Routing with OpenBSD using OpenOSPFD and OpenBGPD

Claudio Jeker <claudio@openbsd.org>, The OpenBSD Project

Abstract

OpenBSD includes OpenBGPD and OpenOSPFD, two secure and powerful routing daemons. Combined with additional network components like pf, IPSec, carp(4) and trunk(4) this results in a very powerful routing and network platform.

History

OpenBGPD development started in October 2003. Henning Brauer contacted me when he started to write a BGP daemon that did not suck. At that time he already had 3000 lines of code including the basic framework and a minimal session engine. Around then we had many issues with our Zebra based BGP routers at work. Andre Oppermann and I designed the route decision engine and I began to write some code. Two month later a first basic version of OpenBGPD got imported into OpenBSD and I got my @openbsd.org account. OpenBSD 3.5 included the first release of OpenBGPD.

One year later in December 2004 Esben Norby contacted Henning and afterwards me about his xOSPFD he was working on. Esben used OpenBGPD as a starting point for his xOSPFD but faced some troubles with the "imsg" subsystem. His request came at the right time because my need for a good and compatible OSPF implementation became more and more apparent. A few days later I sent Esben a huge diff to fix his imsg issues and started to hack on that project as well. One month later I imported OpenOSPFD into the OpenBSD source tree. OpenOSPFD is part of OpenBSD since the 3.7 release.

In the two years of OpenBGPD development around 20'000 lines of code and documentation where written in about 2000 commits -- not including bgpctl.

OpenOSPFD is currently at around 10'000 lines of code and documentation done in about 650 commits.

Overview

Protocols

The Internet is split into regions called Autonomous Systems (AS). Each AS is under the control of a single administrative entity -- for example a university or an ISP. The edge routers of these AS use an Exterior Gateway Protocol (EGP) to exchange routing information between AS. Currently BGP4, the Border Gateway Protocol is the only EGP in widespread use. Routers within an AS use an Interior Gateway Protocol to exchange local routing information. There are different IGPs. OSPF, IS-IS, and RIP are the most commonly used. It is possible and not uncommon to have multiple IGPs running inside one AS.

BGP

The Border Gateway Protocol (BGP) is a path distance vector protocol. BGP is used to exchange routing information between AS. It is probably the most important core infrastructure protocol of the Internet. BGP routers use persistent TCP connections to connect to the direct neighbors. Updates are only exchanged with these direct neighbors and these neighbors decide for them self if this update needs to be sent to further routers. BGP has sophisticated filtering capabilities to control the traffic flow. The decision process uses more than one metric to decide which is the most preferred route. Currently around 185'000 routes are exchanged over a BGP full view. Only one route out of a number available one's can be active.

