

# Содержание

<b>3 Настройка DPDK версии .....</b>	<b>3</b>
Подготовка системы .....	3
Конфигурирование СКАТ .....	4
Задание псевдонимов девайсов .....	5
Конфигурирование в Hyper-V .....	6
Кластеры .....	7
Число ядер (потоков) .....	7
Загрузка потока диспетчера .....	10
dpdk_engine=0: Один диспетчер .....	11
dpdk_engine=1: Диспетчера по направлению .....	11
dpdk_engine=2: Поддержка RSS .....	12
dpdk_engine=3: Диспетчер на мост .....	12



# 3 Настройка DPDK версии

DPDK (Data Plane Development Kit) позволяет работать с сетевыми картами напрямую, фактически без посредничества ядра Linux, тем самым повышается производительность решения. DPDK поддерживает намного больше моделей сетевых карт, чем pf\_ring, и намного более богатый интерфейс работы с ними, что позволяет реализовать различные схемы работы с картами, подходящие как для 10G трафика, так и для трафика 25G, 40G, 100G и т.д.

## Подготовка системы

Сетевые карты, с которыми будет работать СКАТ, выведены из-под управления операционной системы и поэтому как Ethernet устройства для нее не видны. DPDK адресует Ethernet устройства по их PCI идентификаторам, которые можно получить командой:

```
lspci -D|grep Eth  
  
0000:04:00.0 Ethernet controller: Intel Corporation 82599ES 10-Gigabit  
SFI/SFP+ Network Connection (rev 01)  
0000:04:00.1 Ethernet controller: Intel Corporation 82599ES 10-Gigabit  
SFI/SFP+ Network Connection (rev 01)
```

Эта команда выведет список всех PCI-устройств типа ethernet. Каждая строка начинается с системного идентификатора PCI-устройства, - именно эти PCI-идентификаторы являются уникальными идентификаторами сетевой карты в DPDK.

Список карт в режиме DPDK можно проверить командой:

```
driverctl list-overrides  
  
0000:04:00.0 vfio-pci  
0000:04:00.1 vfio-pci
```

При необходимости карты можно вывести из режима DPDK командой, при этом для них активируется штатный драйвер Linux

```
driverctl unset	override 0000:04:00.0  
driverctl unset	override 0000:04:00.1
```

После работ со штатных драйвером не забудьте вернуть их обратно под управление DPDK командой

```
driverctl -v set	override 0000:04:00.0 vfio-pci  
driverctl -v set	override 0000:04:00.1 vfio-pci
```



При переводе карт в режим DPDK будьте внимательны и не переведите случайно управляющий интерфейс сервера в режим DPDK - связь с сервером



сразу прервется!

В старых инсталляциях вместо vfio-pci использовался драйвер igb\_uio, что можно увидеть в выводе команды

```
driverctl list-overrides  
0000:04:00.0 igb_uio
```



В этом случае рекомендуется перейти на использование драйвера vfio-pci для чего выполнить команды

```
echo "options vfio enable_unsafe_noiommu_mode=1" >  
/etc/modprobe.d/vfio-noiommu.conf  
driverctl -v set-override 0000:04:00.0 vfio-pci
```

для всех устройств из списка возвращаемого list-overrides

## Конфигурирование СКАТ

После того, как система настроена для работы с DPDK, можно приступить к конфигурированию СКАТ. Интерфейсы конфигурируются парами «вход»-«выход» (для последующего удобства конфигурирования опций интерфейс «вход» должен быть обращен во внутреннюю сеть оператора, а «выход» в сторону аплинка). Каждая пара образует сетевой мост, прозрачный на уровне L2. В качестве имен интерфейсов выступают PCI-идентификаторы с заменой ':' на '-' (так как символ ':' в имени интерфейса зарезервирован в СКАТ для разделения интерфейсов в одном кластере) и без начального префикса "0000:" - он у всех одинаковый:

```
# Вход - порт 41:00.0  
in_dev=41-00.0  
# Выход - порт 41:00.1  
out_dev=41-00.1
```

Такая конфигурация задает единственный мост 41-00.0 ↔ 41-00.1

Можно указывать группу интерфейсов через ':'

```
in_dev=41-00.0:01-00.0:05-00.0  
out_dev=41-00.1:01-00.1:05-00.1
```

Эта группа образует следующие пары (мосты):

41-00.0 ↔ 41-00.1  
01-00.0 ↔ 01-00.1  
05-00.0 ↔ 05-00.1

В парах должны быть устройства одинаковой скорости; недопустимо объединять в пару 10G и 40G карты. Однако, в группе могут быть интерфейсы разной скорости, например, одна пара 10G, другая - 40G.

