Network Report

2017313260 이재민

1. Development environments

OS: windows 10, virtualbox 6.1.6 r137129 (Qt5.6.2) (given mininet)

Programming Language: python3

Version: python 3.7.4

파일 크기: 13024kb

2. code 설명

공통 함수 write(...): 로그 파일에 로그를 작성하는 함수.

<<sender.py>>

0. main

소켓을 연결해주고 각 인자들을 fileSender함수에 연결해준다.

1. fileSender: sender.py에서 파일을 전송하는 함수

send_file에 전송하고자 하는 파일 srcFilename 을 open

logFile에 작성하고자 하는 logfile open

보내고자 하는 데이터를 담을 packet_queue

1024크기 만큼 잘라서 packet_queue에 담는다.

startTime을 기록하고 send_handler class를 호출해준다.

클래스 내부함수 run_send로 전송을 시작한다.

2. make_packet: 전송하려는 데이터에 헤더를 붙인다.

헤더의 길이는 10으로 잡고, 이 안에 패킷 번호를 담아서 packet을 만든다.

class 선언

3. timer class: 타임아웃을 측정하고 RTT, RTO를 갱신할 수 있는 class이다.

A. class 내부 변수

sendtime = 각 패킷을 보낸 시간을 저장하는 list

retransmit = retransmit된 패킷을 확인하는 list

first_try = RTO, SRTT, RTTVAR이 첫 번째 시도에만 계산식이 다르므로 if문을 사용하기 위한 bool

B. 함수 save_sendtime

각 패킷을 저장한 시간을 sendtime[]에 저장한다. 만약 len(sendtime)와 패킷 번호가 같다면 새로운 패킷이기 때문에, append로 time을 저장한다. 그렇지 않다면 sendtime[ack]에 직접 접근해서 time을 저장한다.

C. 함수 time_out

현재 시간에서 sendtime에 저장되어 있는 '패킷을 보낸 시간'을 빼서, RTO보다 크면 타임아웃이 생긴 것이므로 RTO를 갱신하고 True를 반환한다. 그렇지 않다면 False를 반환한다.

RTO의 최대값은 60, 최소값은 0.2s로 잡았다.

D. 함수 RTTcal

retransmit된 packet이 아닐 경우에 표준 RTT 계산을 따라서 계산

4. send_handler: sender가 패킷을 보내고 ack를 받기 위한 class

A. class 내부 변수

sender_socket: 연결되는 socket

recvAddr: 연결되는 주소의 tuple

windowSize: windowsize

srcFilename: 전송하려는 파일 이름

dstFilename: 전송받는 파일 이름

logFile: 로그파일

packet_queue: 전송하기 위한 file을 패킷 단위로 저장해 놓은 queue

startTime: sender가 전송을 시작한 시간

recent_ack: receriver로부터 ack를 받고 갱신된 ack. window사이즈 만큼 보내는 시작지점.

duplicate_ack: duplicate를 확인하기 위한 int값

timer: timer class

lock: 패킷을 보낸 후에 ack를 받지않고 반복적으로 패킷을 계속 보내는 것을 막기 위한 bool값

retransmit_lock: retransmit 해주어야 하는지에 대한 bool값

B. send_to 함수: 패킷을 보내는 메인 함수 (0번부터 보낸다)

```
sender_socket.send(dstFilename.encode())
writePkt(self.logFile, self.startTime, -1, "sent")
break_time = time.time()
while self.recent_ack != -1:
    if time.time() - break_time > self.timer.RTO:
        sender_socket.send(dstFilename.encode())
        writePkt(self.logFile, self.startTime, -1, "sent")
        break_time = time.time()
```

파일 이름을 -1번 패킷으로 우선적으로 전송한다. 그래서 ack로 -1이 반환되면 while문을 탈출해서 전송을 멈추고 다음 과정으로 넘어간다.

