Где сохранить пакет?

Release 1.0

Jun 09, 2021

Оглавление

Марат Сибгатулин aka eucariot.

"Где сохранить пакет" - книга о том, как устроена аппаратура для передачи данных в современных сетях - о чипах и о памяти. Доставка пакета от отправителя к получателю - это его последовательная передача от одного маршрутизатора к другому. Каждый узел должен самостоятельно принять решение, как поступить с этим пакетом.

Он должен также принять во внимание загрузку линии и не отправлять пакет, если он будет отброшен. А значит он должен его хранить, пока принимает решение и пока не освободится линия.

В этой книге мы поговорим про архитектуру сетевых устройств и чипов, из которых они собраны, о том, что происходит с пакетом с момента, как он попадает на входной интерфейс устройства и до момента, когда он его покидает.

Вторая половина (начиная с 5-й главы) будет посвящена архитектурам памяти и всевозможным ухищрениям, призванным выжать из сети максимальную полосу при минимальных задержках и потерях.

Книга является частью проекта linkmeup, где группа отчаянных энтузиастов пишет статьи и подкасты.

Есть два цикла статей:

- СДСМ - Сети Для Самых Маленьких
- АДСМ - Автоматизация Для Самых Маленьких

И несколько подкастов:

- telecom - подкасты про всё те же сети. Раз в месяц мы болтаем с каким-нибудь крутым экспертом на интересную нам тему. А 25-го числа выкладываем в самый всеобщий доступ.

В телеком выпуске у нас уже побывали: Mellanox Technologies, Microsoft, Arista, Cisco, Huawei, Элтекс, D-Link, Ericcson и ещё много кто, да не обидятся неупомянутые, ибо несть им числа.

- sysadmins - ветвление по админо-девопсо-эсарйешно-эникейной линии.
- В рамках эксперимента у нас работает **бар "256-й TTL"**:

Поссиелки - за виртуальной пинтой допрашиваем обладателей активных экспертных сертификатов (не только циско).

По'уехавшие - истории эмигрантов и экспатов о том, где жить хорошо - а там, наверно, хорошо - ведь они там остались.

Шоты - короткие интервью с разными людьми на узкие темы, например, перед или по следам доклада.

По каким дням и в какие часы он работает - никто не знает. Заходите - вдруг вам повезёт и будет открыто.

- В 2018-м мы провели уникальную для нашей сферы сессию онлайн-собеседований в крупные компании.
- И тогда же, допросив захваченного в плен HR, выяснили дела в тылу противника.

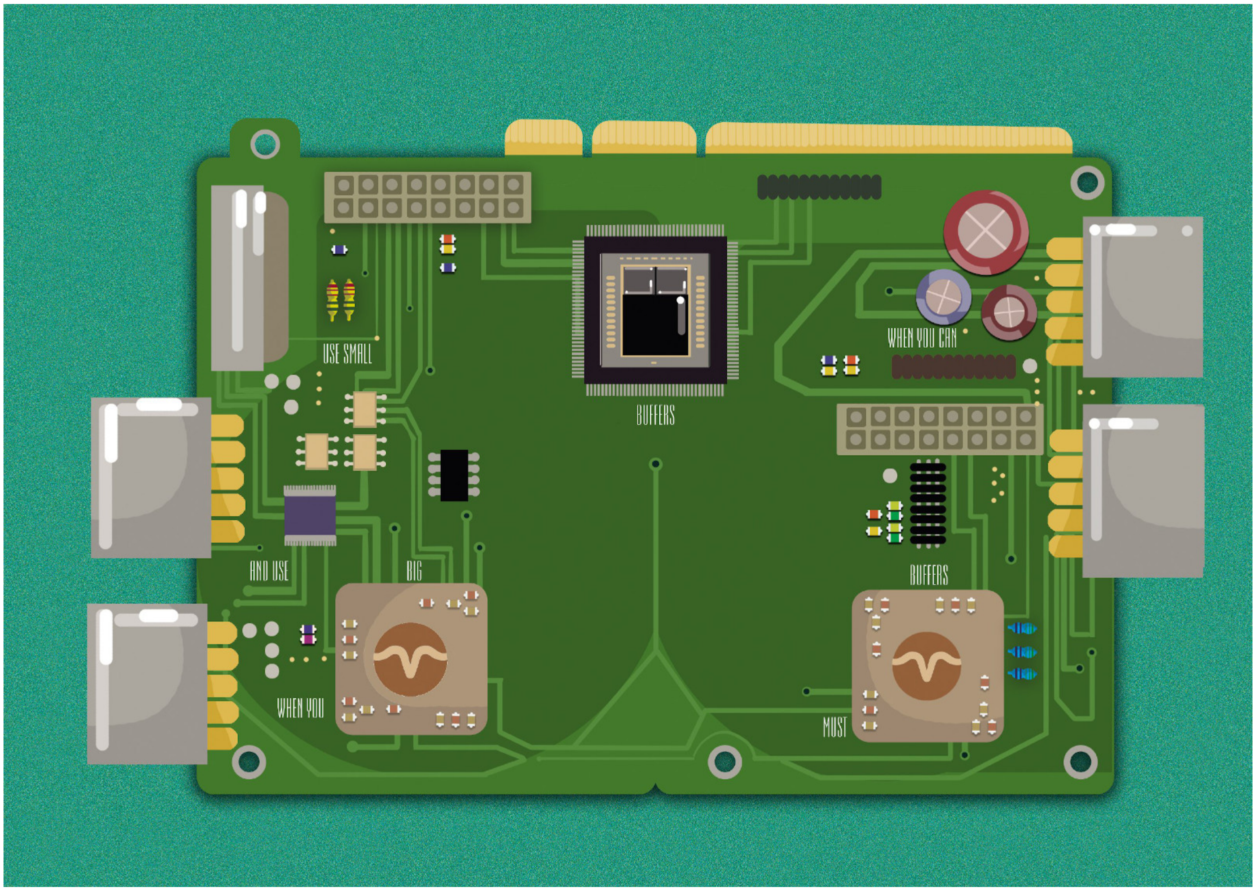
По любым вопросам можно писать на info@linkmeup.ru

Вы можете поддержать костёр энтузиазма подкинув угля через Патреон или другим удобным способом.

1

0. Введение

Самый мощный сетевой чип сегодня умеет обрабатывать 25 600 000 000 000 бит за одну секунду. 30 миллиардов транзисторов должны провернуть уйму работы, доставляя биты со входного интерфейса на выходной. Новые типы интерфейсов, увеличивающиеся скорости, утончающиеся техпроцессы, новые функции - бросают ежедневные вызовы инженерам, разработчикам, производителям. У меня давно уже был праздный интерес к аппаратной начинке сетевых коробок, а в этот раз я решил поковыряться отвёрткой ещё глубже и, сняв крышечку с сетевых микросхем, взглянуть на Pipeline'ы да SerDes'ы. Как вы уже, вероятно, поняли, в этой статье будет запредельно много сленга, англицизмов и слов на английском, потому что переводить их на русский неблагодарно, а порой и кощунственно.



Но знаете, какая мысль действительно не даёт мне покоя всю дорогу? Как вся эта сложнейшая конструкция: миллиарды транзисторов, размером около десятка нанометров, напечатанных на пластине размером с большую печеньку, которая полдюжиной тысяч ног припаяна к многослойной плате, напичканной мириадами других микрочипов, сопротивлений и конденсаторов и являющейся частью многоюнитовой модульной коробки, управляемой операционной системой, состоящей из сотен прошивок для разных чипов, инструкций по работе с ними, реализаций стандартизированных и проприетарных протоколов и механизмов поддержания жизнедеятельности многочисленных вентиляторов, блоков питания, линейных карт, фабрик коммутации, интерфейсов, процессоров, памяти - вообще хоть как-то работает, а тем более работает настолько стабильно в течение многих лет.

2

1. Терминология

Задача этого параграфа не объять все непонятные слова, употребляемые в статье, а лишь внести некую ясность в неразбериху русско-английских терминов.

Чип коммутации , сетевой процессор, (Packet) Forwarding Engine, PFE микросхема, способная коммутировать пакет из входа в нужный выход с нужным набором заголовков.

Pins, пины, ножки металлические контакты на микросхеме для соединения с основанием.

Data Path путь внутри устройства (или чипа), по которому передвигается пользовательский трафик.

Lookup, лукап, поиск поиск адресата в таблицах (FIB, LFIB, ARP Adjacencies, IPv6 ND Table итд.)

Pipeline или конвейер набор действий, которые происходят с пакетом по мере его продвижения от входа в чип до выхода из него.

Single-chip, одночиповый устройство, внутри которого только один чип.

Fixed, фиксированный, pizza-box немодульный коммутатор. Обычно внутри него нет фабрик коммутации. Часто эти термины используются как синоним Single-chip, хотя это не совсем верно - внутри может стоять два (back-to-back) чипа или даже больше.

SerDes, Сериализация/Десериализация процесс перевода данных из параллельного низкоскоростного интерфейса (МГц) в последовательный высокоскоростной (ГГц) и наоборот. Например, из чипа в интерфейс или из чипа в фабрику.

Память физическая микросхема для хранения.

Буфер некий участок памяти, выделенный для хранения пакетов. *Здесь и далее в производных словах, таких как “буферов”, ударение на “У”*.

Очередь абстракция над буфером, позволяющая виртуально выстраивать пакеты в упорядоченную очередь. Фактически в памяти они, конечно, хранятся "как попало".

OCB - On-Chip Buffer память, встроенная в чип.

Programmable ASIC ASIC, логику работы которого можно изменить путём перепрошивки.

Programmable Pipeline чип с возможностью для сторонних компаний программировать конкретные части ASIC в ограниченных пределах с помощью предоставляемого вендором компилятора.

Packaging упаковка кремниевого кристала в корпус для предотвращения механических повреждений и коррозии.

Advanced Packaging различные технологии, позволяющие внутри одного чипсета компоновать чипы различного вида и назначения. Например, совмещение ASIC и внешней памяти или ASIC и оптического трансивера.

IP Core Intellectual Property. Часть микросхемы, реализующая определённую функцию, например, SerDes.

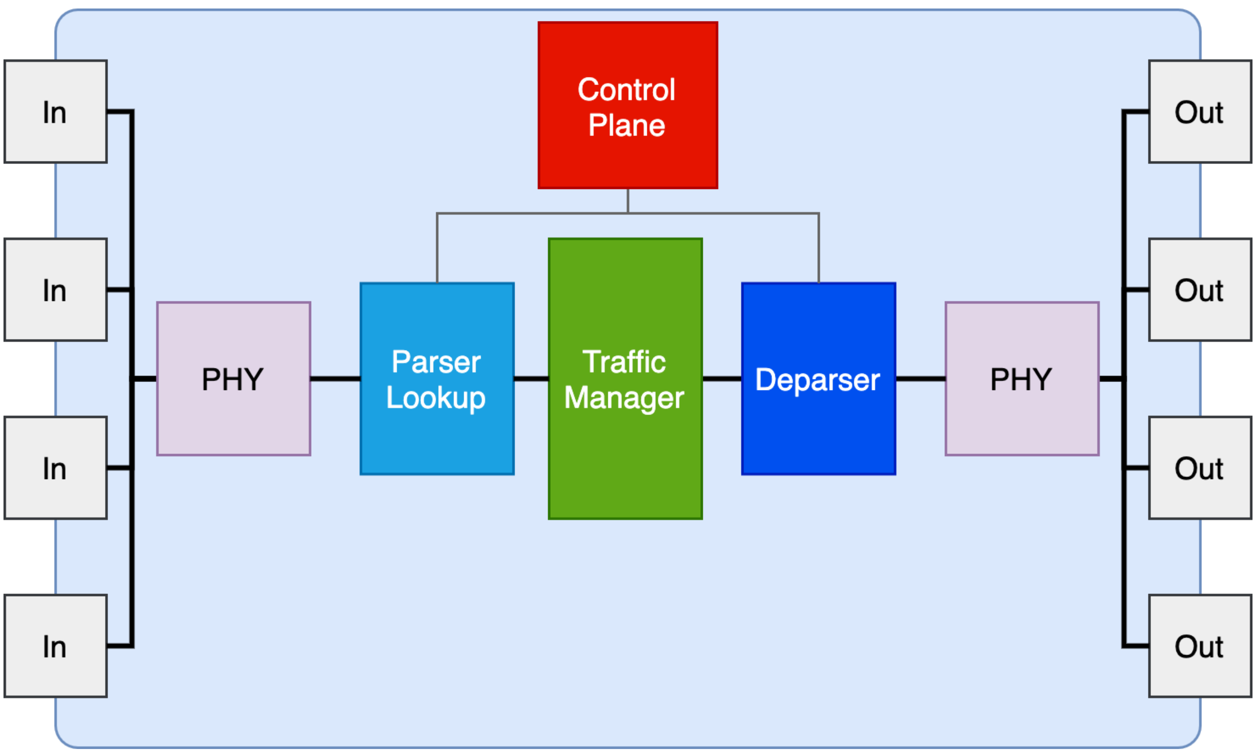
BUM Broadcast, Unknown Unicast, Multicast.

3

2. Архитектура сетевых устройств

Итак, что такое сетевое устройство?

Будь то коммутатор, маршрутизатор, файрвол, балансировщик, программный или аппаратный, его задача - доставить пакет со входа на правильный выход, и состоит оно из следующих частей:



Электрический или оптический сигнал, попадая на устройство через входной физический порт, восстанавливается до потока битов, из него вычленяются отдельные Ethernet-кадры, далее на основе заголовков (Ethernet, IP, MPLS итд) (или иногда содержимого) принимается решение о том, в какой выходной порт этот пакет должен быть далее отправлен и с каким набором заголовков. На своём пути от входного к выходному порту пакет ещё проходит через модуль Traffic Manager, где с ним могут происходить следующие вещи: буферизация, полисинг, шейпинг, обработка по приоритетам.

Это путь самого пакета.

И отдельно от пути пакета - Control Plane Module, который отвечает за то, чтобы путь вообще появился. Это всяческие протоколы маршрутизации, обмена меток и прочее.

Это компоненты, которые присутствуют всегда и во всех сетевых устройствах.

Реализация же этих функций уже зависит от того, о чём именно мы говорим.

Например, на обычном x86 всю работу, кроме PHY могут взять на себя CPU и оперативка.

Более типично, что функции канального уровня заберёт на себя NIC - Ethernet, проверка контрольных сумм.

А можно в компьютер доставить SmartNIC'и, которые аппаратно могут делать, например, туннелирование.

Но мы не будем сегодня про программные реализации сетевых функций. Поговорим о старых добрых материальных коробках, которым всё равно никуда никогда не деться.

Вообще об этом я уже писал, поэтому повторяться не буду. Точнее буду, но не сильно. Точнее сильно, но я добавлю здесь ещё смысла.

Обычно на каждый блок задач выделяется специализированный чип.

Так, всем Control Plane'ом занимаются всё те же **CPU + память**.

Организация взаимодействия со средой передачи и преобразование битов в сигнал и наоборот - специальные чипы **PHY**. Почти всегда они реализуются на ASIC.

Разбор заголовков и поиск пути - **Packet Forwarding Engine**. Это может быть ASIC, Network Processor, реже FPGA и даже CPU. К ним в помощь идёт или обычная память **RAM** или специальная **CAM/TCAM** для хранения таблиц лукапа.

Traffic Manager - если вынесен отдельно, то опять же - узкоспециализированные ASIC'и и плюс к ним память. Но может быть встроенным в чип коммутации.

Один из вариантов реализации (single-chip устройство):

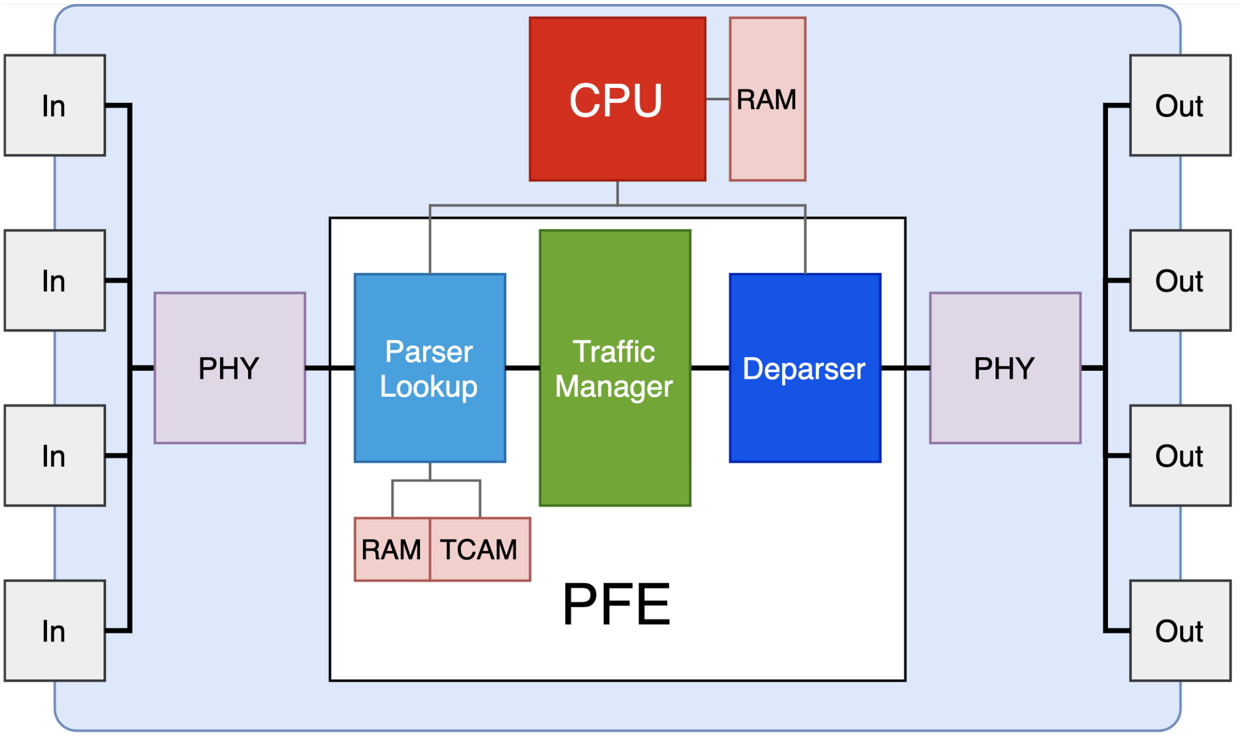


Рис.1: Single-chip устройство

Рис.2: Single-chip коммутатор Mellanox SN2700 на чипе Spectrum

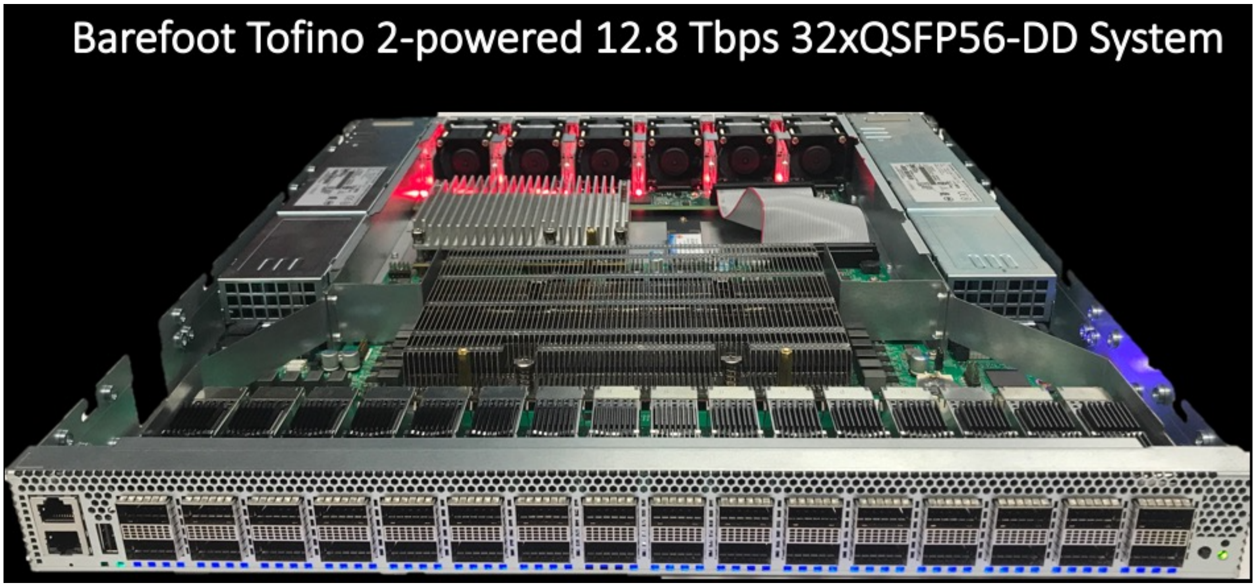


Рис.3: Single-ship коммутатор на базе чипа Barefoot Tophino.

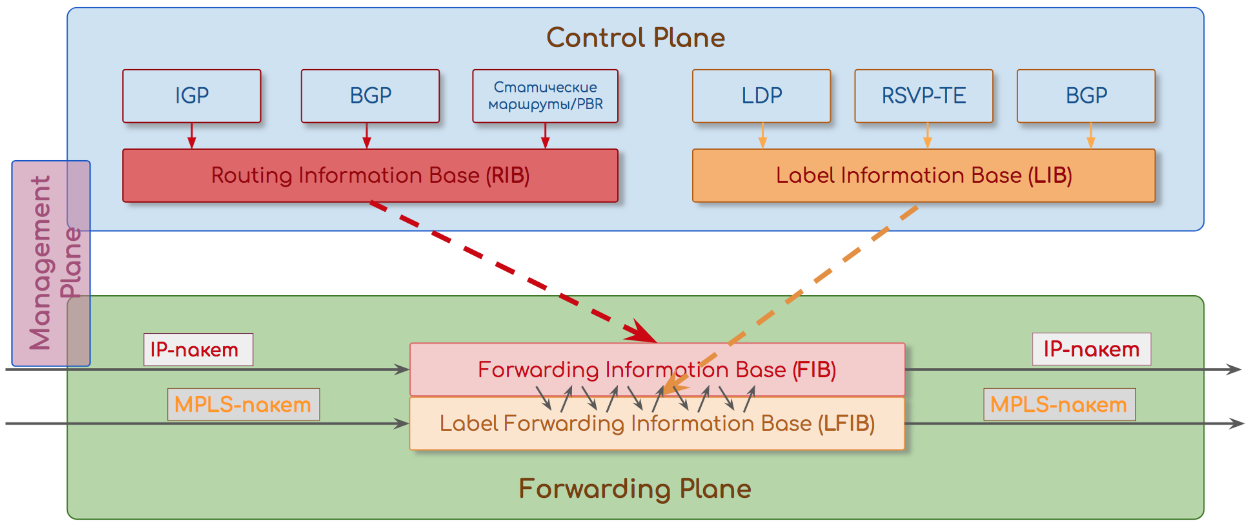
На этом этапе важно понять принципиальную разницу между плоскостями внутри устройства:

Уровни и плоскости

Это настолько важные вещи, что мы не можем не дать им определения.

В работе оборудования можно выделить три уровня/плоскости:

- **Forwarding/Data Plane**
- **Control Plane**
- **Management Plane**



Forwarding/Data Plane

Плоскость пересылки.

Главная задача сети - доставить трафик от одного приложения другому. И сделать это максимально быстро, как в плане пропускной способности, так и задержек.

Соответственно главная задача узла - максимально быстро передать вошедший пакет на правильный выходной интерфейс, успев поменять ему заголовки и применив политики.

Поэтому существуют заранее заполненные таблицы передачи пакетов - таблицы коммутации, таблицы маршрутизации, таблицы меток, таблицы соседств итд.

Реализованы они могут быть на специальных чипах CAM, TCAM, работающих на скорости линии (интерфейса). А могут быть и программными.

Примеры:

1. Принять Ethernet-кадр, посчитать контрольную сумму, проверить есть ли SMAC в таблице MAC-адресов. Найти DMAC в таблице MAC-адресов, определить интерфейс, передать кадр.
2. Принять MPLS-пакет, определить входной интерфейс и входную метку. Выполнить поиск в таблице меток, определить выходной интерфейс и выходную метку. Свопнуть. Передать.
3. Пришёл поток пакетов. Выходным интерфейсом оказался LAG. Решение, в какие из интерфейсов их отправить, тоже принимается на Forwarding Plane.

Разница между Data и Forwarding Plane

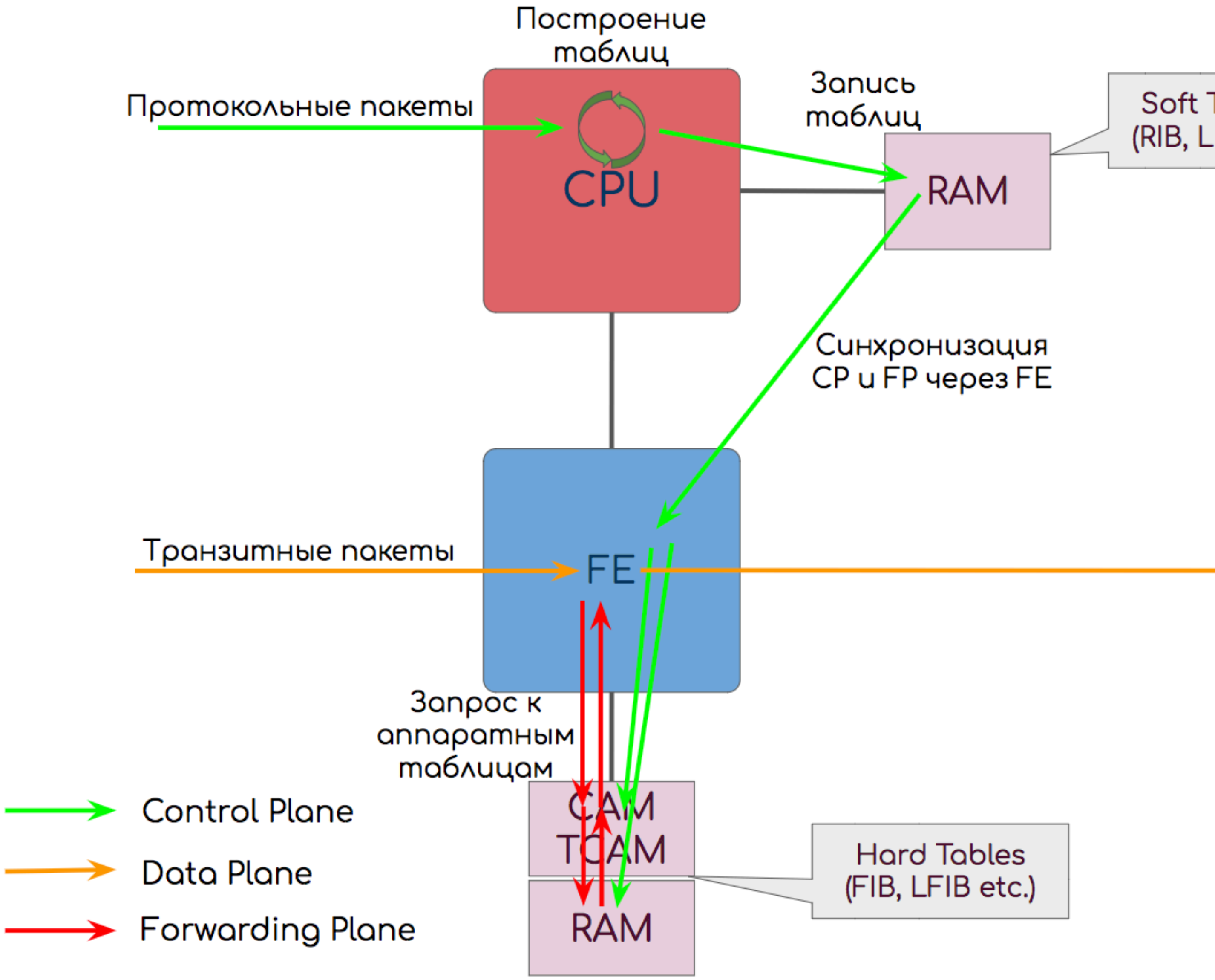
В абсолютном большинстве случаев считается, что Data и Forwarding Plane - это одно и то же.

Однако иногда их разделяют.

Тогда Data Plane означает именно манипуляции с полезной нагрузкой: процесс доставки пакета от входного интерфейса к выходному и обработку его в буферах.

А Forwarding Plane - это обработка заголовков и принятие решения о пересылке.

Примерно так:



Control Plane

Плоскость управления.

Всему голова. Она **заранее** заполняет таблицы, по которым затем будет передаваться трафик.

Здесь работают протоколы со сложными алгоритмами, которые дорого или невозможно реализовать аппаратно.

Например, алгоритм Дейкстры реализовать на чипе можно, но сложно. Так же сложно сделать выбор лучшего маршрута BGP или определение FEC и рассылку меток. Кроме того, для всего этого пришлось бы делать отдельный чип или часть чипа, которая практически не может быть переиспользована.

В такой ситуации лучше пожертвовать сабсекундной сходимостью в пользу удобства и цены.

Поэтому ПО запускается на CPU общего назначения.

Получается медленно, но гибко - вся логика программируема. И на самом деле скорость на Control Plane не важна. Однажды вычисленный маршрут инсталлируется в FIB, а дальше всё на скорости линии.

Вопрос скорости Control Plane возникает при обрывах, флуктуациях на сети, но он сравнительно успешно решается механизмами TE HSB, TE FRR, IP FRR, VPN FRR, когда запасные пути готовятся заранее на том же Control Plane.

Примеры:

1. Запустили сеть с IGP. Нужно сформировать Hello, согласовать параметры сессий, обменяться базами данных, просчитать кратчайшие маршруты, инсталлировать их в Таблицу Маршрутизации, поддерживать контакт через периодические Keepalive.

2. Пришёл BGP Update. Control Plane добавляет новые маршруты в таблицу BGP, выбирает лучший, инсталлирует его в Таблицу Маршрутизации, при необходимости пересылает Update дальше.

3. Администратор включил LDP. Для каждого префикса создаётся FEC, назначается метка, помещается в таблицу меток, анонсы уходят всем LDP-соседям.

4. Собрали два коммутатора в стек. Выбрать главный, проиндексировать интерфейсы, актуализировать таблицы пересылок - задача Control Plane.

Работа и реализация Control Plane универсальна: ЦПУ + оперативная память: работает одинаково хоть на стоечных маршрутизаторах, хоть на виртуальных сетевых устройствах.

Эта система - не мысленный эксперимент, не различные функции одной программы, это действительно физически разделённые тракты, которые взаимодействуют друг с другом.

Началось всё с разнесения плоскостей на разные платы. Далее появились стекируемые устройства, где одно выполняло интеллектуальные операции, а другое было лишь интерфейсным придатком.

Вчерашний день - это системы вроде Cisco Nexus 5000 Switch + Nexus 2000 Fabric Extender, где 2000 выступает в роли выносной интерфейсной платы для 5000.

Где-то в параллельной Вселенной тихо живёт SDN разлива 1.0 - с Openflow-like механизмами, где Control Plane вынесли на внешние контроллеры, а таблицы пересылок заливаются в совершенно глупые коммутаторы.

Наша реальность и ближайшее будущее - это наложенные сети (Overlay), настраиваемые SDN-контроллерами, где сервисы абстрагированы от физической топологии на более высоком уровне иерархии.

Разделение на Control и Forwarding Plane позволило отвязать передачу данных от работы протоколов и настройки сети, а это повлекло значительное повышение масштабируемости и отказоустойчивости.

Так один модуль плоскости управления может поддерживать несколько интерфейсных модулей.

В случае сбоя на плоскости управления механизмы GR, NSR, GRES и ISSU помогают плоскости пересылки продолжать работать будто ничего и не было.

Management Plane

Плоскость или демон наблюдения.

Не всегда его выделяют в самостоятельную плоскость, относя его задачи к Control Plane, а иногда, выделяя, называют Monitoring или Config.

Этот модуль отвечает за конфигурацию и жизнедеятельность узла. Он следит за такими параметрами, как:

- Конфигурация устройства
- Температура
- Утилизация ресурсов

- Электропитание
- Скорость вращения вентиляторов
- Работоспособность плат и модулей.

Примеры:

1. Упал интерфейс - генерируется авария, лог и трап на систему мониторинга
2. Поднялась температура чипа - увеличивает скорость вращения вентиляторов
3. Обнаружил, что одна плата перестала отвечать на периодические запросы - выполняет рестарт плат - вдруг поднимется.
4. Оператор подключился по SSH для снятия диагнонстической информации - CLI также обеспечивается Management Plane'ом.
5. Приехала конфигурация по Netconf - Management Plane проверяет и применяет её. При необходимости инструктирует Control Plane о произошедших изменениях и необходимых действиях.

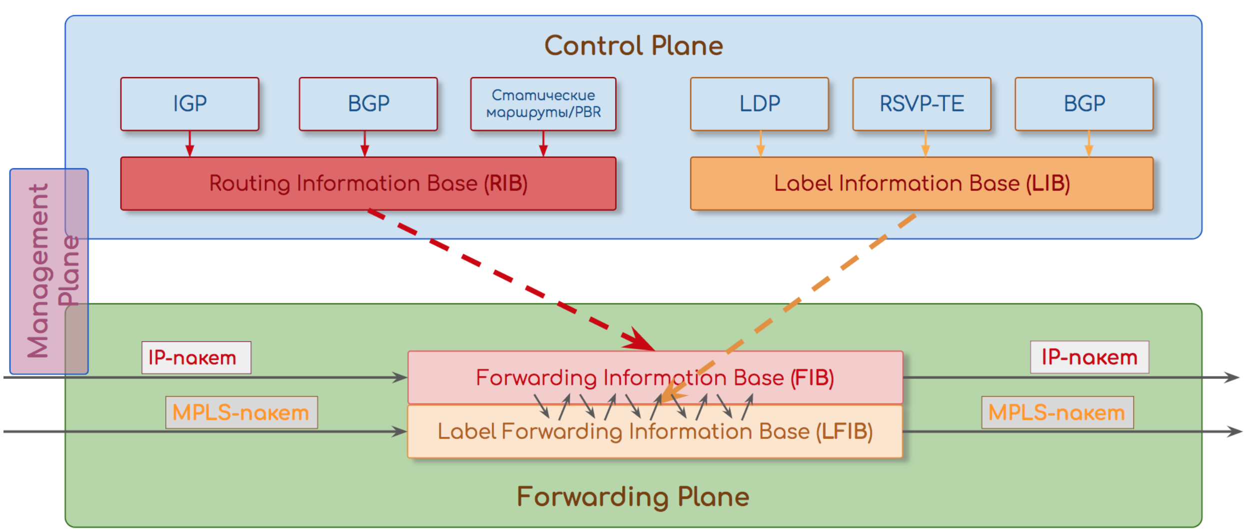
Итак:

Forwarding Plane - передача трафика на основе таблиц пересылок - собственно то, из чего оператор извлекает прибыль. Требуется скорость.

Control Plane - служебный уровень, необходимый для формирования условий для работы Forwarding Plane. Требуется интеллект

Management Plane - модуль, следящий за общим состоянием устройства и обеспечивающий интерфейс конфигурации.

Вместе они составляют самодостаточный узел в сети пакетной коммутации.



Разделение на Control и Forwarding/Data Plane - не абстрактное - их функции действительно выполняют разные чипы на плате.

Так Control Plane обычно реализован на связке CPU+RAM+карта памяти, а Forwarding Plane на ASIC, FPGA, CAM, TCAM.

И перед тем, как продолжить, взглянем на то, как выглядят модульные коробки:

Архитектура модульных устройств

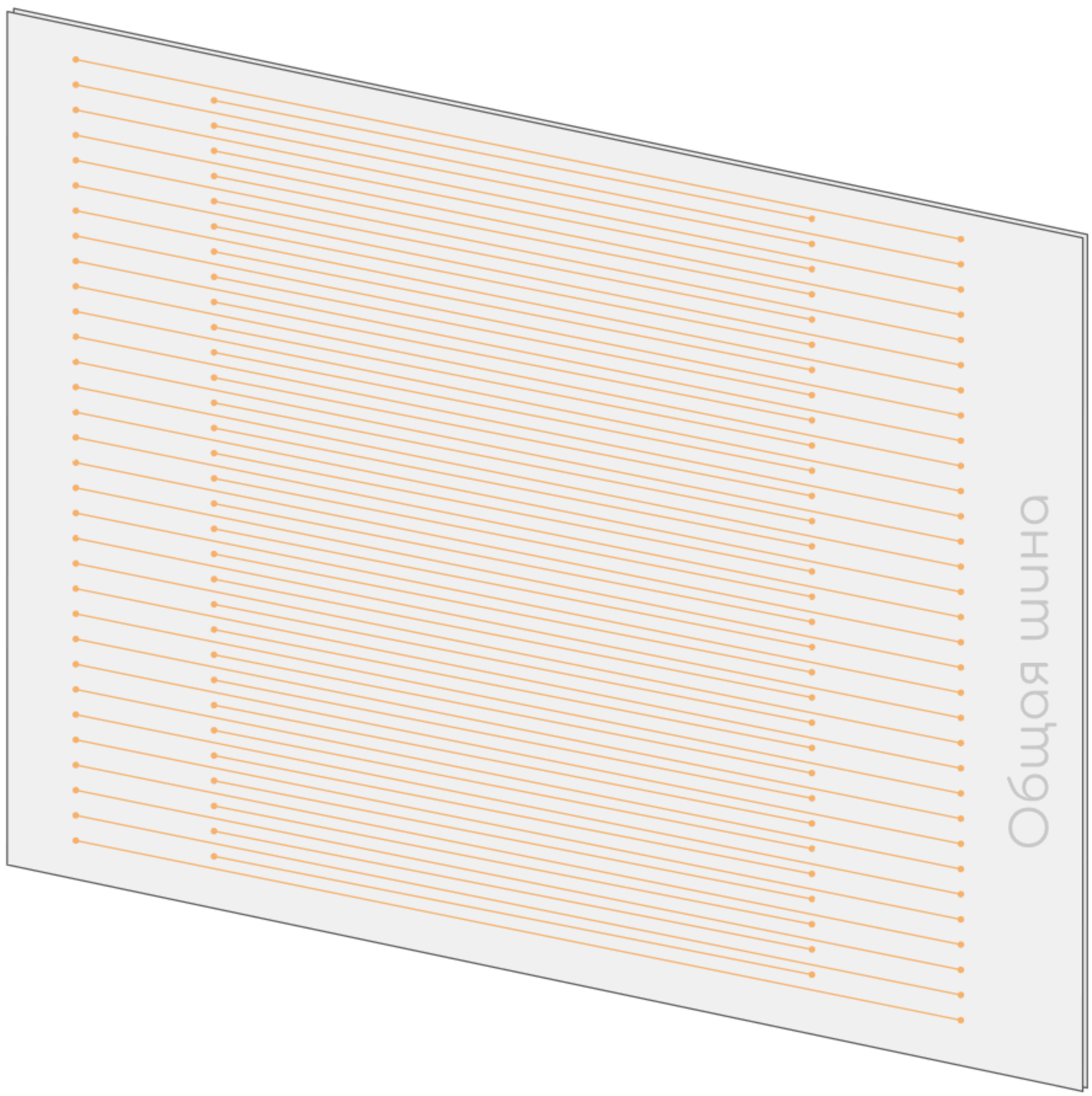
В сетях датацентров долгое время правили огромные монстры размером со стойку - модульные коммутаторы с высокой плотностью портов. Сегодня они уступают место многоуровневым IP-фабрикам, собранным из однаюнитовых одночиповых коробок. Но такие шкафы никуда не уходят с роли бордеров и из сетей операторов. Поэтому мы разберём и их устройство. Тем более, что некоторые вещи будут актуальны и для коммутаторов попроще.

Для этой модели возьмём модульное шасси, состоящее из общей шины, интерфейсных и управляющих модулей и фабрики коммутации.

- *Общая шина*
- *Управляющий модуль*
- *Интерфейсный модуль или линейная карта*
- *Фабрика коммутации*

Общая шина

Общая шина (она же Back Plane, она же Midplane) устройства, связывающая друг с другом все модули. Обычно, это просто батарея медных контактов без каких-либо микросхем.



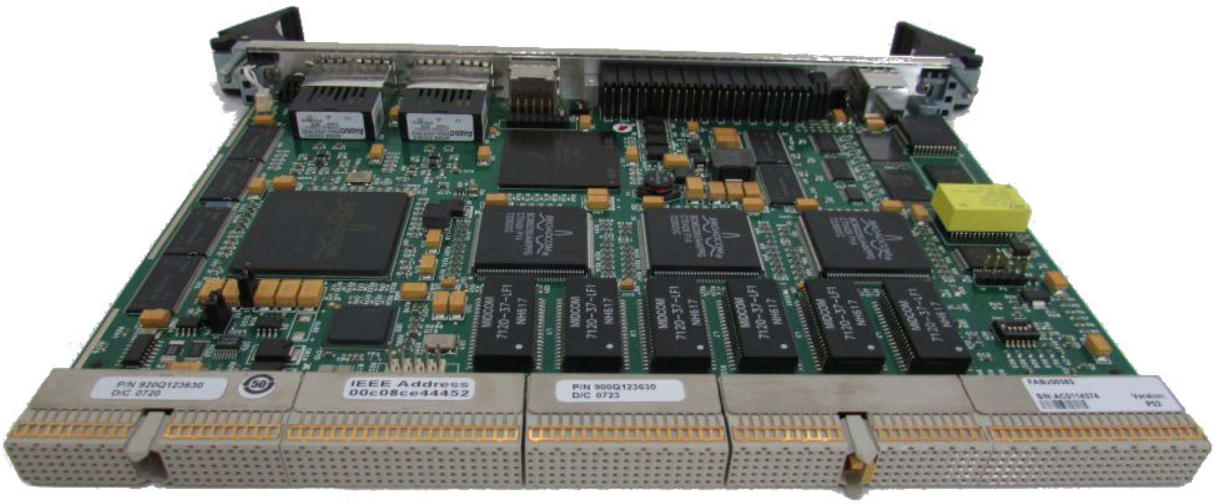


Рис.4: Так выглядит задняя часть платы, которая вставляется в шину

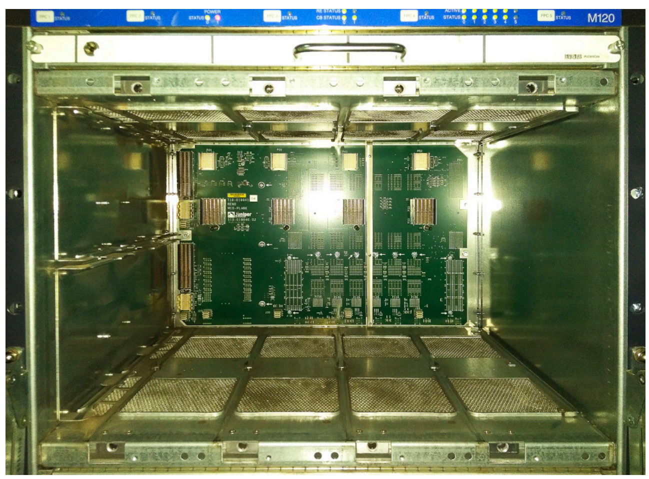
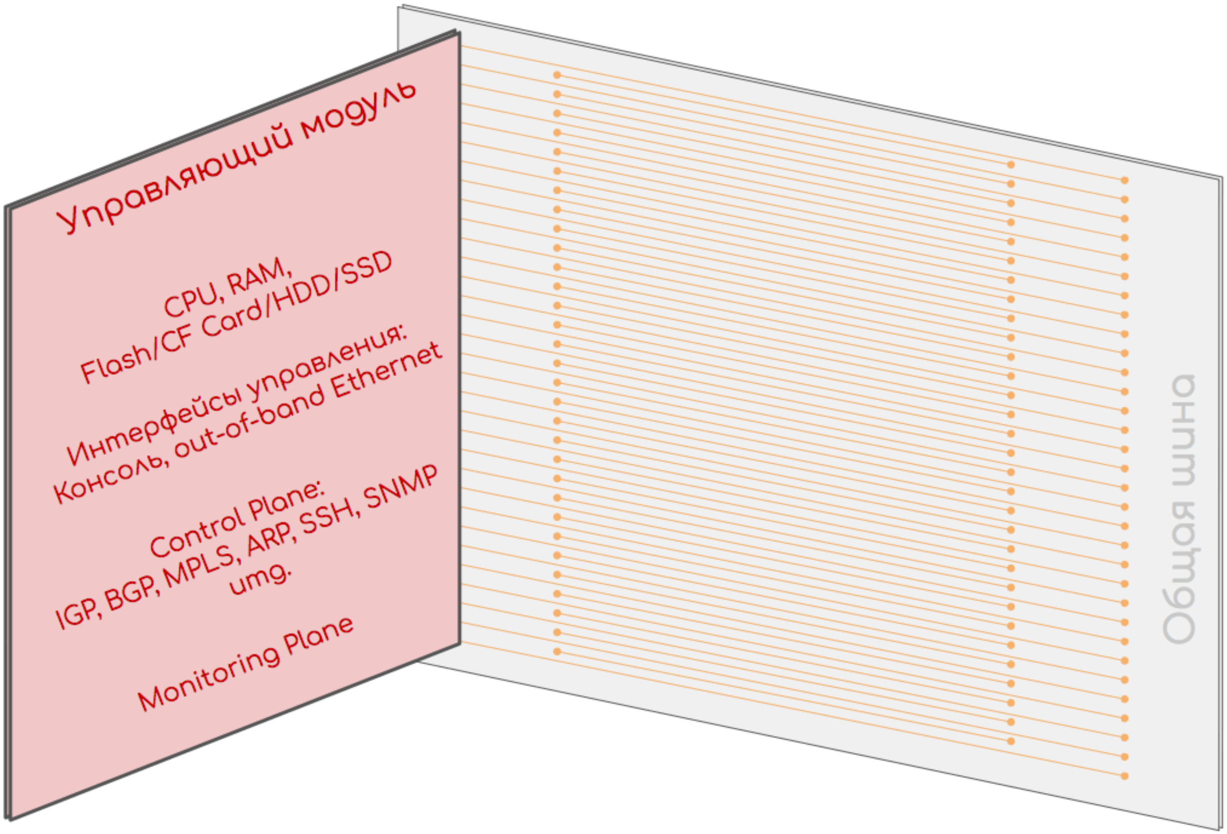


Рис.5: Так выглядит общая шина Juniper

Рис.6: Так выглядит общая шина Huawei

Вот эти две полоски посередине, набитые медными контактами - слоты для фабрик коммутации.

Управляющий модуль



На нём расположены CPU, оперативная память, постоянная память для хранения ПО, конфигурации и логов, интерфейсы для управления.

Он отвечает за Management Plane и за Control Plane.

С ним мы работаем, когда подключаемся к устройству по telnet/ssh.

Он загружает ПО в оперативную память и запускает все другие модули при подаче питания.

Он следит за Heart beat других модулей - специальными пакетами, получение которых говорит о том, что модуль жив и работоспособен.

Он же может перезагрузить модуль, если Heart beat не получил (как программно, так и выключить питание на плате).

Протокольные пакеты доставляются на CPU и тот, обработав их, совершает какое-то действие, как то: записать обновления в таблицы коммутации, сформировать ответный пакет, запросить информацию о каком-либо компоненте итд.

Управляющий модуль занимается расчётом SPF, LSP, установлением соседств по разным протоколам. Он записывает таблицы коммутации в Soft Tables оперативной памяти.

Так выглядят управляющие платы разных устройств:

Рис.7: Cisco ASR9000 RSP (Routing and Switching). Та половина, что ближе к нам. Та, что ближе к шине, - это фабрика коммутации.

