# Table of Contents

**Soft-Router (routing function)** ..... 3
***General Description*** ..... 3
The Internal Router Architecture ..... 3
VRF General Description ..... 5
VRF Lite in Stingray Service Gateway ..... 5
***TAP Subnets Configuration*** ..... 7
***Creating veth interfaces*** ..... 9
***SSG Settings*** ..... 11
SSG Configuration ..... 11
Setting veth-Interface Names ..... 16
LAG Support ..... 17
***Multi-path routing (ECMP)*** ..... 19
***Specifics of addresses announcement*** ..... 19
Subscriber announcements and NAT pool ..... 19
Announcement of L3 subscriber addresses ..... 21
***Root Daemon Configuration (BIRD, FRR, etc.)*** ..... 21
BIRD configuration example ..... 22
***Router Troubleshooting*** ..... 25
Tracing ..... 26
***CLI commands*** ..... 26

# Soft-Router (routing function)

## General Description

Stingray Service Gateway (SSG) itself does not build the routing table. It delegates this work to proven specialised tools. The example uses the BIRD root daemon. The router daemon processes the required routing protocols (BGP, OSPF, etc.) and uses them to build a common routing table which it loads into the kernel. SSG performs routing of packets using this table.

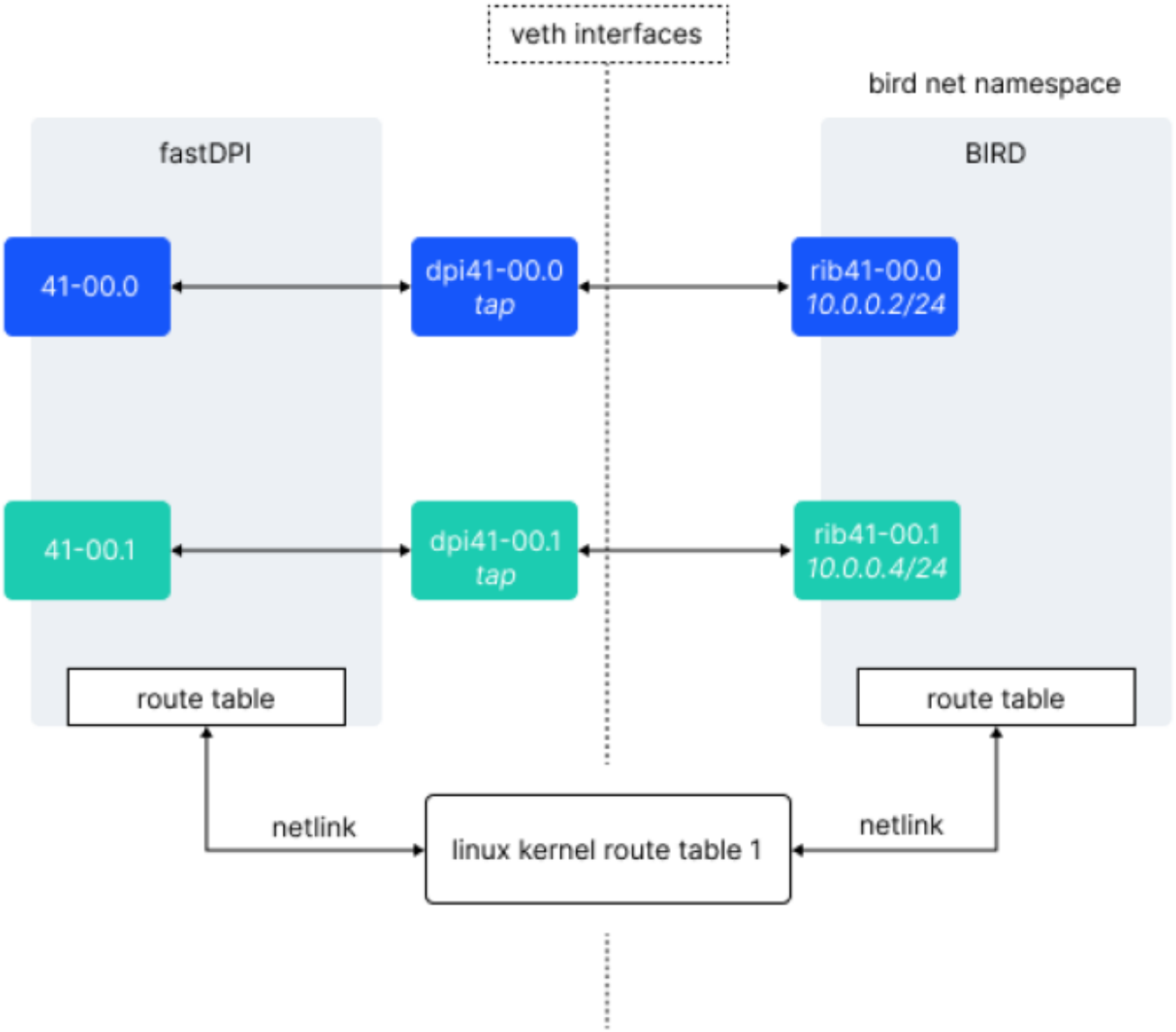
Instead of BIRD, any other daemon that builds a routing table in the Linux kernel can be used, such as FRRouting, QUAGGA, Juniper CRPD, etc. since Stingray Service Gateway only uses the standard Linux interface to read the routing table and is therefore compatible with any daemon.

In future versions, in order to save memory, it is possible to introduce optional specialized APIs for communication with a particular daemon to bypass kernel route table construction and communicate with the daemon directly.

Since BIRD builds the routing table in the OS kernel, to avoid application of these rules by the Linux server itself, the BIRD root daemon runs in a separate net namespace (in the diagram it is `bird` netns). Routing protocol packets are received by SSG in/out devices in general traffic. For each in/out device, a veth pair of "shadow" interfaces with predefined names is created: the DPI interface of the veth pair works as a TAP interface, the rib interface works as a normal system interface in BIRD's netns.

System requirements are described in the Hardware Requirements and Performance section.

### The Internal Router Architecture




Data from the kernel route table is read (rtnetlink) in the router's RIB. RIB is a prefix tree, it is convenient to modify it by kernel route table change events (delete/add entries). But it is impossible to use RIB in routing because it does not support multithreaded access from work threads (it requires locking, which is unacceptable). Therefore, in SSG, RIB is in router thread and unavailable for worker threads.

