

Student ID	2017314626
Name	이재훈

Report for Assignment #2

*export to pdf when submit

1. Environment

- Describe your development environment information in detail. (Versions of operating systems, what kinds of language or tools that you used, and etc).

OS: Ubuntu 18.04

Language: python 3.6.9

Tool: vscode

2. Design

- Explain how to design your program in the view of data structures and algorithms.

본 과제에서는 sender 에서 receiver 로의 전송을 위해 logHandler.py 를 이용하였습니다. 또한 udp 프로토콜을 사용합니다.

■ P1: RDT 1.0

Rdt 1.0 의 경우, **corruption** 및 **packet loss** 가 없으며, 최대 파일 크기가 1kb 이기 때문에, sender 에서 1200 의 버퍼 크기로 파일을 rb 모드로 읽은 후, PASender 를 통해 receiver 로 전송을 합니다. 이 때 로그도 작성합니다. Receiver 에서는 10090 포트로 소켓을 바인드하며, sender 로 부터 파일을 받은 후, wb 모드로 write 후 프로그램을 종료합니다.

■ P2: RDT 2.2

Rdt 2.2 의 경우, **corruption** 이 일어날 수 있습니다. **corruption** 은 receiver 에서 체크섬을 통해 판단하며, ack 를 통해 sender 에서도 재전송 여부를 결정할 수 있습니다. 패킷의 헤더에는 체크섬과 **sequential number** 가 포함되었습니다. Window 의 크기가 1 이기 때문에, seq_num 은 0 과 1 을 바뀌가며 씁니다. 체크섬은 struct 모듈을 이용하여 구현되었습니다. 모든 데이터를 2 바이트(16 비트)씩 나누기 위해, 만약 데이터의 길이가 홀수바이트라면, 마지막에 b'\x00' 을 더해줍니다. 그 후 데이터를 2 바이트 단위로 다 더해주는데, 이 때 16 비트보다 넘치는 자리수(carry)는 마지막 자리수에 다시 더해줍니다. 마지막으로 이를 0xFFFF(1111111111111111)과 ^ (xor) 처리를 하여 체크섬이 완성됩니다. 체크섬을 위한 데이터는 헤더+payload, 즉 seq_num+체크섬+payload 가 모두 필요합니다. (리시버에서 헤더의 corruption 도 판단할 수 있어야 하기 때문에). 그러나 체크섬을 구하기 위해서 체크섬이 필요하다는 딜레마가 생깁니다. 따라서 체크섬을 구할

때는 체크섬이 0 이라고 가정을 하고 구한 후, receiver 로 send 를 할 때 앞에서 구한 체크섬 값으로 패킷을 변경 후 보냅니다. receiver 에서도 역과정으로 체크섬 값을 저장한 후, 체크섬을 0 으로 간주한 후 모든 데이터를 2 바이트 씩 더하는 계산을 합니다. 그 후 해당 값을 sender 에서 받은 체크섬 값과 더했을 때 0xFFFF(65535)가 된다면 corruption 이 없는 것이고, 그렇지 않으면 corruption 이 일어난 것으로 판단합니다. 다시 sender 를 설명하자면, sender 에서는 새로운 데이터를 보낼 때마다 seq_num 을 바꿔가며 (0,1) 헤더에 포함 시킵니다. 그 후 헤더+payload 패킷을 receiver 에 보내게 되면, ack 를 기다립니다. 이 때 receiver 에서는 새로운 데이터를 받고 체크섬을 통해 해당 패킷이 corrupt 됐는지 판단합니다. 만약 corrupt 되지 않았다면 데이터를 저장하고 해당 패킷의 seq_num(0,1)을 ack 로 sender 에게 보내 줍니다. Sender 는 자신이 보낸 seq_num 과 일치하는 ack 를 받으면, corruption 이 없다고 판단 후 새로운 seq_num 으로 새로운 데이터를 전송하게 됩니다. 또한 헤더 자체가 corrupt 되어 체크섬이 숫자가 아닌 경우도 있습니다. 숫자가 아닌 경우는, try except 를 통해 corruption 을 판단하였습니다. 만약 receiver 에서 corruption 이 있다고 판단한다면, 앞에서 보냈던 ack (prv_ack) 변수를 재전송하여 sender 에게 corruption 을 알립니다. 또한 sender 에서는 파일의 마지막을 b'', 즉 빈 패킷을 통해 알려줍니다. receiver 는 빈 패킷을 받으면, 빈 패킷을 받았다고 ack 를 보낸 후 종료하게 됩니다. sender 는 ack 을 받은 후 종료합니다. 이런 모든 통신에는 로그가 저장됩니다.

