

Содержание

20 Роутер	3
Общее описание	3
Внутренняя архитектура роутера	4
Системные требования	5
Задание имен veth-интерфейсов	5
Конфигурирование подсетей TAP	6
Создание veth-интерфейсов	7
Поддержка LAG	9
Конфигурирование Роут-демона (BIRD, FRR, etc.)	11
Конфигурирование fastDPI	11
Обязательные параметры	11
Дополнительные параметры	14
Отладка Роутера	15
CLI-команды	16

20 Роутер



Это предварительное описание, в дальнейшем по результатам тестов может быть существенно изменено



Роутер работает только в версии СКАТ для CentOS-8, и только в режиме L2 BRAS

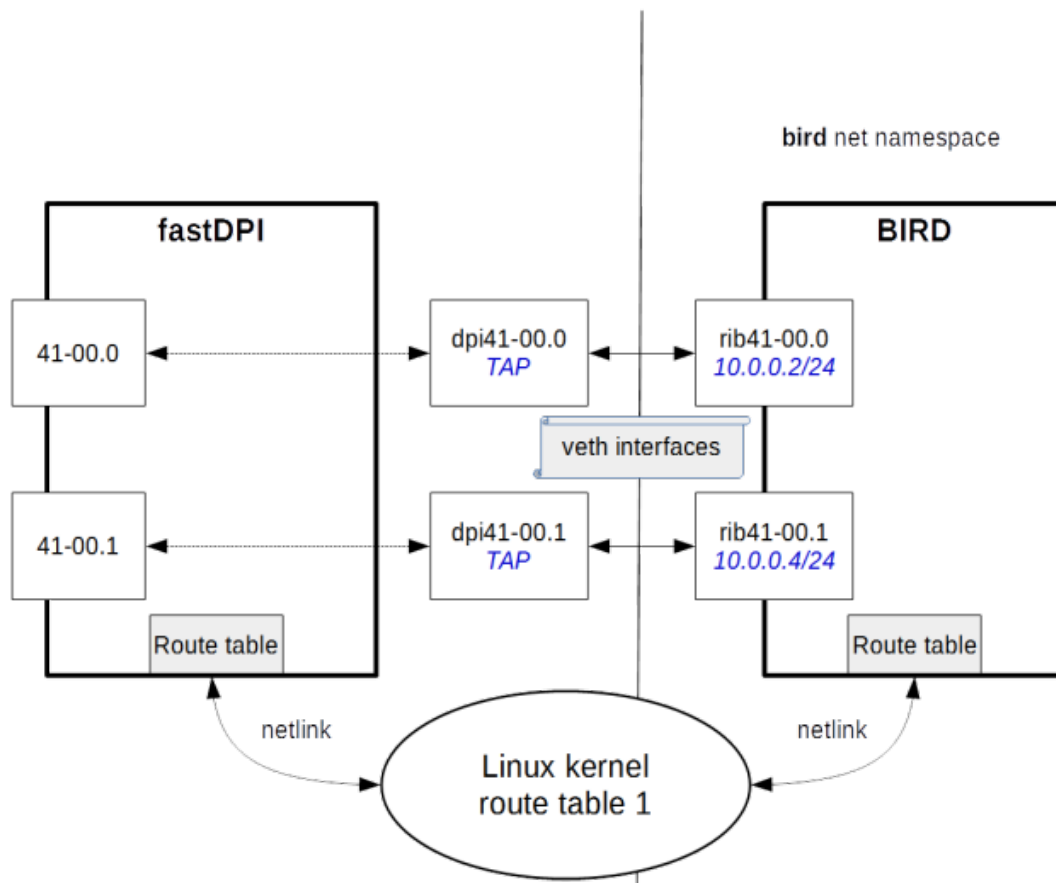
<html><div class="menu"></div></html>

Общее описание



Подробнее о решении: [Обзор Роутера](#)

Сам СКАТ не занимается построением таблицы маршрутизации. Он делегирует эту работу проверенным специализированным инструментам, в примере использован роут-демон BIRD. Роут-демон обрабатывает требуемые протоколы маршрутизации (BGP, OSPF и пр.) и по ним строит общую таблицу маршрутизации, которую загружает в ядро. СКАТ по этой таблице осуществляет маршрутизацию пакетов.



