Practical 2 – Network Configuration and Testing
Coursework Weight: 40%

In this practical, you will be building, configuring and testing computer networks. This exercise is designed to give you a better understanding of how and why computer networks work the way that they do, the ways that they can be configured and built, and to give you an appreciation of what goes on behind the scenes when your software application uses this resource.

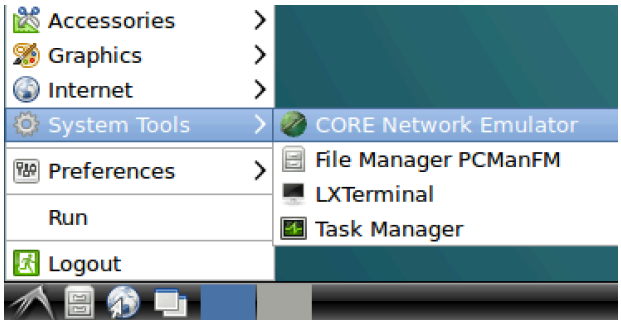
For the purposes of this practical, we will be using the Common Open Research Emulator (CORE) network emulator. This is a software-based network emulation tool, that allows us to design, build and run networks without the need for physical hardware setups. It can also be used to recreate networks far larger than what would be possible in a lab-based environment. It also has an easy to use graphical interface, which makes working with the tool all that much more intuitive. Finally, because CORE runs unmodified protocol stacks, the traffic that traverses our virtual networks is close to that which we would observe in the real world and is considered particularly genuine in this respect.

To get started with CORE, we will be using the same virtual machine image as used in Coursework 1. The image is deployed on all of the B076 lab machines. To use this, open a terminal window, and type:

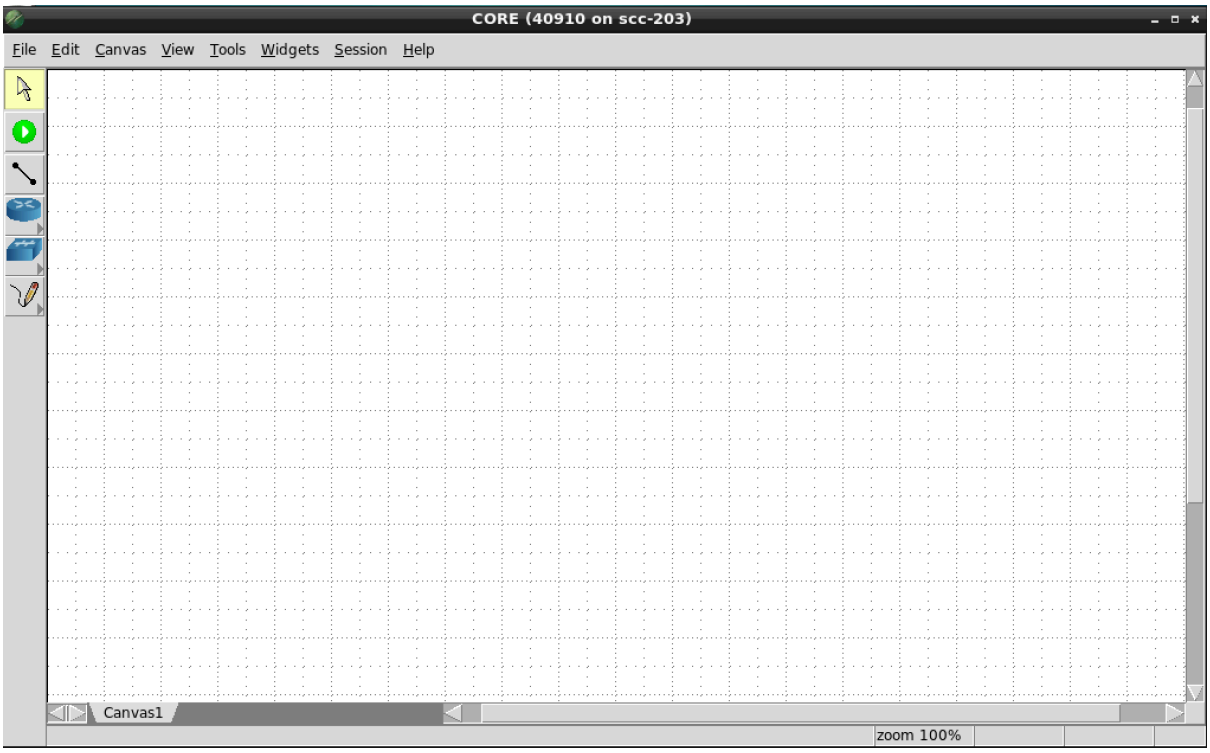
```
svagrant up scc-203
```

The username and password for this machine is: `vagrant`, `vagrant`. This image will map your H: drive into your home directory (as `hdrive`). Be sure to save all of your work here, as the storage on the virtual machine is not persistent. This virtual machine has a graphical environment enabled.

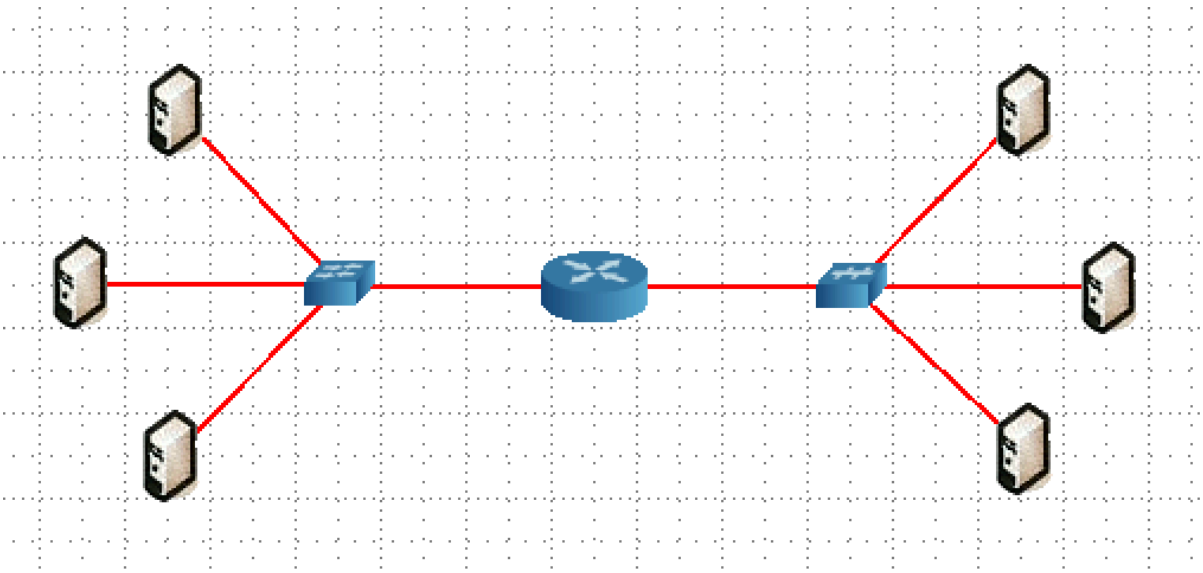
Once this has loaded, you can login with username: `vagrant` and password: `vagrant`. Find the menu icon in the bottom left, and select *System Tools* → *CORE Network Emulator*:



This should start the CORE emulator, which will load with a blank canvas, similar to this:



From here, we can load a CORE scenario file. To do this, click *File* → *Open* and navigate to the file you wish to load. To get you started, we have provided you with such a file on Moodle. Once you have loaded this, your canvas should look like this:



CORE supports a number of different node types, including hubs, switches and routers. These can be connected together using links to build a network. Finally, CORE also supports the creation of hosts, which represent end points in the network (much like the PC or laptop you are using now). Together, these nodes enable the emulation of a full computer network.

The scenario file we have provided contains a wide range of these node types. Using the icons as used in the emulator, each is identified below:

Hub

Router

Switch (note the suitable difference on the top of the icon)

Link

Host

The topology we have provided consists of six hosts, one hub, one switch and one router. Three of the hosts are connected to a switch on one side, whilst the other three hosts are connected to a hub. The switch is then connected to the router, and similarly, the hub is connected to the router on the other side of the network.