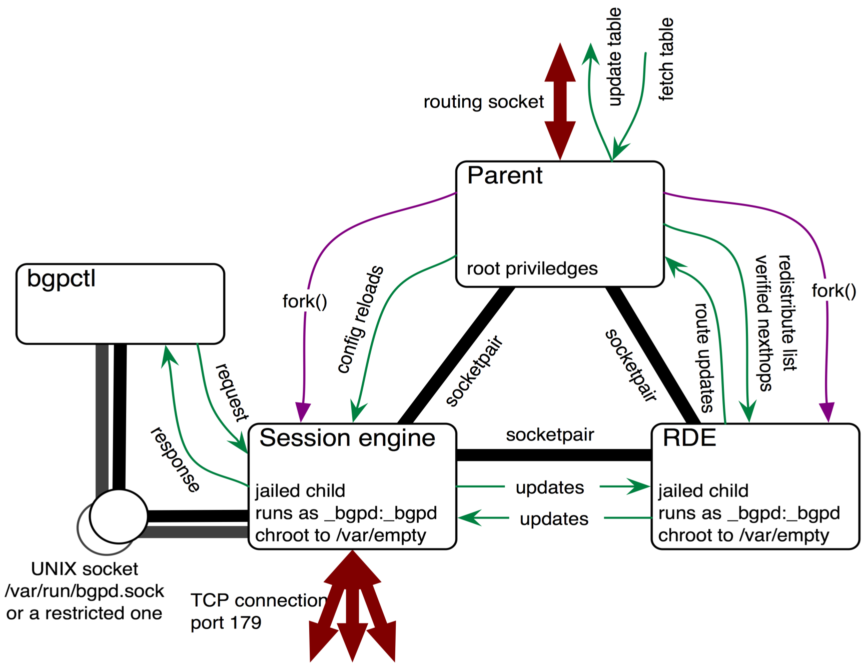
OSPF

The Open Shortest Path First (OSPF) protocol is a link-state, hierarchical routing protocol. It is probably the most used IGP in the world. It is capable of doing neighbor discovery on different types of networks with minimal need for configuration. OSPF encapsulates its routing messages directly on top of IP as its own protocol type (89). TCP connections are not used because the link-state flooding algorithm already includes its own way for reliable communications -- adding to OSPF's complexity. Most obvious the massive use of IP multicast in OSPF makes TCP infeasible and OSPF hard to implement.

Internals

OpenBGPD

OpenBGPD is split into three processes. The parent process, the Session Engine (SE), and the Route Decision Engine (RDE). Both SE and RDE run in a chroot jail and drop privileges on startup. Only the parent process runs with root privileges -- this is necessary to update the routing table.

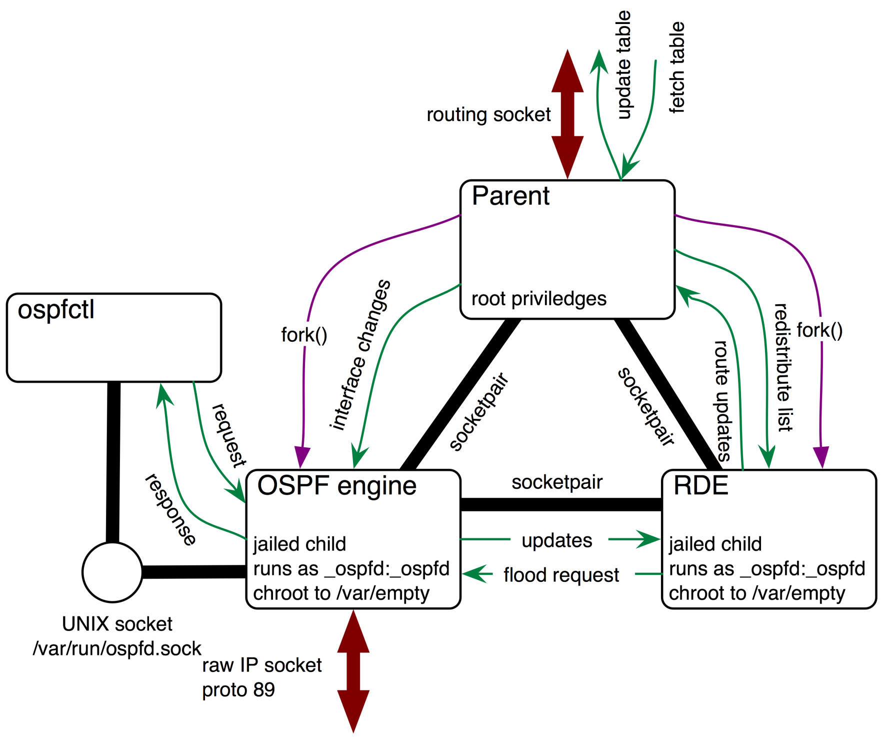


The Session Engine communicates with all neighbors and sends out the keep-alive messages. Updates are passed to the Route Decision Engine where the filtering and routing table calculations are done. Changes in the RIB -- the Routing Information Base -- are sent to the parent process to update the FIB -- the Forward Information Base -- and to the SE to send updates out to the neighbors.

The system scheduler ensures that complex calculations done in the RIB do not starve the session management which needs to send out periodic keep-alive messages.

OpenOSPFD

The design of OpenOSPFD is heavily based on the three process design of OpenBGPD. Again there is the parent process and the Route Decision Engine (RDE). The Session Engine was renamed to OSPF Engine (OE) but the task is similar. Again only the parent process runs with root privileges, the other two processes run in a chroot jail and drop privileges on startup.



The OSPF Engine communicates with the neighbors and runs all the finite state machines. The periodic hello messages are generated by the OE. Updates are passed to the Route Decision Engine. The RDE calculates the SPF (shortest path first) tree and based on this info the RIB is generated. Updates are passed to the parent process for the FIB and to the OE so that the updates are flooded on through the network.

