Linux Routers and Community Networks

Laboratory Manual

Llorenç Cerdà-Alabern
Universitat Politènica de Catalunya
Barcelona, Spain

http://personals.ac.upc.edu/llorenc
llorenc@ac.upc.edu

July, 2015
Contents

Preface and acknowledgments 4

1 Basic Network Configuration 5
   1.1 Description .......................................................... 5
   1.2 Unix basic commands ............................................. 5
   1.3 Linux basic networking configuration ......................... 7
   1.4 Domain Name System, DNS ...................................... 9
   1.5 WiFi ..................................................................... 10
   1.6 Troubleshooting ..................................................... 11
   1.7 OpenWrt Essentials ................................................. 12
   1.8 Lab setup ............................................................... 19

2 RIP and OSPF 22
   2.1 Description .......................................................... 22
   2.2 IOS fundamentals .................................................. 22
   2.3 Quagga set up ....................................................... 22
   2.4 Basic commands .................................................... 23
   2.5 RIP review ............................................................ 24
   2.6 RIP configuration .................................................. 25
   2.7 RIP Lab setup ....................................................... 26
   2.8 OSPF review ........................................................ 28
   2.9 OSPF configuration ................................................. 28
   2.10 OSPF Lab setup ................................................... 29

3 Firewall configuration 31
   3.1 Description .......................................................... 31
   3.2 Iptables operation .................................................. 31
   3.3 Creation of rules with the iptables command ............... 32
   3.4 Commands ............................................................ 32
   3.5 Expressions ........................................................... 32
   3.6 Types of rules ....................................................... 33
   3.7 Quick edition of iptables configuration ..................... 34
   3.8 iptables Lab setup ............................................... 35

4 Community Networks 38
   4.1 Introduction ........................................................ 38
   4.2 WiFi Ad-Hoc Mode ................................................ 38
   4.3 Create a node in Guifi.net ........................................ 39
   4.4 qMp ................................................................. 39
   4.5 Lab setup ............................................................. 43

5 Network Management 45
   5.1 Introduction ........................................................ 45
   5.2 Nagios Installation ................................................ 45
   5.3 Nagios Configuration ............................................ 46
   5.4 Nagios Plugins .................................................... 46
Preface and acknowledgments

On May 2015 Eva Vidal and Jesús Berdun proposed me to participate in a project supported by the Centre de Cooperació per al Desenvolupament, CCD\(^1\) of Universitat Politècnica de Catalunya, UPC\(^2\). The aim of the project was carrying out educational and deployment activities at Mekelle Institute of Technology (MIT), Ethiopia. The activities would be in the area of computer networking and wireless community networks. During the project I would go to Mekelle together with 3 students from UPC.

I agreed to participate in the project giving a summer school of one week at MIT, Ethiopia, and I prepared this laboratory manual to this purpose. The laboratory classes are addressed to students with theoretical background in Computer Networks. The laboratories have been prepared to cover basic concepts in Linux routers administration. More specifically, the course focuses on OpenWrt. OpenWrt\(^3\) is a highly extensible GNU/Linux distribution originally designed for embedded devices (typically wireless routers). OpenWrt is free, open source and continuously improved by a wide and very active community. This is demonstrated by the fact that the list of routers supporting OpenWrt is continuously growing\(^4\). Since the first OpenWrt release on January 2004, the number of packages and contributions have constantly increased. Nowadays OpenWrt is extremely rich in networking features and it is supported by hardware ranging from low performance ADSL routers to specialized router boards able to work at Gbps.

I was myself attracted by wireless community networks as a way to do experimental research in computer networks. In November 2012 I put one antenna in the roof of my house in Barcelona, and I joined a wireless community available in my suburb called Guifisants\(^5\). Since that time I reach the Internet through it, with better performance that what I had with a conventional ADSL line. I have written several papers about wireless community networks, and my interest on this topic has constantly increased. Indeed, at the time of writing I already have 4 antennas in my roof\(^6\). In Guifisants we run a branch of OpenWrt called qMp, from Quick Mesh Project. Currently, Guifisants has around 60 nodes. Guifisants is part of a larger community network started in 2004 which currently has more than 28,000 operative nodes deployed all over Spain called Guifi.net\(^7\). However, in contrast to most Guifi.net infrastructure, which has been deployed using the traditional OSPF and BGP routing protocols used in the Internet, in Guifisants it is used a mesh routing protocol designed specifically for wireless networks called BMX6.

In order to prepare the labs I have used my experience in the laboratories I do as teacher of Computer Networks at the Facultat Informàtica de Barcelona\(^8\), and what I have learned as a community network activist. The manual is very succinct, so it can hardly be used as a self teaching document. Nevertheless, it might be useful in preparing computer networks labs, or as cheatsheet for users starting with OpenWrt.

The sessions have been prepared to be carried out in a lab equiped with 30 PCs running linux, Internet access and 10 routers like TP-LINK TL-WDR4300. The sessions are foreseeing to be followed by 30 to 50 students, who will be distributed in 10 groups, each using 3 PCs and 1 router.

I conclude giving my acknowledgments to Eva and Jesús, for contacting me to participate in this challenging project; to CCD and the people that economically supports CCD, making this project possible; to the EU CONFINE project\(^9\) for its support purchasing the Ubiquiti Nanostations; to our contacts in MIT, Ethiopia: Desta Gebre and Meles G., Dean of MIT for their trust and support; to the students that helped preparing the trip and come with me to Mekelle: Massiel Coves, Joan Pallarès-Sadó and Oriol Lleopart; and to all other people that I’m sure will help us during our stay in Mekelle.


\(^{1}\)http://www.upc.edu/ccd
\(^{2}\)http://www.upc.edu
\(^{3}\)http://wiki.openwrt.org/toh/start
\(^{4}\)http://guifisants.net
\(^{5}\)http://dsg.ac.upc.edu/node/608
\(^{6}\)http://guifi.net
\(^{7}\)http://www.fib.upc.edu
\(^{8}\)http://confine-project.eu
1.1 Description
This Lab consists of installing OpenWrt, accessing OpenWrt using command line and manually configuring networking interfaces and static routing.

1.2 Unix basic commands

1.2.1 Directories
- cd: Change directory.
- mkdir: make directory, remove directory: rmdir
- ls: list directory.
- rm: remove file, remove directory and its contents: rm -r directory
- chown: change owner file/directory.
- cat, more, less: dump file content.
- df -h: list disk partitions.
- grep: globally search regular expression (regex) and print, filter file content using regex.

```
Listing 1.1: grep example, regex '.' means any character.
~# cat network | grep eth.
```  
```
Listing 1.2: find examples.
~# find . -name network
./config/network
./init.d/network
~# find /etc -name *.conf -exec grep nameserver {} \; -print
nameserver 127.0.0.1 /etc/resolv.conf
```

- find: find file. Examples:

```
Listing 1.3: sed examples.
~# sed 's/day/night/' <old >new
~# iw wlan1 scan | sed -n '/BSS/;/SSID/p
BSS 02:ca:ff:ee:ba:be(on wlan1) -- joined
  TSF: 0 usec (0d, 00:00:00)
  freq: 5825
  beacon interval: 100 TUs
  capability: IBSS (0x0002)
  signal: 0.00 dBm
  last seen: 1687530 ms ago
  SSID: qMp
BSS 02:ca:ff:ee:ba:be(on wlan1)
  TSF: 28625233345727246586 usec (33131057d, 02:42:52)
  freq: 5240
  beacon interval: 100 TUs
  capability: IBSS (0x0002)
```

- sed: stream editor. As grep, it uses regex. The usage is more complex than previous commands. In the following there are two typical usage examples: First it creates file new by substituting (command s) day by night in file old. Then sed is used to find text blocks that match 2 addresses: it is done a scan on interface wlan1, and sed prints blocks of text (command p) starting with BSS at the beginning of the line and ending with SSID.
1.2.2 Processes
- ps, top: show running processes.
- kill pid: kill process with process id pid.
- killall cmd: kill command cmd.

1.2.3 Basic editing with vi
- Two modes of operation:
  - insert mode: enter text.
  - command mode: issue editing commands.
- To switch into insert mode:
  - i insert before cursor.
  - I insert beginning of line.
  - a append after cursor.
  - A append at end of line.
- To switch into command mode, press ESC. If you are unsure which mode you are in, press ESC.
- Saving and exiting:
  - :q quit.
  - :q! quit discarding changes.
  - :w save file. After each change, it is recommended to save (there is no undo). To abort changes and quit :q!.
  - :w file write to file.
- Deleting:
  - x character.
  - dw word.
  - dd line.
- Copy paste:
  - yy yank (copy) line
  - yy yank word
  - y$ yank to the end of line
  - v starting point for yank (y) or delete (d).
  - p paste word after cursor, or line below.
  - P paste word before cursor, or line above.
- Moving:
  - 0 beginning of line.
  - $ end of line.
  - H top of screen.
  - gg beginning of file.
  - L last line of screen.
  - G end of file.
  - :n go to line n.
- Repeat command, e.g. remove 3 characters: 3x.

1.2.4 ssh and scp
Secure shell (ssh) allows a remote command line shell on Linux/Unix systems, like OpenWrt. In contrast to telnet, ssh provides encryption and better authentication. Here there is a brief summary, more info, e.g. in [17]. Secure copy (scp) uses ssh to send files between hosts. OpenWrt comes with small size ssh server daemon called Dropbear.
ssh session to configure the router

- OpenWrt
- Dropbear ssh server

- Configuration files: ~/.ssh

```
Listing 1.4: Connect to IP.
~# ssh 192.168.1.1 -l root
```

```
Listing 1.5: Generates key pair (id_rsa, id_rsa.pub).
~# ssh-keygen -t rsa
```

ssh key pair can be used to connect without password appending id_rsa.pub to remote file .ssh/authorized_keys, as in the following:

```
Listing 1.6: Connect without password.
~# cat id_rsa.pub | ssh root@192.168.1.1 'cat - >> .ssh/authorized_keys'
```

```
Listing 1.7: Recursively copy to remote host.
~# scp -r config root@192.168.1.1:/tmp
```

1.3 Linux basic networking configuration

1.3.1 ifconfig

```
Listing 1.8: List all interfaces.
~# ifconfig -a
```

```
Listing 1.9: Assign IP address.
~# ifconfig eth0 102.168.1.2 netmask 255.255.255.240
```

1.3.2 route

```
Listing 1.10: List routing table.
~# route -n
```

```
Listing 1.11: Add a default route.
~# route add default gw 192.168.2.1
```

```
Listing 1.12: Add/delete a static route.
~# route add -net 10.0.0.0 netmask 255.255.255.0 gw 192.168.2.1
~# route del -net 10.0.0.0 netmask 255.255.255.0 gw 192.168.2.1
```

1.3.3 ip

The traditional ifconfig and router UNIX command have been superseded in Linux by ip command in Linux. It is convenient to know both, since ip might not be available in very basic installations. On the other hand, some advanced Linux Routers and Community Networks Lab 1: Basic Network Configuration
Networking features are only available through \texttt{ip}.

