

Computer Network HW 5

학번 / 이름: 2014310407 / 이 준 혁

1. 개발 환경(both sender and receiver)

- 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 는 IP 주소와 시작 window 크기를 입력 받아 해당 크기를 시작으로 패킷을 receiver 에게 내보내고, congestion 에 따른 적절한 control 을 하며, 로그 파일에 이를 기록.

2.1.2 receiver 는 BLR 과 queue_size 를 입력 받음

2.1.3 NEM 은 주어진 BLR 의 속도로 sender 에게 받은 패킷을 receiver module 에게 내보냄 이 내용을 로그 파일에 기록.

2.1.4 RM 은 패킷을 받아 TCP 와 같은 형식으로 sender 에게 cumulative ACK 을 전송. Jain Fairness index 와 각 sender 에게 받은 receiving rate 를 계산 후 로그 파일에 기록

2.1.5 Multiple sender 를 통해 실행하더라도 문제 없이 동작되도록 Concurrent 하게 구성

2.2 구현 세부 설명

2.2.1. Sender.py

2.2.1.1 함수 및 모듈 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.

- logfilewrite 함수: 현재 시간, average RTT, sending rate, goodput 을 로그 파일에 쓴다.

- SEM 함수: drop 과 duplicate ack 을 감지해 이에 알맞은 congestion control 을 하며, packet 을 receiver 에게 보내는 함수이다.

- 사용한 모듈 및 함수: socket 의 모든 함수, threading 모듈의 Thread 함수, time 모듈, sys 의 getsizeof 함수, os 모듈의 _exit 함수

2.2.1.2 내용 설명

저번 과제와 동일하게 port number 는 10080, 패킷을 보낼 때 쓰일 buffersize 는 1400, 헤더를 만들 때 쓰일 headersize 는 48 로 고정했다. 추가적으로 sending rate, goodput, average RTT 를 저장 할 전역변수 리스트를 만들고 이를 이용해 값에 access 하며 계산했다.

메인 함수가 실행되면, 1400 이라는 크기의 패킷을 만들기 위해 W0 으로만 이루어진 데이터를 만들었다. 그 후 IP address, start window size 를 입력으로 받아 값을 저장했다. 이후 로그 파일을 열고, 소켓을 만들어서 바인딩 후 연결을 알리는 메시지를 보냈다. 그리고 로그 파일에 쓰기 위한 쓰레드(logfilewrite)와 패킷을 보내기 위한 쓰레드(SEM)을 만들었다. 이후 메인 함수 내부에서는, input 함수를 이용해 입력을 대기하다가 stop 으로 입력이 들어오면 receiver 에게 종료를 알리고 os 모듈의 _exit 함수를 이용해 프로그램을 종료시켰다.

SEM 함수는 실행되면 우선 샘플 패킷을 보내고 주고받음으로 sample rtt 를 계산했다. 이후 최초 devRTT 는 0.01 으로 두고 timeout 값을 계산했다. 그리고 나서 반복적으로 다음과 같이 함수를 실행했다. 여기서 window 는 보낸 패킷들의 sequence number 를 저장하는 리스트이고, is_first_pkt 은 timeout 이후에 최초로 보내는 패킷일 때 1 을, 아닌 경우에는 0 으로 설정해 놓는다. 또한 is_first_pkt_sent 를 통해 타임아웃 이후에 패킷을 보냈는 지 여부를 알 수 있다. 이 때 보낸 첫 번째 패킷의 값을 sampleAck 에 저장해 둔다.

1. 만약 윈도우가 윈도우 크기만큼 다 차지 않았다면, 다음 패킷을 보낸다. 이 때 윈도우 크기가 0 인 경우에는 가장 최근에 받은 ack 의 다음 값으로, 윈도우에 보낸 패킷의 sequence number 가 남아 있는 경우에는 그에 따라 가장 최근에 보낸 패킷의 다음 패킷을 보낸다. 만약 현재 보내려고 하는 패킷이 timeout 이후에 일어난 패킷이라면 timeout 을 재설정한다. 이 과정은 윈도우가 다 찰 때까지 반복한다. 패킷을 보낼 때 마다, sending rate 를 1 증가시킨다.
2. 이후 소켓을 통해 receiver 로부터 ack 를 대기한다. 여기서 timeout 이 일어난 경우 10.로 간다.
3. Timeout 이 일어나지 않은 경우, 만약 2.번에서 받은 ack 가 timeout 이후에 처음으로 보낸 패킷이라면, 시간을 측정해 rtt 와 devrtt 를 계산해 이를 바탕으로 avgrtt 를 계산한다.
4. 또 만약 지금 시간이 timeout 이후에 첫 번째로 보낸 패킷의 예상 도착 시간보다 뒤에 있다면, 첫 번째 패킷에 대한 timer 가 끝난 것이므로 timer 를 갱신하고, window size 를 1 증가시킨다.
5. 만약 4.번이 아닌 경우에는, slow start 를 위해 ssthreshold 값인 8 이하인 경우에는 윈도우 크기를 두 배로 증가시킨다.(즉, 1,2,3,4 인 경우에만 두 배로 늘리는 것)
6. 이후 받은 ack 가 현재 윈도우의 첫 번째 값과 같다면, 즉 in-order 한 ack 를 받았다면, 해당 값을 window 에서 pop 하고, 1 로 돌아간다. 아닌 경우 7.로 간다.
7. 받은 ack 가 윈도우의 첫 번째 값보다 크다면, cumulative ack 에 의해 window 의 현재 받은 값보다 작은 값들은 모두 정상 송신되었음을 알 수 있다. 따라서 받은 ack 에 해당하는 값이 나오거나 윈도우가 빌 때까지 pop 한 후에, 1 로 돌아 간다. 아닌 경우 8.로 간다.

