Basic TCP/IP networking

Grado en Informática 2025/2026

Departamento de Ciencias de la Computación
y Tecnologás de la Información
Facultad de Informática
Universidad de Coruña

Antonio Yáñez Izquierdo

Contents I

1 Basic network configuration
- solaris 11
- openbsd
- FreeBSD
- NetBSD
- linux
- ubuntu and devuan linux
- fedora linux
- linux: Interacting with Network Manager
- linux's nmcli
- linux's netplan

2 Routes

3 Network interface aliasing

4 Access control

Contents II

- stopping and disabling services
- Access control at the packet level
- Access control at the application level
- Access control without `inetd`

5 inetd
- inetd configuration

6 inetd and tcpwrappers
- tcpwrappers
- inetd and tcpwrappers in Solaris

Basic network configuration

basic IP v4 configuration

- to properly configure a machine using ipv4 we have to configure
  - the machine name
  - the Network Interface Cards
  - the routes
  - the dns (if using it)

basic NIC configuration

- The basic things we have to configure for a Network Interface Card are
  - its ip address
  - its netmask (number of bits in its ip address that correspond to network address)
  - its broadcast address

ways to configure the network

- there are two ways to configure the network
  - **manual configuration:** we configure manually each of the paramaters, either directly using the comand line or through the boot scripts
  - **using dhcp:** the network interface card *asks for its configuration* to a machine in the network (the *dhcp server*). This can be done directly through the comand line or using the boot scripts
- most systems have a graphic utility to configure the network, which can be used to configure either manually or via *dhcp*. We won't deal with those utilities, neither will we deal with the wireless configuration options

`ifconfig`

- the comand **ifconfig** used to be the chosen way to configure network interfaces. It has been superseded in a number of unix flavours (`ip` in linux, `ipadm` solaris 11)
- it is usually still available, either installed in the base system or a separate package
- it is usually located at **/sbin/ifconfig**
- it can configure interfaces both manually or using dhcp
- `ifconfig -a` (or -A, depending on the unix variant) shows the actual configuration of the Network Interface Cards

configuring the dns

- the configuration of the *dns* resides on the file `/etc/resolv.conf`
- this file has the options to the *resolver* configuration. The most common options are
  - **nameserver** to specify the address of a domain name server, up to 3 can be defined
  - **domain** (optional) to sepecify the local domain. Short names are supposed to be from this domain

configuring the dns

- example of `/etc/resolv.conf` file

```
domain dc.if.udc.es.
nameserver 193.144.51.10
nameserver 192.144.48.30
```

- Solaris 11 has that file, but the resolver sevice is configured through `svccfg`. Some linux distrbutions (for example, ubuntu) use `systemd-resolved`, making that file a symbolic link to `/run/systemd/resolve/stub-resolv.conf`

the /etc/hosts file

- this file contains the locally defined ip addresses of hosts
- its format is

  ```
  ip_address      host_name     aliases
  ```

- example of /etc/hosts

```
127.0.0.1       localhost
192.168.1.99    abyecto.dc.fi.udc.es    abyecto
```

the /etc/nsswitch.conf file

- usually called the name service switch file
- this file is used to determine the sources from where to obtain name-service information of several categories: hosts, users, mail aliases ...
- it also specifies the order in which this sources of information should be queried
- in the following example, the hosts ips are first searched for in the local files, then the dns is queried

```
passwd:         compat
group:          compat
shadow:         compat

hosts:          files dns
networks:       files
```

Basic network configuration
→solaris 11

network configuration in Solaris 11

- Up to version 11.3 Solaris 11 used profile based network configuration. By default there were two NCP (Network Configuration Profiles): the DefaultFixed NCP and the Automatic NCP.
- Additional profiles can be created by the system administrator
- we could switch between the profiles with the **netadm** command

```
# netadm enable -p ncp DefaultFixed
                  //to activate the DefaultFixed profile
# netadm enable -p ncp Automatic
                  //to activate the Automatic profile
```

network configuration in Solaris 11

- in the DefaultFixed profile we could use the commands **dladm** and **ipadm** to configure the network
- in the Automatic profile we used the commands **netcfg** and and **netadm** to create and manage network configuration.
- **netadm list** used to show the active NCP on the system. In present versions it shows the avaiable ENMs
- changes made through these commands are persistent across reboots, so there's no need to deal with configuration files.

network configuration in Solaris 11

- In present version of Solaris 11, we have ENMs (External Network Modifiers)
- An ENM manages applications that are responsible for creating network configuration that is external to the system's primary network configuration
- ENMs can be created and/or destroyed with `netcfg`.

network configuration in Solaris 11

- `netcfg` can also be used to set the activation mode for the ENMs
  - `manual`: the ENM is activated or deactivated by the system administrator (or a role) with the `netadm enable` or `netadm disable` command
  - `conditional`: the ENM is activated by the system based on changes in the conditions of the ENM (such as: plugging or unplugging an ethernet cable, obtaining or losing a DHCP lease, detecting a new wireless network,...)
- For the default network configuration (the DefaultFixed profile in previous versions od Solaris 11) we'll use mostly the `ipadm` command

network configuration in Solaris 11

- network interfaces are created with the **ipadm** command.

```
# ipadm create-ip net0
```

