Архитектура распределенного коммутатора

Оригинал

В этом документе рассматриваются принципы устройства и ограничения подсистемы распределенного коммутатора (Distributed Switch Architecture или DSA), её взаимодействия с другими подсистемами, а также вопросы разработки драйверов и нерешенные вопросы разработки самой подсистемы.

Оглавление

Принципы устройства

Подсистема DSA предназначалась в основном для поддержки коммутаторов Ethernet компании Marvell (MV88E6xxx, линейка Link Street), использующих Linux, но развилась и для поддержки продукции других производителей.

Исходным принципом разработки было сохранение возможности применять неизмененные инструменты Linux, такие как bridge, iproute2, ifconfig для настройки и опроса коммутирующих портов сетевого устройства или обычного сетевого устройства.

Коммутатор Ethernet обычно имеет множество портов на передней панели и 1 или несколько портов CPU или управления. Подсистема DSA в настоящее время полагается на наличие порта управления, соединённого с контроллером Ethernet, способным получать кадры Ethernet от коммутатора. Это очень распространённое решение для всех видов коммутаторов Ethernet в продукции для дома и небольшого офиса (Small Home and Office) - маршрутизаторах, шлюзах и даже стоечных коммутаторах (top-of-rack). Этот хост-контроллер Ethernet далее называется ведущим (master) и cpu в описании и коде DSA.

D в аббревиатуре DSA означает Distributed (распределенный), поскольку подсистема разрабатывается для обеспечения возможности настройки и управления каскадом коммутаторов с использованием восходящих и нисходящих каналов Ethernet между ними. Эти порты называются портами dsa в терминологии и коде DSA. Набор соединённых между собой коммутаторов называется деревом коммутаторов (switch tree).

Для каждого порта на передней панели коммутатора DSA создаёт специализированные сетевые устройства, которые служат конечными точками для управления и потоков данных, используемыми сетевым стеком Linux. Эти специализированные сетевые интерфейсы называются ведомыми (slave) в терминологии и коде DSA.

Идеальным вариантом применения DSA является случай с поддержкой коммутатором Ethernet тега (switch tag), являющегося свойством оборудования, когда коммутатор помещает свой тег для каждого кадра Ethernet, получаемого или передаваемого в конкретные порты, чтобы помочь интерфейсу управления понять:

- из какого порта поступил кадр;
- по какой причини кадр был переслан;
- как передать созданный CPU трафик в конкретные порты.

Подсистема поддерживает коммутаторы, не способные вставлять и вырезать теги, но свойства в этом случае могут быть несколько ограничены (разделение трафика происходит на основе VLAN ID по портам).

Отметим, что в настоящее время DSA не создаёт сетевых интерфейсов для портов cpu и dsa.

- Портом cpu является сторона коммутатора Ethernet, обращённая к управляющему контроллеру, поэтому создание порта привело бы к дублированию функций, поскольку было бы 2 интерфейса для одного канала (conduit) master netdev и cpu netdev.
- Портами dsa являются просто каналы (conduit) между двумя и более коммутаторами и, поскольку они могут служить корректными сетевыми интерфейсами, в этой модели имеют смысл лишь нисходящие интерфейсы и восходящий интерфейс верхнего уровня.

Протоколы для тегов коммутаторов

DSA поддерживает фирменные протоколы тегов многих производителей, программно определяемый протокол для тегов и режим без тегов (tag-less, DSA_TAG_PROTO_NONE). Точный формат зависит от производителя, но обычно теги содержат:

- идентификатор порта Ethernet, связанного с приёмом или передачей кадра;
- указание причины, по которой кадр был передан в интерфейс управления.

Все протоколы тегов включены в файлы net/dsa/tag_*.c и реализуют методы структуры dsa_device_ops, описанные ниже.

Протокол тегов обычно относится к одной из указанных ниже категорий.

1. Зависящий от коммутатора заголовок кадра размещается перед заголовком Ethernet, сдвигая вправо (с точки зрения первичного анализатора кадров DSA) поля MAC DA, MAC SA, EtherType и все данные L2 (payload).
2. Зависящий от коммутатора заголовок кадра размещается перед полем EtherType, сохраняя позиции MAC DA и MAC SA с точки зрения первичного анализатора кадров DSA, но сдвигает поля EtherType и данные L2 вправо.
3. Зависящий от коммутатора заголовок кадра размещается в конце пакета, сохраняя на месте все заголовки кадра и не меняя представление кадра с точки зрения первичного анализатора DSA.

Протокол тегов может помечать все кадры тегами коммутатора одного размера или применять теги переменного размера (например, пакеты с временными метками PTP могут требовать расширения тегов коммутатора, а также возможно использование тегов разного размера для передачи и приёма). В любом случае драйвер протокола тегов должен заполнять структуру dsa_device_ops::needed_headroom и/или dsa_device_ops::needed_tailroom, размер которой в октетах равен большему из размеров заголовка/трейлера в кадре. DSA автоматически корректирует MTU на первичном интерфейсе с учётом дополнительных полей, чтобы пользовательские порты DSA могли поддерживать стандартное значение MTU (размер данных L2) в 1500 октетов. Свойства needed_headroom и needed_tailroom

применяются для запроса у сетевого стека (по возможности - best-effort) выделения пакетов с большим пространством, чтобы вталкивание тега коммутатора при передаче пакета не вызывало повторного выделения из-за нехватки памяти.

Хотя от приложений не ожидается синтаксический анализ связанных с DSA заголовков кадров, формат протокола тегов в линии представляет двоичный интерфейс приложений (Application Binary Interface или ABI), раскрываемый ядром пользовательскому пространству для декодирования (такой как libpcap). Драйвер протокола тегов должен помещать в поле proto структуры dsa_device_ops значение, однозначно описывающее характеристики требуемого взаимодействия между оборудованием коммутатора и драйвером пути данных - смещение каждого битового поля в заголовке кадра и любую обработку с учётом состояния, требуемую для работы с кадром как может требоваться, например, для меток времени PTP).

С точки зрения сетевого стека все коммутаторы внутри дерева коммутаторов DSA используют один протокол тегов. Если пакет проходит через узлы коммутации (fabric) нескольких коммутаторов, зависящий от коммутатора тег вставляет первый такой модуль, получивший пакет. Заголовок обычно содержит сведения от типе (является ли кадр управляющим и его нужно пересылать его в CPU или просто кадром данных для пересылки). Кадры управления следует декапсулировать лишь на программном пути данных, а кадры данных могут автономно пересылаться в направлении других пользовательских портов других коммутаторов того же модуля коммутации (fabric) и в этом случае декапсулировать пакет должен внешний (последний) порт коммутатора.

Отметим, что в некоторых случаях формат тегов, используемых коммутатором-листом (не соединён напрямую с CPU) отличается от формата, который видит сетевой стек. Примером этого могут служить деревья коммутаторов Marvell, где порт CPU можно настроить на использование формата DSA или Ethertype DSA (EDSA), а каналы DSA настроены на применение более короткого (без Ethertype) формата кадров DSA для снижения издержек при автономной пересылке пакетов. Если дерево коммутаторов DSA настроено для протокола тегов EDSA, операционная система по-прежнему будет видеть пакеты с тегами EDSA от листовых коммутаторов, которые помечают пакеты более короткими заголовками DSA. Это обусловлено тем, что коммутатор Marvell, напрямую соединённый с CPU настроен на преобразование тегов между DSA и EDSA (это просто добавление или удаление ETH_P_EDSA EtherType и некоторых октетов заполнения).

Возможно каскадирование коммутаторов DSA даже при несовместимости протокола тегов. В таком случае в модуле коммутации (fabric) не будет каналов DSA и каждый коммутатор будет отдельным деревом коммутаторов DSA. Каналы DSA рассматриваются просто как пара из ведущего DSA (обращённый наружу порт восходящего коммутатора DSA) и порта CPU (обращённый внутрь порт нисходящего коммутатора DSA).

Протокол тегов присоединённого коммутатора DSA можно увидеть из атрибута sysfs dsa/tagging ведущего коммутатора DSA с помощью команды

```
cat /sys/class/net/eth0/dsa/tagging
```

Если оборудование и драйвер позволяют это, протокола тегов дерева коммутатора DSA можно сменить в процессе работы. Это делается путём записи имени нового протокола тегов в тот же атрибут устройства sysfs, который указан выше (DSA master и все подключённые порты коммутатора должны быть на это время переведены в состояние down).

Желательно, чтобы все протоколы тегов можно было протестировать с помощью макетного драйвера dsa_loop, который можно присоединить к любому сетевому интерфейсу. Цель заключается в том, чтобы любой сетевой интерфейс был способен передавать один и тот же пакет одним способом, а средству маркировки следует одинаково декодировать один и тот же пакет, независимо от драйвера, применяемого для пути управления в коммутаторе и драйвера, применяемого для DSA master.

Передача пакета осуществляется через вызов функции xmit. В переданной структуре sk_buff элемент *skb содержит указатель skb->data на skb_mac_header(skb), т. е. MAC-адрес получателя, а в переданной структуре net_device элемент *dev представляет виртуальный пользовательский сетевой интерфейс DSA, аппаратному «дубликату» которого должен быть направлен пакет (т. е. swp0). Задача этого метода состоит в подготовке skb так, чтобы коммутатор понимал, какой выходной порт использовать для пакета (и не передавал его в другие порты). Обычно это реализуется вталкиванием заголовка кадра. Проверка достаточности пространства в голове и хвосте skb не требуется при корректном указании needed_headroom и needed_tailroom, поскольку DSA гарантирует наличие свободного пространства перед вызовом этого метода.

Для приёма пакета служит функция rcv. В переданной структуре sk_buff элемент *skb содержит указатель skb->data на skb_mac_header(skb) + ETH_ALEN октетов, т. е. на место, где был бы первый октет EtherType, если бы в кадре не было тега. Роль этого метода заключается в использовании заголовка кадра, настройке skb->data для указания на реально первый октет EtherType и изменения skb->dev для указания виртуального пользовательского сетевого интерфейса DSA, соответствующего физическому порту коммутатора, на котором был получен пакет.

Поскольку протоколы тегов категории 1 и 2 нарушают программное (а чаще всего и аппаратное) разбиение пакета в DSA master, такие свойства, как RPS[1] на DSA будут нарушены. Модель DSA справляется с этим, устанавливая ловушку в диссекторе потоков и сеняя смещение, по которому можно найти заголовок IP в кадре с тегом, как его видит DSA master. Это автоматизируется на основе значения overhead в протоколе тегов. Если не все пакеты имеют одинаковый размер, модуль работы с тегами может реализовать метод flow_dissect структуры dsa_device_ops и переопределить принятое по умолчанию поведение, задавая корректное смещение в каждом принятом пакете. Модули включения тегов в конец кадра не вызывают проблем для диссекторов потоков.

Выгрузка контрольных сумм должна работать с тегами категории 1 и 2, когда драйвер DSA master декларирует NETIF_F_HW_CSUM в vlan_features и просматривает csum_start и csum_offset. В этих случаях DSA меняет начало и смещения на величину размера тега. Если драйвер DSA master продолжает использовать устаревшее значение NETIF_F_IP_CSUM или NETIF_F_IPV6_CSUM в vlan_features, выгрузка может работать лишь при условии, когда оборудование выгрузки уже ожидает этот конкретный тег (возможно по соответствию производителя). Ведомые устройства DSA наследуют эти флаги от ведущего порта и драйвер корректно возвращается к программной обработке контрольных сумм, когда заголовок IP размещается не там, где его ждёт оборудование. Если описанная проверка неэффективна, пакеты могут проходить через сеть без подходящей контрольной суммы (в поле контрольной суммы будет указано значение для псевдозаголовка IP). Для категории 3, когда оборудование разгрузки не ждёт

[1]Receive Packet Steering - управление приёмом пакетов.

использования тегов коммутатора, контрольная сумма должна рассчитываться до вставки тега (т. е. в модуле работы с тегами). Иначе DSA будет включать концевой тег в расчёт (программный или аппаратный) контрольной суммы. Когда коммутатор вырежет тег при передаче, он оставит некорректную контрольную сумму на месте.

В силу разных причин (чаще всего из-за тегов категории 1, связанных с не понимающими DSA ведущими устройствами, может искажаться то, что ведущий считает MAC DA) протокол тегов может требовать от DSA работы в неразборчивом (promiscuous) режиме для приёма кадров независимо от значения MAC DA. Это можно сделать установкой свойства promisc_on_master в структуре dsa_device_ops. Отметим, что это предполагает драйвер ведущего устройства без поддержки DSA, что является нормой.

Ведущие сетевые устройства

Ведущие сетевые устройства - это обычные, не изменённые драйверы сетевых устройств Linux для интерфейсов Ethernet CPU/управления. Таким драйверам иногда может требоваться знать включён ли DSA (например, для включения или выключения конкретных свойств разгрузки), но подтверждена работа подсистемы DSA со стандартными для отрасли драйверами e1000e, mv643xx_eth и т. п. без внесения в них изменений. Такие сетевые устройства часто называют «канальными» (conduit), поскольку они служат каналом между процессором хоста и оборудованием коммутатора Ethernet.

