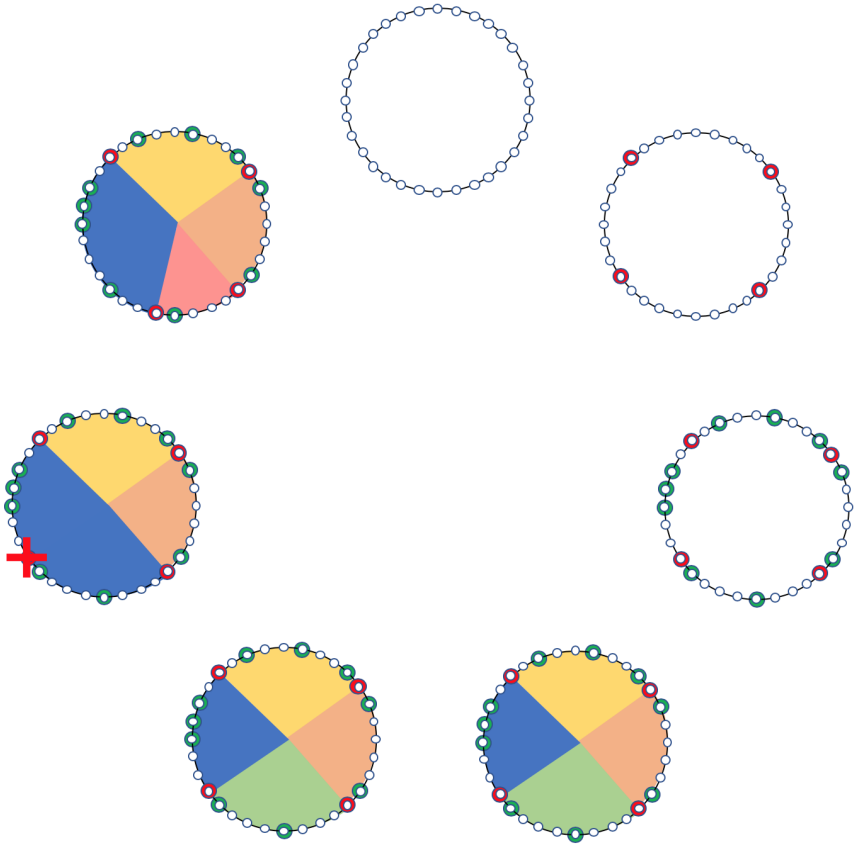
ECMP и превратности балансировки на сетевом оборудовании

Такой привычный и понятный ECMP. Знакомый каждому сетевику с младенческого возраста. Ну о чём тут ещё можно говорить? ECMP особо не вызывает ни теоретических ни практических вопросов в сетях операторов и в энтерпрайзе, где счёт различным путям идёт на единицы. А вот датацентры дышат ECMP, и тут уже приходится считаться: например, закупая ToR-коммутаторы, стоит в спеке обратить внимание на количество ECMP-групп и общее количество доступных некстхопов, а для бордеров - на поддержу Resilient Hashing.

Содержание

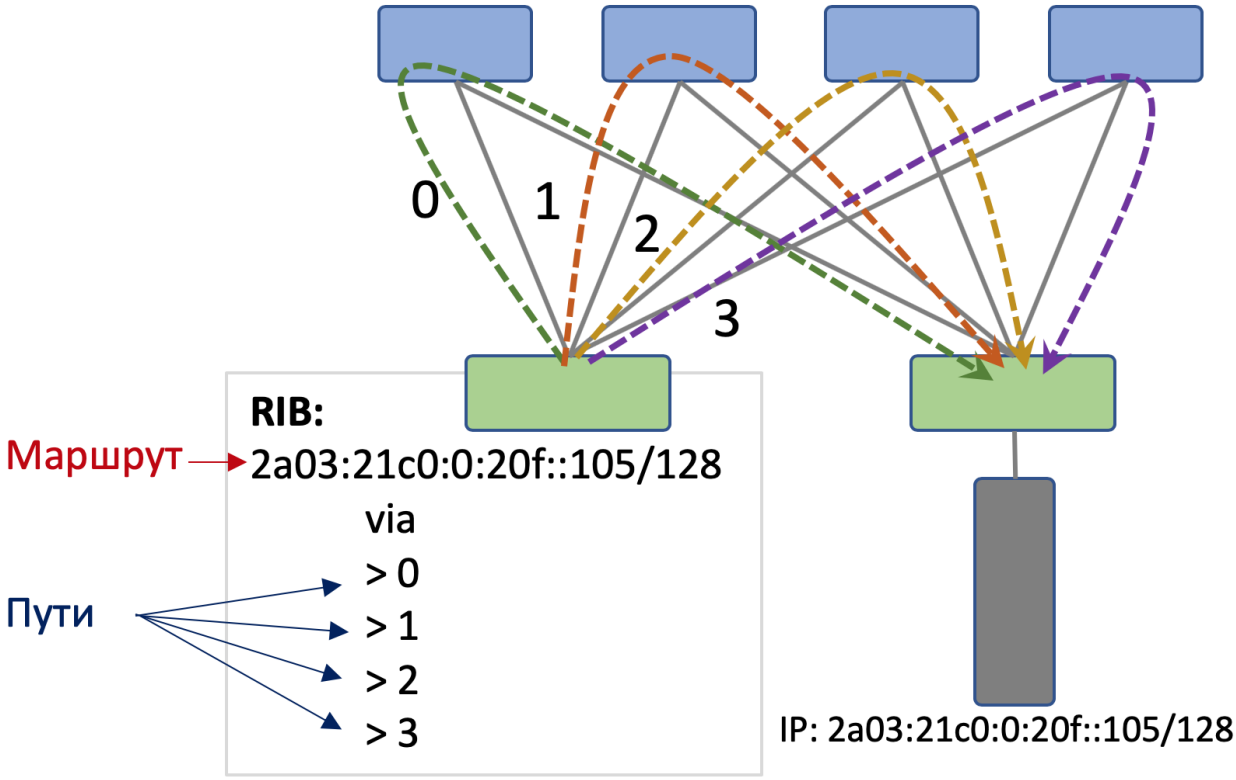
- Балансировка нагрузки
  - Static Hashing
- Особенности статического хэширования
- Anycast
- Resilient/Consistent Hashing
- Поляризация трафика при балансировке
- Ещё про Anycast
- Балансировка MPLS
- Мыши и слоны. Динамическая балансировка
- Равноценные пути
- ECMP-группы
- ECMP-группы в случае MPLS



Что, по сути своей, такое ECMP? Это **балансировка** трафика по **равноценным** путям. Разберёмся сначала со словом "балансировка".

Балансировка нагрузки

А давайте-ка выровняем информационное поле? В такой сети, как ниже, 2a03:21c0:0:20f::105/128 - это маршрут, а 0, 1, 2 и 3 - это возможные пути.

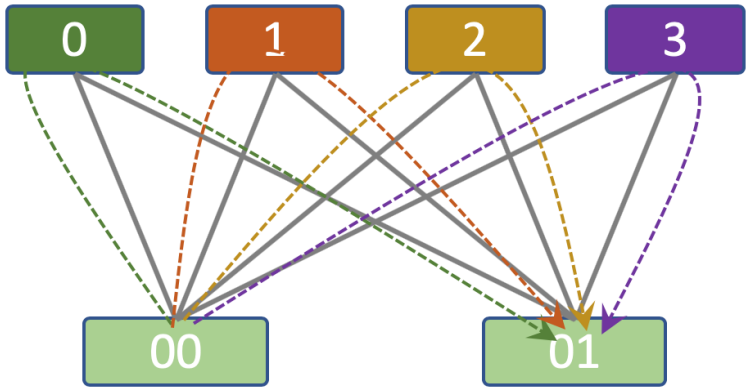


Далее, во всей статье именно это я и буду иметь в виду, говоря "маршрут" и "путь". "Путь" и "некстхоп", при этом, будут в данном контексте синонимами. А ещё иногда буду называть их бакетами - это термин из теории хэширования.

Она может осуществляться либо по пакетам, либо по потокам.

Первый способ - per packet - предполагает примитивный round robin: один пакет в один линк, другой - в другой, и т.д. по кругу. Он давно себя дискредитировал: даже в идеальном случае, задержки доставки будут отличаться, и на получателях будет перманентный реордеринг пакетов. И вроде бы TCP создан с ним справляться, но реордеринг заставляет дольше хранить сегменты в буферах, что может снижать скорость и негативно влиять на приложение. Ну и кроме TCP, у нас есть, как минимум, UDP или какой-нибудь CES, которые устроят вам вакханалию в самый неподходящий момент (всегда). В реальной эксплуатации едва ли вы где-то увидите пакетную балансировку.

