Содержание

20 Router

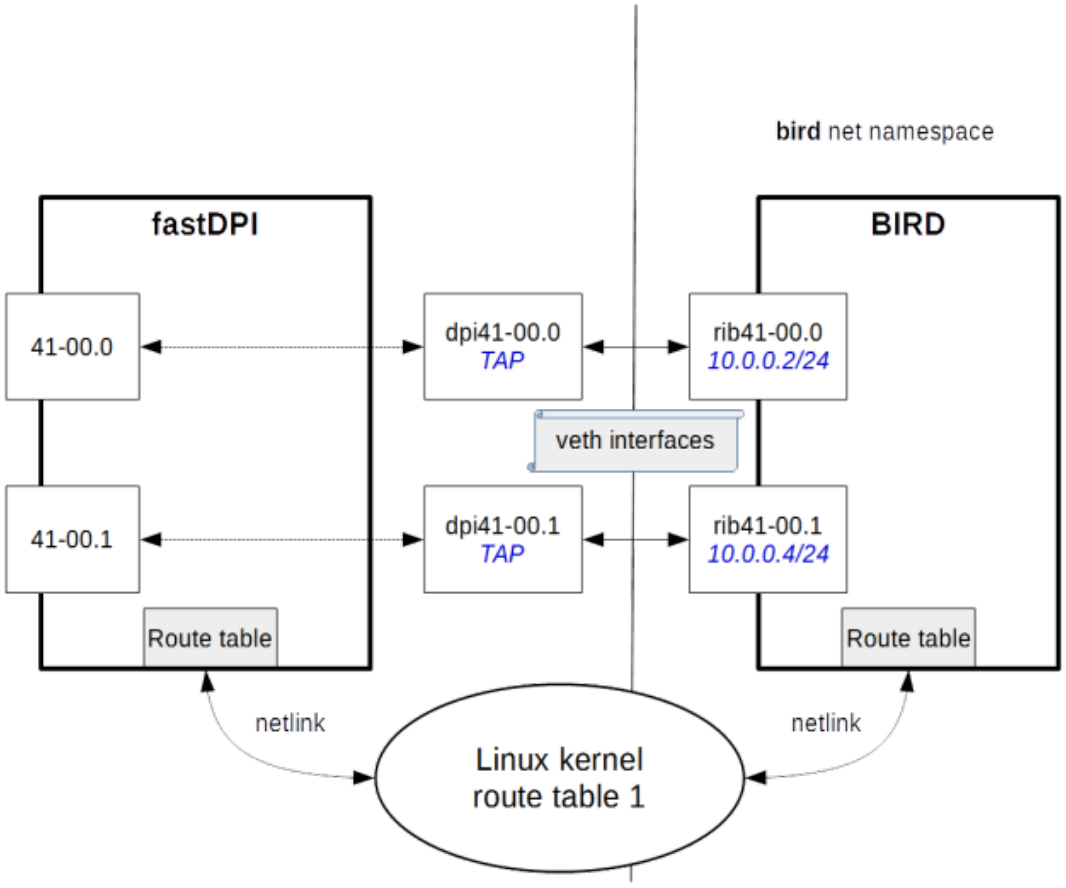
This is a provisional description, and may be substantially modified in the future based on test results

The router only works with the CentOS-8 version of Stingray Service Gateway, and only in L2 BRAS mode

<html> </html>

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.

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 - 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 new one. By doing this, the worker threads will meet any changes that have occurred in the routing table.

System Requirements

Router mode in SSG requires quite a lot of memory, especially with BGP full view. Plus, memory is required for the BIRD daemon that builds the routing table via BGP, OSPF, etc. Router mode (especially BGP full view) requires at least 4-8G additional memory.

Setting veth-nterface 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

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

```
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
}
```

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.

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.

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.

Root Daemon Configuration (BIRD, FRR, etc.)

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

fastDPI 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.

