

Computer Network HW 4

학번 / 이름: 2014310407 / 이 준 혁

1. 개발 환경

- OS: Windows 10 64bit
- Programming language: Python 3.7.1 – **64bit**
- IDE: Pycharm 2018.1.4

2. 구현

2.1 구현 내용 요약

2.1.1. sender 가 패킷을 window size 개수만큼 보내고, ACK 을 받아서 다음 패킷을 보내는 것 구현 후 로그 파일에 해당 내용들 기록

2.1.2. Goodput 을 계산 후 로그 파일에 기록

2.1.3. 입력 받은 확률만큼 패킷을 drop 시키고 로그파일에 drop 된 내용을 기록

2.1.4. Timer 를 이용해 Timeout 을 detect 하고 재 전송. 이 내용을 로그 파일에 기록

2.1.5. 3 Duplicate ACK 을 받으면 다음 패킷 전송. 이 내용을 로그 파일에 기록

2.1.6. Server 와 receiver 가 각각 concurrent 하게 파일을 주고받을 수 있도록 구현

2.2 구현 세부 설명

2.2.0. 함수 설명

- getfile 함수: file 이름과 file 을 여는 방법에 대한 인자를 받아, 파일을 열어 그 포인터를 반환한다. 만약 해당 파일을 여는 데 문제가 있는 경우 0 을 반환한다.
- addheader 함수: 패킷 숫자(sequence number)를 고정된 크기의 헤더에 넣기 위한 작업으로, 패킷 숫자와 뒤에 이어 붙일 데이터를 입력으로 받아 헤더를 입력숫자+W0W0W0WW0... 꼴로 만든다. 이후 이 크기가 48(헤더의 크기)이 될 때까지 반복 후 데이터 앞에 이어 붙여서 반환한다.
- makeheader 함수: addheader 함수와 유사하나 데이터의 이어 붙임 없이 헤더만 반환한다.
- logfilewrite 함수: 로그 파일 포인터, 패킷 / ACK 여부, 패킷 번호, 로그파일에 기록할 때의 시간, 상태(received, sent, timeout..)등을 입력으로 받아 로그 파일에 쓴다.
- filetransfer, filereceive 함수: 각각 sender.py, receiver.py 에서 concurrent 하게 동작하는 파일 전송 / 수신함수.

2.2.1. Sender.py

사용한 패키지 및 함수: socket 의 모든 함수, threading 패키지, time 패키지, sys 의 getsizeof 함수

우선 port number 는 10080, 패킷을 보낼 때 쓰일 buffersize 는 1400, 헤더를 만들 때 쓰일 headersize 는 48 로 고정했다. 그리고 메인 함수가 실행되면, IP address, Timeout value, window size 를 입력으로 받아 값을 저장했다. 이후 반복적으로 파일 이름을 받아서, 파일이름과 위의 IP, timeout 값, window 크기를 변수로 가지는 filetransfer 함수를 멀티스레딩으로 병렬적으로 실행했다.

filetransfer 함수는 인자로 입력 받은 file 이름을 바탕으로 해당 파일을 receiver 에게 전송하는 함수이다. 각 파일마다 전송해야 할 소켓이 필요하므로 socket 을 UDP 로 열었다. 이후 파일을 열어서, 패킷 단위로 쪼개는데, 이때 sequence number 를 저장할 header 를 별도로 앞에 붙여야 하므로 header size 를 제외한 크기만큼 읽은 후 저장했다.

이후 파일 이름을 전송하고, 마지막 파일에 대한 ACK 를 받을 때 까지 다음을 반복했다.

