Name: Sayan Acharya

Roll: 002010501009

Group: A1

Assignment: 7

Deadline: 1st - 4th November 2022

Submission date: 4th November

Code:(ARP)

```
import socket,threading
MAX_DATA_SiZE=4096

print_lock=threading.Lock()

def _print(*args,**kwargs):
  print_lock.acquire()
  print(*args,**kwargs)
  print_lock.release()

class arp_entry:
  def __init__(self,mac_id,entry_type,max_time):
    self.mac_id=mac_id
    self.entry_type=entry_type
    self.max_time=max_time

class Device:
  def __init__(self,server_addr,reply_server_addr,mac_id,ip_addr=-1,):

    self.server_addr=server_addr #real ip addr/port
    self.reply_server_addr=reply_server_addr

    self.sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)# udp socket
    self.sock.bind(self.server_addr)

    self.reply_sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)# udp socket
    self.reply_sock.bind(self.reply_server_addr)

    self.mac_id=mac_id
    self.ip_addr=ip_addr #pretend ip addr
    self.arp_table=dict()
    self.connec_devices=[]
    self.active_threads=[]

  def add_arp_entry(self,ip_addr,arp_entry_val):
    self.arp_table[ip_addr]=arp_entry_val

  def add_device(self,device_server_addr):
    self.connec_devices.append(device_server_addr)

  def start_device(self):
    t=threading.Thread(target=self.listen,args=[])
    self.active_threads.append(t)
```

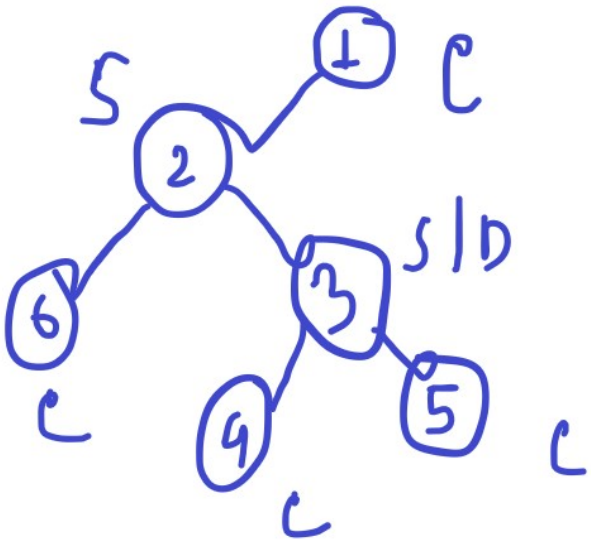
```
    t.start()

  def listen(self):
    while True:#ip,reply_sock_ip,reply_sock_port
      data,parent_addr=self.sock.recvfrom(MAX_DATA_SiZE)
      data=str(data,encoding='utf-8').split(',')
      parent_reply_sock=(data[1],int(data[2]))


      #send data to all devices except the parent device
      for device in self.connec_devices:
        if device == parent_addr:
          continue

self.sock.sendto(f"{data[0]},{self.reply_server_addr[0]},{self.reply_server_addr[1]}".encode('utf-8'
),device)


      ans=None

      #get the output of the data sent. format: found,mac_id

      #leaf case

      if self.ip_addr==data[0]:
        ans=f"{self.mac_id}"
      else:
        #non leaf node case
        for i in range(len(self.connec_devices)-1):
          ret_data,addr=self.reply_sock.recvfrom(MAX_DATA_SiZE)
          string=str(ret_data,encoding='utf-8')
          split_string=string.split(',')
          if int(split_string[0])==1:
            ans=split_string[1]

      if ans:
        what_to_send=f"1,{ans}"
      else:
        what_to_send="0,Nothing"

      #send the ans back to parent
      self.sock.sendto(what_to_send.encode('utf-8'),parent_reply_sock)

  def query_ip(self,q_ip):
```

```
    if q_ip not in self.arp_table:

      for i in self.connec_devices:
        _print(f"trying device: {i}")

self.sock.sendto(f"{q_ip},{self.reply_server_addr[0]},{self.reply_server_addr[1]}".encode('utf-8'),i)
        ans,addr=self.reply_sock.recvfrom(MAX_DATA_SiZE)
        ans=str(ans,encoding='utf-8').split(',')

        if int(ans[0])==1:
          self.arp_table[q_ip]=arp_entry(ans[1],'dynamic',-1)
          break
      if not ans or int(ans[0])==0:
        _print(f"not found any mac_id for ip:{q_ip}")
      else:
        _print(f"found. mac_id:ip found <-> {ans[1]},{q_ip}")
    else:
      _print(f"ip already in table. mac_id:ip found <-> {self.arp_table[q_ip].mac_id},{q_ip}")
```

