Project Design: Based on Transport Layer

Introduction

In lab 2 of CS3611, you have implemented **RDT 3.0** based on **UDP** and simulated its transmitting procedure via Python programs. Designed on basis of Lab 2, this project calls for further comprehension of pipelining in transmission, where you are expected to work on **Go-Back-N (GBN)** protocol and **Selective Repeat (SR)** protocol with similar methodology (the **lab part**), as well as study into any possible application or implication in regard (the **research part**).

Part 1 - Lab

1.1 GBN protocol

Provided codes

> **GBN_sender.py**
> **GBN_receiver.py**
> **utils.py**

Instructions

- You need to implement GBN transmission via Python programs.
- Similar to Lab 2, you are required to complete the given skeleton codes so that an input string of lowercase letters will be transmitted from the sender to the receiver and converted capital in manner of **GBN** protocol.
- You need to hand in the complete code files along with screenshots of your result of simulation (refer to **sample simulation**), and explain how they illustrate the processing mechanism especially in case of packet corruption and losses, verifying that you have actually implemented the protocol with accuracy.

Notes

- Corruption and loss of ACKs are dismissed for convenience.
- You should not modify `utils.py` except setting the simulator.
- You may refer to the FSM and illustrations in our textbook [Kurose, Ross] (fig. 3-19, 3-21, page 145~148).
- (Advisably) In the given codes, `### fill in here ###` marks the ***exact*** and ***only*** places to be substituted with your own codes.

Sample simulation for GBN

> Sender input: `shanghai`
> Receiver output: `SHANGHAI`
> Probability of corruption: `0.1`
> Probability of loss: `0.1`

```
# Sender side output:

Input sentence:
shanghai
Send:  {"pktID": 1, "data": "s", "checksum": True}
Waiting ACK1...
Send:  {"pktID": 2, "data": "h", "checksum": True}
Receive:  {'pktID': 1, 'data': 'ACK', 'checksum': True}
Send:  {"pktID": 3, "data": "a", "checksum": True}
Waiting ACK2...
Send:  {"pktID": 4, "data": "n", "checksum": True}
Receive:  {'pktID': 1, 'data': 'ACK', 'checksum': True}
Send:  {"pktID": 5, "data": "g", "checksum": True}
Receive:  {'pktID': 1, 'data': 'ACK', 'checksum': True}
Receive:  {'pktID': 1, 'data': 'ACK', 'checksum': True}
Waiting ACK2...
Waiting ACK2...
Timeout. Resend the packages
Resend:  {"pktID": 2, "data": "h", "checksum": True}
Resend:  {"pktID": 3, "data": "a", "checksum": True}
Resend:  {"pktID": 4, "data": "n", "checksum": True}
Resend:  {"pktID": 5, "data": "g", "checksum": True}
Receive:  {'pktID': 2, 'data': 'ACK', 'checksum': True}
Send:  {"pktID": 6, "data": "h", "checksum": True}
Receive:  {'pktID': 3, 'data': 'ACK', 'checksum': True}
Send:  {"pktID": 7, "data": "a", "checksum": True}
Receive:  {'pktID': 4, 'data': 'ACK', 'checksum': True}
Send:  {"pktID": 8, "data": "i", "checksum": True}
Receive:  {'pktID': 5, 'data': 'ACK', 'checksum': True}
Send:  {"pktID": 9, "data": "\n", "checksum": True}
Receive:  {'pktID': 6, 'data': 'ACK', 'checksum': True}
Receive:  {'pktID': 6, 'data': 'ACK', 'checksum': True}
Receive:  {'pktID': 6, 'data': 'ACK', 'checksum': True}
Receive:  {'pktID': 6, 'data': 'ACK', 'checksum': True}
Waiting ACK7...
Waiting ACK7...
Timeout. Resend the packages
Resend:  {"pktID": 7, "data": "a", "checksum": True}
Resend:  {"pktID": 8, "data": "i", "checksum": True}
Resend:  {"pktID": 9, "data": "\n", "checksum": True}
Receive:  {'pktID': 7, 'data': 'ACK', 'checksum': True}
Receive:  {'pktID': 8, 'data': 'ACK', 'checksum': True}
Receive:  {'pktID': 9, 'data': 'ACK', 'checksum': True}
Transmission completed!
```

```
# Receiver side output:

{'pktID': 1, 'data': 's', 'checksum': True}  delivered.
Package lost.
{'pktID': 3, 'data': 'a', 'checksum': False}  dropped.
{'pktID': 4, 'data': 'n', 'checksum': True}  dropped.
{'pktID': 5, 'data': 'g', 'checksum': True}  dropped.
{'pktID': 2, 'data': 'h', 'checksum': True}  delivered.
{'pktID': 3, 'data': 'a', 'checksum': True}  delivered.
{'pktID': 4, 'data': 'n', 'checksum': True}  delivered.
{'pktID': 5, 'data': 'g', 'checksum': True}  delivered.
```

```
{'pktID': 6, 'data': 'h', 'checksum': True}  delivered.
{'pktID': 7, 'data': 'a', 'checksum': False}  dropped.
{'pktID': 8, 'data': 'i', 'checksum': True}  dropped.
{'pktID': 9, 'data': '\n', 'checksum': True}  dropped.
{'pktID': 7, 'data': 'a', 'checksum': True}  delivered.
{'pktID': 8, 'data': 'i', 'checksum': True}  delivered.
{'pktID': 9, 'data': '\n', 'checksum': True}  delivered.
The received messages through GBN:
SHANGHAI
```