Basic Setups

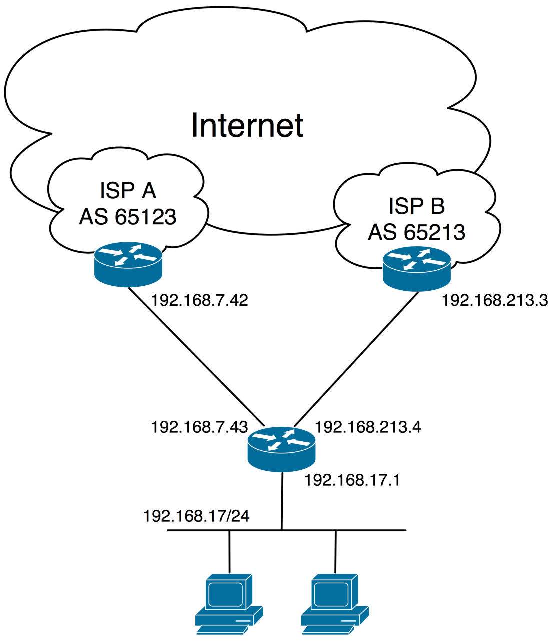
bgpd(8) and ospfd(8) are used on routers so the first thing you need to do is to edit /etc/sysctl.conf and enable IP forwarding by uncommenting the "net.inet.ip.forwarding=1" line. This is important for ospfd as the link-state protocol requires that all OSPF routers are capable of forwarding traffic.

Additionally the network interfaces should be configured correctly as described in hostname.if(5).

OpenBGPD

The most common need for a BGP setup is multi-homing. It gives maximum redundancy by having independent uplinks via different ISPs. If one ISP runs into problems there is still the other link that is hopefully not affected. This is where BGP kicks in. It calculates the full routing table based on the views of the uplink ISPs. If one link fails the routes of this provider are no longer considered and the traffic flows via a different link.

Sounds easy but before firing up bgpd(8) some prerequisites need to made. First of all you need an AS number and provider independent address space. Both can be requested at one of the local RIRs (www.ripe.net for Europe) but it is often easier to ask your ISP for help because filling out the request forms is not that simple. Additionally you need at least two different upstream providers.



With all the necessary information available it is easy to configure OpenBGPD.

The config file is split in three sections:

- global configuration
- neighbor configuration
- filters

Global Config

```
# global config
AS 65042
router-id 192.168.17.1
# announce our PI address space
network 192.168.17/24
```

Three things need to be set in the global config. First of all the AS needs to be set. Optionally the router-id is set this is not necessary but I prefer to use the local IP address as router-id. The third statement is used to announce the assigned IP block telling everybody that 192.168.17/24 belongs to AS 65042. Without that info the IP traffic will not find his way back. Additionally the listen address can be restricted but this is not needed.

Neighbor Config

```
# neighbor config
neighbor 192.168.7.42 {
     descr          "ISP A"
     remote-as 65123
}

neighbor 192.168.213.3 {
     descr          "ISP B"
     remote-as 65213
     tcp md5sig     password "74#senEjTYmhQ/f"
}
```

Two neighbors are configured. The only required info is the IP in the neighbor statement and the remote-as number. The rest is eye-candy (descr) and added security (tcp-md5 protection). For a simple setup nothing else is needed and you could now fire up bgpd.

Filters

But not so fast probably we should setup some filters so that only valid routes are accepted by bgpd. This sounds complex but luckily the default /etc/bgpd.conf file includes already a good default filter set.

```
# filter out prefixes longer than 24 or shorter than 8 bits
deny from any
allow from any prefixlen 8 - 24

# do not accept a default route
deny from any prefix 0.0.0.0/0

# filter bogus networks
deny from any prefix 10.0.0.0/8 prefixlen >= 8
deny from any prefix 172.16.0.0/12 prefixlen >= 12
deny from any prefix 192.168.0.0/16 prefixlen >= 16
deny from any prefix 169.254.0.0/16 prefixlen >= 16
deny from any prefix 192.0.2.0/24 prefixlen >= 24
deny from any prefix 224.0.0.0/4 prefixlen >= 4
deny from any prefix 240.0.0.0/4 prefixlen >= 4
```

For each update message processed by the filter, the filter rules are evaluated in sequential order, from first to last. The last matching allow or deny rule decides what action is taken.