1. Timer 를 설정한다.
2. 가장 마지막으로 ack 를 받은 다음 패킷들을 윈도우 크기의 개수만큼 패킷을 보낸다.
3. 마지막으로 보낸 패킷에 대한 ACK 가 오기 전까지 Receiver 에게서 ACK 를 계속 받는다.
4. 3.에서 Timeout 이 발생하는 경우, Timeout 을 기록하고 Timer 를 재설정 한 후, Timeout 이 일어난 sequence number 에 대한 패킷을 재전송 후 기록한다. 만약 여기서도 Timeout 이 발생한다면 Timer 를 설정하고 재전송하는 과정을 반복한다. 재전송된 패킷에 대한 올바른 ACK 가 올 때 까지 반복한다. 제대로 된 ACK 를 받는다면 2.로 간다.
5. 3.에서 Timeout 이 발생하지 않았다면 받은 패킷을 기록한 후 이전에 받은 패킷과 같은 지 확인한다. 그 후 그렇게 같은 패킷을 받은 것이 세 번인지 확인한다.
6. 5.에서 맞다고 나온 경우 duplicate ack 을 기록한다. 그리고 추가적으로 수신될 duplicate Ack 를 받기 위해 timer 를 설정해서 그 기간동안 오는 Ack 를 다 받고, 무시한다. 이후 다음 packet 을 보내기 위해 1 로 돌아간다.
7. 5.에서 아니라고 나온 경우 받은 ACK 가 가장 최근에 받은 ACK 보다 큰 지 판단한다. 받은 ACK 가 더 크다면, 그 값을 가장 최근에 받은 ACK 로 설정하면 되지만, 아닌 경우에는 무시한다.

1~7 이 끝나면, receiver 에게 끝났다는 메시지를 보낸다. 이후 Goodput 과 파일 전송 종료를 기록 후 파일과 소켓을 모두 닫는다.

2.2.2. Receiver.py

사용한 패키지 및 함수: socket 의 모든 함수, threading 패키지, random 패키지, time 패키지, sys 의 getsizeof 함수

Sender.py 와 같이 buffersize, headersize, port 번호를 고정했고, 추가적으로 전역 변수인 loss 확률과 demultiplexing 을 위한 shrdque 를 선언했다.

main 함수가 실행되면, loss 확률을 입력으로 받는다. (잘못된 경우 프로그램 종료) 이후 receiver 의 모든 통신에 필요한 socket 을 하나 연다. 포트 번호를 10080 으로 바인딩한 후, 소켓의 buffer size 를 출력하고 이후 1000000 으로 변경한다.

이후 반복문을 통해 socket 을 통해 입력을 받는다. 이 때의 입력은 각기 다른 스레드에서 오는 것이기 때문에, 다음과 같은 작업을 통해 demultiplexing 을 수행해 주었다. 우선 각 스레드에서 온 address 정보를 저장하는 addrque 를 만들었다.

1. 보낸 주소에서 처음으로 전송한 경우; 즉, addrque 에 주소가 없는 경우, addrque 에 주소를 append 하고, 이 때 처음 온 값은 파일 명이므로 파일 명을 인자로 해서 filereceive 함수를 멀티스레딩으로 concurrent 하게 실행했다.
2. 보낸 주소에서 전송한 이력이 있는 경우; 즉, addrque 에 해당되는 주소가 있는 경우, shrdque 에 보낸 내용과 주소를 append 했다.

filereceive 함수는 인자로 입력 받은 file 이름을 바탕으로 해당 파일을 sender 에게 전송 받는 함수이다. 파일과 로그 파일을 연 후, 다음과 같은 방식을 통해 전송 받은 파일을 함수로 가져왔다. Shrdque 에 있는 튜플들을 살펴보면, 만약 주소 값이 해당 함수가 인자로 받은 주소 값과 일치한다면 데이터를 가져오고, 그 값을 shrdque 에서 지운다. 이후 이 값에 대해 작업을 수행한다.



만약 받은 데이터가 파일 전송의 끝을 알리는 내용이라면, 입력을 더 이상 받지 않았다. 아닌 경우, header 를 분리해서 sequence number 를 알아냈다. (이 때 python **32bit** 인 경우 정상 작동하지 않을 수 있기에, 64bit 인 점을 명시했습니다.) 이 패킷을 받았다는 것을 기록한 후, loss 를 결정했다. loss 값이 임의로 선택한 값보다 작은 경우, drop 이 발생했다고 판단해서 기록 후 아무것도 수행하지 않고 continue 를 통해 다시 데이터를 받기로 했다.

