Task 1.1: ICMP Ping

The first task is to recreate the *ping* client discussed in Lecture 2: *Performance: Delay, Loss & Throughput*. Remember that *ping* is a tool used to measure delay and loss in computer networks. It does this by sending messages to another host. Once a message has reached that host, it is sent back to the sender. By measuring the amount of time taken to receive that response, we can determine the delay in the network. Similarly, by tracking the responses returned from our messages, we can determine if any have been lost in the network.

ping traditionally uses Internet Control Message Protocol (ICMP) messages to achieve this behaviour. More details can be found in RFC777. For this task, we will be sending echo request messages (with an ICMP type code of 8). These requests are useful to us because on reaching the client, the client will respond with an echo reply message (with an ICMP type code of 0). By timing the period of time elapsed between sending the request and receiving the reply, we can accurately determine the network delay between the two hosts.

Remember, you are recreating ping without the use of external libraries; they are explicitly prohibited!

Implementation Tips

There are several aspects to consider when writing your implementation. Carefully think about the logic required; use a whiteboard if need be. A *ping* client sends one ICMP echo request message at a time and waits until it receives a response. Measuring the time between sending the message and receiving it will give us the network delay incurred in transit. Repeating this process provides us with a number of delay measurements over time, showing any deviation that may occur.

To assist you in your implementation, we have provided skeleton code for this task. This can be found on the Moodle page. It contains suggested functions, as well as an overview of functionality to be implemented by each. These are given as comments and are to be treated as **guidance only**. Note that you may have to change the parameters passed to each function as you advance with the task. The following Python libraries will also be useful to your implementation:

https://docs.python.org/3.8/library/socket.html

https://docs.python.org/3.8/library/struct.html

https://docs.python.org/3.8/library/time.html

https://docs.python.org/3.8/library/select.html

https://docs.python.org/3.8/library/binascii.html

Note that to run a privileged socket, such as socket.SOCK_RAW, your script must be run with elevated privileges (such as sudo). Note that an alternative solution using unprivileged sockets, such as socket.SOCK_DGRAM, is also acceptable.

We have provided you with a checksum function (included in the skeleton code) which can be freely used in your solutions without penalty. It is important that when passing a packet to this function, the checksum field must contain a dummy value of 0. Once the checksum has been calculated, it can be immediately inserted in the packet to send.

The ICMP header contains both an identifier and a sequence number. These can be used by your application to match an echo request with its corresponding echo reply. It is also worth noting that the data included in an echo request packet will be included in its entirety within the corresponding echo reply. Use these features to your advantage.

Be warned that some servers will reject ICMP requests with an identifier of 0, so

ensure that this is set to a value > 0.

Debugging and Testing

Any host, whether this be a PC, laptop, phone or server, should respond to a message generated by your application. In reality, this is not always the case, as both networks and hosts can choose to disregard these packets, and may do so for a number of reasons (including security). For the purposes of this task, using any well-known server is acceptable. As an example, the following popular sites will respond to an echo request: lancaster.ac.uk, baidu.com, or youkou.com. These will all return with relatively low delays. To rigorously test your application, using servers located further afield will usually return larger delays. For example, the US Department of Education (www.ed.gov) can be queried.

To confirm that the server you have chosen to test with does in fact respond to ICMP echo request messages, feel free to use the existing built-in ping tool to verify reachability.

Once you are sending packets, you can use the Wireshark tool to inspect these. Wireshark will also let you investigate the packets that you receive back. In both cases, it provides a useful method to ensure that these contain the expected information. This is a very useful tool for debugging, especially if you are getting unexpected errors; this will show exactly what is being sent from your script. Wireshark is a free and widely-used network protocol analyzer. Once started, you can capture packets on the eth0 interface (as highlighted in Figure 1). It may also be useful to filter packets to icmp only, using the filter bar found towards the top of the interface (also highlighted in Figure 1).