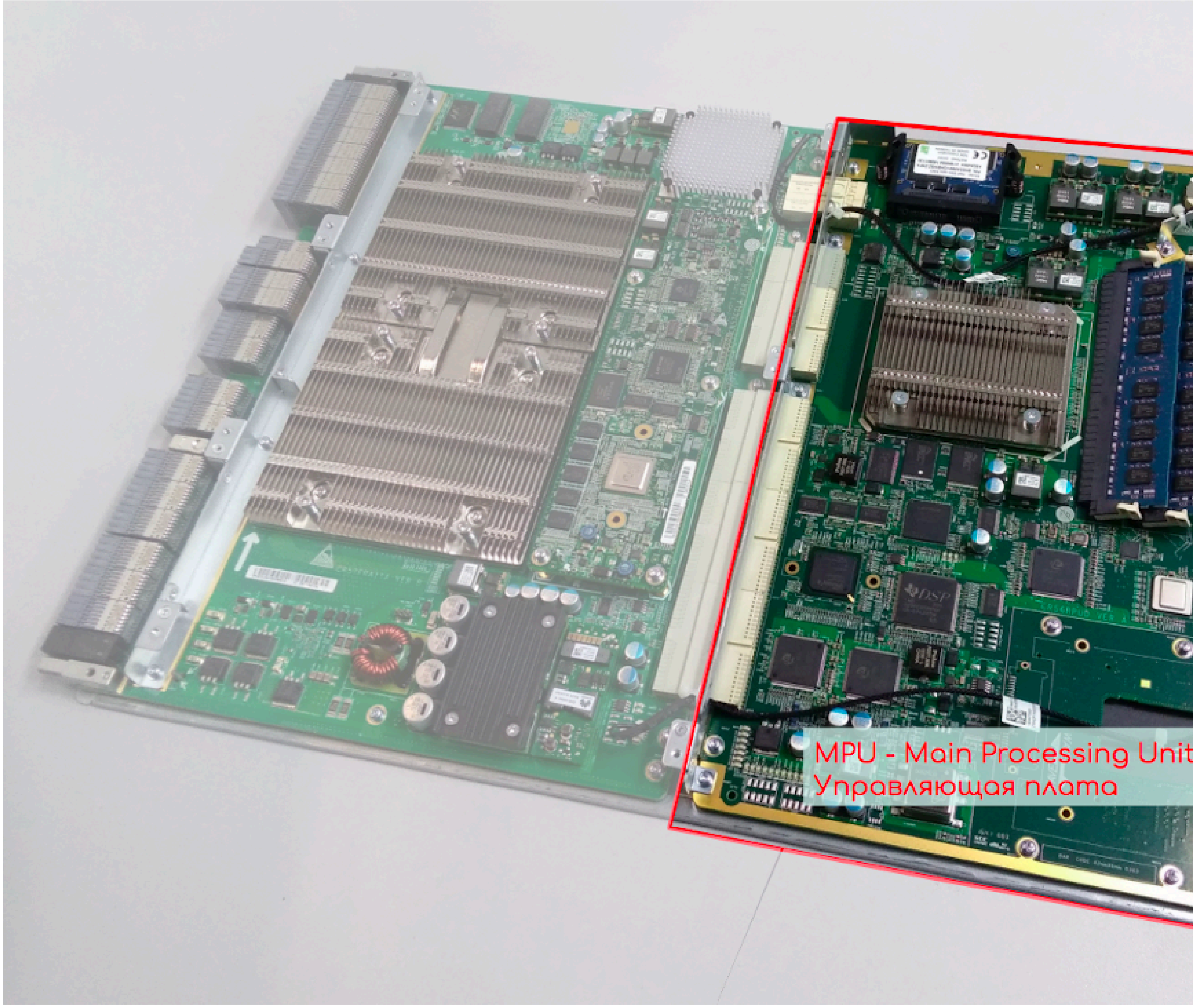


Рис.8: Huawei NE40E-X8 MPU

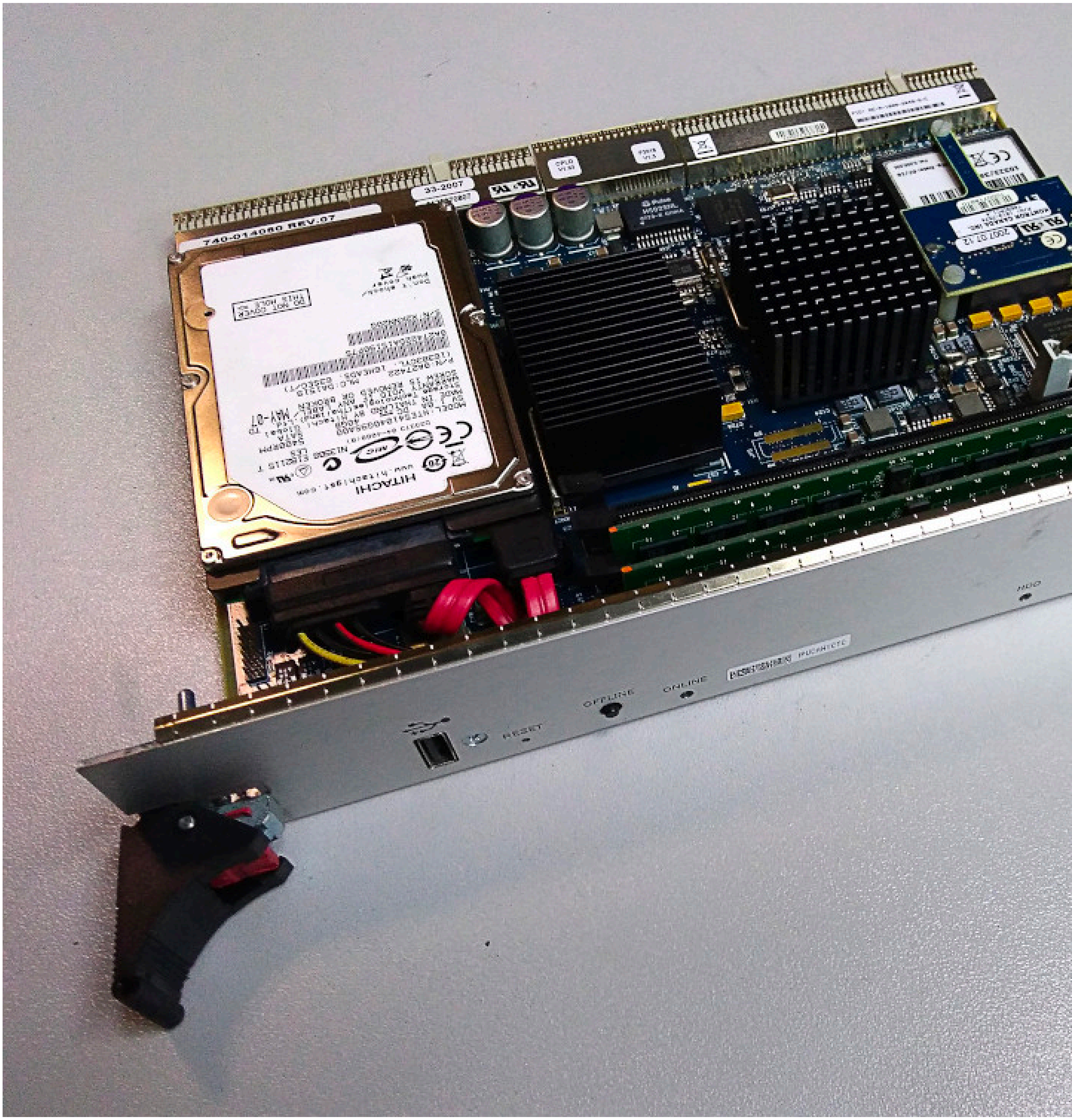


Рис.9: Juniper RE100

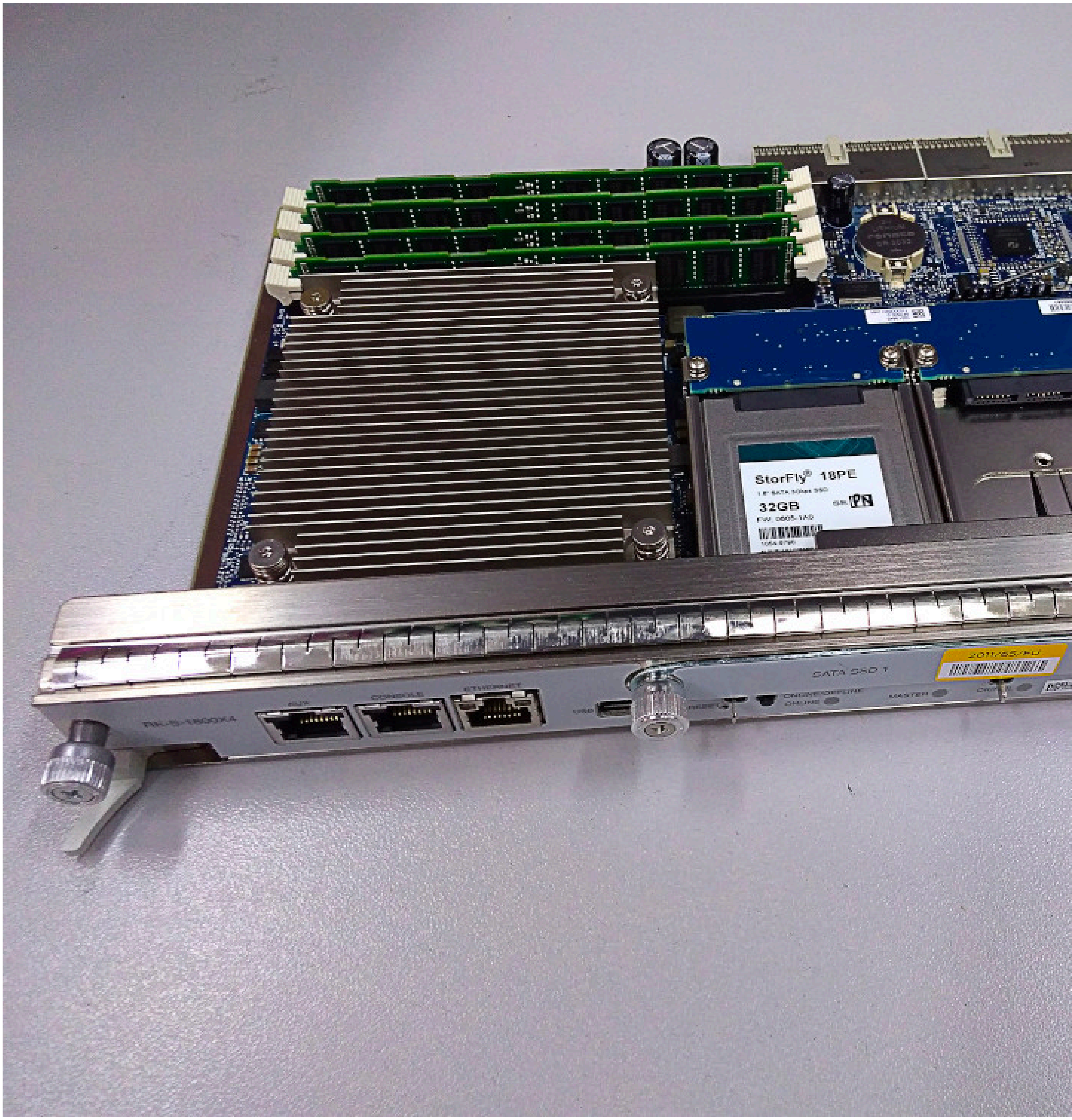


Рис.10: RE Juniper 1800

На всех фотографиях вы можете легко найти CPU, RAM и батарейку BIOS. На некоторых есть

HDD, на других Compact Flash. Да, вы правы - это обычный ПК. Причём современные управляющие платы действительно имеют производительность на уровне компьютера 5-6 летнего возраста.

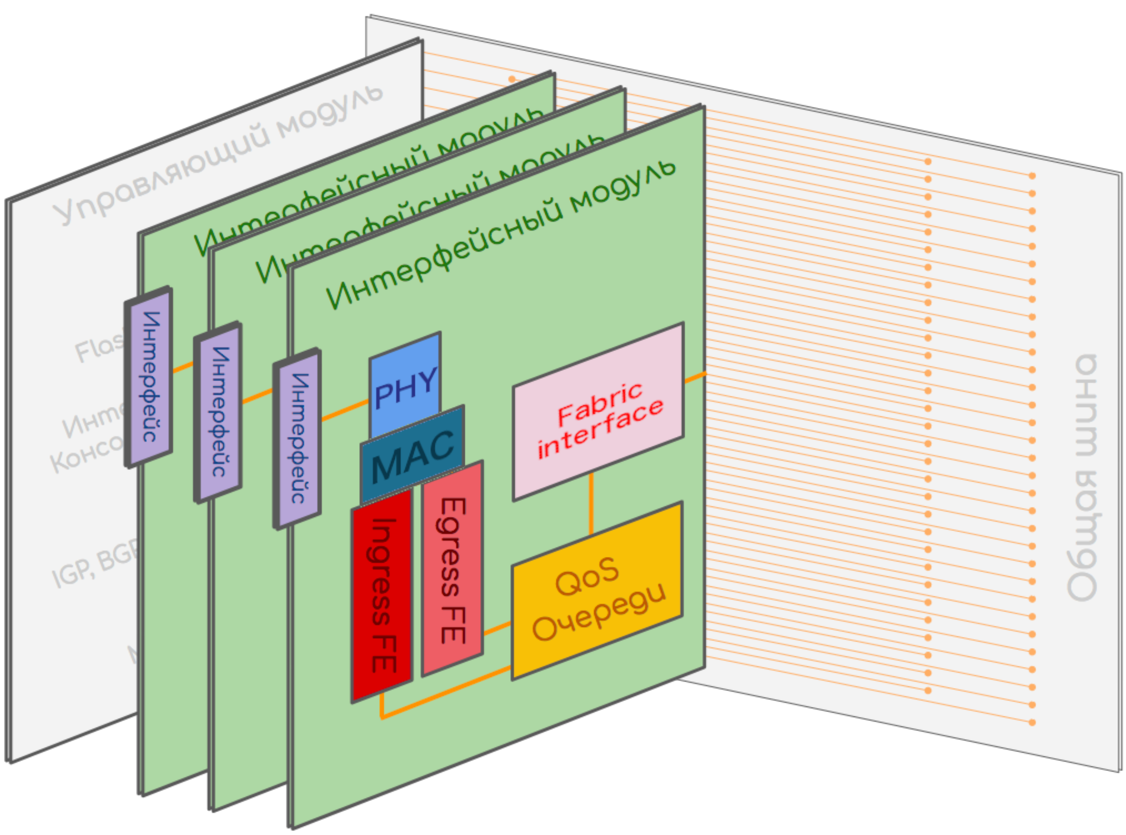
Интерфейсный модуль или линейная карта

Это модуль, который несёт на себе физические интерфейсы и FE (чип коммутации) и выполняет функции Forwarding Plane

По большому счёту линейнкая карта модульного коммутатора - это тот же самый стоечный коммутатор только в форм-факторе платы.

Кроме интерфейсов и чипа коммутации в нём так же есть и CPU, и память, и блоки TM итд. Просто вместо обычных аплинков у неё коннекторы в общую шину, ведущие в другие платы, и ещё ей нужно общаться с управляющей платой, чтобы синхронизировать своё состояние.

Поскольку вся книга посвящена устройству коммутаторов, сильно акцентироваться я в этом месте не буду.



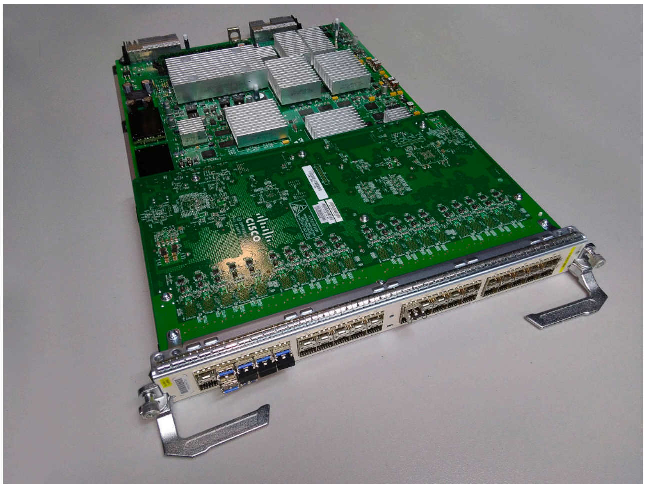
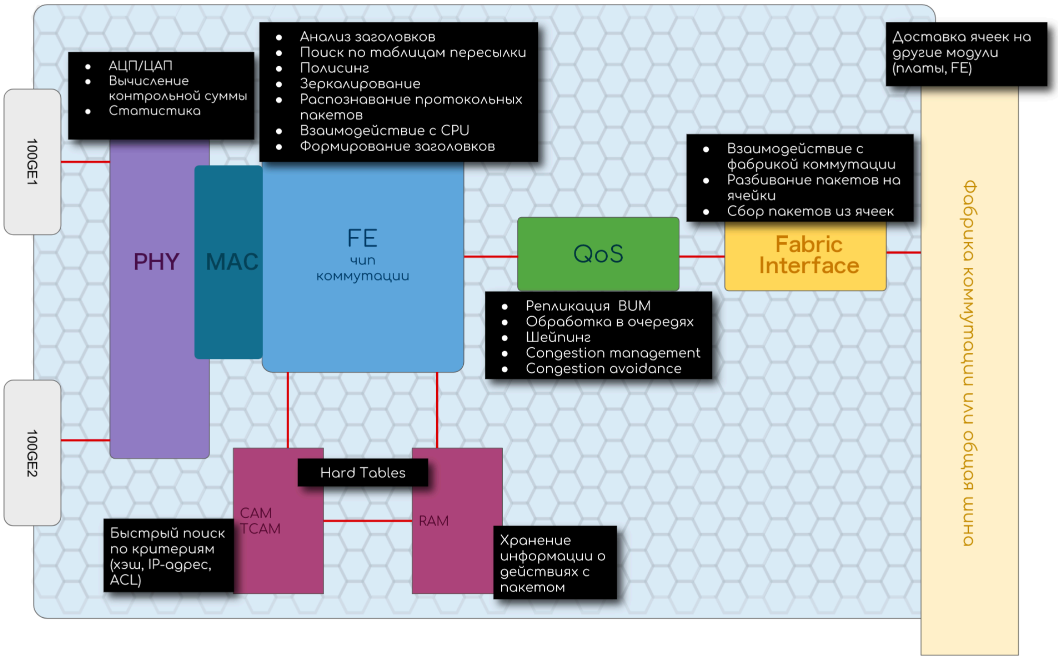


Рис.11: Так выглядит линейная карта Cisco

Плата состоит из многих компонентов, которые могут быть реализованы как в одном чипе (System-on-Chip), так и на множестве отдельных в зависимости от класса устройства и архитектуры.



Основные части линейной карты:

- Собственно интерфейс

- PHY
- MAC
- FE - Forwardin Engine (Ingress/Egress)
- TM - Traffic Manager (QoS)
- Fabric Interface

Иногда часть этих компонентов может выноситься на отдельный извлекаемый модуль - **PIC** (Physical Interface Card). | Например, на нём могут быть интерфейсы, PHY и MAC.

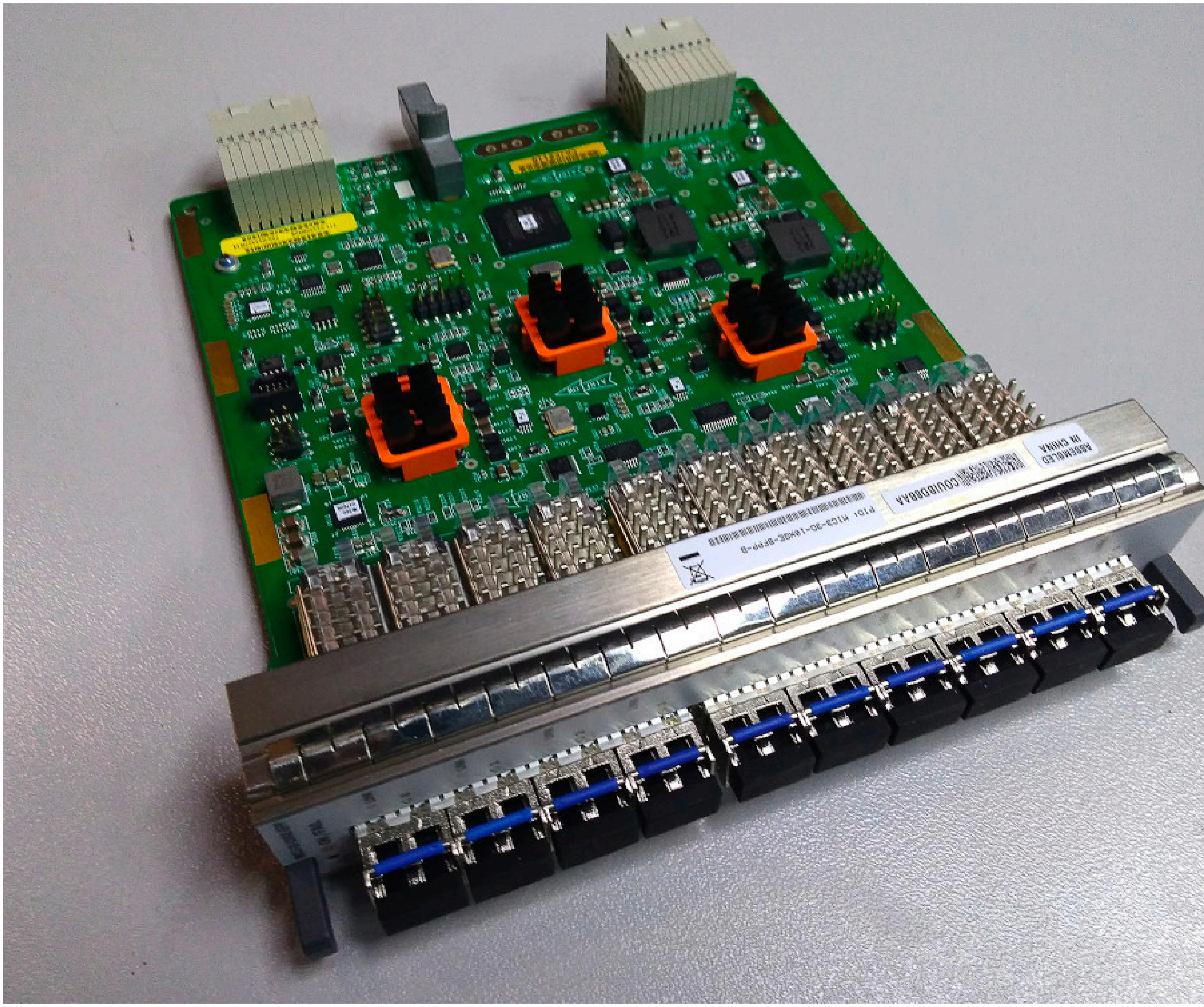


Рис.12: Так выглядит Juniper MIC (Modular Interface Card), совмещённый с чипом PIC

Интерфейс

Электрический/оптический любого форм-фактора.

Реализует возможность подключения провода в устройство через стандартный тип порта.

PHY

PHY занимается задачами физического уровня:

- Конвертация сигнала между средами (оптика-медь), если это нужно
- Восстановление битов из сигналов и наоборот
- Коррекция ошибок
- Синхронизация

Если интерфейс обычный медный, интегрированный, то чип PHY находится на плате и металлическими дорожками соединён с самим интерфейсом.

Если же интерфейс предполагает использование трансивера, то чип PHY обычно находится уже на этом трансивере.

MAC

Модуль MAC выполняет задачи канального уровня - Ethernet. Чаще всего является частью чипа коммутации, но не всегда.

FE - Forwarding Engine

FE или чип коммутации реализует все функции, связанные с коммутацией и маршрутизацией:

- Запросы к CAM/TCAM
- Трансформация Soft Table в Hard Table
- Принятие решения о передаче пакета (ACL, полисинг)
- Коммутация/Маршрутизация
- Маркировка приоритетов
- Зеркалирование
- Обнаружение протокольных пакетов
- Обработка сигналов/пакетов от CPU.

Далее **ВНИМАНИЕ**! Это один из очень важных моментов!

Во-первых, FE делится на **Ingress FE** и **Egress FE**. Первый обрабатывает соответственно пакеты на входном тракте, второй - на выходном.

С одной стороны это разделение терминологическое - пакет пришёл на Ingress FE и далее должен быть отправлен на Egress FE, возможно, другой платы.

С другой, разделение - зачастую вполне физическое: внутри одного FE чипа живут эти две сущности: Ingress и Egress. Это и логично - ведь плата может быть как точкой входа, так и точкой выхода.

Это же верно и для одночиповых коробок, монолитный чип коммутации резделяется на входной и выходной тракты.

Во-вторых, именно входной FE входной линейной карты определяет всю дальнейшую судьбу пакета в пределах узла:

- Вид будущих заголовков
- Приоритет внутри узла и при передаче вовне
- Выходной FE и интерфейс
- Какой именно из физических членов LAG или ECMP

с небольшой оговоркой, что выходной тракт всё-таки может ещё произвести репликацию пакета или зарезать его по ACL

В-третьих, FE должен идентифицировать протокольные и исключительные пакеты в транзитном трафике и передавать их на CPU.

Соответственно и получать пакеты (или инструкции) от CPU - тоже его работа.

Рядом с FE находятся CAM, TCAM и RAM, куда FE обращается в поиске выходного интерфейса и проверки ACL.

Они хранят Hard Tables.

QoS или TM - Traffic Manager

В одночиповых коммутаторах TM обычно является частью чипа коммутации, в модульных может стоять отдельно.

Управлению трафиком, памяти и буферам посвящена львиная часть этой книги, поэтому здесь коснусь этого только вскользь.

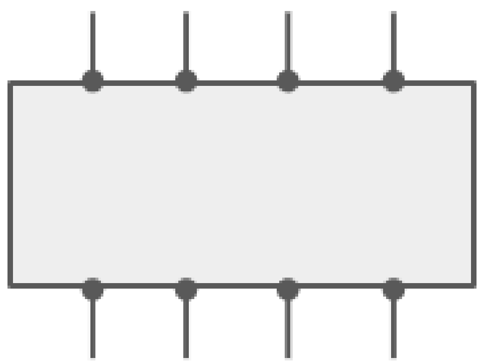
Задача TM - выделять очереди, следить за перегрузками и рапоряжаться трафиком таким образом, как того требует дизайн QoS и маркировка пакетов (Congestion Avoidance и Congestion Management). Он же занимается репликацией BUM-трафика.

Fabric Interface

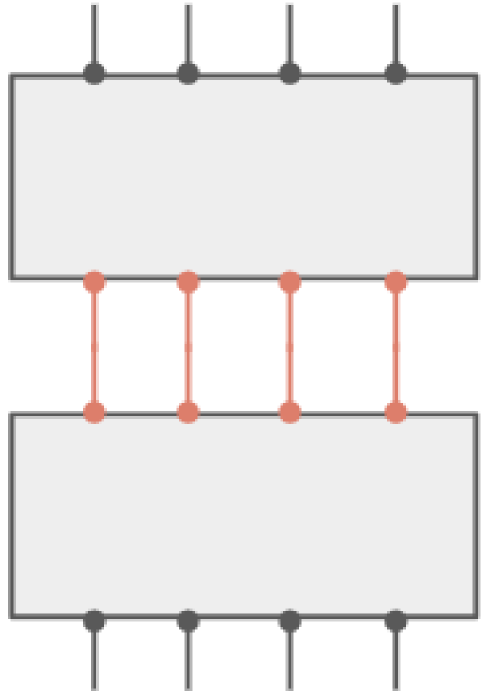
Чип, который обеспечивает взаимодействие с фабрикой коммутации. Может быть как частью FE, так и самостоятельным. Зачастую трафик на фабрику отправляется не как есть, а разбивается на ячейки одинакого размера, чтобы максимально равномерно загрузить все существующие линии. Это, а также передача метаданных - задача данного чипа.

Фабрика коммутации

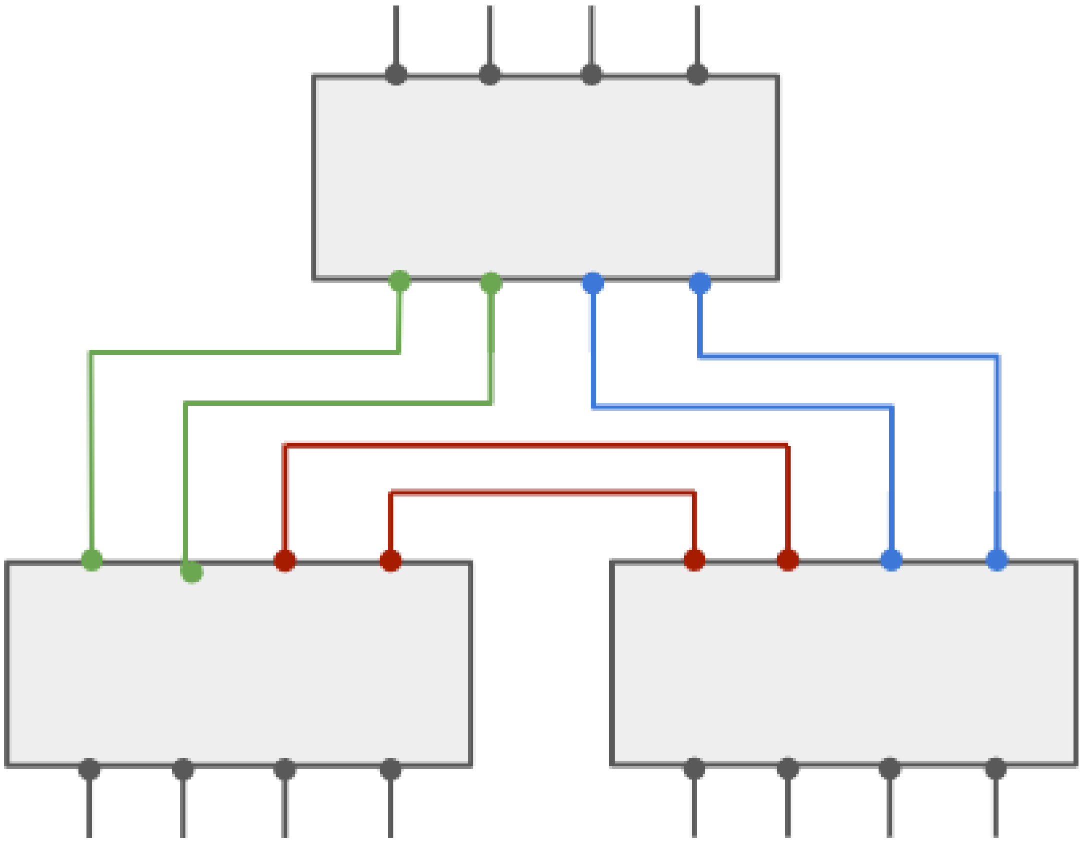
Если мы возьмём Hi-End маршрутизатор операторского класса, то обычно в нём может насчитываться до двух десятков интерфейсных плат, в каждой из которых установлен как минимум один чип коммутации FE. Каждый чип коммутации смотрит частью своих ног в сторону интерфейсов, а частью в сторону задней шины. И ног там предостаточно, потому что медная среда имеет свой предел по пропускной способности - нам не хватит одного-двух выходов.



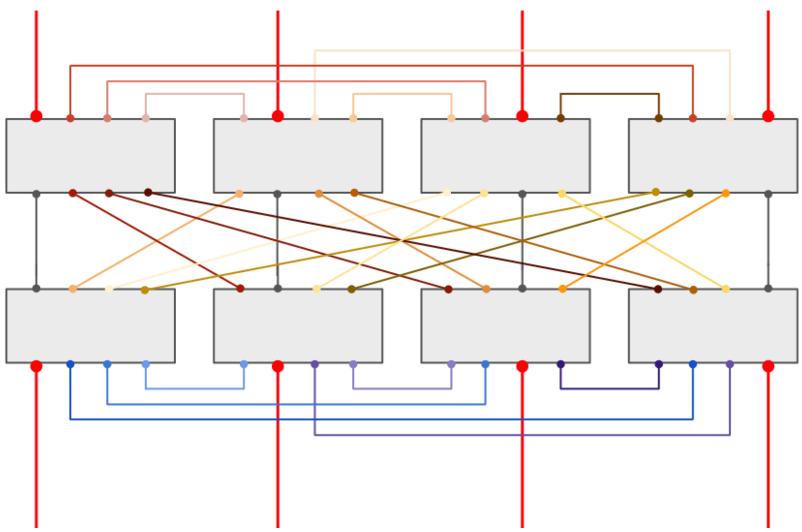
Как связать друг с другом два чипа коммутации? Ну просто же:

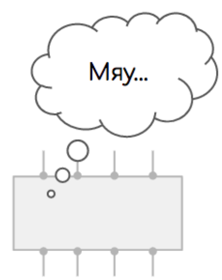


Как связать друг с другом три чипа? Ну, наверное, как-то так?



Как связать 8?





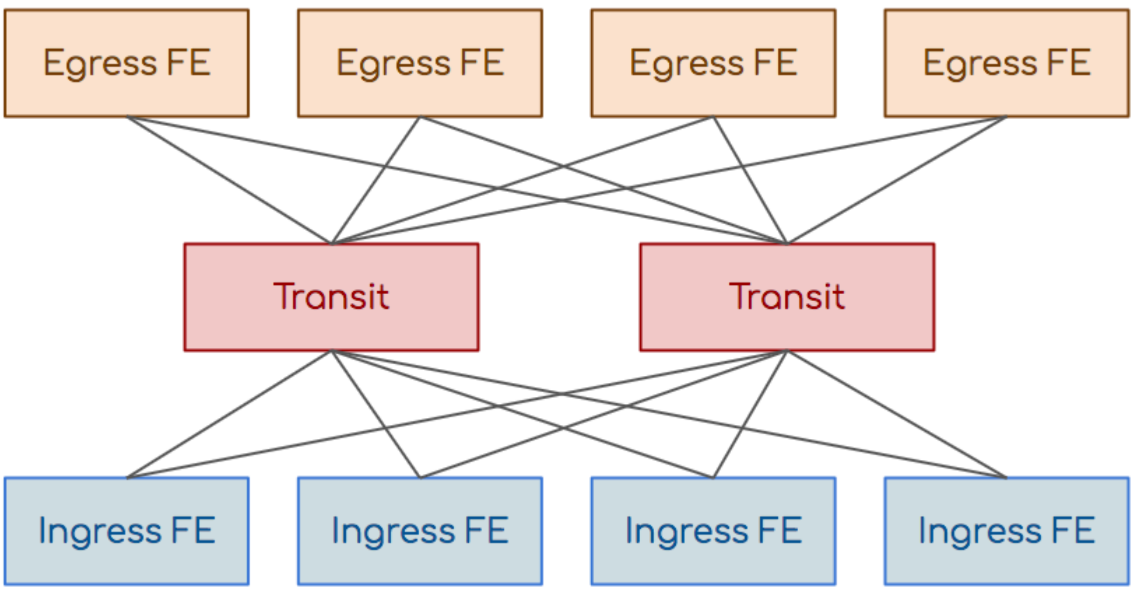
Уверены? Ничего не смущает?

Пропускная способность системы из 8 чипов остаётся той же, что и у пары - ведь каждый раз мы уменьшаем количество ног для связи.

Второй момент, как нам вообще создать полносвязную топологию, если чипов, допустим, 16, и каждый из них имеет по 32 контакта? 16*15/2 пучков кабелей по 32 жилы в каждом?

Эта проблема была адресована неблокирующимся сетям Клоза или сетям без переподписки.

У нас есть входные коммутационные элементы (Ingress FE), выходные (Egress FE) и транзитные. Задача транзитных - связать входные с выходными. Любой входной связан с любым выходным через транзитный.



Входные и выходные не связаны друг с другом напрямую, транзитные также не имеют связи.

Вот этим и напичканы фабрики коммутации в современных маршрутизаторах - очень тупые ASIC, которые только и умеют, что быстро перекладывать пакеты со входа на выход.

Плата коммутации подключается к задней шине и имеет связность со всеми другими платами.

Обычно они работают в режиме N+1 - то есть все разделяют нагрузку, но при выходе из строя одной платы, оставшиеся берут всё на себя.

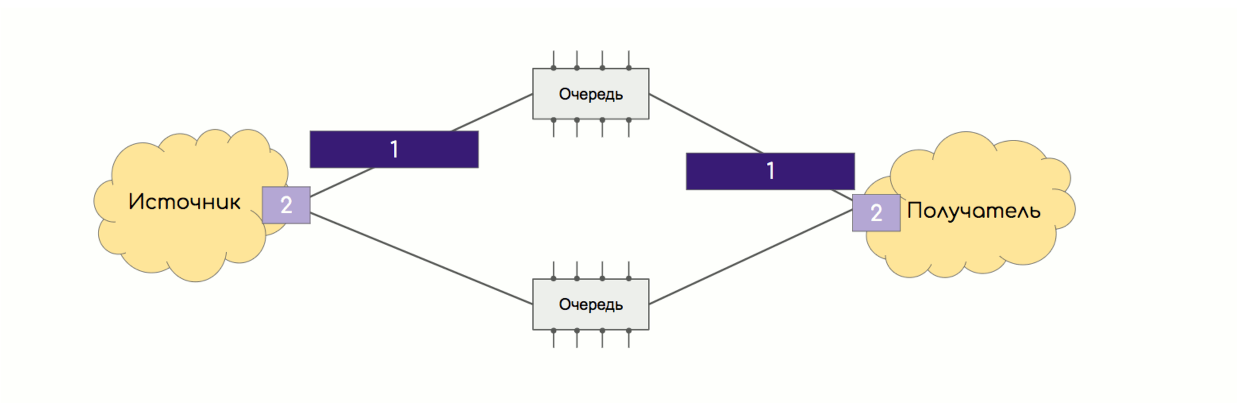
Кстати, сами платы можно вполне назвать верхним каскадом иерархии фабрики Клоза.

Очень подробно топологии Клоза и их применения в современных ДЦ я разбирал в статье Как построить Гугл. Или сети современных датацентров

Остался только вопрос по ячейкам. Ну и перекладывали бы эти ASICи пакеты сразу, зачем их ещё нарезать?

Здесь можно провести аналогию с ECMP. Если кто-то когда-либо настраивал попакетную балансировку между различными путями, то он, наверняка, помнит, сколько боли это доставляло. Неупорядоченная доставка пакетов, с которой с горем пополам справляется TCP, может основательно поломать IP-телефонию или видео, например.

Проблема в попакетной балансировке в том, что два пакета одного потока спокойно могут пойти разными путями. При этом один из них маленький и очень быстро долетит до получателя, а другой акселерат-переросток - застрянет в узком буфере. Вот они и разупорядочились.



То же происходит и на фабрике.

Неплохой метод борьбы с этим - попоточная балансировка - вычисляется хэш по кортежу значений (SMAC, DMAC, SIP, DIP, Protocol, SPort, DPort, MPLS-метка итд.) и все пакеты одного потока начинают передаваться одним путём.

Но это работает неидеально. Зачастую один очень жирный поток может нагрузить один линк в то время, как другие будут простаивать. И с этим можно смириться на сети оператора, но нельзя в пределах этого синего ящика.

Элегантное решение выглядит следующим образом:

Пакеты нарезаются на ячейки одинакового маленького размера.

Ячейки балансируются поячеечно. То есть одна ячейка сюда, другая - туда, третья - в следующий линк итд.

Каждая ячейка пронумерована, поэтому, когда она приходит на нужный FE - легко собирается обратно в целостный пакет.

Поскольку расстояние от входа до выхода примерно одинаковое, размеры ячеек одинаковые, время их доставки тоже примерно одинаковое.

Идея Чарльза Клоза, которая сначала была реализована на телефонных станциях, затем была заимствована в Ethernet-коммутаторы и далее маршрутизаторы, ныне нашла своё место в сетях ЦОДов, заменив собой классическую трёхуровневую модель.

Так выглядят фабрики коммутации

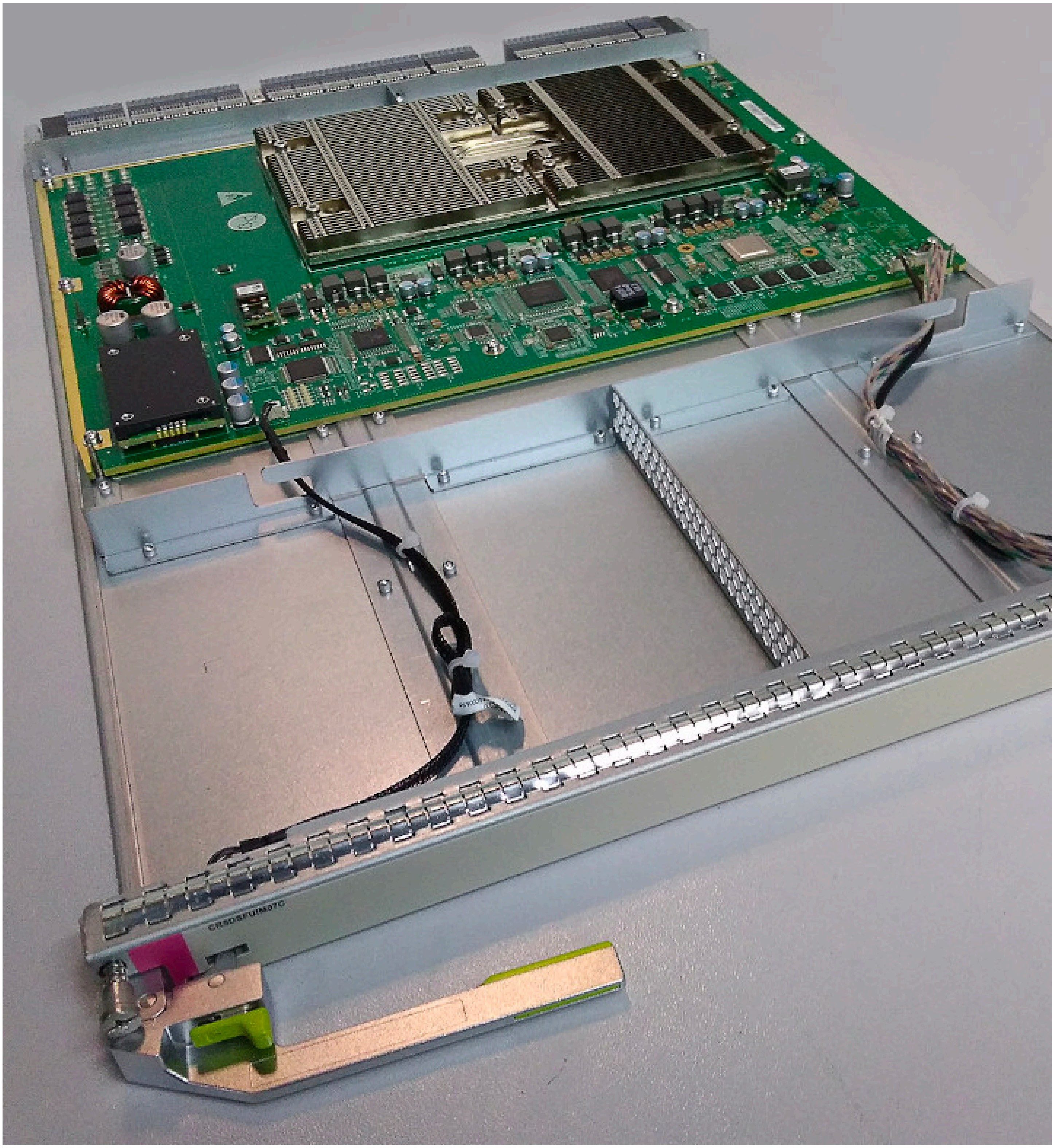


Рис.13: Huawei NE40E-X16

Часто фабрика совмещается с управляющим модулем в одном слоту для экономии пространства в шасси и оптимизации вентиляции.

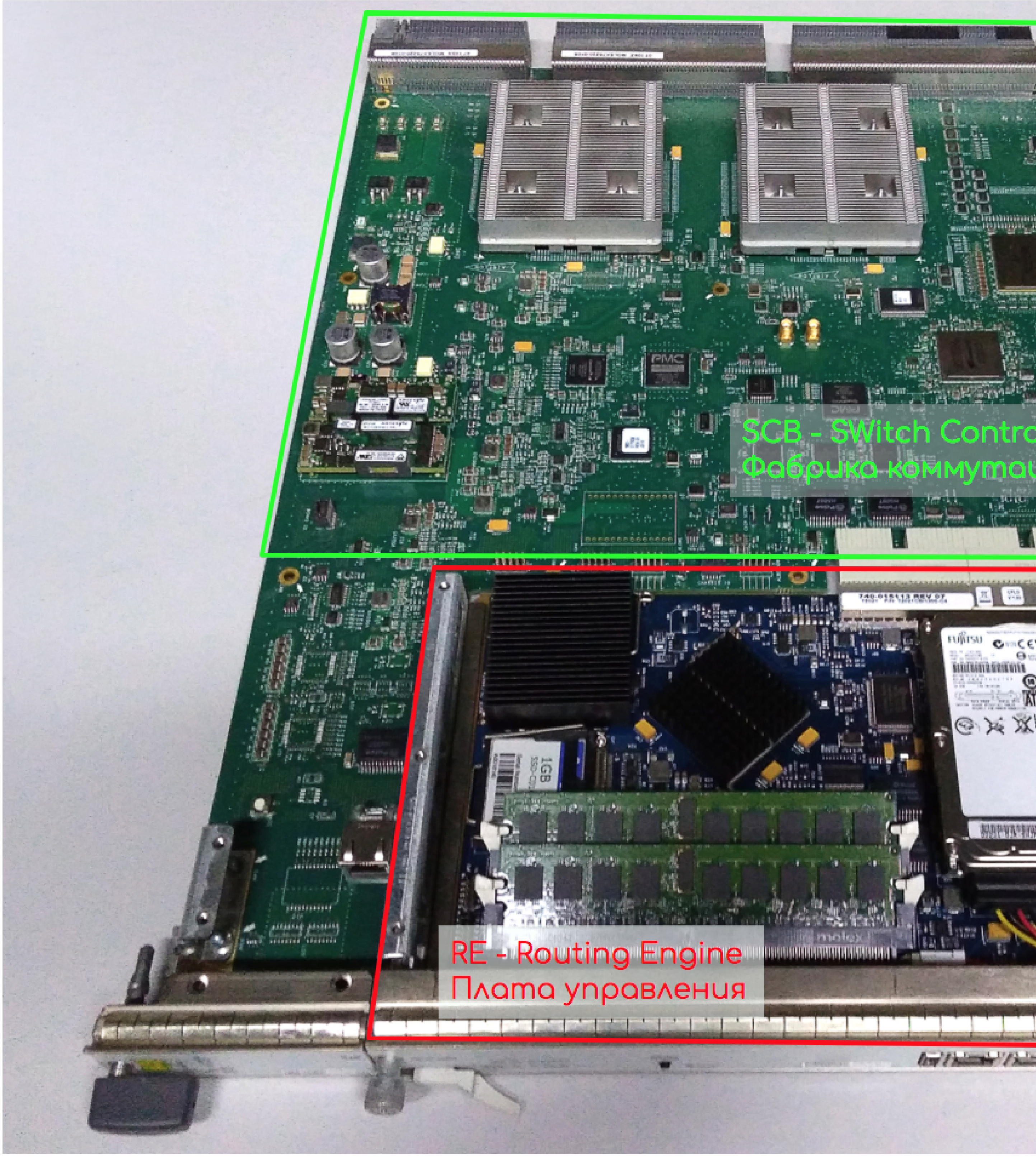


Рис.14: Juniper

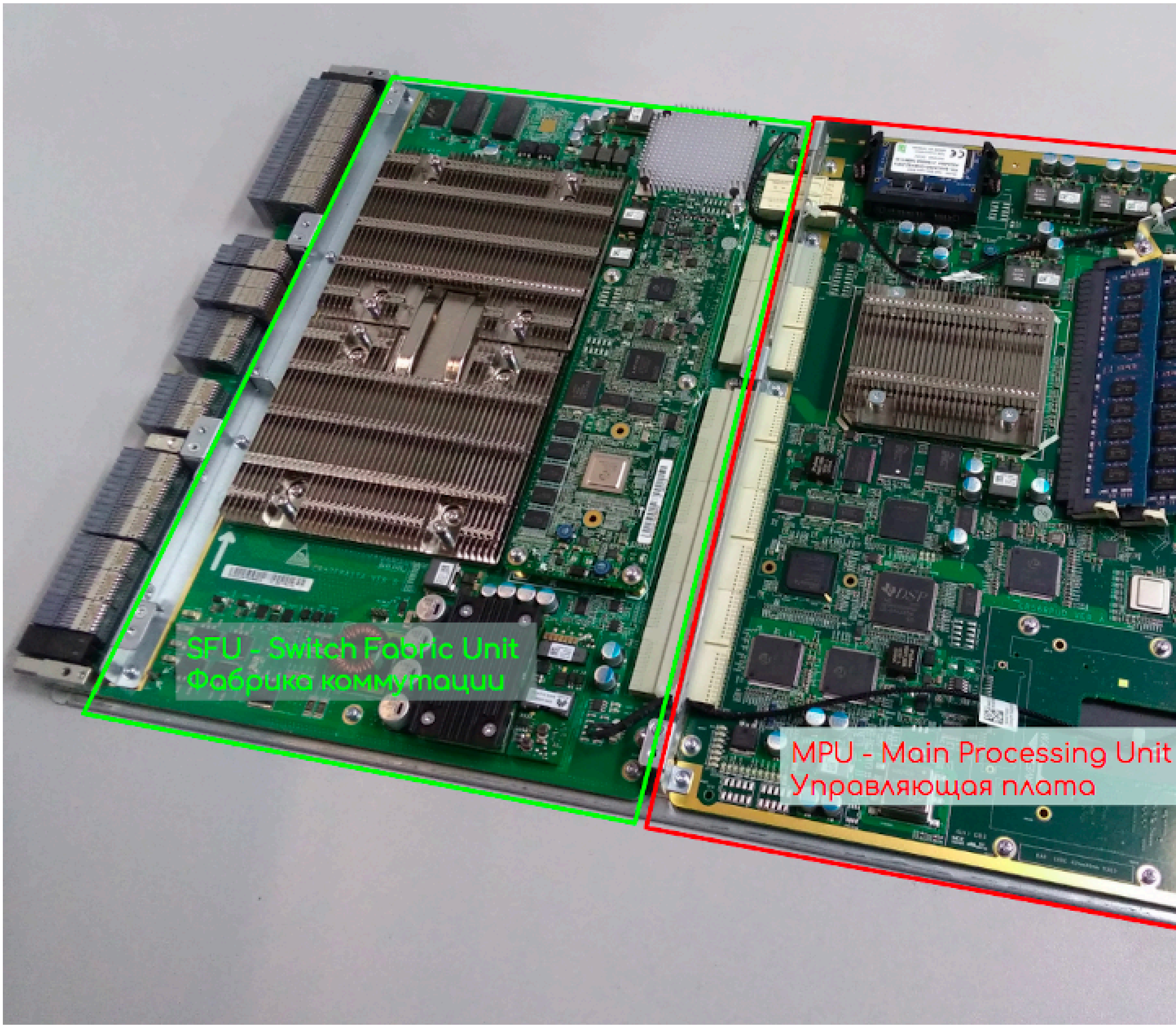


Рис.15: Huawei NE40E-X8

Поскольку вся книга посвящена обработке и хранению пакетов, я сосредоточусь на PFE и TM, которые и могут вносить вариативные задержки в доставку.

4
3. Типов-Чипов

Чипов чудовищное многообразие и простого короткого ответа на вопрос, какой выбрать - нет. Речь и про типы, и про изготовителей, и про серии, и про характеристики. Ниже я попытаюсь разложить все эти штуки на составляющие детали.

А начнём с типов чипов.

Компромиссы: скорость, функциональность, гибкость, цена

В IT всё есть компромисс. Всегда приходится чем-то жертвовать во благо другого.

Вот классический цискин треугольник про компромиссы:

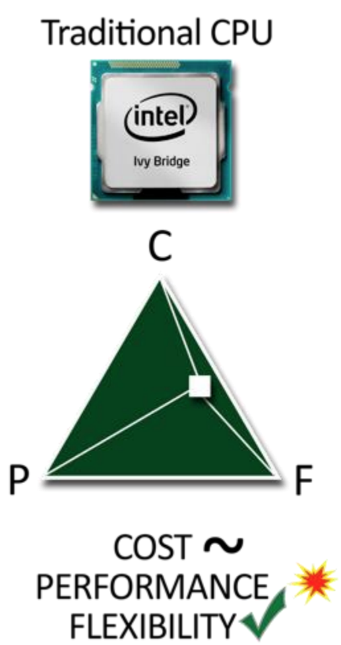


Рис.1: *Здесь не хватает ещё NP и Programmable ASIC*.
Источник

Вендоры всегда балансируют в периметре этого треугольника.

Нельзя сделать CPU, проворачивающий через себя 25 Тб/с.

Нельзя сделать универсальный ASIC - зачастую они могут быть аппаратно ограничены по функциям (например, **или** VXLAN Lookup **или** IP).

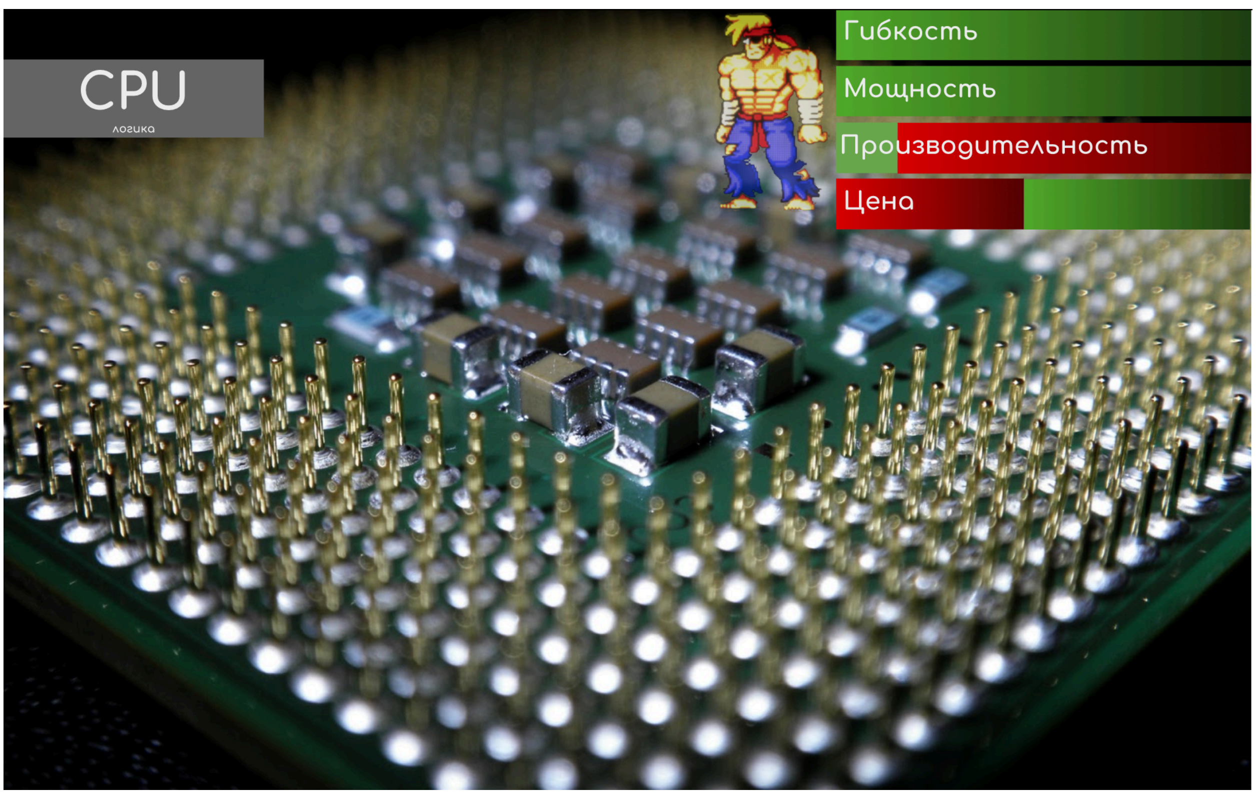
Нельзя сделать дешёвый FPGA.

