Assignment 2

과목명	컴퓨터네트워크
	화6 목5
담당교수	이혁준 교수님
학과	컴퓨터정보공학부
학년	3학년
학번	2019202009
이름	서여지
제출일	21.06.04(금)

KWANGWOON UNIVERSITY

1. Introduction

이 과제는 전송계층에서 동작하는 Reliable data transport 프로토콜인 Go-Back-N을 구현하는 것이다. 프로토콜의 시뮬레이터는 A와 B side를 갖는다. A와 B side는 각각 20개의 문자로 구성된 message를 layer5에서 받아 layer3로 전송하여 통신한다. 시뮬레이터는 세 종류의 event를 생성하여 해당하는 동작을 수행하는 함수를 호출한다.

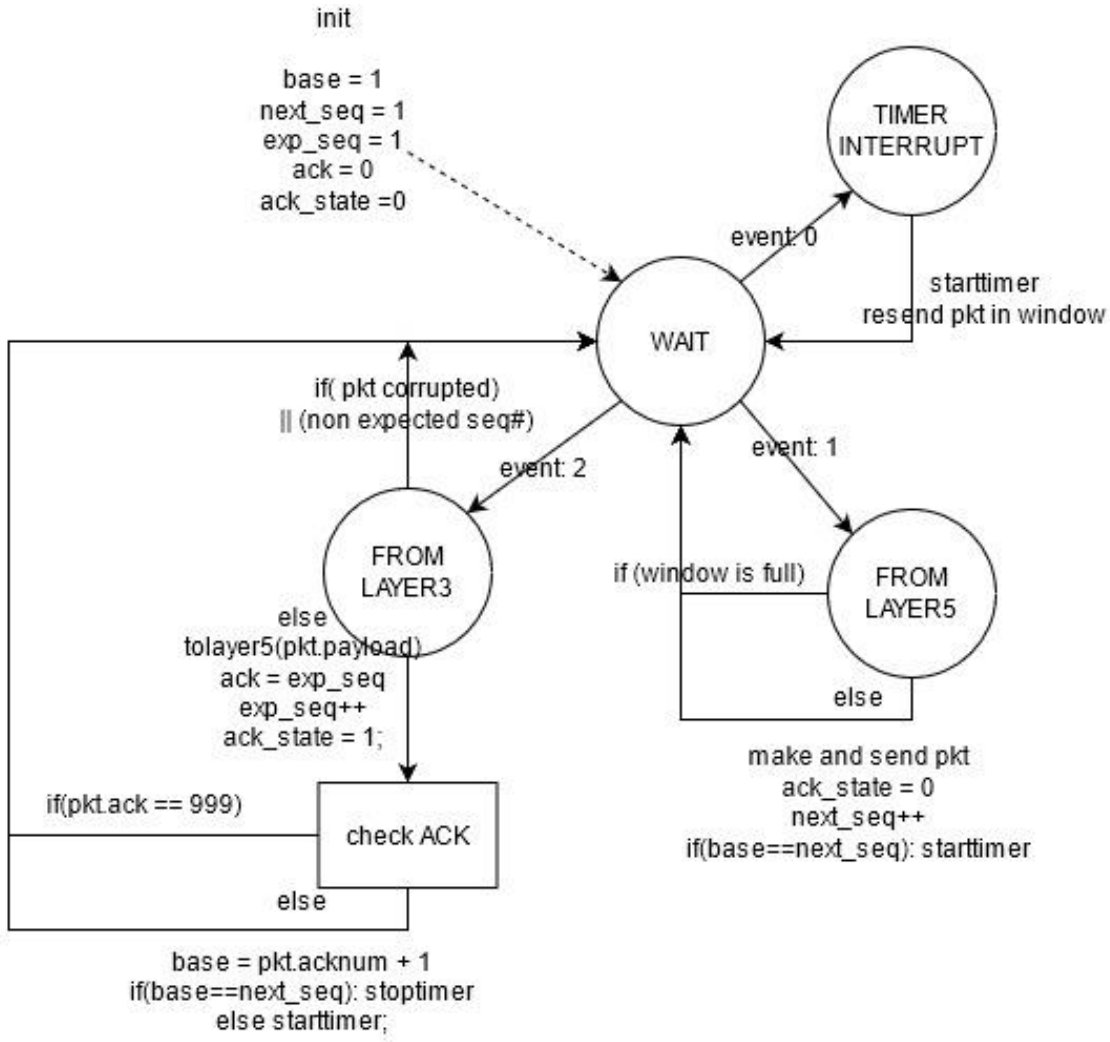
0번 event는 TIMER_INTERRUPT이고 AorB_timerinterrupt함수를 호출하여 window 내의 packet을 재전송 한다. 1번 event는 FROM_LAYER5이고, AorB_output함수를 호출하여 packet을 생성하여 layer3으로 보내는 동작을 한다. 마지막으로 2번 event는 FROM_LAYER3으로, AorB_input함수를 호출하여 받은 packet을 확인하여 정보를 layer5로 전송하거나 drop 한다. event를 통해 호출되는 함수와 시뮬레이터를 처음 실행했을 때 호출되는 AorB_init함수를 A와 B side 모두에 대하여, 총 8개의 함수를 작성하는 것이 이번 과제의 내용이다.