Вместо BIRD можно использовать любой другой демон, строящий таблицу маршрутизации в ядре Линукс, например, [FRRouting](#), так как SKAT для вычитывания таблицы маршрутизации использует только стандартный интерфейс Линукс и поэтому совместим с любым демоном.



В будущих версиях в целях экономии памяти возможно появление опциональных специализированных API для связи с тем или иным демоном, чтобы миновать построение kernel route table и взаимодействовать с демоном напрямую, минуя ядро.

Так как BIRD строит таблицу маршрутизации в ядре ОС, то, во избежание применения этих правил самим Линукс-сервером, роут-демон BIRD работает в отдельном net namespace (на схеме это bird netns). Пакеты протоколов маршрутизации поступают на in/out-девайсы SKATa в общем трафике. Для каждого in/out-девайса создается veth-пара "теневых" интерфейсов с predetermined именами: dpi-интерфейс veth-пары работает как TAP-интерфейс, rib-интерфейс - как обычный системный интерфейс в netns BIRD'a.

Внутренняя архитектура роутера

Данные из kernel route table вычитываются (rtnetlink) в RIB роутера. RIB - это префиксное дерево, его удобно модифицировать по событиям изменения (удаления/добавления записей) route table ядра. Но использовать RIB в маршрутизации нельзя, так как он не поддерживает

многопоточный доступ со стороны рабочих потоков (требуется блокировка, что неприемлемо). Поэтому в СКАТ RIB живет в потоке роутера и недоступен рабочим потокам.

Рабочие потоки используют FIB. Эта структура заточена на многопоточный поиск (LPM - longest prefix match), но не предназначена для модификаций (удаления/добавления новых записей). FIB можно только построить с нуля по RIB и затем использовать для LPM. Поэтому в СКАТ существуют два FIB - текущая (которая сейчас используется для роутинга рабочими потоками) и "будущая". СКАТ раз в `router_fib_refresh` секунд проверяет, не было ли изменений в RIB с момента построения текущей FIB. Если изменения были, СКАТ строит (в потоке роутера) новую FIB на месте "будущей", а затем переключает текущую FIB на новую. Тем самым рабочие потоки увидят все изменения, которые произошли в таблице маршрутизации.

Системные требования

Режим роутера в СКАТ требует довольно много памяти, особенно при BGP full view. Плюс память требуется демону BIRD, строящему таблицу маршрутизации по BGP, OSPF и пр. Для режима роутера (особенно для BGP full view) потребуется дополнительно как минимум 4G - 8G памяти.

Задание имен veth-интерфейсов

В `fastdpi.conf` описываются все TAP-интерфейсы, связанные с девайсами:

```
# Описание одного интерфейса роутера
# ВНИМАНИЕ! '{' должен быть на той же строке, что и имя секции router_device!
router_device {
    # Имя девайса из in_dev/out_dev
    device=
    # Имя TAP-интерфейса для девайса (default='dpi' + device)
    #tap=
    # Имя парного TAP-интерфейса в netns для девайса (default='rib' + device)
    #peer=
# ВНИМАНИЕ! '}' должен быть на отдельной строке!
}
```

Например, для такой конфигурации

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

где к роутеру подключен только `out_dev`, описание будет такое:

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

router_device {
    # Имя девайса из in_dev/out_dev
```

```

device=41-00.1
# Имя TAP-интерфейса для девайса (default='dpi' + device)
tap=tap41
# Имя парного TAP-интерфейса в netns для девайса (default='rib' + device)
peer=bgp41
}

```

Можно не задавать имена tap и peer интерфейсов (в этом случае подразумеваются имена по умолчанию), но описать router_device нужно:

```

in_dev=41-00.0
out_dev=41-00.1
# TAP для out_dev:
router_device {
    device=41-00.1
}

# TAP для in_dev
router_device {
    device=41-00.0
}

```

В этом случае предполагаются такие имена TAP-интерфейсов:

- для in_dev=41-00.0: со стороны СКАТа dpi41-00.0, со стороны BIRD rib41-00.0
- для out_dev=41-00.1: со стороны СКАТа dpi41-00.1, со стороны BIRD rib41-00.1

Конфигурирование подсетей TAP

Для каждого router_device обязательно должно быть указано, какие подсети отводятся на TAP (фактически, это отведение пакетов протоколов маршрутизации на BIRD). СКАТ будет выделять из общего трафика на девайсе обращения к этим подсетям и направлять все такие пакеты на соответствующий TAP-интерфейс.

Подсети задаются параметрами subnet (для IPv4) и subnet6 (для IPv6) в описании router_device. Каждая подсеть задается отдельным параметром subnet/subnet6. Всего в описании router_device может быть до 16 разных subnet параметров и до 16 разных subnet6. Например, такая конфигурация

```

router_device {
    # Имя девайса из in_dev/out_dev
    device=41-00.1
    # Имя TAP-интерфейса для девайса (default='dpi' + device)
    tap=tap41
    # Имя парного TAP-интерфейса в netns для девайса (default='rib' + device)
    peer=bgp41

    # Какие IPv4-подсети отводим на TAP
    subnet=10.0.2.0/30
}

```

```
subnet=8.8.8.0/29
```

```
# Какие IPv6-подсети отводим на TAP
```

```
subnet6=2001::1/124
```

```
# link-local адрес интерфейса, с которым взаимодействует bird
```

```
subnet6=fe80::82d:cff:fe5f:9453/128
```

```
}
```

задает две IPv4-подсети для девайса 41-00.1, которые будут отводиться на TAP-интерфейс tap41, и одну IPv6-подсеть плюс link-local адрес интерфейса, с которым взаимодействует bird.



Если используется IPv6, следует учитывать, что в IPv6 большую роль играют link-local адреса, которые также должны быть указаны в параметрах subnet6

OSPF использует мультикастные адреса 225.0.0.5 и 224.0.0.6, поэтому если на router_device используется протокол OSPF, эти адреса тоже следует задать в описании router_device:

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

```
# OSPF multicast
```

```
subnet=225.0.0.5/32
```

```
subnet=225.0.0.6/32
```

```
}
```



В параметре router_device должна быть указана хотя бы одна IPv4 или IPv6-подсеть

Создание veth-интерфейсов



Все, что описано в данном разделе, - создание veth-интерфейсов, запуск BIRD и пр. - должно быть задано в скриптах загрузки системы и выполняться **до** запуска fastdpi.

Пусть у нас в fastdpi.conf заданы следующие девайсы:

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

Предположим, что нам требуется настроить в BIRD протокол BGP для uplink, то есть на девайсе 41-00.1.



Теневые veth-интерфейсы нужно создавать для каждого in/out-девайса, в трафике которого идут пакеты протоколов маршрутизации, то есть которые требуют конфигурирования в BIRD. Если девайс не участвует в маршрутизации (как `in_dev=41-00.0` в этом примере), для него не надо создавать veth-пару

Чтобы перенаправить BGP-трафик с `41-00.1` на BIRD, который работает в `bird netns`, нам нужно создать veth-пару теневых для `41-00.1` интерфейсов.

Создаем `bird netns` (имя `bird` здесь выбрано произвольно, вы можете использовать другое имя `netns`), в котором будет работать BIRD:

```
ip netns add bird
```

Создаем veth пару:

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

rib-интерфейс должен иметь IP-адрес (и IPv6, если IPv6 поддерживается). Этот адрес будет адресом BGP-пира для BGP-соседа.