- network interfaces can be destroyed with:

```
# ipadm delete-ip net0
```

- once created we can assign them an static ipv4 address

```
# ipadm create-addr -T static -a 192.168.1.7/24 net0/addr
```

- or have them configured through dhcp

```
# ipadm create-addr -T dhcp net0/addr
```

network configuration in Solaris 11

- to disable one interface.

  ```
  # ipadm disable-if net0
  ```

- to see the addresses asigned to the interfaces

  ```
  # ipadm show-addr
  ```

- to see the status of the interfaces

  ```
  # ipadm show-if
  ```

- to see the physical properties of each datalink

  ```
  # dladm show-phys
  ```

network configuration in Solaris 11

- to stablish the default route

```
# route -p add default 192.168.2.1
```

- the dns is configured thorugh the services facilities (service `network/dns/client`)

```
# svccfg -s network/dns/client setprop \
  config/nameserver = net_address: 193.144.51.10
```

- as is the nswitch (service `name-service/switch`)

```
# svccfg -s name-service/switch setprop \
  config/host = astring: '("files dns")'
```

network configuration in Solaris 11

- the hostname is implemented as a property of the `system/identity` service
- to change the hostname

```
# svccfg -s system/identity:node setprop\
 config/nodename=nuevonombre
# svccfg -s system/identity:node refresh
# svcadm restart system/identity:node
```

network configuration in Solaris 11

- tambien es posible importar el el servicio correspondiente la configuración desde los ficheros habituales
  - service `network/dns/client` (suponiendo que `/etc/resolv.conf` existe y tienen una configuración válida)

    ```
    # /usr/sbin/nscfg import -f dns/client
    # svcadm enable dns/client
    ```
  - service `name-service/switch` (suponiendo que `/etc/nsswitch.conf` existe y tienen una configuración válida)

    ```
    # /usr/sbin/nscfg import -f name-service/switch
    # svcadm refresh name-service/switch
    # svcadm refresh name-service/cache
    ```

Basic network configuration
→openbsd

NIC configuration in openBSD

- the interfaces are named after the driver in the kernel that manages them.
  - example: the kernel uses the *em* driver for *Intel(R) PRO/1000* NICS. Cards of this type will get the names em0, em1 ...
- **dhclient interface_name** configures the card *interface_name* using dhcp.

NIC configuration in openBSD

- **ifconfig interface_name inet address netmask broadcast** configures the card *interface_name* with address *address*, netmask *netmask* and broadcast address *broadcast*.l

```
#ifconfig em0 inet 192.168.1.100 255.255.255.0  192.168.1.255
```

- **ifconfig interface_name up** brings the interface up

NIC configuration in openBSD at boot time

- if we want to get the interfaces automatically configured at boot time (via `/etc/netstart`) we'll use the following files
  - interfaces using dhcp
    - `/etc/hostname.interface_name` file containing the word **dhcp** (see `hostname.if` man page)
    - from version 7.0 `/etc/hostname.interface_name` file containing the word **autoconf** (see `hostname.if` man page)

NIC configuration in openBSD at boot time

- interfaces configured manually
  - `/etc/hostname.interface_name` file containing the necessary parameters passed to *ifconfig* to configure the interface. If we'd want to configure an 'inet6' interface we would use *inet6* instead of *inet* in the `/etc/hostname.interface_name` file

```
# cat /etc/hostname.em0
inet 192.168.1.100  255.255.255.0  192.168.1.255
#
```

NIC configuration in openBSD at boot time

- `/etc/myname` Contains the complete name of the system
- `/etc/mygate` Contains the ip address of the default router.

Basic network configuration
→FreeBSD

NIC configuration in FreeBSD

- the interfaces are named after the driver in the kernel that manages them.
  - example: the kernel uses the *le* driver for *PCNet PCI-II* NICS. Cards of this type will get the names le0, le1 ...
- **dhclient interface_name** configures the card *interface_name* using dhcp.

NIC configuration in freeBSD

- **ifconfig interface_name inet/inet6 address netmask broadcast** configures the card *interface_name* with address *address*, netmask *netmask* and broadcast address *broadcast*.I

```
#ifconfig le0 inet 192.168.1.100 255.255.255.0  192.168.1.255
```

- **ifconfig interface_name up** brings the interface up

NIC configuration in freeBSD at boot time

- if we want to get the interfaces automatically configured at boot time we do so adding a line for each interface in the file `/etc/rc.conf`
- we use one line for each interface to be configured.
- The line assigns to a shell variable named `ifconfig_`*`interface_name`* the parameters that would be passed to *ifconfig*

NIC configuration in freeBSD at boot time

- The following lines in `/etc/rc.conf` get the interfaces em0, le0 and le1 configured at boot time

```
ifconfig_em0="DHCP"
ifconfig_le0="inet 192.168.1.3 netmask 255.255.255.0"
ifconfig_le1="inet 10.0.0.1 netmask 255.255.0.0"
```

network configuration in freeBSD at boot time

- We can also configure both the hostname ad the default route through variables in `/etc/rc.conf`
- Example

```
hostname="machine.ningunsitio.org"
defaultrouter="192.168.2.1"
```

