NOTE TECHNIQUE

STORMSHIELD NETWORK SECURITY

ROUTAGE DYNAMIQUE BIRD

Produits concernés : SNS 4.x

Date : 09 décembre 2019

Référence : sns-fr-routage_dynamique_bird_note_technique_V4

Table des matières

Introduction 3
Configuration via l'interface d'administration Web 3
Environnement Bird / Stormshield Network 4
Birdc/birdc6: contrôle distant 4
Configuration 8
Règles de syntaxe 8
Interaction avec le routage Stormshield Network 9
Configurations simples 11
RIP 11
OSPF 12
BGP 15
Configuration « BGP_simple » 15
Authentification 17
Configuration avancée 20
Configuration BIRD 21
Annexe A : Tunnels VPN Hub'n Spoke routés via BGP 25
Configuration des tunnels 25
Configuration BGP du site principal (Hub) 26
Configuration BGP du site satellite Spoke A 27
Configuration BGP du site satellite Spoke B 28
Vérification des tables de routage 28
Annexe B : Connectivité Amazon VPC 30
Configuration Amazon 30
Configuration des tunnels 31
Configuration BGP 31

Introduction

Ce document a pour objet de guider l'administrateur d'un Firewall Stormshield Network dans la configuration et l'exploitation du module de routage dynamique intégré BIRD.

Dans un premier temps sont décrits l'environnement de configuration ainsi que les modes d'interaction avec le moteur de routage. On détaille ensuite une configuration typique simple pour les trois protocoles de routage BGP, RIP & OSPF. Ces exemples sont l'occasion de découvrir la structure de configuration des protocoles ainsi que des éléments périphériques, filtrage et affichage des états. La dernière section se concentre sur une configuration plus complexe.

Notez que BIRD propose de multiples options de configurations notamment pour l'échange de routes entre process, leur filtrage ou une pseudo virtualisation des instances de routage. Ces éléments avancés et spécifiques à BIRD sont exclus du périmètre du document. De même l'utilisation des ROA BGP n'est pas couverte mais notez qu'elle est bien supportée par BIRD.

Configuration via l'interface d'administration Web

L'utilisation du routage dynamique peut être activée ou désactivée depuis le module **Routage** de l'interface d'administration Web.

Les onglets *Routage Dynamique* et *Routage Dynamique IPv6* (si le support IPv6 est activé) permettent l'édition des fichiers de configuration *bird.conf* en IPv4 et *bird6.conf* en IPv6.

Lorsque la configuration est envoyée au firewall, et en cas d'erreur de syntaxe, un message indiquant le numéro de ligne en erreur informe de la nécessité de corriger la configuration.

> **ATTENTION**
> Aucune sauvegarde du fichier de configuration précédent n'est effectuée avant sa modification. Il est donc conseillé de le copier (CTRL+A/CTRL+C), et de le coller dans un éditeur de texte pour sauvegarder le fichier.

Toutefois, l'éditeur de l'interface graphique ne permet pas d'accéder aux modes interactifs Birdc et birdc6, permettant le contrôle du routage dynamique (tests de fonctionnement de nouvelle configuration via une configuration temporaire et visualisation des états).

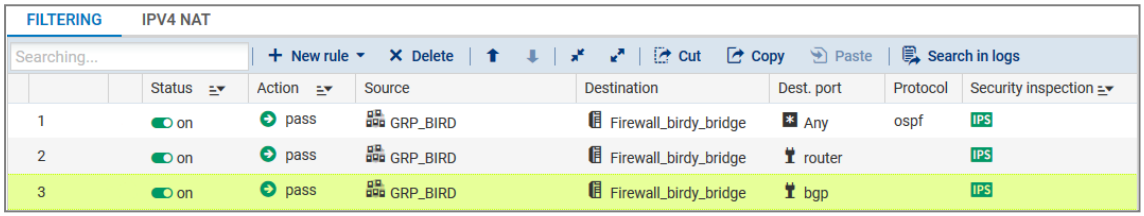
Environnement Bird / Stormshield Network

Dans une configuration sortie d'usine, le module de routage BIRD n'est pas activé.

Il est possible de faire coexister le routage du firewall Stormshield Network et le routage dynamique BIRD. Par exemple, la zone interne peut être gérée avec un protocole de routage dynamique et la zone externe avec les fonctionnalités de routage du firewall (routage statique, passerelles, routage par règles (PBR), objets routeur).

Pour cela, consultez la section Interaction avec le routage Stormshield Network.

Dans l'interface d'administration, des règles de filtrage sont nécessaires pour autoriser les flux de routage BIRD.



Exemple de configuration avancée

Depuis la version de firmware 1.0 et le support d'IPv6, la configuration du routage dynamique s'effectue au sein de deux fichiers, selon la version IP des réseaux concernés:

- **/usr/Firewall/ConfigFiles/Bird/bird.conf** pour les réseaux et routes IPv4.
- **/usr/Firewall/ConfigFiles/Bird/bird6.conf** pour les réseaux et routes IPv6.

