Project - Principles of Reliable Data Transfer

Due Date: November 30, 2023

1 Project Overview

This project aims to improve your theoretical understanding of reliable data transfer protocols by implementing a simple one.

1.1 RDTv2.2 Requirements

You will implement the reliable data transfer protocol RDTv2.2 (already covered in the class). The key characteristics of RDTv2.2 are:

- It is a stop-and-wait protocol.
- It uses positive acknowledgment with retransmission.
- It is an alternating bit protocol that uses 1 bit (0 or 1) for packet sequence number
- It tolerates only packet corruption.
- It fails to handle packet loss and packet-out-of-order.

1.2 Submission Instructions

1. You will complete this project as a team of two students (at most 3 students per team).
2. You must use the provided skeleton-code files to complete this project (see the code appendix section)
3. You must submit the following files:
    - **main.py** the main script to start and test the RDT2.2 protocol.
    - **sender.py** implement the Reliable Data Transfer Protocol V2.2 sender side.
    - **receiver.py** implement the Reliable Data Transfer Protocol V2.2 receiver side
    - **network.py** implement the network layer that delivers packets and acknowledgments between sender and receiver
    - **Project Report** a pdf file describes your implementation of the RDTv2.2. The report MUST include the pseudo-code of the RDT sender and receiver sides. The finite state machine diagrams for the sender and receiver. Description of any changes you have made to the skeleton code files. Test cases and screenshots for the execution of the test cases. The report must contain the names of all team members and only ONE submission per team. Use the provided report template
4. Sharing of code or solution between teams is not allowed. Copy/Paste code from the Internet (without understanding), and proper citation is not allowed. Any of these behaviours will be considered academic misconduct and result in a zero grade for the project.

2 Bonus

- You can get up to 10 points bonus divided equally among the team members. Students can use the bonus points to boost their scores on the quizzes they have attended.
- To obtain the bonus, you need to extend your implementation of the RDT V2.2 by implementing RDTV3.0, which can handle packet loss using timers.
- This will require you to modify the **network.py** file to simulate the packet loss.

3 Report Template

Here are the key sections for

1. Team members' names (students' ids, tutorial number, and email addresses) and a short description of the contribution of each member.
2. FSM diagrams for the sender and receiver (do not copy and paste from other sources).
3. The pseudo-code of the RDT sender and receiver sides, and use proper pseudo-code format that is programming language agnostic (do not dump source code)
4. List any changes you have made to the skeleton code and explain why these changes were necessary **NOTE:** you are not allowed to edit the **network.py** file
5. Test case of your implementation and screenshots of executing these test cases and their results.

4 Grading Scheme

- 30% Project Report
- 70% RDTv2.2 Implementation

5 Appendixes

Here is the code skeleton

5.1 RDTv2.2 Main

```python
from network import NetworkLayer
from receiver import ReceiverProcess
from sender import SenderProcess, RDTSender
import sys

if __name__ == '__main__':
    args = dict([arg.split('=', maxsplit=1) for arg in sys.argv[1:]])
    print(args)
    msg = args['msg']
    prob_to_deliver = float(args['rel'])
    delay = int(args['delay'])
    debug = bool(int(args['debug']))
    corrupt_pkt = True
    corrupt_ack = True
    if debug:
        corrupt_pkt = bool(int(args['pkt']))
        corrupt_ack = bool(int(args['ack']))

    SenderProcess.set_outgoing_data(msg)

    print(f'Sender is sending:{SenderProcess.get_outgoing_data()}')

    network_serv = NetworkLayer(reliability=prob_to_deliver, delay=delay, pkt_corrupt=corrupt_pkt,
                                ack_corrupt=corrupt_ack)

    rdt_sender = RDTSender(network_serv)
    rdt_sender.rdt_send(SenderProcess.get_outgoing_data())

    print(f'Receiver received: {ReceiverProcess.get_buffer()}')
```