Basic network configuration
→NetBSD

NIC configuration in NetBSD

- **ifconfig interface_name inet address netmask broadcast** configures the card *interface_name* with address *address*, netmask *netmask* and broadcast address *broadcast*.l

```
#ifconfig wm0 inet 192.168.1.100 255.255.255.0  192.168.1.255
```

- **ifconfig interface_name up** brings the interface up
- the name of the *dhcp* client is `dhcpcd` and is usually started as a daemon

NIC configuration in NetBSD at boot time

- if we want to get the interfaces automatically configured at boot time
  - interfaces using dhcp
    - the `/etc/ifconfig.interface_name` file contains the parameters we want to pass to ifconfig
    - We enable the client daemon *dhcpcd* with `dhcpcd=‘‘YES’’` in the file `/etc/rc.conf`
    - We pass the interface to configure in the variable `dhcpcd_flags` in the file `/etc/rc.conf`

NIC configuration in openBSD at boot time

- interfaces configured manually
  - `/etc/ifconfig.interface_name` file containing the necessary parameters passed to *ifconfig* to configure the interface. If we'd want to configure an 'inet6' interface we would use *inet6* instead of *inet* in the `/etc/ifconfig.interface_name` file

    ```
    # cat /etc/ifconfig.wm0
    inet 192.168.1.100  255.255.255.0
    #
    ```

  - We can also configure the interfaces with the appropiate variable `ifconfig_ifacename` in `/etc/rc.conf`

    ```
    # cat /etc/rc.conf
    ......
    ifconfig_wm0=‘‘inet 192.168.1.100  255.255.255.0’’
    ....
    ```

NIC configuration in NetBSD at boot time

- `/etc/myname` Contains the complete name of the system in case the variable `hostname` in `/etc/rc.conf` is not set
- `/etc/mygate` Contains the ip address of the default router in case the variable `defaultroute` in `/etc/rc.conf` is not set

Basic network configuration
→linux

Naming Network Interfaces in linux

linux distros have followed several namimg strategies for namimg NICS

1. linux used to name their NICs eth0, eth1 . . . and the order was defined by which one got detected first
   - this makes order dependent on module loading, and changing one NIC for other could change all the names
2. the names *eth0, eth1, eth2 . . .* are asigned to the interfaces **THE FIRST TIME** the kernel recognices them. This is stored in the file `/etc/udev/rules.d/70-persistent-net.rules`, where it can be changed if necessary.

Naming Network Interfaces in linux

3 they get the names like em*N*, emp*N*p*M*, ens*N*, p*N*p*M*. This new name scheme does not make names dependent on the type of card, its mac or when it is detected; the names are generated depending on how (where) they are connected to the system which makes it easier to substitute interfaces

Naming Network Interfaces in linux

- as of today's present versions
  - *devuan* uses criteria labeled *3* although interfaces are named *eth0, eth1 . . .*
  - *ubuntu and debian* use criteria labeled *3*
  - *fedora* uses criteria labeled *3*

NIC configuration in devuan linux

```
abyecto:/home/antonio# cat /etc/udev/rules.d/70-persistent-net.rules
# This file was automatically generated by the /lib/udev/write_net_rules
# program, run by the persistent-net-generator.rules rules file.
#
# You can modify it, as long as you keep each rule on a single
# line, and change only the value of the NAME= key.

# PCI device 0x11ab:0x4363 (sky2)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="00:24:be:40:5c:
ATTR{type}=="1", KERNEL=="eth*", NAME="eth0"

# PCI device 0x8086:0x4232 (iwlagn)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="00:24:d6:0e:ae:
ATTR{type}=="1", KERNEL=="wlan*", NAME="wlan0"
abyecto:/home/antonio#
```

NIC configuration in devuan linux

- **dhclient interface_name** configures the card *interface_name* using dhcp.
- **ifconfig interface_name inet address addr netmask netmk broadcast bcast** configures the card *interface_name* with address *addr*, netmask *netmk* and broadcast address *bcast*
- the *ifconfig* command in linux is being superseded with the *ip* command so although still available in *devuan* it may not be available in other distros, unless explicitly installed

```
#ifconfig eth0 inet 192.168.1.100 netmask 255.255.255.0 broadcast 192.168.1.255
```

- **ifconfig interface_name up** brings the interface up (same as *ifup*)

NIC configuration in devuan linux at boot time

- if we want to get the interfaces automatically configured at boot time (via `/etc/init.d/networking`)
- debian systems and derivatives will look for the file `/etc/network/interfaces` (see `interfaces` man page)
- `/etc/hostname` Contains the name of the system (either the fully qualified domain name or just the nodename)

NIC configuration in devuan linux at boot time

- Sample `/etc/network/interfaces` with just one NIC manually configured

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
#allow-hotplug eth0
auto eth0
iface eth0 inet static
 address 192.168.1.99
 netmask 255.255.255.0
 network 192.168.1.0
 broadcast 192.168.1.255
 gateway 192.168.1.1
```

NIC configuration in devuan linux at boot time

- Sample `/etc/network/interfaces` with just two NICs

```
root@abyecto:~# cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo eth0 eth1
iface lo inet loopback

