STORMSHIELD

TECHNICAL NOTE

STORMSHIELD NETWORK SECURITY

BIRD DYNAMIC ROUTING

Product concerned: SNS 3.x

Date: November 2019

Reference: sns-en-bird_dynamic_routing_technical_note_V3

Table of contents

Introduction

The aim of this document is to guide the administrator of a Stormshield Network firewall in configuring and operating the embedded BIRD dynamic routing module.

To begin with, the configuration environment as well as the interaction modes with the routing engine will be described. Next, a simple typical configuration for the three routing protocols BGP, RIP & OSPF will be explained. These examples provide an opportunity for learning about the configuration structure of protocols, peripheral elements, filtering and status displays. The last section focuses on a more complex configuration.

Take note that BIRD offers multiple configuration options, especially for the exchange of routes between processes, their filters or a pseudo-virtualization of routing instances. These advanced elements are specific to BIRD and are not included in the scope of the document. Likewise, the use of BGP ROAs will not be covered but they are supported by BIRD.

Configuration via the web administration interface

Dynamic routing can be enabled or disabled in the **Routing** module of the web administration interface.

The *Dynamic routing* and *Dynamic routing IPv6* (if IPv6 support has been enabled) tabs make it possible to edit configuration files *bird.conf* in IPv4 and *bird6.conf* in IPv6.

When the configuration is sent to the firewall, and if there are syntax errors, a message indicating the row containing an error will inform the user of the need to correct the configuration.

> **WARNING**
> The previous configuration file will not be backed up before it has been modified. You are therefore advised to copy it (CTRL+A/CTRL+C), and paste it into a text editor in order to back up the file.

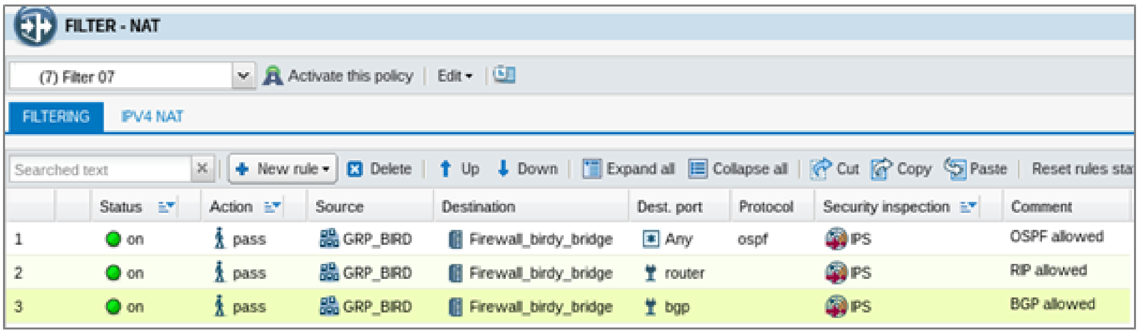
However, the interface editor does not allow access to Birdc and birdc6 interactive modes, which enable control over dynamic routing (testing the operation of a new configuration through a temporary configuration and viewing statuses).

BIRD / Stormshield Network environment

In a factory configuration, the BIRD routing module is not enabled. Routing of Stormshield Network firewall may be made to coexist with BIRD dynamic routing. For example, the internal zone may be managed with a dynamic routing protocol and the external zone with Firewall's routing features (static routing, gateways, routing by rule (PBR), router objects).

To do so, please refer to the section Interaction with Stormshield Network routing.

In the administration interface, filtering rules are necessary to allow BIRD traffic.



	Status	Action	Source	Destination	Dest. port	Protocol	Security inspection	Comment
1	● on	pass	GRP_BIRD	Firewall_birdy_bridge	Any	ospf	IPS	OSPF allowed
2	● on	pass	GRP_BIRD	Firewall_birdy_bridge	router		IPS	RIP allowed
3	● on	pass	GRP_BIRD	Firewall_birdy_bridge	bgp		IPS	BGP allowed

Example of advanced configuration

From version 1.0 of the firmware onwards and on versions supporting IPv6, dynamic routing takes place in two files, depending on the IP version of the networks concerned:

- **/usr/Firewall/ConfigFiles/Bird/bird.conf** for IPv4 networks and routes.
- **/usr/Firewall/ConfigFiles/Bird/bird6.conf** for IPv6 networks and routes.

Two separate operations are required for starting BIRD.

The BIRD module must first be defined as active in the following file:

/usr/Firewall/ConfigFiles/Bird/global. This can be done by setting the "state" variable to "1" in the section [bird] for IPv4 routing, and/or in the section [bird6] for IPv6 routing.

```
[bird]
state=1

[bird6]
state=1
```

This operation ensures that dynamic routing stays enabled when the firewall is restarted.

Next, to start BIRD or to reload its configuration after a modification, use the command "enbird". If the configuration contains syntax errors, the command will indicate them and will not enable the configuration.

```
V50XXA0D0000073>enbird
V50XXA0D0000073>
```

birdc/birdc6: remote control

BIRD and BIRD6 have an interactive mode: ***birdc*** for BIRD Client, and ***birdc6*** for BIRD6 Client. Launch this mode by calling **birdc** or **birdc6**, depending on the IP version of dynamic routing that you wish to control.

```
V50XXA0D0000073>birdc
BIRD 1.6.0 ready.
bird>
```

```
V50XXA0D0000073>birdc6
BIRD 1.6.0 ready.
bird>
```

NOTE
Further on in this document, all examples will be shown for the ***birdc*** interactive mode. They can all be transposed for the ***birdc6*** interactive mode.

The BIRD interactive mode does not allow modifying the configuration file, but allows viewing BIRD states, testing the proper operation of a new configuration by enabling backtracking, and creating a temporary configuration.

“Show” commands

The character “?” allows you to display the list of available options.

```
bird> show ?
show interfaces                            Show network interfaces
show memory                                Show memory usage
show ospf ...                              Show information about OSPF protocol
show protocols [<protocol> | "<pattern>"]  Show routing protocols
show roa ...                               Show ROA table
show route ...                             Show routing table
show static [<name>]                       Show details of static protocol
show status                                Show router status
show symbols ...                           Show all known symbolic names
bird>
```

Example: Show all routes.

```
bird> show route
0.0.0.0/0  via 192.168.97.1 on em0 [static1 21:11] * (200)
100.100.100.100/32 via 192.168.97.101 on em0 [static1 21:11] * (200)
           via 192.168.97.101 on em0 [router1 20:50 from 100.100.100.100]
           (100/?) [AS65001?]
1.1.1.0/24 via 192.168.97.1 on em0 [router2 21:08 from 192.168.97.102] * (100/?)
           [?]
1.1.3.0/24 via 192.168.97.1 on em0 [router2 21:08 from 192.168.97.102] * (100/?)
           [?]
2.2.2.0/24 via 192.168.97.101 on em0 [router1 20:50 from 100.100.100.100] *
           (100/?) [AS65001?]
2.2.4.0/24 via 192.168.97.101 on em0 [router1 20:50 from 100.100.100.100] *
           (100/?) [AS65001?]
bird>
```

Example: Show all routes by protocol instance. In this case, the instance is **router2**.

```
bird> show route protocol router2
```

```
1.1.1.0/24 via 192.168.97.1 on em0 [router2 14:14 from 192.168.97.102] * (100/?)
[?]
1.1.3.0/24 via 192.168.97.1 on em0 [router2 14:14 from 192.168.97.102] * (100/?)
[?]
bird>
```

In **birdc**, most of the commands are common to all the protocols. Therefore for example, routes announced to a neighboring BCP will be viewed by a command that calls on the export filter.

```
bird> show route export router1
172.16.0.0/24 blackhole [static1 13:20] * (200)
bird>
```

Debug

Show commands provide a lot of information on instances. They allow diagnosing problems, whether they are due to a faulty configuration, a network issue or any other problem.

```
bird> show protocols all router1
[……]
BGP state: Active
          Neighbor address: 100.100.100.100
          Neighbor AS: 65001
          Start delay: 2/5
          Last error: Socket: Connection closed
bird>
```

