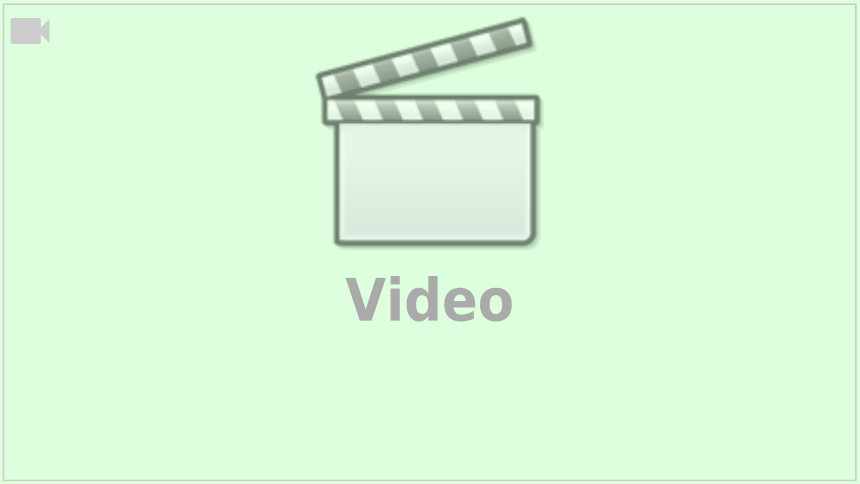
Содержание

Soft-Router (функция маршрутизации)

Общее описание

Подробнее о решении:



.

Сам СКАТ не занимается построением таблицы маршрутизации. Он делегирует эту работу проверенным специализированным инструментам, в примере внутренней архитектуры роутера 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.

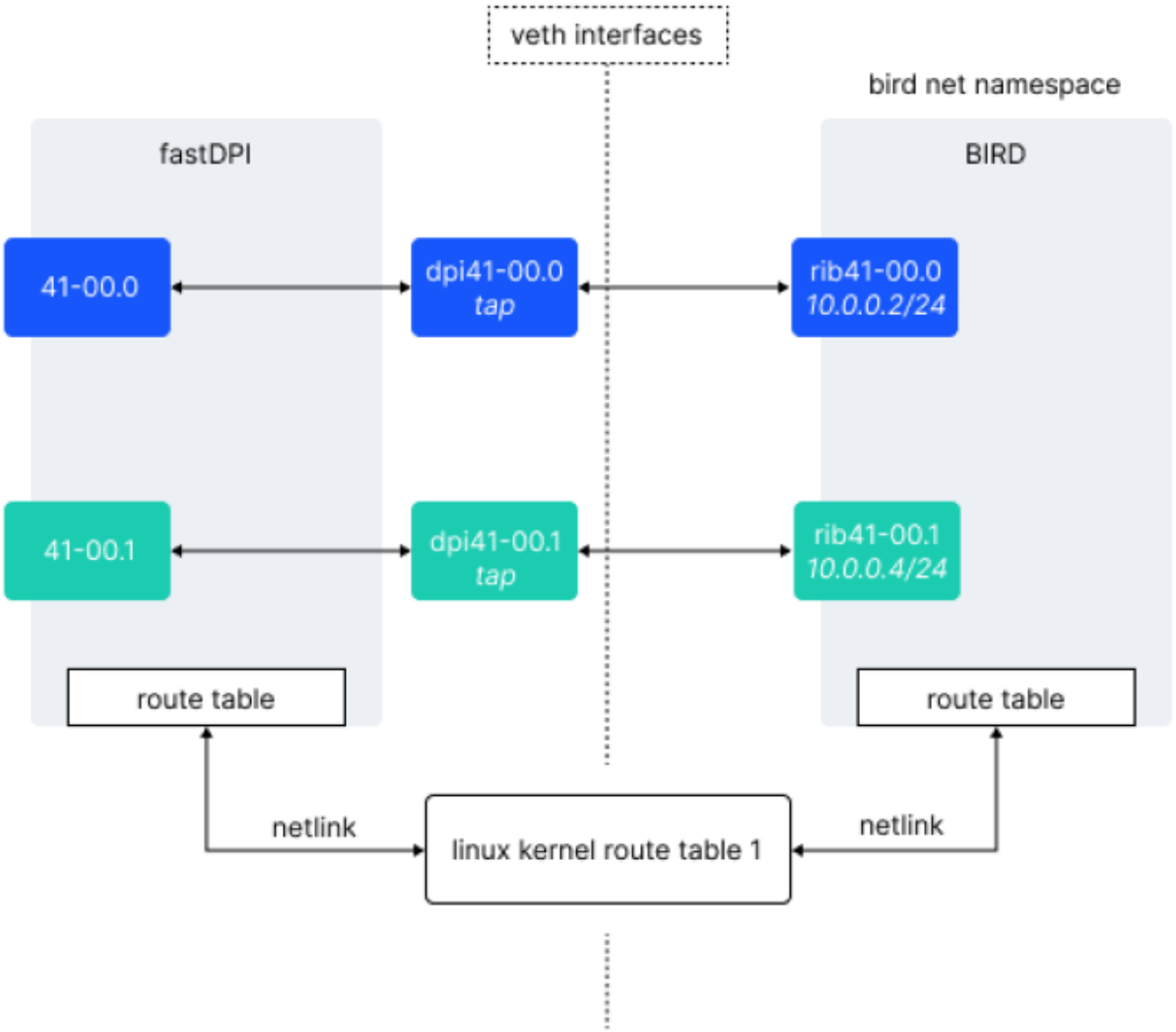
Вместо BIRD можно использовать любой другой демон, строящий таблицу маршрутизации в ядре Linux, например, FRRouting, QUAGGA, Juniper CRPD и другие. СКАТ для вычитывания таблицы маршрутизации использует только стандартный интерфейс Linux, поэтому совместим с любым демоном.

