SYS2 (NET) Laboratory 4 : FTP, Tunnels and UDP

These weekly lab scripts have been written as a self-study resource i.e. an activity to be worked on at your own pace, in your own time. The timetabled practical sessions are an opportunity to get advice and guidance. Therefore, you may not always finish each lab during the timetabled session, but do finish each lab's tasks in your own time. Hardware labs are open Mon-Fri, 9:00-17:00.

The aim of this lab is to introduce one of the core functions of a network i.e. file transfer. Before starting this lab make sure you have watched video **Lectures 2A and 3A**. Back in the early days of the Internet this capability allowed users to access resources at home that previously had only been available in schools or libraries e.g. to download programs, documents, images, things we now take for granted. File transfers can be implemented in a number of different ways, in this lab we shall first look at how existing command line tools can be used to implement this function and then how we can write our own using sockets and python. From the beginning of the Internet the main workhorse protocol was the File Transfer Protocol (FTP). However, there are also a range of standard network command line tools that can be used to transfer files. We will again be using Wireshark to examine how these packets are transmitted across our Pi system and also benchmark our networks performance e.g. its bandwidth, by sending an image from the PC through one or more Raspberry Pi. Next, we will move down the protocol stack to the transport layer, as shown in figure 1 and look at how this functionality can be "replaced" by a simple socket based python communication libraries using the User Datagram Protocol (UDP). At the end of this practical you will understand how to:

- Transfer files using FTP applications (command line and GUI)
- Identify and understand FTP protocol commands and status codes using Wireshark.
- Construct an SSH tunnel and transfer files across multiple machines
- Use the socket API in python to transfer a file using UDP.
- Use port numbers. This is the most important element of this lab!

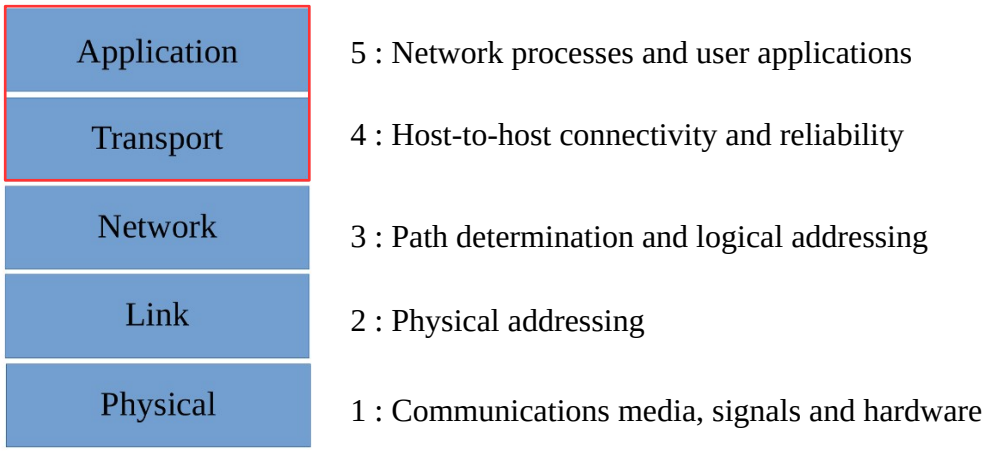

Figure 1 : The Internet protocol stack

Task 1

The File Transfer Protocol (FTP) is one of the founding members of the Internet protocols. Initial developed in 1971, with multiple updates (RFC 959 : https://datatracker.ietf.org/doc/html/rfc959).

This protocol differs from previous protocols we have looked at as it uses two ports, one for control (persistent, port 21) and one for data transfers (non-persistent, port 20).

The FTP protocol is based on a request / response message pair i.e. our familiar client / server model. A client sends a request for a file and the server sends this data or an error code. These commands are send over a persistent connection using port 21 (does time-out eventually if no activity). Requested data is sent over a separate non-persistent connection, on server port 20 to the client's next free incoming port i.e. Active mode. Alternatively, the client can request a port on the server to connect to i.e. Passive mode.

Note, a network connection is uniquely defined by 4 attributes: 2 IP addresses and 2 ports, as shown in figure 2 i.e. the client's source (SRC) IP and Port and the server's destination (DST) IP and Port.

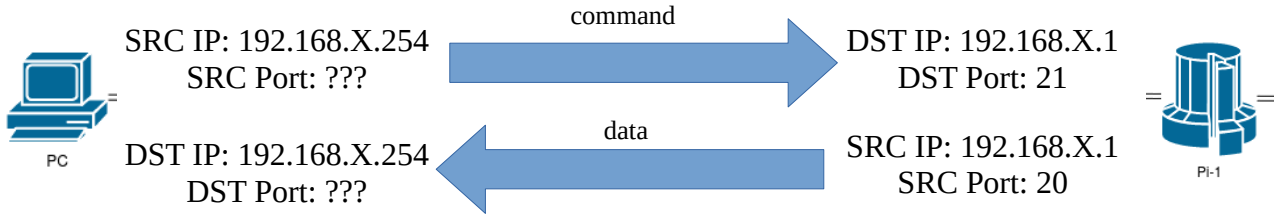

Figure 2 : FTP active mode IP and ports

A selection of commonly used FTP request messages (commands):

- **USER** : user name of the connecting client (in this case pi)
- **PASS** : password of the connecting client (in this case 12345)
- **PASV** : enter passive mode, server returns data port to connect to
- **LIST** : return data containing a list of files and folders in the current directory
- **PWD** : print working directory, return data containing the name of the current dir
- **CWD** : change working directory
- **HOST** : return data containing the name of the ftp server (host)
- **RETR** : retrieve a file, returns data contained in requested file
- **STOR** : send data and store as a file on server in current working directory
- **QUIT** : close connections

Note, for more information on these and other commands used in the file transfer protocol refer to:

https://en.wikipedia.org/wiki/List_of_FTP_commands

FTP response messages return a status code and text message e.g. if a valid request was received : FTP 200 OK, other codes fall under our normal general categories:

- 1xx Positive Preliminary — action is being initiated
- 2xx Positive Completion — command was successfully completed.
- 3xx Positive Intermediate — command accepted, more information is needed.
- 4xx Transient Negative Completion — command failed, retry
- 5xx Permanent Negative Completion — command failed

Note, these are comparable to HTTP, SMTP codes.

On the PC click on the start button and select the command prompt.

`-> Command Prompt`

To start a FTP session on Pi-1, at the command prompt enter the following command :

```
telnet 192.168.X.1 21          (X=Box/Desk, port=21)
```

The number 21 at the end is the port number i.e. we are connecting to the FTP server's command port. To log into the FTP server enter the following commands:

```
user pi
pass 12345
```

You should then see the login successful message as shown in figure 3.