The following rule-set allows only prefixes with a prefix length between a /8 and /24. Then the rule-set explicitly denies the default route plus other non routeable networks like those defined in RFC 1918. The example is using RFC 1918 addresses so it will not correctly work out of the box because of the filters but that's on purpose.

bgpctl

Now we're ready to start bgpd but first let's add 'bgpd_flags=""' to /etc/rc.conf.local so that bgpd is automatically started after a reboot.

Now it's time to have a look at the sessions with bgpctl.

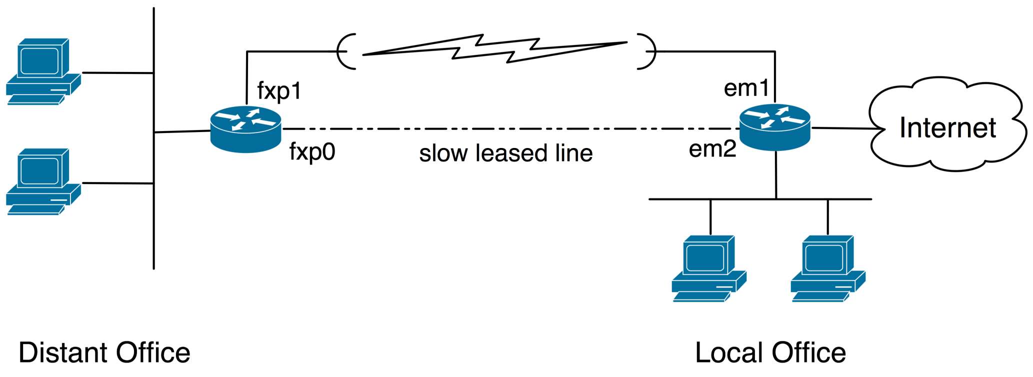
```
router:~> bgpctl show
Neighbor                 AS    MsgRcvd    MsgSent    OutQ  Up/Down  State/PrefixRcvd
ISP A                 65123      38703        138       0 01:09:02 182394
ISP B                 65213      36533        136       0 01:07:12 182952
```

Everything looks good and so the basic bgpd setup is finished.

OpenOSPFD

OSPF is used to calculate the local routing table in an automatic way. While it is possible to add and remove static routes manually for small networks it becomes infeasible for larger ones. Another feature of routing protocols is to adapt to network issues and reroute traffic if an alternate path exists.

In our first example a distant office is connected via two links to the main office. One of the links is a not so reliable but fast wireless link the other is a slow leased line. If the wireless link is up and running traffic should use this fast route. The leased line is used only as backup.



Two OpenBSD systems running OpenOSPFD are used as routers at each end of these links. Their setups are similar but not equal. First let's have a look at the router in the distant office.

Distant Office: ospfd.conf

```
# global configuration
router-id 0.0.0.2
redistribute connected

# areas
area 0.0.0.0 {
     auth-type crypt
     auth-md 1 "L&Y6)@Vc(6I4=Gq"
     auth-md 2 "IsJ_-LE:al17a"
     auth-md-keyid 1

     # main link
     interface fxp1 { metric 10 }
     # backup link
     interface fxp0 { metric 100 }
}
```

Again there is a global config. The router-id can be selected automatically but it is good style to set a router-id to make it easier to identify routers. In our little network we use 0.0.0.1 and 0.0.0.2 in bigger networks it is better to use the main IP address of the router as id. To inform the other side about the networks connected to this router we need to redistribute them. This is done by "redistribute connected". It is possible to redistribute static routes by adding a "redistribute static" line.

The next section configures areas. All examples will only use one area. 0.0.0.0 is the backbone area, if multiple areas are used all other areas need to be directly connected to this backbone area.

The next four lines are used to secure the OSPF setup. Only auth-type crypt is secure enough to protect against packet insertion attacks. Two keys are defined and accepted but the first one is used for signing outgoing packets. The setup is easy so use it :).

Now we define the interfaces. The main link(fxp1) uses a low metric so that it is preferred over the backup link (fxp0).