Drop 이 나지 않은 경우는 다음과 같은 알고리즘을 통해 행동을 달리했다.

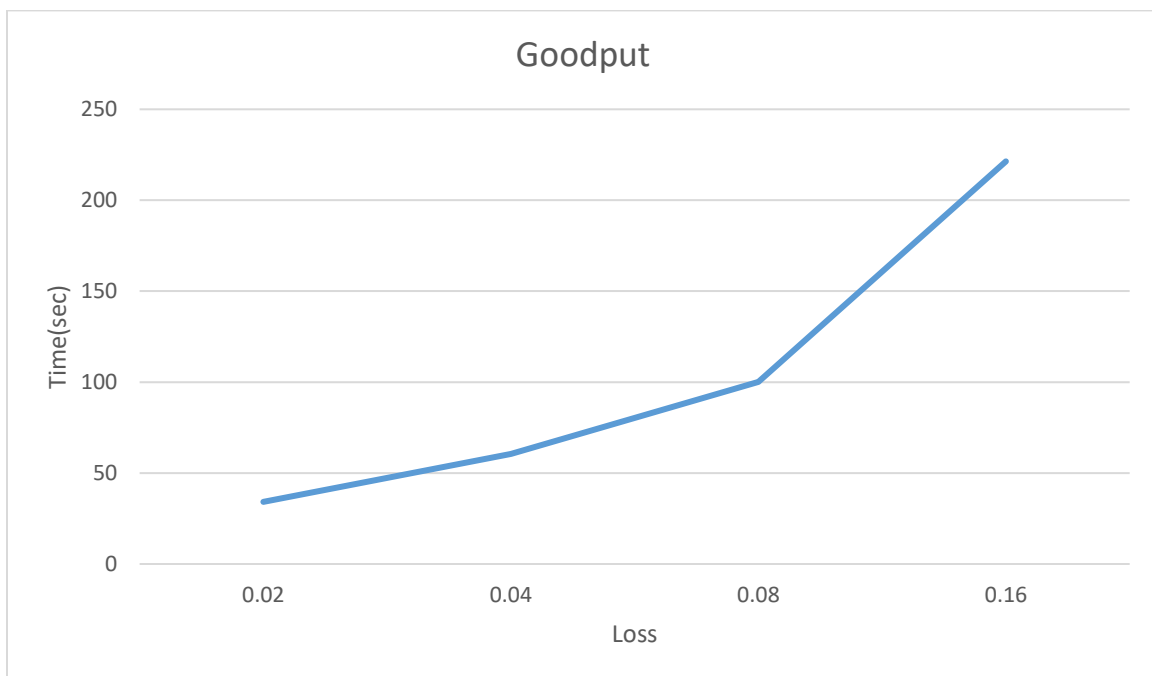
1. 받은 sequence number 가 가장 최근에 보낸 ACK 보다 1 큰 경우: 정상적으로 ACK 가 온 것이다. 그러므로 받은 파일을 write 한 후, 버퍼에 저장되어 있는 제대로 받았지만 out of order 라서 아직 쓰이지 않은 데이터들을 하나씩 쓰면서, ACK 값을 하나씩 증가시킨다. 이를 버퍼가 비거나 버퍼안의 데이터의 sequence number 가장 최근 ACK 값보다 1 초과로 큰 경우까지 시행했다.
2. 받은 sequence number 가 가장 최근에 보낸 ACK 보다 같거나 작은 경우: 이 경우 정상적으로 보낸 ACK 에 대해서 다시 패킷이 온 것이므로, 가장 최근에 보낸 ACK 를 보내 주고, 로그 파일에 기록했다.
3. 받은 sequence number 가 가장 최근에 보낸 ACK 보다 1 초과로 큰 경우: 앞의 패킷에서 drop 이 일어난 것이고, 때문에 sequence number 와 데이터를 버퍼에 저장한다. sender 에게는 가장 최근에 보낸 ACK 를 보내준다.