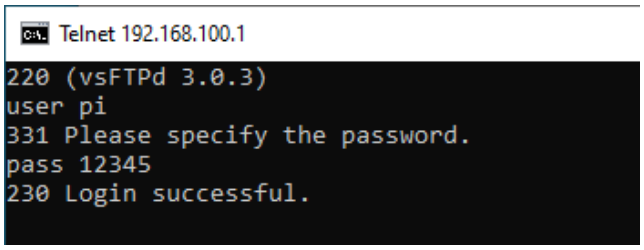

Figure 3 : FTP login

Note, remember by using Telnet you are talking directly to the FTP server running on Pi-1 using the FTP protocol, you are not running an FTP application. For more information on status codes i.e. 220, 331 and 230, refer to:

https://en.wikipedia.org/wiki/List_of_FTP_server_return_codes

To see the root FTP directory enter the following command (Print Working Directory):

```
pwd
```

This will return the home directory for the user `pi`. To see the files and folders in the directory we would enter the command :

```
list
```

However, rather than seeing a listing you will receive an 425 error code :(. This demonstrates the separation of command and data in FTP i.e. you can not send data over the port 21 connection i.e. the command channel. For the PC to receive data it would need to be listening for a connection from the FTP server on the Pi from port 20. Today the Internet is a scary place, so most PCs will have a firewall that will block these types of connection requests. Therefore, rather than the server trying to connect to a port on the client we can get the PC to request a port on the Pi which it should connect to i.e. the passive FTP mode.

To request a data port enter the command:

```
pasv
```

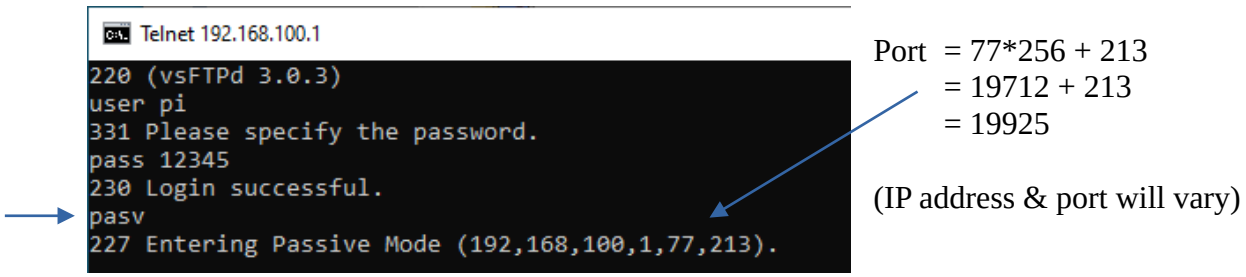

Figure 4 : FTP passive command

This will return the IP address and port number that the client should connect to, to receive the requested data. The returned port number uses base-256 format, therefore to convert this into a decimal value you need to perform the calculation shown in figure 4.

Note, the port number returned is randomly selected by the server from the pool of free ports. The port used will be different for each data transfer i.e. it is closed after the data has been transferred. For more information on this protocol and its active / passive modes of operation refer to:

https://en.wikipedia.org/wiki/File_Transfer_Protocol

To automatically convert the port text string into the port number we can use the python program shown in Appendix A. You can download this program from the VLE: `ftpPort.py`. Launch a python command prompt by clicking on the start button and selecting :

`-> Anaconda`

In this command prompt change to the drive / directory containing this python program using the `cd` command. Alternatively, type "`cd` " at the command line, then within a file browser drag and drop the directory. Next, run this program using the returned IP + port string e.g. for the example in figure 4 this will be :

```
python ftpPort.py 192,168,100,1,77,213
```

The python program will print the command string needed to connect to this port e.g. for the above command this will be:

```
telnet 192.168.100.1 19925          (IP and port number will vary)
```

Repeat this process for the IP + port string returned by your system. Next, start a **new** command prompt i.e. process, on the PC to receive the requested data, click on the start button and select:

`-> Command Prompt`

At the command prompt cut and paste the command string to connect to the FTP server's data port.

Then within the original FTP command terminal re-enter the command:

```
list
```

This will now transfer the directory listing data into the data terminal. Once this transfer is complete the data connection is closed i.e. data port is non-persistent. The command port is persistent, therefore, to close this FTP connection enter in the command terminal:

```
quit
```

To capture the network packets involved in this FTP transfer, open a VNC session on Pi-1. Then within the VNC session start Wireshark, click on the menu icon and select :

`-> Internet -> Wireshark`

This will open the Wireshark GUI. Select the network interface (NIC) to capture packets on, left click on (highlight) the Ethernet 1 (`eth1`) in the Interface list. In the Wireshark GUI enter the packet protocol filter for FTP :

```
ftp
```

as shown in figure 5 and click on the Apply button. To start capturing network packets click on the Start icon  on the top toolbar. Repeat the steps previously described to perform the FTP `list` command. Finally, click on the Stop icon  on the top toolbar to stop capturing network traffic.

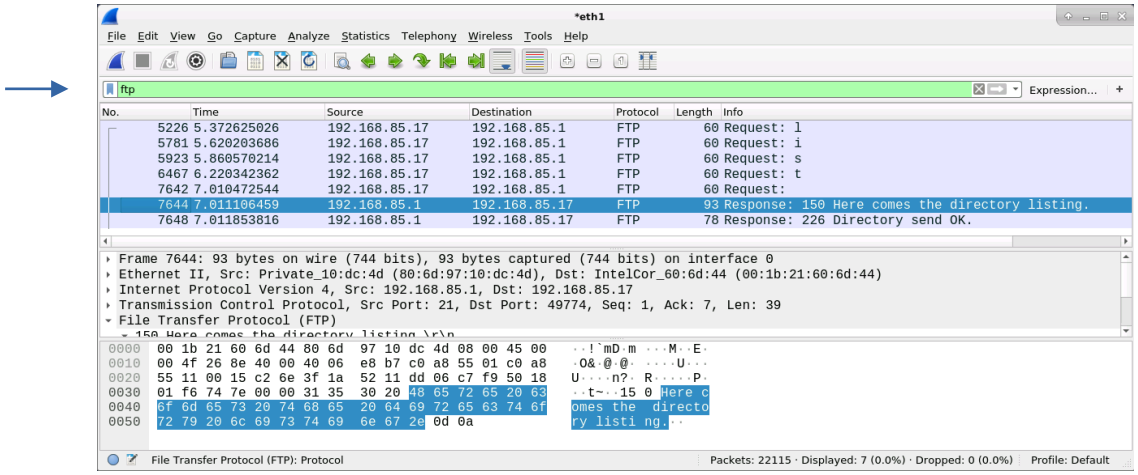

Figure 5 : FTP list command

Task : examine your FTP `list` command capture in Wireshark, expand the FTP section i.e. the request and response packets, do you understand what sequence of steps are being performed in this transfer? What ports are used by the client and server to perform this command, can you identify these in the Wireshark trace? Are ports 20 and 21 used? What packets transferred the `list` data?