Le démarrage de BIRD requiert deux opérations distinctes.

Le module BIRD doit tout d'abord être défini comme actif dans le fichier suivant : **/usr/Firewall/ConfigFiles/Bird/global**. Cela se réalise par le passage à « 1 » de la variable « state » dans la section [bird], pour le routage IPv4, et/ou dans la section [bird6], pour le routage IPv6.

```
[bird]
state=1

[bird6]
state=1
```

Cette opération assure la persistance de l'activation du routage dynamique lors du redémarrage du firewall.

Ensuite, pour démarrer BIRD ou pour recharger sa configuration suite à une modification, utilisez la commande « enbird ». Si la configuration contient des erreurs de syntaxe, la commande les signale et n'active pas la configuration.

```
V50XXA0D0000073>enbird
V50XXA0D0000073>
```

Birdc/birdc6: contrôle distant

BIRD et BIRD6 disposent d'un mode interactif : ***birdc*** pour BIRD Client, et ***birdc6*** pour BIRD6 Client. Lancez ce mode en appelant **birdc** ou **birdc6**, selon la version IP du routage dynamique que vous souhaitez contrôler.

```
V50XXA0D0000073>birdc
```

```
BIRD 1.6.7 ready.
bird>
```

```
V50XXA0D0000073>birdc6
BIRD 1.6.7 ready.
bird>
```

NOTE

Dans la suite de ce document, tous les exemples sont présentés pour le mode interactif ***birdc***. Ils sont entièrement transposables pour le mode interactif ***birdc6***.

Le mode interactif de BIRD ne permet pas de modifier le fichier de configuration, mais de visualiser les états de BIRD, de tester le bon fonctionnement d'une nouvelle configuration en permettant de revenir en arrière, et de créer une configuration temporaire.

Commandes « Show »

Le caractère "?" vous permet d'afficher la liste des options disponibles.

```
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>
```

Exemple :

Affichage de toutes les 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>
```

Exemple :

Affichage des routes par Instance de protocole. Dans ce cas, l'instance est **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>
```

Dans ***birdc***, la plupart des commandes sont communes à l'ensemble des protocoles. Ainsi par exemple, les routes annoncées à un voisin BGP sont visualisées par une commande qui fait appel au filtre d'export.

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

Debug

Les commandes *Show* donnent de nombreux renseignements sur les instances. Elles permettent de diagnostiquer les problèmes, qu'ils soient dus à une mauvaise configuration, un problème de réseau, ou autre.

```
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>
```

Pour activer la réception des messages systèmes sur la console, entrez la commande echo all puis echo off pour stopper ces 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
```

Les événements de *debug* sont visualisés globalement ou par exemple par instance de protocole. L'exploitation des commandes de *debug* est un outil intéressant qui complète efficacement les commandes de visualisation d'états.

```
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
```

Test temporaire d'une nouvelle configuration

On souhaite tester une nouvelle configuration **bird_a_tester.conf**. Pour cela, lancez BIRD en utilisant une configuration **bird.conf** dont le fonctionnement est validé, puis lancez **birdc**.

Pour vérifier la syntaxe du fichier sans l'appliquer:

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

Ensuite, appliquez temporairement cette configuration pendant 60 secondes par la commande :

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

La nouvelle configuration s'applique. Si le firewall n'est plus joignable ou sans confirmation de la part de l'administrateur, la configuration précédente sera ré-appliquée automatiquement au bout de 60 secondes.

Si la nouvelle configuration est considérée comme valide, on peut la confirmer grâce à :

```
bird> configure confirm
```

Si la nouvelle configuration n'est pas validée et que le firewall est encore joignable, on peut revenir en arrière immédiatement grâce à :

```
bird> configure undo
```

Configuration

Toute implémentation demande à minima à ce que les lignes suivantes soient configurées afin de définir un environnement basique de coopération avec le système.

```
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
}
```

Nous ne nous attarderons pas ici sur le détail de chaque ligne de configuration. Si vous désirez en obtenir des explications exhaustives, consultez la documentation en ligne de BIRD à l'adresse :

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

Les notions les plus importantes sont celle d'instance de protocole et celle de filtre.

Une instance de protocole peut être soit BGP soit RIP soit OSPF et définit une configuration appropriée. On peut définir plusieurs instances, éventuellement pour un même protocole.

Chaque instance de protocole est connectée à une table de routage interne à BIRD. Cette connexion est contrôlée par deux filtres qui peuvent accepter, refuser ou modifier les routes.

Le filtre d'exportation contrôle les routes transmises de la table de routage interne à BIRD vers le protocole, le filtre d'importation fait de même dans l'autre sens.

Attention à mettre en œuvre un filtrage précis des routes. L'utilisation d'export ou d'import complets de routes (par exemple, import all;) entre instances de protocole peut avoir des effets destructeurs.

Règles de syntaxe

- Le texte sur la ligne placé après # est un commentaire
- Le texte entouré de /* et */ est un commentaire
- Les blocs de plusieurs options sont placés entre accolades {}
- Chaque option se termine par un point-virgule ;
- La configuration est sensible à la casse.

La configuration suivante comporte deux erreurs de syntaxes.