1.2 SR protocol

Provided codes

> SR_sender.py
> SR_receiver.py
> utils.py

Instructions

- You need to implement SR transmission via Python programs.
- Similar to Lab 2, you are required to complete the given codes so that an input string of lowercase letters will be transmitted from the sender to the receiver and converted capital in manner of **SR** protocol.
- Pay close attention to the differences between SR and GBN implementation.
- You need to hand in the complete code files along with screenshots of your result of simulation (refer to **sample simulation**), and explain how they illustrate the processing mechanism especially in case of packet corruption and losses, verifying that you have actually implemented the protocol with accuracy.

Notes

- Corruption and loss of ACKs are dismissed for convenience.
- You should not modify `utils.py` except setting the simulator.
- You may refer to the FSM and illustrations in our textbook [Kurose, Ross] (fig. 3-23, 3-24 and 3-25, page 148~151).
- (Advisably) In the given codes, `### fill in here ###` marks the ***exact*** and ***only*** places to be substituted with your own codes.

Sample simulation for SR

> Sender input: `shanghai`
> Receiver output: `SHANGHAI`
> Probability of corruption: `0.1`
> Probability of loss: `0.1`

```
# Sender side output:

Input sentence:
shanghai
Send:  {"pktID": 1, "data": "s", "checksum": True}
Waiting ACK1...
```

```
Send:  {"pktID": 2, "data": "h", "checksum": True}
Waiting ACK1...
Send:  {"pktID": 3, "data": "a", "checksum": True}
packet 1 timeout.
Resend:  {"pktID": 1, "data": "s", "checksum": True}
Receive:  {'pktID': 2, 'data': 'ACK', 'checksum': True}
Send:  {"pktID": 4, "data": "n", "checksum": True}
Receive:  {'pktID': 3, 'data': 'ACK', 'checksum': True}
Receive:  {'pktID': 1, 'data': 'ACK', 'checksum': True}
Send:  {"pktID": 5, "data": "g", "checksum": True}
Receive:  {'pktID': 4, 'data': 'ACK', 'checksum': True}
Send:  {"pktID": 6, "data": "h", "checksum": True}
Receive:  {'pktID': 5, 'data': 'ACK', 'checksum': True}
Send:  {"pktID": 7, "data": "a", "checksum": True}
Receive:  {'pktID': 6, 'data': 'ACK', 'checksum': True}
Send:  {"pktID": 8, "data": "i", "checksum": True}
Receive:  {'pktID': 7, 'data': 'ACK', 'checksum': True}
Send:  {"pktID": 9, "data": "\n", "checksum": True}
Receive:  {'pktID': 8, 'data': 'ACK', 'checksum': True}
Receive:  {'pktID': 9, 'data': 'ACK', 'checksum': True}
Transmission completed!
```

```
# Receiver side output:

Package lost.
{'pktID': 2, 'data': 'h', 'checksum': True}
 buffered.
{'pktID': 3, 'data': 'a', 'checksum': True}
 buffered.
{'pktID': 1, 'data': 's', 'checksum': True}
 buffered.
packet 1  delivered.
packet 2  delivered.
packet 3  delivered.
{'pktID': 4, 'data': 'n', 'checksum': True}
 buffered.
packet 4  delivered.
{'pktID': 5, 'data': 'g', 'checksum': True}
 buffered.
packet 5  delivered.
{'pktID': 6, 'data': 'h', 'checksum': True}
 buffered.
packet 6  delivered.
{'pktID': 7, 'data': 'a', 'checksum': True}
 buffered.
packet 7  delivered.
{'pktID': 8, 'data': 'i', 'checksum': True}
 buffered.
packet 8  delivered.
{'pktID': 9, 'data': '\n', 'checksum': True}
 buffered.
packet 9  delivered.
The received messages through GBN:
SHANGHAI
```