To enable the reception of system messages on the console, enter the command *echo all* then *echo off* to stop these logs.

```
bird> echo all
bird> >>> KRT: Error sending route 0.0.0.0/0 to kernel: No such process
>>> KRT: Error sending route 100.100.100.100/32 to kernel: No such process
>>> Next hop address 100.100.100.100 resolvable through recursive route for
100.100.100.100/32
>>> KRT: Error sending route 1.1.1.0/24 to kernel: No such process
```

Debug events are viewed globally or for example by protocol instance. The use of *debug* commands is a useful tool that effectively completes the commands for viewing states.

```
bird> debug router2 all
bird> echo all
>>> router2 < added 0.0.0.0/0 via 192.168.97.1 on em0
>>> router2 < replaced 100.100.100.100/32 via 192.168.97.101 on em0
>>> router2 > updated 1.1.1.0/24 via 192.168.97.1 on em0
>>> router2 < rejected by protocol 1.1.1.0/24 via 192.168.97.1 on em0
>>> router2 > updated [best] 1.1.1.0/24 via 192.168.97.1 on em0
>>> router2 < replaced 2.2.2.0/24 via 192.168.97.101 on em0
>>> router2 < replaced 2.2.4.0/24 via 192.168.97.101 on em0
```

Temporary testing of a new configuration

We would like to test a new configuration **bird_to_be_tested.conf**. To do so, launch BIRD using a **bird.conf** configuration whose operation has been validated, then launch **birdc**.

To check the syntax of the file without applying it:

```
bird> configure check "bird_a_tester.conf"
```

Next, temporarily apply this configuration for 60 seconds using the command:

```
bird> configure "bird_a_tester.conf" timeout 60
```

The new configuration will be applied. If the firewall can no longer be contacted or there is no confirmation from the administrator, the previous configuration will be automatically reapplied after 60 seconds.

If the new configuration is considered valid, it can be confirmed using:

```
bird> configure confirm
```

If the new configuration has not been validated and the firewall can still be contacted, it is possible to immediately backtrack using:

```
bird> configure undo
```

Configuration

The configuration of at least the following lines is required in order to define a basic environment for cooperating with the system.

```
protocol   kernel {
           persist;                # Don't remove routes on BIRD shutdown
           scan time 20;           # Scan kernel routing table every 20 seconds
           export all;             # Default is export none
           learn;                  # Learn all alien routes from the kernel
           preference 254;         # Protect kernel routes with a high
                                   preference
}
protocol   device {
           scan time 10;           # Scan interfaces every 10 seconds
}
```

We will not go into detail here about each configuration line. If you wish to obtain comprehensive explanations, please refer to the online documentation on BIRD at:

http://bird.network.cz/?get_doc&f=bird.html.

The most important concepts are of the protocol instance and filtering.

A protocol instance may either be BGP, RIP or OSPF and defines an appropriate configuration. Several instances may be defined, even for the same protocol.

Each protocol instance is connected to an internal routing table in BIRD. This connection is monitored by two filters that can accept, reject or modify routes.

The export filter monitors routes sent from the internal routing table to BIRD towards the protocol, and the import filter does the same in the opposite direction.

Ensure that you implement an accurate route filter. The use of full import or export routes (for example, import all) between protocol instances may produce destructive results.

Syntax rules

- Text written after the # is a comment
- Text framed by /* and */ is a comment
- Blocks of several options are placed in curly brackets {}
- Each option ends with a semi-colon ;
- Configuration is case-sensitive.

The following configuration contains two syntax errors.

```
router id 192.168.97.219;

protocol kernel
{
persist;                          # Don't remove routes on bird shutdown
```

```
6  scan time 20;                   # Scan kernel routing table every 20 seconds
7  export all;                     # Default is export none
8  }
9
10 protocol device
11 {
12 scan time 10;                   # Scan interfaces every 10 seconds
13
14 protocol static
15 {
16 route 0.0.0.0/0 via 10.200.0.1;
17 route 172.16.0.0/24 drop
18 }
```

Error message 1

```
V50XXA0D0000073>enbird
bird: /usr/Firewall/ConfigFiles/Bird/bird.conf, line 14: syntax error
```

If a closing curly bracket is omitted, the error will mention the first line of the next block, a line that does not correspond to an authorized command of the unclosed block.

The character "**}**" therefore has to be inserted in line 13.

Error message 2

```
V50XXA0D0000073> enbird
bird: /usr/Firewall/ConfigFiles/Bird/bird.conf, line 18: syntax error
```

The character "**;**"therefore has to be inserted at the end of line 17 in the example.

Interaction with Stormshield Network routing

Thanks to the default configuration on Stormshield Network firewalls, firewall's routing has priority over dynamic routing (maximum preference of 254).

WARNING: known restriction
During the reconfiguration of firewall's routes, they will be temporarily erased and BIRD/BIRD6 can then configure its own routes. Firewall's routing therefore has to be protected using an export filter on the pseudo-protocol kernel.

This is an example of a filter that will protect the default route and the static route 1.2.3.0/24:

```
filter protect_Stormshield_routes
{
 if (net = 0.0.0.0/0) || (net = 1.2.3.0/24)
 then reject;
 else accept;
}

protocol kernel
{
 (...)
 export filter protect_Stormshield_routes;
}
```

Giving dynamic routing priority over Stormshield Network routing

If dynamic routing should have priority over Stormshield Network routing, the routes obtained by dynamic routing (BGP, OSPF or RIP protocol) must have a preference value higher than the routes obtained by the system (pseudo-protocol kernel).

The preference value of kernel must therefore be reduced, for example to 1:

```
protocol kernel
{
 (...)
 preference 1;
}
```

Routing the interfaces of firewall

If the firewall's interfaces have been configured with different sub-networks, and you wish to send the sub-networks of the interfaces via BIRD, the pseudo-protocol *direct* will be used.

By default, all interfaces are taken into account. The number of interfaces taken into account can be restricted using the attribute *interface*.

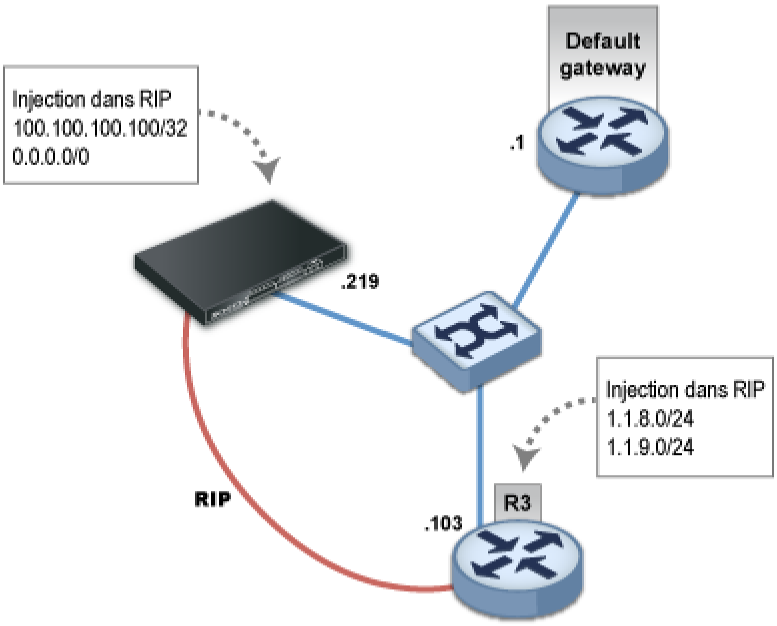
```
protocol direct
{
interface "-vlan*", "*"; # Exclude VLANs
}
```

Simple configurations

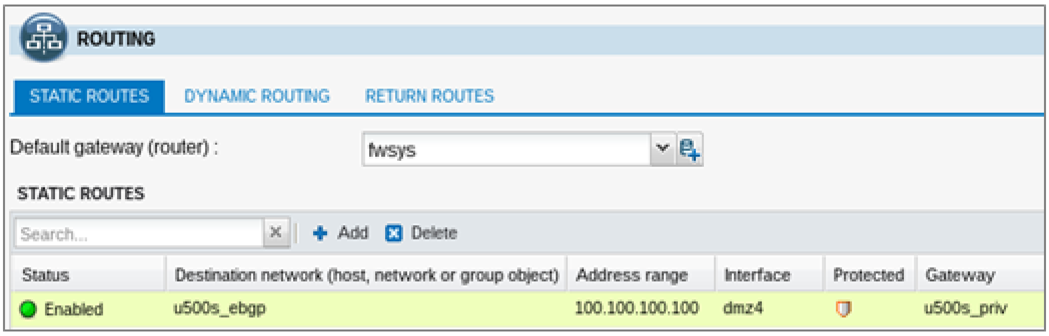
RIP

The version supported is RIPv2.