The worker threads use FIB. This structure is designed for multi-threaded search (LPM – longestLPM - longest prefix match), but is not designed for modifications (deletion/addition of new records). FIB can only be built from scratch by RIB and then used for LPM. Therefore, there are two FIBs in SSG – the current one (which is currently used for routing by worker threads) and the 'future one'. SSG checks every `router_fib_refresh` seconds to see if there have been any changes to the RIB since the current FIB was built. If there were changes, SSG builds (in router thread) new FIB in place of “future” one, and then switches current FIB to a new one. By doing this, the worker threads will meet any changes that have occurred in the routing table.

**Default values in the Router**

**`router_max_ip4_route_count = 1000000.`** The maximum number of IPv4 routes for a particular

VRF.
**router_max_ip6_route_count = 200000.** The maximum number of IPv6 routes for a particular VRF.
**router_multipath_page = 8192.** The maximum number of pages for multi-path route distribution.
**router_fib_refresh = 15s.** FIB update Interval.
**router_arp_cache_size = 1024.** ARP cache size.

## VRF General Description

SSG implements VRF Lite – it shares routing tables, but does not put individual VRF traffic into a unique tunnel (MPLS, VXLAN).

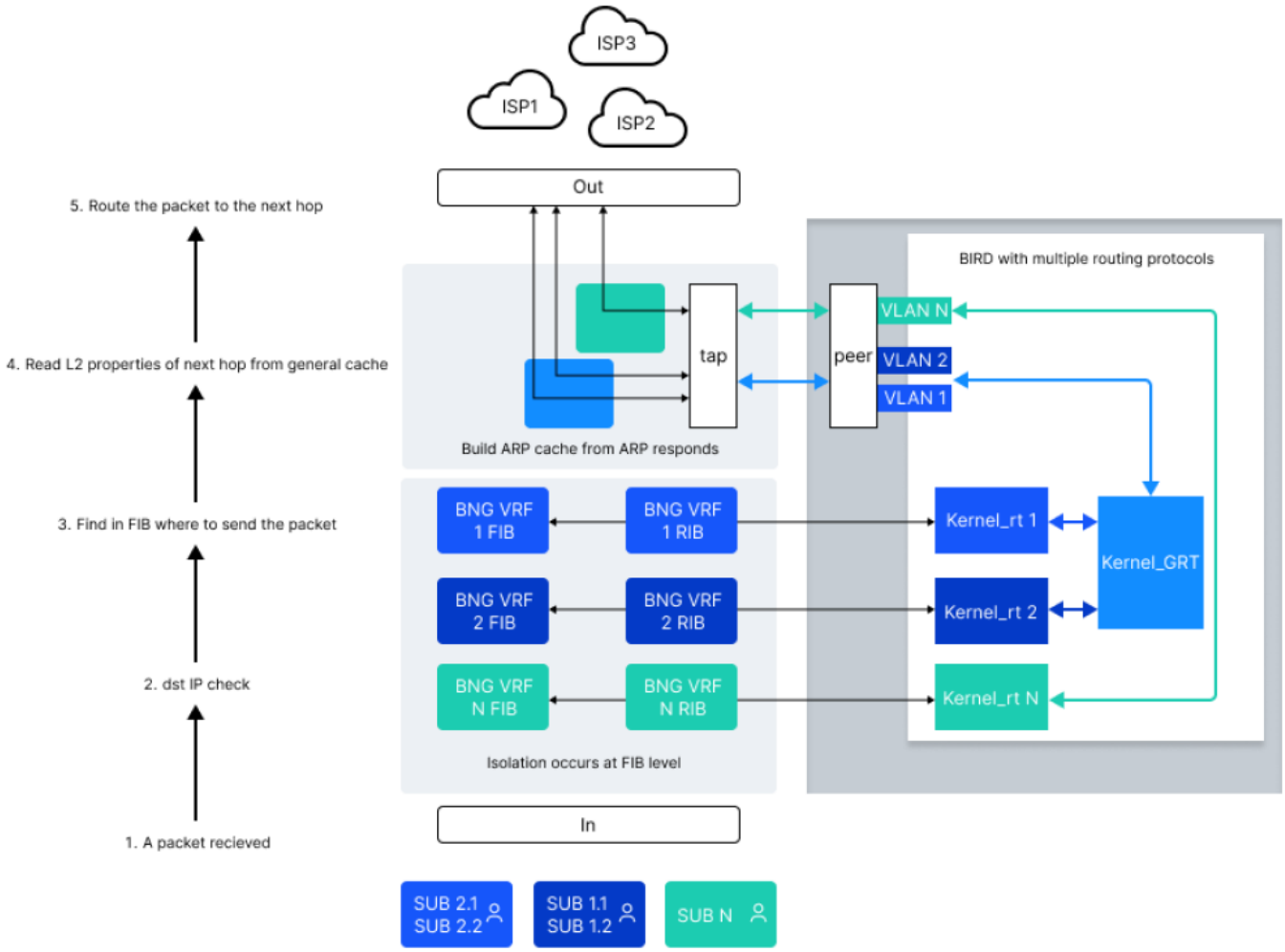
VRF (Virtual Routing and Forwarding instance) — is a routing virtualization mechanism. VRF allows you to create virtual routers on the same physical device with independent routing tables, interface lists, and other parameters. This enables virtualized isolated environments and ensures that each VRF has independent settings and does not share parameters with other VRFs and the physical device. Communication between different VRFs is possible, but it is strictly local to the device, and the VRF on one router is not connected in any way to the VRF on another.

Each VRF is a separate VPN and does not interconnect with other VRFs.

An example is the traffic allocation from an IPTV set-top box that is also located in the L2 domain with BRAS, as well as the CPE (Customer Premises Equipment – network equipment that provides user access to the ISP's network). The IPTV set-top box only accesses local resources, the CPE accesses the Internet.

VRF may be called Virtual Routing Instance in Juniper, VRF in Cisco and MikroTik, but they are all routing virtualization mechanism with similar features.

## VRF Lite in Stingray Service Gateway




The prefix Lite means that SSG only separates routing tables, but does not place the traffic of an individual VRF into a unique tunnel (MPLS, VXLAN). VRF Lite allows isolating the services provided between each other and optimizing routing when using different links.

VRF Lite is implemented in SSG by using Soft-Router, which builds a RIB table and writes/reads from the table. SSG diverts traffic to the Linux kernel for the routing daemon. FastDPI provides the FIB table construction.