Part 2 - Research

Optional directions

1. Recall what you have implemented in course lab 2, try to demonstrate differences and improvement (in efficiency, reliability, etc.) from **stop-and-wait** protocol to **gliding window** protocol and from **GBN** protocol to **SR** protocol. A practical way is to base experiments on the existing programs, i.e., you may try different values for the parameterized factors of each protocol (sequence number space, sequence length, window size, timeout, loss prob., corruption prob., etc.), record the statistical data at your disposal (transmission time, number of packets sent, etc.) and interpret with these data how the parameters affect the performance of different transmission protocols. A report containing comparison and analysis **based on simulation** may be appropriate to present.
2. Try to improve your simulation. For instance, consider two-way transmission via GBN, support handling cases when packets and ACKs get lost/corrupted simultaneously, etc.
3. Try similar methodology to implement **TCP congestion control algorithm** referring to its FSM description. Consider its relationship with the protocols practiced in former parts. Then, try to capture TCP flows using **Wireshark**, plot figures showing window congestion, calculate instantaneous/average throughput and compare the results with that in ideal/simulative condition.
4. Anything innovative related to computer networking is encouraged.

Appendix - Skeleton codes for Part 1

`GBN_sender.py`

```
from socket import *
import json
import time
from utils import Timer

def notcorrupt(rcvpkt):
    return rcvpkt['checksum']

def getacknum(rcvpkt):
    return rcvpkt['pktID']

def make_pkt(pktID, data, checksum):
    sendData = {}
    sendData['pktID'] = pktID
    sendData['data'] = data
    sendData['checksum'] = checksum
    sendMessage = json.dumps(sendData)
    return sendMessage

serverName = 'localhost'
serverPort = ### fill in here ###

clientSocket = ### fill in here ###
clientSocket.setblocking(False)
```

```
timer = Timer(2) # timeout = 2s, changeable
N = 4 # window size, changeable

messages = input("Input sentence: \n")
messages = messages + '\n'
base = ### fill in here ###
nextseqnum = ### fill in here ###

while True:
    if nextseqnum <= len(messages) and nextseqnum < base + N:
        sndpkt = make_pkt(nextseqnum, messages[nextseqnum - 1], True)
        clientSocket.sendto(### fill in here ###)
        print("Send: ", sndpkt)
        if base == nextseqnum:
            timer.start_timer()
        nextseqnum += 1

    if timer.if_timeout():
        print("Timeout. Resend the packages")
        timer.start_timer()
        for i in range(### fill in here ###):
            sndpkt = make_pkt(### fill in here ###)
            clientSocket.sendto(### fill in here ###)
            print("Resend: ", sndpkt)
        continue

    try:
        rcvpkt = clientSocket.recv(1024)
        rcvpkt = json.loads(rcvpkt.decode())
        print("Receive: ", rcvpkt)
        if rcvpkt['pktID'] == len(messages):
            print("Transmission completed!")
            break
        if notcorrupt(rcvpkt):
            base = getacknum(rcvpkt) + 1
            if ### fill in here ###:
                timer.stop_timer()
            else:
                timer.start_timer()
        continue

    except BlockingIOError:
        print(f"Waiting ACK{base}...")
        time.sleep(1)
        continue
```

GBN_receiver.py

```
import json
import random
from socket import *
from utils import ACK, simulator
```

```
def notcorrupt(rcvpkt):
    return rcvpkt['checksum']

def hasseqnum(rcvpkt, expectedseqnum):
    return rcvpkt['pktID'] == expectedseqnum

def make_pkt(pktID, data, checksum):
    sendData = {}
    sendData['pktID'] = pktID
    sendData['data'] = data
    sendData['checksum'] = checksum
    sendMessage = json.dumps(sendData)
    return sendMessage

serverSocket = ### fill in here ###
serverPort = ### fill in here ###
serverSocket.bind(### fill in here ###)

strBuffer = ''
expectedseqnum = 1
default_sndpkt = make_pkt(0, ACK, True)

while True:
    if len(strBuffer) != 0 and strBuffer[-1] == '\n':
        print("The received messages through GBN:\n{}\n".format(strBuffer))
        strBuffer = ''
        break

    rcvpkt, address = serverSocket.recvfrom(1024)
    rcvpkt = json.loads(rcvpkt.decode())
    rcvpkt = simulator(rcvpkt)
    if rcvpkt is None:
        print("Package lost.")
        continue

    if notcorrupt(rcvpkt) and hasseqnum(rcvpkt, expectedseqnum):
        message = rcvpkt['data'].upper()
        strBuffer = strBuffer + message
        print(rcvpkt, " delivered.")
        """
        Send ACK, and...
        """
        ### Fill in here ###

    else:
        print(rcvpkt, " dropped.")
        """
        Send ACK
        """
        ### Fill in here ###
    continue
```