```
1  router id 192.168.97.219;
2
3  protocol kernel
```

```
4  {
5  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 }
```

Message d'erreur 1

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

Si une accolade de fermeture de bloc est oubliée, l'erreur mentionnera la première ligne du bloc suivant, ligne ne correspondant pas à une commande autorisée du bloc non fermé.

Il faut donc insérer le caractère « **}** » en ligne 13

Message d'erreur 2

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

Il faut donc insérer le caractère « **;** » à la fin de la ligne 17 sur l'exemple.

Interaction avec le routage Stormshield Network

Grâce à la configuration fournie par défaut sur les firewalls Stormshield Network, le routage du firewall est prioritaire sur le routage dynamique (préférence maximale de 254).

> **ATTENTION : limitation connue**
> **Pendant la reconfiguration des routes du firewall, celles-ci sont temporairement effacées et BIRD/BIRD6 peut alors configurer ses propres routes. Il faut donc protéger le routage du firewall grâce à un filtre d'export sur le pseudo-protocole *kernel*.**

Voici un exemple de filtre qui va protéger la route par défaut et la route statique 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;
```

```
}
```

Routage dynamique prioritaire sur le routage Stormshield Network

Si l'on veut que le routage dynamique soit prioritaire sur le routage Stormshield Network, il faut que les routes obtenues par routage dynamique (protocole BGP, OSPF ou RIP) aient une valeur de préférence plus élevée que les routes obtenues par le système (pseudo-protocole *kernel*).

Il faut donc diminuer la valeur de préférence de *kernel*, par exemple à 1:

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

Routage des interfaces du firewall

Si les interfaces du firewall sont configurées avec des sous-réseaux différents, et que l'on souhaite transmettre les sous-réseaux des interfaces via BIRD, on utilise le pseudo-protocole *direct*.

Par défaut, toutes les interfaces sont prises en compte. On peut restreindre l'ensemble des interfaces prises en compte grâce à l'attribut *interface*.

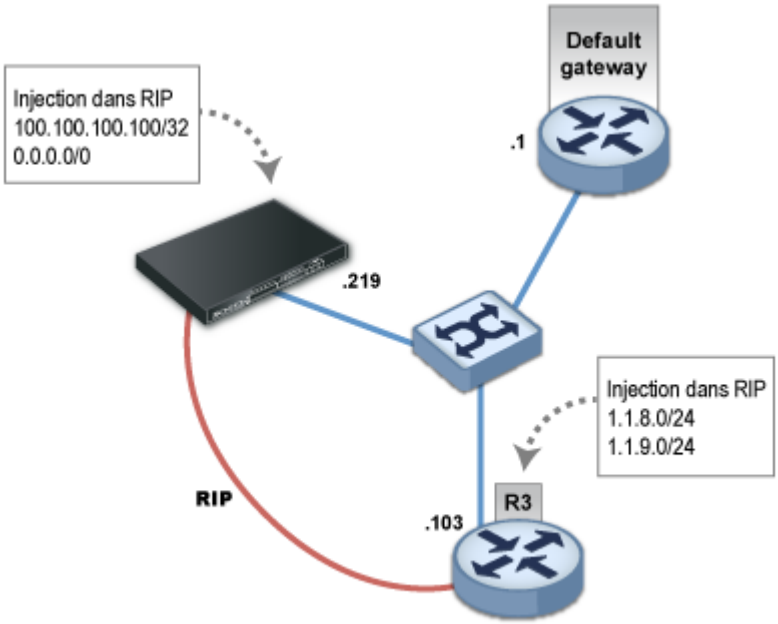
```
protocol direct
{
interface "-vlan*", "*"; # Exclut les VLANs
}
```

Configurations simples

RIP

La version supportée est RIPv2.

Voici ci-dessous la configuration « RIP_simple ».