The routing daemon runs in isolation to keep the OS from becoming a router. SSG sends all signaling to the router, the router itself builds routing rules. Then, using these rules, SSG starts forwarding and routing packets.

The current version (12.3) does not support L3VPN and MPLS, but you can configure special context routing if necessary.

VRF Lite in SSG is service 254, that may or may not be subscriber-enabled. By default, the subscriber falls into the VRF which is specified in SSG.

**Behavior**

Routing in the SSG can be done using the FRR or BIRD routing daemons, which is a separate process that handles the exchange of dynamic routing signaling protocols (BGP, OSPF). In the context of VRF Lite, BIRD suits better since the SSG runs on the Linux architecture. When using BIRD, it is possible to

handle route leaking in the routing tables.

This section covers the single namespace and multiple routing tables option, which is more suitable for the BIRD routing daemon. There is also a concept with multiple namespaces and a single table. This concept is more typical of FRR.

GRT (Global routing table) in this scheme is a conventional name, it is exactly the same kernel route table as the others in the scheme. The table is named so for the convenience and complexity of testing. It uses own routing rules from several routers, they can be multiple or have their own contexts.

All rules and filters are configured in the router (in this case, BIRD). When BIRD starts, when SSG is running and BGP neighboring is up, some incoming routes are folded into the Kernel_rt N table, others are folded into GRT and then flow into Kernel_rt 1 and Kernel_rt 2 according to specified rules.

The SSG describes the same VRFs targeting specific tables with routing rules. Each instance reads from the tables and builds isolated FIBs based on those tables.

The SSG has its own unique ARP cache, which it builds based on responses to ARP requests. An SSG can have multiple ARP tables, multiple VRFs can have one common table.

Due to **isolation**, the same subnet can be accessed through different hops and different routers and routed differently. After the FIB is built, when traffic flows, first the subscriber is authorized and placed in the required VRF. The FIB can be updated after the subscriber is authorized. The subscriber can be moved between VRFs dynamically after the subscriber is authorized by CoA.

The Default gateway for this subscriber will also be moved to another VRF and may become unavailable to other subscribers (who may already be using the network).

Actions when traffic is outgoing from a subscriber:

1. Find which VRF the subscriber belongs to.
2. Find the destination of the packet.
3. In the FIB, find which hop is to terminate the packet and terminate it. **Relevant only when creating flow, then nexthop is memorized for flow**.

Actions on incoming traffic:

1. Analyze dest.
2. Check subscriber status.
3. Test for packet origination.

For other groups of subscribers everything happens similarly in isolation, according to the rules. Announcement is performed from the VRF on which the subscriber is located.

## TAP Subnets Configuration

For each `router_device` it is mandatory to specify which subnets are allocated to the TAP (in fact, this is an allocation of routing protocol packets to the BIRD). SSG will allocate calls to these subnets

from the general traffic on the device and route all such packets to the appropriate TAP interface.

Subnets are defined by the `subnet` (for IPv4) and `subnet6` (for IPv6) parameters in the `router_device` description. Each subnet is defined with a separate `subnet`/`subnet6` parameter. You can have up to 16 different `subnet` parameters and up to 16 different `subnet6` parameters in the router_device description. For example, the following configuration specifies two IPv4 subnets for device `41-00.1` to be allocated to the TAP interface `tap41`, and one IPv6 subnet plus the link-local address of the interface with which bird communicates:

```
router_device {
        # Device name from in_dev/out_dev
    device=41-00.1
        # TAP interface name for the device (default='dpi' + device)
    tap=tap41
        # Name of the paired TAP interface in netns for the device
(default='rib' + device)
    peer=bgp41

        # Which IPv4 subnets to allocate to TAP
    subnet=10.0.2.0/30
    subnet=8.8.8.0/29

        # Which IPv6 subnets to allocate to TAP
    subnet6=2001::1/124
        # link-local the address of the interface with which bird
communicates
    subnet6=fe80::82d:cff:fe5f:9453/128
}
```

If IPv6 is used, note that link-local addresses play a major role in IPv6, which should also be specified in the `subnet6` parameters

OSPF uses the multicast addresses 224.0.0.5 and 224.0.0.6, so if the `router_device` uses OSPF, these addresses should also be specified in the `router_device` description:

```
router_device {
    device=41-00.1
    tap=tap41
    peer=bgp41

    # OSPF multicast
    subnet=224.0.0.5/32
    subnet=224.0.0.6/32
}
```

At least one IPv4 or IPv6 subnet must be specified in the `router_device` parameter

## Creating veth interfaces

Everything described in this section - creating veth interfaces, running BIRD, etc. - should be set in the system boot scripts and run **before** fastdpi is started.

Suppose we have the following devices specified in fastdpi.conf:

```
in_dev=41-00.0
out_dev=41-00.1
```

Suppose we need to configure the BGP protocol for uplink in BIRD, i.e. on the `41-00.1` device.

Shadow veth interfaces must be created for each in/out device whose traffic includes routing protocol packets, i.e. which require configuration in BIRD. If the device is not involved in routing (like `in_dev=41-00.0` in this example), no veth pair needs to be created for it.

To redirect BGP traffic from `41-00.1` to BIRD, which runs on `bird` netns, we need to create a veth shadow pair for the `41-00.1` interfaces.

Create `bird` netns (the name `bird` is arbitrarily chosen here, you may use a different netns name) which will run BIRD:

```
ip netns add bird
```

Create the veth pair:

```
ip link add dpi41-00.1 type veth peer name rib41-00.1 netns bird
```

The rib interface must have an IP address (and IPv6 if IPv6 is supported). This address will be the BGP peer address for the BGP neighbour.

```
ip netns exec bird ip address add 10.0.0.4/24 broadcast 10.0.0.255 dev
rib41-00.1
ip netns exec bird ip address add 2098::4/124 dev rib41-00.1
# enable ARP on the interface
ip netns exec bird ip link set dev rib41-00.1 arp on

# set tx checksum offload off - turn off checksum calculation on the
interface
# noticed that the CRC calculation on the interface may not be correct (at
least on some CentOS-8 kernel builds)
ip netns exec bird ethtool -K rib41-00.1 tx off
```

The IP address of the rib interfaces must be different from the SSG IP address given by

the `bras_arp_ip` and `bras_ipv6_address` parameters. Furthermore, to avoid confusion, the `bras_arp_ip` and `bras_ipv6_address` addresses should not be part of any subnet allocated to the TAP interfaces.