Below is the configuration “RIP_simple”.



We will configure a default route and a static route to 100.100.100.100/32:


ROUTING

STATIC ROUTES | DYNAMIC ROUTING | RETURN ROUTES

Default gateway (router) : fwsys

STATIC ROUTES

Search... ✕ | + Add | ☒ Delete

Status	Destination network (host, network or group object)	Address range	Interface	Protected	Gateway
● Enabled	u500s_ebgp	100.100.100.100	dmz4	●	u500s_priv

RIPv2 is configured by specifying “multicast” as the RIP mode associated with the interface “em0”. Requests by neighbors to send tables will be honored.

```
router id 192.168.97.219;
protocol kernel {
  persist;                          # Don't remove routes on bird shutdown
  scan time 20;                     # Scan kernel routing table every 20 seconds
  export all;                       # Default is export none
  learn;                            # Learn all alien routes from the kernel
  preference 254;                   # Protect kernel routes with a high
                                    preference
}
```

```
protocol device {
  scan time 10;                    # Scan interfaces every 10 seconds
}
filter ripexport {
  if (net = 0.0.0.0/0) || (net = 100.100.100.100/32)
  then accept;
  else reject;
}
protocol rip MyRIP {
  debug all;
  interface "em0" { mode multicast; };
          mode multicast;
          authentication none;
          };
  honor always;
  authentication none;
  import all;
  export filter ripexport;
}
```

The state of the protocol instance:

```
bird> show protocols all MyRIP
name                 proto     table      state      since     info
MyRIP                RIP       master     up         10:08:56
Preference: 120
Input filter: ACCEPT
Output filter: ripexport
Routes: 4 imported, 5 exported, 3 preferred
Route change stats:  received  rejected   filtered   ignored  accepted
Import updates:      147       0          0          140      7
Import withdraws:    0         0          ---        0        0
Export updates:      11        0          3          ---      8
Export withdraws:    0         ---        ---        ---      0
bird>
```

Learned routes:

```
bird> show route primary protocol MyRIP
192.168.97.0/24  via 10.200.45.250 on eth0 [MyRIP 10:29:19] ! (120/2)
1.1.9.0/24       via 10.200.45.250 on eth0 [MyRIP 10:29:19] * (120/2)
1.1.8.0/24       via 10.200.45.250 on eth0 [MyRIP 10:29:19] * (120/2)
```

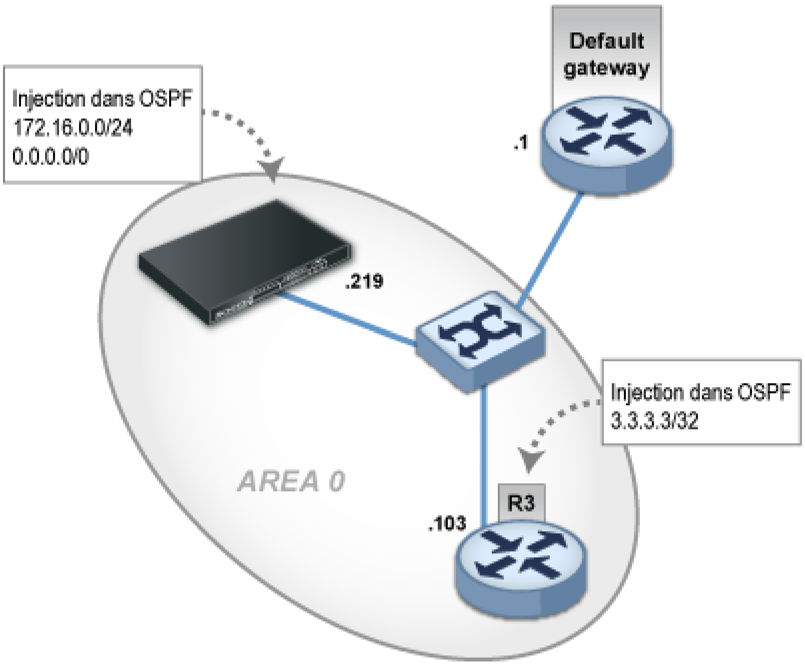
Below are the routes received by the neighbor. Note that the default route has been received. Ordinarily, other routers on the market would reject the export of this route. However, in this case, it has to be filtered explicitly.

```
bird> show route primary protocol MyRIP
0.0.0.0/0 via 192.168.97.219 on eth0 [MyRIP 10:36] * (120/2)
100.100.100.100/32 via 192.168.97.101 on eth0 [MyRIP 10:36 from 192.168.97.219] *
(120/2)
```

OSPF

The versions supported are OSPFv2 for IPv4 and OSPFv3 for IPv6.

Below is the configuration “OSPF simple”.



It consists of deploying an area 0 on a LAN where a possible neighbor is explicitly indicated. All routes are imported from OSPF and redistributed in OSPF. The route of the sub-network directly linked to the interface em3 (172.16.0.0/24) and the default route are redistributed in OSPF.

```
router id 192.168.97.219;
protocol kernel {
  persist;                      # Don't remove routes on bird shutdown
  scan time 20;                 # Scan kernel routing table every 20 seconds
  export all;                   # Default is export none
  learn;                        # Learn all alien routes from the kernel
  preference 254;               # Protect kernel routes with a high preference
}
protocol device {
  scan time 10;                 # Scan interfaces every 10 seconds
}
protocol direct {
  interface "em3";
}
filter ospfexport {
  if (source = RTS_DEVICE) || (net = 0.0.0.0/0)
  then accept;
  else reject;
}
protocol ospf MyOSPF {
  export filter ospfexport;
  import all;
  area 0.0.0.0 {
  stub no;
```

```
          interface "em4" {
            type broadcast;
            neighbors {
            192.168.97.103 eligible;
            };
          };
  };
}
```

NOTE
You are also advised to set the parameter "`priority 0`" in the *interface* section of the OSPF node configuration in order to disable the firewall's role in elections for Designated Router / Backup Designated Router roles.

The following command indicates that the neighborhood has been established (indicated by the state “full”).