Кроме того при таком количестве портов, сегодня вступают в игру энергопотребление, место в стойке, тепловыделение и, конечно, цена.

Итак, на сегодняшний день существуют следующие типы чипов, которые могут быть использованы в качестве пакетных процессоров:

Чипы коммутации

CPU — Central Processing Unit



- Неограниченная гибкость
- Неограниченная функциональность
- Приемлемая цена
- Полный провал по производительности

Идея в том, что всё управляется кодом. Инструкции записываются в оперативную память. Для обработки каждого пакета потребуется сходить много раз из CPU в RAM.

С другой стороны для изменения логики работы достаточно переписать программу. Обновления CPU не требуется.

Область применения: домашние и SOHO маршрутизаторы, устройства уровня доступа, файрволы, DPI итд.

Например, абсолютно все рутеры Mikrotik используют CPU для маршрутизации пакетов.

Иными словами CPU годится там, где не гонимся за ультраскоростями, а важен широкий набор функций и невысокая цена.

Note: Впрочем, не без исключений: бывают, что и большие штуки коммутируют в CPU.

Важное замечание: CPU является необходимой частью любого сетевого устройства, потому что берёт на себя задачи Control Plane. А это автоматически означает, что ему придётся работать с протокольным трафиком: OSPF, BGP, LDP, LLDP итд. Кроме того, есть exception трафик - когда у пакета TTL истёк или когда у него стоит бит Router Alert. Ещё CPU нужно самому генерировать трафик тех же протоколов - Self-generated.

Можно ли считать это участием в коммутации? Скорее да, чем нет.

Программная маршрутизация на CPU

Для последнего десятилетия характерна тенденция к маршрутизации в софте. Для всех сетевых функций предлагаются программные альтернативы. Отсюда и DMA, DPDK, VPP, SR-IOV, которые и правда позволяют творить невиданные прежде вещи.

Более того, современные CPU обладают дополнительными блоками инструкций. У Intel это **SSE** - Streaming SIMD Extensions, позволяющие значительно ускорить обработку трафика.

Тут обычные CPU уже заходят в зону NP (Network Processor) - процессоров, которые можно программировать на языках высокого уровня вроде C, и обладающих большим набором спец. инструкций для работы с сетевым трафиком.

Одним из узких мест современных процессоров ещё является шина доступа - PCIe. В один процессор сейчас более-менее можно загнать 100 с небольшим Гбит/с.

Но как бы производители программных решений ни продвигали идею, что "все можно сделать в софте", однако скорости выше 500 Гбит/с пока что можно достигать только с помощью специализированных асиков.

И давайте ещё прикинем. Выдержка с сайта про VPP:

> **Tip:** Recent testing of FD.io release 17.04 shows impressive gains in performance on Intel's newest platform when switching and routing layer 2⁄3 traffic. With the prior generation Intel® Xeon® Processor E7-8890v3, FD.io testing showed aggregate forwarding rate of 480 Gbps (200 Mpps) for 4-Socket machine (using 4 of E7-8890v3 CPU configuration); however, the same FD.io tests run on two 2-Socket blades (e.g. a modern 2RU server) with the new Intel® Xeon® Platinum 8168 CPUs (using four of 8168 CPUs in two by two-socket configuration), within the same power budget, show increase of forwarding rate to 948 Gbps (400 Mpps) benefiting from the PCIe bandwidth increase of the new CPUs, and the overall decrease in cycles-per-packet due to CPU micro-architecture improvements.

Xeon E7-8890v3: Рекомендуемая цена $7174.

4 проца по 18 ядер = 72 ядра = 480 Gbps (200 Mpps)

$28696 только за процы

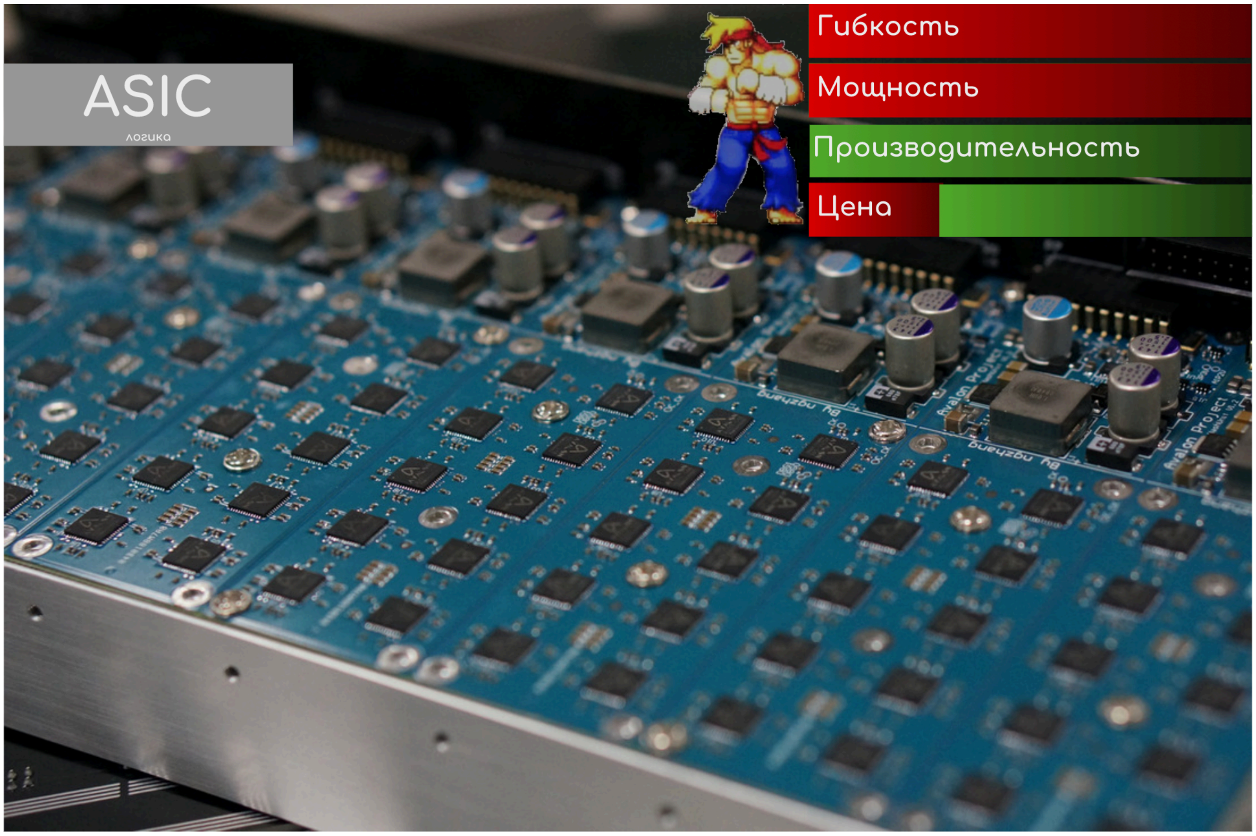
Xeon Platinum 8168. Рекомендуемая цена% $5890.

2 проца по 24 ядра = 48 ядер = 948 Gbps (400 Mpps)

$11780 только за процы

Без обвязки. А ещё кушать электричества он будет как голодный шакал. Не самый дешёвый получится рутер. Зато гибкий.

ASIC — Application Specific Integrated Circuit



- Околонулевая гибкость
- Ограниченная функциональность
- Низкая цена
- Ультравысокая производительность

Идея в том, что инструкции закодированы аппаратно в виде транзисторов.

Сначала очень долго пишется код, реализующий логику, на специальном языке программирования, вроде Verilog, далее он преобразуется в интегральную схему, отлаживается, проверяется и отправляется в тираж. После этого поменять что-то в логике чипа можно, только произведя новый чип.

Каждый пакет обрабатывается, просто прогоняясь по конвейеру из транзисторов, совершающих заранее определённые действия. Это называется Pipeline.

Область применения: почти любые коммутаторы и многие маршрутизаторы.

Note:

Впрочем, не без исключений: Juniper в своей линейке маршрутизаторов MX многие годы использует ASIC Trio.

Который, кстати, согласно книге об MX является на самом деле набором ASIC'ов:

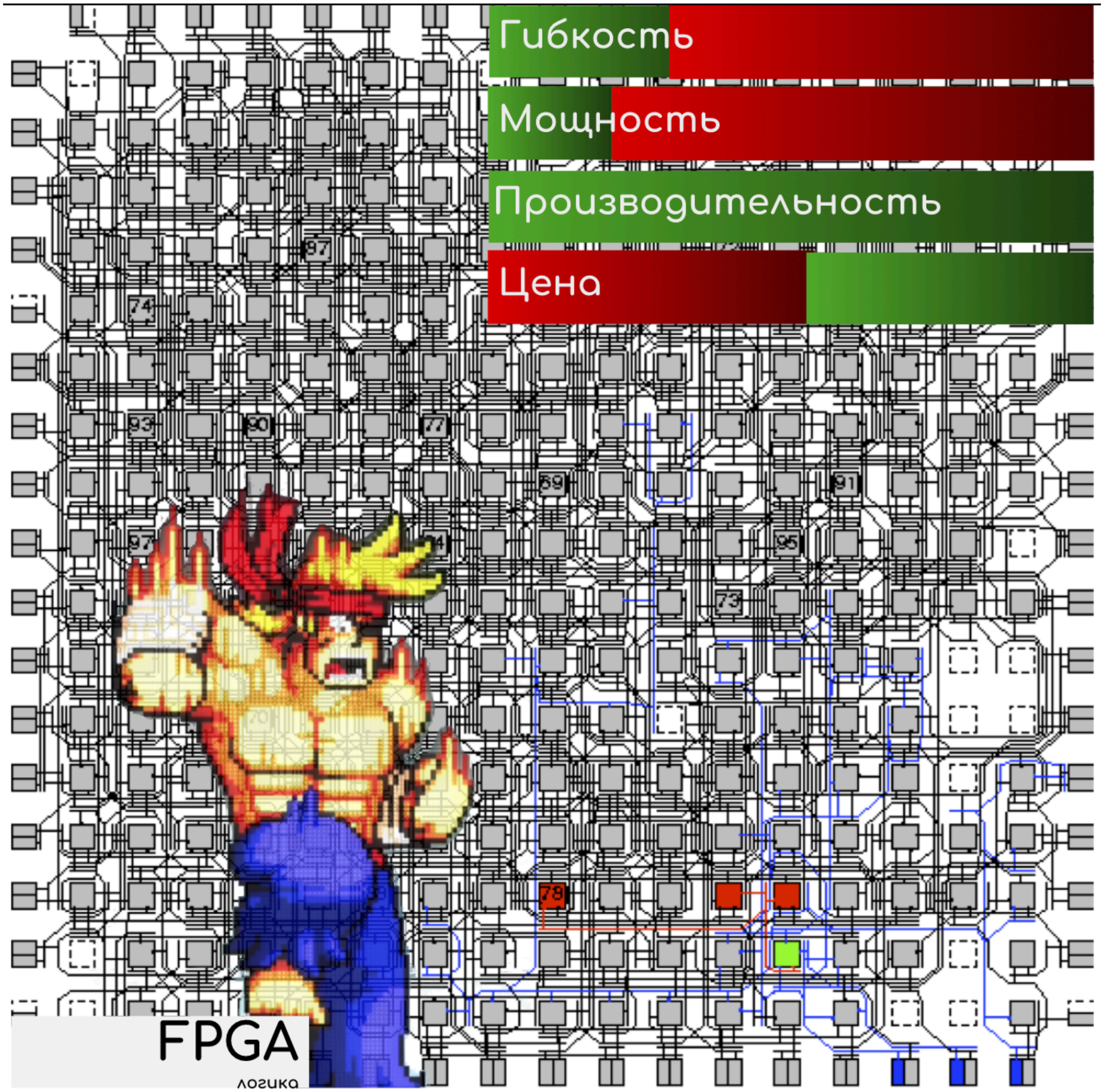
PFEs are made of several ASICs, which may be grouped into four categories:

- Routing ASICs: LU or XL Chips. LU stands for Lookup Unit and XL is a more powerful (X) version.
- Forwarding ASICs: MQ or XM Chips. MQ stands for Memory and Queuing, and XM is a more powerful (X) version.
- Enhanced Class of Service (CoS) ASICs: QX or XQ Chips. Again, XQ is a more powerful version.
- Interface Adaptation ASICs: IX (only on certain low-speed GE MICs) and XF (only on MPC3E)

Речь здесь о классических ASIC - **Applicaton Specific** Integrated Circuit - статических кусках кремния с аппаратной логикой.

Последние лет 10 в области сетевых микросхем произошёл сдвиг в направлении программируемых ASIC'ов, о которых чуть позже.

FPGA — Field Programmable Gate Array



Русский термин - **ПЛИС** - Программируемая Логическая Интегральная Схема.

- Вполне удовлетворительная гибкость
- Вполне удовлетворительная функциональность
- Цена успешного полёта Апполона до Луны и обратно
- Отличная производительность

В отличие от ASIC'ов, где на транзисторах реализованы сами функциональные блоки, в FPGA транзисторами реализуются базовые строительные блоки - регистры, память, LUTы. Из которых потом **можно** создавать нужные функциональные блоки.

Что это даёт?

А то, что FPGA полностью программируемый - логику работы блоков, из которых он состоит, можно поменять. Для этого потребуется обновить прошивку чипа, что менее удобно, чем загрузить программу в CPU, но гораздо удобнее, чем перепаять ASIC.

Так, если поддержка какой-то функции (условный Geneve) не была заложена изначально, её всегда можно добавить потом новой прошивкой.

Однако за такую программируемость приходится дорого платить.

Область применения: POC или низкоскоростные решения для энтерпрайз-сегмента.

Note: Впрочем, не без исключений: собеседовался я как-то раз в контору, в которой модульную коробку для операторов собирали полностью на FPGA, включая фабрику.

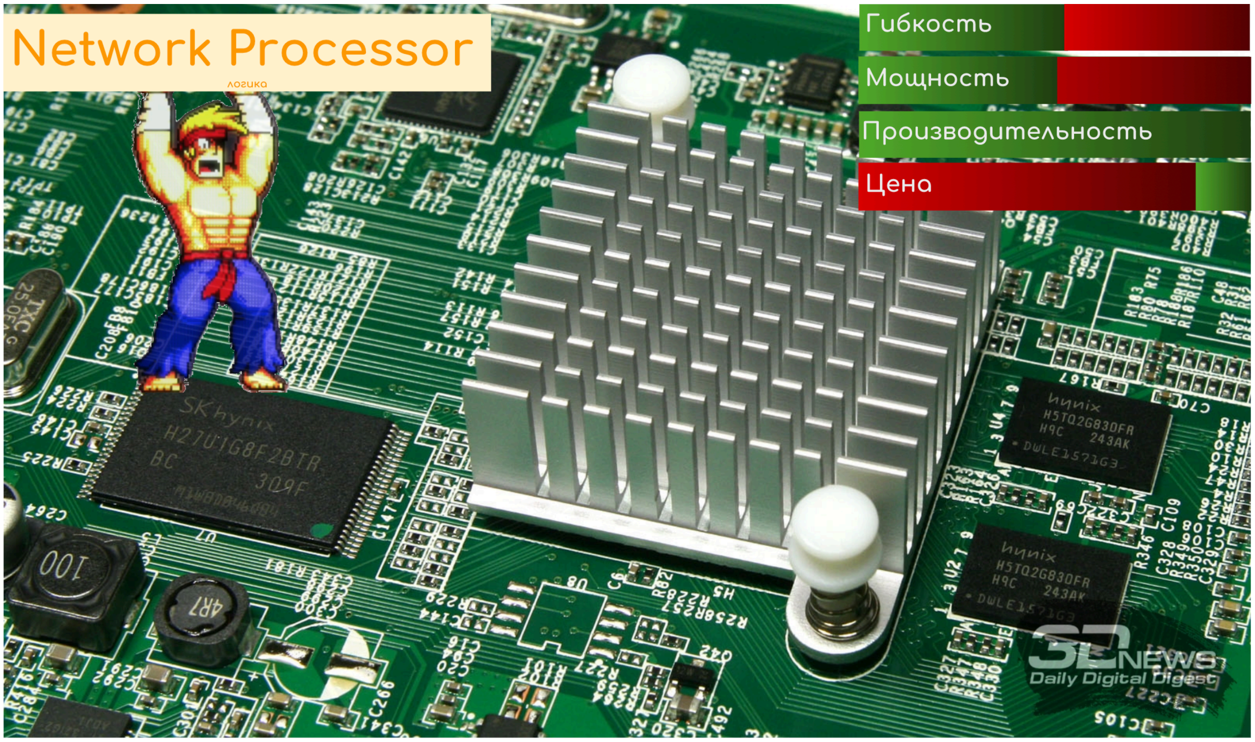
У этого даже есть основания: задолго до появления Programmable ASIC'ов на FPGA можно было делать любую обработку пакетов. И даже через несколько лет после производства плисину легко перепрошить и получить поддержку новой функции.

Автору неизвестны вендоры, которые бы на ПЛИС сделали PFE на скорости более 100 Гбит/с, по всей видимости, потому что частная компания не обладает для этого достаточным капиталом.

Но для рынка энтерпрайз такие решения могут подойти вполне.

Однако, я слышал, что в процессе разработки ASIC возможен такой подход, когда сначала разрабатывается FPGA, программируется нужным образом, тестируется, а потом с неё делают слепок для производства ASIC. Но пруфов нет.

NP - Network Processor



- Отличная гибкость
- Отличная функциональность
- Цена весьма высокая
- Производительность весьма высокая

NP или **NPU** - Network Processor Unit.

Идея в том, что это почти CPU, который однако заточен под сетевые задачи и изготавливается специально под них.

Он, как и CPU, обычно состоит из нескольких ядер, каждое из которых отвечает за свой сегмент. Для изменения логики так же достаточно переписать код приложения.

NP позволяет делать более сложные штуки - например выполнять циклы (чего лишены ASIC и FPGA), делать NAT, почти любые инкапсуляции, пушить и попать условно произвольное число меток итд.

Долгое время NP позиционировался, как серебряная пуля для всех сетевых приложений.

Но производительность уступает ASIC'ам и FPGA.

Большим преимуществом является то, что писать программы для NP можно на C. Это значительно ускоряет процесс, кроме того, где-то можно переиспользовать код.

Область применения: маршрутизаторы агрегации и ядра.

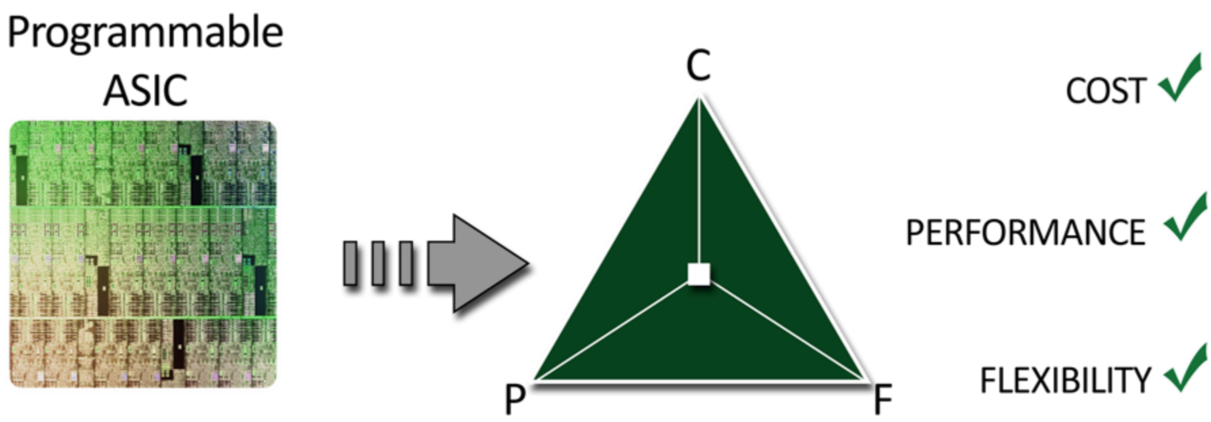
Note: Впрочем, не без исключений: например Smart-NIC Netronome в начале своего пути использовал Intel IXP.

Programmable ASIC

- Приемлемая гибкость
- Ограниченная функциональность
- Низкая цена
- Ультравысокая производительность.

А вот это уже настоящая серебряная пуля последнего десятилетия.

“Почему бы нам не взять ASIC и сделать его немножечко программируемым?” - таким вопросом, видимо, задались разработчики и выдали замечательную вещь, которую циска в своём треугольнике поместила в самую середину, хотя это и не совсем так, потому что производительность программируемого ASIC’а такая же, как и у обычного. Им удалось вырваться из 2D.



Область применения: коммутаторы, маршрутизаторы.

Большинство датацентровых коммутаторов и некоторые маршрутизаторы уровня границы ДЦ работают на программируемых асиках.

Далее взглянем на применение различных чипов коммутации в коммутаторах ДЦ:

Чипы для датацентровых коммутаторов

Чтобы упростить себе жизнь, я продолжу далее разговор только об ASIC'ах под датацентровые коммутаторы, не пытаясь обнять Джабба Хатта.

До недавних пор на этой ниве пахал только Broadcom со своей оружейной палатой: Tomahawk и Trident - и израильскими городами: Qumran, Jericho итд.

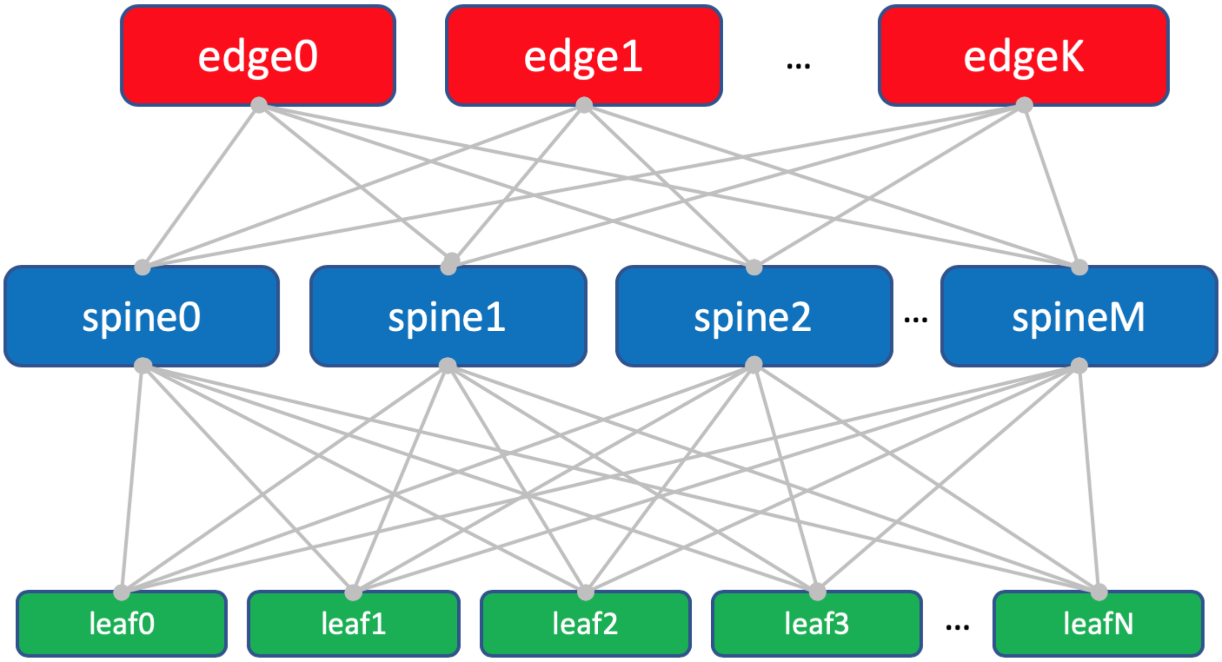
Выбор - особо не разбежишься - ну или разрабатывать своё (как делают Huawei, Juniper и Cisco)

Сегодня конкуренцию ему пытаются составить Mellanox со своими собственными чипами Spectrum (ныне уже Nvidia), Innovium Teralynx, Barefoot Tophino (ныне Intel). Своим появлением эти компании раскачивают рынок и провоцируют среди вендоров тренд на переход от внутренних разработок к готовым чипам их производства.

Мы в конце книги взглянем на их модельные ряды, но пока давайте обсудим, чем же чипы характеризуются и могут отличаться друг от друга.

А для этого надо понять, какие они задачи решают.

В датацентровых сетях есть три основных типа устройств:



Spine сравнительно простая железка, требующая самый минимум функций - её задача просто молотить трафик. Очень много и очень быстро. Зачастую это просто IP-маршрутизация. Но бывают и топологии, в которых Spine играет чуть более важную роль (VXLAN anycast gateway). Но обычная практика - держать конфигурацию спайнов максимально простой.

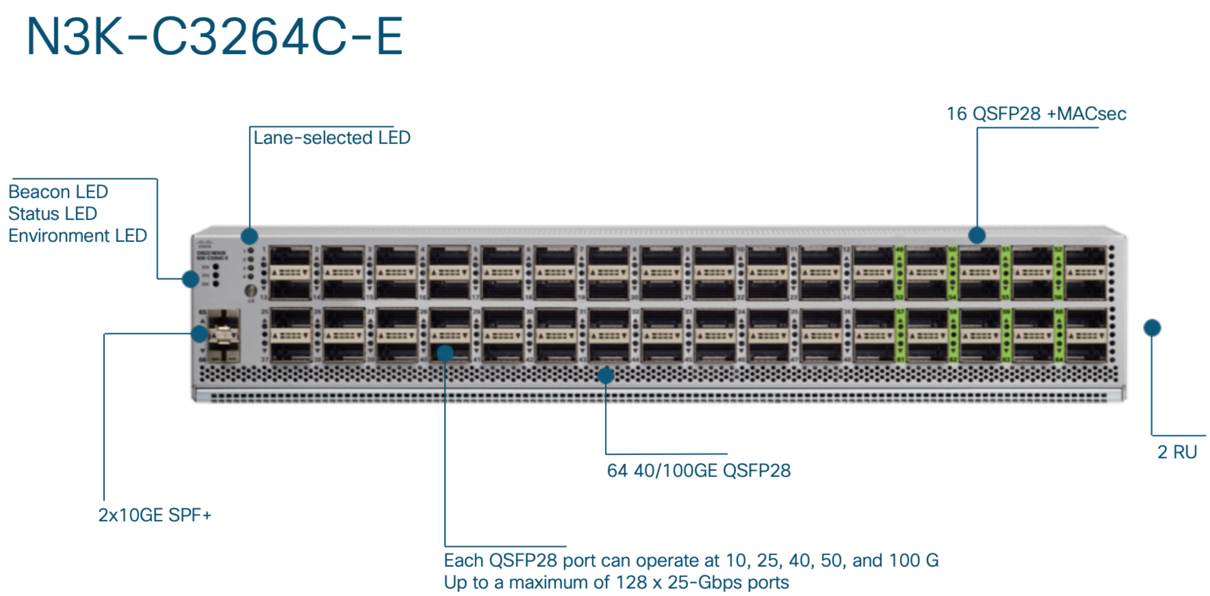
Leaf чуть более требователен к функциям. Может терминировать на себе VXLAN или другие оверлеи. Здесь могут реализовываться политики QoS и ACL. Зато не нужна такая большая пропускная способность, как для спайнов. Кроме того, в некоторых сценариях (VXLAN) leaf знает о сервисах за подключенными машинами (клиентских сетях, контейнерах), соответственно, ему нужно больше ресурсов FIB для хранения этой инфорации.

Edge-leaf это устройства границы сети ДЦ и здесь уже фантазия ограничивается только свободой мысли сетевых архитекторов - MPLS, RSVP-TE, Segment Routing, всевозможные VPN'ы. При этом наименее требовательны к производительности.

На каждом устройстве, соответственно, разные требования к возможностям чипов - как по пропускной способности, так и по набору функций и по количеству ресурсов для хранения чего-либо.

И надо сказать, вендоры чипов и железа добились тут поразительных успехов.

Типичный спайн сегодня - это 64-128 100GE портов на 2-4 юнита с энергопотреблением около 400 Вт. И ценой порядка пары десятков тысяч долларов.



Производителям чипов приходится нелегко не только из-за попыток найти золотую середину, но и из-за возрастающих скоростей передачи данных и конкуренции.

Средняя скорость аплинков с торов сегодня 200-800 Гб/с. Чтобы собрать минимально рабочую сеть ДЦ, нужны спайны с пропускной способностью 3,2 Тб/с.

Всё более и более производительные чипы нужно выпускать уже примерно каждые полтора-два года.

Рис.2: Актуализированная мной картинка из видео PP.

Конкурирующие производители чипов идут ноздря в ноздрю - почти одновременно у всех (Broadcom, Mellanox, Innovium, Barefoot) выходят микросхемы с почти идентичными характеристиками, а вслед за ними и коммутаторы с ними.

Ещё одним компромиссным вопросом является размер буфера, но об этом мы поговорим попозже.

Помимо скорости и обязательных функций по маршрутизации и оверлеям, есть ещё много менее заметных вещей, которые ожидают потребители.

Мы про них говорить сегодня не будем, но не упомянуть было бы ошибкой.

Это, например, **телеметрия** в реальном времени: наблюдать за утилизацией буферов, видеть бёрсты, дампы отброшенных пакетов, профиль трафика по размерам и типам пакетов.

Кроме того, сегодня набирает популярность **INT** - Inband Network Telemetry.

Для многих незаметно, но уже почти жизненно важно, начинает работать **динамическая балансировка трафика**: чип отслеживает потоки (flows) и дробит их на флоулеты (flowlets) - короткие куски трафика одного потока, разделённые между собой паузой в несколько миллисекунд. Эти флоулеты он может динамически распределять по разным путям (ECMP или членам LAG), чтобы обеспечить более равномерную балансировку. Особенно важно это для Elephant Flows, оккупирующих один интерфейс.

Пользователям всё чаще хочется иметь возможность **управлять распределением буфера**, ну а **перераспределение** ресурсов FIB - это уже функциональность, отсутствие которой будет вызывать вопросы.

В условиях датацентров ECMP и балансировка силами сети - это воздух, вендорам нужно обеспечить нужное количество как **ECMP-групп**, так и общее **количество Next-hop'ов**.

Поэтому нет одного чипа или тем более SoC, решающего сразу все задачи.

Под каждую роль разработаны свои чипы. Одни из них ориентированы на пропускную способность, другие на широкую функциональность, третьи на низкие задержки.

Посочувствуем же бедным вендорам и будем выбирать долларом.

И на микросхемы памяти:

Чипы памяти

Отдельно от микросхем логики (и вместе с тем с ними рядом), хочу посмотреть поближе на память.

У микросхем памяти широкое применение в сетевых устройствах.

Они нужны для Control Plane: стоят рядом с CPU, хранят инструкции процессора и просчитанные таблицы маршуртизации, коммутации, меток итд - это обычная RAM-память.

На модульных железках они нужны для этих целей и на управляющих и на линейных картах. А состояние таблиц синхронизируется между ними.

Это не те таблицы, которыми пользуется чип коммутации - они слишком медленные, поэтому отдельная память устанавливается для хранения так называемых Hard Tables - таблиц для Forwardig Plane. Soft Tables из обычной памяти преобразются в Hard Tables.

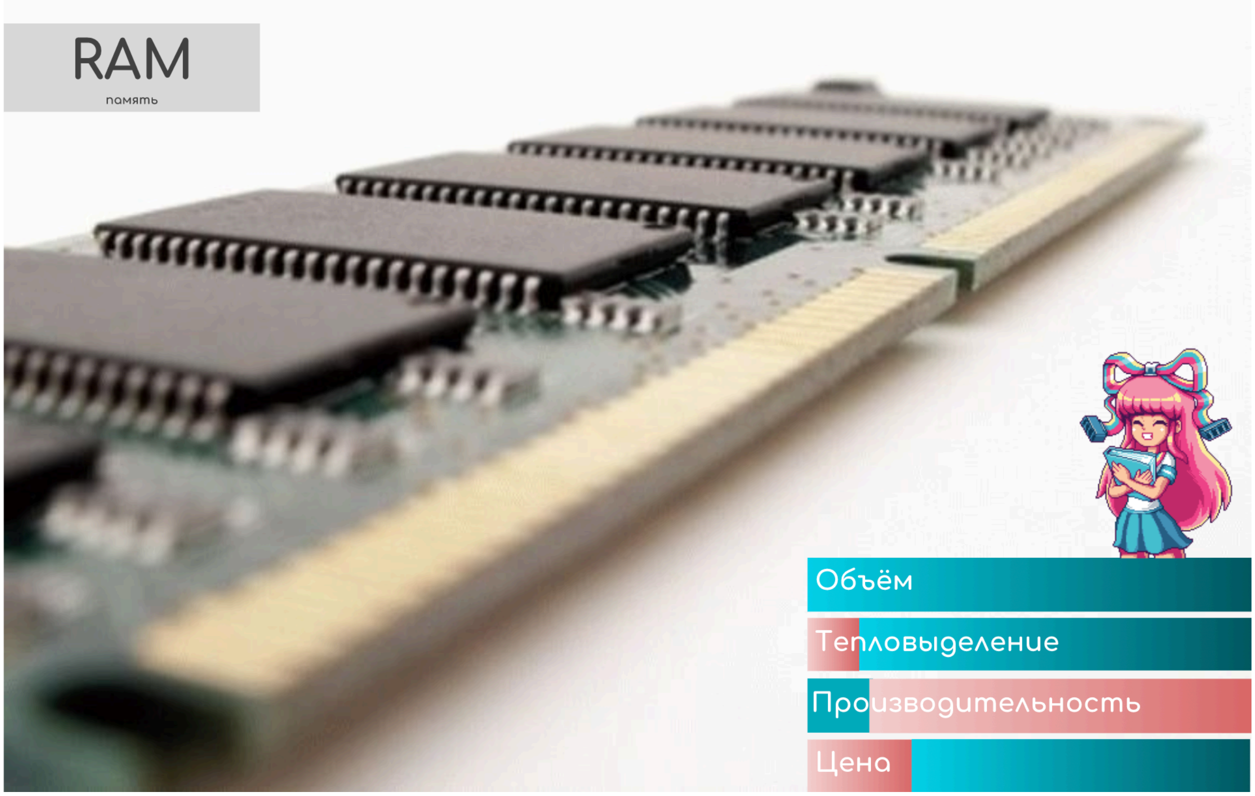
Для быстрого поиска по таблицам (за константное время) были изобретены специальные виды памяти: CAM и TCAM.

Различного рода память нужна для буферизации пакетов на входе и на выходе из устройства, и для их хранения, пока обрабатываются заголовки. Это RAM, который может быть в формате GDDR-5/6, HMC, HBM/2.

Ниже мы взглянем на них чуть подробнее.

RAM - Random Access Memory

Классическая оперативная память - куда без неё?



Мы ей адрес ячейки - она нам содержимое.

В ней хранятся, так называемые **Soft Tables** (программные таблицы) - таблицы маршрутизации, меток, MAC-адресов.

Когда вы выполняете команду *“show ip route”*, запрос идёт именно в оперативку к Soft Tables.

CPU работает именно с оперативной памятью - когда он посчитал маршрут, или построил LSP - результат записывается в неё. А уже оттуда изменения синхронизируются в **Hard Tables** в RAM/CAM/TCAM.

Кроме того, периодически происходит синхронизация всего содержимого всех таблиц на случай, если вдруг по какой-то причине инкрементальные изменения не спустились корректно.

Soft Tables не может быть непосредственно использован для передачи данных, потому что слишком медленно - обращение к оперативке идёт через ЦПУ и требуется алгоритмический поиск, затратный по времени. *С оговоркой на NFV*.

Второй немаловажный компонент сетевого устройства, реализованный на RAM - это очереди: входные, выходные, интерфейсные. Пока пакет ожидает своего черёда на обработку, обрабатывается в чипе или ожидает своей отправки, он томится в RAM.

Эта память может быть на кристалле чипа коммутации, может быть под одной с ним крышкой, а может и просто находиться неподалёку на плате.

Типы этой памяти меняются в зависимости от задач: SDRAM, DDR, GDDR5, GDDR6, HMC, HBM.

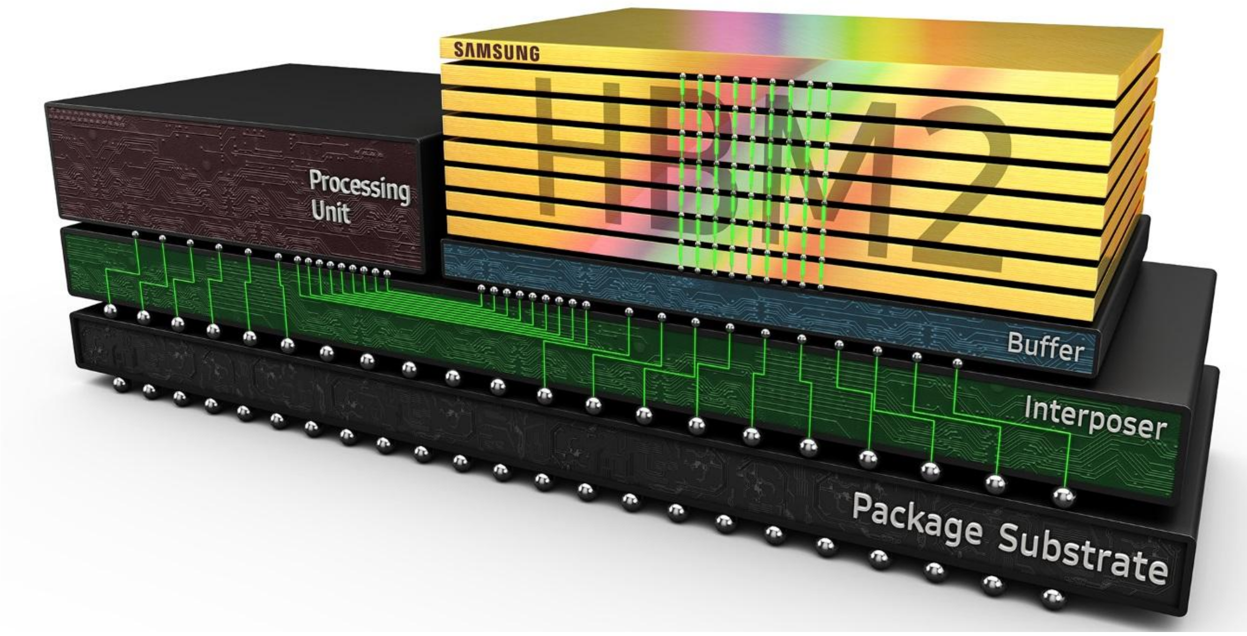


Рис.3: У меня очевидная слабость к 3д-рендерам

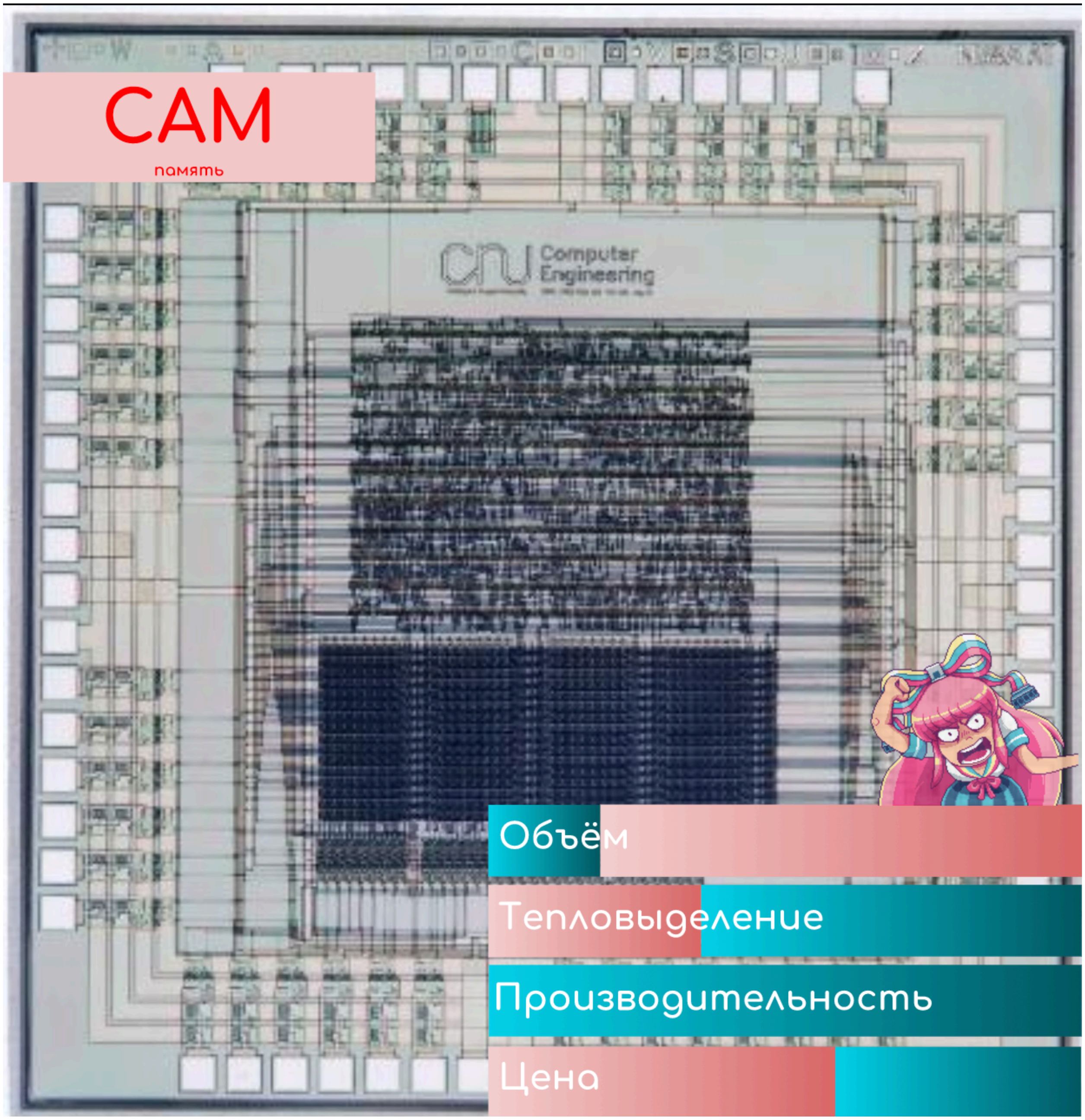
К вопросу о RAM я ещё неоднократно вернусь ниже в секции "Packaging и Advanced Packaging" и вся глава "Память и буферы" посвящена этому.

CAM - Content-Addressable Memory

Это особо-хитрый вид памяти.

Вы ей - значение, а она вам - адрес ячейки.

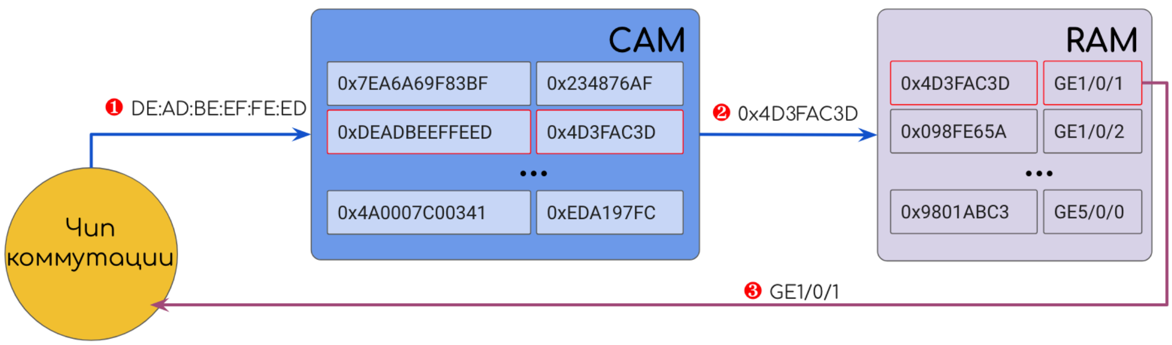
Content-Addressable означает, что адресация базируется на значениях (содержимом).



Значением, например, может быть, например DMAC. CAM прогоняет DMAC по всем своим записям и находит совпадение. В ячейке CAM будет указатель на адрес ячейки в классической RAM, стоящей рядом, где хранится номер выходного интерфейса. Дальше устройство обращается к этой ячейке и отправляет кадр, куда положено.

Для достижения максимальной скорости CAM и RAM располагаются очень близко друг к другу.

Note: Не путать данную RAM с RAM, содержащей Soft Tables, описанной выше - это разные компоненты, расположенные в разных местах.



Прелесть CAM в том, что она возвращает результат за фиксированное время, не зависящее от количества и размера записей в таблице - O(1), выражаясь в терминах сложностей алгоритмов.

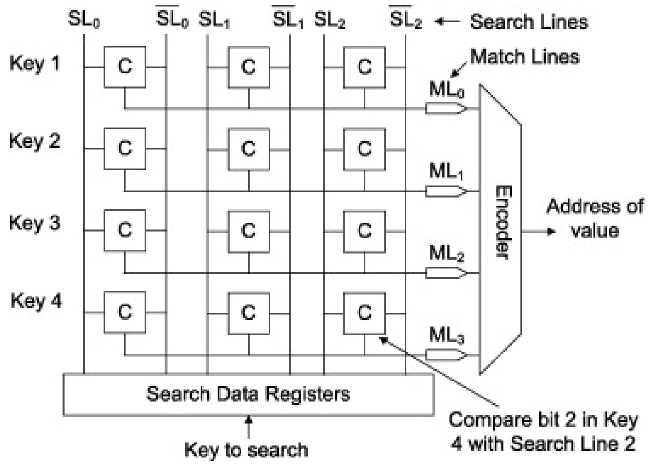
Достигается это за счёт того, что значение сравнивается **одновременно** со всеми записями. Одновременно! А не перебором.

На входе каждой ячейки хранения в CAM стоят сравнивающие элементы (мне очень нравится термин компараторы), которые могут выдавать 0 (разомкнуто) или 1 (замкнуто) в зависимости от того, что на них поступило и что записано.

В сравнивающих элементах записаны как раз искомые значения.

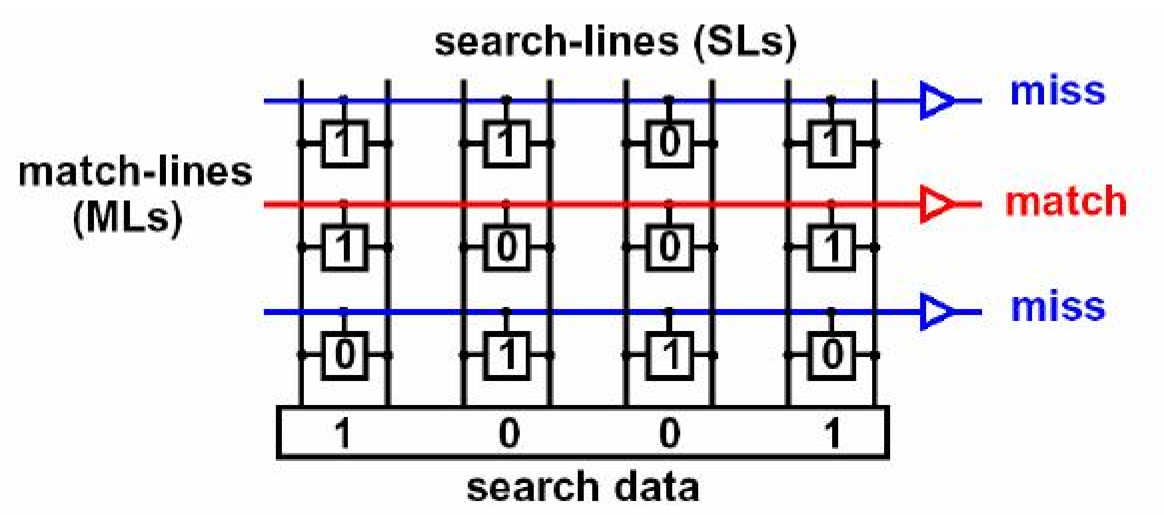
Когда нужно найти запись в таблице, соответствующую определённому значению, это значение прогоняется одновременно через ВСЕ сравнивающие элементы. Буквально, электрический импульс, несущий значения, попадает на все элементы, благодаря тому, что они подключены параллельно. Каждый из них выполняет очень простое действие, выдавая для каждого бита 1, если биты совпали, и 0, если нет, то есть замыкая и размыкая контакт. Таким образом та ячейка, адресом которой является искомое значение, замыкает всю цепь, электрический сигнал проходит и запитывает её.

Вот архитектура такой памяти:



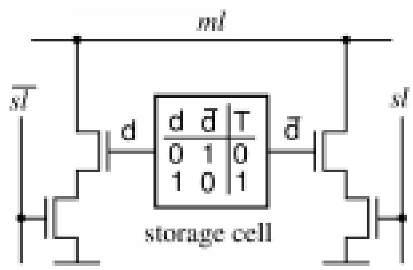
Источник

Вот пример работы



Источник

А это схема реализации:



Источник

Это чем-то похоже на пару ключ-замок. Только ключ с правильной геометрией может поставить штифты замка в правильные положения и провернуть цилиндр. Вот только у нас много копий одного ключа и много разных конфигураций замков. И мы вставляем их все одновременно и пытаемся провернуть, а нужное значение лежит за той дверью, замок которой ключ откроет.

Для гибкого использования CAM мы берём не непосредственно значения из полей заголовков, а вычисляем их хэш. Хэш-функция используется для следующих целей:

1. Длина результата значительно меньше, чем у входных значений. Так пространство MAC-адресов длиной 48 бит можно отобразить в 16-ибитовое значение, тем самым в 2^32 раза уменьшив длину значений, которые нужно сравнивать, и соответственно, размер CAM. Основная идея хэш-функции в том, что результат её выполнения для одинаковых входных данных всегда будет одинаков (например, как остаток от деления одного числа на другое - это пример элементарной хэш функции).

2. Результат её выполнения на всём пространстве входных значений - это ± плоскость - все значения равновероятны. Это важно для снижения вероятности конфликта хэшей, когда два значения дают одинаковый результат. Конфликт хэшей, кстати, весьма любопытная проблема, которая описана в парадоксе дней рождения. Рекомендую почитать Hardware Defined Networking Брайна Петерсена, где помимо всего прочего он описывает механизмы избежания конфликта хэшей.

3. Независимо от длины исходных аргументов, результат будет всегда одной длины. То есть на вход можно подать сложное сочетание аргументов, например, DMAC+EtherType, и для хранения не потребуется выделять более сложную структуру памяти.

Именно хэш закодирован в сравнивающие элементы. Именно хэш искомого значения будет сравниваться с ними. По принципу CAM схож с хэш-таблицами в программировании, только реализованными на чипах.

В этот принцип отлично укладывается также MPLS-коммутация, почему MPLS и сватали в своё время на IP.

Например:

1. Пришёл самый первый Ethernet-кадр на порт коммутатора.

2. Коммутатор извлёк SMAC, вычислил его хэш.

3. Данный хэш он записал в сравнивающие элементы CAM, номер интерфейса откуда пришёл кадр в RAM, а в саму ячейку CAM адрес ячейки в RAM.

4. Выполнил рассылку изначального кадра во все порты.

5. Повторил пп. 1-5 ….

6. Заполнена вся таблица MAC-адресов.

7. Приходит Ethernet-кадр. Коммутатор сначала проверяет, известен ли ему данный SMAC (сравнивает хэш адреса с записанными хэшами в CAM) и, если нет, сохраняет.