Ловушки сетевого стека

При использовании master netdev с DSA в сетевой стек помещается небольшая ловушка для обработки подсистемой DSA используемого коммутатором Ethernet протокола тегов. DSA делает это путём регистрации в сетевом стеке конкретного (фиктивного) типа Ethernet (далее skb->protocol), который называют также ptype или packet_type. Ниже показана типичная последовательность приёма кадра Ethernet ведущим сетевым устройством (например, e1000e).

1. Получение сигнала прерывания:
   - вызов функции приёма;
   - базовая обработка пакета (определение размера, статуса и т. п.);
   - подготовка пакета к обработке на уровне Ethernet вызовом функции eth_type_trans.
2. net/ethernet/eth.c

```
eth_type_trans(skb, dev)
        if (dev->dsa_ptr != NULL)
                -> skb->protocol = ETH_P_XDSA
```

3. drivers/net/ethernet/*

```
netif_receive_skb(skb)
        -> итерации по зарегистрированным packet_type
                -> вызов обработчика для ETH_P_XDSA, вызов dsa_switch_rcv()
```

4. net/dsa/dsa.c

```
-> dsa_switch_rcv()
        -> вызов зависящего от протокола тегов обработчика в net/dsa/tag_*.c
```

5. net/dsa/tag_*.c
   - проверка и вырезание (протокола) тега для определения порта-источника;
   - нахождение сетевого устройства по порту;
   - вызов eth_type_trans() с ведомым сетевым устройством DSA;
   - вызов netif_receive_skb().

После этого ведомые сетевые устройства DSA получают обычные кадры Ethernet, которые может обрабатывать сетевой стек.

Ведомые сетевые устройства

Ведомые сетевые устройства, создаваемые DSA, собираются в стек «поверх» их ведущего сетевого устройства и каждый из этих сетевых интерфейсов отвечает за выполнение функций конечной точки управления и потоков данных для (каждого) порта на передней панели коммутатора. Эти интерфейсы специализированы на

- вставку и удаление тега коммутатора (при наличии) при передаче и приёме через соответствующие порты;
- запрос у коммутатора операций ethtool (статистика, состояние канала, WoL[1], дампы регистров …);
- управление внешним или внутренним PHY (канал, автосогласование и т. п.).

Эти ведомые сетевые устройства имеют указатели на пользовательские функции net_device_ops и ethtool_ops, что позволяет DSA вводить уровень разделения между сетевым стеком/ethtool и реализацией драйвера коммутатора.

При передаче кадра с этих ведомых сетевых устройств DSA будет просматривать, какой протокол тегов коммутатора зарегистрирован в данный момент с этими сетевыми устройствами и вызывать соответствующую подпрограмму отправки, которая позаботится о добавлении подходящего тега коммутатора в кадры Ethernet.

Кадры затем помещаются в очередь на передачу с использованием функции ведущего сетевого устройства ndo_start_xmit(). Поскольку кадры содержат соответствующий тег, коммутатор Ethernet сможет обрабатывать эти входящие кадры от интерфейса управления и доставлять в физический порт коммутатора.

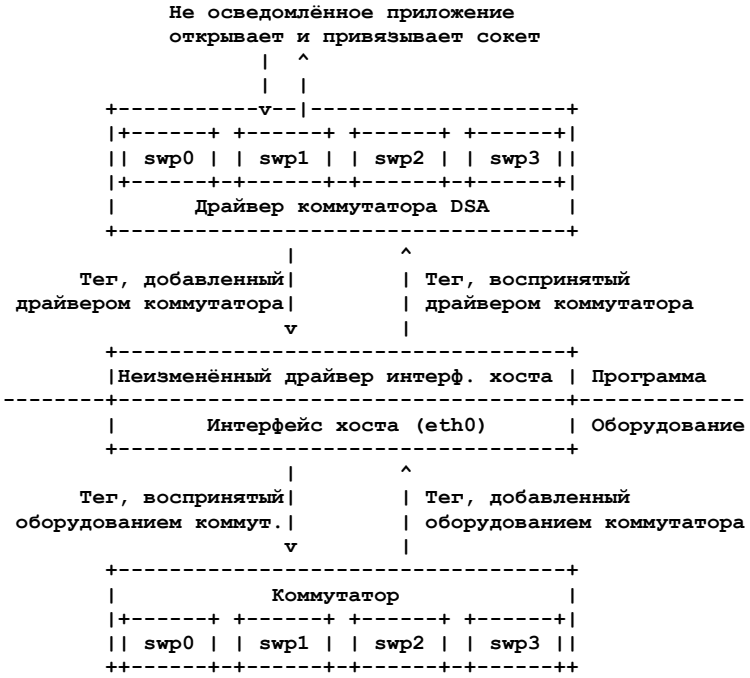
При использовании нескольких портов CPU можно разместить устройство LAG (bonding/team) между ведомыми устройствами DSA и физическими DSA master. Устройство LAG также будет DSA master, но ведомые устройства LAG

[1] Wake-on-LAN - пробуждение из ЛВС.

продолжают оставаться DSA master (просто с ними не связывается физический порт; это нужно для восстановления в случае потери LAG DSA master). Таким образом, путь данных LAG DSA master используется асимметрично. На RX обработчик ETH_P_XDSA, который вызывает dsa_switch_rcv(), вызывается редко (на физическом DSA master, LAG slave). Поэтому путь данных RX в LAG DSA master не используется. С другой стороны, TX работает линейно - dsa_slave_xmit вызывает dsa_enqueue_skb, а эта функция вызывает dev_queue_xmit в направлении LAG DSA master. Последняя вызывает dev_queue_xmit в направлении одного или другого физического DSA master и в обоих случаях пакет выходит из системы по аппаратному пути в направлении коммутатора.

Графическое представление

Вид DSA с точки зрения сетевого устройства показан на рисунке ниже.



Ведомая шина MDIO

Для записи и считывания со встроенного в коммутатор PHY в DSA создаётся ведомая шина MDIO, которая позволяет конкретному сетевому драйверу перехватывать и перенаправлять операции чтения и записи MDIO для конкретного адреса PHY. В большинстве подключённых к MDIO коммутаторов эти функции будут использовать прямую или косвенную адресацию PHY для возврата значений стандартных регистров MII встроенных в коммутатор PHY, позволяя библиотеке PHY возвращать статус канала, страница партнёра по соединению, результаты автосогласования и т. п.

Для коммутаторов Ethernet, имеющих внутренние и внешние шины MDIO, ведомую шину MII можно использовать для мультиплексирования и демультиплексирования операций чтений и записи MDIO в направлении внешних или внутренних устройств MDIO, к которым коммутатор может быть подключён, - внутренние и внешние PHY и даже внешние коммутаторы.

Структуры данных

Структуры данных DSA определены в файлах include/net/dsa.h и net/dsa/dsa_priv.h.

- Структура dsa_chip_data содержит данные конфигурации платформы для данного коммутирующего устройства. Эта структура описывает родительское устройство коммутатора, его адрес, а также различные свойства портов (имена, метки) и указывает таблицу маршрутизации (при каскадировании коммутаторов).
- Структура dsa_platform_data содержит данные конфигурации платформы, которые могут указывать набор структур dsa_chip_data при каскадировании коммутаторов. Структура должна указывать ведущее сетевое устройство, к которому подключено это дерево коммутаторов.
- Структура dsa_switch_tree назначенная ведущему сетевому устройству в dsa_ptr. Эта структура ссылается на структуру dsa_platform_data и указывает протокол тегов, поддерживаемый деревом коммутаторов, а также ловушки функций приёма и передачи, которые следует вызывать и сведения о подключённом напрямую коммутаторе (порт CPU). Для указания отдельных коммутаторов дерева применяется набор структур dsa_switch.
- Структура dsa_switch описывает коммутирующее устройство в дереве, ссылаясь на dsa_switch_tree в качестве обратного указателя, а также указывает ведомые и ведущее сетевые устройства резервную структуру dsa_switch_ops.
- Структура dsa_switch_ops содержит указатели на функции, описанные ниже.

Ограничения для устройств

Отсутствие сетевых устройств CPU/DSA

DSA в настоящее время не создаёт ведомых сетевых устройств для портов CPU и DSA, как указано выше. Это может вызывать проблемы в некоторых случаях:

- невозможность извлечь значения счётчиков статистики порта CPU с использованием ethtool, что может осложнять отладку коммутаторов MDIO, подключённых через интерфейс xMII;
- невозможность настроить для порта CPU параметры канала с помощью подключённого к нему контроллера Ethernet (см. http://patchwork.ozlabs.org/patch/509806/);
- невозможность настроить конкретные VLAN ID и транковые VLAN между коммутаторами в каскаде.

Распространённые ошибки при использовании DSA

После того, как ведущее сетевое устройство настроено для DSA (dev->dsa_ptr отлично от NULL) и размещённый за ним коммутатор ожидает протокол тегов, этот сетевой интерфейс можно использовать лишь в качестве интерфейса «трубы» (conduit). Передача пакетов напрямую через этот интерфейс (например, создание сокета с использованием этого интерфейса) не потребует проходить через функцию передачи протокола тегов, поэтому ожидающий тег коммутатор Ethernet на другой стороне обычно будет отбрасывать такие пакеты.

Взаимодействия с другими подсистемами

Ниже указаны подсистемы, используемые DSA в настоящее время:

- библиотека MDIO/PHY - drivers/net/phy/phy.c, mdio_bus.c
- switchdev - net/switchdev/*
- дерево устройств для различных функций of_*;
- devlink - net/core/devlink.c.

Библиотека MDIO/PHY

Сетевые устройства, раскрываемые DSA, не обязательно сопрягаются с устройствами PHY (struct phy_device, заданная в include/linux/phy.h), но подсистема DSA обрабатывает все возможные комбинации:

- внутренние устройства PHY, встроенные в оборудование коммутаторов Ethernet;
- внешние устройства PHY, подключённые через внутреннюю или внешнюю шину MDIO;
- внутренние устройства PHY, подключённые через внутреннюю шину MDIO;
- специальные устройства PHY без автоматического согласования и управления через: SFP, MoCA (fixed PHY).

Настройка PHY выполняется с помощью функции dsa_slave_phy_setup() и базовая логика приведена ниже.

- При использовании дерева устройств (Device Tree) устройство PHY отыскивается с помощью стандартного свойства phy-handle и при обнаружении это устройство PHY и регистрируется функцией of_phy_connect().
- Если используется дерево устройств и устройство PHY является «фиксированным» (т. е. соответствует определению PHY, управляемого не через MDIO, как указано в файле Documentation/devicetree/bindings/net/fixed-link.txt), PHY регистрируется и подключается «прозрачно» с использованием специального драйвера фиксированной шины MDIO.
- Если устройство PHY встроено в коммутатор, как это часто бывает в автономных коммутаторах, PHY проверяется с использованием специальной шины MII, созданной DSA.

SWITCHDEV

DSA напрямую использует SWITCHDEV при взаимодействии с уровнем моста и, более конкретно, с фильтрацией VLAN при настройке VLAN на портах ведомых сетевых устройств. На сегодняшний день DSA поддерживает в SWITCHDEV лишь объекты FDB и VLAN.

Devlink

DSA регистрирует одно устройство devlink на физический коммутатор в модуле коммутации (fabric). Для каждого устройства devlink каждый физический порт (пользовательские порты, порты CPU, каналы DSA и неиспользуемые порты) раскрывается как порт devlink.

Драйверы DSA могут использовать указанные ниже свойства devlink.

Регионы

Отладочное средство, позволяющее пользовательскому пространству выводить определяемые драйвером области аппаратной информации в низкоуровневом двоичном формате. Поддерживаются глобальные регионы и регионы по портам. Возможен экспорт регионов devlink даже для данных, которые уже тем или иным способом раскрыты стандартным программам пользовательского пространства iproute2 (ip-link, bridge), такие как таблицы адресов и VLAN. Это может быть полезно, например, когда таблицы содержат связанные с оборудованием детали, которые не видны через абстракцию iproute2, или для проверки таблиц на портах, не являющихся пользовательскими, которые не видны iproute2, поскольку для них не регистрируются сетевые интерфейсы.

Параметры

Свойство, которое позволяет пользователям настраивает некоторые низкоуровневые регуляторы, относящиеся к устройству. Драйвер может использовать применимые базовые параметры или добавлять связанные с устройством параметры devlink.

Ресурсы

Средство мониторинга, позволяющее пользователям видеть степень использования некоторых аппаратных таблиц в устройстве, таких как FDB, VLAN и т. п.

Общие буферы

Свойство QoS для настройки и разделения памяти и резервирования кадров по портам на входном и выходном направлении так, чтобы объёмный трафик с низким приоритетом не препятствовал обработке высокоприоритетного важного трафика.

Дополнительные сведения можно найти в файлах каталога Documentation/networking/devlink/.

Дерево устройств

В DSA имеется фиксированная привязка, описанная в файле Documentation/devicetree/bindings/net/dsa/dsa.txt. Вспомогательные функции библиотеки PHY/MDIO, такие как of_get_phy_mode(), of_phy_connect(), часто применяются для запроса связанных с PHY деталей по портам - соединения интерфейсов, размещение шины MDIO и т. п.

Разработка драйверов

Драйверы коммутаторов DSA должны реализовать структуру dsa_switch_ops, содержащую описанные ниже элементы.

Проба, регистрация и срок действия устройства

Коммутаторы DSA являются обычными структурами device на шинах (платформа, SPI, I2C, MDIO и пр.). Платформа DSA не участвует в их зондировании ядром устройства.