The dpi interface **should have neither IPv4 nor IPv6 addresses**, as SSG uses it as a TAP interface and it is not required to have addresses on it (indeed, it may even get in the way if the interface itself starts emitting packets):

```
ip link set dev dpi41-00.1 arp off
# Disable IPv6 on dpiXXX interfaces (so that there is not even a link-local
address)
echo 1>/proc/sys/net/ipv6/conf/dpi41-00.1/disable_ipv6
```

Finally, bring up all created interfaces:

```
ip link set dpi41-00.1 up
ip netns exec bird ip link set lo up
ip netns exec bird ip link set rib41-00.1 up
```

Do not forget the firewall:

```
firewall-cmd --zone=internal --add-source=10.0.0.1/24
firewall-cmd --zone=internal --add-rich-rule='rule family=ipv4 source
address=10.0.0.1/24 accept'
```

Remember that BIRD must be run in `bird` netns:

```
ip netns exec bird /usr/local/sbin/bird
```

The state of the veth interfaces is controlled by SSG: if device `41-00.1` is link down, SSG will link down the veth interfaces of that device; as soon as link `41-00.1` is up, SSG will link up the veth interfaces.

What about the VLAN?
SSG sends packets to the rib interfaces "as is" without any conversion. It means that if you have a VLAN, you have to use Linux to create vlan interfaces on the rib interface and bind the bird to those vlan interfaces.
In fastdpi.conf vlan interfaces created on a rib interface must not be listed anywhere - you must specify the two ends of the veth pair as `tap` and `peer`.

MTU
SSG **does not set** the MTU on the veth interfaces. When configuring the veth interfaces, the MTU must be set using the standard Linux tools.

# SSG Settings

## SSG Configuration

### Mandatory parameters

To enable the routing function, you need to activate the parameter in fastdpi.conf

```
    # [cold] enabling the router
    # Boolean parameter:
    # 0, false, off - router is off (default)
    # 1, true,  on - router is on
    # Does not allow changes on-the-fly via reload
router=1
```

Next you need to specify in which netns BIRD runs and the number of the kernel routing table it builds:

```
    # [cold] net namespace in which BIRD is running
router_netns=bird
    # [cold] Number of the kernel routing table that fastDPI uses
router_kernel_table=1
```

The following BRAS parameters must also be set, even if none of the BRAS modes are enabled:

```
    # Stingray Virtual MAC Address
bras_arp_mac=00:E0:ED:43:84:42

    # Stingray Virtual IP Address
bras_arp_ip=188.227.73.40

    # If IPv6 is used, virtual IPv6 addresses must be set:

    # Sets the global IPv6 address of the Stingray
bras_ipv6_address=2098::1

    # Sets the IPv6 link-local address of the Stingray (prefix FE80::/10)
    # If this parameter is not set explicitly, it is calculated by
bras_arp_mac
#bras_ipv6_link_local
```

These three parameters are **mandatory** to enable the router. The other parameters listed below are optional and are for fine-tuning the router in Stingray Service Gateway.

**Additional parameters**

The maximum number of routes is set by the parameters:

```
    # [cold] Maximum number of routes in IPv4 route table
    # Default value = 1000000
#router_max_ip4_route_count=1000000

    # [cold] Maximum number of routes in IPv6 route table
    # Default value = 200000
#router_max_ip6_route_count=200000
```

When starting in the router mode, SSG preallocates memory for the internal route table in accordance with these parameters. It is recommended to set these options (if necessary) with 20-30% reserve to ensure that the preallocated memory will be enough during the router operation.

The forwarding information base (FIB) in SSG is updated every `router_fib_refresh` seconds:

```
    # [hot] FIB update period, seconds
    # Default value - every 15 seconds
#router_fib_refresh=15
```

It makes no sense to set this parameter too small (less than 5 seconds).

The maximum size of neighbor cache (ARP cache) and the timeout for updating the records of this cache is set by the parameters:

```
    # [cold] Max size of ARP cache (neighbor cache for IPv6)
    # Default value - 1024, max = 32K
#router_arp_cache_size=1024
```

The Stingray SG contains separate neighbor caches for IPv4 and IPv6, each of size `router_arp_cache_size`.

The Stingray SG itself does not send ARP requests for obsolete cache entries. Instead, it relies on updates from the Linux kernel: Stingray monitors the ARP responses coming to the TAP-interfaces subnet address, and updates its ARP cache in accordance with these responses. The same applies to IPv6 (monitoring ICMPv6 neighbor discovery).

The router runs in a separate thread on a separate CPU core. At start Stingray sets parameters of this thread by default, which can be changed by parameters:

```
    # [cold] Adding to the priority of the router's service flow (increasing
the priority)
#router_sched_add_prio=0

    # [cold] Router thread binding kernel, -1 - autodetection
#router_bind_core=-1
```

Do not change these parameters unless absolutely necessary; it is better to let SSG determine the

core and priority itself. For example, explicitly specifying a core for router `router_bind_core` may be useful if there are not enough cores; then you can explicitly bind router to a core to which some other service thread (ajb, ctl) is bound.


Never bind the router to the kernel of a worker thread or dispatcher!

### VRF Configuration

The `router_netns` and `router_kernel_table` settings define the default VRF and are used when describing other VRFs as default values for the corresponding VRF parameters.

When preparing the fastDPI for router mode, the administrator needs to create the necessary netns and TAP interfaces to divert traffic to route-demons. In the `fastdpi.conf` configuration the ready (existing) netns and TAP-interfaces are specified, only in this case the SSG will start.

Each VRF is specified by a separate new section in the router configuration:

#### Routing table description (VRF)

```
router_vrf {
        id=
        netns=
        kernel_table=
    neighbor_cache=
        }
```

```
router_default_vrf=
```

**`id`** — string, unique ID of VRF. For a subscriber in Radius VSA authorization a VRF can be specified - it is this the ID. The maximum size is 15 characters.
**`netns`** — name of netns from which VRF is calculated. If not specified, it is considered equal to the `router_netns` option. This netns contains peer TAP interfaces for this VRF.
**`kernel_table`** — number of kernel routing table for this VRF. If not specified, it is considered equal to the `router_kernel_table` option.
**`router_default_vrf`** — string, ID default VRF. Default VRF is used for subscribers that do not have the `vrf_id` property.
**`neighbor_cache`** — string, the name of the ARP cache for this default VRF, each VRF has its own ARP/Neighbor cache isolated from the others. If you want several different VRFs to share a common ARP/Neighbor cache, you should specify the same value of the `neighbor_cache` option in the description of these VRFs.

