Computer Networks
CIE 447

Course Project Initial Report
Spring 2023

Done By Team 9:

Amr Ahmed Elmasry	201901202	Ahmed Ibrahem	201902227
Youssef Elshabrawy	201900667	Eman Allam	201900903

Introduction:

UDP is a simple, unreliable transport protocol that enables applications to send data packets to other devices on a network without establishing a connection. However, to add reliability to this protocol, Go-Back-N algorithm will be used. Go-Back-N is a protocol used for error control, where the sender retransmits the whole window packets if it receives a negative acknowledgement for any of them. There are three sources of errors: errors in ACK, lost ACK, and out-of-order ACK. To address the issue of out-of-order ACK, the sender retransmits the entire ordered window once a wrong ACK is received while In the case of lost ACK, a timeout is waited for and then the entire window is retransmitted. When there is an error in the ACK, the entire window is retransmitted immediately. This report will examine our implementation of segmenting an image, transmitting it, and managing the receiver.

1) Sender Implementation

1.1) Initialization

```
from binascii import hexlify, unhexlify
import time
from Decapsulation import deSegment_ack, deSegment

#initializations
file_name = 'SmallFile.png'
Ip_address = ''
port_no = 5555
MSS = 1008 #maximum segment size
N = 5 #window size of Go-Back-N
time_out = 1 #time out value in seconds
ID_packet = 0
ID_file = 0
data_size = MSS - 8 #size of the payload in bytes
#byte ya3ny rkamen hexa
no_of_hexa = data_size * 2 #length of the payload in hexa
trailer = b'0' * 8
```

In this part, we import libraries and functions and also initialize some important variables for later use in the code such as filename, port number, MSS, window size, timeout, packet ID, file ID, etc..

1.2) Read and Format The Image

```
#reading the image in bytes and turning it into hexa digits
data = ''
raw_data = ''
with open(file=file_name, mode='rb') as image:
    raw_data = image.read()
    data = hexlify(raw_data)

print(f'data size: {len(data)}')
```

Then, we read the image in bytes and save it in “raw_data”, then we change its format to hexadecimal digits as bytes in the variable “data”.

1.3) Socket Creation

```
#From here the server part
from socket import *

#socket initialization
data_skt = socket(AF_INET, SOCK_DGRAM)
ack_skt = socket(AF_INET, SOCK_DGRAM)

server_IP = 'localhost' #directly hyakhod IP al machine
port_no = 5555

data_skt.bind((server_IP, port_no)) #declare port ll server

N = N * no_of_hexa #window size in hexa
```

Now, we initialize the sockets and declare the server IP and the port number. We also bind the server IP to that socket.

1.4) Image Selection By Receiver

```
print('Here we go')
message, sender_address = data_skt.recvfrom(4096) #unknown recv_socket

confirmation = 'Here you go!'
ack_skt.sendto(confirmation.encode(), sender_address)

message = message.decode()

base = 0 #base of the window
thres = 0 #first unsent packet
iterations = 0 #number of iterations (one iteration means finishing the
last_ack = -1 #last acknowledged packet
```

Inside a while loop, the sender receives the filename to be sent to the receiver. We also initialize the base of the window and threshold to zeros while keeping the last acknowledgement variable set to -1 as there is no received acknowledgment yet.

1.5) Sending Algorithm

```
for j in range(thres, base + N, no_of_hexa): #sending the rest of the window
    if j + no_of_hexa >= len(data):
        print('last')
        trailer = b'f' * 8
    print(f'segment no. {ID_packet}')
    segment_data = data[j : j + no_of_hexa]
    segment = hexlify((ID_packet % 2**16).to_bytes(2)) + hexlify(ID_file.to_bytes(2)) + segment_data + trailer
    #time.sleep(0.0001)
    data_skt.sendto(segment, sender_address)

    ID_packet += int(len(segment_data) / 2)
    iterations = ID_packet // 2**16 #the iteration number of the last sent packet

thres = base + N
```

Based on the chosen file to be sent to the receiver, we start a for loop to send all possible packets until the window is full or the data is over. We form a packet of size MSS bytes. The packet is composed of packet ID, file ID, data (payload), and trailer. We then send this packet and increment the packet ID to hold the next byte to be sent next time. After sending the window is over, we update the threshold to start from the next available packet to be sent.

1.6) Acknowledgement Reception and Decapsulation

```
ack = ack_skt.recv(4096) #receive acknowledgement

ID_ack , _ID_file = deSegment_ack(ack) #decapsulation of the acknowledgment

print(f'last ack {last_ack}, ack no. {ID_ack}')
```

In this part, we receive the acknowledgement and decapsulate it into acknowledgement ID and file ID.