Some conventions: it is enough to write unambiguous arguments (e.g. l instead of link), show and list argument are equivalent. Type \texttt{ip help} for global help, or \texttt{ip OBJECT help}, for help about \texttt{OBJECT}. Examples:

<table>
<thead>
<tr>
<th>Listing 1.13: \texttt{ip}.</th>
</tr>
</thead>
<tbody>
<tr>
<td># list IPv6 addresses</td>
</tr>
<tr>
<td>root@OpenWrt:~# \texttt{ip -6 a l}</td>
</tr>
<tr>
<td>1: lo: &lt;LOOPBACK,UP,LOWER_UP&gt; mtu 65536</td>
</tr>
<tr>
<td>inet6 ::1/128 scope host</td>
</tr>
<tr>
<td>valid.lft forever preferred.lft forever</td>
</tr>
<tr>
<td>2: eth0: &lt;BROADCAST,MULTICAST,UP,LOWER_UP&gt; mtu 1500 qelen 1000</td>
</tr>
<tr>
<td>inet6 fe80::12fe:edff:feaf:635e/64 scope link</td>
</tr>
<tr>
<td>valid.lft forever preferred.lft forever</td>
</tr>
<tr>
<td>5: eth0.1@eth0: &lt;BROADCAST,MULTICAST,UP,LOWER_UP&gt; mtu 1500</td>
</tr>
<tr>
<td>inet6 fe80::12fe:edff:feaf:635e/64 scope link</td>
</tr>
<tr>
<td>valid.lft forever preferred.lft forever</td>
</tr>
<tr>
<td>6: eth0.2@eth0: &lt;BROADCAST,MULTICAST,UP,LOWER_UP&gt; mtu 1500</td>
</tr>
<tr>
<td>inet6 fe80::12fe:edff:feaf:635e/64 scope link</td>
</tr>
<tr>
<td>valid.lft forever preferred.lft forever</td>
</tr>
</tbody>
</table>

\# assign an ip address

root@OpenWrt:~# \texttt{ip a a 192.168.50.5 dev eth1}

\# delete an ip address

root@OpenWrt:~# \texttt{ip a d 192.168.50.5 dev eth1}

\# list routing table

root@OpenWrt:~# \texttt{ip r l}

default via 147.83.34.2 dev eth1

default via 147.83.34.2 dev eth1 proto static metric 1024

147.83.34.0/24 dev eth1 proto kernel scope link src 147.83.34.125

192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.1

\# add a routing table entry

root@OpenWrt:~# \texttt{ip r a 10.10.20.0/24 via 192.168.1.1 dev eth0}

\# delete a routing table entry

root@OpenWrt:~# \texttt{ip r d 10.10.20.0/24 via 192.168.1.1 dev eth0}

\# add default route

root@OpenWrt:~# \texttt{ip r a default via 192.168.1.1}

1.3.4 \texttt{arp}

<table>
<thead>
<tr>
<th>Listing 1.14: List stations from the network with whom datagrams have been exchanged.</th>
</tr>
</thead>
<tbody>
<tr>
<td># legacy arp command</td>
</tr>
<tr>
<td>#-arp</td>
</tr>
<tr>
<td>IP address</td>
</tr>
<tr>
<td>192.168.1.234</td>
</tr>
<tr>
<td>192.168.1.2</td>
</tr>
<tr>
<td>192.168.1.20</td>
</tr>
</tbody>
</table>

\# using ip

\#-ip neigh l

192.168.1.234 dev br-lan lladdr 00:24:e8:2c:74:e2 STALE

192.168.2.1 dev wlan0 lladdr d4:ca:6d:ad:0:0:cd:0 STALE

192.168.1.20 dev wlan0 lladdr dc:9f:db:3:81:cd STALE

1.3.5 Check for connectivity: ping, traceroute, mtr

<table>
<thead>
<tr>
<th>Listing 1.15: ping.</th>
</tr>
</thead>
<tbody>
<tr>
<td># ping 8.8.8.8</td>
</tr>
<tr>
<td>PING 8.8.8.8 (8.8.8.8): 56 data bytes</td>
</tr>
<tr>
<td>64 bytes from 8.8.8.8: seq=0 ttl=45 time=54.281 ms</td>
</tr>
<tr>
<td>64 bytes from 8.8.8.8: seq=1 ttl=45 time=58.195 ms</td>
</tr>
<tr>
<td>^C</td>
</tr>
<tr>
<td>--- 8.8.8.8 ping statistics ---</td>
</tr>
<tr>
<td>2 packets transmitted, 2 packets received, 0% packet loss</td>
</tr>
<tr>
<td>round-trip min/avg/max = 54.281/56.238/58.195 ms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing 1.16: traceroute.</th>
</tr>
</thead>
<tbody>
<tr>
<td># traceroute -n 8.8.8.8</td>
</tr>
<tr>
<td>traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 38 byte packets</td>
</tr>
<tr>
<td>1 10.228.205.1 5.807 ms 2.924 ms 2.875 ms</td>
</tr>
</tbody>
</table>
### Domain Name System, DNS

The OpenWrt router has a DNS/DHCP server provided by dnsmasq. It returns the IP addresses defined in `/etc/hosts`, and by default uses the file `/tmp/resolv.conf.auto` for resolving internet’s DNS queries:
1.5 WiFi

- To configure WiFi we shall use the web interface. See sections 1.7.7 and 1.7.8.
1.6 Troubleshooting

1.6.1 tcpdump

tcpdump examples (see figure 1.1):

Listing 1.25: Checking activity in eth0.

```bash
~# tcpdump -ni eth0
```

Listing 1.26: Capturing packets from a given host.

```bash
~# tcpdump -ni eth0 host 192.168.2.1
```

Listing 1.27: Capturing SYN and FIN packets: 2 first less significant bits of byte 13 (see figure 1.2.)

```bash
~# tcpdump -ni wlan0 "tcp[13] & 3 != 0"
```

Figure 1.1: tcpdump of a TCP segment.

Figure 1.2: TCP header.
Listing 1.28: Capturing dhcp packets in verbose mode.

```bash
# tcpdump -vni br-lan port 67
tcpdump: listening on br-lan, link-type EN10MB (Ethernet), capture size 65525 bytes
14:24:56.571518 IP (tos 0, ttl 64, id 0, offset 0, flags [none], proto UDP (17), length 328)
0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from 10:fe:ed:af:63:5e, length 300, xid 0x6aaf8c2c, Flags [none]
  Vendor-rfc1048 Extensions
  Magic Cookie 0x63825363
  DHCP-Message Option 53, length 1: Discover
  MSZ Option 57, length 2: 576
  Parameter-Request Option 55, length 7:
    Subnet-Mask, Default-Gateway, Domain-Name-Server, Hostname
  Domain-Name, BR, NTP
  Vendor-Class Option 60, length 12: "udhcp 1.22.1"
14:24:56.573758 IP (tos 0, ttl 64, id 42768, offset 0, flags [none], proto UDP (17), length 328)
192.168.1.1.67 > 192.168.1.232.68: BOOTP/DHCP, Reply, length 300, xid 0x6aaf8c2c, Flags [none]
  Your-IP 192.168.1.232
  Server-IP 192.168.1.1
  Vendor-rfc1048 Extensions
  Magic Cookie 0x63825363
  DHCP-Message Option 53, length 1: Offer
  Server-ID Option 54, length 4: 192.168.1.1
  Lease-Time Option 51, length 4: 43200
  RN Option 58, length 4: 21600
  RB Option 59, length 4: 37800
  Subnet-Mask Option 1, length 4: 255.255.255.0
  BR Option 28, length 4: 192.168.1.255
  Default-Gateway Option 3, length 4: 192.168.1.1
  Domain-Name-Server Option 6, length 4: 192.168.1.1
  Domain-Name Option 15, length 3: "lan"
```

1.6.2 System logs

In OpenWrt:

Listing 1.29: logread.

```bash
# logread
Tue Jun 9 13:25:41 2015 daemon.notice netifd: Interface 'wwan' is enabled
Tue Jun 9 13:25:41 2015 kern.info kernel: [ 1003.310000] wlan0: RX AssocResp from 0c:82:68:cd:a1:06 (capab=0x431 status=0 aid=2)
Tue Jun 9 13:25:41 2015 kern.info kernel: [ 1003.320000] wlan0: associated
Tue Jun 9 13:25:41 2015 daemon.notice netifd: Network device 'wlana' link is up
Tue Jun 9 13:25:41 2015 daemon.notice netifd: Interface 'wlana' has link connectivity
Tue Jun 9 13:25:41 2015 daemon.notice netifd: Interface 'wlana' is setting up now
Tue Jun 9 13:25:42 2015 daemon.notice netifd: wlan0 (2489): udhcpc (v1.22.1) started
Tue Jun 9 13:25:42 2015 daemon.notice netifd: wlan0 (2489): Sending discover...
Tue Jun 9 13:25:45 2015 daemon.notice netifd: wlan0 (2489): Sending select for 192.168.2.233...
Tue Jun 9 13:25:45 2015 daemon.notice netifd: wlan0 (2489): Lease of 192.168.2.233 obtained, lease time 43200
Tue Jun 9 13:25:45 2015 daemon.notice netifd: Interface 'wlana' is now up
Tue Jun 9 13:25:45 2015 user.notice firewall: Reloading firewall due to ifup of wlan0 (wlan0)
Tue Jun 9 13:26:15 2015 daemon.info dnsmasq[1210]: reading /tmp/resolv.conf.auto
Tue Jun 9 13:26:15 2015 daemon.info dnsmasq[1210]: using local addresses only for domain lan
Tue Jun 9 13:26:15 2015 daemon.info dnsmasq[1210]: using nameserver 192.168.1.1#53
Tue Jun 9 13:26:15 2015 daemon.info dnsmasq[1210]: using nameserver 192.168.2.1#53
```
1.7.1 Flashing OpenWrt

2. Download the image [https://downloads.openwrt.org](https://downloads.openwrt.org).
3. Flash following instructions for your model.
4. Default IP address: 192.168.1.1

In the following we assume Barrier Breaker 14.07.

1.7.2 Accessing an OpenWrt router using link local IPv6

When installing an OpenWrt router it is convenient to access the router through the link local IPv6 address of the router interface. Link local IPv6 is computed by a standard algorithm from the interface MAC address [6, Appendix A]. Therefore, this address will not change, even if the firmware is reflashed. Using the link local IPv6 address it is not necessary to know the default IPv4 address of the router, and it can be safely changed during the initial configuration.

1. Check the IPv6 link local of the local interface:

```
~# ip -6 a l dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc 9000
    inet6 fe80::250:daff:fec9:6ec7/64 scope link
      valid_lft forever preferred_lft forever
```

2. If no IPv6 link local is assigned, check that the kernel variable `net.ipv6.conf.eth0.disable_ipv6` is set to 0:

```
~# sysctl -a | grep net.ipv6.conf.eth0.disable_ipv6
net.ipv6.conf.eth0.disable_ipv6 = 0
```

3. ping to ff02::1 *All nodes on the local network segment* to figure out the IPv6 link local address of the OpenWrt router:

```
~# ping6 -c 2 ff02::1%eth0
PING ff02::1%eth0(ff02::1) 56 data bytes
64 bytes from fe80::12fe:edff:feaf:635e: icmp_seq=1 ttl=64 time=0.279 ms
64 bytes from fe80::12fe:edff:feaf:635e: icmp_seq=2 ttl=64 time=0.029 ms (DUP!)
--- ff02::1%eth0 ping statistics ---
2 packets transmitted, 2 received, +1 duplicates, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.027/0.111/0.279/0.118 ms
```

Alternatively, can try to ping to IPv6 multicast all routers:

```
~# ping6 -c 2 ff02::1%eth0
PING ff02::1%eth0(ff02::1) 56 data bytes
64 bytes from fe80::12fe:edff:feaf:635e: icmp_seq=1 ttl=64 time=0.279 ms
--- ff02::1%eth0 ping statistics ---
2 packets transmitted, 2 received, +1 duplicates, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.027/0.111/0.279/0.118 ms
```

4. First we need to assign a password using `telnet` (otherwise the node is not accessible using `ssh`):

```
~# telnet fe80::12fe:edff:feaf:635e%eth0
root@OpenWrt:/# passwd
Changing password for root
New password: 13f
Bad password: too short
Retype password: 13f
Password for root changed by root
root@OpenWrt:/#
```

5. The URL is [http://[fe80::12fe:edff:feaf:635e%eth0](http://[fe80::12fe:edff:feaf:635e%eth0)], but most browsers does not support link local IPv6 addresses. To solve this problem we can use the port forwarding feature of ssh (option -L). For instance, the following...
command connects to host `fe80::12fe:edff:feaf:635e%eth0` and use it to forward connections to local port 8080 to port 80 and host `::1`, which is the loopback address, i.e. the same remote host.

Listing 1.35: Forwarding ports with `ssh`.

```bash
~# ssh -L 8080::1:80 fe80::12fe:edff:feaf:635e%eth0 -l root
```

The authenticity of host 'fe80::12fe:edff:feaf:635e%eth0 (fe80::12fe:edff:feaf:635e%eth0)' can't be established.

RSA key fingerprint is b1:b5:d3:f0:1b:03:5c:dc:ff:bd:2a:c5:14:45:c3:76.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'fe80::12fe:edff:feaf:635e%eth0' (RSA) to the list of known hosts.

BusyBox v1.22.1 (2014-09-20 22:01:35 CEST) built-in shell (ash)

Enter 'help' for a list of built-in commands.

root@fe80::12fe:edff:feaf:635e%eth0's password: 13f

BusyBox v1.22.1 (2014-09-20 22:01:35 CEST) built-in shell (ash)

Enter 'help' for a list of built-in commands.

root@fe80::12fe:edff:feaf:635e%eth0's password: 13f

Warning: Permanently added 'fe80::12fe:edff:feaf:635e%eth0' (RSA) to the list of known hosts.

6. Now we can use `http://localhost:8080` to access the OpenWrt router:

1.7.3 OpenWrt Network Interfaces


The Linux kernel distinguishes two types of interfaces:

- **Physical Network Interfaces:** As soon as the device driver is loaded into the Kernel a corresponding physical network interface becomes present, e.g. `eth0`, `wlan0`, etc.
- **Virtual Network Interfaces:** Introduced for the sake of flexibility. Associated with a physical network interface, another virtual interface, or be stand alone such as the loopback interface `lo`.

Linux interface tools (some):

- `ip`: advanced features not available through the legacy `ifconfig` and `route`.
- `vconfig`: VLAN administration.
- `brctl`: bridge administration.

Notes:

- The Unified Configuration Interface, UCI, is a small C utility designed to centralize configuration in OpenWrt.
- `/etc/config/network` is the network configuration file.
- `/etc/config/wireless` is the wireless configuration file.
- UCI creates an abstraction layer for configuring network interfaces: In `/etc/config/network` you allocate a name like `lan` or `wan`. Then this name is consistently used through the entire UCI configuration.

In the following there are the virtual interfaces created by OpenWrt installed in a TL-WDR4300 router (left), and the physical interfaces associated with the LAN interface (right).
1.7.4 Installing additional packages
1. You need Internet connectivity System -> Software -> Update list -> Available packages -> Find tcpdump

1.7.5 netperf

netperf\(^1\) allows throughput measurements. Install the package netperf as explained in section 1.7.4 and activate the server:

```
System -> Startup -> netserver -> Enabled & Start
```

In the following netperf is used to estimate the throughput of a link using the IPv6 link local with a 1 s test. The result is 92.75 Mbps.

```
~# netperf -l 1 -H fe80::12fe:edff:feaf:635e%eth0
MIGRATED TCP STREAM TEST from ::0 (::) port 0 AF_INET6 to fe80::12fe:edff:feaf:635e%eth0 () port 0 AF_INET6 : demo
Recv Send Send Send Socket Socket Message Elapsed
Size Size Size Size bytes bytes bytes secs. 10^6bits/sec
87380 16384 16384 1.05 92.75
```

1.7.6 Gnu screen

gnu screen\(^2\) is a useful tool that allows multiplexing a physical terminal in different windows. This allows executing processes in parallel. For instance, ping in one window and capture the packets with tcpdump in another one. It is convenient to create the .screenrc initialization file of Listing 1.37, which adds a footer bar highlighting the title of the window which is currently selected.

\(^1\)Netperf home page: http://www.netperf.org
\(^2\)https://www.gnu.org/software/screen
Listing 1.37: Gnu screen initialization file.

```
# echo 'caption always "%{= kw}%-= BW% %=n %t%{-}%+w %-= @%H - %LD %d %%LM - %c"' > .screenrc
```

The screen basic commands are the following (C-a c means type Control and a simultaneously, release and type c):

- **C-a c** create new window.
- **C-d** close window.
- **C-a 0** change to window number 0 (likewise for windows 0 to 9).
- **C-a p** change to previous window.
- **C-a n** change to next window.
- **C-a k** kill current window.
- **C-a A** rename current window.

### 1.7.7 Configuring a WiFi AP

1. **Network -> wifi -> Edit**
2. **Advanced Settings**

   ![Advanced Settings](image)

3. **Save & Apply, Enable**

   ![Save & Apply](image)

Note: see appendix A for a list of 802.11 channels.
1.7.8 Configuring a wifi station

1. Network -> wifi
2. scan
3. Join Network, assign to firewall-zone lan
4. submit.

1.7.9 Configuring VLANs

We will use the OpenWrt web interface: Network -> Switch (figure 1.3).

Figure 1.3: Default TP-Link TL-WDR4300 VLAN configuration with OpenWrt.
Listing 1.38: Default TP-Link TL-WDR4300 VLAN configuration with OpenWrt.

```plaintext
root@OpenWrt:~# ifconfig
br-lan Link encap:Ethernet HWaddr 10:FE:ED:AF:63:5E
  inet addr:192.168.5.1 Bcast:192.168.5.255 Mask:255.255.255.0
  inet6 addr: fe80::12fe:edff:feaf:635e/64 Scope:Global
  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
  RX packets:0 errors:0 dropped:0 overruns:0 carrier:0
  collisions:0 txqueuelen:0
  RX bytes:0 (0.0 B) TX bytes:7484 (7.3 KiB)
eth0 Link encap:Ethernet HWaddr 10:FE:ED:AF:63:5E
  inet6 addr: fe80::12fe:edff:feaf:635e/64 Scope:Link
  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
  RX packets:2417 errors:0 dropped:0 overruns:0 frame:0
  TX packets:2415 errors:0 dropped:0 overruns:0 carrier:0
  collisions:0 txqueuelen:1000
  RX bytes:320134 (312.6 KiB) TX bytes:552392 (539.4 KiB)
  Interrupt:4
eth0.1 Link encap:Ethernet HWaddr 10:FE:ED:AF:63:5E
  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
  RX packets:0 errors:0 dropped:0 overruns:0 frame:0
  TX packets:34 errors:0 dropped:0 overruns:0 carrier:0
  collisions:0 txqueuelen:0
  RX bytes:0 (0.0 B) TX bytes:3580 (3.4 KiB)
eth0.2 Link encap:Ethernet HWaddr 10:FE:ED:AF:63:5E
  inet6 addr: fe80::12fe:edff:feaf:635e/64 Scope:Link
  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
  RX packets:2417 errors:0 dropped:0 overruns:0 frame:0
  TX packets:2377 errors:0 dropped:0 overruns:0 carrier:0
  collisions:0 txqueuelen:0
  RX bytes:276628 (270.1 KiB) TX bytes:538720 (526.0 KiB)
```

Note: Several interfaces can be bundled in a bridge. By default, eth0.1 belongs to the bridge br-lan:

Listing 1.39: Default bridge configuration in OpenWrt.

```plaintext
root@OpenWrt:~# brctl show
bridge name  bridge id    STP enabled interfaces
br-lan        7ffe.10feedaf635e no           eth0.1
```

Linux Routers and Community Networks 18 Lab 1: Basic Network Configuration
1.8 Lab setup

1. Building a straight-through patch cord.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Pair</th>
<th>Cable</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>white/orange</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>orange</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>white/green</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>blue</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>white/blue</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>green</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>1</td>
<td>white/brown</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>2</td>
<td>brown</td>
</tr>
</tbody>
</table>

Table 1.1: RJ45 EIA/TIA-568B pinout.

2. Install first ssh client/server in the PCs (if necessary):

```
Listing 1.40: Install ssh.
~# apt-get update
~# apt-get install openssh-client
~# apt-get install openssh-server
```

3. Connect to the router through port 1. Flash the OpenWrt router (section 1.7.1, page 13).

4. Change the router password to 13f and access the router using the browser with the IPv6 link local through ssh (section 1.7.2, page 13). Some configuration will be made using the Web Interface, WI, and some using the Command Line Interface, CLI.

5. Disable the firewall using the WI:

```
System -> Startup -> firewall -> Enable & Stop
```

6. Connect the Internet Ethernet port of the router to the Internal network. The router should be configured by DHCP and have access to the Internet. Install the packages: tcpdump, ip, netperf and screen (section 1.7.4, page 15). Activate the netperf server (section 1.7.5, page 15).

7. Using the WI, configure Ethernet ports 1, 2, 3, 4 of the router in VLANs 1, 2, 3, 4 (section 1.7.9, page 17), and the Internet port in VLAN 5. Note that originally the Internet port of the router is configured in VLAN 2.

```
Network -> Switch -> Add -> ··· -> Save & Apply
```

8. List the physical interfaces using the CLI, you will see that only the pre-configured VLANs are listed:

```
Listing 1.41: Default interfaces.
root@OpenWrt:~# ip l l
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
   link/ether 10:fe:ed:af:63:5e brd ff:ff:ff:ff:ff:ff
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
   link/ether 10:fe:ed:af:63:5f brd ff:ff:ff:ff:ff:ff
```

Lab 1: Basic Network Configuration
4: wlan1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
   link/ether 10:fe:ed:af:63:60 brd ff:ff:ff:ff:ff:ff
5: br-lan: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
   link/ether 10:fe:ed:af:63:5e brd ff:ff:ff:ff:ff:ff
6: eth0.1@eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-lan state UP mode DEFAULT group default
   link/ether 10:fe:ed:af:63:5e brd ff:ff:ff:ff:ff:ff
7: eth0.2@eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
   link/ether 10:fe:ed:af:63:5e brd ff:ff:ff:ff:ff:ff

9. We need to create virtual interfaces associated with the physical interfaces for OpenWrt to set them up. Using the WI,
create the virtual interfaces LAN1, LAN2, LAN3, LAN3 associated with physical interfaces eth0.1, eth0.2, eth0.3, eth0.4. Configure them with unmanaged protocol. Associate the wan interface with eth0.5. In the end, delete the LAN interface, reboot and connect again. The new interfaces should show up configured only with IPv6 link local addresses:

Listing 1.42: Configured VLANs.

```
root@OpenWrt:~# reboot; exit
~# ssh -L 8080:[fe80::12fe:edff:feaf:635e%eth0]:80 fe80::12fe:edff:feaf:635e%eth0 -l root
root@OpenWrt:~# ip a l
```

10. Now, the objective if configuring the network of figure 1.4. First, configure the router names (R1, R2, · · ·) using the WI:

```
System -> Hostname -> Save & Apply
```

11. Configure the WiFi interfaces using the WI (sections 1.7.7, and 1.7.8, page 17), and use the CLI to configure the wired part. Try with traditional commands (ifconfig, router), and ip command.

Linux Routers and Community Networks

Lab 1: Basic Network Configuration
12. Each router must end up with the 37 networks shown in figure 1.4 and a default route towards the laptop that represents the Internet. All hosts must be reachable. All PCs must be also configured manually, setting the router where they are connected as default gateway.

13. Convection for the gateways: a router in the same column to reach networks in the same side, a router in the same row to reach networks in the opposite side. For instance, R1 chooses R5 to reach 10.5.1.0/24, and R10 to reach 10.6.1.0/24.
Figure 1.4: Lab setup, static routing.
2.1 Description
Quagga [15] is an open source routing software package that provides routing protocols support such as RIP, OSPF, IS-IS and BGP. Quagga is a branch of the original project called zebra. Quagga provides a Cisco IOS-like interface. In this lab we will review RIP and OSPF using Quagga.

2.2 IOS fundamentals

- Two modes (see figure 2.1):
  - exec: allows inspecting the router, e.g. show commands.
  - configuration: allows editing the router configuration.
- In configuration modes you edit the running-config pseudofile.
- To delete any configuration command of the running config, execute the same command from the corresponding configuration mode preceded by no.
- The prompt indicates the mode, e.g. >, #, #(config-if), etc.
- Case insensitive.
- ? for help.
- TAB for command completion.
- Allows abbreviated commands as long there is no ambiguity. E.g. sh for show, or conf term for configure terminal.
- Quagga specific: accept address/mask notation, e.g. 10.0.0.1/24.

2.3 Quagga set up
The Quagga Software Routing Suite comes as a set of daemons:
- zebra: general configuration.
- ripd: RIP daemon.
- ospfd: OSPF daemon

Use telnet to connect to the daemons:

```
root@OpenWrt:~# /etc/init.d/quagga start
quagga.init: Starting zebra ... done.
quagga.init: Starting ripd ... done.
quagga.init: Starting ospfd ... done.
root@OpenWrt:~# telnet localhost zebra
```

Figure 2.1: Configuration modes.

```
root@OpenWrt:~# /etc/init.d/quagga start
```
Password: zebra

OpenWrt>
OpenWrt> enable
OpenWrt# ?
clear    Reset functions
copy     Copy configuration
disable  Turn off privileged mode command
disable  Echo a message back to the vty
disable  End current mode and change to enable mode.
disable  Exit current mode and down to previous mode
disable  Description of the interactive help system
disable  Print command list
disable  Send a message to enabled logging destinations
disable  Negate a command or set its defaults
disable  Exit current mode and down to previous mode
disable  Show running system information
disable  Set terminal line parameters
disable  Write who is on vty
disable  Write running configuration to memory, network, or terminal

2.4 Basic commands

Listing 2.2: Interfaces

OpenWrt# show interface
Interface br-lan is up, line protocol detection is disabled
index 5 metric 1 mtu 1500
flags: <UP,BROADCAST,RUNNING,MULTICAST>
inet 192.168.5.1/24 broadcast 192.168.5.255
inet6 fd20:1d78:f920::1/60
inet6 fe80::12fe:edff:feaf:635e/64
Interface dummy0 is down
index 11 metric 1 mtu 1500
flags: <BROADCAST,NOARP>
HWaddr: 06:31:12:18:c8:5c
Interface eth0 is up, line protocol detection is disabled
index 2 metric 1 mtu 1500
flags: <UP,BROADCAST,RUNNING,MULTICAST>
inet6 fe80::12fe:edff:feaf:635e/64
Interface lo is up, line protocol detection is disabled
index 1 metric 1 mtu 65536
flags: <UP,LOOPBACK,RUNNING>
ineth 127.0.0.1/8
inet6 ::1/128
...

Listing 2.3: Assign IP address

OpenWrt# conf term
OpenWrt(config)# int dummy0
OpenWrt(config-if)# ip add 10.0.0.1/24

Listing 2.4: Remove IP address

OpenWrt(config-if)# no ip add 10.0.0.1/24

Listing 2.5: Routing table

OpenWrt# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
O - OSPF, I - IS-IS, B - BGP, H - HLSL, o - OLSR,
b - BATMAN, A - Babel,
> - selected route, * - FIB route
K   0.0.0.0/0 via 192.168.1.1, eth0.2
C   10.0.0.0/8 is directly connected, dummy0
C++ 127.0.0.0/8 is directly connected, lo
C++ 192.168.1.0/24 is directly connected, eth0.2
C++ 192.168.5.0/24 is directly connected, br-lan

Linux Routers and Community Networks

Lab 2: RIP and OSPF
2.5 RIP review

The Routing Information Protocol (RIP) is one of the oldest and more simple routing protocols [7]. In summary, it works as follows:

- The hop count metric is the number of jumps until the destination: 1 if the destination is a network directly connected, 2 if it has to go through a router, etc.
- The routers send periodically (each 30 seconds) a broadcast RIP message in each interface with the known destinations and metrics. They are sent with UDP, source and destination port: 520.
- If we stop receiving RIP messages from a neighbour (180 seconds), we assume that it is down.
- The metric’s value of infinity is 16.
- RIP version 2 introduces these changes: The netmask is added to the destinations sent in the messages. The messages...
are sent to the multicast address: 224.0.0.9 (all RIPv2 routers).

2.5.1 RIP Convergence Problems

- Depending on the route update message order, convergence problems may arise (*Count to Infinity*):

![Diagram of RIP Convergence Problems]

- Evolution of D=N4 entry when R3 fails:

<table>
<thead>
<tr>
<th>G</th>
<th>M</th>
<th>R3 fails</th>
<th>G</th>
<th>M</th>
<th>R1 upd</th>
<th>G</th>
<th>M</th>
<th>R2 upd</th>
<th>G</th>
<th>M</th>
<th>R1 upd</th>
<th>G</th>
<th>M</th>
<th>R2 upd</th>
<th>G</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1: R2 3</td>
<td>→</td>
<td>R2 3</td>
<td>→</td>
<td>R2 3</td>
<td>→</td>
<td>R2 5</td>
<td>→</td>
<td>R2 5</td>
<td>→</td>
<td>R2 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2: R3 2</td>
<td>→</td>
<td>R3 16</td>
<td>→</td>
<td>R1 4</td>
<td>→</td>
<td>R1 4</td>
<td>→</td>
<td>R1 6</td>
<td>→</td>
<td>R1 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Solutions to RIP Convergence Problems:

- *Split horizon*: When the router sends the update, removes the entries having a gateway in the interface where the update is sent.
- *Triggered updates*: Consists of sending the update before the 30 seconds timer expires, when a metric changes in the routing table.

2.6 RIP configuration

- **network command**:
  - Indicates the interfaces that have to send or process RIP messages.
  - Indicates which directly connected networks must be advertised. The network addresses have to be indicated without using subnetting (the mask of the class is assumed). For example, if the interface uses the IP address IP 10.5.4.2/24 we only need to announce the A class 10/8 with the format `network 10.0.0.0`.
  - Quagga uses RIPv2 by default, and masks must be provided to network command.
  - By default, CISCO routers do *route summarization*. The summarization is done at the class boundary. For example, if in the routing tables we have the subnets 10.0.1.0/24 and 10.0.2.0/24, when sending a RIP message to the net 192.168.0.0/24 it will be sent 10.0.0.0/8.

- In order for the router to be advertise static routes (including the default route), we have to execute the command `redistribute static`.

- The router uses two metrics: the administrative metric and the routing algorithm metric. If several routes to a same destination exist, the route with the lower administrative metric is chosen. For example, RIP has administrative metric 120 and OSPF 110.

**Listing 2.11: Administrative metric**

```
R1# sh ip ro
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route
R 17.16.4.0/24 [120/2] via 172.16.1.2, e0, 00:00:07
...```

- Verification command: `show ip rip status`.

**Listing 2.12: RIP configuration example**

```
root@OpenWrt:~# telnet localhost ripd
Hello, this is Quagga (version 0.99.22.3).
User Access Verification
Password: zebra
OpenWrt> enable
OpenWrt# configure terminal
```
2.7 RIP Lab setup

1. Install the packages kmod-dummy, quagga, quagga-ospfd, quagga-ripd and quagga-zebra (see section 1.7.4, page 15).

2. Rename the file /etc/quagga/ospfd.conf to avoid starting the ospfd daemon:

```
~# mv /etc/quagga/ospfd.conf /etc/quagga/ospfd.conf.dst
```

3. Start quagga daemons (check that ospfd does not start):

```
root@OpenWrt:/etc/quagga# /etc/init.d/quagga start
quagga.init: Starting zebra ... done.
quagga.init: Starting ripd ... done.
root@OpenWrt:/etc/quagga#
```

4. Have a look to the configuration files: /etc/quagga.
5. Configure the network of figure 2.2 using quagga.
6. Check the routing tables. Does RIP quagga daemon do route-summarization?
7. Use traceroute to figure out the path to different destinations.
8. Check the RIP messages sent by the router using tcpdump:

```
~# tcpdump -vni eth0.1 port 520
```

9. Identify the networks advertised in the RIP update messages captured with tcpdump. Observe the behavior of Split Horizon.
10. Disable Split Horizon in one interface, and observe the routes that are advertised by in the update messages.

```
ripd# conf term
ripd(config)# int eth0
ripd(config-if)# no ip rip split-horizon
```

11. Disconnect one network and observe the trigger updates and metric 16.
Figure 2.2: Lab setup, RIP routing.
2.8 OSPF review

Open Shortest Path First (OSPF) [8] is a routing protocol standardized inside the IETF with the aim of having a non-proprietary high performance interior gateway protocol. In summary, it works as follows:

- It is a link state protocol type: This means the router monitors and sent to other routers in the network information on networks and routers directly connected neighbors (link state refers to that information). Networks can be 4 types: Point-to-point, broadcast, non-broadcast multiaccess (NBMA) or point-to-multipoint.
- Each router maintains a database with information on the network topology. Each entry in the database is the information received from a router.
- Each router sends its local information to all other routers in the network using flooding. These messages are called Link State Advertisements (LSAs). The flooding routing is basically send datagrams for all interfaces except that the message has arrived. Thus, the message spread throughout the network, without using routing tables.
- Routers use the algorithm Shortest Path First (SPF) to calculate optimal routing entries, depending on the information stored in the database.
- The metric is dimensionless (does not represent the number of hops). The infinite metric is $0xFFFF$.
- A hello protocol, comprising sending signaling packets periodically. This protocol allows routers discover neighbors, and whether any of them becomes inaccessible.
- Unlike other routing protocols, OSPF does not carry data via a transport protocol, such as the User Datagram Protocol (UDP) or the Transmission Control Protocol (TCP). Instead, OSPF encapsulates messages into IP datagrams directly using IP protocol number 89.
- To reduce the number of floodings in the broadcast network with more than 1 router will choose a Designated Router (DR) and a Backup Designated Router (BDR). The DR is the only router in the broadcast domain that sends LSAs to the rest of the network.
- Each router is identified by a 32-bit number called Router ID (RID). Usually the IP address of the router greatest value is chosen. If an address to the dummy interface is assigned, it is chosen although not the highest value. It is advisable to assign an IP address to a dummy interface in order to keep the RID when changing router addresses.
- For the election of the DR and BDR can use a priority (default is 1, if it is equal to 0 means that the router can not be elected DR, BDR). In case of equal priority, the router higher RID is chosen.
- The protocol allows grouping a set of contiguous networks and routers in an area. All networks inside an area can be aggregated in a single prefix. The use of multiple areas increases scalability and reduces the traffic generated by the protocol.
- There must be a backbone area 0, to which all other areas are connected. Area 0 cannot be discontiguous. If any other area is not contiguous to area 0, virtual links must be used.
- Routers can be Internal Routers (IR), if they have all the interfaces in the same area or Area Border Router (ABR) if they have interfaces in more than one area.

2.9 OSPF configuration

- First you should configure an IP dummy interface in order to fix the RID (see listing 2.3).
- network command works similarly to RIP, but specifying the area.
- Area route aggregation is achieved using the range command in ABR routers.
- Default route is distributed using the command default-information originate.
- Verification commands: show ip ospf ?.

Listing 2.17: OSPF configuration example

```
~# telnet localhost ospfd
Entering character mode
Escape character is `^]'.
Hello, this is Quagga (version 0.99.22.3).
User Access Verification
Password: zebra
OpenWrt> enable
OpenWrt# configure terminal
OpenWrt(config)# hostname ospfd
ospfd(config)# router ospf
ospfd(config-router)# network 10.0.1.0/24 area 0
ospfd(config-router)# network ...
ospfd(config-router)# area 1 range 172.16.0.0/16
ospfd(config-router)# default-information originate
```
2.10 OSPF Lab setup

1. Reboot the routers to clean the configuration of RIP Lab.
2. Rename the desired daemons to start:

Listing 2.18: Quagga daemons

~# mv /etc/quagga/ripd.conf /etc/quagga/ripd.conf.dst
~# mv /etc/quagga/ospfd.conf.dst /etc/quagga/ospfd.conf

3. Start quagga daemons (check that ripd does not start):

Listing 2.19: Quagga daemons

root@OpenWrt:/etc/quagga# /etc/init.d/quagga start
quagga.init: Starting zebra ... done.
quagga.init: Starting ospfd ... done.
root@OpenWrt:/etc/quagga#

4. Have a look to the configuration files: /etc/quagga.
5. Configure the network of figure 2.3 using quagga:
   (a) Assign IP addresses to interfaces using zebra daemon.
   (b) Configure OSPF using ospfd daemon.
6. Check the routing tables.
7. Check the routing metrics.
8. Use traceroute to figure out the path to different destinations.
9. Activate area range aggregation and check routing table entries.
10. Capture OSPF messages sent by the router using tcpdump:

Listing 2.20: OSPF messages

~# tcpdump -vni eth0.1 proto 89

11. Disconnect one network and observe the LSA messages captured with tcpdump, and the changes in the routing tables.
Figure 2.3: Lab setup, OSPF routing.
3.1 Description
At the heart of Linux firewall configuration there is the `iptables` command. The objective of this lab is getting familiar with the basic usage of `iptables`. Look at the tutorial [3] available in the Internet for more details.

3.2 Iptables operation
The `iptables` command allows you to filter and/or modify some fields of the packets as they move through different stages (or chains) of the IP layer of the Linux machine. These built-in chains are PREROUTING, FORWARD, POSTROUTING, INPUT and OUTPUT. When the packet moves through one of these chains, the IP layer consults the tables where the `iptables` command makes possible to add rules. An action, or target, is applied to packets that match the expression specified in a rule. These tables are mangle, filter and nat. Figure 3.1 shows the built-in chains that a packet follows from the moment in which is received from the network or generated by a local process. The figure also shows the tables that are in every chain as well as the order in which are consulted.

- The **mangle** table allows you to add rules that change some of the fields of the packets, such as TTL or TOS.
- The **nat** table allows you to add rules that change the IP addresses of the packets. The types of rules are: SNAT or MASQUERADE to change the source address and DNAT to change the target address.
- The **filter** table allows you to add rules for filtering. The types of rules are: DROP to discard a packet and ACCEPT to accept it.

Figure 3.1: Processing of IP datagrams in a Linux router.
3.2.1 User specified chains
User can create user-chains. If a packet enters a built-in chain, e.g. FORWARD in the filter table, a rule can specify a jump to a user-chain within the same table (see figure 3.2). Upon jumping to a user-chain, rules of the user-chain are followed until the user-chain traversal is either ended by a target (which can be a jump to another user chain) or the end of the chain is reached. In the later case, the packet is sent back to the invoking chain.

![Figure 3.2: User chains.](image)

3.3 Creation of rules with the iptables command

**Listing 3.1: Generic iptables command format**

```
iptables [-t <table>] <command> <expression> -j <rule type>
```

If the table is not indicated, the rule is considered to be applied to filter table. The expression (or match) identifies which are the packets in which the rule has to be applied. The type of rule (or target) identifies what has to be done with the packet matching the rule: DROP, ACCEPT, SNAT, MASQUERADE, DNAT or jump to a user-chain.

3.4 Commands
- `-A <chain>`: adds a rule to the table of the chain `<chain>`. For example: `iptables -A INPUT ...`
- `-D <chain> <n>`: removes the rule `<n>` from a table of the chain `<chain>`. The rules are listed from 1 on. For example: `iptables -D INPUT 1`
- `-I <chain> <n> ...`: inserts a rule in the line that is specified. For example: `iptables -I INPUT 1 --dport 80 -j ACCEPT`
- `-L <chain>`: lists the rules of a table of `<chain>`. Accepts the option `-n` in order to get it not to translate the numerical addresses into names, `-v` so that it will be more verbose and `-line` to show the number of the rule. For example: `iptables --line -nvL INPUT`
- `-P <chain>`: specifies the type of rule that a table of `<chain>` has by default. If this command is not executed, the rule by default will be ACCEPT. For example, to get the filter table of the INPUT chain to discard everything, you will have to execute: `iptables -P INPUT DROP`.

3.5 Expressions
They can be generic (not specific to a protocol), TCP, UDP, ICMP and special.

3.5.1 Generic expressions
- `-p <protocol>`: identifies a protocol. filter of the INPUT chain. For example: `iptables -A INPUT -p tcp ...` will add a rule that will be applied for all the TCP packets to the filter table of the INPUT chain. The number of protocol or the name (file /etc/protocols) can be used to identify the protocols. For identifying all the protocols, the name ALL can be used (value by default, if `-p` is not used) or you can invert the expression with the operator !, for example, `-p ! tcp` means any protocol but TCP.
- `-s <@IP source>`: identifies the entire packet with `<@IP source>`. A mask can be added to identify a range of the addresses and the operator ! can be used for denying the expression. For example, `iptables -A INPUT -s ! 192.168.0.0/24 ...` will add a rule to the filter table of the INPUT chain that will be applied for all the packets with a source @IP that doesn’t belong to the range 192.168.0.0/24.
- `-d <@IP destination>`: the same as stated before for the target address.
- `i <input-interface>`: identifies an input interface of packets. It can be only applied in the chains `INPUT`, `FORWARD` and `PREROUTING`. For example, `iptables -A INPUT -i eth0 ...` will add a rule that will be applied for all the packets that will arrive from the `eth0` interface.

- `o <output-interface>`: the same as stated before for an output interface. It can be only applied in the chains `OUTPUT`, `FORWARD` and `POSTROUTING`.

### 3.5.2 TCP Expressions

They can be only applied for the TCP protocol, therefore, the expression must begin with `-p tcp`.

- `--sport <src-port>`: identifies the TCP packets provided with `<src-port>`. A number or any of the names of `/etc/services` can be used. The range: `initial:final` can be also specified. Otherwise, the initial value that will be indicated is 0 and, if not, the final value that will be indicated is 65535. For example: `iptables -A INPUT -p tcp --sport 1024: ...` will apply the rule for all the TCP packets with a port number ≥ 1024.

- `--dport <dst-port>`: the same as stated before for the destination port.

- `--tcp-flags <flag list> <flags list to 1>`: look at the packets of the `<flag list>` which flags have only a value equal to 1 if they are included in `<flags list to 1>`. Flags can be `SYN`, `FIN`, `ACK`, `RST`, `URG`, `PSH`, `ALL`, `NONE`. For example: `iptables -p tcp --tcp-flags SYN,RST,ACK SYN ...` will apply the rule for all the packets with the flag `SYN` enabled and the `RST` and `ACK` to 0.

### 3.5.3 UDP Expressions

They can be only applied for the UDP protocol; therefore, the expression must begin with `-p udp`.

- `--sport <src-port>`: the same option as stated before for the TCP protocol.

- `--dport <dst-port>`: the same option as stated before for the TCP protocol.

### 3.5.4 ICMP Expressions

They can be only applied for the ICMP protocol; therefore, the expression must begin with `-p icmp`.

- `--icmp-type <type>`: identifies the `icmp` packets of the type `<type>`. If you execute `iptables -p icmp --help` a list of the possible types will be given.

### 3.5.5 State Expression

This sort of expression is regarded as `no implicit`, and it is necessary to add `-m state` in order to be able to apply it. The states can be: `NEW`, `ESTABLISHED` and `RELATED`. Basically, when a received packet is identified as being from a new connection, this connection will register it as `NEW`. This affects to the TCP, UDP and ICMP packets (even if UDP and ICMP are not addressed to the connection. If a packet is received in the contrary way bound to a connection registered as `NEW`, then the connection will be registered as `ESTABLISHED`. Finally, if a new connection -that is interpreted as a result of the existing one- is detected (for example, because it is deduced that an ICMP error packet has been generated as a result of an already established connection), then the connection will register it as `RELATED`.

- `--state <state list>`: identifies the packets of a connection which is in one of the states of `<state list>`. For example: `iptables -m state --state RELATED,ESTABLISHED ...` will apply the rule to the packets of the connections located in one of these states.

### 3.6 Types of rules

As it has been introduced in section 2, the types of rules can be: `ACCEPT`, `DROP`, `SNAT`, `MASQUERADE`, `DNAT` or jump to a user-chain. The two firsts are obvious: they accept or discard the packet. In case of being accepted, the rest of the rules of the table will be left aside and you will continue with the rules of the remaining tables and chains that still have to be gone through.

#### 3.6.1 The DNAT rule

This type of rule is used with the option `-to-destination` to indicate the address where the translation has to be done. It can be put just in the nat table of the `PREROUTING` and `OUTPUT` chains. An address or a range of addresses can be specified (and the connections will be randomly distributed among the specified addresses, in other words, a load-balancing will be done). In case of an expression TCP or UDP a port or a range of ports can also be indicated (to distribute the connections among the range of ports). For example:
will change the target IP address of the packets that identifies the expression for an address of the range 192.168.10.10-192.168.10.20 and the range of the port 80-100. The same translation for address/port will be applied to all the packets of a same connection.

3.6.2 The SNAT rule
It has an analogous syntax to the one of DNAT, but with the option \texttt{--to-source}. It can be only put in the nat table of the \texttt{POSTROUTING} chain. For instance:

```
Listing 3.3: SNAT
~# iptables -t nat -A POSTROUTING -p tcp -o ppp0 -j SNAT --to-source 200.10.10.10-200.10.10.20:1024-32000
```

will change the source address of the packets that go out of the ppp0 interface with one of the addresses of the range 200.10.10-200.10.10.20 and the port of the range 1024-32000. If the range of ports is not indicated it will be necessary to change it (because PAT is done), then those ports lower than 512 will be mapped into another port lower than 512. And the same happens for the ranges comprised between 512-1023 and $\geq$1024 (to preserve the semantics of the port). The same translation for address/port will be employed for all the packets of a same connection.

3.6.3 Masquerade
The MASQUERADE target is used as the SNAT target, but it does not require any \texttt{--to-source} option. You simply provide an interface, and its address is applied to all outgoing packets. This is useful in dial-up connections, or DHCP connections, which have dynamic IP addresses. The \texttt{--to-ports} option is used to set the source port of outgoing packets.

```
Listing 3.4: SNAT
~# iptables -t nat -A POSTROUTING -p tcp -o ppp0 --to-ports 1024-20000 -j MASQUERADE
```

3.7 Quick edition of iptables configuration
The command:

```
Listing 3.5: Saving iptables configuration to file
~# iptables-save > cc
```

dumps all the configuration command that iptables is executing and readdress them to the file cc. You can edit this file and replace the new configuration of iptables with the command:

```
Listing 3.6: Restoring iptables configuration from file
~# iptables-restore cc
```
3.8 iptables Lab setup

![Lab setup, iptables.]

3.8.1 Configuration of the network

1. Install Apache server in the PCs:

```
Shebangs
~# apt-get update
~# apt-get install apache2
```

2. Each group $i, i = 1, \ldots, 10$ will configure the network of figure 3.3.
3. Disable the default OpenWrt firewall configuration (see section 1.8, page 19).
4. The PC $\text{ext}$ represents a host in the Internet. $\text{ext}$ must be configured with no default route such that it can only reach
the public address of the router $\text{Ri}$.
5. The PC $\text{serv}$ represents a server in the DMZ.
6. The PC $\text{int}$ represents a host in the internal network.
7. Configure the network DMZ and the Internal network. Verify that the router $\text{Ri}$, $\text{serv}$ and $\text{int}$ can reach each other,
but $\text{ext}$ cannot reach $\text{serv}$ and $\text{int}$.

3.8.2 Configuration of the firewall

8. Execute the following command in $\text{Ri}$:

```
Shebangs
Ri:~# iptables -t nat -A POSTROUTING -o eth0.5 -j SNAT --to-source 100.0.i.1
```

9. Check that now $\text{serv}$ and $\text{int}$ can reach $\text{ext}$ but not vice-versa, why? Check by executing tcpdump in $\text{eth0.1}$,
$\text{eth0.2}$ and $\text{eth0.5}$ of $\text{Ri}$ how the firewall change the IP addresses.
10. Execute the following command in $\text{Ri}$:

```
Shebangs
Ri:~# iptables -t nat -A PREROUTING -p tcp -i eth0.5 -d 100.0.i.1 --dport ssh -j DNAT --to-destination 10.i.1.2
```

11. Check that now $\text{ext}$ can connect to the ssh server of $\text{serv}$ (for example, by executing $\text{ssh root@100.0.i.1}$), but if
it pings to $\text{serv}$ it doesn’t reply, why?
12. Check that $\text{ext}$ has not access to any other service of the $\text{serv}$ (for example, web).
13. Which is the rule that should be executed in order to get to ping from $\text{ext}$ to $\text{serv}$?, and to get to connect to the web
server of $\text{serv}$? Try the rules.
14. Execute the commands in $\text{Ri}$ (do not delete the previous ones):

```
Shebangs
Ri:~# iptables -P INPUT DROP
Ri:~# iptables -P OUTPUT DROP
Ri:~# iptables -P FORWARD DROP
```

15. Check that none of the PCs is now reachable. Why?
16. Execute the commands in Ri:

```
Listing 3.11: Forwarding rules
Ri:=# iptables -A FORWARD -i eth0.2 -o eth0.5 -j ACCEPT
Ri:=# iptables -A FORWARD -i eth0.5 -o eth0.2 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

17. Check that now serv can ping to ext but not vice-versa. Why?
18. Check that now serv can access to any of the services of ext (for example, try ssh) but not vice-versa. Why?
19. Execute the command in Ri:

```
Listing 3.12: Forwarding rules
Ri:=# iptables -A FORWARD -p tcp -i eth0.5 -o eth0.2 -d 10.i.1.2 -j ACCEPT
```

20. Check that now that ext can connect to the ssh server of serv but int cannot. Why?
21. Check that ext only can access to the ssh server of serv (check for example that ext cannot access the web server of serv), why?
22. Configure the firewall in order to get ext to ping serv.
23. Execute the commands in Ri:

```
Listing 3.13: Forwarding rules
Ri:=# iptables -A FORWARD -i eth0.1 -o eth0.2 -j ACCEPT
Ri:=# iptables -A FORWARD -i eth0.2 -o eth0.1 -j ACCEPT
```

24. Now serv and int can see each other, but they don’t reach Ri. Why?
25. Execute the commands in Ri:

```
Listing 3.14: Forwarding rules
Ri:=# iptables -A INPUT -p icmp -i eth0.1 -d 10.i.2.1 -j ACCEPT
Ri:=# iptables -A OUTPUT -p icmp -o eth0.1 -s 10.i.2.1 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

26. Now int can ping to Ri, but cannot have access to any of the services of Ri. Why? Execute the commands that are necessary to get serv to have the same restrictions when accessing to Ri than those that int has.

### 3.8.3 Firewall design

Remove the configuration of the firewall and design a configuration that meets the following:

1. int is the only host which can connect to the ssh server of Ri. int has not to be able to connect to any other service of Ri.
2. int is able to connect to the ssh server of ext, but neither any other of the services of ext nor to any other host of the Internet. ext must not be able to start a connection to int, nor even to do a ping.
3. ext is able to connect to the web server of serv, and ping serv, but neither any other of the services of serv nor to any other host of the internal network.

### 3.8.4 Firewall Configuration in OpenWrt

1. Using the WI, assign LAN1, LAN2, LAN3 and LAN4 to the lan firewall-zone:

   ![Network -> Interfaces -> Edit -> Assign firewall-zone](image)

2. Start the OpenWrt firewall (System -> Startup -> firewall -> Enable -> Start) and use `iptables-save`
to observe the configuration.
3. Use the OpenWrt firewall (Network -> Firewall) to setup some options (e.g. Masquerading), and check the iptables configuration changes with `iptables-save`.
Lab. 4: Community Networks

4.1 Introduction

Ad-hoc networks do not rely on a pre-existing infrastructure. Instead, the nodes are possibly mobile and self-organize and participate in routing. This concept was already introduced in the 1970s and produced extensive research in the 90s, when many new routing algorithms were proposed (see e.g. [2]). Later, the concept of wireless mesh networks (WMNs) was introduced. WMNs are described in [1] as:

Wireless mesh networks (WMNs) consist of mesh routers and mesh clients, where mesh routers have minimal mobility and form the backbone of WMNs. They provide network access for both mesh and conventional clients. The integration of WMNs with other networks such as the Internet, cellular, IEEE 802.11, IEEE 802.15, IEEE 802.16, sensor networks, etc., can be accomplished through the gateway and bridging functions in the mesh routers. Mesh clients can be either stationary or mobile, and can form a client mesh network among themselves and with mesh routers. WMNs are anticipated to resolve the limitations and to significantly improve the performance of ad hoc networks, wireless local area networks (WLANs), wireless personal area networks (WPANs), and wireless metropolitan area networks (WMANs). They are undergoing rapid progress and inspiring numerous deployments. WMNs will deliver wireless services for a large variety of applications in personal, local, campus, and metropolitan areas. Despite recent advances in wireless mesh networking, many research challenges remain in all protocol layers.

In summary, mesh networks have some sort of infrastructure with wireless and wired links and use some self-forming, self-healing routing protocol. Wireless Community Networks (WCN) can be considered an example of mesh networks. WCNs are networks built by a community of users that install wireless antennas on their houses, typically on the roof. WCN are non-profit networks deployed and maintained by their own users. Nowadays there are inexpensive WiFi devices that have fostered the deployment of such networks (see list of WCN in Wikipedia [20]). A relevant example is Guifi.net [13, 18], the largest currently existing community network, having more than 28,000 operative nodes.

Guifi.net was started in 2004. At that time mesh routing protocols implementations were scarce and unstable. This has motivated that most of Guifi.net’s infrastructure have been deployed using OSPF and BGP routing protocols. However, there are initiatives inside Guifi.net that are experimenting with routing mesh protocols, which have been designed for these type of networks. For this purpose a branch of OpenWrt called Quick Mesh Project (QMP) [16, 5] is being developed. QMP is native IPv6 and uses BMX6 [11, 12] as routing protocol. QMP is open source and can be free downloaded from [16].

4.2 WiFi Ad-Hoc Mode

Nodes in qMp are configured in ad-hoc mode.

![Infrastructure BSS (infrastructure mode)](image1)

BSSID: All stations share a common BSSID. In qMp this is 02:CA:FF:EE:BA:BE.

![Independent BSS, IBSS (ad-hoc mode)](image2)
Beacons: One of the stations assumes the responsibility for sending the beacon. After receiving a beacon frame, each station waits for the beacon interval and then sends a beacon if no other station does so after a random time delay. The random delay rotates the responsibility for sending beacons.

4.3 Create a node in Guifi.net
- Register in https://guifi.net -> Create content -> guifi.net node
- Add a device to the node (you need to provide the MAC address).
- Save the node url. In the example below: https://guifi.net/en/node/79331
- We will use the addresses obtained from Guifi.net. In the example below: 10.229.65.1/27

More info: http://en.wiki.guifi.net/wiki/Special:AllPages

- qMp provides 2 interfaces accessible at the bottom menu: qMp and the standard OpenWrt menu: Administration:

4.4 qMp
Basic documentation: http://qmp.cat/Documentation

4.4.1 qMp fundamentals
- It is a branch of OpenWrt [14].
- Native IPv6.
- Uses a mesh routing protocol called BMX6 [11, 12].
- IPv4 connections are enabled via tunnels over the ULA-based IPv6 network.

BMX6 is an open-source mesh routing protocol being developed by Axel Neumann. It originated from a branch of the mesh routing protocol called BATMAN [4]. Design principles of BMX6 include:
- Designed to optimize the communication between neighbor nodes for minimal overhead.
- It uses UDP messages (port 6240) to discover other nodes and disseminate node and routing information.
- Proactive protocol: nodes maintain routes to all destination nodes running BMX6 proactively.
- Distance-vector: nodes don’t have a global topology information, but choose next hop to each destination based on aggregated information, similarly to RIP and BGP.
4.4.2 BMX6 daemon
/usr/sbin/bmx6 is the BMX6 daemon, which is automatically started by qMp. Furthermore, qMp starts by default the crontab job /etc/qmp/bmx6health.sh which is executed every minute in order to check that the BMX6 daemon is properly running. The BMX6 can be requested from the CLI. In the following there are some common invocations:

```
Listing 4.1: Queering the BMX6 daemon.
```

```
~# bmx6 -c status
status:
version compat name primaryIp tun6Address tun4Address uptime cpu nodes
```

```
~# bmx6 -c originators
originators:
name blocked primaryIp routes viaIp viaDev metric lastDesc lastRef
BCN-GS-Salou2bis-3a56 0 fd66:66:66:9:618:d6ff:fe65:3a56 1 fe80::227:22ff:feb8:b828 eth1.2 37748K 5588
```

```
~# bmx6 -c tunnels
tunnels:
name net min max hyst rating minBw tunName tunRoute remoteName advNet
community6 ::/0 32 48 20 100 960 --- --- --- --- --- --- ---
inet6 ::/0 0 0 20 100 960 --- --- --- --- --- --- ---
cloud 10.0.0.0/8 24 128 20 100 960 bmxC4tmain 10.139.95.254/32 GSgVr-bc89-RC8da9 10.139.95.254/32
cloud 10.0.0.0/8 24 128 20 100 960 bmxC4tmain 10.140.48.3/32 GSsalou25-NS7b40 10.140.48.3/32
cloud 10.0.0.0/8 24 128 20 100 960 bmxC4tmain 10.140.48.15/32 GSfeliucasanova24-NSb19a 10.140.48.15/32
```

```
~# bmx6 -c interfaces
interfaces:
devName state type rateMin rateMax llocalIp globalIp primary
eth1.1 UP ethernet 1000M 1000M fe80::4e5e:cf:fee9:fc8a/64 fd66:66:66:7:4e5e:cf:fee9:fc8a/64 1
eth1.2 UP ethernet 1000M 1000M fe80::4e5e:cf:fee9:fc8a/64 fd66:66:66:6:4e5e:cf:fee9:fc8a/64 0
```

```
~# bmx6 -c links
links:
name llocalIp viaDev rxRate txRate bestTxLink routes wantsOgms
UPCc6-65ab fe80::227:22ff:feb9:b5ab eth1.1 100 93 1 13 1
G5gV-nb-207c fe80::26a4:3cff:fe67:207c eth1.2 100 100 1 4 1
UPC-CN-D6-105-5259 fe80::fad1:11ff:fec4:5259 eth1.1 100 93 1 10 1
UPCaliX6-e81c fe80::20:bff:feb1:e81c eth1.1 100 93 1 1 1
```

```
~# bmx6 -c parameters
plugin bmx6_config.so
plugin bmx6_json.so
plugin bmx6_sms.so
ipVersion 6
dev eth1.1
dev eth1.2
tunDev tmain
/tun4Address 10.1.24.97/27
/tun6Address 2012:0:0:fc89::1/64
```

4.4.3 Flashing qMp

1. Download the qMp image [http://fw.qmp.cat](http://fw.qmp.cat)
2. Flash qMp from OpenWrt System -> Backup/Flash:
   1. Choose File -> Flash Image
   2. Proceed
   3. Default IP address: 172.30.22.1
   4. Default root password: 13f

Linux Routers and Community Networks 41 Lab 4: Community Networks
4.4.4 qMp Configuration

Basic qMp configuration is easy using the WI and the Node configuration menu, as show in the following. Most options are self-explanatory, and additional comments are provided in the WI. See figures below. In the sub-menu qMp gateways it can be configured the nodes that advertise a default route: check Enable this rule in the section INET4_OFFER type offer network 0.0.0.0/0, and similarly for IPv6.

- Configure qMp easy setup

![Gsponsor:ns-a1a5](image)

**qMp easy setup**

This page provides a fast and simple way to configure the basic settings of a qMp node. Use the fields below to specify the network mode, the IP addressing and the interface modes.

**Network mode**

qMp nodes can operate in two different modes, depending on the kind of network to deploy. According to your needs, you can choose between:

- roaming mode, for quick, temporal deployments. User devices connected to the network can roam between Access Points without loosing connectivity. However, they cannot see other devices connected to the Mesh.
- community mode for static, long-term deployments (such as community networks). User devices connected to the network get an IP address from a specific range and are accessible from the rest of the Mesh. However, roaming between stations is not possible.

![Community](image)

- "Roaming" mode for quick, temporal network setups.
- "Community" mode for community networks and long-term deployments.

**Node name**

Choose a name for this node. It will be used to identify the device in the network. Use only alphanumeric characters, spaces are not allowed.

![Gsponsor:ns-a1a5](image)

- The name of this node. Four hex numbers will be appended, according to the device’s MAC address.

**IP address and network mask**

Specify the IP address and the network mask for this node, according to the planification of your community or your network deployment. End-user devices will get an IP address within the valid range determined by these two values.

![10.228.205.129](image)

- Main IPv4 address for this node.

![255.255.255.248](image)

- Network mask to be used with the IPv4 address above.

**Interface modes**

Select the working mode of the network interfaces.

- **LAN mode** is used to provide connectivity to end-users (a DHCP server will be enabled to assign IP addresses to the devices connecting) Mesh mode is used on interfaces connected to an internet uplink or any other gateway connection.
- **WAN mode** is used on wireless interfaces to act as an access point and provide connectivity to end-users.

![Wired interface eth1](image)

- **Mesh mode** is used on interfaces connected to a mesh node.

![Use mesh in all wired devices](image)

- If this option is enabled, all the node’s wired network devices will be used for meshing (recommended).

![Wireless interface wlan0](image)

- **Ad hoc (mesh)** mode.

- **Primary network device**

![Gsponsor:ns-a1a5](image)

- The name of the node’s primary network device. The last four digits of this device’s MAC address will be appended to the node name.

- Configure qMp node basic settings

![Gsponsor:ns-a1a5](image)

**qMp node basic settings**

**Node identity**

Use this page to define basic qMp settings, like the node’s name.

**Node name**

- The name for this node (use alphanumeric characters, without spaces).

**Primary network device**

- The name of the node’s primary network device. The last four digits of this device’s MAC address will be appended to the node name.
Configure qMp network settings

qMp network settings

Network mode
Select the working mode of the wired network interfaces:
- LAN mode is used to provide end-users connectivity and a DHCP will be enabled to assign IP addresses to the devices connecting.
- WAN mode is used on interfaces connected to an Internet up-link or any other gateway connection.

LAN and WAN modes are mutually exclusive. Do not set an interface in both LAN and WAN modes.

LAN mode
- wan0
- eth1
- eth0

Interfaces used to provide end-user connectivity (DHCP server)

WAN mode
- wan0
- eth1
- eth0

Interfaces connected to an Internet up-link or any other gateway (DHCP client)

Mesh interfaces
Select the devices that will be used in the mesh network. It is recommended to select them all.

MESH devices
- wan0
- eth1
- eth0

Devices used for meshing (it is recommended to check them all)

Special settings
Use this section to disable VLAN tagging in certain interfaces or to exclude them from qMP.

VLAN-untagged devices

Devices that will not be used with VLAN tagging (it is recommended to leave it blank)

Excluded devices

Devices that will not be used by qMP

Configure Advanced network settings
4.4.5 qMp Verification

Verification can be done through the Mesh menu. The sub-menus Status, Nodes, Links and Tunnels are basically dumps of the bmx6 commands status, originators, links and tunnels, respectively (see section 4.4.2). The Graph sub-menu builds a map of the qMp nodes and links.

4.5 Lab setup

The objective is configure the network of figure 4.3 with the guidelines given in the following:

- qMp routers (indoor and outdoor) are configured with the IP addresses gathered from Guifi.net.
- Activate the mesh protocol in all links where there are other qMp devices.
- WiFi interfaces in 2.4 GHz are configured as APs.
- LAN and WiFi in 2.4 GHz interfaces are bridged.
- Hosts are configured by DHCP from the indoor qMp routers.
- Configure the qMp node having Internet access to advertise a default route.

Verification:

- Check that all nodes are reachable.
- Verify BMX6 using the CLI.
- Use mtr -n @IPv6 to figure out the path to node @IPv6 (for different nodes). IPv6 addresses can be found in Mesh -> Nodes. Note that the path cannot be discovered using IPv4, since it is an IPv4 over IPv6 tunnel.
- Check that nodes can access the Internet.
Figure 4.3: Lab qMp.
5.1 Introduction

*Network management* refers to the network operation, administration, maintenance and provisioning. These issues are essential for the network to run smoothly. *Maintenance* is specially critical to avoid network degradation. This is particularly true in community networks, which are deployed with low cost devices and low redundancy. Routers, and specially if they are outdoor, can hang, crash, have hardware failures and other sort of malfunctioning problems. Most of the times simply rebooting some daemon, or the device is enough to have it working again. Therefore, it is essential to have monitoring tools that detect network failures, such that the network operators (the users in community networks) can take the appropriate actions. There is a large number of monitoring tool solutions having many different features (see Wikipedia page on this topic [19]). In this lab we will investigate two monitoring tools: *nagios* [10] and *munin* [9]. Both are free, open source and available in most Linux distributions (e.g. ubuntu). Nagios focuses on alerting, munin on graphing.

5.2 Nagios Installation

1. Install `nagios3` along with all the required dependencies including `apache2`, `postfix` and others. Execute as `root` user.

   ```
   ~# apt-get update
   ~# apt-get install -y nagios3
   `

   Listing 5.1: Installation in Debian/ubuntu

2. If not requested by the installation command above, create a `nagiosadmin` user account so that you can login and administer the site. Give the password 13f

   ```
   ~# htpasswd -c /etc/nagios3/htpasswd.users nagiosadmin
   ```

   Listing 5.2: nagiosadmin user

   If everything goes fine, after installation a default monitoring setup is already running, accessible by the web browser at http://localhost/nagios3:
5.3 Nagios Configuration

Nagios basic building blocks are:

- **Hosts**: normally a computer.
- **Services**: anything to check in a host.
- **Contacts**: someone to notify events.

Upon installing nagios localhost and some services are already configured, which are accessible choosing Hosts and Services on the left menu:

![Nagios Interface](image)

Configuration is done by manually editing the configuration files: `/etc/nagios3/conf.d/*.cfg`. All these files are read upon nagios daemon initialization.

**Listing 5.3: Configuration files.**

```
# ls -al /etc/nagios3/conf.d/*.cfg
-rw-r--r-- 1 root root 1695 jul 29 2011 contacts_nagios2.cfg
-rw-r--r-- 1 root root 418 jul 29 2011 extinfo_nagios2.cfg
-rw-r--r-- 1 root root 1152 jul 29 2011 generic-host_nagios2.cfg
-rw-r--r-- 1 root root 1803 jul 29 2011 generic-service_nagios2.cfg
-rw-r--r-- 1 root root 678 jul 29 2011 hostgroups_nagios2.cfg
-rw-r--r-- 1 root root 2167 jul 29 2011 localhost_nagios2.cfg
-rw-r--r-- 1 root root 657 jul 29 2011 services_nagios2.cfg
-rw-r--r-- 1 root root 1609 jul 29 2011 timeperiods_nagios2.cfg
```

5.4 Nagios Plugins

Plugins are programs used to gather information about hosts and services. For instance, `check_ping` plugin to check whether a host responds to pings is typically used to check reachability. Nagios basic installation comes with a set of Core plugins, but many others are contributed by nagios users¹.

- **Plugins directory**: `/usr/lib/nagios/plugins`
- **Plugins configuration files**: `/etc/nagios3/plugins/config`

**Listing 5.4: check_ping plugin help**

```
# /usr/lib/nagios/plugins/check_ping --help
check_ping v2.1.1 (monitoring-plugins 2.1.1)
Copyright (c) 1999 Ethan Galstad <nagios@nagios.org>
Copyright (c) 2000-2007 Monitoring Plugins Development Team
<devel@monitoring-plugins.org>
Use ping to check connection statistics for a remote host.
Usage:
check_ping -H <host_address> -w <wrta>,<wpl>% -c <crta>,<cpl>%
[-p packets] [-t timeout] [-4|-6]
Options:
 -h, --help
   Print detailed help screen
 -V, --version
   Print version information
 --extra-opts=[section][@file]
   Read options from an ini file. See
   https://www.monitoring-plugins.org/doc/extra-opts.html
   for usage and examples.
 -4, --use-ipv4
   Use IPv4 connection
 -6, --use-ipv6
```

¹See [https://exchange.nagios.org/directory/Plugins/Linux Routers and Community Networks](https://exchange.nagios.org/directory/Plugins/Linux Routers and Community Networks)
Use IPv6 connection
-H, --hostname=HOST host to ping
-w, --warning=THRESHOLD warning threshold pair
-c, --critical=THRESHOLD critical threshold pair
-p, --packets=INTEGER number of ICMP ECHO packets to send (Default: 5)
-L, --link show HTML in the plugin output (obsolete by urlize)
-t, --timeout=INTEGER Seconds before connection times out (default: 10)

THRESHOLD is \(<rta>,<pl>\)% where \(<rta>\) is the round trip average travel
time (ms) which triggers a WARNING or CRITICAL state, and \(<pl>\) is the
percentage of packet loss to trigger an alarm state.

This plugin uses the ping command to probe the specified host for packet loss
(percentage) and round trip average (milliseconds). It can produce HTML output
linking to a traceroute CGI contributed by Ian Cass. The CGI can be found in
the contrib area of the downloads section at http://www.nagios.org/
Send email to help@monitoring-plugins.org if you have questions regarding
use of this software. To submit patches or suggest improvements, send email
to devel@monitoring-plugins.org

The check-host-alive, used by default to check reachability of hosts, is defined in terms of check_ping plugin:

Listing 5.5: check-host-alive command

```sh
# cat /etc/nagios3/plugins/config/ping.cfg | sed -n '/check-host-alive/,/}/p'
# 'check-host-alive' command definition
define command{
    command_name check-host-alive
    command_line /usr/lib/nagios/plugins/check_ping -H '\$HOSTADDRESS$' -w 5000,100% -c 5000,100% -p 1
}
```

5.5 Adding nagios hostgroups and hosts

Add a hostgroup following the examples of hostgroups_nagios2.cfg

Listing 5.6: hostgroup

```sh
# cat /etc/nagios3/conf.d/hostgroups_nagios2.cfg
define hostgroup{
    hostgroup_name Indoor-routers
    alias qMp routers
}
```

Add a host following the examples of generic-host_nagios2.cfg and localhost_nagios2.cfg. First, create a new
configuration file with the example configuration:

Listing 5.7: New configuration file

```sh
# cat /etc/nagios3/conf.d/generic-host_nagios2.cfg localhost_nagios2.cfg > indoor-qmp-nodes.cfg
```

Edit indoor-qmp-nodes.cfg

Listing 5.8: New configuration file with qmp indoor routers

```sh
# cat /etc/nagios3/conf.d/indoor-qmp-nodes.cfg
```

Linux Routers and Community Networks

Lab 5: Network Management
## 5.6 Adding nagios contacts

1. Add a new contact:

   ```
   ~# cat /etc/nagios3/conf.d/contacts_nagios2.cfg
   define contact{
     contact_name lloren
     alias qMp administrator
     service_notification_period 24x7
     host_notification_period 24x7
     service_notification_options w,u,c,r
     host_notification_options d,r
     service_notification_commands notify-service-by-email
     host_notification_commands notify-host-by-email
     email lloren@ac.upc.edu
   }
   ```

2. Add the contact to the host definition template:

   ```
   ~# cat /etc/nagios3/conf.d/indoor-qmp-nodes.cfg
   # Generic host definition template - This is NOT a real host, just a template!
   define host{
     use qmp-router ; Name of host template to use
     host_name tp4300-1
     address 10.228.205.129
   }
   ```

Restart nagios:

Listing 5.9: Restart Nagios

```
~# /etc/init.d/nagios3 restart
Restarting nagios3 monitoring daemon: nagios3
~#
```

and the new host should show up:
3. restart nagios.
   ```bash
   # /etc/init.d/nagios3 restart
   ```

NOTE: The SMTP daemon must be able to send emails. Configuration in Debian:
```bash
# dpkg-reconfigure exim4-config
```

### 5.7 Monitoring nagios services on a remote host

We will use *Nagios Remote Plugin Executor*, NRPE.