Parameters for this VRF:

**`max_ip4_route_count`** — maximum number of IPv4 routes.
**`max_ip6_route_count`** — maximum number of IPv6 routes.
**`multipath_page`** — maximum number of pages for multi-path route distribution. One page can hold 64 different multi-path routes. *One group can be placed on several pages (if the number of routes in*

*the group is more than 64, if there are no restrictions on the number of routes in the router daemon)*
**`fib_refresh`** — FIB refresh interval.
**`arp_cache_size`** — ARP cache size.

These parameters define the required memory and refresh frequency for this VRF. Default values can be specified here (taken from global options) or overridden.

- If there are **no** `router_vrf` sections in the configuration, the mode of operation remains the same: back compatibility, meaning that one VRF is described in SSG, which is the default.
- If the configuration **has** `router_vrf` sections, the mode of operation is VRF support. In this case exactly one VRF must be default, i.e. the option `router_default_vrf=id of default` VRF must be set.

**`bras_vrf_isolation`** — VRF isolation. L2 BRAS does not isolate subscribers from different VRFs: If this mode is enabled (1), subscribers from different VRFs will be isolated from each other. Default value: 0. When this option is enabled:

1. subscriber's ARP to gateway — processed by fastDPI only if the subscriber and gateway are in the same VRF
2. gateway ICMP ping — processed by fastDPI only if the subscriber and the gateway are in the same VRF.
3. local interconnect — applies only if both subscribers are in the same VRF

**`bras_egress_filtering`** — filtering of outgoing traffic subs→inet (bitmask). By default it is disabled (0). When enabled, the packet will be dropped if the following conditions are met:

1. the subscriber's IP address (srcIP) is unknown to L2 BRAS
2. bras_term_by_as = 0
3. subscriber's AS is not local

#### Description of router_device

A VRF id is added to the `router_device` description. Description of one router interface:

```
router_device {
    device=
    tap=
    peer=
    vrf=
    subnet=
    subnet=
    subnet6=
    subnet6=
}
```

**`device`** — name of the device from `in_dev/out_dev`.
**`tap`** — name of the TAP interface for the device (`default='DPI' + device`).
**`peer`** — name of the paired TAP interface in netns for VRF (`default='rib' + device`).
**`vrf`** — identifier of the VRF where the peer is located. If not specified, it is considered equal to the default VRF.

**subnet** and **subnet6** — are subnets diverted from the general traffic to the TAP-device. At least one `subnet` or `subnet6` parameter must be set for `router_device`!

**subnet** — listing of IPv4 subnets diverted from the general traffic to the TAP device.
**subnet6** — listing of IPv6 subnets diverted from the general traffic to the TAP device.

Each IPv4 and IPv6 subnet is specified separately in the `subnet` and `subnet6` parameter. There can be a maximum of 16 `subnet` and 16 `subnet6` parameters for one `router_device`.

Subnets for the same port must not overlap.
For example, BGP1 from VRF1 is subnet 10.20.30.0/24, BGP2 from VRF2 is subnet 10.20.30.0/20.

#### VRF subscriber management

VRF ID is received by fastDPI at authorization in the new service 254.

`VasExperts-Service-Profile = "254:VRF_ID"`

Here “VRF_ID” is the VRF identifier.

### Example of VRF configuration in SSG

```
in_dev=0b-00.0
out_dev=13-00.0

scale_factor=1

ctrl_port=29000
ctrl_dev=lo

federal_black_list=0
black_list_sm=1
black_list_redirect=http://operator.com/blockpage.html

num_threads=1

router=1
router_vrf {
        id=ROUTER
        netns=router
        kernel_table=100
        neighbor_cache=shared
}

router_vrf {
        id=ROUTER2
```

```
        netns=router
        kernel_table=101
        neighbor_cache=shared
}
router_device {
        device=13-00.0
        vrf=ROUTER
        tap=dpi
        peer=rib
        subnet=224.0.0.5/30
        subnet=192.168.123.69/32
}
router_device {
        device=13-00.0
        vrf=ROUTER2
        tap=dpi
        peer=rib
        subnet=192.168.123.70/32
}
router_subs_announce=6

enable_auth=1
auth_servers=127.0.0.1%lo:29002
bras_enable=1
bras_arp_ip=10.10.102.189
bras_arp_mac=00:0c:29:f5:85:47
bras_dhcp_mode=1
bras_dhcp_server=10.10.99.3%ens256;reply_port=67
bras_pppoe_enable=1
bras_pppoe_session=100
bras_ppp_auth_list=2,3,1

enable_acct=1
netflow=4
netflow_timeout=300
bras_pppoe_service_name=demoDPI
```

### Setting veth-Interface Names

The fastdpi.conf describes all TAP-interfaces associated with the devices:

```
# Description of one router interface
# WARNING! '{' must be on the same line as the router_device section name!
router_device {
        # Device name from in_dev/out_dev
    device=
        # TAP interface name for the device (default='dpi' + device)
    #tap=
        # Name of the paired TAP interface in netns for the device
(default='rib' + device)
```

```
    #peer=
# WARNING! '}' must be on a separate line!
}
```

For example, for this configuration

```
in_dev=41-00.0
out_dev=41-00.1
```

where only `out_dev` is connected to the router, the description would be:

```
in_dev=41-00.0
out_dev=41-00.1

router_device {
        # Device name from in_dev/out_dev
    device=41-00.1
        # TAP interface name for the device (default='dpi' + device)
    tap=tap41
        # Name of the paired TAP interface in netns for the device
(default='rib' + device)
    peer=bgp41
}
```

It is possible not to specify the names of the `tap` and `peer` interfaces (default names are implied in this case), but the `router_device` should be described:

```
in_dev=41-00.0
out_dev=41-00.1

   # TAP for out_dev:
router_device {
    device=41-00.1
}

   # TAP for in_dev
router_device {
    device=41-00.0
}
```

In this case the TAP interface names are assumed to be as follows:

- for `in_dev=41-00.0`: `dpi41-00.0` on the SSG side, `rib41-00.0` on the BIRD side
- for `out_dev=41-00.1`: `dpi41-00.1` on the SSG side, `rib41-00.1` on the BIRD side

## LAG Support

Stingray Service Gateway 10.1 adds support for link aggregation in the router.