5.2 RDTv2.2 Sender

```python
class SenderProcess:
    """ Represent the sender process in the application layer  """

    __buffer = list()

    @staticmethod
    def set_outgoing_data(buffer):
        """ To set the message the process would send out over the network
        :param buffer:  a python list of characters represent the outgoing message
        :return: no return value
        """
        SenderProcess.__buffer = buffer
        return

    @staticmethod
    def get_outgoing_data():
        """ To get the message the process would send out over the network
        :return:  a python list of characters represent the outgoing message
        """
        return SenderProcess.__buffer


class RDTSender:
    """ Implement the Reliable Data Transfer Protocol V2.2 Sender Side """

    def __init__(self, net_srv):
        """ This is a class constructor
            It initialize the RDT sender sequence number  to '0' and the network layer services
            The network layer service provide the method udt_send(send_pkt)
        """
        self.sequence = '0'
        self.net_srv = net_srv

    @staticmethod
    def get_checksum(data):
        """ Calculate the checksum for outgoing data
        :param data: one and only one character, for example data = 'A'
        :return: the ASCII code of the character, for example ASCII('A') = 65
        """
        # TODO provide your own implementation
        checksum = None  # you need to change that
        return checksum

    @staticmethod
    def clone_packet(packet):
        """ Make a copy of the outgoing packet
        :param packet: a python dictionary represent a packet
        :return: return a packet as python dictionary
        """
        pkt_clone = {
            'sequence_number': packet['sequence_number'],
            'data': packet['data'],
            'checksum': packet['checksum']
        }
        return pkt_clone

    @staticmethod
    def is_corrupted(reply):
        """ Check if the received reply from receiver is corrupted or not
        :param reply: a python dictionary represent a reply sent by the receiver
        :return: True -> if the reply is corrupted | False ->  if the reply is NOT corrupted
        """
        # TODO provide your own implementation
        pass

    @staticmethod
    def is_expected_seq(reply, exp_seq):
        """ Check if the received reply from receiver has the expected sequence number
        :param reply: a python dictionary represent a reply sent by the receiver
        :param exp_seq: the sender expected sequence number '0' or '1' represented as a
    character
        :return: True -> if ack in the reply match the   expected sequence number otherwise
    False
        """
        # TODO provide your own implementation
        pass

    @staticmethod
    def make_pkt(seq, data, checksum):
        """ Create an outgoing packet as a python dictionary
        :param seq: a character represent the sequence number of the packet, the one
    expected by the receiver '0' or '1'
        :param data: a single character the sender want to send to the receiver
        :param checksum: the checksum of the data the sender will send to the receiver
        :return: a python dictionary represent the packet to be sent
        """
        packet = {
            'sequence_number': seq,
            'data': data,
            'checksum': checksum
        }
        return packet

    def rdt_send(self, process_buffer):
        """ Implement the RDT v2.2 for the sender
        :param process_buffer:  a list storing the message the sender process wish to send
    to the receiver process
        :return: terminate without returning any value
        """

        # for every character in the buffer
        for data in process_buffer:

            checksum = RDTSender.get_checksum(data)
            pkt = RDTSender.make_pkt(self.sequence, data, checksum)
            reply = self.net_srv.udt_send(pkt)

        print(f'Sender Done!')
        return
```

