COSC 264 Problem Set
IPv4 Networking

Andreas Willig
andreas.willig@canterbury.ac.nz

July 13, 2020

---

1 Introduction

The goal of this problem sheet is that you gain some first experience with some basic Linux networking tools and make your first steps into IPv4 routing by setting up static routes for a given network.

This sheet is designed to be worked on in the first two weeks of term four. Please note that some work needs to be done **in advance** of the labs, e.g. familiarizing yourself with the `quagga` manual, see below.[1]

You will perform the IPv4 experiments in a set of virtual machines called `alpha`, `beta`, `gamma`, `christchurch`, `hamilton`, `dunedin`, `wellington` and `auckland`. All these machines run the Ubuntu Linux operating system. They are connected to multiple networks. The pre-configured login name and passwords for all the virtual machines are:

- login name: `student`
- password: `TipTapToe`

1.1 Lab setup

You will create the host-only networks at first and then import the VMs. Here is a step-by-step guide for setting up your work environment.

[1] This does not mean you should read the manual cover to cover, just enough to understand the major capabilities of the software and to be able to find the right place in the documentation if you have specific questions.

- Open a shell, start the VirtualBox

  ```
  $ virtualbox
  ```

- Make sure there is no existing host-only network
  - Choose menu `File/Preferences`
  - Click on `Network` in the left part of the open dialog
  - Click on `Host-only Networks`
  - Remove the existing host-only network if there is any
- Open another shell, create the host-only networks by running the following script

  ```
  $ /netfs/share/bin/create_vbox_vnets
  ```

- In the same shell, now import the VMs by running the following scripts one by one

  ```
  $ /netfs/share/bin/cosc264vm-alpha
  $ /netfs/share/bin/cosc264vm-auckland
  $ /netfs/share/bin/cosc264vm-beta
  $ /netfs/share/bin/cosc264vm-christchurch
  $ /netfs/share/bin/cosc264vm-dunedin
  $ /netfs/share/bin/cosc264vm-gamma
  $ /netfs/share/bin/cosc264vm-hamilton
  $ /netfs/share/bin/cosc264vm-wellington
  ```

- Start all VMs.
- Log into all VMs as user `student` and type the valid password. After you have logged in, please enter the command

  ```
  /sbin/ifconfig
  ```

  which displays the list of the available interfaces. The interface names are printed on the left side. You will very likely see an interface named `lo`, which is the so-called **loopback interface**. You should also see at least one (and on the routers two or three) interfaces named `enpxsx` (like `enp0s3`, `enp0s8` etc), which refer to Ethernet interfaces. Here, `en` stands for Ethernet which followed by a `p` for PCI slot and `s` for hotplug PCI-E slot. This naming scheme incorporates physical/-geographical location of the connector of the hardware which makes the interface names predictable.

  Note: the traditional interface naming scheme is to assign names beginning with `eth0`, `eth1`, ... for Ethernet interfaces based on the probes by the driver. Therefore, it is not predictable as `eth0` on one boot may become `eth1` on the next boot.

Now you are good to go.

2 IPv4 Networking and Routing

The goal of this set of problems is to familiarize you with the operation and behaviour of IP routing and forwarding under the Linux operating system. You will use the `quagga` routing suite (see `www.quagga.net`), which is a key component of a range of commercial Linux-based router products.[2] You will find a pdf version of the `quagga` users manual on the learn website of this course.

Again, you will perform the lab work entirely under the Linux operating system. However, similar to commercial Linux-based routers, you will not have a graphical user interface at your disposal, but you will have to work on the Linux command line.

You will work on a test network that consists of eight machines: three end hosts named `alpha`, `beta` and `gamma`, and five routers, named `christchurch`, `auckland`, `hamilton`, `dunedin` and `wellington`. These machines are connected through networks. All these machines run as virtual machines under `Oracle VM VirtualBox`, in addition to that `VirtualBox` also emulates the Ethernet networks by which the machines are connected. Each machine runs an instance of the Ubuntu 16.04 Server LTS operating system which includes the relevant tools. Specifically, the routers have installed the `quagga` software suite, the end hosts have not.

The test network does not have a DNS service configured. This implies that you will have to identify all machines / interfaces by their IP addresses. Furthermore, the test network does not have access to the real Internet.