Регистрация коммутатора с точки зрения драйвера означает передачу действительного указателя на структуру dsa_switch функции dsa_register_switch(), обычно из функции проверки драйвера коммутатора. Ниже указаны элементы предоставляемой структуры, которые должны иметь действительные значения:

- ds->dev используется для синтаксического анализа данных узла OF или платформы;
- ds->num_ports используется для создания списка портов этого коммутатора и проверки индексов портов, предоставляемых в узле OF;
- ds->ops содержит указатель на структуру dsa_switch_ops, содержащую реализации методов ;
- ds->priv - обратный указатель на частную структуру данных драйвера, которая может извлекаться во всех последующих обратных вызовах (callback) методов DSA.

Кроме того, могут указываться перечисленные ниже флаги структуры dsa_switch для задания конкретного поведения драйвера со стороны ядра DSA. Описание поведения приведено в комментариях к файлу include/net/dsa.h.

- ds->vlan_filtering_is_global;
- ds->needs_standalone_vlan_filtering;
- ds->configure_vlan_while_not_filtering;
- ds->untag_bridge_pvid;
- ds->assisted_learning_on_cpu_port;
- ds->mtu_enforcement_ingress;
- ds->fdb_isolation.

DSA сохраняет внутри массив деревьев (групп) коммутаторов, являющийся глобальным для ядра, и связывает с деревом структуру dsa_switch при регистрации. Идентификатор дерева, с которым связан коммутатор, определяется первым числом u32 number свойства dsa,member узла OF коммутатора (0 при отсутствии). Идентификатор коммутатора в дереве определяется вторым числом u32 в том же свойстве OF (0 при отсутствии). Регистрация нескольких коммутаторов с совпадающими идентификаторами коммутатора и дерева некорректна и вызывает ошибку. При использовании данных платформы разрешён 1 коммутатор и одно дерево.

Для дерева с несколькими коммутаторами зондирование выполняется асимметрично. Первые N-1 вызывающих dsa_register_switch() лишь добавляют свои порты в список портов дерева (dst->ports) и каждый порт имеет обратный указатель на связанный с ним коммутатор (dp->ds). Эти коммутаторы заранее завершают свои вызовы dsa_register_switch(), поскольку функция dsa_tree_setup_routing_table() определяет, что дерево ещё не завершено (не все порты, указанные каналами DSA, присутствуют в списке портов дерева). Дерево становится полным, когда последний коммутатор вызывает dsa_register_switch() и это запускает эффективное продолжение инициализации (включая вызов ds->ops->setup()) для всех коммутаторов этого дерева, как часть контекста вызова функции зондирования последним коммутатором.

Обратные к регистрации действия выполняются при вызове dsa_unregister_switch(), удаляющем порты коммутатора из списка портов дерева. При отмене регистрации первого коммутатора все дерево уничтожается.

Для драйверов коммутаторов DSA обязательна реализация обратного вызова shutdown() соответствующей шины и вызов dsa_switch_shutdown() из неё (минимальный вариант полного демонтажа, выполняемого dsa_unregister_switch()). Причина заключается в том, что DSA сохраняет ссылку на ведущее сетевое устройство и при отвязывании (unbind) основного устройства при отключении ссылка DSA будет препятствовать завершению операции.

Должна вызываться функция dsa_switch_shutdown() или dsa_unregister_switch(), но не обе, и модель драйвера устройства разрешает вызывать метод шины remove(), даже если уже вызвана функция shutdown(). Поэтому ожидается реализация в драйверах метода взаимного исключения remove() и shutdown() путём установки drvdata = NULL после вызова любой из этих функций и проверка значения drvdata (NULL) перед выполнением какого-либо действия.

После вызова dsa_switch_shutdown() или dsa_unregister_switch() обратные вызовы через предоставленный указатель на dsa_switch_ops не могут выполняться и драйвер может освободить структуры данных, связанные с dsa_switch.

Настройка коммутатора

get_tag_protocol

Указывает, какой тип протокола тегов поддерживается и должно содержать действительное значение из перечисляемых dsa_tag_protocol. Возвращаемая информация не является статической, драйверу передаётся номер порта CPU и протокол тегов встроенного вышележащего коммутатора стека на случай аппаратных ограничений на формат протокола тегов.

change_tag_protocol

Когда у принятого по умолчанию протокола тегов возникают проблемы с ведущим или иные проблемы, драйвер может поддерживать смену протокола в процессе работы через свойство дерева устройств или sysfs. После этого вызову get_tag_protocol следует возвращать текущий используемый протокол.

setup

Функция установки (setup) коммутатора, отвечающая за установку приватной структуры dsa_switch_ops со всем необходимым - планы регистров, прерывания, мьютексы, блокировки и т. п. Предполагается, что эта функция должным образом настроит конфигурацию коммутатора для отделения сетевых интерфейсов друг от друга, т. е. им следует быть изолировать самому оборудованию коммутатора, обычно путём создания для каждого VLAN ID по порту и разрешая пересылку лишь между портом CPU и конкретным портом. Не используемые платформой порты следует отключать. После вызова этой функции коммутатор предполагается полностью настроенным и готовым к выполнению запросов любого вида. Рекомендуется выполнять программный сброс коммутатора в этой функции настройки для исключения настроек, которые мог установить прежний программный агент, такой как начальный загрузчик или микрокод. Методом, отвечающим за отказ от всех применимых выделений или операций, выполненных здесь, является teardown.

port_setup и port_teardown

Методы для создания и удаления структур данных по портам. Они обязательны для таких операций, как регистрация и дерегистрация регионов портов devlink. Порт может быть удалён лишь в том случае, когда он был создан ранее. Порт может быть установлен на время зондирования, а затем сразу удалён, например, в случае отсутствия PHY. При этом зондирование коммутатора DSA продолжается без такого порта.

port_change_master

Метод, с помощью которого можно изменить связность (аффинность - ассоциация для целей завершения трафика) пользовательского порта и порта CPU. По умолчанию все пользовательские порты из дерева связаны с первым доступным портом CPU, имеющим смысл для данного порта (в большинстве случаев это означает, что все пользовательские порты в дереве связаны с одним портом CPU, за исключением топологии H, описанной в представлении 2c0b03258b8b). Аргумент port указывает индекс пользовательского порта, а master представляет новое ведущее устройство DSA master net_device. Порт CPU, связанный с новым ведущим, можно определить поиском в struct dsa_port *cpu_dp = master->dsa_ptr. Кроме того, ведущий может быть устройством LAG, в котором все ведому устройства являются физическими DSA master. Ведущие LAG DSA имеют действительный указатель master->dsa_ptr, однако он не уникален и является дубликатом dsa_ptr первого физического DSA (LAG slave). В случае LAG DSA последующий вызов port_lag_join будет выполняться отдельно для физических портов CPU, связанных с физическими DSA master, с запросом на создание аппаратной группы LAG, связанной с интерфейсом LAG.

Управление PHY и каналом

get_phy_flags

Некоторые коммутаторы имеют интерфейсы для различных типов Ethernet PHY. Если драйвер PHY библиотеки PHY нужны сведения, которые он не может получить сам (например, из регистров, отображённых на память коммутатора), этой функции следует возвращать 32-битовую маску «флагов», которые являются приватными для драйвера коммутатора и драйвера Ethernet PHY из каталога drivers/net/phy/*.

phy_read

Функция вызывается шиной MDIO ведомого DSA для считывания регистров MDIO порта коммутатора. При неудаче считывания функция возвращает 0xffff для каждой операции чтения. Для Ethernet PHY встроенных коммутаторов этой функции следует разрешать считывание статуса канала, результаты автосогласования, страницы партнёра по каналу и т. п.

phy_write

Функция вызывается шиной MDIO ведомого DSA для записи в регистры MDIO порта коммутатора. При невозможности записи функция возвращает отрицательный код ошибки.

adjust_link

Функция вызывается библиотекой PHY, когда ведомое сетевое устройство присоединяется к устройству PHY. Функция отвечает за подобающую настройку параметров канала для порта коммутатора - скорости, дуплекса, паузы на основе представленных в phy_device значений.

fixed_link_update

Функция вызывается библиотекой PHY и, в частности, драйвером фиксированного PHY, спрашивающим у драйвера коммутатора параметры канала, которые невозможно согласовать автоматически или получить путём чтения регистров PHY через MDIO. Это особенно полезно для определённых типов оборудования, таких как QSGMII, MoCA или иные типы не управляемых через MDIO устройств PHY, где информация получается вне основного соединения.

Операции ethtool

get_strings

Функция ethtool для запроса строк драйвера, которые обычно содержат статистику, приватные флаги и т. п.

get_ethtool_stats

Функция ethtool для запроса статистики по порта. DSA накладывает статистику ведомых сетевых устройств на общую статистику - счётчики RX/TX из сетевого устройства со статистикой драйвера коммутатора по портам.

get_sset_count

Функция ethtool для запроса числа элементов статистики.

get_wol

Функция ethtool для получения настроек Wake-on-LAN по портам. В некоторых реализациях функция может также запрашивать настройки Wake-on-LAN ведущего сетевого устройства, если этот интерфейс нужен для участия в Wake-on-LAN.

set_wol
Функция ethtool для настройки Wake-on-LAN по портам.

set_eee
Функция ethtool для настройки параметров EEE (Green Ethernet) порта коммутатора, которая может вызывать библиотеку PHY для включения EEE на уровне PHY, если это уместно. Этой функции следует включать EEE на контроллере MAC и в логике обработки данных порта коммутатора.

get_eee
Функция ethtool для запроса настроек EEE порта коммутатора, которой следует возвращать состояние EEE контроллера MAC порта коммутатора и логики обработки данных, а также запрашивать у PHY текущие настройки EEE.

get_eeprom_len
Функция ethtool, возвращающая для данного коммутатора размер EEPROM в байтах.

get_eeprom
Функция ethtool, возвращающая для данного коммутатора содержимое EEPROM.

set_eeprom
Функция ethtool, записывающая указанные данные в EEPROM данного коммутатора.

get_regs_len
Функция ethtool, возвращающая для данного коммутатора размер регистров.

get_regs
Функция ethtool, возвращающая для коммутатора Ethernet содержимое внутренних регистров. Для этой функции может потребоваться код из пользовательского пространства для точного вывода значений и регистров.

Управление питанием

suspend
Функция, вызываемая устройством платформы DSA, когда система переходит в режим приостановки (suspend). Функции следует приостанавливать все действия коммутатора Ethernet, оставляя активными порты, участвующие в Wake-on-LAN, а также дополнительную логику пробуждения, если она поддерживается.

resume
Функция, вызываемая устройством платформы DSA при возобновлении работы системы (resume). Функции следует восстанавливать все действия коммутатора Ethernet и реконфигурировать коммутатор, чтобы он находился в полностью активном состоянии.

port_enable
Функция, вызываемая функцией ndo_open ведомого устройства DSA при административной активации порта. Функции следует полностью включать данный физический порт. DSA заботится о маркировке порта BR_STATE_BLOCKING, если порт входит в мост, и BR_STATE_FORWARDING - в противном случае, а также распространении этих изменений в оборудование.

port_disable
Функция, вызываемая функцией ndo_open ведомого устройства DSA при административном отключении порта. Функции следует полностью выключать данный физический порт. DSA заботится о маркировке порта BR_STATE_DISABLED и распространении этих изменений в оборудование, если отключенный порт входит в мост.

Адресные базы данных

Предполагается, что коммуникационное оборудование имеет таблицу для записей FDB, однако не все записи активны в каждый момент времени. База адресных данных является подмножеством (разделом) записей FDB, которые активны (могут быть сопоставлены с изучением адресов на RX или поиском в FDB на TX) в зависимости от состояния порта. В этом документе база адресных данных иногда называется идентификаторами фильтрации (FID или Filtering ID), хотя базовая реализация может выбрать все, что поддерживает оборудование. Например, все порты, принадлежащие мосту, не поддерживающему VLAN (в данный момент), предполагаются изучающими адреса отправителей в адресной базе, связанной драйвером с этим мостом (а не с другими мостами без поддержки VLAN). При пересылке и поиске в FDB пакета, принятого на порту моста без поддержки VLAN, следует обеспечивать возможность поиска не связанной с VLAN записи FDB с тем же MAC DA, что и в пакете, имеющуюся для другого порта того же моста. В тоже время, процесс поиска в FDB должен быть способен не найти (игнорировать) запись FDB с тем же MAC DA, что и в пакете, если эта запись указывает порт другого моста без поддержки VLAN (связана с другой базой адресов).

Каждой VLAN каждого выгруженного (offloaded) моста с поддержкой VLAN следует иметь связанную базу адресных данных, которая является общей для всех портов этой VLAN, но не относится к портам других мостов, с тем же VID.

В этом контексте базе данных без VLAN соответствуют все пакеты независимо от VLAN ID (поиск лишь по адресу MAC), а в базе данных с VLAN сопоставление выполняется по VLAN ID из заголовка 802.1Q (или pvid, если тега нет).

На уровне моста записи FDB без поддержки VLAN имеют специальное значение VID 0, а в записях FDB с поддержкой VLAN идентификаторы VID отличны от 0. Отметим что не знающий о VLAN мост может иметь записи FDB с VLAN (ненулевой идентификатор VID), а мост с поддержкой VLAN может иметь записи FDB без VLAN. Как и аппаратные, программные мосты поддерживают раздельные базы адресных данных и выгружают в оборудование записи FDB, относящиеся к этим базам, через switchdev асинхронно относительно моментов их активации (деактивации).