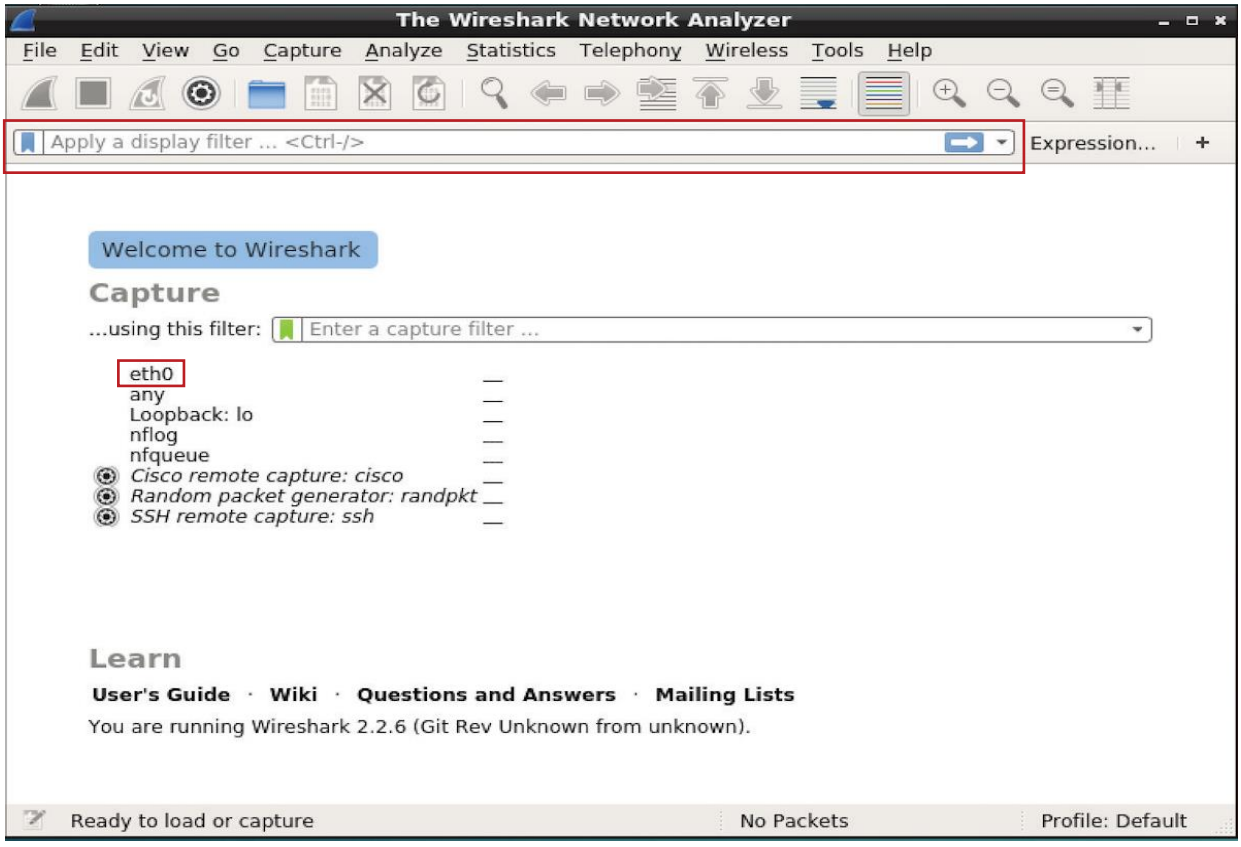

Figure 1: Wireshark Interface

Marking Criteria

You will be awarded the majority of marks for a functioning replica of the **ping** tool. That is, you can successfully send and receive ICMP *echo* messages, timing the delay between. This is then reported in the terminal window. Your application should continue to perform these measurements until stopped.

If you are unsure about the accuracy of the delay measured by your own tool, use the built- in **ping** tool to confirm your results. We're not expecting the results to be perfectly identical (delay changes all the time), but showing that they are close is expected.