Hint, you will need to alter the protocol filter to be able to see the directory listing data. Filter terms can be combined using logical operators e.g. `and`, `or` etc, try using the filter:

```
ftp or ftp-data or tcp.port==XXX,
```

where XXX is the port returned by the python program.

IMPORTANT : before continuing make sure you understand how ports are used when passing packets across a network. Do you understand the difference between a system port and a dynamic port? If you are not sure do ask for help.

Task 2

To simplify the process of transferring files using the FTP protocol we can use an FTP application. On the PC click on the start button and select the FileZila application.

`->  FileZila`

This will launch the FileZila GUI, as shown in figure 6. You can now connect to the FTP server on Pi-1 by entering your normal login details in the top panel and then clicking the `Quickconnect` button.

- Hosts : `192.168.X.1` `(where X=Box/Desk)`
- Username : `pi`
- Password : `12345`
- Port : `21` `(FTP command port)`

A pop-up may appear asking if you are happy to connect across an unencrypted link, click OK.

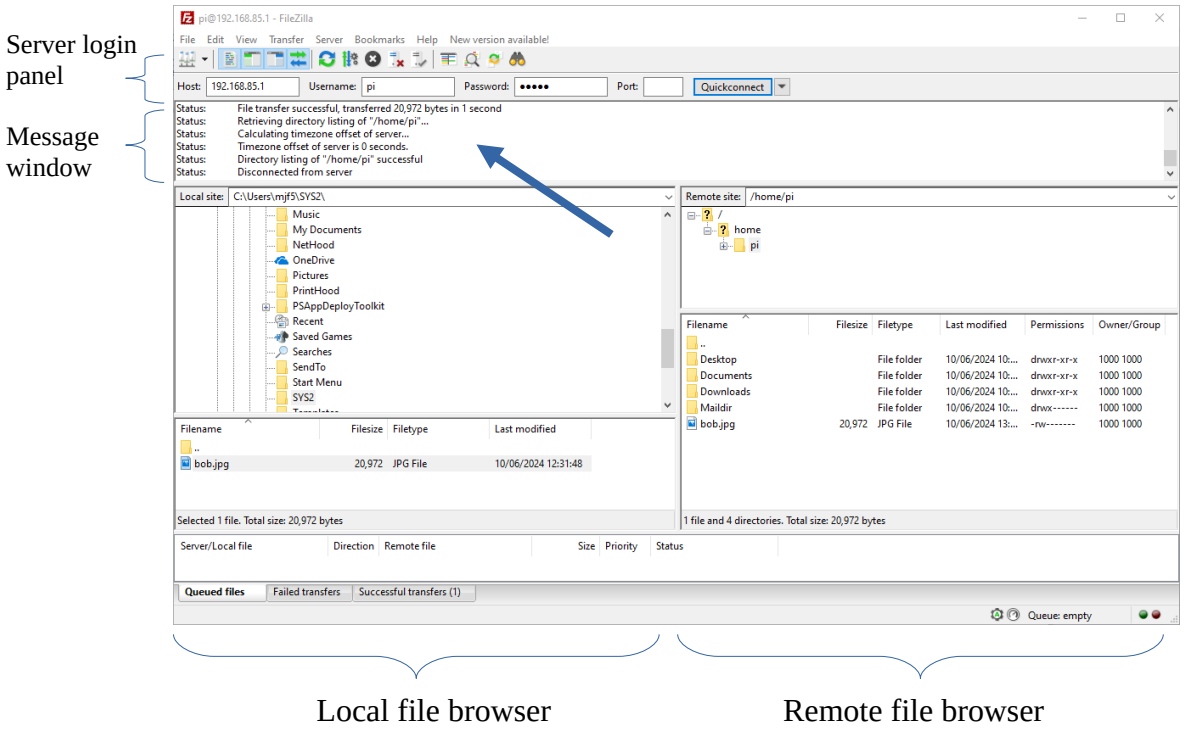

Figure 6 : FileZila application

IMPORTANT, remember that this GUI is just a graphical frontend to our standard FTP client i.e. when you click on a button this request is converted into FTP protocol commands from task 1, as shown in figure 7.

