Протокол Spanning Tree (STP)

Описание

Протокол Spanning Tree (STP) обеспечивает отсутствие петель в топологии любой сети.

Оглавление

1. Протокол Spanning Tree – самое важное

Протокол Spanning Tree (STP) обеспечивает отсутствие петель в топологии любой сети. Помимо предотвращения петель, STP изолирует угрозу от широковещательного шторма в сетях на втором уровне модели OSI (L2). Разберемся в терминах:

КАКИЕ БЫВАЮТ ПОРТЫ?

Можно смело выделить 3 вида портов в рамках протокола Spanning Tree. А именно:

1. Корневой порт (root port)
2. Выделенный порт (designated port)
3. Блокированный (альтернативный порт)

СТАТУСЫ ПОРТОВ

Порт коммутатора может находиться в различных статусах, в зависимости от результата сходимости Spanning Tree:

- **Блокирован (Blocking)** - как видно из названия, данный порт находится в статусе блокировки. Это означает, что порт не участвует в приеме и пересылке фреймов. Все BPDU сообщения от соседних коммутаторов исключаются.

  > **BPDU (Bridge Protocol Data Unit)** это фреймы, необходимые для обмена сообщениями между коммутаторами для выбора корневого (root) устройства в рамках механизма протокола STP (Spanning Tree Protocol).

- **Слушает (Listening)** – коммутатор все еще не участвует в процессе передачи фреймов с данными, но получает и отправляет сообщения BPDU.
- **Учится (Learning)** – в данном состоянии порт начинает фиксировать MAC – адреса устройств.
- **Пересылка (Forwarding)** – в состоянии пересылки, коммутатор может отправлять и принимать фреймы BPDU параллельно с заполнением таблицы MAC - адресов.
- **Выключен** – порт выключен администратором.

ЭТАПЫ ПРОТОКОЛА STP

1. Выбор «корневого» (root) коммутатора.
2. Выбор «корневого» (root) порта.
3. Назначение «выделенного» (designated) порта.
4. Блокировка остальных портов в рамках алгоритма STP.

ВЫБОР КОРНЕВОГО КОММУТАТОРА

Коммутатор с наименьшим идентификатором (ID) выбирается как корневое устройство. Идентификатор коммутатора (switch ID) состоит из следующих компонентов: .

1. Номер приоритета .
2. MAC – адрес коммутатора

Например: 24577.00:00:00:00:00:01 / Приоритет. MAC – адрес

В процессе выбора корневого коммутатора, первым делом сравнивается приоритет. Если у двух коммутаторов одинаковых приоритет, то выбор базируется на MAC – адресе устройства.

Выбор корневого порта

Корневой порт выбирается на основании наименьшей «стоимости» пути к корневому коммутатору. Стоимость пути выражается из стоимости линков, ведущих к корневому коммутатору.

Важно отметить:

1. Корневые порты назначаются только на не корневых коммутаторах.

2. Один не корневой коммутатор может иметь только один корневой порт.

Выбор назначенного порта

Порт коммутатора, который имеет кратчайший путь к корневому коммутатору - называется «назначенным».

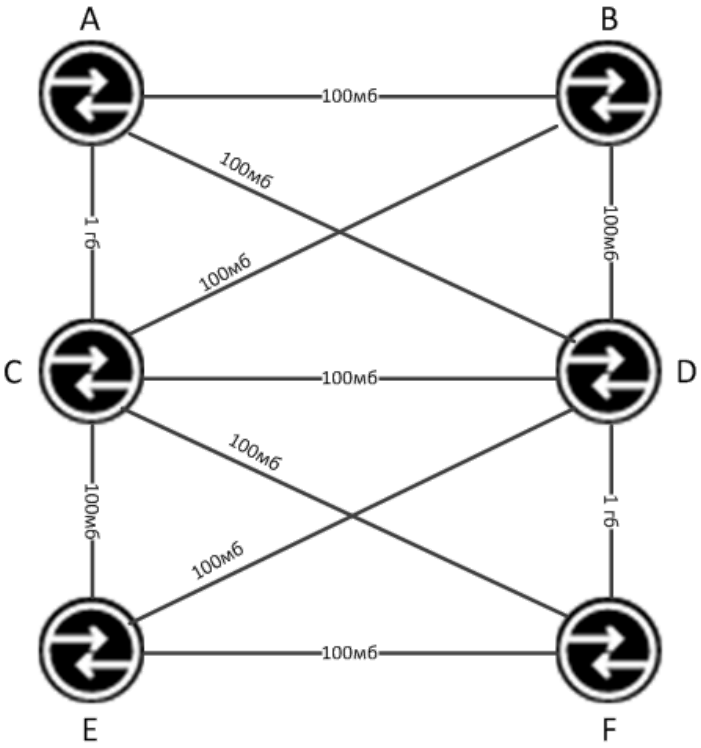
1. Каждый сегмент (путь) имеет свой назначенный порт.
2. Назначенные порты определяются на всех коммутаторах (корневых и нет).

Если два порта имеют одинаковую стоимость, сначала учитывается идентификатор устройства (Bridge ID), а затем идентификатор порта (Port ID).

Все остальные порты переходят в альтернативный статус и блокируются.