8. 만약 6, 7 모두 아니면서 현재 받은 ack 가 가장 최근에 받은 ack 와 같다면, duplicate ACK 의 값을 하나 증가시킨다. duplicate ack 이 3 이 되었다면, 윈도우를 초기화하고 window size 를 반으로 줄인다. 이후 timeout 의 절반 시간동안 나머지 ack 을 받고, 1 로 돌아간다. 이 마저도 아니라면 9.로 간다.
9. 받은 값이 6,7,8 어느 값도 아니라면 의미 없는 ack 이므로 2 로 돌아가서 다시 수신을 대기한다.
10. 만약 2 번에서 timeout 이 일어난 경우 congestion 이 심한 경우이므로 window size 를 1 로 줄이고 윈도우를 초기화한다. 이후 1 로 돌아간다.

congestion control 을 요약하자면, 4 이하인 경우에는 ack 를 받자마자 바로 window 크기를 두 배로 늘리고, 이후에는 매 RTT 마다 window 크기를 하나씩 늘린다. 이후에 만약 duplicate ack 이 발생한 경우 window 크기를 절반으로 줄이고 window 를 초기화하고, 아예 timeout 이 발생하면 window 크기를 1 로 줄이고 window 를 초기화한다.

2.2.2. Receiver.py

2.2.2.1 함수 및 모듈 설명

- avgquecal 함수: que utilization 을 매 0.1 초마다 계산해서 저장하는 함수
- 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.
- NEMque 함수: NEM 의 queue 가 비지 않았을 경우, 해당 queue 를 RM 이 접근할 수 있는 dictionary 에 옮기는 함수. 1 / BLR 을 입력으로 받아, 매 1 / BLR 초마다 시행한다. (즉, BLR 이 50 이라면 매 1/50 초마다 실행됨.)
- jfindex 함수: 리스트를 입력으로 받아, 리스트의 Jain Fairness Index 를 계산해서 반환한다. 만약 list 가 비었거나 모든 값이 0 이라면 NA 를 반환한다.

- logfilewriteNEM 함수: 현재 시간, incoming rate, sending rate, average queue

utilization(avgqueecal 함수를 통해 계산)을 로그 파일에 쓰는 함수이다.

- logfilewriteRM 함수: 현재 시간, Jain Fairness Index, 각 ip 와 포트에서의 receiving rate 를 로그 파일에 쓰는 함수이다.

- RM: receiver module 으로, NEM 으로부터 받은 패킷을 TCP manner cumulative ack 를 전송한다. sender 가 한 명 연결 될 때마다 multithreading 으로 하나씩 실행된다.

- 사용한 모듈 및 함수: socket 의 모든 함수, threading 모듈의 Thread 함수, time 모듈

2.2.2.2 내용 설명

Sender.py 와 같이 buffersize, headersize, port 번호를 고정했고, 추가적으로 다음과 같은 전역 변수를 선언했다.

1. nemque: NEM 에서 RM 으로 내보내기 전에 패킷들을 저장하는 큐.

2. avgque: que utilization 이 계산되어서 저장되어 있는 리스트. 평균 queue utilization 을 계산할 때 사용한다.

3. shrddic: key 값은 (ip: port)이고, value 값은 리스트로써 현재 RM 에 전송된 packet 이다. 다만 value 의 첫 번째 값은 현재 해당 ip 와 port 로 들어온 패킷 수, 즉 receiving rate 이다.

4. IN_FO: incoming rate 와 forwarding rate 를 저장하는 list 이다.

main 함수가 실행되면, BLR 과 queue size 를 입력으로 받는다. 이후 receiver 의 모든 통신에 필요한 socket 을 하나 연다. 포트 번호를 10080 으로 바인딩한 후, 소켓의 buffer size 를 1000000 으로 변경한다.

그리고 멀티쓰레딩을 이용해 BLR 의 속도로 NEM 에서 RM 으로 패킷을 전송하는 NEMque, average que utilization 을 저장하는 queSize 함수를 concurrent 하게 실행한다. 또한 RM, NEM 의 로그 파일을 쓰는 두 함수를 시행한다. 그리고 메인 함수 안에 NEM 을 다음과 같이 실행했다.

socket 으로부터 입력을 받은 다음에 packet 의 sequence number 만 뽑아 냈다. 이후 이 값이 무엇인지에 따라 다음과 같이 demultiplexing 했다. 또한 sender 들의 (ip, port) 튜플, 즉 address 를 보관하기 위해 addrque 를 만들었다.