## Задание псевдонимов девайсов

В СКАТ 9.6 появилась возможность задавать псевдонимы (alias) девайсов. Вызвано это тем, что DPDK поддерживает большое количество девайсов, не только PCI, но и, например, vmbus-девайсы (Hyper-V) или виртуальные девайсы vdev. Кроме того, каждый DPDK-драйвер поддерживает свой набор конфигурационных параметров для тонкой настройки. Синтаксис описания таких девайсов несовместим с синтаксисом задания в `in_dev/out_dev`, поэтому введено понятие псевдонима девайса.

Суть псевдонима очень проста: вы описываете необходимый девайс в отдельном параметре и задаете этому описанию имя. Далее в параметрах `in_dev`, `out_dev`, `tap_dev` вы указываете это имя - псевдоним девайса.

Каждый псевдоним задается отдельным параметром `dplk_device`:

```
dplk_device=alias:bus:device-description
```

здесь:

- `alias` - задает псевдоним девайса (например, `eth1`). В псевдониме допустимы только буквы и цифры.
- `bus` - тип шины: `pci`, `vmbus`, `vdev`
- `device-description` - описатель девайса в синтаксисе, принятом в DPDK

Например:

```
# eth1 - псевдоним PCI-девайса 41:00.0
dplk_device=eth1:pci:41:00.0
# eth2 - псевдоним PCI-девайса 41:00.1
dplk_device=eth2:pci:41:00.1
```

```
in_dev=eth1
out_dev=eth2
```

Это описание эквивалентно следующему:

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

Отметим, что в `dplk_device` PCI-девайс задается в каноническом виде `41:00.0`.



Для PCI-девайсов задание в `in_dev/out_dev` через псевдонимы не обязательно, можно использовать прежнюю нотацию.

Если требуется подключить Hyper-V девайсы (а это не PCI, а VMBus-девайсы), то использование псевдонимов обязательно. Пример:

```
dplk_device=subs1:vmbus:392b7b0f-dbd7-4225-a43f-4c926fc87e39
dplk_device=subs2:vmbus:58f75a6d-d949-4320-99e1-a2a2576d581c,latency=30
```

```
dpdk_device=inet1:vmbus:34f1cc16-4b3f-4d8a-b567-a0eb61dc2b78
dpdk_device=inet2:vmbus:aed6f53e-17ec-43f9-b729-f4a238c49ca9, latency=30
in_dev=subs1:subs2
out_dev=inet1:inet2
```

Здесь мы не только задаем псевдоним, но и указываем аргумент `latency=30` для DPDK-драйвера. В принципе, каждый драйвер DPDK поддерживает свой набор аргументов, см. [документацию DPDK](#) соответствующей версии (версия DPDK, с которой собран СКАТ, выводится в `fastdp_alert.log` при старте, а также при вызове `fastdp -ve`). Следует отметить, что бездумное задание аргументов для драйвера может привести к труднообнаружимым ошибкам и потере работоспособности СКАТА, поэтому не советуем пользоваться этой возможностью без консультаций с нашей техподдержкой.

## Конфигурирование в Hyper-V

Начиная с версии 9.6, СКАТ поддерживает работу в виртуальной машине Hyper-V. На гостевой CentOS-8 должны быть установлены:

```
# Поддержка multi-queue - необходима для СКАТ
dnf install kernel-modules-extra
```



Host-система (Windows) должна поддерживать multiple channel для виртуализованных сетевых карт

Девайсы в Hyper-V являются VMBus-, а не PCI-девайсами, поэтому им требуется особый перевод в режим DPDK. Каждый девайс (интерфейс) идентифицируется своим уникальным идентификатором UUID, поэтому сначала нужно узнать UUID всех интерфейсов, с которыми будет работать СКАТ. Затем нужно перевести девайс в DPDK-режим:

```
# переводим интерфейсы eth0 и eth2 в DPDK-режим
for DEV in eth0 eth2
do
    # получаем UUID девайса
    DEV_UUID=$(basename $(readlink /sys/class/net/$DEV/device))
    # переводим в DPDK compatible mode
    driverctl -b vmbus set-override $DEV_UUID uio_hv_generic

    # Device appears in
    # /sys/bus/vmbus/drivers/uio_hv_generic/$DEV_UUID

    echo "$DEV uuid=$DEV_UUID"
done
```

