

Open Source Routing Suites

Hege Trosvik, University of Oslo, USIT

Abstract

Traditionally, implementing IP routing meant buying an expensive piece of hardware from a small set of routing equipment vendors and an equally expensive piece of closed source software to run on this proprietary hardware. With the increased focus on and popularity of open source software, those days are gone and there are now several fairly complete open source IP routing stack implementations available. Most of these run on standard PC hardware and are quite simple to build, install and configure. There are still plenty of reasons to buy routers from the traditional router vendors, but for enterprises with the necessary skills and time looking for an inexpensive, low to medium performance router, there are several open source routing implementations worth looking at.

In the iLabs Open Source Software Initiative we have implemented four of these open source routing solutions. This paper describes our network and the basic configurations and effort required to set up these four open source routing solutions.

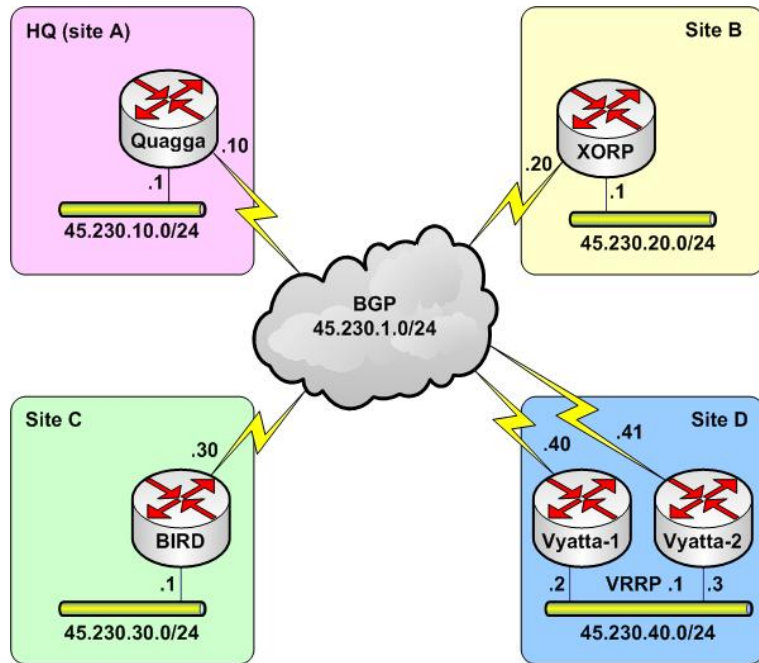
Network Topology

Our network simulates a company with three sites (A, B, C) running mainly open source operating systems and applications on all servers and the majority of desktops to show how a complete infrastructure can be implemented with open source applications. A fourth, newly acquired site (D), had an infrastructure based almost entirely on closed source software at the time of the acquisition. This allows us to show the integration of open source software with a closed source infrastructure.

Three of the four sites run different open source routing suites; Quagga, XORP and BIRD on 1RU i86 PC platforms with Fedora Core 4 (FC4). The fourth site has two PCs that were pre-installed with Vyatta OFR software (XORP based) running VRRP for redundancy.

All routers are configured as BGP peers to the routers at the other sites. **Figure 1** shows the network topology.

Figure 1 OSS Network Topology



Quagga

Quagga (<http://www.quagga.net/>) is an open source routing software suite that supports IPv4 as well as IPv6, provides implementations of OSPFv2, OSPFv3, RIP v1 and v2, RIPv3 and BGPv4, and runs on several flavors of Unix. Support for multicast and MPLS is in the process of being implemented. Quagga is a fork of GNU Zebra which was developed by Kunihiro Ishiguro. The Quagga Routing Software Suite is provided under the gnu General Public License (GPL). The programming language is C.

Quagga has a modular design consisting of several independent daemons that work together to build the kernel routing table. There are separate daemons for RIP, OSPF and BGP, and the zebra daemon communicates with the individual routing daemons providing communication between them and updating the kernel routing table. Each of the daemons has its own configuration file which can either be edited with a text editor or through the zebra Command Line Interface (CLI), `vttysh`. Interface information, routing tables etc. is also easily accessible through `vttysh`.

We tried to use a publicly available RPM for XORP 1.2 on FC4, but soon realized that downloading the source and building and installing by hand was a much better option. We installed xorp under `/usr/local/xorp/` which is the default. XORP hides the internal structure from the user and one process manages the whole XORP router, `xorp_rtmgr` (XORP Router Manager). XORP has one configuration file, by default `config.boot`, but you can make `xorp_rtmgr` use a different configuration file by starting it with the `-b` argument. We used `/home/xorp/config.boot` as the config file and created an init file where xorp was started with `xorp_rtmgr -b /home/xorp/config.boot`.

The configuration can be edited by editing the file directly or with the `xorpsh`. The syntax is Juniper like and pretty straight forward to edit and read for anybody who is somewhat familiar with JunOS. **Figure 3** shows the `config.boot` configuration file we used in our setup. XORP is b-gw in **figure 1** and is peering BGP with each of the three other routers in the picture.

The XORP project is looking at supporting custom hardware and software forwarding architectures in the future. The project also has ties to the Click (<http://pdos.csail.mit.edu/click>) projects which is an open source project to build a flexible, configurable and modular router with separate modules, or “elements”, for simple router functions like packet classification, queuing, scheduling and interfacing with network devices.