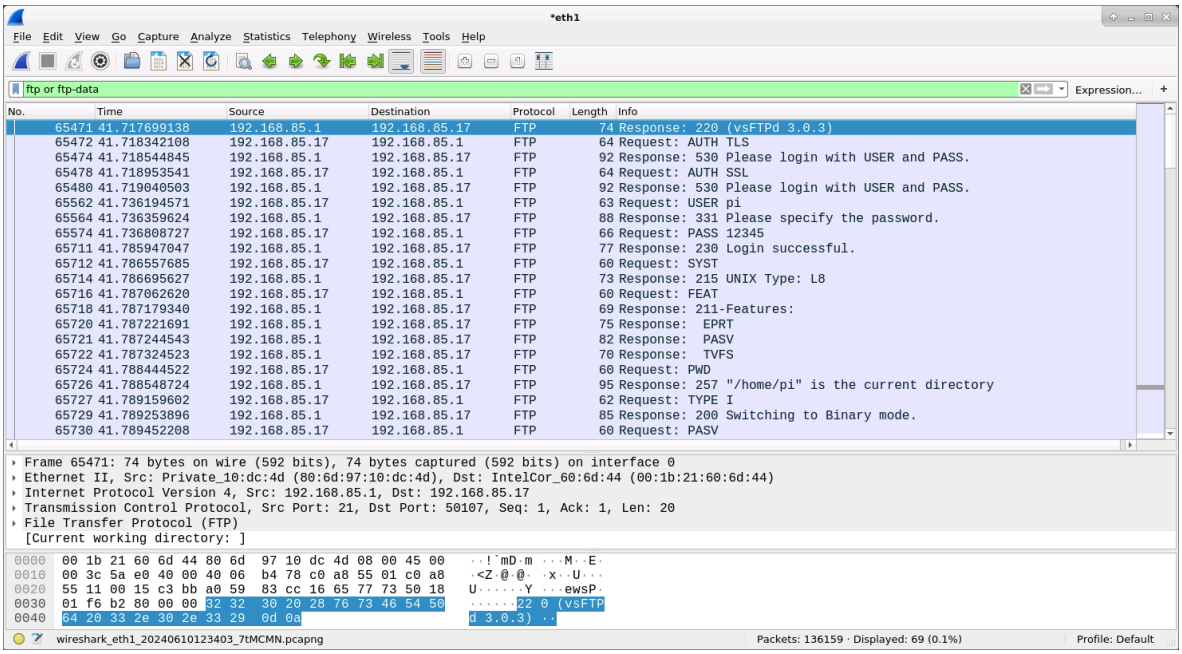

Figure 7 : some of the FTP packets used to transfer bob.jpg

From the VLE download the image of Bob: `bob.jpg`, saving it to your working directory e.g. in figure 6 I used `C:\Users\<USERNAME>\SYS2`. To transfer this file from the PC to Pi-1 simply left click, hold and drag the image of Bob from the local file browser to the remote file browser window i.e. the Pi-1 home directory.

Task : confirm the image of Bob has been transferred to Pi-1 by opening an `SSH` terminal and performing an `ls` command i.e. the file `bob.jpg` should be present in the folder: `/home/pi`. To transfer this image from Pi-1 to PC, simply left click, hold and drag the image of Bob from the remote file browser to the local file browser window.

To capture the network packets involved in this FTP transfer, open a VNC session on Pi-1 and start Wireshark as previously described, selecting the network interface (NIC) Ethernet 1 (`eth1`) in the Interface list. Click on the Start icon on the top toolbar to start capturing network packets. Using the current FTP connection in FileZila re-transfer the image of Bob, overwriting the previous copy. Next, in Wireshark click on the Stop icon on the top toolbar to stop capturing network traffic. Finally, in FileZila GUI click on the disconnect icon to end the FTP session.

Note, the FTP packet traces produced when writing and reading the image of Bob will be significantly larger than the previous FTP `list` example, as FileZila will query the FTP server to find out what operating modes it supports and what files are present in the remote home directory.

Task : examine the FTP packet captures in Wireshark when writing and reading `bob.jpg` to and

from Pi-1. Can you find the packets containing your username and password? Is this data encrypted or sent as plain text i.e. can you see the username and passwords in Wireshark?

Task 3

From a security point of view FTP is a dated protocol, however, it is still a useful protocol to have to hand when moving files across a network. Therefore, to simplify these types of operations there are a range of command line tools to automate these operations using scripts / batch files etc.

To see these types of transfers in action and get some free classic sci-fi books / audio books you can access the lab's FTP server. Connect the Raspberry Pi base node to the lab's network using the **GREEN** cable, as described in lab 1.

Open a VNC session to a Raspberry Pi using `VNC Viewer`.

`-> RealVNC Viewer`

In the VNC remote desktop open the file browser on the **Pi** by clicking on the file browser icon  in the lower toolbar. To mount the lab's FTP server into the Pis file space enter the following path (highlighted in **RED** in figure 8) :

`ftp://192.168.100.4`

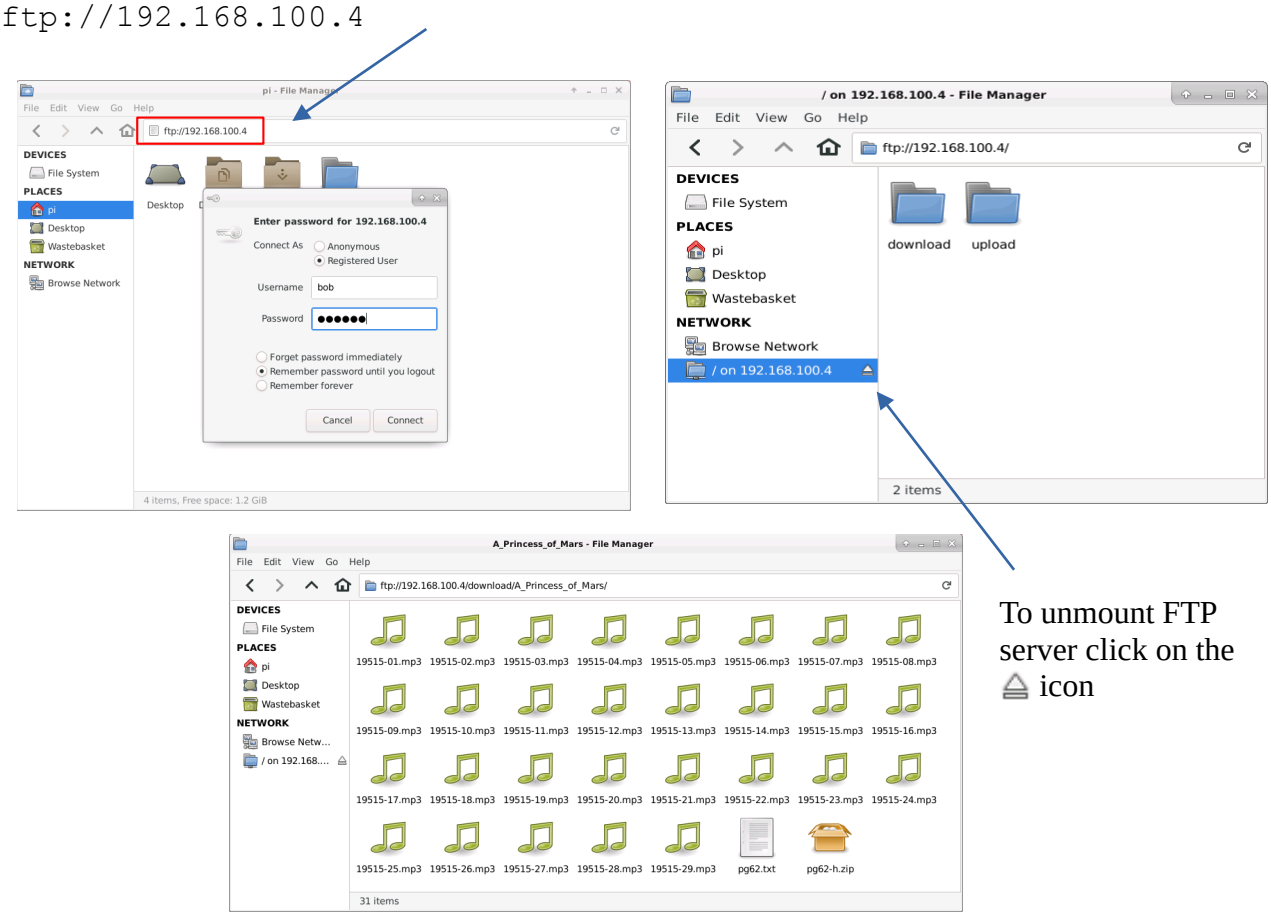

Figure 8 : FTP via file browser.