8. Извлекает DMAC, считает его хэш.
9. Данный хэш он прогоняет через все сравнивающие элементы CAM и находит единственное совпадение.
10. Узнаёт номер порта, отправляет туда изначальный кадр.

Резюме:

- Ячейки CAM адресуются хэшами.
- Ячейки CAM содержат (обычно) адрес ячейки в обычной памяти (RAM), потому что хранить конечную информацию в CAM - дорого.
- Каждая ячейка CAM имеет на входе сравнивающий элемент, который сравнивает искомое значение с хэш-адресом. От этого размер и стоимость CAM значительно больше, чем RAM.
- Проверка совпадения происходит **одновременно** во всех записях, отчего CAM дюже греется, зато выдаёт результат за константное время.
- CAM+RAM хранят **Hard Tables** (аппаратные таблицы), к которым обращается чип коммутации.

TCAM - Ternary Content-Addressable Memory

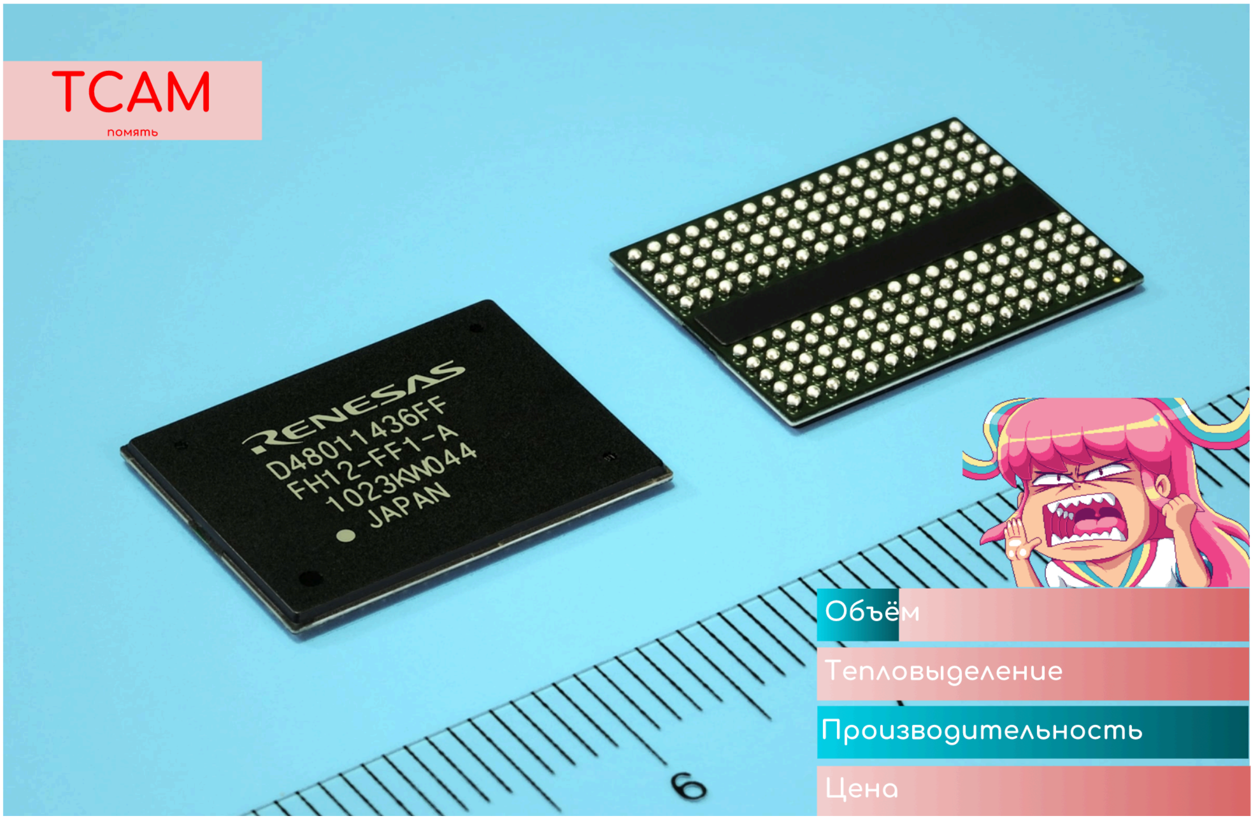
CAM весьма неплохо подходит для поиска MAC-адресов, где результат всегда заведомо один. А что не так с IP

Если мы возьмём описанный выше CAM, то на любой Destinataion IP он очень редко сможет вернуть 1 во всех битах.

Дело в том, что DIP - это всегда один единственный адрес, а маршруты в таблице маршрутизации - это подсеть или даже агрегация более мелких маршрутов. Поэтому полного совпадения быть почти не может - кроме случая, когда есть маршрут /32.

Перед разработчиками чипов стояло два вопроса:

- Как это в принципе реализовать?
- Как из нескольких подходящих маршрутов выбрать лучший (с длиннейшей маской)?



Ответом стал TCAM, в котором **"T"** означает "троичный"". Помимо **0** и **1** вводится ещё одно значение **X3** - "не важно" (CAM иногда называют BCAM - Binary, поскольку там значения два - 0 и 1).

Тогда результатом поиска нужной записи в таблице коммутации будет содержимое той ячейки, где самая длинная цепочка 1 и самая короткая "не важно".

Например, пакет адресован на DIP 10.10.10.10.

В Таблице Маршрутизации у нас следующие маршруты:

```
0.0.0.0/0
10.10.10.8/29
10.10.0.0/16
10.8.0.0/13
Другие
```

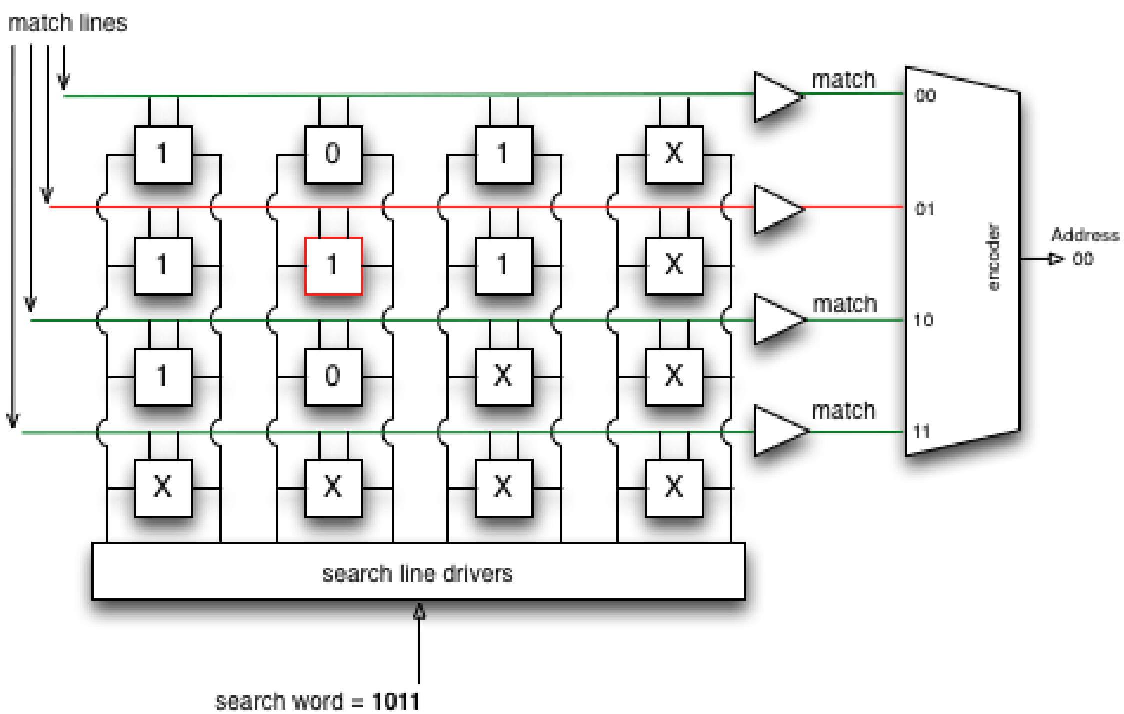
В сравнивающие элементы TCAM записываются биты маршрута, если в маске стоит 1, и "не важно", если 0.

При поиске нужной записи TCAM, как и CAM, прогоняет искомое значение одновременно по всем ячейкам. Результатом будет последовательность 0, 1 и "не важно".

Только те записи, которые вернули последовательность единиц, за которыми следуют "не важно" участвуют в следующем этапе селекции.

Далее из всех результатов выбирается тот, где самая длинная последовательность единиц - так реализуется правило Longest prefix match.

Очевидно, что мы-то своим зорким взглядом, сразу увидели, что это будет маршрут 10.10.10.8/29.



Источник

Решение на грани гениальности, за которое пришлось заплатить большую цену. Из-за очень высокой плотности транзисторов (у каждой ячейки их свой набор, а ячеек должны быть миллионы) они греются не меньше любого CPU - нужно решать вопрос отвода тепла.

Кроме того, их производство стоит очень дорого, и не будет лукавством сказать, что стоимость сетевого оборудования раньше зачастую определялась наличием и объёмом TCAM.

Внимательный читатель обратил внимание на вопрос хэш-функций - ведь она преобразует изначальный аргумент во что-то совершенно непохожее на исходник, как же мы будем сравнивать 0, 1 и длины? Ответ: хэш функция здесь не используется. Описанный выше алгоритм - это сильное упрощения реальной процедуры, за деталями этого любознательного читателя отправлю к той же книге Hardware Defined Networking.

Алгоритмический поиск

Прогресс, как говорится, не стоит на месте. Появляются новые алгоритмы, совершенствуются старые. Чипы становятся всё более производительными иногда даже более дешёвыми. TCAM - вещь элегантная, но экономически не очень выгодная. И как только первый вендор реализовал алгоритмический поиск и стал продавать своё оборудование дешевле конкурентов с TCAM, все начали делать так же.

Суть этого подхода в том, что вместо прожорливого до денег и электричества TCAM ставится отдельный ASIC, реализующий алгоритмический лукап IP-адреса (или чего-то ещё) и тот же RAM рядом.

TCAM всё ещё применяется в сетевом оборудовании для узкоспециализированных задач.

Подробнее послушать о алгоритмах этого класса и деревьях можно в подкасте с Кодом Безопасности.

5

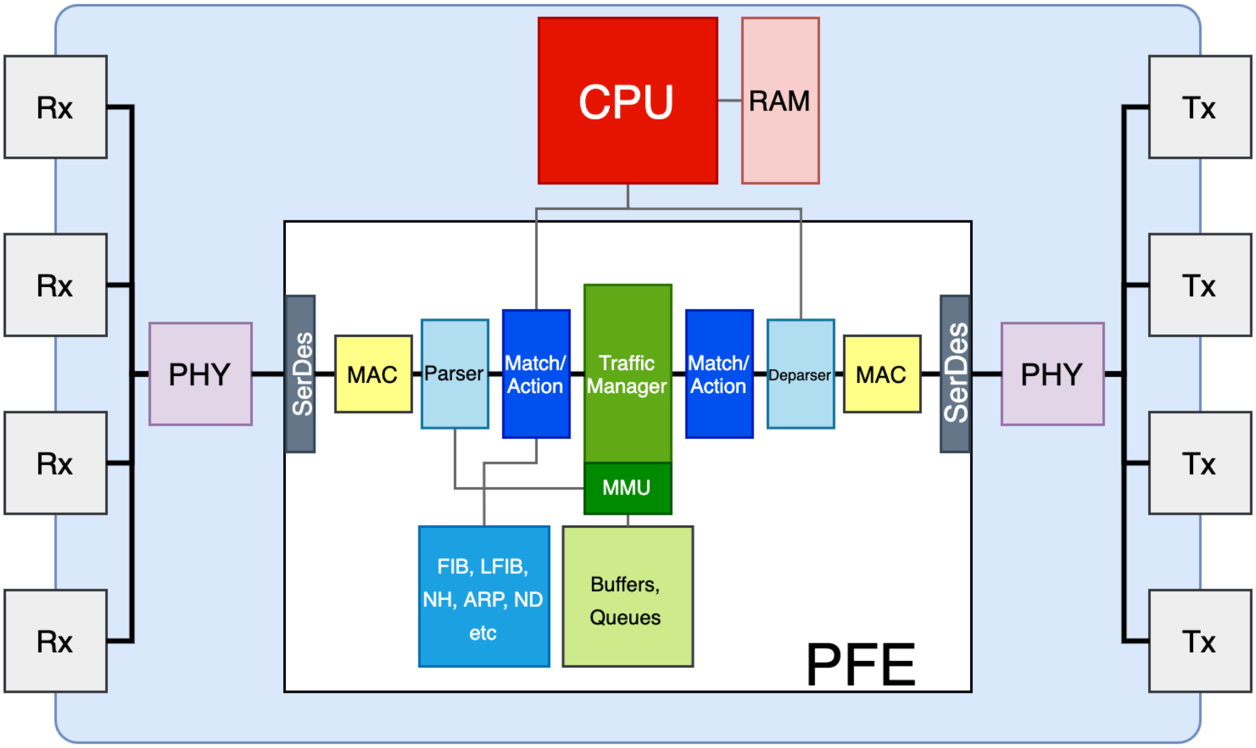
4. Архитектура сетевых ASIC

Сначала мы взглянем на физиологию чипа - из каких компонентов он состоит. А далее разберёмся что с пакетом в этих компонентах происходит.

Физическое устройство

Итак, для успешной коммутации пакета нужны следующие блоки:

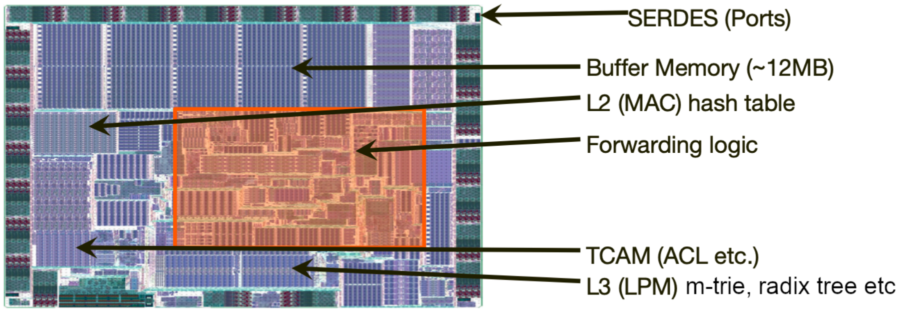
- Парсер заголовков (Parser)
- Лукап (Match): FIB/LFIB, Nexthop-группы, ARP Adjacencies, IPv6 ND Tables, ACL итд
- Блоки преобразований (Action)
- Блок управления памятью (TM/MMU)
- Сборщик пакета (Deparser)
- SerDes
- Память для буферизации пакетов
- Блок, реализующий MAC
- Чип PHY
- Физические порты/трансиверы



Крупными мазками: оптический или электрический сигнал попадает на порт (**Rx**), тот его передаёт на **PHY**. Модуль PHY реализует функции физического уровня и передаёт биты на входные пины PFE, где сигнал блоками **SerDes** преобразуется в удобоваримый для чипа вид. Блок **MAC** из потока битов восстанавливает Ethernet-кадр и передаёт его **парсеру**. Парсер отделяет необходимые ему заголовки и передаёт их на анализ следующему блоку **Match/Action**. Тот их исследует и применяет нужные действия - отправить на правильный порт, на CPU, энкапсулировать, дропнуть итд. Тело пакета всё это время хранилось в **буферах**, управляемых **MMU**, и теперь пришло время **Traffic Manager'у** проводить все обряды QoS. И потом процесс раскручивается в обратную сторону. Снова **Match/Action**. Потом собрать пакет с новыми заголовками (**Deparser**), преобразовать кадр в поток битов (**MAC**), сериализовать (**SerDes**), осуществить действия физического уровня (**PHY**) и передать через выходной порт (**Tx**) в среду.

В простейшем случае вообще почти все блоки являются частью одного монолитного кристалла кремния. То есть они - продукт одного процесса печати на вафле.

Отдельные, составляющие чип компоненты, реализующие законченный набор функций, называются IP-core (не тот, что ты мог подумать, сетевой инженер!). То есть SerDes, MAC, TM - это всё отдельные IP-core. Зачастую они производятся сторонними компаниями, специализирующимися конкретно на данных компонентах, а потом встраиваются в микросхему. Особенно это касается SerDes - сложнейшей детали, в которую вендоры сетевых чипов не готовы вкладывать силы R&D. Один из крупных производителей SerDes - Inphi.


SERDES (Ports)
Buffer Memory (~12MB)
L2 (MAC) hash table
Forwarding logic
TCAM (ACL etc.)
L3 (LPM) m-trie, radix tree etc

Источник

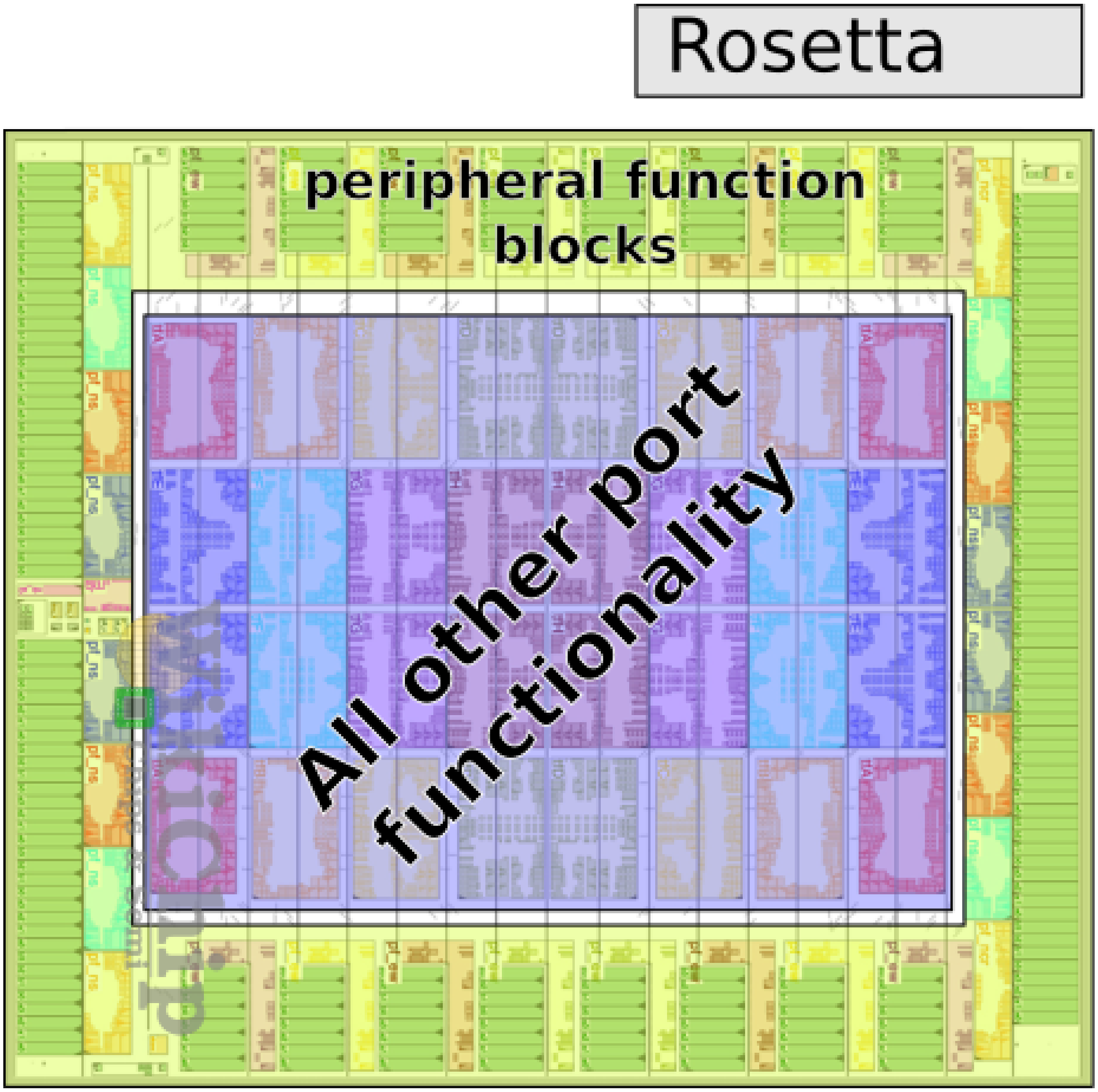
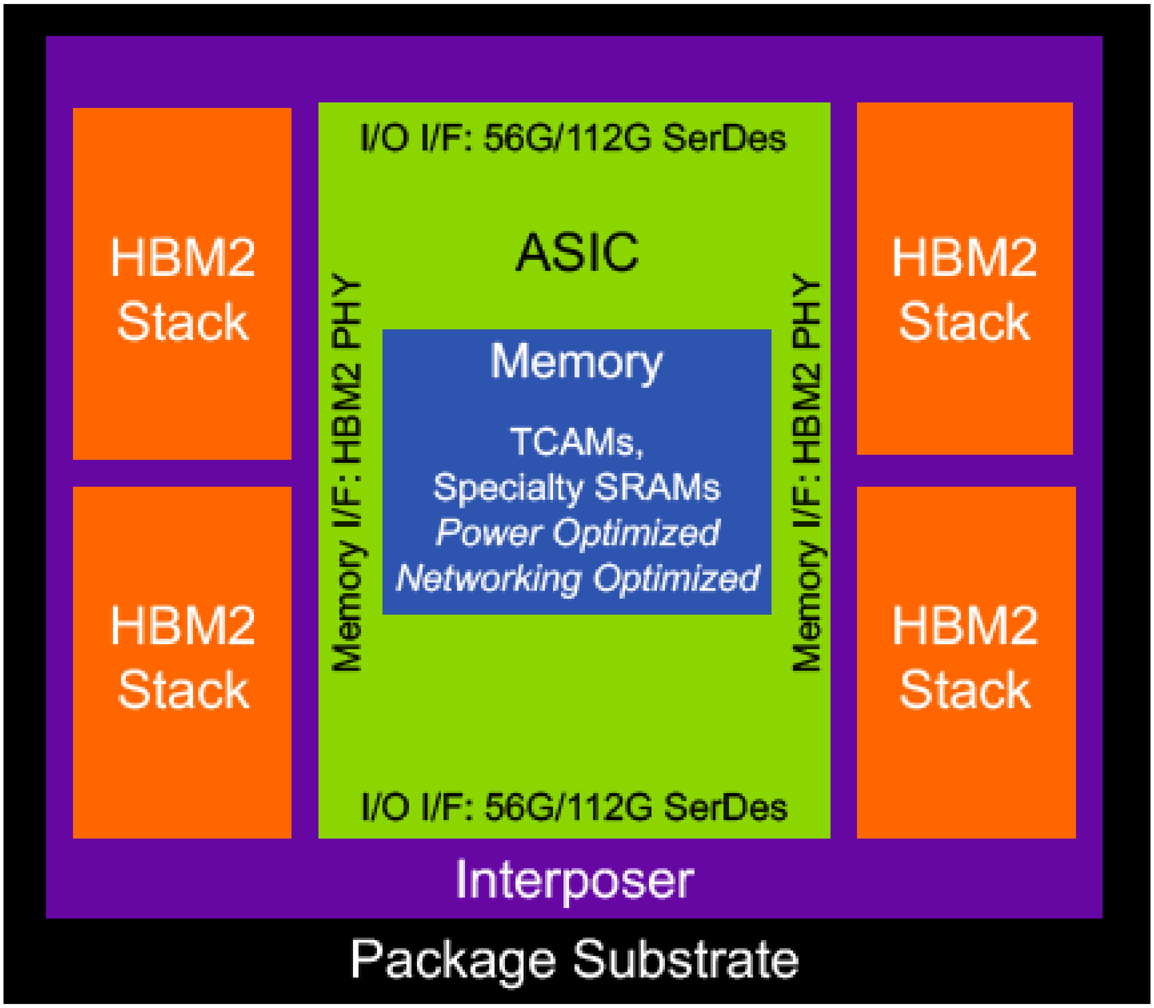
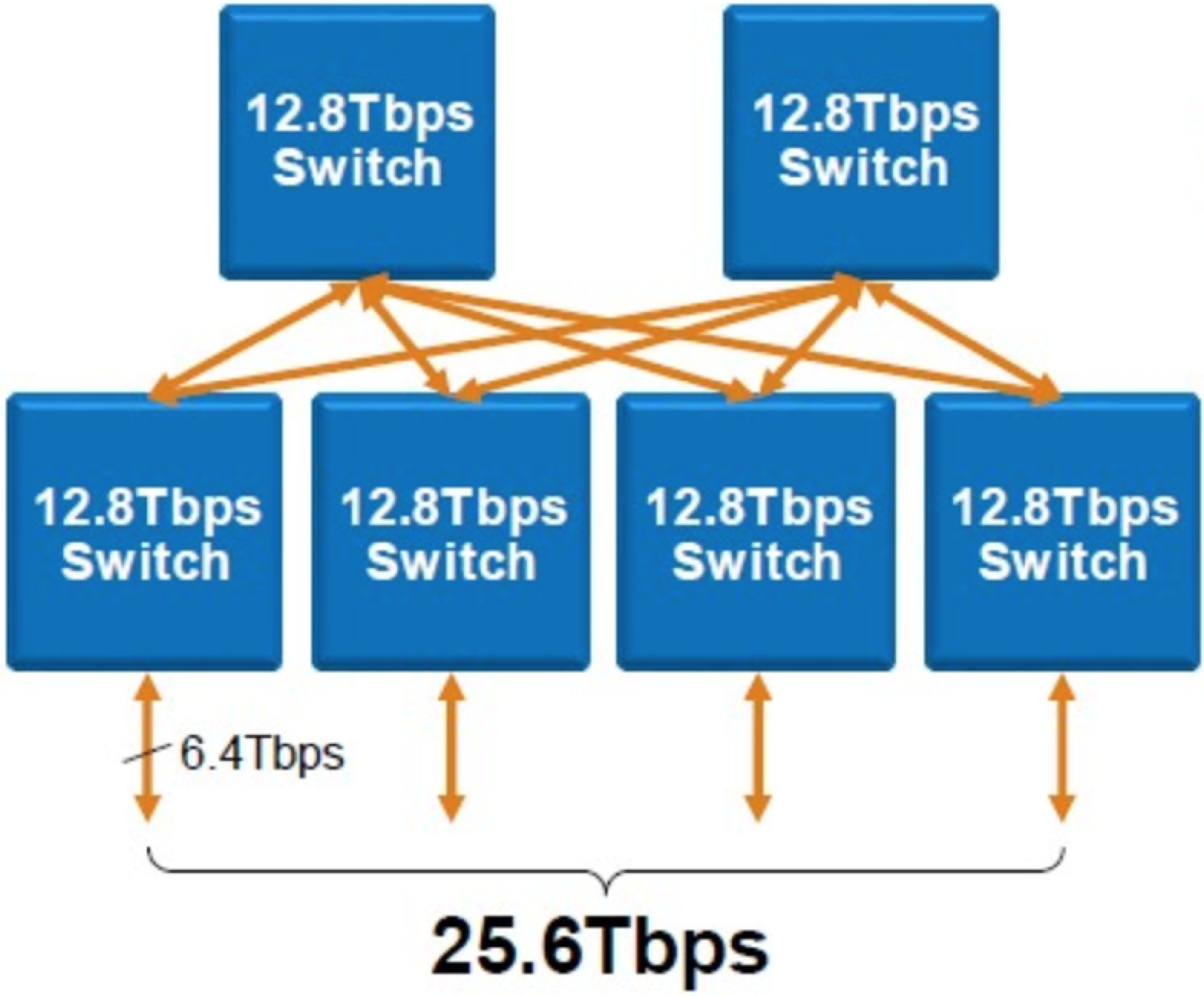


Рис.1: Монолитный чип Rosetta.
Источник

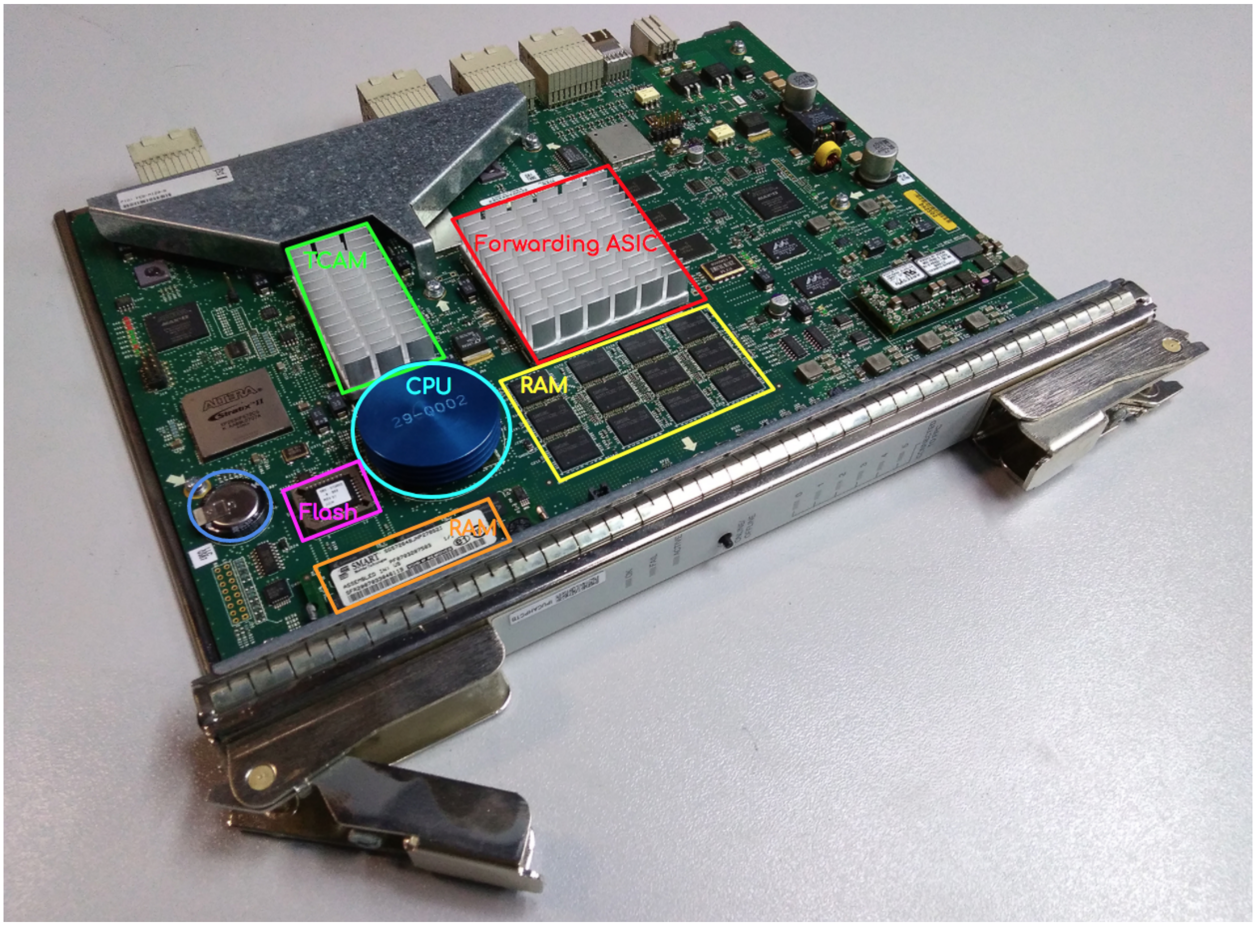
Другой распространённый вариант: в одном чипе сочетать несколько разных кристаллов с интерконнектом между ними. Так, например, память HBM коммерческого производства выносится за пределы кристалла сетевого ASIC:



Под крышкой одного производительного чипа могут быть собраны несколько, так называемых, менее производительных чиплетов (chiplet), которые, объединённые в фабрику, дают бо́льшую пропускную способность:

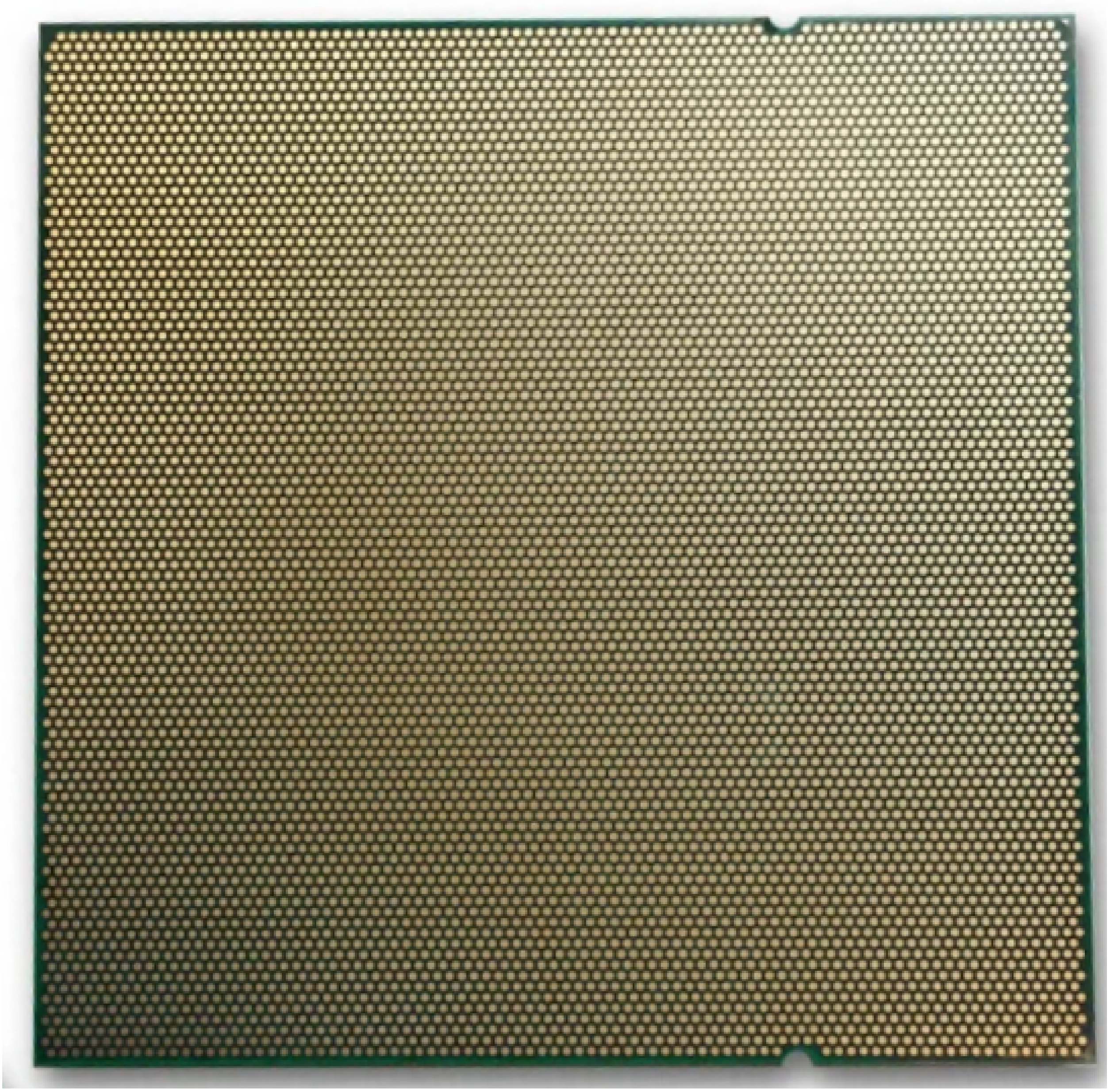


Для некоторых решений рядовая практика - вообще все ресурсы выносить за пределы сетевого ASIC'а:



В случае Juniper, кстати, их Trio - это не один ASIC - это их набор, каждый из которого реализует свои функции.

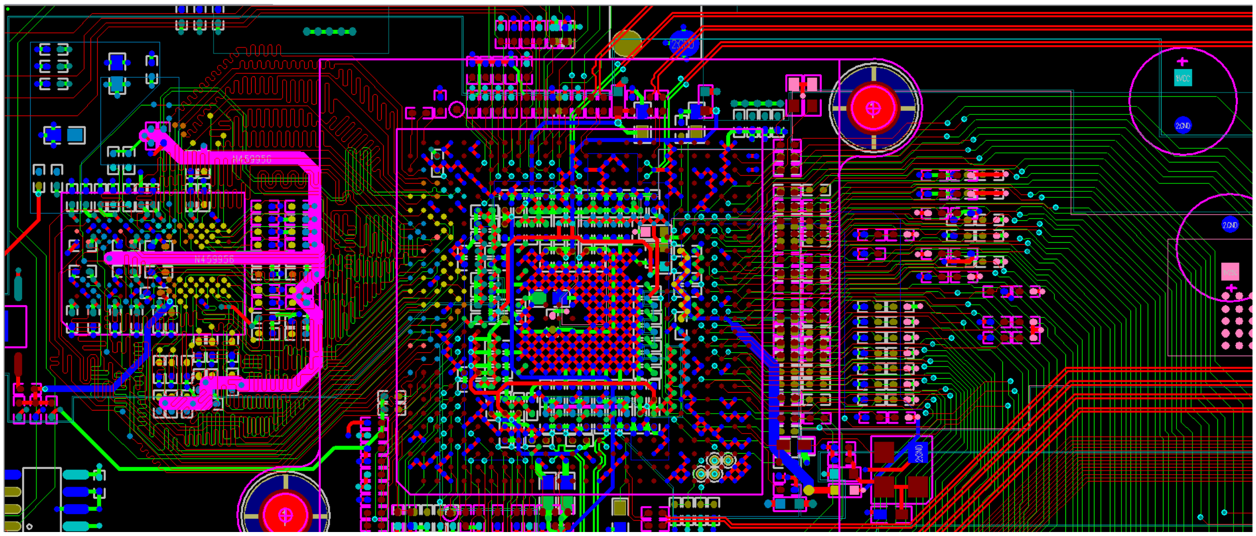
Но как бы ни был устроен сам чип, ему нужно общаться с миром. И поэтому на животике у него есть несколько тысяч пинов:



Одни пины нужны для того, чтобы подключить к чипу интерфейсы. Другие - чтобы подключиться к внешней памяти (CAM/TCAM/RAM), если она есть. Третьи - к фабрике коммутации, если коробка модульная.

Два пина образуют дифференциальную пару для передачи данных в одном направлении. То есть две пары пинов нужны для полнодуплексной передачи.

Вот так оно потом выглядит в программах для проектирования (для случая на порядок более простой микросхемы):



Источник

Теперь пришло время разобраться с тем, что же такое загадочный SerDes. Нет, это не ножки на микросхеме.

SerDes

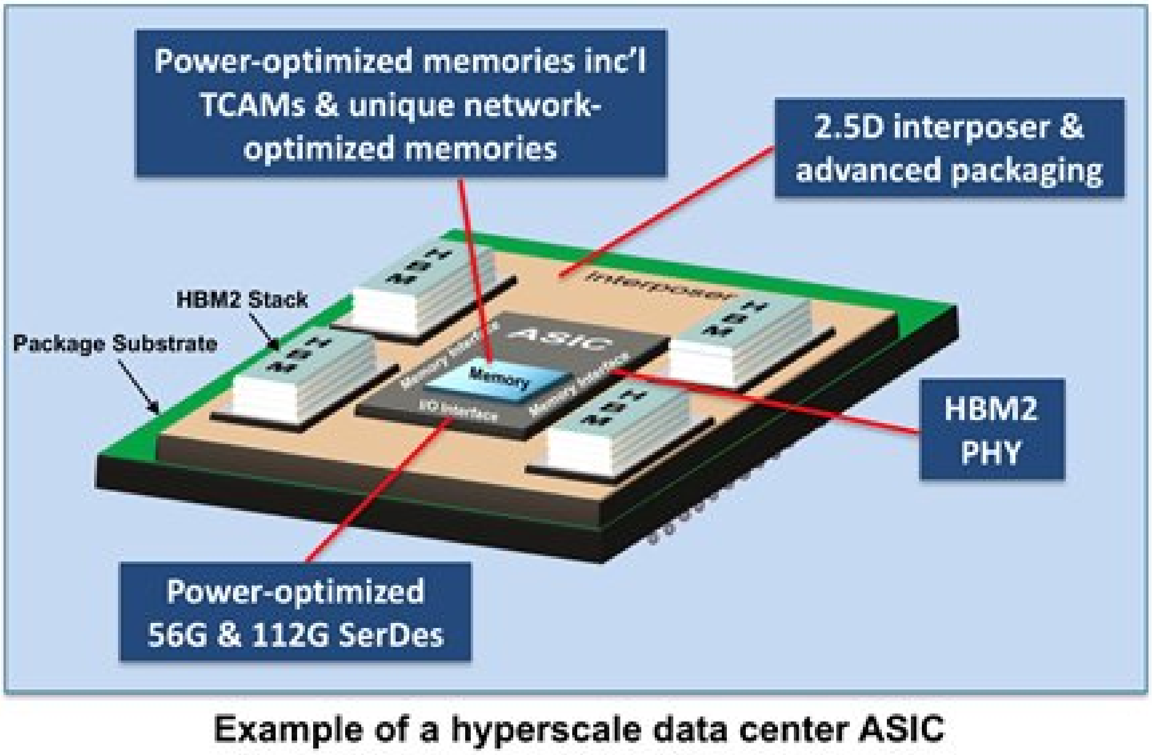
Если коротко - то это блоки (IP-core) сетевого ASIC'а, которые позволяют получить сигнал с пинов и, наоборот, передать его туда.

Теперь с этимологией. Аналогично "модему" и "кодеку", ставшим уже такими родными в кириллическом написании, SerDes составлен из двух слов: **Serializer-Deserializer**. Так чего же он сериализует и десериализует?

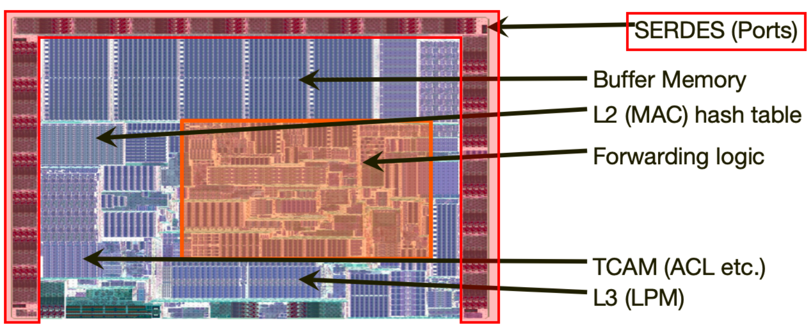
Всё дело в скорости работы сетевых чипов. Независимо от их типа (ASIC, NP, CPU) - их частота находится в пределах сотен Мгц - единиц Ггц. А частота передачи данных с порта 10Гб/с - 10 Ггц (100 Гбит/с = 4x25ГГц). Соответственно, каким-то образом нужно понизить частоту внутри чипа. И как раз это достигается тем, что сигнал из пары вводных пинов распараллеливается на множество внутренних линий - десериализуется.

В обратную сторону - сигнал с нескольких линий нужно сериализовать в пару пинов.

Блоки SerDes всегда являются составными частями кристалла сетевого чипа.



Источник



Источник

Один SerDes - это 4 пина на пузике асика - два для Tx, два для Rx.

Скорость одного SerDes'а - величина скачкообразно растущая с годами . Наиболее распространённые сегодня - это 10Гб/с и 28Гб/с. Но в скором будущем датацентровый масс-маркет начнут заполонять устройства с 56Гб/с SerDes и даже со 112Гб/с.

Выглядит довольно сложным. Для чего же вообще устраивать эту сериальную вакханалию, а не сделать просто пинов по числу реальных линий в чипе?

Зачем?

Ну, давайте прикинем, спайн-коммутатор с 64x100Гб/с портами несёт под кожухом ASIC ёмкостью 6,4 Тб/с.

Если каждые 4 пина чипа могут обеспечить 28Гб/с в полном дуплексе, значит, на нём должно быть 64*4*4=1024 пинов.

Это уже как минимум 32x32 пинов на 40 см^2, не считая питания и земли. И они там сидят довольно плотненько. Легко ли будет кратно нарастить их количество?

Однако проблема не только и не столько с количеством пинов.

Тенденция к серийным интерфейсам намечается уже давно.

На смену переферийным параллельным портам пришли серийные (USB на смену параллельному)

На смену IDE - SATA (Serial ATA)

На смену SCSI - SAS (Serial SCSI)

На смену PCI - PCI Express

Это жжж неспроста. Ведь казалось бы, чем больше проводов, тем больше данных можно передать за единицу времени?

На самом деле нет - при параллельной передаче с повышением скорости всё острее и острее встают вопросы синхронизации между этими самым проводами. Схемотехники даже связанные дорожки на платах проектируют так, чтобы они были максимально одинаковой длины.

В какой-то момент задача усложнилась настолько, что дальнейшее развитие пошло по увеличению полосы пропускания обычной дифф-пары вместо параллелизма. В частности для 100Гб/с другого варианта просто не существует.

Модуляция

То, каким образом множество параллельных сигналов укладывается в один, зависит от метода модуляции.

Для SerDes с пропускной сопособностью 10Гб/с и 28Гб/с используется **NRZ** - Non-Return-to-Zero.

С ним же добились и 56Гб/с. Но всё же стандартом *de facto* для 56Гб/с и *de iure* для 112Гб/с является **PAM4** - 4 -level Pulse Amplitude Modulation.

Есть и ленивые пессимистичные взгляды в сторону PAM8.

Итого, учитывая современные реалии (NRZ), для того, чтобы запитать данными интерфейс 100Гб/с нужно подвести к нему 4 SerDes'а по 28Гб/с (или 16 дорожек). Отсюда и берётся "лейновость" 100Ж-портов: 4 лейна - это 4 канала по 28Гб/с.

И это то, что позволяет 1x100Ж порт разбить на 4x25Ж с помощью гидры.

В случае PAM4 для 100Ж нужно только 2 SerDes'а по 56Гб/с, то есть два лейна.

GearBox'ы

Сложности с переходом на новые методы модуляции заключаются в том, что устройства на разных сторонах должны использовать одинаковые, либо нужно ставить дополнительные конвертеры. То есть просто подключить сотками коммутатор с PAM4 к NRZ не получится.

Но когда индустрия бросала своих участников, не предлагая им решений? Для того, чтобы устройства с разными модуляциями могли взаимодействовать друг с другом, изобрели коробки передач, которые из малого количество высокоскоростных линий делают много помедленнее и наоборот.

Так, в новейшие коммутаторы, выпускаемые сегодня, ставят дополнительные чипы, чтобы их можно было использовать в сети с более старым оборудованием.

Использование гирбоксов также упростит вендорам и переход на 400G - не придётся менять ASIC - достаточно заменить/убрать гирбокс.

Ну а потом от вещей мирских перейдём к тому, сколько кругов пакет проходит в чипе.

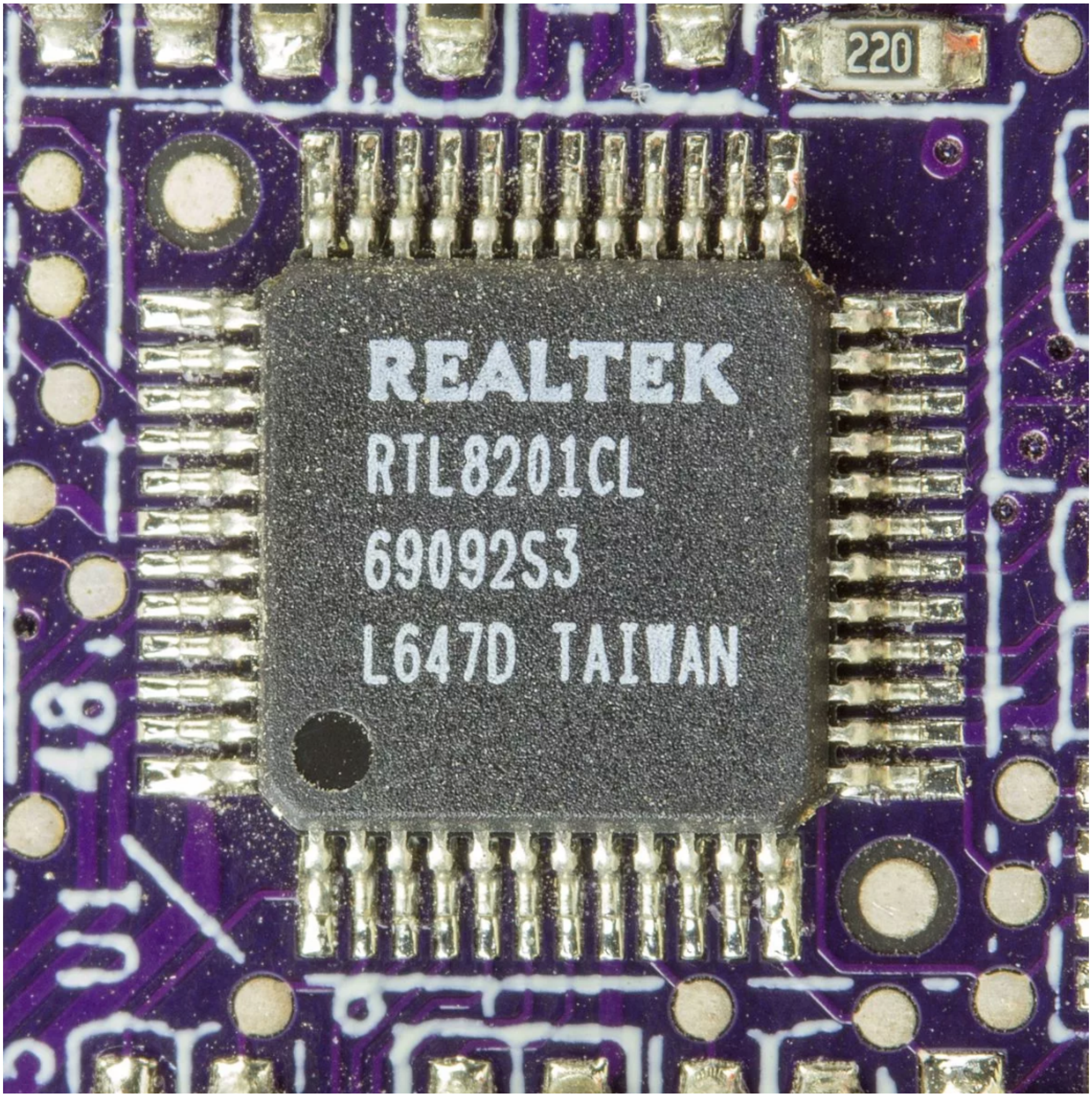
PHY

Этот зверь тоже по-своему интересен.
Его задачи незамысловаты:

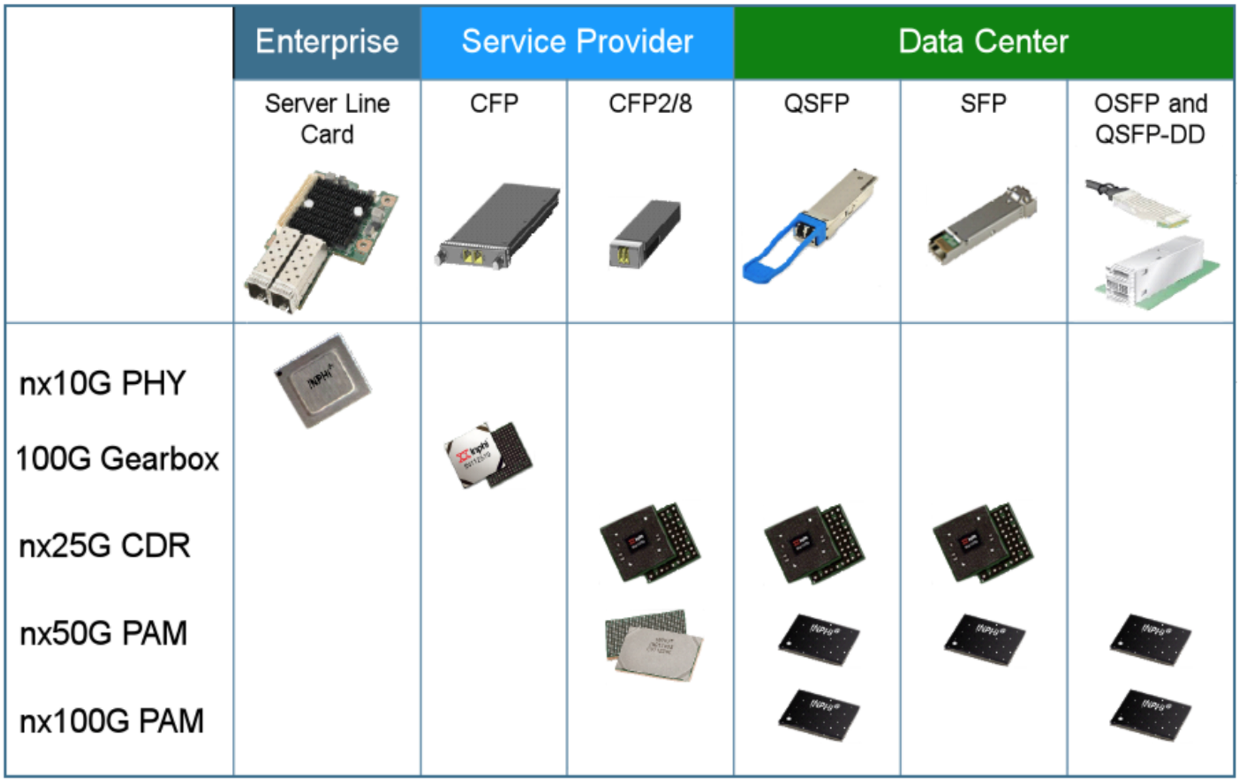
- Конвертация сигнала между средами (оптика-медь), если это нужно
- Восстановление битов из сигналов и наоборот
- Коррекция ошибок
- Синхронизация
- И другие задачи физического уровня.

