MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY

DEPARTMENT OF ICT

Lab Report No : 07

Course Code : ICT-3208

Course Title : Network Planning and Designing Lab

Lab Report Name : Python for Networking Lab

Submitted by	***Submitted to***
Name: Md. Shoriful Islam Sakib ID : IT-17043 Session : 2016-2017 3rd Year 2nd Semester	Nazrul Islam Assistant Professor, Department of ICT,MBSTU Santosh,Tangail-1902

Python for Networking

Objectives :

- Install python and use third-party libraries
- Interact with network interfaces using python
- Getting information from internet using Python

1. Describe some networking common terms ?

i. Packet

v. Protocol ii. Network Interface

vi. Firewall iii. LAN

vii. NAT iv. WAN

viii. VPN

Ans:

i**.Packet:** A packet is, generally speaking, the most basic unit that is transfered over a network. When communicating over a network, packets are the envelopes that carry your data (in pieces) from one end point to the other. Packets have a header portion that contains information about the packet including the source and destination, timestamps, network hops, etc. The main portion of a packet contains the actual data being transfered. It is sometimes called the body or the payload.

ii.**Network Interface :** A network interface can refer to any kind of software interface to networking hardware. For instance, if you have two network cards in your computer, you can control and configure each network interface associated with them individually.A network interface may be associated with a physical device, or it may be a representation of a virtual

interface. The "loopback" device, which is a virtual interface to the local machine, is an example of this.

iii.**LAN :** LAN stands for "local area network". It refers to a network or a portion of a network that is not publicly accessible to the greater internet. A home or office network is an example of a LAN.

iv.**WAN :** WAN stands for "wide area network". It means a network that is much more extensive than a LAN. While WAN is the relevant term to use to describe large, dispersed networks in general, it is usually meant to mean the internet, as a whole. If an interface is said to be connected to the WAN, it is generally assumed that it is reachable through the internet.

V. **Protocol :** A protocol is a set of rules and standards that basically define a language that devices can use to communicate. There are a great number of protocols in use extensively in networking, and they are often implemented in different layers. Some low level protocols are TCP, UDP, IP, and ICMP. Some familiar examples of application layer protocols, built on these lower protocols, are HTTP (for accessing web content), SSH, TLS/SSL, and FTP. Port: A port is an address on a single machine that can be tied to a specific piece of software. It is not a physical interface or location, but it allows your server to be able to communicate using more than one application.

vi. **Firewall :** A firewall is a program that decides whether traffic coming into a server or going out should be allowed. A firewall usually works by creating rules for which type of traffic is acceptable on which ports. Generally, firewalls block ports that are not used by a specific application on a server.

vii. **NAT :** NAT stands for network address translation. It is a way to translate requests that are incoming into a routing server to the relevant devices or servers that it knows about in the LAN. This is usually implemented in physical LANs as a way to route requests through one IP address to the necessary backend servers.

viii. **VPN :** VPN stands for virtual private network. It is a means of connecting separate LANs through the internet, while maintaining privacy. This is used as a means of connecting remote systems as if they were on a local network, often for security reasons.

2. Exercises :

Ex-1 : Enumerating interfaces on your machine.

Source code :

```
import     sys
import
socket  import
fcntl   import
struct  import
array

SIOCGIFCONF  =  0x8912  #from  C  library  sockios.h
STUCT_SIZE_32 = 32
STUCT_SIZE_64 = 40

PLATFORM_32_MAX_NUMBER =   2**32
DEFAULT_INTERFACES = 8
 def
list_interfaces():
interfaces  =  []
    max_interfaces  =  DEFAULT_INTERFACES
    is_64bits  =  sys.maxsize  >  PLATFORM_32_MAX_NUMBER
struct_size  =  STUCT_SIZE_64  if  is_64bits  else
STUCT_SIZE_32     sock  =  socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)     while  True:
```

```
        bytes  =  max_interfaces  *  struct_size
interface_names  =  array.array('B',  '\0'  *  bytes)
sock_info  =
fcntl.ioctl(sock.fileno(),SIOCGIFCONF,struct.pack('iL', bytes,
interface_names.buffer_info()[0])  )
        outbytes  =  struct.unpack('iL',
sock_info)[0]          if  outbytes  ==  bytes:
max_interfaces  *=  2          else:
break
        namestr  =  interface_names.tostring()
for  i  in  range(0,  outbytes,  struct_size):
            interfaces.append((namestr[i:i+16].split('\0',  1)
[0]))          return  interfaces if     name    ==  '   main
':     interfaces  =  list_interfaces()     print  ("This
machine    has    %s network    interfaces:    %s."%
(len(interfaces), interfaces))
```