Local Office: ospfd.conf

The other box has a very similar setup

```
# global configuration
router-id 0.0.0.1
redistribute default

# areas
area 0.0.0.0 {
     auth-type crypt
     auth-md 1 "L&Y6)@Vc(6I4=Gq"
     auth-md 2 "IsJ_-LE:al17a"
     auth-md-keyid 1

     # main link
     interface em1 { metric 10 }
     # backup link
     interface em2 { metric 100 }
     # possible additional interfaces
}
```

The only things that changed are the router-id, the interface names and the redistribute line. Instead of announcing the connected networks we just announce the default route so that all traffic from the distant office is sent to this router.

We need to add 'ospfd_flags=""' to /etc/rc.conf.local on both routers so that ospfd is started on reboots. Now ospfd can be started on both routers.

ospfctl

With ospfctl we can monitor the neighbors and the calculated routing table.

```
First router at distant office

# ospfctl show nei
ID              Pri State        DeadTime Address         Iface     Uptime
0.0.0.1         1   FULL/DR      00:00:39 192.168.254.5   fxp0      00:00:16
0.0.0.1         1   FULL/DR      00:00:39 192.168.254.1   fxp1      00:00:06

# ospfctl show rib
Destination          Nexthop            Path Type    Type       Cost     Uptime
0.0.0.1              192.168.254.1      Intra-Area   Router     10       00:01:14
192.168.254.0/30     192.168.254.2      Intra-Area   Network    10       00:01:19
192.168.254.4/30     192.168.254.6      Intra-Area   Network    100      00:01:29
0.0.0.0/0            192.168.254.1      Type 1 ext   Network    110      00:01:14

Now the local router
# ospfctl show nei
ID              Pri State        DeadTime Address         Iface     Uptime
0.0.0.2         1   FULL/BCKUP   00:00:30 192.168.254.6   em2       00:03:10
0.0.0.2         1   FULL/BCKUP   00:00:30 192.168.254.2   em1       00:03:00

# ospfctl show rib
Destination          Nexthop            Path Type    Type       Cost     Uptime
0.0.0.2              192.168.254.2      Intra-Area   Router     10       00:03:43
```

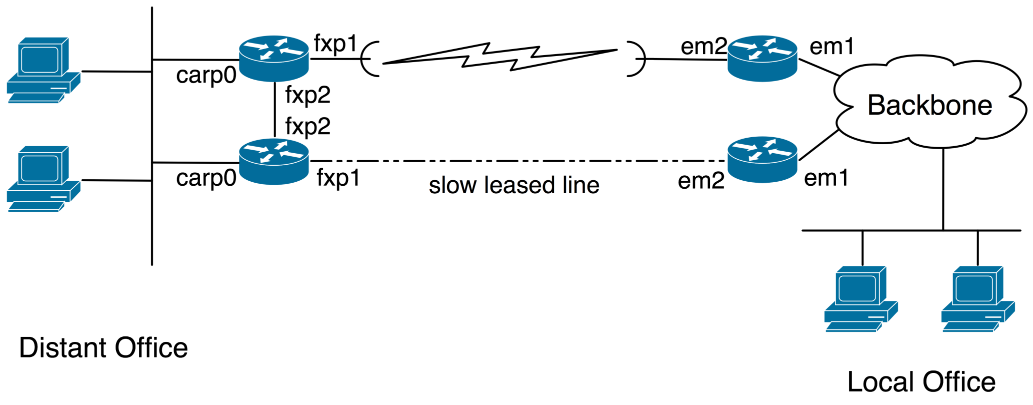
```
192.168.254.0/30     192.168.254.1     Intra-Area   Network   10       00:03:53
192.168.254.4/30     192.168.254.5     Intra-Area   Network   100      00:04:03
192.168.2.0/24       192.168.254.2     Type 1 ext   Network   110      00:03:43

AS external LSA -- those routes with Path Type "Type 1 ext" have a default
metric of 100.
```

Optimising Setups

In the BGP case and the OSPF case we have redundant links but only one router. The routers are now the single point of failure. So let's optimise the setups to add additional redundancy.

OpenOSPFD

In most cases it is simple to add additional routers to get more redundancy. In the previous example we could use one dedicated router for each link end so a total of four routers. Sounds good, let's add these routers.



Now we have two routers at each place but how do the other systems know where to send the traffic. On the backbone side this is not a big problem because the routers there will run OSPF and so get the correct information but the workstations in the office LAN are not able to use OSPF. This cries for carp(4) the common address redundancy protocol.