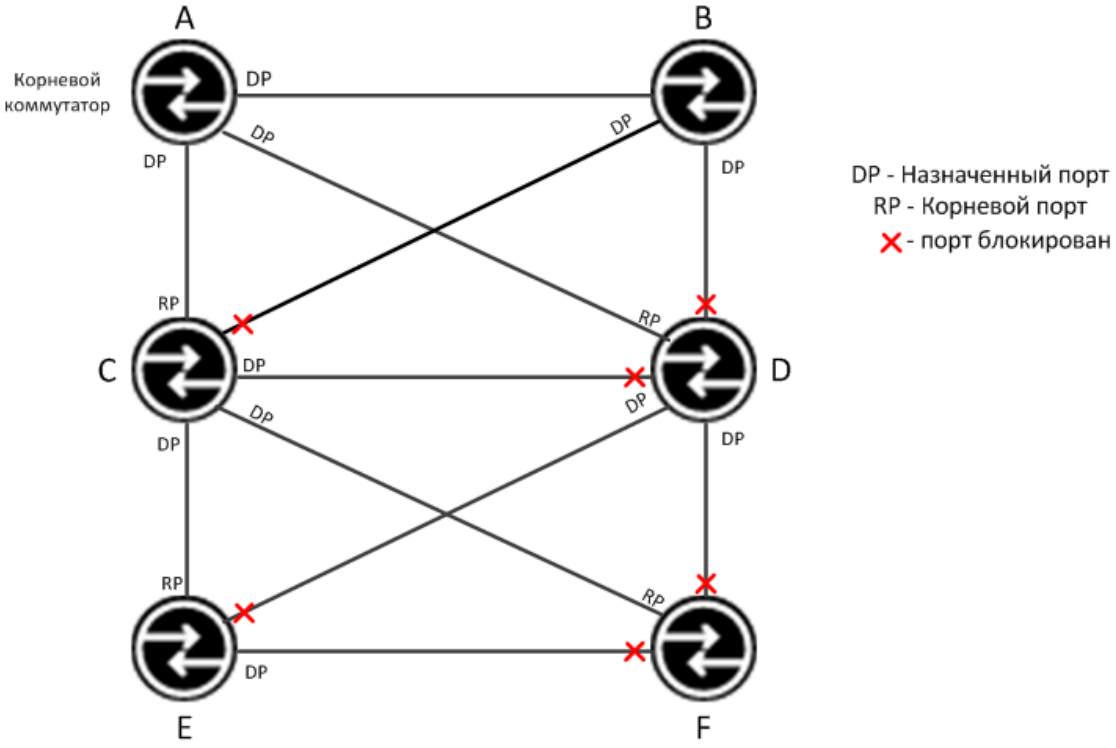
Пример

До запуска алгоритма Spanning Tree:

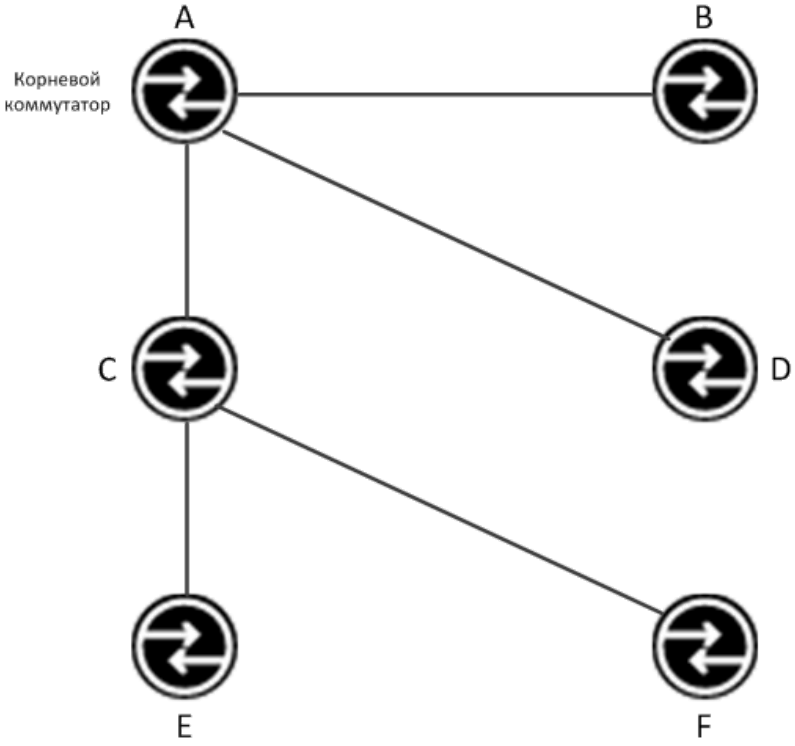


Коммутатор	Идентификатор ID
A	32768.aa.aa.aa.aa.aa.aa
B	32768.bb.bb.bb.bb.bb.bb
C	32768.cc.cc.cc.cc.cc.cc
D	32768.dd.dd.dd.dd.dd.dd
E	32768.ee.ee.ee.ee.ee.ee
F	32768.ff.ff.ff.ff.ff.ff

Выбор портов



Финальная топология



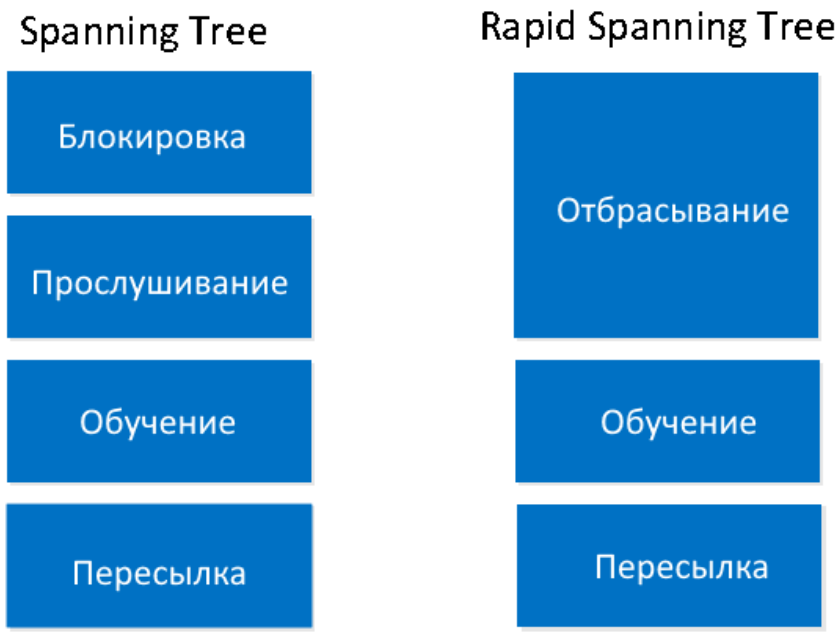
2. Полный обзор RSTP (Rapid Spanning Tree)

Классический стандарт связующего дерева работает нормально, но в настоящее время для современных сетей он слишком медленный 🐌

В настоящее время мы наблюдаем в наших сетях все больше и больше маршрутизации. Протоколы маршрутизации, такие как OSPF и EIGRP, намного быстрее адаптируются к изменениям в сети, чем **spanning-tree**. Чтобы не отставать от скорости этих протоколов маршрутизации, была создана еще одна разновидность связующего дерева... (**rapid spanning tree**) быстрое связующее дерево.