The preparation problems given in this section have the goal to familiarize you with the set of tools that we will use in the routing lab. It is important that you work through this section (both reading it and doing the exercises) **before** the actual lab, since otherwise you will risk to have insufficient time to get everything done!! However, you should not expect that the notes below will contain *everything* you need – in fact, you are expected to figure out several details on your own.

2.1 Elementary Unix/Linux networking tools

You will need to familiarize yourself with a number of basic tools for diagnosing, printing and configuring network-related information under Unix/Linux. There are four popular tools that every network engineer should be familiar with: `ping`, `ifconfig`, `route` and `traceroute`.

The `ifconfig` tool can be used to configure network interfaces and to print information related to these interfaces. Here we restrict to looking at information printed by this tool. In the user account pre-configured for this lab you may find that the shell just prints an error message when you enter the command `ifconfig`. This is because the

[2] It is important to note, though, that many of the Internet routers, especially on the backbone, use proprietary hardware architectures and operating systems.

`ifconfig` executable is not stored in one of the directories which are included in a user's search path.[3] Instead you have to enter the command

```
/sbin/ifconfig
```

to see the existing interfaces and their main properties. In contrast, the `ping` and `traceroute` tools should be accessible without giving the `/sbin/` prefix.

The `traceroute` tool displays a list of intermediate routers between your host and the final destination. Nowadays `traceroute` often fails to do this properly since many institutions run firewalls blocking the packets important for `traceroute` (the same is also true for `ping`). However, `traceroute` will be available in your test network.

The `route` tool displays the current contents of the forwarding table and also allows to add or delete routes from the command line.

Problem 2.1 (Review problem)**.**

- Read the man page for `ifconfig`. Which information is included in the output?
- Read the man page for `ping`.
- Read the man page for `traceroute`.
- How is `ping` implemented, i.e. which IP facilities or protocol features does `ping` use? What do the related IP datagrams look like?
- How is `traceroute` implemented? Describe the key approach of its implementation.
- What does a `traceroute` user assume about the displayed route when using this tool? Is this assumption always true? Please explain.
- Read the man page for `arp`.
- Read the man page for `route`.

Solution 2.1.

- `traceroute`:

[3] The search path is a list of directories in which the shell looks for executables when you submit a command. You can see the list of directories in which the shell searches for an executable by giving the command `printenv PATH`. Actually, `PATH` is an environment variable and can be changed. Consult a tutorial for the `bash` shell and look for the commands `setenv` and `export`.

  - Implementation: `traceroute` is called with a destination IP address or hostname (in the latter case `traceroute` then performs DNS lookup). It sends an IP packet to the destination with a TTL of 1. The first router decrements TTL to zero, notices it is not the destination and returns an ICMP error packet ("time exceeded"), which includes his (the routers) own IP address. Having received the ICMP packet, the sender sends the next packet with TTL=2 and so on.
  - Assumption: it is implicitly assumed that all packets between source and destination follow the same path, which of course is not true in general. Reasons could include path changes after routing / weight updates, usage of load-balancing router pairs.

- `ping` implementation: sender sends an ICMP echo request packet to destination. This ICMP request is embedded into an IP datagram. The destination sends an ICMP echo reply datagram back to the sender. This generally allows to check whether routing between source and destination works.

2.2 The `quagga` routing software

The `quagga` routing package (see `www.quagga.net`) provides a basic framework for IP routing plus several routing demons for individual dynamic routing protocols, e.g. RIPv2 or OSPF. However, `quagga` also allows you to do static routing. The `quagga` routing package is pre-installed on all the router virtual machines (i.e. `christchurch`, `auckland`, etc.) but is not available on the end hosts `alpha`, `beta`, and `gamma`.

Generally speaking, when an IP router has decided to forward an IP datagram, it consults its **forwarding table**. The forwarding table contains several entries, one entry per known IP network (also called an **IP prefix**). An IP prefix here refers to a combination of an IP network address and netmask. For each IP prefix the forwarding table stores the outgoing interface (on a linux-based router e.g. `enp0s3`) and the IP address of the next-hop router, which must be reachable through a directly attached subnetwork (i.e. over the outgoing interface). The forwarding table is all that the router (or the Linux kernel) needs for forwarding IP datagrams. There are fundamentally two different ways of **populating** the forwarding table: static routing and dynamic routing.

