어 해당 message는 drop되지 않는다. 동시에 B는 timer를 정지한다.



이후 B_input이 ack를 받은 후 B_output에서 새로운 데이터와 함께 통신이 양호했다는 화답을 위해 ack=3를 전송한다. 이후 A_input은 이전에 보내진 값들에 대하여 이를 무시하며 A_input을 통해 기댓값이던 ACK=3가 수신된 후에야 timer를 정지한다.(기존 interrupt 보내던 행위를 정지한다) 이후 A_output을 통해 새로운 data와 함께 B에 화답을 주기 위해 ack3를 전송한다.

```
EVENT time: 61.259808,  type: 0, timerinterrupt   entity: 1
B_timerinterrupt : resend packet (seq = 3) :ffffffffffffffffffff
          TOLAYER3: packet being corrupted

EVENT time: 65.183876,  type: 2, fromlayer3  entity: 1
B_input : recv.packet (seq = 4) : gggggggggggggggggg
B_input : got ACK(ack = 3)
B_input : stoptimer.

EVENT time: 65.998566,  type: 2, fromlayer3  entity: 0
A_input : Packet corrupted. Drop.

EVENT time: 67.158730,  type: 0, timerinterrupt   entity: 0
A_timerinterrupt : resend packet (seq = 4) :gggggggggggggggggg

EVENT time: 68.963501,  type: 2, fromlayer3  entity: 0
A_input : recv.packet (seq = 4) : gggggggggggggggggg
A_input : got NAK(ack = 4) Drop.

EVENT time: 73.023468,  type: 1, fromlayer5  entity: 0
A_output: send packet(seq=5) :hhhhhhhhhhhhhhhhhhhh
A_output: send ACK(ack=4)

EVENT time: 77.158730,  type: 0, timerinterrupt   entity: 0
A_timerinterrupt : resend packet (seq = 4) :ggggggggggggggggg
          TOLAYER3: packet being corrupted
A_timerinterrupt : resend packet (seq = 5) :hhhhhhhhhhhhhhhhhhhh
          TOLAYER3: packet being corrupted

EVENT time: 81.667442,  type: 2, fromlayer3  entity: 1
B_input : recv.packet (seq = 5) : hhhhhhhhhhhhhhhhhhhh
B_input : got ACK(ack = 4)
Warning: unable to cancel your timer. It wasn't running.
B_input: starttimer.

EVENT time: 82.129585,  type: 2, fromlayer3  entity: 0
A_input : Packet corrupted. Drop.

EVENT time: 84.165901,  type: 2, fromlayer3  entity: 0
A_input : Packet corrupted. Drop.

EVENT time: 87.158730,  type: 0, timerinterrupt   entity: 0
A_timerinterrupt : resend packet (seq = 4) :ggggggggggggggggggg
          TOLAYER3: packet being lost
A_timerinterrupt : resend packet (seq = 5) :hhhhhhhhhhhhhhhhhhhh
```

이후 B_input이 원하던 ACK를 받고 해당 seq를 저장한 후 timer를 정지한다. A_input은 B에서 보낸 corrupt된 packet은 여전히 잘 처리하는 모습이다. 이후 A_output을 통해 ACK를 보냄과 동시에 data를 보낸다. 이후 ACK가 오지 않은 동안 interrupt를 통해 재전송을 진행했다. B_input을 통해 해당 값이 받아진다.(이때 B에서 interrupt를 진행하지 않으므로 timer가 stop할 timer가 없다) 이후 timer를 작동시킨다. A는 이전에 보내진 값들에 대해 Drop을 진행한다. 이후 A_timerinterrup에 의해 전송된 packet이 lost되지만 이미 B에서 input을 통해 처리되었다.