For aggregated channels, packets that need to be diverted to a TAP interface can go to any device

that is part of a LAG. To avoid creating the same TAP interface for each device in a LAG, the router takes into account which devices are included in the LAG and for all such devices does a diversion of traffic of the specified subnets to the TAP (to the BIRD daemon).

Each LAG is defined by a separate section in fastdpi.conf which lists all the devices included in the LAG:

```
    # In/out devices, combined in a LAG
in_dev=01-00.0:02-00.0
out_dev=01-00.1:02-00.1

    # Describing the LAG towards the inet
lag {
       # Optional LAG name, used only for log output
    name=inet
       # Each device included in a LAG is described by a separate device
parameter
    device=01-00.1
    device=02-00.1
}

    # Description of one router interface
router_device {
        # Device name from out_dev. Only for this device create a veth pair
of TAP interfaces
    device=01-00.1
        # TAP interface name for the device (default='dpi' + device)
    tap=tap0
        # Name of the paired TAP interface in netns for the device
(default='rib' + device)
    peer=peer0
        # Subnets diverted from total traffic to the TAP device (example)
    subnet=10.0.10.0/26
    #...other subnets...
}
```

With this description, traffic to TAP `tap0` will be diverted from both `01-00.1` and `02-00.1` devices specified in the `lag` section, according to the rules (subnets) specified for `01-00.1` in `router_device`.

The `lag` section must contain at least two devices, and all devices must be of the same direction (either all facing inet or all facing subs). One device may only be in one LAG (or not in any LAG at all). If the router works both inet and subs direction (e.g. BGP on the inet side and OSPF inside the network, subs side), two `lag` sections are described:

```
# LAG towards inet
lag {
    name=inet
    device=01-00.1
    device=02-00.1
}
```

```
# LAG towards subs
lag {
    name=subs
    device=01-00.0
    device=02-00.0
}
```

and a separate `router_device` section is configured for each.

A maximum of 10 different lag sections can be configured in total.

The `lag` section in fastdpi.conf is a cold parameter, requiring a fastdpi restart when the LAG description is changed.

## Multi-path routing (ECMP)

Support for multi-path routing (ECMP) is added in Stingray Service Gateway 10.2.

SSG performs traffic balancing (round-robin) at the flow level for all routes from the multi-path. Balancing at the flow level means that a specific flow will be assigned to one of the routes from the multi-path, and the selected route will not change until the end of this flow (unless the composition of the multi-path is changed by external events from the routing daemon).

No configuration is required to enable multi-path in SSG. ECMP support is enabled in the configuration parameters of the routing daemon. For example, in BIRD, ECMP support is enabled by specifying `merge paths yes` in the `kernel` protocol, see https://bird.network.cz/?get_doc&v=20&f=bird-6.html#ss6.6.