Note, open this link on the **Pi** not the PC. The Windows File-Explorer does sort of work with FTP, but not very reliably :). To transfer file on the PC use either SCP, WinSCP or Fillzilla.

You will then be prompted for a username and password

```
username : bob
password : asdzxc
```

You will then see the FTP servers top level folders: Download and Upload. Double click on the download folder and you will see a collection of sci-fi classics e.g. A Princess of Mars, or if you prefer your sci-fi Disneyfied: John Carter :).

Uploading and downloading files is now just a simple process of dragging and dropping. When finished, to disconnect from the FTP server i.e. unmount, you need to click on the ⏏ icon.

Note, unmounting the FTP server may take a while to work, for some reason it seems to be issue on the year's Pi image :(. The FTP server's Download folder is read-only, to upload files to the FTP server you have to use the Upload folder :).

Task : can you connect to the lab's FTP server using a file browser on the Pi and PC? Each Pi has its own FTP server. Using these servers can you transfers files between each Pi e.g. from Pi-1 to Pi-2, from Pi-2 to Pi-3 and from Pi-3 to Pi-1?

Two other useful command line tools are `wget` and `curl`. These tools are very useful when you need to batch download files and you don't want to do this manually. Some examples of these command line tools using FTP are soon below i.e. download the file `listing` from the lab server.

```
wget --ftp-user=bob --ftp-password='asdzxc' ftp://192.168.100.4/listing
```

```
curl --user bob:asdzxc ftp://192.168.100.4/listing
```

Note, Firefox discontinued FTP support a few years back. This makes sense from a general public usage, security point of view, but for those who know when and when not to use FTP its a pain :(. `wget` and `curl` can also be used to to access files using HTTP, very handy when scraping data from ftp servers, webpages etc. A Bash script example is soon below:

```
#!/bin/sh
for i in $(seq 1 10)
do
   wget --ftp-user=bob --ftp-password='asdzxc' \
          ftp://192.168.100.4/test/test-$i.txt
done
```

Figure 9 : automatic download script

You can also use HTTP for file transfers, as shown in figure 10. Open a web browser on **Pi-1**, then enter the URL:

```
http://192.168.100.4
```

when prompted use the same username and password as the previous FTP server.

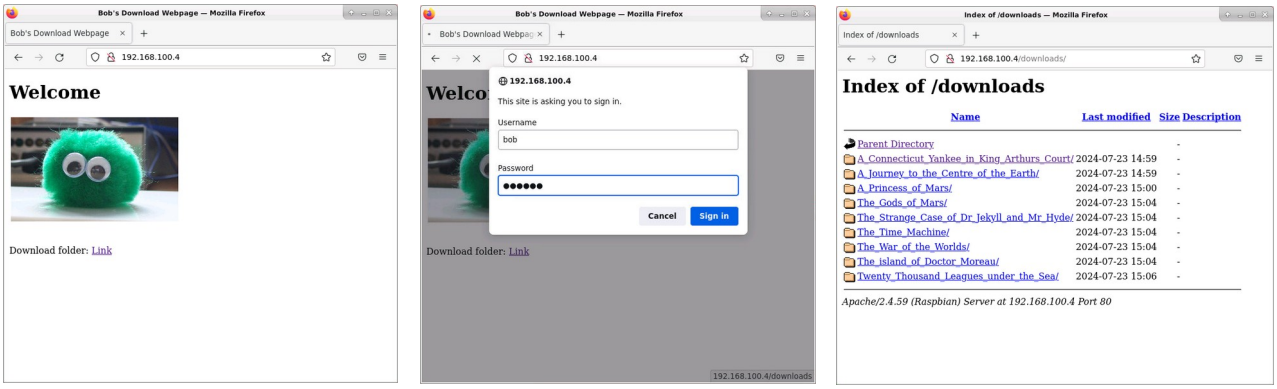

Figure 10 : HTTP file downloads

Task : download a file from the webpage shown in figure 10. Capture this transfer in Wireshark. How was this data transferred?

IMPORTANT : make sure that you can transfer files between the PC and the different Raspberry Pi as you will need to do this in future labs. To perform these transfers you can use `http`, `curl`, `wget`, `scp`, or `ftp` etc. To test your file transfer skills can you transfer an ebook from the labs FTP server to the PC? Can you transfer files between the different Raspberry Pi on the base system?

For more information on how to setup an FTP and other file servers:

http://simplecpudesign.com/networking_file_server/index.html

Task 4

From the previous task you should have seen that during the FTP login process your username and password are sent as plain text i.e. not encrypted. The image data is also not encrypted, so this data will be visible to any routers these packets pass through on the network. Therefore, to state the obvious you should not use FTP on the Internet. This raises the question: how do you perform these types of file transfers safely? There are a number of choices. You can use the encrypted variants of FTP e.g. Secure FTP (SFTP), these create an encrypted channel across which you can send your data.

Note, unencrypted FTP network traffic is not visible to all hosts on the local network e.g. Pi-2 and Pi-3. These hosts will not be able to see the FTP username and password, as these packets are only sent to the FTP server, they are not broadcasted to everyone. To use the SFTP protocol in FileZila you just need to specify port 22 i.e. SSH, when entering your login detail in the top panel.

In addition to using SFTP you can also manually setup up an encrypted channel using the SSH protocol i.e. static port forwarding. To do this we will be using the PCs loopback interface (`lo`).

https://en.wikipedia.org/wiki/Localhost

The loopback network interface is a virtual network interface, an interface that allows you to run a network service without the need for a physical NIC. These virtual interfaces are assigned an IP address in the range 127.0.0.1 to 127.255.255.254, which gives you quite a few to choose from :).

Note, network traffic sent to these interfaces is not transmitted onto the physical network i.e. it will not appear on `eth0` or `eth1`, it is internal to the OS. The loop back interface allows different local processes / application to communicate as if they were connected to their own private network.

