IPv6 Bootcamp

COSC301 Laboratory Manual

Reading guide

With the exception of Section 4, “Diagnostic and Query Tools”, you should be able to read most of this lab prior to coming to perform the lab. This will help you understand the material again (just like a text-book, the second reading is usually much more informative than the first).

There are many things you just can’t ignore, such as death and taxes; and there are things that you probably should ignore, such as requests from your users to install the latest greatest version of Mahjong on their workstation; but there are also things that you could ignore for now and learn later, which is what a lot of people have done with IPv6. This is a great and terrible illustration of the chicken and egg problem: most people aren’t bothering too much about IPv6 because not many are asking for it; and not many people are asking for it because not many people are offering it. Meanwhile, four of the five regional Internet registries (RIR) managed by the Internet Assigned Numbers Authority (IANA) have run out of IPv4 addresses. The RIRs are responsible for distributing publically routable IPv4 address blocks to Internet Service Providers (ISP). The Asia Pacific Network Information Center (APNIC), responsible for New Zealand, Australia, China, India, among others, has around 8 million IPv4 address remaining. These are distributed in blocks of 1024 addresses at a time. With approximately 300 new requests per year, this is expected to take a long time to run out completely[1]. The APNIC Labs [https://labs.apnic.net/?p=1107] has a report on the latest figures.

IPv6 is both simpler and more complex at the same time. Most of the complexity comes from general inexperience and ongoing development or misconceptions in most of the industry, but especially with regard to the various transition mechanisms to help IPv6 work in an IPv4 world, and IPv4 work in the coming IPv6 world. Some management areas become more complicated, such as DNS, while others ought to become simpler (such as address management).

We don’t have a lot of time in this lab to explore a lot of IPv6, so we shall cover only the very basics. We shall look at IPv6 enablement of various network services at the same time as we cover IPv4. In this lab, we shall look mostly at the client and how to manage IPv6 on the client side. Here is a brief overview of what we shall cover in this lab:

1. Practice enabling and disabling IPv6 on Linux, and how to control autoconfiguration etc.

2. Observe how IPv6 works with router advertisements using Wireshark, which is a network traffic analyser (“network sniffer”).

3. Practice the use of basic IPv6 diagnostic and query tools, namely **ping6** and **ip**.

1. Preparation

In the previous lab on basic network interface configuration, you should have restored to the snapshot you took at the beginning of that lab. In this section, we want to ensure that everything is still as we expect, and to add a new virtual appliance which we shall using just for today.

[1] See https://www.apnic.net/community/ipv4-exhaustion/graphical-information/ for more information.

In this lab we shall only be needing Client1 and a new virtual appliance from the `resources` folder. The new virtual appliance is called `COSC301-v6bootcamp-radv-version.ova`, which we shall call Radv. “Radv” is the virtual machine acting as our router advertiser. Radv is running the router-advertisement daemon (**radvd**).

On Client1, ensure that only one network adaptor is present, and that it is connected to the Internal Network “COSC301 Internal Network 1”. Start Client1, and ensure that **ifconfig -a** shows only one network interface (excluding the loopback interface), and that it is called “eth0”, and that it has no IPv4 address (no “inet” line, though it will have one “inet6” line). Basically, you should see this:

```
$ /sbin/ifconfig -a
eth0      Link encap:Ethernet  HWaddr 08:00:27:99:c2:7d
          inet6 addr: fe80::a00:27ff:fe99:c27d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1368 (1.3 KB)  TX bytes:1836 (1.8 KB)

lo        … Seen this before …
```

If you find an interface called “eth1” or “intnet*”, then you need to complete the Cleanup section from the previous lab. Reboot Client1 if you needed to make any modifications.

Next we need to start importing Radv. In VirtualBox’s main window, import the Radv appliance from the `Appliances` in the `resources` folder. If you cannot remember how, consult the notes from the "Introduction" lab. The filename will be named similarly to `COSC301-v6bootcamp-radv-version.ova`. Use the latest version available unless otherwise instructed. Accept the default settings as presented in the import dialogs.

The operation will take several minutes to complete. While you are waiting for the import operation to complete, go onto Section 2, “Enabling and Disabling IPv6”. We will come back to Radv in Section 3, “Observing Router Advertisements”.

2. Enabling and Disabling IPv6

In Linux, the act of enabling IPv6 is generally very trivial, as it is most often enabled by default. The tricky thing is disabling it if you don’t want it. Even trickier still is only enabling it on certain interfaces. In this section, we look at how support gets added to an IPv6 interface, how we can remove it if required, and how we can control some aspects of how IPv6 configures itself.

In Linux, the kernel support for IPv6 *might be* contained in a kernel module called `ipv6.ko` (the `.ko` extension meaning “kernel object”). Support for IPv6 is often made available as a loadable module; although in the case of Ubuntu 9.10 and later, it is built into the kernel, which complicates matters somewhat if you need to disable it.