Когда пользовательский порт работает в автономном режиме, его драйверу следует настроить порт на использование отдельной базы данных, называемой приватной базой порта. Она отличается от описанных выше форм базы данных и ей следует как можно меньше затруднять операции автономного порта (входящий пакет, исходящий пакет в порт CPU). Например, ей не следует пытаться изучать на входе адреса MAC SA, поскольку обучение является функцией уровня моста, а это автономный порт и адреса будут бессмысленно занимать пространство. Без изучения адресов приватной базе порта следует быть пустой в тривиальной реализации и в таком случае все входящие пакеты следует просто пересылать в порт CPU.

Порты DSA (каскад) и CPU называют также разделяемыми (shared) поскольку они обслуживают несколько адресных баз, а база данных, с которой следует связывать пакет, обычно встраивается в тег DSA. Это означает, что порт CPU может одновременно транспортировать пакеты от автономного порта (классифицированные оборудованием в 1 адресной базе) и от порта моста (классифицированы в другой базе адресов).

Драйверы коммутаторов, соответствующие определенным критериям, могут оптимизировать тривиальную конфигурацию, удаляя порт CPU из домена лавинной рассылки в коммутаторе и просто программируя оборудование записями FDB, указывающими на порт CPU, для которого известен интерес программ в пакетах с этими адресами MAC. Пакеты, не соответствующие известной записи FDB, не будут доставляться в порт CPU, что сэкономит вычислительные ресурсы CPU требуемые для создания skb лишь с целью последующего отбрасывания.

DSA может выполнять фильтрацию адресов хостов для указанных ниже типов адресов.

- Первичные индивидуальный MAC-адреса портов (dev->dev_addr). Эти адреса связаны с приватной базой адресов порта и драйвер уведомляется об их установке через port_fdb_add в направлении порта CPU.
- Вторичные индивидуальные и групповые MAC-адреса портов (добавленные через dev_uc_add() и dev_mc_add()). Эти адреса тоже связаны с приватной базой адресов соответствующего пользовательского порта
- Локальные/постоянные записи FDB моста (BR_FDB_LOCAL). Это MAC-адреса портов моста, для которых пакеты должны обрабатываться локально, без пересылки. Адреса связаны с базой данных этого моста.
- Статические записи FDB моста, установленные для внешних (не DSA) интерфейсов, присутствующих в одном мосту с портами коммутатора DSA. Эти адреса также связаны с базой данных этого моста.
- Динамически полученные записи FDB для внешних интерфейсов, присутствующих в одном мосту с портами коммутатора DSA, лишь при установке драйвером значения true для ds->assisted_learning_on_cpu_port. Адреса связаны с базой данных этого моста.

Для различных операций, описанных ниже, DSA предоставляет структуру dsa_db, которая может иметь один из указанных ниже типов.

DSA_DB_PORT
Устанавливаемая или удаляемая запись FDB (или MDB) для частной базы пользовательского порта db->dp.

DSA_DB_BRIDGE
Запись, относящаяся к одной из баз данных моста db->bridge. Предполагается, что драйвер разделяет базы без VLAN и базы на основе VID в этом мосту.

DSA_DB_LAG
Запись, относящаяся к адресной базе LAG db->lag. Отметим, что DSA_DB_LAG в настоящее время не используется, а в будущем может быть удалена.

Драйверам, использующим аргумент dsa_db в port_fdb_add, port_mdb_add и т. п., следует объявлять ds->fdb_isolation = true.

DSA связывает каждый выгруженный мост и каждую выгруженную группу LAG с идентификатором на основе 1 (struct dsa_bridge :: num, struct dsa_lag :: id) для подсчёта адресов на совместно используемых портах. Драйверы могут использовать (piggyback) схему нумерации DSA (идентификатор считывается через db->bridge.num и db->lag.id) или реализовать свою схему.

Только драйверы, объявляющие поддержку изоляции FDB, уведомляются о записях FDB на порту CPU, относящихся в базам данных DSA_DB_PORT. Для совместимости адреса DSA_DB_BRIDGE сообщаются драйверам, даже не поддерживающим изоляцию FDB, однако в этом случае db->bridge.num и db->lag.id имеют значение 0 (чтобы указать отсутствие изоляции для целей подсчета).

Отметим, что от драйвера коммутатора не требуется реализация адресных баз данных для каждого автономного пользовательского порта. Поскольку записи FDB в приватных базах портов всегда указывают порт CPU, не возникает риска неверных решений о пересылке. В таких случаях все автономные порты могут пользоваться общей базой, но подсчёт ссылок на отфильтрованные хостом адреса(без удаления записей FDB для MAC-адресов порта, которые используются другими портами) возлагается на драйвер, поскольку DSA не знает, что базы портов на деле являются общими. Это можно обеспечить вызовом dsa_fdb_present_in_other_db() и dsa_mdb_present_in_other_db(). Недостатком является то, что списки входной (RX) фильтрации пользовательских портов фактически являются общими, а это означает, что пользовательский порт A может воспринять пакет с MAC DA, который не следует принимать, лишь потому, что этот MAC-адрес включён в RX-фильтр пользовательского порта B. Однако такие пакеты все равно будут отбрасываться программно.

Уровень моста

Выгрузка плоскости пересылки моста необязательна и обслуживается описанными ниже методами. Эти методы могут отсутствовать и возвращается -EOPNOTSUPP или ds->max_num_bridges может быть ненулевым и превышенным и в этом случае присоединение порта моста остаётся возможным, но пересылка пакетов будет выполняться программно, а порты программного моста должны быть настроены так же, как при автономной работе, т. е. все функции моста (изучение адресов и т. п.) должны быть отключены, а все принятые пакеты должны пересылаться лишь в порт CPU.

Порт начинает выгрузку плоскости пересылки после того, как он успешно выполняет метод port_bridge_join, и прекращает это делать после вызова port_bridge_leave. Выгрузка моста означает автономное изучение адресов (записей FDB) в соответствии с состоянием порта программного моста и автономную пересылку (или лавинную рассылку) полученных пакетов без вовлечения порта CPU. Это необязательно даже при выгрузке порта моста. Предполагается, что драйверы протокола тегов вызывают dsa_default_offload_fwd_mark(skb) для пакетов, которые уже пересылались автономно в домене пересылки входного порта коммутатора. DSA с помощью dsa_port_devlink_setup() считает все порты коммутатора, являющиеся частью одного дерева, относящимися к одному домену пересылки моста (способными автономно пересылать пакеты между собой).

Выгрузка выходного (TX) процесса пересылки моста отличается от простой выгрузки плоскости пересылки и относится к способности определённых комбинаций драйверов и протоколов тегов передавать один блок skb, приходящий от функции пересылки устройства моста, одному или нескольким выходным портам (без программного клонирования).

Пакеты, для которых мост запрашивает такое поведение, называют пакетами плоскости данных и они имеют skb->offload_fwd_mark = true в функции xmit драйвера протокола тегов. Для пакетов плоскости данных выполняется поиск в

FDB, аппаратное обучение на порту CPU, а состояние STP для порта не переопределяется. Кроме того, репликация пакетов плоскости данных (групповая и лавинная рассылка) обеспечивается оборудованием, а драйвер моста будет передавать 1 блок skb для каждого пакета, который может реплицироваться.

При включённой выгрузке TX-пересылки драйвер протокола тегов отвечает за внедрение пакетов в плоскость данных оборудования в направлении домена моста (FID), к которому относится порт. Порт может не понимать VLAN и в таком случае идентификатор FID должен совпадать с FID, используемым драйвером для адресной базы без VLAN, связанной с этим мостом. Мост может понимать VLAN и в этом случае гарантируется, что пакет имеет тег VLAN с VLAN ID, где мост обрабатывает этот пакет. Оборудование отвечает за снятие тега VID на выходных портах без тегов и сохранение его на выходных портах с тегами.

port_bridge_join
Функция уровня моста, вызываемая при добавлении в мост данного порта коммутатора. Этой функции следует делать все, что требуется на уровне коммутатора, чтобы разрешить добавление порта в соответствующий логический домен для приёма/передачи трафика других членов моста. Установка для аргумента tx_fwd_offload значения true ведёт к выгрузке и процесса TX-пересылки данного моста.

port_bridge_leave
Функция уровня моста, вызываемая при удалении данного порта коммутатора из моста. Этой функции следует делать все, что требуется на уровне коммутатора, чтобы исключить удаляемый порт из приёма/передачи трафика остальных членов моста.

port_stp_state_set
Функция уровня моста, вызываемая при расчёте статуса STP данного порта коммутатора уровнем моста. Статус порта следует передавать оборудованию коммутатора для пересылки, блокирования и изучения трафика.

port_bridge_flags
Функция уровня моста, вызываемая, когда порт должен настроить свои параметры, например для лавинной рассылки неизвестного трафика или изучения адресов отправителей. Драйвер коммутатора отвечает за начальную организацию автономных портов с отключённым изучением адресов и лавинной рассылкой на выходе для всех типов трафика, после чего ядро DSA уведомляет о любых изменениях флагов портов моста при добавлении или исключении портов моста. В настоящее время DSA не управляет флагами порта моста для порта CPU. Предполагается, что изучение адресов следует включать статически (если это поддерживается оборудованием) на порту CPU, а также следует включать лавинную рассылку в направлении порта CPU, поскольку в ядре DSA нет явного механизма фильтрации адресов.

port_fast_age
Функция уровня моста, вызываемая при необходимости очистки автоматически полученных (обучение) записей FDB на порту. Функция вызывается при смене статуса STP, при котором следует изучать адреса, на статус STP, где этого не следует делать, при выходе из моста или при запрете изучения адресов с помощью port_bridge_flags.

Фильтрация VLAN в мостах

port_vlan_filtering
Функция уровня моста, вызываемая при включении или отключении на мосту фильтрации VLAN. Если на аппаратном уровне не требуется конкретных действий, этот обратный вызов (callback) можно не реализовать. При включении фильтрации VLAN оборудование должно быть запрограммировано на отклонение кадров 802.1Q, в которых VLAN ID выходит за рамки запрограммированных разрешённых отображений и правил VLAN ID. Если на порту коммутатора не запрограммировано PVID, должны отвергаться и кадры без тегов. При отключённой фильтрации коммутатор должен воспринимать любые кадры 802.1Q независимо от их VLAN ID, а также кадры без тегов.

port_vlan_add
Функция уровня моста, вызываемая при настройке VLAN (с тегами или без них) для данного порта коммутатора. Порт CPU становится членом VLAN лишь в том случае, когда порт другого (foreign) моста также входит в VLAN (и пересылка должна выполняться программно), или VLAN помещается в группу VLAN самого устройства моста для целей завершения (bridge vlan add dev br0 vid 100 self). Ссылки на VLAN общих (shared) портов учитываются и VLAN удаляется, если пользователей не остаётся. Драйвер не обязан вручную устанавливать VLAN на порту CPU.

port_vlan_del
Функция уровня моста, вызываемая при a VLAN is removed from the given switch port

port_fdb_add
Функция уровня моста, вызываемая при желании моста установить запись в базе пересылки FDB. На оборудовании коммутатора следует запрограммировать указанный адрес с VLAN ID в базе пересылки, связанной с этим VLAN ID.

port_fdb_del
Функция уровня моста, вызываемая при желании моста удалить запись из базы пересылки FDB. На оборудовании коммутатора следует запрограммировать удаление указанного MAC-адреса из VLAN ID, если он был отображён на данный порт в базе пересылки.

port_fdb_dump
Функция обхода моста, вызываемая ndo_fdb_dump на физических интерфейсах портов DSA. Поскольку DSA не пытается синхронизировать аппаратные записи FDB с программным мостом, этот метод служит средством просмотра записей, видимых на пользовательских портах в аппаратной базе данных. Возвращаемые этой функцией записи имеют флаг self в выводе команды просмотра FDB моста.

port_mdb_add
Функция уровня моста, вызываемая при желании моста установить запись в групповой базе. На оборудовании коммутатора следует запрограммировать указанный адрес с VLAN ID в базе пересылки, связанной с этим VLAN ID.

port_mdb_del
Функция уровня моста, вызываемая при удалить запись из групповой базы. На оборудовании коммутатора следует запрограммировать удаление указанного MAC-адреса из VLAN ID, если он был отображён на данный порт в базе пересылки.

Агрегирование каналов

Агрегирование каналов реализовано в сетевом стеке Linux драйверами bonding и team, моделируемыми как виртуальные стековые сетевые интерфейсы. DSA может выгружать группы агрегирования каналов (link aggregation

group или LAG) в оборудование, поддерживающее эту функцию, и поддерживать мосты между физическими портами и LAG, а также между LAG. Интерфейс bonding/team, содержащий несколько физических портов, представляет собой логический порт, хотя в настоящее время в DSA нет явной концепции логического порта. Поэтому события присоединения и отключения LAG от моста трактуются как присоединение или отключение всех отдельных физических портов данной группы LAG. Атрибуты (фильтрация VLAN, статус STP и т. п.) и объекты (VLAN, записи MDB) порта switchdev, выгруженные в LAG как порт моста, обрабатываются аналогично - DSA выгружает объект или атрибут порта switchdev для всех членов LAG. Статические записи FDB моста для LAG ещё не поддерживаются, поскольку API драйвера DSA не включает концепции идентификатора логического порта.

port_lag_join
Функция, вызываемая при добавлении данного порта в LAG. Драйвер может возвращать -EOPNOTSUPP и в таких случаях DSA будет возвращаться к программной реализации, когда весь трафик из этого порта передаётся в CPU.

port_lag_leave
Функция, вызываемая при удалении данного порта из LAG и возвращающая порт в режим автономного.

port_lag_change
Функция, вызываемая при изменении статуса любого из членов LAG. Функция хэширования при этом должна выполнить повторную балансировку для использования лишь активных физических портов, входящих в LAG.