Go-Back-N프로토콜은 Reliable data transmission을 위하여 cumulative ACK과 timer를 이용하고, timer의 종료까지 ACK을 받지 못한 window 내에 존재하는 모든 packet을 재전송한다. ACK를 받으면 해당 packet과, 해당 packet의 seq#보다 작은 seq#를 가진 모든 패킷을 window에서 제외하고, window는 새로운 packet을 전송할 수 있게 된다. 반면 window가 가득 찼을 때 layer5에서 새로운 message를 전송하는 것을 시도하면 해당 message는 drop된다. 또한 이번 과제에서는 data packet에 ACK 정보를 함께 전송하는 piggyback을 이용하며, 전송할 message가 없는 경우에도 ACK을 단독으로 전송할 수 없다.

시뮬레이터는 실행 시 사용자에게 보낼 메시지의 수, packet이 loss될 확률, packet의 data가 corrupt될 확률, packet을 받은 뒤 받은 packet의 message를 layer5로 전송하는데 걸리는 시간, 시뮬레이션 정보의 출력 여부를 입력 받는다. 이때 packet message를 layer5로 전송하는데 걸리는 시간은 A혹은 B side가 다음 packet을 전송하거나 수신하기까지의 시간에 영향을 준다.

2. FSM Flow chart

A와 B side의 구조는 동일하다.



3. Code description

A. 전역변수 사용 용도 설명

WINDOW_SIZE와 TIMEOUT은 각각 GBN의 send window의 크기와 timeout 기간을 설정한다. A와 B side는 각각 base, next_seq, ack, exp_seq, ack_state를 전역변수로 갖는다. base는 send window에서 가장 작은 seq#를 갖는 packet, 즉 ACK를 받지 못한 packet의 seq#이고, next_seq는 다음에 생성될 packet의 seq#이다. ack는 가장 마지막에 받은 packet의 seq#이고, exp_seq는 다음에 수신할 packet의 seq#이다. 마지막으로 ack_state는 다음 보내는 packet에 piggyback해서 보낼 ACK가 존재하는지 여부를 의미한다.

send window는 pkt 배열로 구현한 A_sndpkt와 B_sndpkt이다. 함수에서 send window를 이용하는 경우 packet의 seq#를 WINDOW_SIZE로 나눈 나머지의 값을 index로 하여 해당 위치에 packet의 내용을 저장하여 사용한다.

B. int makeChecksum (struct pkt *p)

인자로 전달된 packet의 checksum을 계산하여 반환한다. 이때 인자로 전달되는 packet은 checksum을 제외한 모든 data가 삽입되어있어야 한다. checksum함수는 seqnum과 acknum, payload를 16bit씩 나눈 값을 모두 1의 보수 덧셈을 한 뒤, 값을 not연산 하여 checksum을 계산한다.

C. A_init(), B_init()

base, next_seq, exp_seq, ack, ack_state의 값을 초기화한다.

D. A_output(struct msg message), B_output(struct msg message)

현재 send window에 빈 자리가 있다면 인자로 전달받은 message를 전달하기 위한 packet을 생성하여 layer3으로 전달한다. timer가 실행되고있지 않다면 timer를 동작시킨다. ack를 전달했으므로 ack_state를 0으로 바꾸고, next_seq를 1증가시킨다.

E. A_input(struct pkt packet), B_input(struct pkt packet)

인자로 전달된 packet의 checksum을 확인하여 corrupt 여부를 확인한다. packet의 data에 오류가 없고, seq#가 exp_seq와 일치하는 경우 수신한 packet의 내용을 layer5로 전송한다. ack를 갱신하고, exp_seq의 값을 1 증가시킨다. 또한 ack_state를 1로 바꾸어 다음에 보내는 packet에 ack를 포함하도록 한다. 반면 seq#가 일치하지 않는 경우 다른 동작을 하지 않고 timeout을 기다린다. packet에 ACK가 포함된 경우 base값을 갱신하여 send window를 이동시킨다. 모든 packet에 대한 ack를 받았다면 timer를 중단하고, 아닌 경우 timer를 재시작한다.

