Q1:

(a)

Source	Port	Destination	Port	Protocol	Action
*	*	64.37.180.23	80	TCP	Allow
*	*	64.37.180.23	443	TCP	Allow
*	*	64.37.160.63	22	TCP	Allow
*	*	*	*	*	Deny

(b)

Source	Port	Destination	Port	Protocol	Action	Notes
10.20.2.63	*	*	*	*	Deny	Blocking TCP/UDP Int_Web
10.100.X.X	*	*	53	TCP	Allow	Allow TCP internal Network
10.100.X.X	*	*	8080	TCP	Allow	Allow HTTP
10.100.X.X	*	*	8443	TCP	Allow	Allow HTTPS
10.20.2.22	*	*	53	TCP	Allow	Allow Remote SSH

(c)

www.companyxyz.com should resolve to the external IP for the Ext_Web which is 64.37.180.23. This IP is external so external networks can communicate with this IP address. If we used the internal IP 10.20.2.63 only those in the internal network can connect to this IP address.

Remote.companyxyz.com should resolve to the external IP of remote access which is 64.37.160.63. This external IP can be connected to by external networks and internal networks. If the internal IP was used 10.20.2.22 only those in the internal network can connect to this IP address.

(d)

The company must port forward the internal IP and the ports, The IP 10.10.1.20 and the ports 8080 and 8443 must be port forwarded (using TCP as the service). Port forwarding allows external networks to connect to a specific service in an internal network.

(e)

Source	Port	Destination	Port	Protocol	Action
10.20.2.22	*	10.100.X.X	22	TCP	Allow
10.20.2.22	*	10.100.X.X	*	*	Deny

(f)

Yes, as EXT_FW allows access to the EXT_WEB server, if appropriate SSH is setup on the EXT_WEB it can be directly accessed by SSH assuming port forwarding has also been done.

(g) Int_Web is only allowed to respond to requests from the internal network. The user cannot directly connect to this network from home. The only way to connect to this server is to connect to a machine on the internal network by ssh into RemoteAccess (64.37.160.63:22) and to then ssh into Int_Web. Thus, a direct connection from home directly cannot work.

(h)

Source	Port	Destination	Port	Protocol	Action
10.100.X.X	*	10.20.2.22	22	TCP	Allow
10.100.X.X	*	10.20.2.63	8080	TCP	Allow
10.100.X.X	*	10.20.2.63	8443	TCP	Allow
10.100.X.X	*	*	53	TCP	Allow
*	*	*	*	*	Deny

(i)

The Guest_Wifi and Home_Eth networks are two independent subnetworks with their own IP ranges. These two networks cannot communicate with each other.

A segregated Wifi network will allow you to prevent guests to access your private Samba Share folders and access to the printer. This segregated network ensures that those that want to see your private family documents must be physically connected to your server.

Give the laptop a static IP address and create a firewall rules that allows connection from the allocated IP address to the subnetwork of Home_WAP.

The external IP (80.245.7.8) will show on the webservers log. This is because when the user is sending a request to the webserver the home router will send the request to the webserver using the external IP address and provide it a port number. This process is done by using a method known as Network Address Translation (NAT), this maps a single IP address space into another in this case we are mapping the internal IP into the external IP. NAT modifies the IP header whilst it is in transit from the router.

Question 2:

(a) L/(R1) + L/(R2)

(b) (i) 2 Users

(ii) As there is a 2Mbps link and each user is continuously transmitting 1Mbps there can be up to two maximum simultaneous transmissions without packet delay as the total 2Mbps link is being used. If more than two users are using this link for instance three users, a link of 3Mbps is required which is more than the available bandwidth therefore, there will be a packet queue formed at the router as packets will be queuing faster than the packets that are being transported across the network.

(iii) 0.2

(iv) $0.2^3 = 0.008$

(c) (i) 500Kbps
(ii) (4*8*10^6)/(500*1000) = 64 seconds
(iii) 100Kbps, 320 seconds

(d) The first packet will arrive the destination at N(L/R) time. The second packet will arrive at N(L/R) + L/R time. Where P is the last packet to reach the destination the overall equation must be N(L/R) + (P-1)(L/R) = (N + P – 1)(L/R).

(e) The IP address and port number of a process of the other host.

(f) TCP has a reliable data transfer ensuring a connection using three way TCP.
Security is not provided by either this must be done in the application layer
A guarantee of an arrival time not provided by either
Guarantee that a certain throughput is maintained is not provided by either.

(g) A TCP server needs two sockets as this protocol work by request and response. TCP sends data to the client and requests a response from the client, since data is being transported between both machines it requires two sockets. A TCP server only needs two sockets even with n simultaneous connections because server hosts can support many simultaneous TCP connections. TCP requires the IP and port of both the source and destination, when the packet is received at the destination machine, they can be demultiplexed into different sockets.

(h) (i)

One RTT for handshake + (One RTT for handshake + One RTT for page request + page data) + 5(One RTT for handshake + One RTT for handshake + One image request)

$(0.1 + 0.1 + \frac{1*1000*8}{1*10^6}) + 5*(0.1 + 0.1 + \frac{50*1000*8}{1*10^6})$ = **3.208s**

(ii)
Getting the first webpage is required to get other data.
Getting first page: One RTT for handshake + (One RTT for handshake + One RTT for page request + page data)
$= (0.1 + 0.1 + \frac{1*1000*8}{1*10^6}) = 0.208$
Now the Images can be done in parallel
(One RTT for handshake + One RTT for handshake + One image request)

$= *(0.1 + 0.1 + \frac{50*1000*8}{1*10^6}) = 0.6$

Total time taken = 0.208 + 0.6 **= 0.808s**

(iii)
One RTT for handshake + (one RTT + document) + 5 (one RTT + image)
$= (0.1 + 0.1 + \frac{1*1000*8}{1*10^6}) + 5*(0.1 + \frac{50*1000*8}{1*10^6})$ = **2.708s**

(i) **(i)** Source: 33000, Destination: 80
(ii) Source: 80, Destination 33000
(iii) No, as HTTP is a TCP protocol so UDP cannot be used to contact the server. HTTP relies on the three-way TCP handshake.

(j) 01011101 11110010
00110010 00001101

UDP Checksum requires it to be the 1's complement, in this case there are two bits that are not correct. This segment is not correctly received. The receiver must request the packet again.

Question 3

(a) (i) gaia.cs.umass.edu/cs453/index.html
(ii) HTTP 1.1.
(iii) Persistent because of Connection:keep-alive
(iv) Mozilla/5.0 Browser The browser type is needed as different browsers handle HTTP messages differently.

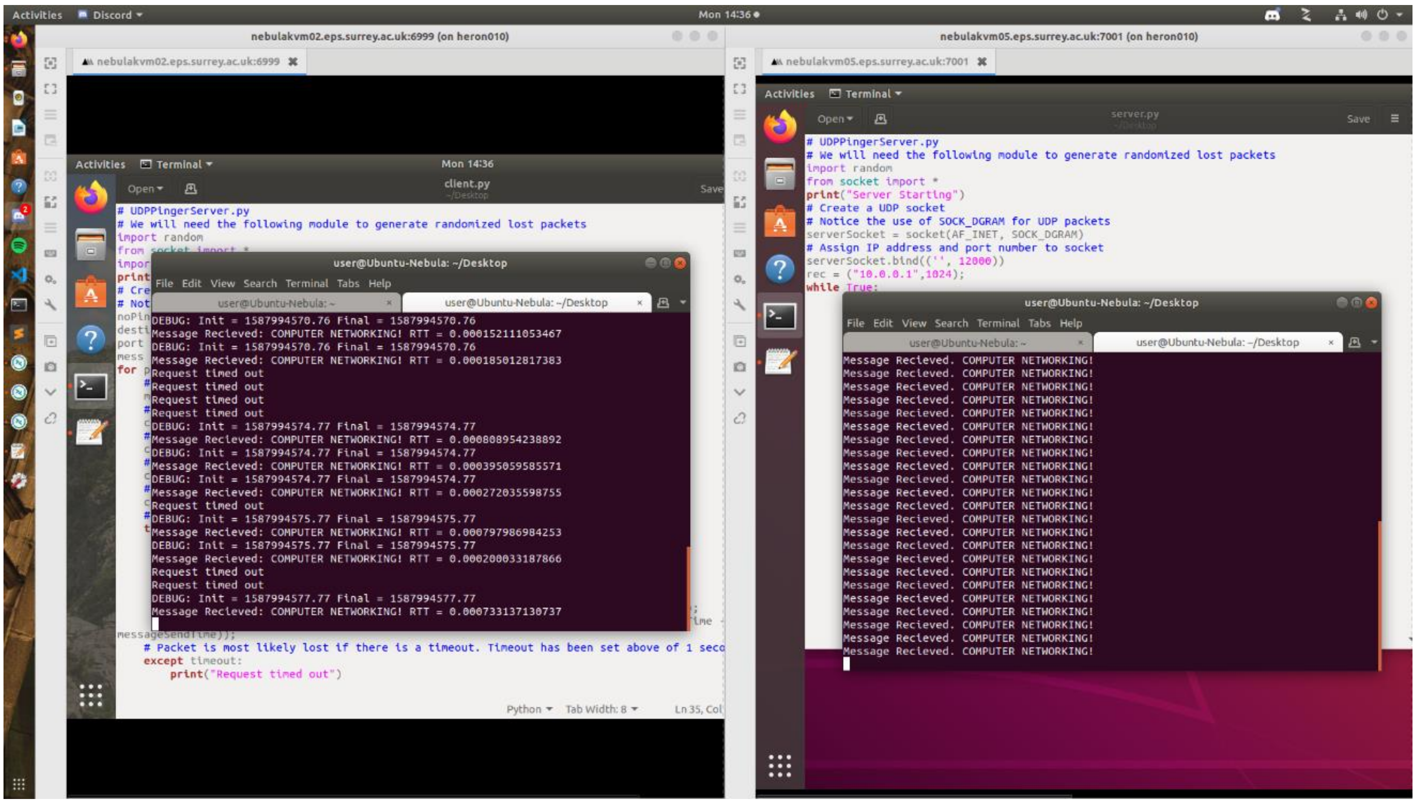
(b) (i) Server was successfully able to find the document as a 200 OK response was returned. Tue, 07 Mar 200812:39:45GMT
(ii) Sat, 10 Dec2005 18:27:46GMT
(iii) 3874 bytes
(iv) The first 5 bytes are "<!doc".Yes the server has agreed to keep the persistent connection as there is the "Connection: Keep-Alive" in the response.