1. 새로운 sender 가 연결을 시작한 경우: addrque 에 새로운 주소를 넣고, 그 주소에 해당하는 RM 을 멀티쓰레딩으로 실행했다. (sender 하나당 RM 함수 하나 실행)
2. 기존 sender 가 연결을 종료한 경우: addrque 에서 해당 주소를 지우고, 이를 RM 에 즉각적으로 전송해 줬다.
3. 1,2 가 아닌 경우, 즉 제대로 된 packet 이 온 경우: NEM 에서 RM 으로 전송하는 queue 에 받은 패킷을 저장했다.

위 1,2,3 을 계속 반복함으로써 NEM 을 구현했다.

반면 RM 은 멀티쓰레딩으로 실행되는 함수로서 구현했는데, 우선 입력으로 socket 과 주소를 받았다. 이후 shrddic 을 통해 입력을 처리했는데, 만약 그 값이 연결 종료면 shrddic 에서 자신의 주소를 지우고 종료했고, sample RTT 를 측정하는 거라면 그에 맞게 반응했다. 둘 다 아니라 sequence number 를 받았다면, 다음과 같은 알고리즘으로 행동했다.

1. 받은 sequence number 가 가장 최근에 보낸 ACK 보다 1 큰 경우: 정상적으로 패킷이 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 주었다.
3. 받은 sequence number 가 가장 최근에 보낸 ACK 보다 1 초과로 큰 경우: 앞의 패킷에서 drop 이 일어난 것이고, 때문에 sequence number 를 버퍼에 저장한다. sender 에게는 가장 최근에 보낸 ACK 를 보내준다.

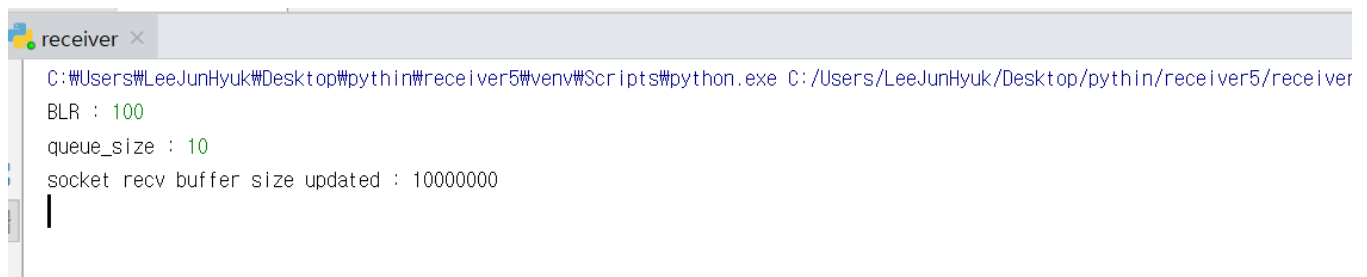
모든 과정에서 receiving rate 는 1 씩 증가시켜줬다. 이 과정을 별도의 종료 조건 없이 반복적으로 구현했다.

3. 실습(구현 결과)

차트를 못 만들어서 로그 파일을 첨부했습니다. 죄송합니다..

4. 실행 방법

1. 다음과 같이 receiver program 을 실행시킨 후 BLR 과 queue_size 를 입력한다. receiver program 이 실행된다.



```
receiver x
C:\Users\LeeJunHyuk\Desktop\pythin\receiver5\venv\Scripts\python.exe C:/Users/LeeJunHyuk/Desktop/pythin/receiver5/receiver
BLR : 100
queue_size : 10
socket recv buffer size updated : 10000000
|
```

2. 다음과 같이 sender program 을 실행시킨 후 IP, Start window size 값을 입력한다. sender program 이 실행된다.

```
sender x
C:\Users\LeeJunHyuk\AppData\Local\Programs\Python\Python37-32\python.exe "C:/Users/LeeJunHyuk/OneDrive/coding/Python/2018 Fall Network/server5/sender.py"
Receiver IP address: 203.252.33.36
start initial window size : 5
Sending Start
samplertt : 0.048870086669921875
```

3. 다음과 같이 concurrent 하게 실행 가능하다.

```
sender x sender x
C:\Users\LeeJunHyuk\AppData\Local\Programs\Python\Python37-32\python.exe "C:/Users/LeeJunHyuk/OneDrive/coding/Python/2018 Fall Network/server5/sender.py"
Receiver IP address: 203.252.33.36
start initial window size : 5
Sending Start
samplertt : 0.05485272407531738
```

4. 마지막으로 다음과 같이 sender 프로그램에서 stop 을 입력하면 종료된다.

```
sender x sender x
C:\Users\LeeJunHyuk\AppData\Local\Programs\Python\Python37-32\python.exe "C:/Users/LeeJunHyuk/OneDrive/coding/Python/2018 Fall Network/server5/sender.py"
Receiver IP address: 203.252.33.36
start initial window size : 5
Sending Start
samplertt : 0.05485272407531738
stop

Process finished with exit code 0
```