The neighbor has been declared a “Designated Router” (indicated by the state “dr”)

```
bird> show ospf neighbors
MyOSPF:
Router ID      Pri     State    DTime  Interface   Router IP
192.168.97.103 1       full/dr  00:34  em4         192.168.97.103
bird>
```

Routes received:

```
bird> show route protocol MyOSPF
3.3.3.3/32 via 192.168.97.103 on em4 [MyOSPF 16:17:38] * E2 (150/10/10000)
[192.168.97.103]
192.168.97.0/24 dev em4 [MyOSPF 16:15:43] * I (150/10) [192.168.97.219]
bird>
```

The OSPF topology can be displayed:

```
bird> show ospf topology
area 0.0.0.0
        router 192.168.97.103
                 distance 10
                 network 192.168.97.0/24 metric 10
        router 192.168.97.219
                 distance 0
                 network 192.168.97.0/24 metric 10
        network 192.168.97.0/24
                 dr 192.168.97.103
                 distance 10
                 router 192.168.97.103
                 router 192.168.97.219
bird>
```

As well as the LSA database:

```
bird> show ospf lsadb
Global
Type   LS ID           Router          Age    Sequence      Checksum
0005   3.3.3.3         192.168.97.103  501    8000000a      ec8a
0005   172.16.0.255    192.168.97.219  1150   80000001      81b6
0005   0.0.0.0         192.168.97.219  1150   80000001      37f1
Area    0.0.0.0
Type   LS ID           Router          Age    Sequence      Checksum
0001   192.168.97.103  192.168.97.103  455    8000000a      2254
0002   192.168.97.103  192.168.97.103  456    80000006      9384
0001   192.168.97.219  192.168.97.219  1144   8000041b      0bf8
bird>
```

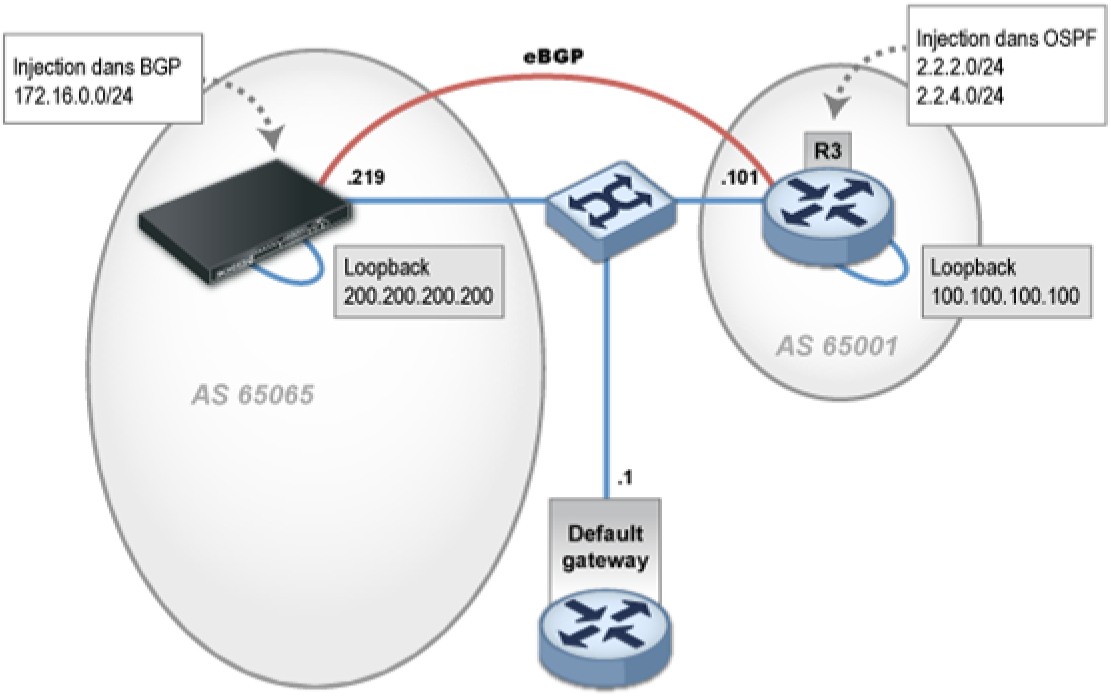
NOTE
Note that the type of LSA is shown on the left whereas it is generally used as a horizontal delimiter by the usual display standards.

BGP

"BGP_simple" configuration

The version supported is BGPv4 for IPv4 and IPv6.

Below is the configuration "BGP_simple".



The configuration "BGP_simple" is implemented as follows:

```
router id 192.168.97.219;
```

```
protocol kernel {
         persist;                # Don't remove routes on bird shutdown
         scan time 20;           # Scan kernel routing table every 20 seconds
         export all;             # Default is export none
         learn;                  # Learn all alien routes from the kernel
         preference 254;         # Protect kernel routes with a high preference
}
protocol device {
         scan time 10;           # Scan interfaces every 10 seconds
}
protocol direct {
         interface "em3";
}
protocol bgp router1 {
         description "My 1st BGP uplink";
         local as 65065;
         neighbor 100.100.100.100 as 65001;
         multihop 5;
         hold time 180;
         keepalive time 60;
         export where source = RTS_DEVICE;
         import all;
         default bgp_local_pref 100;
         source address 200.200.200.200;
}
```

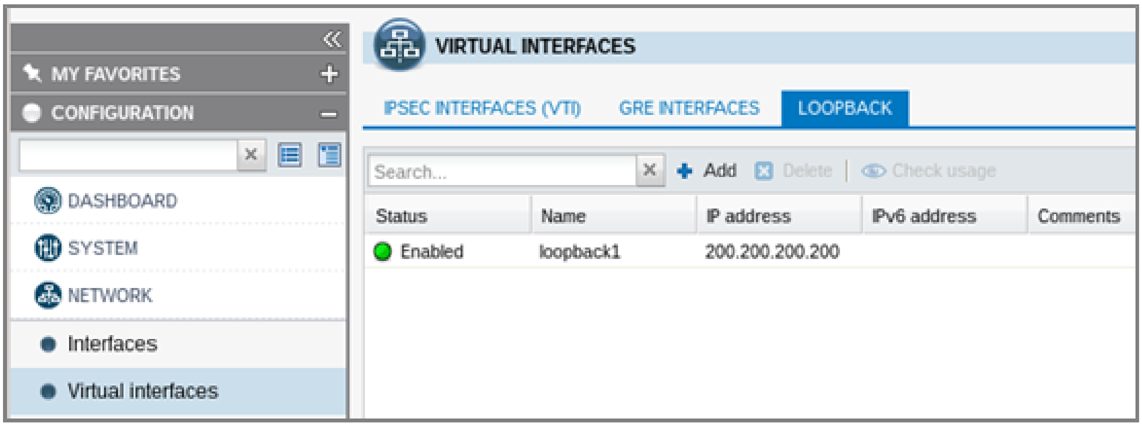
Explanations

Unlike most routers on the market, the local AS has to be specified for each BGP instance.

According to the best practices, this eBGP session is set up between *loopback* interfaces and not physical interfaces. It is therefore necessary to configure the IP of the local *loopback* in question (200.200.200.200/32), to specify this address as the source and a static route to the neighbor's *loopback*.

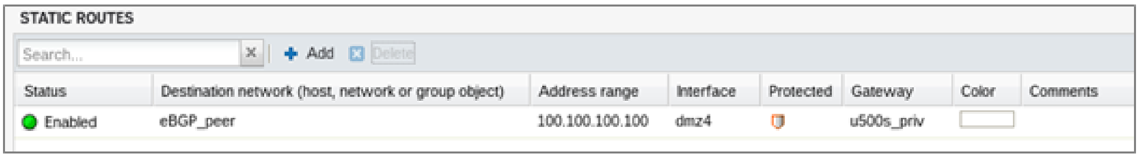
Loopback virtual interfaces

In version 2, the web administration interface allows configuring loopback interfaces via the **Virtual interfaces** module, in the *Loopback* tab.



Status	Name	IP address	IPv6 address	Comments
● Enabled	loopback1	200.200.200.200		

It is advisable to declare the static route to the remote loopback on the firewall outside the Bird configuration via the **Routing** module in the *Static routes* tab, in order to prevent BGP traffic from being blocked by "IP address spoofing" alarms.



STATIC ROUTES

Status	Destination network (host, network or group object)	Address range	Interface	Protected	Gateway	Color	Comments
● Enabled	eBGP_peer	100.100.100.100	dmz4	▨	u500s_priv	□	

Once again, we will select only the sub-network 172.16.0.0/24 directly linked to the interface em3 as the route to announce to our neighbors.