Второй - балансировка по потокам (per flow) . Тут всегда пакеты одной сессии отправляются по одному и тому же пути. То есть, в случае TCP/UDP учитывает (SIP, DIP, Proto, SPort, DPort) - так называемый 5-Tuple. Для каждого пакета высчитывается хэш на основе этих полей. На основе этого хэша трафик отправляется по тому или иному пути. Для одного и того же 5-Tuple всегда будет получаться один и тот же хэш, соответственно, выбираться один и тот же путь. В простейшем случае формула выбора пути следующая: hash % path_count - или, иными словами, остаток от деления хэша на число доступных путей. Например, есть 4 пути и 5 TCP-потоков. Результаты хэш-функции: 4, 8, 15, 16 и 23. Остатки от деления на 4: 0, 0 и 3, 0 и 3. Соответственно, потоки разложатся так:



Хэш потока	Путь
4	0
8	0
15	3
16	0
23	3

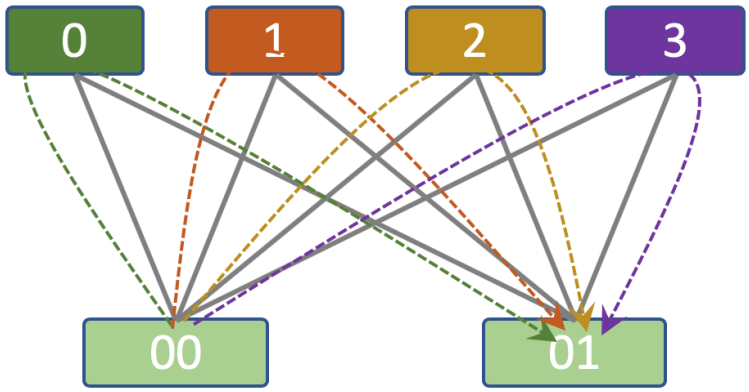
При большом числе потоков, они сравнительно равномерно раскладываются по путям.

> Раньше у некоторых вендоров было два ограничения: строгая рекомендация по использованию числа линков в ECMP, кратного степени двойки, и ограничение на количество путей в ECMP. Оба были вызваны аппаратными возможностями. На сегодняшний день это уже неактуально.

Описанный подход называется **статическим хэшированием**, поскольку **независимо от обстоятельств** чип применят хэш-функцию на заголовки пакетов и возвращает путь, на который указал остаток от деления.

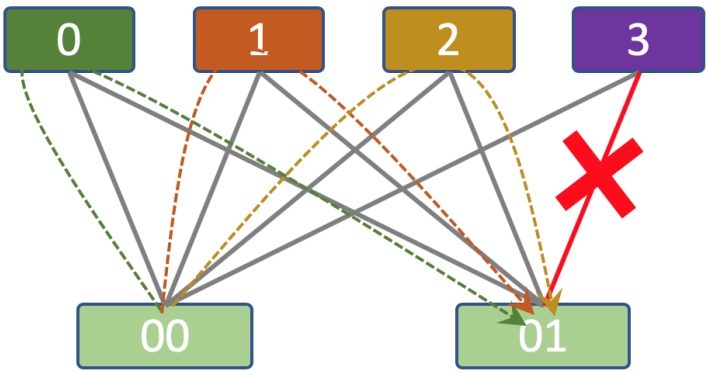
Особенности статического хэширования

Статическое хэширование замечательно работает на статической же сети, в которой ничего не флапает. Однако, стоит убрать или добавить путь, как вся таблица пересчитается. Разберёмся на примере. У Вани было 4 некстхопа и 5 потоков. На основе простейшего алгоритма по остатку от деления потоки распределяются так:



Хэш потока	Путь
4	0
8	0
15	3
16	0
23	3

Маша забирает у Вани 1 поток, выдернув провод. И вот как меняется распределение:



Хэш потока	Старый путь	Новый Путь
4	0	1
8	0	2
15	3	0
16	0	1
23	3	2

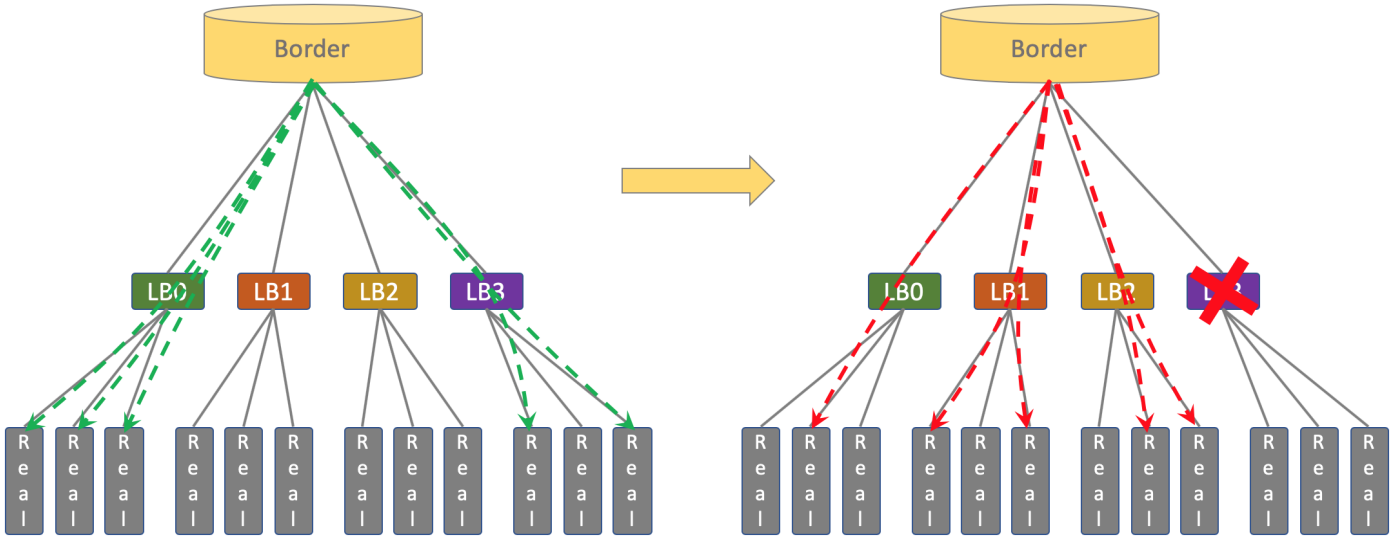
Все потоки перераспределились. В случае операторов связи, нас это мало волнует - ну, может, задержки немного изменятся. На фабрике в ДЦ (например, спайн выключился) для обычных маршрутов - аналогично.

Anycast

Ситуация кардинально меняется, когда речь заходит об Anycast"е. Напомню, что это один и тот же префикс, обычно /32, который анонсируется разными устройствами. И трафик на данный префикс отправляется к ближайшему устройству. То есть, запросы к одному IP-адресу могут фактически обрабатываться разными серверами, на которых он настроен. Такая конфигурация используется для

балансировки нагрузки. И только таким способом можно обслужить терабиты трафика и миллиарды пользователей, когда имеется максимум пара 25Гб/с порт на одном сервере. Заводим 100 бэкендов за одним IP-адресом и получаем возможность обработать терабит трафика.

Дополнительные плюсы anycast - трафик всегда идёт на ближайшие хост с этим адресом, что уменьшает задержку и общую утилизацию сети - ведь трафику нужно пройти меньший путь. Если на бордере есть несколько равноценных путей на один адрес, он будет по ECMP слать часть запросов по одному пути, часть - по другому. Так они и будут попадать на разные сервера. Допустим, у нас в датацентре есть 4 L3/L4-балансировщика трафика, анонсирующих один и тот же IP-адрес, за которым скрывается web-сервис. Каждый из них отправляет полученные запросы на свой пул реальных серверов.

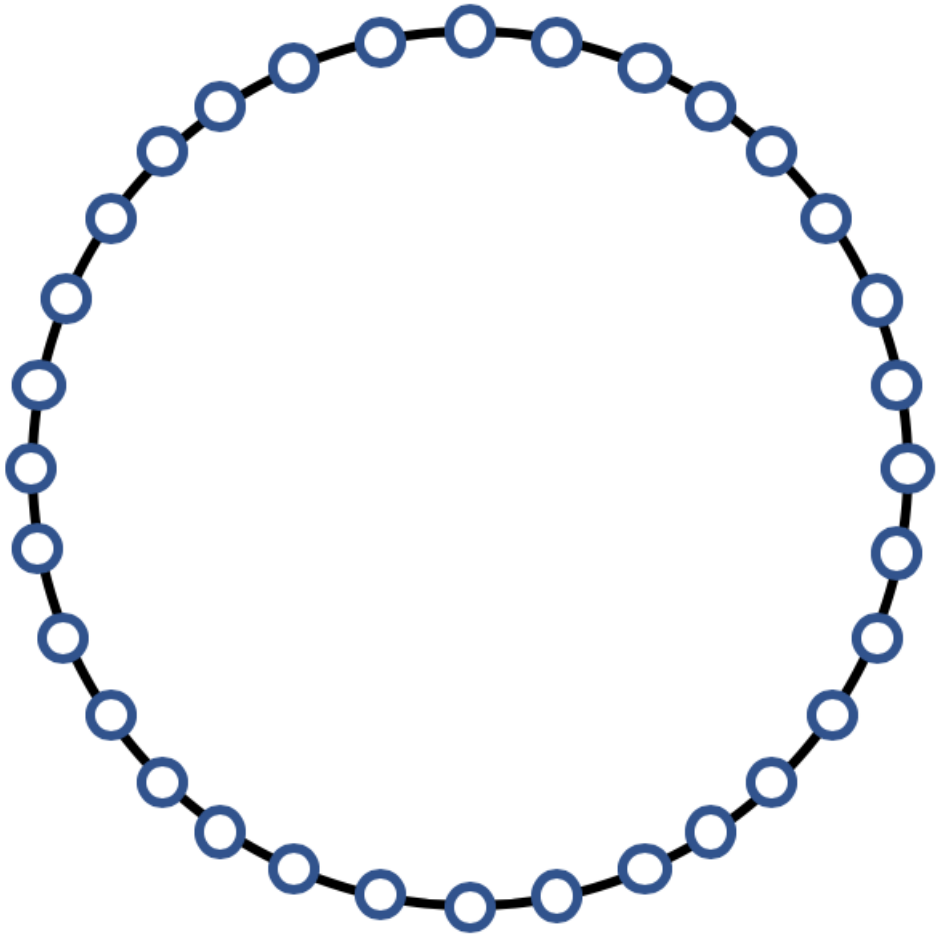


В этой ситуации перераспределение потоков при выпадении одного балансировщика будет означать, что запросы будут прилетать на сервера, на которых нет установленной сессии, а значит - это ресет сессии, переустановка и весь процесс сначала. То есть была, например, загрузка многогигабайтного файла в хранилище, осталось пара мегабайтов, моргнул балансировщик - и всё! То же самое и при добавлении нового балансировщика. И, конечно, это не только недовольство пользователей, но и резкий рост нагрузки на мощности - переустановление сессий, повторные операции. Возрастает риск получить тут масштабное каскадное падение. С этой проблемой призван справиться механизм **Resilient Hashing**.

Resilient/Consistent Hashing

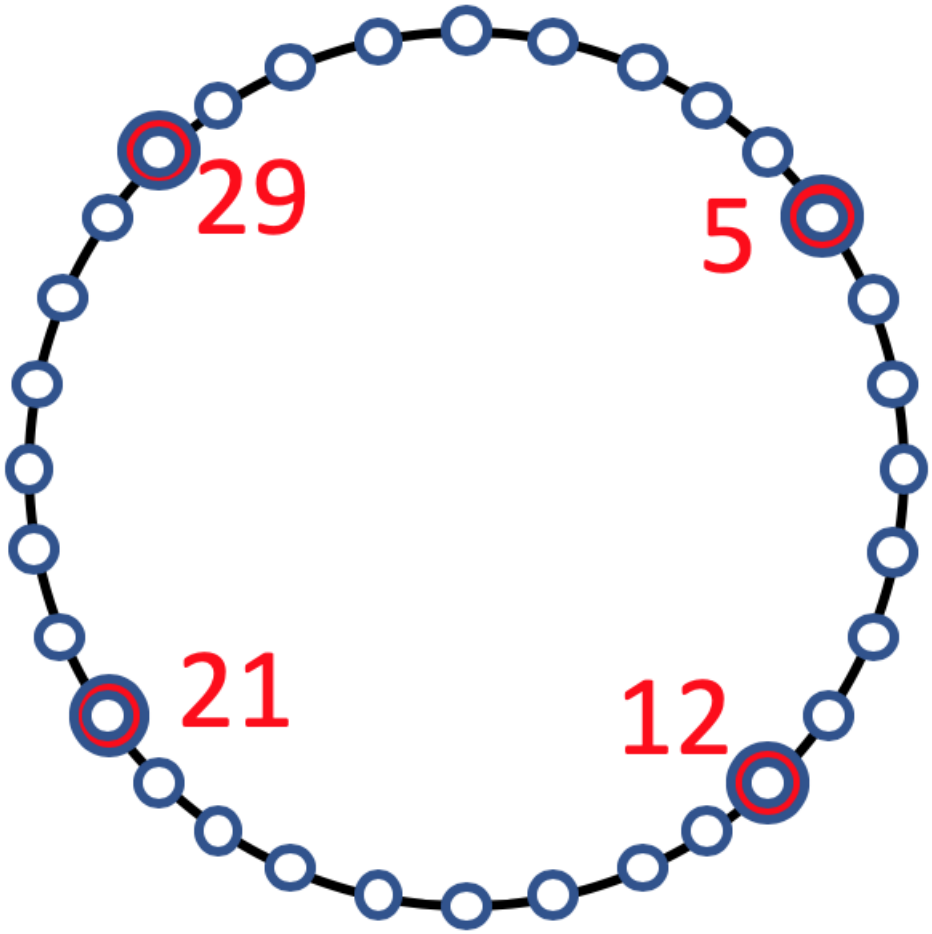
Читается, как ризилиент! Били ограничения статического хэширования по всем и больно, поэтому некий Каргер в MIT описал такой алгоритм хэширования, который позволял с минимальными нарушениями переживать изменение количества бакетов. Под бакетами (или слотами) в нашем случае будем иметь в виду доступные пути. И вот как это достигается.

Представим себе окружность. Она разбита на M делений. M - заведомо большое число (типа 2^32).

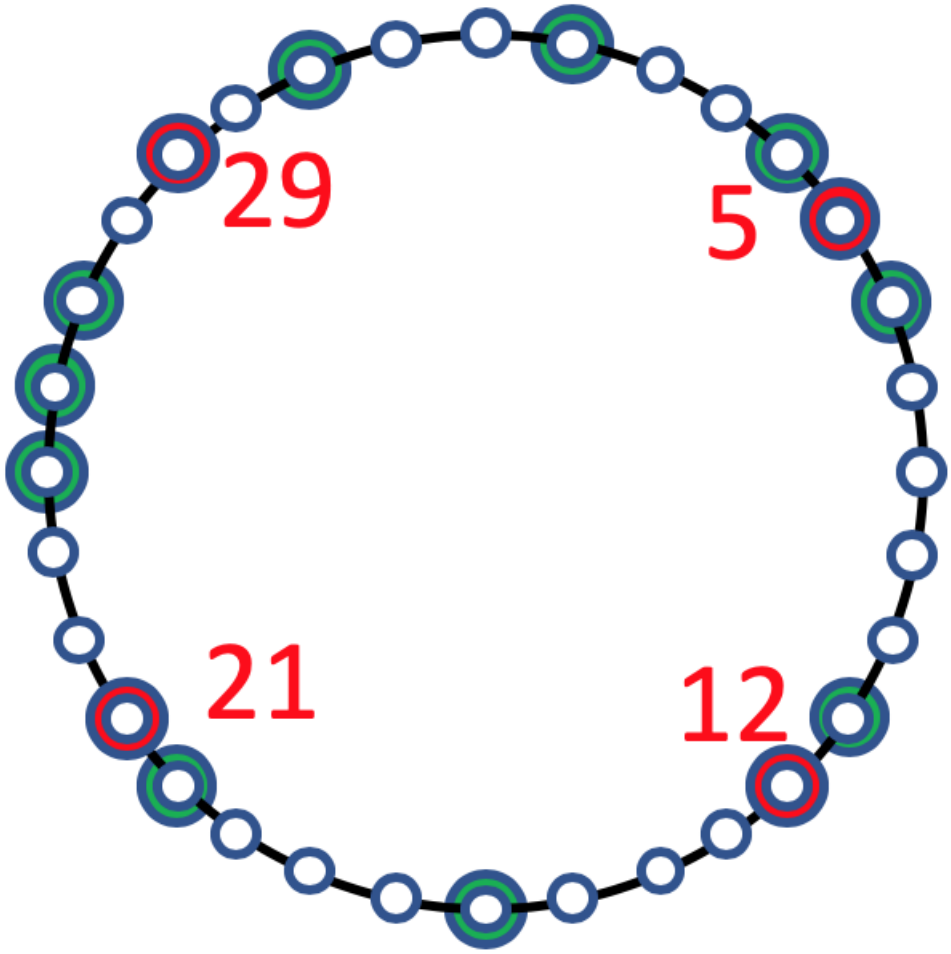


У нас N бакетов (путей). Каждому из этих N бакетов сопоставлено какое-то число в диапазоне [o, M], то есть они рассредоточены по окружности.

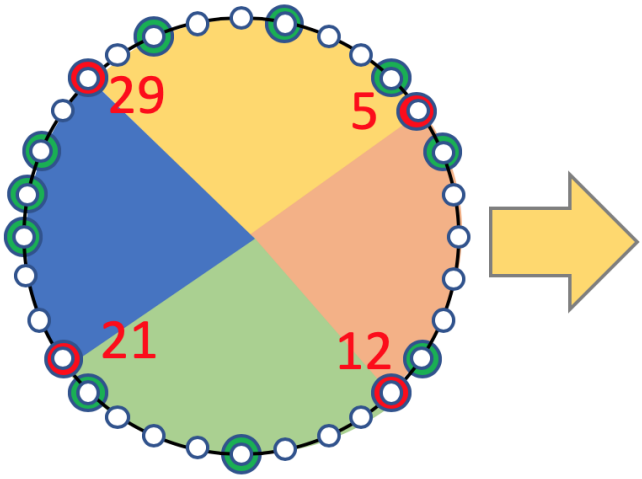
Положим равномерно.



Ключи (хэши от 5-Tuple) тоже находятся в диапазоне [0, M] и тоже наносятся на эту же окружность.



Следующий ближайший к ключу бакет ставится ему в соответствие и заносится в таблицу. Если следующее значение больше M - начинаем с 0.

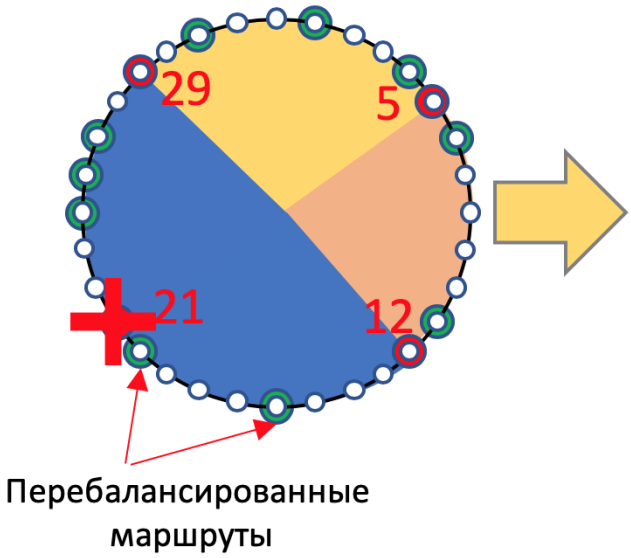


Хэш потока	Путь
1, 4, 31	5
6, 11	12
16, 20	21
24, 25, 26	29

В чём разница со статическим хэшированием, спросите вы? А давайте посмотрим.