Rapid spanning tree - это не революция spanning tree, а его эволюция. Некоторые вещи были изменены для того, что бы ускорить процесс, но с точки зрения конфигурации - это то же самое, что классический spanning tree . Я называю оригинальное spanning tree "**классическим spanning tree**".

АЗЫ RAPID SPANNING TREE



Помните состояние портов spanning tree? У нас есть блокирующее, прослушивающее, обучающее и пересылающее состояние порта. Это первое различие между spanning tree и rapid spanning tree. Rapid spanning tree имеет только три состояния портов:

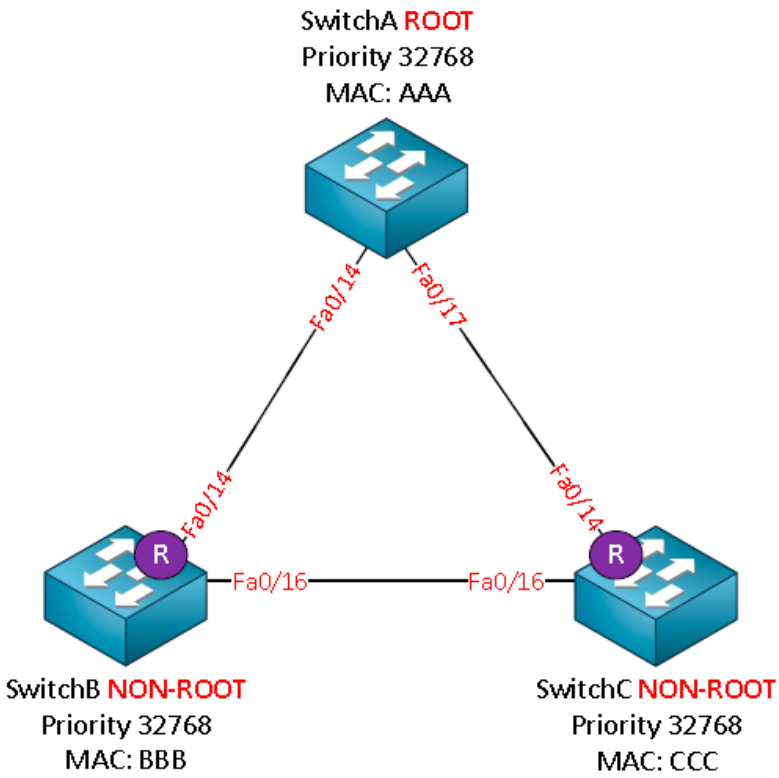
- **Отбрасывание (**Discarding**)**;
- **Обучение** (Learning);
- **Пересылка** (Forwarding).

Вы уже знакомы с состоянием порта в режиме обучения и пересылки, но отбрасывание - это новое состояние порта. В основном он объединяет в себе блокировку и прослушивание состояния порта.

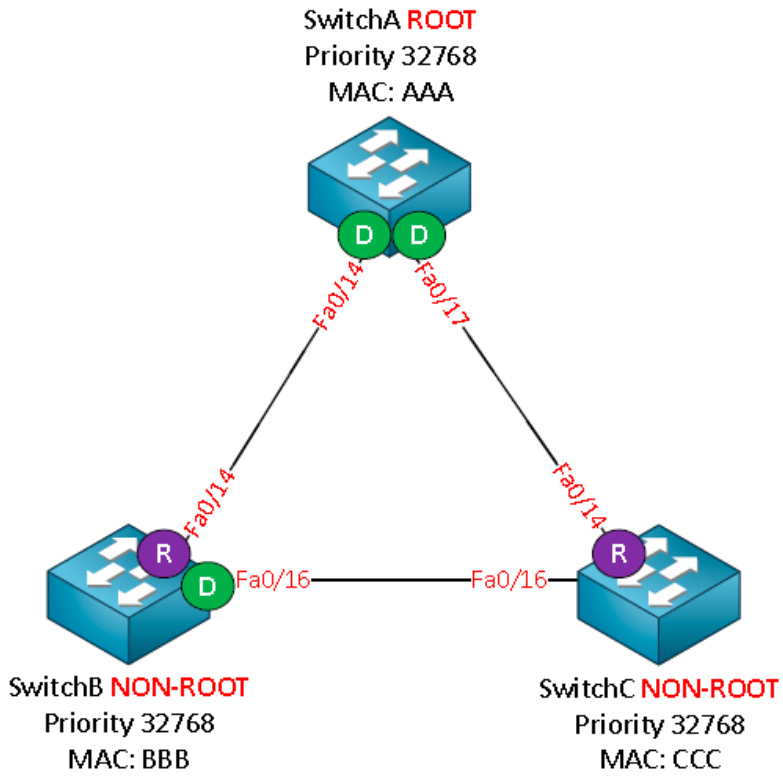
Spanning Tree	Rapid Spanning Tree	Порт активен в топологии?	Изучает MAC - адрес?
Блокировка	Отбрасывание	Нет	Нет
Прослушивание	Отбрасывание	Да	Нет
Обучение	Обучение	Да	Да
Пересылка	Пересылка	Да	Да

Вот хороший обзор с различными состояниями портов для spanning tree и rapid spanning tree. В таблице отображено состояние портов: **активны ли** они и узнают ли они MAC-адреса или нет.