Here, an anonymous export filter has been defined directly in the “export” instruction, with the keyword "where". This export filter selects routes with RTS_DEVICE as their source, meaning routes obtained by the “direct” pseudo-protocol.

The value of the *hold-time* has been specified as 180s, the usual market value. BIRD implements by default 240s. The value of the *keepalive* duration does not need to be specified (calculated at 1/3 of the *hold-time*) but we are mentioning it explicitly for clarity. Likewise for the default *local-preference*.

The command “show protocols” below confirms that the session is indeed working.

```
bird> show protocols router1
name      proto   table   state  since   info
router1   BGP     master  up     12:47   Established
bird>
```

The neighbor has received the routes:

```
bird> show route protocol router1
100.100.100.100/32  via 192.168.97.101 on em0 [router1 13:09 from 100.100.100.100]
(100/?)  [AS65001?]
2.2.2.0/24          via 192.168.97.101 on em0 [router1 13:09 from 100.100.100.100]
*(100/?)[AS65001?]
2.2.4.0/24          via 192.168.97.101 on em0 [router1 13:09 from 100.100.100.100]
*(100/?)[AS65001?]
bird>
```

On the BGP neighbor, the route announced and released by the filter is received. As for the route 1.1.1.1/32, it has been blocked.

```
@router1:~$ show ip bgp
BGP table version is 0, local router ID is 2.2.2.2
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
                          r RIB-failure,   S Stale,       R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete

Network                   Next Hop         Metric LocPrf Weight          Path
*> 2.2.2.0/24             0.0.0.0          1             32768           ?
*> 2.2.4.0/24             0.0.0.0          1             32768           ?
*> 100.100.100.100/32     0.0.0.0          1             32768           ?
*> 172.16.0.0/24          200.200.200.200                0               65065 i
*> 192.168.97.0           0.0.0.0          1             32768           ?
Total number of prefixes 5
@router1:~$
```

Authentication

TCP-MD5 authentication can be set up between BGP routers in BIRD.

This method allows protecting BGP sessions by authenticating frames in the TCP header in compliance with RFC2385.

This is represented by the addition of a *"password"* directive*"password"* and *"setkey no"* directives in the configuration of the BGP router in the files **/usr/Firewall/ConfigFiles/Bird/bird.conf** (dynamic routing of IPv4 packets) and **/usr/Firewall/ConfigFiles/Bird/bird6.conf** (dynamic routing of IPv6 packets). For example:

```
protocol bgp
{
  local as 65065;
  neighbor 100.100.100.100 as 65001;
  export where source = RTS_DEVICE;
  import all;
  source address 200.200.200.200 ;
  password "very_secret";
  setkey no;
}
```

Next, this SA (Security Association) needs to be declared in the file **ConfigFiles/Bird/global** in the section [BGPAuth]. This section groups IPv4 and IPv6 security associations.

Each SA is described in it in the form of a line with the following structure (fields separated by commas without spacing):

```
[BGPAuth]
IP_local_router,IP_remote_router1,password_1
IP_local_router,IP_remote_router2,password_2
```

In our example:

```
[BGPAuth]
100.100.100.100,200.200.200.200,very_secret
```

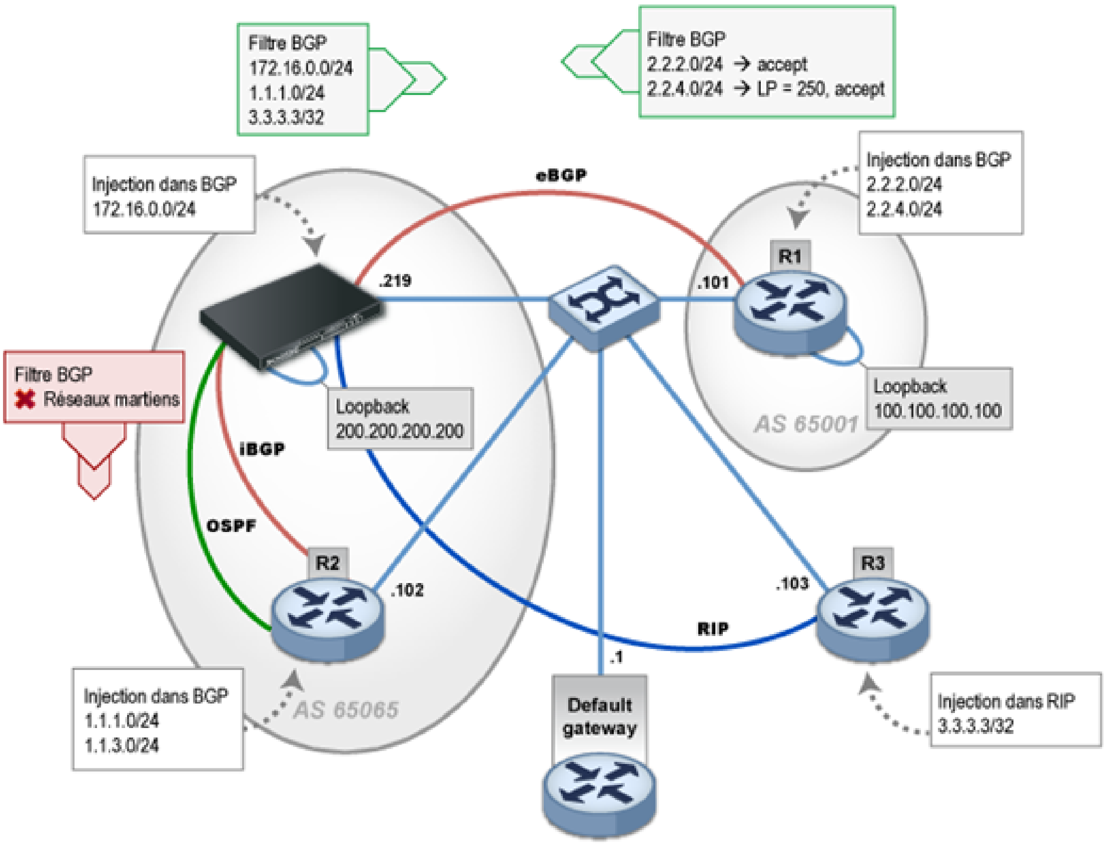
NOTE
Passwords must not contain spaces or the equal sign ('=').

The command `showSAD` allows viewing BGP SAs (as well as IPSec SAs). The field "**tcp mode=any**" indicates an SA relating to BGP authentication, while the field "**A: tcp-md5**" indicates the authentication type used.

```
showSAD
200.200.200.200 100.100.100.100
        tcp mode=any spi=4096(0x00001000) reqid=0(0x00000000)
        A: tcp-md5 32383738 35
        seq=0x00000000 replay=0 flags=0x00000040 state=mature
        created: Apr 22 12:59:49 2014 current: Apr 22 12:59:53 2014
        diff: 4(s) hard: 0(s) soft: 0(s)
        last: Apr 22 12:59:52 2014 hard: 0(s) soft: 0(s)
        current: 0(bytes) hard: 0(bytes) soft: 0(bytes)
        allocated: 1 hard: 0 soft: 0
        sadb_seq=1 pid=6330 refcnt=1
100.100.100.100 200.200.200.200
        tcp mode=any spi=4096(0x00001000) reqid=0(0x00000000)
        A: tcp-md5 32383738 35
        seq=0x00000000 replay=0 flags=0x00000040 state=mature
        created: Apr 22 12:59:49 2014 current: Apr 22 12:59:53 2014
        diff: 4(s) hard: 0(s) soft: 0(s)
        last: hard: 0(s) soft: 0(s)
        current: 0(bytes) hard: 0(bytes) soft: 0(bytes)
        allocated: 0 hard: 0 soft: 0
        sadb_seq=0 pid=6330 refcnt=1
```

Advanced configuration

The advanced configuration is implemented here. This configuration combines three simple configurations and also includes an iBGP link set parallel to the OSPF link.