Сценарий 1. Удаление пути

Допустим, путь №21 исчезает. Тогда с окружности удаляется эта точка, а все сессии, перебегают на путь №29.

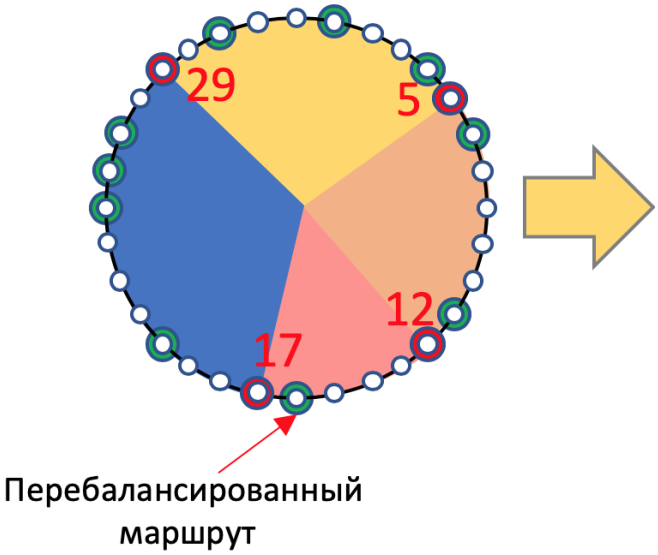


Хэш потока	Путь
1, 4, 31	5
6, 11	12
	21
24, 25, 26, 16, 20	29

При этом, остаются неизменными все сессии, что лежат через пути №5, 12 и 29. То есть, фактически, дёрнутся только те, которые и так дёрнулись бы.

Сценарий 2. Добавление пути

Теперь добавляется путь №17. Тогда все сессии, что находятся между путями 12 и 17, которые до этого шли через путь 29, перенаправятся в путь №17.



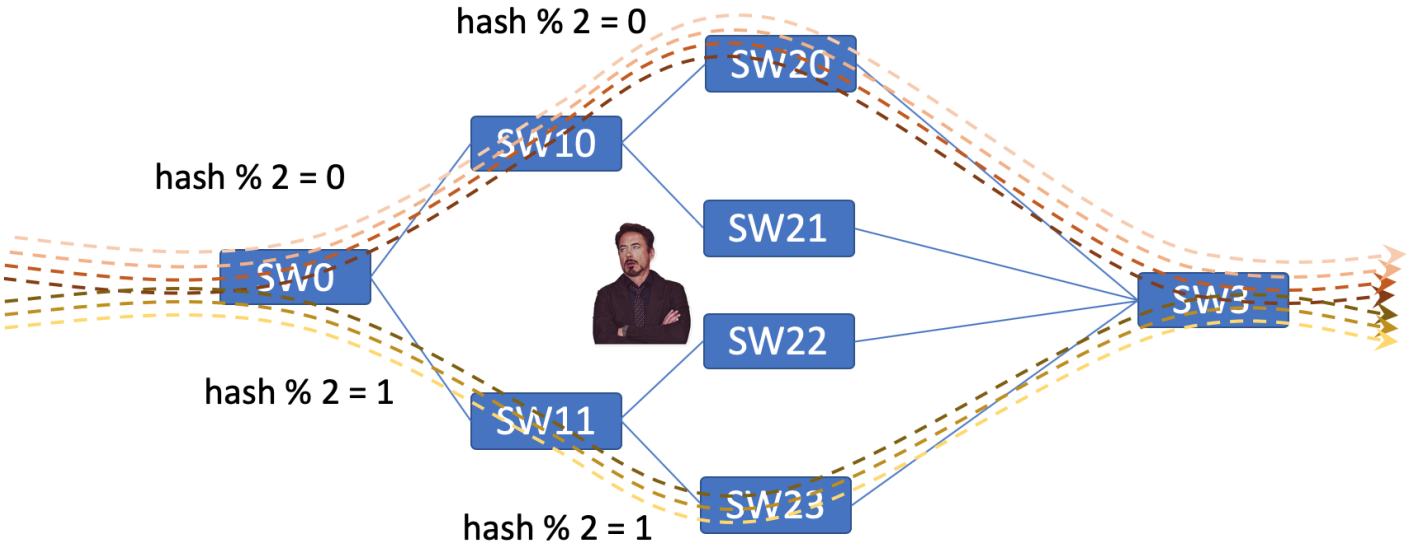
Хэш потока	Путь
1, 4, 31	5
6, 11	12
24, 25, 26, 20	29
16	17

И опять же ничего не случится с теми, которые идут через пути 5 и 12. И будет затронута только часть маршрутов, идущих через 29. Безусловно, это простая реализация, в которой, например, не поддерживается равномерность балансировки, но она даёт представление, как реализуется Resilient Hashing. С фактической его работой на сетевом оборудовании не всё так просто, поэтому не все устройства его поддерживают. А те, что поддерживают, дают лишь ограниченное число записей.

> Как это иногда бывает, возникла некая путаница с терминологией, с которой мне пришлось долго повозиться. Как я уже сказал выше, в 1997-м году Каргер опубликовал статью, описывающую как распределять запросы по меняющемуся пулу web-бэкендов, в которой ввёл термин **Consistent Hashing**. С тех пор в мире разработки все алгоритмы, решающие эту задачу, стали называть Consistent Hashing"ом. Но в мире сетевых устройств этого термина словно бы не существует. То ли из-за того, что тут он реализован аппаратно, то ли название как-то кем-то защищено, то ли назло ветрам, но в сетях это называется Resilient Hasing"ом. Что, кстати, на мой взгляд, более точно описывает поведение. В общем, так и повелось, что когда мы говорим о балансировщиках трафика - это Consistent Hashing. Когда мы говорим об ECMP на сетевом оборудовании - это Resilient Hashing. И ровно ни в одном документе не удалось найти упоминания одновременно двух названий, чтобы сопоставить одно ли это и то же, конкурирующие алгоритмы или одно является разновидностью другого. Буду надеяться, что теперь по запросу "consistent hashing vs resilient hashing" будет всплывать эта статья.

Поляризация трафика при балансировке

Вернёмся к ECMP. Забавно, что самая что ни на есть естественная сторона хэш-функции - выдавать одинаковый результат для одинаковых входных данных - может стать и помехой.

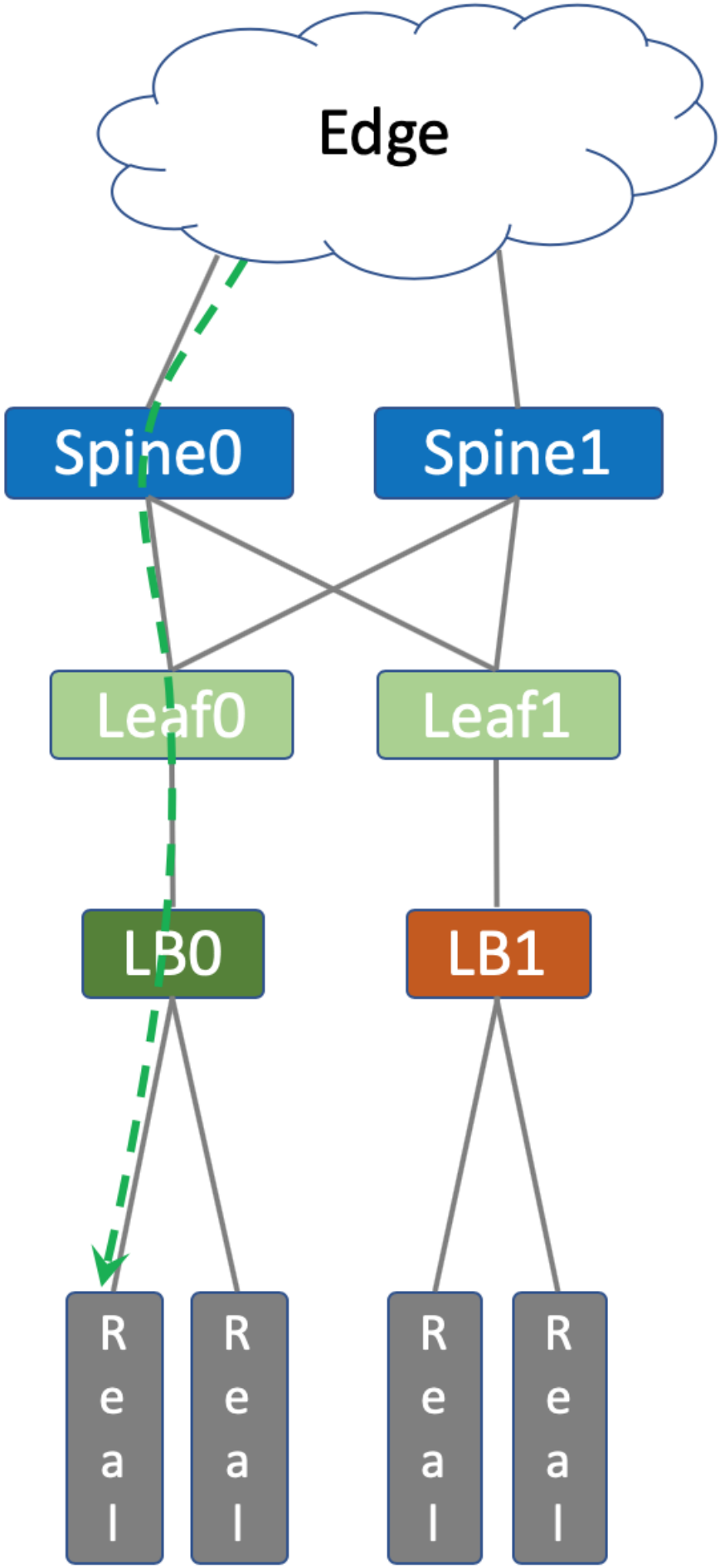


Именно это и происходит в случае последовательных балансировок. Трафик пришёл на SW0, разбалансировался между SW10 и SW11. А теперь его ещё и SW10 должен побалансировать. Ирония в том, что на SW10 трафик уже пришёл с определённым (отобранным) 5-Tuple, то есть поляризованный. И если на него натравить ту же хэш-функцию, то, предсказуемо, весь трафик разложится в одно плечо, а во второе - не попадёт ничего. Для выхода из этой ситуации в работу хэш-функции вносятся изменения. Они могут называться и работать по-разному: Salt, Seed, Shift - но их задача одна: выдать результат отличный от того, что получится на других устройствах.

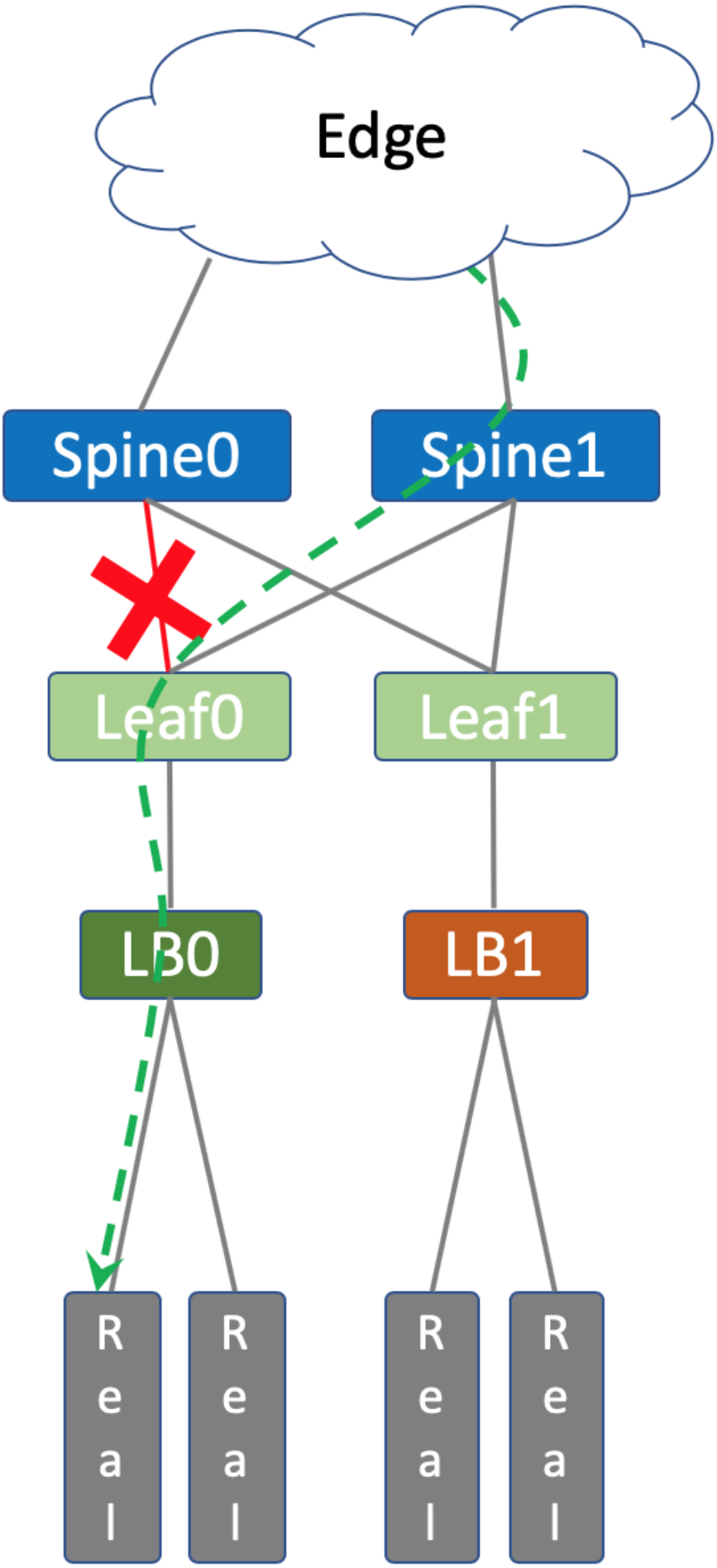
Обычно этот **Seed/Salt/Shift** генерируется случайным образом (или нет) при первом запуске устройства, и далее не меняется. Но тут всё зависит сугубо от вендора. Это опять же (забавно) идёт вразрез с предыдущим параграфом про эникаст, где нам хотелось иметь одинаковый результат на разных устройствах. Но тут тогда неплохо придерживаться такой политики: Shift/Salt/Seed один и тот же для устройств одного уровня, и разный - для разных. Если, конечно, производитель вообще позволяет настраивать.

Ещё про Anycast

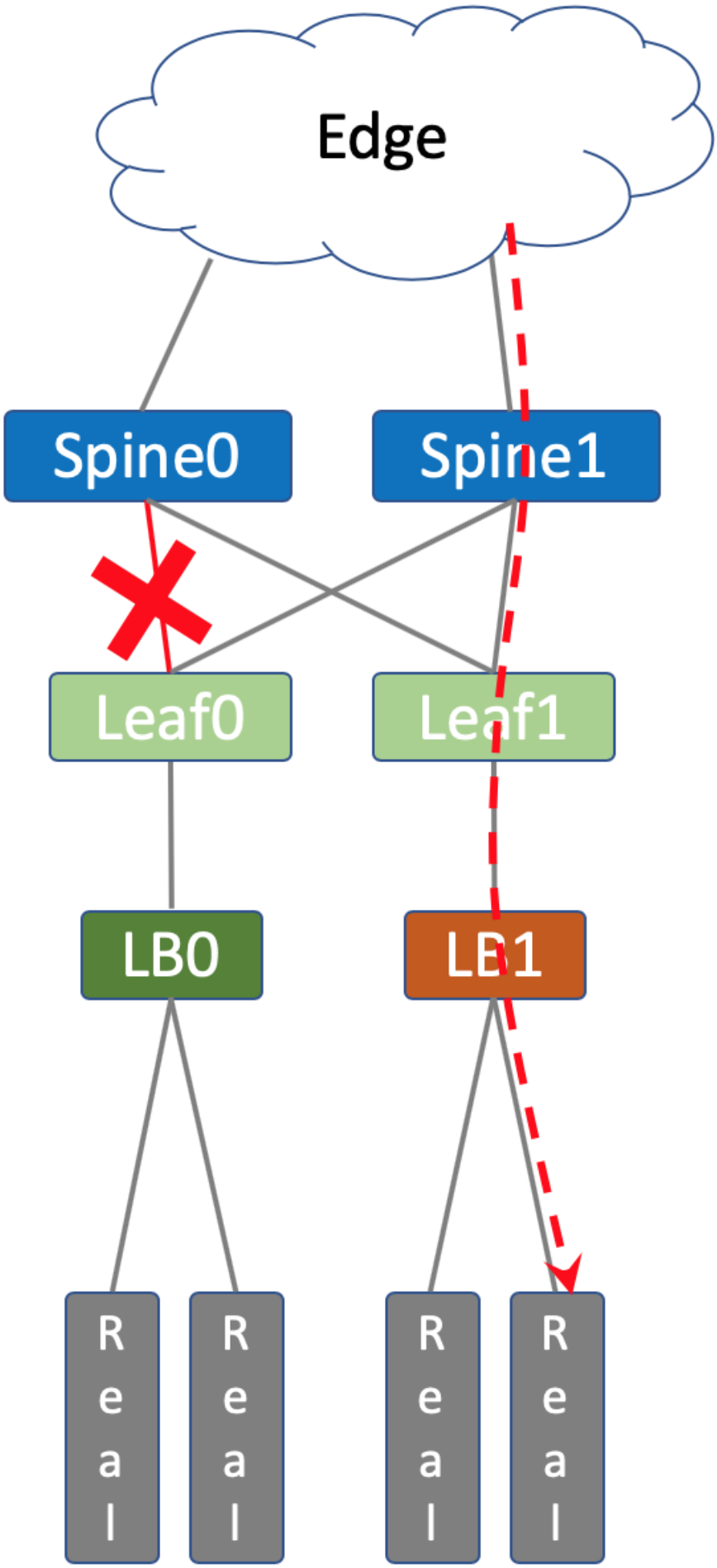
Тот, кто нам мешает, тот нам и поможет. Когда мы говорим про Anycast, есть интересные нюансы. Например, два балансировщика трафика - LB0 и LB1 - подключены в два разных Leaf-коммутатора. Каждый балансер отвечает за свою группу реальных бэкендов. Есть одна сессия, которая по своему хэшу пошла на Leaf0 и, соответственно, попала на LB0.



И вдруг падает линк Leaf0<->Spine0. Трафик сессии перетекает на Spine1, где от его 5-Tuple считается хэш. И вот, если хэш-функция одна и та же, и порты подключения Leaf0 и Leaf1 одни и те же, то сессия автоматически снова ляжет через Leaf0 на LB0 и, конечно, тот же самый реальных бэкенд.



Но если хэш получился другой, или соответствие путей хэшам получилось иным, то трафик сессии может попасть на LB1, а значит, другие бэкенды - и получим разрыв.



То есть, для этого сценария нам полезно использовать одни и те же порты для подключения Leaf к Spine.

MPLS Load Balancing

Не IP единым, как говорится. Много где бегает MPLS. Возможно, это не совсем относится к датацентровым сетям, но мы же не только про них говорим.

В MPLS-метке не так уж много разнообразия. Она указывает или на некстхоп, или на операции демультиплексирования/коммутации, и ничего не знает о внутренних протоколах пакета. Так, транспортная одинакова для всего трафика, идущего на конкретный FEC. Сервисная имеет чуть больше деталей. Однако и она скрывает за собой или вообще весь трафик одного VPN, или, в лучшем случае (если это L3), весь префикс. Понимая это, производители научили свои чипы заглядывать под заголовки MPLS. Но лишь под определённое их число. Кто-то может заглянуть под 3, кто-то - под 4. Максимум - под 5, не больше. Поэтому, если у вас на сети Carrier Supports Carrier с каким-нибудь TE Fast Reroute, вам придётся быть очень аккуратным при планировании балансировки.

> Ограничение количества меток объясняется тем, что ASIC"и обычно не поддерживают циклы и перебирать стек итеративно не получится. Да и в случае NP (сетевых процессоров), циклы непредсказуемо меняют задержку обработки пакета и пропускную способность, потому что заголовок придётся прогнать через контур неизвестное число раз - а это тоже нежелательно.

Но как понять, что нагрузка под заголовком MPLS - IP-пакет? Очень просто: анализируем первые 4 бита. Если в них 0x4 или 0x6, значит это IPv4 или IPv6 - и тогда берётся стандартный 5-Tuple (SIP, DIP, Protocol, SPort, DPort). Это делают все вендоры и не стесняются в этом признаться. Любой другой способ категоризации нижележащего протокола будет несравнимо дороже. Если дальше идёт Ethernet, то следующим полем будет DMAC. И, как бы тут, упаси боже, чтобы DMAC начинался на 0x4/0x6 - тогда такую шикарную балансировку можно получить, что свет белый не взвидишь. Поэтому для сервисов L2VPN может добавляться Control Word, который гарантирует, что в первых четырёх битах не будет 0x4/0x6.

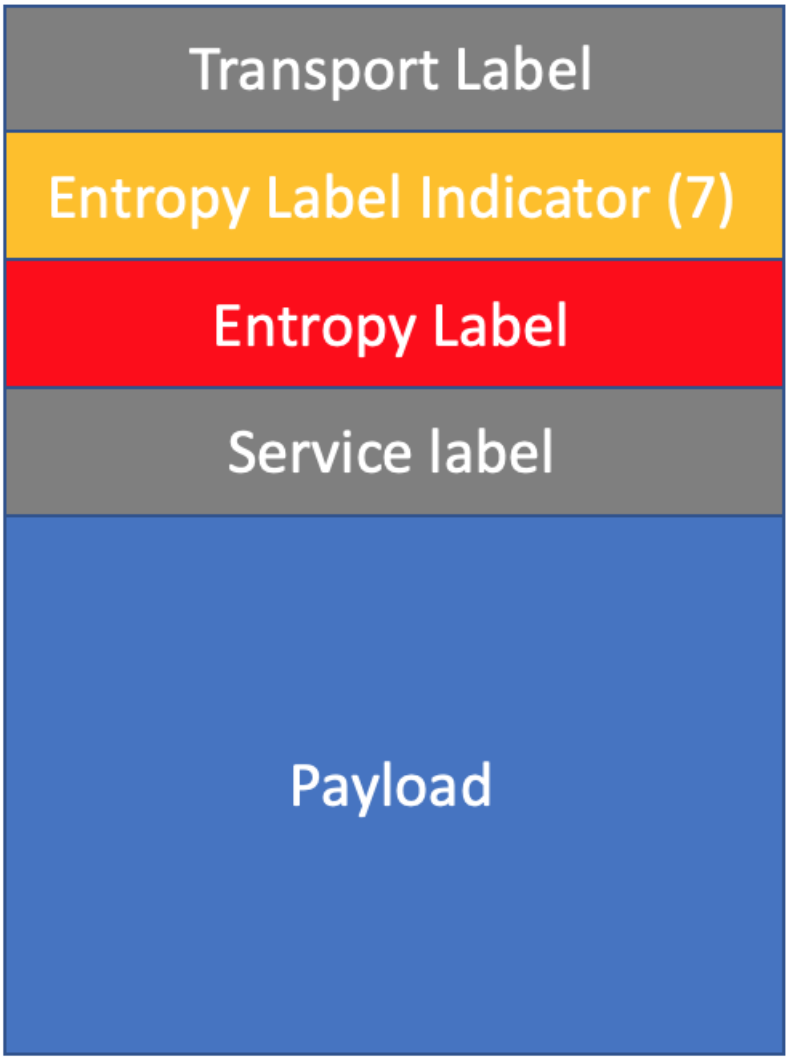
Некоторые вендоры умеют определять Ethernet-заголовок и, соответственно балансировать по 2-Tuple (DMAC, SMAC). В совсем плачевной ситуации, когда ничего не понятно с типом нагрузки, можно брать для расчёта хэша:

- Сколько-то битов нагрузки (например, 16).
- Самую нижнюю метку.
- Две нижние метки.
- Весь стек меток.
- Любые комбинации пунктов выше.

И всё же, что делать, если меток больше трёх, если тип нагрузки неизвестен и P-маршрутизатор не может определить, как балансировать? На помощь спешит **Entropy Label** RFC6790.

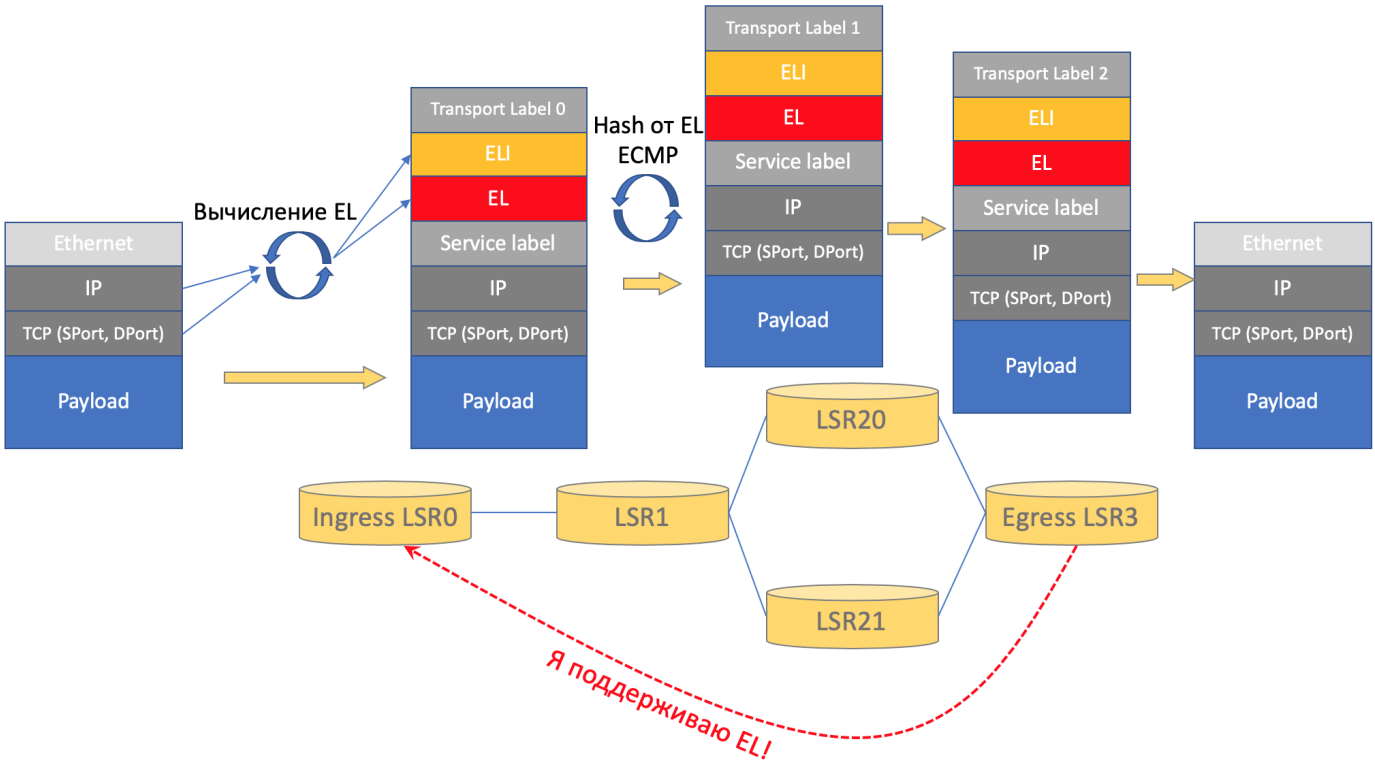
Entropy Label

Entropy Label (EL) - специальная метка, несущая информацию о нагрузке, позволяющая побалансировать MPLS-трафик более гранулярно и не допустить реордеринга. Грубо говоря, для каждой TCP-сессии будет генерироваться собственная EL и вставлять в стек меток. Поскольку в заголовке MPLS никакого специального флага указания, для чего метка служит, нет, прямо над EL в стеке меток вставляется ещё одна - **ELI - Entropy Label Indicator** - это зарезервированная метка **7**. То есть, любое транзитное устройство, видящее метку 7, или понимает, что это ELI, или не понимает, и тогда не обращает внимания.