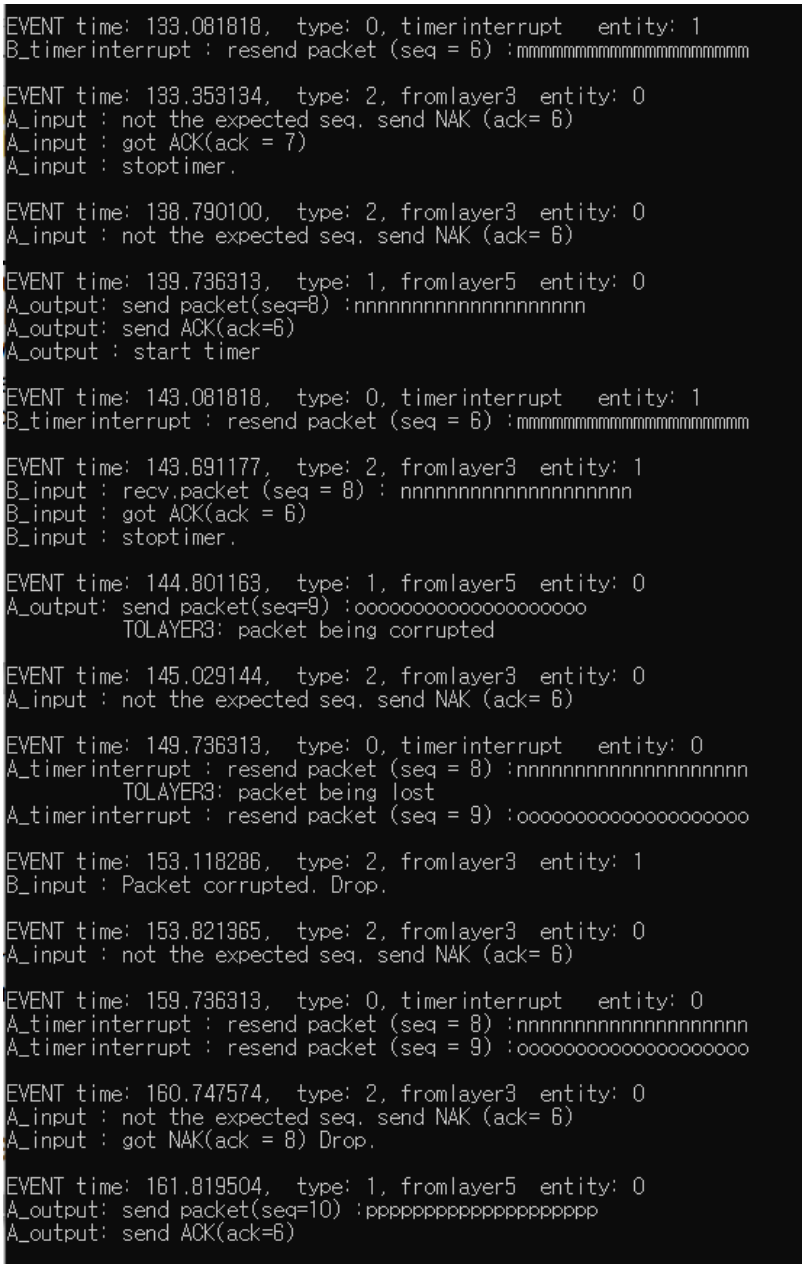
```
EVENT time: 88.319412,  type: 2, fromlayer3  entity: 0
A_input : recv.packet (seq = 5) : hhhhhhhhhhhhhhhhhhhh
A_input : got ACK(ack = 4)
A_input: starttimer.

EVENT time: 91.667442,  type: 0, timerinterrupt   entity: 1

EVENT time: 91.735588,  type: 1, fromlayer5  entity: 1
B_output: send packet(seq=4) :iiiiiiiiiiiiiiiiiiii
B_output: send ACK(ack=5)
          TOLAYER3: packet being corrupted

EVENT time: 93.451096,  type: 2, fromlayer3  entity: 0
A_input : Packet corrupted. Drop.

EVENT time: 98.319412,  type: 0, timerinterrupt   entity: 0
A_timerinterrupt : resend packet (seq = 5) :hhhhhhhhhhhhhhhhhhhh
          TOLAYER3: packet being lost

EVENT time: 101.667442,  type: 0, timerinterrupt   entity: 1

EVENT time: 107.163918,  type: 1, fromlayer5  entity: 0
A_output: send packet(seq=6) :jjjjjjjjjjjjjjjjjjj
A_output: send ACK(ack=5)

EVENT time: 108.319412,  type: 0, timerinterrupt   entity: 0
A_timerinterrupt : resend packet (seq = 5) :hhhhhhhhhhhhhhhhhhhh
          TOLAYER3: packet being corrupted
A_timerinterrupt : resend packet (seq = 6) :jjjjjjjjjjjjjjjjjjj
          TOLAYER3: packet being lost

EVENT time: 111.667442,  type: 0, timerinterrupt   entity: 1

EVENT time: 113.081825,  type: 2, fromlayer3  entity: 1
B_input : recv.packet (seq = 6) : jjjjjjjjjjjjjjjjjjj
B_input : got ACK(ack = 5)
B_input: starttimer.
```