■ P3: RDT 3.0

Rdt 3.0 은 corruption+packet loss 가 모두 일어납니다. Rdt 2.2 와 data corruption 판단 방식은 동일합니다. Packet loss 판단을 위해서는, socket 의 settimeout 기능이 사용됩니다. settimeout(n)을 설정하면, n 초 동안 소켓 통신이 없을 시 예외를 일으킵니다. 따라서 0.01 초 동안 ack 가 오지 않는다면, sender 는 이를 packet loss 로 판단하고 데이터를 재전송합니다. 또한 만약 정상적인 ack 를 받았다면 타이머를 다시 none 으로 설정하고, 다음 데이터를 보낸 후 다시 0.01 초로 설정합니다. 잘못된 ack 가 올 경우에는 무시하고 timeout 을 기다립니다. receiver 에서는 packet loss 가 일어나면 그것에 대한 반응은 할 수 없습니다.(받은 패킷이 없기 때문에) 따라서 corruption 에 대한 처리는 2.2 와 똑같으며, timeout 후 다시 오는 패킷이 corruption 이 없다면 정상적인 ack 를 보내줍니다.

■ P4: RDT 3.0 + pipelining

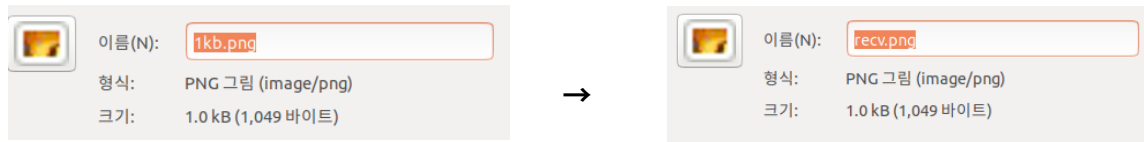
Rdt 3.0 에 파이프라이닝까지 구현한 버전입니다. 파이프라이닝의 구현을 위해, tosendNum 이라는 변수를 설정하였습니다. 해당 변수는 현재 sender 가 보낼 수 있는 패킷의 수로, 윈도우 안에서 보내지 않은 패킷의 수를 뜻하기도 합니다. 무한루프 안에서, 만약 tosendNum 이 1 이상이라면, 새로운 데이터를 전송하고 tosendNum 을 1 줄입니다. 또한 해당 데이터는 윈도우의 다음 주소에 저장됩니다. 이것은 즉 윈도우를 이동하는 것입니다. 이러한 윈도우의 갱신이 일어날 때마다 wd_idx(다음 데이터를 저장할 윈도우의 index)와