Драйверы, которым выгодно иметь идентификатор, связанный с каждой выгруженной группой LAG, могут заполнять значение ds->num_lag_ids из метода dsa_switch_ops::setup. LAG ID, связанный с интерфейсом bonding/team в этом случае может быть извлечён драйвером коммутатора DSA с помощью функции dsa_lag_id.

IEC 62439-2 (MRP)

Протокол резервирования среды (Media Redundancy Protocol или MRP) - это протокол управления топологией, оптимизированный для восстановления при отказах в кольцевых сетях, некоторые компоненты которого реализованы как функции драйвера моста. MRP использует управляющие PDU (Test, Topology, LinkDown/Up, Option), передаваемые по групповым MAC-адресам из диапазона 01:15:4e:00:00:0x с EtherType = 0x88e3. В зависимости от роли узла в кольце (MRM - Media Redundancy Manager, MRC - Media Redundancy Client, MRA - Media Redundancy Automanager) для некоторых MRP PDU может требоваться локальное завершение, для других же - пересылка. MRM может выигрывать от выгрузки в оборудование создания и передачи некоторых MRP PDU (Test).

Обычно экземпляр MRP может создаваться на базе любого сетевого интерфейса, однако в случае устройств с выгруженным путём данных, таких как DSA, требуется, чтобы оборудование (даже не поддерживающее MRP) было способно извлекать MRP PDU из модуля коммутации (fabric) до того, как драйвер сможет обработать их в программной реализации. На сегодняшний день в DSA нет драйверов, понимающих MRP, поэтому прослушиваются лишь минимальные объекты switchdev, требуемые для корректной работы программ, которые указаны ниже.

port_mrp_add u port_mrp_del
Уведомляет драйвер о создании или удалении экземпляра MRP с некоторым идентификатором кольца, приоритетом, основным и резервным портом.

port_mrp_add_ring_role u port_mrp_del_ring_role
Функция, вызываемая при смене роли экземпляра MRP между MRM и MRC. Это определяет, какие MRP PDU следует захватывать (trap) программно, а какие - автономно пересылать.

IEC 62439-3 (HSR/PRP)

Протокол параллельного резервирования (Parallel Redundancy Protocol или PRP) обеспечивает резервирование сети путём дублирования и последовательной нумерации пакетов, передаваемых через 2 независимые сети L2 (которые не знают о наличии в конце пакетов тега PRP), и исключения дубликатов у получателя. Протокол бесшовного резервирования высокой доступности (High-availability Seamless Redundancy или HSR) использует похожие концепции, но все узлы, передающие резервный трафик, знают о тегах HSR (HSR использует заголовок с EtherType - 0x892f) и соединены в кольцо. HSR и PRP используют кадры наблюдения для мониторинга работоспособности сети и обнаружения узлов.

В Linux протоколы HSR и PRP реализованы в драйвере hsr, который создаёт виртуальный стековый сетевой интерфейс с двумя портами. Драйвер реализует лишь базовые роли узла с двойным подключением, реализующего HSR (Doubly Attached Node implementing HSR или DANH) или PRP (Doubly Attached Node implementing PRP или DANP), а роли RedBox и QuadBox не поддерживаются (поэтому мостовое соединения сетевого интерфейса hsr с физическим портом коммутатора не даёт ожидаемого результата).

Драйверу, способному выгружать некоторые функции DANP или DANH, следует объявлять соответствующие свойства netdev, как указано в файле Documentation/networking/netdev-features.rst. Кроме того, должны быть реализованы указанные ниже методы.

port_hsr_join
Функция, вызываемая при добавлении данного порта коммутатора в DANP/DANH. Драйвер может возвращать -EOPNOTSUPP и в таких случаях DSA будет возвращаться к программной реализации, когда весь трафик из этого порта передаётся в CPU.

port_hsr_leave
Функция, вызываемая при удалении данного порта коммутатора из DANP/DANH и возвращающая порт в режим автономного.

Продолжение работы

Объединение базы кода SWITCHDEV и DSA

SWITCHDEV должным образом заботится об абстрагировании сетевого стека для оборудования, поддерживающего выгрузку, но не использует строгую модель драйвера устройства коммутации. DSA применяет достаточно строгую модель драйвера устройства и работает с большинством специфических для коммутатора функций. В какой-то момент эти две подсистемы могут быть слиты, принимая лучшее из обеих.

Драйвер коммутатора Ethernet Broadcom RoboSwitch

Семейство коммутаторов Ethernet Broadcom RoboSwitch используется во многих маршрутизаторах xDSL, кабельных модемах и устройствах multimedia. Фактическая реализация поддерживает устройства BCM5325E, BCM5365, BCM539x, BCM53115 и BCM53125, а также BCM63XX.

Детали реализации

Файлы размещены в каталоге drivers/net/dsa/b53/ и реализуют драйвер DSA. Описание подсистемы и её функций приведено в файле Documentation/networking/dsa/dsa.rst.

Коммутатор, по возможности, настраивается на поддержку 4-байтовых тегов коммутатора Broadcom, которые коммутатор помещает в каждый пакет, пересылаемый на интерфейс CPU, а сетевому интерфейсу CPU следует вставлять аналогичный тег во все пакеты, поступающие в порт CPU. Формат тегов описан в файле net/dsa/tag_brcm.c.

Конфигурация устройства зависит от включения или отключения маркировки пакетов тегами.

Имена интерфейсов и примеры сетевой конфигурации соответствуют описанию в параграфе Варианты конфигурации.

Конфигурация с поддержкой тегов

Желательно применять конфигурацию на основе тегов. Она не является специфической для драйвера DSA b53 и будет работать как и с другими драйверами DSA, поддерживающими теги (см . Конфигурация с поддержкой тегов).

Конфигурация без поддержки тегов

Более старые модели (5325, 5365) применяют иной формат тегов, который ещё не поддерживается. Для 539x и 531x5 требуется управляемый режим и особая обработка, которая также пока не поддерживается. В этих случаях теги не поддерживаются и коммутатору нужна другая конфигурация, несколько отличающаяся от описанной в параграфе Конфигурация без поддержки тегов.

Коммутатор b53 помечает тегами порт CPU во всех VLAN, поскольку иначе программирование VLAN без тегов PVID фактически изменит принятый по умолчанию идентификатор PVID порта CPU и сделает его безтеговым, что нежелательно.

В отличие от конфигурации, описанной в параграфе Конфигурация без поддержки тегов, нужно удалить принятую по умолчанию VLAN 1 на ведомом интерфейсе для одиночного порта и шлюза, а в конфигурации моста не требуется дополнительная настройка VLAN.

Отдельный порт

Конфигурацию можно задать лишь через теги VLAN и настройку моста. По умолчанию применяется vid 1.

```
# Установка тегов для трафика на порту CPU
ip link add link eth0 name eth0.1 type vlan id 1
ip link add link eth0 name eth0.2 type vlan id 2
ip link add link eth0 name eth0.3 type vlan id 3

# Ведущий интерфейс должен быть поднят до ведомых портов.
ip link set eth0 up
ip link set eth0.1 up
ip link set eth0.2 up
ip link set eth0.3 up

# Активация ведомых интерфейсов
ip link set wan up
ip link set lan1 up
ip link set lan2 up

# Создание моста
ip link add name br0 type bridge

# Активация фильтров VLAN
ip link set dev br0 type bridge vlan_filtering 1

# Добавление портов в мост
ip link set dev wan master br0
ip link set dev lan1 master br0
ip link set dev lan2 master br0

# Установка тегов для трафика на портах
bridge vlan add dev lan1 vid 2 pvid untagged
bridge vlan del dev lan1 vid 1
bridge vlan add dev lan2 vid 3 pvid untagged
bridge vlan del dev lan2 vid 1

# Настройка VLAN
ip addr add 192.0.2.1/30 dev eth0.1
ip addr add 192.0.2.5/30 dev eth0.2
ip addr add 192.0.2.9/30 dev eth0.3

# Активация моста
ip link set br0 up
```

Мост

```
# Установка тегов для трафика на порту CPU
```

```
ip link add link eth0 name eth0.1 type vlan id 1

# Ведущий интерфейс должен быть поднят до ведомых портов.
ip link set eth0 up
ip link set eth0.1 up

# Активация ведомых интерфейсов
ip link set wan up
ip link set lan1 up
ip link set lan2 up

# Создание моста
ip link add name br0 type bridge

# Активация фильтров VLAN
ip link set dev br0 type bridge vlan_filtering 1

# Добавление портов в мост
ip link set dev wan master br0
ip link set dev lan1 master br0
ip link set dev lan2 master br0
ip link set eth0.1 master br0

# Настройка моста
ip addr add 192.0.2.129/25 dev br0

# Активация моста
ip link set dev br0 up
```

Шлюз

```
# Установка тегов для трафика на порту CPU
ip link add link eth0 name eth0.1 type vlan id 1
ip link add link eth0 name eth0.2 type vlan id 2

# Ведущий интерфейс должен быть поднят до ведомых портов.
ip link set eth0 up
ip link set eth0.1 up
ip link set eth0.2 up

# Активация ведомых интерфейсов
ip link set wan up
ip link set lan1 up
ip link set lan2 up

# Создание моста
ip link add name br0 type bridge

# Активация фильтров VLAN
ip link set dev br0 type bridge vlan_filtering 1

# Добавление портов в мост
ip link set dev wan master br0
ip link set eth0.1 master br0
ip link set dev lan1 master br0
ip link set dev lan2 master br0

# Установка тегов для трафика на портах
bridge vlan add dev wan vid 2 pvid untagged
bridge vlan del dev wan vid 1

# Настройка VLAN
ip addr add 192.0.2.1/30 dev eth0.2
ip addr add 192.0.2.129/25 dev br0

# Активация моста
ip link set br0 up
```

Драйвер коммутатора Ethernet Broadcom Starfighter 2

Аппаратные модули коммутаторов Ethernet Broadcom Starfighter 2 применяются в нескольких типах устройств:

- шлюзы xDSL, такие как BCM63138;
- приставки (Set Top Box) для потокового вещания и multimedia, такие как BCM7445;
- кабельные модемы и домашние шлюзы, такие как BCM7145/BCM3390.

Коммутатор обычно имеет от 5 до 13 портов, обеспечивая ряд встроенных и настраиваемых интерфейсов:

- 1 встроенный интерфейс Gigabit PHY;
- 4 встроенных Gigabit PHY;
- 4 внешних Gigabit PHY с мультиплексором MDIO;
- встроенный MoCA PHY
- несколько внешних интерфейсов MII/RevMII/GMII/RGMII;

Коммутатор также поддерживает функции контроля перегрузки, позволяющие при отказе MoCA не терять пакеты в процессе перевыбора роли MoCA, а также оказывать обратное влияние на сетевой интерфейс CPU хоста при подключении нисходящих интерфейсов на более низкой скорости.

Блок аппаратного коммутатора обычно имеет интерфейс с доступом MMIO и содержит набор регистров (субблоков):

- SWITCH_CORE - общие регистры коммутатора;
- SWITCH_REG - регистр внешних интерфейсов коммутатора;
- SWITCH_MDIO - контроллер внешней шины MDIO (в SWITCH_CORE имеется ещё один для непрямого доступа к PHY);
- SWITCH_INDIR_RW - блок вспомогательных 64-битовых регистров;
- SWITCH_INTRL2_0/1 - контроллеры прерываний L2;
- SWITCH_ACB - блок управления допуском;
- SWITCH_FCB - блок управления преодолением отказов.

Детали реализации

Драйвер находится в файле drivers/net/dsa/bcm_sf2.c и реализован как драйвер DSA. Подсистема и её функции описаны в файле Documentation/networking/dsa/dsa.rst.

Коммутатор SF2 настраивается на вставку 4-байтовых тегов коммутатора Broadcom, которые помещаются в каждый пакет, пересылаемый на интерфейс CPU, а сетевому интерфейсу CPU следует вставлять аналогичный тег во все пакеты, поступающие в порт CPU. Формат тегов описан в файле net/dsa/tag_brcm.c.

В целом драйвер SF2 является обычным драйвером DSA, но имеет некоторые особенности, описанные ниже.

Зондирование дерева устройств

Драйвер устройства платформы DSA зондируется с использованием специальной строки совместимости, представленной в файле net/dsa/dsa.c. Это связано с тем, что подсистема DSA в настоящее время регистрируется как драйвер устройства платформы. DSA предоставляет требуемые указатели device_node, которые становятся доступными функции установки драйвера устройства для организации ресурсов, таких как диапазоны регистров и прерывания. В настоящее время это работает очень хорошо, поскольку ни одной из используемых драйвером функций of_* не требуется привязка структуры device к структуре device_node, но в будущем это может измениться.

