README

1. How to Run the Code?

1. Run Basic:

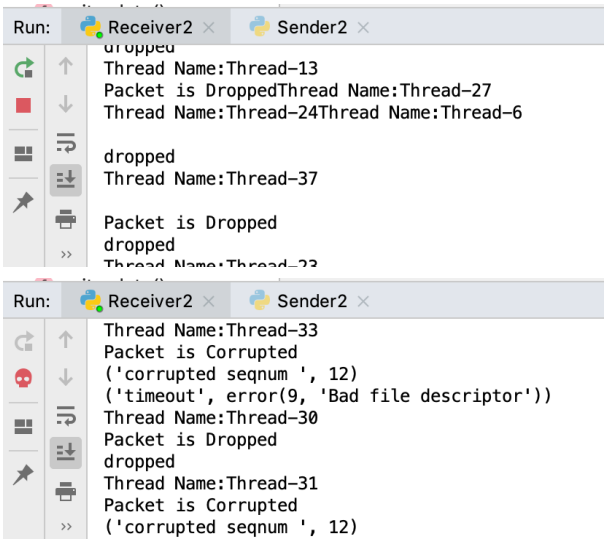
    - Note: the timeout between a. and b. is 5 secs.
    - For text:
        - a. Run Receiver.py, using "python Receiver.py ./text/rece_file.txt localhost 6789"
        - b. Run Sender.py, using "python Sender.py ./text/send_file.txt localhost 6789"
    - For image:
        - a. Run Receiver.py, using "python Receiver.py ./imgs/rece_img.jpeg localhost 6789"
        - b. Run Sender.py, using "python Sender.py ./imgs/imgPA2.jpeg localhost 6789"

2. Run Bonus Part:

    - a. **Directly** Run Receiver2.py, type in the filename you want to send, eg: "./text/rece_file.txt", or "./imgs/rece_img.jpeg".

    - b. Run Sender2.py in **Terminal**, for **text file**, using "python Sender2.py ./text/send_file.txt localhost 6789", or for **image**, using "python Sender2.py ./imgs/imgPA2.jpeg localhost 6789".

    - c. After Sender.py sends all the files, **Terminate** Receiver2.py manually, otherwise will wait a long time. Press the **Red** Terminate Button twice as follows:



    - d. Check the output file ~

2. Details of Bonus Part

Check the Second Part of "Overview of Code"

Overview of Code

1. Basic Sender.py

1.1 Split File into Chunks, then Make Packets.

```
def make_pkt_from_data(file_name):
    f = read_file(file_name, chunk_size=DATA_LENGTH)  # DATA_LENGTH)
    ...
    return packet_with_csum
```

1.2 Using Sliding Window

1.2.1 Send Packets in Current Window

```
def transmit_wnd(clientSocket, ip, port, pkts, send_start, send_end):
    try:
        i = send_start
        while i <= send_end:  # window
            clientSocket.sendto(pkts[i], (ip, port))
            checked_sum, received_sum, seqnum, flag, data =
extract_packet(pkts[i])
            print ("transmit_wnd: send ", i, "-th pkt, ", "seqnum of pkt
is ", seqnum)
            i += 1
    except timeout:
        print('timeout, closing socket')
        clientSocket.close()
```

1.2.2 Update Window and Packets to be Sended

```
# moving sliding window
# Once received an ack, the window will slide one or more steps, and send
one or more packets.
send_start = max(wnd_start, send_end + 1)
if ack_seq < n: # n is the total number of packets.
    wnd_start = ack_seq
    if ack_seq + WND_SIZE < n:
        send_end = wnd_start + WND_SIZE
    else:
        send_end = n - 1
```

1.3 500ms Timeout

```
@timeout(0.5)
def wait_ack(clientSocket):
    while True:
        try:
            ack, serverAddress = clientSocket.recvfrom(2048)
```

```
            if ack:
                checked_sum, received_sum, ack_seq, flag, data =
extract_packet(ack)
                print ["wait_ack received ack, ack_seq is", ack_seq]
                return ack

        except timeout:
            print('timeout, closing socket')
            clientSocket.close()
```