Output :

```
    import  fcntl
ModuleNotFoundError: No module named 'fcntl'
```

Ex-2 : Finding the IP address for a specific interface on your machine.

Source code :

```
import
argparse
import      sys
import   socket
import    fcntl
import   struct
import array
  def  get_ip_address(ifname):     s  =
socket.socket(socket.AF_INET,  socket.SOCK_DGRAM)
return  socket.inet_ntoa(fcntl.ioctl(s.fileno(),
0x8915,     struct.pack('256s',  ifname[:15]))[20:24])
    if     __name__=='__main__':                    parser        =
argparse.ArgumentParser(description='Python  networking  utils')
    parser.add_argument('--ifname',
action="store",  dest="ifname",required=False)
given_args  =  parser.parse_args()     ifname  =
given_args.ifname
    print  ("Interface  [%s]  -->  IP:  %s"  %(ifname,
get_ip_address(ifname)))
```

Output:

```
    import  fcntl
ModuleNotFoundError: No module named 'fcntl'
```

Ex-3 : Finding whether an interface is up on your machine.

Source code :

```
import  argparse
import    socket
import    struct
import     fcntl
import nmap

SAMPLE_PORTS = '21-23'
 def  get_interface_status(ifname):     sock  =
socket.socket(socket.AF_INET,  socket.SOCK_DGRAM)
ip_address  =
socket.inet_ntoa(fcntl.ioctl(sock.fileno(),0x8915,
struct.pack('256s',  ifname[:15]))[20:24]) nm  =
nmap.PortScanner() nm.scan(ip_address,  SAMPLE_PORTS)
return  nm[ip_address].state()
  if       name       ==    '     main     ':         parser    =
argparse.ArgumentParser(description='Python  networking  utils')
    parser.add_argument('--ifname',  action="store",
dest="ifname",     required=True)
    given_args  =  parser.parse_args()
ifname  =  given_args.ifname
    print  ("Interface  [%s]  is:  %s"  %(ifname,
get_interface_status(ifname)))
```

Output :

```
    import  fcntl
ModuleNotFoundError: No module named 'fcntl'
```

Ex-4 : Detecting inactive machines on your network .

Source code:

```
import argparse import
time    import    sched
from
scapy.layers.inet
```

```
import sr, srp, IP, UDP, ICMP, TCP, ARP, Ether
#from scapy.all
# import sr, srp, IP, UDP, ICMP, TCP, ARP, Ether

RUN_FREQUENCY = 10

scheduler = sched.scheduler(time.time, time.sleep)
  def
detect_inactive_hosts(scan_hosts):

    global scheduler
    scheduler.enter(RUN_FREQUENCY, 1, detect_inactive_hosts,
(scan_hosts, ))       inactive_hosts = []       try:
ans, unans = sr(IP(dst=scan_hosts) / ICMP(), retry=0,
timeout=1)          ans.summary(lambda (s, r):
                        r.sprintf("%IP.src% is
alive"))              for inactive in unans:
print("%s     is     inactive" % inactive.dst)
inactive_hosts.append(inactive.dst)

        print("Total %d hosts are inactive" %
(len(inactive_hosts)))

        except KeyboardInterrupt: exit(0)
 if
__name__=='__main__':
    parser = argparse.ArgumentParser(description='Python
networking utils')
    parser.add_argument('--scan-hosts', action="store",
dest="scan_hosts",required=True)

given_args = parser.parse_args()
scan_hosts = given_args.scan_hosts
scheduler.enter(1, 1, detect_inactive_hosts, (scan_hosts,))
scheduler.run()
```