# The primary network interface
allow-hotplug eth0
iface eth0 inet dhcp
# internal network
allow-hotplug eth1
iface eth1 inet static
 address 192.168.1.100
 netmask 255.255.255.0
 network 192.168.1.0
 breadcast 192.168.1.255
```

NIC configuration ubuntu server

- *ubuntu* is, as is devuan linux, a derivative of the debian linux distribution
- the configuration of network interfaces is also done using the `/etc/network/interfaces` file
- the naming of the interfaces is of type `empNsM` as is in fedora linux
- the `/etc/resolv.file` is a symbolic link to `/var/run/resolvconf/resolv.conf`

Basic network configuration
→ubuntu and devuan linux

debian derivatives

- ubuntu and devuan are distibutions based on debian. Their configuration files resemble that of the debian distibution
- ubuntu now names interfaces the *fedora way* but the configuration files for basic networking are the same as in debian
- As of version 8 (code name *jessie*) debian has switched to a *systemd* based distrubution. *devuan* is a fork of debian without the need to install *systemd*. As far as basic network configuration is concerned, they are the same.

Basic network configuration
→fedora linux

NIC configuration in fedora linux

- fedora changed de naming of the interfaces in a linux system using the *biosdevname* program.
- They are no longer named *eth0, eth1,...*; they get the names `em`*N*, `emp`*N*`p`*M*, `ens`*N*, `p`*N*`p`*M*
- This new name scheme does not make names dependent on the type of card or its mac, the names are generated depending on how (where) they are connected to the system which makes easier to substitute interfaces

NIC configuration in fedora linux

- We can use the same commands as in debian linux to configure the interfaces
  - **dhclient interface_name** configures the card *interface_name* using dhcp.
  - **ifconfig interface_name inet address addr netmask netmk broadcast bcast** configures the card *interface_name* with address *addr*, netmask *netmk* and broadcast address *bcast*

NIC configuration in fedora linux

- *fedora linux* reommends to use the command `ip addr` or `ip link` to configure the interface instead of *ifconfig*
- Examples
  - `ip addr show` shows the configuration of the interfaces
  - `ip link set p2p1 down` brings the interface *p2p1* down
  - `ip link set p2p1 up` brings the interface *p2p1* up
  - `ip addr add 192.168.2.100 dev p2p1` adds address 192.168.2.100 to interface *p2p1*
  - `ip addr del 192.168.2.100 dev p2p1` removes address 192.168.2.100 to interface *p2p1*
- with this command we can easily assign more than one address to one interface

NIC configuration at boot time

- if we want to get the interfaces automatically configured at boot time we should take into account the following files
- **/etc/sysconfig/network**. A file defining the following variables.

```
NETWORKING=yes.or.no
HOSTNAME=fully.qualified.name
GATEWAY=ipaddr.of.the.gateway
GATEWAYDEV=interface
```

NIC configuration at boot time

- **/etc/sysconfig/network-scripts/ifcfg-interface_name**.A file for each interface to be configured in the system. It defines, among others, the following variables

```
DEVICE=name
BOOTPROTO=protocol (can be none, bootp or dhcp)
IPADDR=address
NETMASK=mask
BROADCAST=address
```

- Las opciones de los archivos de configuración están descritas en (`man ifup` proporciona la información)

```
/usr/share/doc/initscripts/sysconfig.txt
```

Basic network configuration
→linux: Interacting with Network Manager

Interacción con Network Manager

- *Network Manager* is a package that gets installed in desktop environments.
- It consists of a *daemon* execiting as `root` and a *font-end* dependant on the desktop environment
- *Network Manager* It tries to manage all of the NICs not already managed
- To find out which interfaces are managed by *Network Manager*

  `#nmcli dev status`
- *Network Manager's* configuration resides in `/etc/NetworkManager/NetworkManager.conf`

Interfaces NOT managed by Network Manager

If we want to have a NIC not managed by *Network Manager* we must do the following

debian (and derivatives): Configure the NIC via `/etc/network/interfaces` and have `/etc/NetworkManager/NetworkManager.conf` contain the following

```
[main]
plugins=ifupdown
[ifupdown]
managed=false
```

Interfaces NOT managed by Network Manager

fedora Add to the file `ifcfg-iface` the lines `NM_CONTROLLED=no` and `HWADDR=??:??:??:??:??:??` and add the following to `/etc/NetworkManager/NetworkManager.conf`

```
[main]
plugins=ifcfg-rh
```

Interfaces managed Network Manager

- If we want to temporarily stop *Network Manager* we can do it with the following commands (depending on the distro)

```
# service NetworkManager stop
# /etc/init.d/network-manager stop
# systemctl stop NetworkManager.service
```

- Should we want to disable it (it does not start when the O.S. boots),