main.py:

```
from helper import Device
from typing import List
loop_back_addr="127.0.0.0"

all_devices:List[Device]=[]
all_clients=[]

with open('info.txt','r') as f:
  lines=[i.strip('\r\n').strip('\n') for i in f.readlines()]
  def get_input():
    index=0
    while index<len(lines):
      yield lines[index]
      index+=1
    return
  iter_obj=get_input()
  n=int(next(iter_obj))
  for i in range(n):
    output=next(iter_obj).split()
    print(output)
    dev=Device((output[0],int(output[1])),((output[2],int(output[3]))),output[4],output[5])
```

```
    dev.start_device()
    all_devices.append(dev)
    if output[4]=='C':
      all_clients.append(dev)
  m=int(next(iter_obj))
  for i in range(m):
    a,b=next(iter_obj).split()
    a=int(a)
    b=int(b)
    all_devices[a-1].add_device(all_devices[b-1].server_addr)
    all_devices[b-1].add_device(all_devices[a-1].server_addr)

while True:
  inp=input() # device_index,ip_addr
  if inp.upper()=="EXIT":
    break
  inp=inp.split()
  all_devices[int(inp[0])-1].query_ip(inp[1])
```




```
D:\NetworkLab\Assignment7\ARP>python main.py
['127.0.0.1', '8001', '127.0.0.1', '9001', '01:00:5E:00:00:01', '193.0.0.1', 'C']
['127.0.0.1', '8002', '127.0.0.1', '9002', '01:00:5E:00:00:02', '193.0.0.2', 'S']
['127.0.0.1', '8003', '127.0.0.1', '9003', '01:00:5E:00:00:03', '193.0.0.3', 'S']
['127.0.0.1', '8004', '127.0.0.1', '9004', '01:00:5E:00:00:04', '193.0.0.4', 'C']
['127.0.0.1', '8005', '127.0.0.1', '9005', '01:00:5E:00:00:05', '193.0.0.5', 'C']
['127.0.0.1', '8006', '127.0.0.1', '9006', '01:00:5E:00:00:06', '193.0.0.6', 'C']
1 193.0.0.4
trying device: ('127.0.0.1', 8002)
found. mac_id:ip found <-> 01:00:5E:00:00:04,193.0.0.4
```

Code:(DHCP)

```
import socket,threading,time

MAX_PACKET_SIZE=4096
print_lock=threading.Lock()

def _print(*args,**kwargs):
  print_lock.acquire()
  print(*args,**kwargs)
  print_lock.release()


class Packet:
  keys=[
    'src_ip',
    'src_port',
    'dest_ip',
    'dest_port',
    'pid',
    'data',
  ]
  def __init__(self,
               src_ip,src_port,
               dest_ip,dest_port,
               pid,data):
    self.src_ip=src_ip
    self.src_port=src_port
    self.dest_ip=dest_ip
    self.dest_port=dest_port
    self.pid=pid
    self.data=data

    self.mappa=dict(zip(Packet.keys,[
      src_ip,src_port,
      dest_ip,dest_port,
      pid,data,
    ]))

  def __getitem__(self,key_val):
    return self.mappa[key_val]

  def encode(self,format='utf-8'):
    string=f"{self.src_ip} {self.src_port} {self.dest_ip} {self.dest_port} {self.pid} {self.data}"
```

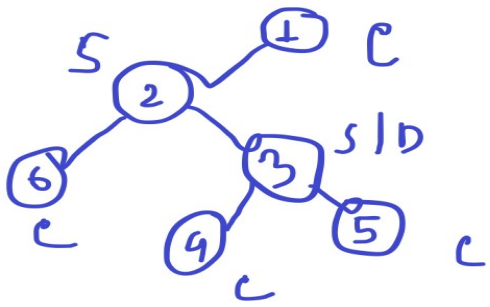
```
    return string.encode(format)

  @classmethod
  def decode(cls,string:str,format='utf-8'):
    parts=str(string,encoding=format).split()
    return Packet(parts[0],int(parts[1]),parts[2],int(parts[3]),parts[4],' '.join(parts[5::]))


class Device:
  def __init__(self,ip,port,is_dhcp=False):
    self.ip=ip
    self.port=port
    self.sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
    self.sock.bind((ip,port))
    self.is_dhcp=is_dhcp
    if self.is_dhcp:
      self.dhcp_mutex=threading.Lock()
      self.dhcp_table=dict()

    self.dhcp_ip='?'
    self.dhcp_port=0
    self.active_dhcp_request=False

    self.routing_table=dict()
    self.connected_devices=[]
    self.ip_count=0
    self.pid=0
    self.pid_table=dict()
    self.assigned_ip='?'

    self.threads=[]

  def get_addr(self):
    return (self.ip,self.port)

  def add_device(self,ip,port):
    self.connected_devices.append((ip,port))

  def start_device(self):
    t=threading.Thread(target=self.listen,args=[])
    t.setDaemon(True)
    t.start()
    self.threads.append(t)
```