CORE also enables the display of various types of information on the canvas. This can be configured using *View* → *Show* and selecting the items of information to be shown. This includes useful information such as IP addresses (which are automatically assigned by the emulator), node labels (used to identify each node) and interface names (useful when a device has multiple interfaces visible, such as the router). For the purposes of this practical we will not be using IPv6 addresses, so these can be disabled here too.

Once you are happy with the setup and information displayed, you can run the emulation using the following *Start* button:

This brings the emulation into execution mode, whereby the hosts, links and networking elements are active. At this point, if setup correctly, the network should be able to send packets, and communication between one or more hosts should be possible. To end the emulation and stop the activity within the network, the *Stop* button can be used:

Now you have a basic introduction to the operation of CORE. In this practical, you will be using this emulator extensively to conduct a number of experiments and tasks. These are detailed in the remainder of this document.

Submission and Assessment

The submission for all work completed in this practical is due by the end of Week 19. Please submit a single document, totalling no more than 1500 words, addressing each of the tasks contained within this document. Make careful use of screenshots to aid your answers in each case. Time should be spent evenly between each task, but as with Practical 1, Tasks 1.1 and 2.1 build upon each other in Tasks 1.2 and 2.2 (respectively). As such, it is recommended that you attempt both tasks.

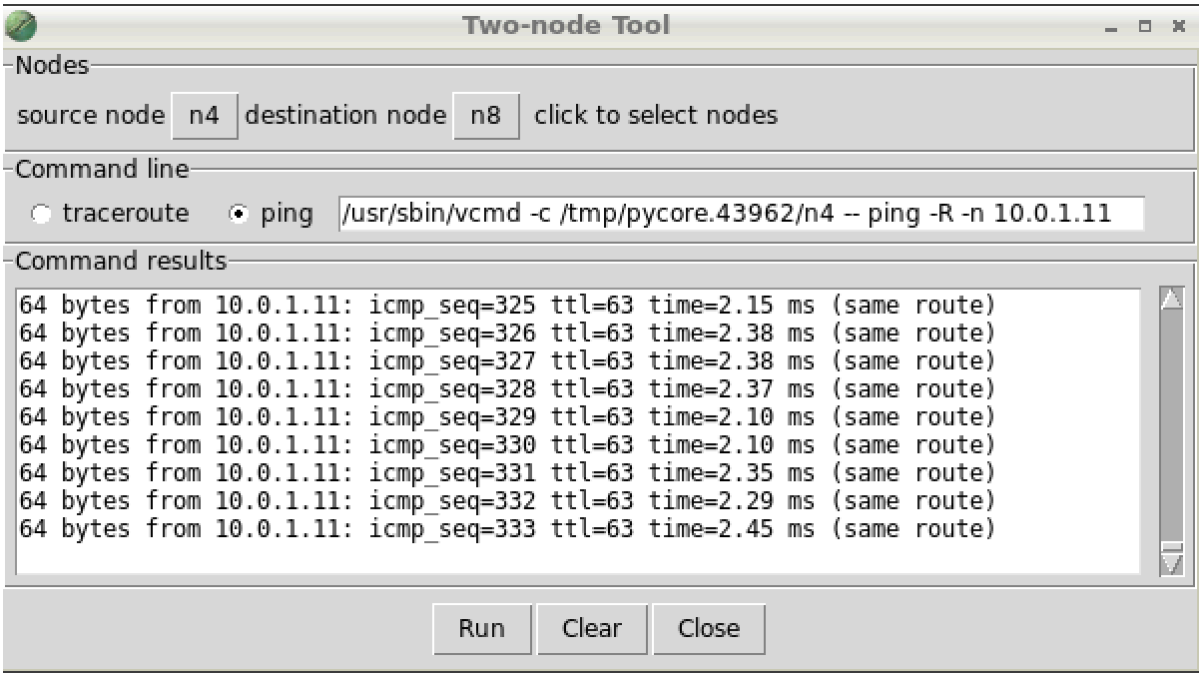
Marking Criteria

In each case, marks will be awarded for satisfactorily demonstrating that you have answered the questions contained within each task. The answers to the questions posed within should be well evidenced use carefully selected screenshots to enhance the written answer. A small number of additional marks will be awarded for overall structure and clarity within the document.