이후 A_input이 이전에 보낸 값과 ACK=4을 통해 화답을 잘 받았으며 받은 data에 대해 timer를

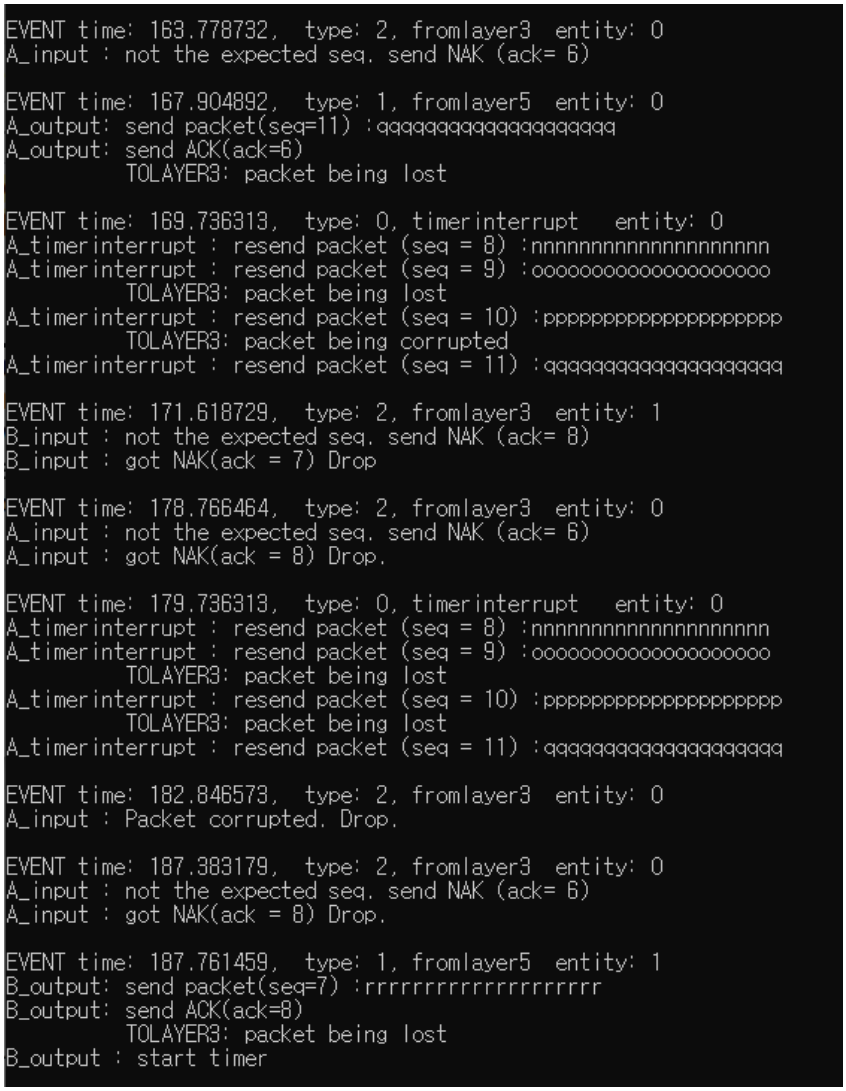
시작한다. 이는 이후 interrupt에서 재전송됨을 확인할 수 있다. 이후 B_output을 통해 ack-5를 통해 화답하려하나 corrupt된다. 이후 A_input에 의해 corrupt된 message가 drop되며 A_timerinterrupt를 통해 다시 ACK를 요청하나 lost된다. 이후 A_output을 통해 B에 대해 화답하고 data를 전송한다. 이후 timerinterrupt로 재전송하나 corrupt와 lost가 진행된다. 이후 B에서 ACK5에 대한 값을 받게 된다. 이후 timer를 시작해 재전송을 위해 준비한다.

```
EVENT time: 115.714203,  type: 2, fromlayer3  entity: 0
A_input : Packet corrupted. Drop.

EVENT time: 118.319412,  type: 0, timerinterrupt   entity: 0
A_timerinterrupt : resend packet (seq = 5) :hhhhhhhhhhhhhhhhhhhh
A_timerinterrupt : resend packet (seq = 6) :jjjjjjjjjjjjjjjjjjj

EVENT time: 118.502151,  type: 1, fromlayer5  entity: 0
A_output: send packet(seq=7) :kkkkkkkkkkkkkkkkkkk
A_output: send ACK(ack=5)

EVENT time: 122.041992,  type: 2, fromlayer3  entity: 1
B_input : recv.packet (seq = 7) : kkkkkkkkkkkkkkkkkkk
B_input : got NAK(ack = 6) Drop

EVENT time: 123.002144,  type: 2, fromlayer3  entity: 0
A_input : not the expected seq. send NAK (ack= 5)
A_input : got NAK(ack = 5) Drop.

EVENT time: 123.081825,  type: 0, timerinterrupt   entity: 1

EVENT time: 125.970680,  type: 2, fromlayer3  entity: 0
A_input : recv.packet (seq = 6) : jjjjjjjjjjjjjjjjjjj
A_input : got ACK(ack = 5)
A_input: starttimer.

EVENT time: 126.333809,  type: 1, fromlayer5  entity: 1
B_output: send packet(seq=5) :||||||||||||||||||||
B_output: send ACK(ack=7)

EVENT time: 127.805412,  type: 1, fromlayer5  entity: 1
B_output: send packet(seq=6) :mmmmmmmmmmmmmmmmmmmm
         TOLAYER3: packet being lost
B_output : start timer
Warning: attempt to start a timer that is already started
```