Если хочется знать больше, и не пугают забористые тексты со страшными картинками: PHY Interface for PCI Express, SATA, USB 3.1, DisplayPort, and Converged IO Architectures.

Что действительно любопытно и достойно обсуждения - так это его расположение. Если порт медный - RJ45, то чип PHY - это ASIC, установленный на плате.



Если порт оптический, то в подавляющем большинстве случаев эти функции возьмёт на себя DSP PHY, встроенный в трансивер (та самая штука, называемая нами модулем и вставляемая в дырку в коммутаторе).



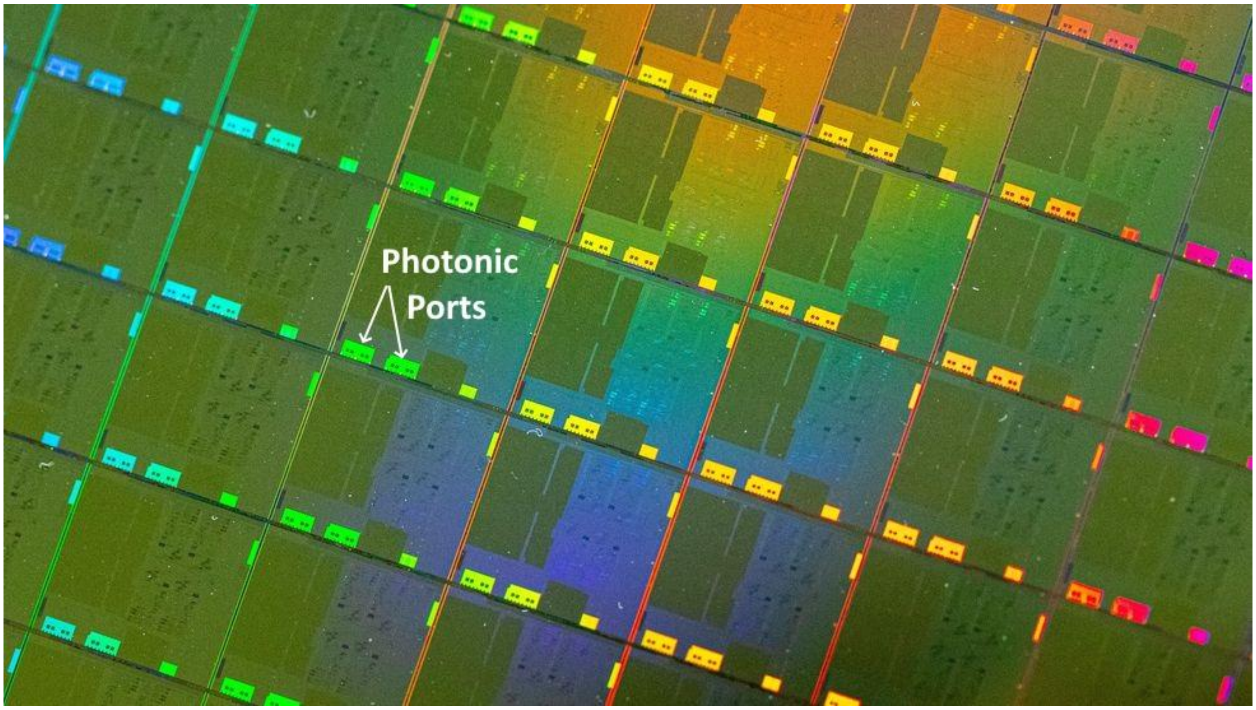
Источник

Однако тенденции последних лет - это **Silicon Photonics**.

Самый производительный коммерческий чип сегодня выдаёт 25,6 Тб/с. Это серьёзнейший инженерный вызов разработчикам. И нет никаких оснований полагать, что гиперскейлеры и экзаскейлеры умерят свои аппетиты и решат остановиться на этом. Скорость будет расти.

Чип PHY находится на трансиверах, а SerDes - на кристалле сетевой микросхемы. Сигнал между ними идёт по электропроводящей среде - по металлической дорожке. С увеличением скоростей растёт и конструктивная сложность и потребляемое электричество. Рано или поздно (скорее, рано) мы во что-нибудь упрёмся.

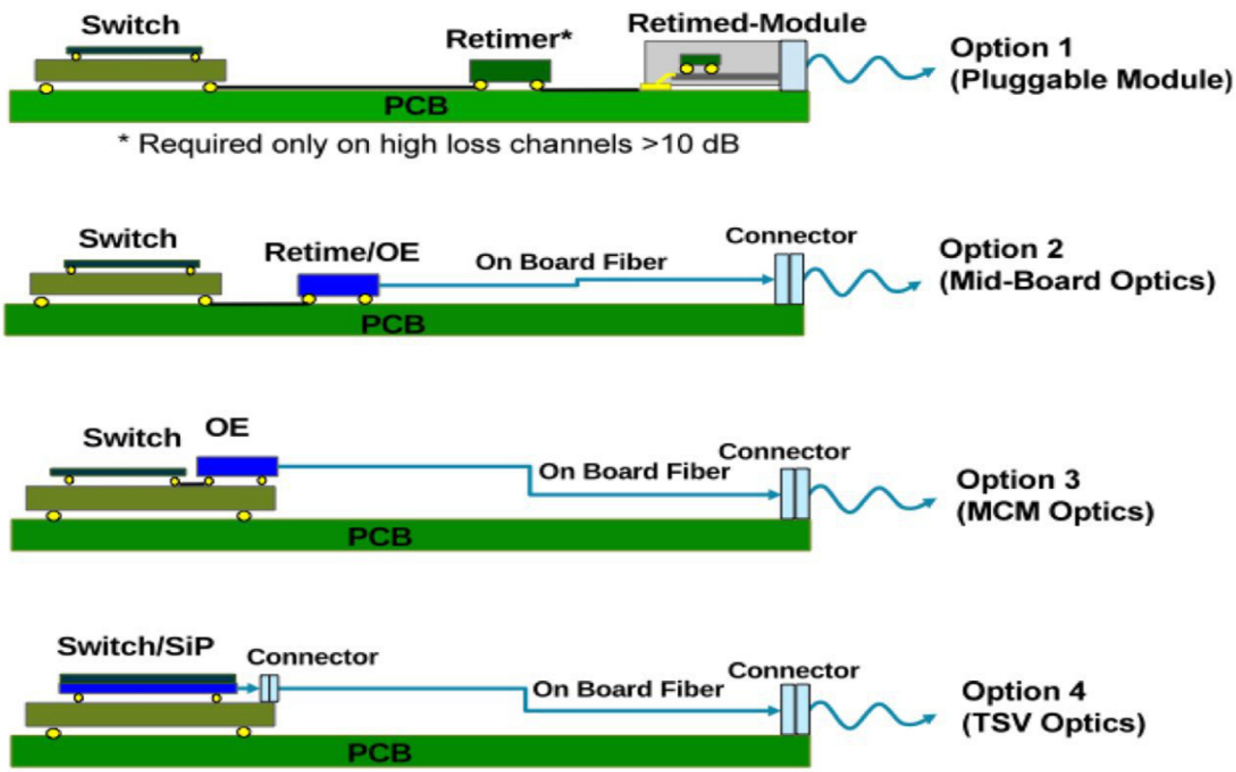
В случае silicon photonics микросхема PHY переносится внутрь самого чипа коммутации. В кристалл "встраиваются" фотонные порты, позволяющие осуществлять коммуникации между чипами на скорости света через оптическую среду.



Источник

Идея не новая, и только ожидавшая своего времени, а именно, когда технологии достигнут нужного уровня зрелости. Проблема была в том, что материалы и процессы, используемые для производства фотонных чипов, были фундаментально несовместимы с процессом производства кремниевых чипов - CMOS.

Из возможных альтернативных решений: установка на плату отдельного чипа, преобразующего электрический сигнал в оптический, или его установка внутрь сетевого чипа, но не на сам кристалл (всё ещё требует конвертации среды).

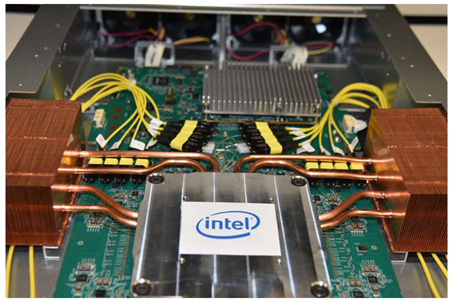


Но эта технологическая плотина смыта упорными разработками в этом направлении, и в скором будущем микроэлектронику ждут большие изменения.

Весьма занимательная статья с историей вопроса и сегодняшними реалиями.

Note: Кстати, был у нас в гостях Compass EOS, которые разработали co-packaged optics ещё до того, как это стало модным. Но, увы, они настолько опередили своё время, что оказались в тот момент никому не нужны. И постигла их ужасающе печальная судьба быть купленным не то Ростелекомом, не то Роснано. Впрочем, возможно, история звучала совсем иначе :).

Note: И прямо во время написания этой статьи 5-го марта 2x20 Intel опубликовал в своём блоге новость о том, что они продемонстрировали первый свитч, в котором им удалось интегрировать свой интеловский silicon photonics в барефутовский Tofino2.



Источник

Напоследок хочется немного времени посвятить прям совсем кишочкам, скрытым под металлической крышкой.

Packaging и Advanced Packaging

Packaging (корпусирование) - это процесс упаковки кристалла в корпус, который защитит его от физических повреждений и коррозии.

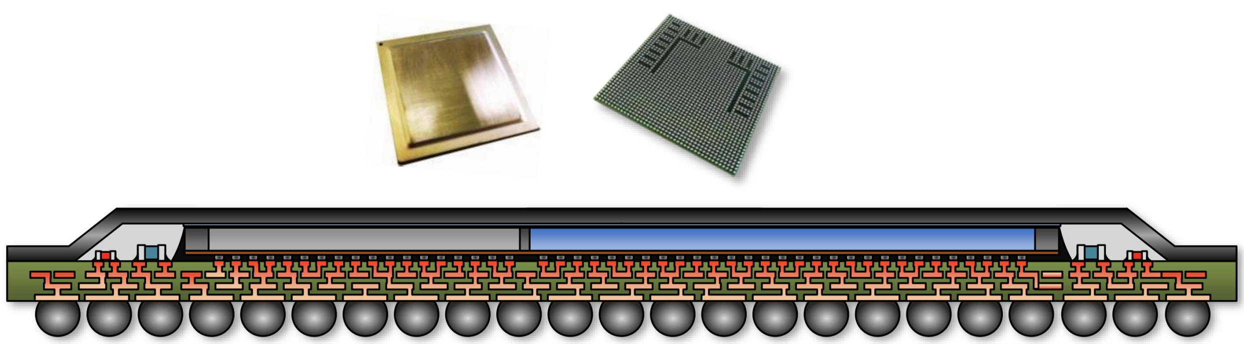


Рис.2: Корпусированный набор чипов.
Источник

Я честно предпринял попытку разобраться в этом вопросе, но уже на втором часе чтения стало понятно, что, если хоть немного глубже общих понятий погружаться, то это будет требовать ещё одной статьи такого же размера, чтобы не прослыть дилетантом.

Когда-то давно, внутри одной микросхемы на плате был один кристалл. Даже не так - когда-то давно в микросхеме было несколько полупроводниковых элементов.

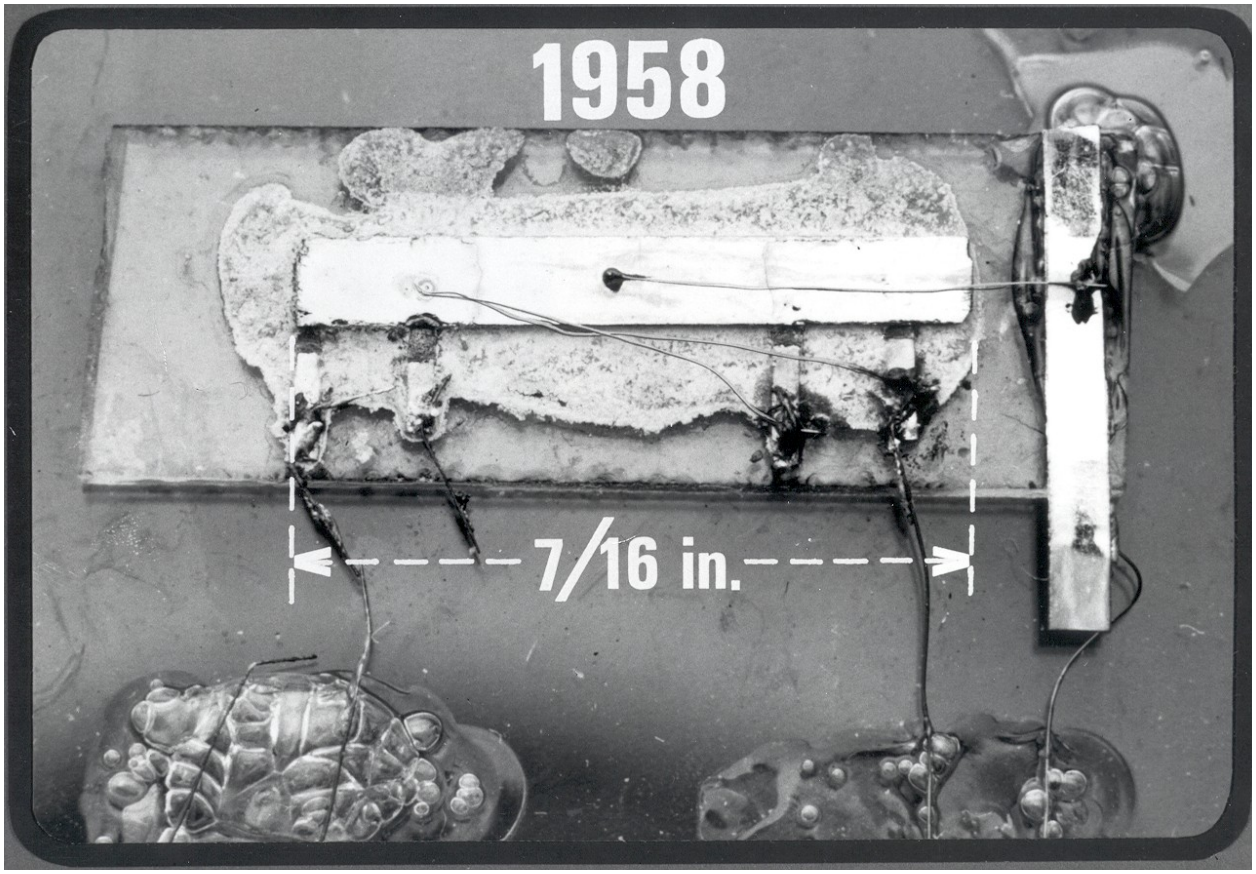
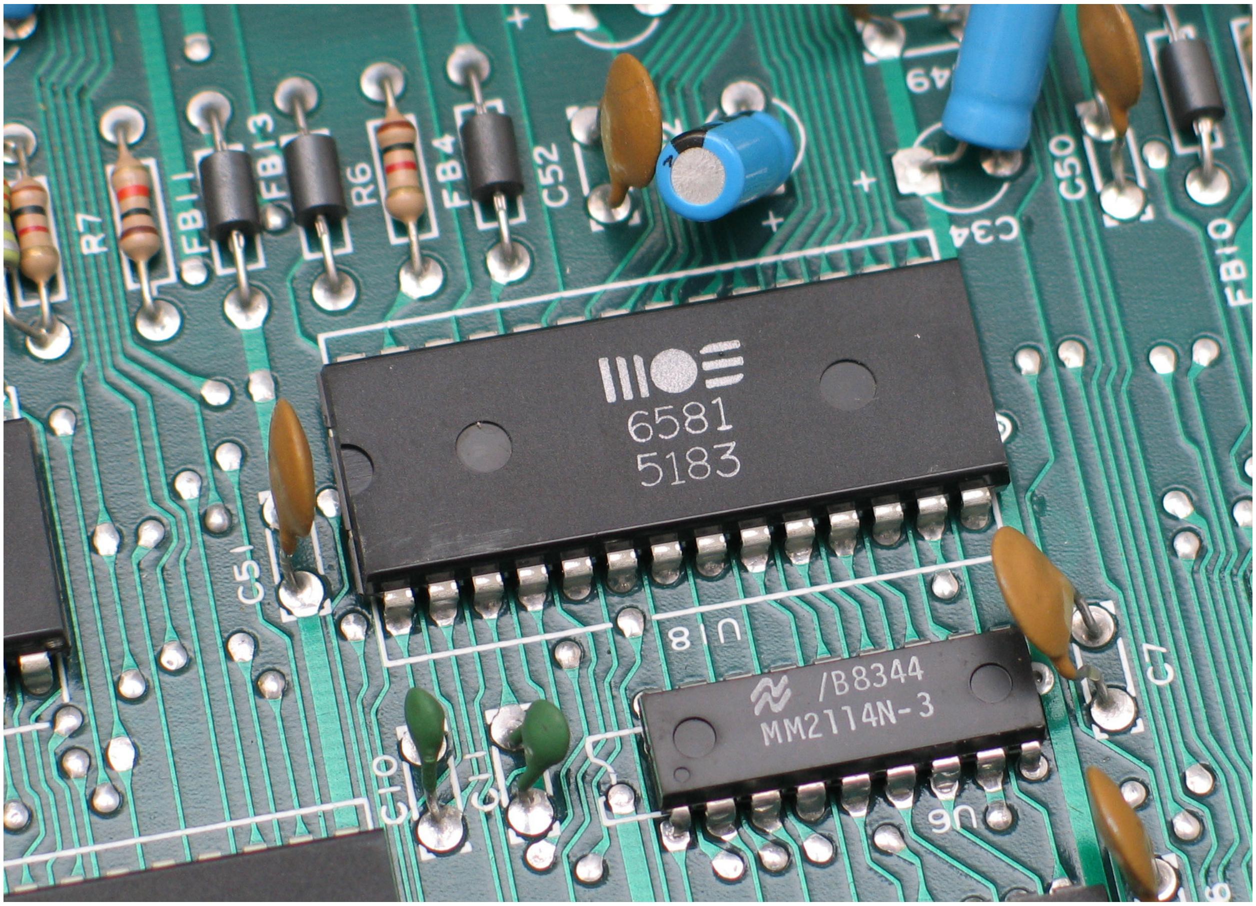


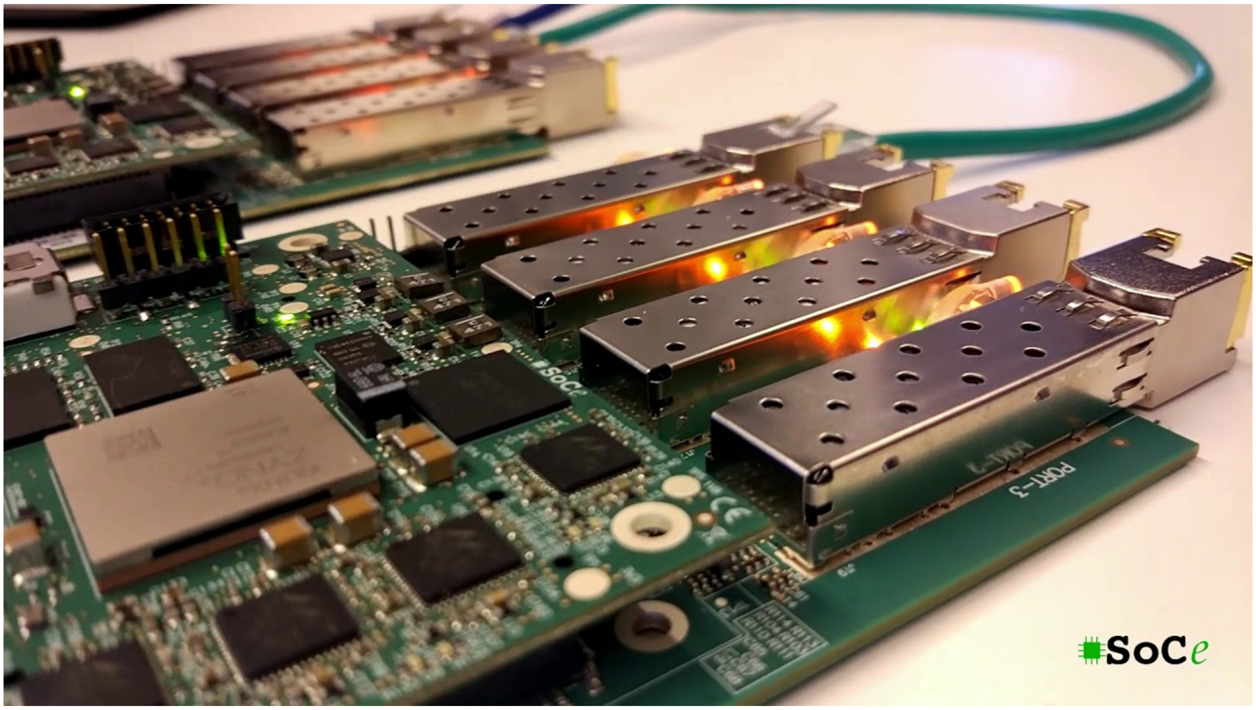
Рис.3: Прототип первой интегральной схемы. Джэк Килби.
Источник

Нужно две микросхемы - ставим их рядом, соединяем дорожками.



Источник

Нужно много-много микросхем - делаем большие платы или собираем платы в несколько слоёв.



Со временем транзисторы становились меньше, а микросхемы сложнее - внутри одного кристалла могли совмещаться несколько логических блоков - это были первые SoC - System On Chip - ЦПУ+Память+какие-нибудь специфические задачи. Но сильно много туда всё равно не засунешь.

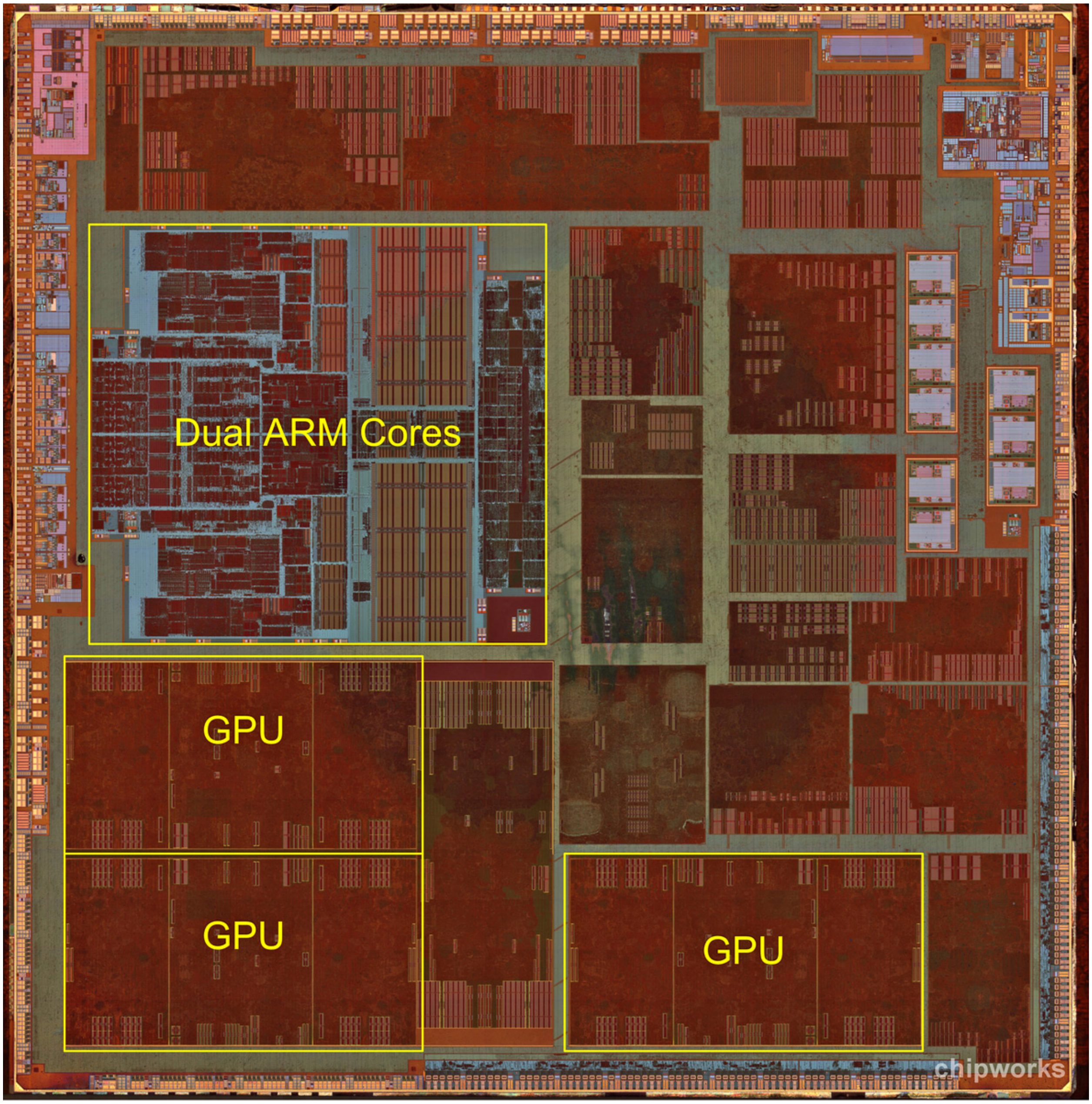


Рис.4: SoC Apple A6.
Источник

Потом научились, нарезав вафли на отдельные кристаллы, каждый из них запаковывать в отдельный пластиковый пакет, ставить один на другой и корпусировать вместе. Толщина таких бутербродов получалась большой - заметно больше, чем толщина ядра.

Что сделало возможными семимиллиметровые телефоны - так это Wafer-Level Packaging (**WLP**) или ещё это называют **Advanced Packaging** - технологии, позволяющие на одной вафле делать несколько микросхем и связывать их друг с другом. Сюда относятся: 2D, 2.5D, 3D, Fan-Out, 2.1D итд.

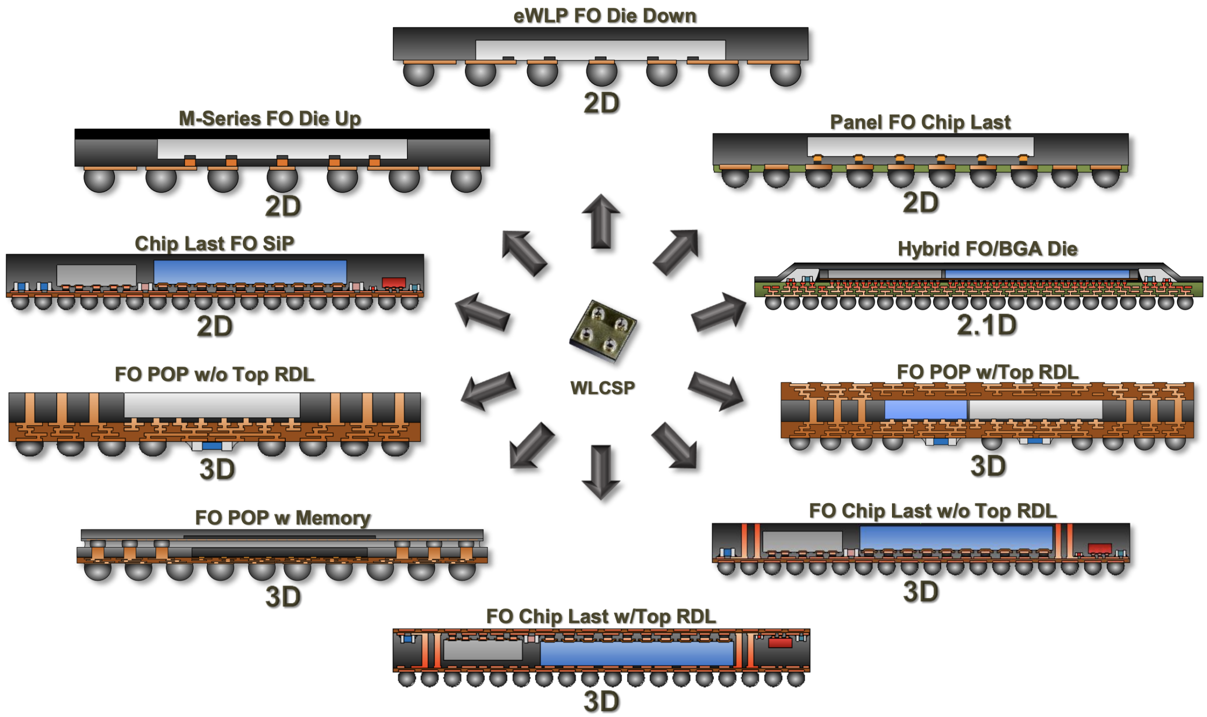
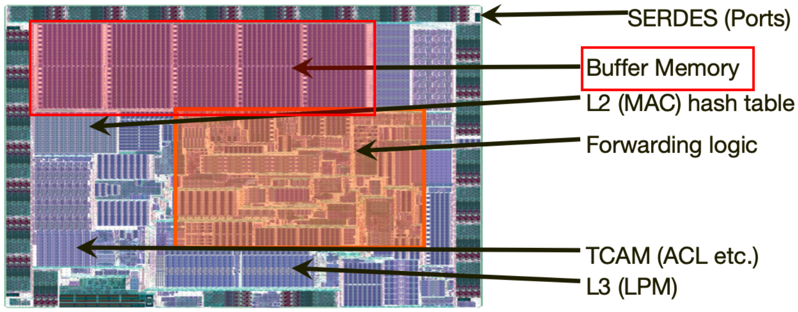


Рис.5: Неполное многообразие способов компоновки.
Источник

Каждый отдельный чип называется **чиплетом** (**chiplet**). Отличаются технологии Advanced Packaging'а друг от друга расположением чиплетов (рядом или один на другом), наличием и отсутствием интерпозера, материалом интерпозера (кремний, стекло, органика), видом электрических связей между кристаллами (металлические дорожки или TSV) и массой других. Суть их всех в том, что такая более плотная компоновка позволяет уменьшить размер, увеличить количество шин передачи данных, а соответственно и пропускную способность, значительно сократить энергопотребление и тепловыделение, отказавшись от длинных металлических дорожек.

Зачем я вообще в это решил копнуть? Во-первых, теперь немного понятнее становится что такое co-packged optics из предыдущего параграфа - это чиплет, реализующий оптический интерфейс, упакованный по одной из этих схем внутрь чипсета с ASIC'ом. А, во-вторых, дальше я буду рассказывать о памяти и буферах. Также я скажу о том, что каждый коммутационный ASIC обладает небольшим объёмом встроенной памяти - On-Chip Buffer. Она является составной частью асика коммутации.



Однако в некоторых случаях в помощь к ней добавляют объёмную внешнюю память, в которой пакеты будут храниться, если они не помещаются во встроенную. И пока это была какая-нибудь GDDR5, вынесенная отдельно на печатной плате в виде самостоятельного чипа, всё было понятно.

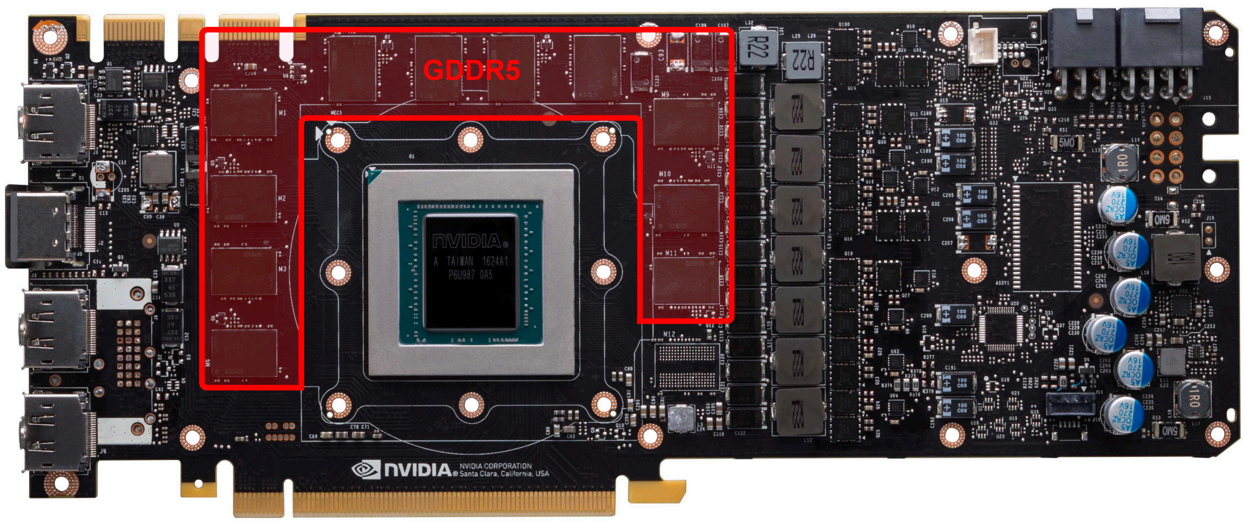
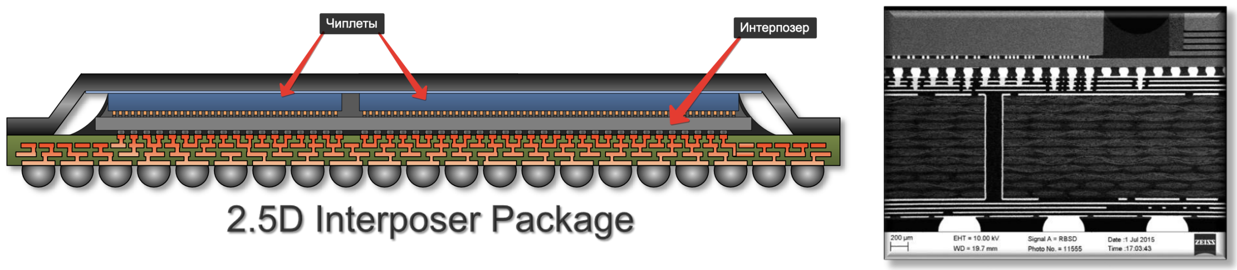


Рис.6: Своего родного сетевого не нашлось, но вот тут карточка от Энвидии с вынесенными GDDR5.

Но сегодня, с появлением WLP, когда под одной крышкой собраны и логика и память, границы начинают размываться. Некоторые вендоры спекулируют тем, что внешняя память co-packaged с коммутационным ASIC'ом, и утверждают, что она On-Chip.

Так, например, выглядит 2.5D корпусирование, где один чиплет соединён с другим через кремниевый интерпозер:



Источник

2.5D - это развитие классического подхода 2D. Оно всё ещё требует дорогостоящего выхода за пределы чиплета и прохода по длинным металлическим дорожкам. И кроме того обладает сравнительно невысокой пропускной способностью, ограничивающей максимальную скорость, на которой может работать чип в условиях, близких к перегрузкам.

Другой вариант - это уже 3D WLP:

Источник

HBM - High Bandwidth Memory - как раз и является High Bandwidth благодаря очень широким шинам взаимодействия между плотно скомпонованными чиплетами.

Расположенные друг над другом они соединены через **TSV - Through-Silicon Via** - это микроскопические проходные отверстия в кристаллах, залитые металлом (например, вольфрамом), которые позволяют чипам общаться друг с другом. Они значительно короче линий в интерпозере, что позволяет их сделать гораздо более эффективными.

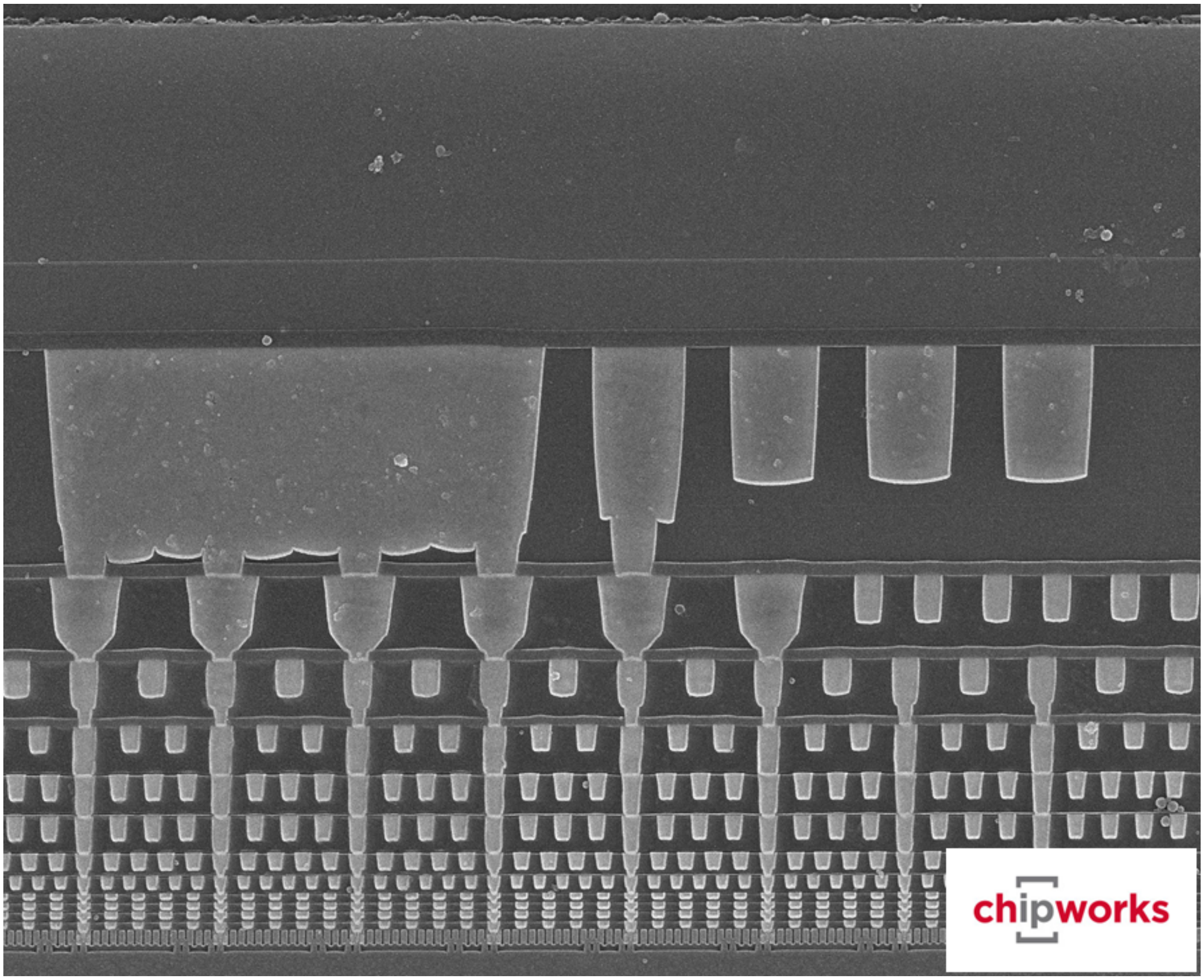
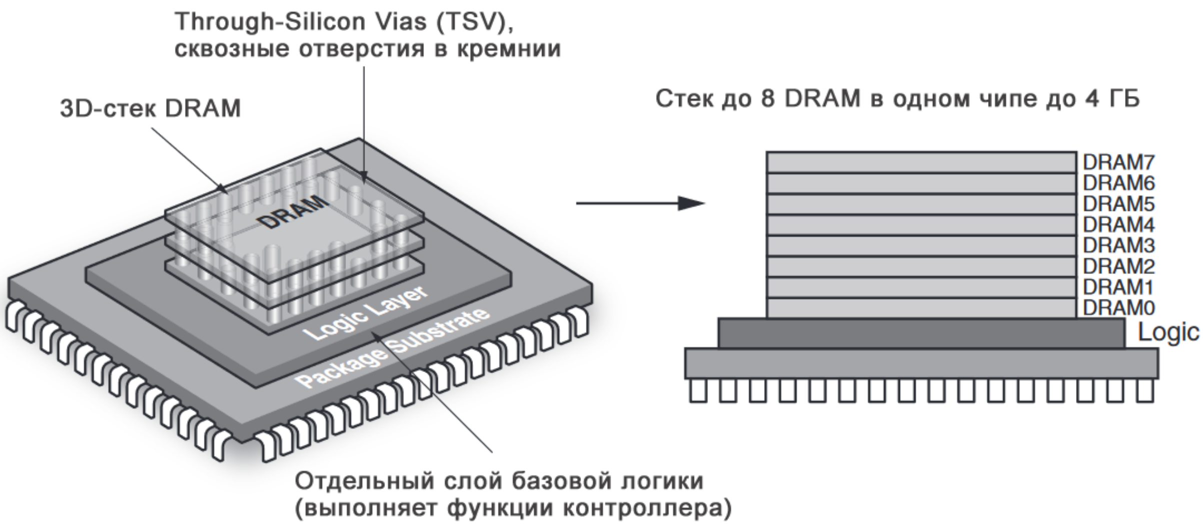


Рис.8: TSV
Источник

HBM весьма уверенно пробивает себе дорогу сегодня в мир Deep-Buffer!!! коробок.

Кстати по такой же 3D-технологии корпусирования была создана память **HMC - Hybrid Memory Cube**. Это бутерброд из склееных друг с другом плашек DRAM, пронизанных TSV.



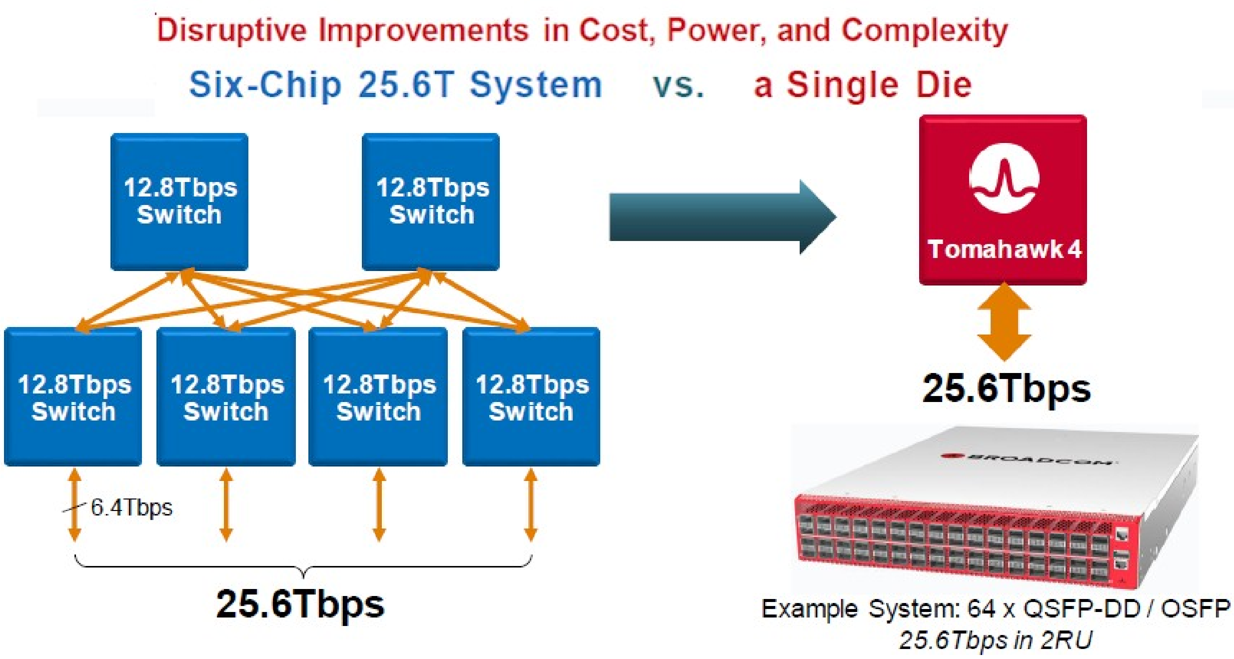
Источник

HMC - это коммерческая память производства Micron, от которой ныне отказались в пользу HBM и GDDR6.

Считать ли такую компоновку действительно On-Chip или нет - вопрос открытый, но складывается мнение, что в будущем необходимая от ASIC'а скорость коммутации будет расти всё равно быстрее, чем пропускная способность интерфейса до пусть даже лежащего плотно на ней чиплета HBM.

В чуть более узком, коммутационном смысле, под чиплетами могут иметься в виду составные части сетевых чипсетов, в которых высокая пропускная способность достигается не более производительным кристаллом, а набором нескольких менее производительных чиплетов, соединённых друг с другом через внутреннюю фабрику.

Бродком, кажется, считает это малодушием, продолжая изготавливать свои ультра-производительные чипы на монолитном кристалле.



Источник

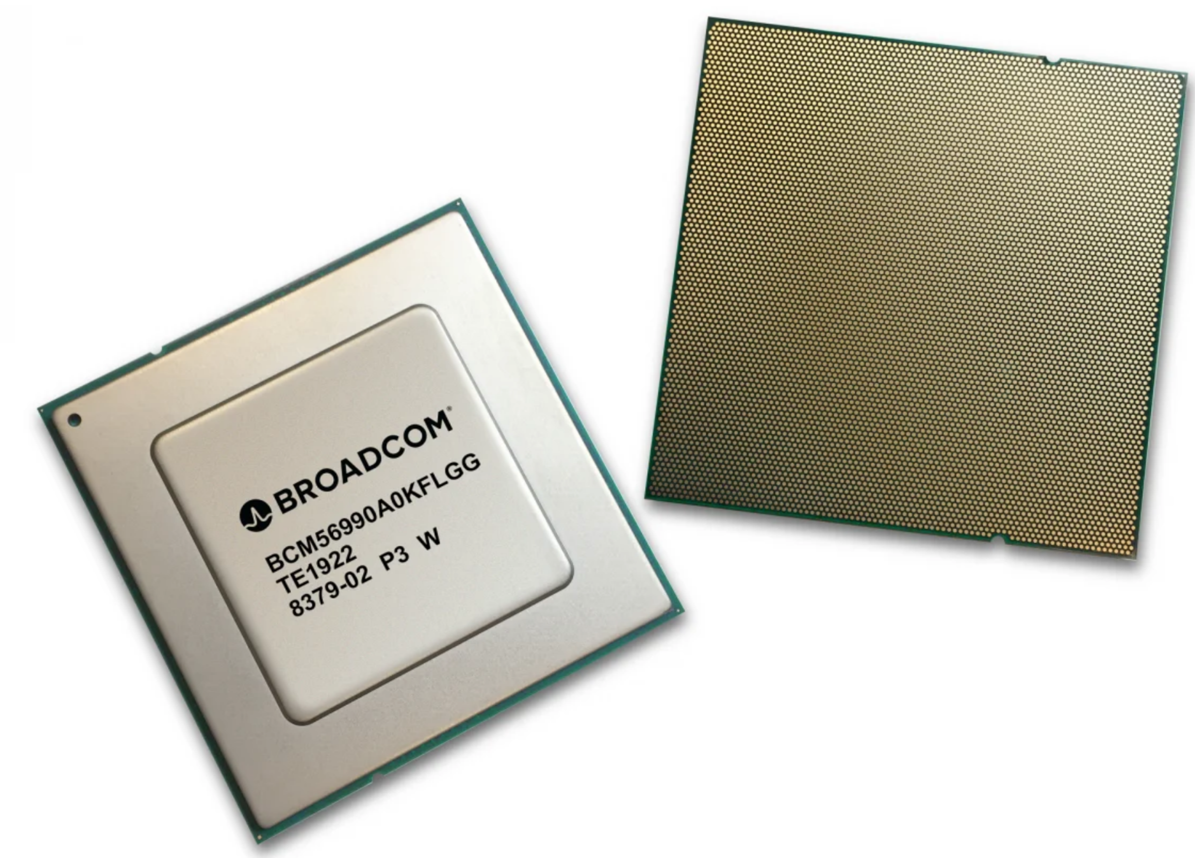
Дальнейшее чтиво:

- Understanding chiplet in one article
- Advanced packaging: five trends to watch in 2017
- Fan Out – Simple to Complex

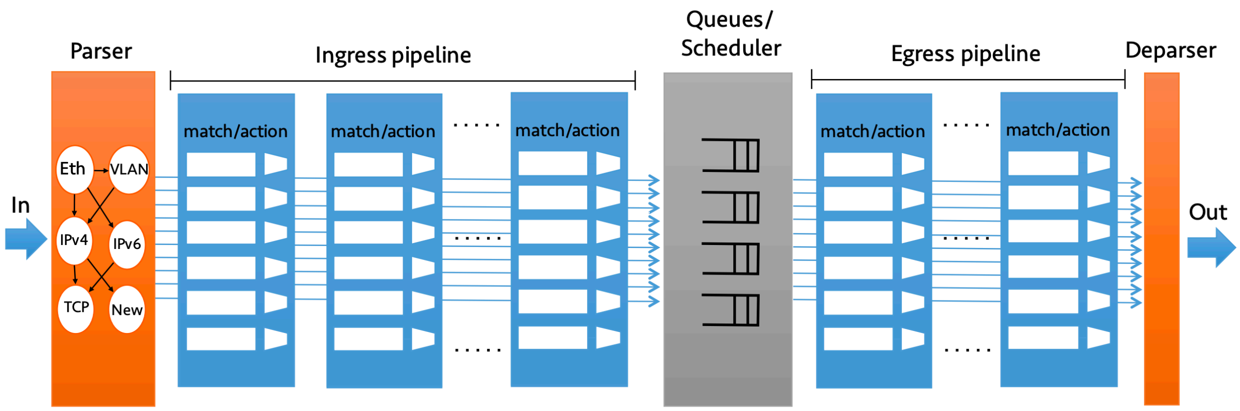
Логическое устройство

Типичный сетевой ASIC представляет из себя конвейер, по которому пакет передаётся от входного интерфейса к выходному, а по пути с ним случаются приключения. Английский термин для этого - Pipeline.