При необходимости интерфейс может быть переведен обратно в kernel-режим так:

```
ETH0_UUID=<eth0_UUID>
driverctl -b vmbus unset-override $ETH0_UUID
```

Далее конфигурируем СКАТ - задаем девайсы в `fastdapi.conf`. При этом используем [псевдонимы](#) для указания UUID, которые мы только что узнали:

```
# eth0 UUID=392b7b0f-dbd7-4225-a43f-4c926fc87e39  
dpdk_device=eth0:vmbus:392b7b0f-dbd7-4225-a43f-4c926fc87e39  
# eth2 UUID=34f1cc16-4b3f-4d8a-b567-a0eb61dc2b78  
dpdk_device=eth2:vmbus:34f1cc16-4b3f-4d8a-b567-a0eb61dc2b78
```

```
# далее везде используем псевдонимы eth0 и eth2 при указании девайсов  
in_dev=eth0  
out_dev=eth2
```

## Кластеры

DPDK-версия СКАТ поддерживает кластеризацию: можно указывать, какие интерфейсы входят в каждый кластер. Разделителем кластеров является символ '|'

```
in_dev=41-00.0|01-00.0:05-00.0  
out_dev=41-00.1|01-00.1:05-00.1
```

Этот пример создает два кластера:

- кластер с мостом 41-00.0 ↔ 41-00.1
- кластер с мостами 01-00.0 ↔ 01-00.1 и 05-00.0 ↔ 05-00.1

Кластеры являются в большей мере наследием pf\_ring-версии СКАТ: в pf\_ring кластер является базовым понятием, означающим "один поток диспетчера + RSS потоков-обработчиков", и это чуть ли не единственный способ масштабирования. Недостатком кластерного подхода является то, что кластеры физически изолированы друг от друга: невозможно переслать пакет с интерфейса X кластера #1 на интерфейс Y кластера #2. Это может являться значительным препятствием в режиме L2 BRAS СКАТ.

В DPDK кластеры также изолированы друг от друга, но в отличие от pf\_ring здесь кластер - понятие во многом логическое, наследуемое от pf\_ring. DPDK намного гибче, чем pf\_ring, и позволяет строить сложные многомостовые конфигурации со множеством диспетчеров без использования кластеров. Фактически, единственным аргументом "за" кластеризацию в DPDK-версии СКАТ является случай, когда у вас к СКАТ подключены две независимые сети А и В, которые никоим образом не должны взаимодействовать друг с другом.



Совет: вместо использования кластеров рассмотрите переход на другой `dpdk_engine`, более подходящий под вашу нагрузку

Далее при описании конфигураций предполагается, что есть только один кластер (то есть кластеризация не используется).

## Число ядер (потоков)

Ядра CPU являются, пожалуй, самым критичным ресурсом СКАТ. Чем больше физических ядер имеется в системе, тем больший трафик сможет обрабатывать СКАТ.



СКАТ не использует Hyper-Threading: учитываются только реальные физические ядра, а не логические

Для работы СКАТу нужны следующие потоки:

- потоки обработки - обрабатывают входящие пакеты, пишут в TX-очереди карты;
- потоки диспетчера - читают RX-очереди карты и раскидывают входящие пакеты по потокам обработки;
- служебные потоки - выполняют отложенные (продолжительные) действия, принимают и обрабатывают команды fdpi\_ctrl и CLI, связь с PCRF, отправка netflow
- системное ядро - выделено для работы операционной системы.

Потоки обработки и диспетчера не могут располагаться на одном ядре. При старте СКАТ привязывает потоки к ядрам. СКАТ по умолчанию выбирает число потоков-обработчиков в зависимости от скорости интерфейса:

10G - 4 потока

25G - 8 потоков

40G, 50G, 56G - 16 потоков

100G - 32 потока

Для группы число потоков равно сумме числа потоков для каждой пары; например, для таких карт

```
# 41-00.x - 25G NIC
# 01-00.x - 10G NIC
in_dev=41-00.0:01-00.0
out_dev=41-00.1:01-00.1
```

будет создано 12 потоков обработки (8 для 25G карты и 4 для 10G)

В fastdpi.conf можно явно указать число потоков на мост с помощью параметра num\_threads:

```
# 41-00.x - 25G NIC
# 01-00.x - 10G NIC
in_dev=41-00.0:01-00.0
out_dev=41-00.1:01-00.1

num_threads=4
```

Такая конфигурация создаст 8 (num\_threads=4 \* 2 моста) потоков обработки.