On configure une route par défaut et une route statique vers 100.100.100.100/32 :

STATIC ROUTES

Searching... + Add ✕ Delete

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

On configure RIPv2 en spécifiant « multicast » comme mode RIP associé à l'interface « em0 ».
Les requêtes d'envoi des tables demandées par les voisins sont honorées.

```
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;
          authentication none;
          };
  honor always;
  authentication none;
  import all;
  export filter ripexport;
}
```

Affichage de l'état de l'instance de protocole :

```
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>
```

Affichage des routes apprises :

```
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)
```

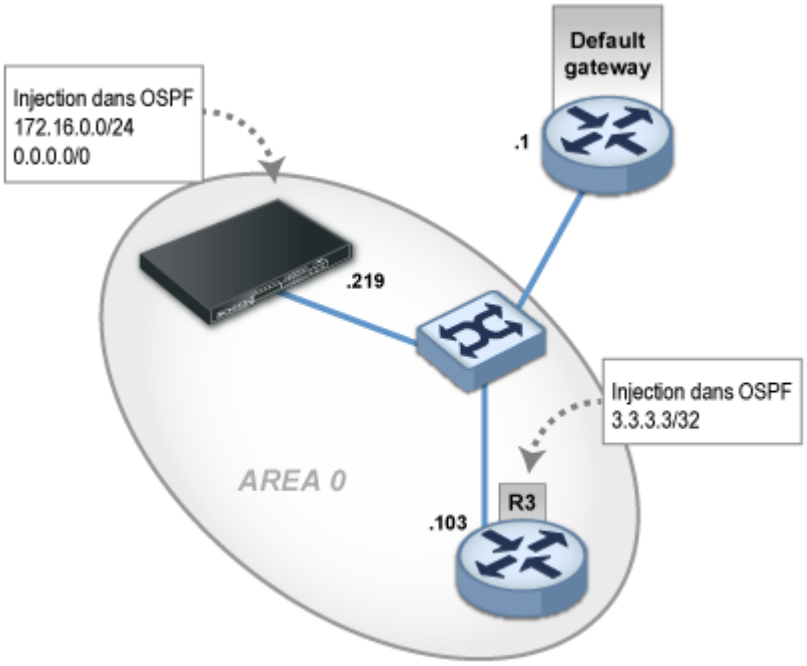
Voici ci-dessous les routes reçues par le voisin. Notez que la route par défaut est reçue. L'export de cette route est en effet normalement rejeté par les routeurs du marché. Ici, il est nécessaire de le filtrer explicitement.

```
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

Les versions supportées sont OSPFv2 pour IPv4 et OSPFv3 pour IPv6.

Voici ci-dessous la configuration « OSPF_simple ».



Elle consiste à déployer une aire 0 sur un LAN où l'on désigne explicitement un voisin éventuel. Toutes les routes sont importées d'OSPF. On redistribue dans OSPF la route du sous-réseau directement relié à l'interface em3 (172.16.0.0/24), ainsi que la route par défaut.

```
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

Il est conseillé de positionner le paramètre "`priority 0`" dans la section *interface* de la configuration du noeud OSPF afin de désactiver la participation du firewall aux élections pour les rôles de Designated Router / Backup Designated Router.

La commande suivante indique que le voisinage est bien établi (indiqué par l'état « full »).

Le voisin est déclaré comme « Designated Router » (indiqué par l'état « 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 reçues:

```
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>
```

On peut afficher la topologie OSPF :

```
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>
```

Ainsi que la base de données LSA :

```
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

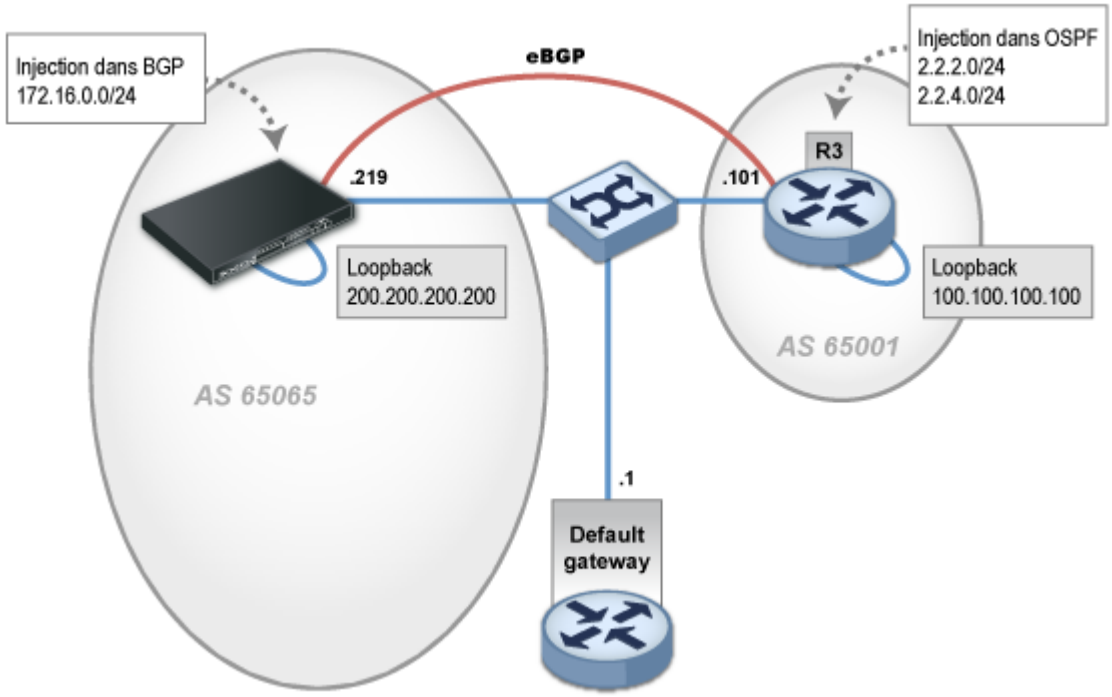
Notez que le type de LSA est présenté à gauche alors qu'il sert généralement de délimiteur horizontal dans les conventions d'affichage traditionnelles.

BGP

Configuration « BGP_simple »

La version supportée est BGPv4 pour IPv4 et IPv6.

Voici ci-dessous la configuration « BGP_simple ».



La configuration « BGP_simple » est implémentée de la façon suivante :

