QoS

Содержание

- Вступление.
- Модели QoS.
  - Best Effort.
  - Integrated Services.
  - Differential Services.
- Классификация и маркировка трафика согласно DiffServ.
  - Behavior Aggregate.
  - Interface-based.
  - Multi-Field.
- CoS.
- DSCP.
- Очереди.
- Ограничение трафика.
- Работа QoS на коммутаторах SNR.
  - Классификация и маркировка трафика на чипсете Realtek.
  - Классификация и маркировка трафика на чипсете Marvell.
  - Классификация и маркировка трафика на чипсете Broadcom.

Вступление

По мере увеличения скорости и развития технологий в сетях передачи данных, все больше и больше появлялось возможностей использовать эту скорость для передачи различных типов информации.

Выделенные каналы, которые были построены в США в начале и середине XX века для передачи голосовой информации, а также каналы для передачи телевизионного сигнала были дороги в обслуживании и аренде, находились в монополии нескольких компаний.

Дешевизна Ethernet привела к его массовому распространению по всему миру. Разрабатывался Ethernet с учетом того, что если США подвергнется ядерной атаке, то сеть все равно должна продолжать работать.

Появление Ethernet постепенно стало вытеснять старые линии передачи данных и активно внедряться в локальные сети офисов разных компаний. Золотое правило капитализма: где есть спрос — есть и предложение, а значит, если уже существует сеть внутри компании, построенная на Ethernet, то почему бы не использовать ее для передачи голосовой информации или видео вместо дорогих линий монополистов? Это привело к развитию других протоколов, например, VoIP.

В небольших сетях объемы передаваемой информации изначально были невелики, а скорости Ethernet хватало на всё, чтобы не волноваться насчет какой-то там приоритезации трафика. Однако, внутренние сети компаний росли, количество сотрудников увеличивалось, объемы информации неумолимо ширились. Это привело к потерям/задержкам/джиттеру информации внутри сети. Чтобы как-то управлять этими показателями появился QoS.

Quality of Service (QoS) - технология приоритизации трафика. Разделение трафика на классы и предоставление классам различных приоритетов в обслуживании.

Более важный трафик будет обработан быстрее, и задержки при его прохождении по сети будут минимальны.

Качество обслуживания (QoS) относится к любой технологии, которая управляет трафиком данных для уменьшения потери пакетов, задержки и джиттера в сети.

- **Потери**
  Когда сетевые каналы становятся перегруженными, маршрутизаторы и коммутаторы начинают отбрасывать пакеты. Когда пакеты теряются во время связи в реальном времени, например, при голосовых или видеовызовах, в этих сеансах могут возникать дрожание и разрывы в речи. Пакеты могут быть отброшены, когда очередь или линия пакетов, ожидающих отправки, переполняется.
- **Джиттер**
  Разница в задержках между доставкой последовательных пакетов.



Слишком сильный джиттер может ухудшить качество голосовой и видеосвязи.

- **Задержки**

Это время, необходимое пакету для перемещения от источника к месту назначения. Задержка должна быть как можно ближе к нулю. Если голосовой вызов по IP имеет большую задержку, пользователи могут столкнуться с эхом и перекрывающимся звуком.



Инструмент QoS просматривает заголовки пакетов для определения приоритетов пакетов. Заголовки пакетов — это биты информации, которые сообщают сетевым устройствам, что содержит пакет, куда он направляется (IP-адрес пункта назначения) и для чего он будет использоваться.

Модели QoS

Существуют три модели реализации QoS:

- Best Effort.
- Integrated Services.
- Differential Services.

Best Effort - модель QoS, в которой все пакеты имеют одинаковый приоритет, а доставка пакетов не гарантируется. Best Effort применяется, когда в сетях не настроены политики QoS или когда инфраструктура не поддерживает QoS. Это поведение устройств по умолчанию, т.е. QoS не используется, трафик обслуживается и передается на основе обрабатывающего его устройства — как получится. Нет гарантий по потерям, джиттеру и задержкам.

Integrated Services(IntServ) - модель QoS, которая резервирует полосу пропускания по определенному пути в сети. Приложения запрашивают у сети резервирование ресурсов, а сетевые устройства контролируют поток пакетов, чтобы убедиться, что сетевые ресурсы могут принимать пакеты.

Для реализации IntServ требуются устройства с поддержкой IntServ и используется протокол резервирования ресурсов (RSVP) для резервирования сетевых ресурсов. IntServ имеет ограниченную масштабируемость и высокое потребление сетевых ресурсов. Из-за того, что у данной модели очень плохая масштабируемость, а также то, что требуется выделение мощностей устройств в сети для резервирования ресурсов — данная модель не сильно прижилась.

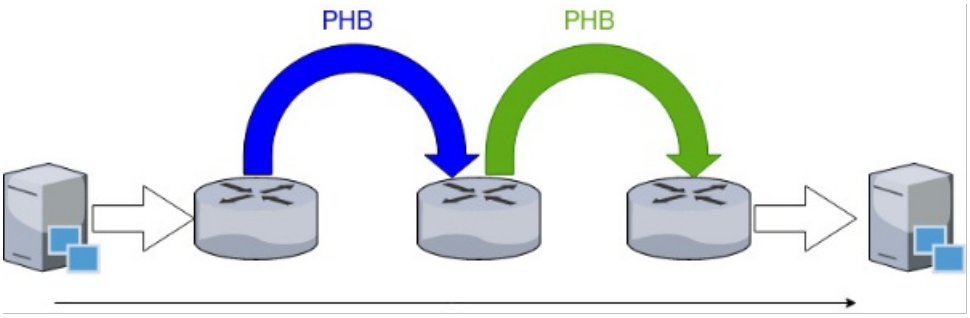
Differentiated Services (DiffServ) - модель QoS, в которой сетевые элементы, такие как маршрутизаторы и коммутаторы, настроены на обслуживание нескольких классов трафика с разными

приоритетами. Сетевой трафик должен быть разделен на классы в зависимости от конфигурации.

При поступлении пакета на устройство он классифицируется и к нему применяется набор условий, на основе которых пакету будет оказан разный уровень сервиса.

Например, голосовому трафику можно назначить более высокий приоритет, чем другим типам трафика. Пакетам назначаются приоритеты с использованием **Differentiated Services Code Point** (DSCP) для классификации. DiffServ также использует **per-hop behavior** для применения к пакетам методов QoS, таких как создание очередей и приоритизация.

Per-Hop Behavior(PHB): каждое промежуточное устройство в сети самостоятельно принимает решение о том, как обработать поступивший пакет.

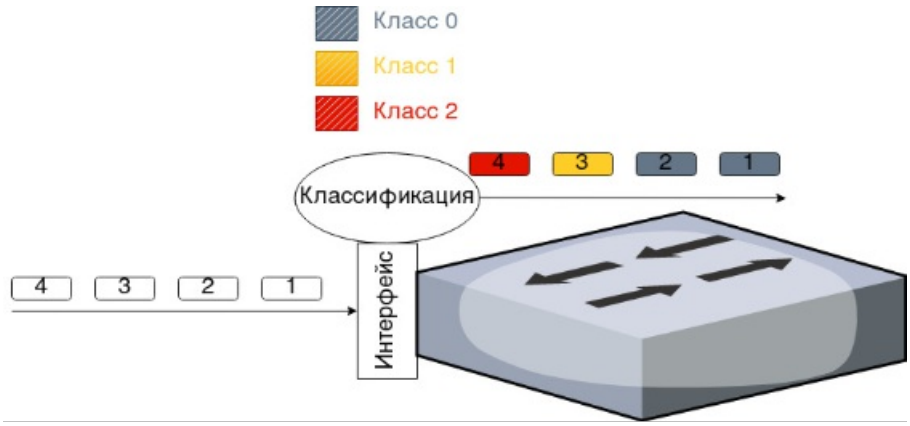


Каждый узел самостоятельно классифицирует поступающие пакеты, поэтому сначала нужно определить к какому классу относится пакет.

Именно данная модель QoS(DiffServ) получила наибольшее распространение, т. к. является масштабируемой.

Классификация и маркировка трафика согласно DiffServ

В зависимости от оборудования и его возможностей, мы определяем набор классов, которые сможем предоставить для трафика.



Для классификации трафика существует несколько вариантов:

1. Доверие существующей метки. **Behavior Aggregate** (**BA**). При поступлении на порт пакета, оборудование проверяет заголовок и, на основании поля заголовка, понимает к какому какому классу относится пакет и обрабатывает его соответствующе.
2. Классификация на основе интерфейса. **Interface-based** - все, что приходит на порт помещать в один какой-то определенный класс. То есть, мы классифицируем все, что поступает на порт каким-то определенным классом.
3. Присвоение класса на основе заголовков пакета. **Multi-Field** - при поступлении пакета мы не доверяем имеющейся маркировки и производим перемаркировку пакета.

Когда оборудование получает пакет, то, в большинстве случаев, оно опирается на тот заголовок, который будет использован для коммутации пакета. Например, если это ethernet-коммутатор, то будет использован 802.1p. Если это маршрутизатор, то IP DSCP.

Таким образом, администратор сети может определить набор классов для предоставления сервисов

и назначить им некоторое цифровое значение.

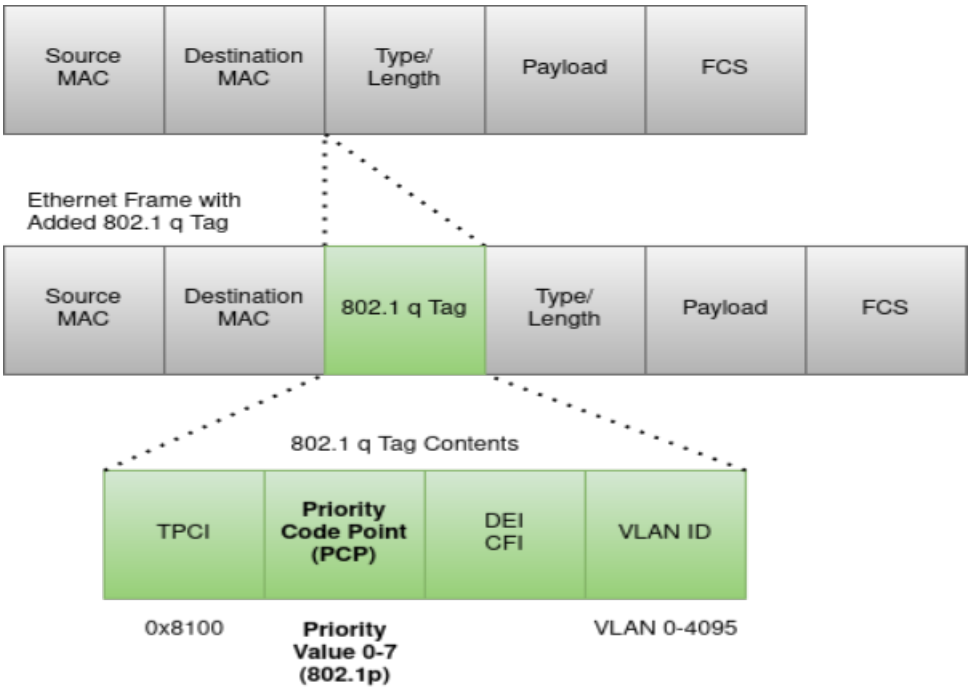
Для классификации трафика используются стандартные поля в заголовках:

- 3-битный(802.1p) Class of Service (CoS) в Ethernet;
- 6-битный Differentiated Services Code Point (DSCP) в IP.

CoS

Для маркировки пакетов на L2-уровне используется специальное поле в соответствии со стандартом 802.1p.

Данное поле имеет величину 3 бита и расположено в в 4-х байтовом заголовке 802.1Q.



Тег определяет следующие восемь уровней приоритета, которые подают сигналы сетевым устройствам относительно класса обслуживания, который должен получить кадр:

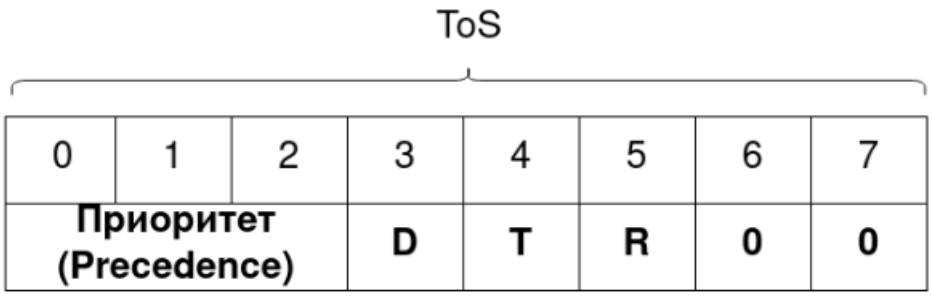
- **Priority 7 -** (самый высокий приоритет) **-** Трафик управления сетью.
- **Priority 6 -** Межсетевой контроль (маршрутизация).
- **Priority 5 -** Голос.
- **Priority 4 -** Видео.
- **Priority 3 -** Важные приложения (Call Signaling).
- **Priority 2** - Данные с высоким приоритетом (Critical data).
- **Priority 1 -** Best Effort.
- **Priority 0 -** (самый низкий приоритет) **-** Некритичный трафик, такой как: резервные копии, некоторая электронная почта и т. д.

DSCP

Для маркировки пакетов на L3-уровне используется специальное поле DSCP.

Архитектура дифференцированного обслуживания (DiffServ) предоставляет сетевым устройствам средства для классификации трафика на основе кода DiffServ (DSCP) и сопоставления трафика с определенной обработкой пересылки QoS.

Классификация трафика сначала поддерживалась 8-битным полем ToS (Type of Service) в заголовке IPv4. Поле было разделено на два подполя: три старших бита присваивали пакету один из восьми классов приоритета, пять младших битов определяли характеристики трафика.



D - delay. Требование ко времени задержки

T - throughput. Требование к пропускной способности

R - reliability. Требование к надежности

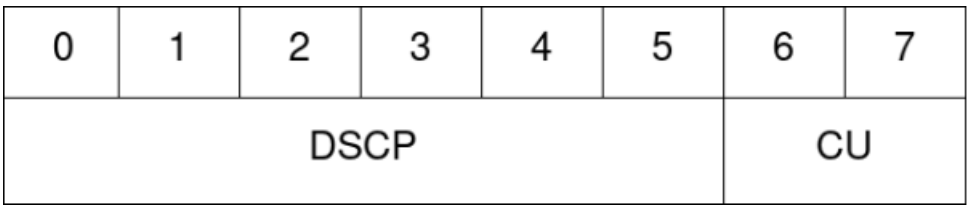
Приоритет принимает следующие значения:

111 — Network Control
110 — Internetwork Control
101 — CRITIC/ECP
100 — Flash Override
011 — Flash
010 — Immediate
001 — Priority
000 — Routine

Позже, поле ToS было переназначено для переноса DSCP в старших 6 битах и поля Explicit Congestion Notification (ECN) в младших 2 битах.

DSCP - это механизм, используемый для классификации сетевого трафика в IP-сетях. Он использует 6-битное (значения 0-63) поле дифференцированных услуг (поле DS или DSCP) в заголовке IP для целей классификации пакетов.

DSCP позволяет разделять потоки на 64 класса.



CU - Currently Unused.

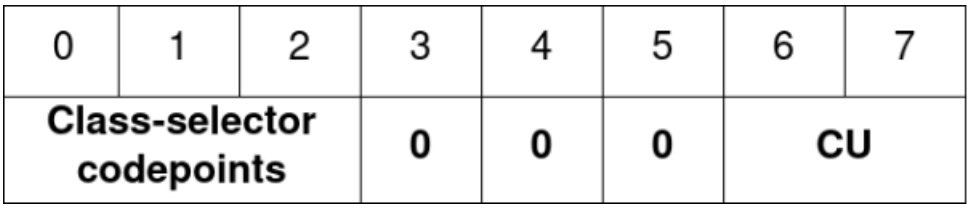
Per-Hop Behavior

Так как изначально устройства опирались на первоначальную классификацию поля ToS, то учитывали они только IP-приоритет (Precedence). После появления дифференцированных услуг(DSCP) встал вопрос совместимости этих устройств в сети. Устройства должны были понимать какой приоритет передается между ними.

Текущие определения значений DSCP включают четыре PHB:

Class selector PHB

Когда младшие 3 бита DSCP установлены на 000, селектор класса PHB обеспечивает обратную совместимость с IP-приоритетом на основе ToS.



В данном случае используются только три первых бита аналогично полю IP Precendece.
3 бита Class Selector позволяют определить 8 классов.
Кодовые точки выбора класса:

Class selector name	DSCP value	IP Precedence value	IP Precedence name
Default / CS0	000000	000	Routine
CS1	001000	001	Priority
CS2	010000	010	Immediate
CS3	011000	011	Flash
CS4	100000	100	Flash Override
CS5	101000	101	Critic/Critical
CS6	110000	110	Internetwork Control
CS7	111000	111	Network Control

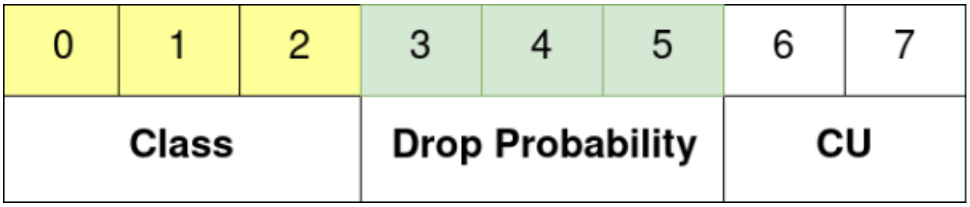
Assured forwarding (AF) PHB

Предоставляет четыре очереди для четырех классов трафика (AFxy): AF1y, AF2y, AF3y и AF4y. Для каждой очереди резервируется заданная полоса пропускания. Если объем трафика в определенной очереди превышает зарезервированную полосу пропускания для этой очереди, очередь заполняется и, в конечном итоге, приводит к отбрасыванию пакетов.

В каждом классе AFxy, y определяет предпочтение (или вероятность) отбрасывания пакета (Drop Probability). Некоторые пакеты отмечены минимальной вероятностью/предпочтительностью отбрасывания, некоторые – средней, а остальные – максимальной вероятностью/предпочтительностью отбрасывания. Всего есть 3 уровня для приоритета отбрасывания.

Обратите внимание, что большие числа здесь не лучше, чем меньшие, потому что они подразумевают более высокий приоритет отбрасывания.

Например, AF21 = класс трафика 2, приоритет отбрасывания 1. Значения класса трафика (1–4) имеют возрастающие значения приоритета, при этом трафик, помеченный как AF11, имеет более низкий приоритет, чем AF41.



Drop	Class 1	Class 2	Class 3	Class 4
Low	001010 (AF11)	010010 (AF21)	011010 (AF31)	100010 (AF41)
Medium	001100 (AF12)	010100 (AF22)	011100 (AF32)	100100 (AF42)
High	001110 (AF13)	010110 (AF23)	011110 (AF33)	100110 (AF43)

Старшие 3 бита поля DSCP установлены на 001, 010, 011 или 100 (они также называются AF1, AF2, AF3 и AF4), AF PHB используется для обслуживания гарантированной полосы пропускания.

Expedited forwarding (EF) PHB

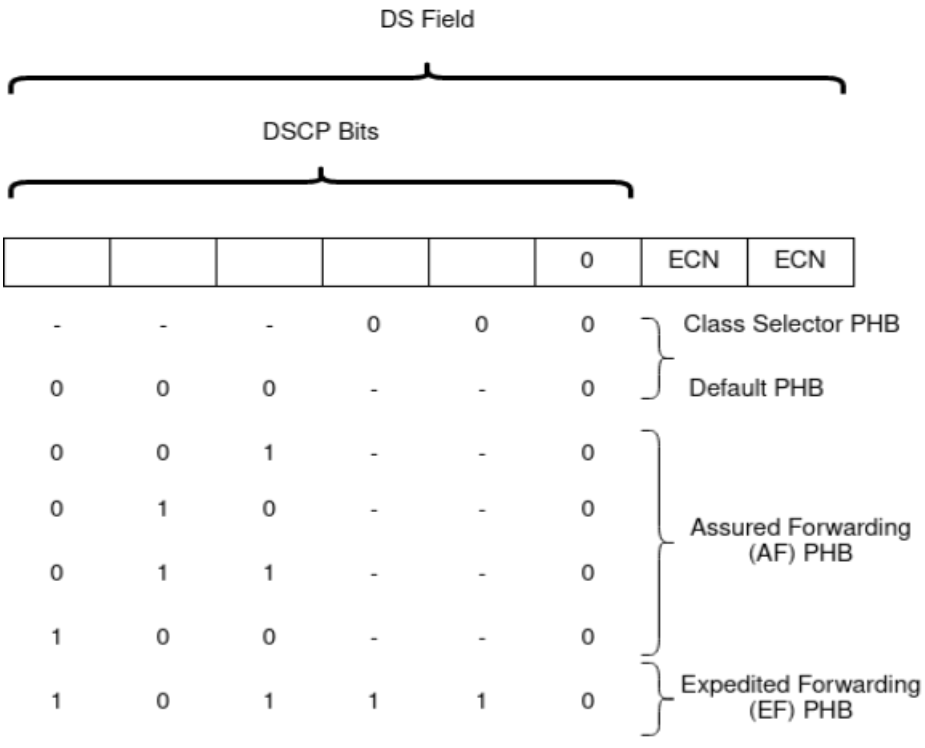
EF PHB обеспечивает сервис с низкой задержкой и должен свести к минимуму джиттер и потери. Полоса пропускания, выделенная для EF, должна быть ограничена, чтобы другие классы трафика также могли быть обработаны, иначе всю полосу займет EF. Очередь, выделенная для EF, должна иметь наивысший приоритет, чтобы назначенный ей трафик проходил быстро и не вызывал значительных задержек и потерь.

Expedited forwarding (EF) PHB - старшие 3 бита поля DSCP установлены на 101 (все поле DSCP установлено на 101110, десятичное значение 46), EF PHB обеспечивает обслуживание с малой задержкой.

Default PHB

3 старших бита поля DiffServ/DSCP установлены на 000, Default PHB используется для best effort (BE). Если значение DSCP-пакета не сопоставлено с PHB, оно, следовательно, назначается Default PHB.

Результирующее изображение, которое отражает все 4 PHB, которые соотносятся с полем DSCP:



Использование специальных полей в L2 и L3-заголовке предоставляет возможность промаркировать пакеты. Обработать их и поместить в определенную очередь в зависимости от их класса.

Очереди

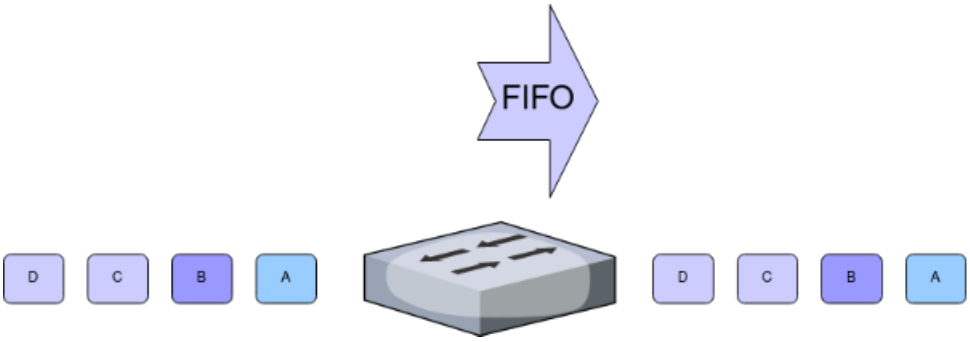
После того, как мы произвели классификацию пакетов они попадают в определенную очередь соответствующую их классу. В очередях пакеты будут обрабатываться по разному в зависимости от метода, который был выбран администратором. Например, более требовательный к задержкам голосовой или видеотрафик должен быть классифицирован приоритетной очередью. Объем очередей ограничен.

Различные методы обработки очередей:

Итак, трафик у нас распределился по очередям в соответствии со своим классом, чтобы покинуть устройство. Все пакеты будут отправлены в один интерфейс.

Если отдавать приоритет только определенному типу трафика, например, видео или VoIP, то остальной трафик так и не будет обработан. Поэтому необходимо по-разному извлекать пакеты из очередей, чтобы достичь определенного уровня сервисов.

- **FIFO (first in – first out)**
  При таком методе очереди отсутствуют, весь трафик обрабатывается в одной очереди. Пакеты обрабатываются в порядке их поступления, т.е. если пакет пришел первым на коммутатор, то он и будет обработан первым. Таким образом, на коммутаторе пакеты не разделяются по классам обслуживания, а значит и QoS не работает, а значит и нет DiffServ-обслуживания.
  Очевидные плюсы данного подхода: не надо заморачиваться за проектирование сети и выделение каких-то классов. Очевидные минусы: при заполнении очереди, рост задержек и джиттера у трафика, а как мы помним есть трафик, который к этим задержкам чувствителен.



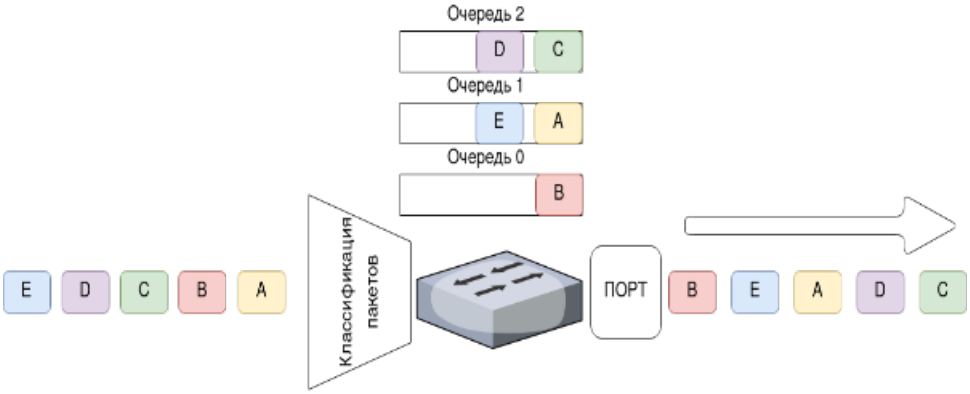
- **Strict Priority queuing (PQ – Priority Queuing)**
  В данном методе обработки очередей пакеты распределяются по очередям в соответствии их классу.

  Сначала обрабатываются пакеты из очереди с наивысшим приоритетом. Когда очередь с наивысшим приоритетом будет обработана, диспетчер обработки очередей переходит к следующей по приоритету очереди, и так далее, до очереди с наименьшим приоритетом.

  Однако, если в момент обработки менее приоритетной очереди придет пакет в более приоритетной очереди, диспетчер обработки пакетов переключится на более приоритетную очередь, и, только обработав ее, вновь вернется к менее приоритетной очереди.

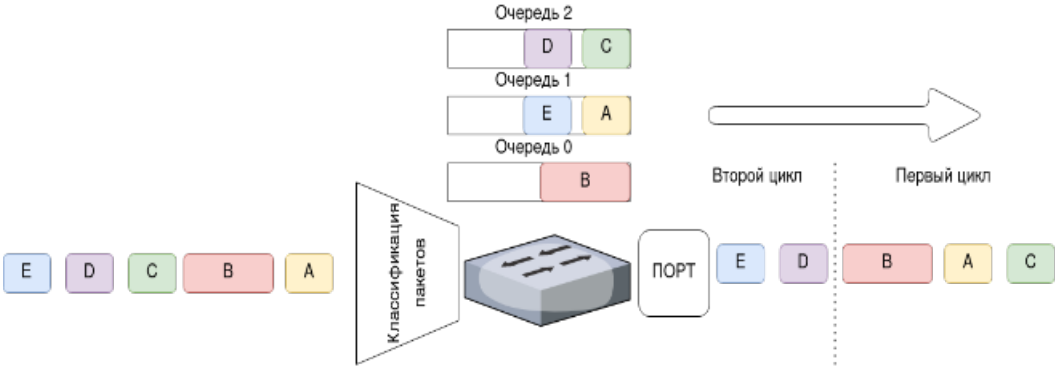
  Очевидные плюсы такого метода: пакеты, которым нужна маленькая задержка, будут обрабатываться в приоритетной очереди, а значит будут обработаны быстрее.

  Очевидные минусы: пока приоритетная очередь не будет полностью обработана, диспетчер не перейдет к обработке менее приоритетной очереди. Если в приоритетной очереди постоянно присутствуют пакеты, в менее приоритетной очереди будут скапливаться пакеты и никогда не будут обработаны, а значит при переполнении менее приоритетной очереди, пакеты начинают отбрасываться.



- **RR - Round-Robin**

  Данный метод балансировки предполагает извлечение по равному числу пакетов из каждой очереди каждый цикл опроса.



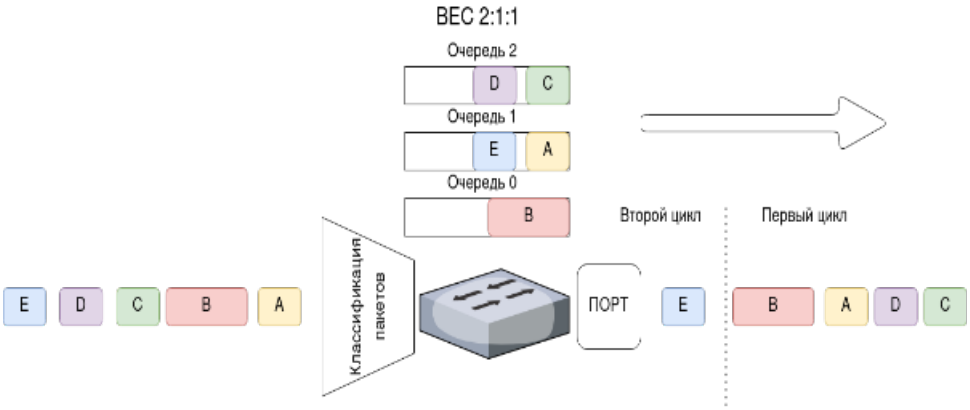
Метод сразу не сильно прижился, т.к. он является нечестным по отношению к различным потокам, например, потоки с пакетами большего размера могли утилизировать полосу больше. Плюсы: очень простой. Минусы: никакой справедливости распределения полосы.

- **WRR - Weighted Round Robin**

  Основная концепция WRR заключается в том, что он обрабатывает пакеты, которым требуется разная пропускная способность. WRR достигает этого, позволяя продвигать несколько пакетов из очереди каждый раз, когда диспетчер обращается к ней.

  WRR также решает проблему с PQ, в которой одна или несколько очередей может "голодать". WRR делает это, позволяя продвигать по крайней мере один пакет из каждой очереди, содержащей пакеты, в каждом цикле обработки очередей.

  Количество пакетов, которые будут обслуживаться за каждый цикл, определяется весом очереди. Таким образом, обрабатывается не равное количество пакетов, как в RR, а число пакетов кратное весу очереди.



Если рассмотреть пример, представленный выше, то для очереди 2 вес выставлен в 2, а 1 и 0 очереди вес имеет 1. Поэтому, в данному случае, т.к. у второй очереди вес больше по отношению к остальным очередям, то будет передано 2 пакета. Тогда как у 1 и 0 очереди будет передано по 1 пакету в первый цикл работы диспетчера.

- **DWRR — Deficit Weighted Round Robin**

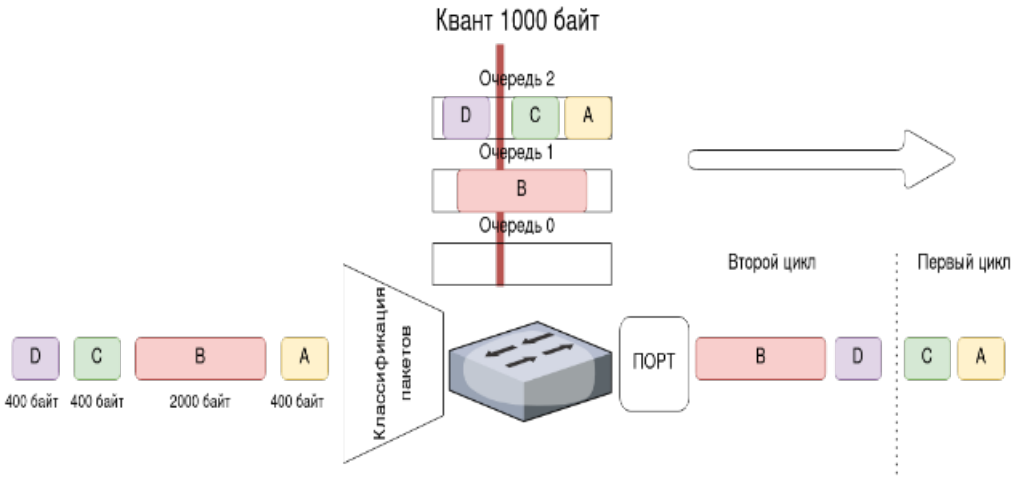
  Как можно заметить из метода обработки очередей RR, из нее вынимается по равному числу пакетов. Соответственно, в очередях с пакетами большего объема вынимается и больший объем информации, а значит утилизация полосы будет больше у пакетов с большим объемом - это нечестно по отношению к очередям, в которых могут быть пакеты меньшего объема. В конце концов может случиться так, что пакеты с меньшим объемом важнее, а мы не уделяем им должного внимания.

  Отчасти, метод WRR решает этот вопрос, т.к. мы самостоятельно задаем вес очередей, а значит для более приоритетной очереди мы можем задать вес больше. Однако, пакеты в сети неоднородные и может случиться так, что очереди с приоритетными пакетами утилизируют большую полосу, а другие очереди будут “голодать”. То есть, нет динамического распределения веса очередей.

  DRR позволяет более равномерно распределять нагрузку между очередями за счет использования **кванта**.

  Квант это своего рода кредит, который выдается для очереди в байтах. Если объем пакета меньше или равен кванту, то пакет пропускается. Если объем пакета больше кванта, то пакет не пропускается. После одного цикла опроса очередей квант увеличивается на изначальное значение.

  Пример: квант = 1000 байт. В 1 очереди три пакета по 400 байт. Во 2 очереди один пакет 2000 байт. В первый цикл опроса каждой очереди доступен квант в 1000 байт. Из 1 очереди два пакета по 400 байт “влазят” в квант 1000 байт, поэтому проходят: 1000-(400+400) = 200. Третий пакет не влазит в отведенный квант, т.к. 400+400+400 = 1200, а квант 1000 байт. Получается, что для первой очереди остается не задействовано еще 200 байт “кредита”. Это значение в 200 байт добавляется к следующему циклу опроса.
  Во 2 очереди пакет в первом цикле опроса вообще не будет обработан и не продвинется из очереди, т.к. его объем 2000 байт, что больше кванта в 1000 байт. Т.к. квант не был израсходован он переносится на второй цикл опроса. Во второй цикл опроса квант для первой очереди будет иметь уже значение в 1200 байт(1000 байт изначальный квант + 200 байт от предыдущего опроса). Оставшийся третий пакет из первой очереди проходит 1200 - 400 = 800 байт.
  Пакет из второй очереди тоже проходит, т.к. 1000 байт квант + 1000 байт квант оставшийся от предыдущего цикла опроса: 1000+1000 = 2000 байт квант - как раз хватает под объем пакета во второй очереди, который также имеет 2000 байт объема.



Так работает DRR. DWRR использует такой же принцип, однако, квант для каждой очереди выбирается свой, что позволяет более гибко настраивать обслуживание каждой конкретной очереди.

Таким образом, все очереди получают гарантированную полосу, независимо от размера пакетов в ней.

Ограничение трафика

Предположим, мы заключили договор с клиентом на предоставление услуг связи и подключили его в гигабитный порт на нашем оборудовании. Однако, по договору с клиентом мы предоставляем ему 300 Мбит/с.

Клиент не обязан заботиться о том, чтобы со своей стороны как-то ограничивать трафик, поэтому функцию по ограничению трафика мы можем произвести в точке подключения клиента.

Traffic Policing

Заключается в выполнении действия (обычно передача/прохождение) для пакетов, которые соответствуют указанной скорости, и выполнении другого действия (обычно отбрасывания) для пакетов, которые нарушают эту скорость.



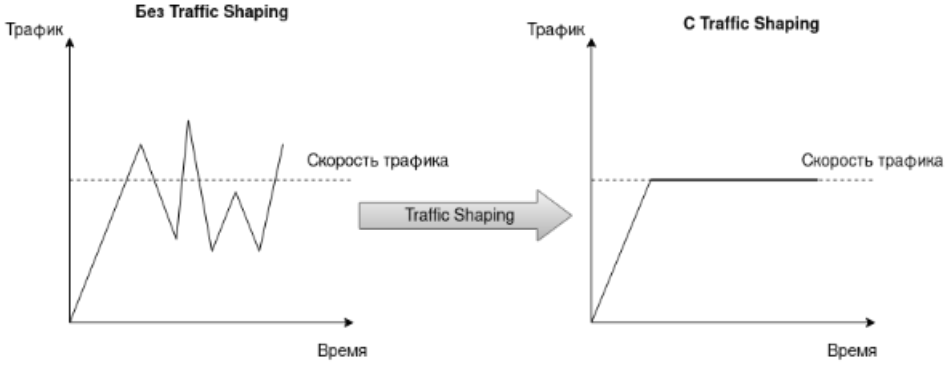
Однако, полисер может допускать всплески трафика. Поэтому все, что выходит за рамки полисера не обязательно будет отброшено. Допускаются всплески трафика или небольшое превышение заданной полосы - достигается это методом Token Bucket.

Полисер работает совместно с измерителем трафика (внутренняя логика работы коммутатора). Происходит покраска пакетов в зеленый, желтый или красный цвет. На основе цвета будут производиться действия над пакетами, например, пропустить пакет, перемаркировать пакет или отбросить пакет.

Traffic Shaping

Ограничение скорости с помощью шейпинга происходит за счет буферизации трафика. Трафик буферизируется и изымается из буфера с постоянной заданной скоростью. Если пакеты поступают со скоростью, которая ниже выходной скорости, то пакеты не скапливаются в буфере и пропускаются.

Если скорость поступления пакетов выше, чем скорость шейпера, то пакеты начинают накапливаться в буфере и отдаваться с постоянной скоростью. Поэтому в случае, если будет всплеск трафика, то он будет буферизирован и растянут по времени.



Работа QoS на коммутаторах SNR

Классификация и маркировка трафика на чипсете Realtek

Behavior Aggregate (BA)

Мы доверяем существующей метке.

По умолчанию коммутаторы SNR-S2962(5) и SNR-S2982(5)G доверяют значению CoS. Доверие метки CoS можно изменить в настройках порта с помощью команды **mls qos trust <cos/dscp>**.

Доверие метки CoS можно изменить в настройках порта с помощью команды **mls qos trust <cos/dscp>**.

Обратите внимание, что команда mls qos trust cos явно не видна на интерфейсе, т.к. она является командой по умолчанию:

```
sh run int e1/0/5
!
Interface Ethernet1/0/5
switchport mode trunk
!
sh mls qos interface e1/0/5
Ethernet1/0/5:
Default COS: 0
Trust: COS
…
```

```
sh run int e1/0/5
!
Interface Ethernet1/0/5
switchport mode trunk
!
sh mls qos interface e1/0/5
Ethernet1/0/5:
Default COS: 0
Trust: COS
…
```

Можно не доверять метке, которая есть в пакете. В этом случае пакет будет ассоциироваться с Default COS:

```
(config)#int e1/0/5
```

```
(config-if-ethernet1/0/5)#no mls qos trust cos
sh mls qos int e1/0/5
Ethernet1/0/5:
Default COS: 0
Trust:
```

После того, как мы отказались от доверия метки CoS эта информация стала явно отражаться на порту:

```
sh run int e1/0/5
!
Interface Ethernet1/0/5
no mls qos trust cos
switchport mode trunk
!
```

Также, можно установить доверие метки DSCP вместо CoS:

```
sh mls qos int e1/0/5
Ethernet1/0/5:
Default COS: 0
Trust:
…
(config)#int e1/0/5
(config-if-ethernet1/0/5)#mls qos trust dscp
!
Interface Ethernet1/0/5
no mls qos trust cos
mls qos trust dscp
switchport mode trunk
!
sh mls qos inte e1/0/5
Ethernet1/0/5:
Default COS: 0
Trust: DSCP
…
```

Если режим доверия установлен одновременно и для значения CoS, и для значения DSCP - приоритет отдается значению DSCP.

```
(config)#int e1/0/5
(config-if-ethernet1/0/5)#mls qos trust cos
(config-if-ethernet1/0/5)#mls qos trust dscp
!
Interface Ethernet1/0/5
mls qos trust dscp
switchport mode trunk
!
sh mls qos int e1/0/5
Ethernet1/0/5:
Default COS: 0
Trust: COS DSCP
…
```

Итак, к нам поступил пакет и, в зависимости от настройки, которая у нас есть на интерфейсе, мы доверяем метке CoS / DSCP / не доверяем метке вообще.
Дальнейшее продвижение пакета внутри коммутатора будет происходить на основе внутреннего соответствия метки (internal priority).
Если значения внутренних приоритетов не менялись администратором сети, то они совпадают с изначальными значениями меток приоритета.

Посмотреть внутренние значения приоритета можно с помощью команды ‘sh mls qos maps’ в глобальной конфигурации:

```
sh mls qos maps
Ingress COS-TO-Internal-Priority map:
COS:  0   1   2   3   4   5   6   7
----------------------------------------
INTP: 0   1   2   3   4   5   6   7
```

Ingress DSCP-TO-Internal-Priority map:
d1 : d2 0 1 2 3 4 5 6 7 8 9
0: 0 0 0 0 0 0 0 0 1 1
1: 1 1 1 1 1 1 2 2 2 2
2: 2 2 2 2 3 3 3 3 3 3
3: 3 3 4 4 4 4 4 4 4 4
4: 5 5 5 5 5 5 5 5 6 6
5: 6 6 6 6 6 6 7 7 7 7
6: 7 7 7 7

В ‘Ingress COS-TO-Internal-Priority map’ у нас идет соответствие CoS к его INTP. В нашем случае администратор сети не менял значения. На порту, куда поступил пакет, мы доверяем метке CoS. Если придет пакет с CoS=2, то внутренняя метка у него тоже будет 2.

Далее, пакет должен поступить на исходящий интерфейс на коммутаторе. Предположим, наш исходящий интерфейс номер 4, пакет должен попасть в соответствующую исходящую очередь согласно его внутреннему приоритету (Internal-Priority).

Значения Internal-Priority соответствуют значению исходящей очереди:

sh mls qos interface e1/0/4
Ethernet1/0/4:
Default COS: 0
Trust: COS
Egress Internal-Priority-TO-Queue map:
INTP: 0 1 2 3 4 5 6 7

Queue: 0 1 2 3 4 5 6 7

Если пакет имеет INTP = 2, то он попадет в Queue = 2.

Все это также справедливо, если мы доверяем метке DSCP: к нам на порт поступил пакет с DSCP = 46. На порту, куда поступил пакет, есть команда доверия метки DSCP:

На порту, куда поступил пакет, есть команда доверия метки DSCP:

!
Interface Ethernet1/0/5
mls qos trust dscp
switchport mode trunk
!

Смотрим, какой Internal-Priority получит пакет на основе Ingress DSCP-TO-Internal-Priority map:

d1 : d2 0 1 2 3 4 5 6 7 8 9
0: 0 0 0 0 0 0 0 0 1 1
1: 1 1 1 1 1 1 2 2 2 2
2: 2 2 2 2 3 3 3 3 3 3
3: 3 3 4 4 4 4 4 4 4 4
4: 5 5 5 5 5 5 5 5 6 6
5: 6 6 6 6 6 6 7 7 7 7
6: 7 7 7 7

На первый взгляд, ничего непонятно для человека, который впервые столкнулся с данной таблицей. Давайте разберемся: поле DSCP может иметь значение от 0 до 63 (2 в 6 степени, т.к. поле DSCP занимает 6 бит в заголовке IPv4).

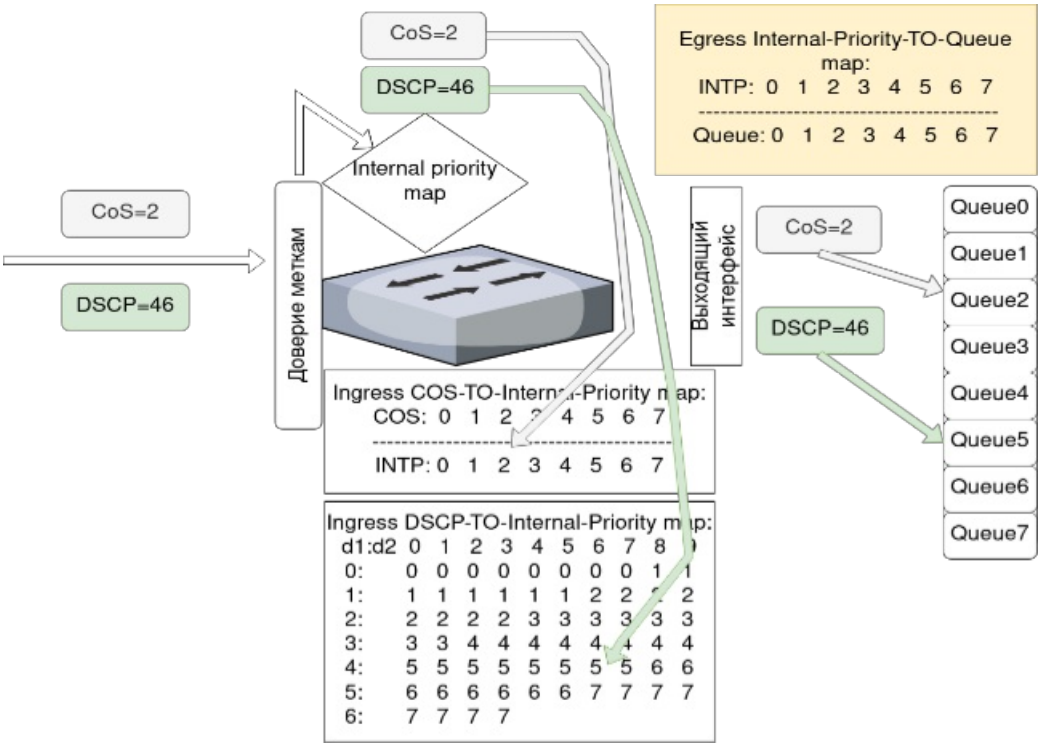
DSCP = d1d2: 00, 01...63. Таким образом, d1 могут принимать значения от 0 до 6, а значения d2 от 0 до 9. Пересечение в таблице d1 и d2 формирует значение самого DSCP и очередь, в которую он попадет.

```
Ingress DSCP-TO-Internal-Priority map:
d1 : d2  0   1   2   3   4   5   6   7   8   9
0:       0   0   0   0   0   0   0   0   1   1
1:       1   1   1   1   1   1   2   2   2   2
2:       2   2   2   2   3   3   3   3   3   3
3:       3   3   4   4   4   4   4   4   4   4
4:       5   5   5   5   5   5   5   5   6   6
5:       6   6   6   6   6   6   7   7   7   7
6:       7   7   7   7
```

На приведенном выше примере мы видим, что пакет с DSCP = 46 будет иметь Internal-Priority = 5, а значит он попадет в исходящую очередь Queue: 5.

Egress Internal-Priority-TO-Queue map:
INTP: 0   1   2   3   4   5   6   7

Queue: 0   1   2   3   4   5   6   7



Значение DSCP у пакета может быть изменено на коммутаторе. Для этого существует таблица ‘Ingress DSCP-TO-DSCP map’:

[Изображение недоступно]

Как мы видим из вывода данной информации, у нас идет соотношение один в один, т.е. при поступления пакета с DSCP = 26, в таблице у него будет такое же соответствие. При изменении соответствия DSCP на коммутаторе, например, DSCP 26 на DSCP 46:

(config)#mls qos map dscp-dscp 26 to 46

Мы получим следующую таблицу:

[Изображение недоступно]

После этого DSCP 26 перестает существовать на коммутаторе и все пакеты с DSCP 26 будут изменены на DSCP 46. Однако, пакеты будут помещены в исходящую очередь на основе первоначального DSCP, с которым поступил пакет, а именно 26, а значит по карте DSCP-TO-Internal-Priority map он попадет в 3 очередь. А на выходе из очереди пакет уже будет иметь DSCP 46.

Interface-based

Все, что приходит на порт, помещаем в один какой-то определенный класс. Предположим, мы уверены, что к порту подключено какое-то определенное устройство или мы знаем, какой трафик будет поступать на порт. Мы бы хотели весь трафик красить с определенной меткой CoS или DSCP.

Для этого мы воспользуемся policy:

```
(config)#class-map NAG
(config-classmap-nag)#match vlan 5
(config)#policy-map NAG
(config-policymap-nag)#class NAG
(config-policymap-nag-class-nag)#set cos 3
```

или

```
(config)#policy-map NAG
(config-policymap-nag)#class NAG
(config-policymap-nag-class-nag)#set ip dscp 46
(config)#int e1/0/7
(config-if-ethernet1/0/7)#service-policy input NAG
sh run
class-map NAG
match vlan 5
!
policy-map NAG
class NAG
set cos 3
exit
Interface Ethernet1/0/7
service-policy input NAG
switchport mode trunk
switchport trunk allowed vlan 5
```

Как будет обработан пакет и в какую очередь он будет помещен, зависит от того, доверяем ли мы метке CoS/DSCP или нет.

Если мы доверяем метке CoS/DSCP:

1. Пакет поступает на порт.
2. Коммутатор доверяет метке, с которой он поступил.
3. На основе карт ‘Ingress COS-TO-Internal-Priority map’ или ‘Ingress DSCP-TO-Internal-Priority map’ пакету присваивается INTP. На основе INTP пакет помещается в исходящую очередь.
4. Происходит перемаркировка пакета на основе policy, который был задан на порту, куда изначально поступил пакет.
5. Пакет покидает коммутатор с новой меткой, заданной в policy.

Таким образом, сначала происходит распределение пакетов по очередям на основе пришедшей метки и только потом метка будет изменена. То есть пакет будет обработан на основе пришедшей метки, а не той метки, которая была задана в policy.

Если мы не доверяем метке CoS/DSCP:

1. Пакет поступает на порт.
2. Коммутатор не доверяет метке, с которой он поступил.
3. Пакет ассоциируется с дефолтным CoS, который равен 0.
4. Все пакеты попадают в низкоприоритетную очередь.
5. Происходит перемаркировка пакета на основе policy, который был задан на порту.
6. Пакет покидает коммутатор с новой меткой, заданной в policy.

Таким образом, если к нам на порт придет пакет, например, с CoS = 2, то пакет сохранит метку CoS, но попадет все равно в низкоприоритетную очередь.

Если к нам на порт придет пакет с CoS = 2 и на порту будет полисер, в котором указано, что все пакеты будут перемаркированы в CoS=3, то пакет все равно будет помещен в низкоприоритетную очередь, а на выходе уже будет иметь CoS = 3.

[Изображение недоступно]

Multi-Field

При поступлении пакета мы не доверяем имеющейся маркировки и производим перемаркировку пакета.

При поступлении пакета мы не доверяем имеющейся маркировки и производим перемаркировку пакета.

```
!
ip access-list extended nag
  permit icmp any-source any-destination
  exit
ip access-list extended nag2
  permit ip any-source any-destination
  exit
!
class-map nag
match access-group nag
!
class-map nag2
match access-group nag2
!
policy-map nag
class nag
set cos 4
exit
class nag2
set ip dscp 46
set internal-priority 2
exit
!
(config)#int e1/0/5
(config-if-ethernet1/0/5)#no mls qos trust cos
(config-if-ethernet1/0/5)#no mls qos trust dscp
!
Interface Ethernet1/0/5
no mls qos trust cos
service-policy input nag
switchport mode trunk
switchport trunk allowed vlan 3
!
```

В данном примере мы выделяем из потока данных пакеты ICMP и пакеты IP, устанавливаем им различные метки. На порту мы не доверяем никаким меткам, поэтому все пакеты будут обработаны с дефолтной нулевой меткой CoS = 0.

Затем, пакеты будут обработаны согласно ‘Ingress COS-TO-Internal-Priority map’ и помещены в одну очередь:

```
 Ingress COS-TO-Internal-Priority map:
COS:  0   1   2   3   4   5   6   7
-----------------------------------------
INTP: 0   1   2   3   4   5   6   7
```

Однако, мы можем в полисере использовать ‘set internal-priority’, который явно укажет на то, с каким INTP обработать пакет внутри коммутатора. Таким образом, если мы установим internal-priority для пакетов в 2, то пакет попадет в исходящую очередь соответствующую INTP = 2.

Policing

На коммутаторах с чипом realtek, policy очень прост в возможностях и обладает небольшим функционалом. Полисер можно применять только в направлении in.

В качестве примера рассмотрим один из вариантов его настройки. Мы будем весь трафик, который поступает на порт во влане 3, ограничивать по скорости в 10 мбит/с.

Полисер состоит из двух частей:

1. **<bits_per_second>** - committed information rate (CIR) – скорость передачи данных, настраивается в Кбит/с.
2. **<normal_burst_bytes>** - committed burst size (CBS) – допустимый всплеск поступления информации, превышающий CIR, измеряется в Кбайт.

Policy burst (CBS) – для него можно настроить два значения и в дальнейшем применять их в полисере. Значение по умолчанию 1024.

В итоге, полисер будет выглядеть следующим образом:

policy <bits_per_second> burst-group <burst-group-id>

Настроим наш полисер:

```
(config)#policy burst 1 1000
(config)#class-map nag
(config-classmap-nag)#match vlan 3
!
(config)#policy-map nag
(config-policymap-nag)#class nag
(config-policymap-nag-class-nag)#policy 10000 burst-group 1
!
(config)#int e1/0/3
(config-if-ethernet1/0/3)#service-policy input nag
!
Interface Ethernet1/0/3
service-policy input nag
switchport mode trunk
!
#sh mls qos int e1/0/3
Ethernet1/0/3:
Default COS: 0
Trust: COS
Attached Policy Map for Ingress: nag
...
#sh policy-map nag
Policy Map nag, used by 1 time(s)
  Class Map name: nag
policy CIR: 10000 CBS: 1000
    conform-action:
     transmit
    exceed-action:
     drop
```

Очереди

Коммутаторы SNR поддерживают несколько способов обработки очередей:

- SP.
- WRR.
- WDRR.

Очереди настраиваются в рамках конфигурации самого интерфейса:

```
(config)#int e1/0/7
(config-if-ethernet1/0/7)#mls qos queue algorithm ?
  sp Strict priority
  wdrr  Weighted deficit round robin
  wrr   Weighted round robin
```

По умолчанию способ обработки очередей: Queue Algorithm: WRR.

Проверить соответствие INTP очереди, какой метод обработки очереди сейчас конфигурирован на порту и какие веса имеют очереди можно с помощью команды sh mls qos int <interface>:

```
#sh mls qos int e1/0/7
Ethernet1/0/7:
Default COS: 0
Trust: COS
Egress Internal-Priority-TO-Queue map:
INTP:    0   1   2   3   4   5   6   7
-----------------------------------------
Queue: 0   1   2   3   4   5   6   7
Queue Algorithm: WRR
Queue weights:
Queue  1 2 3 4 5 6 7 8
-----------------------------------------------------
WrrWeight  1 2 3 4 5 6 7 8
WdrrWeight 1 2 4 8 16 32 64 64
```

Здесь мы хорошо видим, что у нас есть прямое соотношение между INTP и Queue, т.е., например, если по внутренней логике работы коммутатора пакету при его классификации был присвоен INTP = 1, то он попадет в 1 Queue (исходящую естественно). После чего, пакеты в очередях будут обработаны согласно алгоритмам.

Если мы хотим при алгоритмах WRR или WDRR выделить приоритетную очередь (SP), то в этом случае вес этой очереди надо выставить в 0.

```
(config)#int e1/0/7
config-if-ethernet1/0/7)#mls qos queue wrr weight 1 0 3 4 5 6 7 8
sh mls qos int e1/0/7
Ethernet1/0/7:
Default COS: 0
Trust: COS
Egress Internal-Priority-TO-Queue map:
INTP:    0   1   2   3   4   5   6   7
-----------------------------------------
Queue: 0   1   2   3   4   5   6   7
Queue Algorithm: WRR
Queue weights:
Queue  1 2 3 4 5 6 7 8
-----------------------------------------------------
WrrWeight       1 0 3 4 5 6 7 8
WdrrWeight      1 2 4 8 16 32 64 64
```

Обратите внимание на строку WrrWeight, вес второй очереди изменился на 0. Все пакеты, которые будут попадать в эту очередь, будут обработаны в приоритете, т.е. 2 выходящая очередь становится SP.

Важно отметить, что в выводе веса очередей они начинаются не с 0, а с 1, т.е. 1…8, тогда как соотношение INTP к Queue начинается с 0…7. Поэтому, если пакет соответствовал INTP = 1, то он соответствует второй очереди в рамках обработки алгоритма.

Классификация и маркировка трафика на чипсете Marvell

На коммутаторах с чипсетами Marvell логика работы QoS расширена и отличается от коммутаторов на чипсете Realtek.

Рассмотрим пример:

```
sh mls qos int e1/0/7
Ethernet1/0/7:
Default COS: 0
Default int-Prio: 0
Trust:
Pass-through-cos:NONE
Pass-through-dscp-exp:NONE
Queue Algorithm: WDRR
Queue weights:
Queue  1 2 3 4 5 6 7 8
----------------------------------------------------------
WdrrWeight     1 1 1 1 1 1 1 0
```

1. По умолчанию, мы не доверяем никакой метке на порту.
2. Алгоритмов обработки очередей у нас поддерживаются два: SP и WDRR, ну и, соответственно, комбинация SP+WDRR.
3. Есть такое расширенное понятие как int-Prio, которое включает в себя значения от 0 до 119. Это внутренняя метка на основе которой обрабатывается пакет внутри коммутатора.
4. Есть такие понятия как Pass-through-cos и Pass-through-dscp-exp - сохранение метки, с которой поступил пакет на коммутатор. Т.е. при поступлении пакета с какой-нибудь меткой, она может быть заменена после прохождения внутренней логики коммутатора, а данная команда позволяет сохранить пакет с меткой, с которой он поступил.

При поступлении пакета на порт происходит его классификация на основе меток. Мы либо доверяем CoS/DSCP, либо нет. Если мы не доверяем меткам и никаких дополнительных настроек мы не производим, тогда пакет классифицируется меткой CoS = 0, и имеет int-Prio = 0.

Во внутренней логике коммутатора предусмотрено 0…119 внутренних приоритетов(int-Prio):

```
d1 : d2  0   1   2   3   4   5   6   7   8   9
0:   0   0   0   0   0   0   0   0   1   1
1:   1   1   1   1   1   1   2   2   2   2
2:   2   2   2   2   3   3   3   3   3   3
3:   3   3   4   4   4   4   4   4   4   4
4:   5   5   5   5   5   5   5   5   6   6
5:   6   6   6   6   6   6   7   7   7   7
6:   7   7   7   7   0   0   0   0   0   0
7:   0   0   1   1   1   1   1   1   1   1
8:   2   2   2   2   2   2   2   2   3   3
9:   3   3   3   3   3   3   4   4   4   4
10:  4   4   4   4   5   5   5   5   5   5
11:   5   5   6   6   6   6   6   6   6   6
```

По умолчанию распределение внутренних меток выглядит следующим образом между CoS и DSCP:

```
Ingress COS-TO-Internal-Priority map:
COS:  0   1   2   3   4   5   6   7
-----------------------------------------
INTP: 0   8   16  24  32  40  48  56

Ingress DSCP-TO-Internal-Priority map:
d1 : d2  0   1   2   3   4   5   6   7   8   9
0:   0   1   2   3   4   5   6   7   8   9
1:  10  11  12  13  14  15  16  17  18  19
2:  20  21  22  23  24  25  26  27  28  29
3:  30  31  32  33  34  35  36  37  38  39
4:  40  41  42  43  44  45  46  47  48  49
5:  50  51  52  53  54  55  56  57  58  59
6:  60  61  62  63
```

В случае с CoS, значения идут не один в один как на чипсетах Realtek, в случае с DSCP значения совпадают.

Behavior Aggregate (BA)

Итак, к нам поступил пакет, мы его классифицируем согласно внутренней метки Ingress COS-TO-Internal-Priority map или Ingress DSCP-TO-Internal-Priority map.

После этого, пакет с внутренней меткой передается далее согласно внутренней логике коммутатора. Определяется выходной интерфейс. На данном этапе нам надо определить с какой меткой будет передан пакет на выходной интерфейс и в какую очередь он попадет.

По картам Egress Internal-Priority-TO-COS или Egress Internal-Priority-TO-DSCP определяется с какой меткой будет передан пакет, будет ли она перезаписана на другую метку или останется такой же:

Egress Internal-Priority-TO-COS map:

d1 : d2	0	1	2	3	4	5	6	7	8	9
0:	0	0	0	0	0	0	0	0	1	1
1:	1	1	1	1	1	1	2	2	2	2
2:	2	2	2	2	3	3	3	3	3	3
3:	3	3	4	4	4	4	4	4	4	4
4:	5	5	5	5	5	5	5	5	6	6
5:	6	6	6	6	6	6	7	7	7	7
6:	7	7	7	7	7	7	7	7	7	7
7:	7	7	7	7	7	7	7	7	7	7
8:	7	7	7	7	7	7	7	7	7	7
9:	7	7	7	7	7	7	7	7	7	7
10:	7	7	7	7	7	7	7	7	7	7
11:	7	7	7	7	7	7	7	7	7	7

Egress Internal-Priority-TO-DSCP map:

d1 : d2	0	1	2	3	4	5	6	7	8	9
0:	0	1	2	3	4	5	6	7	8	9
1:	10	11	12	13	14	15	16	17	18	19
2:	20	21	22	23	24	25	26	27	28	29
3:	30	31	32	33	34	35	36	37	38	39
4:	40	41	42	43	44	45	46	47	48	49
5:	50	51	52	53	54	55	56	57	58	59
6:	60	61	62	63	63	63	63	63	63	63
7:	63	63	63	63	63	63	63	63	63	63
8:	63	63	63	63	63	63	63	63	63	63
9:	63	63	63	63	63	63	63	63	63	63
10:	63	63	63	63	63	63	63	63	63	63
11:	63	63	63	63	63	63	63	63	63	63

Например, мы доверяем метке CoS на порту, куда поступил пакет с CoS = 7.
На основе карты Ingress COS-TO-Internal-Priority map ему будет присвоен Intp = 56.

Ingress COS-TO-Internal-Priority map:

COS:	0	1	2	3	4	5	6	7
INTP:	0	8	16	24	32	40	48	56

На основе карты Egress Internal-Priority-TO-Queue map мы сравнили полученный Intp (который был у нас 56) и определили выходящую очередь - 7.

На выходе мы должны понять будет ли происходить перемаркировка пакета на основе карты Egress Internal-Priority-TO-COS map. Согласно этой карте, Intp 56 соответствует CoS = 7. То есть, на входе мы доверились метке CoS, присвоили пакету внутренний приоритет 56, на выходе сравнили внутренний приоритет с картой соответствия внутреннего приоритете с меткой CoS и получили значение такое же – CoS = 7.

Если мы изменим значение карты Egress Internal-Priority-TO-COS map для intp 56 с 7 на 0:

(config)#mls qos map intp-cos 56 to 0

[Изображение недоступно]

То, при поступлении пакета на порт с CoS = 7, мы присвоим ему Intp = 56 по карте Ingress COS-TO-Internal-Priorit. Пакет на основе intp = 56 попадет в приоритетную 7 очередь по карте Egress Internal-Priority-TO-Queue map, а на выходе на основе карты Egress Internal-Priority-TO-COS будет иметь CoS = 0.

Аналогичные действия могут быть произведены для пакетов, если мы доверяем метке DSCP.

[Изображение недоступно]

Interface-based

Все, что приходит на порт, помещать в один какой-то определенный класс.
У коммутаторов на чипсете marvell есть особенность: если мы создадим полисер и будем выставлять пакетам значения CoS или DSCP, то мы не сможем навесить их в направлении in, только на out.

Пример:

```
class-map nag
match vlan 3
!
policy-map nag
class nag
set cos 5
exit
!
Interface Ethernet1/0/1
switchport mode trunk
!
(config)#int e1/0/1
(config-if-ethernet1/0/1)#service-policy input nag
Error: QoS setting dscp/cos is supported only on out direction!
```

Коммутатор оперирует значением INTP, которое присваивается пакетам при поступлении на порт. Значит, если мы не доверяем меткам, то по умолчанию INTP будет равен 0.
Если мы хотим, чтобы, например, все пакеты во влане 3 помещались в определенный класс, то мы с вами будем им менять INTP.

Пример:

```
class-map nag
match vlan 3
!
policy-map nag
class nag
set internal-priority 46
exit
!
Interface Ethernet1/0/1
service-policy input nag
switchport mode trunk
!
```

В данном примере мы все пакеты, которые поступают к нам на порт матчим по влану 3. И этим пакетам мы присваиваем внутренний приоритет 46.

Неважно: доверяем ли мы меткам на порту или нет. Если поступит пакет без любой метки, то внутренний приоритет у него будет 46. Если поступит пакет с меткой и мы доверяем метке – она учитываться не будет и пакету будет присвоен внутренний приоритет 46.

Далее, у нас уже на основе карты ‘Egress Internal-Priority-TO-Queue map’ определяется очередь, в которую попадет пакет, а значения CoS и DSCP пакета будут соответствовать картам Egress Internal-Priority-TO-COS map и Egress Internal-Priority-TO-DSCP map.

Согласно этим картам, у нас INTP 46 соотносится с CoS = 5, а DSCP = 46.

Если мы изменим карту ‘Egress Internal-Priority-TO-Queue map’ так, чтобы пакеты с INTP = 46 попадали вместо 5 очереди во 2, то CoS = 5 и DSCP = 46 у нас метки сохранятся, но пакеты уже будут помещены во 2 очередь.

[Изображение недоступно]

Multi-Field

Мы решили, что не будем вообще доверять никаким меткам и берем всё в свои руки: сами будем решать, какой трафик мы будем выделять из потока и красить его той меткой, которой захотим.

Как мы с вами помним из примера Interface-Based, у marvell есть особенность: если мы создадим полисер и будем выставлять пакетам значения CoS или DSCP, то не сможем навесить их в направлении in, только на out. Мы с вами будем им менять значение INTP.

Выделим два потока: icmp и обычный ip-трафик.

Просто для примера назначим icmp-трафику внутренний приоритет равный 24, а ip-трафику 46. Также, ограничим трафик ip простым полисером в 10 мбит/с – все что больше, у нас будет отбрасываться.

```
ip access-list extended nag
  permit icmp any-source any-destination
  exit
ip access-list extended nag2
  permit ip any-source any-destination
  exit
!
class-map nag
match access-group nag
!
class-map nag2
match access-group nag2
!
policy-map nag
class nag
set internal-priority 24
exit
class nag2
set internal-priority 46
policy 10000 1000 exceed-action drop
exit
!
int e1/0/1
(config-if-ethernet1/0/1)#no mls qos trust cos
(config-if-ethernet1/0/1)#no mls qos trust dscp
(config-if-ethernet1/0/1)#service-policy input nag
sh mls qos int e1/0/1
Ethernet1/0/1:
Default COS: 0
Default int-Prio: 0
Trust:
Pass-through-cos:NONE
Pass-through-dscp-exp:NONE


Attached Policy Map for Ingress: nag
Classmap       classified          green              yellow             red (in packets)
nag            NA                  NA                 NA                 NA
nag2           NA                  NA                 NA                 NA


Queue Algorithm: WDRR
```

```
Queue weights:
Queue      1    2    3    4    5    6    7    8
-------------------------------------------------------
WdrrWeight 1    1    1    1    1    1    1    0
```

Весь icmp-трафик будет окрашен по внутреннему приоритету 24, обработан по внутренней логике коммутатора и будет передан на какой-то исходящий интерфейс.
Согласно карте Egress Internal-Priority-TO-Queue map 24 соответствует 3 очереди.

Весь ip-трафик будет:

- Окрашен по внутреннему приоритету в 46.
- Ограничен скоростью в 10 мбит/с.

Согласно карте Egress Internal-Priority-TO-Queue map 46 соответствует 5 очереди.

Egress Internal-Priority-TO-Queue map:

```
Egress Internal-Priority-TO-Queue map:
d1 : d2  0   1   2   3   4   5   6   7   8   9
0:       0   0   0   0   0   0   0   0   1   1
1:       1   1   1   1   1   1   2   2   2   2
2:       2   2   2   2   3   3   3   3   3   3
3:       3   3   4   4   4   4   4   4   4   4
4:       5   5   5   5   5   5   5   5   6   6
5:       6   6   6   6   6   6   7   7   7   7
6:       7   7   7   7   0   0   0   0   0   0
7:       0   0   1   1   1   1   1   1   1   1
8:       2   2   2   2   2   2   2   2   3   3
9:       3   3   3   3   3   3   4   4   4   4
10:      4   4   4   4   5   5   5   5   5   5
11:      5   5   6   6   6   6   6   6   6   6
```

Например, исходящий интерфейс у нас был 1/0/2:

```
sh mls qos queue statistics int e1/0/2
Port:Ethernet1/0/2
-----------------------------------------
Queue     Passed(packet) Dropped(packet)
0         180            0
1         0              0
2         0              0
3         1635075    0
4         0              0
5         452536     0
6         0              0
7         0              0
```

Как мы видим, трафик был классифицирован и попал в соответствующую очередь.

Policing

Как мы уже с вами ранее обсудили, при использовании полисеров трафик измеряется и ему присваивается цвет: зеленый, желтый или красный. На основе цвета пакета над ним могут производиться действия: пропустить, перемаркировать, отбросить.

Для измерения и ограничения скорости используется механизм Token Bucket, который может работать режимах:

1. Single Bucket - Single Rate - Two Color Marking – красит трафик в зеленый и красный.
2. Dual Bucket - Single Rate - Three Color Marking – красит трафик в три цвета.
3. Dual Bucket – Dual Rate - Three Color Marking – красит трафик в три цвета.

В данной статье мы не будем углубляться в принцип работы Token Bucket.

Полисер может содержать следующие переменные:

policy <bits_per_second> <normal_burst_bytes> [pir <peak_rate_bps>] | <maximum_burst_bytes> [{ exceed-action ACTION | violate-action ACTION }]

<bits_per_second> - committed information rate (CIR) – скорость передачи данных, настраивается в Кбит/с.

<normal_burst_bytes> - committed burst size (CBS) – допустимый всплеск поступления информации, превышающий CIR, измеряется в Кбайт.

<maximum_burst_bytes> - peak burst size (PBS) – разрешенный размер всплесков во время пиков, измеряется в Кбайт.

pir <peak_rate_bps> - peak information rate (PIR) - максимальная средняя скорость, измеряется в Кбит/с. Если PIR не настроено, то работает Dual Bucket - Single Rate, если настроено, то работает Dual Bucket – Dual Rate.

violate-action: действие, при котором идет превышение PIR.

exceed-action: действие, при котором идет превышение CIR, но PIR нет.

ACTION:

drop/transmit: отбросить/передать пакет.

set-internal-priority: установить внутренний приоритет для пакета.

policied-intp-transmit: изменить внутренний приоритет пакета на основе карт intp-to-intp.

Single Bucket Mode:

Если у нас настроено только CIR и CBS, то у нас используется метод обработки пакетов Singl Bucket с покраской трафика в два цвета: зеленый и красный.

policy <bits_per_second> <normal_burst_bytes> ({exceed-action ACTION})

В таком случае пакеты, которые попадают у нас в скорость, которую мы задаем, будут покрашены в зеленый цвет и продвинуты дальше.

Если пакеты будут превышать заданные параметры в полисере, то они будут покрашены в красный цвет и к ним применится **exceed-action** (по умолчанию drop).

Рассматривать **drop/transmit**, **set-internal-priority** мы не будем, т.к. они говорят сами за себя: дропнуть или передать пакет, либо установить внутренний приоритет на основании которого в дальнейшем будет обработан пакет.

При использовании **policied-intp-transmit** мы опираемся на две карты: Internal-Priority-TO-Internal-Priority-YELLOW map и Internal-Priority-TO-Internal-Priority-RED map. При превышении установленного лимита пакеты будут покрашены в красный цвет(т.к. мы используем **Single Bucket Mode**) и им будет присвоен внутренний приоритет на основании карты Internal-Priority-TO-Internal-Priority-RED map.

Например, мы хотим ограничить скорость на порту подключения в 10 Мбит/с (плюс всплеск на 1000 Кбайт - CBS). Все пакеты, которые будут поступать на порт с 3 вланом будут измерены и, в случае превышения скорости, будут покрашены в красный цвет и им будет присвоен внутренний приоритет на основе карты Internal-Priority-TO-Internal-Priority-RED map.

```
class-map nag
match vlan 3
!
policy-map nag
class nag
policy 10000 1000 exceed-action policied-intp-transmit
exit
!
Interface Ethernet1/0/2
service-policy input nag
switchport mode trunk
!
```

Как мы с вами видим, никаким меткам на порту мы не доверяем, значит у нас пакеты будут иметь внутренний приоритет 0.
На основании карты Internal-Priority-TO-Internal-Priority-RED map мы видим, что 0 соответствует 0:

Internal-Priority-TO-Internal-Priority-RED map:
d1 : d2 0   1   2   3   4   5   6   7   8   9
0:      0   1   2   3   4   5   6   7   8   9
1:     10 11 12 13 14 15 16 17 18 19
2:     20 21 22 23 24 25 26 27 28 29
3:     30 31 32 33 34 35 36 37 38 39
4:     40 41 42 43 44 45 46 47 48 49
5:     50 51 52 53 54 55 56 57 58 59
6:     60 61 62 63 64 65 66 67 68 69
7:     70 71 72 73 74 75 76 77 78 79
8:     80 81 82 83 84 85 86 87 88 89
9:     90 91 92 93 94 95 96 97 98 99
10:   100 101 102 103 104 105 106 107 108 109
11:   110 111 112 113 114 115 116 117 118 119

То есть, ничего не изменится. Однако, мы можем менять данную карту. Присвоим внутреннему 0 приоритету 26:

(config)#mls qos map intp-intp red 0 to 26
Internal-Priority-TO-Internal-Priority-RED map:
d1 : d2 0   1   2   3   4   5   6   7   8   9
0:     26   1   2   3   4   5   6   7   8   9
1:     10 11 12 13 14 15 16 17 18 19
2:     20 21 22 23 24 25 26 27 28 29
3:     30 31 32 33 34 35 36 37 38 39
4:     40 41 42 43 44 45 46 47 48 49
5:     50 51 52 53 54 55 56 57 58 59
6:     60 61 62 63 64 65 66 67 68 69
7:     70 71 72 73 74 75 76 77 78 79
8:     80 81 82 83 84 85 86 87 88 89
9:     90 91 92 93 94 95 96 97 98 99
10:   100 101 102 103 104 105 106 107 108 109
11:   110 111 112 113 114 115 116 117 118 119

На данном этапе пакеты, которые будут соответствовать скорости в 10 мбит/с, будут иметь внутренний приоритет 0 и покрашены в зеленый цвет. Пакеты, которые будут превышать 10 Мбит/с будут покрашены в красный цвет и на основе карты им будет присвоен внутренний приоритет 26.

Так как внутренний приоритет для “красных” пакетов изменился на 26, то согласно Egress Internal-Priority-TO-Queue map он попадет в 3 очередь:

Egress Internal-Priority-TO-Queue map:

d1 : d2 0   1   2   3   4   5   6   7   8   9
0:      0   0   0   0   0   0   0   0   1   1
1:      1   1   1   1   1   1   2   2   2   2
2:      2   2   2   2   3   3   3   3   3   3
3:      3   3   4   4   4   4   4   4   4   4
4:      5   5   5   5   5   5   5   5   6   6
5:      6   6   6   6   6   6   7   7   7   7
6:      7   7   7   7   0   0   0   0   0   0
7:      0   0   1   1   1   1   1   1   1   1
8:      2   2   2   2   2   2   2   2   3   3
9:      3   3   3   3   3   3   4   4   4   4
10:     4   4   4   4   5   5   5   5   5   5
11:     5   5   6   6   6   6   6   6   6   6

Пакеты с 0 приоритетом попадут в 0 очередь (в данном случае порт 1 является нашим магистральным портом, куда пакеты улетают):

sh mls qos queue statistics int e1/0/1
Port:Ethernet1/0/1

Queue   Passed(packet) Dropped(packet)
0   17000   0
1   0   0
2   0   0
3   135483   0
4   0   0
5   0   0

6 0 0
7 0 0

Данный пример показывает, как можно манипулировать трафиком, если он превышает установленный лимит в случае использования конкретного действия exceed-action policied-intp-transmit.

Dual Bucket Mode:

Если у нас настроено CIR, CBS, PIR и PBS, то включается Dual Bucket – Dual Rate - Three Color Marking. В случае, если PIR у нас не настраивается, то включается Dual Bucket - Single Rate - Three Color Marking.

policy <bits_per_second> <normal_burst_bytes> [pir <peak_rate_bps>] |

<maximum_burst_bytes> [{exceed-action ACTION | violate-action ACTION }]

В обоих случаях у нас используется покраска трафика в три цвета: зеленый, желтый и красный. В случаях, когда у нас скорость трафика превышает CIR, но не превышает PIR, то это **exceed-action** и пакеты окрашиваются в желтый цвет. В случае, если скорость превышает уже и PIR, то это **violate-action**, что означает, что пакеты будут покрашены в красный цвет.

Как и в предыдущем примере, **exceed-action** и **violate-action** могут иметь значения **drop/transmit**, **set-internal-priority, policied-intp-transmit**. Соответственно при нарушении какой-либо из настроенных скоростных режимов у нас пакеты могут быть отброшены, пропущены, присвоен ЯВНЫЙ внутренний приоритет или внутренний приоритет на основе карт intp-intp.

Мы рассмотрим пример с внутренними картами, т.к. другие действия интуитивно понятны. Мы немного модифицируем наш полисер и укажем в нем дополнительно PIR и PBS.

```
class-map nag
match vlan 3
!
policy-map nag
class nag
policy 10000 1000 pir 20000 10000 exceed-action policied-intp-transmit violate-action policied-intp-transmit
accounting
exit
!
Interface Ethernet1/0/2
service-policy input nag
switchport mode trunk
!
```

Изменим значение внутренних приоритетов для желтой и красной карт Internal-Priority-TO-Internal-Priority-YELLOW map и Internal-Priority-TO-Internal-Priority-RED map соответственно:

```
mls qos map intp-intp red 0 to 26
mls qos map intp-intp yellow 0 to 48
Internal-Priority-TO-Internal-Priority-YELLOW map:
d1 : d2  0   1   2   3   4   5   6   7   8   9
0:     48   1   2   3   4   5   6   7   8   9
1:     10  11  12  13  14  15  16  17  18  19
2:     20  21  22  23  24  25  26  27  28  29
3:     30  31  32  33  34  35  36  37  38  39
4:     40  41  42  43  44  45  46  47  48  49
5:     50  51  52  53  54  55  56  57  58  59
6:     60  61  62  63  64  65  66  67  68  69
7:     70  71  72  73  74  75  76  77  78  79
8:     80  81  82  83  84  85  86  87  88  89
9:     90  91  92  93  94  95  96  97  98  99
10:   100 101 102 103 104 105 106 107 108 109
11:   110 111 112 113 114 115 116 117 118 119
```

Internal-Priority-TO-Internal-Priority-RED map:

```
d1 : d2  0   1   2   3   4   5   6   7   8   9
0:     26   1   2   3   4   5   6   7   8   9
1:     10  11  12  13  14  15  16  17  18  19
2:     20  21  22  23  24  25  26  27  28  29
```

```
3:    30  31  32  33  34  35  36  37  38  39
4:    40  41  42  43  44  45  46  47  48  49
5:    50  51  52  53  54  55  56  57  58  59
6:    60  61  62  63  64  65  66  67  68  69
7:    70  71  72  73  74  75  76  77  78  79
8:    80  81  82  83  84  85  86  87  88  89
9:    90  91  92  93  94  95  96  97  98  99
10:  100 101 102 103 104 105 106 107 108 109
11:  110 111 112 113 114 115 116 117 118 119
```

Так как мы меткам не доверяем на порту, то внутренний приоритет у пакетов будет 0. Пакеты, которые не превышают установленный порог в 10 мбит/с + CBS будут окрашены в зеленый и пройдут без изменений. Пакеты, которые превысят порог в 10 мбит/с, но не превысят PIR в 20 мбит/с, попадут под exceed-action и будут окрашены в желтый цвет, и по карте Internal-Priority-TO-Internal-Priority-YELLOW map будут иметь внутренний приоритет 48. Пакеты, которые превысят PIR попадут под violate-action и будут окрашены в красный цвет.

Дополнительно мы с вами настроили accounting для policy-map nag, с помощью которого мы сможем увидеть подсчет пакетов, которые были окрашены в зеленый, желтый, красный цвет:

```
sh mls qos interface e1/0/2 policy
Ethernet1/0/2:
Attached Policy Map for Ingress: nag
Classmap       classified          green             yellow            red (in packets)
nag           3691999           375735           127540            3188724
```

На основе карты Egress Internal-Priority-TO-Queue map, у нас зеленые пакеты попадают в 0 очередь, т.к. внутренний приоритет у них не менялся и оставался 0, желтые пакеты попадают в 6 очередь, т.к. у них был изменен приоритет на 48, а красные - в 3 очередь, т.к. внутренний приоритет у них был 26 (в данном случае, порт 1 является нашим магистральным портом, куда пакеты улетают. Количество пакетов в green и red с очередями 0 и 3 не совпадает, т.к. счетчик был использован из предыдущего примера):

```
Egress Internal-Priority-TO-Queue map:

d1 : d2  0   1   2   3   4   5   6   7   8   9
0:       0   0   0   0   0   0   0   0   1   1
1:       1   1   1   1   1   1   2   2   2   2
2:       2   2   2   2   3   3   3   3   3   3
3:       3   3   4   4   4   4   4   4   4   4
4:       5   5   5   5   5   5   5   5   6   6
5:       6   6   6   6   6   6   7   7   7   7
6:       7   7   7   7   0   0   0   0   0   0
7:       0   0   1   1   1   1   1   1   1   1
8:       2   2   2   2   2   2   2   2   3   3
9:       3   3   3   3   3   3   4   4   4   4
10:      4   4   4   4   5   5   5   5   5   5
11:      5   5   6   6   6   6   6   6   6
```

```
sh mls qos queue statistics int e1/0/1
Port:Ethernet1/0/1
-----------------------------------------
Queue     Passed(packet) Dropped(packet)
0         135166         0
1         0              0
2         0              0
3         1049563        0
4         0              0
5         0              0
6         127540         0
7         0              0
```

Рассмотренные примеры касались полисинга, который применялся на входящий трафик.

Для исходящего трафика требуется применять полисинг на out. Для маркировки трафика на out также могут быть использованы методы **Single Bucket Mode и Dual Bucket Mode,** однако настраиваются они немного по другому.

Single bucket mode:

policy <bits_per_second> <normal_burst_bytes> ({action ACTION} | exceed-action drop | transmit})

Dual bucket mode:

policy <bits_per_second> <normal_burst_bytes> [pir <peak_rate_bps>] | <maximum_burst_bytes> [{action ACTION | violate-action drop | transmit}]

ACTION definition:

policied-cos-to-cos-transmit
policied-cos-to-dscp-transmit
policied-dscp-exp-to-cos-transmit
policied-dscp-exp-to-dscp-transmit

Отличие заключается в том, что для пакета сначала можно применить action, а затем уже exceed-action или violate-action.

Действие action опирается опять же на набор карт:

COS-TO-COS-GREEN/YELLOW/RED map
COS-TO-DSCP-GREEN/YELLOW/RED map
DSCP-TO-COS-GREEN/YELLOW/RED map
DSCP-TO-DSCP-GREEN/YELLOW/RED map

В обычных случаях нам нет нужды использовать данные карты, т.к. мы применяем к трафику однозначные действия: пропустить (transmit) или отбросить (drop).

Пример такой настройки:

```
class-map nag2
match vlan 3
!
policy-map nag2
class nag2
policy 10000 1000 pir 20000 10000 exceed-action transmit violate-action drop
exit
!
Interface Ethernet1/0/1
service-policy output nag2
switchport mode trunk
```

В данном примере у нас просто идет измерение трафика и, в случае его превышения, у нас он пропускается до определенного порога, а в случае превышения значения PIR отбрасывается(violate-action drop).

Однако, мы можем воспользоваться методом перемаркировки этого трафика:

int e1/0/1 – в данном случае у нас выступает в качестве магистрального интерфейса. Через него проходят пакеты дальше по сети в сторону аплинка на out.

Single bucket mode:

```
class-map nag
match vlan 3
!
policy-map nag
class nag
policy 10000 1000 action policied-cos-to-cos-transmit exceed-action transmit
accounting
exit
!
Interface Ethernet1/0/1
service-policy output nag
switchport mode trunk
!
sh mls qos int e1/0/1
…
…
Attached Policy Map for Egress: nag
Classmap       classified       green       yellow       red (in packets)
```

```
nag          0                0                    NA               0
```

Итак, в случае, если трафик не превышает заданных у нас параметров в 10 Мбит/с, то он окрасится в зеленый цвет и к нему примениться карта COS-TO-COS-GREEN map, т.к. policied-cos-to-cos-transmit говорит нам как раз опираться на нее:

```
COS-TO-COS-GREEN map:
COS:  0   1   2   3   4   5   6   7
----------------------------------------
COS:  0   1   2   3   4   5   6   7
```

Остальной трафик, который не попадает под ограничения, будет покрашен в красный цвет.

Пакеты, которые поступают от клиентов на коммутатор через другие порты будут иметь CoS = 0 (меткам на портах мы не доверяем), а это по карте COS-TO-COS-GREEN map также равняется 0. Запустим трафик, который превышает значение в 10 мбит/с и проверим:

```
sh mls qos int e1/0/1
…
…
Attached Policy Map for Egress: nag
Classmap       classified          green              yellow             red (in packets)
nag            207142              22395              NA                 184747
sh mls qos queue statistics int e1/0/1
Port:Ethernet1/0/1
----------------------------------------
Queue     Passed(packet) Dropped(packet)
0         207871    0
1         0         0
2         0         0
3         0         0
4         0         0
5         0         0
6         0         0
7         0         0
```

Часть пакетов попала под ограничение и они были покрашены в зеленый цвет, остальные - в красный. Все пакеты - и красные, и зеленые - по итогу имели на выходе значение CoS = 0, т.к. по умолчанию эти значения совпадают с картами.

Изменим карту COS-TO-COS-GREEN map так, чтобы пакеты с CoS = 0 перемаркировывались в CoS = 1 и вновь запустим траффик превышающий 10 мбит/с:

```
(config)#mls qos map cos-cos green 1 1 2 3 4 5 6 7
sh mls qos maps cos-cos green
sh mls qos maps cos-cos green
COS-TO-COS-GREEN map:
COS:  0   1   2   3   4   5   6   7
----------------------------------------
COS:  1   1   2   3   4   5   6   7
Green remarking: Disable
sh mls qos interface e1/0/1
…
…
Attached Policy Map for Egress: nag
Classmap       classified          green              yellow             red (in packets)
nag            245926              28077              NA                 217849
```

На выходе у нас снова будут пакеты с CoS = 0. Почему так произошло? Обратите внимание на строчку: “Green remarking: Disable” в выводе “sh mls qos maps cos-cos green”.

Внимание! Чтобы происходила перемаркировка траффика, покрашенного в зеленый цвет, необходимо указать команду: “mls qos egress green remark”.

После применения данной команды зеленые пакеты на выходе из порта будут иметь CoS = 1, красные так и останутся с CoS = 0 (карту COS-TO-COS-RED map то мы ведь не меняли).

В случае, если мы хотим, чтобы у нас перемаркировывался еще и трафик, который красится в красный цвет, то нам надо поменять настройки для карты COS-TO-COS-RED map:

```
(config)#mls qos map cos-cos red 1 1 2 3 4 5 6 7
sh mls qos maps cos-cos red
COS-TO-COS-RED map:
COS:  0   1   2   3   4   5   6   7
---------------------------------------
COS: 1   1   2   3   4   5   6   7
```

После применения данных настроек, пакеты, которые будут покидать коммутатор и окрашены в красный цвет, также будут иметь CoS = 1.

В случае, если мы изменим условия передачи пакетов в полисере и вместо “exceed-action transmit” укажем “exceed-action drop”, то пакеты, которые покрашены в зеленый цвет будут покидать коммутатор, а красные отбрасываться:

```
policy-map nag
class nag
policy 10000 1000 action policied-cos-to-cos-transmit exceed-action drop
accounting
exit
```

Dual bucket mode:

```
class-map nag
match vlan 3
!
policy-map nag
class nag
policy 10000 1000 pir 20000 10000 action policied-dscp-exp-to-dscp-transmit violate-action drop
accounting
exit
!
Interface Ethernet1/0/1
service-policy output nag
switchport mode trunk
!
```

Мы снова не доверяем ни CoS, ни DSCP на клиентских портах, а значит их значения равны 0. Сделано это просто для удобства примера.

Настроены значения PIR и PBS для включения в работу dual bucket.

В текущем примере мы будем опираться уже на показатели DSCP, поэтому применим команду “policied-dscp-exp-to-dscp-transmit”. Зеленый и желтый трафик будут переданы, а красный отброшен (violate-action drop).

Принцип работы в данном примере точно такой же, как и в single bucket, только здесь еще трафик может быть окрашен в желтый цвет.

Для перемаркировки трафика в данном примере будут служить карты:

```
DSCP-TO-DSCP-GREEN
DSCP-TO-DSCP-YELLOW
DSCP-TO-DSCP-RED
```

По умолчанию значения DSCP в картах у нас совпадают один в один, например:

```
sh mls qos maps dscp-dscp green
DSCP-TO-DSCP-GREEN map:
d1 : d2  0   1   2   3   4   5   6   7   8   9
0:      0   1   2   3   4   5   6   7   8   9
1:     10  11  12  13  14  15  16  17  18  19
2:     20  21  22  23  24  25  26  27  28  29
3:     30  31  32  33  34  35  36  37  38  39
4:     40  41  42  43  44  45  46  47  48  49
5:     50  51  52  53  54  55  56  57  58  59
6:     60  61  62  63
```

В картах YELLOW и RED такие же значения.
Запустим трафик, который превышает значения и 10 мбит/с и 20 мбит/с (PIR).

```
sh mls qos int e1/0/1
...
```

...

Attached Policy Map for Egress: nag
Classmap classified green yellow red (in packets)
nag 271489 30613 52027 188849
sh mls qos queue statistics int e1/0/1
Port:Ethernet1/0/1

Queue Passed(packet) Dropped(packet)
0 271557 0
1 0 0
2 0 0
3 0 0
4 0 0
5 0 0
6 0 0
7 0 0

На выходе у нас зеленые, желтые и красные пакеты будут иметь DSCP = 0.

Изменим карты:

(config)#mls qos map dscp-dscp green 0 to 26
(config)#mls qos map dscp-dscp yellow 0 to 48

Не забываем указать команду: mls qos egress green remark.

sh mls qos maps dscp-dscp green
DSCP-TO-DSCP-GREEN map:
d1 : d2 0 1 2 3 4 5 6 7 8 9
0: 26 1 2 3 4 5 6 7 8 9
1: 10 11 12 13 14 15 16 17 18 19
2: 20 21 22 23 24 25 26 27 28 29
3: 30 31 32 33 34 35 36 37 38 39
4: 40 41 42 43 44 45 46 47 48 49
5: 50 51 52 53 54 55 56 57 58 59
6: 60 61 62 63
Green remarking: Enable.
sh mls qos maps dscp-dscp yellow

DSCP-TO-DSCP-YELLOW map:
d1 : d2 0 1 2 3 4 5 6 7 8 9
0: 48 1 2 3 4 5 6 7 8 9
1: 10 11 12 13 14 15 16 17 18 19
2: 20 21 22 23 24 25 26 27 28 29
3: 30 31 32 33 34 35 36 37 38 39
4: 40 41 42 43 44 45 46 47 48 49
5: 50 51 52 53 54 55 56 57 58 59
6: 60 61 62 63

На выходе из интерфейса пакеты, которые были покрашены в желтый цвет Будут иметь DSCP = 48, а зеленые DSCP = 26, а красные у нас будут отброшены.

Важно понимать, что при использовании полисинга на out, пакеты сначала будут помещены в исходящую очередь на основе внутреннего приоритета, полученного на входе в устройство. И только на выходе из очереди у пакетов изменится метка CoS и/или DSCP. Именно поэтому в примерах, которые мы рассмотрели, при выводе команды “sh mls qos queue statistics int e1/0/1” все пакеты у нас проходили через очередь 0, т.к. на входе на клиентских портах отсутствовало доверие меткам, а значит CoS = DSCP = 0.

pass-through-cos и pass-through-dscp

Еще одна особенность чипсетов на Marvell, это возможность применения таких команд как: pass-through-cos и pass-through-dscp.

При поступлении трафика на порт, он может иметь определенную метку CoS и/или DSCP. На основе данной метки пакету будет присвоен INTP. На выходе DSCP метка и CoS метка будет выбрана на основе INTP.

Например, на входе DSCP был 34. Мы меняем ему внутреннюю метку на 56 и пакет на выходе будет иметь метку DSCP = 56.

Это не всегда полезно. Если мы хотим сохранить метку, которая была на входе, но внутри коммутатора обработать по другой внутренней метке, то мы можем использовать следующие команды: pass-through-cos и pass-through-dscp.

Пример:

Например, на коммутатор у нас на порт поступает трафик с DSCP = 34.
Метке DSCP мы доверяем и настраиваем команду pass-through-dscp-exp.

```
Interface Ethernet1/0/2
pass-through-dscp-exp
mls qos trust dscp
switchport mode trunk
switchport trunk allowed vlan 3
!
```

Внутри коммутатора мы будем изменять метку 34 на 56.

```
mls qos map dscp-intp  34 to 56
Ingress DSCP-TO-Internal-Priority map:

d1 : d2  0   1   2   3   4   5   6   7   8   9
0:     0   1   2   3   4   5   6   7   8   9
1:     10  11  12  13  14  15  16  17  18  19
2:     20  21  22  23  24  25  26  27  28  29
3:     30  31  32  33  56  35  36  37  38  39
4:    40  41  42  43  44  45  46  47  48  49
5:     50  51  52  53  54  55  56  57  58  59
6:     60  61  62  63
```

Внутри коммутатора пакет должен быть обработан согласно INTP = 56, а значит он попадет в 7 очередь согласно карте :

Egress Internal-Priority-TO-Queue map:

```
d1 : d2  0   1   2   3   4   5   6   7   8   9
0:      0   0   0   0   0   0   0   0   1   1
1:      1   1   1   1   1   1   2   2   2   2
2:      2   2   2   2   3   3   3   3   3   3
3:      3   3   4   4   4   4   4   4   4   4
4:      5   5   5   5   5   5   5   5   6   6
5:      6   6   6   6   6   6   7   7   7   7
6:      7   7   7   7   0   0   0   0   0   0
7:      0   0   1   1   1   1   1   1   1   1
8:      2   2   2   2   2   2   2   2   3   3
9:      3   3   3   3   3   3   4   4   4   4
10:     4   4   4   4   5   5   5   5   5   5
11:     5   5   6   6   6   6   6   6   6   6
```

Покидая коммутатор из 7 очереди, пакет должен иметь метку 56, однако этого не происходит и метка DSCP у него остается = 34 из-за того, что на порту у нас была команда, которая указывала на то, что метка, с которой поступил пакет, должна сохраняться: pass-through-dscp-exp.

[Изображение недоступно]

Очереди

Коммутаторы на чипсетах Marvell поддерживают несколько способов обработки очередей:

- SP.
- WDRR.

В рамках конфигурации интерфейса мы можем выбрать алгоритм обработки очередей:

```
(config)#int e1/0/1
(config-if-ethernet1/0/1)#mls qos queue algorithm ?
  sp    Strict priority
  wdrr  Weighted deficit round robin
```

По умолчанию, способ обработки очередей: Queue Algorithm: WDRR.
На основе INTP, который мы определили, когда пакет к нам поступил на коммутатор, этот пакет попадет в определенную очередь.

Проверить соответствие INTP очереди, какой метод обработки очереди сейчас конфигурирован на порту и какие веса имеют очереди можно с помощью нескольких команд:

```
sh mls qos maps intp-queue
Egress Internal-Priority-TO-Queue map:

d1 : d2  0   1   2   3   4   5   6   7   8   9
0:       0   0   0   0   0   0   0   0   1   1
1:       1   1   1   1   1   1   2   2   2   2
2:       2   2   2   2   3   3   3   3   3   3
3:       3   3   4   4   4   4   4   4   4   4
4:       5   5   5   5   5   5   5   5   6   6
5:       6   6   6   6   6   6   7   7   7   7
6:       7   7   7   7   0   0   0   0   0   0
7:       0   0   1   1   1   1   1   1   1   1
8:       2   2   2   2   2   2   2   2   3   3
9:       3   3   3   3   3   3   4   4   4   4
10:      4   4   4   4   5   5   5   5   5   5
11:      5   5   6   6   6   6   6   6   6   6
```

```
sh mls qos int e1/0/1
Ethernet1/0/1:
Default COS: 0
Default int-Prio: 0
Trust:
Pass-through-cos:NONE
Pass-through-dscp-exp:NONE
Queue Algorithm: WDRR
Queue weights:
Queue        1    2    3    4    5    6    7    8
-------------------------------------------------------
WdrrWeight 1     1    1    1    1    1    1    0
```

Как мы с вами можем заметить, поддерживается 8 очередей. По умолчанию, вес очередей у всех одинаковый, кроме последней восьмой очереди. Вес восьмой очереди равняется нулю, что делает ее приоритетной очередью. Естественно, вес этой очереди мы можем изменить также на 1 или другое значение, которое доступно для настройки.

Вес очереди для WDRR мы можем изменить только глобально, отдельно для портов его нет возможности настроить.

```
(config)#mls qos queue weight 1 2 3 4 5 6 7 0
```

Вес очереди настраивается по-порядку. То есть 1 соответствует первой очереди, 2 – второй очереди и т.д.

```
sh mls qos interface e1/0/1
Ethernet1/0/1:
Default COS: 0
Default int-Prio: 0
Trust:
Pass-through-cos:NONE
Pass-through-dscp-exp:NONE
Queue Algorithm: WDRR
Queue weights:
Queue        1    2    3    4    5    6    7    8
-------------------------------------------------------
WdrrWeight 1     2    3    4    5    6    7    0
```

Важно: на коммутаторах под управлением чипсетов Marvell приоритетные очереди могут быть назначены только в конце. То есть, если мы хотим сделать две приоритетные очереди, то вес 7 и 8 очереди надо изменить на 0. Мы не можем сделать две приоритетные очереди, которые находятся не в конце, например 1 1 0 1 1 1 1 0 – это надо помнить.

Таким образом, по карте Egress Internal-Priority-TO-Queue map у нас определяется исходящая очередь, а по алгоритму обработки очередей у нас определяется как пакеты будут изыматься из очередей.

Queue statistics

Отдельно хотелось бы отметить возможность просмотра статистики очередей на чипах Marvell. Включить данную функцию можно командой mls qos queue statistics enable.

После включения данной команды появится возможность просмотра количества пакетов, которые попали в ту или иную исходящую очередь.

Для просмотра можно использовать команду sh mls qos queue statistics {interface}.

[Изображение недоступно]

Помимо количества пакетов, которые попали в ту или иную очередь, данная команда показывает пакеты, которые были дропнуты, если им не хватило буфера.

Классификация и маркировка трафика на чипсете Broadcom

Рассмотрим некоторые особенности на примере коммутатора SNR-S300X-24FQ:

```
#sh mls qos int e1/0/17
Ethernet1/0/17:
Default COS: 0
Trust: COS
Pass-through-cos:NONE
Egress Internal-Priority-TO-Queue map:
INTP:  0   1   2   3   4   5   6   7
-----------------------------------------
Queue: 0   1   2   3   4   5   6   7
Queue Algorithm: WRR
Queue weights:
Queue     1    2    3    4    5    6    7    8
-------------------------------------------------------
WrrWeight  1    2    3    4    5    6    7    8
WdrrWeight 1    2    4    8    16   32   64   64
...
```

1. По умолчанию, мы доверяем метке CoS на порту. Если одновременно настроено доверие метке CoS и DSCP, то приоритет отдается метке DSCP.
2. Алгоритмы обработки очередей у нас поддерживаются три: SP, WDRR, WRR.
3. Internal priority имеет значение такое же, как на коммутаторах на чипсете Realtek от 0 до 63.
4. Есть такое понятие как Pass-through-cos - сохранение метки CoS, с которой поступил пакет на коммутатор. Т.е. при поступлении пакета с какой-нибудь меткой она может быть заменена после прохождения внутренней логики коммутатора, а данная команда позволяет сохранить пакет с меткой, с которой он поступил.
5. На коммутаторе есть возможность просматривать статистику пакетов, которые распределяются по очередям: Queue statistics.

При поступлении пакета на порт, происходит его классификация на основе меток. Мы либо доверяем CoS/DSCP, либо нет.

Если мы не доверяем меткам и никаких дополнительных настроек не производим, тогда пакет классифицируется меткой CoS = 0, и имеет int-Prio = 0.

Behavior Aggregate (BA)

Мы доверяем существующей метке. По умолчанию мы доверяем метке CoS. Доверие метке можно

изменять в настройках порта с помощью команды mls qos trust dscp/cos. На основе метки CoS или DSCP происходит маркировка трафика внутренним приоритетом Intp. Для CoS это значения от 0 до 7, для DSCP это значения от 0 до 63. На основе внутреннего приоритета в дальнейшем будет выбрана исходящая очередь, а значит и приоритет обработки такого пакета.

Внутренний приоритет назначается согласно картам queue maps.

Если мы доверяем метке CoS:

К нам поступил пакет с CoS = 2.

```
Interface Ethernet1/0/3
switchport mode trunk
switchport trunk allowed vlan 3
!
#sh mls qos int e1/0/3
Ethernet1/0/3:
Default COS: 0
Trust: COS
Pass-through-cos:NONE
...

...
```

Согласно карте ‘Ingress COS-TO-Internal-Priority map’ пакету будет присвоен INTP = 2.

```
Ingress COS-TO-Internal-Priority map:
COS:  0   1   2   3   4   5   6   7
-----------------------------------------
INTP: 0   1   2   3   4   5   6   7
```

Исходящий интерфейс для пакета у нас является порт Ethernet1/0/4.

```
sh mls qos int e1/0/4
...

...

Egress Internal-Priority-TO-Queue map:
INTP:  0   1   2   3   4   5   6   7
-----------------------------------------
Queue: 0   1   2   3   4   5   6   7
```

Согласно карте ‘Egress Internal-Priority-TO-Queue map’ пакет с INTP = 2 опадет во 2 queue.

Если мы доверяем метке DSCP:

К нам поступил пакет с DSCP = 24.

```
#sh mls qos int e1/0/3
Ethernet1/0/3:
Default COS: 0
Trust: COS DSCP
Pass-through-cos:NONE
...
...
```

Согласно карте ‘Ingress DSCP-TO-Internal-Priority map’ пакету будет присвоен INTP = 3.

```
Ingress DSCP-TO-Internal-Priority map:
d1 : d2  0   1   2   3   4   5   6   7   8   9
0:       0   0   0   0   0   0   0   0   1   1
1:       1   1   1   1   1   1   2   2   2   2
2:       2   2   2   2   3   3   3   3   3   3
3:       3   3   4   4   4   4   4   4   4   4
4:       5   5   5   5   5   5   5   5   6   6
5:       6   6   6   6   6   6   7   7   7   7
6:       7   7   7   7
```

Исходящий интерфейс для пакета у нас является порт Ethernet1/0/4.

sh mls qos int e1/0/4
…
…

Egress Internal-Priority-TO-Queue map:

```
INTP:  0   1   2   3   4   5   6   7
-----------------------------------------
Queue: 0   1   2   3   4   5   6   7
```

Согласно карте ‘Egress Internal-Priority-TO-Queue map’ пакет с INTP = 3 попадет в 3 queue.

[Изображение недоступно]

Если мы ничего не меняли в queue maps, то пакеты будут отправлены с тем же CoS – если мы доверяем CoS или с тем же DSCP, если мы доверяем DSCP. Однако, есть моменты, на которые стоит обратить внимание:

1. Карта ‘Ingress DSCP-TO-DSCP map’.

[Изображение недоступно]

Если к нам пришел пакет с DSCP = 27 и мы доверяем метке DSCP на порту, то на основе карты ‘Ingress DSCP-TO-Internal-Priority map’ пакет попадет в 3 очередь:

Ingress DSCP-TO-Internal-Priority map:

```
d1 : d2  0   1   2   3   4   5   6   7   8   9
0:       0   0   0   0   0   0   0   0   1   1
1:       1   1   1   1   1   1   2   2   2   2
2:       2   2   2   2   3   3   3   3   3   3
3:       3   3   4   4   4   4   4   4   4   4
4:       5   5   5   5   5   5   5   5   6   6
5:       6   6   6   6   6   6   7   7   7   7
6:       7   7   7   7
```

Однако, на выходе будет иметь DSCP согласно карте ‘Ingress DSCP-TO-DSCP map’, а значит = 24. По умолчанию, как мы можем видеть, в карте ‘Ingress DSCP-TO-DSCP map’ соответствие идет не 1 к 1, а, например, значения от 24 до 31 принимают значения 24. Значения в карте ‘Ingress DSCP-TO-DSCP map’ можно изменить.

2. Если мы доверяем только метке CoS и к нам пришел пакет с меткой CoS и DSCP, то пакет получит внутренний приоритет на основе метки CoS, попадет в соответствующую очередь и будет передан далее. При этом метка DSCP в пакете сохраняется той, с которой он пришел.

3. Если мы доверяем и метке CoS и метке DSCP или только метке DSCP, то пакет будет обработан согласно карте ‘Ingress DSCP-TO-Internal-Priority map’, т.е. по метке DSCP. При этом, если пришел пакет с меткой CoS = 1, а меткой DSCP = 30, то согласно

Ingress DSCP-TO-Internal-Priority map:

```
d1 : d2  0   1   2   3   4   5   6   7   8   9
0:       0   0   0   0   0   0   0   0   1   1
1:       1   1   1   1   1   1   2   2   2   2
2:       2   2   2   2   3   3   3   3   3   3
3:       3   3   4   4   4   4   4   4   4   4
4:       5   5   5   5   5   5   5   5   6   6
5:       6   6   6   6   6   6   7   7   7   7
6:       7   7   7   7
```

пакет будет помещен в 3 очередь, на выходе будет иметь DSCP = 24 по карте ‘Ingress DSCP-TO-DSCP map’, а CoS с 1 изменится на 3, т.к. INTP = 3 по карте ‘Ingress DSCP-TO-Internal-Priority map’. Т.е. метка CoS изменится согласно внутреннему приоритету, основанному на карте DSCP.

Interface-based

Все, что приходит на порт помещать в один какой-то определенный класс.

Предположим, мы бы хотели весь трафик красить с определенной меткой CoS или DSCP.

Для этого мы воспользуемся policy:

```
class-map nag
match vlan 3
!
policy-map nag
class nag
set cos 3
exit
!
Interface Ethernet1/0/3
service-policy input nag
switchport mode trunk
switchport trunk allowed vlan 3
```

Или

```
class-map nag
match vlan 3
!
policy-map nag
class nag
set ip dscp 46
exit
!
Interface Ethernet1/0/3
service-policy input nag
switchport mode trunk
switchport trunk allowed vlan 3
```

Как будет обработан пакет и в какую очередь он будет помещен, зависит от того, доверяем ли мы метке CoS/DSCP или нет.

Если мы доверяем метке CoS/DSCP:

1. Пакет поступает на порт.
2. Коммутатор доверяет метке, с которой он поступил.
3. На основе карт ‘Ingress COS-TO-Internal-Priority map’ или ‘Ingress DSCP-TO-Internal-Priority map’ пакету присваивается INTP. На основе INTP пакет помещается в исходящую очередь.
4. Происходит перемаркировка пакета на основе policy, который был задан на порту, куда изначально поступил пакет.
5. Пакет покидает коммутатор с новой меткой, заданной в policy.

Таким образом, сначала происходит распределение пакетов по очередям на основе пришедшей метки, и только потом метка будет изменена. То есть, пакет будет обработан на основе пришедшей метки, а не той метки, которая была задана в policy.

Три момента, которые были указаны в разделе **Behavior Aggregate** сохраняются, но стоит учитывать, что policy приоритетнее карты ‘Ingress DSCP-TO-DSCP map’, поэтому на выходе DSCP метка будет той, которая указана в policy.

Если мы не доверяем метке CoS и DSCP:

1. Пакет поступает на порт.
2. Коммутатор не доверяет метке, с которой он поступил.
3. Пакет ассоциируется с дефолтным CoS, который равен 0.
4. Все пакеты попадают в низкоприоритетную очередь.
5. Происходит перемаркировка пакета на основе policy, который был задан на порту.
6. Пакет покидает коммутатор с новой меткой, заданной в policy.

[Изображение недоступно]

Если к нам поступил пакет с меткой CoS = 1 и меткой DSCP = 30 и мы меняем полисером метку CoS на 3:

```
policy-map nag
class nag
set cos 3
exit
!
```

Т.к. мы не доверяем меткам, пакет попадет в 1 очередь (т.е. самую низкоприоритетную). При этом, на выходе будет иметь CoS = 3, а DSCP = 30. Как мы видим, значение DSCP остается без изменения.

Однако, если мы будем менять в полисере значение DSCP:

```
policy-map nag
class nag
set ip dscp 46
exit
!
```

И к нам придет пакет с меткой CoS = 1 и меткой DSCP = 30, то на выходе пакет попадет в 1 очередь (т.е. самую низкоприоритетную), будет иметь метку CoS = 0, а DSCP = 46. Таким образом, метка CoS не сохраняется в отличие от DSCP и сбрасывается в 0.

Вывод: если мы не доверяем меткам на порту и к нам пришел пакет с метками CoS и DSCP, то:

- А) пакет будет помещен в низкоприоритетную очередь;
- Б) метка DSCP сохранится, метка CoS сбрасывается в 0.

Multi-Field

При поступлении пакета мы не доверяем имеющейся маркировке и производим перемаркировку пакета.

Как мы с вами выяснили из предыдущих пунктов, если мы не доверяем никакой метке на порту, то все пакеты будут попадать в одну низкоприоритетную очередь. Если пакет имел какую-нибудь метку CoS, то она не будет сохранятся. Если какую-нибудь метку DSCP, то она сохранится при прохождении пакета через коммутатор.

Мы с вами будет менять у одних пакетов метку CoS, а у других метку DSCP. Для IP-пакетов будем ставить метку CoS = 3, а для IP пакетов DSCP = 46. Пакеты попадут в низкоприоритетную очередь.

Однако для IP-пакетов внутренний приоритет мы установим в 5.

```
!
ip access-list extended nag
  permit icmp any-source any-destination
  exit
ip access-list extended nag2
  permit ip any-source any-destination
  exit
!
class-map nag
match access-group nag
!
class-map nag2
match access-group nag2
!
policy-map nag
class nag
set cos 3
exit
class nag2
set ip dscp 46
```

```
set internal-priority 5
exit
!
(config)#int e1/0/3
(config-if-ethernet1/0/5)#no mls qos trust cos
(config-if-ethernet1/0/5)#no mls qos trust dscp
!
Interface Ethernet1/0/3
no mls qos trust cos
service-policy input nag
switchport mode trunk
switchport trunk allowed vlan 3
!
```

Таким образом, при поступлении ICMP-пакета мы установим ему метку CoS = 3 на выходе, пакет попадет в 1 низкоприоритетную очередь. Если каким-то образом ICMP имел метку DSCP, то она сохранится на выходе.

При поступлении IP-пакета, его метка DSCP на выходе будет 46 и пакет будет обработан в 5 очереди, т.к. INTP = 5. Если пакет имел какую-нибудь метку CoS, то она не сохранится, CoS = 0.

Policing

При использовании полисеров трафик измеряется и ему присваивается цвет: зеленый, желтый или красный. На основе цвета пакета над ним могут производиться действия: пропустить, перемаркировать, отбросить.

Для измерения и ограничения скорости используется механизм Token Bucket, который может работать режимах:

1. Single Bucket - Single Rate - Two Color Marking – красит трафик в зеленый и красный.
2. Dual Bucket - Single Rate - Three Color Marking – красит трафик в три цвета.
3. Dual Bucket – Dual Rate - Three Color Marking – красит трафик в три цвета.

Полисер может содержать следующие переменные:

policy <bits_per_second> <normal_burst_bytes> [pir <peak_rate_bps>] | <maximum_burst_bytes> [{conform-action ACTION | exceed-action ACTION |violate-action ACTION }]

<bits_per_second> - committed information rate (CIR) – скорость передачи данных, настраивается в Кбит/с.

<normal_burst_bytes> - committed burst size (CBS) – допустимый всплеск поступления информации, превышающий CIR, измеряется в Кбайт.

<maximum_burst_bytes> - peak burst size (PBS) – разрешенный размер всплесков во время пиков, измеряется в Кбайт.

pir <peak_rate_bps> - peak information rate (PIR) - максимальная средняя скорость, измеряется в Кбит/с. Если PIR не настроено, то работает Dual Bucket - Single Rate, если настроено, то работает Dual Bucket – Dual Rate.

conform-action: действие, при котором не превышается CIR.

violate-action: действие, при котором идет превышение PIR.

exceed-action: действие, при котором идет превышение CIR, но PIR нет.

ACTION:

drop/transmit: отбросить/передать пакет.

set-dscp-transmit: установить значение DSCP в пакете.

set-internal-priority: установить внутренний приоритет для пакета.

set-cos-transmit: установить значение CoS в пакете.

Single Bucket Mode:

Если у нас настроено только CIR и CBS, то у нас используется метод обработки пакетов Singl Bucket с покраской трафика в два цвета: зеленый и красный.

policy <bits_per_second> <normal_burst_bytes> ({conform-action ACTION |exceed-action ACTION})

В таком случае пакеты, которые попадают у нас в скорость, которую мы задаем, будут покрашены в зеленый цвет и продвинуты дальше.

Если пакеты будут превышать заданные параметры в полисере, то они будут покрашены в красный цвет и к ним применится **exceed-action** (по умолчанию drop).

Рассмотрим простой пример **Single Bucket Mode**.

Мы будем матчить пакеты с вланом 3 на порту 1/0/3. Ограничивать скорость пакетов 10 мбит/с с всплеском до 1000 Кбайт. Все, что свыше данного порога будет отброшено.

```
class-map nag
match vlan 3
!
policy-map nag
class nag
policy 10000 1000 conform-action transmit exceed-action
drop
exit
!
Interface Ethernet1/0/3
service-policy input nag
switchport mode trunk
switchport trunk allowed vlan 3
!
```

Dual Bucket Mode:

Если у нас настроено CIR, CBS, PIR и PBS, то у нас включается Dual Bucket – Dual Rate - Three Color Marking. В случае, если PIR у нас не настраивается, то включается Dual Bucket - Single Rate - Three Color Marking.

policy <bits_per_second> <normal_burst_bytes> [pir <peak_rate_bps>] | <maximum_burst_bytes> [{conform-action ACTION | exceed-action ACTION |violate-action ACTION }]

В обоих случаях у нас используется покраска трафика в три цвета: зеленый, желтый и красный. В случаях, когда у нас скорость трафика превышает CIR, но не превышает PIR, то это **exceed-action** и пакеты окрашиваются в желтый цвет, в случае, если скорость превышает уже и PIR, то это **violate-action**, что означает, что пакеты будут покрашены в красный цвет.

В качестве примера мы рассмотрим просто случайную конфигурацию для наглядности.

Мы не будем применять команды **drop/transmit,** т.к. если мы укажем в **exceed-action и violate-action** команду drop, то трафик сверх CIR будет отбрасываться.

Рассмотрим пример **Dual Bucket Mode**.

Матчим пакеты на 3 порту с вланом 3. Пакеты, которые попадают в зеленую зону просто продвигаем дальше. Пакеты, которые попадают в желтую зону – устанавливаем им CoS = 2. Пакеты, которые попадают в красную зону устанавливаем им приоритет CoS = 1. Как мы видим, пакеты у нас вообще не отбрасываются, нет действия drop.

Изначально на порт поступают пакеты с CoS = 3

```
class-map nag
match vlan 3
!
policy-map nag
```

```
class nag
policy 10000 1000 pir 20000 1000 conform-action transmit exceed-action set-cos-transmit 2 violate-action
set-cos-transmit 1
accounting
exit
!
Interface Ethernet1/0/3
service-policy input nag
switchport mode trunk
switchport trunk allowed vlan 3
!
```

Таким образом, если на порт у нас будет поступать пакеты с CoS = 3 и скорость этих пакетов не превышает CIR, то метка CoS у них сохраняется. Если пакеты превышают CIR, но не превышают PIR, то они метятся желтыми и им устанавливается CoS = 2.

Остальные пакеты превышающие PIR попадают под красную зону и им устанавливается CoS = 1.

Полисер можно применять, в том числе, и на направление out.

Также, стоит отметить, что изменение CoS в приведенном выше примере не влияет на попадание в очередь. То есть, т.к. у нас все пакеты на порт 3 приходили с меткой CoS = 3, то всем пакетам присваивался INTP = 3, затем происходила перемаркировка пакетов на CoS = 2 и CoS = 1, но все пакеты попадали в 3 очередь согласно INTP.

Мы могли бы назначить ‘set-internal-priority <x>’ в action, чтобы пакеты попадали в определенную очередь.

pass-through-cos

Как мы с вами уже обсуждали выше, мы можем:

- Не доверять никаким меткам на портах.
- Доверять только метке CoS.
- Доверять и метке CoS, и метке DSCP (но в этом случае приоритет у DSCP).
- Доверять только метке DSCP.

Предположим, к нам поступил пакет со следующими метками:

- CoS = 3
- DSCP = 46

Мы не производим никаких перемаркировок внутри коммутатора, не используем никаких дополнительных настроек и возможностей.

1. Если меткам мы не доверяем на порту, то с пакета снимается метка CoS и устанавливается CoS = 0, а вот метка DSCP сохраняется, т.е. остается равно 46.
2. Если мы доверяем только метке CoS на порту, то метка CoS и метка DSCP сохраняются, т.е остаются CoS = 3 и DSCP = 46. Обработка пакета по очередям основывается на метке CoS.
3. Если мы доверяем метке и CoS и DSCP, то в этом случае приоритет появляется у метки DSCP. Обработка пакета по очередям основывается на метке DSCP.

При этом метка CoS на выходе будет другой.

Согласно карте ‘Ingress DSCP-TO-Internal-Priority map’ внутренний приоритет пакету будет присвоен на основании его метке DSCP = 46. Метке 46 соответствует INTP = 5.

[Изображение недоступно]

Покидая коммутатор, метка CoS уже будет иметь значение согласно внутреннему приоритету = 5, а вот метка DSCP будет иметь значение не 46, а 40.

Это связано с тем, что после поступления пакета с меткой DSCP = 46, коммутатор обращается с карте ‘Ingress DSCP-TO-DSCP map’:

[Изображение недоступно]

Согласно этой карте, значения DSCP от 40 до 47 принимают значения 40. Поэтому на выходе метка DSCP у нас изменится с 46 на 40.
Мы можем изменить значения в карте: (config)#mls qos map dscp-dscp 46 to 46.
В этом случае пакет на выходе будет иметь метку CoS = 5 и метку DSCP = 46.

4. Если мы доверяем только метке DSCP, то в этом случае происходит все тоже самое, что описано в пункте 3.

Но что, если нам нужно, чтобы метка CoS сохранялась как и метка DSCP? В этом случае мы можем использовать команду **pass-through-cos** на порту, куда к нам поступает пакет.

За пример возьмем порт 3:

```
sh mls qos int e1/0/3
Ethernet1/0/3:
Default COS: 0
Trust:
Pass-through-cos: YES
```

Как мы с вами видим, Pass-through-cos активирована на порту. Меткам мы на порту не доверяем и при поступлении к нам пакета с меткой CoS = 3 она сохранится (как и метка DSCP). Пакеты будут обработаны и попадут в низкоприоритетную очередь, т.к. мы меткам не доверяем, но метки сами на выходе сохраняются.

Если мы доверяем только метке CoS, то поведение не меняется – сохраняется и метка CoS, и метка DSCP.

Если мы доверяем и метке CoS, и метке DSCP или только метке DSCP, то метка CoS у нас сохраняется CoS = 3, а не меняется на CoS = 5.

[Изображение недоступно]

Очереди

Коммутатор поддерживает несколько способов обработки очередей:

- SP.
- WRR.
- WDRR.

Очереди настраиваются в рамках конфигурации самого интерфейса:

Очереди настраиваются в рамках конфигурации самого интерфейса:

```
(config)#int e1/0/4
(config-if-ethernet1/0/4)#mls qos queue algorithm ?
  sp    Strict priority
  wdrr  Weighted deficit round robin
  wrr   Weighted round robin
```

По умолчанию, способ обработки очередей: Queue Algorithm: WRR.

Проверить соответствие INTP очереди, какой метод обработки очереди сейчас конфигурирован на порту и какие веса имеют очереди, можно с помощью команды sh mls qos int <interface>:

```
sh mls qos int e1/0/4
Ethernet1/0/4:
Default COS: 0
Trust: COS
Pass-through-cos:NONE
Egress Internal-Priority-TO-Queue map:
```

```
INTP:  0   1   2   3   4   5   6   7
-----------------------------------------
Queue: 0   1   2   3   4   5   6   7
Queue Algorithm: WRR
Queue weights:
Queue      1     2     3     4     5     6     7     8
-----------------------------------------------------------
WrrWeight  1     2     3     4     5     6     7     8
WdrrWeight 1     2     4     8     16    32    64    64
```

Здесь мы хорошо видим, что у нас есть прямое соотношение между INTP и Queue, т.е., например, если по внутренней логике работы коммутатора, пакету, при его классификации, был присвоен INTP = 1, то он попадет в 1 Queue (исходящую естественно). После чего, пакеты в очередях будут обработаны согласно алгоритмам.

Если мы хотим при алгоритмах WRR или WDRR выделить приоритетную очередь (SP), то в этом случае вес этой очереди надо выставить в 0.

```
(config)#int e1/0/4
(config-if-ethernet1/0/4)#mls qos queue wdrr weight 1 2 4 0 16 32 64 64
sh mls qos int e1/0/4
Ethernet1/0/4:
Default COS: 0
Trust: COS
Pass-through-cos:NONE
Egress Internal-Priority-TO-Queue map:
INTP:  0   1   2   3   4   5   6   7
-----------------------------------------
Queue: 0   1   2   3   4   5   6   7
Queue Algorithm: WRR
Queue weights:
Queue      1     2     3     4     5     6     7     8
-----------------------------------------------------------
WrrWeight  1     2     3     4     5     6     7     8
WdrrWeight 1     2     4     0     16    32    64    64
```

Обратите внимание на строку WdrrWeight - вес четвертой очереди изменился на 0. Все пакеты, которые будут попадать в эту очередь, будут обработаны в приоритете, т.е. 4 выходящая очередь становится SP.

Важно отметить, что в выводе веса очередей они начинаются не с 0, а с 1, т.е. 1…8, тогда как соотношение INTP к Queue начинается с 0…7. Поэтому, если пакет соответствовал INTP = 1, то он соответствует второй очереди в рамках обработки алгоритма.

Queue statistics

Также, на коммутаторе можно включить статистику просмотра очередей. Включить данную функцию можно командой mls qos queue statistics enable.

После включения данной команды появится возможность просмотра количества пакетов, которые попали в ту или иную исходящую очередь.

Для просмотра можно использовать команду sh mls qos queue statistics {interface}.

[Изображение недоступно]

Помимо количества пакетов, которые попали в ту или иную очередь, данная команда показывает пакеты, которые были дропнуты, если им не хватило буфера.