1.4 Fast Retransmission

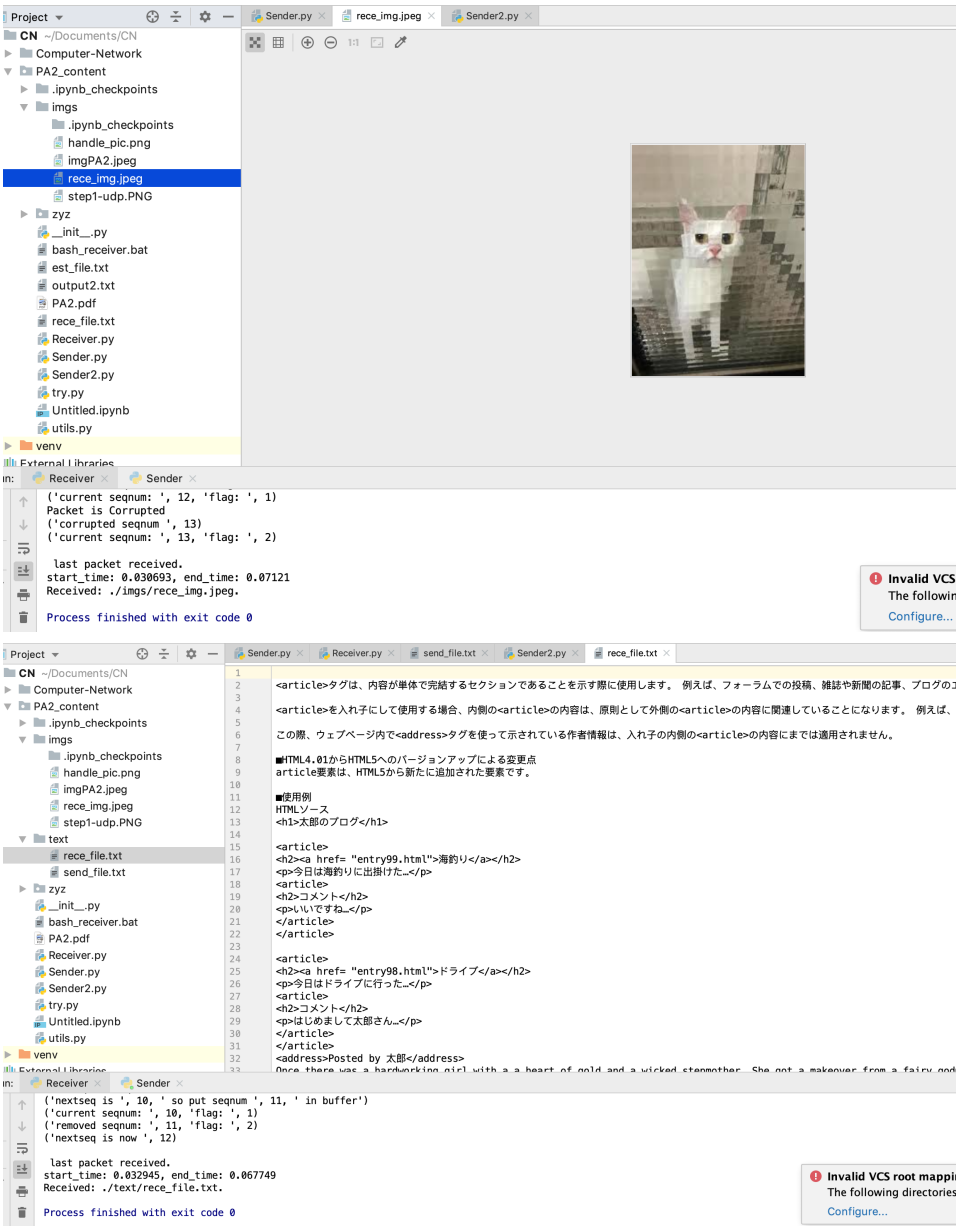
```
# Using an Dictionary to record
if ack_seq not in ack_dic:
    ack_dic[ack_seq] = 1
else:
    ack_dic[ack_seq] += 1
if ack_dic[ack_seq] >= 3:
    wnd_start = ack_seq
    raise ValueError("Duplicate Ack, Will Do Fast Retransmit")
```

1.4 Only Retransmit the Oldest Unacknowledged Packet

```
def retransmit_oldest(clientSocket, pkts, ip, port, i):
    ack = None
    while not ack:
        try:
            clientSocket.sendto(pkts[i], (ip, port))
            ack = wait_ack(clientSocket)  # must return an ack
            checked_sum, received_sum, ack_seq, flag, data =
extract_packet(ack)
            checked_sum_pkt, received_sum_pkt, pkt_seq, flag_pkt, data_pkt
= extract_packet(pkts[i])
            print "retransmit pkt:", i, " pkt_seq is:", pkt_seq, " receive
ack seq is: ", ack_seq
            if checked_sum != received_sum:
                raise ValueError("corrupt ack")
            print "retransmit succeed"
            return ack

        except Exception as e:  # RuntimeError:
            print "retransmit error:", e
```

1.5 Can Handle Arbitrary Files.



1.6 Avoid Crashing in the First Place

```
ack = None
while not ack:
    try:
        transmit_wnd(clientSocket, ip, port, [self.pkts[0]], 0, 0) # the
first pkt
        ack = wait_ack(clientSocket)
    except Exception as e:
        print "the first packet's error: ", e
```

2. Adding Bonus Part

2.1 Congestion Control (10 points)

```
THRES_HOLD = 10
SLOW_START_WND_SIZE = 1
if if_retransmit:
    ack = retransmit_ack
    # adjust window
    if wnd_size_var >= THRES_HOLD and wnd_size_var > 0:
        wnd_size_var /= 2
    if wnd_size_var < THRES_HOLD and wnd_size_var > 0:
        wnd_size_var = SLOW_START_WND_SIZE
    if_retransmit = False
if ack:
    if wnd_size_var + 1 < THRES_HOLD:
        wnd_size_var += 1
    else: #CA
        wnd_size_var += int(1/wnd_size_var)
```