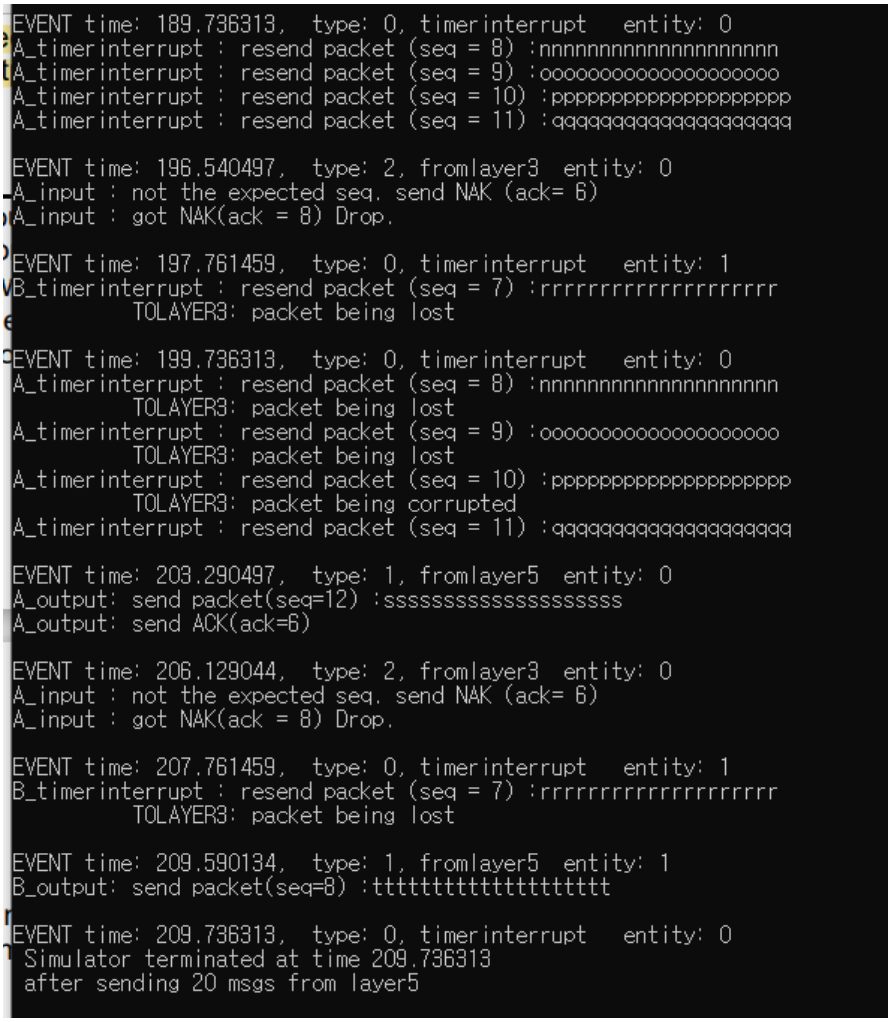
이후 A_timerinterrupt를 통해 packet이 알맞게 전송되고 A_output을 통해 ACK=5에 대한 화답을 한다. 이후 B_input을 통해 ack=5로 된 값을 받게 되는데 이는 B가 원하는 ack가 아니므로 drop 된다. 이후 A_input을 통해 ack=5가 받아지게 된다. B_output은 ACK=7을 전송함과 동시에 data만 있는 packet을 전송하려하나 lost된다. 이는 timer start로 재전송 될 것이다.