The client's network consists of the router R2, R3 and Stormshield Network firewall. The router R1 is an external BGP neighbor. This network represents a realistic architecture model, except for the fact that all the routers are physically connected by a single LAN.

We will implement a standard filter policy for:

- announcing only public BGP networks to outside the network,
- not spreading internal or Martian networks in the internal BGP,
- tagging one of the learned routes en eBGP with a local-preference of 250. This measure is generally implemented to monitor load balancing between several eBGP neighbors,
- announcing only one default route in OSPF,
- announcing only one default route in RIP.

The networks announced by R2 and R3 are announced respectively via BGP and RIP. The use of OSPF to announce the default route is only for learning purposes.

BIRD configuration

Below is the equivalent configuration file in BIRD.

```
router id 192.168.97.219;

function is_locormartians()
        prefix set martians;
        {
        martians = [ 169.254.0.0/16+, 172.16.0.0/12+, 192.168.0.0/16+,
10.0.0.0/8+, 224.0.0.0/4+, 240.0.0.0/4+ ];
        # default
        if net.ip = 0.0.0.0 then return true;
        # LIR not authorized
        if (net.len < 8) || (net.len > 24) then return true;
        # martians
        if net ~ martians then return true;
        # local
        if net = 100.100.100.100/32 then return true;
        return false;
}

filter out_eBGP {
        if net ~ [ 172.16.0.0/24, 3.3.3.3/32, 1.1.1.0/24 ]
        then accept;
        else reject;
}

filter out_iBGP {
        if ( is_locormartians() )
        then reject;
        else accept;
}

filter lp_tag_in {
        if net = 2.2.4.0/24 then {
        bgp_local_pref = 250;
        accept;
        } else accept;
}

filter default_ok {
        if net = 0.0.0.0/0 then {
        accept;
        } else reject;
}

protocol kernel {
        persist;                 # Don't remove routes on bird shutdown
        scan time 20;            # Scan kernel routing table every 20 seconds
        export all;              # Default is export none
        learn;                   # Learn all alien routes from the kernel
        preference 254;          # Protect kernel routes with a high preference
        }

protocol device {
        scan time 10;            # Scan interfaces every 10 seconds
}

protocol direct {
        interface "em3";
}

protocol rip MyRIP {             # You can also use an explicit name
        debug all;
        interface "em4" { mode multicast; };
                                 mode multicast;
                                 authentication none;
                                 };
        honor always;
        authentication none;
        import all;
        export filter default_ok;
}

protocol ospf MyOSPF {
        export filter default_ok;
        import all;
        area 0.0.0.0 {
        stub no;
        interface "em4" {
        type broadcast;
          };
        };
}
protocol bgp router1 {
        debug all;
        description "My 1st BGP uplink";
        local as 65065;
        neighbor 100.100.100.100 as 65001;
        multihop 5;
        hold time 180;
        keepalive time 60;
        export filter out_eBGP;
        import filter lp_tag_in;
        source address 200.200.200.200;
}
protocol bgp router2 {
        description "My local BGP neighbor";
        local as 65065;
        neighbor 192.168.97.102 as 65065;
        keepalive time 60;
        next hop self;
        export filter out_iBGP;
        import all;
}
```

NOTE

You are also advised to set the parameter "`priority 0`" in the *interface* section of the OSPF node configuration in order to disable the firewall's role in elections for Designated Router / Backup Designated Router roles.

Stormshield Network Firewall's routing table

```
bird> show route
0.0.0.0/0           via 192.168.97.1 on em4 [kernel1 14:37:15] * (254)
100.100.100.100/32  via 192.168.97.101 on em4 [kernel1 14:37:15] * (254)
3.3.3.3/32          via 192.168.97.103 on em4 [MyRIP 14:37:06] * (120/2)
192.168.97.0/24     dev em4 [MyOSPF 14:01:33] * I (150/10) [192.168.97.102]
                    via 192.168.97.102 on em4 [router2 14:01:17] (100/10) [i]
1.1.1.0/24          via 192.168.97.102 on em4 [MyOSPF 14:01:36] * E2
                    (150/10/10000) [192.168.97.102]
                    via 192.168.97.102 on em4 [router2 14:01:17] (100/10) [i]
1.1.3.0/24          via 192.168.97.102 on em4 [MyOSPF 14:01:36] * E2
                    (150/10/10000) [192.168.97.102]
                    via 192.168.97.102 on em4 [router2 14:01:17] (100/10) [i]
2.2.2.0/24          via 192.168.97.101 on em4 [router1 13:54:12 from
                    100.100.100.100] * (100/?) [AS65001i]
2.2.4.0/24          via 192.168.97.101 on em4 [router1 14:01:17 from
                    100.100.100.100] * (100/?) [AS65001i]
172.16.0.254/32     dev lo0 [kernel1 14:37:15] * (254)
192.168.97.219/32   dev lo0 [kernel1 14:37:15] * (254)
172.16.0.0/24       dev em3 [direct1 13:54:11] * (240)
10.200.45.254/32    dev lo0 [kernel1 14:37:15] * (254)
```

In order to check the local-preference on the router 2.2.4.0/24, the details of the routes for the instance of the protocol router1 will be displayed.

```
bird> show route protocol router1 all
2.2.2.0/24 via 192.168.97.101 on em4 [router1 13:54:12 from 100.100.100.100] *
(100/?) [AS65001i]
 Type: BGP unicast univ
 BGP.origin: IGP
 BGP.as_path: 65001
 BGP.next_hop: 100.100.100.100
 BGP.local_pref: 100
2.2.4.0/24 via 192.168.97.101 on em4 [router1 14:01:17 from 100.100.100.100] *
(100/?) [AS65001i]
 Type: BGP unicast univ
 BGP.origin: IGP
 BGP.as_path: 65001
 BGP.next_hop: 100.100.100.100
 BGP.local_pref: 250
```

Router R3 – show IP route

You will notice here that the default route is also announced.

```
@router3:~$ show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
 I - ISIS, B - BGP, > - selected route, * - FIB route
R>* 0.0.0.0/0 [120/2] via 192.168.97.1, eth0, 00:06:15
C>* 1.1.8.0/24 is directly connected, lo
C>* 1.1.9.0/24 is directly connected, lo
S>* 3.3.3.3/32 [1/0] is directly connected, Null0, bh
C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.97.0/24 is directly connected, eth0
@router3:~$
```

In the event this traffic has to be routed symmetrically - for example when there is NAT – the BIRD configuration has to be adapted in order to announce the firewall as the next hop. The changes can be made in the filter "default_ok" which is used for announcing the default route to R3 via RIP and to R2 via OSPF:

```
filter default_ok
{
  if net = 0.0.0.0/0 then
  {
  dest = RTD_UNREACHABLE;  # annonce le firewall comme next-hop pour cette route
  accept;
  }
}
```

To impose a gateway other than the firewall itself, you will need to use the directive:

```
gw = <ip>;
```

Router R2 – show IP route

```
@router2:~$ show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
 I - ISIS, B - BGP, > - selected route, * - FIB route
O>* 0.0.0.0/0 [110/10000] via 192.168.97.1, eth0, 22:26:17
C>* 1.1.1.0/24 is directly connected, lo
C>* 1.1.3.0/24 is directly connected, lo
B>* 2.2.2.0/24 [200/1] via 100.100.100.100 (recursive via 192.168.97.1), 00:02:04
B>* 2.2.4.0/24 [200/1] via 100.100.100.100 (recursive via 192.168.97.1), 00:02:04
C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.97.0/24 is directly connected, eth0
@router2:~$
```