가장 최근에 받은 self.recent_ack 를 recent_ack에 저장해둔다. (지속적인 갱신으로 인한 패킷손실을 막음)

```
if self.recent_ack == len(self.packet_queue) - 1:
    self.sender_socket.sendto(b'₩0', self.recvAddr)
    break
```

만약 self.recent_ack 이 len(self.packt_queue) – 1 이라면 ack로 마지막 패킷 번호를 받은 것이기 때문에 전송이 완료된 것이다. 그러므로 끝내는 packet을 전송하고 while문을 탈출한다.

```
if self.lock:
    if len(self.timer.sendtime) - 1 == recent_ack:
        self.lock = False
    if self.timer.time_out(recent_ack):
        writePkt(self.logFile, self.startTime, recent_ack + 1, 'timeout since {:.3f}
        self.timer.RTO = 2 * self.timer.RTO
        if self.timer.RTO >= 60:
            self.timer.RTO = 60
        self.retransmit_lock = True
        self.lock = False
```

lock이 True일 때, self.timer.sendtime에 담겨있는 개수와 recent_ack이 같다면 보낸 패킷에 대한 ack를 정상적으로 받은 것이기 때문에 lock을 False로 바꾼다.

만약 timeout이 생기면 이는 retransmit 해주어야 하므로 retransmit_lock을 True로 바꾸고, lock을 False로 바꿔준다.

```
else:
    for i in range(self.windowSize):
        ack_packet = i + recent_ack + 1
        if ack_packet >= len(self.packet_queue):
            break

        packet = make_packet(ack_packet, self.packet_queue[ack_packet])
        self.timer.save_sendtime(ack_packet)
        self.sender_socket.sendto(packet, self.recvAddr)
        if self.retransmit_lock:
            writePkt(self.logFile, self.startTime, ack_packet, "retransmitted")
            self.retransmit_lock = False
        else:
            writePkt(self.logFile, self.startTime, ack_packet, "sent")
    if self.recent_ack == len(self.packet_queue) - 1:
        self.sender_socket.sendto(b'W0', self.recvAddr)
        break

    self.lock = True
```

windowSize만큼 패킷을 전송해야 하므로 for문을 돌려준다.

그리고 패킷을 만들고, 전송함과 함께 timer.sendtime에 시간을 저장한다.

만약 retransmit_lock이 True라면 retransmit를 로그에 출력하고 값을 False로 바꾼다. False라면 그냥 sent한다.

for문 도중에 packet 번호가 len(packet_queue)보다 크게 되면 for문을 탈출한다.

for문을 탈출하고 self.recent_ack 이 len(self.packt_queue) – 1 과 같게 된다면 ack로 마지막 패킷 번호를 받은 것이기 때문에 전송이 완료된 것이다. 그러므로 끝내는 packet을 전송하고 while문을 탈출한다.

C. recv_from: ACK를 받는 메인 함수

recv_ack에 receiver로부터 받은 ACK을 저장한다. 만약 직전에 받은 recent_ack과 다르다면, 정상적으로 패킷이 전송되고 ACK을 받은 것이므로, duplicate_ack은 0이 된다.

만약 같다면 duplicate_ack를 1씩 더하고, 반복해서 받다가 duplicate_ack가 3이 되면 3duplicated를 로그에 출력한다. 그리고 retransmit 해주어야하므로 recent_ack를 recv_ack로 바꿔준다. 그리고 duplicate_lock을 True로 바꾸고 duplicated를 띄우거나 retransmit를 여러 번 해주지 않도록 한다. 또한 retransmit를 해주어야하므로 lock을 False로, retransmit_lock을 True로 바꾼다.

만약 recent_ack보다 recv_ack이 더 크다면, 다음 패킷을 정상적으로 받았다는 뜻이므로, duplicate_lock을 False로 바꾸고 recent_ack를 recv_ack으로 바꿔준다. 그리고 RTT를 갱신해준다.

만약 recent_ack가 이 len(self.packt_queue) – 1 과 같게 된다면 ack로 마지막 패킷 번호를 받은 것이기 때문에 전송이 완료된 것이다. 그러므로 while문을 탈출한다.

D. run_send

send_to와 recv_from을 각각의 쓰레드에서 작동시키고, 종료되면 로그파일에 종료됨을 알림과 동시에 goodput과 avgRTT를 출력한다.