```
# chkconfig NetworkManager off
# update-rc.d network-manager remove
# insserv -r network-manager
# systemctl disable  NetworkManager.service
```

Basic network configuration
→linux's nmcli

Network Manager

- As a matter of fact, network configuration in *fedora* (as in many other linux distros) is done via Network Manager.
- Network Manager consists of
  - a daemon service, that can be started or stopped using systemctl (or the appropiate init script)
  - an applet to be used by the graphical environment
  - a command line interface, **nmcli**, to configure network manager

Network Manager

- So if we are on a system that uses Network Manager to manage the network connections we can choose to
  - configure the network via the `interfaces`, `ifcfg-`*`ifname`* `...` files, provided we have configured network manager to use (or accept) those files
  - use the applet for Network Manager provided in our graphical environment
  - use Network Manager Command Line Interface, **nmcli**, to configure our interfaces via a script or using the command line

Uso de `nmcli`

- The following steps we will show the use of `nmcli` to configure an ethernet connection with three static ips. Neither route nor dns will be configured for that connection
- `nmcli's` general syntax is

```
# nmcli [OPTIONS...] OBJECT [COMMAND] [ARGS...]
```

- `OBJECT` can be any of **"help, general, networking, radio, connection, device, agent, monitor"**, and can be written in abbreviated form
- `COMMAND`, the command to execute, can also be written in abbreviated form

Uso de `nmcli`

- for example, to show the current connections we can use either of these

```
# nmcli connection show
# nmcli con show
# nmcli c s
```

Creating a connection with nmcli

- First whe look what devices we have in the system to create the connection

```
# nmcli device
DEVICE  TYPE      STATE         CONNECTION
enp0s3  ethernet  connected     Wired connection 1
enp0s8  ethernet  connected     TARJETA1
enp0s9  ethernet  disconnected  --
lo      loopback  unmanaged     --
#
```

- we see that device enp0s9 is not connected

Creating a connection with `nmcli`

- we create a connection named NUEVACONEXION

```
# nmcli con add type ethernet ifname enp0s3 con-name NUEVACONEXION
Connection 'NUEVACONEXION' (df460578-b37d-4228-8b2c-e937864c23b0) successfully
#
```

Creating a connection with `nmcli`

- We configure its static addresses adn activate the conection

```
# nmcli con modify NUEVACONEXION ipv4.addresses 192.168.2.103/24,
          192.168.20.103/24,192.168.200.103/24
# nmcli con modify NUEVACONEXION ipv4.method static
# nmcli con up NUEVACONEXION
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkM
#
```

Creating a connection with `nmcli`

- it shows in `nmcli device`

```
# nmcli dev
DEVICE  TYPE      STATE      CONNECTION
enp0s3  ethernet  connected  Wired connection 1
enp0s8  ethernet  connected  TARJETA1
enp0s9  ethernet  connected  NUEVACONEXION
lo      loopback  unmanaged  --
#
```

Creating a connection with `nmcli`

- it shows in `connection show`

```
# nmcli connection show
NAME                UUID                                  TYPE      DEVICE
Wired connection 1  3c27a730-7940-3cbc-a9be-5f648781ce0b  ethernet  enp0s3
NUEVACONEXION       19426329-47c2-4fb5-a4bc-c48720d556cf  ethernet  enp0s9
TARJETA1            7f5394f0-1d49-40d4-a92a-3e9c279752f0  ethernet  enp0s8
#
```

- this connection configuration is incorporated into Network Manager so it's presistent accross system reboots

Basic network configuration
→linux's netplan

netplan

- some (*new*) linux distros use `netplan`
- netplan uses its own configuration YAML files (in `/etc/netplan` directory) to generate the configuration files for the *network renderers* which are the ones actually configuring the network
- currently two renderers are supported
  - NetworkManager
  - Systemd-networkd

netplan

- after modifying its configuration files we can issue the command `'netplan apply'` that
  - generates the addecuate configuration for one of the *network renderers*
  - makes the *network renderers* reload their configuration
- the commands `'netplan try'` `'netplan generate'` allow us to check the configuration before applying

netplan

- files in `/etc/netplan` directory are processed in order, so they usually are named with names starting with '10_', '20_', ...
- the format of this file, for ipv4 is

```
network:
    version: 2
    renderer: networkd/NetworkManager
    ethernets:
       DEVICE_NAME:
          dhcp4: yes/no
          address: [IP/NETMASK]
          address: [IP/NETMASK]
          ....
          gateway4: GATEWAY
          nameservers:
             addresses: [NAMESERVER, NAMESERVER]
```

sample netplan file

- the following file allows NetworkManager to configure all interfaces using dhcp

```
network:
version: 2
renderer: NetworkManager
```

sample netplan file

- the following file configures several interfaces with systemd-networkd, one of them with dhcp and the others with several aliases

