Problem Statement

Implement three data link layer protocols, Stop and Wait, Go Back N Sliding Window and Selective Repeat Sliding Window for flow control.

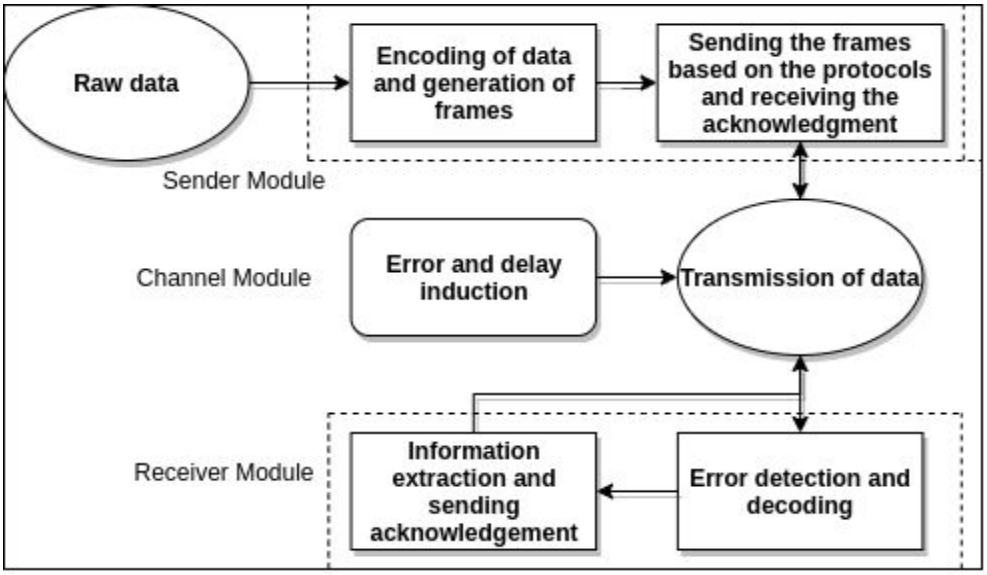
Design description

`Purpose of the program:` The program tries to simulate the environment of data communication at data link layer for analysing the performances of various flow control modules namely Stop and Wait , Go Back N Sliding Window and Selective Repeat Sliding Window. The key points are -

1. Encoding Data using error handling modules.
2. Sending data over the channel as per the protocol
3. Induction of errors into the data while transmitting.
4. Receiving data from the channel.
5. Error detection and if no error is detected, extracting the original data.
6. Sending Acknowledgement through the channel.
7. Receiving the acknowledgement and repeating the above steps until all the frames are sent successfully.

Now, due to simulator constraints, it is very less probable for any error or delay to be induced int the data. Hence, a channel module is created to induce errors and delay based on the analysis requirements. To achieve a more realistic view, errors are induced randomly into the sent data. In the later sections of the report, a complete analysis of the results over various possible test cases is shown for further examination of different methods used.

`Structural design:`



1. The raw data is read from the input file.
2. CRC is used to encode the data.
3. The program sends the data onto the buffer file.

4. The channel module induces error and a delay on the transmitted data.
5. Receiver module receives the data from the intermediate file.
6. Error is being detected using the scheme used to generate the code words.
7. If the program does not detect any error, an acknowledgement is sent to the sender via channel.
8. The sender module receives the acknowledgement and sends the next frame.
9. The channel module corrupts the data and/or fails the transmission based on a certain probability.

Input and output format:

1. The input is read from the file “input.txt”.
2. The logs are shown in a file named “temp_file.txt”.
3. The received message is stored in the file “output.txt”.

Code Snippet

Stop and Wait

Sender Module:

The sender opens a socket and binds to localhost and predefined port.
Input is taken from the user and using the CRC module in Ass1.

Iterate over the frames, and for each frame, **send** the frame using the method**.**

```
def _send_one_frame(self, frame, conn, corrupt_simulation=False):
      # Put in corruptions
      if randrange(0,10) <=2 and corrupt_simulation:
          frame = self.crc._corrupt_frame(frame)
      conn.sendall(str(frame).zfill(8).encode('utf-8'))
      # conn.flush()
      print('{} sent'.format(str(frame).zfill(8)))
```

Then it waits for an ack

```
def _receive_ack(self, conn):
      print('waiting for ack')
      ack = int(conn.recv(1))
      print('Received ack {}'.format(True if ack == 1 else False))
      if ack == 1:
          return True
      else:
          return False
```

If the ack is False, the frame is sent again.

Corruption simulation

After the frames are made, a random bit is flipped.

Receiver Module:

The receiver waits for data.When a frame arrives, it decodes and checks for error.

```
def _receive_one_frame(self):
      frame = self.socket.recv(8).decode('utf_8')
      print('Frame {} received'.format(frame))
      if frame:
          return int(frame)
      return None
```

If the frame is not corrupt it sends a True ack, else a False ack.

```
def _send_ack(self, ack):
      if ack:
          self.socket.send('1'.encode('utf-8'))
      else:
          self.socket.send('0'.encode('utf-8'))
      print('Ack sent {}'.format(ack))
```

If the frame was valid it added is to the list of received frames. Finally all the frames are assembled and the received text is dumped into a file.

Go Back N

Sender Module:

It maintains a window size, with a low and high pointer. After sending all frames in that window it waits for ack for TTL time. If TTL expires it resends the frames

```
while 1:
 x = open(tempfile, 'rb')
  fcntl.flock(x, fcntl.LOCK_EX )
  status = pickle.load(x)
  fcntl.flock(x, fcntl.LOCK_UN)  # print(status)
if status['transfer_complete']:
  print( "File transfer Complete ..")
  print( "Total time in seconds : " + str(time.time() - start_time))
  Break
print('Trying to send {}'.format(seq_no_to_send))
  message_to_send = pack_data(seq_no_to_send, frames[seq_no_to_send])
  client_socket.sendto(message_to_send,(server_host,port))
  # set_timeout(send_seq_no,tempfile)
  seq_no_to_send += 1
  if seq_no_to_send > status['window_high']:
    seq_no_to_send = status['window_low']
```

In another thread it receives the ack. If ack is found for a seq no, the window slides forwards.

```
def rcv_thread(soc):
    global window_size
    global total_frames
    global tempfile
    new_proc = os.fork()
    if new_proc == 0:
        print("Receive Thread " + str(os.getpid()) + " created..")
        while 1:
            while os.stat(tempfile).st_size == 0:
                Continue
            status = pickle.load(open(tempfile, 'rb'))
            print(status)
            message, server_addr = soc.recvfrom(max_buff)
            message = pickle.loads(message)
            seq_no, ack = message['seq_num'], message['ack']
            print('\033[93mAck Rec {} value {}\033[0m'.format(seq_no, ack))
            if seq_no == total_frames-1:
                status['transfer_complete'] = True
                Break
            if ack:
                status['window_low'] = seq_no+1
                status['window_high'] = status['window_low'] + window_size
                x = open(tempfile, 'wb')
                fcntl.flock(x, fcntl.LOCK_EX )
                pickle.dump(status, open(tempfile, 'wb') )
                fcntl.flock(x, fcntl.LOCK_UN)
```

Receiver module:

Waits for frames and when found sends an ack if valid to the sender with the seq no received

```
while 1:
  msg, client_address = s.recvfrom(65535)
  data = pickle.loads(msg)
  got_seq_no, frame = data['seq_num'], data['frame']
  print('Packet Recv seq_no {} '.format(got_seq_no))
  if random() < probability: # Random packet drop simulation
    decoded = crc.decode([frame], verbose=True)
    if decoded is not None: # Checksum is correct
      if got_seq_no == exp_seq_no: # Send ack
        to_send = rdt_send(got_seq_no)
        if to_send:
          s.sendto(to_send, client_address)
          f_write(decoded,output_file)
          exp_seq_no = exp_seq_no + 1
      elif got_seq_no > exp_seq_no: # Future packet received, hence dropped
        print ("Packet loss, sequence number = " + str(got_seq_no))
      elif got_seq_no < exp_seq_no: # Repeat sent
        to_send = rdt_send(exp_seq_no
```

```
        if to_send:
        # print ("Retransmitted ACK - " + str(got_seq_no))
          s.sendto(to_send, client_address)
    else:
      print ("Codeword invalid. Packet dropped.")
  else: # Random packet drop simulation
    print ("Packet loss, sequence number = " + str(got_seq_no))
```

Selective Repeat ARQ

Receiver module:

It maintains a window size, with a low and high pointer. After receiving acknowledgement for the leftmost frame in that window it slides the window else waits for ack for TTL time. If TTL expires it resends the frames.

```
while 1:
 x = open(tempfile, 'rb')
  fcntl.flock(x, fcntl.LOCK_EX )
  status = pickle.load(x)
  fcntl.flock(x, fcntl.LOCK_UN)  # print(status)
if status['transfer_complete']:
  print( "File transfer Complete ..")
  print( "Total time in seconds : " + str(time.time() - start_time))
  Break
print('Trying to send {}'.format(seq_no_to_send))
  message_to_send = pack_data(seq_no_to_send, frames[seq_no_to_send])
  client_socket.sendto(message_to_send,(server_host,port))
  # set_timeout(send_seq_no,tempfile)
  seq_no_to_send += 1
  if seq_no_to_send > status['window_high']:
    seq_no_to_send = status['window_low']
```

In another thread it receives the ack. If ack is found for a seq no, the window slides forwards.

```
def rcv_thread(soc):
    global window_size
    global total_frames
    global tempfile
    new_proc = os.fork()
    if new_proc == 0:
        print("Receive Thread " + str(os.getpid()) + " created..")
        while 1:
            while os.stat(tempfile).st_size == 0:
                Continue
            status = pickle.load(open(tempfile, 'rb'))
            print(status)
            message, server_addr = soc.recvfrom(max_buff)
            message = pickle.loads(message)
            seq_no, ack = message['seq_num'], message['ack']
```

```
            print('\033[93mAck Rec {} value {}\033[0m'.format(seq_no, ack))
            if seq_no == total_frames-1:
                status['transfer_complete'] = True
                Break
            if ack:
                status['window_low'] = seq_no+1
                status['window_high'] = status['window_low'] + window_size
                x = open(tempfile, 'wb')
                fcntl.flock(x, fcntl.LOCK_EX )
                pickle.dump(status, open(tempfile, 'wb') )
                fcntl.flock(x, fcntl.LOCK_UN)
```

Receiver module:

Waits for frames and when found sends an **ack** if valid to the sender with the seq no. received. If the frame received is not the expected one it sends **NAK** for the expected frame.

```
while 1:
  msg, client_address = s.recvfrom(65535)
  data = pickle.loads(msg)
  got_seq_no, frame = data['seq_num'], data['frame']
  print('Packet Recv seq_no {} '.format(got_seq_no))
  if random() < probability: # Random packet drop simulation
    decoded = crc.decode([frame], verbose=True)
    if decoded is not None: # Checksum is correct
      if got_seq_no == exp_seq_no: # Send ack
        to_send = rdt_send(got_seq_no)
        if to_send:
          s.sendto(to_send, client_address)
          f_write(decoded,output_file)
          exp_seq_no = exp_seq_no + 1
      elif got_seq_no > exp_seq_no: # Future packet received, hence dropped
        print ("Packet loss, sequence number = " + str(got_seq_no))
      elif got_seq_no < exp_seq_no: # Repeat sent
        to_send = rdt_send(exp_seq_no
        if to_send:
        # print ("Retransmitted ACK - " + str(got_seq_no))
          s.sendto(to_send, client_address)
    else:
      print ("Codeword invalid. Packet dropped.")
  else: # Random packet drop simulation
    print ("Packet loss, sequence number = " + str(got_seq_no))
```

Output Log

STOP and WAIT

Input

00110101010101111111000011

Log

Simulating StopAndWait Protocol for sender

ENCODING

Frame 000 Str '001' Data ['00110000', '00110000', '00110001'] CRC 00001
Frame 003 Str '101' Data ['00110001', '00110000', '00110001'] CRC 00110
Frame 006 Str '010' Data ['00110000', '00110001', '00110000'] CRC 00100
Frame 009 Str '101' Data ['00110001', '00110000', '00110001'] CRC 00110
Frame 012 Str '011' Data ['00110000', '00110001', '00110001'] CRC 00111
Frame 015 Str '111' Data ['00110001', '00110001', '00110001'] CRC 00000
Frame 018 Str '000' Data ['00110000', '00110000', '00110000'] CRC 00010
Frame 021 Str '011' Data ['00110000', '00110001', '00110001'] CRC 00111
Frame 024 Str '\n ' Data ['00001010', '00100000', '00100000'] CRC 00001
25264523 sent
waiting for ack
Received ack False
25264521 sent
waiting for ack
Received ack True
25788814 sent
waiting for ack
Received ack True
25266564 sent
waiting for ack
Received ack True
25788815 sent
waiting for ack
Received ack False
25788814 sent
waiting for ack
Received ack True
25266575 sent
waiting for ack
Received ack True
25790857 sent
waiting for ack
Received ack False
25790856 sent
waiting for ack
Received ack True
25264512 sent
waiting for ack
Received ack False
25264514 sent
waiting for ack
Received ack True
25266575 sent
waiting for ack
Received ack True
05308673 sent

waiting for ack
Received ack True

Simulating StopAndWait Protocol for receiver
Frame 25264523 received
Frame no.0 is corrupted. corrupted value: 110000001100000011000101l
Ack sent False
Frame 25264521 received
Ack sent True
Frame 25788814 received
Ack sent True
Frame 25266564 received
Ack sent True
Frame 25788815 received
Frame no.0 is corrupted. corrupted value: 110001001100000011000111l
Ack sent False
Frame 25788814 received
Ack sent True
Frame 25266575 received
Ack sent True
Frame 25790857 received
Frame no.0 is corrupted. corrupted value: 110001001100010011000100l
Ack sent False
Frame 25790856 received
Ack sent True
Frame 25264512 received
Frame no.0 is corrupted. corrupted value: 110000001100000011000000l
Ack sent False
Frame 25264514 received
Ack sent True
Frame 25266575 received
Ack sent True
Frame 05308673 received
Ack sent True
Frame received
001101010101011111000011

Go Back N

Input

001101010101011111000011

Log

Sender
{'transfer_complete': False, 'window_high': 10, 'window_low': 2}
Ack Rec 2 value True
{'transfer_complete': False, 'window_high': 11, 'window_low': 3}
Ack Rec 3 value True
{'transfer_complete': False, 'window_high': 12, 'window_low': 4}
Ack Rec 3 value True
{'transfer_complete': False, 'window_high': 12, 'window_low': 4}
Ack Rec 4 value True
{'transfer_complete': False, 'window_high': 13, 'window_low': 5}
Ack Rec 5 value True
{'transfer_complete': False, 'window_high': 14, 'window_low': 6}
Ack Rec 5 value True
{'transfer_complete': False, 'window_high': 14, 'window_low': 6}
Ack Rec 5 value True
{'transfer_complete': False, 'window_high': 14, 'window_low': 6}
Ack Rec 6 value True

{'transfer_complete': False, 'window_high': 15, 'window_low': 7}
Ack Rec 7 value True
{'transfer_complete': False, 'window_high': 16, 'window_low': 8}
Ack Rec 7 value True
{'transfer_complete': False, 'window_high': 16, 'window_low': 8}
Ack Rec 7 value True
{'transfer_complete': False, 'window_high': 16, 'window_low': 8}
Ack Rec 7 value True
{'transfer_complete': False, 'window_high': 16, 'window_low': 8}
Ack Rec 8 value True

Receiver

Listening for client requests ...
Packet Recv seq_no 0
Packet loss, sequence number = 0
Packet Recv seq_no 1
Packet loss, sequence number = 1
Packet Recv seq_no 2
Packet loss, sequence number = 2
Packet Recv seq_no 3
Packet loss, sequence number = 3
Packet Recv seq_no 4

DECODING

Frame 000 Data 1100000011000100110001 Str '011'
Packet loss, sequence number = 4
Packet Recv seq_no 5
Packet loss, sequence number = 5
Packet Recv seq_no 6

DECODING

Frame 000 Data 1100000011000000110000 Str '000'
Packet loss, sequence number = 6
Packet Recv seq_no 7

DECODING

Frame 000 Data 1100000011000100110001 Str '011'
Packet loss, sequence number = 7
Packet Recv seq_no 0

DECODING

Frame 000 Data 1100000011000000110001 Str '001'
Packet Recv seq_no 1
Packet loss, sequence number = 1
Packet Recv seq_no 2
Packet loss, sequence number = 2
Packet Recv seq_no 3
Packet loss, sequence number = 3
Packet Recv seq_no 4

DECODING

Frame 000 Data 1100000011000100110001 Str '011'
Packet loss, sequence number = 4
Packet Recv seq_no 5

Analysis of Algorithm

RTT (in ms)

Frame No.	**Stop and Wait**
0	5
1	8
2	7
3	2010
4	9
Average:	407.8

Discussion

Stop and Wait	**Go Back N**	**Selective Repeat ARQ**
Sender sends one frame and wait for the acknowledgment from the receiver side	Sender sends more than one frame to the receiver side and re-transmit the frame which is/are damaged or suspected.	Sender sends more than one frame to the receiver side and re-transmit the frame which is/are damaged or suspected.
The receiver just receives frame one by one and sends the acknowledgement.	The receiver receives the frames in an ordered manner and sends ack to the sender.	The receiver receives the frames in an unordered manner and sends ack and nak to the sender.
Less efficient.	Better than Stop and Wait in terms of efficiency.	Better than Go Back N in terms of efficiency.
Half Duplex Algorithm	Full Duplex Algorithm	Full Duplex Algorithm

Comments

- The assignment helps the student to understand the flow of the transmission of data in a network. It also helps the student to understand pros and cons of using a flow control module and helps choose the required module appropriately in real life scenarios.
- The assignment was not too hard nor too easy from a student's point of view. This task helped understand the concepts of socket programming. The concepts of threading and multiprocessing was tested as well.