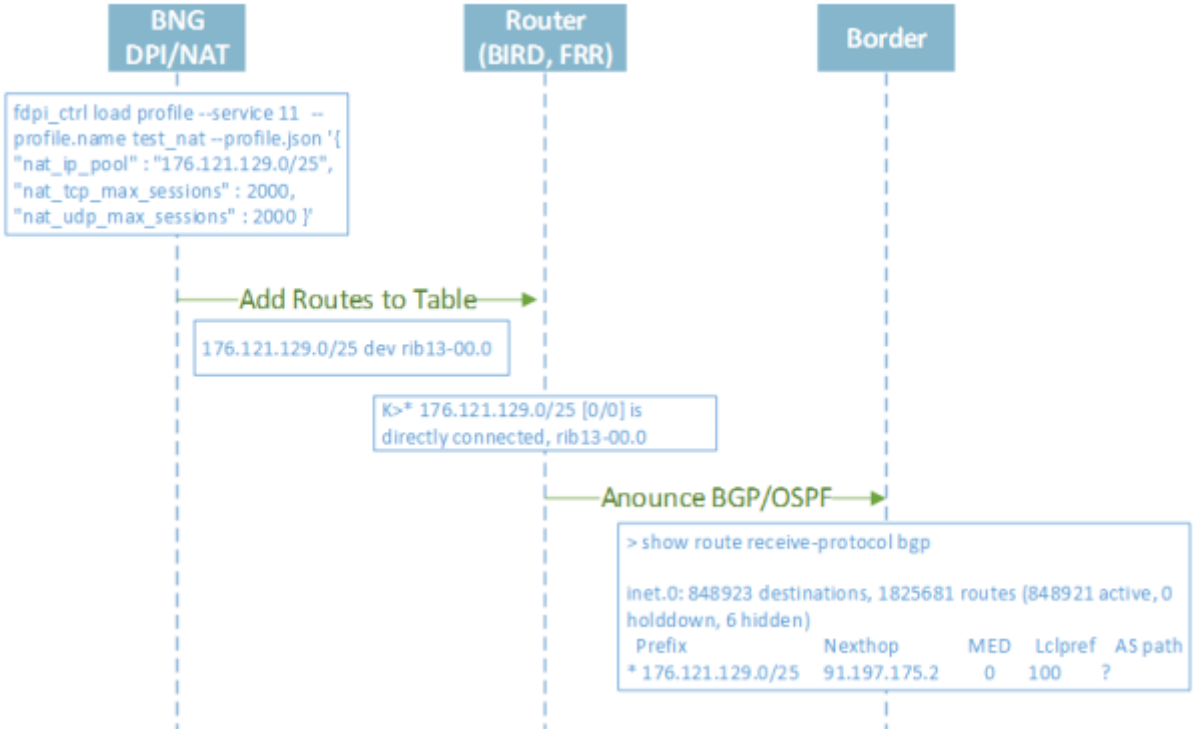
## Specifics of addresses announcement

### Subscriber announcements and NAT pool

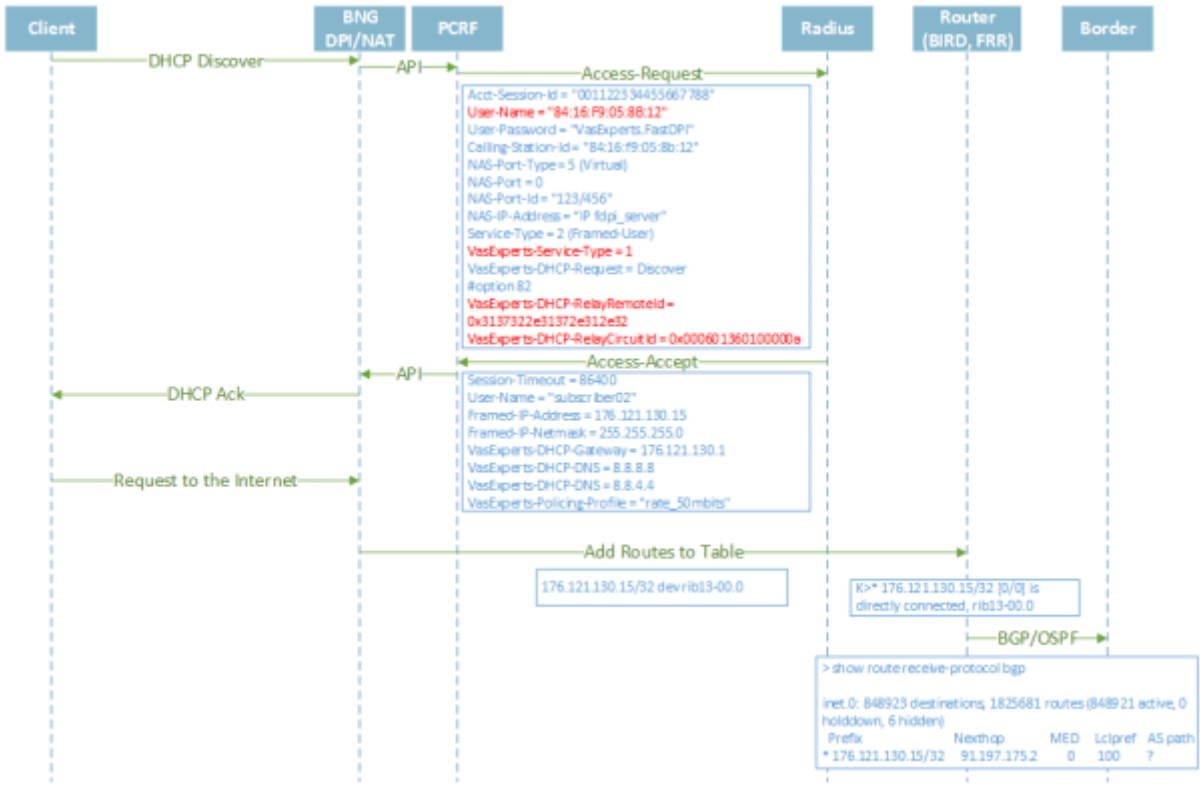
Announcing subscriber addresses is enabled by a parameter in fastdpi.conf:

```
    # [cold] Subscriber address announcement flags
    # Bit mask
    # Values:
    #   1 - announce the subscriber's address towards subs
    #   2 - announce the subscriber's address towards the inet (if the
subscriber does not have NAT connected)
    #   4 - announce NAT subnets towards the inet
    #   8 - announce subscriber gateways (direction is set by 1 and 2 flags)
    # Default value: 0 - not to announce anything
#router_subs_announce=0
```

```
    # [hot] Metric for subscriber address announcements
    # Default value = 32
#router_subs_metrics=32
```




NAT public address subnets are announced only towards the inet when the SSG starts and when adding/removing/modifying NAT profiles.




Subscriber addresses can be announced both towards inet and subs side. But if the subscriber's IP address is in the private address range and the service 11 is assigned to it, the subscriber's address is

not announced towards the inet (so be careful when defining private address ranges). The announcement is made to the BIRD routing table for all TAP-devices of the allowed direction, then BIRD picks up the changes and announces them to the appropriate protocols according to its configuration.

### Announcement of L3 subscriber addresses

In SSG 10.2, the following algorithm of announcing IP-addresses of L3-subscribers operates:

- by default, the address announcement is performed for successful authorization (accept) and is NOT performed for unsuccessful authorization (reject)
- fastdpi.conf parameter `auth_announce_reject` allows to globally allow announcements for reject:

```
# [hot] To announce (1) or not (0) the unauthorized (Reject) subscribers
# Default value - 0 (do not announce)
#auth_announce_reject=0
```

- added new Radius-attribute `VasExperts-Route-Announce`: value 0 - do not announce subscriber's address, value 1 - announce. This attribute has higher priority than `auth_announce_reject`.

To sum up, when deciding whether to announce the IP-address of the subscriber or not, the following algorithm applies:

- If the `VasExperts-Route-Announce` attribute is explicitly set in the Radius response (Accept or Reject), the announcement will be made for `VasExperts-Route-Announce=1` and will NOT be made for `VasExperts-Route-Announce=0`
- If the `VasExperts-Route-Announce` attribute is not set in the Radius response:
  - For Access-Accept (successful authorization), the subscriber's IP address will be announced
  - For Access-Reject (unsuccessful authorization), the IP address is announced only if fastdpi.conf-parameter `auth_announce_reject=1`


This algorithm is applied at the stage of calling the router announcement function; the router's fastdpi.conf parameter `router_subs_announce` regulates, whether or not the subscriber's IP address is announced, and in which direction.

This algorithm is also used for ARP authorization and GTP authorization, that is, for all types of authorization, where the subscriber's IP address is known at the time of the authorization call and does not change. For other types of L2 authorization (DHCP, PPP, - all where the IP address is explicitly **allocated**), he `VasExperts-Route-Announce` attribute is not taken into account, the announcement occurs after the IP address is allocated to the subscriber.

## Root Daemon Configuration (BIRD, FRR, etc.)

The settings of the Root daemon (BIRD, FRR, etc.) and SSG must be consistent: the root daemon must create a kernel route table with the number given by the `router_kernel_table` parameter.

Supported Routing Daemons:

1. BIRD: official documentation. BIRD version 2 and higher is supported. Version 1 is not supported.
2. FRRouting: official documentation
3. QUAGGA: official documentation
4. Juniper CRPD: official documentation

The example shows a special case, for more information please refer to the documentation of the specific root daemon. In the context of VRF Lite, BIRD suits better since the SSG runs on the Linux architecture.