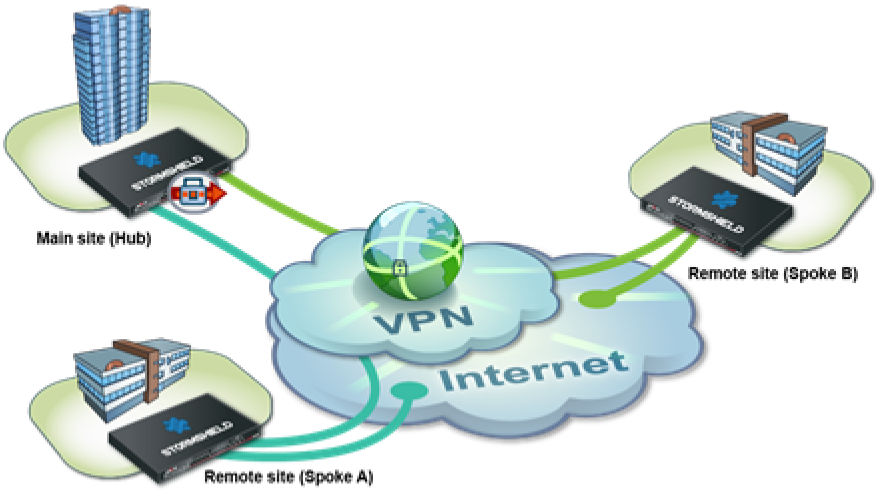
Router R1 – show IP route

```
@router1:~$ show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
I - ISIS, B - BGP, > - selected route, * - FIB route
S>* 0.0.0.0/0 [1/0] via 192.168.97.1, eth0
B>* 1.1.1.0/24 [20/0] via 200.200.200.200 (recursive via 192.168.97.219),
00:00:29
C>* 2.2.2.0/24 is directly connected, lo
C>* 2.2.4.0/24 is directly connected, lo
B>* 3.3.3.3/32 [20/0] via 200.200.200.200 (recursive via 192.168.97.219),
00:00:08
C>* 100.100.100.100/32 is directly connected, lo
C>* 127.0.0.0/8 is directly connected, lo
B>* 172.16.0.0/24 [20/0] via 200.200.200.200 (recursive via 192.168.97.219),
00:00:29
C>* 192.168.97.0/24 is directly connected, eth0
S>* 200.200.200.200/32 [1/0] via 192.168.97.219, eth0
@router1:~$
```

Appendix A: Hub & Spoke VPN tunnels routed via BGP

Below is an example of BGP dynamic routing in the context of a VPN network in a Hub and Spoke star configuration.

Tunnels configuration



For the configuration of a Hub & Spoke IPSec policy, please refer to the *HOW TO* mentioned below. In our case, the differences from this procedure consist of configuring traffic endpoints through virtual interfaces, instead of remote networks in the IPSec policy (see following paragraph).

Go to http://documentation.stormshield.eu/. Refer to *HOW TO: IPSec VPN - Hub and Spoke Configuration*, and refer to case no. 1: *internal traffic via IPSec tunnels*.

Main site

TunnelA

Local network:	Ipsec1 interface (172.16.0.1)
Peer:	Site_SpokeA
Remote network:	Remote_tunnelA (172.16.0.2)

TunnelB

Local network:	Ipsec2 interface (172.16.0.5)
Peer:	Site_SpokeB
Remote network:	Remote_tunnelB (172.16.0.6)

Spoke A

Local network:	Ipsec1 interface (172.16.0.2)
Peer:	Site_FW_Hub
Remote network:	Remote_tunnelA (172.16.0.1)

Spoke B

Local network:	Ipsec1 interface (172.16.0.6)
Peer:	Site_FW_Hub
Remote network:	Remote_tunnelA (172.16.0.5)

BGP configuration of the main site (Hub)

```
protocol direct {
}

protocol kernel {
        learn;                  # Learn all alien routes from the kernel
        persist;                # Don't remove routes on bird shutdown
        scan time 20;           # Scan kernel routing table every 20 seconds
        import all;             # Default is import all
        export all;             # Default is export none
        preference 254;         # Protect existing routes
}

# This pseudo-protocol watches all interface up/down events.
protocol device {
        scan time 10;           # Scan interfaces every 10 seconds
}

filter f_import {
        if source = RTS_BGP then
        accept;
        else
        reject;
}

filter f_export {
        # local shared networks and BGP routes
        if( (net = 192.168.0.0/24) || (source = RTS_BGP) ) then
        accept;
        else
        reject;
}

router id <ip_pub_hub>;

template bgp star {
        local as 65000;
        import filter f_import;
        export filter f_export;
        hold time 5;
        multihop;
        rr client;
        next hop self;
}

protocol bgp router_spokeA from star {
        neighbor 172.16.0.2 as 65000;
```

```
         source address 172.16.0.1;
}

protocol bgp router_spokeB from star {
         neighbor 172.16.0.6 as 65000;
         source address 172.16.0.5;
}
```

BGP configuration of the Spoke A satellite site

```
protocol direct {
}

protocol kernel {
          learn;                # Learn all alien routes from the kernel
          persist;              # Don't remove routes on bird shutdown
          scan time 20;         # Scan kernel routing table every 20 seconds
          import all;           # Default is import all
          export all;           # Default is export none
          preference 254;       # Protect existing routes
}

protocol device {
          scan time 10;         # Scan interfaces every 10 seconds
}

filter filter_export_net {
          if(net = 192.168.1.0/24) then {
          accept;
          }
          else reject;
}

router id <ip_pub_spokeA>;

  protocol bgp router_tunnel1 {
          local as 65000;
          neighbor 172.16.0.1 as 65000;
          hold time 5;
          multihop;
          import all;
          export filter filter_export_net;
          source address 172.16.0.2;
}
```

BGP configuration of the Spoke B satellite site

```
protocol direct {
}
```

```
protocol kernel {
          learn;                # Learn all alien routes from the kernel
          persist;              # Don't remove routes on bird shutdown
          scan time 20;         # Scan kernel routing table every 20 seconds
          import all;           # Default is import all
          export all;           # Default is export none
          preference 254;       # Protect existing routes
}

protocol device {
          scan time 10;         # Scan interfaces every 10 seconds
}

filter    filter_export_net {
          if(net = 192.168.2.0/24) then {
          accept;
          }
          else reject;
}
router id <ip_pub_spokeB>;

protocol bgp router_tunnel2 {
          local as 65000;
          neighbor 172.16.0.5 as 65000;
          hold time 5;
          multihop;
          import all;
          export filter filter_export_net;
          source address 172.16.0.6;
}
```

Verification of routing tables

Routing table on the main site (Hub):

```
bird> show route
0.0.0.0/0        via 10.60.0.254 on em0 [kernel1 10:16] * (254)
10.60.3.127/32   dev lo0 [kernel1 10:16] * (254)
192.168.0.0/24   dev em1 [direct1 10:16] * (240)
192.168.1.0/24   dev em2 [direct1 10:16] * (240)
192.168.1.0/24   via 172.16.0.2 on enc1 [router_tunnelA 10:22]*(100/0)[AS65001i]
192.168.2.0/24   via 172.16.0.6 on enc1 [router_tunnelB 10:21]*(100/0)[AS65002i]
192.168.0.254/32 dev lo0 [kernel1 10:16] * (254)
192.168.1.254/32 dev lo0 [kernel1 10:16] * (254)
172.16.0.0/30    dev lo1 [direct1 10:16] * (240)
10.60.0.0/16     dev em0 [direct1 10:16] * (240)
172.16.0.4/30    dev lo2 [direct1 10:16] * (240)
```