Опосредованный доступ к MDIO

Ограничения при разработке коммутаторов Broadcom требуют для внешних коммутаторов Broadcom, подключённых к SF2, использовать ведомую шину DSA MDIO для их настройки. По умолчанию адреса псевдо-PHY коммутатора SF2 и внешнего коммутатора будут отслеживать входящие транзакции MDIO, поскольку они находятся по одному адресу (30), что ведёт к «двойному» программированию. Использование DSA и установка ds->phys_mii_mask перенаправляют операции чтения и записи на адреса псевдо-PHY внешних коммутаторов Broadcom. В новых версиях оборудования SF2 добавлен настраиваемый адрес псевдо-PHY, что позволяет преодолеть это ограничение.

Мультимедиа через интерфейсы CoAxial (MoCA)

Интерфейсы MoCA достаточно специфичны и требуют использования микропрограммного блока, который загружается в процессоры MoCA для обработки пакетов. Оборудование коммутатора включает логику для утверждения и отмены состояний соединения интерфейса MoCA при каждом отключении коаксиального кабеля MoCA или перезагрузке микрокода. Драйвер SF2 полагается на такие события для корректной установки статуса несущей интерфейса MoCA и передачи сведений об этом сетевому стеку.

Интерфейсы MoCA поддерживаются с использованием фиксированных/эмулируемых устройств PHY библиотеки PHY и драйвер коммутатора регистрирует для таких PHY обратный вызов (callback) fixed_link_update, который отражает состояние канала, полученное от обработчика прерываний.

Управление питанием

По мере возможности драйвер SF2 пытается минимизировать потребление энергии, используя комбинацию:

- отключения внутренних буферов и памяти;
- отключения логики обработки пакетов;
- перевода встроенных PHY в режим IDDQ/low-power;
- снижение тактовой частоты ядра коммутатора в зависимости от числа активных портов;
- включение и анонсирование EEE;
- отключение логики обработки данных RGMII при обрыве (down) канала связи.

Wake-on-LAN

Функция пробуждения из сети (Wake-on-LAN или WoL) в настоящее время реализована с использованием логики включения контроллера Ethernet MAC на процессоре хоста. При запросе Wake-on-LAN определяется пересечение запроса пользователя и возможностей WoL на интерфейсе Ethernet хоста и используется результат. При остановке и восстановлении работы на уровне системы в целом отключаются лишь порты, не участвующие в Wake-on-LAN.

Драйвер коммутатора Ethernet LAN9303

LAN9303 - это 3-портовый коммутатор Ethernet 10/100 Мбит/с со встроенными PHY для двух внешних портов Ethernet. Третий порт является интерфейсом RMII/MII для первичного сетевого интерфейса хоста (например, постоянного канала).

Детали драйвера

Драйвер реализован как драйвер DSA, см. Documentation/networking/dsa/dsa.rst. Привязки к дереву устройств описаны в файле Documentation/devicetree/bindings/net/dsa/lan9303.txt.

Управление LAN9303 возможно через MDIO и I2C, поддерживаемые драйвером.

При запуске драйвер настраивает устройство для предоставления двух раздельных сетевых интерфейсов (принятое по умолчанию состояние устройства DSA). Из-за аппаратных ограничений в этом режиме не поддерживается аппаратное изучение MAC-адресов.

Когда оба пользовательских порта присоединены к одному мосту, включается обычное аппаратное изучение MAC-адресов. Это означает, что индивидуальный трафик пересылается на аппаратном уровне. Широковещательные и групповые кадры пересылаются лавинно на аппаратном уровне. В этом режиме поддерживается также протокол STP. Драйвер поддерживает операции FDB/MDB, что означает поддержку протокола IGMP.

Если один из пользовательских портов отключается от моста, порты переходят в исходных режим раздельной работы.

Ограничения драйвера

- Не реализована поддержка фильтрации VLAN.
- Оборудование не поддерживает связанные с VLAN записи FDB.

Драйвер коммутатора Ethernet NXP SJA1105

Обзор

NXP SJA1105 - это семейство из 10 автомобильных коммутаторов с управлением SPI:

- SJA1105E - первое поколение без TTEthernet;
- SJA1105T - первое поколение с TTEthernet;
- SJA1105P - второе поколение без TTEthernet и SGMII;
- SJA1105Q - второе поколение с TTEthernet, но без SGMII;
- SJA1105R - второе поколение без TTEthernet, но с SGMII;
- SJA1105S - второе поколение с TTEthernet и SGMII;
- SJA1110A - третье поколение с TTEthernet, SGMII, встроенными PHY 100base-T1 и 100base-TX PHY;
- SJA1110B - третье поколение с TTEthernet, SGMII, 100base-T1, 100base-TX;
- SJA1110C - третье поколение с TTEthernet, SGMII, 100base-T1, 100base-TX;
- SJA1110D - третье поколение с TTEthernet, SGMII, 100base-T1.

Поскольку коммутаторы являются компонентами автомобиля, их интерфейс настройки ориентирован на принцип «установил и забыл» с минимальным динамическим взаимодействием при работе. Для этого нужно создать статическую конфигурацию программными средствами, упаковать её с CRC и заголовками таблиц и передать через SPI. Статическая конфигурация состоит из нескольких таблиц, часть которых может изменяться (перенастраиваться) в процессе работы. Некоторые таблицы обязательны, другие - нет.

Таблица	*Обязательна*	*Перенастройка*
Планирование	нет	нет
Точки входа планирования	При включённом планировании	нет
Поиск VL	нет	нет
Правила VL	При включённом поиске VL	нет
Пересылка VL	При включённом поиске VL	нет
Поиск L2	нет	нет
Правила L2	да	нет
Поиск VLAN	да	да
Пересылка L2	да	частичная (полная на P/Q/R/S)
Конфигурация MAC	да	частичная (полная на P/Q/R/S)
Параметры планирования	При включённом планировании	нет
Параметры точек входа планирования	При включённом планировании	нет
Параметры пересылки VL	При включённой пересылке VL	нет
Параметры поиска L2	нет	частичная (полная на P/Q/R/S)
Параметры пересылки L2	да	нет
Параметры синхронизации часов	нет	нет
Параметры AVB	нет	нет
Общие параметры	да	частичная
Смена тегов	нет	да
Параметры xMII	да	нет
SGMII	нет	да

Конфигурации доступны только для записи и программы не могут считывать их из коммутатора за исключением редких случаев.

Драйвер создаёт статическую конфигурацию во время зондирования и всё время хранит её в памяти как «тень» состояния оборудования. При необходимости изменить аппаратные настройки обновляется и статическая конфигурация. Если изменённую конфигурацию можно передать в коммутатор через интерфейс динамической реконфигурации, она передаётся, в противном случае выполняется сброс коммутатора и перепрограммирование с обновлённой статической конфигурацией.

Свойства коммутации

Драйвер поддерживает настройку аппаратных правил пересылки L2 для портов моста. Домен пересылки, широковещания и лавинной рассылки может быть ограничен двумя способами - на уровне пересылки L2 (изоляция одного порта моста от других) или принадлежности порта к VLAN (изоляция портов внутри одного моста). Окончательное решение о пересылке, принимаемое оборудованием, является логическим пересечением (AND) этих двух наборов правил.

Оборудование помечает весь трафик тега VLAN по портам (pvid) или декодирует сведения о VLAN из тегов 802.1Q. Расширенная классификация VLAN не поддерживается. После определения тега VLAN кадры проверяются по правилам принадлежности и отбрасываются на входе, если они не соответствуют какой-либо VLAN. Это доступно в случае подключения портов коммутатора к мосту с vlan_filtering = 1.

Обычно оборудование не настраивается в части поддержки VLAN, но изменение TPID, по которому коммутатор ищет теги 802.1Q, позволяет сохранить семантику моста с vlan_filtering = 0 (принимать весь трафик, независимо от тегов), поэтому такой режим тоже поддерживается.

Поддерживается разделение портов коммутатора между несколькими мостами (например, 2 + 2), но всем мостам следует использовать общий уровень осведомлённости о VLAN (значение 0 или 1 для vlan_filtering у всех).

Поддерживается определение топологии и обнаружение петель с помощью протокола STP.

Выгрузка

Планирование с учётом времени

Коммутатор поддерживает вариант усовершенствований для планирования трафика, заданный в IEEE 802.1Q-2018 (ранее 802.1Qbv). Это означает возможность использования коммутатора для обеспечения детерминированной задержки приоритетного трафика, передаваемого вместе (in-band) с событием gate-open в сетевом планировании. Этой возможностью можно управлять через выгрузку tc-taprio (flags 2). Отличие от программной реализации taprio состоит в том, что последняя способна управлять лишь трафиком от CPU, не не пересылаемыми автономно потоками.

Устройство поддерживает 8 классов и отображает входящие кадры на один из них на основе битов VLAN PCP (при отсутствии VLAN используются принятые по умолчанию по портам). Как описано выше, в зависимости от значения vlan_filtering, поле EtherType, распознаваемое коммутатором как VLAN, может иметь значение 0x8100 или настраиваемое значение, используемое драйвером для тегов. Поэтому коммутатор игнорирует VLAN PCP при использовании в автономном режиме или в мосту с vlan_filtering=0, поскольку он не распознаёт EtherType 0x8100. В этих режимах внедрение в конкретную очередь на передачу (TX) возможно лишь для сетевых устройств DSA, которые на выходу включают теги в поле PCP. При vlan_filtering=1 поведение меняется на обратное, выгруженные потоки могут направляться в очереди TX на основе VLAN PCP, но сетевые устройства DSA уже не могут этого делать. Для внедрения кадров в аппаратную очередь TX при включённой поддержке VLAN требуется создать субинтерфейс VLAN на ведущем порту DSA и передавать обычные (0x8100) кадры с тегами VLAN в направлении коммутатора с соответствующей установкой VLAN PCP.

Трафик управления (с DMAC 01-80-C2-xx-xx-xx или 01-19-1B-xx-xx-xx) является исключением и коммутатор всегда обрабатывает его с фиксированным приоритетом без учёта битов VLAN PCP, если они установлены. Для трафика управления в настоящее время используется класс 7 (высший приоритет), который не настраивается в драйвере.

Ниже приведён пример настройки планирования с циклом 500 мсек на выходном порту swp5. Шлюз для управляющего трафика (7) открыт на 100 мсек, для остальных классов - на 400 мсек.

```
#!/bin/bash

set -e -u -o pipefail

NSEC_PER_SEC="1000000000"

gatemask() {
        local tc_list="$1"
        local mask=0

        for tc in ${tc_list}; do
                mask=$((${mask} | (1 << ${tc})))
        done

        printf "%02x" ${mask}
}

if ! systemctl is-active --quiet ptp4l; then
        echo "Please start the ptp4l service"
        exit
fi

now=$(phc_ctl /dev/ptp1 get | gawk '/clock time is/ { print $5; }')
# Фазовое выравнивание базового времени относительно начала следующей секунды.
sec=$(echo "${now}" | gawk -F. '{ print $1; }')
base_time="$(((${sec} + 1) * ${NSEC_PER_SEC}))"
```

```
tc qdisc add dev swp5 parent root handle 100 taprio \
        num_tc 8 \
        map 0 1 2 3 5 6 7 \
        queues 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 \
        base-time ${base_time} \
        sched-entry S $(gatemask 7) 100000 \
        sched-entry S $(gatemask "0 1 2 3 4 5 6") 400000 \
        flags 2
```

Можно применить выгрузку tc-taprio на нескольких выходных порта. Однако имеется аппаратное ограничение, связанное с невозможностью одновременных событий шлюзов на двух портах. Драйвер проверяет согласованность планирования с этим ограничением и при необходимости выдаёт ошибку. Для предотвращения ошибок нужен анализ планирования, выходящий за рамки этого документа.

Маршрутизация (redirect, trap, drop)

Коммутатор способен выгружать основанное на потоках перенаправление в набор портов, заданных пользователем. В коммутаторе это реализовано путём использования виртуальных каналов (Virtual Link или VL) - концепции TTEthernet.

Коммутатор поддерживает для VL 2 типа ключей:

- виртуальные каналы с поддержкой VLAN соответствуют MAC-адресу получателя, VLAN ID и VLAN PCP;
- виртуальные каналы без поддержки VLAN соответствуют лишь MAC-адресу получателя.

Состояние осведомлённости моста о VLAN (vlan_filtering) не может быть изменено, пока установлены правила для VL. В одном правиле может применяться несколько действий. Если нужна лишь маршрутизация, драйвер создаёт «некритичный» виртуальный канал. Если список действий включает также tc-gate (см. ниже), виртуальный канал становится «критичным по времени» (извлекает буферы кадров из зарезервированного раздела памяти и т. п.).

Поддерживается 3 действия по маршрутизации: ловушка (trap), отбрасывание (drop) и перенаправление (redirect).