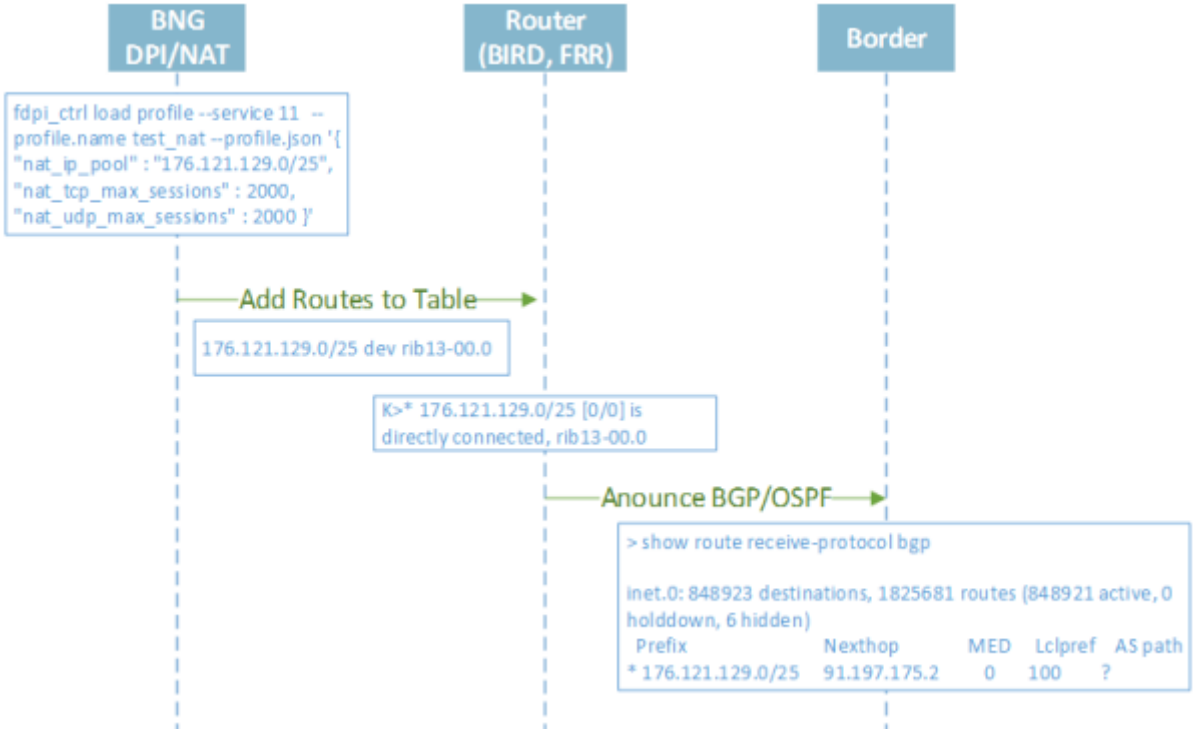
Subscriber announcements and NAT pool

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

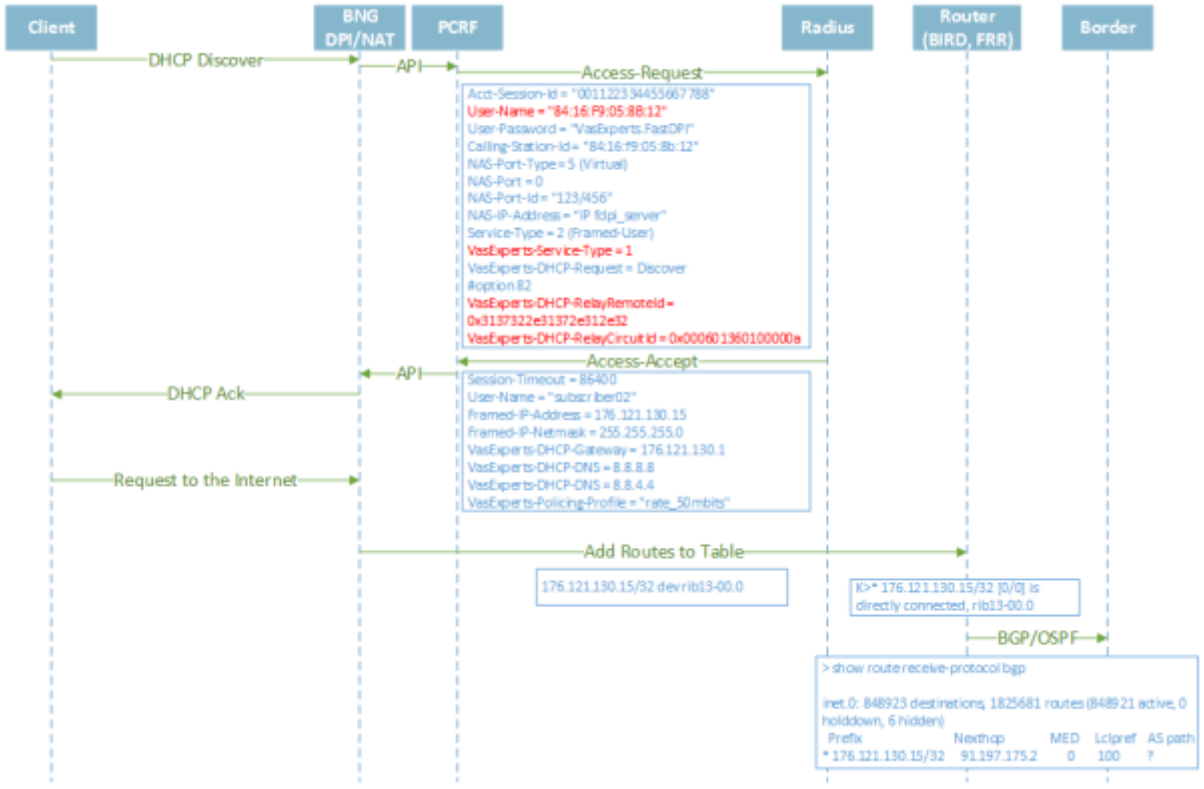
```
    # [cold] Subscriber address announcement flags
    # Bit mask
    # Values:
    #   1 - announce the subscriber's address towards subs
    #   2 - announce the subscriber's address towards the inet (if the
subscriber does not have NAT connected)
    #   4 - announce NAT subnets towards the inet
```

```
    #   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.

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!

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.

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`.