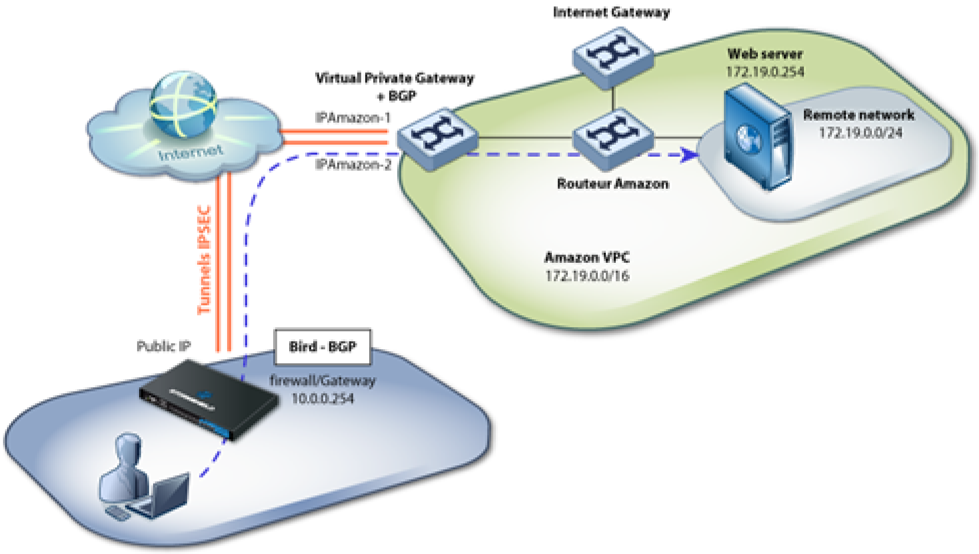
Routing table on spokeA:

```
bird> show route
0.0.0.0/0      via 10.60.0.254 on em0 [kernel1 13:32] * (254)
```

```
192.168.0.0/24 via 172.16.0.1 on enc1 [router_tunnelA 13:32] * (100/0) [i]
192.168.2.0/24 via 172.16.0.1 on enc1 [router_tunnelA 13:32] * (100/0) [i]
192.168.1.0/24 dev em1 [direct1 13:32] * (240)
172.16.0.0/30  dev lo1 [direct1 13:32] * (240)
10.60.3.128/32 dev lo0 [kernel1 13:32] * (254)
10.60.0.0/16   dev em0 [direct1 13:32] * (240)
```

Appendix B: Amazon VPC connectivity

The aim here is to link up a local network to an Amazon VPC (Virtual Private Cloud). To do so, Amazon enables the creation of two routed tunnels between the local firewall and the Amazon cloud, and by routing this traffic via BGP.



Amazon configuration

Follow the steps below:

1. Create an Amazon VPC,
2. Create a sub-network in this VPC,
3. Configure routing in this VPC,
4. Create a dynamic VPN connection to the UTM via the object Amazon Virtual Private Gateway,
5. Create ACLs to allow local traffic going to the web server,
6. Routing: enable route propagation to the VPC's routing table.

Extract from configuration help provided by Amazon during the configuration of the service:

```
The Customer Gateway inside IP address should be configured on your tunnel
interface.

Outside IP Addresses:
- Customer Gateway              : IP publique Firewall/Gateway
- Virtual Private Gateway       : IPAmazon-1

Inside IP Addresses
- Customer Gateway              : 169.254.254.66/30
- Virtual Private Gateway       : 169.254.254.65/30

Configure your tunnel to fragment at the optimal size:
- Tunnel interface MTU          : 1436 bytes

#4: Border Gateway Protocol (BGP) Configuration:
```

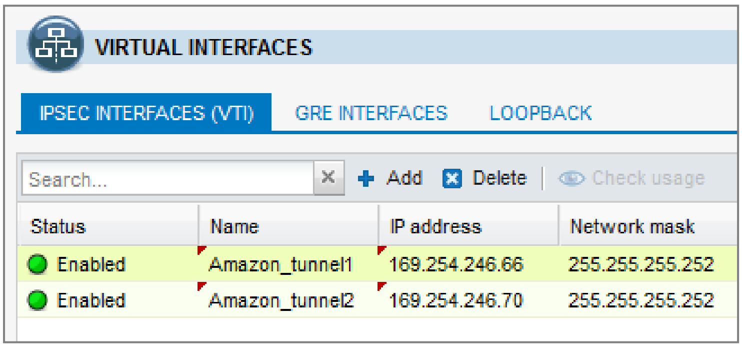
```
The Border Gateway Protocol (BGPv4) is used within the tunnel, between the
inside IP addresses, to exchange routes from the VPC to your home network. Each
BGP router has an Autonomous System Number (ASN). Your ASN was provided to AWS
when the Customer Gateway was created.

BGP Configuration Options:
- Customer Gateway ASN          : 65000
- Virtual Private Gateway ASN   : 9059
- Neighbor IP Address           : 169.254.254.65
- Neighbor Hold Time            : 30

Configure BGP to announce routes to the Virtual Private Gateway. The gateway
will announce prefixes to your customer gateway based upon the prefix you
assigned to the VPC at creation time.
```

Configuration of tunnels

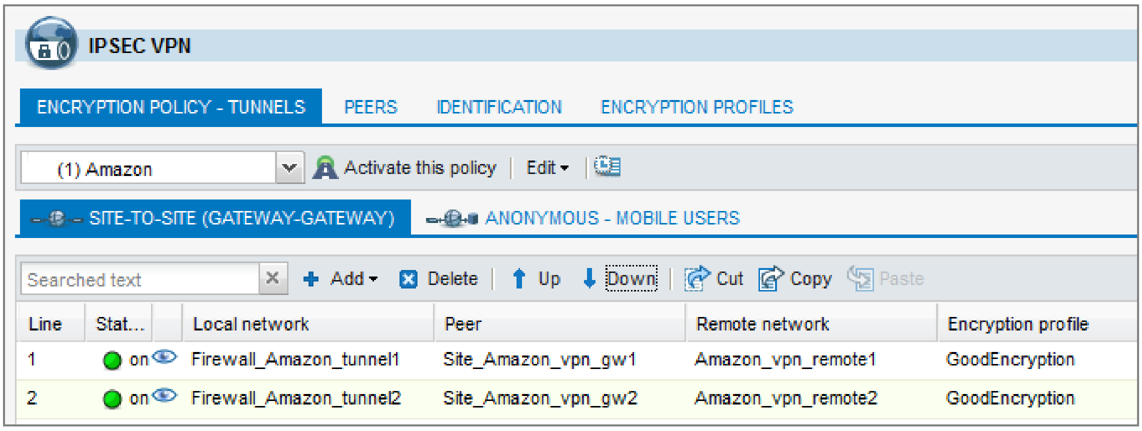
In the **Virtual interfaces** module (**Network** section), the *IPSec interfaces* tab allows you to define the interfaces concerned:



Status	Name	IP address	Network mask
● Enabled	Amazon_tunnel1	169.254.246.66	255.255.255.252
● Enabled	Amazon_tunnel2	169.254.246.70	255.255.255.252

In the IPsec VPN module (VPN section), under the Site to site (gateway-gateway) tab, you will be able to define the tunnels below using the following objects:

Site_Amazon_vpn_gw1:	IPAmazon-1
Site_Amazon_vpn_gw2:	IPAmazon-2
Amazon_vpn_remote1:	169.254.254.65
Amazon_vpn_remote2:	169.254.254.69



Line	Stat...	Local network	Peer	Remote network	Encryption profile
1	● on	Firewall_Amazon_tunnel1	Site_Amazon_vpn_gw1	Amazon_vpn_remote1	GoodEncryption
2	● on	Firewall_Amazon_tunnel2	Site_Amazon_vpn_gw2	Amazon_vpn_remote2	GoodEncryption

BGP configuration

We have chosen to export only the network 10.0.1.0/24

```
filter filter_net_in {
  if(net = 10.0.1.0/24) then {
  accept;
  }
  else reject;
}

protocol bgp router1 {
  local as 65000;
  neighbor 169.254.254.65 as 9059;
  hold time 30;
  multihop;
  import all;
  export filter filter_net_in;
  source address 169.254.254.66;
}

protocol bgp router2 {
  local as 65000;
  neighbor 169.254.254.69 as 9059;
  hold time 30;
  multihop;
  import all;
  export filter filter_net_in;
  source address 169.254.254.70;
}
```

STORMSHIELD

documentation@stormshield.eu

All images in this document are for representational purposes only, actual products may differ.