СКАТ при планировании ядер учитывает NUMA node, к которой относятся ядра и карта: если карта на NUMA node 0, СКАТ назначит потоки обработчиков и диспетчеров также на NUMA node 0. Если ядер в NUMA node не хватает, СКАТ не запустится

Кроме потоков-обработчиков, для работы нужен также как минимум один поток-диспетчер (и значит, еще как минимум одно ядро), читающий rx-очереди интерфейсов. Задача диспетчера - чтобы пакеты, относящиеся к одному flow, попадали в один и тот же поток обработчика. Внутренняя архитектура работы с одним или множеством диспетчеров разительно отличается, поэтому СКАТ предоставляет несколько движков, конфигурируемых параметром `dpdk_engine` файла конфигурации `fastdp.conf`:

- `dpdk_engine=0` - read/write движок по умолчанию, один диспетчер на все;
- `dpdk_engine=1` - read/write движок с двумя потоками-диспетчерами: на каждое направление по диспетчеру;
- `dpdk_engine=2` - read/write движок с поддержкой RSS: для каждого направления создается `dpdk_rss` диспетчеров (по умолчанию `dpdk_rss=2`), таким образом, общее количество диспетчеров =  $2 * dpdk_rss$ ;
- `dpdk_engine=3` - read/write движок с отдельным диспетчером на каждый мост.

Далее подробно описываются все эти движки, особенности их настройки и области применения, но сначала - общее замечание про потоки диспетчеров.

## Явная привязка к ядрам

Можно задать в `fastdp.conf` явную привязку потоков к ядрам. За это отвечают параметры:

- `engine_bind_cores` - список номеров ядер для потоков-обработчиков
- `rx_bind_core` - список номеров ядер для потоков диспетчеров

Формат задания этих списков одинаков:

```
# 10G карты - 4 потока обработчика, 1 диспетчер на кластер
in_dev=01-00.0|02-00.0
out_dev=01-00.1|02-00.1

# Привязываем потоки обработки для кластера #1 к ядрам 2-5, диспетчер - к ядру 1
#   для кластера #2 к ядрам 7-10, диспетчер - к ядру 6
engine_bind_cores=2:3:4:5|7:8:9:10
rx_bind_core=1|6
```

Для бескластерного задания:

```
# 10G карты - 4 потока обработчика на карту
in_dev=01-00.0:02-00.0
out_dev=01-00.1:02-00.1
# 2 диспетчера (по направлениям)
dpdk_engine=1

# Привязка потоков обработчиков и диспетчеров
engine_bind_cores=3:4:5:6:7:8:9:10
rx_bind_core=1:2
```

Как уже отмечалось, потоки обработчиков и диспетчеров должны иметь выделенные ядра; не допускается привязывать несколько потоков к одному ядру, - СКАТ при этом выругается в

fastdpi\_alert.log и не будет запускаться.



Явная привязка к ядрам может быть применена только в экстренных случаях; обычно достаточно автоматической привязки. Для выяснения номеров ядер советуем запустить СКАТ с автоматической привязкой (без параметров `engine_bind_cores` и `rx_bind_core`) и посмотреть в `fastdpi_alert.log` дамп топологии системы: номер ядра - это `lcore`



При явной привязке СКАТ строго следует заданным в `fastdpi.conf` параметрам и не учитывает NUMA node, что может негативно сказаться на производительности (минус 10% - 20%)

## Загрузка потока диспетчера

Значение загрузки потока диспетчера, близкое к 100%, не говорит о том, что диспетчер не справляется: DPDK предполагает, что данные с картычитываются потребителем (это как раз и есть диспетчер) без каких-либо прерываний типа "пришли данные", поэтому диспетчер постоянно опрашивает состояние rx-очередей интерфейсов на наличие пакетов (так называемый `poll mode`). Если в течение N циклов опроса не принято ни одного пакета, диспетчер засыпает на несколько микросекунд, что вполне достаточно для снижения нагрузки на ядро до единиц процентов. Но если пакеты поступают раз в N-i циклов опроса, диспетчер не будет переходить в режим сна, и загрузка ядра будет 100%. Это нормально.



Загрузку потоков СКАТА можно посмотреть следующей командой:

```
top -H -p `pidof fastdpi`
```

Истинное состояние каждого диспетчера можно увидеть в `fastdpi_stat.log`, - в него, помимо прочего, периодически выводится статистика по диспетчерам вида:

```
[STAT      ][2020/06/15-18:17:17:479843]  [HAL][DPDK] Dispatcher statistics
abs/delta:
RX packets          drop (worker queue full)          | empty NIC RX |
Cluster #0:        0/0           0.0%/  0.0% | 98.0%/95.0% |
100500000/100500
```

здесь `empty NIC RX` - это и есть процент холостых опросов rx-очередей карт - абсолютный процент (с начала работы СКАТ) и относительный (дельта с последнего вывода в stat-лог). 100% - значит, входных пакетов нет, диспетчер работает вхолостую. Если относительный процент меньше 10 (то есть в более чем 90% опросов интерфейсов есть входные пакеты) - диспетчер не справляется и надо рассмотреть вариант с другим движком, где большее число диспетчеров.