2.2 Adjust Timeout according to Variable RTT (5 points)

```
class RttUtils:
    def __init__(self, sample_rtt):
        self.sample_rtt = sample_rtt
        self.dev_rtt = 0
        self.timeout_interval = 0

    def update(self, estimated_rtt):
        estimated_rtt = 0.875 * estimated_rtt + 0.125 * self.sample_rtt
        self.dev_rtt = 0.75 * self.dev_rtt + 0.25 * abs(self.sample_rtt -
estimated_rtt)
        self.timeout_interval = estimated_rtt + 4 * self.dev_rtt
        return self.timeout_interval, estimated_rtt

# Usage
self.transmit_wnd(clientSocket, ip, port, [self.pkts[0]], 0, 0)
self.rtt[0] = time.time()
ack = self.wait_ack(clientSocket)
sample_rtt = time.time() - self.rtt[0]  # rtt[0] is the send time for pkt
0

rtt_utils = RttUtils(sample_rtt)
self.timeout_interval, self.estimated_rtt =
rtt_utils.update(self.estimated_rtt)
timeout_interval = self.timeout_interval
```

2.3 Add Threads to Implement Bidirectional Transfer (5 points)

> After adding threads, the transmission is as much faster for sender, in real time, it takes about 3 secs, in CPU time, it takes about 0.002 secs.

2.3.1 Server Side (Receiver2.py)

```
class MyUDPRequestHandler(SocketServer.DatagramRequestHandler):
    # Override the handle() method
    def handle(self):
        """
        1. This part is the main process part for Server.
        2. need to change some variable into global, otherwise in one
thread it updates, in another is not.
        3. can't use yield and rerurn the yield data, can write data
everytime or use a String variable to record
        4. I use UDPServerObject.shutdown(), but it still need serveral
minutes to shutdown.
        """
UDPServerObject = SocketServer.ThreadingUDPServer(ServerAddress,
MyUDPRequestHandler)
UDPServerObject.serve_forever(poll_interval=0.5)
```

2.3.2 Client Side (Sender2.py)

```
ThreadList = []
ThreadCount = 20
# Create as many connections as defined by ThreadCount
for index in range(ThreadCount):
    ThreadInstance = threading.Thread(target=sender.main())
    ThreadList.append(ThreadInstance)
    ThreadInstance.start()
# Main thread to wait till all connection threads are complete
for index in range(ThreadCount):
    ThreadList[index].join()
```

Console Output

> In a normal execution, the console output will be like: For the submit version of code, I delete all the output, except Exception.

```
# Server side:
Thread Name:Thread-1
Thread Name:Thread-4
Thread Name:Thread-3
Packet is Corrupted
('corrupted seqnum ', 2)
Thread Name:Thread-7
('current seqnum: ', 2, 'flag: ', 1)
Thread Name:Thread-5
('current seqnum: ', 3, 'flag: ', 1)
Thread Name:Thread-6
Packet is Corrupted
('corrupted seqnum ', 4)
```

```
Thread Name:Thread-1
...

# Client side:
('transmit_wnd: send ', 0, '-th pkt, ', 'seqnum of pkt is ', 0)
('pkts csum is', 2630644351, 'pkt received_sum is: ', 2630644351)
['wait_ack received ack, ack_seq is', 1]
sample_rtt: 0.120254039764
estimated_rtt: 0.120254039764
timeout_interval: 0.120254039764

current window size: 1
 to be sended pkt in current window is from pkt 1  to 1
('transmit_wnd: send ', 1, '-th pkt, ', 'seqnum of pkt is ', 1)
('pkts csum is', 266770715, 'pkt received_sum is: ', 266770715)
reliable transmit error: function [wait_ack] timeout [0.120254039764
seconds] exceeded!
retransmit error: function [wait_ack] timeout [0.120254039764 seconds]
exceeded!
current window size: 1
['wait_ack received ack, ack_seq is', 1]
['wait_ack received ack, ack_seq is', 1]
reliable transmit error: function [wait_ack] timeout [0.120254039764
seconds] exceeded!
retransmit error: function [wait_ack] timeout [0.120254039764 seconds]
exceeded!
current window size: 1
['wait_ack received ack, ack_seq is', 1]
received ack is:  l ack_seq is:  1
sample_rtt: 0.134088993073
estimated_rtt: 0.121983408928
modified timeout_interval: 0.134088993073

current window size: 2
 to be sended pkt in current window is from pkt 2  to 3
('transmit_wnd: send ', 2, '-th pkt, ', 'seqnum of pkt is ', 2)
('pkts csum is', 3007635358, 'pkt received_sum is: ', 3007635358)
('transmit_wnd: send ', 3, '-th pkt, ', 'seqnum of pkt is ', 3)
('pkts csum is', 1517250782, 'pkt received_sum is: ', 1517250782)
['wait_ack received ack, ack_seq is', 1]
received ack is:  l ack_seq is:  1
sample_rtt: 0.207488059998
estimated_rtt: 0.132671490312
modified timeout_interval: 0.207488059998

...

Process finished with exit code 0
```