```
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;
}
```

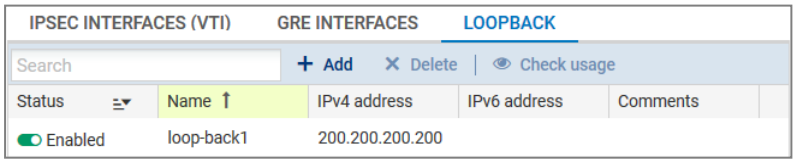
Explications

Contrairement à la majorité des routeurs du marché, il est nécessaire de spécifier l'AS local pour chaque instance BGP.

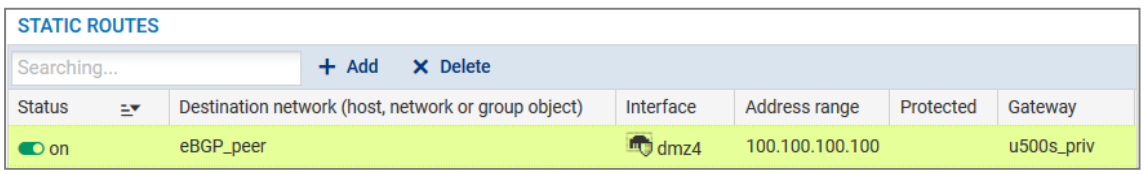
Selon les bonnes pratiques, on monte cette session eBGP entre des interfaces *loopbacks* et non pas les interfaces physiques. Il est donc nécessaire de configurer l'IP de la *loopback* locale en question (200.200.200.200/32), de spécifier cette adresse comme source et une route statique vers la *loopback* du voisin.

Interfaces virtuelles Loopback

L'interface d'administration web permet de configurer les interfaces de type loopback via le module **Configuration** > **Réseau** > **Interfaces virtuelles**, onglet *Loopback*.



Il est conseillé de déclarer la route statique vers la loopback distante sur le firewall en dehors de la configuration BIRD, via le module **Configuration** > **Réseau** > **Routage**, onglets *Routes statiques*, afin d'éviter que le trafic BGP soit bloqué par des alarmes "Usurpation d'adresse IP".



A nouveau, on sélectionne seulement le sous-réseau 172.16.0.0/24 relié directement à l'interface *em3* comme route à annoncer à nos voisins.

Ici on a défini un filtre d'export anonyme, directement dans l'instruction "export", grâce au mot-clé "where". Ce filtre d'export sélectionne les routes dont la source est RTS_DEVICE, c'est-à-dire les routes obtenues par le pseudo-protocole direct.

La valeur du *hold-time* est spécifiée à 180s, valeur habituelle du marché. BIRD implémente par défaut 240s. Il n'est pas nécessaire de spécifier la valeur du délai de *keepalive* (calculé à 1/3 du *hold-time*) mais nous le mentionnons explicitement pour plus de lisibilité. De même pour la *local-preference* par défaut.

La commande « show protocols » ci-dessous confirme que la session est bien fonctionnelle.

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

Les routes sont bien reçues du voisin :

```
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>
```

Sur le voisin BGP on reçoit bien la route annoncée et libérée par le filtre. La route 1.1.1.1/32 est pour sa part, effectivement bloquée.

```
@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:~$
```

Authentification

Il est possible de mettre en œuvre une authentification TCP-MD5 entre routeurs BGP au sein de BIRD.

Cette méthode permet ainsi la protection des sessions BGP par authentification des trames dans l'entête TCP conformément à la RFC2385.

Cela se traduit par l'ajout *"password"* dans la configuration du routeur BGP au sein des fichiers **/usr/Firewall/ConfigFiles/Bird/bird.conf** (routage dynamique des paquets IPv4) et **/usr/Firewall/ConfigFiles/Bird/bird6.conf** (routage dynamique des paquets IPv6). Par exemple :

```
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;
}
```

Il est ensuite nécessaire de déclarer cette association de sécurité (SA : Security Association) dans le fichier **ConfigFiles/Bird/global** au sein de la section [BGPAuth]. Cette section unique regroupe les associations de sécurité IPv4 et IPv6. Chaque SA y est décrite sous la forme d'une ligne structurée comme suit (séparation des champs par des virgules et sans espace):