Хотя с виду и не скажешь:



В самом общем виде Pipeline выглядит так:



Источник

Пакет проходит через все стадии как минимум один раз, но для реализации сложных действий, вроде дополнительной инкапсуляции (VXLAN), может отправиться повторно.

Parser

Сначала на вход попадает пакет с неизвестным набором заголовков. Парсер разбирает все заголовки, отделяя их от собственно данных. Если это L2-коммутатор, то его заинтересует только заголовки Ethernet и VLAN. Если это MPLS-коммутатор, он заглянет в MPLS заголовки. Для L3 соответственно IPv4 и IPv6. Если это VXLAN-терминатор, ему понадобится UDP и собственно заголовок VXLAN. Для целей ECMP и ACL, парсер заглянет в UDP/TCP.

Строго говоря, на какие заголовки и какие поля в них надо смотреть, определяет фантазия разработчика. Сколько заголовков забрать? Как правило парсер вынимает фиксированное для чипа значение байтов от пакета и разделывает уже их. И это значение является одним из ограничений чипа.

Почему парсинг заголовков задача нетривиальная рассказывается в совместном исследовании Стэнфорда и Майкрософта: Design Principles for Packet Parsers.

Pre-Ingress processing

Иногда логически выделяют этот блок, который совершает действия, не являющиеся ни парсингом, ни как таковым лукапом - он кладет пакет в нужный VRF, ставит внутренний Traffic Class итд.

Ingress Match-Action

Когда парсер, разобрался с чем он имеет дело, он передаёт заголовки дальше - в модули Match-Action. Здесь происходит lookup. Для L2 поищем MAC'и, для L3 - прошерстим FIB, для MPLS - просмотрим в LFIB. И здесь же принимается решение, что с пакетом делать дальше: доложить CPU, пропустить/дропнуть, побалансировать, в какой порт отправить, какие заголовки навесить, с каким приоритетом внутри чипа обработать, полисить ли/шейпить ли его итд.

Собственно действие записано в той же таблице, в которой происходит lookup.

Это если коротко. А если чуть подлиннее, то:

L2 short pipeline

Приходящий L2-пакет всегда ассоциируется с VLAN. Тег VLAN проверяется через **VLAN Lookup Table**. Если VLAN lookup успешен, просматриваются таблицы: **VLAN STP**, **VLAN Port Bitmap**, **Port Filtering Mode** (PFM). Если же тег неверный, то пакет сбрасывается (или нет). После пакет проходит стандартную обработку: запомнить SRC MAC, посмотреть в таблице DST MAC, - но при этом могут быть применены дополнительные флаги, например, отправить на CPU unknown sender.

L3 short pipeline

Если DST MAC является MAC'ом самого устройства, то процессинг передается в L3 модуль. Следующий шаг - Destination Lookup. Сначала используется **L3 Table Lookup**, в этой таблице как правило directly attached хосты. Если адрес найден, то выдается index в **L3 Interface Table**, в котором выходной порт, MAC, VLAN. Если же в L3 table не найден адрес, то делается LPM поиск (Longest Prefix Match). Результат такого поиска - index в L3 Table Lookup таблице, который должен использоваться для форвардинга. После удачного поиска, чип поменяет SA/DA/VID пакета (L2), посчитает FCS, поменяет TTL и IP checksum.

Traffic Manager + MMU

Может также называться Buffer Manager. В этом блоке происходят следующие операции:

- Постановка пакетов в очередь
- Их хранение (буферизация)
- Контроль перегрузок
- Диспетчеризация
- Репликация

Он состоит из двух частей - MMU и TM. Первый отвечает за управление памятью и буферами, второй - за QoS и мультикаст.

MMU - Memory Management Unit - компонент чипа, который управляет физической памятью. Одна из его функций аналогична MMU (блоку управления памятью) обычного компьютера - доступ приложений к физической памяти и её защита. Но список его обязанностей гораздо шире, поскольку заточен он на работу именно с пакетами. Он отслеживает как память распределяется между интерфейсами и насколько она занята в каждый момент времени, можно ли поместить пакет в буфер, если да, то в какой, как разбить его на более мелкие ячейки, ну и, конечно, как его оттуда забрать.

TM - Traffic Manager - решает более высокоуровневые задачи - выделение очередей, помещение в них трафика, диспетчеризация, шейпинг, полисинг, управление перегрузками. В общем, всё, что относится к QoS, а так же к мультикасту.

Note:

С мультикастом история, право, интересная (как всегда). Репликацией мультикастовых пакетов занимается блок TM. В модульных устройствах это происходит в два этапа: сначала на входном чипе создаётся столько копий, сколько выходных чипов должны получить этот пакет, а затем на выходных чипах ещё столько копий, сколько портов на этой плате должны его получить. Делается это для того, чтобы лишними копиями не загружать фабрику.

> Любопытный момент с буферизацией и контролем перегрузок: входной чип должен учитывать занятость выходного порта, прежде, чем отправлять пакет, потому что именно входная плата управляет VOQ. Поскольку Traffic Manager оперирует не самими пакетами, а по сути информацией о них, то ему необязательно делать сразу копий по числу выходных портов, а достаточно записать об этом информацию в VOQ.

MMU - это не совсем часть TM - это, скорее, два взаимодействующих друг с другом блока.

Память и буферизация на сетевых устройствах - это настолько масштабная тема, что ей я посвятил отдельную статью, которая выйдет на nag.ru прямо следом за этой. В ней мы разберёмся с устройством и архитектурой памяти, видами и расположением буферов, арбитражем и поднимем самый горячий вопрос современности - что лучше: большие и маленькие буферы.

В разговорах о буферах связка TM+MMU - одни из важнейших блоков ASIC'а (или внешний чип), поэтому к ним мы ещё вернёмся позже.

Egress Match-Action

Далее над заголовками пакетов могут быть совершены дополнительные акты - например, выходной ACL, туннельные инкапсуляции, сбор статистики итд.

Deparser

К этому моменту на основе результатов обработки в блоках Match-Action сформирован список новых заголовков, и он может воссоединиться с телом пакета. Сам пакет теперь готов в последний путь внутри этого чипа, чтобы выйти через выходной интерфейс.

Кроме того, здесь может собираться дополнительная статистика о длине пакетов и сообщаться блоку TM и зеркалироваться исходящий трафик.

Вышеуказанные стадии могут быть выполнены в пределах одного чипа, а могут быть и разнесены на разные. Так, в случае single-chip-коробки - все они скомпонованы в один кусочек силикона, площадью с фотку на документы. На модульных коробках Parser и Ingress Match-Action - это входной чип коммутации, Egress Match-Action и Deparser - выходной, TM стоит отдельно между чипом коммутации и фабрикой, и может быть разделён на Ingress и Egress. Кроме того в модульных устройствах могут существовать ещё и отдельные чипы Fabric Interface, которые разбивают пакеты на ячейки одного размера и отправляют в фабрику.

Pipeline

Пайплайном называется весь процесс доставки пакета от парсера до депарсера. Что хорошо - можно делать несколько Match-Action подряд, например, отлукапив сначала Ethernet, потом

IP, потом ещё и TCP для ECMP.

Что плохо - число этих действий строго ограничено - ASIC вещь достаточно детерминированная. Это ведёт к тому, что некоторые вещи становятся аппаратно невозможны. К примеру, на старых Trident'ах нельзя было сделать и VXLAN и IP lookup последовательно для одного пакета. Или в другой ситуации коробка у меня не могла снять метку, сделать рекурсивный IP-lookup и навесить две новые метки.

Однако у таких трудностей есть как минимум три решения:

1. Второй чип. Тогда можно и разнести невозможные прежде операции на два этапа. И история знает такие решения.
2. Рециркуляция. Многие чипы позволяют закольцевать выход чипа на вход и прогнать пакет дважды. Тогда на второй итерации ему можно задать уже другой набор Match-Action. Но за это придётся заплатить - удвоенной задержкой и уменьшенной полосой пропускания чипа. А ещё можно упереться и в пропускную способность самого рециркулятора.
3. Купить другой чип… Другой коммутатор… Поменять работу…

Programmable Pipeline

В большинстве современных коммутаторов конвейер обработки пакетов запечён производителем в софт (если не в кремний) - он фиксирован и может быть изменён только вендором чипсета.

> **Attention:**
>
> Не путать с Programmable ASIC. Программируемые микросхемы - уже давно реальность. Многие сетевые чипы - это ASIC с возможность программирования. Но эта возможность есть только у производителя микросхемы.
>
> Порграммируемый конвейер же - это возможность изменять логику работы чипа в определённых пределах, которую предоставляет производитель микросхемы покупателям.

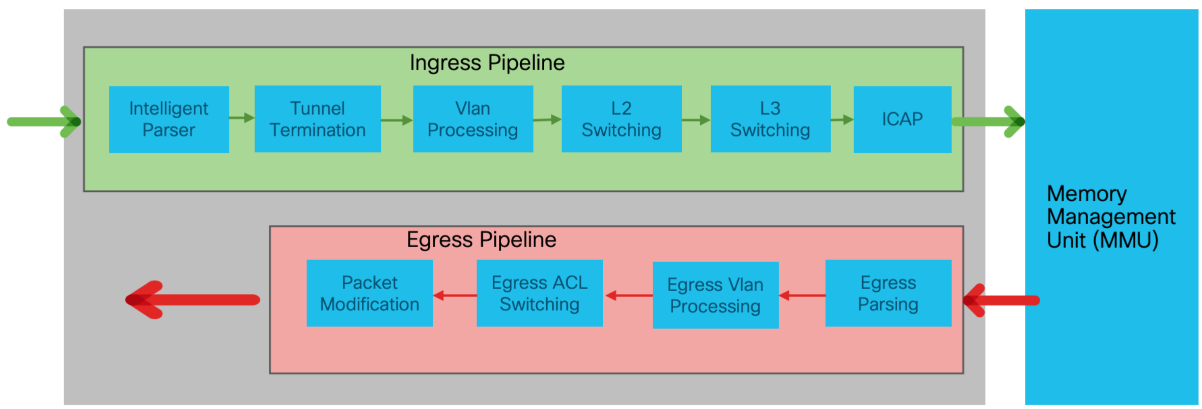
Не так давно появился Barefoot Tofino, у которого полностью программируемый Pipeline - с ним можно задавать совершенно любые условия для парсера, поля для Match и действия для Action - хоть калькулятор пишите, или распределённое хранилище на кластере коммутаторов.

На сегодняшний день выпускать сетевую микросхему на рынок без возможности программирования Pipeline'а, становиться плохим тоном. Так, последние чипы Broadcom тоже уже программируемы.

Не то чтобы теперь каждый домовой оператор кинется переписывать себе пайплайны, нанимая студентов для разработки под P4 или NPL, но это возможность, которая позволяет вендорам железа и крупным потребителям вроде гугла быть гораздо более гибкими. Так, например, если в вашей сети все линки p2p, то зачем вам Ethernet? тратить на него такты ASIC'а ещё - просто выкидываем его.

Правда "можно всё запрограммировать" превращается в наших реалиях "придётся всё запрограммировать". На сегодняшний день готовых конструктивных блоков, вроде парсинга Ethernet, IP, подсчёта статистики итд - не существует - всё с нуля. Если не использовать вендорские бинари, то весь Pipeline **придётся** написать самому. А если использовать, то ничего за пределами SDK не запрограммируешь.

Но на большинстве современных коммутаторов всё ещё фиксированный конвейер, который выглядит примерно так:



Дальнейшее чтиво:

- Cisco всё ещё делает классную документацию, а презентации с Cisco Live - кладезь технических сокровищ. Например, они рассказывают о бродкомовских чипах больше, чем сам Бродком (если, конечно, вы не подписали NDA кровью): Cisco Nexus 3000 Switch Architecture
- Вот у P4 есть неплохое описание конвейера и его программируемости: P4 Tutorial, Hot Chips 2017.
- Более фундаментальное и низкоуровневое описание **RMT** - Reconfigurable Match Tables, необходимых для возможности программирования: Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN.

6

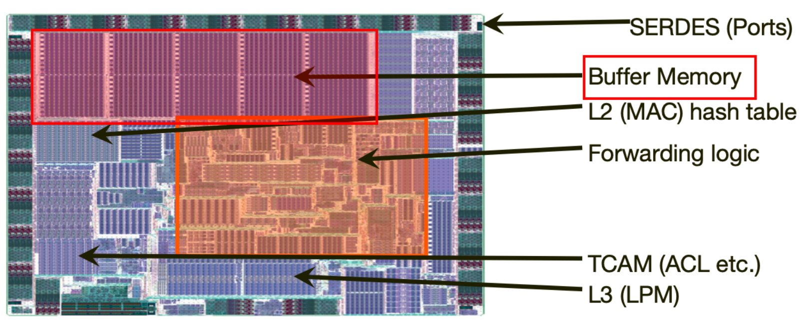
5. Память и буферы

Ну вот мы и добрались до ниточки, с которой клубок начал распутываться.

Всё, что было выше, касалось заголовков пакетов - они отделялись и подвергались алгоритмическим экзекуциям в пайплайне.

А где же прохлаждались их тела всё это время?

В сетевых чипсетах есть встроенная память (**OCB - On Chip Buffer**) как раз для хранения тел. Её размер в силу физических ограничений очень мал (до 100Мб), но для большинства задач - это разумный компромисс.



Note:

Бывают редкие исключения, где встроенная память имеет больший размер.

Но всегда были и будут сценарии, в которых нужна память гораздо большего объёма, в этом случае прибегают к использованию внешней относительно чипа коммутации памяти. Такие устройства называются Deep Buffer.

И о них и вообще о компромиссах мало-много памяти поговорим отдельно.

Таким образом на входе в чип парсер отделяет заголовки от тел, первые отдаёт на анализ в блок Match-Action, вторые - складывает в буфер.

На выходе новые заголовки пришивают к извлечённым обратно телам и отправляют на выход.

Будучи неоднократно обвинённым при рецензировании этой статьи, чувствую необходимость ещё раз повторить это: внутри чипа коммутации заголовки отделяются от тела пакета. В то время, как изначальные заголовки анализируются, помогают сделать лукап, уничтожаются, формируются новые, тело находится в одном месте физической памяти, не перемещаясь. Даже в тот момент, когда Traffic Manager выстраивает пакеты в очередь согласно их приоритетам, производит их диспетчеризацию и Congestion Avoidance, фактически он работает со внутренними временными заголовками, не двигая пакеты по памяти.

Управляет доступом к физической памяти **MMU - Memory Management Unit**. Он довольно похож на MMU в компьютерах (по сути является им). Программа обращается к указателю, чтобы извлечь данные из памяти, MMU транслирует это в реальный адрес ячейки и возвращает данные.

MMU занимается размещением пакетов в буферах, их извлечением или отбрасыванием. Он же контролирует разделение памяти на области (dedicated, shared, headroom, voq) и их загрузку.

За более верхнеуровневое управление очередями и перегрузками отвечает блок **TM - Traffic Manager**.

Есть два подхода к размещению тел в буферах: **Store-and-Forward** и **Cut-Through**.

Store-and-Forward vs Cut-Through

SF - Store and Forward предполагает, что чип сначала получает полностью пакет, сохраняет в буфере, а уже потом занимается вивисекцией.

После анализа заголовков пакет помещается в правильную выходную очередь.

Исторически это первый метод коммутации. Его преимущество в том, что он может, получив пакет целиком, проверить контрольную сумму и выбросить побитые.

CT - Cut-Through, напротив, сразу после получения первых нескольких десятков байтов пакета, позволяющих принять решение о его судьбе, отправляет его в выходную очередь, не дожидаясь его полной доставки.

Это позволяет сэкономить до нескольких микросекунд на обработке пакета внутри коробки, ценою, однако, отсутствия проверки целостности - ведь FCS-то в конце. Эта задача перекладывается на протоколы более высокого уровня (или на следующий хоп).

Такой режим используется для приложений, требующих ультра-коротких задержек.

Tip:

К слову на сегодняшний день для коммутаторов 2-го уровня время обработки в пределах устройства порядка 200нс, 3-го - 250нс.

Для коммутаторов 1-го уровня (фактически патч-панель) это время - около 5 нс.

Многие производители сегодня по умолчанию устанавливают режим Cut-Through, поскольку ошибки на Ethernet сегодня явление сравнительно нечастое, а приложение обычно само может обнаружить проблему и запросить переотправку (или не запрашивать, кстати).

Дальнейшее чтение:

- Весьма глубокий подкоп под режимы коммутации от циски: Cisco Nexus 5000 Series Switches

Перегрузки: причины и места

Буфер нужен на сетевом устройстве не только для того, чтобы похранить тело пакета, пока его заголовки перевариваются в кишках чипа.

Как минимум нужно сгладить поток пакетов до скорости выходного интерфейса. Грубо говоря, пока один пакет сериализуется для передачи в интерфейс, второму нужно где-то подождать.

Для этой задачи обычно хватит совсем небольшой FIFO очереди.

Другая важнейшая сетевая задача - перегрузки (congestion). Если в один интерфейс одновременно сваливается много пакетов, их тоже нужно сохранить.

Причём перегрузки - это наша повседневная легитимная реальность, а не что-то крайне редкое, что нужно перетерпеть и станет попроще.

Во времена коммутации каналов такой проблемы не стояло - для любой общающейся пары всегда было зарезервировано строго необходимое количество ресурсов.

Сегодня совершенно законно на порт может прийти больше трафика, чем тот готов сиюминутно пропустить.

Причины перегрузок

Самая простая - из высокоскоростного интерфейса трафик должен слиться в более низкоскоростной - из **10G в 1**, например.

Другая причина, очень распространённая в сетях крупных ДЦ, особенно в тех, где развёрнуты кластеры Map-Reduce - это **Incast**. Это ситуация, в которой одна машина отправляет запрос на десятки/сотни/тысячи, а те все разом начинают отвечать, и пакеты стопорятся на узком интерфейсе в сторону машины-инициатора.

Более общий случай - трафик с нескольких входящих портов должен влиться в один исходящий - **Backpressure**.

Прочие типы всплесков трафика, которые ещё называют бёрстовыми или просто бёрстами (**Bursts**).

Поэтому однозначно нужно побольше памяти для буферизации. Но фактически это место, где пакеты обрастают задержкой.

Так на заре 1G буферизация вызывала массу головной боли у трейдеров, чьи приложения получали свой бесценный трафик с задержкой и джиттером.

И поэтому тут уже недостаточно везде FIFO. Это задача, во благо которой трудится QoS.

Если случилась конгестия, то нужно иметь возможность требовательный к задержкам трафик, пропустить первым, чувствительный к потерям - не дропнуть, а наименее ценным пожертвовать.

Ещё нужно уметь пополисить и пошейпить.

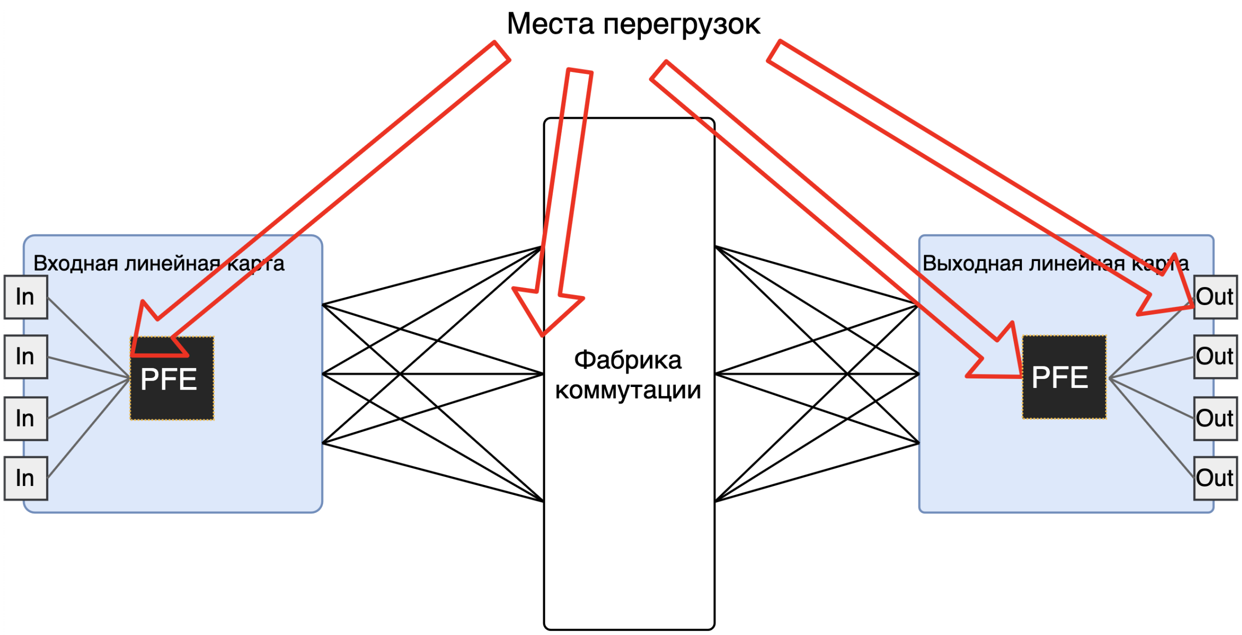
Но в каком месте располагать эту память и где реализовывать QoS?

Места возникновения перегрузок

Их по большому счёту 4:

1. **на входном чипе** - если со стороны интерфейсов на него поступает больше, чем он способен обработать.
2. **на фабрике коммутации** (если коробка модульная) - если линейные карты пытаются отправить на фабрику больше, чем она способна обработать.
3. **на выходной линейной карте**, если фабрика пытается передать на линейную карту больше, чем её чип способен обработать

4. **на выходном интерфейсе** - если чип шлёт в интерфейс больше, чем тот способен сериализовать.



Но никто не хочет бороться с перегрузками в четырёх местах.

Поэтому обычно

А) Чип делают такой производительности, чтобы он смог обработать весь трафик, даже если тот начал одновременно поступать со всех портов на этой линейной карте. Поэтому для устройства со 128 портами 100Гб/с используется чип с производительностью 12,8Тб/с.

Очевидно бывают и исключения. Тогда или имеем непредсказуемые потери, или (чаще) невозможность использовать часть портов.

Б) Фабрику так же делают без переподписки, чтобы она могла провернуть весь трафик, который пытаются в неё передать все линейные карты, даже если они делают это одновременно на полной скорости. Таким образом не нужно буферизировать трафик и перед отправкой на фабрику. Более того, её приходится делать не просто равной сумме пропускных способностей входных/выходных интерфейсов, а больше - там ведь помимо самих пакетов передаются ещё метаданные.

В) Управление перегрузками на выходном чипе и выходном интерфейсе сводят в одно место.

> **Attention:**
>
> На самом деле фабрика без передподиски (или неблокируемая) - это та ещё спекуляция, к которой нередко прибегают маркетологи.

> Для некоторых сценариев, например, CIOQ даже со speedup фабрики в пару раз от необходимого есть строгие результаты, показывающие, при каких условиях она будет неблокируемой.
>
> Можно почитать у достопочтенных выпускников MIT и Стэнфорда: On the speedup required for combined input and output queued switching.

Архитектура буферов

И вот тут на сцену выходит TM - Traffic Manager, который реализует функции QoS (и некоторые другие). Он может быть частью чипа коммутации, а может быть отдельной микросхемой - для нас сейчас важно то, что он заправляет буферами.

Буфер - это с некоторыми оговорками обычная память, используемая в компьютерах. В ней в определённой ячейке хранится пакет, который чип может извлечь, обратившись по адресу.

Любой сетевой ASIC или NP обладает некоторым объёмом встроенной (on-chip) памяти (порядка десятков МБ). Так называемые Deep-Buffer свитчи имеют ещё внешнюю (off-chip) память, исчисляемую уже гигабайтами. И той и другой управляет модуль чипа - MMU.

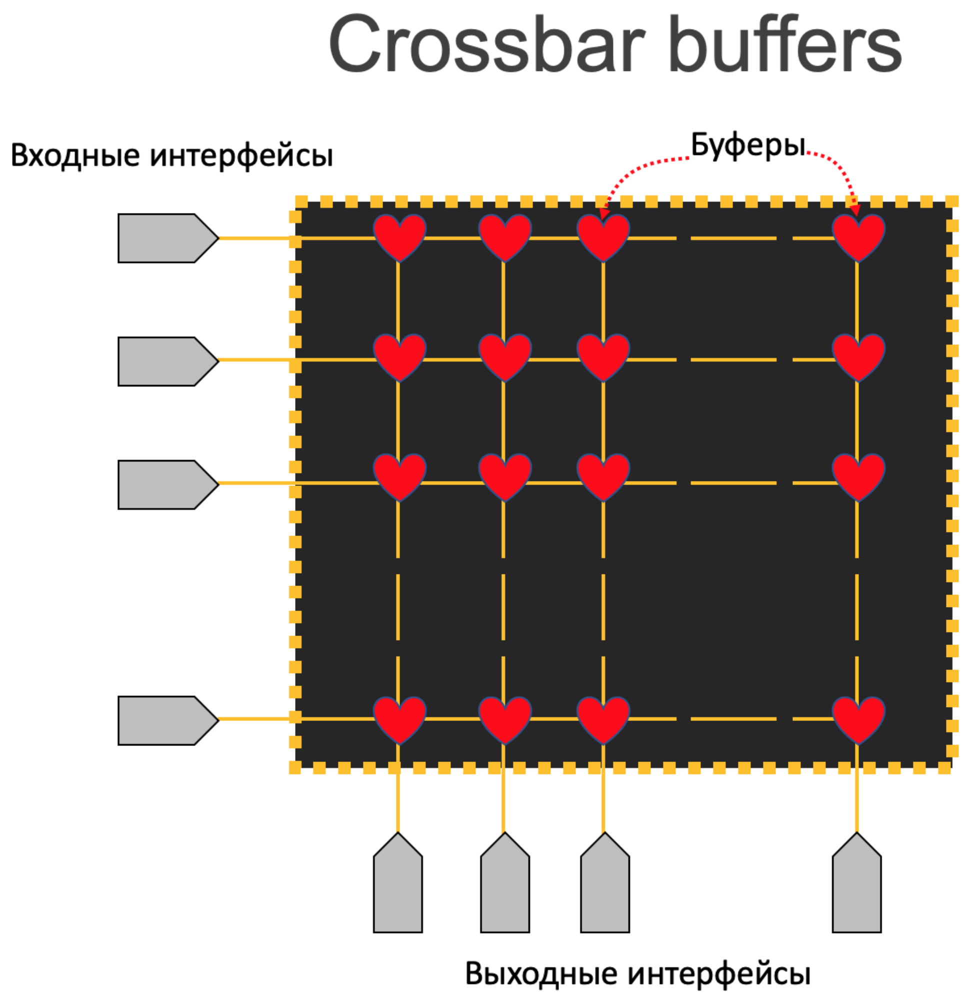
В целом для нас пока местонахождение не имеет значения - взглянем на это попозже. Важно то, как имеющейся памятью чип распоряжается, а именно, где и какие очереди он создаёт и какие AQM использует.

И тут практикуют:

- *Crossbar*
- *Shared Buffer*
- *Output Queueing*
- *Input Queuing*
- *Combined Input and Output Queueing*
- *Virtual Output Queueing*

Crossbar

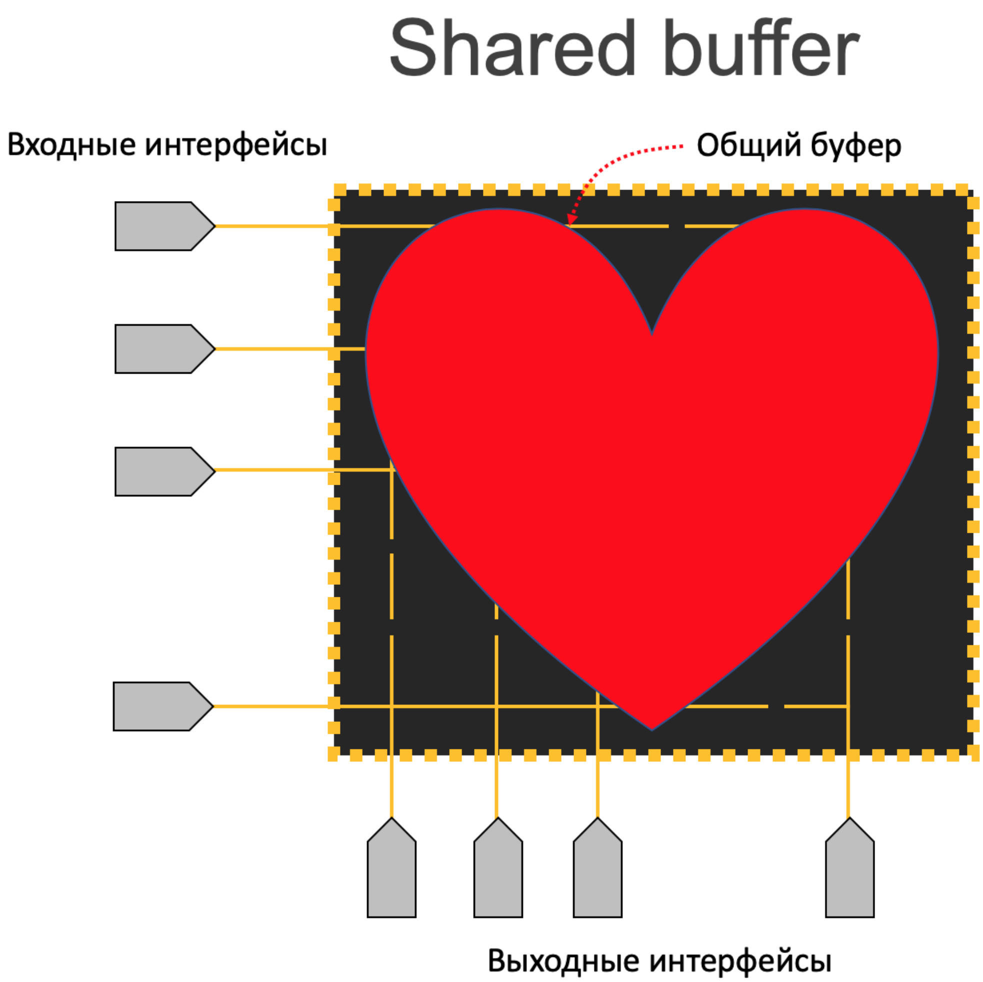
Идея в том, чтобы для каждой пары (входной интерфейс - выходной интерфейс) выделить аппаратный буфер.



Это, скорее, умозрительный эксперимент, потому что в плане сложности, стоимости реализации и эффективности это проигрышный вариант.

Shared Buffer

По числу существующих в мире коробок этот вариант, однозначно, на первом месте.



Используется Shared Buffer на немодульных устройствах без фабрики коммутации, в которых установлен один чип (обычно, но может быть больше).

Аппаратно - это память (обычно SRAM), встроенная прямо в чип - она так и называется **on-chip** (OCB). Много туда не засунешь, поэтому объём до 100 МБ.

Зачастую это единственная память, которая в одночиповых устройствах используется для буферизации.

Пусть, однако, эта кажущаяся простота не вводит вас в заблуждение - для того, чтобы в десятки мегабайтов поместить трафик сотни портов 100Гб/с, да ещё и обеспечить отсутствие потерь, за ними должны скрываться годы разработок и нетривиальная архитектура.

А так оно и есть - я чуть ниже неглубоко вас окуну.

Итак, есть соблазн эту память взять и просто равномерно разделить между всеми портами. Такой статический дизайн имеет право на жизнь, но сводит на нет возможность динамически абсорбировать всплески трафика.

Гораздо более привлекательным выглядит следующий вариант:

Dedicated + Shared

Из доступной памяти каждому порту выделяется определённая небольшая часть - это **Dedicated Buffer**. За каждым портом кусочек памяти законодательно закреплён и не может быть использован другими портами. То есть при любых обстоятельствах у порта будет свой защищённый кусочек. Минимальный размер Dedicated Buffer где-то настраивается, где-то нет. Но лучше без основательного понимания в дефолты не лезть.

Доля каждого порта в абсолютных цифрах очень маленькая - порядка единиц кБ.

Гарантируемый минимум выделяется для хранения как входящих пакетов, так и выходящих.

Остальная часть памяти как раз общая - **Shared Buffer** - может быть использована любым портом по мере необходимости. Из неё динамически выделяются куски для тех интерфейсов, которые испытывают перегрузку.

Например, если чип пытается на один из интерфейсов передать больше трафика, чем тот способен отправлять в единицу времени, то эти пакеты сначала заполняют выделенный для этого порта буфер, а когда он заканчивается, автоматически начинают складываться в динамически выделенный буфер из общей памяти. Как только все пакеты обработаны, память освобождается.

Под общий буфер может быть отдано 100% той памяти, что осталась после вычитания из неё выделенных для портов кусочков (Dedicated). Но она так же может быть перераспределена - за счёт общего буфера можно увеличить выделенные. Так, если выделить 80% под Shared, то оставшиеся 20% равномерно распределятся по Dedicated.

Наличие Shared Buffer'а решает огромную проблему, позволяя сглаживать всплески трафика, когда перегрузку испытывает один или несколько интерфейсов.

Однако вместе с тем за общую память начинаются соревноваться разные порты одновременно. И серьёзная перегрузка на одном порту может вызвать потери на другом, которому нужно было всего лишь несколько килобайтов общей памяти, чтобы не дропнуть пакет.

Одним из способов облегчить эту ситуацию является увеличение выделенных буферов за счёт уменьшения общего.

Но это всегда зона компромиссных решений - сокращая размер общей памяти, мы уменьшаем и объёмы всплесков, которые чип может сгладить.

Кроме того Lossless трафик требует к себе ещё более щепетильного отношения.

Поэтому зачастую, помимо Dedicated и Shared буферов, резервируют ещё **Headroom buffers**.

Headroom buffers

Это последний способ сохранить пакеты, когда даже общий буфер уже забит. Естественно, он тоже отрезается от общей памяти, поэтому на первый взгляд выглядит не очень логичным откусить от общей памяти кусок, назвать его по-другому и сказать, мол, мы всё оптимизировали.

На самом деле Headroom буферы решают довольно специфическую задачу - помочь lossless приложениям с **PFC** - Priority-based Flow Control.

PFC - это механизм Ethernet Pause, который умеет притормаживать не всю отправку, а только по конкретным приоритетам Ethernet CoS.

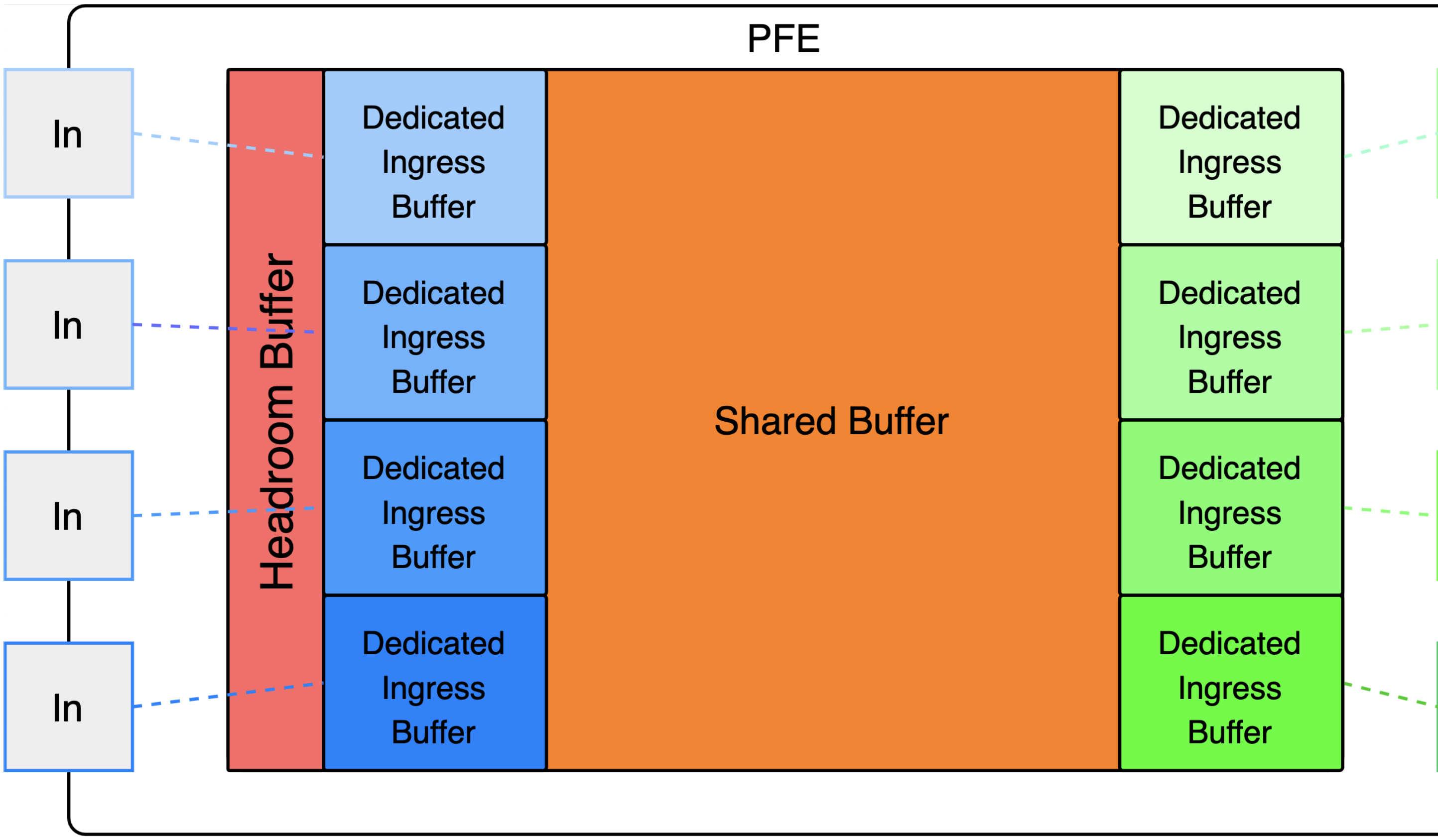
Например, два приложения на отправителе: RoCE и репликация БД. Первое - чувствительная к задержкам и потерям вещь, второе - массивные данные.

Коммутатор, заметив заполнение общего буфера, отправляет Pause для no-drop трафика, например, RoCE, чтобы тот не пришёл к закрытым воротам и не был отброшен.

Задача буфера Headroom здесь в том, чтобы сохранить in-flight пакеты приоритетной очереди (те, что сейчас в кабеле), пока Pause летит к отправителю с просьбой притормозить.

То есть пакеты репликации начнут дропаться, когда заполнится общий буфер, а пакеты RoCE будут складываться в Headroom.

> Помимо lossless headroom бывает и headroom для обычного трафика, чтобы помочь сохранить более приоритетный. Но это на домашнее задание.



При наступлении перегрузки буферы будут задействованы в следующем порядке.

Для входящего best-effort трафика:

1. Dedicated buffers
2. Shared buffers

Для входящего lossless трафика:

1. Dedicated buffers
2. Shared buffers
3. Lossless headroom buffers

Для всего исходящего трафика:

1. Dedicated buffers
2. Shared buffers

Разумеется, описанное выше лишь частный пример, и от вендора к вендору ситуация может различаться (разительно).

Например бродкомовские чипы (как минимум Trident и Tomahawk) имеют внутреннее разделение памяти по группам портов. Общая память делится на порт-группы по 4-8 портов, которые имеют свой собственный кусочек общего буфера. Порты из одной группы, соответственно буферизируют пакеты только в своём кусочке памяти и не могут занимать другие. Это тоже один из способов снизить влияние перегруженных портов друг на друга. Такой подход иногда называют **Segregated Buffer**.

Admission Control

Admission Control - входной контроль - механизм, который следит за тем, можно ли пакет записывать в буфер. Он не является специфичным для Shared-буферов, просто в рамках статьи - это лучшее место, чтобы о нём рассказать.

Формально Admission Control делится на Ingress и Egress.

Задача **Ingress Admission Control** - во-первых, вообще убедиться, что в буфере есть место, а, во-вторых, обеспечить справедливое использование памяти.

Это означает, что у каждого порта и очереди всегда должен быть гарантированный минимальный буфер. А ещё несколько входных портов не оккупируют целиком весь буфер, записывая в него всё новые и новые пакеты.

Задача **Egress Admission Control** - помочь чипу абсорбировать всплески, не допустив того, чтобы один или несколько выходных портов забили целиком весь буфер, получая всё новые и новые пакеты с кучи входных портов.

В случае Shared Buffer оба механизма срабатывают в момент первичного помещения пакета в буфер. То есть никакой двойной буферизации и проверки не происходит.

Как именно понять, сколько буфера занято конкретным портом/очередью и главное, сколько ещё можно ему выдать?

Это может быть статический порог, одинаковый для всех портов, а может быть и динамически меняющийся, регулируемый параметром **Alpha**.

Alpha

Итак, почти во всех современных чипах память распределяется динамически на основе информации о том, сколько общей памяти вообще свободно и сколько ещё можно выделить для данного порта/очереди.

На самом деле минимальной единицей аккаунтинга является не порт/очередь, а регион (в терминологии Мелланокс). Регион - это кортеж: *(входной порт, Priority Group на входном порту, выходной порт, Traffic Class на выходном порту)*.

Каждому региону назначается динамический порог, сколько памяти он может под себя подмять. При его превышении, очевидно, пакеты начинают дропаться, чтобы не влиять на другие регионы.

Этот порог вычисляется по формуле, множителями которой являются объём свободной на данный момент памяти и параметр **alpha**, специфичный для региона и настраиваемый:

```
Threshold [Bytes] = alpha * free_buffer [Bytes]
```

Его значение варьируется от 1/128 до примерно 8 с шагом x2. Чем больше эта цифра, тем больший объём свободной памяти доступен региону.

Например, если на коммутаторе 32 региона, то:

при alpha=1/64 каждому региону будет доступна 1/64 часть свободной памяти, и даже при максимальной утилизации они все смогут использовать только половину буфера.

при alpha=1/32 вся память равномерно распределится между регионами, ни один из них не сможет влиять на другие, а при полной утилизации 100% памяти будет занято.

при alpha=1/16 каждый регион может претендовать на больший объём памяти. И если все регионы разом начнут потреблять место, то им всем не хватит, потому что памяти потребовалось бы 200%. То есть это своего рода переподписка, позволяющая сглаживать всплески.

Предполагаем тут, что значение alpha одинаково для всех регионов, хотя оно может быть настроено отдельно для каждого.

При получении каждого пакета, механизм Admission Control вычисляет актуальный порог для региона, которому принадлежит пакет. Если порог меньше размера пакета, тот отбрасывается.

Если же больше, то он помещается в буфер и уже не будет отброшен никогда, даже если регион исчерпал все лимиты. Объём свободной памяти уменьшается на размер пакета.

Это происходит для каждого приходящего на чип пакета.

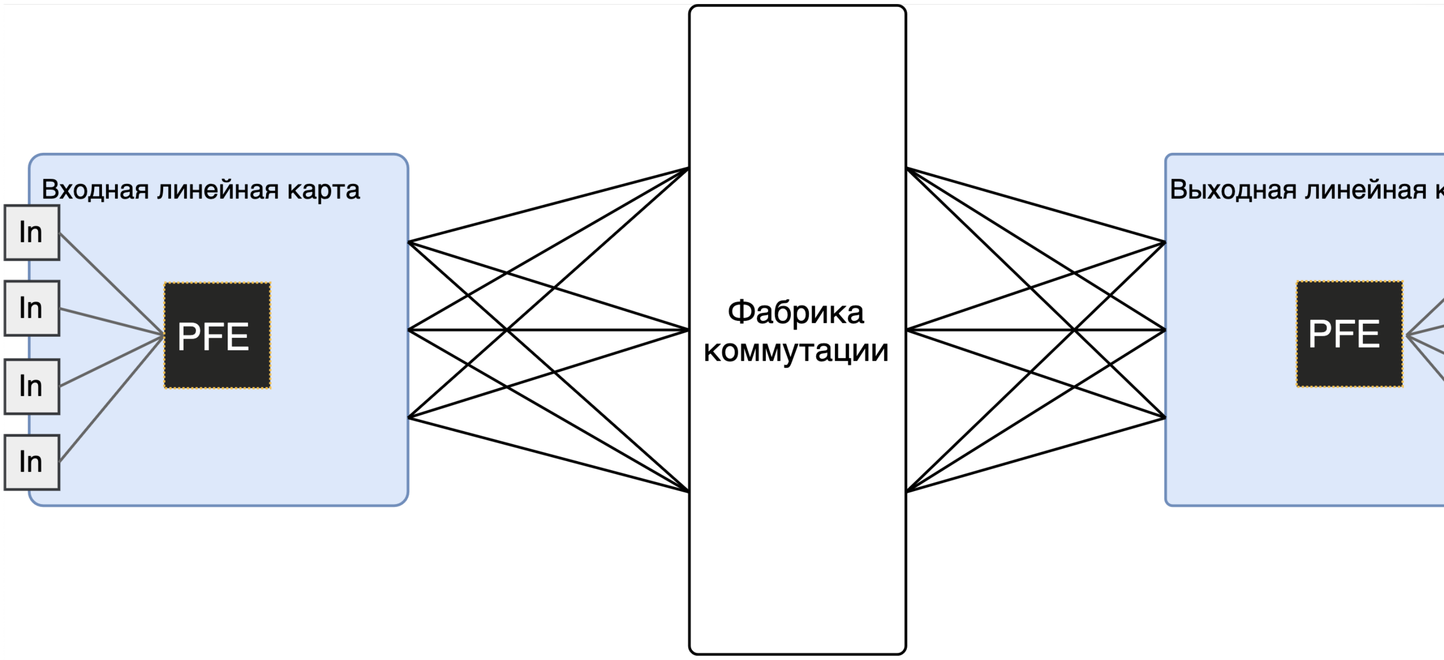
Написанное выше об Admission Control и Alpha может быть справедливо не только для Shared Buffers, но и для других архитектур, например, VoQ.

Дальнейшее чтиво:

- Если в жизни не хватает страданий: Design and Implementation of a Shared Memory Switch Fabric
- Understanding the Alpha Parameter in the Buffer Configuration of Mellanox Spectrum Switches
- Programming Guide'ы коммерческих микросхем (NDA кровью, помним, да?).

Crossbar и Shared Buffer - это архитектуры, которые могут использоваться для устройств фиксированной конфигурации (возможно, даже multi-chip), но не подходят для модульных.
Взглянем же теперь на них.

Дело в том, что они состоят из нескольких линейных карт, каждая из которых несёт как минимум один самостоятельный чип коммутации.
И этот чип, будь то ASIC, NP или даже CPU не может в своей внутренней памяти динамически выделять буферы для тысяч очередей выходных интерфейсов - кишка тонка.

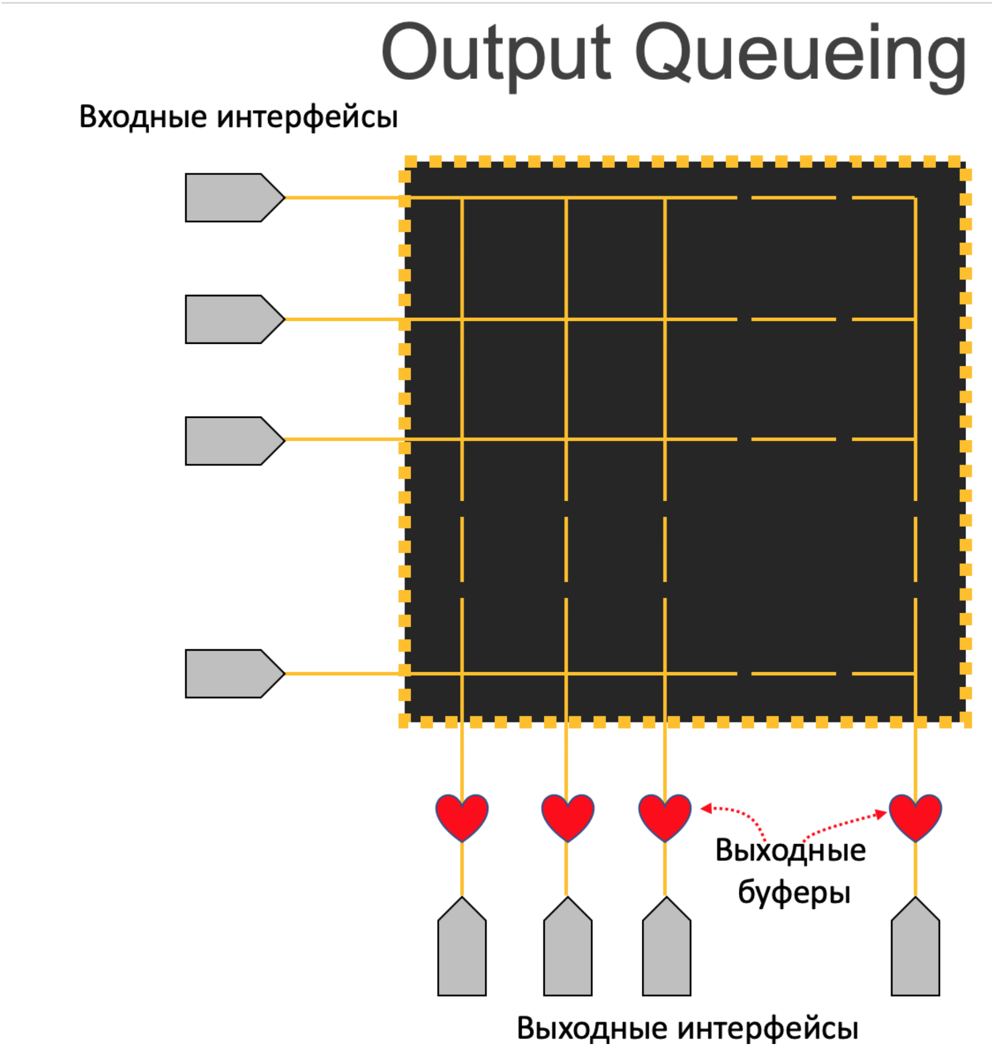


Далее поговорим про архитектуры памяти для модульных шасси.

Output Queueing

Наиболее логичным кажется буферизировать пакеты как можно ближе к месту возможного затора - около выходных интерфейсов.

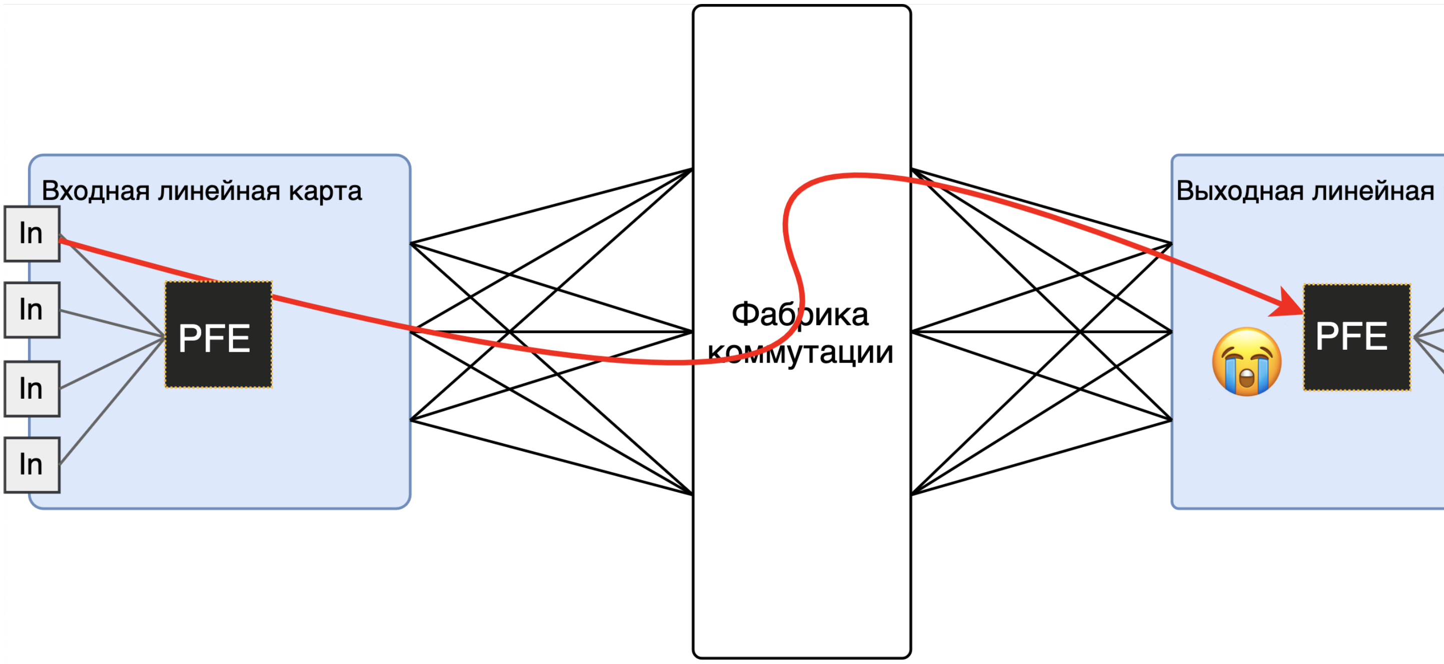
Кому как не выходному чипу знать о здоровье своих подопечных интерфейсов, обслуживать по несколько QoS очередей для каждого и бороться с перегрузками?