Ex-5 : Pinging hosts on the network

with ICMP Source code :

```
import      os
import
argparse
import   socket
import   struct
```

```
import   select
import  time

ICMP_ECHO_REQUEST = 8 # Platform specific DEFAULT_TIMEOUT = 2
DEFAULT_COUNT = 4
 class
Pinger(object):
    def     init    (self,  target_host,
count=DEFAULT_COUNT, timeout=DEFAULT_TIMEOUT):
self.target_host  =  target_host self.count  =  count
self.timeout  =  timeout
        def         do_checksum(self,
source_string):     sum = 0
      max_count    =    (len(source_string)/2)*2
count  =  0          while  count  <  max_count:
val  =  ord(source_string[count  +  1])  *  256  +
ord(source_string[count])       sum = sum + val
sum = sum & 0xffffffff     count = count + 2
     if max_count < len(source_string):          sum = sum
+ ord(source_string[len(source_string) - 1])          sum
= sum & 0xffffffff

    sum = (sum >> 16) + (sum &
0xffff)     sum = sum + (sum >> 16)
answer = ~sum     answer = answer &
0xffff
    answer = answer >> 8 | (answer << 8 & 0xff00)
return answer
     def receive_pong(self, sock, ID,
timeout):
        """
        Receive  ping  from  the  socket. """

    time_remaining = timeout
while True:          start_time
= time.time()
          readable   =   select.select([sock],   [],   [],
time_remaining)          time_spent  =  (time.time()  -
start_time)     if readable[0] == []:  # Timeout return

    time_received = time.time()
recv_packet, addr = sock.recvfrom(1024)
icmp_header = recv_packet[20:28]
    type, code, checksum, packet_ID, sequence = struct.unpack(
        "bbHHh", icmp_header
    )
    if packet_ID == ID:          bytes_In_double =
struct.calcsize("d")     time_sent =
```

```
struct.unpack("d", recv_packet[28:28 +
bytes_In_double])[0]
    return time_received - time_sent
     time_remaining = time_remaining -
time_spent     if time_remaining <= 0:
return
     def send_ping(self, sock,
ID):
        """
        Send  ping  to  the  target  host """

    target_addr = socket.gethostbyname(self.target_host)

    my_checksum = 0


ID, 1)

#  Create  a  dummy  heder  with  a  0  checksum. header =
struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, my_checksum,
 bytes_In_double = struct.calcsize("d")
data = (192 - bytes_In_double) * "Q"
data = struct.pack("d", time.time()) +
data

#  Get  the  checksum  on  the  data  and  the  dummy  header.
my_checksum  =  self.do_checksum(header  +  data)  header  =
struct.pack(
"bbHHh", ICMP_ECHO_REQUEST, 0, socket.htons(my_checksum), ID,
1
)
packet = header + data
sock.sendto(packet, (target_addr, 1))
  def
ping_once(self):
    """
    Returns  the  delay  (in  seconds)  or  none  on  timeout.
"""
  icmp =
socket.getprotobyname("icmp")
try:
    sock = socket.socket(socket.AF_INET, socket.SOCK_RAW,
icmp)      except  socket.error, (errno, msg):
if errno == 1:
# Not  superuser,  so  operation  not  permitted msg +=
"ICMP  messages  can  only  be  sent  from  root  user
processes
"
```

```
raise
socket.error(msg)
except   Exception,  e:
print
"Exception:  %s" % (e)

my_ID = os.getpid() & 0xFFFF

self.send_ping(sock, my_ID)
delay = self.receive_pong(sock, my_ID,
self.timeout) sock.close() return delay
  def
ping(self):
    """
    Run  the  ping  process """
  for i in
xrange(self.count):
print
     "Ping   to   %s..." % self.target_host,       try:
delay  =  self.ping_once()  except    socket.gaierror,  e:
print
"Ping   failed.   (socket   error:   '%s')" %
e[1] break  if delay == None:     print
    "Ping  failed.  (timeout  within  %ssec.)" % self.timeout
else:
delay  =  delay  *  1000
print  "Get  pong  in  %0.4fms"  %  delay

   if        name      ==    '     main     ':        parser    =
argparse.ArgumentParser(description='Python             ping')
parser.add_argument('--target-host',
action="store",dest="target_host",               required=True)
given_args    =    parser.parse_args()          target_host    =
given_args.target_host                          pinger         =
Pinger(target_host=target_host)     pinger.ping()
```