front(윈도우의 제일 앞 인덱스) 또한 갱신합니다. 해당 데이터의 **seq_num** 또한 **ack** 판단을 위해 **wd_seq** 리스트의 같은 주소에 저장됩니다. 또한 윈도우가 이동할 때마다 제일 앞 데이터에 대한 타이머를 매번 설정합니다. 또한 **seq_num**의 크기는, 윈도우가 5~10이기 때문에, 10의 2배인 20으로 설정하였습니다(0~19). 또한 파이프라이닝에서는 **recv**가 연속적으로 일어날 수 있기 때문에, (즉 전송->수신->전송->수신이 아닌, 전송 여러 개 → 수신 여러 개가 될 수 있기 때문에) **recv**용 쓰레드를 만들었습니다. 해당 쓰레드에서는 **ack**를 계속적으로 수신하는데, 만약 정상적인 **ack**를 받았다면 데이터를 저장하고, **front**를 **(front+1)%window_s**(윈도우크기)로 갱신합니다. 또한 윈도우를 옆으로 이동시킨 것이기 때문에 보낼 수 있는 것이 하나 늘어나기 때문에, **tosendNum**을 1 증가시킵니다. Go-back N 파이프라이닝에서는, 윈도우를 이동시키면서 타이머 또한 바로 재시작 해야하기 때문에 **settimeout(None)**을 통한 타이머 정지는 없으며, 계속하여 타이머가 0.01 초로 재설정됩니다. 또한 **timeout** 예외 시 **retransmit** 변수를 1로 설정하면, 메인 쓰레드에서 윈도우 내 모든 데이터를 재전송하고 **retransmit** 변수를 0으로 되돌립니다. 파일의 마지막을 판단하는 것 또한 파이프라이닝이 없을 때와는 다릅니다. 파이프라이닝이 없을 때는, 파일의 마지막을 알리는 패킷이 가고 **ack**가 오면 끝이지만, 파이프라이닝이 있다면 파일의 마지막을 알리는 패킷 및 **ack**이 오고 갔더라도, 그것보다 먼저 전송된 파일의 실제 데이터가 **packet loss** 됐을 수도 있기 때문입니다. 따라서 파일의 마지막을 알리는 패킷이 윈도우를 모두 채웠을 때, 파일의 전송이 모두 완료된 것이라고 판단하였습니다. 파일의 마지막을 알리는 데이터를 'finpack123'이라고 설정하였으며, 만약 윈도우가 모두 finpack123으로 찼다면, 더 이상 **ack** 받지 못한 파일의 실제 데이터가 없는 것이기 때문에 'endfinish'라는 패킷을 전송해주어 파일의 전송이 끝났음을 알려줍니다. 해당 패킷에 대한 **ack**는 100으로 설정하였으며, sender가 **ack 100**을 받게 되면 종료하게 됩니다. receiver의 경우, 만약 받아야 할 **seq_num**이 오지 않고 다른 **seq_num**을 가진 패킷이 도착할 경우, 계속해서 가장 최근의 **ack**인 **prv_ack**을 전송합니다. 파이프라이닝의 receiver에서 앞의 receiver들과 다른 점은, finpack에 대한 구분입니다. Finpack123 패킷을 받으면, 해당 패킷의 **seq_num**과 같은 **ack**를 계속해서 알려줍니다. 최종적으로 'endfinish' 패킷을 받으면, **ack 100**을 보낸 후 종료합니다.

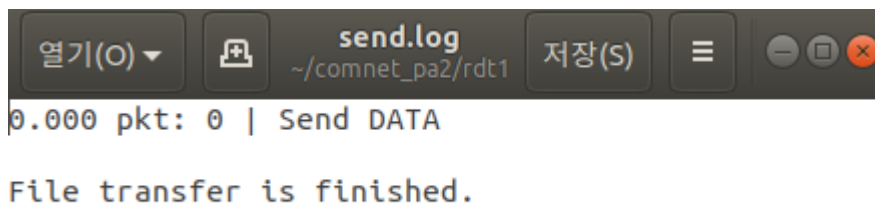
3. Implementation

- Describe which functionalities were successfully implemented. You may attach partial screenshots of output (log files, printed logs in terminal, etc.) that effectively show your program is coping with each situation.