```
network:
    version: 2
    renderer: networkd
    ethernets:
        enp0s3:
            dhcp4: true
        enp0s9:
            addresses:
              -  192.168.1.200/24
              -  192.168.2.200/24
              -  192.168.5.200/24
        enp0s8:
            addresses:
              - 192.168.11.200/24
              - 192.168.12.200/24
              - 192.168.15.200/24
```

Routes

route configuration at boot time

- we'll deal only with the most simple case of routing: A single default route. If we're using dhcp this is configured automatically. If not

solaris	through the `/etc/defaultrouter` file
solaris 11	through the `route -p` command
openBSD	through the `/etc/mygate` file
freeBSD	in the file `/etc/rc.conf`
netBSD	in the file `/etc/rc.conf` or through the `/etc/mygate` file
linux	in the `/etc/network/interfaces` file with the word `gateway` (debian). In the file `/etc/sysconfig/network` (fedora). In the netplan configuration file with *gateway4* option

manipulating the route

- the commands to manipulate the routing table are

solaris	**route** and **routeadm**
BSD	**route**
linux	**route** or **ip route**

- to show the routing table we use **netstat -r**. In linux we can also use **route** or **ip route show** without arguments

Network interface aliasing

interface aliasing

- By interface aliasing we refer to the act of giving a Network Interface Card more than one IP address.
- In **previous** solaris and linux versions (solaris used to call them *logical interfaces*), we would create *'new'* interface with the same_name:number for each new IP to assign to the interface and configure them just like the other interfaces
  - (linux) for eth0 we would use eth0:0, eth0:1,eth0:2 . . . for the different addresses of eth0
  - (solaris $<$ 11) for e1000g0 we would use e1000g0:1, e1000g0:2 e1000g0:3 . . . for the different addresses of e1000g0 (e1000g0:0 would be the interface itself)
- in BSD systems we still use the the option *alias* in `ifconfig`

interface aliasing in debian and derivatives

debian (and derivatives) We have two options

a. We simply add internet addresses to the interface with the *ip* command

```
# ip addr add 192.168.2.100 dev eth0
# ip addr add 192.168.29.18 dev eth0
```

b. we use the old *name:N* with *ifconfig*

```
# ifconfig eth0:1 inet 192.168.2.100
# ifconfig eth0:2 inet 192.168.29.18
```

- Note that address defined with method a), are not seen by *ifconfig*

interface aliasing in debian and derivatives

debian (and derivatives) At boot time, we can use the *name:N* (f.e. eth0:1) approach in `/etc/network/interfaces` or we can assign directly several ip_addresses to the interface

```
iface eth0 inet dhcp

auto eth1
iface eth1 inet static
   address 192.168.1.101
   netmask 255.255.255.0
iface eth1 inet static
   address 192.168.10.101
   netmask 255.255.255.0
iface eth1 inet static
   address 192.168.100.101
   netmask 255.255.255.0
```

interface aliasing

fedora We configure these alias as we would do with a *non-aliased* interface but using the *ip addr* command.

- We simply add internet addresses to the interface

```
# ip addr add 192.168.2.100 dev p2p1
# ip addr add 192.168.29.18 dev p2p1
```

- if we want to get the alias configured at boot time we add IPADDR2, IPADDR3, . . . to the `/etc/sysconfig/network-scripts/ifcgf-`*device* file

```
IPADDR2=192.168.2.150
IPADDR3=192.168.15.22
```

- It would also be possible to create ifcfg-*device:N* files in the `etc/ifconfig/network-scripts` for the aliased interfaces

interface aliasing

openBSD We use the option **alias** of **ifconfig**

```
# ifconfig em0 alias 10.1.2.4 255.0.0.0
```

In the **/etc/hostname.interface_name** file we add one line for each alias of the NIC

```
# cat /etc/hostname.em0
inet 192.168.1.100 255.255.255.0 192.168.1.255
inet alias 10.1.2.4 255.0.0.0
```

interface aliasing

freeBSD We use the option **alias** of **ifconfig**

```
# ifconfig le0 10.1.2.4 255.0.0.0 alias
```

- We add a line to `/etc/rc.conf` to get the alias configured at boot time

```
# cat /etc/rc.conf
ifconfig_le0="inet 192.168.1.3 netmask 255.255.255.0"
ifconfig_le0_alias0="inet 192.168.30.11 netmask 255.255.255.255"
```

interface aliasing

NetBSD We use the option **alias** of **ifconfig**

```
# ifconfig wm0 alias 10.1.2.4 255.0.0.0
```

- To get it done at boot time we can
  - the ifconfig_xxN variables in `/etc/rc.conf`. A semicolon (;) represents several lines

```
# cat /etc/rc.conf
ifconfig_wm1="inet 192.168.1.3 255.255.255.0;alias 192.168.30.11 255.255.255.0"
```

- add alias lines to the file `/etc/ifconfig.ifacename`
- use the file `/etc/aliases` or the ifaliases_xxN variables in `/etc/rc.conf`

interface aliasing

solaris 11 We just use the command `ipadm create-addr` to add one address to an existing interface

```
# ipadm create-addr -T static -a 192.168.5.44  net1/alias0
```

- it will appear in the configuration