Пример 2. Передача кадров, полученных на swp2 с DA 42:be:24:9b:76:20, в CPU и swp3. Этот тип ключа (только DA) относится к отключённой поддержке VLAN на порту

```
tc qdisc add dev swp2 clsact
tc filter add dev swp2 ingress flower skip_sw dst_mac 42:be:24:9b:76:20 \
        action mirred egress redirect dev swp3 \
        action trap
```

Пример 2. Отбрасывание кадров, полученных на swp2 с DA of 42:be:24:9b:76:20, VID 100 и PCP 0

```
tc filter add dev swp2 ingress protocol 802.1Q flower skip_sw \
        dst_mac 42:be:24:9b:76:20 vlan_id 100 vlan_prio 0 action drop
```

Входные правила с учётом времени

Аппаратные возможности TTEthernet в коммутаторе могут быть ограничены для работы в соответствии с пунктом «Фильтрация и правила по потокам» (Per-Stream Filtering and Policing или PSFP) стандарта IEEE 802.1Q-2018 (ранее 802.1Qci). Это означает возможность жёсткого контроля допуска по времени для множества (до 1024) потоков, задаваемых MAC-адресом получателя, VLAN ID и VLAN PCP. Пакеты, полученные за пределами ожидаемого окна приёма, отбрасываются. Этой возможностью можно управлять путём выгрузки действия tc-gate. Поскольку действия по маршрутизации присущи виртуальным каналам в TTEthernet (явная маршрутизация критичного ко времени трафика без отдачи на откуп FDB, лавинной рассылке и т. п.), действие tc-gate не может возникать само по себе, если запросить у sja1105 выгрузку его. За этим действием должно следовать 1 или несколько действий redirect или trap.

Создадим, например, расписание tc-taprio, согласованное по фазе с расписанием tc-gate (часы должны быть синхронизированы стеком приложения 1588, но это выходит за рамки документа). Ни один из пакетов, доставленных отправителем, не будет отброшен. Отметим, что окно приёма больше окна передачи (в этом примере намного больше) для компенсации задержки распространения в канале (которую может определить стек приложения 1588).

Получатель (sja1105)

```
tc qdisc add dev swp2 clsact
now=$(phc_ctl /dev/ptp1 get | awk '/clock time is/ {print $5}') && \
        sec=$(echo $now | awk -F. '{print $1}') && \
        base_time="$(((sec + 2) * 1000000000))" && \
        echo "base time ${base_time}"
tc filter add dev swp2 ingress flower skip_sw \
        dst_mac 42:be:24:9b:76:20 \
        action gate base-time ${base_time} \
        sched-entry OPEN  60000 -1 -1 \
        sched-entry CLOSE 40000 -1 -1 \
        action trap
```

Отправитель

```
now=$(phc_ctl /dev/ptp0 get | awk '/clock time is/ {print $5}') && \
        sec=$(echo $now | awk -F. '{print $1}') && \
        base_time="$(((sec + 2) * 1000000000))" && \
        echo "base time ${base_time}"
tc qdisc add dev eno0 parent root taprio \
        num_tc 8 \
        map 0 1 2 3 4 5 6 7 \
        queues 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 \
        base-time ${base_time} \
        sched-entry S 01  50000 \
        sched-entry S 00  50000 \
        flags 2
```

Для планирования работы входных шлюзов применяется тот же механизм, что и для выгрузки tc-taprio, поэтому сохраняются ограничения, связанные с невозможностью одновременного (в течение одного интервала в 200 нсек) срабатывания двух шлюзов (tc-gate или tc-taprio).

Для удобства можно использовать создаваемые по времени виртуальные каналы через несколько портов с помощью блоков потоков. В этом случае ограничения на одновременный запуск не применяются, поскольку в системе имеется единое планирование для общего виртуального канала.

```
tc qdisc add dev swp2 ingress_block 1 clsact
tc qdisc add dev swp3 ingress_block 1 clsact
tc filter add block 1 flower skip_sw dst_mac 42:be:24:9b:76:20 \
        action gate index 2 \
        base-time 0 \
        sched-entry OPEN 50000000 -1 -1 \
        sched-entry CLOSE 50000000 -1 -1 \
        action trap
```

Доступна также аппаратная статистика для каждого потока (pkts учитывает число отброшенных кадров, указывающее сумму кадров, отброшенных из-за временных ограничений, отсутствия портов назначения и превышения MTU). Счётчики байтов недоступны.

Ограничения

Коммутаторы семейства SJA1105 всегда обрабатывают VLAN. При отключённой поддержке VLAN на порту кадры содержат внутренние теги VLAN в зависимости от того, является порт автономным или относится к мосту с VLAN.

Ключи виртуальных каналов фиксированы {MAC DA, VLAN ID, VLAN PCP}, но драйвер запрашивает VLAN ID и VLAN PCP, если порт относится к мосту с поддержкой VLAN. В иных случаях драйвер автоматически указывает VLAN ID и VLAN PCP в зависимости от того, является порт автономным или относится к мосту без VLAN, и воспринимает лишь ключи tc-flower без поддержки VLAN (MAC DA).

Имеющиеся ключи tc-flower, выгруженные с использованием виртуальных каналов, перестают работать, при возникновении одного из указанных ниже обстоятельств:

- автономный ранее порт присоединяется к мосту (с VLAN или без таковых);
- порт относится к мосту, в котором меняется статус поддержки VLAN;
- порт, входивший в мост, становится автономным;
- порт был автономным, но другой порт присоединился к мосту с VLAN и это поменяло статус поддержки VLAN в мосту.

Драйвер не может отменить все эти операции, а также не может обновлять или удалять имеющиеся фильтры tc-flower. Поэтому для корректной работы фильтры tc-flower следует устанавливать лишь после настройки пересылки на порту и удалять из пользовательского пространства перед внесением в них каких-либо изменений.

Привязки дерева устройств и конструкция платы

Этот раздел основан на файле Documentation/devicetree/bindings/net/dsa/nxp,sja1105.yaml и демонстрирует некоторые предостережения при работе с коммутаторами.

Роль RMII PHY и сигнализация по отдельному каналу

По спецификации RMII тактовые сигналы 50 МГц подаются от MAC или внешнего генератора (но не от PHY). Однако спецификация предоставляет достаточно свободы и устройства часто выходят за её пределы. Некоторые PHY идут наперекор спецификации и предоставляют вывод, на которых они подают сигнал 50 МГц, пытаясь быть полезными. Коммутатор SJA1105 настраивается только двоичным путём и в роли RMII MAC он тоже пытается управлять тактовым сигналом. Чтобы этого не происходило, коммутатор нужно перевести в режим RMII PHY, однако это ведёт к некоторым нежелательным последствиям. По спецификации RMII устройство PHY может передавать дополнительные внеполосные сигналы через RXD[1:0]. Практически это несколько дополнительных кодовых слов ((/J/ и /K/), передаваемых до преамбулы каждого кадра. В MAC нет такого механизма внеполосной сигнализации, определяемого спецификацией RMII. Поэтому при переводе порта SJA1105 в роль PHY для исключения двух источников тактовых сигналов неизбежно возникает соединение RMII PHY-to-PHY. Коммутатор SJA1105 полностью эмулирует интерфейс PHY и генерирует символы /J/ и /K/ перед преамбулой кадра, которые предполагаются непонятными для реального PHY. Поэтому PHY просто кодирует дополнительные символы, полученные от SJA1105 в роли PHY, в линию 100Base-Tx. На другом конце линии некоторые партнёры по каналу могут отбрасывать эти дополнительные символы, но другие могут «подавиться» ими и отбросить весь кадр Ethernet. Это выглядит как потеря пакета на некоторых каналах. Поэтому в режиме RMII коммутатор SJA1105 должен быть способен служить тактовым генератором при соединении с PHY.

RGMII стационарных каналов и внутренние задержки

Второе поколение устройств имеет настраиваемые линии задержки (часть MAC), которые можно использовать для организации корректного бюджета времени RGMII. При включении питания линии могут смещать часы Rx и Tx по фазе на величину от 73,8 до 101,7 градуса. Сложность заключается в том, что линии задержки нужно привязывать к тактовому сигналу со стабильной частотой. Это означает, что между тактовыми сигналами старой и новой частоты должно быть не менее 2 мксек тишины. В противном случае синхронизация теряется и линии задержки нужно сбрасывать (выключать и включать снова). В RGMII тактовая частота зависит от скорости канала (125 МГц для 1000 Мбит/с, 25 МГц для 100 Мбит/с и 2,5 МГц для 10 Мбит/с), которая может меняться в процессе автоматического согласования. В ситуации, когда порт коммутатора подключён к партнёру по стационарному каналу RGMII и состояние этого канала не контролируется Linux (например, другая микросхема SoC), линии задержки остаются разблокированными (и неактивными) без ручного вмешательства (ifdown/ifup на порту коммутатора). В результате этого в режиме RGMII внутренние задержки в коммутаторе будут надёжными лишь в случаях, когда скорость канала к

партнёру не меняется или меняется согласованно с портом коммутатора (практически, обе стороны канала находятся под управлением одной системы Linux). В части изменения скорости стационарного канала следует помнить, что есть контроллеры Ethernet, которые при перезапуске устанавливают режим 100 Мбит/с и требуют смены тактовой частоты для перехода не гигабитную скорость.

Шина MDIO и управление PHY

SJA1105 не имеет шины MDIO и не выполняет автонастройки по основному каналу, поэтому от коммутационного устройства не поступает уведомлений о статусе канала. Плате нужно «перехватывать» (hook) PHY коммутатора, подключённые к какой-либо шине MDIO, доступной для Linux внутри системы (например, к шине MDIO ведущего DSA). Управление состоянием канала в этом случае выполняется драйвером «вручную» путём синхронизации (с помощью команд SPI) скорости канала MAC с установками, согласованными PHY.

SJA1110 поддерживает ведомую точку доступа MDIO, через которую с хоста можно обращаться к внутренним 100base-T1 PHY. Однако эта точка не используется драйвером, а доступ к внутренним PHY 100base-T1 и 100base-TX осуществляется через команды SPI, моделируемые в Linux как виртуальные шины MDIO.

Микроконтроллер, подключённый к порту 0 в SJA1110, имеет контроллер MDIO, работающий в режиме первичного (master), однако драйвер его не поддерживает, поскольку микроконтроллер отключается при работе драйвера Linux. Дискретным PHY, подключённым к портам коммутатора, следует иметь интерфейс MDIO, подключённый к контроллеру MDIO хост-системы, а не коммутатора, как в случае SJA1105.

Матрица совместимости портов

Матрица совместимости портов SJA1105 показана в таблице.

Порт	***SJA1105E/T***	***SJA1105P/Q***	***SJA1105R/S***
0	xMII	xMII	xMII
1	xMII	xMII	xMII
2	xMII	xMII	xMII
3	xMII	xMII	xMII
4	xMII	xMII	SGMII

Матрица совместимости портов SJA1110 показана в следующей таблице.

Порт	***SJA1110A***	***SJA1110B***	***SJA1110C***	***SJA1110D***
0	RevMII (uC)	RevMII (uC)	RevMII (uC)	RevMII (uC)
1	100base-TX или SGMII	100base-TX	100base-TX	SGMII
2	xMII или SGMII	xMII	xMII	xMII или SGMII
3	XMII, SGMII или 2500base-X	XMII, SGMII или 2500base-X	xMII	SGMII или 2500base-X
4	SGMII или 2500base-X	SGMII или 2500base-X	SGMII или 2500base-X	SGMII или 2500base-X
5	100base-T1	100base-T1	100base-T1	100base-T1
6	100base-T1	100base-T1	100base-T1	100base-T1
7	100base-T1	100base-T1	100base-T1	100base-T1
8	100base-T1	100base-T1	-	-
9	100base-T1	100base-T1	-	-
10	100base-T1	-	-	-

Настройка коммутатора из пользовательского пространства

Настройка конфигурации коммутатора DSA в настоящее время не интегрирована с основными пакетами настройки конфигурации сети и должна выполняться вручную.

Варианты конфигурации

Для настройки коммутатора DSA требуется выполнить несколько команд и ниже в качестве примеров рассматриваются три базовых случая.

Одиночный порт
: Каждый порт коммутатора выступает как отдельный настраиваемый порт Ethernet.

Мост
: Каждый порт коммутатора является частью одного настраиваемого моста Ethernet.

Шлюз
: Каждый порт коммутатора, за исключением одного восходящего порта, является частью одного настраиваемого моста Ethernet. Восходящий порт является отдельным настраиваемым портом Ethernet.

Все настройки выполняются с помощью команд iproute2[1].

Через DSA каждый порт коммутатора обслуживается как обычный интерфейс Linux Ethernet. Порт CPU в коммутаторе соединён с микросхемой Ethernet MAC. Соответствующий интерфейс Linux Ethernet называется ведущим (master), остальные - ведомыми. Чтобы ведомые интерфейсы могли принимать и передавать трафик, должен быть включён ведущий интерфейс. В ядрах до версии v5.12 состоянием ведущего интерфейса явно управлял пользователь, но в ядрах, начиная с v5.12, поведение изменилось:

- при активации ведомого интерфейса DSA ведущий интерфейс активируется автоматически;
- при отключении (down) ведущего интерфейса все ведомые интерфейсы DSA отключаются автоматически.

В этом документе используются указанные ниже интерфейсы Ethernet.

eth0
: Ведущий интерфейс.

eth1
: Второй ведущий интерфейс.

[1] См. https://www.kernel.org/pub/linux/utils/net/iproute2/

lan1
Ведомый интерфейс.

lan2
Второй ведомый интерфейс.

lan3
Третий ведомый интерфейс.

wan
Ведомый интерфейс, выделенный для восходящего трафика.