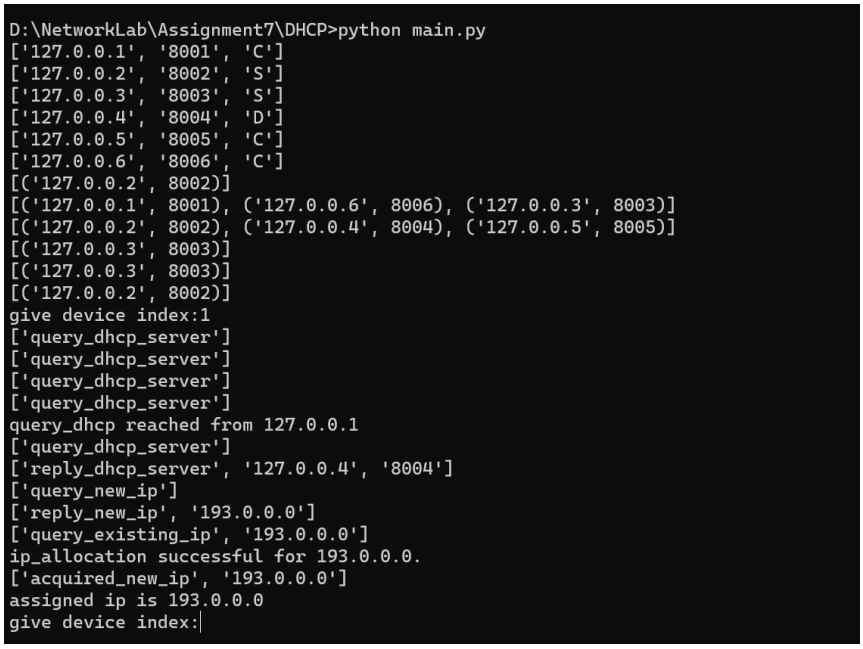
```
def listen(self):
  while True:
    packet,addr=self.sock.recvfrom(MAX_PACKET_SIZE)
    packet=Packet.decode(packet)
    t=threading.Thread(target=self.handle_packet,args=[packet,addr])
    t.setDaemon(True)
    t.start()
    self.threads.append(t)

def add_to_routing_table(self,packet,addr):#(ip)->(ip,port)
  tup=(packet['src_ip'])
  if tup not in self.routing_table:
    self.routing_table[tup]=addr

def give_new_ip(self):
  assert(self.is_dhcp)

  self.dhcp_mutex.acquire()

  assigned_ip=self.ip_count
  self.ip_count+=1
  ip= f"193.0.0.{assigned_ip}"

  self.dhcp_mutex.release()

  return ip

def get_new_ip(self):
  t_start=time.time()
  while time.time()-t_start<=10:
    if self.assigned_ip=='?' and not self.active_dhcp_request:
      self.active_dhcp_request=True
      self.send_packet(Packet(self.ip,self.port,'?',0,0,f"query_dhcp_server"),('?',0))

  if self.assigned_ip=='?':
    _print("dhcp server finding or ip allocation failed!")
  else:
    _print(f"assigned ip is {self.assigned_ip}")


def handle_packet(self,packet:Packet,addr):#this addr is previous socket
  self.add_to_routing_table(packet,addr)
  data=packet['data']
```

```
    if self.is_dhcp:
      if packet.dest_ip=='?':
        if data=="query_dhcp_server":
          _print(f"query_dhcp reached from {packet['src_ip']}")

self.send_packet(Packet(self.ip,self.port,packet['src_ip'],packet['src_port'],packet['pid'],f"reply_dh
cp_server {self.ip} {self.port}"),(packet['src_ip'],packet['src_port']))

    data=data.split()

    if packet.dest_ip==self.ip or packet.dest_ip=='?': #reached destination
      _print(data)
      if data[0]=="reply_dhcp_server":
        self.dhcp_ip=packet['src_ip']
        self.dhcp_port=packet['src_port']


self.send_packet(Packet(self.ip,self.port,self.dhcp_ip,self.dhcp_port,0,f"query_new_ip"),(self.dh
cp_ip,self.dhcp_port))

      elif data[0]=="query_new_ip":

self.send_packet(Packet(self.ip,self.port,packet['src_ip'],packet['src_port'],packet['pid'],f"reply_ne
w_ip {self.give_new_ip()}"),(packet['src_ip'],packet['src_port']))

      elif data[0]=="reply_new_ip":
        task_pid=self.pid
        self.pid+=1
        self.assigned_ip=data[1]

        #send an arp to see if exists
        self.send_packet(Packet(self.ip,self.port,'?',8000,task_pid,f"query_existing_ip
{data[1]}"),('?',8000))
        self.pid_table[task_pid]='?'

        t_start=time.time()
        is_failed=False
        while time.time()-t_start<=5:
          if self.pid_table[task_pid]!='?' and self.pid_table[task_pid]=='1':
            #got the ans_to_query as invalid_ip
            self.assigned_ip='?'#the assigned ip
            is_failed=True
            break
          time.sleep(0.5)
```

```
        if not is_failed:
          _print(f"ip_allocation successful for {data[1]}.")
          self.send_packet(Packet(self.ip,self.port,self.dhcp_ip,self.dhcp_port,0,f"acquired_new_ip
{self.assigned_ip}"),(self.dhcp_ip,self.dhcp_port))
        else:
          _print(f"ip_allocation failed for {data[1]}.")
          self.active_dhcp_request=False

      elif data[0]=="query_existing_ip":
        if data[1]==self.assigned_ip:

self.send_packet(Packet(self.ip,self.port,packet['src_ip'],packet['src_port'],packet['pid'],"reply_exi
sting_ip 1"))

      elif data[0]=="reply_existing_ip":
        self.pid_table[packet['pid']]=data[1]

      elif data[0]=='acquired_new_ip':
        self.dhcp_table[data[1]]=-1 # time limit
        self.active_dhcp_request=False

      elif packet.dest_ip!=self.ip:#on route
        self.send_packet(packet,(packet['dest_ip'],packet['dest_port']),set([addr]))

    elif packet.dest_ip!=self.ip:#on route
      self.send_packet(packet,(packet['dest_ip'],packet['dest_port']),set([addr]))

  def send_packet(self,packet,addr,exclude_list=set()):
    if addr in self.routing_table:
      self.sock.sendto(packet.encode(),self.routing_table[addr[0]])
    else:
      for dev in self.connected_devices:
        if dev not in exclude_list:
          self.sock.sendto(packet.encode(),dev)

main.py:
          from DHCP import Device

all_devices=[]
all_clients=[]
dhcp_server=None

with open('info.txt','r') as f:
```

```
  lines=[i.strip('\r\n').strip('\n') for i in f.readlines()]
  def get_input():
    index=0
    while index<len(lines):
      yield lines[index]
      index+=1
    return
  iter_obj=get_input()
  n=int(next(iter_obj))
  for i in range(n):
    output=next(iter_obj).split()
    print(output)
    dev=Device(output[0],int(output[1]),is_dhcp=(output[-1]=='D'))
    dev.start_device()
    all_devices.append(dev)
    if output[-1]=='C':
      all_clients.append(dev)
  m=int(next(iter_obj))
  for i in range(m):
    a,b=next(iter_obj).split()
    a=int(a)
    b=int(b)
    all_devices[a-1].add_device(*all_devices[b-1].get_addr())
    all_devices[b-1].add_device(*all_devices[a-1].get_addr())

for i in range(n):
  print(all_devices[i].connected_devices)

while True:
  inp=input("give device index:") # device_index
  if inp.upper()=="EXIT":
    break
  all_devices[int(inp)-1].get_new_ip()
print("ending loop")
```





Discussion:

ARP: In this method, the computer pings the whole network to find out if there is a device with the given ip. If there is an ip, then it returns the mac_id of the device and then that is stored in the querying pc's table

DHCP: In this method, the computer pings to find the DHCP server. If present, then the DHCP server is further queried to get a new ip. Then the querying machine sends an ARP to find out if the ip exists in the network already. If the ip does not exist, then the machine accepts the ip and sends ip acceptance request to the DHCP.

Comment:

With this assignment, I got to know a lot about the various protocols used in the Internet