OpenOSPFD will honour the state of the carp(4) interface and only the router that is carp master will announce the network to the other routers. There is one problem left OpenOSPFD is not able to preempt the carp(4) interface if one of the OSPF links goes down. Currently the best way to solve this issue is to add a direct connection between the two carp routers.

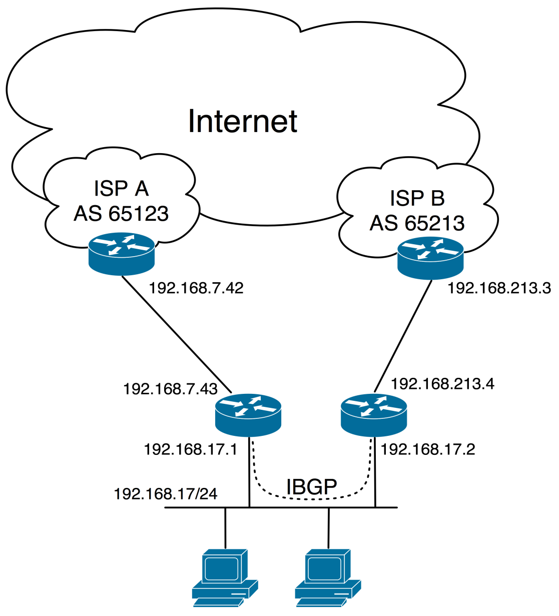
Here is the example config of one router. The other one has an equal config.

```
# global configuration
router-id 0.0.0.2

# areas
area 0.0.0.0 {
     auth-type crypt
     auth-md 1 "L&Y6)@Vc(6I4=Gq"
     auth-md 2 "IsJ_-LE:al17a"
     auth-md-keyid 1

     # main link
     interface fxp1 { metric 10 }
     # cross link between the two routers
     interface fxp2 { metric 20 }
     # announce local LAN via carp
     interface carp0
}
```

The big change is that "redistribute connected" got replaced with a "interface carp0". This ensures that the announced network depends on the interface state of carp0. It is not recommended to use "redistribute connected" with carp(4) because the connected route is attached to the parent interface and so depends on the link state of that interface and not of the carp(4) one. The result would be that the router with the backup carp(4) interface will announce the network as well.

OpenBGPD

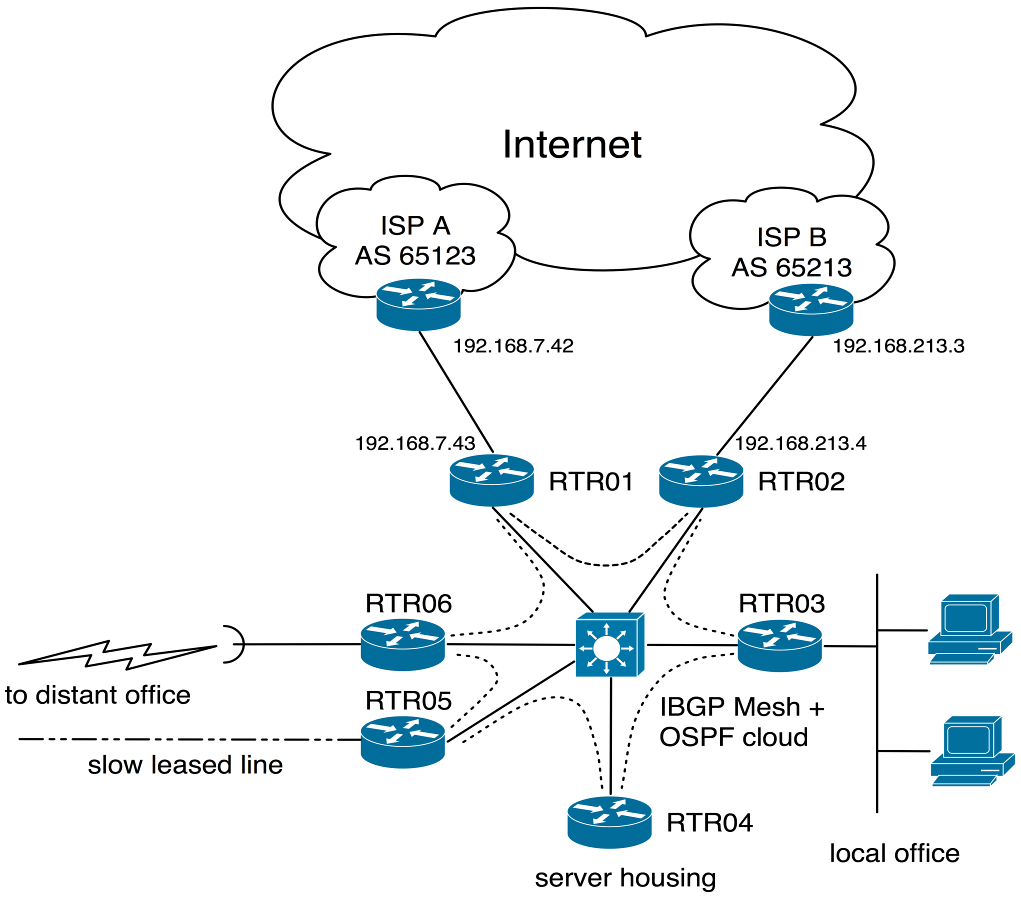
Using multiple BGP routers is not that trivial. Sure you could carp(4) your uplinks and use the "depend on" config statement but this will not result in a clean fail-over as all sessions will be reset.