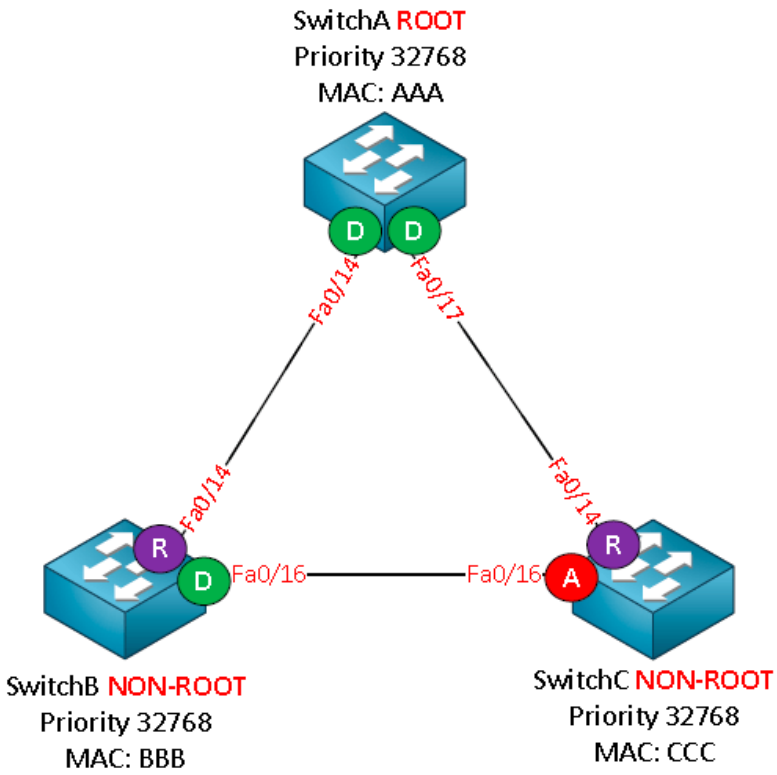
Помните ли вы все остальные роли портов, которые есть у spanning tree? Давайте сделаем небольшой обзор, и будет показано отличие от rapid spanning tree.



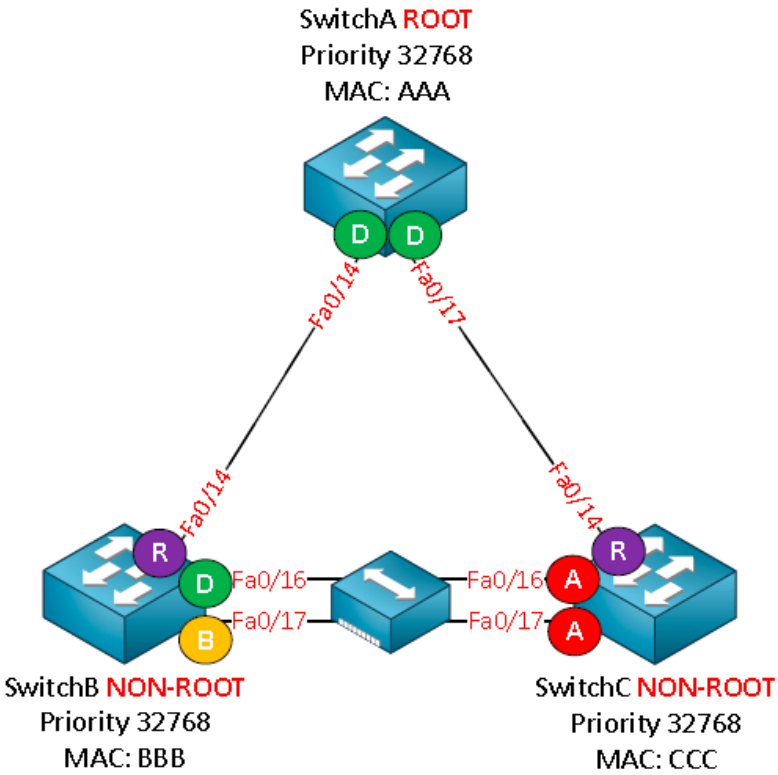
Коммутатор с лучшим ID моста (priority + MAC -адрес) становится корневым мостом. Другие коммутаторы (**non-root**) должны найти кратчайший путь стоимости к корневому мосту. Это корневой порт. Здесь нет ничего нового, все это работает аналогично и в rapid spanning tree.



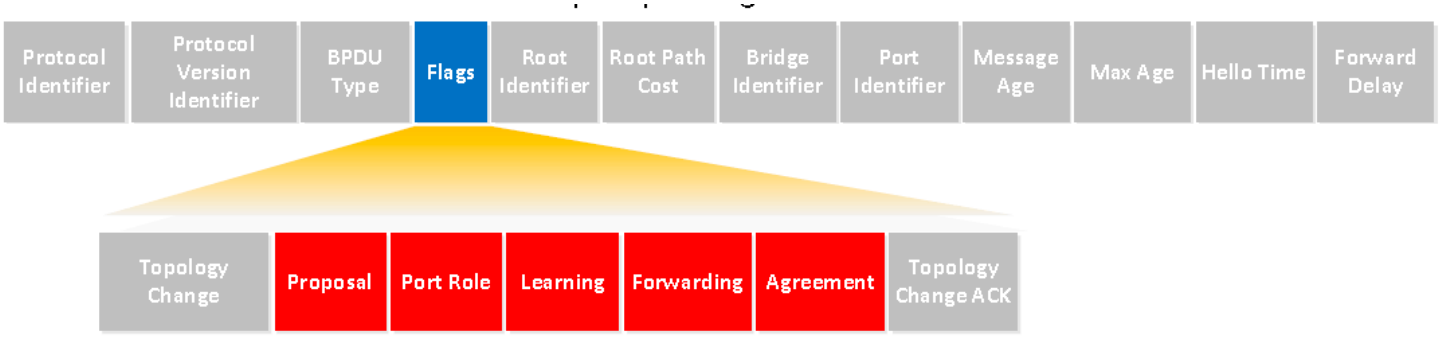
На каждом сегменте может быть только один назначенный порт, иначе мы получим петлю. Порт станет назначенным портом, если он сможет отправить лучший BPDU. Коммутатор A, как корневой мост, всегда будет иметь лучшие порты, поэтому все интерфейсы будут назначены. **Интерфейс fa0/16** на коммутаторе B будет назначенным портом в моем примере, потому что он имеет лучший идентификатор моста, чем коммутатор C. Здесь все еще нет ничего нового по сравнению с классическим связующим деревом.