5.3 RDTv2.2 Receiver

```python
class ReceiverProcess:
    """ Represent the receiver process in the application layer  """
    __buffer = list()

    @staticmethod
    def deliver_data(data):
        """ deliver data from the transport layer RDT receiver to the application layer
        :param data: a character received by the RDT RDT receiver
        :return: no return value
        """
        ReceiverProcess.__buffer.append(data)
        return

    @staticmethod
    def get_buffer():
        """ To get the message the process received over the network
        :return:  a python list of characters represent the incoming message
        """
        return ReceiverProcess.__buffer


class RDTReceiver:
    """" Implement the Reliable Data Transfer Protocol V2.2 Receiver Side """

    def __init__(self):
        self.sequence = '0'

    @staticmethod
    def is_corrupted(packet):
        """ Check if the received packet from sender is corrupted or not
            :param packet: a python dictionary represent a packet received from the sender
            :return: True -> if the reply is corrupted | False ->  if the reply is NOT corrupted
        """
        # TODO provide your own implementation
        pass

    @staticmethod
    def is_expected_seq(rcv_pkt, exp_seq):
        """ Check if the received reply from receiver has the expected sequence number
         :param rcv_pkt: a python dictionary represent a packet received by the receiver
         :param exp_seq: the receiver expected sequence number '0' or '1' represented as a character
         :return: True -> if ack in the reply match the   expected sequence number otherwise False
        """
        # TODO provide your own implementation
        pass


    @staticmethod
    def make_reply_pkt(seq, checksum):
        """ Create a reply (feedback) packet with to acknowledge the received packet
        :param seq: the sequence number '0' or '1' to be acknowledged
        :param checksum: the checksum of the ack the receiver will send to the sender
        :return:  a python dictionary represent a reply (acknowledgement)  packet
        """
        reply_pck = {
            'ack': seq,
            'checksum': checksum
        }
        return reply_pck

    def rdt_rcv(self, rcv_pkt):
        """  Implement the RDT v2.2 for the receiver
        :param rcv_pkt: a packet delivered by the network layer 'udt_send()' to the receiver
        :return: the reply packet
        """

        # TODO provide your own implementation

        # deliver the data to the process in the application layer
        ReceiverProcess.deliver_data(rcv_pkt['data'])

        #reply_pkt = RDTReceiver.make_reply_pkt()
        #return reply_pkt

        return None
```

5.4 Network Layer

```
import random
import time
from receiver import RDTReceiver

"""
NOTE: YOU SHOULD NOT MODIFY THIS CLASS
"""


class NetworkLayer:
    """ The network layer that deliver packets and acknowledgments between sender and
    receiver """

    def __init__(self, reliability=1.0, delay=1.0, pkt_corrupt=True, ack_corrupt=True):
        """ initialize the network layer
        :param reliability: the probability that the network layer will deliver the message
    correctly
        :param delay: the round trip time for sending a packet and receive a reply
        :param pkt_corrupt: sender packets will be corrupted
        :param ack_corrupt: receiver acknowledgments will be corrupted
        """
        self.reliability = reliability
        self.packet = None
        self.reply = None
        self.delay = delay
        self.pkt_corrupt = pkt_corrupt
        self.ack_corrupt = ack_corrupt
        self.recv = RDTReceiver()  # connect the network layer to the receiver

    def get_network_reliability(self):
        """ show network layer reliability
        :return: a float number represent the current network reliability
        """
        return self.reliability

    def __packet_corruption_probability(self):
        """ calculate the probability that a pocket will be corrupted
        :return: True if the probability greater than the network reliability
        """
        ran = random.uniform(0, 1)
        if ran > self.reliability:
            return True
        return False

    def __corrupt_packet(self):
        """ Corrupt the sender packet, it could corrupt the seq_num, the data or the
    checksum
        :return: no return value
        """
        ran = random.randint(1, 90)
        if ran < 30:
            self.packet['sequence_number'] = chr(random.randint(ord('2'), ord('9')))
            return
        if ran < 60:
            self.packet['data'] = chr(random.randint(ord('!'), ord('}')))
            return
        if ran < 90:
            self.packet['checksum'] = random.randint(ord('!'), ord('}'))

    def __corrupt_reply(self):
        """ Corrupt the receiver reply (acknowledgments) packet
        :return: no return value
        """
        ran = random.randint(1, 100)
        if ran < 50:
            self.reply['ack'] = chr(random.randint(2, 9))
        else:
            self.reply['checksum'] = chr(random.randint(ord('2'), ord('9')))

    def udt_send(self, frame):
        """ implement the delivery service of the unreliable network layer
        :param frame: a python dictionary represent the a sender's packet or a receiver's
    reply
        :return: the receiver's reply as a python dictionary returned to the sender
        """

        # TODO: You may add ONLY print statements to this function for debugging purpose
        self.packet = frame
        s_test = self.__packet_corruption_probability()

        if s_test and self.pkt_corrupt:
            self.__corrupt_packet()

        time.sleep(self.delay)

        # bridge|connect the RDT sender and receiver
        self.reply = self.recv.rdt_rcv(self.packet)

        r_test = self.__packet_corruption_probability()
        if r_test and self.ack_corrupt:
            self.__corrupt_reply()

        return self.reply
```