### BIRD configuration example

```
# Specify additional features to be tested:

# Any public address
function is_public() {
    if net !~ [ 10.0.0.0/8+, 172.16.0.0/12+, 192.168.0.0/16+, 100.64.0.0/10+
] then
        return true;
    return false;
}

# Any private address
function is_private() {
    if net ~ [ 10.0.0.0/8+, 172.16.0.0/12+, 192.168.0.0/16+, 100.64.0.0/10+
] then
        return true;
    return false;
}

# Default gateway
filter default_gw {
    if net ~ [0.0.0.0/0] then
        accept;
    reject;
}

# Specify the filters:
# Routes that are not acquired from other routing protocols (and the prefix
is not /32)
filter exclude_external_routes {
    if (source = RTS_INHERIT) && (net.len != 32) then
        accept;
    reject;
}
```

```
# Exclude routes from other routing protocols, public subnets, private
subnets – not /32
filter exclude_ext_1_ip {
    if (source = RTS_INHERIT) && (is_public() || (is_private() && (net.len
!= 32))) then
        accept;
    reject;
}


log "/var/log/bird.log" all;
router id 192.168.123.65;

debug protocols all;

# Describe the tables
ipv4 table grt;
ipv4 table bird00;
ipv4 table bird01;

protocol device {
}


protocol direct {
    disabled;        # Disables by default
    ipv4;            # Connection to the default IPv4 table
    ipv6;            # ... and to the default IPv6 table
}

# Describe kernel "protocols"
protocol kernel kernel_grt {
    ipv4 {            # Connect the protocol to the IPv4 table
          table grt;
          import all;    # Import to table, default is import all
          export all;    # Export to protocol. Default value - no export
    };
    scan time 5;
    learn;            # Examine incoming routes from the kernel table
    kernel table 99;    # Kernel table to synchronize with (default: main)
}
protocol kernel kernel_bird00 {
    ipv4 {            # Connect the protocol to the IPv4 table over the link
          table bird00;
          import all;    # Import to table, default is import all
          export all;    # Export to protocol. Default value - no export
    };
    scan time 5;
    learn;            # Examine incoming routes from the kernel table
    kernel table 100;    # Kernel table to synchronize with (default: main)
}
```

```
protocol kernel kernel_bird01 {
    ipv4 {            # Connection of the protocol to the IPv4 table
          table bird01;
          import all;    # Import to table, default is import all
          export all;    # Export to protocol. Default value - no export
    };
    scan time 20;
    learn;            # Examine incoming routes from the kernel table
    kernel table 101;    # Kernel table to synchronize with (default: main)
}

# Another instance for IPv6 that skips default settings
protocol kernel {
    ipv6 { export all; };
}

protocol static {
    ipv4;            # IPv4 channel with default parameters again

}

# OSPF protocols (each instance with its own table)
protocol ospf v2 ospf_grt {
        tick 1;
        rfc1583compat no;
    stub router no;
    ecmp yes limit 16;
        ipv4 {
        table grt;
                import all;
        export all;
        };
        area 0.0.0.0 {
                networks {
                        192.168.123.64/30;
                };
                interface "rib.102" {
                        cost 1;
                        rx buffer large;
            type broadcast;
            authentication none;
               };
         };
};
protocol ospf v2 ospf_bird01 {
        tick 1;
        rfc1583compat no;
    ecmp yes limit 16;
        ipv4 {
        table bird01;
```

```
                import all;
                export all;
        #export filter exclude_ext_1_ip;
        };
        area 0.0.0.0 {
                networks {
                        192.168.123.68/30;
                };
                interface "rib.202" {
                        cost 1;
                        rx buffer large;
            type broadcast;
            authentication none;
               };
         };
}

# Describe routing "protocols" that are designed to "flip" routes between
tables (using filters)
protocol pipe grt_bird00 {
    table grt;
    peer table bird00;
    import all;
    export filter default_gw;
}
protocol pipe grt_bird01 {
    table grt;
    peer table bird01;
    import all; # filter exclude_ext_1_ip;
    export all; #filter default_gw;
}
```

## Router Troubleshooting

SSG Router for debugging purposes can record traffic from BIRD to pcap:

```
    # [hot] Recording pcap from the router's TAP interfaces
    # Note: You can also record traffic with the tcpdump utility (specify
the TAP interface name).
    #       But the problem is that tcpdump does not work with interfaces in
DOWN mode,
    #       hat is, tcpdump cannot record traffic
    #       when the interface goes from DOWN to UP.
    # AP interface names with ';' or 'all' (record from all)
    # For each TAP interface, a separate pcap file named
tap_<interface_name>_xxx.pcap is created
    # in the directory specified by the ajb_udpi_path parameter (by default
/var/dump/dpi)
#router_tap_pcap=all|the list of TAP interfaces separated by ';'
```

```
    # [hot] Direction of packets for pcap recording from TAP interfaces
    # Values:
    # 1 - TAP -> inward (packets from the TAP interface)
    # 2 - outward -> TAP (packets towards the TAP interface)
    # 0 or 3 - all directions
#router_tap_pcap_dir=0

    # [hot] TAP pcap rotation interval, seconds
    # 0 - is taken from the ajb_udpi_ftimeout parameter (ajb_udpi_ftimeout
is set in minutes)
#router_tap_pcap_rotate=0
```

You can also enable recording to pcap the data exchange with the kernel (rtnetlink):

```
    # [hot] To record rtnetlink messages в pcap or not
    # 0 - recording disabled
    # 1 - recording enabled
    # Prefix of pcap files = "rtnl"
#router_rtnl_pcap=0
```

Moreover, if packets are recorded to pcap by address mask (`ajb_save_ip`), the router will also record the resulting packet to pcap after routing is applied. That is, there will be two entries in pcap for one incoming packet: the first entry is the original packet, the second is the sent packet.

### Tracing

```
    # [hot] Trace flags of different parts of the router
    # 0x0001          Transit of rtnetlink FSM processing states (BIRD
alerts)
    # 0x0002          FSM events of rtnetlink handler (BIRD alerts)
    # 0x0004          data dump rtnetlink
    # 0x0010          RIB trace (route info base construction)
    # 0x0020          Route table trace
    # 0x0040          FIB trace
    # 0x0080          ARP cache trace
    # 0x0100          TAP trace
    # 0x0200          TAP pcap trace
    # 0x0400          announce trace
#router_trace=0
```

## CLI commands

Stingray has a set of CLI commands to view the current router status. For a complete list of commands see:

```
fdpi_cli help router
```

RIB and FIB dump commands output a lot of data, because these structures can contain hundreds of thousands of records in case of BGP full view. Therefore, when calling these commands we advise to redirect the output to the file

Also keep in mind that the routing table for BGP, OSPF, etc. is built by BIRD, which has its own command line utility `birdc` and its own configuration file with a developed system of commands for filtering, setting static routes etc.

In addition, the standard Linux `ip` utility gives you full control over the kernel route table. When using the `ip` utility, remember to specify the correct netns (`router_netns`) and routing table number (`router_kernel_table`).