Если устройство понимает ELI, то знает, что прямо под ней - EL, считает хэш от метки EL и балансирует трафик на основе него. И уже не нужно делать DPI до, собственно, тела пакета. Если же не понимает, то для него 7 - это просто какая-то метка. Если необходимо, заглядывает по возможности в нагрузку и балансирует на основе этой информации. И просто свопает транспортную метку, оставляя ELI/EL в стеке. Egress, очевидно, должен в любом случае уметь разбираться с EL/ELI.

Как это работает? При сигнализации туннеля Egress LSR сообщает Ingress LSR"у, что он умеет в Entropy Label и их можно ему слать (а можно и не слать) (в LDP это **Entropy Label Capability**, но вообще любой протокол распространения меток это умеет). Ingress LSR, в свою очередь, знает о нагрузке гораздо больше, чем любой транзитный LSR, поэтому он лучше понимает, как можно трафик балансировать. Кроме того, он работает на более низких скоростях, чем транзитные, и может позволить себе больше операций над пакетом. Поэтому Ingress LSR получает пакет и вешает сервисную метку. Далее, понимая, что Egress LSR готов обработать EL, формирует EL, сверху приклеивает ELI, потом транспортную - и в дальний путь. Соответственно, как я уже сказал выше, транзитные LSR могут обработать пакет независимо от того, поддерживают они EL или нет.



Я намеренно не описываю детали с PHP и другие варианты Entropy Label, чтобы не усложнять повествование. Но, говоря про внесение энтропии в стек меток, нельзя не упомянуть FAT Flow Label. FAT - потому, что Flow Aware Transport of psewdowires. FAT применим только к L2VPN-сервисам и использует специальную версию LDP для сигнализации возможности обрабатывать FAT-метку. Отличается она от EL тем, что устанавливается в самый низ стека под метку PW и, соответственно, только Egress PE её видит при обработке пакета - а уж он то знает, что её нужно попнуть. Транзитные P-маршрутизаторы для вычисления хэша используют эту метку, как самую нижнюю, без каких-либо подозрений. Проблемы у этого подхода 2 - во-первых, только для L2VPN, во-вторых, FAT-метка лежит на дне стека, а это как минимум 3 штуки (транспортная, сервисная и FAT), и, если P-маршрутизатор не может заглянуть больше, чем под 3, а у нас их больше, то будет неловко.

Мыши и слоны

Вопрос "Elephant vs Mice flows "стоит почти столько же, сколько существуют сети передачи данных. Одновременно с короткими TCP-сессиями открытия linkmeup.ru через сеть провайдера идут долгие тяжёлые закачки. Одновременно с HTTP-запросом на объект в Object Storage в сети ДЦ идёт массивная репликация БД. Но в сети провайдера по статистике количество одновременных сессий лежит в диапазоне от нескольких тысяч до десятков и сотен тысяч, среди которых большое количество и Mice, и Elephant flows. И это всё по ECMP может раскладываться сравнительно равномерно. В сети ДЦ ситуация радикально другая - количество сессий может быть и меньше тысячи, среди них буквально единицы-десятки Elephant flows. При статической балансировке велика вероятность, что массивные потоки лягут в один интерфейс и создадут очень неравномерную балансировку. Per-packet балансировка позволяет одинаково нагрузить все линки, но выливается в реордеринг. Как бы нам и один поток разложить в разные линки, и пакеты при этом доставлять упорядоченно? На помощь приходит статистика. Оказывается, что на маленьком масштабе времени TCP имеет взрывной характер - то есть, всплеск-пауза-всплеск-пауза-и т.д. Такие отдельные всплески внутри одного flow назвали **flowlet**.

И вот, если пауза между двумя соседними флоулетами больше определённого порога, то их можно безопасно отправить в разные линки, таким образом эффективно разложив один поток на несколько путей. Что же до длины паузы, то она должна быть больше, чем возможная разница во времени доставки пакетов по различным путям ECMP. Например, если эта разница лежит в диапазоне 300-500 мкс, то 1 мс паузы должно быть достаточно, чтобы исключить любой реордеринг.

Juniper называет эту технологию AFS - Adaptive Flowlet Splicing. Соответственно, при выборе пути

теперь учитывается не только статический хэш от заголовков пакета, но и загрузка интерфейсов и заполненность очереди. Чем менее загружен интерфейс и меньше пакетов в очереди, тем более вероятно, что при следующей возможности флоулет переключится в него.

Чем дальше в детали, тем ближе к чипо-вендорному разнообразию, но, если в общих словах, то реализуется это примерно так. Для того, чтобы отслеживать паузы, для каждого потока вводится элемент, называемый Hash Bucket Table. В каждой записи в нём есть поле с меткой времени, когда последний раз через него прогонялся пакет, и поле с некстхопом. При получении нового пакета, чип эту метку сравнивает с заданным временны́м порогом: если больше - то можно назначать новый интерфейс, если меньше, то слать в уже указанный в записи. По схожим принципам работает динамическая приоритизация пакетов (DPP - Dynamic Packet Prioritization). Но это уже совсем другая история.

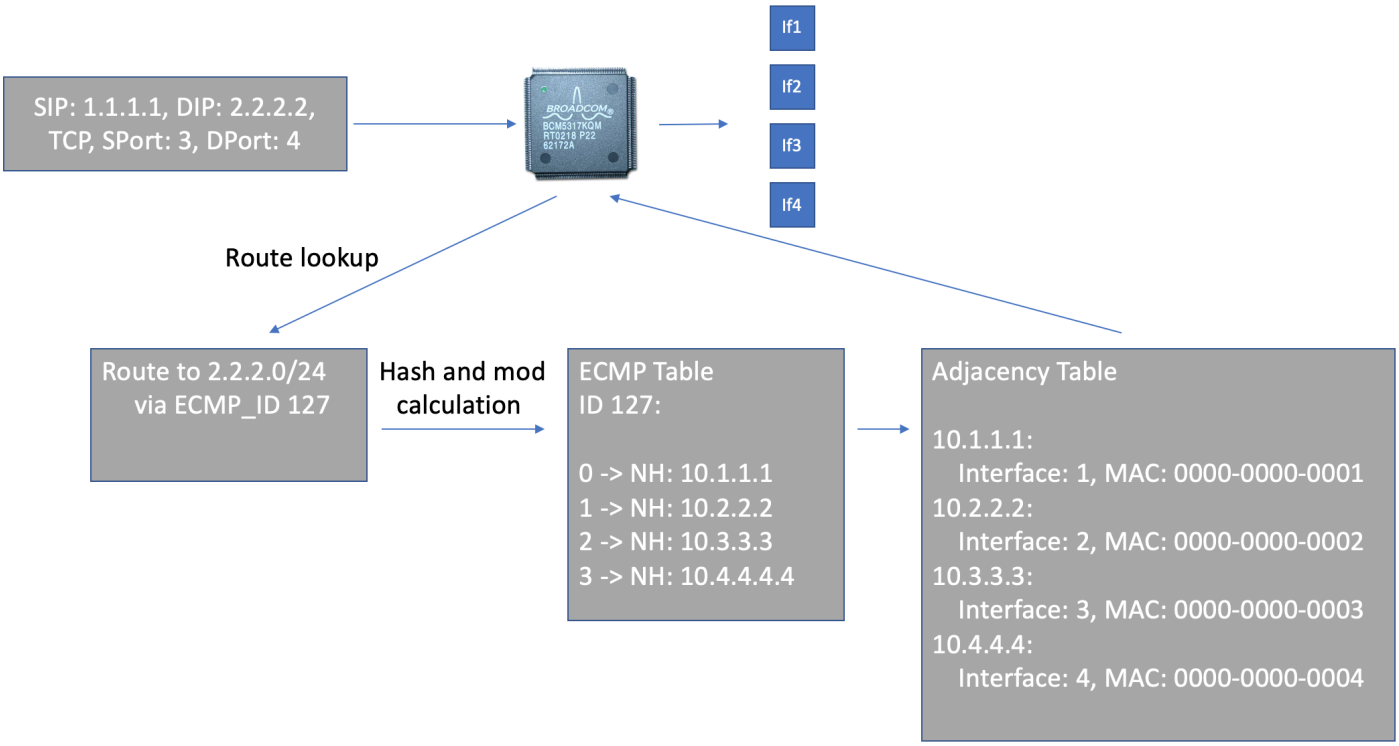
Теперь мы готовы обратиться к слову "равноценный" во фразе "Что, по сути своей, такое ECMP? Это **балансировка** трафика по **равноценным** путям".