```
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
# включаем ARP на интерфейсе
ip netns exec bird ip link set dev rib41-00.1 arp on

# set tx checksum offload off - выключаем расчет контрольной суммы на интерфейсе
# замечено, что расчет CRC на интерфейсе может быть некорректным (по крайней мере, на
некоторых сборках ядра CentOS-8)
ip netns exec bird ethtool -K rib41-00.1 tx off
```



IP-адрес rib-интерфейсов должны отличаться от IP-адреса SKAT, задаваемого параметрами `bras_arp_ip` и `bras_ipv6_address`. Более того, во избежание непонятных ситуаций адреса `bras_arp_ip` и `bras_ipv6_address` не должны входить ни в какую подсеть, отводимую на TAP-интерфейсы

dpi-интерфейс **не должен** иметь ни IPv4, ни IPv6 адреса, так как в SKAT он используется как TAP-интерфейс и наличие на нем адресов не требуется (более того, даже может мешать, если сам интерфейс начнет "излучать" пакеты):

```
ip link set dev dpi41-00.1 arp off
# Disable IPv6 on dpiXXX interfaces (чтобы не было даже link-local адреса)
echo 1>/proc/sys/net/ipv6/conf/dpi41-00.1/disable_ipv6
```

Наконец, поднимаем все созданные интерфейсы:

```
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
```

Не забываем про 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'
```

Не забывайте, что BIRD должен быть запущен в bird netns:

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



Состоянием линков veth-интерфейсов управляет SKAT: если девайс 41-00.1 link down, то SKAT переведет в link down veth-интерфейсы этого девайса; как только линк 41-00.1 поднимется, SKAT переведет veth-интерфейсы в link up

Как быть с VLAN?



SKAT пересылает пакеты в rib-интерфейсы "как есть", без всякого преобразования. Это значит, что в случае наличия VLAN, нужно средствами Linux создать vlan-интерфейсы на rib-интерфейсе, а уже к этим vlan-интерфейсам привязать bird

В fastdpi.conf vlan-интерфейсы, созданные на rib-интерфейсе, не должны нигде фигурировать, - вы должны в качестве tap и peer указать два конца veth-пары.

MTU



SKAT **не** выставляет MTU на veth-интерфейсах. При конфигурировании veth-интерфейсов следует задать MTU штатными средствами Линукса.

Поддержка LAG

В SKAT 10.1 добавлена поддержка агрегации каналов в роутере.

Для агрегированных каналов пакеты, которые нужно отводить на TAP-интерфейс, могут прийти на любой девайс, входящий в LAG. Чтобы не создавать фактически одинаковые TAP-интерфейсы для каждого девайса из LAG, роутер учитывает, какие девайсы входят в LAG и для всех таких девайсов делает отвод трафика указанных подсетей на TAP (к демону BIRD).

Каждый LAG задается отдельной секцией в fastdpi.conf, в которой перечисляются все девайсы, входящие в LAG:

```
# Входные/выходные девайсы, объединенные в LAG
```

```

in_dev=01-00.0:02-00.0
out_dev=01-00.1:02-00.1

# Описываем LAG в сторону inet
lag {
    # Необязательное имя LAG, используется только для вывода в лог
    name=inet
    # Каждый девайс, входящий в LAG, описывается отдельным параметром device
    device=01-00.1
    device=02-00.1
}

# Описание одного интерфейса роутера
router_device {
    # Имя девайса из out_dev. Только для этого девайса делаем veth-пару TAP-
интерфейсов
    device=01-00.1
    # Имя TAP-интерфейса для девайса (default='dpi' + device)
    tap=tap0
    # Имя парного TAP-интерфейса в netns для девайса (default='rib' + device)
    peer=peer0
    # Подсети, отводимые из общего трафика на TAP-девайс (пример)
    subnet=10.0.10.0/26
    #...прочие подсети...
}

```

При таком описании отвод трафика на TAP tap0 будет происходить с обоих девайсов 01-00.1 и 02-00.1, заданных в секции lag, в соответствии с правилами (подсетями), заданными для 01-00.1 в router_device.