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 Linux-сервером, роут-демон BIRD работает в отдельном net namespace (на схеме внутренней архитектуры роутера это `bird netns`). Пакеты протоколов маршрутизации поступают на in/out-девайсы СКАТа в общем трафике. Для каждого in/out-девайса создается veth-пара "теневых" интерфейсов с предопределенными именами: dpi-интерфейс veth-пары работает как TAP-интерфейс, rib-интерфейс - как обычный системный интерфейс в netns BIRD'а.

Системные требования описаны в разделе Требования к оборудованию и производительность.

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 CKAT 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 CKAT существуют два FIB - текущая (которая сейчас используется для роутинга рабочими потоками) и "будущая". CKAT раз в `router_fib_refresh` секунд проверяет, не было ли изменений в RIB с момента построения текущей FIB. Если изменения были, CKAT 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.

Значения по умолчанию в роутере

router_max_ip4_route_count = 1000000. Максимальное количество маршрутов IPv4 для конкретного VRF.
router_max_ip6_route_count = 200000. Максимальное количество маршрутов IPv6 для конкретного VRF.
router_multipath_page = 8192. Максимальное число страниц для распределения multi-path маршрутов.
router_fib_refresh = 15s. Интервал обновления FIB.
router_arp_cache_size = 1024. Размер ARP-кеша.

Общее описание VRF

Статья в блоге: VRF Lite и L3VPN в СКАТ

В СКАТ реализован VRF Lite — он разделяет таблицы маршрутизации, но не помещает трафик отдельного VRF в уникальный туннель (MPLS, VXLAN).

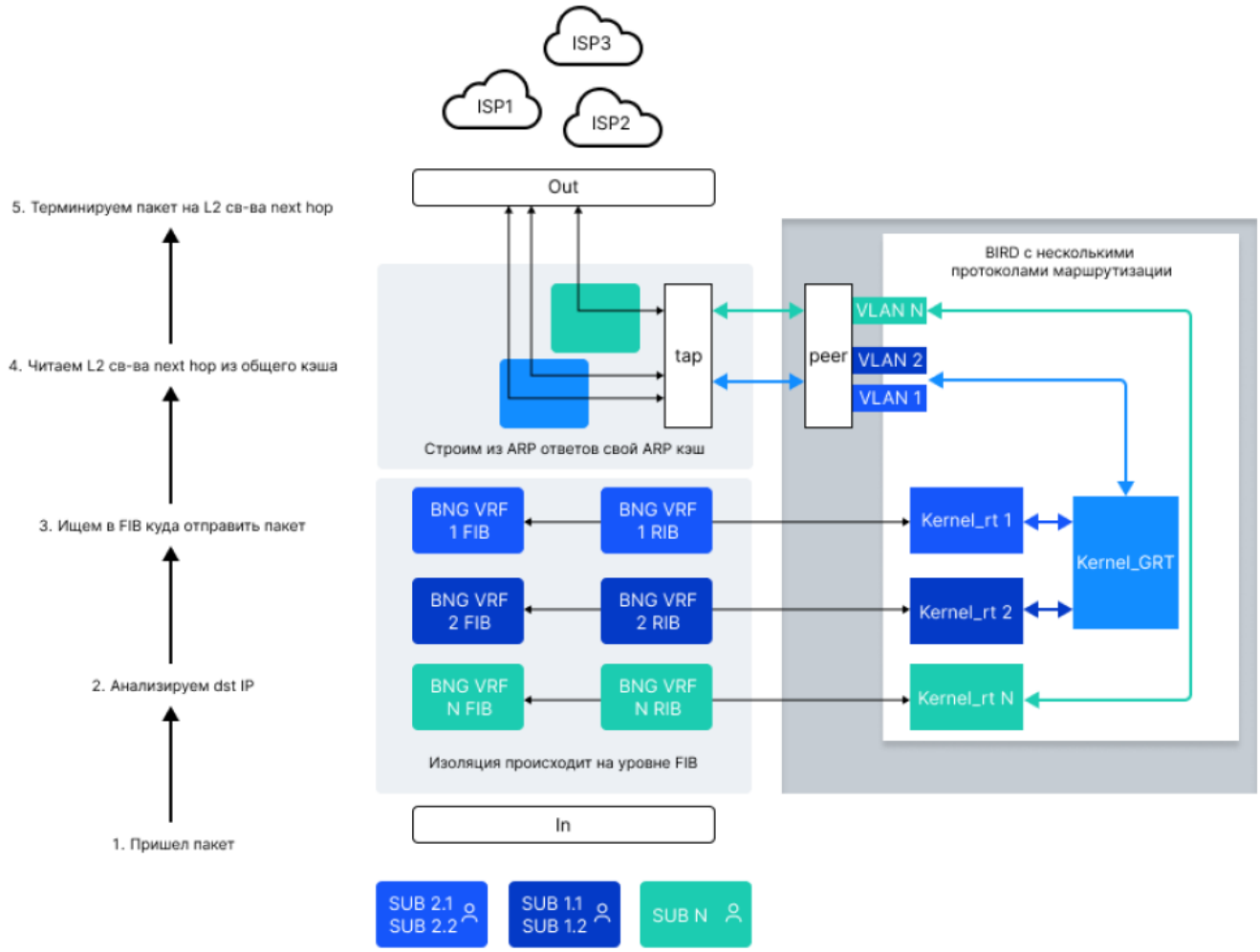
VRF (Virtual Routing and Forwarding instance) — это механизм виртуализации маршрутизации. VRF позволяет создавать виртуальные маршрутизаторы на одном физическом устройстве с независимыми таблицами маршрутизации, списками интерфейсов и другими параметрами. Это позволяет создавать виртуальные изолированные среды и гарантирует, что каждый VRF имеет независимые настройки и не имеет общих параметров с другими VRF и физическим устройством. Коммуникация между разными VRF возможна, но строго локальна для данного устройства, и VRF на одном маршрутизаторе никак не связан с VRF на другом.

Каждый VRF представляет собой отдельный VPN и не пересекается с другими VRF.

Примером может служить выделение трафика от IPTV приставки, которая также находится в L2-домене с BRAS, как и CPE (Customer Premises Equipment — сетевое оборудование, которое обеспечивает доступ пользователя к сети провайдера). IPTV приставка получает доступ только к локальным ресурсам, CPE получает доступ в интернет.

VRF может называться Virtual Routing Instance в Juniper, VRF в Cisco и MikroTik, но всё это — механизм виртуализации маршрутизации с аналогичными функциями.

Реализация VRF Lite в СКАТ



Приставка Lite означает, что CKAT только разделяет таблицы маршрутизации, но не помещает трафик отдельного VRF в уникальный туннель (MPLS, VXLAN). VRF Lite позволяет изолировать предоставляемые сервисы между собой и оптимизировать маршрутизацию при использовании разных каналов.

Реализация VRF Lite в CKAT выполнена с помощью Soft-Router, который строит RIB таблицу и пишет/читает из таблицы. CKAT отводит трафик в ядро Linux для демона маршрутизации. FastDPI обеспечивает построение FIB таблицы.

Демон маршрутизации работает в изолированном пространстве для того, чтобы операционная система не стала роутером. CKAT направляет всю сигнализацию в роутер, роутер сам строит правила маршрутизации и после этого CKAT, учитывая эти правила, начинает заниматься переадресацией и маршрутизацией пакетов.

В текущей версии на данный момент нет поддержки L3VPN и MPLS, но при необходимости можно настроить особую контекстную маршрутизацию. VRF Lite в CKAT — это отдельная услуга 254, которая может быть как подключена абоненту, так и не подключаться. Абонент по умолчанию попадает в указанный в CKAT VRF по умолчанию.

Логика работы

Маршрутизация в CKAT может происходить с помощью демонов маршрутизации FRR или BIRD — это отдельный процесс, который занимается обменом динамической маршрутизации

протоколов сигнализации (BGP, OSPF). В контексте VRF Lite архитектуре Linux, на которой работает CKAT, больше подходит BIRD. При использовании BIRD есть возможность работать с route leaking (перетеканием маршрутов) между таблицами маршрутизации.

В этом разделе рассмотрены варианты с одним пространством имен и несколькими таблицами маршрутизации, что больше подходит для демона маршрутизации BIRD. Также существует концепция с несколькими пространствами имен и с одной таблицей. Эта концепция более характерна для FRR.

GRT (Global routing table) на данной схеме — условное название, это точно такая же kernel route table, как и остальные на схеме. Таблица названа так для удобства тестирования и усложнения проверок. В ней использовались собственные правила маршрутизации от нескольких роутеров, они могут быть множественные, либо иметь свои контексты.

Все правила и фильтры настраиваются в роутере (в данном случае — в BIRD). При запуске BIRD, когда работает CKAT и поднимается BGP-соседство, одни приходящие маршруты складываются в таблицу Kernel_rt N, другие складываются в GRT и потом по заданным правилам перетекают в Kernel_rt 1 и Kernel_rt 2.

В CKAT описаны всё те же N VRF, направленные на определенные таблицы с правилами маршрутизации. Каждый инстанс читает из таблиц, на основе этих таблиц строятся изолированные FIB.

CKAT имеет свой уникальный ARP-кеш, который он строит на основе ответов на ARP-запросы. В CKAT может быть несколько ARP-таблиц, у нескольких VRF может быть одна общая таблица.

Вследствие **изоляции** в рамках CKAT одна и та же подсеть может быть доступна через разные hop и разные роутеры и маршрутизироваться по-разному. После того как построится FIB, когда пойдет трафик — сначала авторизуется абонент и помещается в нужный VRF. FIB может обновляться уже после того, как абонент авторизовался. Перемещать абонента между VRF можно динамически после авторизации абонента с помощью CoA.

Default gateway для этого абонента тоже будет перемещен в другой VRF и может стать недоступен для других абонентов (которые могут уже использовать сеть).

Действия при исходящем от абонента трафике:

1. Найти, к какому VRF принадлежит абонент.
2. Найти, куда направлен пакет.
3. Найти в FIB, на какой hop затерминировать пакет и терминировать. **Актуально только при создании flow, далее nexthop запоминается для flow.**

Действия при входящем трафике:

1. Анализ dest.
2. Проверка состояния абонента.
3. Тест на оригинацию пакета.

Для других групп абонентов все происходит аналогично изолированно, в соответствии с правилами. Анонсирование происходит из того VRF, на котором находится абонент.

Конфигурирование подсетей TAP

Для каждого `router_device` обязательно должно быть указано, какие подсети отводятся на TAP (фактически, это отведение пакетов протоколов маршрутизации на BIRD). CKAT 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=dpi41
        # Имя парного TAP-интерфейса в netns для девайса (default='rib' +
device)
    peer=rib41

        # 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 + 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 224.0.0.5 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=dpi41
    peer=rib41

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

```
}
```

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, они 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 CKAT, 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 CKAT он используется как TAP-интерфейс и наличие на нем адресов не требуется (более того, даже может мешать, если сам интерфейс начнет генерировать и отправлять 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
```

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

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

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

Как быть с VLAN?

CKAT 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

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

Настройка CKAT

Конфигурирование CKAT

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 CKAT
bras_arp_mac=00:E0:ED:43:84:42

    # Виртуальный IP-адрес CKAT
bras_arp_ip=188.227.73.40

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

    # Задает глобальный IPv6 адрес CKATa
bras_ipv6_address=2098::1

    # Задает IPv6 link-local адрес CKATa (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 CKAT.

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
```

CKAT 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) CKAT 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
```

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

CKAT сам не посылает ARP-запросы для устаревших записей кеша. Вместо этого он полагается на обновление кеша со стороны ядра Linux: CKAT 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 CKAT 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!

Конфигурирование VRF

Настройки `router_netns` и `router_kernel_table` задают VRF по умолчанию (default VRF) и применяются при описании других VRF как значения по умолчанию для соответствующих параметров VRF.

При подготовке fastDPI к работе в режиме роутера администратору нужно создать необходимые netns и TAP-интерфейсы для отвода трафика на route-демоны. В конфигурации `fastdpi.conf` указываются уже готовые (существующие) netns и TAP-интерфейсы, только в этом случае запустится СКАТ.

Каждая VRF задается отдельной новой секцией в конфигурации роутера:

Описание таблицы маршрутизации (VRF)

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

```
router_default_vrf=
```

`id` — строка, уникальный id VRF. Для абонента в Радиус VSA авторизации может быть указана VRF — это и есть данный id. Максимальный размер — 15 символов.
`netns` — имя netns, из которого вычитывается VRF. Если не задано — считается равным опции `router_netns`. В этом netns находятся peer TAP-интерфейсы для данной VRF.
`kernel_table` — номер таблицы маршрутизации ядра для данной VRF. Если не задано — считается равным опции `router_kernel_table`.
`router_default_vrf` — строка, id default VRF. Default VRF используется для абонентов, у которых нет свойства `vrf_id`.

neighbor_cache — строка, имя ARP кеша для данной VRF по умолчанию, каждый VRF имеет свой собственный, изолированный от других ARP/Neighbor кеш. Если нужно, чтобы несколько разных VRF имели общий ARP/Neighbor кеш, то следует задать в описании этих VRF одно и то же значение опции `neighbor_cache`.

Параметры для данной VRF:

max_ip4_route_count — максимальное количество маршрутов IPv4.
max_ip6_route_count — максимальное количество маршрутов IPv6.
multipath_page — максимальное число страниц для распределения multi-path маршрутов. Одна страница вмещает 64 разных multi-path маршрута. *На одной странице могут размещаться маршруты нескольких ECMP-групп. Одна группа может быть размещена по нескольким страницам (если количество маршрутов в группе более 64, если нет ограничений на количество маршрутов в демоне роутера)*
fib_refresh — интервал обновления FIB.
arp_cache_size — размер ARP-кеша.

Эти параметры определяют требуемую память и частоту обновления для данного VRF. Здесь можно указать значения по умолчанию (берутся из глобальных опций) или переопределить их.

- Если в конфигурации **нет** секций `router_vrf`, режим работы остается прежним: back compatibility, это означает, что в CKAT описана одна VRF, которая является дефолтной.
- Если в конфигурации **есть** секции `router_vrf`, режим работы — поддержка VRF. При этом ровно одна VRF должна быть дефолтной, то есть должна быть задана опция `router_default_vrf=id` дефолтной VRF.

bras_vrf_isolation - изоляция VRF. L2 BRAS не изолирует абонентов из разных VRF: Если данный режим включен (1), то абоненты из разных VRF будут изолированы друг от друга. Значение по умолчанию: 0. При включении этой опции:

1. ARP абонента к шлюзу - обрабатывается fastDPI только если абонент и шлюз в одном VRF
2. ICMP ping шлюза - обрабатывается fastDPI только если абонент и шлюз в одном VRF
3. local interconnect - применяется только если оба абонента в одном VRF

bras_egress_filtering - фильтрация исходящего трафика subs→inet (битовая маска). По умолчанию отключена (0). При включении этой опции пакет будет дропнут при выполнении следующих условий:

1. IP-адрес абонента (srcIP) неизвестный для L2 BRAS
2. bras_term_by_as = 0
3. AS абонента не local

Описание router_device

В описание `router_device` добавляется id VRF. Описание одного интерфейса роутера:

```
router_device {
    device=
    tap=
    peer=
    vrf=
```

```
    subnet=
    subnet=
    subnet6=
    subnet6=
}
```

device — имя девайса из `in_dev/out_dev`.
tap — имя TAP-интерфейса для девайса (`default='DPI' + device`).
peer — имя парного TAP-интерфейса в netns для VRF (`default='rib' + device`).
vrf — идентификатор VRF, в котором находится peer. Если не указан — считается равным default VRF.

subnet и **subnet6** — подсети, отводимые из общего трафика на TAP-девайс. Для `router_device` обязательно должен быть задан хотя бы один параметр `subnet` или `subnet6`!

subnet — перечисление IPv4-подсетей, отводимых из общего трафика на TAP-девайс.
subnet6 — перечисление IPv6-подсетей, отводимых из общего трафика на TAP-девайс.

Каждая IPv4 и IPv6 подсеть указывается отдельно в параметре `subnet` и `subnet6` соответственно. Всего может быть не более 16 параметров `subnet` и 16 параметров `subnet6` для одного `router_device`.

Подсети для одного порта не должны пересекаться.
Например, BGP1 из VRF1 — подсеть 10.20.30.0/24, BGP2 из VRF2 — подсеть 10.20.30.0/20.

Управление VRF абонентами

VRF ID поступает в fastDPI при авторизации, в новой услуге 254.

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

Здесь “VRF_ID” — идентификатор VRF.

Пример настройки VRF в CKAT

```
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://vasexperts.ru/test/blocked.php
```

```
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
```

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=dpi41
        # Имя парного  TAP-интерфейса в  netns  для девайса  (default='rib' +
device)
    peer=rib41
}
```

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 CKATa `dpi41-00.0`, со стороны BIRD `rib41-00.0`
- для `out_dev=41-00.1`: со стороны CKATa `dpi41-00.1`, со стороны BIRD `rib41-00.1`

Поддержка LAG

В CKAT 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=dpi0
        # Имя парного TAP-интерфейса в netns для девайса (default='rib' +
device)
    peer=rib0
        # 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 девайсы из in_dev, либо все девайсы out_dev). 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.

Секции `lag` в fastdpi.conf - это холодный параметр, требуется рестарт fastdpi при изменении описания LAG.

ARP менеджмент

NeighborDB - это БД nexthop, которую обслуживает сам fastDPI: посылает ARP request для выяснения MAC-адреса nexthop, обновляет ARP кеш для этих nexthop. Для обновления используются следующие опции:

- `router_neighbor_lifetime` — время жизни записи в ARP кеше, в секундах.
  По истечении этого времени запись нужно обновить отправкой ARP request (ICMPv6 NDP в случае IPv6). Значение по умолчанию: 300 секунд
- `router_neighbor_refresh_timeout` — время ожидания ответа на ARP request (сек).
  Значение по умолчанию: 3 секунды
- `router_neighbor_refresh_attempt` — количество безуспешных попыток обновления, нужных для того чтобы перевести запись ARP-кеша в состояние Unreachable (недостижимая); другими словами, сколько подряд надо послать ARP Request без ответа, чтобы запись стала unreachable.
  Значение по умолчанию: 10 попыток
- `router_neighbor_unreachable_refresh` — после того, как запись становится Unreachable, fastDPI периодически пытается оживить её, посылая ARP Request. Данная опция задает этот период в секундах (по умолчанию 30 сек).

Запись nexthop состоит из:

- nexthop IP;
- *Обязательно* — имя порта, через который осуществляется связь с этим nexthop (в случае LAG — один из портов, входящих в LAG);
- *Опционально* — VLAN
- *Опционально* — MAC-адрес nexthop. Записи с явно заданным MAC-адресом nexthop являются static ARP cache record — по ним никогда не посылается ARP-запросов IPv4 и IPv6 NeighborDB хранится в SDR в файле `router.mdb`. SDR (System Data Repository) — это новая сущность наподобие UDR, но содержащая системные настройки. Каталог, где располагается SDR, задается новой опцией `sdr_path`, по умолчанию это `/var/db/dpisdr`.

Управление NeighborDB (добавление/модификация/удаление записей) производится через команды CLI:

- `router neighbor dump` — вывод полного списка всех загруженных записей и их внутреннего состояния
- `router neighbor dump db` — вывод дампа БД router.mdb.
- `router neighbor add` — добавление новой записи (нового nexthop) в NeighborDB.
- `router neighbor update` — модификация существующей записи NeighborDB
- `router neighbor delete` — удаление записи NeighborDB
- `router neighbor refresh` — принудительное обновление (отправка ARP Request) MAC-адреса конкретного nexthop IP или всех nexthop в указанном VRF или всей NeighborDB
- `router neighbor purge` — удаление всех записей NeighborDB (полная очистка NeighborDB).

Синтаксис каждой из этих команд см. `fdpi_cli <command> ?`

Приоритетность в ARP кеше
С появлением NeighborDB в fastDPI появились два независимых источника записей в ARP кеше:

- [прежний] BIRD/FRR: fastDPI отлавливает взаимодействие BIRD/FRR с соседями и соотвественно заполняет свой ARP кеш для nexthop; fastDPI не обновляет статус таких записей отправкой ARP Request, а надеется на то, что это сделает BIRD/FRR
- [новый] NeighborDB: fastDPI сам следит за актуальностью записей, своевременно отправляя ARP Request. **Записи NeighborDB имеют бОльший приоритет**, чем записи от BIRD/FRR.

Поддержка Multi-path routing (ECMP)

В CKAT 10.2 добавлена поддержка multi-path routing (ECMP).

CKAT делает балансировку трафика (round-robin) на уровне flow по всем маршрутам из multi-path. Балансировка на уровне flow означает, что конкретный flow будет закреплен за одним из маршрутов из multi-path и выбранный маршрут не будет изменяться до окончания данного flow (если, конечно, не будет изменен состав multi-path по внешним событиям от демона маршрутизации).

Для включения multi-path в СКАТ не требуется никакой настройки. Поддержка ECMP включается в конфигурационных параметрах демона маршрутизации. Например, в BIRD поддержка ECMP включается указанием `merge paths yes` в протоколе `kernel`, см. https://bird.network.cz/?get_doc&v=20&f=bird-6.html#ss6.6.

Особенности анонсирования адресов

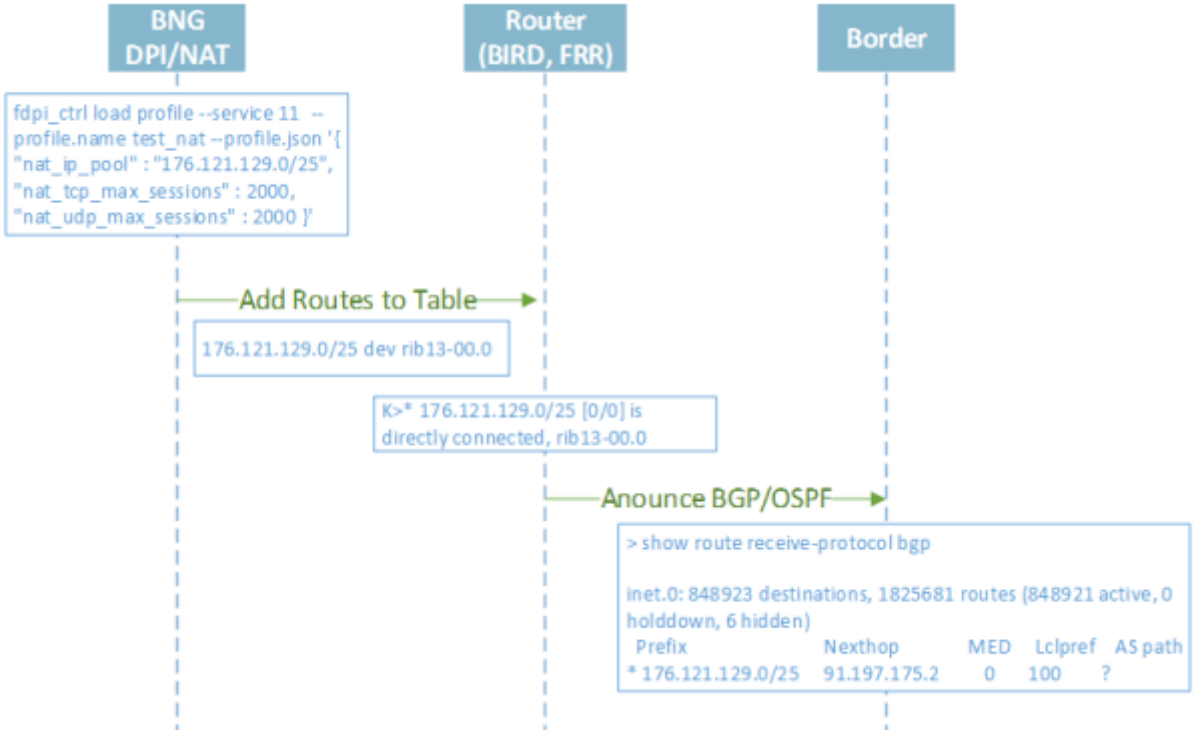
Анонсы абонентов и NAT pool

NAT-пулы анонсируются только в default VRF.
NAT применяется к абоненту, если ему подключена услуга NAT и он относится к VRF, в котором анонсируются NAT пулы.

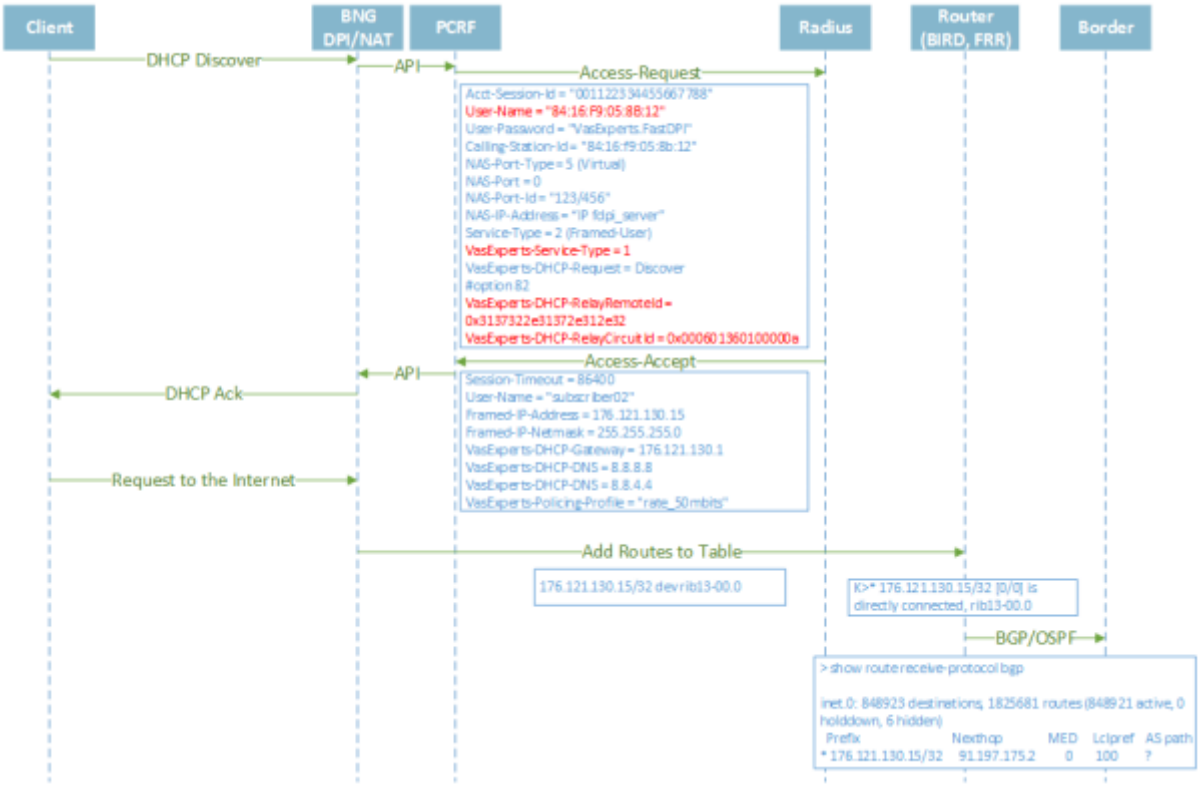
Включение анонсирования адресов абонентов производится параметром в fastdpi.conf:

```
    # [cold] Флаги анонса адресов абонентов
    # Битовая маска
    # Значения:
    #   0x00001 - анонсировать адрес абонента в сторону subs
    #   0x00002 - анонсировать адрес абонента в сторону inet (если у абонента не подключен NAT)
    #   0x00004 - анонсировать NAT-подсети в сторону inet
    #   0x00008 - анонсировать абонентские шлюзы (направление задается флагами 1 и 2)
    #   0x10000 - деанонсировать L3-абонента при наступлении acct idle (закрытие acct-сесси по idle timeout)
    # Значение по умолчанию: 0 - ничего никуда 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 приватных 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 приватных 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.

[CKAT 10.2+] Если в fastdpi.conf задано `nat_exclude_private=1`, то есть не применять NAT для пакетов "приватный IP → приватный IP", то CKAT **будет** анонсировать приватный адрес абонента с подключенным NAT в сторону inet, так как вышестоящий роутер должен знать маршрут до серого IP-адреса абонента.

Описание параметров по ссылке

Анонс адресов L3-абонентов

В CKAT 10.2 действует следующий алгоритм анонсов IP-адресов L3-абонентов:

- по умолчанию, анонс адреса производится для успешной авторизации (accept) и НЕ производится для неуспешной (reject)
- fastdpi.conf-параметр `auth_announce_reject` позволяет глобально разрешить анонсы для reject:

```
# [hot] Анонсировать (1) или нет (0) неавторизованных (Reject) абонентов
# Значение по умолчанию - 0 (не анонсировать)
#auth_announce_reject=0
```

- добавлен новый Радиус-атрибут `VasExperts-Route-Announce`: значение 0 - не анонсировать адрес абонента, значение 1 - анонсировать. Данный атрибут перекрывает (имеет более высокий приоритет), чем `auth_announce_reject`.

Итого при решении вопроса, нужно ли анонсировать IP-адрес абонента или нет, применяется следующий алгоритм:

- если в ответе Радиуса (Accept или Reject) явно задан атрибут `VasExperts-Route-Announce`, анонс будет производиться для `VasExperts-Route-Announce=1` и НЕ будет для `VasExperts-Route-Announce=0`
- если атрибута `VasExperts-Route-Announce` в ответе Радиуса нет:
  - Для Access-Accept (успешная авторизация) IP-адрес абонента анонсируется
  - Для Access-Reject (неуспешная авторизация) IP-адрес анонсируется только если fastdpi.conf-параметр `auth_announce_reject=1`

этот алгоритм применяется на стадии вызова функции анонсирования роутера; будет или нет анонсирован IP-адрес абонента и в какую сторону, — это регулируется fastdpi.conf-параметром роутера `router_subs_announce`.

Данный алгоритм также применяется для ARP-авторизации и GTP-авторизации, то есть для всех типов авторизации, где IP-адрес абонента известен на момент вызова авторизации и не может быть изменен.

Для прочих видов L2-авторизации (DHCP, PPP, — всех, где явно **выдается** IP-адрес) атрибут `VasExperts-Route-Announce` не принимается в расчет, анонс происходит по факту выдачи IP-адреса абоненту.

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

В примере приведен частный случай, за дополнительной информацией обратитесь к документации конкретного роут-демона. В контексте VRF Lite архитектуре Linux, на которой работает СКАТ, больше подходит BIRD.

Пример настройки BIRD

```
# Указать дополнительные функции для проверки:

# Любой публичный адрес
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;
}

# Любой приватный адрес
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;
}

# Шлюз по умолчанию
filter default_gw {
    if net ~ [0.0.0.0/0] then
        accept;
    reject;
}

# Описать фильтры:
# Маршруты, которые не получены из других протоколов маршрутизации (и префикс не /32)
filter exclude_external_routes {
```

```
    if (source = RTS_INHERIT) && (net.len != 32) then
        accept;
    reject;
}

# Исключить маршруты из других протоколов маршрутизации, публичные подсети,
приватные - не /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;

# Описать таблицы
ipv4 table grt;
ipv4 table bird00;
ipv4 table bird01;

protocol device {
}


protocol direct {
    disabled;         # Отключено по умолчанию
    ipv4;             # Подключение к таблице  IPv4  по умолчанию
    ipv6;             # ... и к таблице  IPv6  по умолчанию
}

# Описать "протоколы"  kernel
protocol kernel kernel_grt {
    ipv4 {             # Подключить протокол к таблице  IPv4  по каналу
          table grt;
          import all;     # Импорт в таблицу, по умолчанию - импортировать все
          export all;     # Экспорт в протокол. Значение по умолчанию - экспорт
отсутствует
    };
    scan time 5;
    learn;             # Изучить пришедшие маршруты из  kernel-таблицы
    kernel table 99;     #  Kernel-таблица для синхронизации с (по умолчанию:
основная)
}
protocol kernel kernel_bird00 {
    ipv4 {             # Подключить протокол к таблице  IPv4  по каналу
          table bird00;
```

```
          import all;     # Импорт в таблицу, по умолчанию - импортировать все
          export all;     # Экспорт в протокол. Значение по умолчанию - экспорт
отсутствует
    };
    scan time 5;
    learn;              # Изучить пришедшие маршруты из kernel-таблицы
    kernel table 100;     # Kernel-таблица для синхронизации с (по умолчанию:
основная)
}

protocol kernel kernel_bird01 {
    ipv4 {              # Подключить протокол к таблице IPv4 по каналу
          table bird01;
          import all;     # Импорт в таблицу, по умолчанию - импортировать все
          export all;     # Экспорт в протокол. Значение по умолчанию - экспорт
отсутствует
    };
    scan time 20;
    learn;              # Изучить пришедшие маршруты из kernel-таблицы
    kernel table 101;     # Kernel-таблица для синхронизации с (по умолчанию:
основная)
}

# Другой экземпляр для IPv6, пропускающий параметры по умолчанию
protocol kernel {
    ipv6 { export all; };
}

protocol static {
    ipv4;             # Снова канал IPv4 с параметрами по умолчанию

}

# Протоколы OSPF (каждый инстанс со своей таблицей)
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;
               };
         };
}

# Описать "протоколы" маршрутизации, которые предназначены для "переливания" маршрутов между таблицами (с использованием фильтров)
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;
}
```

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.

Трассировка

```
    # [hot] Флаги трассировки различных частей роутера
    # 0x0001          транзит состояний FSM обработки rtnetlink (оповещения от
BIRD)
    # 0x0002          события FSM обработчика rtnetlink (оповещения от BIRD)
    # 0x0004          дамп данных rtnetlink
    # 0x0010          трассировка RIB (построение route info base)
    # 0x0020          трассировка таблицы маршрутизации
    # 0x0040          трассировка FIB
    # 0x0080          трассировка ARP-кеша
    # 0x0100          трассировка TAP
    # 0x0200          трассировка TAP pcap
    # 0x0400          трассировка объявлений
```

```
#router_trace=0
```

CLI-команды

CKAT 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` и собственный конфигурационный файл с развитой системой команд по фильтрации, заданию static-маршрутов и пр.

Кроме того, стандартная утилита Linux `ip` дает полный контроль над kernel route table. При использовании утилиты `ip` не забывайте указывать правильный netns (`router_netns`) и номер таблицы маршрутизации (`router_kernel_table`).