Также хорошим индикатором, что текущий движок в целом не справляется, является ненулевое значение дельты для показателя drop (worker queue full). Это число отброшенных пакетов, которые диспетчер не смог отправить в поток обработки из-за переполнения входной очереди обработчика. Это значит, что обработчики не справляются с обработкой входящего трафика; причины могут быть две:

- либо слишком мало потоков-обработчиков, надо увеличить параметр num\_threads или выбрать другой движок (параметр dpdk\_engine);
- либо трафик сильно перекошен и большинство пакетов попадает в один-два обработчика, тогда как остальные свободны. В этой ситуации нужно анализировать структуру трафика. Можно попробовать увеличить или уменьшить на единицу число потоков-обработчиков, чтобы хеш-функция диспетчера раскидывала пакеты более равномерно (напомним, что номер потока обработки есть хеш\_пакета mod число\_обработчиков)

## **dpdk\_engine=0: Один диспетчер**

В этом режиме работы СКАТ создает один поток диспетчера на кластер. Диспетчер читает входящие пакеты со всех in\_dev и out\_dev устройств и раскидывает пакеты по потокам обработчиков. Подходит для 10G карт, выдерживает нагрузку до 20G и более (зависит от модели CPU и режима разбора туннелей [check\\_tunnels](#))



Общее число требуемых ядер равно числу обработчиков плюс одно ядро на диспетчера

СКАТ конфигурирует карты следующим образом:

- RX queue count = 1
- TX queue count = число потоков обработки. Потоки обработки пишут напрямую каждый в свою TX-очередь карты.

## **dpdk\_engine=1: Диспетчеры по направлению**

В этом режиме создается два потока диспетчера: один для направления от абонентов в inet (для in\_dev), другой - для направления из inet к абонентам (для out\_dev). Подходит для нагрузок свыше 20G (карты 25G, 40G).



Общее число требуемых ядер равно числу обработчиков плюс два ядра на диспетчера

СКАТ конфигурирует карты следующим образом:

- RX queue count = 1
- TX queue count = число потоков обработки. Потоки обработки пишут напрямую каждый в свою TX-очередь карты.

## **dpdk\_engine=2: Поддержка RSS**

В данном режиме задействуется RSS (receive side scaling) карты. Значение RSS задается в fastdp.conf параметром

```
dpdk_rss=2
```

Значение dpdk\_rss не должно быть менее 2. Для каждого направления создается dpdk\_rss диспетчеров.



Общее число требуемых ядер равно числу обработчиков плюс  $dpdk\_rss * 2$  на диспетчеры

Подходит для мощных карт 50G+ (то есть для СКАТ-100+). Если у вас 50G набрано из нескольких карт группировкой, данный режим вряд ли подойдет, так как для каждой карты из группы требует дополнительно как минимум 2 ядра (при dpdk\_rss=2). Лучше рассмотреть варианты dpdk\_engine=1 или dpdk\_engine=3.

СКАТ конфигурирует карты следующим образом:

- RX queue count = dpdk\_rss
- TX queue count = число потоков обработки. Потоки обработки пишут напрямую каждый в свою TX-очередь карты.

## **dpdk\_engine=3: Диспетчер на мост**

Для каждого моста создается отдельный поток диспетчера. Предназначен для конфигураций со множеством девайсов на входе и выходе:

```
in_dev=01-00.0:02-00.0:03-00.0  
out_dev=01-00.1:02-00.1:03-00.1  
dpdk_engine=3
```

Для данного примера создается три потока диспетчеров:

- для моста 01-00.0 ↔ 01-00.1
- для моста 02-00.0 ↔ 02-00.1
- для моста 03-00.0 ↔ 03-00.1



Общее число требуемых ядер равно числу обработчиков плюс число мостов

Данный движок предназначен для нескольких карт 25G/40G/50G в группе (то есть для СКАТ-100+)

СКАТ конфигурирует карты следующим образом:

- RX queue count = 1
- TX queue count = число потоков обработки. Потоки обработки пишут напрямую каждый в свою TX-очередь карты.