В секции lag должно быть указано не менее двух девайсов, причем все девайсы должны быть одного направления (либо все смотрят в inet, либо все смотрят в сторону subs). Один девайс может входить только в один LAG (или не входить вообще ни в какой). Если роутер работает как в сторону inet, так и в сторону subs (например, BGP на стороне inet и OSPF внутри сети, в сторону subs), то описываются две секции lag:

```

# LAG в сторону inet
lag {
    name=inet
    device=01-00.1
    device=02-00.1
}

# LAG в сторону subs
lag {
    name=subs
    device=01-00.0
    device=02-00.0
}

```

и для каждой конфигурируется отдельная секция router_device.

Всего возможно задать не более 10 различных lag-секций.

Конфигурирование Роут-демона (BIRD, FRR, etc.)

Настройки Роут-демона (BIRD, FRR, etc.) и СКАТ **должны быть согласованы**: роут-демон должен создавать kernel route table с номером, задаваемым параметром `router_kernel_table`.

Поддерживаемые роут-демоны:

1. [BIRD: официальная документация](#). Поддерживается BIRD версии 2 и выше. Версия 1 **не** поддерживается.
2. [FRRouting: официальная документация](#)
3. [QUAGGA: официальная документация](#)
4. [Juniper CRPD: официальная документация](#)

Конфигурирование fastDPI

Обязательные параметры

Чтобы включить функционал роутера, в `fastdpi.conf` необходимо активировать параметр

```
# [cold] Включение функционала роутера
# Булевый параметр:
# 0, false, off - функционал роутера отключен (default)
# 1, true, on - функционал роутера включен
# Не допускает изменения "на лету" через reload
router=1
```

Далее необходимо указать, в каком netns работает BIRD и номер таблицы маршрутизации ядра, которую он строит:

```
# [cold] Имя net namespace, в котором запущен BIRD
router_netns=bird
# [cold] Номер таблицы маршрутизации ядра, которую использует fastDPI
router_kernel_table=1
```

Также обязательно должны быть заданы следующие параметры BRAS, даже если никакой из режимов BRAS не включен:

```
# Виртуальный MAC-адрес СКАТ
bras_arp_mac=00:E0:ED:43:84:42
```

```
# Виртуальный IP-адрес СКАТ
bras_arp_ip=188.227.73.40
```

```
# Если используется IPv6, необходимо задать виртуальные IPv6-адреса:
```

```
# Задает глобальный IPv6 адрес СКАТа  
bras_ipv6_address=2098::1
```

```
# Задает IPv6 link-local адрес СКАТа (префикс FE80::/10)  
# Если данный параметр не задан явно, он вычисляется по bras_arp_mac  
#bras_ipv6_link_local
```