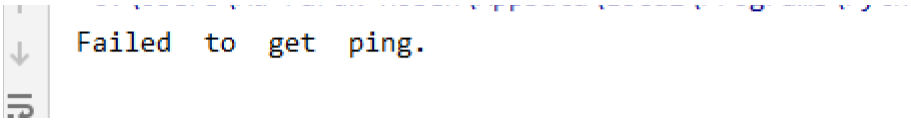
Ex-6 : Pinging hosts on the network with ICMP using pc resources Create

Source code :

```
import subprocess
import  shlex

command_line  =  "ping  -c  1  10.0.1.135"
 if  __name__=='__main__':      args
=          shlex.split(command_line)
try:
subprocess.check_call(args,stdout=subprocess.PIPE,stderr=
subprocess.PIP E)         print  ("Your  pc  is  up!")     except
subprocess.CalledProcessError:          print  ("Failed  to  get
ping.")
```

Output :



Ex-7 : Scanning the broadcast of packets Source code :

```
from  scapy  import  all
from  scapy.layers.inet
import  sr,  srp,  IP,  UDP,  ICMP,  TCP,  ARP,  Ether, sniff
captured_data = dict()
END_PORT = 1000
 def  monitor_packet(pkt):     if  IP  in  pkt:
if  not  captured_data.has_key(pkt[IP].src):
            captured_data[pkt[IP].src]  =  []
       if  TCP  in  pkt:             if
pkt[TCP].sport  <= END_PORT:
            if  not  str(pkt[TCP].sport)  in
captured_data[pkt[IP].src]:
captured_data[pkt[IP].src].append(str(pkt[TCP].sport))

    os.system('clear')
    ip_list  =  sorted(captured_data.keys())
for  key  in  ip_list:         ports=',
'.join(captured_data[key])         if  len
(captured_data[key])  ==  0:             print
('%s'  %  key)             else:
print  ('%s  (%s)'  %  (key,  ports))
```

```
      if       __name__=='__main__':
sniff(prn=monitor_packet,  store=0)
```

Ex-8 : Sniffing packets on your network

Ans: **Tcpdump** is a common packet analyzer that runs under the command line. It allows the user to display TCP/IP and other packets being transmitted or received over a network to which the computer is attached. Distributed under the BSD license,[3] tcpdump is free software.

- Open a linux terminal and check the usage of tcpdump using the command line tcpdum –help
- Using tcpdump get the traffic present in the Ethernet interface of your pc (10 packet only), which is the command line?
- Using the subprocess write a program for sniffing 1 packet of the Ethernet interface? (Save as packet_sniffer.py).

Ex-9 : Performing a basic

Telnet Source code :

```
import  socket TCP_IP
=  '127.0.0.1'
TCP_PORT = 62
BUFFER_SIZE  =  20 #  Normally  1024,  but  we  want  fast
response

s  =  socket.socket(socket.AF_INET,  socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)
conn, addr = s.accept()
print  ('Connection  address:',
addr) while  1:     data  =
conn.recv(BUFFER_SIZE)     if  not
data:
        break
    print  ("received  data:",  data) conn.send(data)  #  echo
conn.close()
```

Output :

```
Connecting to localhost port 1234
Sending Test message: SDN course examples
Received: b'Test message: '
Received: b'SDN course exa'
Received: b'mples'
Closing connection to the server
```

Conclusion :

From this lab, I learn a lot of python code that is related to networking. I also learn telnet. When I do this lab, I face some problem. But I overcome these with the help of my course teacher(Nazrul Islam).