```
[BGPAuth]
IP_routeur_local,IP_routeur_distant1,mot_de_passe_1
IP_routeur_local,IP_routeur_distant2,mot_de_passe_2
```

Dans notre exemple :

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

NOTE
Les mots de passe ne doivent pas contenir d'espace ni de signe égal ('=').

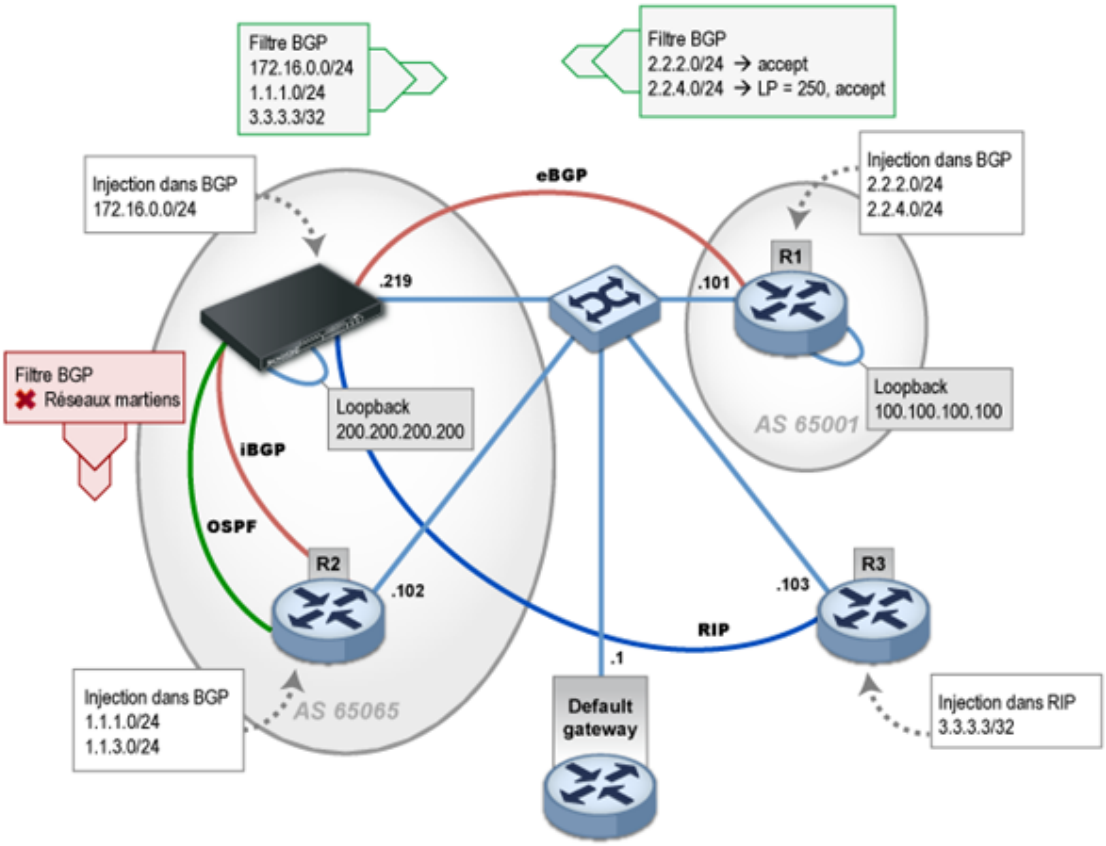
La commande `showSAD` permet de visualiser les SA de BGP (mais aussi du protocole IPSec). Le champ « **tcp mode=any** » indique une SA relative à l'authentification BGP ; le champ « **A : tcp-md5** » indique pour sa part le type d'authentification utilisée.

```
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
```

Configuration avancée

On met ici en œuvre la configuration avancée. Cette configuration réunit les trois configurations simples et comporte en plus une liaison iBGP établie en parallèle de la liaison OSPF.



Le réseau du client comprend le routeur R2, R3 et le firewall Stormshield Network. Le routeur R1 est un voisin BGP externe. Ce réseau représente un cas réaliste d'architecture, à l'exception du fait que tous les routeurs sont connectés physiquement par le biais d'un LAN unique.

On met en œuvre une politique standard de filtrage pour :

- n'annoncer que les réseaux publics en BGP vers l'extérieur,
- ne pas propager de réseaux internes ou martiens dans BGP interne,
- tagguer une des routes apprises en eBGP avec une local-preference de 250 ; cette mesure est généralement mise en œuvre pour contrôler le partage de charge entre plusieurs voisins eBGP,
- n'annoncer dans OSPF qu'une route par défaut,
- n'annoncer dans RIP qu'une route par défaut.

Les réseaux annoncés par R2 et R3 le sont respectivement via BGP et RIP. L'utilisation d'OSPF pour annoncer la route par défaut n'a qu'une utilité pédagogique.

Configuration BIRD

Ci-dessous le fichier de configuration équivalent en 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;
                                        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

Il est conseillé de positionner le paramètre "`priority 0`" dans la section *interface* de la configuration du noeud OSPF afin de désactiver la participation du firewall aux élections pour les rôles de Designated Router / Backup Designated Router.

Table de routage du firewall Stormshield Network