One of the most common questions regarding IPv6 on Linux is how to turn it off, because like anything, if you don’t need it you don’t really want to be running with it. Many sites are not yet ready to support IPv6 and so it doesn’t make much sense to be using IPv6 because if you try using something that is unsupported in a production environment you can expect problems to occur. Running with IPv6 can cause transition mechanisms to be used which invites problems relating to tunneling, performance and security.

> **Note**
>
> Not all of the techniques below are applicable on Client1, which is running Ubuntu 18.04 LTS. Read all the sections, but only try the command shown in the sections with [applicable].

Removing IPv6 addresses when support is built-in… [applicable]

For us, as we are using Ubuntu 18.04 LTS, we don't have a kernel module we can unload as the support is built into the kernel. We could perhaps recompile the kernel and omit IPv6 support, but that is a lot of work and it doesn't help us if we just want to enable IPv6 on specific interfaces, which would be a handy thing to do on a gateway. Compiling your own kernel also creates a maintenance requirement that we could probably do without.

We can manage addresses on an interface using the **ip** tool. It can work with both IPv4 and IPv6 and is a Linux-specific tool, unlike the likes of **ifconfig**, which you can find on most Unix-like systems. Unlike IPv4 commands, which grew up with a shared heritage, IPv6 stacks have been developed more independently and have much different management interfaces on each platform.

Here is just one example of using the **ip** command to delete an address attached to an interface. Thinking back to the types of addresses that were covered in the lecture, what sort of IPv6 address is being deleted here?

```
$ ip -6 addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qlen 1000
    inet6 fe80::a00:27ff:fe97:164a/64 scope link
       valid_lft forever preferred_lft forever
# ip -6 addr del fe80::a00:27ff:fe97:164a/64 dev eth0
```

If we wanted to bring it back, we *could* use **ip -6 addr add address**, as shown in the example below.

> **Note**
>
> To add or delete an IPv6 address, make sure to include the prefix length (/64) explicitly after the address; otherwise the prefix length is defaulted to /128

```
# ip -6 addr add fe80::a00:27ff:fe97:164a/64 dev eth0
```

However, because the above command is generally used for configuring addresses manually, we won't use it here for our link-local address, which should be assigned automatically. We can illustrate that by bringing the interface back up using **ifconfig**:

```
# /sbin/ifconfig eth0 down
# /sbin/ifconfig eth0 up
```

And now our IPv6 address is back, which can be shown by the following command.

```
$ ip -6 addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436
    inet6 ::1/128 scope host
```

```
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qlen 1000
    inet6 fe80::a00:27ff:fe97:164a/64 scope link
       valid_lft forever preferred_lft forever
```

If you don't see this, repeat the above command.

Normally though, we would expect to bring an interface down or up using **ifdown** and **ifup**, as that includes the configuration of IPv4 addresses and other associated tasks. We can't do this at the moment on Client1, because we haven't configured `/etc/network/interfaces`, so it would not work presently.

```
This won't work on Client1 currently
# ifdown eth0
There may be output from any "hooks" that get run
# ifup eth0
There may be output from any "hooks" that get run
```

Passing arguments to the IPv6 subsystem… [applicable]

As soon as the IPv6 support is loaded into the kernel (in the case of built-in support this would be at boot-time) it will perform StateLess Address AutoConfiguration (SLAAC) in order to generate most of its addresses, so if we want to prevent that from happening, or configure how it is done, we need to pass it some arguments.

In the case of kernel modules, we can add arguments inside `/etc/modules` or similar, or by hand when we use **modprobe**. We can find out what parameters are available to a module using **modinfo**, but that assumes that we're dealing with loadable modules. However, for subsystems that are built into the kernel like IPv6, you need to add them to the kernel command-line, which is set by the bootloader (typically, this is GRUB on modern systems). So what parameters can we use, and how do we learn about them?

To find out what parameters are available, you could try looking in various manual pages such as ipv6(7) (more relevant to network programming) and ip(8) (interface management). But in this case, where we are wanting to know about kernel modules and such, the various documentation [http://www.kernel.org/doc/Documentation/] contained in the Linux kernel source is probably going to be more informative. In particular, the file `Documentation/networking/ip-sysctl.txt` documents the sort of thing we're after; here is the relevant extract; you are not expected to understand it just yet:

```
/proc/sys/net/ipv6/* Variables:

IPv6 has no global variables such as tcp_*.  tcp_* settings under ipv4/ also
apply to IPv6 [XXX?].     Hmm, how reliable is this documentation?

bindv6only - BOOLEAN
 Default value for IPV6_V6ONLY socket option,
 which restricts use of the IPv6 socket to IPv6 communication
 only.
  TRUE: disable IPv4-mapped address feature
  FALSE: enable IPv4-mapped address feature

 Default: FALSE (as specified in RFC2553bis)
…
conf/default/*:
 Change the interface-specific default settings.

conf/all/*:
```

```
 Change all the interface-specific settings.

conf/all/forwarding - BOOLEAN
 Enable global IPv6 forwarding between all interfaces.

    This would be important for creating a router, but we don't care about that just yet.
…
conf/interface/*:
 Change special settings per interface (where interface is the name of your network interface).


accept_ra - BOOLEAN
 Accept Router Advertisements; autoconfigure using them.

 Functional default: enabled if local forwarding is disabled.
       disabled if local forwarding is enabled.

accept_ra_defrtr, accept_ra_pinfo
    We can fine-tune what we accept in a router advertisement.

…
autoconf - BOOLEAN
 Autoconfigure addresses using Prefix Information in Router
 Advertisements.

 Functional default: enabled if accept_ra_pinfo is enabled.
       disabled if accept_ra_pinfo is disabled.
…
forwarding - INTEGER
    Configure interface-specific Host/Router behaviour.

    Note: It is recommended to have the same setting on all
    interfaces; mixed router/host scenarios are rather uncommon.

    Possible values are:
    0 Forwarding disabled
    1 Forwarding enabled

    FALSE (0):

    By default, Host behaviour is assumed.  This means:

    1. IsRouter flag is not set in Neighbour Advertisements.
    2. If accept_ra is TRUE (default), transmit Router
    Solicitations.
    3. If accept_ra is TRUE (default), accept Router
    Advertisements (and do autoconfiguration).
    4. If accept_redirects is TRUE (default), accept Redirects.

    TRUE (1):

    If local forwarding is enabled, Router behaviour is assumed.
    This means exactly the reverse from the above:

    1. IsRouter flag is set in Neighbour Advertisements.
    2. Router Solicitations are not sent unless accept_ra is 2.
    3. Router Advertisements are ignored unless accept_ra is 2.
    4. Redirects are ignored.

    Default: 0 (disabled) if global forwarding is disabled (default),
    otherwise 1 (enabled).


…
disable_ipv6 - BOOLEAN
 Disable IPv6 operation.  If accept_dad is set to 2, this value
 will be dynamically set to TRUE if DAD fails for the link-local
 address.
```

```
 Default: FALSE (enable IPv6 operation)

 When this value is changed from 1 to 0 (IPv6 is being enabled),
 it will dynamically create a link-local address on the given
 interface and start Duplicate Address Detection, if necessary.

 When this value is changed from 0 to 1 (IPv6 is being disabled),
 it will dynamically delete all address on the given interface.
…

IPv6 Update by:
Pekka Savola <pekkas@netcore.fi>
YOSHIFUJI Hideaki / USAGI Project <yoshfuji@linux-ipv6.org>
```

Also, in the `Documentation/kernel-parameters.txt` file, you are told that you can pass in arguments to built-in modules (in which case they are no-longer "modules" as such) using `modulename.parameter=value` syntax. So, for example, to run with the `disable_ipv6` option turned on (and hence disabling IPv6 addresses from being assigned), we can add the following argument to the command-line of the kernel:

```
ipv6.disable=1
```

But where do we put it? Assuming we are using the GRUB boot-loader, we need to edit the GRUB configuration. In Ubuntu 18.04 LTS, this is done using the file `/etc/default/grub` and those files in `/etc/grub.d/`. The two sources of files are combined into one using the command **update-grub**, which should be run after modifying either of these sources; the resultant file is stored in `/boot/grub/grub.cfg`, but you should not edit that as it will be overwritten on subsequent calls to **update-grub**. Find the part of `/etc/default/grub` that looks like the following:

```
GRUB_CMDLINE_LINUX=""
```

and change it to the following:

```
GRUB_CMDLINE_LINUX="ipv6.disable=1"
```

Run, with root privileges, the command **update-grub**; the final line of its output message should say `done`. Now reboot the guest operating system; you can do this easily from the command-line using **sudo shutdown -r now**, or from the GUI.

Exercise

As an exercise after you have restarted, make sure there is no IPv6 (`inet6`) address listed in **ifconfig**. If it does not work, double-check if you disabled the network manager.

Let's think about what we have just done. Previously, we had an IPv6 enabled kernel with IPv6 support enabled *by default*. Now, we have an IPv6 enabled kernel, but the interfaces are IPv6 *disabled* by default.

Regardless of whether IPv6 is by default enabled or disabled, we can enable or disable IPv6 interfaces at runtime by modifying the `disable_ipv6` parameter as follows:

```
$ sudo sh -c "echo 0 > /proc/sys/net/ipv6/conf/interface/disable_ipv6"
```

This is the general interface for tuning the networking stack behaviour. The contents of the `/proc` filesystem are special, as they are an interface into the running kernel. We shall revisit

this concept later in the paper when we look at filesystems in more detail. We will use `/proc` more frequently when we encounter firewalls, which are devices that control what traffic can enter or leave an interface.

Now *undo* your changes, because we actually *do* want IPv6 for the remainder of this paper. Verify that you get an IPv6 address when you boot the guest.