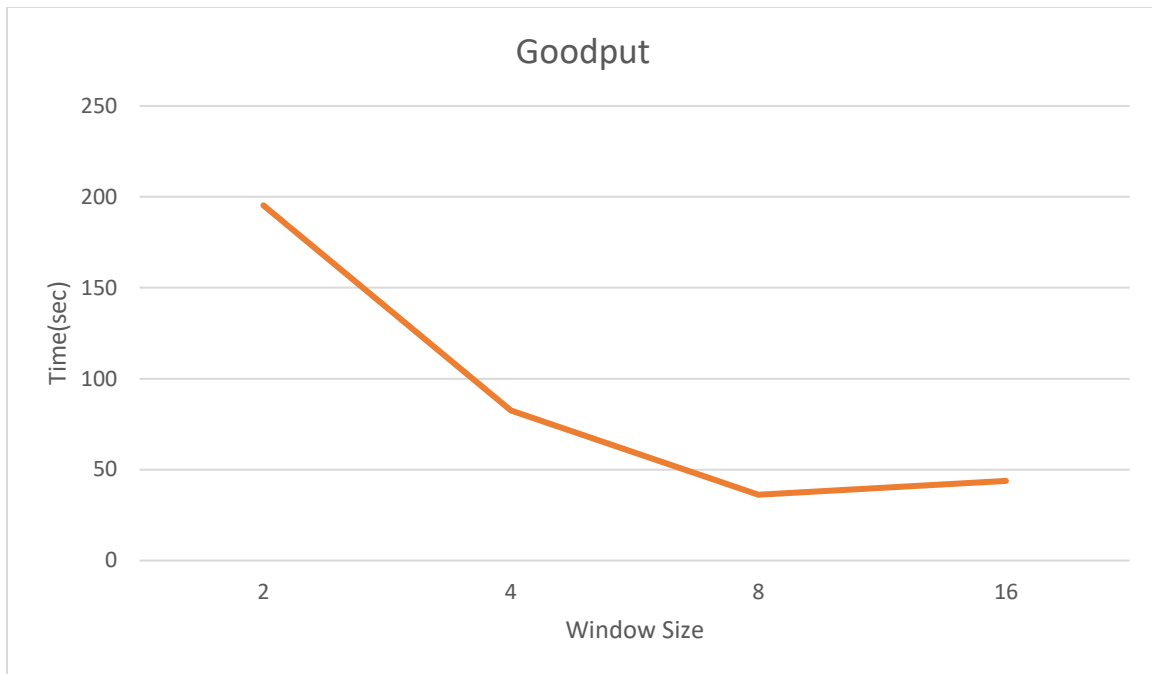
이 과정을 위에 기술한 바와 같이 sender 쪽에서 끝났음을 알리는 메시지가 올 때까지 반복했다. 이후 goodput 을 계산하고 로그 파일에 기록한 후 파일 포인터를 닫고 종료했다.

3. 실습(구현 결과)

 data.jpg		2018-11-20 오후...	JPG 파일	13,099KB
--	---	------------------	--------	----------

- 12.7 MB





4. 실행 방법

```

receiver x
C:\Users\LeeJunHyuk\AppData\Local\Programs\Python\Python37\python.exe C:/Users/LeeJunHyuk/Desktop/pythin/python/receiver.py
packet loss probability :0.02
socket recv buffer size : 65536
socket recv buffer size updated : 10000000
data.jpg
  
```

1. 다음과 같이 receiver program 을 실행시킨 후 packet loss probability 를 입력한다.

```

sender x
C:\Users\ljhjo\AppData\Local\Programs\Python\Python37\python.exe "C:/Users/ljhjo/OneDrive/coding/Python/2018 Fall Network/server4/sender.py"
Receiver IP address: 192.168.0.14
Timeout : 0.05
Window size : 2
Filename : data.jpg
Filename :
  
```

2. 다음과 같이 sender program 을 실행시킨 후 IP, window size, timeout 값을 입력한다.

3. Filename 을 입력하면 전송이 시작된다.