<<receiver.py>>

0. main

소켓을 연결해주고, 파일이름과 주소를 받으면 -1 ACK를 전송한다. 그리고 각 인자를 filieReceiver함수에 연결해준다.

1. fileReceiver: 파일을 받고 ACK를 전송하는 함수

recv_handler class를 호출한다.

class 내부에 있는 run_recv 함수를 실행해서 파일을 받고 ACK를 전송한다.

2. recv_handler: receiver가 ACK을 보내고 패킷을 받기 위한 class

A. class 내부 변수

receiver_socket: 연결되어 있는 소켓

dstFilename: 파일을 전송받아서 저장할 파일이름

recvAddr: 연결된 소켓의 주소 tuple

packet_queue: 패킷을 받아서 저장해두는 queue

logFile: logfile

startTime: 함수를 실행하는 시작시간

B. file_receive: 패킷을 받고 ACK을 전송하는 메인함수

sender에서 패킷을 받고 buffer에 저장한다. 만약 buffer에서 b'₩0', 끝내기 위한 패킷을 받았을 때, while문을 탈출한다.

아니라면 헤더에서 pktnum를 얻고, 나머지 data를 얻는다. 로그에 받은 패킷의 번호를 출력한다.

만약 패킷 번호가 len(packet_queue)와 같다면 손실 없이 정상적으로 패킷을 전송받은 것이므로, packet_queue에 데이터를 저장한다. 그리고 패킷을 받았다고 sender에게 ACK을 전송한다.

다르다면 패킷 손실이 일어난것이므로, packet_queue에 데이터를 담지 않고 받은 패킷에 대해 ACK을 전송한다.

C. write_file

큐에 저장되어 있는 데이터를 dstFilename에다가 쓴다.

D. run_recv

file_recieve와 write_file을 차례로 실행시키고, 끝나면 로그파일에 goodput을 출력한다.

3. Experiment

실험 결과

실험 1. window size를 40으로 고정하고 loss_probabilities를 바꿔가며 실험한 결과이다.

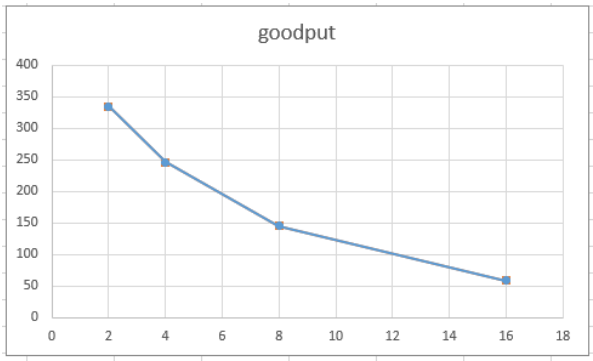
windowSize: 40

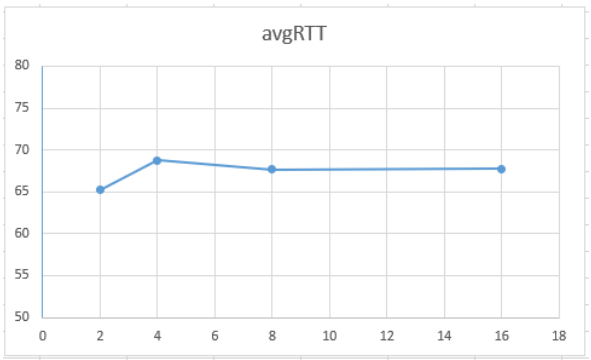
BW: 10Mbps

One Wat Delay: 25ms

Loss_probabilities: 2, 4, 8, 16 (%)