F. A_timerinterrupt(), B_timerinterrupt()

타이머를 다시 실행하고, send window내의 모든 패킷을 seq# 순서대로 재전송한다.

4. Result & Analysis

(1) 일반적인 동작을 확인한다.

```
Enter the number of messages to simulate: 10
Enter  packet loss probability [enter 0.0 for no loss]:0.5
Enter packet corruption probability [0.0 for no corruption]:0.1
Enter average time between messages from sender's layer5 [ > 0.0]:5.0
Enter TRACE:0
B_output: send packet (seq = 1): aaaaaaaaaaaaaaaaaaaa
B_output: start timer.
A_input: recv packet (seq = 1) : aaaaaaaaaaaaaaaaaaaa
```

B가 A에게 packet 하나를 전송하고, A가 그것을 수신하였다.

```
A_output: send packet (seq = 1): bbbbbbbbbbbbbbbbbbbb
A_output: send ACK (ack: 1)
A_output: start timer.
B_input: recv packet (seq = 1) : bbbbbbbbbbbbbbbbbbbb
B_input: got ACK (ack = 1)
B_input: stop timer.
```

A가 B에게 packet에 ACK를 포함하여 전송하고, B가 그것을 수신하였다.

```
A_output: send packet (seq = 2): ccccccccccccccccccc
A_output: send packet (seq = 3): ddddddddddddddddddd
A_timerinterrupt: resend packet (seq = 1): bbbbbbbbbbbbbbbbbbbb
A_timerinterrupt: resend packet (seq = 2): ccccccccccccccccccc
A_timerinterrupt: resend packet (seq = 3): ddddddddddddddddddd
```

A가 B에게 추가로 두 개의 packet을 전송한 뒤, timeout되어 ack를 받지 못한 3개의 packet을 모두 재전송한다.

```
B_input: recv packet (seq = 2) : ccccccccccccccccccc
A_output: send packet (seq = 4): eeeeeeeeeeeeeeeeeee
B_input: recv packet (seq = 3) : ddddddddddddddddddd
A_output: send packet (seq = 5): fffffffffffffffffff
B_input: not the expected seq(4). send NAK (ack = seq 1)
B_input: got NAK(ack = 1). Drop.
```

B는 A가 timeout되기 전 전송한 두 개의 packet을 수신한 뒤, A가 재전송한 packet을 수신하여 해당 packet을 drop하였다.

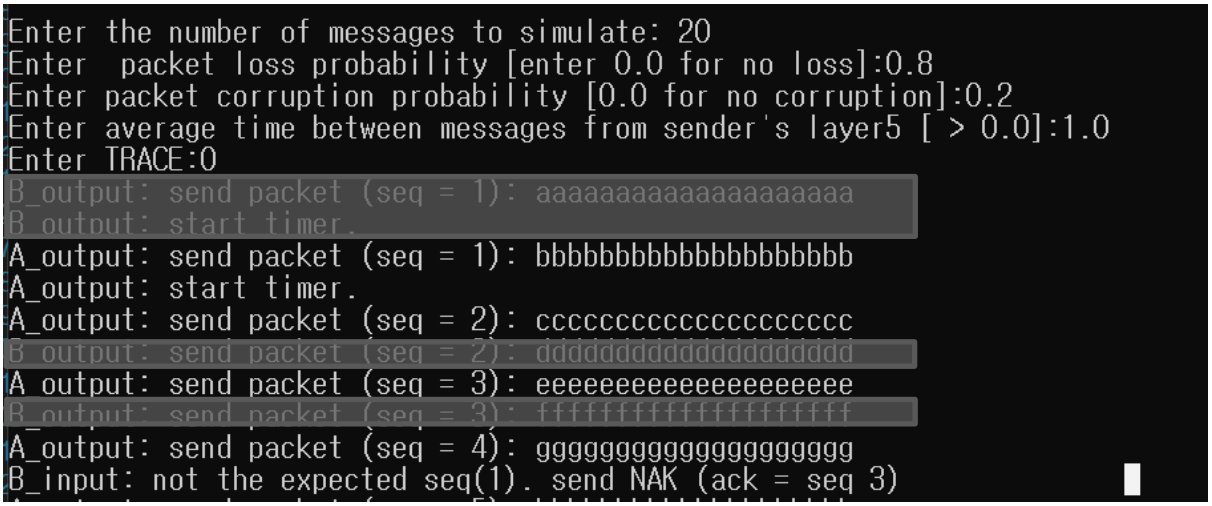
```
B_output: send packet (seq = 2): ggggggggggggggggggg
B_output: send ACK (ack: 3)
B_output: start timer.
```