1.7) Window Shift Based of Received Acknowledgement

```
lower = (base // 2)  % 2**16 #lower base of the window
upper = ((base + N) // 2) % 2**16 #upper base of the window
print(f'lower is {lower}, upper is {upper}')
if lower < upper:
    if ID_ack >= lower and ID_ack < upper: #keda al ack mazbota
        last_ack = ID_ack
        #update al base 3shan yb2a 2odam al last ack
        diff = ID_ack - lower #difference ya3ny mabenhom kam byte
        diff *= 2 #diff mabenhom kam hexa
        base += diff * no_of_hexa
        base += no_of_hexa
    elif ID_ack == last_ack:
        thres = base #3shan ab3at akher window tani
        ID_packet = ID_ack + int(len(segment_data) / 2) #synchronize the packet
    else:
        pass #discard the ack lw heya akbar mn al window aw as8ar mnha except l
else:
    if ID_ack >= lower and ID_ack > upper or ID_ack < lower and ID_ack < upper:
        last_ack = ID_ack
        #update al base 3shan yb2a 2odam al last ack
        if ID_ack < lower:
            ID_ack += 2**16 #3shan akhleha fe mkanha akbar mnha 3shab a7seb al

        diff = ID_ack - lower #difference ya3ny mabenhom kam byte
        diff *= 2 #diff mabenhom kam hexa
        base += diff * no_of_hexa
        base += no_of_hexa
    elif ID_ack == last_ack:
        thres = base #3shan ab3at akher window tani
        ID_packet = ID_ack + int(len(segment_data) / 2) #synchronize the packet
    else:
        pass #discard the ack lw heya akbar mn al window aw as8ar mnha except l
```

After the reception of the acknowledgement, we calculate the lower bound and the upper bound of the window and check if the received acknowledgement lies inside the window, before the window, or after the window. If inside the window, we set the last acknowledgement to this acknowledgment and change the base of the window to the subsequent packet ID. If the received acknowledgement ID is equal to the last acknowledgement then we resend the whole window (out of order at the receiver). Otherwise, we ignore the acknowledgement.

2)Receiver Implementation

2.1) Initialization and Socket Creation

```
from socket import *
from binascii import hexlify, unhexlify
from Decapsulation import deSegment
import random
import time

data_skt = socket(AF_INET, SOCK_DGRAM)
ack_skt = socket(AF_INET, SOCK_DGRAM)

#for sending
server_IP = 'localhost'
port_no = 1234 #choose from registered ports (1024, 49___)

ack_skt.bind((server_IP, port_no))
```

In this part, we import libraries and functions and also initialize the sockets, IP address, port number, and bind the IP address to that port number.

2.2) Choosing Image and Variables declaration

```
data = bytes()
message = input('input message: ')

data_skt.sendto(message.encode(), (server_IP, 5555))

confirmation, sender_address = data_skt.recvfrom(4096)

print(confirmation.decode())

expected = 0
last_correct = 0
```

In this part, we declare a “data” variable to hold the received data. We also ask for which image we want to receive. We initialize another 2 variables, “expected” variable to hold the expected packet ID, and the “last_correct” variable to hold the packet ID of the last correctly received packet.

2.3) Receiving Segments and out-of-order Simulation

```
segment = data_skt.recv(4096) #input buffer size (powers of 2)

ID_packet, ID_file, datum, trailer, length = deSegment(segment) #check trailer thing

if ID_packet < expected: #lw ack l haga adema afkslha
    continue

#some randomness to check out of order packets
rand = random.randint(0, 100)
if rand <= 20 and ID_packet != 0:
    print(f'error at: {ID_packet}')
    ID_packet += 100
```

We then start receiving segments, decapsulate the segment. If the received packet ID is less than the expected we ignore. We also add some sort of randomness to test the out of order reliability feature (packet ID higher than expected).

2.4) Receiving out-of-order Packet

```
if ID_packet != expected: #lw msh al expected ab3at al last correct ack
    ack = hexlify(last_correct.to_bytes(2)) + hexlify(ID_file.to_bytes(2))
    print(f'recieved {ID_packet}, but not as expected {expected}')
    ack_skt.sendto(ack, sender_address)
    continue
```

If the packet ID is not the expected packet ID, we send acknowledgement of the last correctly received packet.

2.5) Receiving in-order Packet

```
#lw expected
print(f'received ID: {ID_packet}')
last_correct = ID_packet
expected += int(length / 2)
expected %= 2**16

data += datum

#send ack
ack = hexlify(ID_packet.to_bytes(2)) + hexlify(ID_file.to_bytes(2))
ack_skt.sendto(ack, sender_address)
```

Otherwise, the received packet ID is the expected, therefore, we update the “last_correct” variable and increment the “expected” variable. Then, we add the received data to the overall received data. After that we send acknowledgement for the received packet.

2.6) Receiving Last Packet and Writing Image

```
    if trailer == b'f' * 8:
        break

with open('receiver3.png', 'wb') as f:
    f.write(data)
```

Finally, if the received segment has a trailer of ones, then the file is complete and we break from the receiving loop. Then, we write the data in an image.

3)Decapsulation

```python
from binascii import hexlify, unhexlify

def deSegment(segment):
    length = len(segment[8 : len(segment) - 8])
    ID_packet = unhexlify(segment[0 : 4])
    ID_file = unhexlify(segment[4 : 8])
    data = unhexlify(segment[8 : len(segment) - 8])
    trailer = segment[-8:]

    return (int.from_bytes(ID_packet), int.from_bytes(ID_file), data, trailer, length)

def deSegment_ack(ack):
    ID_packet = unhexlify(ack[0 : 4])
    ID_file = unhexlify(ack[4 : 8])

    return (int.from_bytes(ID_packet), int.from_bytes(ID_file))
```

These are two functions that we wrote to make the code easier. Their function is to decapsulate the data segments and the acknowledgement segment.