И это правда так.

Но есть одна фундаментальная проблема - в случае перегрузок пакеты будут приходить на Egress PFE, чтобы умирать. Они проделают весь огромный путь от входного интерфейса через фабрику коммутации до выходного буфера через фабрику для того, чтобы узнать, что мест нет и быть печально дропнутыми.

Это бессмысленная и бесполезная утилизация полосы пропускания фабрики.

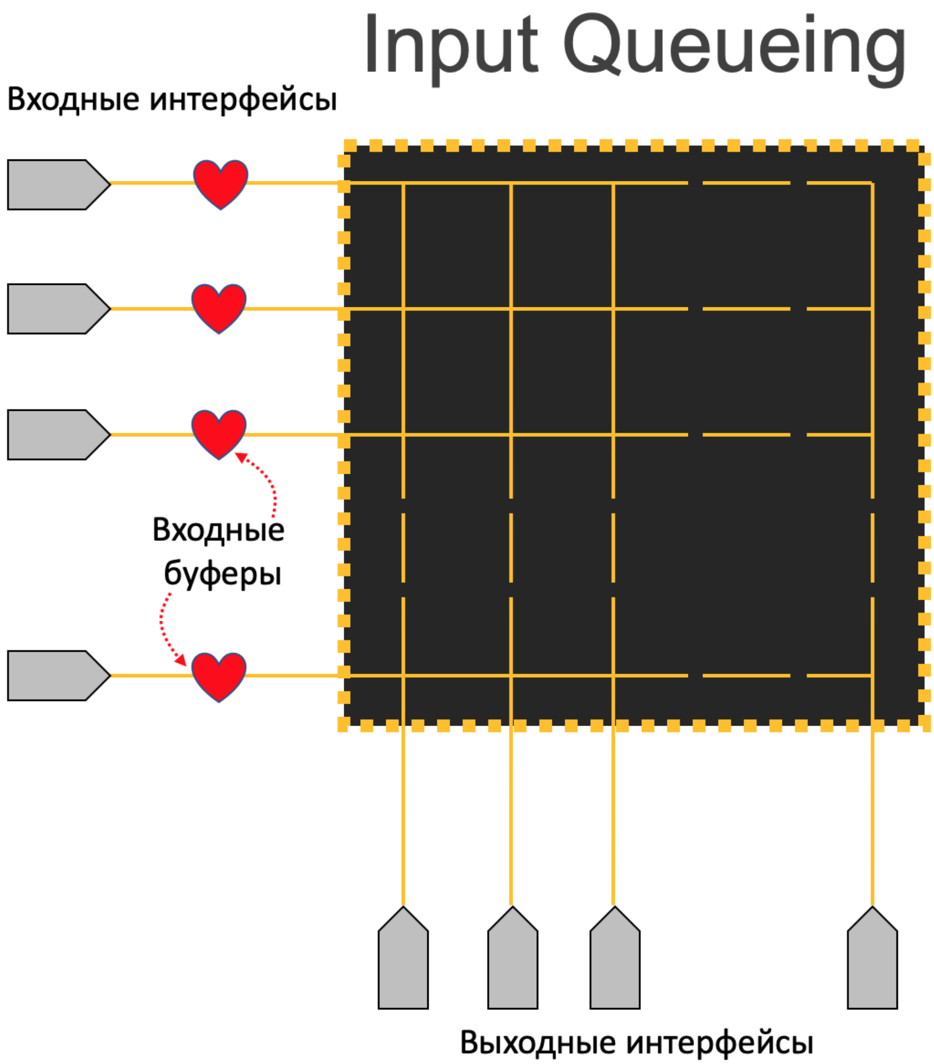


И вот уже вырисовывается следующая логичная мысль - выбросить пакет нужно как можно раньше.

Как было бы здорово, если бы мы могли это сделать на входной плате.

Input Queuing

Более удачным вариантом оказывается буферизировать пакеты на входной плате после лукапа, когда уже становится понятно, куда пакет слать. Если выходной интерфейс заведомо занят, то и смысла гнать камикадзе на фабрику нет.



Постойте! Как же входной чип узнает, что выходной интерфейс не занят?

С точки зрения Data Plane никакой обратной связи, от выходного чипа входному, очевидно, нет. Распространение между ними информации, необходимой для лукапа (некстхопы, интерфейсы, заголовки) производится средствами медленного Control Plane - тоже не подойдёт.

Так вот для сигнализации такой информации между линейными платами появляется арбитр. У разных вендоров он может быть реализован по-разному, но суть его в следующем - входной чип регулярно запрашивает у выходного разрешение на отправку нового блока данных. И пока он его не получит - держит пакеты в своём буфере, не отправляя их в фабрику.

Соответственно выходной чип, получив такой запрос, смотрит на утилизацию выходного интерфейса и решает, готов ли он принять пакет. Если да - отправляет разрешение (**Grant**).

Это на первый взгляд контринтуитивное поведение - каковы же накладные расходы на такой арбитраж, насколько это увеличивает задержки, если на отправку пакета данных нужно дождаться RTT в пределах коробки - пока запрос улетит на выходной чип, пока тот обработает, пока ответ вернётся назад.

Тут некоторые платформы оптимизируют: request/grant'ы присобачиваются к data-пакетам piggyback'ом.

Итога вместо data1 $\rightarrow$ request2 $\rightarrow$ data2 $\rightarrow$ request3 получается data1+request2 $\rightarrow$ data2+request3.

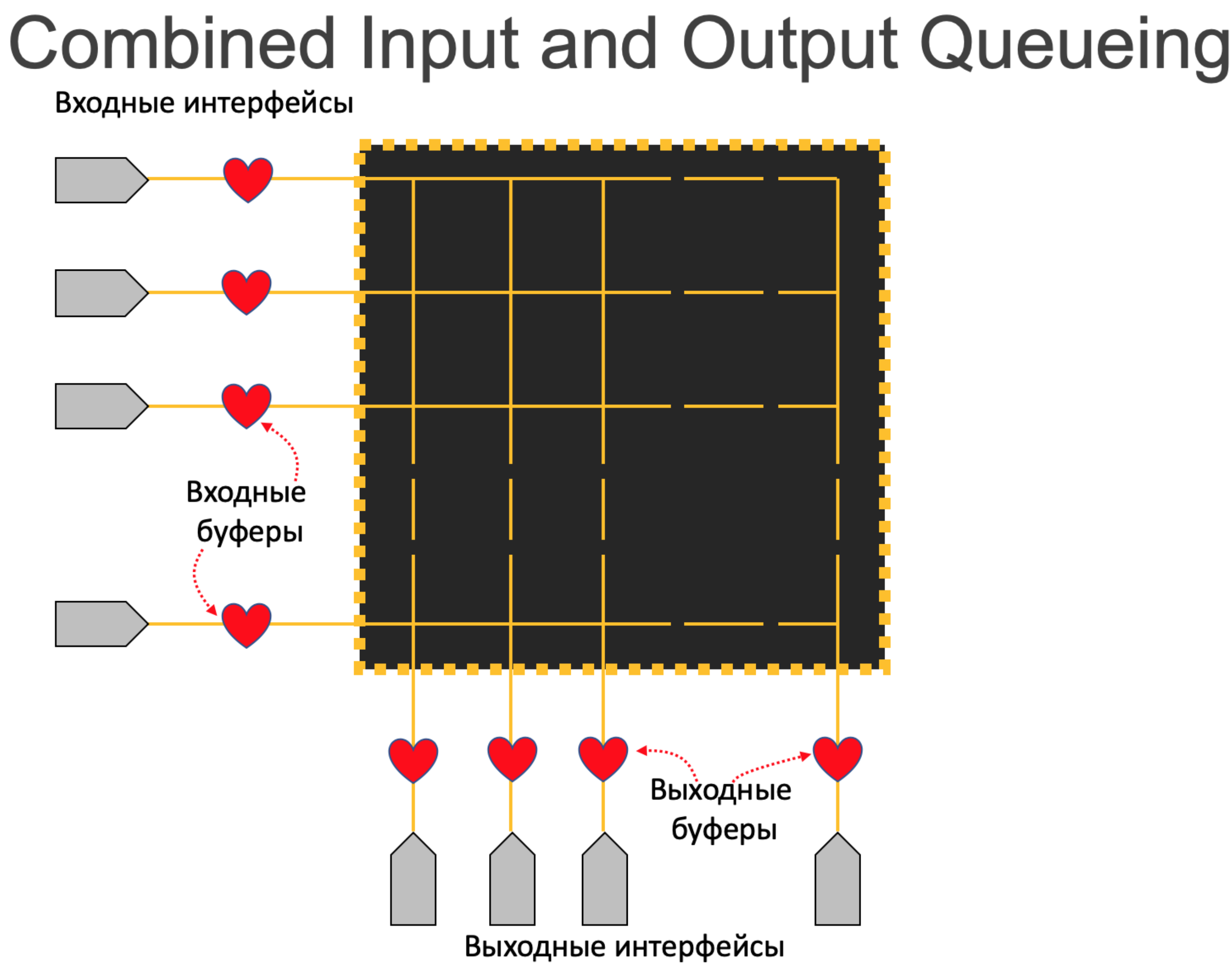
В любом случае тут для меня начинается область магического искусства, но вендоры эту революцию успели совершить и есть масса платформ, на которых арбитр прекрасно со своей задачей справляется.

Хотя обычно он применяется не для Input Queueing в описанном виде.

Дело в том, что эффективность Input Queueing не очень высокая - очень часто придётся ждать, пока интерфейс освободится. Эх, прям вспоминается старый добрый Ethernet CSMA/CD.

Combined Input and Output Queueing

Гораздо выгоднее в этом плане разрешить буферизацию и на выходе. Тогда арбитр будет проверять не занятость интерфейса, а степень заполненности выходного буфера - вероятность, что в нём есть место, гораздо выше.

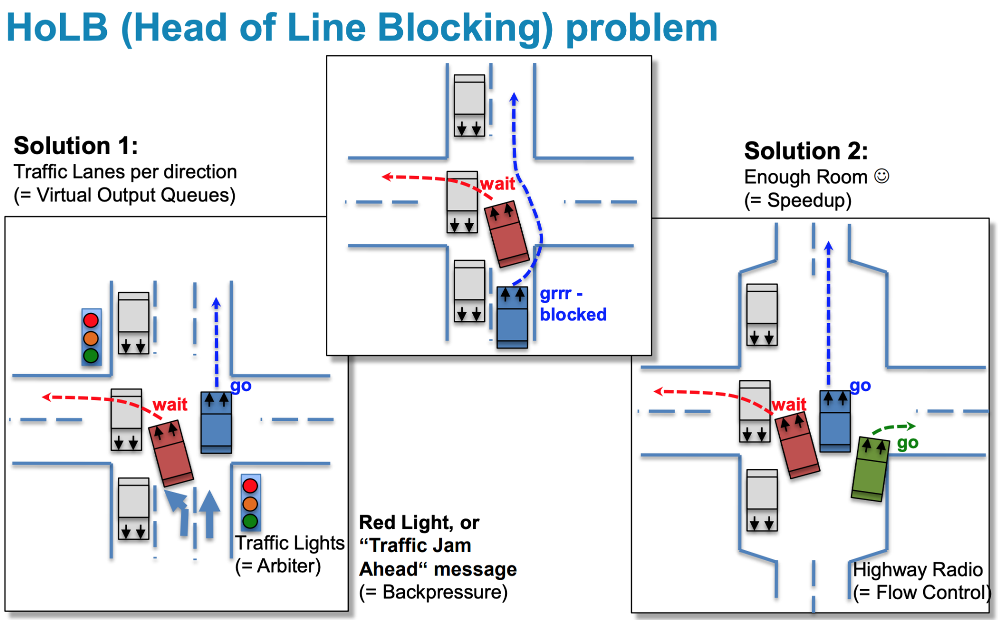


Но такие вещи не даются даром. Очевидно, это и увеличенная цена из-за необходимости реализовывать дважды буферизацию, и увеличенные задержки - даже в отсутствие заторов этот процесс не бесплатный по времени.

Кроме того, для обеспечения QoS придётся хоть какой-то минимум его функций реализовывать в двух местах, что опять же скажется на цене продукта

Но у CIOQ (как и у IQ) есть фундаментальный недостаток, заставивший в своё время немало поломать голову лучшим умам - **Head of Line Blocking**.

Представьте себе ситуацию: однополосная дорога, перекрёсток, машине нужно повернуть налево, сквозь встречный поток. Она останавливается, и ждёт, когда появится окно для поворота. А за ней стоит 17 машин, которым нужно проехать прямо. Им не мешает встречный поток, но им мешает машина, которая хочет повернуть налево.



Источник.

Этот избитый пример иллюстрирует ситуацию HoLB. Входной буфер - один на всех. И если всего лишь один выходной интерфейс начинает испытывать затор, он блокирует полностью очередь отправки на выходном чипе, поскольку один пакет в начале этой очереди не получает разрешение на отправку на фабрику.

Трагическая история, как в реальной жизни, так и на сетевом оборудовании.

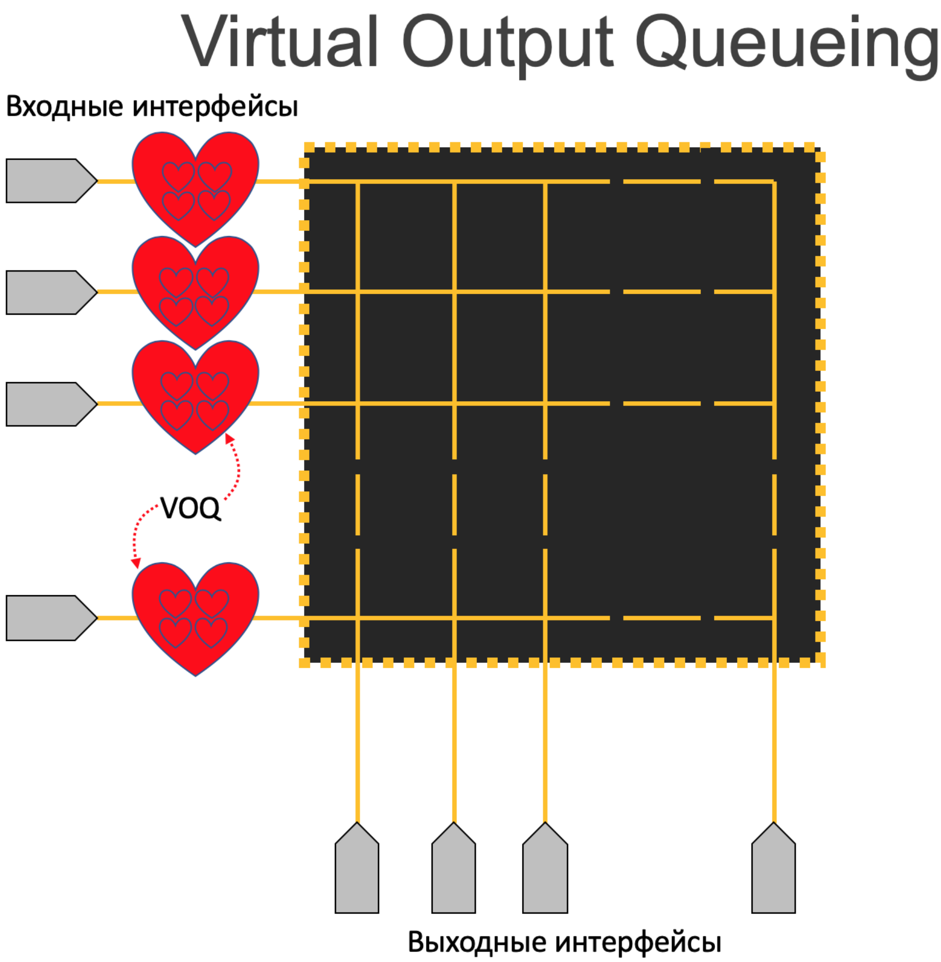
Virtual Output Queueing

Как можно исправить эту дорожную ситуацию? Например, сделав три полосы - одна налево, другая прямо, третья направо.

Ровно то же самое сделали разработчики сетевого оборудования.

Они взяли входной буфер побольше и подробили его на множество очередей.

Для каждого выходного интерфейса они создали по 8 очередей на каждом чипе коммутации. То есть перенесли все задачи по обеспечению QoS на входной чип. На выходном же при этом остаётся самая базовая FIFO очередь, в которой никогда не будет заторов, потому что их контроль взял себя входной чип.



Если взять грубо коробку со 100 интерфейсами, то на каждой плате в буферах нужно будет выделить 800 очередей.

Если в коробке всего 10 линейных карт, то общее число очередей на ней будет 100*8*10 = 8000.

Однако V в VOQ означает виртуальный, не потому, что они как бы выходные, но на самом деле находятся на входных платах, а потому что Output Queue для каждого выходного интерфейса распределён между всеми линейными картами. То есть сумма 10и физических очередей для одного интерфейса на 10 чипах составляет 1 виртуальную.

Собственно из-за распределённого характера этой виртуальной очереди от арбитра и здесь избавиться не получится - разным входным чипам всё же нужно знать, состояние выходной очереди. Поэтому даже несмотря на то, что выходная очередь - это FIFO, выходной чип всё ещё должен давать добро на отправку трафика.

Кстати, что касается трафика, который должен вернуться в интерфейс той же карты, на которую он пришёл изначально, то здесь никаких исключений - он томится в VOQ, пока чип не даст добро переложить его в выходную очередь. С тем только отличием, что пакет не будет отправляться на фабрику. Поэтому перед лицом перегрузок все равны.

Combined Input and Output Queueing w/ Virtual Output Queueing

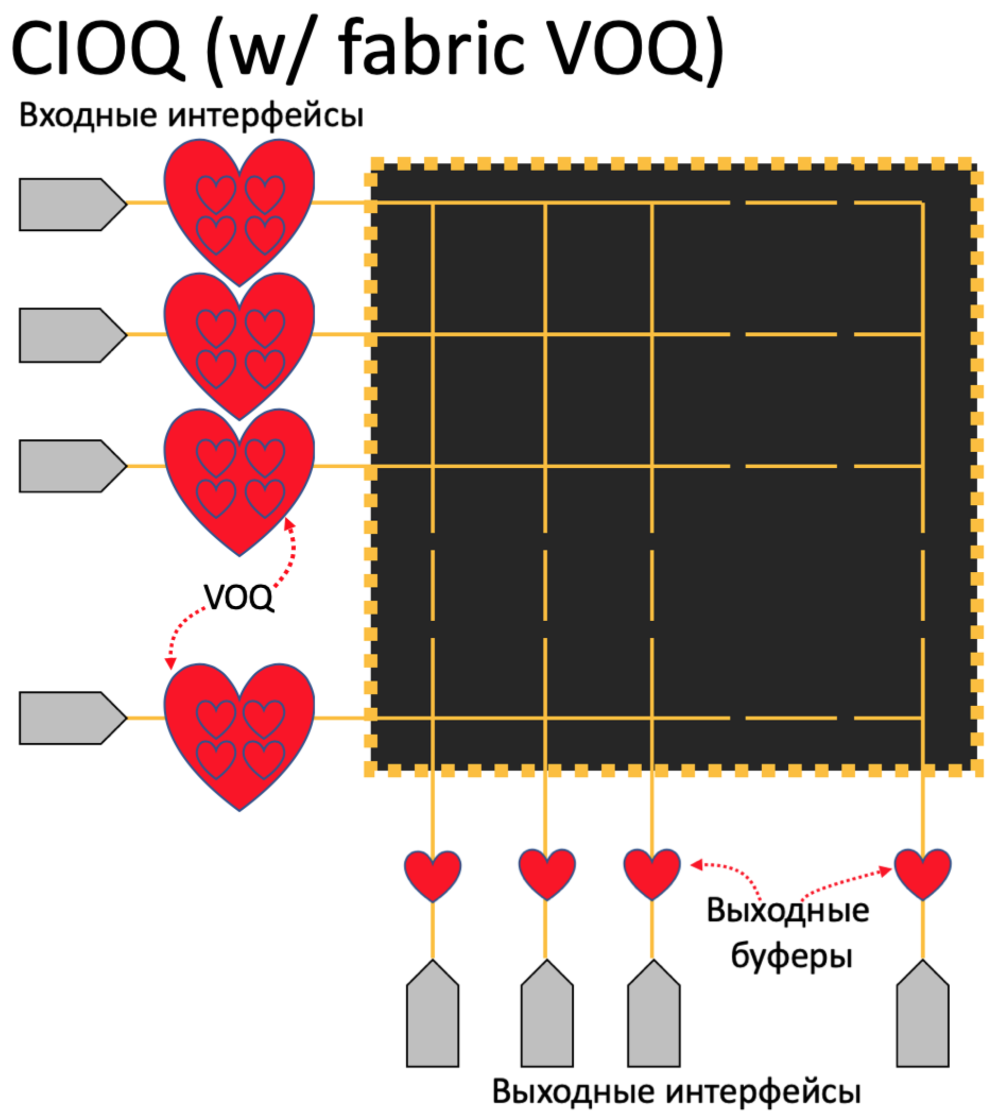
Как сделать еще сложнее? Совместить обе вышепреведённые техники буферизации. VoQ применяется на фабрике, то есть только для виртуальных портов подключающих Switch Fabric к Egress Line Card NPU, таких портов в сторону NPU относительно немного и виртуальная очередь (VoQ) всё так же находится на Ingress Line Card.

Таким образом у каждого Ingress NPU есть виртуальные очереди для каждого из Egress NPU, таких очередей может быть несколько.

После того как трафик прошёл через фабрику минуя fabric VoQ, пакет в любом случае попадёт в очередь на исходящем интерфейсе (Egress Interface Output Queue) и или задержится там если в данный момент интерфейс перегружен, или сбросится если очередь заполнена полностью, или отправится дальше в исходящий интерфейс.

Данная техника не подвержена HoLB эффекту в той же мере что и классическая CIOQ, однако в случае когда трафик идёт на два разных исходящих интерфейса за одним NPU в случае затора на порту фабрики проблема HoLB всё таки может присутствовать.

CIOQ w/ VoQ fabric масштабируется гораздо лучше чем E2E VoQ из-за меньшего количества интерфейсов, а следовательно и очередей, поэтому подходит для PE, BNG и другого оборудования с большим количеством интерфейсов. Однако, недостаток двойной буферизации никуда не делся, как и недостаток размещения памяти в двух местах и как следствие бОльшего потребления энергии.



На сегодняшний день End-to-End VOQ является наиболее прогрессивной технологией, но говорить о её безоговорочной победе пока не приходится. Картинка с NANOG65 (2015):

**System characteristic
vs. Queuing architecture**

	CIOQ	CIOQ (w/ fabric VOQ)	E2E VOQ
Low residency time	✖	✖	✔
High load	✖	✔	✔
Low power footprint	✖	✖	✔
High number of interfaces (each with independent queuing. E.g. BNG, BE)	✔	✔	✖
multi-chassis systems	✔	✔	✔
Examples*	C7500, Early C7600	Juniper MX, Cisco ASR9k, CRS-X ALU 7750/7950*	Juniper PTX Cisco NCS600

Дальнейшее чтиво:

- An Update on Router Buffering
- What is VOQ and why you should care
- Strategies of packet buffering inside Routers
- Understanding CoS Virtual Output Queues (VOQs) on QFX10000 Switches
- High Performance Switches and Routers - если у вас есть лишних 14 к₽.

Shallow vs Deep Buffers

Буферы - это то место, где пакеты можно похранить, вкачав в них смертельную порцию задержки.

Как сказали в видео Packet Pushers - буферы - это религия. Хотя, скорее всего, неортодоксальная, а возможно даже секта.

Чуть позже мы поговорим о том, что такое хорошо, а что такое плохо. А пока посмотрим на реализации.

Shallow - неглубокие - это буферы размером до 100МБ. Обычно это встроенная в кристалл **on-chip** память - **OCB** - On-Chip Buffer. **Deep** - счёт уже идёт на гигабайты. Обычно **off-chip** и подключается к чипу по отдельной шине. И нет ничего посередине.

За последние лет десять производительность чипов выросла на порядки, трафика они теперь перемалывают терабиты в секунду вместо единиц гигабит. А размер памяти не то что не поспевает за этим ростом, он фактически почти стоит на месте.

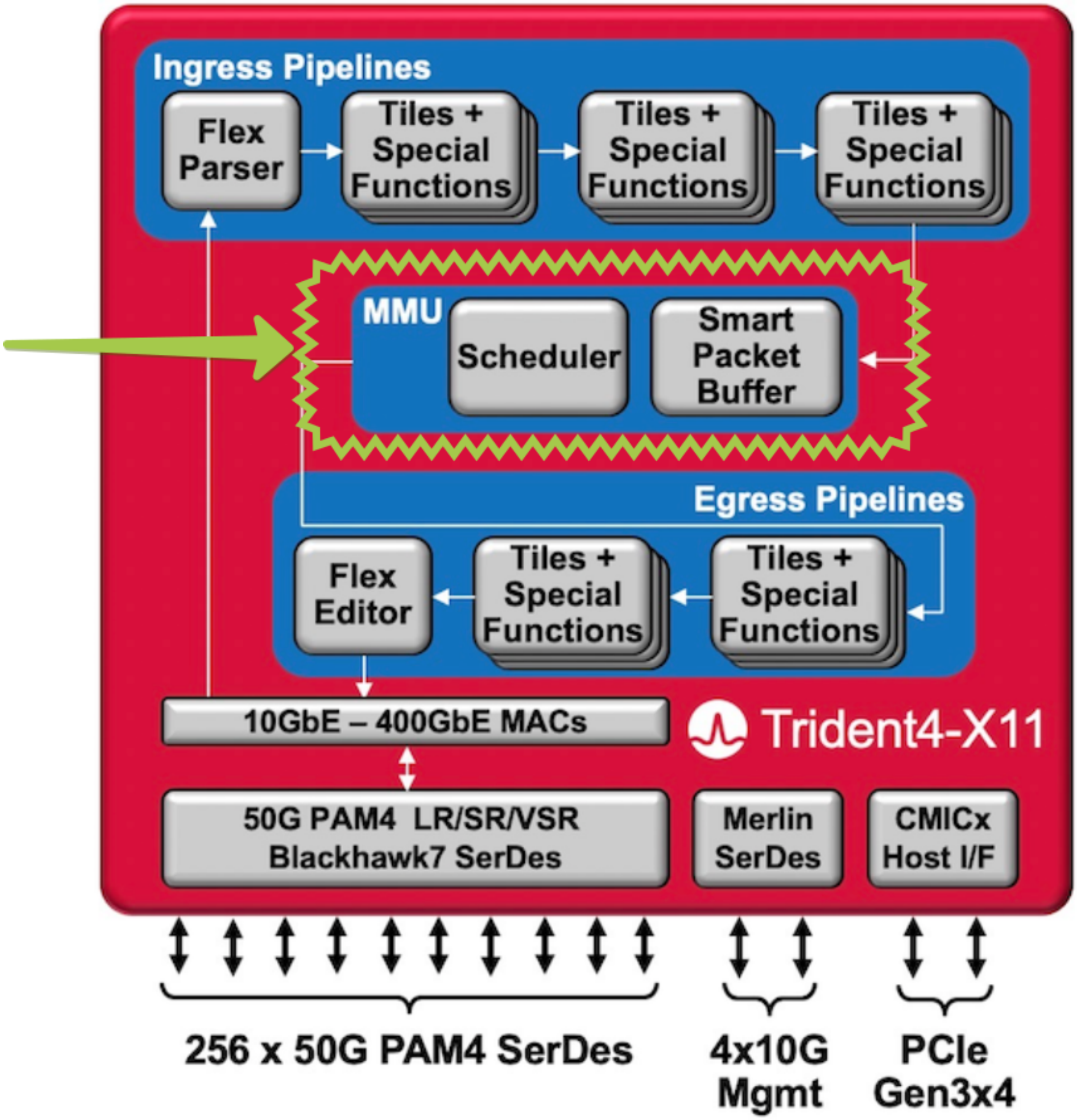
Давайте грубо прикинем: если для гигабитного порта буфер размером 16 мегабайт мог абсорбировать всплеск трафика длительностью примерно 100 мс, то для 100Гб/с - всего лишь 1мс. И это только один порт, фактически же плотность портов тоже растёт и максимальная комплектация для одночипового устройства сегодня - 64 порта 400Гб/с - или 25,6 Тб/с полосы пропускания.

Используя только 64 МБ буфер, такой чип сможет хранить трафик 0.000005 с или 5 мкс.

Такие буферы порой даже называют Extremely shallow buffers.

Их воистину миниатюрный объём обусловлен в первую очередь тем, что они в прямом смысле встроены в чип. Такая память является составной частью микросхемы, и каждый дополнительный мегабайт, разумеется, будет обходиться в лишнюю тысячу долларов, больший размер и тепловыделение. Для справки Broadcom Trident 4 содержит 21 миллиард транзисторов, изготовленных по 7нм техпроцессу (это меньше размера капсида любого существующего вируса) на нескольких квадратных сантиметрах.

Логично вытекающим следствием является скорость работы с этой памятью - она должна соответствовать производительности чипа.



Очевидно, что не для всех задач такие маленькие буферы подходят. В частности модульные коробки с VOQ явно не могут позволить себе дробить 64 Мб на несколько тысяч очередей (*на самом деле могут*).

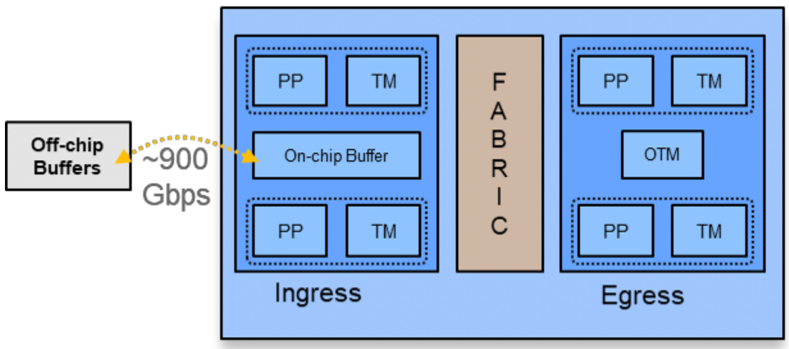
Поэтому рынок предлагает решения с большой внешней памятью (Deep Buffers), размер которой начинается от 1ГБ (обычно от 4ГБ).

Согласно этой таблице существуют коммутаторы (Arista 7280QR-C48) с фантастическими

32-хгигабайтовыми буферами - это уже все сезоны Рика и Морти в неплохом качестве. Но это уже история про | VOQ - всё-таки это память не одного чипа. На моём первом ПК такого объёма был жёсткий диск.

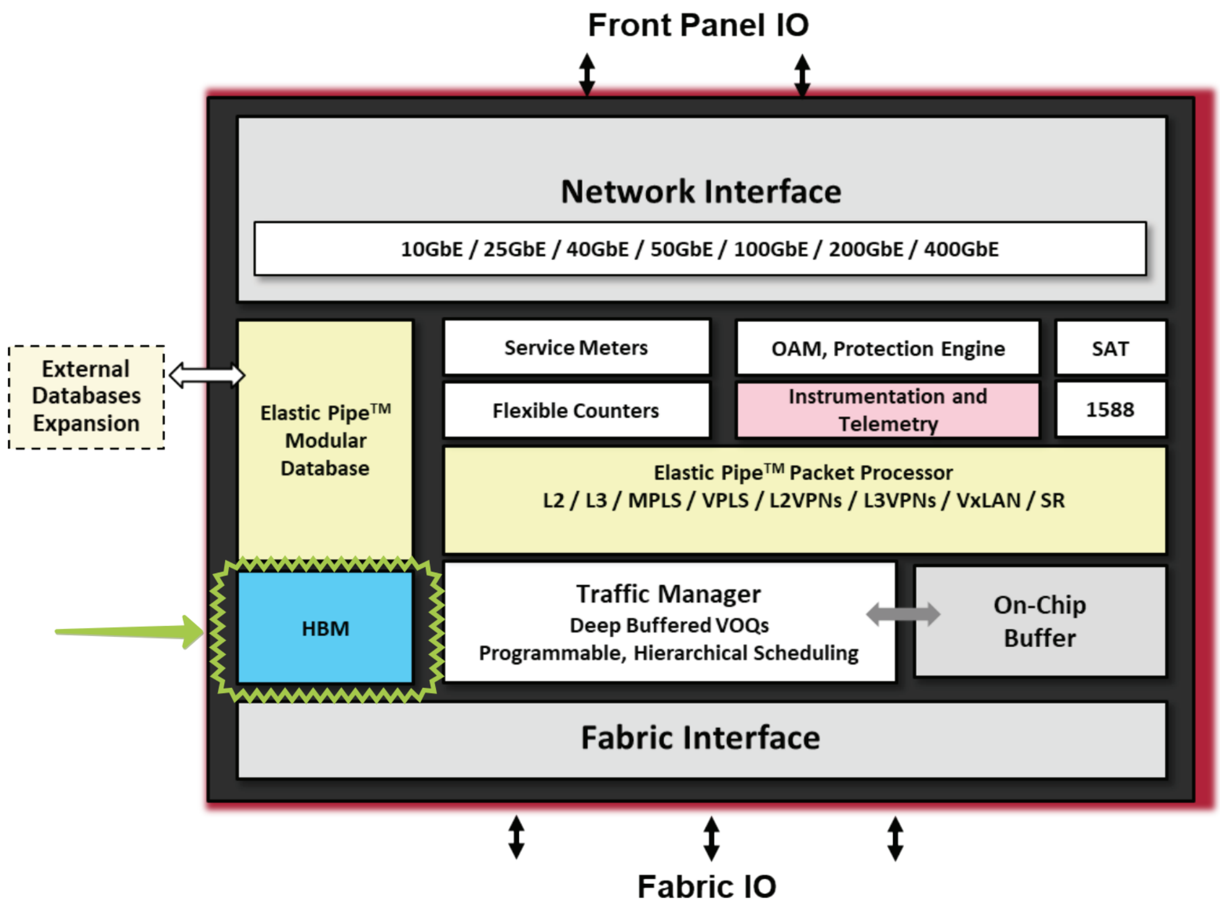
Как такая память реализована зависит уже от чипа и коробки.

Например, Broadcom **Jericho+** сгружает пакеты во внешнюю память в размере 4ГБ. Это обычная широко известная **GDDR5**, использующаяся в видеокартах.

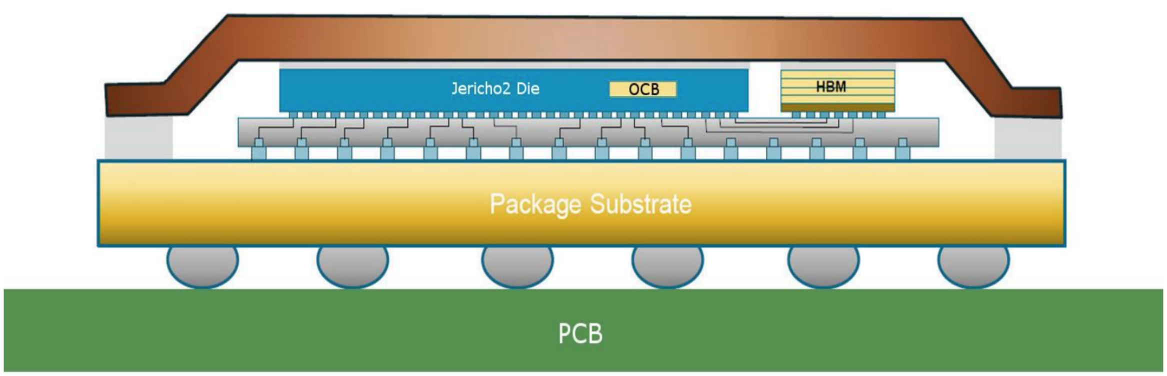


Источник

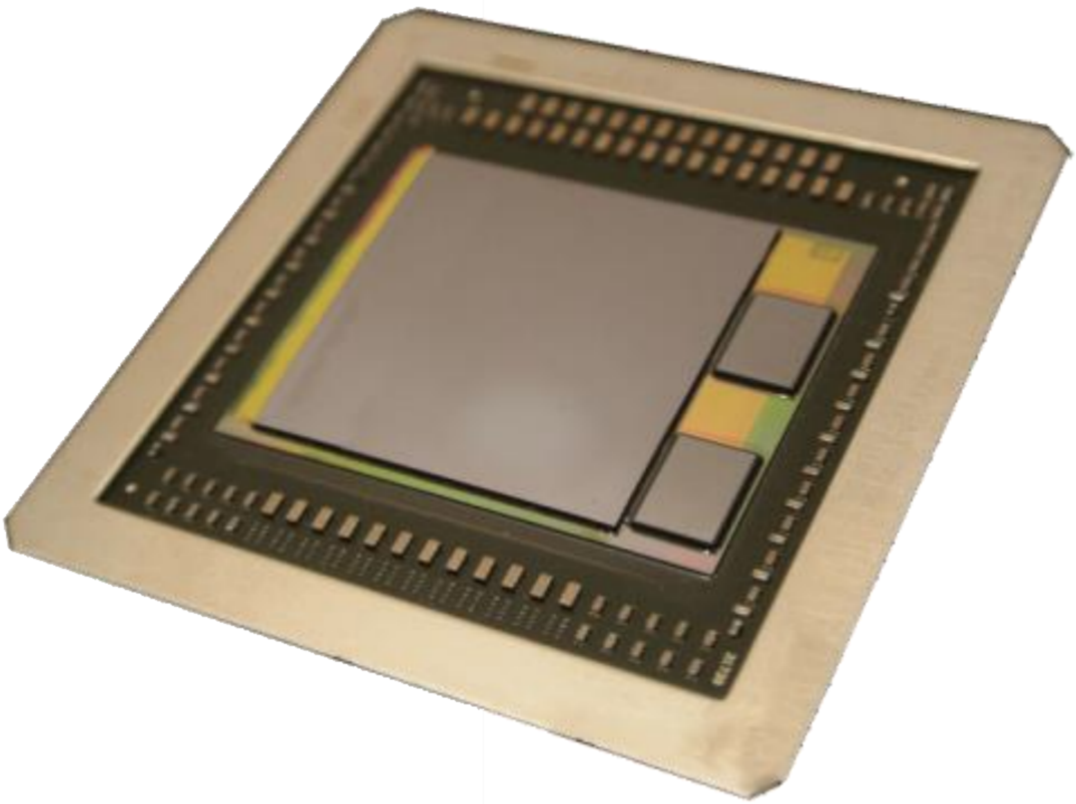
Jericho2 несёт на борту новейшую память **HBM2** - High Bandwidth Memory - размером 8ГБ.



Источник



А вот и фото Jericho2:

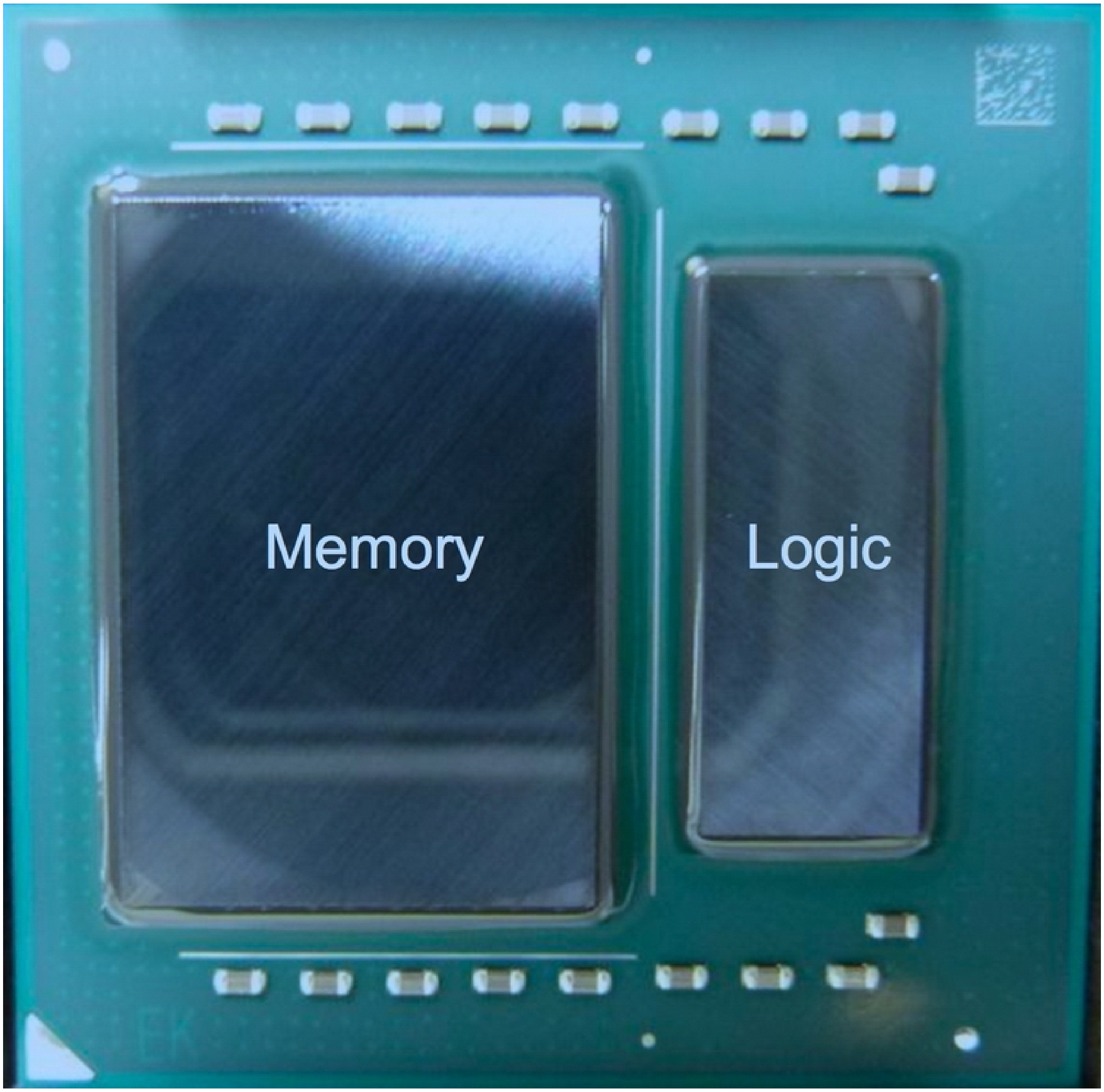


Источник

Juniper PTX и QFX10000 используют чип **Q5** собственного производства с внешней памятью - **HMC** - Hybrid Memory Cube - в размере 4ГБ.

Источник

А вот так выглядит сетевой процессор Cisco с внешней памятью:



Источник

Перечислять можно и дальше.

Что здесь важно отметить, что внешняя память тоже не даётся бесплатно. Во-первых, цена таких решений значительно выше. Во-вторых, пропускная способность обычно ниже. И основное ограничение - канал между чипом коммутации и чипом памяти. Для производимых массово чипов, вроде GDDR5 полоса не превышает 900ГБ в режиме half-duplex. Но это чип, явно не заточенный под задачи сетевых сервисов.

Кастомный джуниперовский HMC обещает 1,25 Тб/с в обоих направлениях.

Если верить вики, то HBM 2-го поколения, используемый в последнем чипе Broadcom Jericho2, выдаёт порядка 2Тб/с.

Но это всё ещё далеко от реальной производительности сетевого ASIC. Фактически шины до этой внешней памяти является узким местом, которое и определяет производительность чипа.

Important:

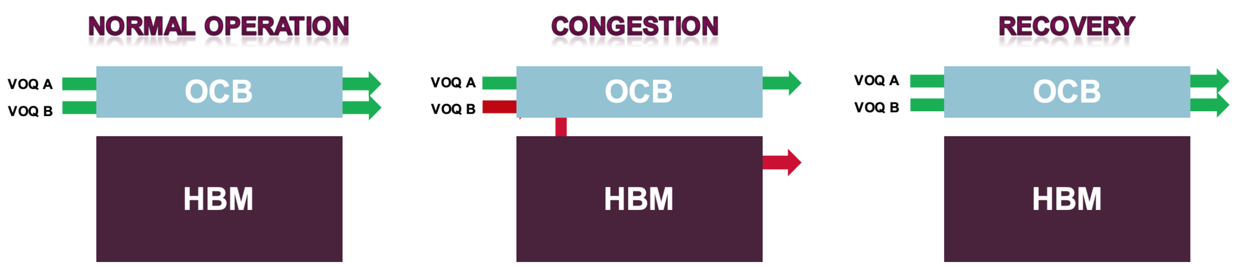
Когда-то мир был лучше и было строгое разделение - Shallow Buffer - это встроенная On Chip память, Deep Buffer - внешняя.

С развитием WLP ситуация начинает меняться. Память HBM становится co-packaged в один чип вместе с комутационным асиком. TSV и 3D Advanced Packaging значительно увеличивают пропускную способность. И нередко в перезентациях вендорово можно увидеть "Deep Buffer" и "On Chip" в одной фразе.

Тут нужно быть осторожным, поскольку шина между асиком и памятью, пусть даже они расположены рядышком на одном интерпозере под общей крышкой, всё ещё является узким местом и ограничивает максимальную пропускную способность.

Hybrid Buffering

Поэтому почти все вендоры сегодня практикуют **гибридную буферизацию**, или, если хотите - динамическую. В нормальных условиях используется только on-chip память, предоставляющая line-rate производительность. А в случае перегрузки пакеты автоматически начинают буферизироваться во внешней памяти. Это позволяет уменьшить стандартные задержки, энергопотребление и в большинстве случаев вписаться в ограниченную полосу пропускания до памяти.



Источник

Note:

Данный параграф отменяет сказанное выше о том, что on-chip памяти не хватит для VOQ. Фактически в случае гибридной буферизации она всё же дробится на тысячи очередей очень маленькой длины, чтобы обеспечить VOQ. Просто в нормальных условиях этой длины хватает, чтобы пропускать трафик мимо внешней памяти.

При этом в первую очередь начнёт офлоадиться на внешнюю память массивный трафик, идущий в низкоприоритетных очередях, а требовательный к задержками будет по-прежнему пролетать фаст-пасом.

Большие буферы - добро или зло?

В целом это довольно старая дилемма. Подольше похранить или пораньше дропнуть?

Конечно, всем приложениям хочется lossless low-latency сеть. Даже жирному некрасивому торренту. Но так не бывает и кем-то нужно жертвовать.

И мы долгое время живём с инертной мыслью , что часть приложений могут потерпеть задержки, а вот терять трафик совсем не хочется. Не в малой степени этому способствовало и то, что потери - это измеримая характеристика с более или менее понятными границами - потерь быть не должно. А что задержка? Вроде можно чётко сказать, что единицы миллисекунд - это хорошо, а секунды - это плохо. А между ними - зона спекуляций. Как оценить влияние вариаций задержки для рядового TCP-трафика?

Поэтому и спрос на устройства с большими буферами есть - никто не хочет терять трафик.

А теперь я выскажу не самое популярное мнение - потери - это хорошо.

Так уж вышло, что один из транспортных протоколов, фиксирует перегрузки, опираясь на потери.

Дроп в очереди на сетевом устройстве означает, что на нём случился затор - это он не по своему капризу. И будет совсем не лишним, если отправители немного приуменьшат свои congestion window.

Именно так и работают все классические (и не очень) реализации TCP Congestion Control.

Соответственно на устройствах с глубокими буферами во время заторов пакеты будут долго копиться, не отбрасываясь. Когда они всё-таки дойдут до получателя и тот их ACKнет, отправитель не только не снизит скорость, но может даже её увеличить, если у него сейчас режим Slow Start или Congestion Avoidance.

Можно взглянуть и дальше: растущая очередь взвинчивает RTT, что соответственно влечёт за собой увеличение RTO таймеров на отправителях, тем самым замедляя обнаружение потерь.

То есть сеть лишается своего практически единственного инструмента управления перегрузками.

И таким образом архитекторы, пытающиеся решить вопрос заторов на сети путём увеличения буферов, усугубляют ситуацию ещё больше.

Ситуация, описанная выше, называется **bufferbloat** - распухание буфера.

Википедия иронично замечает:

> Проект www.bufferbloat.net иронично определил этот термин, как «ухудшение производительности Интернета, вызванное предыдущими попытками её улучшения»

Отбросы - санитары сети. Ко всеобщему удивлению, уменьшение очереди до одного пакета зачастую может кардинально улучшить ситуацию, особенно в условиях датацентра (только не сочтите это за дельный совет).

Note:

Справедливости ради следует заметить, что современные реализации TCP - BBR2, TIMELY ориентируются не только и не столько на потери, сколько на RTT и BDP.

Гугловый QUIC - надстройку над UDP - следует отнести сюда же.

Внутри фабрики датацентра RTT ультракороткий - зачастую меньше 1 мс. Это позволяет среагировать на потерю очень быстро и купировать перегрузку в её зачатке.

Собственно поэтому практически все ASIC'и для датацентровых коммутаторов имеют только крохотную on-chip память.

Хотя и появилась в последние годы тенденция к глубоких буферам и тут.

И этому даже находится объяснение.

Особая история на границе датацентра (или на устройствах доступа в сети провайдера или на магистральных сетях).

Во-первых, это места, которые обычно заведомо строятся с переподпиской, поскольку WAN-линки дорогие, что автоматически означает, что ситуации, в которых трафика приходит больше, чем способен переварить интерфейс, ожидаемы. А значит нужна возможность пакеты хранить и обрабатывать их в соответствии с приоритетами. Большие буферы позволяют сгладить всплески.

Во-вторых, обычно приложения настолько чувствительные к задержкам, никто не будет пытаться растягивать на этот сегмент. Например, RoCE или распределённое хранилище. Для чуть менее чувствительных, таких как телефония, в больших буферах выделяется приоритетная очередь.

В-третьих, тут задержки на устройстве всё ещё делают основной вклад в общее время доставки, но уже не настолько драматический.

Итак, устройства с большим объёмом памяти годятся в места где заложена переподписка или могут появиться заторы.

Что стоит отметить, так это то, что в датацентрах тоже есть ситуации, в которых 16-64 МБ буферов может не хватить, даже несмотря на отсутствие переподписки.

Два типичных примера - это обработка Big Data и Storage.

Анализ Big Data. Кластера Map-Reduce - это сотни и тысячи машин, которые перемалывают параллельно огромные массивы данных по заданию Master-узла, заканчивают примерно одинаково и все разом начинают возвращать ответы на Master-узел. Ситуация называется Incast. Длится она порядка нескольких десятков миллисекунд и потом исчезает.

On-chip память неспособна вместить эти данные - значит будет много дропов, значит ретрансмиты, значит общее снижение производительности.