loss_prob	w_size 40	goodput	1	2	3	4	5	6	7	8	9	10	avg
2			324.536	343.337	329.474	340.306	333.322	336.157	336.633	327.346	344.726	333.033	334.887
4			254.312	239.071	238.839	243.669	243.11	248.453	252.329	252.098	254.978	237.355	246.4214
8			146.367	146.019	148.621	144.299	140.84	145.525	136.959	146.972	155.73	144.346	145.5678
16			61.109	55.363	59.342	57.608	55.98	60.421	56.379	61.218	60.979	56.215	58.4614
loss_prob	w_size 40	avgRTT											avg
2			64.728	64.785	64.528	75.941	62.986	60.063	60.705	65.062	70.484	63.2226	65.25046
4			66.686	66.985	70.814	75.652	62.032	60.08	67.514	72.77	67.313	77.468	68.7314
8			62.729	70.868	68.815	65.571	74.215	65.314	62.566	67.141	70.892	68.349	67.646
16			67.406	65.619	71.746	63.289	63.228	60.421	72.713	66.137	70.237	76.679	67.7475





x축은 loss_probabilities이고, y축은 각각 goodput, avgRTT이다.

retrasmit되는 패킷은 RTT 갱신을 시키지 않으므로 avgRTT는 변화가 거의 없다.

loss_probabilities가 클수록 retransmit경우가 많아져 시간이 오래걸리므로, goodput이 낮게나온다.

실험2. loss_probabilities를 2로 고정하고 window_size를 바꿔가며 실험한 결과이다.

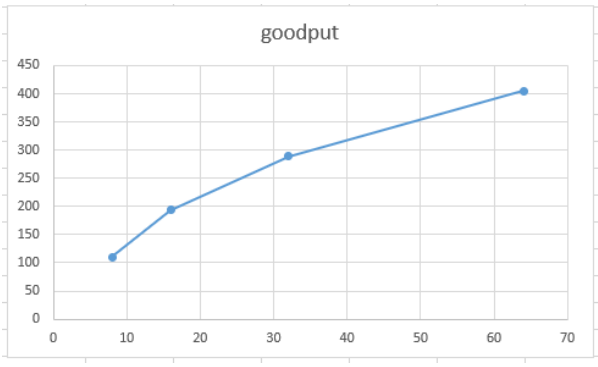
windowSize: 8, 16, 32, 64

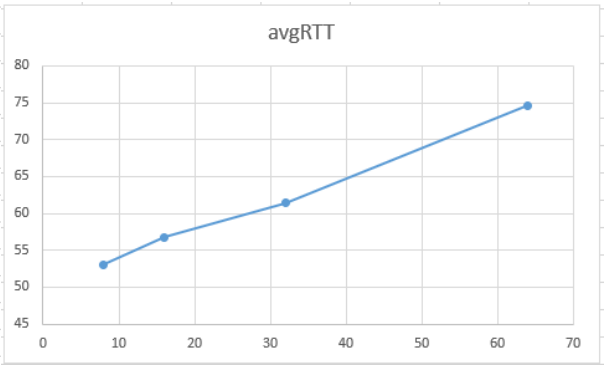
BW: 10Mbps

One Wat Delay: 25ms

Loss_probabilities: 2%

loss_prob	w_size	goodput	1	2	3	4	5	6	7	8	9	10	avg
2	8		109.76	107.487	106.617	110.767	105.557	111.995	112.901	109.884	112.989	109.468	109.7425
	16		192.185	197.942	193.992	191.878	199.288	192.243	187.297	196.226	187.236	195.058	193.3345
	32		290.274	301.806	283.666	281.307	291.651	283.666	287.815	295.588	285.046	294.177	289.4996
	64		380.144	405.703	411.404	405.856	412.053	397.941	421.952	395.418	404.241	410.072	404.4784
loss_prob	w_size	avgRTT											avg
2	8		53.179	53.458	52.982	53.232	52.861	52.803	52.697	53.016	53.837	52.867	53.0932
	16		55.592	56.558	55.707	56.151	55.68	57.887	58.222	56.135	56.985	58.605	56.7522
	32		64.028	66.705	58.866	58.761	59.314	58.866	65.083	60.694	61.654	60.474	61.4445
	64		63.154	72.588	65.561	79.645	83.053	92.589	76.939	79.823	67.414	65.291	74.6057





x축은 window_size이고, y축은 각각 goodput, avgRTT이다.

패킷을 보내고 받는데까지 window_size가 클수록 오래 걸리므로 avgRTT가 커진다.