Instead uplinks are terminated on different routers. But now every router has a different view and this information needs to be consolidated. This is achieved

by setting up a IBGP full mesh. Sessions between routers of the same AS are called IBGP -- interior BGP – sessions.
To get the traffic on the best path out of the network additional IBGP routers need to be setup so that unneeded hops between the two uplink routers can be avoided.

Now IBGP sessions are a bit special because the nexthop is passed unmodified to other IBGP routers. This comes from the fact that an IBGP mesh is layered on a IGP network and so all local routes are known and so all nexthop resolve nicely over the IGP routes. If you do not run a IGP your in trouble because the passed nexthops are not reachable form the other router and so all prefixes sent are marked invalid. A possible workaround is to set the nexthop to "self" and by doing so enforce a reachable nexthop. There are other more complex and error prone ways of running IBGP over a IGP less network that I will not cover here, these setups are for crazy experts only. I do prefer to run a simple IGP on my networks.



So here is the important config for this network.

```
**** RTR01 ****
---- bgpd.conf ----
# global config
AS 65042
router-id 192.168.17.1
# announce our PI address space
network 192.168.17/24

# neighbor config
```

```
neighbor 192.168.7.42 {
     descr         "ISP A"
     remote-as 65123
}

group IBGP {
     remote-as 65042
     neighbor 192.168.17.2 {
          descr    "RTR02"
     }
     neighbor 192.168.17.3 {
          descr    "RTR03"
     }
     neighbor 192.168.17.4 {
          descr    "RTR04"
     }
     neighbor 192.168.17.5 {
          descr    "RTR05"
     }
     neighbor 192.168.17.6 {
          descr    "RTR06"
     }
}

# filter out prefixes longer than 24 or shorter than 8 bits
deny from any
allow from any prefixlen 8 - 24

# do not accept a default route
deny from any prefix 0.0.0.0/0

# filter bogus networks
deny from any prefix 10.0.0.0/8 prefixlen >= 8
deny from any prefix 172.16.0.0/12 prefixlen >= 12
deny from any prefix 192.168.0.0/16 prefixlen >= 16
deny from any prefix 169.254.0.0/16 prefixlen >= 16
deny from any prefix 192.0.2.0/24 prefixlen >= 24
deny from any prefix 224.0.0.0/4 prefixlen >= 4
deny from any prefix 240.0.0.0/4 prefixlen >= 4
----
---- ospfd.conf ----
# global configuration
router-id 0.0.0.1
redistribute connected

# areas
area 0.0.0.0 {
     auth-type crypt
     auth-md 1 "L&Y6)@Vc(6I4=Gq"
     auth-md 2 "IsJ_-LE:al17a"
     auth-md-keyid 1

     # main link
     interface fxp1
}
----

**** RTR02 *****
---- bgpd.conf ----
# global config
AS 65042
router-id 192.168.17.2
```

```
# announce our PI address space
network 192.168.17/24

# neighbor config
neighbor 192.168.213.3 {
     descr          "ISP B"
     remote-as 65213
     tcp md5sig     password "74#senEjTYmhQ/f"
}

group IBGP {
     remote-as 65042
     neighbor 192.168.17.1 {
          descr     "RTR01"
     }
     neighbor 192.168.17.3 {
          descr     "RTR03"
     }
     neighbor 192.168.17.4 {
          descr     "RTR04"
     }
     neighbor 192.168.17.5 {
          descr     "RTR05"
     }
     neighbor 192.168.17.6 {
          descr     "RTR06"
     }
}