(c) (i) 71.192.34.104
(ii) Source: 192.168.1.100, Source Port: 4335
Destination: 63.233.169.104, Destination Port: 80

(d) First TCP SYN segment sent used by the GET at 7.109267? The first SYN packet was sent at 7.075657000
Acknowledgment is received at 7.108986

(e) (i) 6.069168s
Source: 71.192.34.104, Source Port: 4335, Destination IP: 64.233.169.104, Destination Port: 80
The fields that are still the same are the source port, destination port, destination IP address.
(ii) The source IP address got changed. This is because the source IP in the home side was an internal IP address, this had to translated to an external IP address using NAT so external networks can communicate with this computer. The TCP checksum is also modified as this checksum is formed with the data in the header, since the source IP changed the checksum must also change.

Question 4



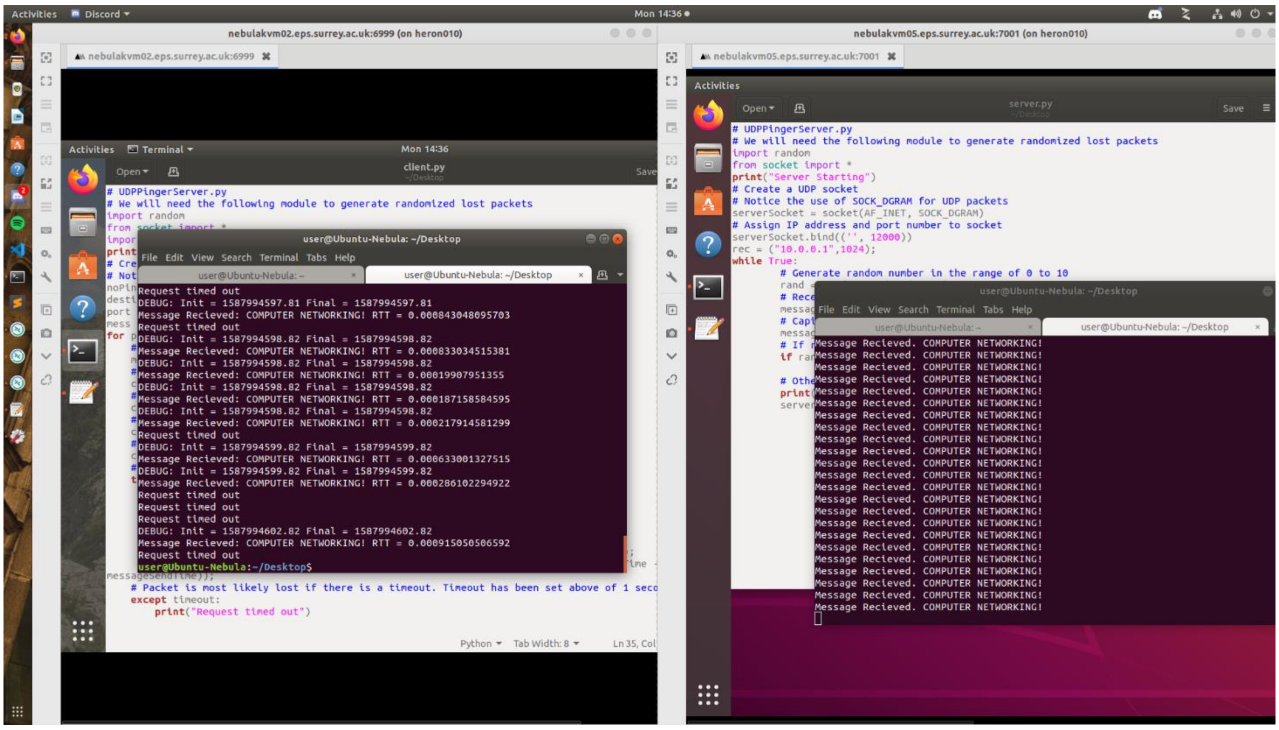
```
# UDPPingerServer.py
# We will need the following module to generate randomized lost packets
import random
```

```
DEBUG: Init = 1587994570.76 Final = 1587994570.76
Message Recieved: COMPUTER NETWORKING! RTT = 0.000152111053467
DEBUG: Init = 1587994570.76 Final = 1587994570.76
Message Recieved: COMPUTER NETWORKING! RTT = 0.000185012817383
Request timed out
Request timed out
Request timed out
Request timed out
DEBUG: Init = 1587994574.77 Final = 1587994574.77
Message Recieved: COMPUTER NETWORKING! RTT = 0.000808954238892
DEBUG: Init = 1587994574.77 Final = 1587994574.77
Message Recieved: COMPUTER NETWORKING! RTT = 0.000395059585571
DEBUG: Init = 1587994574.77 Final = 1587994574.77
Message Recieved: COMPUTER NETWORKING! RTT = 0.000272035598755
Request timed out
DEBUG: Init = 1587994575.77 Final = 1587994575.77
Message Recieved: COMPUTER NETWORKING! RTT = 0.000797986984253
DEBUG: Init = 1587994575.77 Final = 1587994575.77
Message Recieved: COMPUTER NETWORKING! RTT = 0.000200033187866
Request timed out
Request timed out
DEBUG: Init = 1587994577.77 Final = 1587994577.77
Message Recieved: COMPUTER NETWORKING! RTT = 0.000733137130737
```

```
    # Packet is most likely lost if there is a timeout. Timeout has been set above of 1 seco
    except timeout:
        print("Request timed out")
```

```
# UDPPingerServer.py
# We will need the following module to generate randomized lost packets
import random
from socket import *
print("Server Starting")
# Create a UDP socket
# Notice the use of SOCK_DGRAM for UDP packets
serverSocket = socket(AF_INET, SOCK_DGRAM)
# Assign IP address and port number to socket
serverSocket.bind(('', 12000))
rec = ("10.0.0.1",1024);
while True:
```

```
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
```



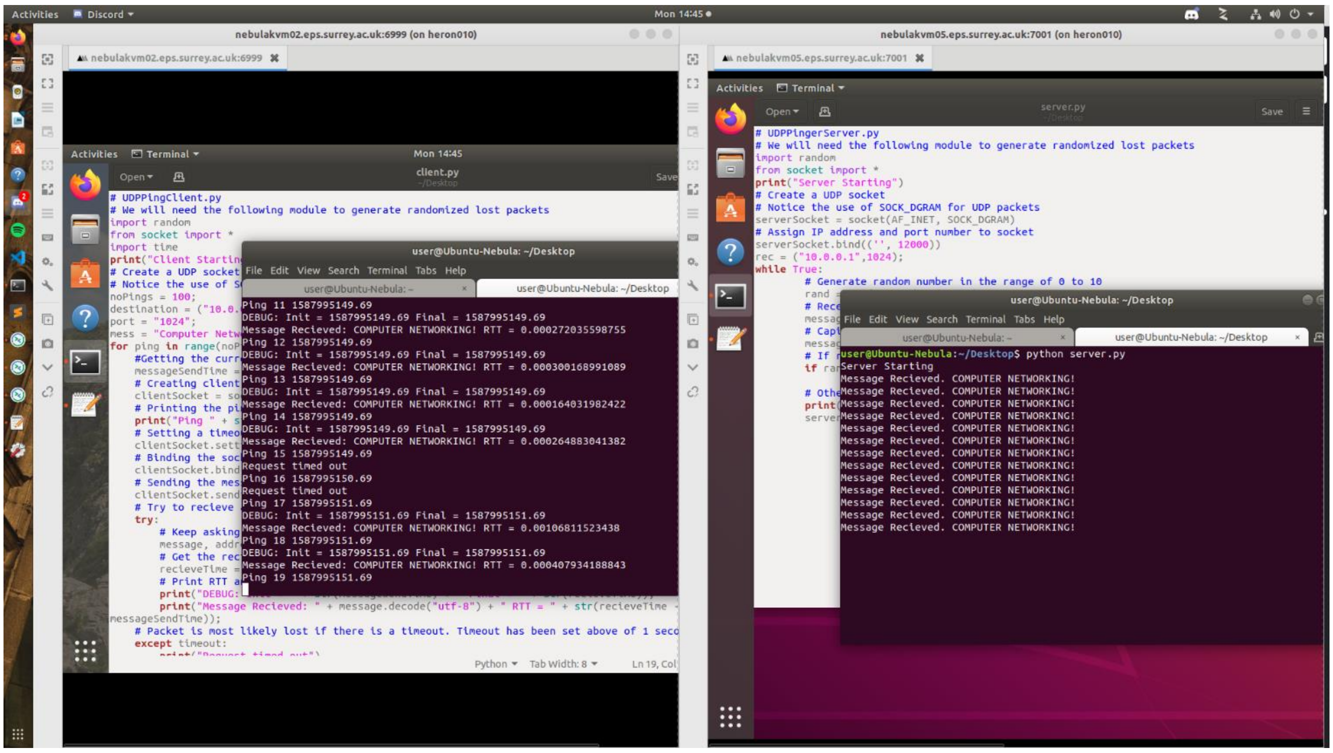
```
# UDPPingerServer.py
# We will need the following module to generate randomized lost packets
import random
```

```
Request timed out
DEBUG: Init = 1587994597.81 Final = 1587994597.81
Message Recieved: COMPUTER NETWORKING! RTT = 0.000843048095703
Request timed out
DEBUG: Init = 1587994598.82 Final = 1587994598.82
Message Recieved: COMPUTER NETWORKING! RTT = 0.000833034515381
DEBUG: Init = 1587994598.82 Final = 1587994598.82
Message Recieved: COMPUTER NETWORKING! RTT = 0.00019907951355
DEBUG: Init = 1587994598.82 Final = 1587994598.82
Message Recieved: COMPUTER NETWORKING! RTT = 0.000187158584595
DEBUG: Init = 1587994598.82 Final = 1587994598.82
Message Recieved: COMPUTER NETWORKING! RTT = 0.000217914581299
Request timed out
DEBUG: Init = 1587994599.82 Final = 1587994599.82
Message Recieved: COMPUTER NETWORKING! RTT = 0.000633001327515
DEBUG: Init = 1587994599.82 Final = 1587994599.82
Message Recieved: COMPUTER NETWORKING! RTT = 0.000286102294922
Request timed out
Request timed out
Request timed out
DEBUG: Init = 1587994602.82 Final = 1587994602.82
Message Recieved: COMPUTER NETWORKING! RTT = 0.000915050506592
Request timed out
user@Ubuntu-Nebula:~/Desktop$
```

```
    # Packet is most likely lost if there is a timeout. Timeout has been set above of 1 seco
    except timeout:
        print("Request timed out")
```

```
# UDPPingerServer.py
# We will need the following module to generate randomized lost packets
import random
from socket import *
print("Server Starting")
# Create a UDP socket
# Notice the use of SOCK_DGRAM for UDP packets
serverSocket = socket(AF_INET, SOCK_DGRAM)
# Assign IP address and port number to socket
serverSocket.bind(('', 12000))
rec = ("10.0.0.1",1024);
while True:
        # Generate random number in the range of 0 to 10
```

```
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
```

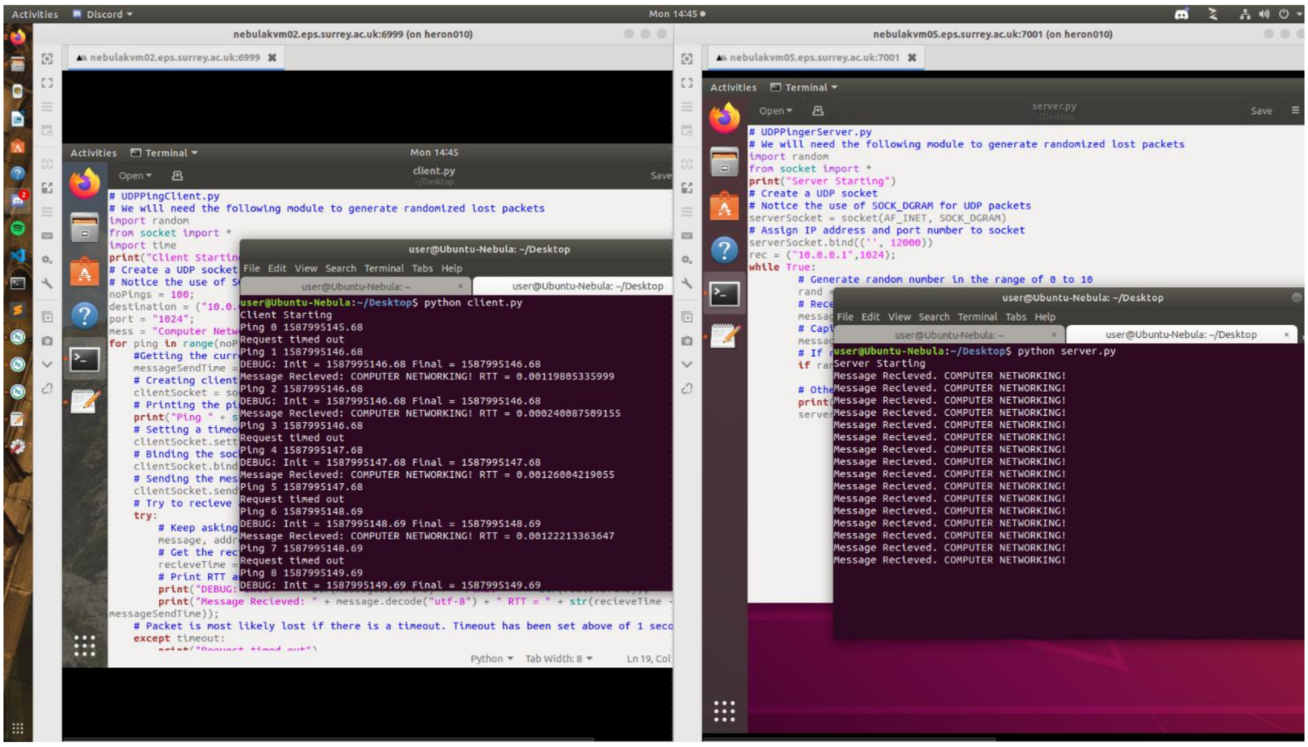


```
# UDPPingClient.py
# We will need the following module to generate randomized lost packets
import random
from socket import *
import time
print("Client Startin
# Create a UDP socket
# Notice the use of S
noPings = 100;
destination = ("10.0.
port = "1024";
mess = "Computer Netw
for ping in range(noP
    #Getting the curr
    messageSendTime =
    # Creating client
    clientSocket = so
    # Printing the pi
    print("Ping " + s
    # Setting a timeo
    clientSocket.sett
    # Binding the soc
    clientSocket.bind
    # Sending the mes
    clientSocket.send
    # Try to recieve
    try:
        # Keep asking
        message, addr
        # Get the rec
        recieveTime =
        # Print RTT a
        print("DEBUG:
        print("Message Recieved: " + message.decode("utf-8") + " RTT = " + str(recieveTime -
messageSendTime));
    # Packet is most likely lost if there is a timeout. Timeout has been set above of 1 seco
    except timeout:
```

```
Ping 11 1587995149.69
DEBUG: Init = 1587995149.69 Final = 1587995149.69
Message Recieved: COMPUTER NETWORKING! RTT = 0.000272035598755
Ping 12 1587995149.69
DEBUG: Init = 1587995149.69 Final = 1587995149.69
Message Recieved: COMPUTER NETWORKING! RTT = 0.000300168991089
Ping 13 1587995149.69
DEBUG: Init = 1587995149.69 Final = 1587995149.69
Message Recieved: COMPUTER NETWORKING! RTT = 0.000164031982422
Ping 14 1587995149.69
DEBUG: Init = 1587995149.69 Final = 1587995149.69
Message Recieved: COMPUTER NETWORKING! RTT = 0.000264883041382
Ping 15 1587995149.69
Request timed out
Ping 16 1587995150.69
Request timed out
Ping 17 1587995151.69
DEBUG: Init = 1587995151.69 Final = 1587995151.69
Message Recieved: COMPUTER NETWORKING! RTT = 0.00106811523438
Ping 18 1587995151.69
DEBUG: Init = 1587995151.69 Final = 1587995151.69
Message Recieved: COMPUTER NETWORKING! RTT = 0.000407934188843
Ping 19 1587995151.69
```

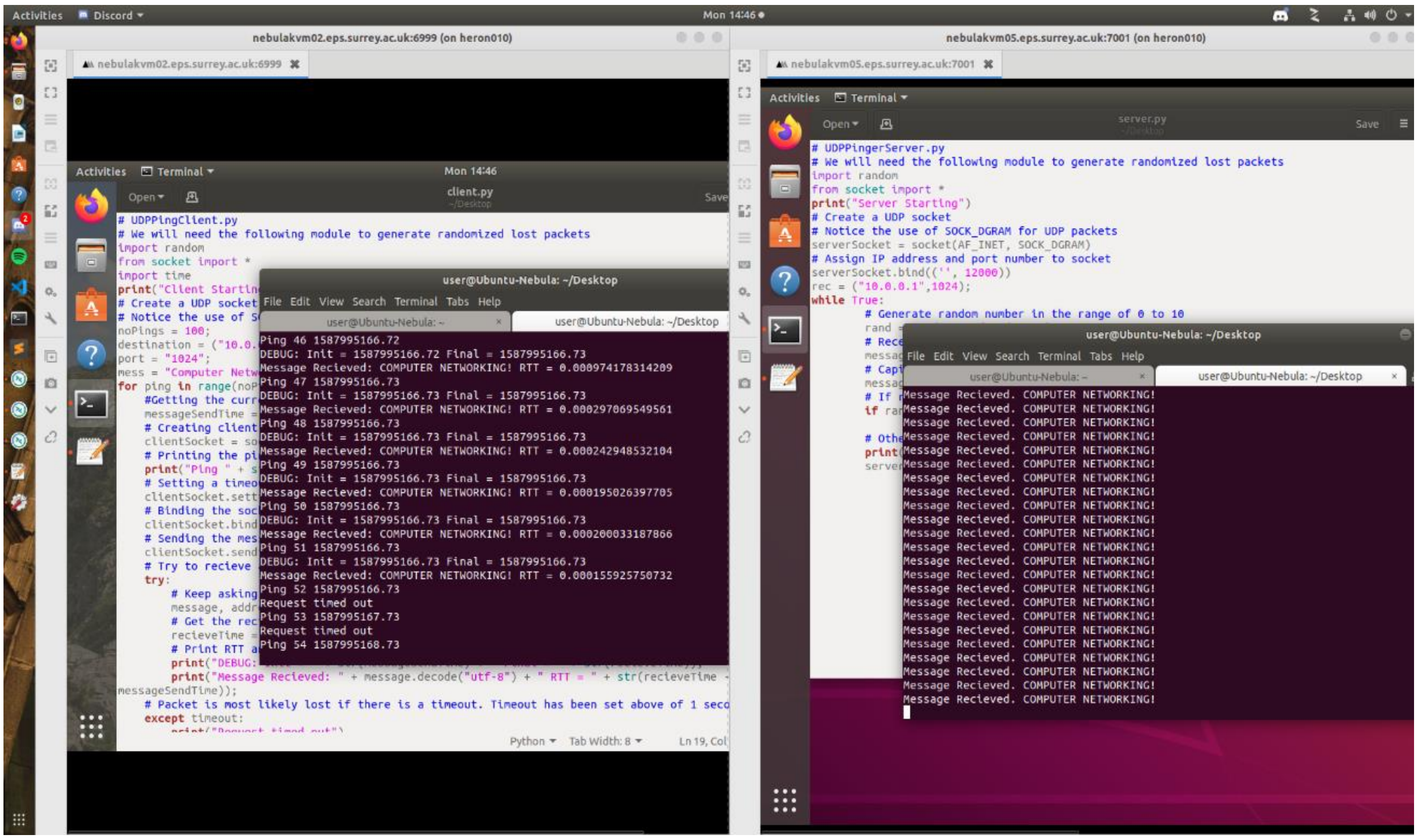
```
# UDPPingerServer.py
# We will need the following module to generate randomized lost packets
import random
from socket import *
print("Server Starting")
# Create a UDP socket
# Notice the use of SOCK_DGRAM for UDP packets
serverSocket = socket(AF_INET, SOCK_DGRAM)
# Assign IP address and port number to socket
serverSocket.bind(('', 12000))
rec = ("10.0.0.1",1024);
while True:
        # Generate random number in the range of 0 to 10
        rand =
        # Rece
        messag
        # Capi
        messag
        # If r
        if rar

        # Othe
        print(
        server
```

```
user@Ubuntu-Nebula:~/Desktop$ python server.py
Server Starting
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
```



```
user@Ubuntu-Nebula:~/Desktop$ python client.py
Client Starting
Ping 0 1587995145.68
Request timed out
Ping 1 1587995146.68
DEBUG: Init = 1587995146.68 Final = 1587995146.68
Message Recieved: COMPUTER NETWORKING! RTT = 0.00119805335999
Ping 2 1587995146.68
DEBUG: Init = 1587995146.68 Final = 1587995146.68
Message Recieved: COMPUTER NETWORKING! RTT = 0.000240087509155
Ping 3 1587995146.68
Request timed out
Ping 4 1587995147.68
DEBUG: Init = 1587995147.68 Final = 1587995147.68
Message Recieved: COMPUTER NETWORKING! RTT = 0.00126004219055
Ping 5 1587995147.68
Request timed out
Ping 6 1587995148.69
DEBUG: Init = 1587995148.69 Final = 1587995148.69
Message Recieved: COMPUTER NETWORKING! RTT = 0.00122213363647
Ping 7 1587995148.69
Request timed out
Ping 8 1587995149.69
DEBUG: Init = 1587995149.69 Final = 1587995149.69
```

```
user@Ubuntu-Nebula:~/Desktop$ python server.py
Server Starting
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
```



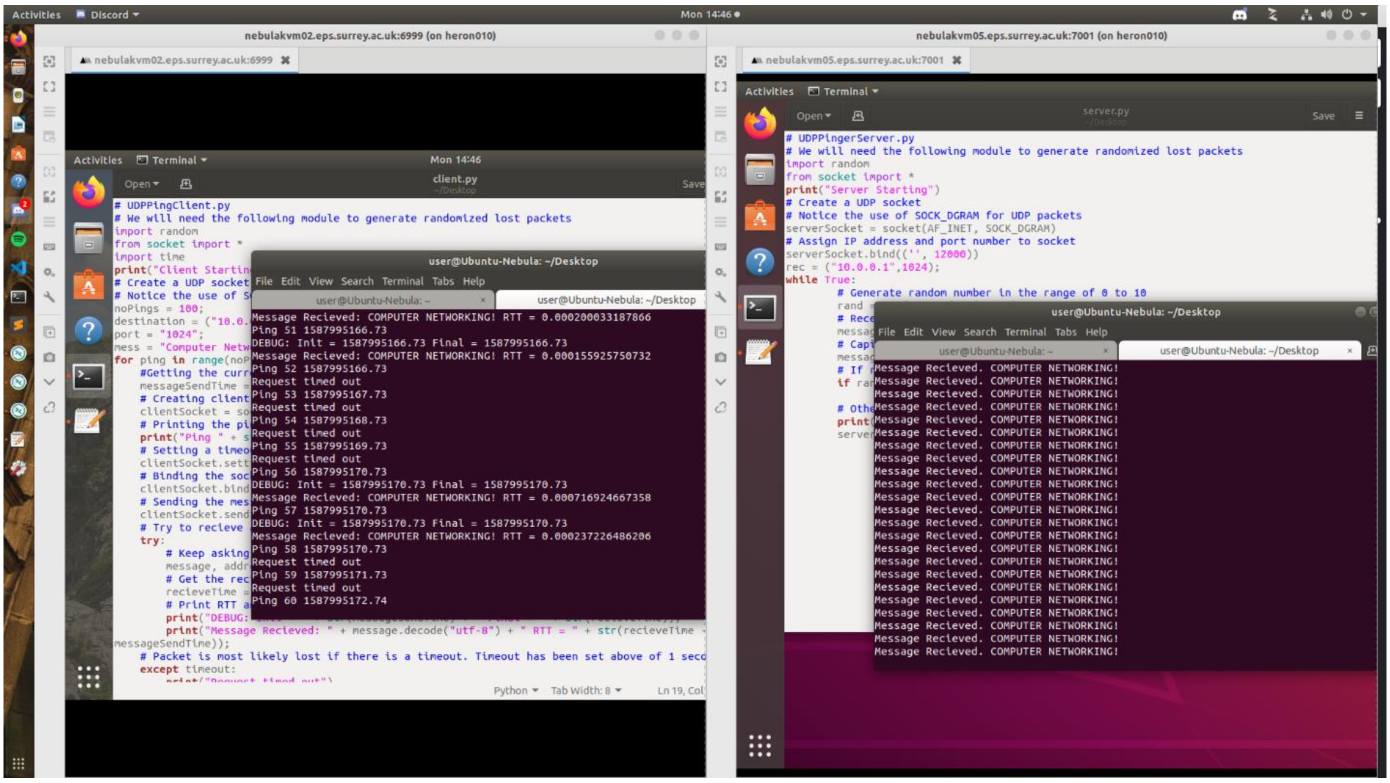
Activities
Discord
Mon 14:46
nebulakvm02.eps.surrey.ac.uk:6999 (on heron010)
nebulakvm05.eps.surrey.ac.uk:7001 (on heron010)
Terminal
Open
client.py
server.py
Save
user@Ubuntu-Nebula: ~/Desktop
File Edit View Search Terminal Tabs Help
user@Ubuntu-Nebula: ~
Python
Tab Width: 8
Ln 19, Col

```
# UDPPingClient.py
# We will need the following module to generate randomized lost packets
import random
from socket import *
import time
print("Client Startin
# Create a UDP socket
# Notice the use of S
noPings = 100;
destination = ("10.0.
port = "1024";
mess = "Computer Netw
for ping in range(noP
    #Getting the curr
    messageSendTime =
    # Creating client
    clientSocket = so
    # Printing the pi
    print("Ping " + s
    # Setting a timeo
    clientSocket.sett
    # Binding the soc
    clientSocket.bind
    # Sending the mes
    clientSocket.send
    # Try to recieve
    try:
        # Keep asking
        message, addr
        # Get the rec
        recieveTime =
        # Print RTT a
        print("DEBUG:
        print("Message Recieved: " + message.decode("utf-8") + " RTT = " + str(recieveTime -
messageSendTime));
    # Packet is most likely lost if there is a timeout. Timeout has been set above of 1 seco
    except timeout:
```

```
Ping 46 1587995166.72
DEBUG: Init = 1587995166.72 Final = 1587995166.73
Message Recieved: COMPUTER NETWORKING! RTT = 0.000974178314209
Ping 47 1587995166.73
DEBUG: Init = 1587995166.73 Final = 1587995166.73
Message Recieved: COMPUTER NETWORKING! RTT = 0.000297069549561
Ping 48 1587995166.73
DEBUG: Init = 1587995166.73 Final = 1587995166.73
Message Recieved: COMPUTER NETWORKING! RTT = 0.000242948532104
Ping 49 1587995166.73
DEBUG: Init = 1587995166.73 Final = 1587995166.73
Message Recieved: COMPUTER NETWORKING! RTT = 0.000195026397705
Ping 50 1587995166.73
DEBUG: Init = 1587995166.73 Final = 1587995166.73
Message Recieved: COMPUTER NETWORKING! RTT = 0.000200033187866
Ping 51 1587995166.73
DEBUG: Init = 1587995166.73 Final = 1587995166.73
Message Recieved: COMPUTER NETWORKING! RTT = 0.000155925750732
Ping 52 1587995166.73
Request timed out
Ping 53 1587995167.73
Request timed out
Ping 54 1587995168.73
```

```
# UDPPingerServer.py
# We will need the following module to generate randomized lost packets
import random
from socket import *
print("Server Starting")
# Create a UDP socket
# Notice the use of SOCK_DGRAM for UDP packets
serverSocket = socket(AF_INET, SOCK_DGRAM)
# Assign IP address and port number to socket
serverSocket.bind(('', 12000))
rec = ("10.0.0.1",1024);
while True:
        # Generate random number in the range of 0 to 10
        rand =
        # Rece
        messag
        # Capi
        messag
        # If r
        if ran

        # Othe
        print(
        server
```

```
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
```



Activities
Discord
Mon 14:46
nebulakvm02.eps.surrey.ac.uk:6999 (on heron010)
nebulakvm05.eps.surrey.ac.uk:7001 (on heron010)
Terminal
Open
client.py
server.py
Save
user@Ubuntu-Nebula: ~/Desktop
File Edit View Search Terminal Tabs Help
user@Ubuntu-Nebula: ~
Python
Tab Width: 8
Ln 19, Col

```
# UDPPingClient.py
# We will need the following module to generate randomized lost packets
import random
from socket import *
import time
print("Client Startin
# Create a UDP socket
# Notice the use of S
noPings = 100;
destination = ("10.0.
port = "1024";
mess = "Computer Netw
for ping in range(noP
    #Getting the curr
    messageSendTime =
    # Creating client
    clientSocket = so
    # Printing the pi
    print("Ping " + s
    # Setting a timeo
    clientSocket.sett
    # Binding the soc
    clientSocket.bind
    # Sending the mes
    clientSocket.send
    # Try to recieve
    try:
        # Keep asking
        message, addr
        # Get the rec
        recieveTime =
        # Print RTT a
        print("DEBUG:
        print("Message Recieved: " + message.decode("utf-8") + " RTT = " + str(recieveTime -
messageSendTime));
    # Packet is most likely lost if there is a timeout. Timeout has been set above of 1 seco
    except timeout:
```

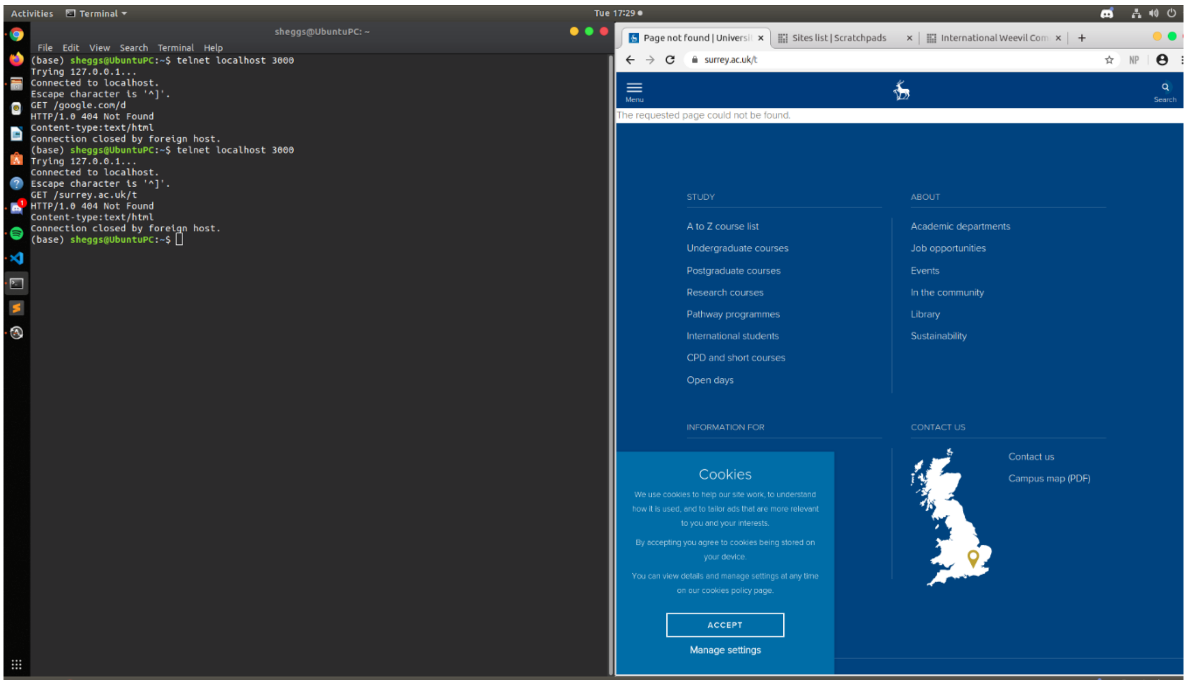
```
Message Recieved: COMPUTER NETWORKING! RTT = 0.000200033187866
Ping 51 1587995166.73
DEBUG: Init = 1587995166.73 Final = 1587995166.73
Message Recieved: COMPUTER NETWORKING! RTT = 0.000155925750732
Ping 52 1587995166.73
Request timed out
Ping 53 1587995167.73
Request timed out
Ping 54 1587995168.73
Request timed out
Ping 55 1587995169.73
Request timed out
Ping 56 1587995170.73
DEBUG: Init = 1587995170.73 Final = 1587995170.73
Message Recieved: COMPUTER NETWORKING! RTT = 0.000716924667358
Ping 57 1587995170.73
DEBUG: Init = 1587995170.73 Final = 1587995170.73
Message Recieved: COMPUTER NETWORKING! RTT = 0.000237226486206
Ping 58 1587995170.73
Request timed out
Ping 59 1587995171.73
Request timed out
Ping 60 1587995172.74
```

```
# UDPPingerServer.py
# We will need the following module to generate randomized lost packets
import random
from socket import *
print("Server Starting")
# Create a UDP socket
# Notice the use of SOCK_DGRAM for UDP packets
serverSocket = socket(AF_INET, SOCK_DGRAM)
# Assign IP address and port number to socket
serverSocket.bind(('', 12000))
rec = ("10.0.0.1",1024);
while True:
        # Generate random number in the range of 0 to 10
        rand =
        # Rece
        messag
        # Capi
        messag
        # If r
        if ran

        # Othe
        print(
        server
```

```
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
Message Recieved. COMPUTER NETWORKING!
```

Proxy Server

404 Images



```
(base) sheggs@UbuntuPC:~$ telnet localhost 3000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /google.com/d
HTTP/1.0 404 Not Found
Content-type:text/html
Connection closed by foreign host.
(base) sheggs@UbuntuPC:~$ telnet localhost 3000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /surrey.ac.uk/t
HTTP/1.0 404 Not Found
Content-type:text/html
Connection closed by foreign host.
(base) sheggs@UbuntuPC:~$
```

surrey.ac.uk/t

The requested page could not be found.

STUDY: A to Z course list, Undergraduate courses, Postgraduate courses, Research courses, Pathway programmes, International students, CPD and short courses, Open days

ABOUT: Academic departments, Job opportunities, Events, In the community, Library, Sustainability

INFORMATION FOR

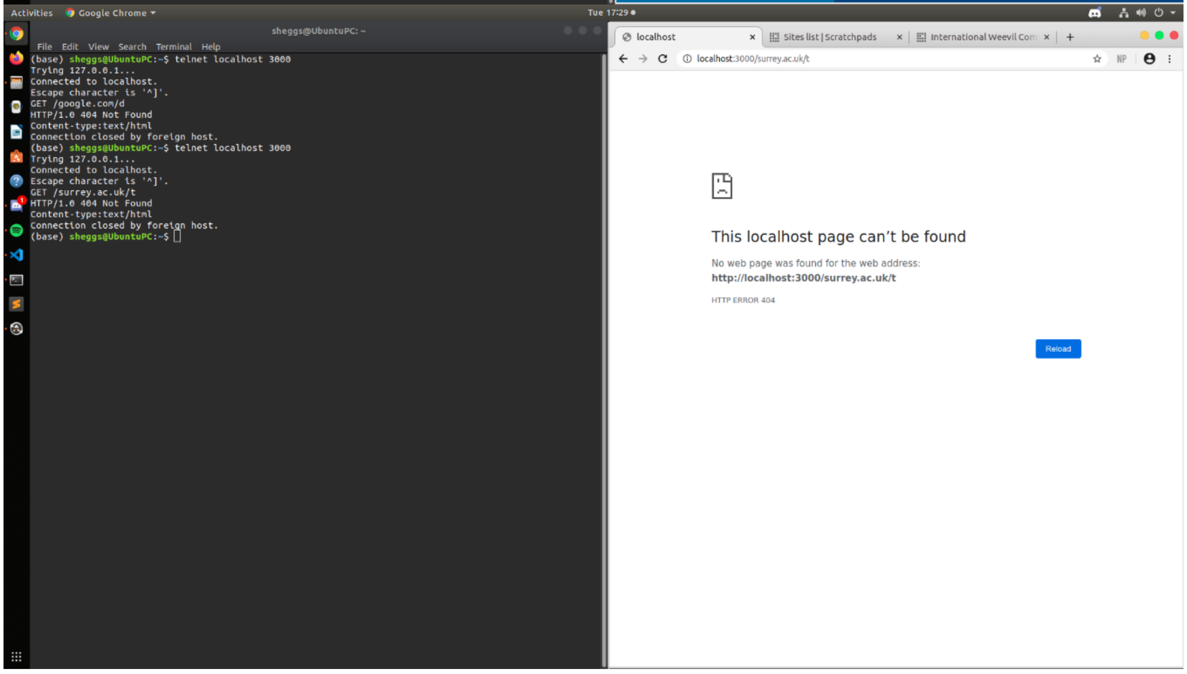
CONTACT US: Contact us, Campus map (PDF)

Cookies

We use cookies to help our site work, to understand how it is used, and to tailor ads that are more relevant to you and your interests.

By accepting you agree to cookies being stored on your device.

You can view details and manage settings at any time on our cookies policy page.



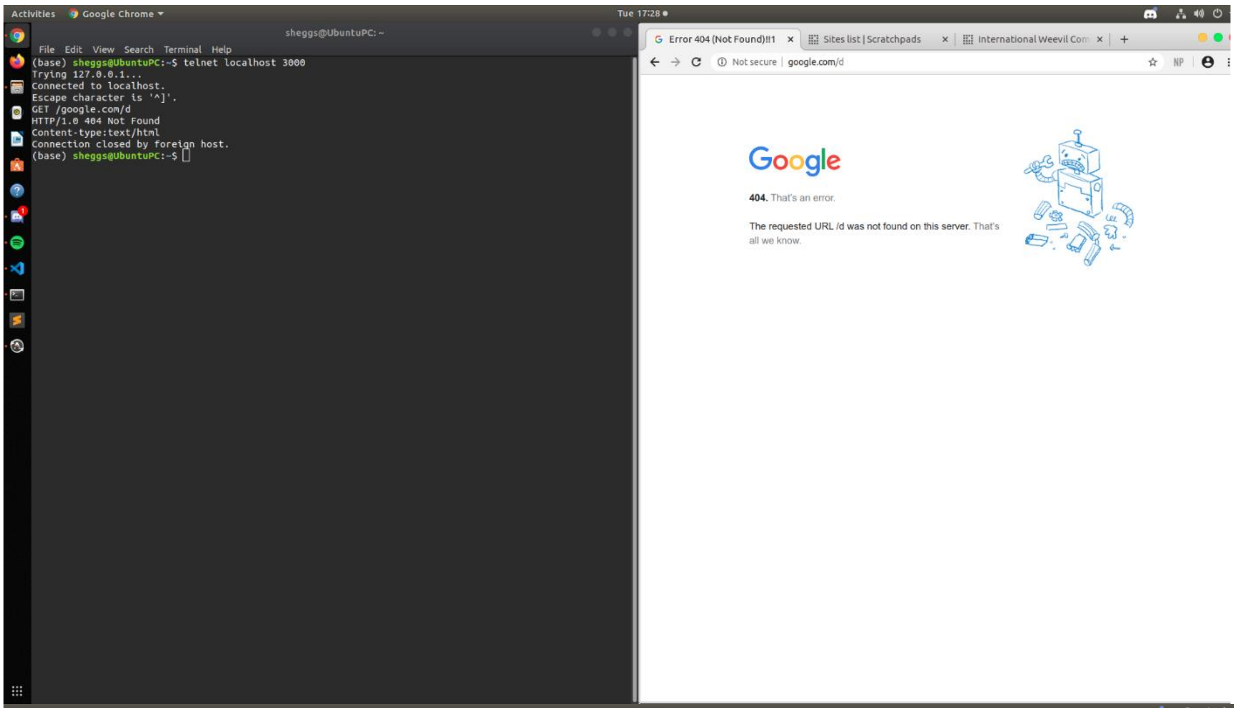
```
(base) sheggs@UbuntuPC:~$ telnet localhost 3000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /google.com/d
HTTP/1.0 404 Not Found
Content-type:text/html
Connection closed by foreign host.
(base) sheggs@UbuntuPC:~$ telnet localhost 3000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /surrey.ac.uk/t
HTTP/1.0 404 Not Found
Content-type:text/html
Connection closed by foreign host.
(base) sheggs@UbuntuPC:~$
```

localhost:3000/surrey.ac.uk/t

This localhost page can’t be found

No web page was found for the web address: **http://localhost:3000/surrey.ac.uk/t**

HTTP ERROR 404



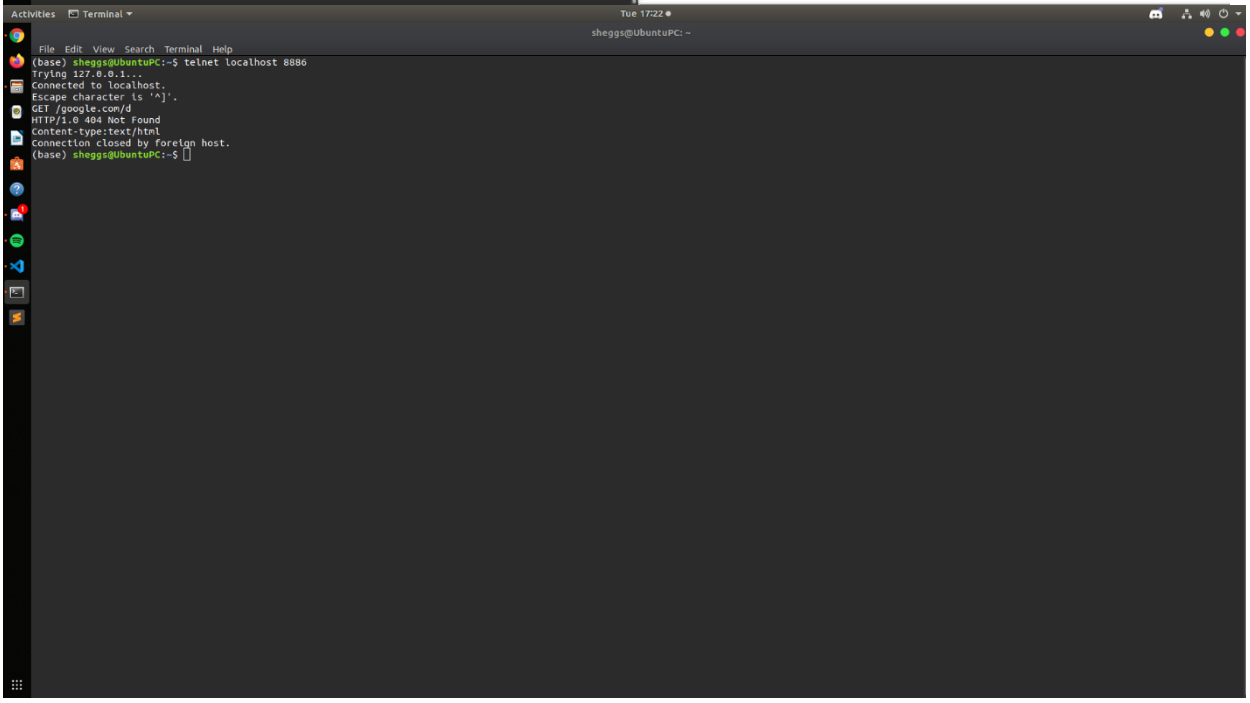
```
(base) sheggs@UbuntuPC:~$ telnet localhost 3000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /google.com/d
HTTP/1.0 404 Not Found
Content-type:text/html
Connection closed by foreign host.
(base) sheggs@UbuntuPC:~$
```

Error 404 (Not Found)!!1 — Not secure | google.com/d

Google

404. That's an error.

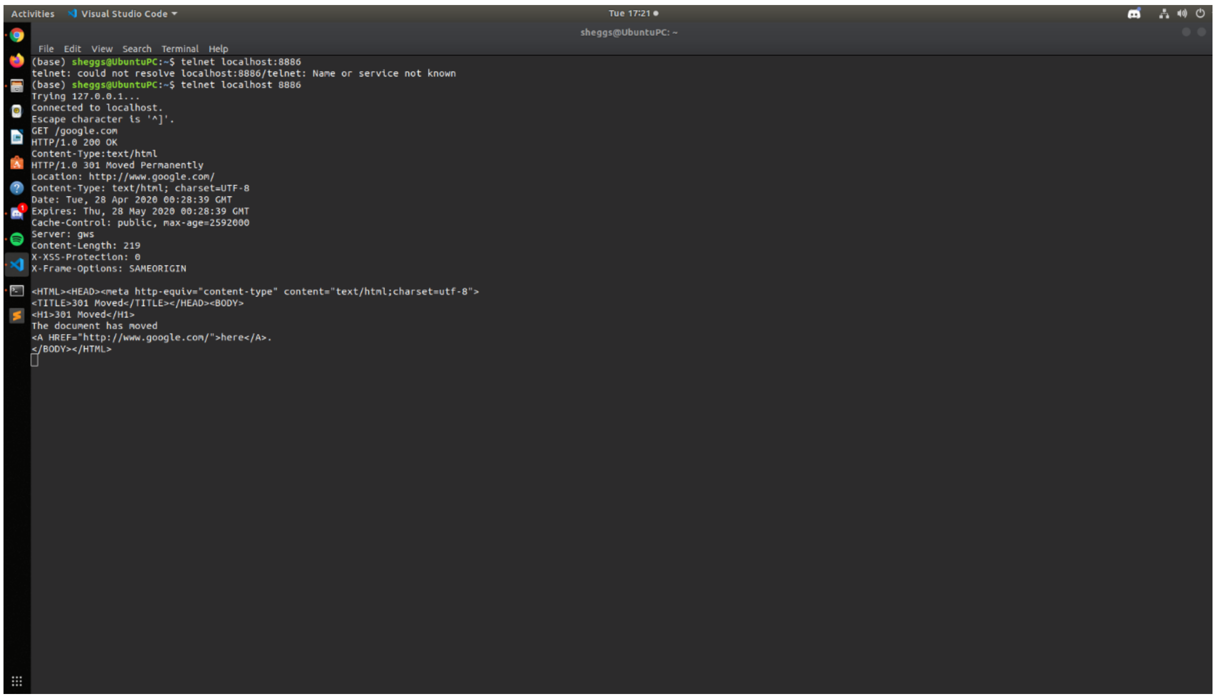
The requested URL /d was not found on this server. That's all we know.



```
(base) sheggs@UbuntuPC:~$ telnet localhost 8886
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /google.com/d
HTTP/1.0 404 Not Found
Content-type:text/html
Connection closed by foreign host.
(base) sheggs@UbuntuPC:~$
```

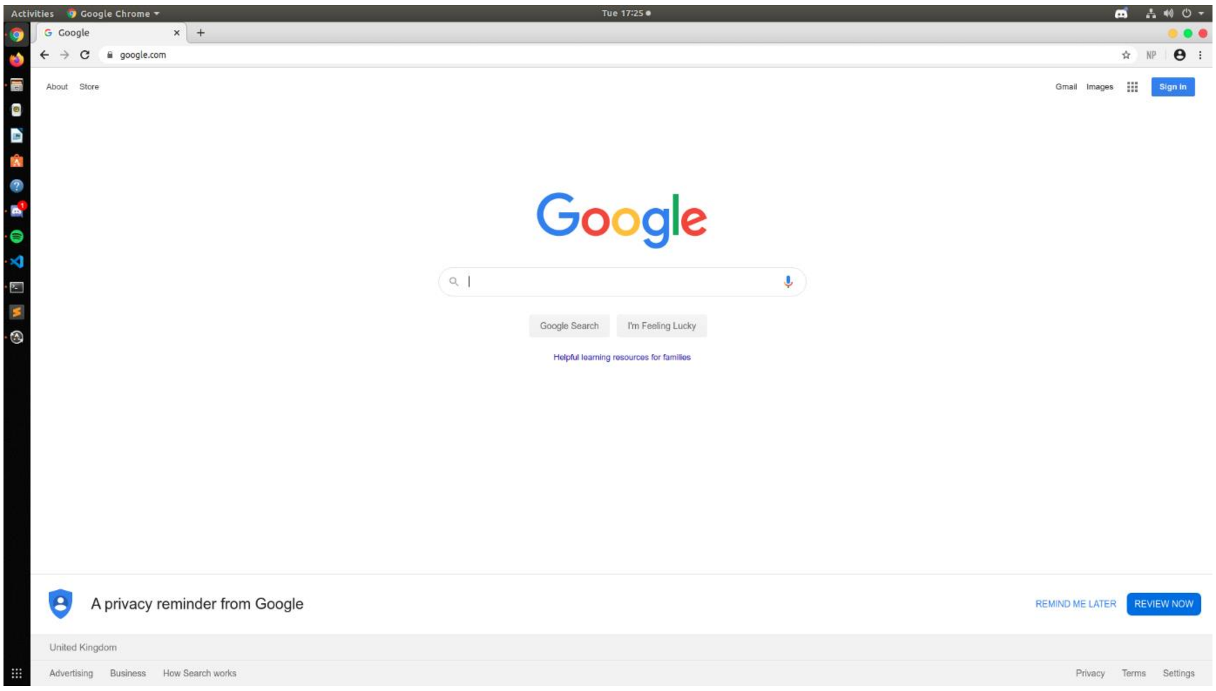
301 Moved

When it is cached (200 request returned but 301 message is shown on page)



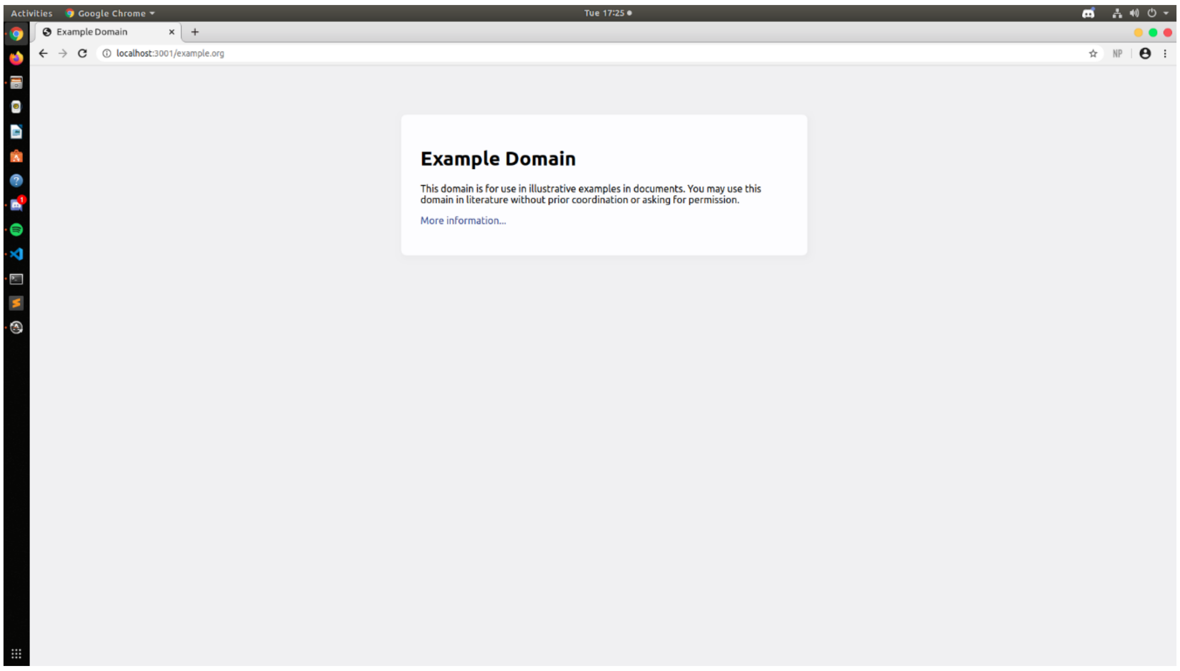
```
(base) sheggs@UbuntuPC:~$ telnet localhost:8886
telnet: could not resolve localhost:8886/telnet: Name or service not known
(base) sheggs@UbuntuPC:~$ telnet localhost 8886
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /google.com
HTTP/1.0 200 OK
Content-Type:text/html
HTTP/1.0 301 Moved Permanently
Location: http://www.google.com/
Content-Type: text/html; charset=UTF-8
Date: Tue, 28 Apr 2020 00:28:39 GMT
Expires: Thu, 28 May 2020 00:28:39 GMT
Cache-Control: public, max-age=2592000
Server: gws
Content-Length: 219
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
```

Redirects the client to google.com (On first try, other attempts the cached result shows the error)



Real 200 responses

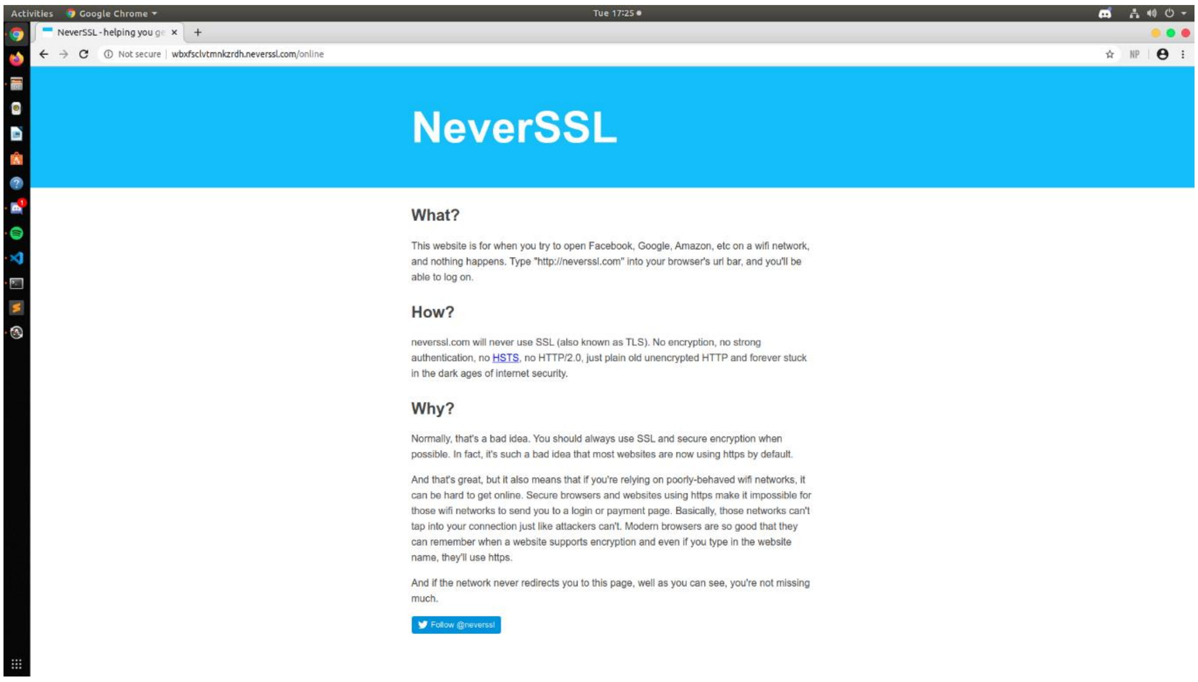
CSS hrefs have been fixed but for some websites it can’t get the required data.



Example Domain

This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.

More information...



NeverSSL

What?

This website is for when you try to open Facebook, Google, Amazon, etc on a wifi network, and nothing happens. Type "http://neverssl.com" into your browser's url bar, and you'll be able to log on.

How?

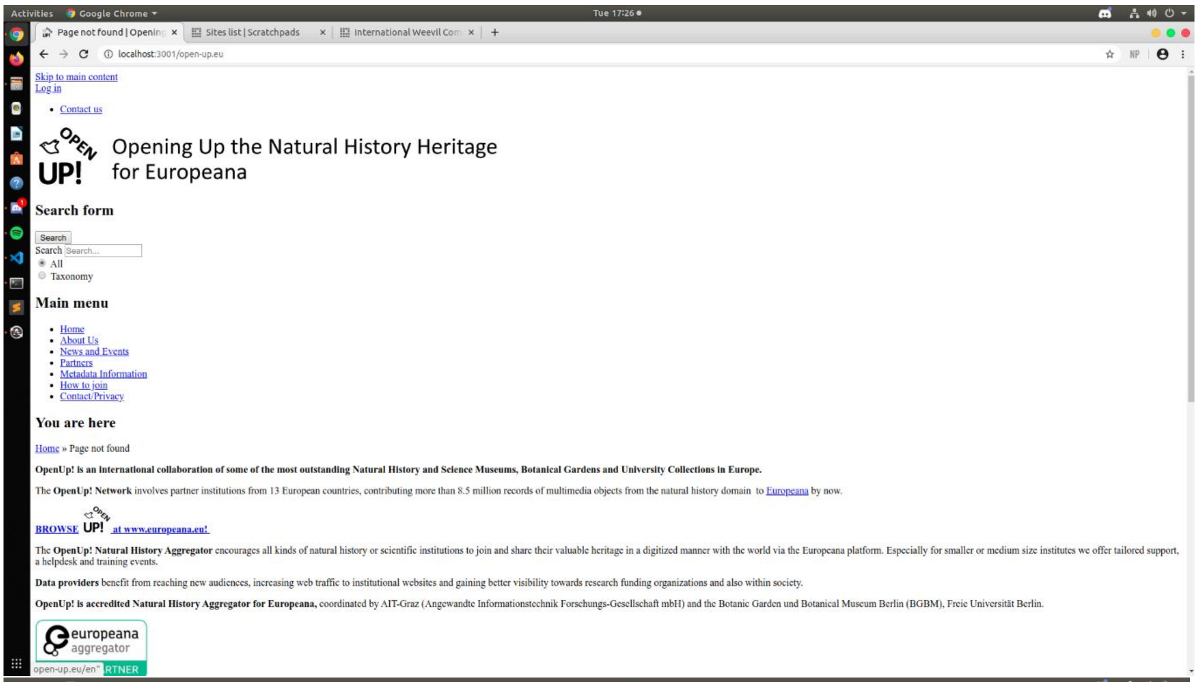
neverssl.com will never use SSL (also known as TLS). No encryption, no strong authentication, no HSTS, no HTTP/2.0, just plain old unencrypted HTTP and forever stuck in the dark ages of internet security.

Why?

Normally, that's a bad idea. You should always use SSL and secure encryption when possible. In fact, it's such a bad idea that most websites are now using https by default.

And that's great, but it also means that if you're relying on poorly-behaved wifi networks, it can be hard to get online. Secure browsers and websites using https make it impossible for those wifi networks to send you to a login or payment page. Basically, those networks can't tap into your connection just like attackers can't. Modern browsers are so good that they can remember when a website supports encryption and even if you type in the website name, they'll use https.

And if the network never redirects you to this page, well as you can see, you're not missing much.



Skip to main content
Log in

- Contact us

Opening Up the Natural History Heritage for Europeana

Search form

Search
Search
All
Taxonomy

Main menu

- Home
- About Us
- News and Events
- Partners
- Metadata Information
- How to join
- Contact/Privacy

You are here

Home » Page not found

OpenUp! is an international collaboration of some of the most outstanding Natural History and Science Museums, Botanical Gardens and University Collections in Europe.

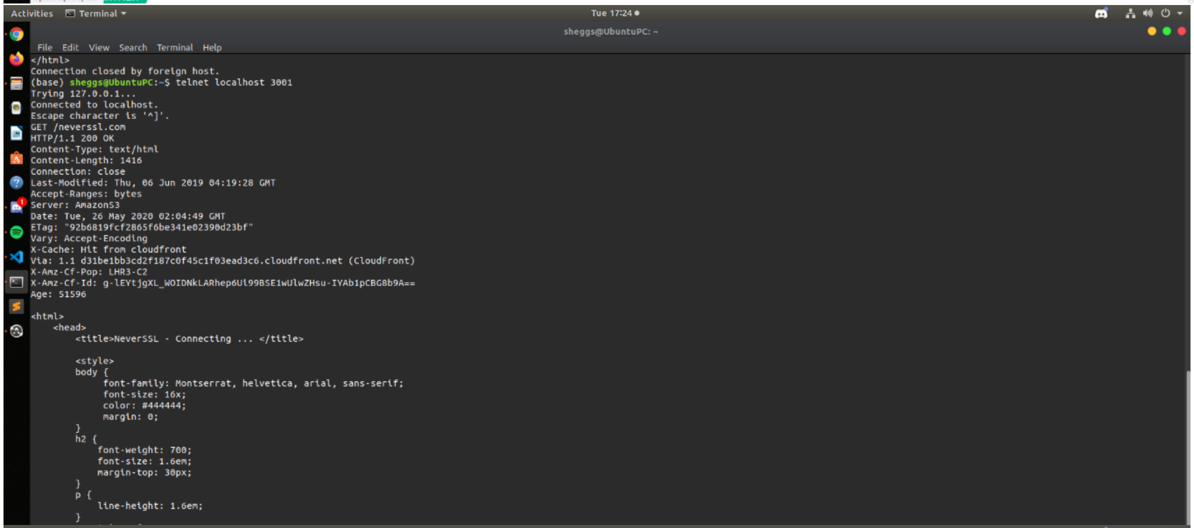
The **OpenUp! Network** involves partner institutions from 13 European countries, contributing more than 8.5 million records of multimedia objects from the natural history domain to Europeana by now.

BROWSE UP! at www.europeana.eu!

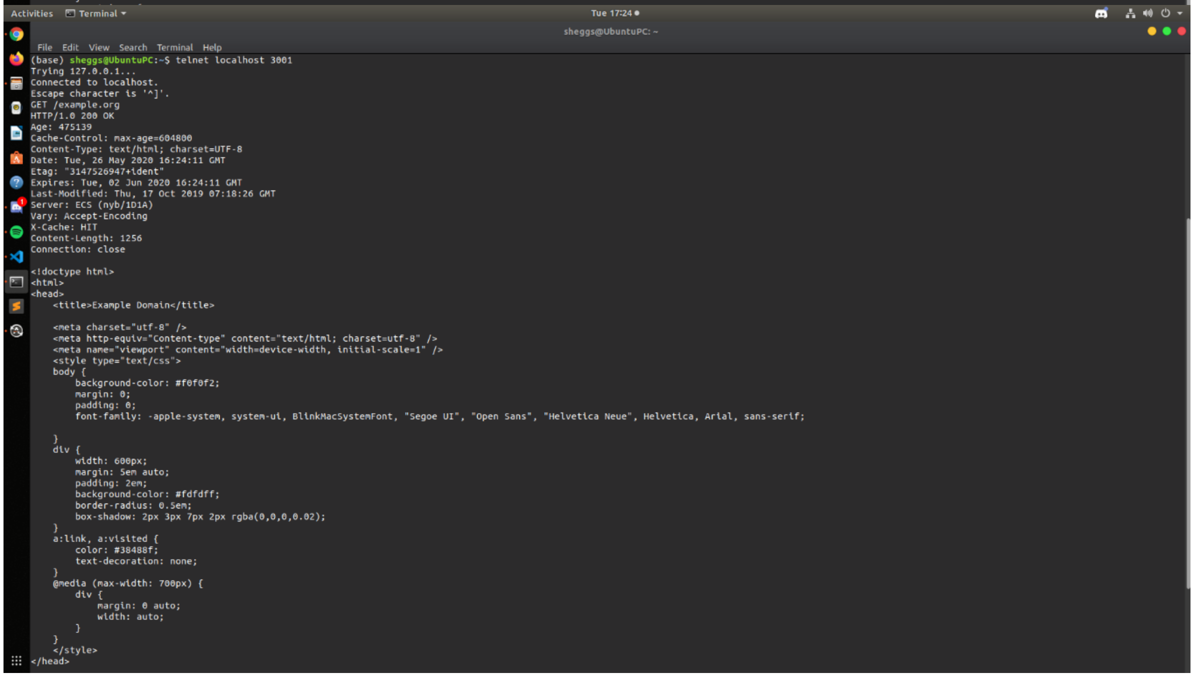
The **OpenUp! Natural History Aggregator** encourages all kinds of natural history or scientific institutions to join and share their valuable heritage in a digitized manner with the world via the Europeana platform. Especially for smaller or medium size institutes we offer tailored support, a helpdesk and training events.

Data providers benefit from reaching new audiences, increasing web traffic to institutional websites and gaining better visibility towards research funding organizations and also within society.

OpenUp! is accredited Natural History Aggregator for Europeana, coordinated by AIT-Graz (Angewandte Informationstechnik Forschungs-Gesellschaft mbH) and the Botanic Garden und Botanical Museum Berlin (BGBM), Freie Universität Berlin.



```
</html>
Connection closed by foreign host.
(base) sheggs@UbuntuPC:~$ telnet localhost 3001
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /neverssl.com
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1416
Connection: close
Last-Modified: Thu, 06 Jun 2019 04:19:28 GMT
Accept-Ranges: bytes
Server: AmazonS3
Date: Tue, 26 May 2020 02:04:49 GMT
ETag: "92b6819fcf2865f6be341e02390d23bf"
Vary: Accept-Encoding
X-Cache: Hit from cloudfront
Via: 1.1 d31be1bb3cd2f187c0f45c1f03ead3c6.cloudfront.net (CloudFront)
X-Amz-Cf-Pop: LHR3-C2
X-Amz-Cf-Id: g-lEYtjgXL_WOIDNkLARhep6Ui99BSE1wUlwZHsu-IYAb1pCBG8b9A==
Age: 51596

<html>
    <head>
        <title>NeverSSL - Connecting ... </title>

        <style>
        body {
             font-family: Montserrat, helvetica, arial, sans-serif;
             font-size: 16x;
             color: #444444;
             margin: 0;
        }
        h2 {
            font-weight: 700;
            font-size: 1.6em;
            margin-top: 30px;
        }
        p {
            line-height: 1.6em;
        }
```



```
(base) sheggs@UbuntuPC:~$ telnet localhost 3001
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /example.org
HTTP/1.0 200 OK
Age: 475139
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Tue, 26 May 2020 16:24:11 GMT
Etag: "3147526947+ident"
Expires: Tue, 02 Jun 2020 16:24:11 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (nyb/1D1A)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 1256
Connection: close

<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 2em;
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
    @media (max-width: 700px) {
        div {
            margin: 0 auto;
            width: auto;
        }
    }
    </style>
</head>
```

Question 5

(a) (i) MTA = Mail Transfer Agent

(ii)

(A) “Too many recipients” has an error code of 452 and not 552 so this error code is incorrect.

(B) Error 552 means too much mail data.

(iii)

(A) 220 message means service ready in the RFC documentation. This message is an initial message, a receiver will send this reply when the connection is completed.

(B) 220 message timeouts are 5 minutes. Length of all objects is not compulsory, the RFC states “implementation techniques that impose no limits on the lengths of these objects should be used”. However, maximum total lengths for local-part, domain, path, command line, reply line, message buffer and recipients’ buffer has been stated and appropriate error respond codes have been allocated 500,501,452 and 522.

(C) A TCP connection must be established for a 220 message to be send. This message can be delayed until the system allows for more mail to be processed. A TCP connection is needed to be started for any SMTP messages to be sent.

(iv) SMTP messages must be queued to be sent from the SMTP client to the SMTP server. SMPTP clients will periodically attempt to send messages to a SMPTP server, mail must be periodically retried by the sender as it cannot respond with an error message under any circumstances. Failure of a message being sent can only be determined after a timeout of a few minutes.

(b) SRV maps hostname and port numbers of services with specified services. This allows the use of subdomains for different services. SRV record allows admins to use several servers for a single domain. Allowing services to move between hosts with little problems.

Question 6

(a) (i)

n = (11)(19) = 209

z = (11-1)(19-1) = 180

(ii)

e = 17 is an acceptable choice as e is less than the value of n and has no common factor with z.

(iii)

de = 1 (mod z)

d(17) = 1 mod 180

d = 53

(iv)

Encryption

m = 7

c = m^e mod n

c = 7^{17} mod 209

c = 182

Decryption

m = c^d mod n

m = 182^{53} mod 209

m = 7

(b) Alice: Secret Key S_A, Public Key (T_A) =(g^S_A) mod p, Shared Key T_B^S_A mod p

Bob: Secret Key S_B, Public Key (T_B) =(g^S_B) mod p, Shared Key T_A^S_B mod p

(i)

S = (T_B^S_A mod p) = (g^S_B)^S_A mod p = g^ S_A mod p ^ S_B = T_A^S_B mod p = S'

(ii)

p = 11, g = 2, S_A = 5, S_B = 12

(T_A) =(g^S_A) mod p = 2^5 mod 11 = 10

(T_B) =(g^S_B) mod p = 2^12 mod 11 = 4

(iii)

S = T_B^S_A mod p S' = T_A^S_B mod p

S = 4^5 mod 11 = 1

S' = 10^12 mod 11 = 1

(iv) Eve will first communicate with Alice and receive the public key T_A and then communicate with Bob and get a public key of T_B. Both Alice and Bob will make a shared key with Eve and then Eve will be the "man-in-the-middle" having access the plain data being sent across the networking and having the ability to send any message to either parties.

(c) Client = 128.238.38.162 Server = 216.75.194.220

(i) Client

(ii) IP = 216.75.194.220, Port = 443

(iii) 283

(iv) Three

(v) Master Secret

(vi) first is bc, last is 29

(d) No longer will work as the MAC that Bob is appending is unique and not shared between Bob and Alice. Alice won't be able to decrypt the message.

Question 7

(a) A host associated with a base station is operating in infrastructure mode. If it has no infrastructure to connect with it is called an ad hoc network. Ad hoc networks have no infrastructure thus the hosts must provide routing, DNS, address assignment and other features. Whilst infrastructure mode must connect to a base station and the host is connected via this base station, so all traditional services are already provided.

(b) A user that is mobile has to switch from one base station to another if they exceed the range of the currently connected base station in a process known as handoff. A user that walks around her house with a laptop is not considered mobile as it doesn't switch between base stations.

(c) Beacon frames send data regarding an AP which consists of its SSID and MAC address. This will be used to aid the wireless station to select an AP to connect to. Beacon frames are transmitted across one of the 11 channels.

(d) SSID, MAC Address

(e) Ethernet has no packet loss whilst 802.11 is more likely to lose packets and higher risk of collisions. Ethernet has a physical link which makes it much more reliable.

(f) Active scanning is by constantly broadcasting a probe frame that will be received by all access points in the range of the wireless host. Passive scanning is scanning channels and waiting for beacon frames.

(g) (i)

No, it will not break down. Both can work parallel over the same channel. If both APs transmitted at the same time, there will be collisions. Maximum transmission rate is 11Mbs.

(ii) Now they are across different channels there are no collisions and the maximum transmission rate is 22Mbps. 11Mbps per channel.
(iii) Host will send a DHCP discovery message via the AP in order to obtain an IP address. Once this IP address is allocated this is now connected to the subnet.

References

[1] https://tools.ietf.org/html/rfc5321#section-4.5.3.1