Коммутатор C получает лучшие **BPDU** на своем интерфейсе fa0/16 от **коммутатора B**, и таким образом он будет заблокирован. Это альтернативный порт, и это все еще то же самое, что и для rapid spanning tree.



Вот вам новый порт, взгляните на интерфейс fa0/17 **коммутатора B**. Он называется резервным портом и является новым для rapid spanning tree. Однако вы вряд ли увидите этот порт в производственной сети. Между коммутатором B и коммутатором C был добавлен хаб. Обычно (без промежуточного концентратора) оба fa0/16 и fa0/17 будут назначены портами. Из-за хаба интерфейсы fa0/16 и fa0/17 **коммутатора B** теперь находятся в одном домене коллизий. Fa0/16 будет выбран в качестве назначенного порта, а fa0/17 станет резервным портом для интерфейса fa0/16. Причина, по которой коммутатор B видит интерфейс fa0/17 в качестве резервного порта, заключается в том, что он получает свои собственные BPDU на интерфейсах fa0/16 и fa0/17 и понимает, что у него есть два соединения с одним и тем же сегментом. Если вы удалите хаб, то fa0/16 и fa0/17 будут назначены портами точно так же, как classic spanning tree.



BPDU отличается для rapid spanning tree. В classic spanning tree поле flags использовало только два бита:

- **Topology change**.;
- **Topology change acknowledgment**.;

Теперь используются все биты поля flags. Роль порта, который создает BPDU, будет добавлена с помощью поля port role, оно имеет следующие параметры:

- **Unknown**;
- **Alternate / Backup port**;
- **Root port**;
- **Designated port**.

Эта BPDU называется BPDUv2. Коммутаторы, работающие со старой версией spanning tree, проигнорируют эту новую версию BPDU. Если вам интересно ... rapid spanning tree и старое spanning tree совместимы! Rapid spanning tree способно работать с коммутаторами, работающими под управлением более старой версии spanning tree.

ЧТО ПОМЕНЯЛОСЬ

BPDU теперь отправляются каждый hello time. Только корневой мост генерирует BPDU в classic spanning tree, и они ретранслировались non-root, если они получали его на свой корневой порт. Rapid spanning tree работает по-разному...все коммутаторы генерируют BPDU каждые две секунды (hello time). Это hello timeпо умолчанию, но вы можете его изменить.

classic spanning tree использует максимального время жизни (20 секунд) для BPDU, прежде чем они будут отброшены. Rapid spanning работает по-другому! BPDU теперь используются в качестве механизма поддержания активности, аналогичного тому, что используют протоколы маршрутизации, такие как OSPF или EIGRP. Если коммутатор пропускает три BPDU от соседнего коммутатора, он будет считать, что подключение к этому коммутатору было потеряно, и он немедленно удалит все MAC-адреса.

Rapid spanning tree будет принимать низшие BPDU. Classic spanning tree игнорирует их. Скорость перехода (время сходимости) является наиболее важной характеристикой rapid spanning tree. Classic spanning tree должно было пройти через состояние прослушивания и обучения, прежде чем оно переведет интерфейс в forwarding состояние, это занимает 30 секунд (таймер по умолчанию). Classic spanning было основано на таймерах.

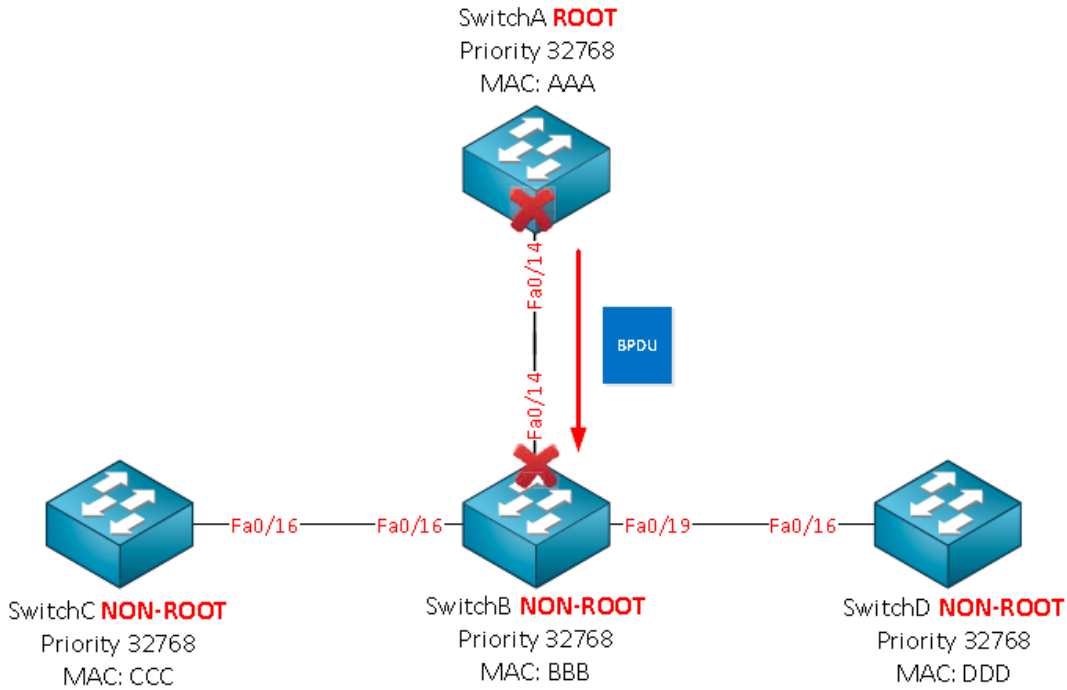
Rapid spanning не использует таймеры, чтобы решить, может ли интерфейс перейти в forwarding состояние или нет. Для этого он будет использовать переговорный (negotiation) механизм. Чуть позже я покажу вам, как это работает.

Помните ли вы понятие portfast? Если мы включим portfast во время запуска classic spanning tree, оно пропустит состояние прослушивания и обучения и сразу же переведет интерфейс в forwarding состояние. Помимо перевода интерфейса в forwarding состояние, он также не будет генерировать изменения топологии, когда интерфейс переходит в состояние UP или DOWN. Мы все еще используем portfast для rapid spanning tree, но теперь он называется пограничным портом (edge port).

Rapid spanning tree может только очень быстро переводить интерфейсы в forwarding состояние на edge ports (portfast) или интерфейсы типа point-to-point. Он будет смотреть на link type, и есть только два ink types:

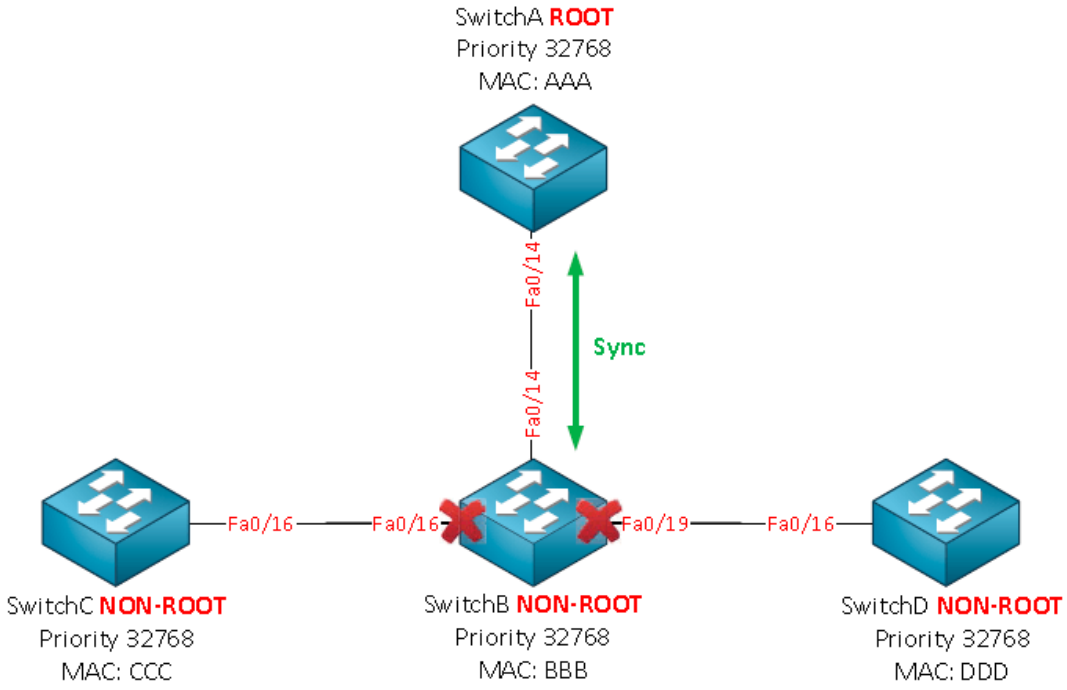
- **Point-to-point (full duplex)**;
- **Shared (half duplex)**.

Обычно мы используем коммутаторы, и все наши интерфейсы настроены как full duplex, rapid spanning tree видит эти интерфейсы как point-to-point. Если мы введем концентратор в нашу сеть, то у нас будет half duplex, который рассматривается как shared interface к rapid spanning-tree.

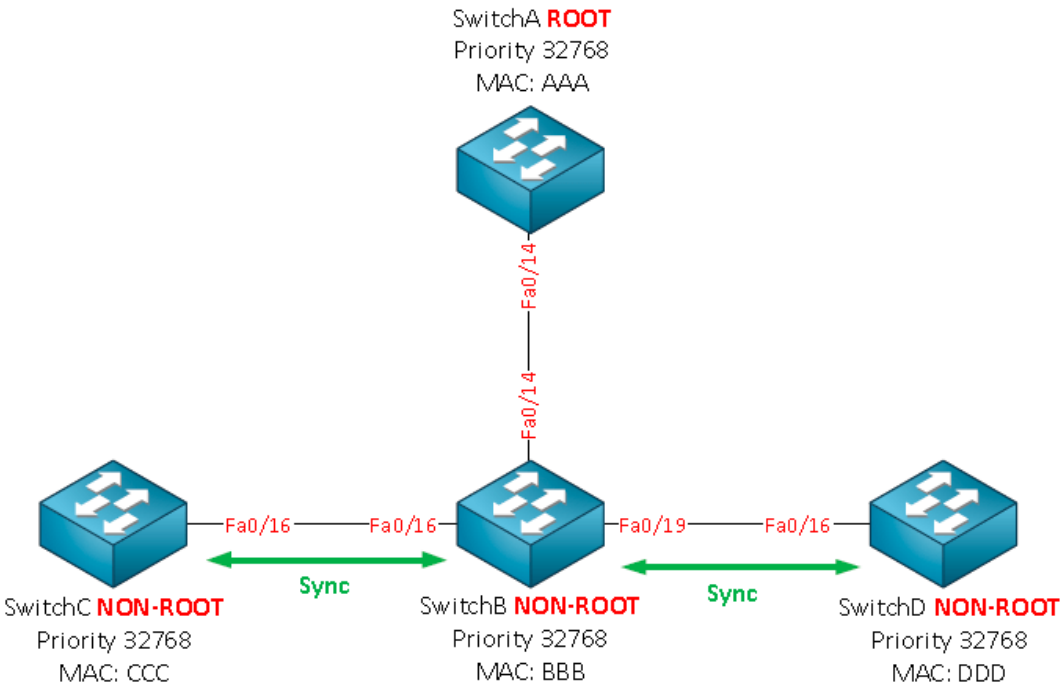


Позвольте мне описать механизм быстрой синхронизации spanning tree, используя рисунок выше. Коммутатор A сверху - это корневой мост. Коммутатор B, C и D- некорневые мосты (non-root).

Как только появится связь между коммутатором A и коммутатором B, их интерфейсы будут находиться в режиме блокировки. Коммутатор B получит BPDU от **коммутатора A**, и теперь будет происходить согласование, называемое синхронизацией.

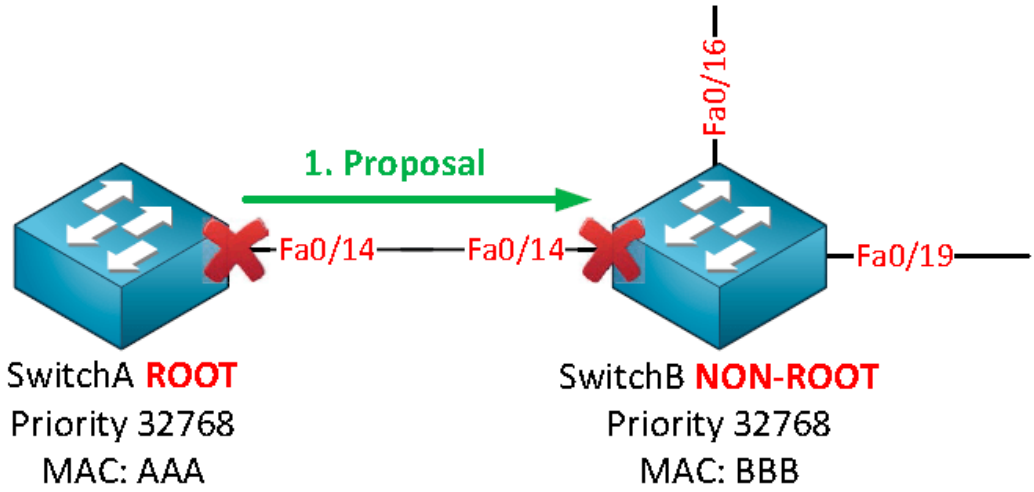


После того, как **коммутатор B** получил **BPDU** от корневого моста, он немедленно блокирует все свои порты, не обозначенные в списке **non-edge**. Non-edge порты - это интерфейсы для подключения к другим коммутаторам, пока **edge** порты- интерфейсы, настроены как portfast. Как только коммутатор B блокирует свои non-edge порты, связь между коммутатором A и коммутатором B переходит в **forwarding** состояние.

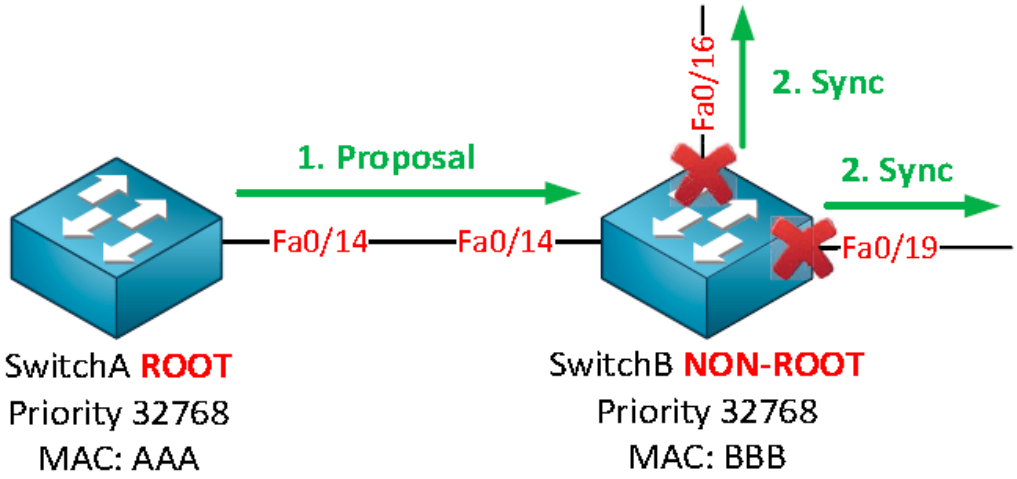


Коммутатор B также выполнит операцию синхронизации как с коммутатором C, так и с коммутатором D, чтобы они могли быстро перейти в forwarding состояние.

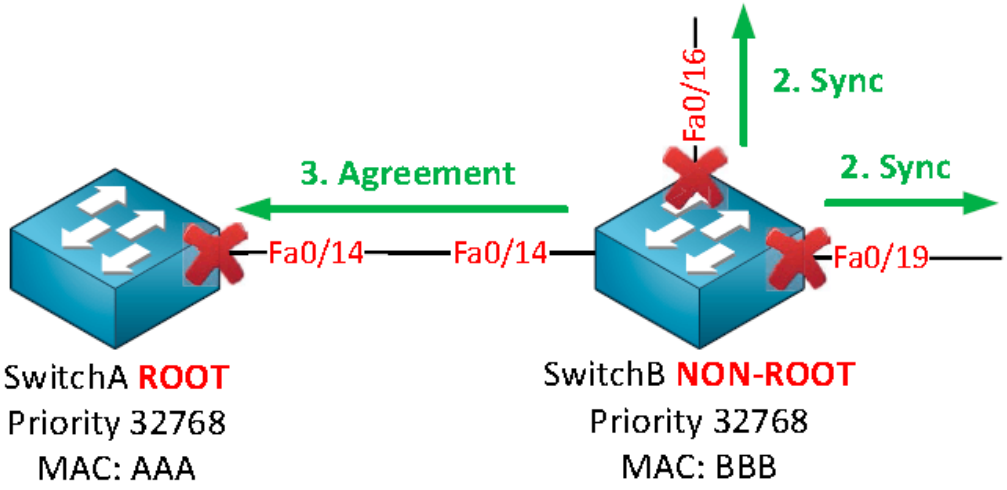
Главное, что следует усвоить здесь, заключается в том, что rapid spanning tree использует этот механизм синхронизации вместо механизма "таймера", который использует classic spanning tree (**прослушивание** → **обучение** → **forwarding**).