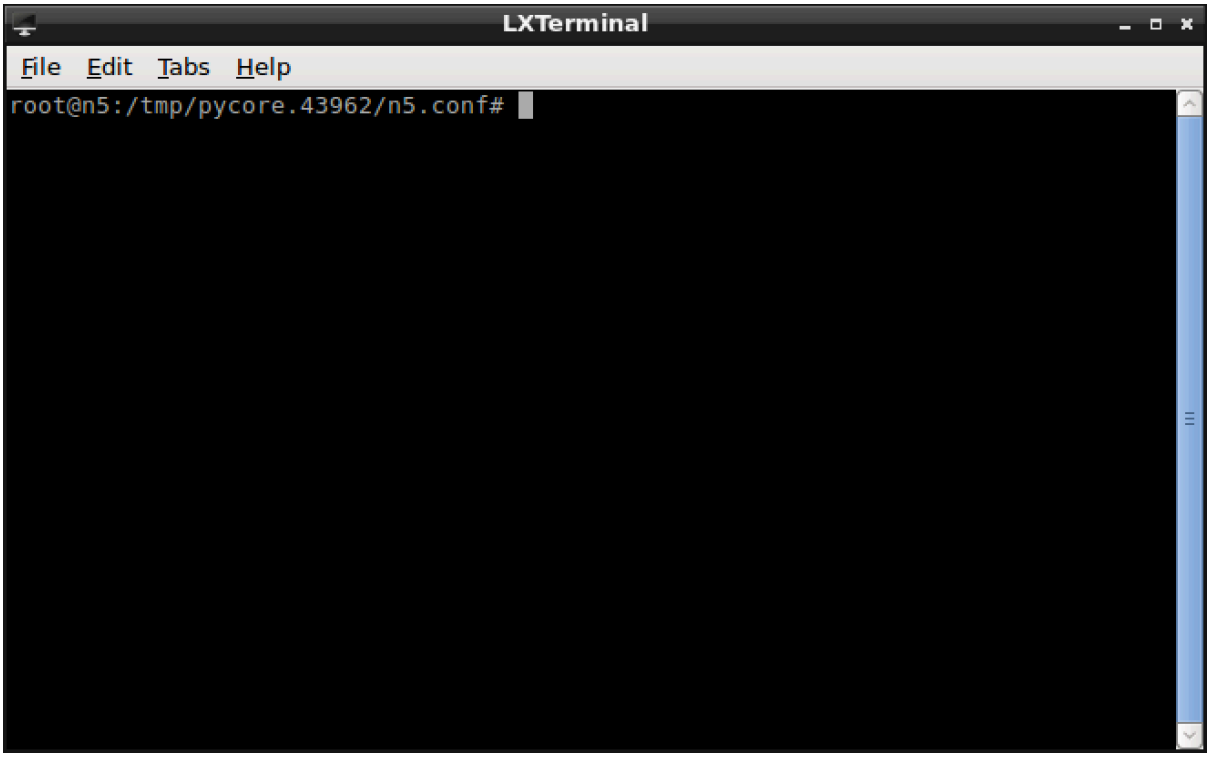
Task 1.1: Network Measurement

In Practical 1, you developed a number of Python-based tools. These replicated functionalities available in common Unix utilities, such as `ping` and `traceroute`. Furthermore, you have seen how tools such as `iperf` can be used to measure throughput between two hosts. CORE allows us to easily use these tools and provides us with a handy window for achieving this. First, click the *Two-node Tool* icon:

This should open a window which enables us to define two nodes in which we can run either a `ping` or `traceroute`. In each case, the results are also shown when the *Run* button is clicked:



Furthermore, CORE also allows us to create terminal windows, which give us access to each host. With the emulation in execution, double-click a host to open its terminal window, which should look something like this:



This allows us to run other tools, such as `iperf3`, as demonstrated in Lecture 3. Check back to the slide material for details on its use. If you prefer, `ping` and `traceroute` can be run from here too.

