Design File

Title and Authors

- Phase 4
- Shiyang Chen, Chun-Jen Lin, Aniket Bhatt

Purpose of The Phase

This phase is a simple implement for RDT 3.0 over an unreliable UDP channel with bit-errors and packet-loss.

We build two program for send and receive respectively. Both programs can intentionally set bit-error and packet-loss probabilities for simulating real channel.

Code Explanation

Protocol

The protocol we used is a modified RDT consisting of 2 types of packet, data packet and end packet. Data packet is of length 1032, including 4 bytes of sequence number(an integer) ,4 bytes of checksum(an integer) and 1024 bytes data payload.

According to the RDT 3.0 protocol, the sequence number will be 0 or 1. The ACK packet has two types: ACK0 and ACK1. The data payload field of ACK1 is filled with consecutive 1. The data payload field of ACK0 is filled with consecutive 0.

The packets may loss during transmission. Therefore, there is a countdown timer at sender side to determine whether the packet is loss. The timer starts after that the data packet is sent. If the time of waiting for ACK is exceeded to the limit, it causes a timeout and the packet will be resent.

The end packet is of length 4, which content is determined 0000. When receiver get the end packet, it means the transmission is over and all connection will be closed.

Functions:

- isCorrupt: To check whether the received packet is corrupted
- is_ACK: To check whether the received ACK packet is the one expected.
- udt_send: A warp of UDP socket, which send the packet over UDP protocol to target machine.
- extract: Extract the packet into sequence number, checksum and data payload
- readFile: Read file into bytes
- data_iter: An iterator which produces fixed-length bytes segment from the input bytes
- checksum: Calculate the checksum of bytes

- make_pkt: Assemble bytes into packet by adding sequence number and checksum ahead
- rdt_send: The overall function controlling sending packet
- rdt_rcv: The overall function controlling receiving packet
- make_error: Intentionally damage some bytes in the packet for experiments.

Sender

Sender program is implemented as a FSM which has four states:

1. WAIT_CALL_0
2. WAIT_CALL_1
3. WAIT_ACK_0
4. WAIT_ACK_1

It calls the function 'send' which is the sending API of RDT 2.2.

Receiver

Receiver program is implemented as a FSM which has two states:

1. WAIT_0
2. WAIT_1

It calls the function 'recv' which is the sending API of RDT 2.2.

Code snippets

timer

The timer is executed at another thread. The main thread send signal to timer thread to start and restart the timer. When timeout, the timer resend the packet.

```
def timer(): # another thread for timer
    global sndpkt, address, send_to_port, t, timeout
    # wait for green light from the main thread
    event.wait()
    couting_time = time.time()
    while event.is_set():
        if (time.time() - couting_time) > timeout:  # timeout
            udt_send(sndpkt, address, send_to_port)  # resend
            print("resend due to timeout")
            couting_time = time.time()
```

rdt_rcv

The rdt_rcv receives the ACK packet from receiver. If the error rate and loss rate are setted, it will randomly damage the received packet or prevent the

received packet to be processed further.

```
def rdt_rcv(address, port) -> bytes:
    global error_prob
    global loss_prob
    rcvpkt, _ = s.recvfrom(2048)
    if random.random() < error_prob:  # intentionally make error to ACK pkt
        return make_error(rcvpkt)
    elif random.random() < loss_prob:  # intentionally make loss to ACK pkt
        return None
    else:
        return rcvpkt
```

send

The send function implement the RDT 3.0 sender FSM and its main logic.

```
def send(data, address, send_to_port, recv_from_port):
    it = data_iter(data)
    global State, sndpkt, t
    State = STATE.WAIT_CALL_0
    print("Start sending")
    while True:
        try:
            if State == STATE.WAIT_CALL_0:
                sndpkt = rdt_send(next(it), address, send_to_port)
                State = STATE.WAIT_ACK_0
                t += 1
                print("Packet No." + str(t) + " sent.")
            elif State == STATE.WAIT_CALL_1:
                sndpkt = rdt_send(next(it), address, send_to_port)
                State = STATE.WAIT_ACK_1
                t += 1
                print("Packet No." + str(t) + " sent.")
            elif State == STATE.WAIT_ACK_0:
                t1 = threading.Thread(target=timer, daemon=True)
                t1.start()       # start the timer
                event.set()                                          # start timer
                rcvACK = rdt_rcv(address, recv_from_port)
                if rcvACK == None:  # timeout
                    time.sleep(0.04)
                elif is_ACK(rcvACK, 0) and not isCorrupt(rcvACK):  # Successful send
                    State = STATE.WAIT_CALL_1
                else:
                    print("Resend Packet No." + str(t))
                    udt_send(sndpkt, address, send_to_port)     # resend
                event.clear()                                       # stop timer
```

```
            elif State == STATE.WAIT_ACK_1:
                t1 = threading.Thread(target=timer, daemon=True)
                t1.start()          # start the waiting timer
                event.set()         # start timer
                rcvACK = rdt_rcv(address, recv_from_port)
                if rcvACK == None:  # timeout
                    time.sleep(0.04)
                elif is_ACK(rcvACK, 1) and not isCorrupt(rcvACK):  # Successful send
                    State = STATE.WAIT_CALL_0
                else:
                    print("Resend Packet No." + str(t))
                    udt_send(sndpkt, address, send_to_port)      # resend
                event.clear()         # stop timer
            else:
                exit(1)  # Unexpected error

        except StopIteration:
            udt_send(b"0000", address, send_to_port)  # end packet
            print("Send finish")
            break  # All data sent
```

make and extract packet

The struct library is used to serialize and deserialize data into binary.

```
def make_pkt(data, seqNum) -> bytes:
    fmt = "!II" + str(PACKET_SIZE) + "s"  # !II1024s network byte order
    chksum = checksum(data)
    return struct.pack(fmt, seqNum, chksum, data)

def extract(pkt):
    # Extract the packet into sequence number, checksum and data payload
    fmt = "!II" + str(PACKET_SIZE) + "s"  # !II1024s network byte order
    seqNum, chksum, data = struct.unpack(fmt, pkt)
    return seqNum, chksum, data
```

checksum

```
def checksum(pkt) -> int:
    sum = 0
    # convert binary data to hexadecimal for checksum
    data_hex = binascii.hexlify(pkt)
    for i in data_hex:
        sum = sum + int(str(i), 16)
    return sum
```

recv

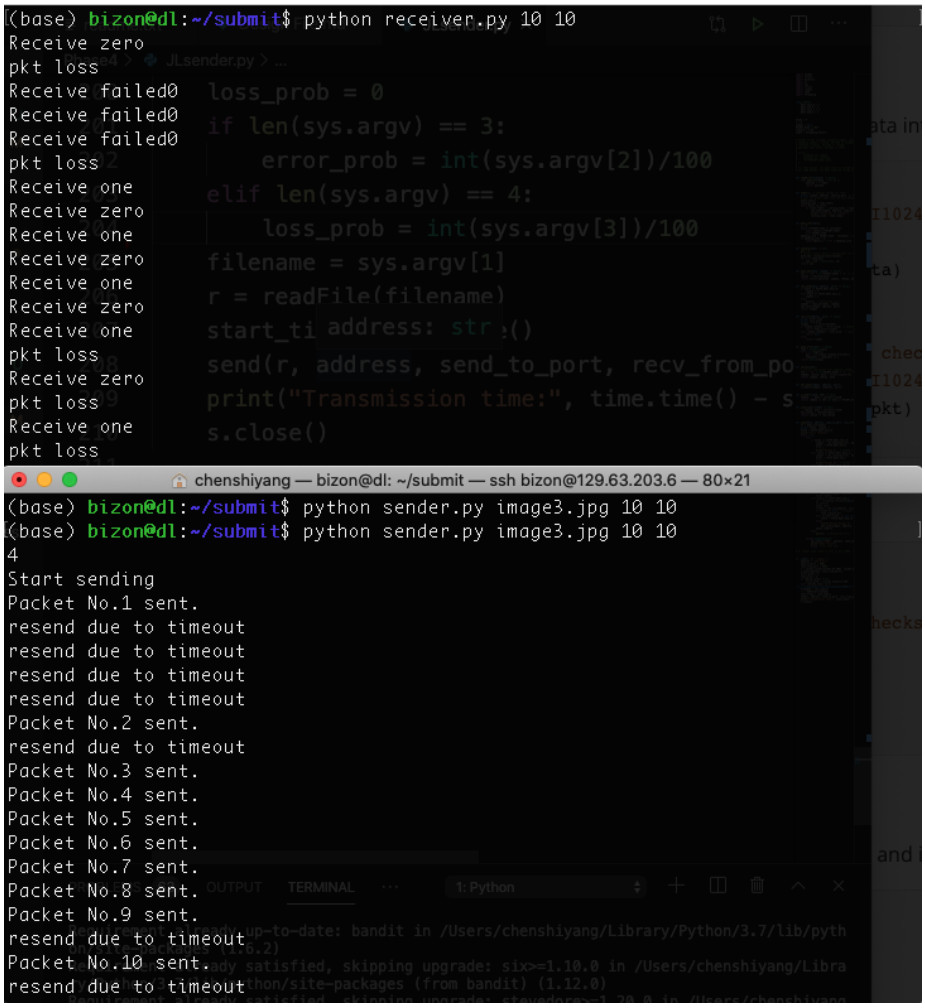
The send function implement the RDT 3.0 sender FSM and its main logic.

```
def recv(send_to_port, recv_from_port):
    State = STATE.WAIT_0
    data = []
    rcvpkt = None
    sndpkt = None
    seqNum = 0
    client = None
    oncethru = 0
    while True:
        if State == STATE.WAIT_0:
            rcvpkt, client = rdt_rcv(recv_from_port)
            if rcvpkt is None:
                print("pkt loss")
                continue  # DATA pkt loss
            if len(rcvpkt) == 4:  # end pkt
                break
            if isCorrupt(rcvpkt) or has_seq(1, rcvpkt):  # receive fail
                print("Receive failed1")
                sndpkt = make_pkt(ACK1, 1)
                udt_send(sndpkt, client[0], send_to_port)  # resend
            else:
                print("Receive zero")
                oncethru = 1
                data.append(extract(rcvpkt)[2])
                sndpkt = make_pkt(ACK0, 0)
                udt_send(sndpkt, client[0], send_to_port)
                State = STATE.WAIT_1
        elif State == STATE.WAIT_1:
            rcvpkt, client = rdt_rcv(recv_from_port)
            if rcvpkt is None:
                print("pkt loss")
                continue  # DATA pkt loss
            if len(rcvpkt) == 4:
                break
            if isCorrupt(rcvpkt) or has_seq(0, rcvpkt):  # receive fail
                print("Receive failed0")
                sndpkt = make_pkt(ACK0, 0)
                udt_send(sndpkt, client[0], send_to_port)  # resend
            else:
                print("Receive one")
                data.append(extract(rcvpkt)[2])
                sndpkt = make_pkt(ACK1, 1)
```

```
                udt_send(sndpkt, client[0], send_to_port)
                State = STATE.WAIT_0
        else:
            exit(1)
    return b"".join(data)
```

Execution Example

Start the programs



Sending complete