예상대로 timerinterrupt에 의해 해당 packet은 재전송되며 A_input은 data는 기대하던 ack를 받게 되고 timer를 정지한다. 이후 예상되지 않는 packet들을 생략하고 ack에 대한 화답으로 ack=6을 전송한다. B는 timer로 계속해 seq=6에 값을 전송하며 ack=6에 해당하는 값을 받고 timer를 정지한다. 이후 A_output이 corrupt되고 input으로 ack가 기대보다 낮은 것들을 받지 않는다. 이후 corrupt된 함수에 대해 다시금 전송한다. B_input은 전송되었던 corrupt값을 받지 않으며 A_input또한 예측되지 않은 seq의 paket을 받지 않는다. 이후 ack를 받지 못한 packet들을 timerinterrupt를 통해 계속해 보낸다. A_input이 drop후 A_output을 통해 ACK를 통한 화답과 함께 새로운 data를 전송한다.



이후 A_output을 통해 새로운 data와 ack를 통한 화답을 실행하나 lost되며 이 때문에 interrupt에 해당 packet이 ack가 없는 packet들과 함께 재전송된다. 이후 B_Output을 통해 이전에 받아진 ack=8에 대해 화답이 시도되나 lost가 진행된다. 허나 timer가 시작되어 이는 다시 전송이 시도될 것이다.



이후 계속 반복이 진행되고, A_output으로 새로운 packet이 전송을 준비한다. 허나 A_input에는 ACK가 돌아오지 않고(계속된 lost발생) A에서 재전송된 seq=8,9,10,11와 A_output을 통해 전송된 seq=12에 대해 화답을 받지 못한다. 또한 B_timerinterrupt로 전송되는 seq=7과 B_output을 통해 전송될 seq=8또한 화답을 받지 못할 것이다.

이를 통해 corrupt에 대한 양 side에 Drop은 잘 이뤄지고 있으며 ACK에 의한 소통 및 interrupt를 통한 재전송과 그로인한 corrupt와 lost로 인한 손실 방지 또한 잘 진행되고 있음을 알 수 있다. ACK와 SEQ에 대한 값 변화도 예측한 대로 진행되었다는 것을 확인했다.

또한 조건이었던 첫 Packet전송 후 corrupt에 의해 손실된 packet을 받을 경우 화답을 ack=0으로 하는 것 또한 지킨 상태로 ACK없는 packet의 전송이 올바르게 진행되었으며 초기 값들 또한 init를 통해 잘 적용되었다는 것을 알 수 있었다.

이를 통해 예상했던 동작 0,1,2,3,4,5가 모두 적합하게 돌아감을 확인할 수 있었다.

고찰:

위 과제를 통해 네트워크 통신이 packet을 통해 진행되는 방식들을 알 수 있었으며 그중 Go Back N을 제작해보며 Go Back N이 어떻게 동작하는지 알 수 있었다. 이를 통해 네트워크에 대한 이해도를 높일 수 있었다.

또한 A와 B가 방향성을 제외하고는 완전히 동일하게 제작함으로서 이 횟수가 늘어나더라도 이를 그대로 사용할 수 있게 하며 함수의 범용성의 중요성을 알고 범용성을 높이는 법 또한 배우게 되었다. 또한 piggyback을 배우며 원래는 따로 전송할 생각밖에 하지 못했으나 이를 같이 전송해 packet수를 줄이는, 다양한 효율을 늘리는 방식에 대해 다시금 고민하게 되었다.

단, 해당 과제를 진행하며 아쉬운 점도 있었다. 대표적으로 구현한 piggyback의 경우 output이 없으면 보낸 packet에 대하여 ack를 받을 수 없다. 이 때문에 message 수가 정해져 있다면 언제나 모든 packet이 ack를 받을 수 없게 된다. 이는 실제로 사용되는 구현에 있어서는 치명적일 것 이다.(이 때문에 input후 tolayer3를 통해 재전송하는 방향을 모색하기도 했다.)

Reference

컴퓨터네트워크 강의자료:Lecture 4.pdf

컴퓨터네트워크 BiGBN_ExtraCredit구현_강의자료_수정본.pdf

컴퓨터네트워크 ProgAssn.pdf

컴퓨터네트워크 BiGBN 강의자료.pdf