```
root@aso:~# ipadm show-addr
ADDROBJ           TYPE     STATE        ADDR
lo0/v4            static   ok           127.0.0.1/8
net0/ipv4         dhcp     ok           10.0.2.15/24
net1/ipv4         static   ok           192.168.1.155/24
net1/alias0       static   ok           192.168.4.44/24
lo0/v6            static   ok           ::1/128
```

Access control

Access control

- By access control we refer to the ability to discard (or reject) not wanted connections
- A machine providing a service is prone to receive attacks to that service.
- We might want to restrict access to that service. We can accomplish this
  - disabling the service
  - at the packet level
  - at the application level (tcpwrappers)

Access control
→stopping and disabling services

Stoping and disabling services

- We can stop any service at any time
  - **BSD systems:** `/etc/rc.d/`*name* `stop`
  - **System V systems:** `/etc/init.d/`*name* `stop`
  - **Linux systemd:** `systemctl stop` *name*`.service`
  - **Solaris** `svcadm disable` *name*

Stoping and disabling services

- We can also *"disable"* a service: That is not having it started when we boot the machine
  - **BSD systems:** Through the appropiante variables in `/etc/rc.conf` (or `/etc/default/rc.conf`)
  - **System V system:** Through the links from `/etc/rcN.d` to `/etc/init.d/` (N being the runlevel)
  - **Linux systemd** `systemctl disable` *name*`.service`
  - **Solaris** `svcadm disable` *name*

Access control
→Access control at the packet level

Access control at the packet level

- Every modern operating system has a packet filtering framework that has rules that allow us to select packets depending on
  - protocol family, protocol, port ...
  - type of packet
  - source or destination address
  - incoming or outgoing interface
  - ...

Access control at the packet level

- Those rules also allow us to do things with the packets:
  - accept
  - reject
  - drop
  - change its source address
  - change its destination address or port
  - ...

Access control at the packet level

- Unfortunately, different unixes have different packet filtering systems, with diferent rules selection packets and differen rules syntax.
  - **PF** in open OpenBSD
  - **IPFW**, **PF** or **IPFILTER** in FreeBSD
  - **NPF** in NetBSD
  - **PF** in Solaris
  - **nftables** (previously **iptables** in Linux) (*ipchains* completely deprecated)

Access control at the packet level

- When packets for a connection are accepted, the connection is stablished
- Rejected packets produce a message of type *"connection refused"*
- Dropped packets produce no response from the machine, and the machine trying to stablish connection finally will report a *time out* type of message

Access control
→Access control at the application level

Access control at the application level

- If we cannot (or we don't want to) control access at the packet level, we can, if we wish to, do that the application level.
- In this case, is the application itself the one that controls the access but **after the connection has been stablished**
- This is usually done by what we call the *tcpwrappers* or *tcpd*

application level: tcpwrappers

- the configuration for the *tcpwrappers* resides in the files `/etc/hosts.allow` and `/etc/hosts.deny`
- the manual page `hosts_access` documents the use of these files
- the basic syntax is lines in the form `daemon:client_list`. For example

```
sshd: 193.144.51.  192.168.2.
ftpd: ALL
```

  - Access will be granted when a (daemon,client) pair matches an entry in the `/etc/hosts.allow` file.
  - Otherwise, access will be denied when a (daemon,client) pair matches an entry in the `/etc/hosts.deny` file.
  - Otherwise, access will be granted.

application level: tcpwrappers

- Some implemetations allow for a more complex working. For example

solaris
- the format of a line is daemon:client_list [:shell_command] (the shell command being optional)
- the client list allows for operators such as EXCEPT

FreeBSD
- only the `hosts.allow` file is used
- the format of a line is daemon:client_list: option: option
- option can be allow, deny, spawn (create a new process), twist (replace the process code) ...
- the client list allows for operators such as EXCEPT

Access control at the application level

- For applications to use this kind control they have to be compiled and linked with the *tcpwrap o tcpd* library. That can be checked `ldd`: it will usually appear as `libtcpd...` or `libwrap...`
- If the application has not been compiled and linked with *tcpwrap o tcpd* we can have it called from `inetd` provided that `inetd` has.
- If nor `inetd` neither the application have been compiled and linked with *tcpwrap o tcpd* we can have `inetd` call `tcpd` and the `tcpd` call the application (classical approach for legacy applications)

Access control
→Access control without `inetd`

Access control without `inetd`

- Today it is customary for servers to listen ports directly without having to rely on `inetd` to call them
- Access control is through files `/etc/hosts.allow` and `/etc/hosts.deny` (or `/etc/hosts.allow` only)
- Servers need to have been linked with tcpwrappers support (which is the usual thing to have)

Access control without inetd

- We can check if that is the case with `ldd` and look for something like `libtcpd` or `libwrap` in the output

```
$ ldd /usr/sbin/sshd
/usr/sbin/sshd:
 libpam.so.6 => /usr/lib/libpam.so.6 (0x80028e000)
 libprivatessh.so.5 => /usr/lib/libprivatessh.so.5 (0x80029e000)
 libutil.so.9 => /lib/libutil.so.9 (0x800346000)
 libbsm.so.3 => /usr/lib/libbsm.so.3 (0x80035e000)
 libblacklist.so.0 => /usr/lib/libblacklist.so.0 (0x80037c000)
 libgssapi_krb5.so.10 => /usr/lib/libgssapi_krb5.so.10 (0x800382000)
 libgssapi.so.10 => /usr/lib/libgssapi.so.10 (0x8003a4000)
 libkrb5.so.11 => /usr/lib/libkrb5.so.11 (0x8003b1000)
 libwrap.so.6 => /usr/lib/libwrap.so.6 (0x800433000)
 libcrypto.so.111 => /lib/libcrypto.so.111 (0x80043f000)
 libc.so.7 => /lib/libc.so.7 (0x800734000)
 .........
```

inetd

inetd configuration

- **inetd** is called the internet superserver
- Some internet services listen directly to their corresponding port, others are started by **inetd**
- When a conexion request arrives on a designated port, **inetd** starts the appropiated server program
- This allows for server programs to run only when needed, thus saving resources on the system
- Two files control the working of **inetd**
  - `/etc/services`
  - `/etc/inetd.conf`

inetd
→inetd configuration

/etc/services

- /etc/inet/services on some systems
- this file has a mapping between the port numbers and protocol to the services names. Info can be found in the services man page. A fragment from an acual /etc/services is shown