3. Observing Router Advertisements

In this section, you will practice using a network analyser called Wireshark[2] to take a close look at what happens when an interface is configured using StateLess Address AutoConfiguration (SLAAC) and to observe other fundamental IPv6 mechanisms.

By now Radv should have finished importing. Go into its Network settings and ensure that its Adaptor 1 is connected to the Internal Network "COSC301 Internal Network 1". Then Start it.

The purpose of this machine is send out IPv6 "router advertisements", so host on the network can learn about their local router(s) and determine a suitable set of addresses to use. Radv will not be used as a router for this lab, we only need it to offer us router advertisements, so we can study what they contain and how it influences Client1. You will not need to configure anything inside this machine.

When you have imported and started the new machine, which is called "Radv", it will boot to a console-login window ("radv login:"). You can minimise the machine, you won't need to interact with it.

Exercise

As an exercise on Client1, run **ifconfig** and **ip addr show** in a terminal window on Client1. You should see that you now have another address in addition to the link-local address (fe80::/64) you will typically always have. This address, which starts with fd6b:, is a "Unique-Local Unicast Address". If you see this address it means the router advertisements from Radv worked with Client1.

Wireshark needs to run with sufficient privileges to capture traffic, which generally—though not necessarily—implies root privilege. However, Wireshark is such a complicated program (due to all the various protocols it tries to understand) that experience has shown it to be something of a security risk, so ideally you would run it as a non-root user that has the appropriate permissions to capture traffic; but currently Wireshark doesn't support separation of privileges when you are capturing and analysing (displaying) the packets at the same time. So just run **wireshark** using a privilege elevator such as **sudo**.

Note

In the related video, we show an alternative, more secure way of doing the capture and analysis, using **tcpdump** for the capture, and **wireshark** for the analysis.

```
# wireshark
```

[2] Previously known as Ethereal.

Tip

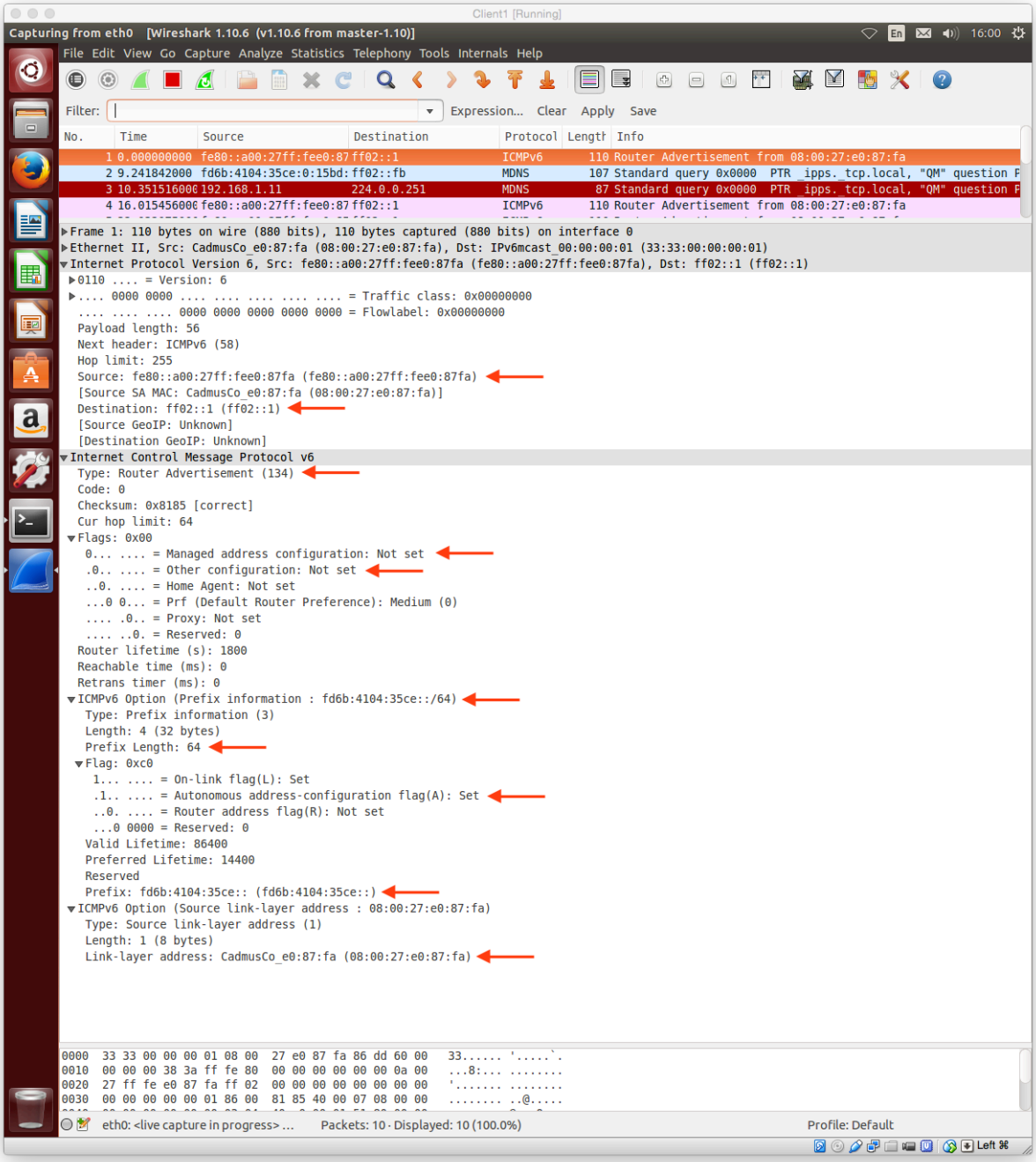
Wireshark will detect that it is running as root and pop up an alert box, which you will need to dismiss before you can continue running the application. Note that the alert box may be hidden behind the Wireshark window.

Start capturing on the first ethernet interface (it should be “eth0”). You should soon see ICMPv6 Routing Advertisements. The **radvd** software running on the machine Radv has been configured to send out routing advertisements rather more frequently than needed, to reduce your waiting time. Stop the capture when you have at least 10 or so. Using the View menu, un-check the Packet Bytes to give yourself more screen real-estate. Alternatively, you can just make the VirtualBox window larger and (due to the fact that the VirtualBox Guest Additions are installed inside the guest) it should change the screen resolution of the guest.

Exercise

As an exercise click on any one of the ICMPv6 Router advertisement packets. In the Packet Details area of the Wireshark window, click on the disclose (+ or little triangle) button recursively to open up the packet so you see that all of the fields and subfields for the “Internet Protocol Version 6” and “Internet Control Message Protocol v6”. Take a screenshot showing the entire contents, you will want to refer to it later.

Figure 1. Detail of IPv6 Router Advertisement in Wireshark



Wireshark showing the full details of a single IPv6 Router Advertisement.

Figure 1, “Detail of IPv6 Router Advertisement in Wireshark” shows the screenshot similar to what you should see, showing the details of a single router advertisement. Right now we’re going to discuss what is significant in this packet.

Here is a brief explanation of the most important parts of the router advertisement, from top to bottom in the packet details view.

IPv6 Source

In the packet shown, this is the "link-local unicast" address of Radv. We can confirm this by logging into Radv as `mal` and running **ifconfig**. (note that your Radv will have a different MAC address from ours) and seeing that the IPv6 address has a MAC address embedded inside of it[3].

IPv6 Destination

ff02::1 is a well-known address commonly called the "all-hosts link-local multicast address". Every host on the local network (the "local link", which is everything up to the first router) will be listening for traffic sent to this address.

Internet Control Message Protocol v6 (ICMPv6) Type

ICMPv6 is a management protocol that is very important to the running of IPv6. As such, there are many different types of messages it could transmit. To identify the type of message being transmitted, the Type field is used. Here, type 134 identifies this message as being a Router advertisement.

ICMPv6 Flags

The "Managed" address config flag specifies whether "Stateful configuration" is to be used. Stateful configuration would typically be understood to mean DHCPv6 (we learn about DHCP in a later lab, but we'll ignore DHCPv6 in this paper). On most networks, this flag would not be set, meaning that the host should use "StateLess Address AutoConfiguration" (SLAAC).

The "Other" stateful config flag means that nodes on this link (local network) should use Stateful configuration (DHCPv6) for things other than address assignment. Thus, if you don't use Stateful configuration for address assignment, you can still use it to advertise other information (such as information about DNS services, but we learn about DNS later in this course).

For example, in the capture we saw on our own network, we saw M=0 and O=0, meaning "Not managed" and "Not other". So we don't use stateful configuration (such as DHCPv6) to try and get an address (instead we just use SLAAC), and neither do we use stateful configuration to find out other, non-address information about the network (eg. DNS settings, which host to send log messages to, and a wide range of other possibilities, some of which we mention further in the lab on DHCP).

These "M" and "O" bits (Managed and Other) are important for understanding how IPv6 address assignment works. The remaining flags are not important for what we are aiming to understand today.

ICMPv6 Option (Prefix information)

There can be a number of optional components to a router advertisement. The most common would be an option advertising what "prefix(es)" are used on this link. A prefix is very much like a network address: in SLAAC, a set of addresses is formed by taking each prefix and adding the interface's EUI-64 host ID (typically formed by the MAC address).

Note that there can be multiple "Prefix information" options included in a router advertisement. An interface can use multiple prefixes to generate multiple IPv6 addresses.

ICMPv6 Prefix length

Somewhat self-explanatory, it tells us that this particular prefix is a /64 (remember that IPv6 addresses are 128-bit). With SLAAC, all advertised prefixes are /64, which makes this particular entry rather less interesting. We'll come back to it when we see the Prefix.

[3]This need not always be the case if you consider the "privacy" addresses.

ICMPv6 Flags
: The only flag we wish to draw to your attention here is the Autonomous Configuration flag (shown as “auto” in the screenshot). Recall that there can be multiple prefixes included in a router advertisement: only the ones marked with this “auto” flag get SLAAC addresses. This is on by default.

ICMPv6 Prefix
: This is the key piece of data in this announcement. Coupled with the prefix length we saw earlier, we see that this the prefix being advertised is fd6b:4104:35ce::/64. Viewed uncompressed, so you can see exactly how long the prefix is in relation to the whole address:

```
fd6b:4104:35ce:0000:0000:0000:0000:0000
     network bits ↤ ↦ host bits
```

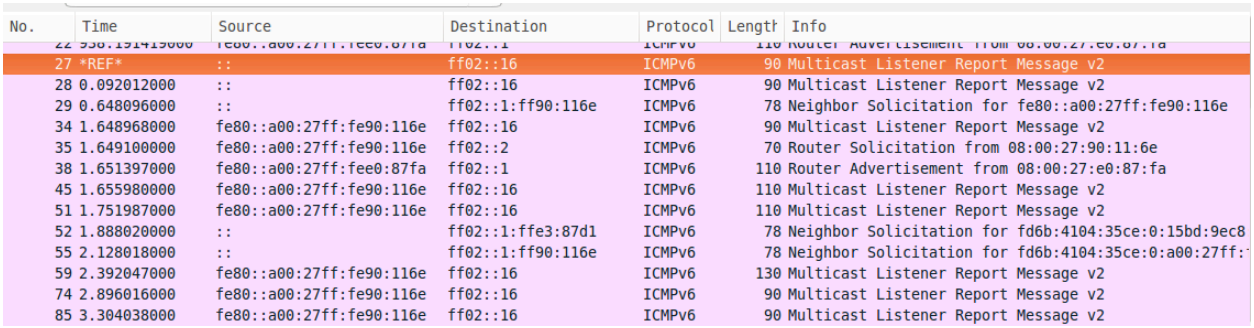
ICMPv6 Source link-layer address option
: This option simply communicates the router’s MAC address to the hosts on the link, so they can pre-cache it in their “neighbour cache” (this is like ARP, although IPv6 doesn’t use ARP but something similar which we don’t want to get into right now). Having it included here means we don’t have to have an extra round-trip on the network to figure out the link-layer (MAC) address of the router, which reduces delay.

So that’s a router advertisement, seen during steady-state whereby nothing particularly interesting is happening such as a new host appearing on the link. Let’s now repeat the capture and this time we shall see everything that happens when an interface comes up.

Exercise

As an exercise resize the Wireshark display so that the packet list area takes up most of the room. Start a new live capture in Wireshark; you may discard your previous capture. When it has started capturing, from a terminal window, as root, disable IPv6 with **echo 1 > /proc/sys/net/ipv6/conf/all/disable_ipv6**. Now remember what the last packet was that is currently shown in Wireshark’s packet list area. When you're ready, run change the 1 to a 0 in the previous command, you should see several packets arrive in the window. Then stop the capture and save the captured packets. Figure 2, “What Wireshark sees when an Interface “Comes Up”” is what we see on our own capture, which may likely be a bit different from yours; in particular, the ordering of some packets can vary.

Figure 2. What Wireshark sees when an Interface “Comes Up”



No.	Time	Source	Destination	Protocol	Length	Info
27	*REF*	::	ff02::16	ICMPv6	90	Multicast Listener Report Message v2
28	0.092012000	::	ff02::16	ICMPv6	90	Multicast Listener Report Message v2
29	0.648096000	::	ff02::1:ff90:116e	ICMPv6	78	Neighbor Solicitation for fe80::a00:27ff:fe90:116e
34	1.648968000	fe80::a00:27ff:fe90:116e	ff02::16	ICMPv6	90	Multicast Listener Report Message v2
35	1.649100000	fe80::a00:27ff:fe90:116e	ff02::2	ICMPv6	70	Router Solicitation from 08:00:27:90:11:6e
38	1.651397000	fe80::a00:27ff:fee0:87fa	ff02::1	ICMPv6	110	Router Advertisement from 08:00:27:e0:87:fa
45	1.655980000	fe80::a00:27ff:fe90:116e	ff02::16	ICMPv6	110	Multicast Listener Report Message v2
51	1.751987000	fe80::a00:27ff:fe90:116e	ff02::16	ICMPv6	110	Multicast Listener Report Message v2
52	1.888020000	::	ff02::1:ffe3:87d1	ICMPv6	78	Neighbor Solicitation for fd6b:4104:35ce:0:15bd:9ec8
55	2.128018000	::	ff02::1:ff90:116e	ICMPv6	78	Neighbor Solicitation for fd6b:4104:35ce:0:a00:27ff:
59	2.392047000	fe80::a00:27ff:fe90:116e	ff02::16	ICMPv6	130	Multicast Listener Report Message v2
74	2.896016000	fe80::a00:27ff:fe90:116e	ff02::16	ICMPv6	90	Multicast Listener Report Message v2
85	3.304038000	fe80::a00:27ff:fe90:116e	ff02::16	ICMPv6	90	Multicast Listener Report Message v2

Wireshark showing the packets that are seen when an interface comes up. The packet marked as *REF* is the first packet to come into the system after we brought up the interface.

Rather than having you read about what each packet is doing, we're going to show you with a video, which will walk you through what is happening in this sequence of packets; or rather, a similar sequence of packets. In this video, which lasts about half an hour, you will encounter various parts of IPv6 that form part of its basic mechanisms. We require that you have some understanding of what is going on, to an extent whereby you could look at such a listing, and with a bit of work, explain basically what is happening. In the self-assessment, you will be asked to summarise some of what you have seen.

The video can be found in the "resources" folder of your desktop, and is called *IPv6 SLAAC* under `Lab resources/IPv6`. However, you should watch the video later.

Watch the video later

This is because you won't need to be in the lab to complete the video-watching task, but you will want to be in the lab for Section 4, "Diagnostic and Query Tools". Doing this will help you use your time more effectively.

4. Diagnostic and Query Tools

This section is very simple; it's just learning to use some simple tools to see what we they can tell us and to get ourselves familiarised with the environment. Without this familiarity, we are blind and lame when we need to diagnose problems. We shall limit ourselves to a very small selection of commands.

Basic ip usage

With IPv6, we generally have a new set of tools, or system specific additions to old tools, such as **ifconfig**. On Linux, the new, preferred and much more capable tool that manages most of the network stack is simply called **ip**.

The **ip** command has extensive help built in, so you generally don't use the manual page for getting documentation. The help system is modal; it depends on what mode you are using. For example, try these commands on Client1 to see the different help you get.

```
$ ip help
Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }
       ip [ -force ] [-batch filename
where  OBJECT := { link | addr | addrlabel | route | rule | neigh | ntable |
                   tunnel | maddr | mroute | monitor | xfrm }
       OPTIONS := { -V[ersion] | -s[tatistics] | -d[etails] | -r[esolve] |
                    -f[amily] { inet | inet6 | ipx | dnet | link } |
                    -o[neline] | -t[imestamp] }
$ ip addr help
Usage: ip addr {add|change|replace} IFADDR dev STRING [ LIFETIME ]
                                                      [ CONFFLAG-LIST]
       ip addr del IFADDR dev STRING
       ip addr {show|flush} [ dev STRING ] [ scope SCOPE-ID ]
                            [ to PREFIX ] [ FLAG-LIST ] [ label PATTERN ]
IFADDR := PREFIX | ADDR peer PREFIX
          [ broadcast ADDR ] [ anycast ADDR ]
          [ label STRING ] [ scope SCOPE-ID ]
SCOPE-ID := [ host | link | global | NUMBER ]
FLAG-LIST := [ FLAG-LIST ] FLAG
FLAG  := [ permanent | dynamic | secondary | primary |
           tentative | deprecated | CONFFLAG-LIST ]
CONFFLAG-LIST := [ CONFFLAG-LIST ] CONFFLAG
CONFFLAG  := [ home | nodad ]
LIFETIME := [ valid_lft LFT ] [ preferred_lft LFT ]
```

```
LFT := forever | SECONDS
```

In order to understand what is being shown, you simply need to understand that `[ … | … ]` denotes an optional choice, `{ … | … }` denotes a mandatory choice, `lowercase` are keywords (mandatory) and `UPPERCASE` are rules ("grammar productions" to use a Computer Science term) that expand (`:=`) further. You can also see that you just need to add **`help`** to the end of the command you are trying to complete; it's pretty simple, though a little bewildering at first. Most of the things you don't need to worry about for simple uses; the **ip** command can allow for quite advanced commands which makes the grammar more useful to have.

Let's use **ip** to look at some useful information. For each example, add **`help`** to the end to see what else you can do. Since we're only interested in IPv6 at present, we'll add the `-6` option so it limits its operation to IPv6. Also, we're not going to show the output of the commands; this makes you have to figure out what they do for yourself.

```
$ ip -6 addr
…
$ ip -6 link
…
$ ip -6 route
…
```

Okay, so that was pretty basic, we just looked at the Layer 3 network address configuration (for IPv6) and the Layer 2 datalink (Ethernet, in this case) information.

We want to briefly look at the glue the binds the IPv6 layer (network layer) to the Ethernet layer (datalink layer). One of the things that makes IPv6 different from IPv4 is that we no-longer use ARP to determine the Ethernet MAC address that is bound to a particular IPv4 address. Instead, IPv6 has its own protocol for this called Neighbour Solicitation. Likewise, the Neighbour Cache has replaced the ARP cache; that's a little simplified, but it will do for now.

```
$ ip -6 neigh help
…
$ ip -6 neigh
fe80:a00:27ff:fe69:9487 dev eth0 lladdr 08:00:27:69:94:87 router STALE
```

We use the above output to illustrate a few things (your output will look different). The basic pattern is an IPv6 address is associated to a link-layer address (in this case, an Ethernet MAC address) on a particular network device. This IPv6 address apparently belongs to a router (but at the moment we don't care about that). Each entry has a Neighbour Unreachability Detection (NUD) state; "stale" means the entry is valid but not particularly trustworthy because of its age. Let's ping Radv and see what changes in our neighbour cache.

Pinging with IPv6

Log in as mal on Radv (the password is the same) and find out Radv's enp0s3 IPv6 link-local address. We suggest you write it down. Now ping Radv from Client1; don't include the prefix length (`/64`) in the link-local address of the command below:

```
$ ping6 -c2 radv-link-local-ipv6-address%interface-id-of-the-local-host
…
```

If you get "connect: Invalid Argument", that indicates that you forgot to include the interface id of the local machine such as `eth0` or `intnet1`, or the interface id was wrong. If instead you got "unknown host", then it means you copied the address wrongly such that it is no longer a valid IPv6 address.

> **Note**
>
> Notice that because we are pinging a link-local address (fe80::/64), which every interface has and is therefore generally ambiguous, we need to use a special syntax to identify the interface we want to use `%eth0`. A number can be used for the interface ID on some systems instead, such as Windows.

What actually went on there? If you had been looking at what was happening on the network, using Wireshark or similar, you should likely see something similar to this, although it can differ depending on what each side already knows. So the packets used look like the following:

1. Client1 wants to send an IPv6 packet to Radv's IPv6 link-local address (layer 3), which is a local delivery. But it probably doesn't yet know Radv's link-layer address (layer 2), so it needs to look it up by using a **ICMPv6 Neighbour solicitation** from Client1's link-local IPv6 address to the Solicited node multicast address for the address that shall be pinged. It also carries with it a notification of Client1's link-layer (Ethernet) address, preventing Radv from having to look it up later and thus saving network traffic.

   This is analogous to an ARP request in IPv4.

2. Radv responds with a **ICMPv6 Neighbour advertisement**. It comes from Radv's link-local IPv6 address to Client1's link-local IPv6 address. It carries a notification of Radv's link-layer (Ethernet) address, which Client1 puts into its Neighbour Cache.

   This is analogous to an ARP reply and the result being cached in the ARP cache.

3. Now Client1 knows everything it needs to in order to make the local delivery of this IPv6 packet, so it sends the **ICMPv6 Echo request** ("ping"), from Client1's link-local IPv6 address to Radv's link-local IPv6 address (which is the address we pinged).

4. Radv has already learned of Client1's link-layer address, so it doesn't need to perform the Neighbour Solication process again, as it has it in its neighbour cache. So Radv can now send the **ICMPv6 Echo response** packet immediately, which is its response to the "ping" (Echo request) packet.

Notice that this is a *local*, or "on-link" delivery; it does not go through a router. We'll ignore the issue of routing until much later.

We've seen some multicast addresses being used; how do we find out which multicast addresses the host is currently interested in? **ip** can help here also:

```
$ ip -6 maddr
…
```

That's it, a few very simple commands to allow us to investigate the local link (layer 2) and IPv6 in a single network.

There is much we haven't covered in this lab, such as assigning static IPv6 addresses, which is really easy, but we'll cover that later.

5. Self-assessment

This lab has been very introductory, and we don't want to labour you with a lot of assessment, except to give you a little to help you check that you have understood some of the new concepts you have seen today.

You should be able to answer the following questions after attending the IPv6 lecture and watching the video as mentioned in Section 3, “Observing Router Advertisements”. You can work on the questions in small groups (to help everyone understand).

1. Describe multicast, as opposed to unicast and broadcast. Also, give an example of an IPv6 multicast address.

2. Give an example of a link-local IPv6 address. Briefly describe how it is generally formed from the Ethernet MAC address.

3. Give an example of a “unique-local” IPv6 address.

4. Describe a Solicited Node [multicast] address. What is its purpose? Give an example. How is it formed?

5. Multicast Listener Discovery version 2 (MLDv2) Snooping is a mandatory feature for any switch that wants to support IPv6[4]What is the motivation for MLDv2 snooping, which in the IPv4 world we might compare with “IGMP Snooping”? We want to see that you understand its purpose with respect to network switches (not routers). You may like to include a diagram with a little table showing real IPv6 multicast addresses (such as two solicited node addresses, and an all-routers address (ff02::2) and switch port numbers.

6. Briefly describe the most important components of a router advertisement.

7. Briefly describe Duplicate Address Detection (DAD). Why is it needed and how is it performed? What do you think the difference is between a “Tentative” address and a “Preferred” address, with regard to DAD?

[4]Or at least, mandatory in the sense that it should be a purchasing requirement.