SR_sender.py

```
from socket import *
import json
import time
from utils import Timer

def notcorrupt(rcvpkt):
    return rcvpkt['checksum']

def getacknum(rcvpkt):
    return rcvpkt['pktID']

def make_pkt(pktID, data, checksum):
    sendData = {}
    sendData['pktID'] = pktID
    sendData['data'] = data
    sendData['checksum'] = checksum
    sendMessage = json.dumps(sendData)
    return sendMessage

serverName = 'localhost'
serverPort = ### fill in here ###

clientSocket = socket(### fill in here ###)
clientSocket.setblocking(False)
N = 4 # window size, changeable

messages = input("Input sentence: \n")
messages = messages + '\n'

timers = [Timer(2) for _ in range(len(messages) + 1)] # Each packet has its own timer
acks = [False for _ in range(len(messages) + 1)] # List to maintain ACK state for each
packet
send_base = ### fill in here ###
nextseqnum = ### fill in here ###

while True:
    if nextseqnum <= len(messages) and nextseqnum < send_base + N:

        '''
        send packet
        '''
        ### fill in here ###

        print("Send: ", sndpkt)

    for i in range(### fill in here ###):
        if timers[i].active and timers[i].if_timeout():
            print(f"packet {i} timeout.")

            '''
            Resend packet
            '''
            ### fill in here ###
```

```
            print("Resend: ", sndpkt)

    try:
        rcvpkt = clientSocket.recv(1024)
        rcvpkt = json.loads(rcvpkt.decode())
        print("Receive: ", rcvpkt)
        if notcorrupt(rcvpkt):
            ackseqnum = getacknum(rcvpkt)

            ### fill in here ###

            if send_base == ackseqnum:
                '''
                move window forward
                '''
                i = send_base + 1
                while ### fill in here ### :

                    ### fill in here ###

                send_base = i
                if i == len(messages) + 1:
                    print("Transmission completed!")
                    break
        continue

    except BlockingIOError:
        print(f"Waiting ACK{send_base}...")
        time.sleep(1)
        continue
```

SR_receiver.py

```
import json
import random
from socket import *
from utils import ACK, simulator

def notcorrupt(rcvpkt):
    return rcvpkt['checksum']

def getseqnum(rcvpkt):
    return rcvpkt['pktID']

def make_pkt(pktID, data, checksum):
    sendData = {}
    sendData['pktID'] = pktID
    sendData['data'] = data
    sendData['checksum'] = checksum
    sendMessage = json.dumps(sendData)
    return sendMessage
```

```
serverSocket = socket(### fill in here ###)
serverPort = ### fill in here ###
serverSocket.bind(('',serverPort))

strResult = ''
buffer = dict()
N = 4 # window size, changeable
rcv_base = ### fill in here ###

while True:
    if len(strResult) != 0 and strResult[-1] == '\n':
        print("The received messages through GBN:\n{}\n".format(strResult))
        strBuffer = ''
        break

    rcvpkt, address = serverSocket.recvfrom(1024)
    rcvpkt = json.loads(rcvpkt.decode())
    rcvpkt = simulator(rcvpkt)
    if rcvpkt is None:
        print("Package lost.")
        continue
    print(rcvpkt)

    if notcorrupt(rcvpkt):
        pktseqnum = getseqnum(rcvpkt)
        if pktseqnum >= rcv_base - N and pktseqnum < rcv_base:

            '''
            send ACK
            '''
            ### fill in here ###

        elif pktseqnum >= rcv_base and pktseqnum < rcv_base + N:

            '''
            send ACK
            '''
            ### fill in here ###

            buffer[pktseqnum] = rcvpkt['data']
            print(" buffered.")
            if rcv_base == pktseqnum:
                '''
                Convert the letter and move window forward
                '''
                i = pktseqnum
                while ### fill in here ### :

                    ### fill in here ###

                    print('packet', i, " delivered.")
                rcv_base = i
```

utils.py

```python
import random
import time

ACK = 'ACK'

def simulator(rcvpkt):
    p = random.random()
    if p < 0.1:
        rcvpkt['checksum'] = False
    elif p > 0.9:
        rcvpkt = None
    return rcvpkt

class Timer:
    def __init__(self, target):
        self.start_time = 0
        self.target = target
        self.active = False

    def start_timer(self):
        self.start_time = time.time()
        self.active = True

    def stop_timer(self):
        self.start_time = 0
        self.active = False

    def if_timeout(self):
        if self.active:
            return (time.time() - self.start_time) > self.target
        else:
            print("Please start the timer at first")
            return False
```