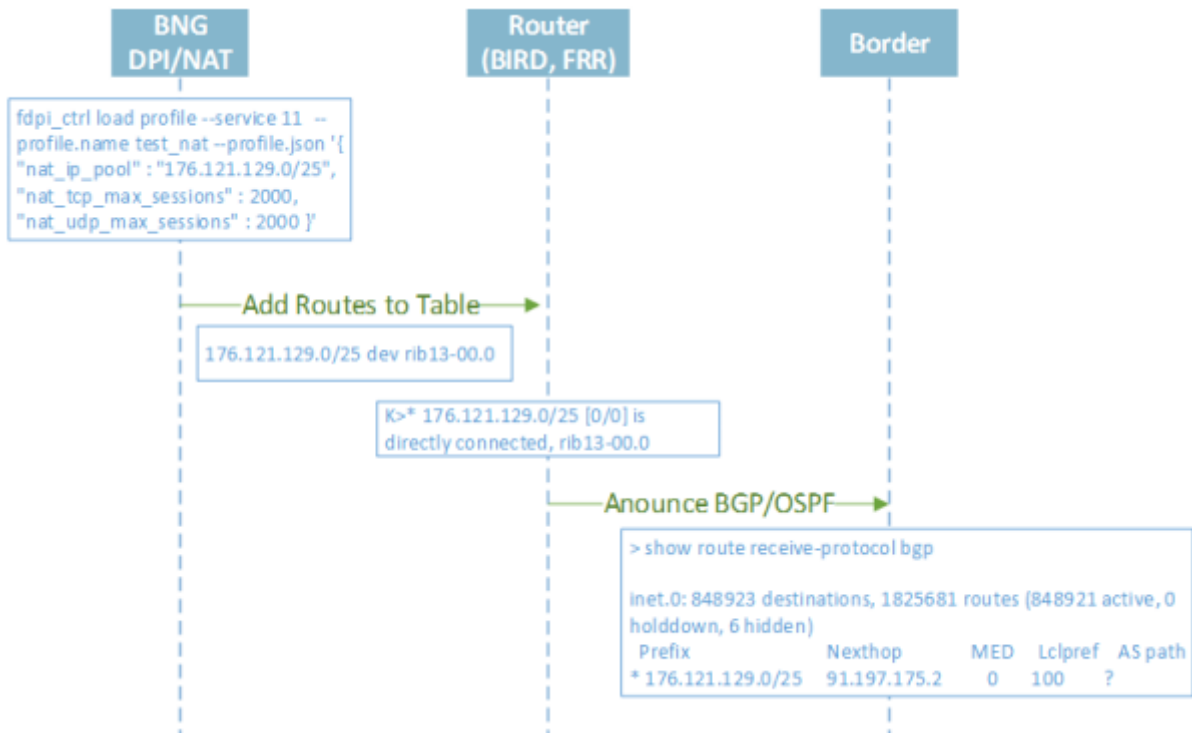


Эти три параметра являются **обязательными** для включения роутера. Прочие параметры, перечисленные ниже, не являются обязательными и предназначены для тонкой настройки роутера в СКАТ.

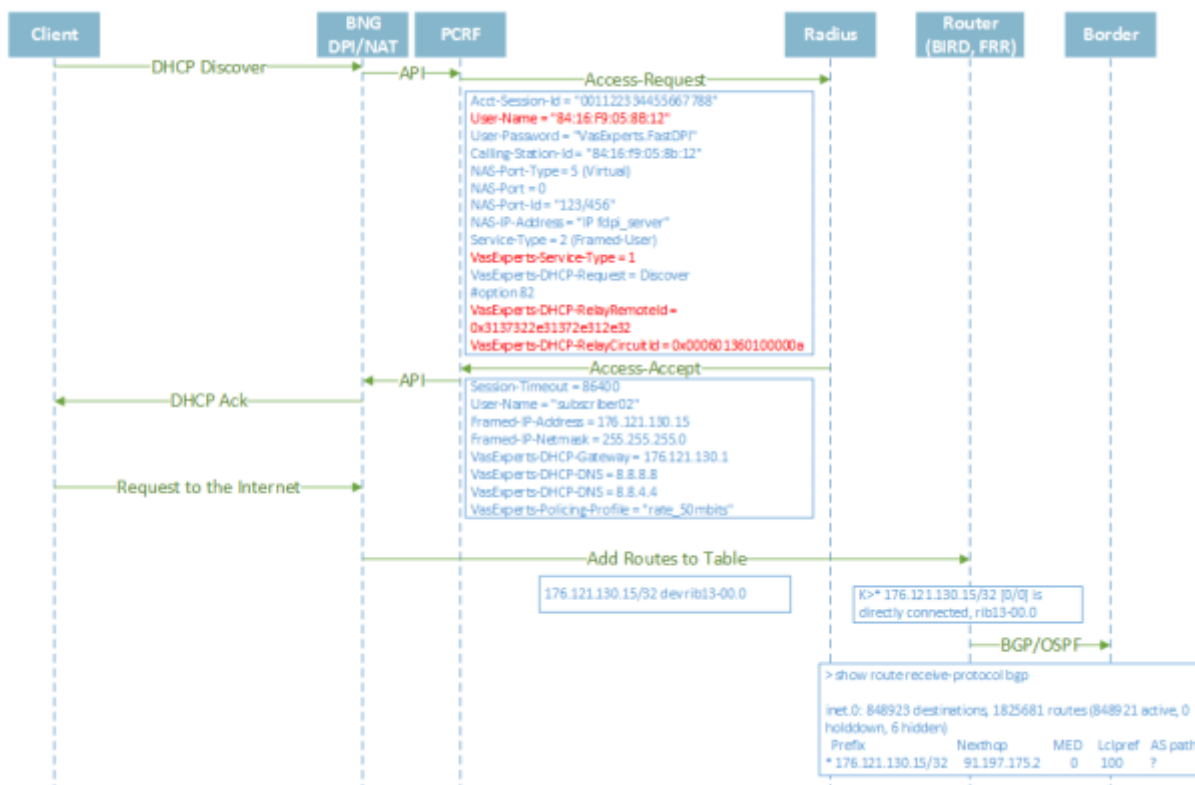
Анонсы абонентов и NAT pool

Включение анонсирования адресов абонентов производится параметром в fastdpi.conf:

```
# [cold] Флаги анонса адресов абонентов  
# Битовая маска  
# Значения:  
# 1 - анонсировать адрес абонента в сторону subs  
# 2 - анонсировать адрес абонента в сторону inet (если у абонента не подключен NAT)  
# 4 - анонсировать NAT-подсети в сторону inet  
# 8 - анонсировать абонентские шлюзы (направление задается флагами 1 и 2)  
# Значение по умолчанию: 0 - ничего куда не анонсировать  
#router_subs_announce=0  
  
# [hot] Метрика для анонсов адресов абонентов  
# Значение по умолчанию = 32  
#router_subs_metrics=32
```



Подсети белых адресов NAT анонсируются только в сторону inet при старте СКАТа и при добавлении/удалении/изменении NAT-профилей.



Адреса абонентов могут анонсироваться как в сторону inet, так и в сторону subs. Но если IP адрес абонента **входит в диапазон частных адресов и на него назначена 11 услуга, т.е. натится**, адрес абонента не анонсируется в сторону inet (так что нужно быть осмотрительными при определении диапазонов частных адресов). Анонс производится в таблицу маршрутизации BIRD для всех TAP-девайсов разрешенного направления, далее BIRD подхватывает изменения и анонсирует их по нужным протоколам в соответствии со своей конфигурацией.

Дополнительные параметры

Максимальное число маршрутов задается параметрами:

```
# [cold] Максимальное число маршрутов в IPv4 route table
# По умолчанию = 1000000
#router_max_ip4_route_count=1000000

# [cold] Максимальное число маршрутов в IPv6 route table
# По умолчанию = 200000
#router_max_ip6_route_count=200000
```

СКАТ при старте в режиме роутера преаллоцирует память для внутренних route table в соответствии с этими параметрами. Советуем устанавливать эти опции (если необходимо) с запасом в 20-30%, чтобы в процессе работы роутера гарантированно хватило преаллоцированной памяти.

Рабочая таблица маршрутизации (FIB) СКАТ обновляется раз в router_fib_refresh секунд:

```
# [hot] Период обновления FIB, секунд
# По умолчанию - раз в 15 секунд
#router_fib_refresh=15
```

Устанавливать данный параметр слишком маленьким (меньше 5 секунд) особого смысла нет.

Максимальный размер neighbor cache (ARP cache) и тайм-аут обновления записей этого кеша задается параметрами:

```
# [cold] Max размер ARP cache (neighbor cache для IPv6)
# По умолчанию - 1024, max = 32K
#router_arp_cache_size=1024
```

СКАТ содержит отдельные neighbor-кеши для IPv4 и IPv6, каждый размером router_arp_cache_size.

СКАТ сам не посылает ARP-запросы для устаревших записей кеша. Вместо этого он полагается на обновление кеша со стороны ядра Линукса: СКАТ мониторит ARP-ответы, приходящие на адрес подсети TAP-интерфейсов, и в соответствии с этими ответами обновляет свой ARP-кеш. То же самое относится и к IPv6 (мониторинг ICMPv6 neighbor discovery).

Роутер работает в отдельном потоке на отдельном ядре CPU. При старте СКАТ задает параметры этого потока по умолчанию, которые могут быть изменены параметрами:

```
# [cold] Добавление к приоритету служебного потока роутера (повышение
приоритета)
#router_sched_add_prio=0

# [cold] Ядро привязки потока роутера, -1 - автоопределение
#router_bind_core=-1
```

Изменять эти параметры надо только в крайнем случае, лучше дать СКАТу самому определить ядро и приоритет. Например, явное указание ядра для роутера `router_bind_core` может пригодиться в случае, если ядер не хватает; тогда можно явно привязать роутер к ядру, к которому привязан какой-то другой служебный поток (`ajb`, `ctl`).



Ни в коем случае не привязывайте роутер к ядру рабочего потока или диспетчера!

Отладка Роутера

Роутер СКАТа в целях отладки может записывать в `pcap` трафик с BIRD:

```
# [hot] Запись pcap с TAP-интерфейсов роутера
# Note: записывать можно и утилитой tcpdump, указав имя TAP-интерфейса.
# Но проблема в том, что tcpdump не работает с интерфейсами в режиме DOWN,
# то есть tcpdump'ом невозможно записать трафик при переходе интерфейса
# из состояния DOWN в состояние UP.
# Имена TAP-интерфейсов через ';' или 'all' (записывать со всех)
# Для каждого TAP-интерфейса создается отдельный pcap-файл с именем
# tap_<имя_интерфейса>_xxx.pcap в каталог, задаваемый параметром
ajb_udpi_path (по умолчанию /var/dump/dpi)
#router_tap_pcap=all|список TAP-интерфейсов через ';'

# [hot] Направление пакетов для записи pcap с TAP-интерфейсов
# Значения:
# 1 - TAP -> вовне (пакеты от TAP-интерфейса)
# 2 - извне -> TAP (пакеты на TAP-интерфейс)
# 0 или 3 - все направления
#router_tap_pcap_dir=0
# [hot] Интервал ротации TAP pcap, секунд
# 0 - берется из параметра ajb_udpi_ftimeout (ajb_udpi_ftimeout задается в
минутах)
#router_tap_pcap_rotate=0
```

Также можно включить запись в `pcap` обмена данными с ядром (`rtnetlink`):

```
# [hot] Записывать или нет rtnetlink messages в pcap
# 0 - не записывать
# 1 - записывать
# Префикс pcap-файлов = "rtnl"
#router_rtnl_pcap=0
```

Кроме того, если включена запись пакетов в `pcap` по маске адреса (`ajb_save_ip`), то роутер будет записывать в `pcap` также результирующий пакет после применения маршрутизации. То есть в `pcap` окажутся две записи для одного входящего пакета: первая запись - исходный пакет, вторая - отправленный пакет.

CLI-команды

СКАТ имеет набор CLI-команд по просмотру текущего состояния роутера. Полный список команд см.

```
fdpi_cli help router
```



Команды дампа RIB и FIB выводят очень много данных, так как эти структуры могут содержать сотни тысяч записей в случае BGP full view. Поэтому при вызове этих команд советуем перенаправлять вывод в файл

Также не забывайте, что построением таблицы маршрутизации по BGP, OSPF и пр. занимается BIRD, у которого есть собственная утилита командной строки `birdc`.