# filter out prefixes longer than 24 or shorter than 8 bits
deny from any
allow from any prefixlen 8 - 24

# do not accept a default route
deny from any prefix 0.0.0.0/0

# filter bogus networks
deny from any prefix 10.0.0.0/8 prefixlen >= 8
deny from any prefix 172.16.0.0/12 prefixlen >= 12
deny from any prefix 192.168.0.0/16 prefixlen >= 16
deny from any prefix 169.254.0.0/16 prefixlen >= 16
deny from any prefix 192.0.2.0/24 prefixlen >= 24
deny from any prefix 224.0.0.0/4 prefixlen >= 4
deny from any prefix 240.0.0.0/4 prefixlen >= 4
----
---- ospfd.conf ----
# global configuration
router-id 0.0.0.2
redistribute connected

# areas
area 0.0.0.0 {
     auth-type crypt
     auth-md 1 "L&Y6)@Vc(6I4=Gq"
     auth-md 2 "IsJ_-LE:al17a"
     auth-md-keyid 1

     # main link
     interface fxp1
}
----
```

```
**** RTR03 *****
---- bgpd.conf ----
# global config
AS 65042
router-id 192.168.17.3

# neighbor config
group IBGP {
     remote-as 65042
     neighbor 192.168.17.1 {
          descr    "RTR01"
     }
     neighbor 192.168.17.2 {
          descr    "RTR02"
     }
     neighbor 192.168.17.4 {
          descr    "RTR04"
     }
     neighbor 192.168.17.5 {
          descr    "RTR05"
     }
     neighbor 192.168.17.6 {
          descr    "RTR06"
     }
}

# no need to re-filter here
----
---- ospfd.conf ----
# global configuration
router-id 0.0.0.3
redistribute connected

# areas
area 0.0.0.0 {
     auth-type crypt
     auth-md 1 "L&Y6)@Vc(6I4=Gq"
     auth-md 2 "IsJ_-LE:al17a"
     auth-md-keyid 1

     # main link
     interface fxp0
}
----

**** RTR04 has almost the same config as RTR03 ****

**** RTR05 ****
---- bgpd.conf is similar to RTR03 -- only the IBGP mesh needs some tweaks ----
---- ospfd.conf ----
# global configuration
router-id 0.0.0.5

# areas
area 0.0.0.0 {
     auth-type crypt
     auth-md 1 "L&Y6)@Vc(6I4=Gq"
     auth-md 2 "IsJ_-LE:al17a"
     auth-md-keyid 1

     # main link
     interface em0
     # slow leased line (backup)
     interface em2 {
          metric 100
     }
}
----

**** RTR06 ****
---- bgpd.conf is similar to RTR03 -- only the IBGP mesh needs some tweaks ----
---- ospfd.conf ----
# global configuration
router-id 0.0.0.6

# areas
area 0.0.0.0 {
     auth-type crypt
     auth-md 1 "L&Y6)@Vc(6I4=Gq"
     auth-md 2 "IsJ_-LE:al17a"
     auth-md-keyid 1

     # main link
     interface em0
     # fast wireless link
     interface em1 {
          metric 10
     }
}
----
```

This are the basic steps for setting up OpenBGPD and OpenOSPFD in a network. This document covers only the very basic setup. More information about OpenBGPD and OpenOSPFD can be found in the OpenBSD man pages for bgpd(8), bgpctl(8), bgpd.conf(5), ospfd(8), ospfctl(8), and ospfd.conf(5).

About the Author

Claudio Jeker studied electrical engineering at the ETH Zurich with a heavy bias in information technology and networking. Since four years he is working for a small company doing network consulting and developement. This includes the developement of PCI cards with T1/E1 and G.SHDSL interfaces.
He is one of the main developers of both OpenBGPD and OpenOSPFD and does additional OpenBSD network stack hacking. Since about two years Claudio Jeker is an OpenBSD developer with full source code commiter status.

Legal Notice