```
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)
```

Afin de pouvoir vérifier la local-preference sur la route 2.2.4.0/24 on affiche le détail des routes de l'instance du protocole router1 :

```
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

On constate ici que la route par défaut est également bien annoncée.

```
@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:~$
```

Dans le cas où ce trafic doit être routé symétriquement - par exemple en cas de NAT - il est nécessaire d'adapter la configuration de BIRD afin d'annoncer le firewall en tant que next-hop. La modification peut se faire dans le filtre « default_ok » qui sert à annoncer la route par défaut à R3 via RIP ainsi qu'à 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;
  }
}
```

Pour imposer une autre passerelle que le firewall lui-même, il faut utiliser la 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:~$
```

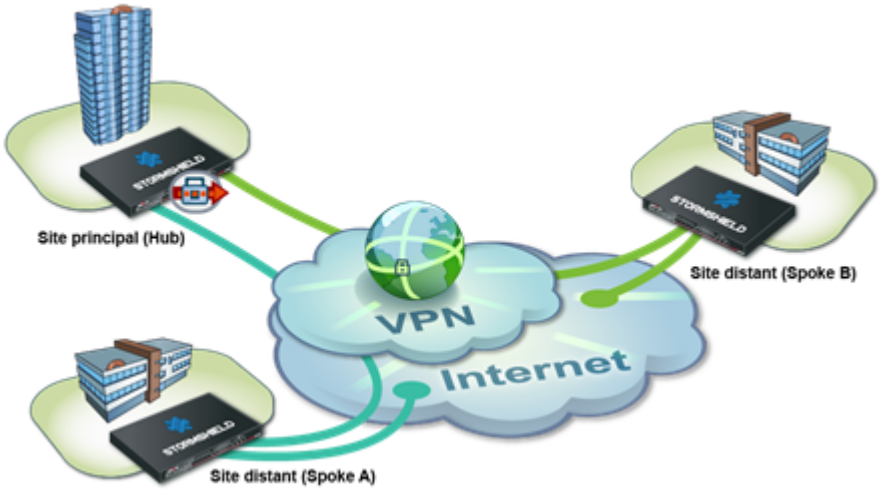
Router R1 – show IP route

```
@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:~$
```

Annexe A : Tunnels VPN Hub'n Spoke routés via BGP

Voici un exemple de routage dynamique BGP dans le cadre d'un réseau VPN en étoile de type Hub and Spoke.

Configuration des tunnels



Pour le paramétrage de la politique IPsec Hub'n Spoke, consultez le *HOW TO* cité ci-dessous. Dans notre cas, les différences de paramétrage par rapport à cette procédure consistent à configurer les extrémités de trafic au moyen d'interfaces virtuelles, au lieu de réseaux distants dans la politique IPsec (voir paragraphe suivant).

Rendez-vous à l'adresse http://documentation.stormshield.eu/. Reportez-vous au *HOW TO : VPN IPSec - Configuration Hub and Spoke*, et consulter le *cas n° 1 : trafic interne via les tunnels IPSec*.

Site principal

TunnelA

Réseau local :	Interface ipsec1 (172.16.0.1)
Correspondant :	Site_SpokeA
Réseau distant :	Remote_tunnelA (172.16.0.2)

TunnelB

Réseau local :	Interface ipsec2 (172.16.0.5)
Correspondant :	Site_SpokeB
Réseau distant :	Remote_tunnelB (172.16.0.6)

Spoke A

Réseau local :	Interface ipsec1 (172.16.0.2)
Correspondant :	Site_FW_Hub
Réseau distant :	Remote_tunnelA (172.16.0.1)

Spoke B

Réseau local :	Interface ipsec1 (172.16.0.6)
Correspondant :	Site_FW_Hub
Réseau distant :	Remote_tunnelB (172.16.0.5)

Configuration BGP du site principal (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;
}
```

Configuration BGP du site satellite Spoke A

```
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;
}
```

Configuration BGP du site satellite Spoke B