Storage. Это штука крайне чувствительная к потерям и тоже гоняющая массивные объёмы данных. В случае хранилки тоже лучше не терять ничего. Но обычно она при этом и к задержкам предъявляет строгие требования, поэтому такие приложения обсудим пониже.

Однако при этом крайне редко они единственные потребители сети в датацентрах, другим приложениям нужна низкая задержка.

Впрочем, это легко решается выделением очередей QoS с ограничением максимальной доступной глубины. И весь вопрос заключается тогда только в том, готова ли компания заплатить за глубокие буферы, возможно, не использовать их и поддерживать конфигурацию QoS.

Но в любой ситуации лучше следовать правилу: use shallow ASIC buffers when you can and use deep buffers when you must.

Критика глубоких буферов:

- Arista’s Big Buffer B.S.
- Incast
- Speeding Applications in Data Center Networks. The Interaction of Buffer Size and TCP Protocol Handling and its Impact on Data-Mining and Large Enterprise IT Traffic Flows

Кстати, показательная таблица типичных задержек:

Duration of Latency	Common Sources
100s of nanoseconds	Nominal latency of ultra-low-latency cut-through switches
microseconds	10G serialization delay, nominal latency of single-chip switches
10s of microseconds	Data Center RTT, Modular routers, 1G serialization
1s of milliseconds	Campus or Metro RTT
10s of milliseconds	Cached CDN RTT, intra-country RTT
100s milliseconds	Inter-continent RTT
1s of seconds	Congested DSL home router or mobile device

Источник

7

6. Low-latency lossless сети

С развитием RoCE, RDMA, nVME over Fabric к сети стали появляться другие, неслыханные доселе требования: и каждый пакет ценен и времени терять нельзя. Одновременно. И хуже того, мы хотим по максимуму утилизировать имеющиеся линки, чтобы они не простаивали.

И все требования оправданы - осуждать их мы здесь не будем.

Так что же можно сделать? На самом деле, как я уже говорил выше, условия сети датацентра хороши тем, что информация о перегрузке очень быстро доносится до отправителей. Поэтому и реакция на потерянный пакет не заставит себя долго ждать.

Но что если нам не отбрасывать пакет, а каким-то другим способом сообщить отправителю, что нужно замедлиться?

Здесь есть несколько конкурирующих подходов.

- *ECN-Based*
- *Bandwidth-Delay Product Based*
- *RTT-BASED*

ECN-Based

Транзитное устройство, готовящееся испытать перегрузку, явным образом сообщает отправителям о том, что нужно охладить свою страсть.

Прозорливые инженеры заложили в IP целых 8 бит под QoS, и только 6 мы задействовали под DSCP, а 2 бита были зарезервированы для целей **ECN - Explicit Congestion Notification**.

Надолго забытый механизм сегодня извлекают из ящика стола, сдувают с него пыль, и вставляют в шкатулку, на которой нацарапано или DCTCP

Транзитное устройство при заполнении буфера больше, чем до определённого порога, выставляет в заголовках IP обрабатываемых пакетов бит CE (Congestion Encountered) и отправляет пакет дальше.

Получатель, увидев в пришедшем пакете этот флаг, сообщает отправителю о перегрузке и о том, что нужно снизить скорость.

Классический TCP может обнаружить только уже существующую перегрузку, а DCTCP, используя ECN, узнаёт о том, что она только приближается, и пробует её избежать.

Есть и другие реализации TCP, поддерживающие ECN, например, HTCP.

Нюанс с ECN-based Congestion Control механизмами в том, что до поры до времени они ничего не знают о надвигающейся перегрузке, а потом должен пройти ещё целый RTT, чтобы отправитель узнал, что какое-то транзитное устройство к ней близко. | К тому времени, как отправители начнут снижать скорость, перегрузка уже может или рассосаться или наоборот дойти до уровня, когда начнутся дропы.

Bandwidth-Delay Product Based

Другие реализации замеряют эффективную полосу сети совместно с RTT, то есть сколько можно ещё напихать в трубу до того, как это создаст затор и увеличит задержку.

Примерами таких протоколов являются BBR и H-TCP.

RTT-BASED

В конце концов есть элегантные механизмы, которые замеряют время прохода трафика туда-обратно.

Идея провальная для MAN/WAN-сегментов, и, честно говоря, при попытке программной вычисления RTT тоже.

TIMELY от Google с аппаратным offload'ом вычисления RTT один из наиболее удачных примеров.

На самом деле, если бы не видео с прекрасной девушкой, рассказывающей про технические детали TIMELY, не знаю даже стал ли бы я упоминать про него. Наслаждайтесь, но берегите уши: TIMELY: RTT-based Congestion Control for the Datacenter.

8

7. Существующие чипы

До начала этого разговора следует сказать о разделении:

- **Custom/In house Silicon**. Это то, с чего всё начиналось - чипы, разработанные внутри R&D вендоров сетевых устройств: Cisco, Juniper итд. Используются они только внутри своего же оборудования и не продаются наружу.
- **Commodity/Merchant Silicon**. Чипы широкого производства. Можно их назвать рыночными или коммерческими. Это чипы, производимые сторонними компаниями: Broadcom, Innovium, Barefoot, Marvell итд.

Commodity/Merchant Silicon

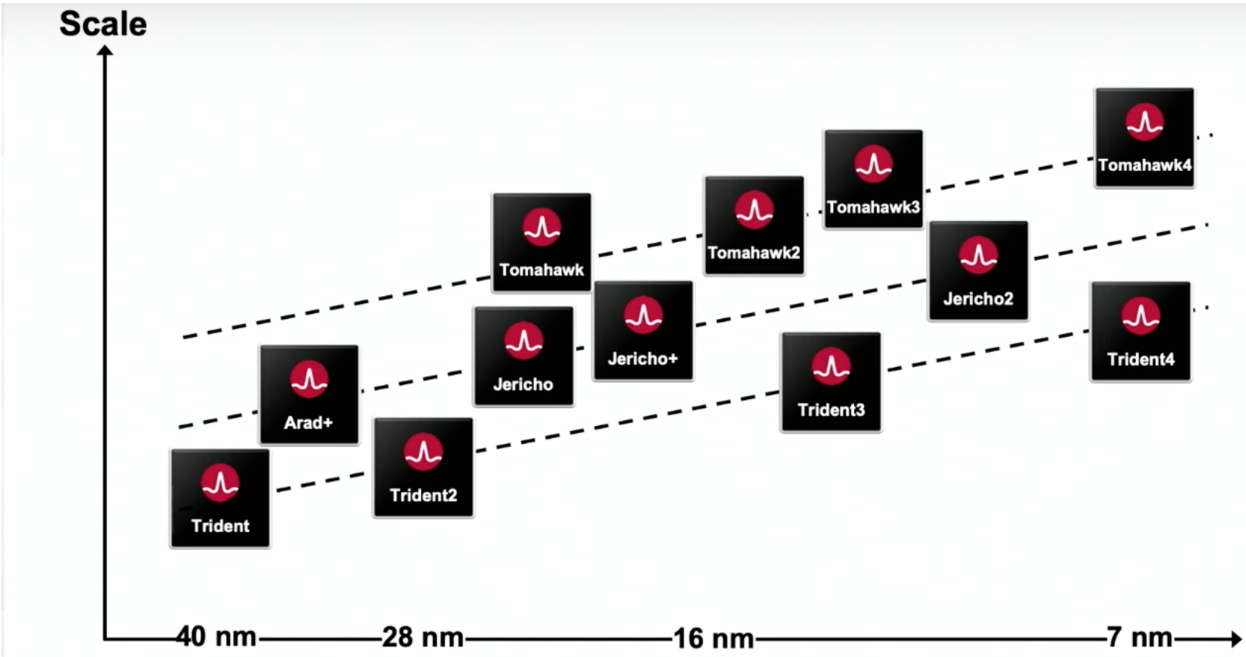
Многие утверждают, что за рыночными чипами будущее (если уже не настоящее). Поэтому давайте с ними и познакомимся сначала.

Broadcom

Broadcom делает уйму разных чипов уже тыщу лет.

Исторически он был одним из первых вендоров, кто начал производство высокоскоростных ASIC для сетевых устройств.

В 2010-м году они выпустили свой чип Trident 64x10G, а в 2020 они начнут поставлять 64x400G.


Scale
Tomahawk4
Tomahawk3
Tomahawk2
Tomahawk
Jericho2
Jericho+
Jericho
Arad+
Trident4
Trident3
Trident2
Trident
40 nm
28 nm
16 nm
7 nm

Все сетевые ASIC Broadcom принадлежат двум семействам: StrataXGS и Strata DNX.

StrataXGS - shallow-buffer чипы преимущественно для датацентровых коммутаторов. Названы в честь ракет.

Это семейство делится на:

Многофункциональные: Trident, Trident2, Trident2+, Trident3, Trident4, Maverick.

Высокоскоростные: Tomahawk, Tomahawk+, Tomahawk2, Tomahawk3, Tomahawk4.

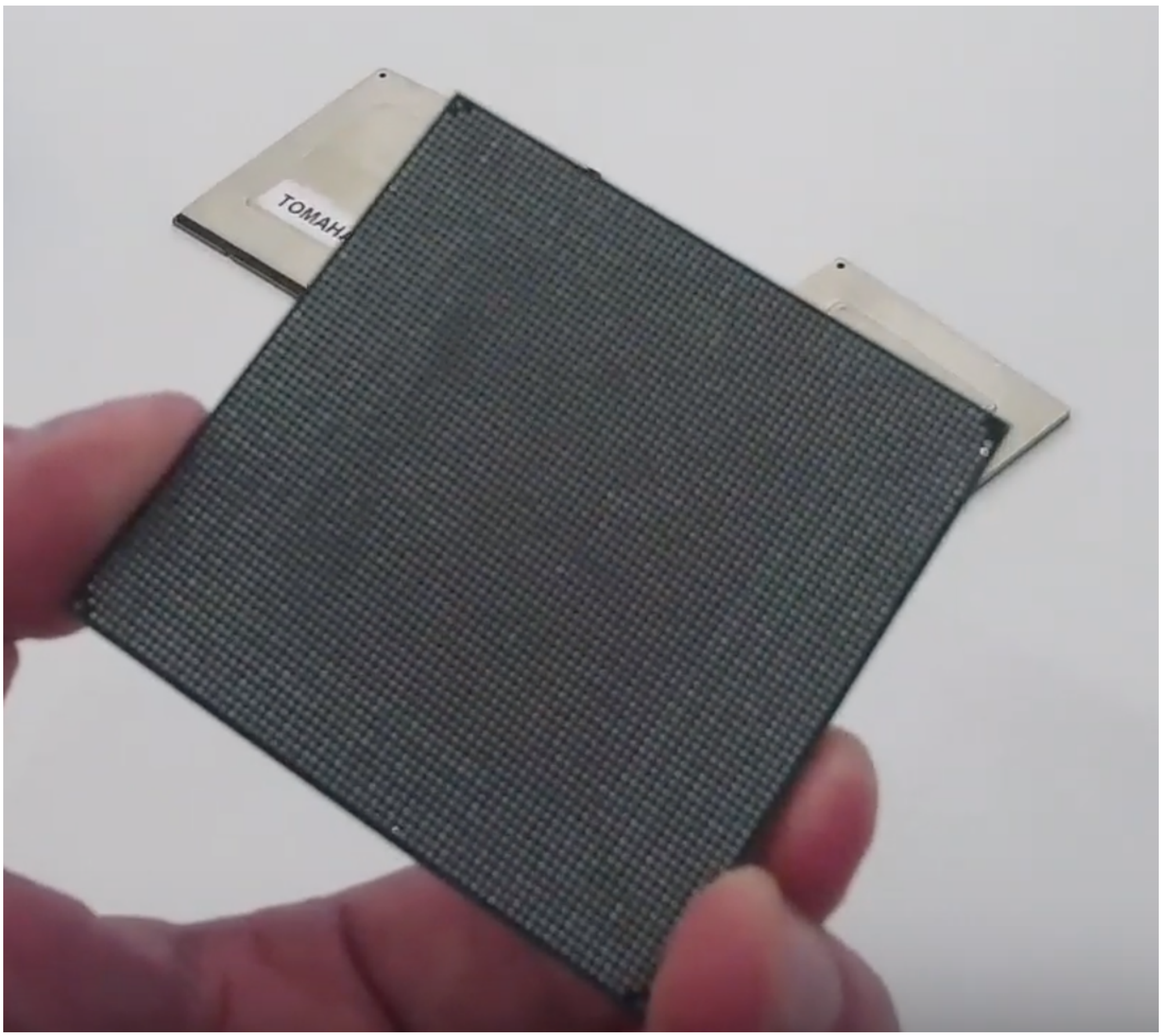
Strata DNX - чипы с глубокими буферами, рассчитанные на маршрутизаторы, модульные коммутаторы. Скорости при этом из всех семейств наиболее низкие. Названы в честь израильских городов: Arad, Qumran, Jericho, Jericho+, Jericho2.

По назначению, грубо говоря, делятся они примерно так:

Trident’ы на роль ToR’ов, где нужно немного интеллекта: EVPN, VXLAN, пожирнее FIB, побольше ACL.

Tomahawk’и на роль Spine’ов - быстро перекладывать много пакетиков из одного интерфейса в другой.

Jericho - граница датацентра, где выход во внешний мир и DCI. Обычно тут не требуются сверхвысокие скорости, потому что основной трафик - это East-West в пределах ДЦ. Зато что здесь требуется, так это весь стек сетевых технологий, большие таблицы и глубокие буферы. VXLAN, MPLS, SR, L3VPN, различные Option’ы и всё прочее, что уже основательно забылось после СДСМ.



Но если уйти за пределы ДЦ в любую сторону - в энтерпрайз, в провайдинг, в операторов, то

Broadcom свои Трезубцы позиционирует уже как универсальные чипы, которые везде и на любом уровне сети сгодятся.

Почти два часа видео весьма технического склада:

- Introduction to Broadcom's Switch Portfolio
- Broadcom Trident4: Disrupting the Enterprise Data Center & Campus
- Broadcom Tomahawk4: Industry's Highest Bandwidth Chip
- Jericho2: Driving Merchant Silicon Revolution
- *Хотел бы я, пожалуй, быть таким седовласым дедком, бегущим по лезвию современных технологий.*

Hint: Кстати, будут у нас в гостях!

Mellanox

Долгое время Broadcom был единоличным властелином всех сердец датацентровых коммутаторов, что позволяло ему диктовать правила игры.

Пока в 2013-м году известный производитель Inifinband-коммутаторов Mellanox не выпустил свой чип **Spectrum** и Ethernet-коммутаторы на его основе. Чип обладал производительностью 3.2 Тб/с и мог обслуживать 32 100Гб порта или 64 порта меньшей скорости.

Это было внезапно.

На сегодняшний день у них продаются свитчи на чипе **Spectrum 2** с мощностью 6,4 Тб/с.

Оба чипа shallow-buffer, рассчитаны на коммутаторы уровней Leaf и Spine - высокая скорость, низкая задержка, не самая богатая функциональность.

Ходят слухи о разработке **Spectrum 3**, от которого ожидается 12,8 Тб/с, что позволит Mellanox'у *почти* догнать Broadcom.

Относить ли Mellanox к числу производителей рыночных ASIC'ов - вопрос в целом дискуссионный, но все делают именно так. Увы (а может и нет), они делает чипы только для своих коммутаторов, и не продают их наружу.

В их пользу говорит то, что Mellanox - это whitebox-коммутаторы, на которое можно устанавливать сторонние операционные системы. И тем самым он участвует в конкурентной борьбе и способствует снижению цен, гонке скоростей и открытости технологий. Мимими.

И кроме того, это так называемая Fabricless компания, которая не имеет заводов, а заказывает изготовление чипов на стороне.

Note: К слову, в 2019-м Мелланокс был куплен Nvidia.

Hint: Кстати, были у нас в гостях.

Barefoot

Совсем молодой игрок на рынке коммерческих ASIC'ов. В 2013-м появился, в 2016-м вышел из тени с чипом Tofino с пропускной способностью 6,5 Тб/с.

Сегодня они готовы продавать уже Tofino 2 - 12,8Тб/с

Чип трудится на коммутаторах Cisco Nexus 3464C, Nexus 34180YC и Arista 7170.

Чипы так же shallow-buffer и рассчитаны на спайны и лифы.

Однако сегодня нельзя просто выйти на рынок и сказать "я лучше Бродкома, купите меня". Нужно что-то предложить.

Barefoot предлагает программируемый Pipeline. Это позволяет, используя специальный язык программирования P4, полностью определять, что будет происходить с пакетом в коммутаторе. Можно написать свою логику без оглядки на существующие стандарты - выбросить Ethernet вообще, заглядывать на любую глубину заголовков, выискивать какие угодно флаги итд. Грубо говоря: если 4 бита со сдвигом от начала пакета в 16 бит равны 0110, то нужно поменять 8 бит со сдвигом 32 бита на 01001001 и отправить в интерфейс 100GE1/0.

Эта гибкость позволяет как вендорам, так и (теоретически) конечным клиентам встраивать в ASIC свою логику, а не довольствоваться встроенными правилами.

Впрочем programmable pipeline - это уже совсем другая история.

Но сегодня программируемым конвейером хвастаются со сцен своих маркетинговых конференций уже почти все.

Note: К слову, в 2019-м Барефут был куплен Intel'ом.

Hint: Кстати, были у нас в гостях.

Marvell

Если загуглите в Яндексе "Marvell switch ASICs", то не так уж много ссылок вас проведут туда, где вам будут рады. Marvell определённо делает интегральные микросхемы, и даже вполне определённо делает сетевые интегральные микросхемы, но назвать его фаворитом этой гонки язык не поворачивается.

У них есть три сетевых ASIC'а, вполне конкурентоспособных по заявленным функциям и мощностям:

- Prestera CX - 12.8 Тб/с, обещают программируемый Pipeline.
- Prestera PX - по всей видимости, что-то около 1 Тб/с на роль тора.
- Falcon - 12.8 Тб/с - видел несколько упоминаний о нём, но даже на самом сайте Marvell'а информации о нём нет.

Пожалуй, из последних предложений вы можете сделать закономерный вывод, что больше про Marvell я ничего не знаю.

Innovium

У Innovium, основанного выходцами из Intel и Broadcom, есть два сетевых чипа: Teralynx 5 и Teralynx 7, обещающих знакомые скорости: 6.4 и 12.8 Тб/с.

Они установлены в паре цискиных коробок: Nexus 3408 и 3432D.

Другие

Есть и другие игроки, не снискавшие успеха среди гиперскейлеров.

Один из примеров - это **Cavium**. Приходилось слышать? Это вендор, купленный не так давно Marvell'ом и производящий NP для энтерпрайз-маршрутизаторов и (!!) базовых станций.

Буквально в феврале, кстати, появилась крайне любопытная новость: Ubiquiti UniFi USW-Leaf Overview 48x 25GbE and 6x 100GbE Switch.

Современный Leaf-коммутатор с 30ГБ SSD за $2000.

Немного пораскрутив публикацию, я обнаружил, что внутри сокрыт некий Taurus, разработанный **Nephos** - дочерней компанией MediaTek. И у них даже вполне любопытное портфолио.

С такой ценой, возможно, появляется новый игрок на рынке.

Для полноты картины приведу так же парочку малоизвестных компаний, которые производят низкоскоростные чипы коммутации на FPGA:

- Ethernity Networks
- Arrive Technologies

Нашлось, кстати, тут место и для отечественных разработок.

Например, вот такого малыша вместе с отладочным комплектом можно приобрести себе для доморощенного L2-коммутатора:

- Миландр 1923КХ028.

Custom silicon

Настало время поговорить про вендоров сетевого оборудования, которые всё ещё в состоянии содержать свой гигантский штат R&D.

Но именно с них когда-то всё и начиналось, рыночных чипов не было, а каждый производитель разрабатывал и изготавливал свои сетевые процессоры и ASIC'и.

И это сложно сегодня, потому что когда другие могут сосредоточиться только на аппаратной обвязке вокруг чипа и софте, другим приходится выделять ресурсы на фундаментальные разработки. Стоит отдать дань уважения вендорам за это.

Note:

Занимательный факт. Во время санкционной войны лета 2019 у Huawei был невоображаемый сценарий вылететь с рынка - американская компания Broadcom заморозила поставку ASIC'ов для их линейки CloudEngine.

Всё, конечно, завершилось хорошо.

Но почти одновременно с этим вышел модульный коммутатор CE16800 на чипах собственного производства, и обещали пицца-боксы.

Ясное дело, что занимались они этой разработкой уже довольно давно, наверно, лет 5.

Однако сей факт намекает на то, что, возможно, не так уж и плоха идея вкладываться в разработку своих чипов.

Очевидно, что и у них не по одному типу ASIC'ов, развивающихся планомерно и интегрирующихся во все новые устройства - они делятся по сериям железок, по их ролям. Не забываем и о том, насколько большие компании любят поглощения.

Я не собираюсь здесь погружаться в детали, перечисляя все их виды и возможности. Пройдёмся только поверхностно по наиболее известным из них.

Juniper

В первую очередь это, конечно, легендарный ASIC Trio, который бьётся внутри всех маршрутизаторов MX.

Марат Бабаян в своё время написал прекрасную статью о его работе.

Внутри их магистральных коробок - **ASIC Express** (ZX, TRITON).

А датацентровые коммутаторы, вроде QFX 10000, заряжены чипом **Q5**. **Q5** - это маректинговое название, внутреннее - **Paradise**. При этом часть PFE, стоящая на линейной плате называется - ELIT, а фабричная - ULTIMAT.

Huawei

У этих ребят тоже давняя история разработки своих чипов.

Когда-то начиналось с интеловских асиков, потом были эксперименты с чипом Marvell, а потом масть пошла. И новые поколения сетевых процессоров стали появляться один за другим.

Они отличаются от поколения к поколению, от класса к классу, от серии к серии.

Но для внешней людей их объединили в две линейки:

- **Ascend** для датацентров
- **Solar** для провайдеров и операторов

На самом деле это весьма распространённая практика. Тот же Broadcom:

Tomahawk3 - известное всем название чипа.

StrataXGS - семейство чипов (не то чтобы на слуху у каждого)

BCM56980 - внутреннее название чипа (не понадобится вам, если не собираетесь препарировать SDK).

Cisco

Можно сказать, что новейшие линейки программируемых чипов это:

- **Cisco Silicon One**, установленный в линейку Cisco 8000.
- **UADP** - Unified Access Data Plane - программируемый ASIC для каталист и некоторых нексусов
- **QFP** - Quantum Flow Processor - для ASR и ESP

А в глубине веков начинается такой, зоопарк, что я просто кину несколько, даже не пытаясь докопаться до глубин:

- Sasquatch, Strider - каталисты 29xx и 3xxx
- K1, K2, K5, K10 - каталисты 4000 и 4500
- EARL1>EARL8 - ещё разнообразные каталисты и Nexus 7000
- Monticello - Nexus 3548
- Big Sur - Nexus 6000
- F3 - Nexus 7000/7700
- nPower X1 - сетевой процессор для NCS.

Замечательные слайды, кстати, о производстве цискиных чипов и их устройстве.

И слайды про использование рыночных чипов в нексусах и их (чипов) архитектуру.

Отечественная микроэлектроника

Похвастаться терабитами, увы не можем, но вот есть 88Гб/с L3-коммутатор от "Цифровых решений" с поддержкой 1Гб/с, 10 Гб/с портов на собственном FPGA: Феникс-1/10G.

Обещали на базе этого FPGA потом выпустить ASIC, но чем история закончилась неизвестно.

Кстати, будут у нас в гостях.

Merchant vs Custom

Несмотря на всё вышеперечисленное, вендоры, конечно же, используют рыночные чипы. Те же Nexus (3xxx) или NCS5500 используют бродкомовские, инновиумские и барефутовские чипы.

Juniper QFX5xxx и Huawei Cloud Engine - бродкомовские.

За

Что заставляет их использовать рыночные вместо собственных?

Ну тупо проще взять готовенькое, впаять в плату и продать. Пресловутый Time To Market - хотя и с оговорками.

Экономия ресурсов на разработке такой сложной штуки, как микросхема по технологии 7нм с производительностью 25,6 Тб/с.

Компания, которая сосредоточена на разработке только чипов, развивает их гораздо быстрее как в плане производительности, так и функциональности.

Кто-то уже постарался и собрал информацию о том, какие функции нужны клиентам и все их реализовал

Против

В таком случае, что вендоров заставляет делать свои чипы, раз всё так хорошо?

А вот и оговорки с TTM - внести изменения в свой ASIC и выпустить следующую исправленную версию - проще и быстрее, чем запускать бюрократический маховик большой сторонней компании.

Кроме того до недавних пор не существовало программируемых рыночных чипов - вендору нужно было полагаться на фиксированный пайплайн или весьма урезанный SDK, предоставляемый поставщиком.

И производители сетевого оборудования сами начали учиться это делать, чтобы новые функции внедрять не перепаиванием асиков, а релизом новой версии софта.

Кроме того, едва ли бродком возьмётся за реализацию проприетарных протоколов или других функций, которые нужны одному покупателю - это же массовый рынок.

Никуда не спрятать вопрос безопасности. Тут своё родное - знаешь каждую закладочку.

Не всегда может устроить реализация QoS на рыночных чипах. На своих можно замутить любые мутки.

Свой SDK - свои разработчики, которые его поправят. Чужой SDK - чужие баги, давайте сдавать, решать, тестировать - времени требуется значительно больше.

В общем, хорошо, что мы находимся сейчас в ситуации высококонкурентного рынка, и мы каждые полтора-два года можем получать ещё более мощные коммутаторы по ещё более низким ценам.

Ну а если вдруг вам надоели все эти наши Клозы и масс-маркет-силиконы, и душа просит чего-то совсем экзотического, то вот почитайте, как строятся суперкомпьютеры для High

Performance Computing'a.

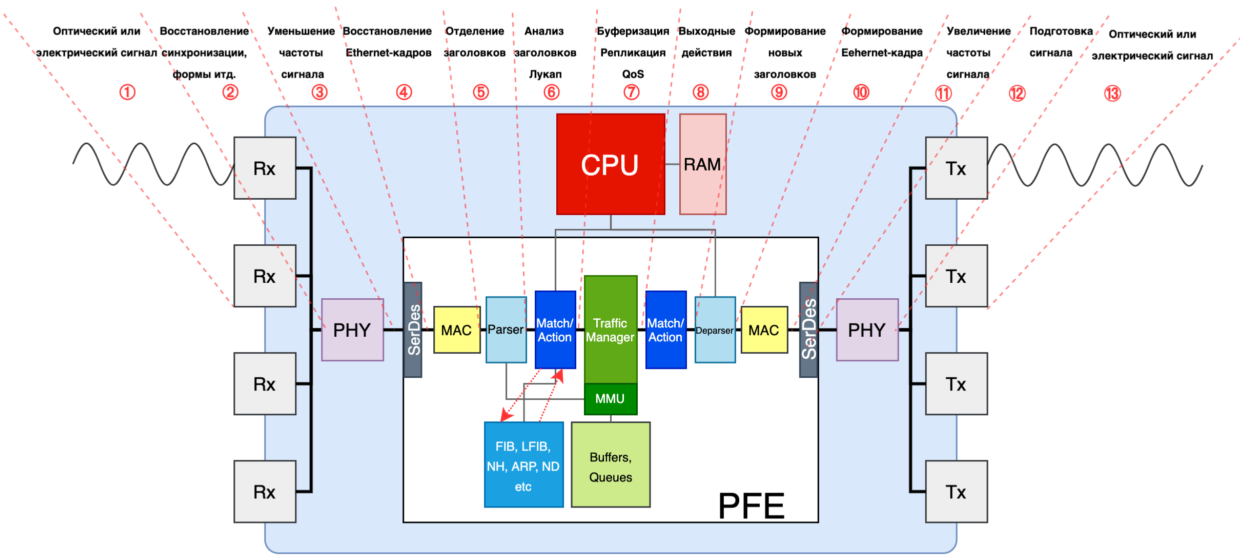
9

8. Путь пакета

Ну что же, кажется мы добрались до того момента, когда можно взглянуть на длинную и нелегкую жизнь, что предстоит прожить сигналу, несущему пакету.

Одночиповое устройство

Сначала рассмотрим пример попроще с single-chip shared buffer коробкой.



1. Оптический или электрический сигнал приходит в **интерфейс**.
2. Далее он попадает на чип **PHY**, который его усиливает, восстанавливает форму сигнала и синхронизацию (3R - Re-amplification, Re-shaping, Re-timing)

3. Из PHY сигнал по медным контактам бежит на **SerDes** чипа коммутации. Частота электрического сигнала здесь всё ещё десятки ГГц (для скоростей 10 Гб/с и выше). SerDes, понижает частоту (за счёт демодуляции NRZ, PAM4) до удобоваримой для чипа и передаёт его на модуль **MAC**.

4. MAC разбивает его на отдельные Ethernet-кадры. В этом ему помогает Inter-Frame Gap и преамбула, которые тут же за ненадобностью отбрасываются. Здесь же вычисляется контрольная сумма кадра, а потом сравнивается с хвостовым заголовком FCS.

   Если совпало, пакет без FCS передаётся дальше. Если нет - тут же отбрасывается, а счётчик ошибок на интерфейсе увеличивается на 1.

5. Далее этот Ethernet кадр ещё со всем заголовками отправляется на **Parser**. Тот отделяет управляющие заголовки от тела. Тело он складывает во **встроенный буфер**. Причём оно расчленяется на ячейки, чтобы плотно упаковаться в доступные сегменты памяти. Управляющими являются те заголовки, на основе которых применяется решение о маршрутизации пакета - Ethernet, MPLS, IP, TCP/UDP (для ACL и LAG/ECMP).

6. Пока тело хранится в буфере, набор заголовков передаётся на анализ в блок **Match-Action**.

   1. Сначала изучается Dst MAC. FE делает лукап в MAC-таблице (например, в памяти CAM). Если DMAC совпадает с MAC'ом самого узла, то кадр передаётся на обработку по конвейеру дальше. Если нет, то кадр коммутируется согласно таблице и на этом всё.

   2. Если MACи совпали, то кадр на основе EtherType передаётся следующему блоку (MPLS или IP).

      Какие же данные из заголовка понадобятся?

      - Значение TTL/Hop Limit.
      - Адрес назначения (IP, MPLS-метка итд.)
      - Приоритет (DSCP, EXP итд.)
      - *Возможно понадобятся и другие поля, такие как адрес источника, информация о вложенных данных (протокол, порт, приложение)*

   3. Пока тело всё ещё прохлаждается во внутричиповом буфере, происходит принятие десятков решений о его судьбе:

      - Уменьшение TTL/Hop Limit на 1 и проверка, не равен ли он 0. Если равен - то запрос на CPU для генерации *ICMP TTL Expired in Transit*.
      - Проверка по ACL - можно ли вообще передавать этот трафик.
      - Принятие решения о назначении пакета на основе Dst IP или верхней MPLS-метки.

Входной тракт чипа коммутации здесь делает лукап в FIB в попытке найти маршрут, соответствующий данному IP-адресу (или в LFIB, для поиска LSP). Будь то поиск в TCAM или запрос в блок алгоритмического поиска результатом будет указатель в RAM, где хранится искомая запись с набором инструкций.

Чип из RAM получает этот набор инструкций: выходной интерфейс, NextHop, инкапсуляции, выходной стек меток итд. При необходимости осуществляется рекурсивный лукап. В большинстве случаев чип коммутации выясняет на этом шаге MAC-адрес Next-Hop'а из таблицы соседей (Adjacencies Tables/ND Tables).

Если некстхопов несколько, а соответственно и несколько комбинаций (выходной интерфейс, инкапсуляция), то чип считает хэш от полей заголовков для определения в какой из членов LAG или ECMP послать этот пакет.

Если адрес назначения локальный, то или парсится следующий заголовок (как это и было выше с Ethernet), или принимаются какие-то меры аппаратные (BFD, например) или пакет передаётся на CPU (BGP, OSFP итд.)

- Преобразование приоритета из заголовков пакета во внутреннюю CoS-метку, в соответствии с которой пакет будет обрабатываться в пределах данного узла. Может навешиваться приоритет обработки в очередях и приоритет отбрасывания пакета.

  Всё это станет метаданными и будет помещено во временный заголовок, сопровождающий пакет до выходного тракта:

  - Выходной интерфейс
  - Приоритет
  - TTL
  - Next Hop/стек меток
  - Другое

4. Для BUM трафика модуль Match-Action определяет список выходных интерфейсов, но не создаёт реплики пакета - эта информация добавляется в метаданные.

Note: Важно понимать, что до завершения этого шага никаких решений о перегрузках, отбрасывании пакетов и очередях не принимается - тело пакета в любом случае сохраняется в буфере на время обработки заголовков и поиска выходного интерфейса

---

7. Эстафета передаётся модулю **TM**. Он ведёт всю бухгалтерию, касающуюся QoS: Congestion Avoidance, Congestion Management, диспетчеризация, управление задержками, шейперы, полисеры итд. Соответственно для данного пакета принимается решение в какую очередь его поместить, в какой цвет окрасить, сколько токенов из бакета он съест. Решения принимаются на основе заголовков пакета и конфигурации QoS.

   Если интерфейс, в который предстоит попасть пакету, испытывает перегрузку, то пакет может быть записан в низко- или высоко- приоритетную очередь в соответствии со своими заголовками. Может быть дропнут, если сработает механизм WRED/Tail Drop. А если настроен PFC, то он может быть записан в Headroom-буфер, чтобы сохранить его любой ценой, когда всякое вакантное место уже отсутствует в общей памяти.

   Если пакету повезло попасть в очередь, то TM должен согласно механизмам диспетчеризации в нужный момент времени извлечь его для отправки

   Если пакет BUM, то тут же на TM родятся его копии, каждая со своим набором метаданных. .. note:: Тут важно понимать, что все манипуляции и передвижения (даже постановка в очередь и извлечение из неё) проводятся как бы виртуально - с метаданными, а не с телом пакета - оно преспокойно лежит всё это время на одном месте (если мы говорим про вариант с Shared Buffer)

8. Далее из модуля TM метаданные о пакете кочуют в **Egress Match-Action**, где над ними производятся дополнительные экзекуции, в виде, например, выходных ACL.

9. **Deparser** получает метаданные, формирует из них стек заголовков. В частности здесь он преобразует внутренний приоритет в значения приоритетов IP/MPLS/Ethernet, записывает правильный TTL, пересчитывает контрольные суммы итд. Так же Deparser извлекает бренное тело из буфера, склеивает его с новыми заголовками, и передаёт блоку MAC.

10. И потом всё раскручивается в другую сторону: **MAC** добавляет IFG, преамбулу, считает новую контрольную сумму кадра и добавляет трейлер FCS.

11. **SerDes** теперь повышает частоту и отправляет сигнал в медные дорожки.

12. **PHY** готовит сигнал для передачи в среду (медь, оптика, радио).

13. С порта счастливый пакет уходит в своё короткое (или не очень) плавание до следующей пристани.

Note: Это, несомненно очень упрощённый пример обработки трафика. Более того, он рассматривает частный пример реализации. Фактически чип MAC может стоять отдельно от чипа коммутации, или наоборот PHY быть его частью. PHY может самостоятельно понижать частоту, и тогда не нужны SerDes. Репликация BUM может происходить на TM, а может на выходном тракте. Дорожки, в случае Silicon Photonics будут не электрическими, а оптическими. И много прочих нюансов

Если заменить Shallow Buffer на Deep Buffer, то в случае перегрузок тело пакета может сохраняться на внешнюю память.

Многочиповое устройство

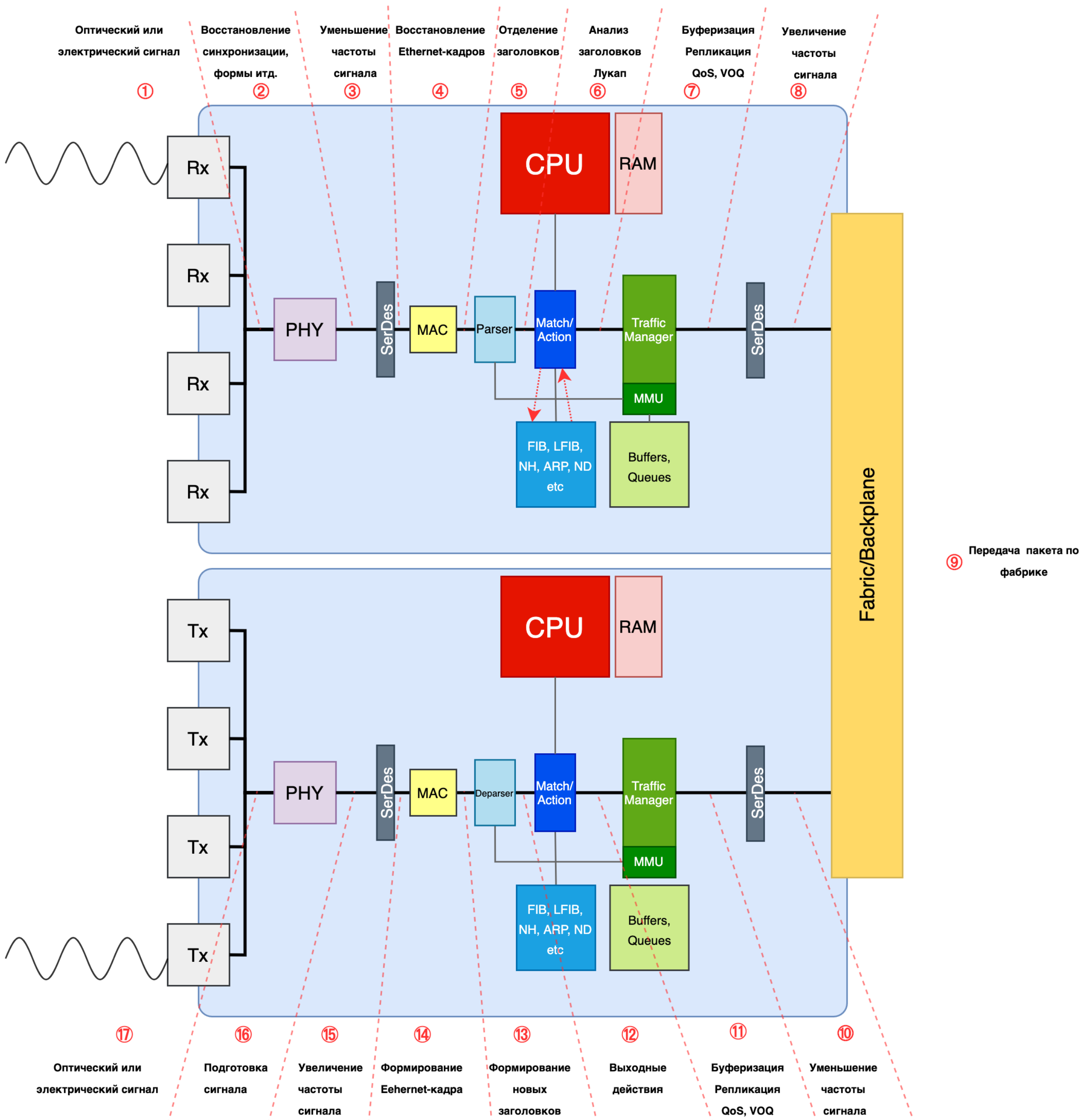
Нередки случаи, когда в одну коробку ставится два-три и больше чипов для увеличения производительности.

Тогда входному чипу коммутации нужно ещё определить и выходной чип во время лукапа и указать его в метаданных. Поскольку обработка будет происходить дважды на двух разных чипах, естественно, тело пакета передаётся между ними и буферизируется дважды. Тут возможны разные архитектуры памяти, но VOQ одна из наиболее удобных - буфер выходного чипа в этом случае используется только как FIFO, а вся нагрузка по обеспечению QOS ложится на входной TM.

Модульное устройство

Значительно усложняется картина в случае шассийных коробок. Добавляется VOQ на входном чипе, фабрика, обработка на выходном, арбитраж с пиггибэком.

Для шассийных коробок процесс изменится так.



Интерфейс входной платы → PHY → SerDes → MAC → Parser

Эти шаги без изменений.

Parser также складывает тело в буфер (встроенный или внешний). Пока ещё здесь нет разницы куда и с каким приоритетом этот пакет полетит.

А вот **Match-Action** должен в момент лукапа теперь определить плюсом к остальным деталям направления и действий с пакетом, ещё и выходной чип. Эта информация кодируется в FIB на этапе его заполнения Control Plane’ом.

То есть чип из RAM должен извлечь следующий набор инструкций: **выходной чип**, выходной интерфейс, NextHop, инкапсуляции, выходной стек меток итд.

И следующие метаданные будут привязаны к пакету:

- Выходной чип
- Выходной интерфейс
- Приоритет
- TTL
- Next Hop/стек меток
- Другое

С BUM трафиком ситуация тоже не меняется - модуль Match-Action определяет список выходных чипов и интерфейсов и вносит это в метаданные.

У **TM** тоже работы прибавляется.

В подавляющем большинстве случаев на линейных картах модульных шасси будет VOQ на входе. То есть для каждого возможного выходного интерфейса (по сути для всех существующих) будет создана очередь на выходном чипе. TM, уже обладая информацией о том, куда пакет должен быть направлен, помещает его в очередь, отвечающую за данный выходной интерфейс. И соответственно поступает с ним так, как того требуют приоритет пакета, условия перегрузок, настроек QoS на данном конкретном порту.

Если используется арбитр (а он используется, бьюсь об заклад), то TM должен ещё и на отправку пакета получить разрешение от выходного чипа, чтобы быть уверенным, что этот пакет не будет отброшен из-за конгестии на выходном порту.

Ну и кроме всего прочего TM должен выполнить репликацию BUM-пакетов. Но не во все порты, за которыми есть получатели этого трафика, а только по числу выходных чипов, за которыми они есть. Что и логично, чтобы не забивать фабрику.

Ну а дальше у пакета недолгий путь через **фабрику коммутации**. В этом путешествии ему не обойтись без верного спутника - временного заголовка с его метаданными.

Пакет обычно шинкуется на ячейки одинакового размера, каждой из них приклеивается этот метазаголовок и порядковый номер, и отправляется на фабрику.

На выходной плате ячейки обратно спекаются в один пакет и попадают на **TM**. В целом в зависимости от реализации здесь чего с ним только может не происходить, но базовый минимум - это просто FIFO очередь (потому что QoS отработал на входной плате) и репликация BUM-пакетов по числу портов-подписчиков.

Следующим пристанищем пакета становится выходной чип коммутации. Здесь могут приниматься ещё какие-то решения о передаче, например, выходные ACL.

Ну а дальше **Deparser** → **MAC** → **SerDEs** → **PHY** → **интерфейс** → **среда**

Локальные пакеты

Бóльшая часть локальных пакетов обрабатываются на ЦПУ.

Напомню, что локальные - это те, которые были созданы на данном узле или которые предназначены именно ему (юникастовые), которые предназначены всем/многим (броадкастовые или мультикастовые). К ним относятся пакеты протоколов Control Plane (BGP, OSPF, LDP, LLDP итд), пакеты протоколов управления (telnet, SSH, SNMP, NetConf итд), пакеты ICMP.

К ним же стоит отнести транзитные протоколы, требующие обработки Control Plane'ом узла (TTL Expired, Router Alert).

Входящие

Вплоть до блока Match-Action с ними происходит всё то же самое, что и с транзитными. Далее чип коммутации, обратившись в таблицу MAC-адресов , видит, что DMAC - это MAC-адрес локального устройства, заглядывает в EtherType. Если это какой-нибудь BPDU или ISIS PDU, то пакет сразу передаётся нужному протоколу.

Если IP - передаёт его модулю IP, который, заглядывая в FIB, видит, что и Dst IP тоже локальный - значит нужно посмотреть в поле Protocol заголовка IPv4 (или Next Header IPv6).

Определяется протокол, принимается решение о том, какому модулю дальше передать пакет - BFD, OSPF, TCP, UDP итд. И так пакет разворачивается до конца, пока не будет найдено приложение назначения.

Когда входной чип коммутации с этим справился, содержимое пакета передаётся на CPU через специальный канал связи.
На этом шаге достаточно интеллектуальные устройства применяют политику по ограничению скорости протокольных пакетов, передаваемых на ЦПУ, чтобы одними только telnet'ами не заDoSить процессор - **CoPP** - Control Plane Policy.

Если данный пакет принёс информацию об изменении топологии (например, новый OSPF, LSA), Control Plane должен обновить Soft Tables (RAM), а затем изменения спускаются в Hard Tables (RAM/CAM/TCAM+RAM).

Если пакет требует ответа, то устройство должно его сформировать и отправить назад изначальному источнику (например, TCP Ack на пришедший BGP Update) или передать куда-то дальше (например, OSPF LSA или RSVP Resv).

Исходящие

Исходящие протокольные пакеты формируются на ЦПУ - он заполняет все поля всех заголовков на основе Soft Tables и далее, в зависимости от реализации, спускает его на Ingress или Egress FE.

> **Attention:**
>
> Из-за того, что пакет сформирован на процессоре, зачастую он не попадает под **интерфейсные** политики. Архитектурно многие операции, выполняющиеся на FE, требуют того, чтобы FE производил Lookup и формировал заголовки.
>
> Отсюда могут быть любопытные и неочевидные следствия, например, их не получится отловить ACL, вы можете не увидеть их в зазеркалированном трафике, они не будут учитываться при ограничении скорости. Но это зависит от вендора и оборудования.
>
> Однако политики, работающие с очередями на CPU их, конечно, увидят.

Есть некоторые протоколы Control Plane, которые всё-таки обрабатываются в железе. Ярким примером может служить BFD. Его таймеры выкручиваются вплоть до 1 мс. CPU, как мы помним, штука гибкая, но неповоротливая, и пока BFD-пакет пройдёт по всему тракту и развернётся до заголовка BFD, пока до процессора дойдёт прерывание, пока тот на него переключится, прочитает пакет, сгенерирует новый, вышлет его, пройдут десятки и сотни миллисекунд - глядь, а BFD-то уже развалился.

Поэтому пакеты BFD в большинстве случаев разбираются на чипе, на нём же и готовится ответ. И только сама сессия устанавливается через CPU.

> ..note:: Большие в этом вопросе пошли ещё дальше, перенеся на железо наиболее рутинные операции. Так, например, Juniper ввёл PPM - Periodic Packet Management, который разделяет функции Control Plane некоторых протоколов между управляющим модулем и интерфейсным:
>
> - Bidirectional Forwarding Detection (BFD)
> - Connectivity Fault Management (CFM)
> - Link Aggregation Control Protocol (LACP)
> - Link Fault Management (LFM)
> - Multiprotocol Label Switching (MPLS)
> - Real-time Performance Monitoring (RPM)
> - Spanning Tree Protocol (STP)
> - Synchronous Ethernet (SYNCE)

- Virtual Router Redundancy Protocol (VRRP)

Note:

История выше отсылает нас к длинным пингам. Иногда инженер проверяет RTT своей сети путём пинга с одного маршрутизатора на другой. Видит вариацию в десятки и сотни мс и, начиная переживать, открывает запросы вендору. Пугаться тут нечего. Обычно ICMP обрабатывается на CPU. И именно занятостью процессора определяется время ответа. При этом корреляция с реальным RTT сети практически нулевая, потому что транзитный трафик на CPU не обрабатывается.

Некоторые современные сетевые устройства могут обрабатывать ICMP-запросы и формировать ICMP-ответы на чипе (NP, ASIC, FPGA), минуя долгий путь до CPU. И вот в этом случае циферки в ping будут адекватны реальности. Но я бы всё же на это не полагался

Кроме того, есть технологии мониторинга качества сети (OAM), работающие аппаратно, например CFM.

10

9. Заключение

Что ещё сказать после 50 мегбайтов? Только то, что многое из того, что вы прочитали в этот раз - частные случаи, которые могут быть (и будут) несправедливы в других ситуациях.

Как только сетевой инженер смещает свой фокус со стандартизированных протоколов в область обработки пакетов, он падает в пропасть бесконечных компромиссов, где нет универсальных ответов, нет RFC, нет исчерпывающих мануалов. И чем глубже он падает, тем страшнее становится разнообразие деталей и нюансов.

Эта книга планировалась небольшой заметкой о буферах - хотелось копнуть неглубокую ямку и выяснить, чем отличаются Shallow Buffer от Deep. Всё началось с небольшой странички, подбивающей информацию о размерах буферов. А потом ссылочка за ссылочкой и размотался клубок. Стало понятно, что без разбора типов чипов не обойтись, дальше пришлось углубиться в архитектуру. Здесь бы и тормознуть, но верёвка уже оборвалась.

Полезные ссылки

В этот раз хоть под кат убирай. Но, поверьте, я оставил тут только самые хорошие источники, прочитанные лично моими глазами и отобранные лично моими руками.

- **Архитектура сетевых устройств**
  - Anatomy of Internet Routers
  - Cisco Nexus 3000 Switch Architecture
  - Juniper Hardware Architecture
  - Hardware Defined Networking

  - Как сделать коммутатор?
  - Сети для самых маленьких. Часть четырнадцатая. Путь пакета
- **Архитектура ASIC**
  - Packet Pushers. Understanding ASICs For Network Engineers (Pete Lumbis
  - Cisco Enterprise ASICs
  - Broadcom Ships Jericho2
  - Broadcom Launches Another Tomahawk Into The Datacenter
  - Programmable Pipeline
  - The Most Important Development Of This Century
  - Understanding chiplet in one article
  - Advanced_packaging_five_trends_to_watch_in_2017.aspx# Advanced packaging: five trends to watch in 2017
  - Fan Out – Simple to Complex
- **Программируемость**
  - Design Principles for Packet Parsers.
  - Fast Programmable Match-Action Processing in Hardware for SDN
  - P4 Tutorial, Hot Chips 2017
- **Архитектура памяти**
  - Packet Buffers. Отсюда по ссылкам разворачиваются разнообразные материалы очень глубоко
  - An Update on Router Buffering
  - NCS 5500 Buffering Architecture
  - Cisco Nexus 5000 Series Switches. Cut-through and Store-and-Forward
  - Understanding CoS Flow Control (Ethernet PAUSE and PFC
  - Quality of Service.Headroom buffers.
- **Реализации очередей**
  - Strategies of packet buffering inside Routers
  - Understanding CoS Virtual Output Queues (VOQs) on QFX10000 Switches
  - What is VOQ and why you should care?
- **Deep Buffers**
  - Not all deep buffer switches are created equal

- Arista’s Big Buffer B.S.
- Speeding Applications in Data Center Networks. The Interaction of Buffer Size and TCP Protocol Handling and its Impact on Data-Mining and Large Enterprise IT Traffic Flows

• Data Center TCP (DCTCP): TCP Congestion Control for Data Centers

• Understanding Microburst

• Ну и немного мотивационных видео от Broadcom
  - Introduction to Broadcom’s Switch Portfolio
  - Broadcom Trident4: Disrupting the Enterprise Data Center & Campus
  - Broadcom Tomahawk4: Industry’s Highest Bandwidth Chip
  - Jericho2: Driving Merchant Silicon Revolution

Спасибы

• Андрею Глазкову (glazgoo) за рецензию и дельные замечания о Shared Buffers и коммерческих чипах

• Михаилу Соколову (insektazz) за разъяснения об устройстве чипов, SerDes и Silicon Photonics

• Александру Клименко (v0lk) за обнаружение точек роста в вопросах Admission Control, Alpha, Pipeline’ов и минусов DCTCP

• Александру Азимову (mitradir) за комментарии о Lossless Ethernet

• Дмитрию Афанасьеву (fl0w) за дополнения ко всем частям статьи

• Виталию Венгловскому (AllTheThingsUndone) за наставление на путь в вопросах On-Chip Buffer, Packaging и других

• Наташе Самойленко за помощь с RTD. Благодаря ей вы можете скачать эту книгу в PDF.

• Артёму Чернобаю за КДПВ

Особо благодарных просим задержаться и пройти на Патреон.

Become my patron on patreon