```
ftp             21/tcp
fsp             21/udp          fspd
ssh             22/tcp                          # SSH Remote Login Protocol
ssh             22/udp
telnet          23/tcp
smtp            25/tcp          mail
time            37/tcp          timserver
time            37/udp          timserver
rlp             39/udp          resource        # resource location
nameserver      42/tcp          name            # IEN 116
whois           43/tcp          nicname
```

/etc/inetd.conf

- This file associates the service name to the program actually providing the service
- The format for one line of this file is

```
service_name  socket_type  protocol  wait/nowait  user.group  program  args
```

/etc/inetd.conf

- As lines started with the # are treated as comments, we can disable one service, by simply comenting out the corresponding line
- Example of the `telnetd` service disabled

```
#telnet    stream  tcp     nowait  root    /usr/sbin/in.telnetd    in.telnetd
```

/etc/inetd.conf

- debian linux does not include `inetd`, it can be installed as package `openbsd-inetd` (usually as a dependence of other network packages)
- BSD systems `inetd` does not start by default (variables inetd and inetd_flags of `rc.conf`). Some provide a sample configuration file at `/etc/examples/inetd.conf`

inetd in fedora linux

- Fedora linux, (as do some distrbutions of linux), does not include `inetd`. It includes `xinetd`, a inetd replacement
- However, it is no necesary to use `xinetd` to use such services such as *telnetd* or *ftpd*
- For example, should have we installed `pure-ftpd` as the ftp server we can enable that ftp service by doing

```
# systemctl enable pure-ftpd
# systemctl start pure-ftpd
```

inetd and tcpwrappers

inetd and tcpwrappers
→tcpwrappers

tcpwrappers

- An aditional layer can be placed between `inetd` and the server program to perform access control based on host name, network address or ident queries
- This layer is usally called `tcpwrappers` or, by the name of the program, `tcpd`.
  - the program `tcpd` gets called by `inetd` and receives the server to start as a parameter
  - `tcpd` checks its configuration files to see if the access must be granted or denied
  - in case the access is granted `tcpd` starts the server program supplied as parameter

tcpwrappers

- the corresponding line for this `telnetd` server using *tcpwrappers* and running for user telnet would look like this. The second line would run the service for user root and without tcpwrappers

```
telnet   stream  tcp  nowait telnet  /usr/sbin/tcpd  /usr/sbin/in.telnetd
telnet   stream  tcp  nowait root    /usr/sbin/in.telnetd  in.telnetd
```

tcpwrappers

- **As seen before** the configuration for the *tcpwrappers* resides in the files `/etc/hosts.allow` and `/etc/hosts.deny`
- the manual page `hosts_access` documents the use of these files
  - Access will be granted when a (daemon,client) pair matches an entry in the `/etc/hosts.allow` file.
  - Otherwise, access will be denied when a (daemon,client) pair matches an entry in the `/etc/hosts.deny` file.
  - Otherwise, access will be granted.

tcpwrappers

- `xinetd` can also implement this access control as can the servers that use *libtcpwrappers* (*libtcpd*) directly
- It is important to remember that implementations differ among different unixes, and that we must **always** check the `hosts_access` (or `hosts.allow`) man page

inetd and tcpwrappers
→inetd and tcpwrappers in Solaris

`inetd` and tcpwrappers in Solaris

- Starting with Solaris 10, the `inetd` services have been integrated in the *smf* (Sevices Managemnent Facility) mainframe
- The file `/etc/inetd.conf` exists on the system but any changes made to it do not change the system behaviour
- If we add a service to `/etc/inetd.conf`, we can convert into a *smf manifest* to me managed by *smf* using the command `inetconv`

`inetd` and tcpwrappers in Solaris

- After that, the service can be managed using the commands **svcadm** and **inetadm**
- *tcpwrappers* can also be activated for all or some of the *inetd* services as a property of *inetd*

inetd and tcpwrappers in Solaris

- to see *inetd* properties

```
# svcprop inetd
# inetadm -p
```

- to list the services managed through inetd

```
# inetadm
```

- to list an *inetd* service

```
# inetadm -l servce_name
```

- to enable or disable an *inetd* service

```
# inetadm -e|-d  service_name
```

- services can also be disabled with *svcadm*

inetd and tcpwrappers in Solaris

- *tcp_wrapers* is treated as a property of *inetd* so, to enable it we must modify that property of *inetd*

  ```
  # inetadm -M tcp_wrappers=TRUE
  ```

- *tcp_wrapers* can be enabled on a per service basis modifying just the property of that service. For example to enable the host access control ONLY for the telnet service we'd do

  ```
  # inetadm -M tcp_wrappers=FALSE
  ```

  to disable *tcp_wrappers*. And then to enable *tcp_wrappers* for the telnet service

  ```
  # inetadm -m telnet tcp_wrappers=FALSE
  ```