With static routing the IP forwarding table is configured manually (by editing configuration files). In contrast, with dynamic routing, on all routers a dedicated piece of software is running, a so-called **routing demon**. Such a routing demon implements a dynamic routing protocol (e.g. RIP, OSPF) which communicates with neighbored routers, exchanges reachability information, and, most importantly, automatically updates the forwarding table used for IP forwarding.

The `quagga` routing package contains first the `zebra` routing demon, which is the sole manager of the Linux kernels forwarding table. On top of that, `quagga` can run one or more of the routing demons that are part of the package, for example the RIP demon, the OSPF demon and so forth. These routing demons modify the kernels forwarding table through the services offered by the `zebra` demon. In addition, the `quagga` routing package comes along with the `vtysh` command, which allows a user to "log in" to the routing package and to issue commands to the `zebra` demon and any routing demon in a command language that resembles the language of CISCO routers running the CISCO IOS ("Internet Operating System").

We will not give a comprehensive tutorial about the `quagga` software. You find a `quagga` manual under `www.quagga.net`, following the documentation link (the manual has also been placed on the learn platform). Furthermore, Google shows up `quagga` tutorials. Nonetheless, in the following you can find some important bits and pieces about the work with the `quagga` software suite:

- You can find all `quagga` configuration files in the directory

  ```
  /etc/quagga
  ```

  To get into this directory you can use the `cd` (change directory) command, thus entering

  ```
  cd /etc/quagga
  ```

  in the command line. The configuration files are all stored in ASCII format, so you can edit them with `vi` (it is thus a great luck that you have already learned about `vi` on the very first problem sheet). When logged in as user `student` you can only read the configuration files but not modify them. If you want to modify a configuration file (e.g. the file `daemons`) you can use the command `sudo vi daemons`.

- Each routing daemon (including `zebra`) insists on finding its own configuration file in the directory `/etc/quagga`. For example, the `zebra` daemon expects a `zebra.conf` file to be present (even if it is empty), similarly the ripd (RIP daemon) expects file `ripd.conf`. To create an empty `zebra.conf` file you can give the command

  ```
  touch zebra.conf
  ```

  These files must have owner and group `quagga` and permissions `644`, which you can achieve with the commands

  ```
  sudo chown quagga:quagga filename
  sudo chmod 644 filename
  ```

  where of course `filename` is the name of the file you want to work with.

- Whenever you have modified any `quagga` configuration file you have to restart the

overall `quagga` service (which is: all currently active `quagga` daemons). To do this, enter the command

```
sudo /etc/init.d/quagga restart
```

This command stops all currently running daemons and starts them afresh. Each daemon reads its configuration file after it started. When a daemon notices a syntax error in its configuration file, the startup is stopped.

The `quagga` software suite offers an own command-line interface to the running daemons, the `vtysh` program.

To simplify our work in the next steps, you can enter the command `sudo -s -H`. This gives you a **root shell** (which you can recognize by the changed prompt). **BEWARE:** This is not for the faint-hearted, you have now the absolute power over the system, there is nothing that prevents you from destroying or misconfiguring it. Be careful, especially when you find yourself typing `rm` on the command line – `rm` stands for "remove" and it deletes files without any possibility of undeleting them. If you want to play it safe, don't perform this step but then prefix all the following commands with `sudo`.

Problem 2.2 (Review problem)**.**

- Please familiarize yourself with the `quagga` users manual. You do not have to read it cover to cover, but you should know it well enough to find relevant pages quickly.

Solution 2.2.

2.3 Discovering network topology

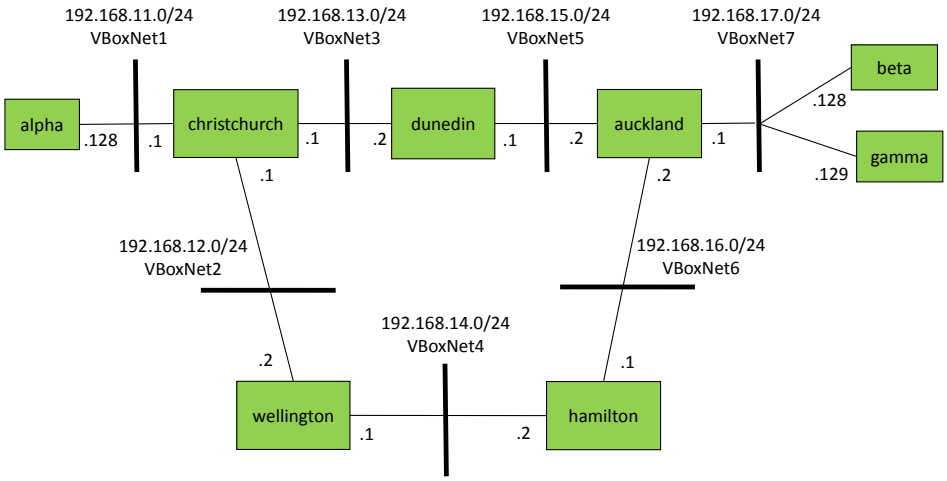
The goal of the first exercise is to let you use the above introduced networking tools to explore the topology of our test network. It is assumed that at the beginning of these exercises you have obtained a fresh copy of the virtual machines and have not started (or even modified) them before. If this is not true, then some of the exercises below might not work out as intended.

Problem 2.3 (Lab problem)**.**

- Switch on all virtual machines (hosts and routers), wait until the last machine has booted before making the next step
- Use `ifconfig` in each machine to discover the topology of the network:
  - How do you use IP addresses and netmasks in this process?
  - Draw a diagram of the topology that shows all subnetworks, all hosts and routers, which interface of each host is attached to which subnetwork, and what the IP address of this interface is.
- Select three different subnetworks and two nodes on either subnetwork. Use the `ping` command on one of these hosts and use the other host as target.
  - `ping` prints for each packet it sends the round-trip time (denoted simply as `time` in the output). You will very likely notice that the first time is larger than the subsequent times. Why? And how can you repeat this experiment and get the same effect without re-starting the machine?
- Log into host `alpha` and ping the IP address of host `gamma`. What happens? And why?
- Shut down all virtual machines. To do this, use from within the machine the command `sudo shutdown -P now`.

Solution 2.3.

- Discovery: when you use `ifconfig` you can see the IP address and the netmask assigned to a host. From this you can compute the subnet address. Simply collect all subnets. The resulting figure could look like this:



- `ping` exercise:
  - higher first packet delay for `ping`: related to ARP request (happens for both default gateway and any other host in the same network). To re-create the same behaviour: clear the ARP cache, using command `arp -d`
- host `alpha` pinging host `gamma`: you get a "network is unreachable" error message. This generally means that an IP router or host has no route for a particular destination network in its forwarding table. In this particular case the problem resides with host `alpha` itself, since `alpha` does not yet have a default route entry.

2.4 Configuring static routing

In the second lab experiment you will learn how to configure static routing in the given network.

Problem 2.4 (Lab problem)**.**

- Re-boot all virtual machines so that they start "empty"
- First consider the three end hosts `alpha`, `beta` and `gamma`:
  - Use the `route` command to display the forwarding table
  - What can you see? And what is missing? How do you configure the missing route(s)? (Hint: it is related to default gateway)
  - If you have configured the missing route(s) on `alpha`, how does the output of a `ping` command towards `gamma` change (compare Problem 2.3)? What happens now and why? Which IP protocol mechanism is involved here? Perform the same configuration step on `beta` and `gamma`.
  - Compare the output of the previous question with the output you get when you `ping` from `alpha` to an IP address that lies in the same subnet as `alpha` but for which there is no host? What happens?
- Continue with the next problem without re-starting the hosts or routers.

Solution 2.4.

- The `route` command on `alpha` shows only one entry. `alpha`'s IP address is perhaps `192.168.11.128` and the entry displayed by `route` is to destination network `192.168.11.0`. This is the home subnetwork for host `alpha`, which is marked as a directly attached subnetwork by the asterisk character. Both the own IP address and the netmask (which is needed for determining the own network address) are fetched from the file `/etc/network/interfaces`. What is missing: the entry for the default gateway. The default gateway can be configured as follows:
  `sudo route add default gw 192.168.11.1 enp0s3`.
  This adds a default router for interface `enp0s3` and also specifies the IP address of this router. The IP address here has to be `christchurchs` IP address on the `192.168.11.0` network. However, this information is lost after a reboot. To make this change permanent, one needs to edit `/etc/network/interfaces`, go to the entry for `enp0s3` and add the following line after the `address` line:
  `gateway 192.168.11.1`
  (use the same indentation as for the previous `address` line). Don't forget to run the commands `ifdown enp0s3` followed by `ifup enp0s3` afterwards to re-initialize the interface after changing the file. The command `ifdown enp0s3` might give a cryptic error message, you can ignore that.
- When host `gamma` is pinged from host `alpha`, the `ping` tool outputs nothing. Indeed, host `alpha` has sent the IP datagram to `gamma` to its default router, which

is router `christchurch`. Router `christchurch`, however, does not have entry for the destination address / network of `gamma` in the forwarding table and also does not have IP forwarding enabled, thus ignores the packet from `alpha`.

- When `alpha` pings an IP address from its own subnetwork but of a non-existent host (for example `192.168.11.47`), it outputs after some time the message "destination host unreachable". Since the chosen destination is in the same subnet, `alpha` tries a local delivery. At the beginning of this, `alpha` will issue an ARP request, which fails (after some re-tries). As a result, the local IP instance cannot deliver.

Problem 2.5 (Lab problem)**.** After we have successfully configured the end hosts, we turn our attention to the routers.

- Log into router `christchurch` and display the forwarding table. What do you see?
- How to enable IP forwarding on a router? Make sure that IP forwarding is enabled on all routers. What is the meaning of this step?
- For all routers do the following:
  - Change into the directory with all `quagga` configuration files: `cd /etc/quagga`. Edit the file `daemons` to enable just the `zebra` daemon and no other daemon. Do not forget to restart the `quagga` daemons (by using the command `/etc/init.d/quagga restart`). In the following we will not remind you anymore about this.
  - Make a backup of the `zebra.conf` configuration file before you configure static routing, call it for example `zebra-empty.conf`
  - After you have convinced yourself that the `zebra` demon runs properly, please modify its configuration file and add static routes to **all** known networks (except directly attached networks). Ensure that always the smallest number of hops is taken and please do not use default routes.
  - Make backup copies of your edited `zebra` configuration files.
- Test your setup with `ping` and `traceroute` for full connectivity.
- Stop routing on router `dunedin`. What does this mean for data transfer between `alpha` and `gamma`?

Solution 2.5.

- You only see entries for the three directly attached networks `.11.0`, `.12.0` and `.13.0`, since routing has not been configured yet.
- The IP forwarding flag tells whether a router should forward foreign IP packets or not. In the latter case it would not act as a router (see the diagram on IP packet processing in the slideset "IP and related protocols"). To check whether IP forwarding is enabled one could issue the command `sysctl net.ipv4.ip_forward`. When the answer is 0, forwarding is not enabled. An alternative command is `cat /proc/sys/net/ipv4/ip_forward`. To enable IP forwarding dynamically you could give the command `sysctl -w net.ipv4.ip_forward=1`. To enable it permanently on a router, you can edit the file `/etc/sysctl.conf` by adding a line saying `net.ipv4.ip_forward = 1`. Actually, the `sysctl.conf` provided by Ubuntu already contains such a line, which needs be uncommented. After editing the sysctl.conf file one either restarts the machine or restarts the sysctl demon by saying `sysctl -p /etc/sysctl.conf`.
  In addition, now that ip forwarding is enabled if we ping from host `alpha` to host `gamma`, the ping tool will output an error message of the form "from 192.168.11.1: ICMP seq = xx, Destination Net Unreachable". Indeed, host `alpha` sent the IP datagram destined for `gamma` to its default router, which is Christchurch. However, Christchurch does not yet have a forwarding table entry for the destination address / network of `gamma` and thus returns an ICMP error message that indicates the destination net unreachable.
- An example `zebra.conf` for router `christchurch` is shown below.
- After router `dunedin` has been stopped, routing between `alpha` and, say, `gamma` should not work any longer (unless you have **not** configured the shortest routes). The problem with static routing is that this cannot be repaired without modifying many configuration files again.

```
 1  hostname christchurch
 2  interface enp0s3
 3  ip address 192.168.11.1/24
 4  interface enp0s8
 5  ip address 192.168.12.1/24
 6  interface enp0s9
 7  ip address 192.168.13.1/24
 8  interface lo
 9  ip route 192.168.14.0/24 192.168.12.2
10  ip route 192.168.15.0/24 192.168.13.2
11  ip route 192.168.16.0/24 192.168.13.2
12  ip route 192.168.17.0/24 192.168.13.2
```