Additional marks will be awarded for the following aspects:

- Taking an IP or host name as an argument
- Once stopped, show minimum, average and maximum delay across all measurements
- Configurable measurement count, set using an optional argument
- Configurable timeout, set using an optional argument
- Measuring and reporting packet loss, including unreachable destinations
- Handling different ICMP error codes, such as *Destination Host Unreachable* and *Destination Network Unreachable*

Please note that the features mentioned above are considered supplementary; you do not have to complete them all, and you can still receive a satisfactory mark without completing *any* of them. They are intentionally challenging and designed to stretch you.

Task 1.2: Traceroute

Building on Task 1.1, the second aspect of this task is to recreate the traceroute tool, again in Python. As discussed in Lecture 2: *Performance: Delay, Loss & Throughput*, this is used to measure latency between the host and each hop along the route to a destination. This too uses an ICMP *echo request* message, but with an important modification: the Time To Live (TTL) value is initially set to 1. This ensures that we get a response from the first hop; the network device closest to the host we are running the script on. When the message arrives at this device, the TTL counter is decremented. When it reaches 0 (in this case at the first hop), the message is returned to the client with an ICMP type of 11. This indicates that TTL has been exceeded. As with the previous task, by measuring the time taken to receive this response, delay can be calculated at each hop in the network. This process can be repeated, increasing the TTL each time, until we receive an *echo reply* back (with an ICMP type of 0). This tells us that we have reached the destination, so we can stop the script.

Implementation Tips

As with the previous task, make sure you think carefully about the logic here. Remember you can build upon your Task 1.1 implementation, although you should submit two separate scripts; one for each subtask.

As before, the checksum function included in the skeleton code can be used without penalty.

The same Python documentation as noted in Task 1.1 will be useful for this task too. Of particular note is the socket.setsockopt(*level*, *optname*, *value*) function, which can be used to set the TTL of a socket (and thus the packets leaving it):

https://docs.python.org/3.8/library/socket.html#socket.socket.setsockopt

For this task, we rely on ICMP Type 11 error messages (TTL Exceeded). Unlike Type 0 messages (Echo Reply), these do not contain an identifier field in the response. Any checking you might do in respect to this identifier will therefore fail (as it is not present).

Debugging and Testing

As with the previous task, every host on the path to your chosen destination should respond to your *echo request* message. In reality, these messages are often filtered, including within the lab network. As a result, it is especially difficult to test this tool with a remote host. Instead, it is suggested that you test with a closer endpoint that is reachable: lancaster.ac.uk. Although the number of hops is small (~5), it can still be used to demonstrate the working of your application. If you run your script whilst attached to a different network, such as that at home, your results likely differ. You will also be able to reach external hosts more easily.

The traceroute utility can be used to confirm the results generated by your own application.

- To run traceroute on Windows, open the command prompt window. Enter the command *tracert,* followed by a space, then the domain name or IP address.
- To run traceroute on Mac, launch the Terminal. Enter the command *traceroute*, followed by a space, then the domain name or IP address.

- For Linux, see the Linux man page for more details:

  https://linux.die.net/man/8/traceroute

As with Task 1.1, Wireshark can be used to inspect the packets leaving your application. Comparing these to those created using the traceroute utility will provide you with a meaningful comparison.

Marking Criteria

The majority of marks will be awarded for ensuring that your implementation behaves in a way similar to the traceroute utility. This includes providing delay measurements for each of the nodes between your machine and the chosen remote host. You are expected to increase the TTL of each message, until you reach this final destination.

Additional marks will be awarded for the following aspects:

- Measuring and reporting packet loss, including unreachable destinations
- Repeated measurements for each node
- Configurable timeout, set using an optional argument
- Configurable protocol (UDP or ICMP), set using an optional argument
- Resolve the IP addresses found in the responses to their respective hostnames

As before, please note that the features mentioned above are considered supplementary; you do not have to complete them all, and you can still receive a satisfactory mark without completing *any* of them. They are intentionally challenging and designed to stretch you.