Using these tools, you are to measure the delay, loss and throughput in the following scenarios: a) between the two hosts connected to the switch, b) two hosts connected to the hub and c) two hosts across the network (using the router). In the case of delay, be clear why the round-trip time is measured as such (paying attention to the delay on each link). For the bandwidth measurement, highlight where the likely bottleneck is between the two hosts.

Task 1.2: Link-layer Transport

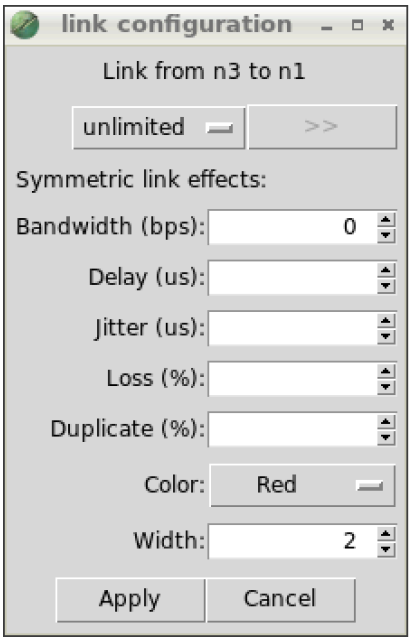
Hubs were once fundamental components of computer networks. However, they have fallen out of use in recent years. Using the sample topology given to you, explain the differences in behaviour between a network switch and a network hub. More details can also be found in the course textbook, Chapter 6.4. Evidence these differences using tools such as `ping` (to generate traffic) and Wireshark (to observe the packets being sent and received). Include any screenshots to help in your explanation. Remember that three hosts are attached to a switch, whilst three are attached to a hub.

Task 2.1: Building a Bigger Network

For the second task, you will be extending the existing network. Using the tools in the emulator, add additional network components to your topology. The following should be added:

- 4 additional routers (making 5 in total) in a full mesh configuration; that is, each router is connected to every other router
- A small network attached to each of these new routers. These should each contain three hosts attached to one switch. Each switch is then attached to one router. These small networks will be similar to those already given in the example topology.

Once you have established this topology, the link characteristics can be changed by clicking on each link, bringing up the following window:



From here, the bandwidth, delay and loss can be individually set. For the purposes of this task, these values should be set as follows:

- For links *between* routers, bandwidth should be set to 1gbps, delay to 5ms and loss to 0.1%
- For links between the switches and the routers, bandwidth should be set to 100mbps, delay to 15ms and loss to 5%
- For links between the hosts and switches, bandwidth should be set to 10mbps, delay to 2ms and loss to 1%

Note that the units given here do not match those required in the CORE GUI; you will have to convert them as appropriate.

As with Task 1.1, using all the tools available to you, demonstrate that you have created the above topology, that it is connected correctly, and that the link characteristics have been set correctly in each case. You may have to run multiple measurements across different network segments and between different hosts to demonstrate this.

Task 2.2: Configuring Routing

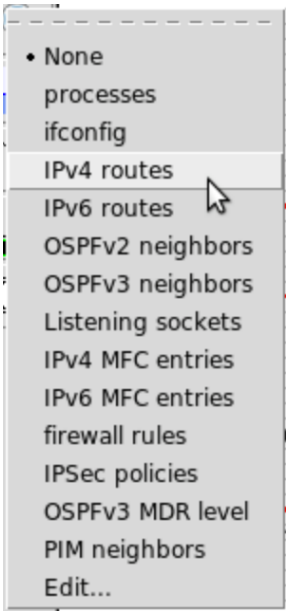
The final task involves manipulating the routing configuration of the routers contained within the topology. This is done to demonstrate how routing works, and how routes can be manipulated to ensure traffic flows through different paths.

A router makes forwarding decisions based upon information that it holds about how to reach certain destinations. This decision is determined by the IP address of an arriving packet. The information is held in a table, which contains a number of entries, each representing another segment of the network, and the accompanying port through which the packet should be sent to reach its destination. This information is spread between routers using standard protocols. In this practical, we will be focusing in OSPF.

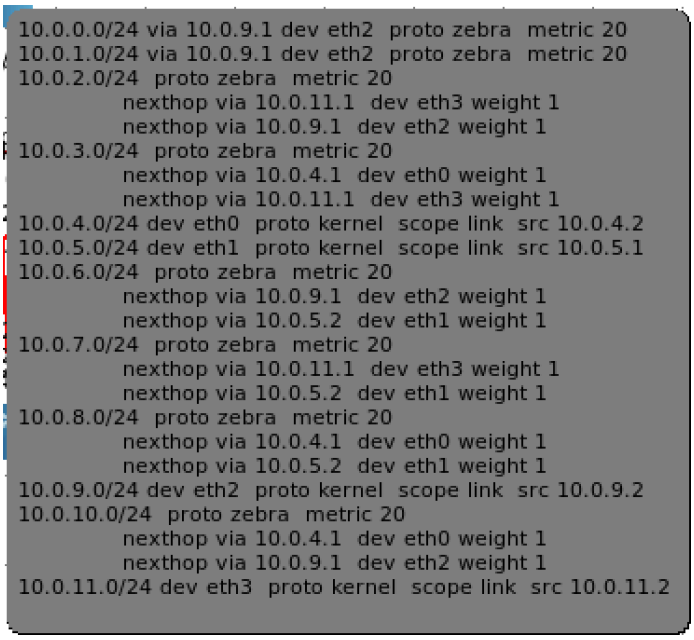
Open Shortest Path First (OSPF) is a routing protocol used internally in large networks; that is, within a single Autonomous System (AS). As an implementation, it monitors the network for changes, including the availability of routers. After such a change, it recalculates the routing structure very quickly. OSPF computes the shortest path to a destination based upon Dijkstra's algorithm. As such, each link has an associated *cost* metric, which is an arbitrary value used to determine this optimal path. For more information, please see Chapter 5.3 of the course text book.

Link metrics are by default set to 10. Given that each link is equal in cost, this means that traffic will always be routed by the shortest path. This is because the shortest path will also have the lowest cost. With the simulation running, it is possible to view the contents of these routing tables by selecting the *Observer Widgets Tool:*

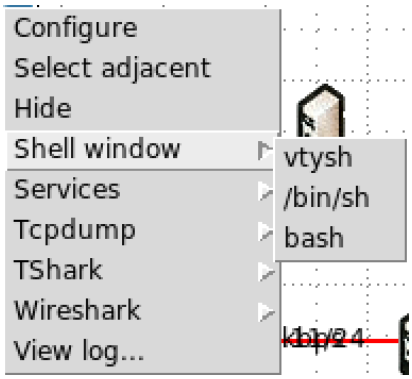
And then selecting *IPv4 routes*:



Now, by hovering over each router, you should be able to see the contents of the routing table, which should look something like this (note the *metric* value):



Within the emulator, it is also possible to change the value of these metrics. To do this, we need to open a terminal for the router. Right-click on one of the router elements and select *Shell window* → *vtysh*:



This should open up an interface, subtly different to those opened previously on the host machines; this terminal interacts directly with the routing daemon running on the node. To change the link cost, we first need to enter the configuration mode:

```
n1# configure terminal
```

Then we need to select the interface on which we want to change the cost.

```
n1(config)# interface eth2
```

Then we can go ahead and change the cost, in this case, keeping the cost at 10.

```
n1(config-if)# ospf cost 10
```

It is worth noting that whilst in this mode, the terminal may give you warning that it is not fully functional. This does not impact this practical assignment and can be escaped using the 'q' character.

For this task, your goal is to manipulate these *cost* values to steer traffic particular ways through your network. In this case, we want you to steer traffic around the maximum number of hops before reaching its destination; that is, traffic from one host (of your choice) travelling towards another host (of your choice, but at least attached to another router) should traverse every other router before arriving there.

To demonstrate that you have established the costs appropriately and that traffic is routing accordingly, we would like you to show a before and after of your topology in action. This includes generating traffic between hosts to demonstrate that the paths have been changed. Furthermore, using screenshots from Wireshark, illustrate how the changes in link cost are propagated between routers using OSPF messages.