Equal Cost

Не волнуйтесь, мы уже почти в конце романа и последняя глава не будет такой большой. Осталось только разобраться с тем, что же подразумевается под равноценными путями. В первую очередь, они, конечно, должны быть все из одного источника. То есть, за место в ECMP маршрутам IS-IS и OSPF придётся сражаться - вместе они там оказаться не могут. Тут всё просто - решение принимается по Administrative Distance. Далее, пути в пределах самого протокола должны быть равноценны. В случае OSPF, например, они должны быть одного типа (intra-area, inter-area, external type-1 или type-2) и, конечно, с одинаковой стоимостью. В случае BGP у маршрутов должны быть одинаковы следующие атрибуты:

- Weight/Preference/Metric/etc.
- Local Preference.
- AIGP.
- Длина AS Path.
- Origin.
- MED.
- EBGP или IBGP.

Зачастую, ECMP обычно выключен по умолчанию, и его нужно включать, явно указывая максимальное число путей для балансировки. Если число фактически доступных ECMP-путей больше, чем настроено, то к ним начинают применяться примерно те же критерии выбора лучшего, что и обычно, чтобы оставить только настроенное количество. В результате, из протокола в RIB импортируется набор равноценных путей. А из RIB они уже инсталируются в FIB и в ASIC"и/память. В некоторых случаях для такого экспорта нужна ещё отдельная конфигурация (так на джунипере это применение политики в секции routing-options -> forwarding-table). Каждый раз, как приходит на чип пакет, происходит лукап маршрута. Если чип находит маршрут указывает на ECMP-группу, он рассчитывает хэш от заголовков, согласно настройкам (2-Tuple, 5-Tuple). На основе результата выбирается соответствующий путь из группы.



Всё просто в первом приближении. Однако, тут на сцену выходит гонка за ресурсами и оптимизацией. В случае мощной фабрики на каждый маршрут легко иметь 8 и больше некстхопов. Если счёт маршрутов идёт на тысячи и десятки тысяч, очень легко упереться в ресурсы аппаратного FIB, поэтому прибегают к ухищрениям. Так появилась концепция ECMP-групп.

ECMP Groups

Если вы попытаетесь найти в Интернете описание того, что же такое ECMP-группы, то самое богатое, что встретите: ECMP-группа - это список уникальных некстхопов, на который ссылаются ECMP-маршруты (дословно с сайта Cumulus: "An ECMP group is a list of unique next hops that are referenced by multiple ECMP routes"). Хотите больше - это уже Programmer"s Guides для конкретных чипов с водяными знаками на страницах, подписями "Confidential" и открывающиеся по личному паролю. Пожалуй, этого краткого определения и достаточно.

Или нет? Мне лично оно непонятно. И каждый раз, как я на него смотрю, мне всё так же непонятно. Попробую сделать пару абзацев ECMP Groups Explained. И давайте для примера возьмём вот такую замысловатую сетоньку.

[ECMP и превратности балансировки на сетевом оборудовании]

Как видите, в ДЦ слева 4 спайна. Соответственно, с каждого лифа по 4 аплинка, то есть 4 некстхопа для маршрутов. Допустим, из другого ДЦ прилетает 100 маршрутов. В стабильной ситуации у каждого из 100 маршрутов 4 возможных некстхопа. У всех 100 - одинаковый список. Это означает, что все они формируют **одну ECMP-группу**.

[ECMP и превратности балансировки на сетевом оборудовании]

То есть, в FIB есть структура, содержащая 4 некстхопа, на которую указывают 100 маршрутов.

Когда происходит лукап адреса назначения в FIB, возвращается именно эта ECMP-группа. На основе заголовка пакета вычисляется значение хэш-функции, и для этого конкретного пакета выбирается определённый маршрут из этой группы и, соответственно, набор действий, которые нужно осуществить над пакетом.

Ещё раз: вместо того, чтобы хранить 4 некстхопа для каждого из 100 маршрутов, мы храним всего одну группу и ссылаемся на неё. Кроме сохранения ресурсов концепция ECMP-групп позволяет очень быстро вносить изменения в FIB - мы правим всего лишь одну запись, вместо перепрограммирования каждого маршрута.

Теперь добавим хаоса: в удалённом ДЦ вылетает линк между лиф-коммутатором и спайном, через который анонсировалось прежде 10 маршрутов. Теперь на локальной стороне 90 маршрутов доступны через всё те же 4 некстхопа, а 10 - через 3.

[ECMP и превратности балансировки на сетевом оборудовании]

Сразу после этого появляется **вторая ECMP-группа**. Первая состоит из тех же 4, и на неё ссылаются 90 маршрутов. Вторая состоит из 3 некстхопов, и на неё ссылаются 10 маршрутов. Соответственно, если выйдет из строя ещё один лиф на удалённой стороне, анонсировавший 20 маршрутов, то на локальных лифах будет три группы: Первая - 4 некстхопа, ссылаются 70 маршрутов Вторая - 3 некстхопа, ссылаются 10 маршрутов. Третья - **другие** 3 некстхопа, ссылаются другие 10 маршрутов.

[ECMP и превратности балансировки на сетевом оборудовании]

Несмотря на то, что число некстхопов во второй и третьей группах одинаковое, их состав разный - поэтому и группы разные. Это почти вырожденный пример того, как формируются ECMP-группы. Реальный мир посложнее. Но всё равно видно, что в случае простого IP нужно очень-очень сильно выпендриться, чтобы как-то значительно их число увеличить. При этом, стандартным для датацентровых коммутаторов является ограничение в 4096 ECMP-групп при максимум 64 NH в группе.

> На самом деле, если постараться, то ~~можно и FIB сломать~~ упереться можно. Флап линка Leaf<->Spine может вызывать высокую частоту обновления маршрутов. Если их много, вполне возможно, что не все они будут приезжать одновременно. И, несмотря на то, что в конечном состоянии должна остаться одна ECMP-группа (например, было 4 некстхопа, стало 3), из-за переходных периодов, могут начать создаваться новые ECMP-группы. У них есть какой-то lease-период, в течение которого они продолжают существовать даже при отсутствии маршрутов на них ссылающихся, поэтому новые группы могут создаваться быстрее, чем удаляются старые.

ECMP-группы при MPLS

А пикантности здесь добавляет MPLS. Как только мы добавим коммутацию по меткам, ситуация заиграет новыми красками. При голом IP некстхоп характеризовался параметрами (Egress_Interface, NH_MAC). В случае наличия метки, некстхоп уже состоит из трёх параметров: (Egress_Interface, NH_MAC, Label). A Label здесь уникален для префикса назначения и пути. То есть, даже если сети за удалёнными лифами будут резолвиться через лупбэки этих лифов, мы, возвращаясь к той же картинке, сходу получим 5 ECMP-групп. Внутри каждой группы по-прежнему сохраняется полный набор некстхопов и конкретный выбирается на основе хэша. Но именно из-за того, что метки, которые нужно записать в стек, разные, приходится держать отдельный набор NH под каждый маршрут.

[ECMP и превратности балансировки на сетевом оборудовании]

Иллюстрация из доклада Дмитрия Афанасьева с Yandex Next Hop 2019. Фактически, в случае MPLS на лифах получается отдельная ECMP-группа на каждый маршрут. Представим теперь 8 ДЦ по 512 стоек в каждом. Даже без всяких внешних маршрутов получается 4096 лупбэков и, соответственно, ECMP-групп. А когда группы кончаются, начинает отбрасываться трафик. Такая беда. Ситуация приобретает ещё более тонкие оттенки, когда мы говорим о Transit LSP. В этом случае, в зависимости от конфигурации и производителя, возможна и ситуация с отдельной ECMP-группой на Transit LSP, что увеличивает их кратно.

> И мы тут пока даже не затрагиваем иерархический ECMP, когда выделяются отдельные группы для оверлейного трафика, отдельные - для андерлейного.

Заключение

Была у меня идея, наконец, немного разобраться с тем, как работают сети. Так начался СДСМ. Потом СДСМ официально закончился. Но тут захотелось разобраться, что такое ECMP. Ну вы поняли. Разделаться полностью с вопросами балансировки задачи не стояло. Поэтому мы пролетели мимо VIP и туннелирования, DSR, L3/L4/L7 балансировщиков, GSLB, Global Unicast. Вопрос глобальной балансировки тянет на ещё одну статью, но для ознакомления рекомендую два видео, на которые я дал ссылки в следующем разделе.

Полезные ссылки

Статья маленькая, а ссылок много.

- Equal Cost Multipath Load Sharing - Hardware ECMP.
- ASR9000/XR: Load-balancing architecture and characteristics.
- Consistent Hash Rings Explained Simply
- Консистентные хэши (Consistent hashing).

- Load-balancing, what you should really know.
- RFC 6790. The Use of Entropy Labels in MPLS Forwarding.
- Maglev: A Fast and Reliable Software Network Load Balancer.
- Adaptive Flowlet Splicing – VCF"s Fine-Grained Adaptive Load Balancing Without Packet Re-ordering.
- RFC 6391, Flow-Aware Transport of Pseudowires over an MPLS Packet Switched Network.
- Балансировка нагрузки в Яндексе – проблемы роста от зарождения и до наших дней – Владимир Неверов.
- HighLoad 2018. Как обслужить миллиард пользователей и отдать терабит трафика / Игорь Васильев (ПАО Сбербанк России).
- BGP Anycast – просто о сложном.
- Who Are the Anycasters?

Спасибы

- Андрею Глазкову за рецензию.
- Дмитрию Афанасьеву за разъяснения.