To create a encrypted tunnel on the PC click on the start button and select the command prompt:

```
-> Command Prompt
```

In a command prompt create an SSH tunnel by entering any of the following commands:

```
ssh -L 8080:192.168.X.1:80 pi@192.168.X.1       (1)
ssh -L 8080:172.16.X.5:80 pi@192.168.X.1        (2)
ssh -L 8080:127.0.0.1:80 pi@192.168.X.1         (3)
```

when prompted enter the normal password : `12345`. In example (2) above, this command will create an SSH tunnel to Pi-1's second network interface (`eth0`) port 80, as shown in figure 11. Here packets sent to port 8080 on the PC will appear on port 80 with `eth0` IP address.

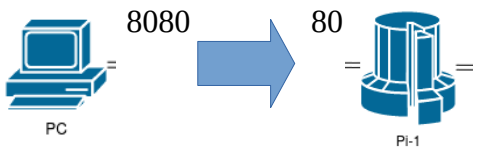

Figure 11 : SSH tunnel

Note, as normal X=desk, the general format of this SSH tunnel command is:

```
ssh -L local_IP:local_port:remote_IP:remote_port user@remote_IP
```

You can connect to this SSH tunnel using a browser on the PC, enter the URL:

```
http://127.0.0.1:8080
```

you will connect to the Pi-1's web server as we did in lab2, however, all data is now encrypted. In example (2) packets will be sent to IP address `172.16.X.5`, as shown in figure 12. To close this tunnel, in the SSH terminal enter the command:

```
exit
```

Note, the browser on the PC can access the opened port 8080 via the loop back interface 127.0.0.1. Packets captured on `eth1` will be sent using the SSH protocol, encrypting the HTTP packets. The unencrypted HTTP packets can be viewed by capturing packets on Pi-1s loopback (`lo`) interface.

Task : capture the encrypted and unencrypted data packets on your system. Examine these packet traces, confirm that no unencrypted HTTP data is sent across `eth1`. Then see if you can capture the raw HTTP packets on the loopback interface. To force the web server on Pi-1 to resend HTML data, within the browser press :

`CTRL+SHIFT+R`

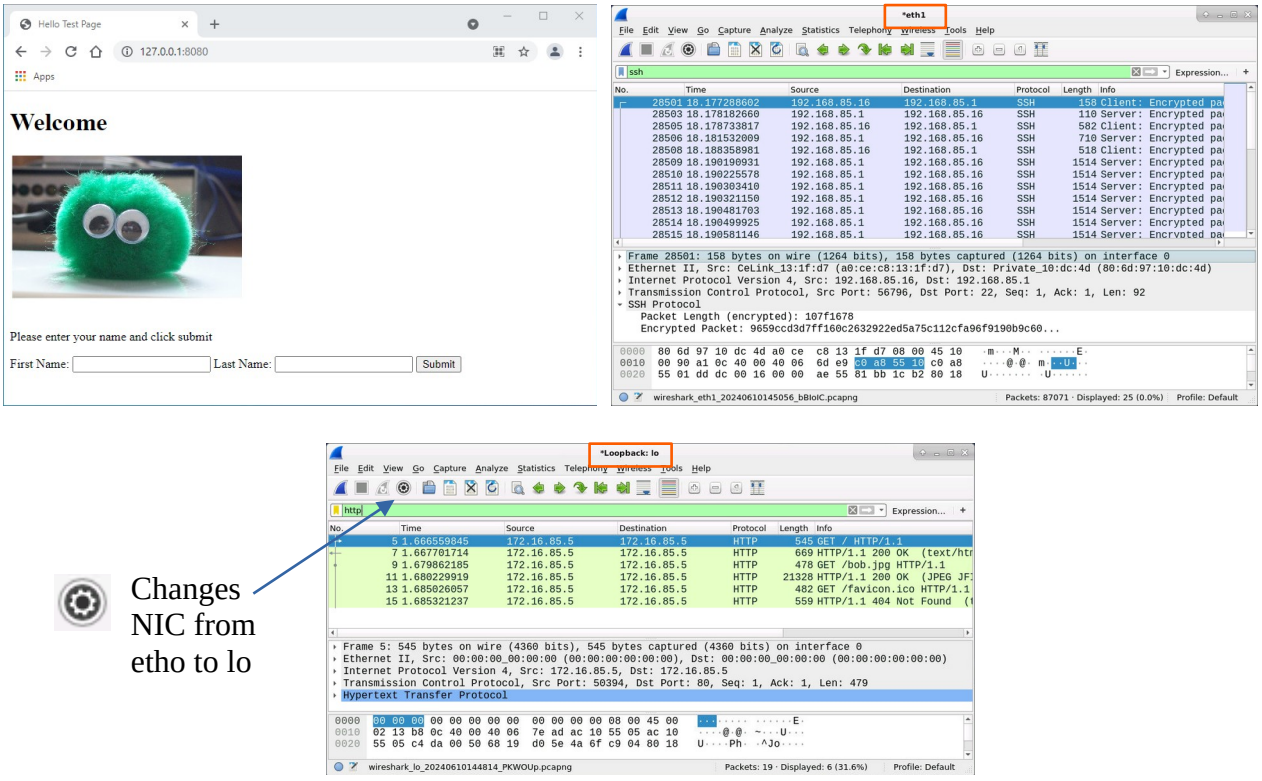

Figure 12 : HTTP over an ssh tunnel

Task 5

The file transfer protocol's (FTP) commands are transmitted across the network using the transport layer protocol : Transmission Control Protocol (TCP). TCP is used for FTP as this protocol ensures data is transferred "correctly" e.g. if packets are corrupted or lost they will be re-transmitted. This functionality does come at a cost of network complexity and increased processing load. TCP is a bit of a complex protocol to start with, therefore, if we assume that the chances of a packet being corrupted or lost on the local network are small we can use a simpler protocol: User Datagram Protocol (UDP), which has a significantly low processing load, but is not fault tolerant. We shall return to TCP in the next lab.

The User Datagram Protocol (UDP) was created in 1983 as a simple, low overhead communications protocol (RFC 768 : https://tools.ietf.org/html/rfc768), replacing TCP for certain non critical functions i.e. low latency, loss tolerant, connectionless applications. The structure of a UDP packet is very simple: Ports, Length and Checksum, as shown in figure 13. UDP uses best-effort delivery service, no handshaking / acknowledgements (ACK), as soon as data is ready it is transmitted i.e. a

fire and forget protocol.

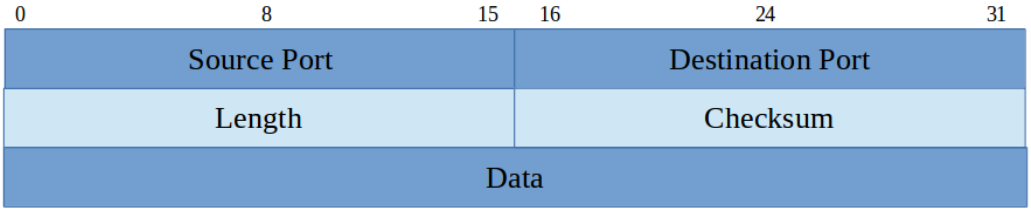

Figure 13 : UDP segment

Note, in RFC documentation UDP packets are referred to as a datagrams, but this name also used to describe network layer packets, so to avoid confusion people commonly refer to “packets” at layer 4 i.e. the transport layer, as segments :).

To show UDP segments in action we can use the `netcat` command: `nc` to transfer the image of Bob between two Raspberry Pi i.e. replicate the FTP functionality in UDP. On the PC click on the start button and select the command prompt.

`-> Command Prompt`

then SSH into Pi-1 by entering the command below using the password (`12345`).

```
ssh pi@192.168.X.1          (X=Box/Desk)
```

Repeat this process, opening a second command prompt on the PC and SSH into Pi-2 by entering the command:

```
ssh pi@192.168.X.2          (X=Box/Desk)
```

Note, to find out more information about `netcat` you can look at its manual page on the Pi by entering the following command in one of the SSH terminals:

```
man nc
```

Use the cursor keys to move through the document and `q` to quit. Within the **Pi-2** SSH terminal enter the following command:

```
nc -u -l -p 8081 > newBob.jpg
```

This is the receiving (**RX**) end of the transfer, where:

- `-u` = udp
- `-l` = listen
- `-p` = port

Note, the symbol “>” indicates redirection i.e. the standard output from the `nc` command is redirected from the screen to a file named `newBob.jpg`. For more information on redirection refer

to this webpage:

https://en.wikipedia.org/wiki/Redirection_(computing)

This program listens for UDP data transmitted to port 8081, saving it to a file. The program that will send this data can again be implemented using `netcat`, using the command:

```
cat bob.jpg | pv | nc -u 192.168.X.2 8081        (X=Box/Desk)
```

This is the transmitting (**TX**) end of the transfer, running on **Pi-1**, sending the data to the receiving (**RX**) program on Pi-2, listening on port 8081, as shown in figure 14.

Note, the file `bob.jpg` should still be present on Pi-1 from the previous tasks. The symbol "`|`" is a pipe, feeding the output of one program into the input of the next. In the above command the file bob.jpg is `cat` into `pv` and then into `nc`. For more information on pipes refer to this webpage:

https://en.wikipedia.org/wiki/Pipeline_(Unix)


```
cat bob.jpg | pv | nc -u 192.168.X.2 8081        nc -u -l -p 8081 > newBob.jpg
```

Figure 14 : NC commands

Task : using these `netcat` commands can you transfer Bob from Pi-1 to Pi-2?

Note, the receiving program needs to be running before the transmitting program starts. For each of the commands `cat`, `pv` and `nc`, you can look at their `man` page to identify what they do.

What you should see is that the transfer occurs i.e. data is written to a file, and pipe view (`pv`) displays transfer statistics. However, `netcat` on Pi-1 and Pi-2 never finish. This is because UDP does not support a mechanism to indicate that a transfer is complete. Examine figure 13, a UDP header only contains port addresses of the client and server, the size of the packet transferred and a checksum to detect errors i.e. no state information. To stop the `netcat` commands in each terminal press:

```
CTRL-C
```

Note, the UDP length field contains the size of the packet, which is a 16bit value, so you could transfer Bob in one packet. However, typically the size of this packet is determined by lower layer protocols e.g. the amount of data a network switch will store. Remember packet switching communications uses store and forward techniques. Switches and routers don't have infinite memory space, so they specify limits on the size of packets they will buffer e.g. typically 1500-ish bytes, otherwise they would run out of memory. We will look at this in more detail in a later lab.

To automatically terminate the `netcat` programs we can add a time-out parameter: `-w<n>`, this will stop these programs after a specified period of inactivity, as shown below:

```
nc -u -l -w2 -p 8081 > newBob.jpg                          (RX)
cat bob.jpg | pv | nc -u -w2 192.168.X.2 8081              (TX)
```

Note, as always the IP address used will vary, but the port can be the same. The time delay is in seconds i.e. for the above commands the `netcat` programs will terminate after 2 seconds of inactivity.

To capture the network packets involved in this UTP transfer, open a VNC session on Pi-1 and start Wireshark as previously described, selecting the network interface (NIC) Ethernet 1 (`eth1`) in the Interface list. In the Wireshark GUI enter the packet protocol filters :

```
udp
```

To start capturing network packets click on the Start icon  on the top toolbar. Ensure that the RX program is running on Pi-2 and then transmit the data from Pi-1.

When the transfer is complete, in Wireshark click on the Stop icon  on the top toolbar to stop capturing network traffic.

Task : can you identify the UDP packets sent to Pi-2? How many UDP packets were sent? What port numbers are used in this transfer i.e. what is port ???? shown in figure 14.

Additional Task

This task talks you through my solutions to the programming task we did last week. You will need to understand how the `udpRX.py` and `udpTX.py` program operate as we will be using them in a later labs.

To transfer an image of Bob using a python program we will be using the application layer network socket API :

https://en.wikipedia.org/wiki/Network_socket

Sockets date back to the early days of Unix, the Berkeley or POSIX socket, typically used for interprocess communications (IPC) using the loopback interface (`127.0.0.1`) . The python code used to transmit and receive this image is shown in Appendix B and C. Below is a short description of the transmit code: `udpTX.py`.

- Lines 1 – 8 : import libs and define constants. Note, for initial testing we will be using the loop-back address to communicate data between two processes running on the PC.
- Line 9 : open socket, `AF_INET` indicates standard internet stack addresses etc and `SOCK_DGRAM` that UDP packets will be used i.e. datagrams.
- Lines 10 – 15 : variables, name of file to read, name of file and size/number of packets sent to receiving program. **Note**, the strange (-(-X//Y)) is a trick to round up to the nearest multiple of Y i.e. // will round down, but if negative will round “up” :).

- Line 16 : send file name used by receiving program i.e. the name of the file to be written to, can be the same as source file if receiving program in a different directory or on a different machine.
- Line 17 : mystery delay :), we will look at this later.
- Line 18 : send the number of packets that will be transmitted, so that the receiver knows when to finish.
- Line 19 : mystery delay :), we will look at this later.
- Lines 20 – 24 : open file to be transmitted.
- Lines 25 – 30 : for loop, transmit packets.
- Lines 31 – 32 : close socket and file.

A short description of the receiver code: `udpRX.py`.

- Lines 1 – 7 : import libs and define constants. Again, for initial test we will be using the loop-back address to communicate data between two processes running on the same machine.
- Line 8 : open socket, `AF_INET` indicates standard internet stack addresses etc and `SOCK_DGRAM` that UDP packets will be used i.e. datagrams.
- Line 9 : tells the socket what IP/Port to listen on
- Lines 11 – 12 : block until packet received i.e. output file name.
- Lines 14 – 15 : block until packet received i.e. number of segments to be transferred.
- Lines 17 – 19 : open output file.
- Lines 21 – 27 : for loop, receive packets and write to file.
- Lines 28 – 31 : close socket and file, display image.
- Lines 33 – 36 : time-out exception, just in case a packet gets lost.

On the PC click on the start button and select the command prompt.

`-> Anaconda`

Make a new subdirectory using the command:

```
mkdir rx
```

Download the python program `udpRX.py` from the VLE and save it to this directory. Next, open a second command prompt, create another subdirectory by entering the command:

```
mkdir tx
```

Download the python program `udpTX.py` and the image `bob.jpg` from the VLE and save them to this directory. In the `rx` directory / prompt run the command:

```
python udpRX.py
```

Then in the `tx` directory/prompt run the command:

```
python udpTX.py
```

If all goes correctly this will transfer the image of Bob across the internal loop-back interface i.e. `127.0.0.1`, from the `tx` subdirectory to the `rx` subdirectory, writing the data to the file `bob-copy.jpg`.

Note, as this transfer was performed on the loopback interface i.e. a virtual NIC, no data will be transmitted on any of the hardware NICs. These transfers are simulated in software, no network hardware will be used.

We can now extend Bob's journey such that he travels out of the PC via Pi-1 as shown in 15.

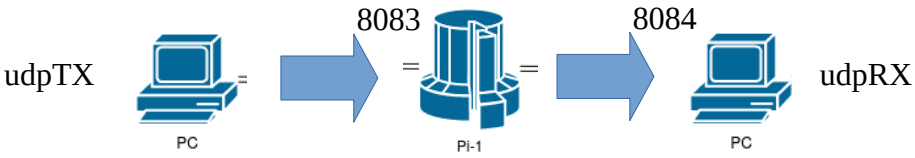

Figure 15 : image transfer route

To automatically transfer the UDP packets across Pi-1 and back to the PC we can use the following command running on Pi-1:

```
nc -u -l -k -p 8083 | pv | nc -u 192.168.X.254 8084
```

SSH into Pi-1 and run the above command. This command combines our previous RX and TX `netcat` commands, piping the output of the receiver into the transmitter, returning the data back to the PC. To connect the python programs to this "channel" modify the python programs running on the PC to connect to these IP addresses / ports:

- `udpTX.py:` `UDP_IP = "192.168.X.1"`
  `UDP_PORT = 8083`
- `udpRX.py:` `UDP_IP = "192.168.X.254"`
  `UDP_PORT = 8084`

IMPORTANT, in the python programs make sure you have replaced the 127.0.0.1 address with the correct 192.168.X.Y address for your desk.

Task : examine these commands and modifications to the python code, do you understand how these will work? Run these commands to transfer an image and examine the network packets using Wireshark.

When you have finished close all open VNC, Telnet and SSH sessions. Then shut down the Raspberry Pi system as described in lab1, unplug and returned to its box.

Appendix A : ftp port calculator

```
import sys

if len(sys.argv) == 1:
  print("Enter FTP pasv string")
  inputString = input()
else:
  inputString = sys.argv[1]

inputString = str(inputString).replace('(','')\
              .replace(')','').replace(' ','').split(',')

ip_addr_0 = inputString[0]
ip_addr_1 = inputString[1]
ip_addr_2 = inputString[2]
ip_addr_3 = inputString[3]

port_high = int(inputString[4])*256
port_low  = int(inputString[5])

outputString = "telnet " + ip_addr_0 + '.' + ip_addr_1 + '.' \
               + ip_addr_2 + '.' + ip_addr_3 \
               + ' ' + str(port_high+port_low)

print( outputString )
```

Figure A1 : FTP port calculator code

Appendix B : UDP transmitter socket

```
1   import socket
2   import os
3   import time
4
5   UDP_IP = "127.0.0.1"
6   UDP_PORT = 8082
7   BUF_SIZE = 1024 * 1
8
9   sockTX = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
10
11  read_file_name="bob.jpg"
12  write_file_name="bob-copy.jpg"
13  file_size = os.path.getsize(read_file_name)
14  number_of_segments = (-(-file_size//BUF_SIZE))
15
16  sockTX.sendto(write_file_name.encode("ascii"), \
                (UDP_IP, UDP_PORT))
17  time.sleep(0.01)
18  sockTX.sendto(str(number_of_segments).encode("ascii"), \
                (UDP_IP, UDP_PORT))
19  time.sleep(0.01)
20
21  print("Sending: " + read_file_name + ":" \
                    + str(number_of_segments))
22
23  f=open(read_file_name,"rb")
24  data = f.read(BUF_SIZE)
25
26  for i in range(0,number_of_segments):
27       sockTX.sendto(data, (UDP_IP, UDP_PORT))
28       time.sleep(0.01)
29       print("tx segment: " + str(i) + " len: " \
                              + str(len(data)))
30
31       data = f.read(BUF_SIZE)
32
33  sockTX.close()
34  f.close()
```

Figure B1 : TX code – udpTX.py

Appendix C : UDP receiver socket

```
1   import socket
2   from PIL import Image
3
4   UDP_IP = "127.0.0.1"
5   UDP_PORT = 8082
6   BUF_SIZE = 1024 * 1
7
8   sockRX = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
9   sockRX.bind((UDP_IP, UDP_PORT))
10
11  data,addr = sockRX.recvfrom(BUF_SIZE)
12  file_name = data.strip()
13
14  data,addr = sockRX.recvfrom(BUF_SIZE)
15  number_of_segments = int(data)
16
17  print("RX File: {0}".format(file_name))
18  print(addr)
19  f = open(file_name,'wb')
20
21  try:
22       for i in range(0,number_of_segments):
23            sockRX.settimeout(2)
24            data,addr = sockRX.recvfrom(BUF_SIZE)
25            f.write(data)
26            print("rx segment: " + str(i))
27       print("Download complete")
28       f.close()
29       sockRX.close()
30       image = Image.open(file_name)
31       image.show()
32
33  except socket.timeout:
34       print("Oops timeout")
35       f.close()
36       sockRX.close()
```

Figure C1 : RX code – udpRX.py