■ P1: RDT 1.0



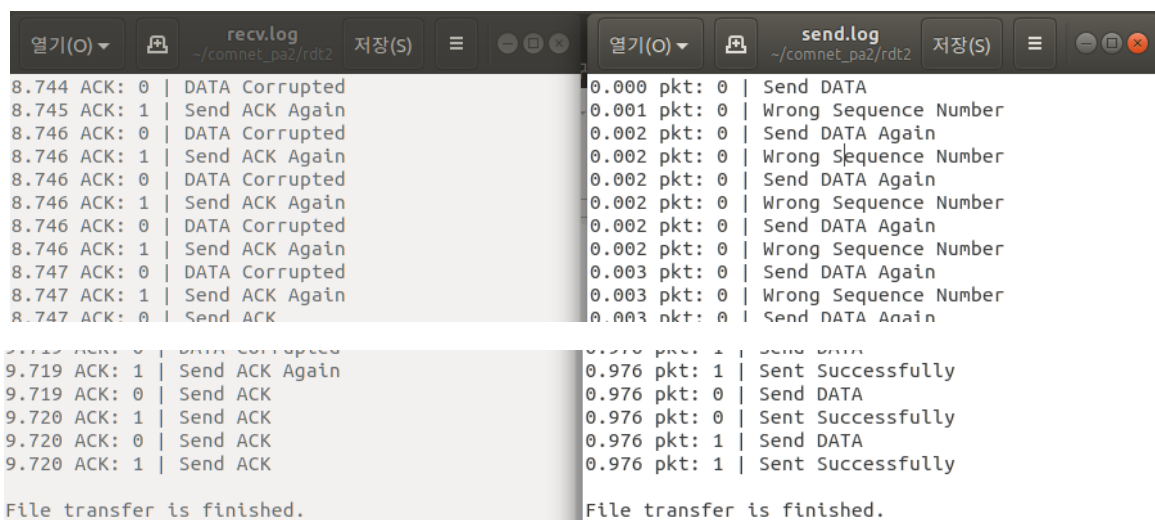
corrupt rate, packet loss rate = 0




■ P2: RDT 2.2



corrupt rate=0.8, packet loss rate = 0



■ P3: RDT 3.0




이름(N): test.pdf

형식: PDF 문서 (application/pdf)

크기: 26.0 MB (25,959,712 바이트)





이름(N): recv.pdf

형식: PDF 문서 (application/pdf)

크기: 26.0 MB (25,959,712 바이트)

corrupt rate=0.2, packet loss rate = 0.2

열기(O) ▾	recv.log ~/comnet_pa2/rdt3	저장(S)	≡	열기(O) ▾	send.log ~/comnet_pa2/rdt3	저장(S)	≡	⌵	⌵	⌵	⌵
3.115 ACK: 0	DATA Corrupted			0.000 pkt: 0	Send DATA						
3.116 ACK: 1	Send ACK Again			0.001 pkt: 0	Wrong Sequence Number						
3.126 ACK: 0	Send ACK			0.012 pkt: 0	TIMEOUT						
3.127 ACK: 1	Send ACK			0.012 pkt: 0	Send DATA Again						
3.127 ACK: 0	Send ACK			0.012 pkt: 0	Sent Successfully						
3.148 ACK: 1	Send ACK			0.012 pkt: 1	Send DATA						
3.149 ACK: 0	Send ACK			0.012 pkt: 1	Sent Successfully						
3.161 ACK: 1	Send ACK			0.013 pkt: 0	Send DATA						
3.161 ACK: 0	Send ACK			0.013 pkt: 0	Sent Successfully						
3.162 ACK: 1	Send ACK			0.013 pkt: 1	Send DATA						
3.163 ACK: 0	Send ACK			0.023 pkt: 1	TIMEOUT						
3.163 ACK: 1	Send ACK			0.023 pkt: 1	Send DATA Again						
3.164 ACK: 0	Send ACK			0.033 pkt: 1	TIMEOUT						
3.165 ACK: 1	Send ACK			0.034 pkt: 1	Send DATA Again						
3.166 ACK: 0	DATA Corrupted			0.034 pkt: 1	Sent Successfully						
180.901 ACK: 0	Send ACK			177.800 pkt: 1	Send DATA						
180.962 ACK: 1	Send ACK			177.871 pkt: 1	TIMEOUT						
180.962 ACK: 0	Send ACK			177.872 pkt: 1	Send DATA Again						
180.963 ACK: 1	Send ACK			177.872 pkt: 1	Wrong Sequence Number						
180.974 ACK: 0	Send ACK			177.883 pkt: 1	TIMEOUT						
180.986 ACK: 1	DATA Corrupted			177.884 pkt: 1	Send DATA Again						
180.986 ACK: 0	Send ACK Again			177.885 pkt: 1	Sent Successfully						
180.998 ACK: 1	Send ACK			177.885 pkt: 0	Send DATA						
180.999 ACK: 0	Send ACK			177.886 pkt: 0	Sent Successfully						
File transfer is finished.				File transfer is finished.							

■ P4: RDT 3.0 + pipelining

corrupt rate=0.2, packet loss rate = 0.2

The image shows two file transfer windows. The left window shows a file named 'test.pdf' (26.0 MB) being received. The right window shows a file named 'recv.pdf' (26.0 MB) being received. Below these are two log windows. The left log window shows the receiver's log (recv.log) with entries for ACKs and Send ACKs. The right log window shows the sender's log (send.log) with entries for Send DATA and packet numbers. Both logs indicate that the file transfer is finished.

이름(N): test.pdf
형식: PDF 문서 (application/pdf)
크기: 26.0 MB (25,959,712 바이트)

→

이름(N): recv.pdf
형식: PDF 문서 (application/pdf)
크기: 26.0 MB (25,959,712 바이트)

recv.log
1.733 ACK: 0 | Send ACK
1.734 ACK: 1 | Send ACK
1.734 ACK: 2 | Send ACK
1.745 ACK: 2 | Send ACK
1.755 ACK: 3 | Send ACK
1.755 ACK: 4 | Send ACK
1.756 ACK: 4 | Send ACK
1.767 ACK: 5 | Send ACK
197.703 ACK: 11 | Send ACK
197.704 ACK: 11 | Send ACK
197.704 ACK: 11 | Send ACK
File transfer is finished.

send.log
0.001 pkt: 0 | Send DATA
0.001 pkt: 1 | Send DATA
0.002 pkt: 2 | Send DATA
0.002 pkt: 3 | Send DATA
0.002 pkt: 4 | Send DATA
0.002 pkt: 5 | Send DATA
0.002 pkt: 6 | Send DATA
0.003 pkt: 7 | Send DATA
195.896 pkt: 9 | Send DATA
195.911 pkt: 10 | Send DATA
195.922 pkt: 11 | Send DATA
File transfer is finished.

파이프라인의 경우, log 를 정확히 구현하지 못하였습니다. 그러나 파일 전송은 정상적으로 됩니다.

4. Execution Screenshot

- **Important:** Insert a screenshot that explicitly shows your program is working correctly with your execution command. TAs will run your program based on this information. (Capturing the initialization part is enough, not the entire operation of scenarios)

The image shows two terminal windows. The top window shows the command 'python receiver1.py recv.png recv.log' being executed. The bottom window shows the command 'python sender1.py 127.0.0.1 1 kb.png send.log' being executed.

```
2017314626jh@jh-PC: ~/comnet_pa2/rdt1
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
2017314626jh@jh-PC:~/comnet_pa2/rdt1$ python receiver1.py recv.png recv.log
2017314626jh@jh-PC:~/comnet_pa2/rdt1$
```

```
2017314626jh@jh-PC: ~/comnet_pa2/rdt1
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
2017314626jh@jh-PC:~/comnet_pa2/rdt1$ python sender1.py 127.0.0.1 1 kb.png send.log
2017314626jh@jh-PC:~/comnet_pa2/rdt1$
```

*제출 할 때는 폴더 및 코드 이름을 P1~P4 와 sender.py, receiver.py 로 수정하여 제출하였습니다. 실행 또한 해당 명으로 가능합니다.

```
2017314626jh@jh-PC: ~/comnet_pa2/rdt2
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
2017314626jh@jh-PC:~/comnet_pa2/rdt2$ python receiver2.py recv.jpg recv.log
starting
received data
end
2017314626jh@jh-PC:~/comnet_pa2/rdt2$

2017314626jh@jh-PC: ~/comnet_pa2/rdt2
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
2017314626jh@jh-PC:~/comnet_pa2/rdt2$ python sender2.py 127.0.0.1 1 1mb.jpg send
.log
2017314626jh@jh-PC:~/comnet_pa2/rdt2$
```

```
2017314626jh@jh-PC: ~/comnet_pa2/rdt3
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
2017314626jh@jh-PC:~/comnet_pa2/rdt3$ python sender3.py 127.0.0.1 9 test.pdf sen
d.log
2017314626jh@jh-PC:~/comnet_pa2/rdt3$

2017314626jh@jh-PC: ~/comnet_pa2/rdt3
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
2017314626jh@jh-PC:~/comnet_pa2/rdt3$ python receiver3.py recv.pdf recv.log
starting
received data
end
2017314626jh@jh-PC:~/comnet_pa2/rdt3$
```

pipelining 이 아닐때는, sender에서 window 인자를 9로 주더라도 코드에서는 window 1로
설정되어 작동합니다.

```
2017314626jh@jh-PC: ~/comnet_pa2/rdt4
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
2017314626jh@jh-PC:~/comnet_pa2/rdt4$ python sender4.py 127.0.0.1 9 test.pdf sen
d.log
파일 전송 완료
파일 전송 완료
파일 전송 완료
파일 전송 완료
파일 전송 완료
파일 전송 완료
파일 전송 완료
파일 전송 완료
파일 전송 완료
2017314626jh@jh-PC:~/comnet_pa2/rdt4$

2017314626jh@jh-PC: ~/comnet_pa2/rdt4
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
2017314626jh@jh-PC:~/comnet_pa2/rdt4$ python receiver4.py recv.pdf recv.log
starting
real finished
received data
end
2017314626jh@jh-PC:~/comnet_pa2/rdt4$
```