B가 다음 message를 전송할 때 누적 ack값을 함께 전송한다.

```
A_input: recv packet (seq = 2) : ggggggggggggggggggg
A_input: got ACK (ack = 3)
A_input: start timer.
```

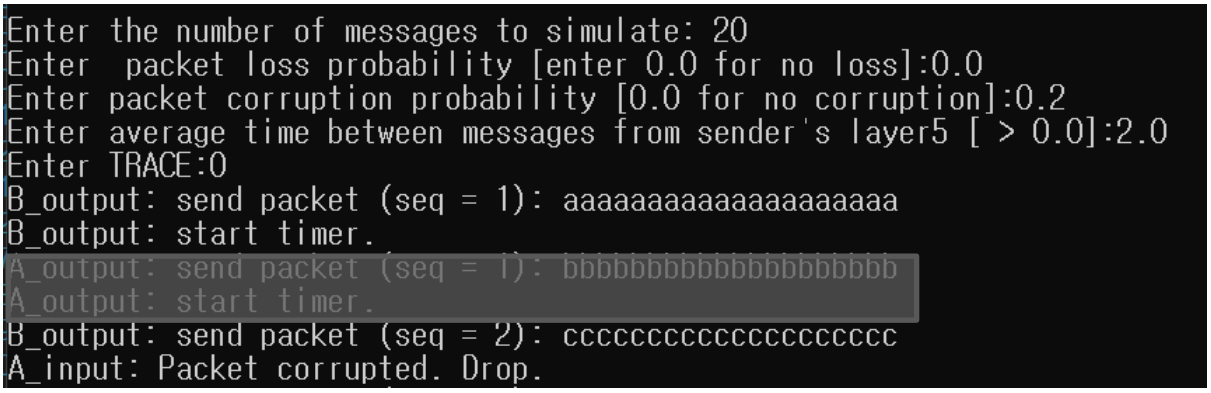
A가 ack를 받은 뒤, window내에 ack을 수신해야하는 packet이 남아있기 때문에 timer를 재시작한다.

(2) packet이 loss되는 확률을 높여서 packet이 분실되었을 때의 동작을 확인한다.



이 경우 A가 전송한 두 개의 packet이 분실되고, 세 번째 packet이 수신되었다.

(3) packet의 내용에 오류가 발생하는 경우를 찾아 관찰한다.



B가 A에게 전송한 packet의 내용에서 오류가 발생하였다.

이어서 수신된 packet은 seq#가 맞지 않기 때문에 drop된다.

5. Consideration

이번 과제는 GBN을 구현하여 그 동작을 직접 확인할 수 있는 과제였다. 기존의 rdt와 달리 한 번에 여러 개의 packet을 전송하기 때문에 그 과정이 복잡하게 느껴졌는데, 과제를 진행하며 protocol에 대해 더 깊이 이해할 수 있게 되었다. 처음 GBN을 구현한 뒤, 시뮬레이션을 실행했을 때, piggyback을 정확히 이해하지 못해서 실험 결과를 이해하는 것에 어려움을 겪었다. A혹은 B side에서 전송할 data가 없는 경우 ack를 보낼 수 없다는 점을 기억할 필요가 있었다.

input 함수에서 buffer를 이용하지 않았다. 제시된 output format에서 Buffer is full event가 존재하는데 이에 대해서 이해하지 못했다. 또한 이 과제에서 send window에 저장되는 pkt은 이후에 값이 갱신되지 않기 때문에 timerinterrupt에서 packet이 재전송되는 경우 갱신되지않은 ACK값이 전송된다.

이 점을 제외하면 과제를 진행하는데 큰 어려움은 없었다. send window를 배열을 이용하여 구현하였는데, 시뮬레이터에서 event를 관리하는 것과 유사하게 포인터를 이용하면 더 적은 자원을 이용하는 결과를 기대할 수 있을 것 같다. 방학기간을 이용하여 해당 코드를 개선할 수 있을 것 같다.