```
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;
}
```

Vérification des tables de routage

Table de routage sur le site principal (Hub) :

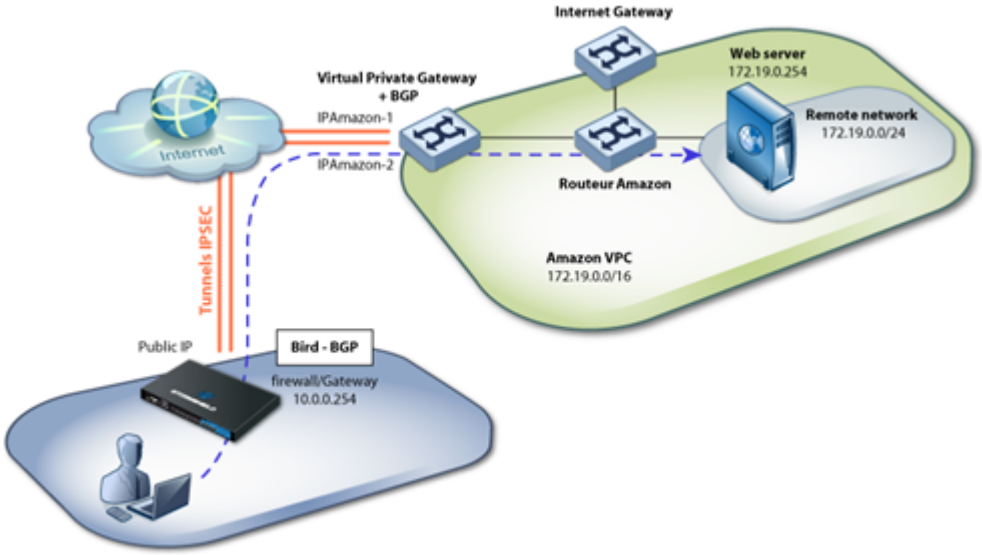
```
bird> show route
0.0.0.0/0         via 10.60.0.254 on em0 [kernel1 10:16] * (254)
10.60.3.127/32    dev lo0 [kernel1 10:16] * (254)
192.168.0.0/24    dev em1 [direct1 10:16] * (240)
192.168.1.0/24    dev em2 [direct1 10:16] * (240)
192.168.1.0/24    via 172.16.0.2 on enc1 [router_tunnelA 10:22]*(100/0)[AS65001i]
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)
```

Table de routage sur 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)
```

Annexe B : Connectivité Amazon VPC

Le but est de relier un réseau local à un VPC Amazon (Virtual Private Cloud). Pour cela, Amazon propose la création de deux tunnels routés entre le firewall local et le Cloud Amazon et de router ce trafic via BGP.



Configuration Amazon

Suivez les étapes suivantes :

1. Créez un VPC Amazon,
2. Créez un sous réseau dans ce VPC,
3. Configurez le routage dans ce VPC,
4. Créez une connexion VPN dynamique vers l'UTM via l'objet Amazon Virtual Private Gateway,
5. Créez les ACLs pour autoriser le trafic local vers le serveur Web,
6. Routage : activez la propagation des routes à la table de routage du VPC.

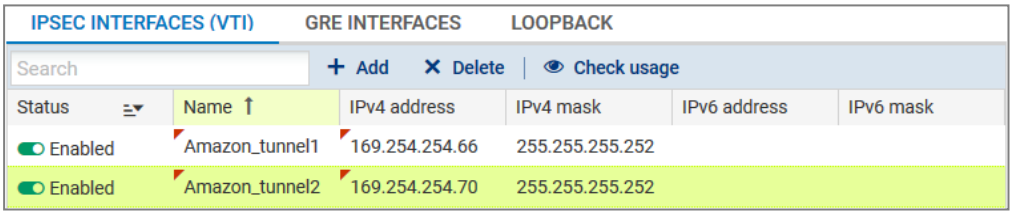
Extrait de l'aide de configuration fournie par Amazon lors de la configuration du service :

```
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.
```

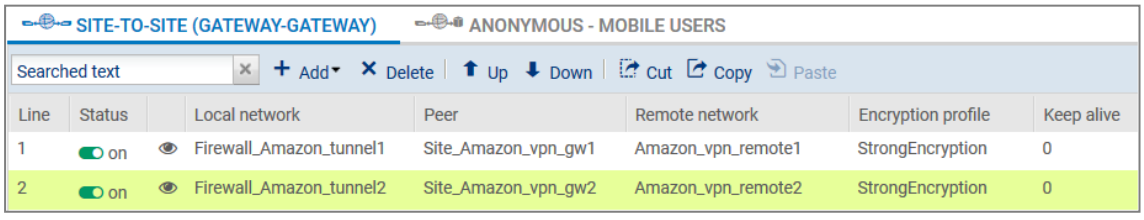
Configuration des tunnels

Dans le module **Configuration** > **Réseau** > **Interface virtuelles**, l'onglet *Interfaces IPsec* vous permet de définir les interfaces concernées :



Dans le module **Configuration** > **VPN** > **VPN IPsec**, onglet *Site à site (gateway-gateway)*, vous pouvez définir les tunnels ci-dessous, à l'aide des objets suivants :

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



Configuration BGP

On choisit d'exporter uniquement le réseau 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;
  }
```

documentation@stormshield.eu