![NRPE Operation](image)

#### Figure 5.1: NRPE Operation.

1. Install the plugin in the server:

   ```bash
   # apt-get update
   # apt-get install nagios-nrpe-plugin
   ```

Check that the following nagios command is added upon installing the NRPE plugin:

   ```bash
   # cat /etc/nagios-plugins/config/check_nrpe.cfg
   # this command runs a program $ARG1$ with arguments $ARG2$
   define command {
     command_name check_nrpe
     command_line /usr/lib/nagios/plugins/check_nrpe -H $HOSTADDRESS$ -c $ARG1$ -a $ARG2$
   }
   # this command runs a program $ARG1$ with no arguments
   define command {
     command_name check_nrpe_larg
     command_line /usr/lib/nagios/plugins/check_nrpe -H $HOSTADDRESS$ -c $ARG1$
   }
   ```

2. Install the NRPE daemon and nagios plugins in the nagios host:

   ```bash
   root@qMp:-# opkg update
   root@qMp:-# opkg install nagios-plugins
   root@qMp:-# opkg install nrpe
   ```
Listing 5.15: Preconfigured NRPE services in the nagios host.

root@qMp:~# egrep '^command' /etc/nrpe.cfg
command_timeout=60
command[check_users]=/usr/libexec/nagios/check_users -w 3 -c 5
command[check_load]=/usr/libexec/nagios/check_load -w 7,4,2 -c 10,5,3
command[check_disk]=/usr/libexec/nagios/check_disk -w 50% -c 25% -p /tmp
command[check_zombie_procs]=/usr/libexec/nagios/check_procs -w 1 -c 3 -s Z
command[check_total_procs]=/usr/libexec/nagios/check_procs -w 25 -c 30

3. Accept connections to port 5666 in the host and activate the NRPE daemon:

   Administration -> Network -> Firewall ->
   Traffic Rules

   Administration -> System -> Startup ->
   Enable & Start

4. Configure the daemon in the host to accept requests from the server. Comment out the following lines in the configuration file:

Listing 5.16: Changes in the NRPE configuration file /etc/nrpe.cfg.

root@qMp:~# vi /etc/nrpe.cfg
# server_address=192.168.1.1
# allowed_hosts=192.168.1.2

root@qMp:~# /etc/init.d/nrpe restart

5. Test that the NRPE is reachable from the nagios server:

Listing 5.17: NRPE test

~# /usr/lib/nagios/plugins/check_nrpe -H 192.168.1.232 -c check_load
OK - load average: 0.28, 0.23, 0.17|load1=0.280;7.000;10.000;0; load5=0.230;4.000;5.000;0; load15=0.170;2.000;3.000;0;

6. Create the file qmp-services.cfg adding a new service for Indoor-routers host group:

Listing 5.18: Installation of nagios-nrpe-plugin in the nagios server.

~# cat /etc/nagios3/conf.d/qmp-services.cfg

# Define a service to "ping" qmp machines
define service{
  name qmp-load-service ; The name of this service template
  use generic-service ; Inherit default values from the generic-service definition
  check_command check_nrpe_1arg!check_load
  service_description qMp-LOAD
  max_check_attempts 5 ; Re-check the service up to 5 times in order to determine its final (hard) state
  normal_check_interval 5 ; Check the service every 5 minutes under normal conditions
  retrycheck_interval 2 ; Re-check the service every 2 minutes until a hard state can be determined
  notification_interval 120 ; Re-notify about service problems every 2 hours
  register 0 ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL SERVICE, JUST A TEMPLATE!
}

define service{
  use qmp-load-service ; Name of service template to use
  hostgroup_name Indoor-routers
}

7. Check the configuration and restart nagios in the server:

Listing 5.19: Check nagios configuration and restart.

~# nagios3 -v /etc/nagios3/nagios.cfg
... Total Warnings: 0
Total Errors: 0
8. Find the new service in nagios Services menu:

```
-# /etc/init.d/nagios3 restart
```

5.8 Nagios Lab setup

1. Install nagios in the PCs and monitor the routers configured as in lab 4.
2. Create a group called **Outdoor-routers**.
3. Add all TP-links to the **Indoor-routers** group.
4. Add all Nanostations to the **Indoor-routers** group.
5. Install NRPE in the nagios server and hosts. Add the preconfigured services `check_load` and `check_total_procs` for all qMp nodes.
6. Disconnect one qMp node and check the changes in nagios.

5.9 Graphic Monitoring with Munin

Munin [9] consists of a **munin-master** that connects every 5 min to **munin-nodes** to gather information. Information is provided by means of **munin plugins** (see figure 5.2). The data is stored using the **round-robin database tool**, RRDtool, in a set of files found in `/var/lib/munin`. The munin-master is accessible over a web interface. Munin is widely used because is easy to install and configure. Additionally, qMp nodes are setup with a basic munin-node configuration.

![Munin System Diagram](image)

**Figure 5.2: Munin system**

5.10 Munin Installation

1. Install the software:

```
Listing 5.20: munin installation.
-# apt-get update
-# apt-get install munin munin-node
-# ln -s /etc/munin/apache.conf /etc/apache2/sites-enabled/munin
-# /etc/init.d/apache2 restart
```

2. By default munin website is only reachable from localhost. Chang the Configuration to be reachable from the desired addresses (all private addresses in the following example):
Listing 5.21: munin installation.

```bash
~# cat /etc/munin/apache24.conf
<Directory /var/cache/munin/www>
  Require local
  Allow from localhost 127.0.0.0/8 172.16.0.0/12 10.0.0.0/8 192.168.0.0/16 ::1
  # Options None
  Options FollowSymLinks SymLinksIfOwnerMatch
  Require all granted
</Directory>
```

```bash
~# chown -R www-data:munin /var/cache/munin/www
~# chmod -R 775 /var/cache/munin/www
~# /etc/init.d/apache2 restart
```

3. Check that the munin website is reachable:

![Munin Graphs](image)

5.11 Adding munin-nodes

1. Check that a qMp munin-node is reachable from the master:

```bash
~# telnet 10.228.205.177 4949
Trying 10.228.205.177...
Connected to 10.228.205.177.
Escape character is '^]'.
# munin node at GSgV-nsl-b828
version
munins node on qMp-635e version: 1.0.4 (munin-lite)
list
cpu load memory processes uptime interrupts irqstats dhcp_leasses mesh_links mesh_nodes net_eth0 net_wlan0 wifi_signal
config load
graph_title Load average
graph_args --base 1000 -l 0
graph_vlabel load
graph_category system
graph_load load
graph_warning 10
graph_critical 120
graph_info The load average of the machine describes how many processes are in the run-queue (scheduled to run "immediately").
graph_fetch load
graph_value 0.04
graph_quit
```

2. Add the node in /etc/munin/munin.conf

```bash
~# cat /etc/munin/munin.conf
...[
[GSgV-nsl-b828]
  address 10.228.205.177
```

3. Around 5 min after the node will show up:
5.12 Munin Lab setup

1. Install munin in the PCs. Check that qMp nodes are already configured as munin-nodes. Add the qMp nodes to the munin server.
## Appendix A: WLAN channels

![Figure A.1: 2.4 GHz band channels](source: Wikipedia)

### 2.4 GHz (802.11b/g/n)

<table>
<thead>
<tr>
<th>Channel</th>
<th>Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2412</td>
</tr>
<tr>
<td>2</td>
<td>2417</td>
</tr>
<tr>
<td>3</td>
<td>2422</td>
</tr>
<tr>
<td>4</td>
<td>2427</td>
</tr>
<tr>
<td>5</td>
<td>2432</td>
</tr>
<tr>
<td>6</td>
<td>2437</td>
</tr>
<tr>
<td>7</td>
<td>2442</td>
</tr>
<tr>
<td>8</td>
<td>2447</td>
</tr>
<tr>
<td>9</td>
<td>2452</td>
</tr>
<tr>
<td>10</td>
<td>2457</td>
</tr>
<tr>
<td>11</td>
<td>2462</td>
</tr>
<tr>
<td>12</td>
<td>2467</td>
</tr>
<tr>
<td>13</td>
<td>2472</td>
</tr>
<tr>
<td>14</td>
<td>2484</td>
</tr>
</tbody>
</table>

### 5 GHz (802.11a/h/j/n/ac)

<table>
<thead>
<tr>
<th>Channel</th>
<th>Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>5180</td>
</tr>
<tr>
<td>38</td>
<td>5190</td>
</tr>
<tr>
<td>40</td>
<td>5200</td>
</tr>
<tr>
<td>42</td>
<td>5210</td>
</tr>
<tr>
<td>44</td>
<td>5220</td>
</tr>
<tr>
<td>46</td>
<td>5230</td>
</tr>
<tr>
<td>48</td>
<td>5240</td>
</tr>
<tr>
<td>52</td>
<td>5260</td>
</tr>
<tr>
<td>56</td>
<td>5280</td>
</tr>
<tr>
<td>60</td>
<td>5300</td>
</tr>
<tr>
<td>64</td>
<td>5320</td>
</tr>
<tr>
<td>100</td>
<td>5500</td>
</tr>
<tr>
<td>104</td>
<td>5520</td>
</tr>
<tr>
<td>108</td>
<td>5540</td>
</tr>
<tr>
<td>112</td>
<td>5560</td>
</tr>
<tr>
<td>116</td>
<td>5580</td>
</tr>
<tr>
<td>120</td>
<td>5600</td>
</tr>
<tr>
<td>124</td>
<td>5620</td>
</tr>
<tr>
<td>128</td>
<td>5640</td>
</tr>
<tr>
<td>132</td>
<td>5660</td>
</tr>
<tr>
<td>136</td>
<td>5680</td>
</tr>
<tr>
<td>140</td>
<td>5700</td>
</tr>
<tr>
<td>149</td>
<td>5745</td>
</tr>
<tr>
<td>153</td>
<td>5765</td>
</tr>
<tr>
<td>157</td>
<td>5785</td>
</tr>
<tr>
<td>161</td>
<td>5805</td>
</tr>
<tr>
<td>165</td>
<td>5825</td>
</tr>
</tbody>
</table>
# List of acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL</td>
<td>Access Control List</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>AP</td>
<td>Access Point</td>
</tr>
<tr>
<td>ARP</td>
<td>Address Resolution Protocol</td>
</tr>
<tr>
<td>ASN</td>
<td>Autonomous System Number</td>
</tr>
<tr>
<td>AS</td>
<td>Autonomous System</td>
</tr>
<tr>
<td>BGP</td>
<td>Border Gateway Protocol</td>
</tr>
<tr>
<td>BSS</td>
<td>Basic Service Set</td>
</tr>
<tr>
<td>BSSID</td>
<td>BSS Identifier</td>
</tr>
<tr>
<td>CLI</td>
<td>Command Line Interface</td>
</tr>
<tr>
<td>DHCP</td>
<td>Dynamic Host Configuration Protocol</td>
</tr>
<tr>
<td>DNAT</td>
<td>Destination Network Address Translation</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>ICMP</td>
<td>Internet Control Message Protocol</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IGRP</td>
<td>Interior Gateway Routing Protocol</td>
</tr>
<tr>
<td>IOS</td>
<td>Internetwork Operating System</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ISM</td>
<td>Industrial, Scientific, and Medical</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organisation for Standardisation</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunications Union</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>MAC</td>
<td>Medium Access Control</td>
</tr>
<tr>
<td>MSS</td>
<td>Maximum Segment Size</td>
</tr>
<tr>
<td>MTU</td>
<td>Maximum Transfer Unit</td>
</tr>
<tr>
<td>NAPT</td>
<td>Network Address and Port Translation</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>NIC</td>
<td>Network Interface Card</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnection</td>
</tr>
<tr>
<td>OSPF</td>
<td>Open Shortest Path First</td>
</tr>
<tr>
<td>PAT</td>
<td>Port and Address Translation</td>
</tr>
<tr>
<td>PDU</td>
<td>Protocol Data Unit</td>
</tr>
<tr>
<td>PPP</td>
<td>Point to Point Protocol</td>
</tr>
<tr>
<td>RFC</td>
<td>Request For Comments</td>
</tr>
<tr>
<td>RIB</td>
<td>Routing Information Base</td>
</tr>
<tr>
<td>RID</td>
<td>Router ID</td>
</tr>
<tr>
<td>RIP</td>
<td>Routing Information Protocol</td>
</tr>
<tr>
<td>RIR</td>
<td>Regional Internet Registries</td>
</tr>
<tr>
<td>RR</td>
<td>Resource Record</td>
</tr>
<tr>
<td>RTO</td>
<td>Retransmission Time-Out</td>
</tr>
<tr>
<td>RTT</td>
<td>Round Trip Time</td>
</tr>
<tr>
<td>SOHO</td>
<td>Small Office Home Office</td>
</tr>
<tr>
<td>STP</td>
<td>Spanning Tree Protocol</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TTL</td>
<td>Time To Live</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>VLAN</td>
<td>Virtual Local Area Network</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
<tr>
<td>WAN</td>
<td>Wide Area Network</td>
</tr>
<tr>
<td>WI</td>
<td>Web Interface</td>
</tr>
<tr>
<td>WDS</td>
<td>Wireless Distribution System</td>
</tr>
<tr>
<td>WLAN</td>
<td>Wireless Local Area Network</td>
</tr>
<tr>
<td>WiFi</td>
<td>Wireless Fidelity</td>
</tr>
</tbody>
</table>
References