Другие интерфейсы Ethernet могут настраиваться аналогично. Адреса IP и сети указаны ниже.

Отдельный порт
- lan1: 192.0.2.1/30 (192.0.2.0 - 192.0.2.3)
- lan2: 192.0.2.5/30 (192.0.2.4 - 192.0.2.7)
- lan3: 192.0.2.9/30 (192.0.2.8 - 192.0.2.11)

Мост
- br0: 192.0.2.129/25 (192.0.2.128 - 192.0.2.255)

Шлюз
- br0: 192.0.2.129/25 (192.0.2.128 - 192.0.2.255)
- wan: 192.0.2.1/30 (192.0.2.0 - 192.0.2.3)

Конфигурация с поддержкой тегов

Конфигурация на основе тегов желательна и поддерживается большинством коммутаторов DSA. Эти коммутаторы способны маркировать входящий и исходящий трафик без использования конфигурации на основе VLAN.

Отдельный порт

```
# Настройка каждого интерфейса.
ip addr add 192.0.2.1/30 dev lan1
ip addr add 192.0.2.5/30 dev lan2
ip addr add 192.0.2.9/30 dev lan3

# Для ядер до v5.12 ведущий интерфейс нужно активировать вручную
# до активации любого из ведомых портов.
ip link set eth0 up

# Активация ведомых интерфейсов.
ip link set lan1 up
ip link set lan2 up
ip link set lan3 up
```

Мост

```
# Для ядер до v5.12 ведущий интерфейс нужно активировать вручную
# до активации любого из ведомых портов.
ip link set eth0 up

# Активация ведомых интерфейсов.
ip link set lan1 up
ip link set lan2 up
ip link set lan3 up

# Создание моста.
ip link add name br0 type bridge

# Добавление портов моста.
ip link set dev lan1 master br0
ip link set dev lan2 master br0
ip link set dev lan3 master br0

# Настройка моста.
ip addr add 192.0.2.129/25 dev br0

# Активация моста
ip link set dev br0 up
```

Шлюз

```
# Для ядер до v5.12 ведущий интерфейс нужно активировать вручную
# до активации любого из ведомых портов.
ip link set eth0 up

# Активация ведомых интерфейсов.
ip link set wan up
ip link set lan1 up
ip link set lan2 up

# Настройка восходящего порта.
ip addr add 192.0.2.1/30 dev wan

# Создание моста.
ip link add name br0 type bridge

# Добавление портов моста.
ip link set dev lan1 master br0
ip link set dev lan2 master br0
```

```
# Настройка моста.
ip addr add 192.0.2.129/25 dev br0

# Активация моста.
ip link set dev br0 up
```

Конфигурация без поддержки тегов

Незначительная часть коммутаторов не способна использовать протокол тегов (DSA_TAG_PROTO_NONE). Их можно настраивать на основе VLAN.

Отдельный порт

Конфигурацию можно задать лишь через теги VLAN и организацию (setup) моста.

```
# Теги трафика на порту CPU.
ip link add link eth0 name eth0.1 type vlan id 1
ip link add link eth0 name eth0.2 type vlan id 2
ip link add link eth0 name eth0.3 type vlan id 3

# Для ядер до v5.12 ведущий интерфейс нужно активировать вручную
# до активации любого из ведомых портов.
ip link set eth0 up
ip link set eth0.1 up
ip link set eth0.2 up
ip link set eth0.3 up

# Активация ведомых интерфейсов.
ip link set lan1 up
ip link set lan2 up
ip link set lan3 up

# Создание моста.
ip link add name br0 type bridge

# Активация фильтров VLAN.
ip link set dev br0 type bridge vlan_filtering 1

# Добавление портов моста.
ip link set dev lan1 master br0
ip link set dev lan2 master br0
ip link set dev lan3 master br0

# Теги трафика на портах.
bridge vlan add dev lan1 vid 1 pvid untagged
bridge vlan add dev lan2 vid 2 pvid untagged
bridge vlan add dev lan3 vid 3 pvid untagged

# Настройка VLAN.
ip addr add 192.0.2.1/30 dev eth0.1
ip addr add 192.0.2.5/30 dev eth0.2
ip addr add 192.0.2.9/30 dev eth0.3

# Активация моста.
ip link set br0 up
```

Мост

```
# Теги трафика на порту CPU.
ip link add link eth0 name eth0.1 type vlan id 1

# Для ядер до v5.12 ведущий интерфейс нужно активировать вручную
# до активации любого из ведомых портов.
ip link set eth0 up
ip link set eth0.1 up

# Активация ведомых интерфейсов.
ip link set lan1 up
ip link set lan2 up
ip link set lan3 up

# Создание моста.
ip link add name br0 type bridge

# Активация фильтров VLAN.
ip link set dev br0 type bridge vlan_filtering 1

# Добавление портов моста.
ip link set dev lan1 master br0
ip link set dev lan2 master br0
ip link set dev lan3 master br0
ip link set eth0.1 master br0

# Теги трафика на портах.
bridge vlan add dev lan1 vid 1 pvid untagged
bridge vlan add dev lan2 vid 1 pvid untagged
bridge vlan add dev lan3 vid 1 pvid untagged
```

```
# Настройка моста.
ip addr add 192.0.2.129/25 dev br0

# Активация моста
ip link set dev br0 up
```

Шлюз

```
# Теги трафика на порту CPU.
ip link add link eth0 name eth0.1 type vlan id 1
ip link add link eth0 name eth0.2 type vlan id 2

# Для ядер до v5.12 ведущий интерфейс нужно активировать вручную
# до активации любого из ведомых портов.
ip link set eth0 up
ip link set eth0.1 up
ip link set eth0.2 up

# Активация ведомых интерфейсов.
ip link set wan up
ip link set lan1 up
ip link set lan2 up

# Создание моста.
ip link add name br0 type bridge

# Активация фильтров VLAN.
ip link set dev br0 type bridge vlan_filtering 1

# Добавление портов моста.
ip link set dev wan master br0
ip link set eth0.1 master br0
ip link set dev lan1 master br0
ip link set dev lan2 master br0

# Теги трафика на портах.
bridge vlan add dev lan1 vid 1 pvid untagged
bridge vlan add dev lan2 vid 1 pvid untagged
bridge vlan add dev wan vid 2 pvid untagged

# Настройка VLAN.
ip addr add 192.0.2.1/30 dev eth0.2
ip addr add 192.0.2.129/25 dev br0

# Активация моста.
ip link set br0 up
```

Управление базой пересылки (FDB)

В имеющихся коммутаторах DSA нет поддержки синхронизации программной базы FDB моста с аппаратными таблицами, поэтому управление этими таблицами выполняется раздельно - команда bridge fdb show запрашивает обе таблицы, а команды bridge fdb add и bridge fdb del применяются к одной или обеим таблицам в зависимости от флагой self и master.

В ядрах до v4.14 управление записями FDB моста поддерживалось DSA из пользовательского пространства лишь с помощью операций обхода моста (обновляют лишь аппаратную таблицу, но не FDB) с флагом self (необязателен).

```
bridge fdb add dev swp0 00:01:02:03:04:05 self static
# или
bridge fdb add dev swp0 00:01:02:03:04:05 static
```

Из-за ошибки реализация обхода моста в DSA не различала статические и локальные записи FDB (статические предназначены для пересылки, а локальные - для завершения в коммутаторе, т. е. передачи в порт хоста). Записи FDB с флагом self (явным или неявным) трактовались DSA как статические, даже будучи локальными.

```
# Приведённая ниже команда
bridge fdb add dev swp0 00:01:02:03:04:05 static
# ведёт себя для DSA как команда
bridge fdb add dev swp0 00:01:02:03:04:05 local
# или сокращённо, поскольку флаг local предполагается, если не задан static.
bridge fdb add dev swp0 00:01:02:03:04:05
```

Последняя команда является некорректным способом добавления статической записи FDB в коммутатор DSA с помощью операции обхода моста и работает лишь благодаря ошибке. Другие драйверы будут считать добавленную этой командой запись FDB локальной и не будут пересылать по ней в отличие от DSA.

В ядрах от v4.14 до v5.14 DSA поддерживает параллельно два режима добавления записей FDB в мост - рассмотренный выше обход моста и новый режим с использованием флага master, задающего установку записей FDB и в программный мост.

```
bridge fdb add dev swp0 00:01:02:03:04:05 master static
```

Начиная с ядра v5.14 в DSA была достигнута более тесная интеграция с программной FDB моста, а поддержка обхода моста (флаг self) была исключена. Это привело к указанным ниже изменениям.

```
# Это единственный способ добавления записи FDB, совместимый
# с ядрами, начиная с v4.14.
bridge fdb add dev swp0 00:01:02:03:04:05 master static
# Эта команда больше не является ошибочной и запись корректно
# считается локальной.
bridge fdb add dev swp0 00:01:02:03:04:05
```

```
# Эта команда больше не включает статическую запись FDB в оборудование.
bridge fdb add dev swp0 00:01:02:03:04:05 static
```

При разработке сценариев рекомендуется использовать флаги master и static для работы с записями FDB моста на интерфейсах коммутатора DSA.

Близость пользовательских портов к портам CPU

Обычно коммутатор DSA подключается к хосту через один интерфейс Ethernet, но при использовании дискретных микросхем коммутации оборудование может допускать использование нескольких портов для повышения пропускной способности при завершении трафика. DSA может использовать несколько портов CPU разными способами. Во-первых, можно статически назначить завершение трафика, связанного с неким пользовательским портом, определённому порту CPU. Таким способом пользовательское пространство может реализовать настраиваемые правила статического распределения нагрузки между пользовательскими портами, распределяя близость (affinity) в соответствии с доступными портами CPU. Во-вторых, можно распределять нагрузку между портами CPU на уровне пакетов, не задавая статического сопоставления пользовательских портов с портами CPU. Это можно реализовать, размещая ведущие DSA на интерфейсе LAG (bonding или team). DSA отслеживает это и создаёт зеркало такой программной группы LAG на портах CPU, обращённых к ведущим DSA, связанным с ведомым устройством LAG.

Для использования нескольких портов CPU в описании микрокода (дерево устройств) коммутатора требуется отметить все каналы между портами CPU и их DSA master с использованием ссылок (phandle) Ethernet. При запуске используется лишь 1 порт CPU и DSA (первый по порядку из описания микрокода, имеющий свойство Ethernet). Настройку системы для использования коммутатором других ведущих выполняет пользователь.

DSA использует механизм rtnl_link_ops (с типом dsa) для возможности смены DSA master пользовательского порта. Атрибут netlink IFLA_DSA_MASTER u32 содержит ifindex ведущего устройства, обслуживающего каждое ведомое устройство. DSA master должен быть действительным кандидатом на основе сведений об узле микрокода или интерфейсом LAG, содержащим только ведомые, являющиеся действительными кандидатами.

Ниже приведены манипуляции, доступные с помощью iproute2.

```
# Просмотр используемого DSA
ip -d link show dev swp0
    (...)
    dsa master eth0

# Статическое распределение порта CPU.
ip link set swp0 type dsa master eth1
ip link set swp1 type dsa master eth0
ip link set swp2 type dsa master eth1
ip link set swp3 type dsa master eth0

# Порты CPU в LAG, использующие явное назначение DSA master.
ip link add bond0 type bond mode balance-xor && ip link set bond0 up
ip link set eth1 down && ip link set eth1 master bond0
ip link set swp0 type dsa master bond0
ip link set swp1 type dsa master bond0
ip link set swp2 type dsa master bond0
ip link set swp3 type dsa master bond0
ip link set eth0 down && ip link set eth0 master bond0
ip -d link show dev swp0
    (...)
    dsa master bond0

# Порты CPU в LAG, полагающиеся на неявный перенос DSA master.
ip link add bond0 type bond mode balance-xor && ip link set bond0 up
ip link set eth0 down && ip link set eth0 master bond0
ip link set eth1 down && ip link set eth1 master bond0
ip -d link show dev swp0
    (...)
    dsa master bond0
```

Отметим, что для портов CPU в LAG использование атрибута netlink IFLA_DSA_MASTER не является обязательным, а DSA реагирует на изменение атрибута IFLA_MASTER у текущего ведущего устройства (eth0) и переносит все пользовательские порты на новый интерфейс верхнего уровня eth0 (bond0). Аналогично, при удалении bond0 с помощью RTM_DELLINK DSA переносит пользовательские порты, связанные с этим интерфейсом, на первое физическое устройство DSA master, выбираемое по описанию микрокода (фактически происходит возврат к начальной конфигурации).

В системе с числом физических портов CPU больше 2 возможно сочетание статической привязки пользовательских портов к портам CPU с LAG между DSA master. Невозможно статически назначить пользовательский порт в направлении DSA master, имеющего какие-либо интерфейсы верхнего уровня (включая устройства LAG, в этом случае ведущим всегда будет LAG).

Разрешается изменение близости пользовательского порта к DSA master (и порту CPU) для обеспечения динамического перераспределения в зависимости от трафика.

Физическим DSA master разрешается в любой момент присоединяться и выходить из LAG, служащего DSA master, однако DSA может отвергать интерфейс LAG в качестве действительного кандидата на роль DSA, если у него нет хотя бы одного физического DSA master в качестве ведомого устройства.

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru