ECE 570 Computer Networks

Project I: A Simple Data Link Layer Protocol

By *Jason Zhao* 4405 3635

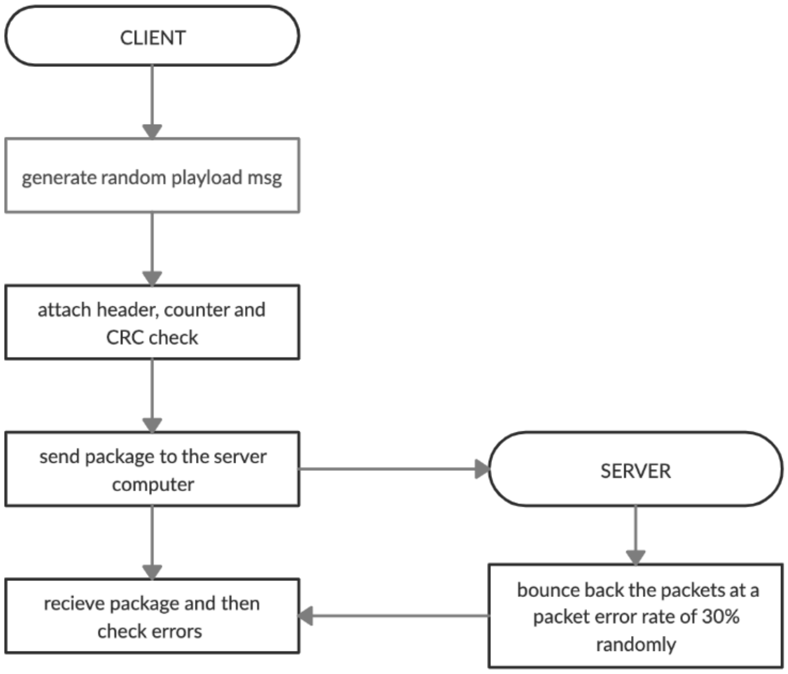
Project Descriptions

This project is composed of 2 parts, the first one is about a random package rending, receiving and analyzing and the second part is zbout reading, formatting and sending a image into packages.

Part 1 Random Package Sending and Receiving

Using UDP socket programs, the client computer sends 100 packages to the server computer. We are required to generate a random payload massage of 256 Bytes and form test packages. Each frame contains a header of 01111110, a 1 Byte frame counter, and is attached with a CRC check by using a divisor 11001 or standard CRC16. The server will bounce back the packets at a packet error rate of 30% randomly. When receiving the packet, perform CRC check at client. The program is required to be using stop and wait automatic repeat request (ARQ) for error control by resenting incorrect packet until it is correctly received.

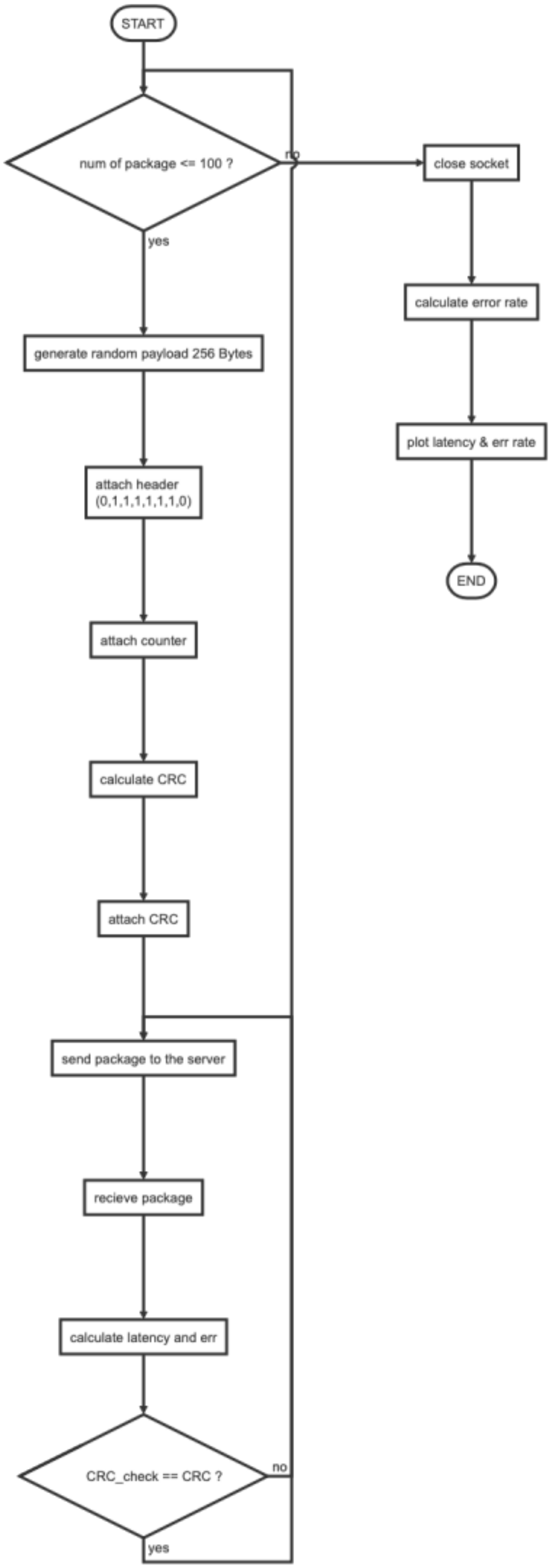
Below is the diagram of the system:


CLIENT
generate random playload msg
attach header, counter and CRC check
send package to the server computer
SERVER
recieve package and then check errors
bounce back the packets at a packet error rate of 30% randomly

The frame configuration shows the structure of the package as below:

Header 1 Byte	Counter 1 Byte	Message 256 Bytes	CRC 2 Byte

Below is the flow chart representing the logic of the programming for this part.


START
num of package <= 100 ?
no
yes
generate random payload 256 Bytes
attach header (0,1,1,1,1,1,1,0)
attach counter
calculate CRC
attach CRC
send package to the server
recieve package
calculate latency and err
CRC_check == CRC ?
no
yes
close socket
calculate error rate
plot latency & err rate
END

Sparse source code:

Several functions are defined for this part so that they can be applied directly in the main body.

Function that makes the package's form to be represented in bt and int:

```
def msg2bt(msg):
    bt=[]
    for i in range(len(msg)):
        for b in msg[i]:
            bt.append(int(b))
    return bt

def bt2int(bt):
    w=2**np.array(range(8))[::-1]
    return np.dot(bt,w)
```

Function that calculates the CRC of the frame: input a divisor (in this situation the divisor is [1,1,0,0,1]) and a message, and the function returns the CRC.

```
def getCRC(bt,div):
    N = len(div) # divisor (length)
    tmp = bt[0:N] # first N bits of the bitstream
    K = len(bt) # message (beatstream length)
    for i in range(K-N+1):
        if tmp[0]==1:
            tmp=[i^j for i,j in zip(div,tmp)]
        if i!=K-N:
            tmp = tmp[1:]+[bt[i+N]]
    return tmp[1:]
```

The server address is given by

```
# server address
sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
server_address=('199.223.115.24',9997)
```

And then the main body that is surpported by the flow chart:

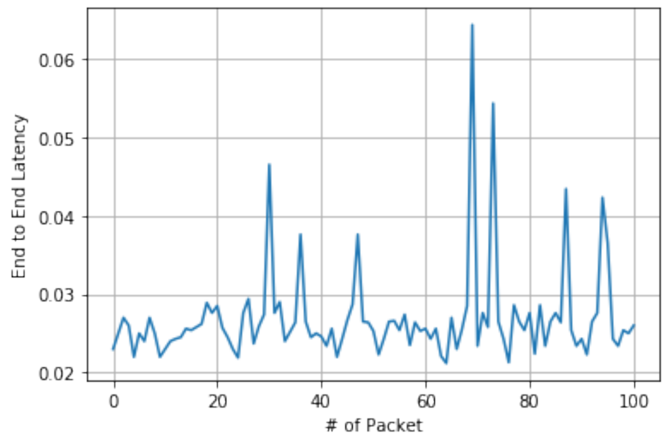
```
# Part1 - packet sending
# Initialize latency, drop and err
latency = []
dp = 0
rc = 0
loss = []
# Main loop
if i <= 100: # Main condition: if # of package is greater than 100, close socket
and go to the plot function
  lst = randint(256, size=256)
  msg = ''
  for n in lst:
    ib = str(bin(n)[2:])
    msg += ib.zfill(0)
  pl = msg2bt(msg)
  header = [0, 1, 1, 1, 1, 1, 1, 0] # header of each frame
  cnt_b = str(bin(i+1))
  counter = msg2bt(cnt_b[2:].zfill(0))
  div = [1, 1, 0, 0, 1] # use the divisor of x^4 + x^3 + 1
  CRC = getCRC(pl, div) # get CRC
  # print(CRC) # test whether the get_CRC function works at this point
  frame = header + counter + pl + CRC
  frame = ''.join(str(e) for e in frame)
  frame = '%d' % int(frame, 2)
  # print(frame)
  print('............' + str(i+1) + '............')
  sent = sock.sendto(frame.encode(), server_address)
  ts1 = time.time()
  print('package sent at: ' + str(ts1) + 'us')
  data, server = sock.recvfrom(1024)
  ts2 = time.time()
  print('package recevied at: ' + str(ts2) + 'us')

  dat = str(data)
  hex_str = dat[2:-1]
  hex_int = int(hex_str, 16)
  pl_r = bin(hex_int)
  pl_r = msg2bt(pl_r[2:])
  pl_r = pl_r[15:-1]
  CRC_check = getCRC(pl_r, div) # define CRC check parameter

  print('Transmitted CRC: ' + str(CRC))
  print('Recieved CRC: ' + str(CRC_check))

  latency.append(ts2 - ts1) # Calculate latency
  # check error:
  if CRC_check == CRC:
    rc += 1
  else: # ELSE: redo the package sending procedure
    dp += 1 # calculate err if check failed
    print('CRC Error!')
    print('............' + str(i+1) + '............')
    sent = sock.sendto(frame.encode(), server_address)
    ts1 = time.time()
    print('package resent at: ' + str(ts1) + 'us')
    data, server = sock.recvfrom(1024)
    ts2 = time.time()
    print('resent package recevied at: ' + str(ts2) + 'us')

    dat = str(data)
    hex_str = dat[2:-1]
    hex_int = int(hex_str, 16)
    pl_r = bin(hex_int)
    pl_r = msg2bt(pl_r[2:])
    pl_r = pl_r[15:-1]
    CRC_check = getCRC(pl_r, div)
    latency[-1] = ts2 - ts1
    if CRC_check == CRC:
      rc += 1
    else:
      dp += 1

    print('Transmitted CRC: ' + str(CRC))
    print('Recieved CRC: ' + str(CRC_check))

  loss.append(dp/(rc + dp) * 100)
else:
  sock.close()
  print('Socket closed')
  print('Final Error rate is: ' + str(loss[-1]))
```

Part that plot the latency and drop rate:

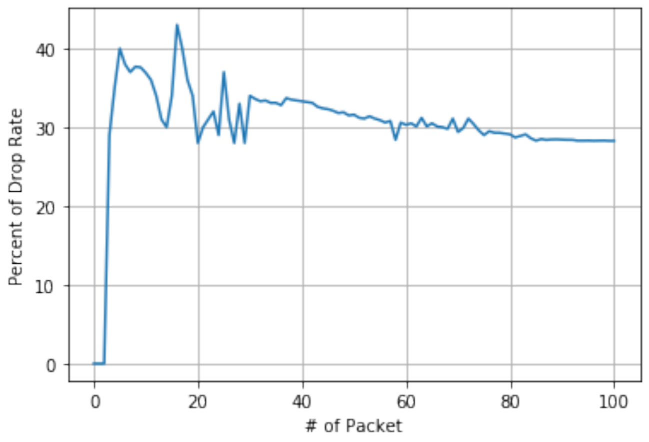
```
# Part 1 - Plotting
# End to End Latency
plt.figure(1)
plt.grid(1)
plt.plot(latency)
plt.xlabel('# of Packet')
plt.ylabel('End to End Latency')
# Drop Rate
plt.figure(2)
plt.grid(1)
plt.plot(loss)
plt.xlabel('# of Packet')
plt.ylabel('Percent of Drop Rate')
```

And the output is as below:

Ene to End Latency:



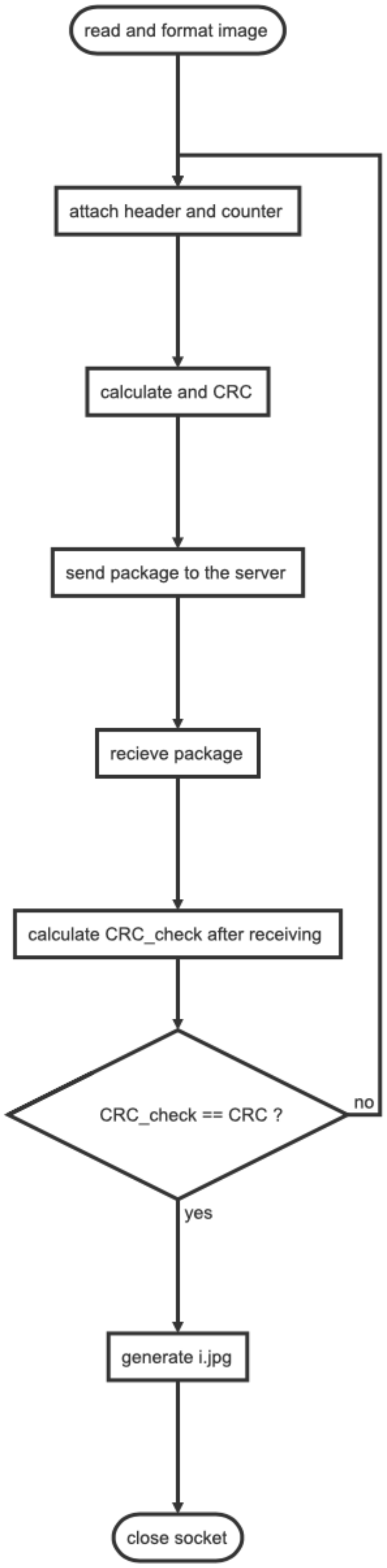
Error Rate:



Part 2 Image Parsing and Sending

All the same with Part 1, in this part though, an image is required to be read in the client program in a binary format and then pack the bit stream of the file into I-frames with sizes of 256 Bytes payload. When is precedure is done, a image file "1.jpg" is created in the project directory.

The flow chart for this part is shown below:



The frame configuration shows the structure of the package as below:

Header	Counter	Message	CRC
1 Byte	1 Byte	256 Bytes	2 Byte

And the source code with the output are shown below:

Read the image and format the image:

```
with open('/umdlogo.jpg','rb') as f:
    buff=f.read()
    msg=['{:08b}'.format(b) for b in buff] # formatting logo as 1 byte
    bt=msg2bt(msg)
    N,cnt,rdata=1024,0,bytes([]) # bit stream size of 1024
    Nf=int(np.ceil(len(bt)/N)) # noise figure
```

Several functions are defined for this part so that they can be applied directly in the main body.

Function that makes the package's form to be represented in bt and int:

```
def msg2bt(msg):
    bt=[]
    for i in range(len(msg)):
        for b in msg[i]:
            bt.append(int(b))
    return bt

def bt2int(bt):
    w=2**np.array(range(8))[::-1]
    return np.dot(bt,w)
```

Function that calculates the CRC of the frame: input a divisor (in this situation the divisor is [1,1,0,0,1]) and a message, and the function returns the CRC.

```
def getCRC(bt,div):
    N = len(div) # divisor (length)
    tmp = bt[0:N] # first N bits of the bitstream
    K = len(bt) # message (beatstream length)
    for i in range(K-N+1):
        if tmp[0]==1:
            tmp=[i^j for i,j in zip(div,tmp)]
        if i!=K-N:
            tmp = tmp[1:]+[bt[i+N]]
    return tmp[1:]
```

The server address is given by

```
# server address
sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
server_address=('199.223.115.24',9997)
```

And then the main body that is surpported by the flow chart:

The following part is served as the main body of the programming of this part, forming the frame by adding header, counter, payload and CRC, send and receive the package and then calculate the corresponding CRC just to check the accuracy of the procedure.

```
# main body
while True:
    # form the frame by adding header, counter, payload and CRC
    frame = list([0,1,1,1,1,1,1,0]) # header

    cntArr = [int(b) for b in '{:08b}' .format(cnt)]
    frame += cntArr

    txData = bt[1024*cnt:1024*(cnt+1)] # make transmitted data size 1024bit
    frame += txData # adding txData to the frame

    crc = getCRC(frame,[1,1,0,0,1]) # perform CRC check
    frame += crc # adding CRC to the frame

    frame +=[0,0,0,0]

    frame=np.array(frame).reshape(int(len(frame)/8),8) # creating frame to be sent
    packet=[bt2int(b) for b in frame] # creating packet to be sent
    sent=sock.sendto(bytes(packet),server_address) # sending the packet to the
server address
    data,server=sock.recvfrom(2048)
    rev=['{:08b}'.format(b) for b in data][:-1]
    rev=msg2bt(rev)

    # calculate CRC after receiving the packet
    rcrc = getCRC(rev, [1,1,0,0,1]) # divisor x^4 + x^3 + 1
    # check if the CRC of the recieved packet == CRC of the package sent
    if (rcrc==crc):
        print('Correct! %d packet' %cnt)
        cnt+=1
        rdata+=data[2:-1]
    else:
        print('Incorrect ! %d packet and resent it' %cnt)
        print(rdata==buff[:len(rdata)]) # print the data

    if cnt==Nf:
        print('Done')
        break
```

The output of the function is shown by

```
with open('1.jpg','wb') as f:
    f.write(rdata)
```

Main Issues that I have met

1. Issue and Solution: As I was not familiar with python language quite well, when I built my program in parts, the error like operator type always occurred. Operators like "<<", ">>" were not properly used. The way to solve these sort of problems is to search stack overflow as well as learn the data structure in detail in a tutorial and then get back and solve the errors.
2. In first few versions of my program, CRC_check always appeared wrong values so that it makes the condition to be always wrong and then keep resending the package for good. The reason for this is the disorder of the main loop that I have to do the reording of the parameters.
3. Length of the packet & CRC and etc. has confused me for several times. So as the calculation of the latency when it is complicated for me todo the operation as well as the order issues.

What I have learned from the project

This project has really been a journey to me ever since I've attended the grad school and the reason for which is that this has been the very first python programming for me, as I did not even touched this language before. Lots of debugging were being conducted and lots of work were being done just to organize the logic of the project well.

The very first skill that I think I've improved is definitely the programming skill. After finishing all this, I started to look back and found that the programming is literally not as plenty and complicated as I might think in the beginning. But afterall, I did learned this language from nothing especially in the communication domain, really appreciate for this.

The next thing I learned is the knowledge of computer networks in practical. How cool it is that I get to touch this field and turn it into something that I can really be doing.

Last, but not intensively, where I think I've learned something, is that I get to edit this report through Markdown. Learning to creat flowchart and so on could definitely be of vital help in my future study as well as research.