BIRD

The BIRD (BIRD Internet Routing Daemon) project (<http://bird.network.cz/>) aims to develop a fully functional dynamic IP routing daemon primarily targeted for UNIX like systems. BIRD supports both IPv4 and IPv6, BGP, RIP, OSPF (IPv4 only) and static routes. BIRD also has a number of advanced features like multiple routing tables, inter-table protocols, soft reconfiguration and route filtering. The BIRD web page also lists a number of features that “surely deserve to be implemented in the future versions of BIRD”, including among others OSPF for IPv6, OSPF NSSA and opaque LSAs, route aggregation and flap dampening, multipath routes, multicast routing protocols and ports to other systems, so this project will also be an interesting one to watch going forward. BIRD has been developed as a school project at Faculty of Math and Physics, Charles University Prague. BIRD is distributed under the GNU General Public License. BIRD is written in GNU C.

BIRD has a simple, modular architecture consisting of various core, library, resource management, protocol, filter and system dependent modules as well as configuration modules and the client which provides access to a CLI. The configuration file is edited manually. The CLI does not modify the configuration, but has been designed as an inspection tool only. The CLI can also disable, enable and restart protocols and load new configuration files on the fly, ensuring smooth transition between configurations.

Figure 4 BIRD configuration file

```
bird# more /etc/bird.conf

# Override router ID
router id 45.230.1.30;

filter bgp_out {
    if net ~ 45.230.30.0/24 then accept;
    else reject;
}

protocol direct {
    interface "*";
}

protocol kernel {
    persist;
    scan time 20;
    export all;
}

protocol device {
    scan time 10;
}

protocol static {
    route 0.0.0.0:0.0.0.0 via 45.230.1.1;
}
```

```
protocol bgp bgp10 {
    local as 65030;
    neighbor 45.230.1.10 as 65010;
    source address 45.230.1.30;
    export filter bgp_out;
    import all;
}

protocol bgp bgp20 {
    local as 65030;
    neighbor 45.230.1.20 as 65020;
    source address 45.230.1.30;
    export filter bgp_out;
    import all;
}

protocol bgp bgp40 {
    local as 65030;
    neighbor 45.230.1.40 as 65040;
    export filter bgp_out;
    import all;
}

protocol bgp bgp41 {
    local as 65030;
    neighbor 45.230.1.41 as 65040;
    export filter bgp_out;
    import all;
}
```

We used the BIRD 1.0.11 RPMs provided, and they installed without problems on FC4. BIRD needs a basic configuration file to start. Once there is a basic configuration file, BIRD can be started manually, optionally with `-c` and its configuration file as argument. Init files were also provided, we used these to enable BIRD at startup. **Figure 4** shows the BIRD config file.

Vyatta

Vyatta (<http://www.vyatta.com>) provides an open source router called the Vyatta OFR (Open, Flexible Router) based on XORP. The OFR supports the same IPv4 unicast routing protocols as XORP; OSPFv2, BGPv4 and RIPv2 in addition to static routes. IPv6 and multicast is not available in the first release, but support is planned for future releases. The OFR also supports many additional features that XORP does not support, including VRRP, SNMP, DHCP, stateful inspection firewall and NAT and also includes both a common CLI for all the features as well as a web GUI.

The OFR software currently comes as a complete system bundled with Debian Linux and runs on commodity PC hardware. There are also rumors that Vyatta is planning to offer an appliance with their OFR software, but no details are available at this point.

We received two pre-installed rack-mountable PCs with the OFR version 0.5 code. Configuring them was quite easy since we already knew how to configure XORP. The XORP CLI is also very similar to the Juniper CLI. One of the features we found very useful on the OFR is VRRP. In our iLabs network, we run two OFRs in site D with VRRP between them for redundancy. **Figure 5** shows the interface portion of one of our OFRs as an example of the VRRP configuration. The routing configuration is similar to the XORP configuration shown in **figure 3** above.

The OFR consists of a number of different components from a number of sources, nicely integrated and packaged. All of the different components have licenses defined as open source licenses, but the exact license type varies among the components. The OFR will certainly be interesting to watch going forward with features like IPSec VPN, WAN optimization, intrusion detection and Server load balancing on the roadmap in addition to IPv6 and multicast as mentioned above. According to the Vyatta web site, "Vyatta" is an old Sanskrit word meaning "open" which must mean they are pretty committed to keeping the OFR code open.

Conclusion

In general, installing the software for any of these routing stacks is fairly simple, but the configs might require a bit of tweaking before you get them right, even if you are used to working with the commonly used proprietary routers.

None of these implementations provide their own forwarding, but rather they rely on the forwarding of the underlying host operating system. One thing to look out for is that not all of these implementations will set the kernel to do ip forwarding and you may have to do this manually. This can be done temporarily by doing `echo 1 > /proc/sys/net/` or permanently by editing `/etc/sysctl.conf`.

Most of the open source routing solutions have implementations of the most common unicast routing protocols, some of them have IPv6 support and few have multicast support. None of the solutions we worked with have support to create GRE tunnels through their CLIs, XORP and OFR can create subinterfaces through their CLIs, and most of them can utilize VLANs and tunnels created by the operating system.

Except for the OFR, all of these solutions are pure routing solutions with no firewall. With the current industry trend towards integrated solutions, there is reason to expect that network operators will soon be looking for open source products with integrated routing and security.

```

root@vyatta-1# edit interfaces
[edit interfaces]
root@vyatta-1# show
  loopback lo {
  }
  ethernet eth0 {
    address 45.230.1.40 {
      prefix-length: 24
      broadcast: 45.230.1.255
    }
  }
  ethernet eth1 {
    address 45.230.40.2 {
      prefix-length: 24
      broadcast: 45.230.40.255
    }
    vrrp {
      virtual-address: 45.230.40.1
      priority: 100
    }
  }
[edit interfaces]

```

Figure 5 Vyatta OFR VRRP configuration