Давайте увеличим масштаб механизма синхронизации rapid spanning tree, подробно рассмотрев коммутатор A и коммутатор B. Сначала интерфейсы будут заблокированы до тех пор, пока они не получат BPDU друг от друга. В этот момент коммутатор B поймет, что коммутатор A является корневым мостом, потому что он имеет лучшую информацию BPDU. Механизм синхронизации начнется, потому что коммутатор A установит **proposal bit** в поле flag BPDU.

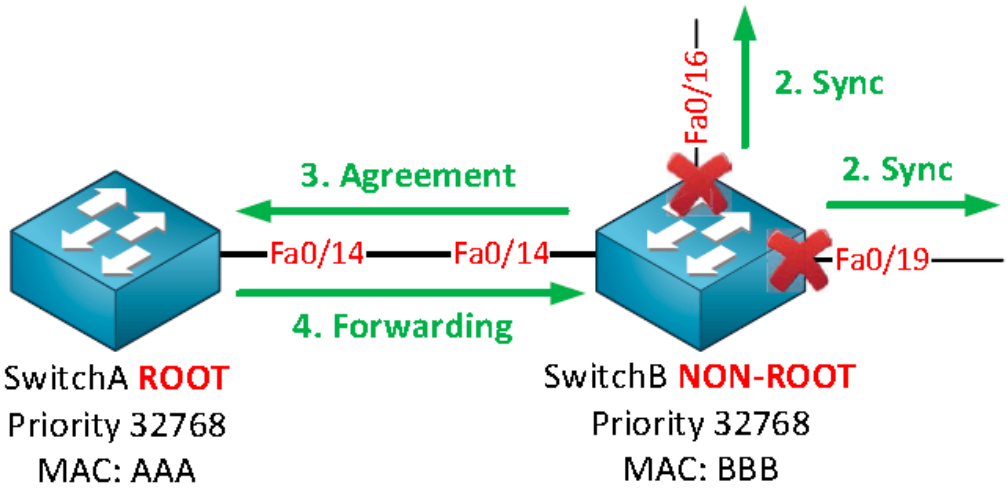


Коммутатор B получает предложение от **коммутатора A** и понимает, что он должен что-то сделать. Он заблокирует все свои non-edge интерфейсы и запустит синхронизацию в направлении **коммутатора C** и **коммутатора D**.

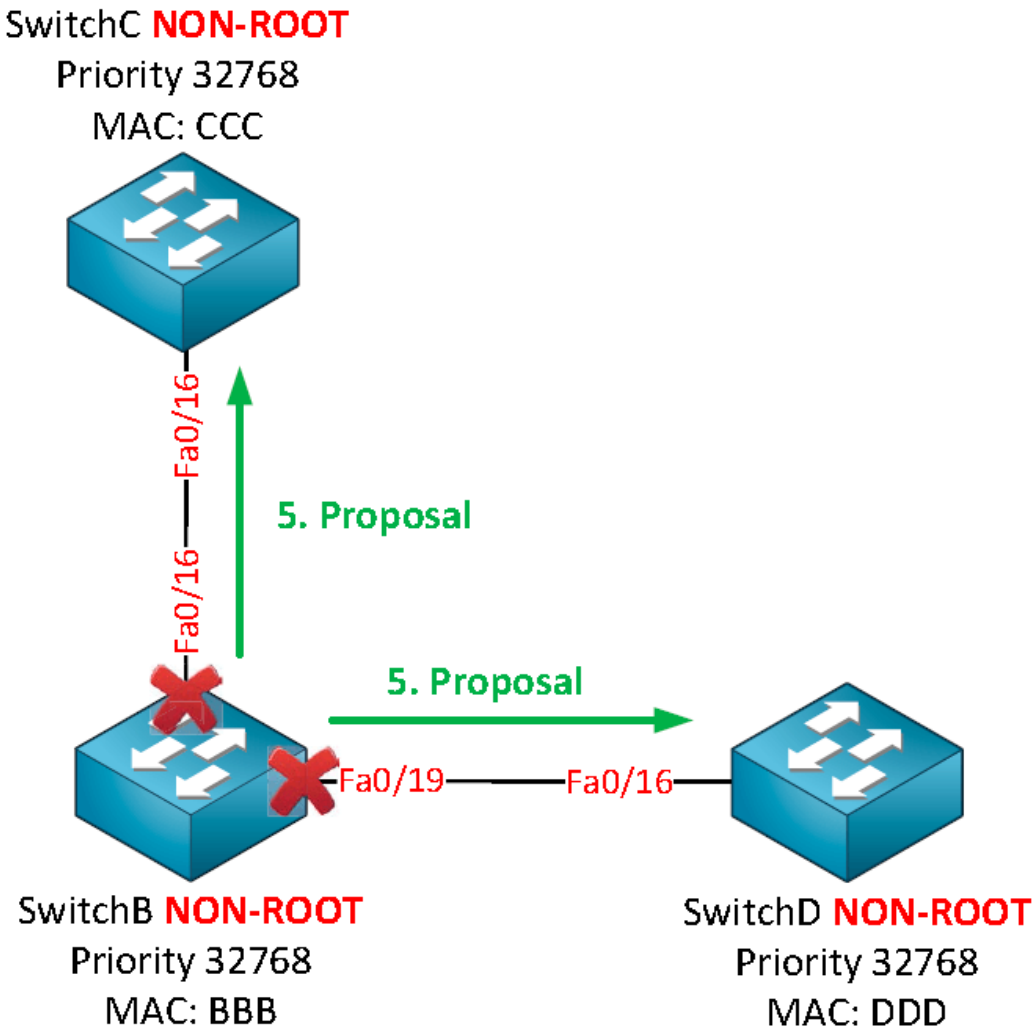


Как только коммутатор B перевед свои интерфейсы в режим синхронизации, это позволит коммутатору A узнать об этом, отправив соответствующее соглашение.

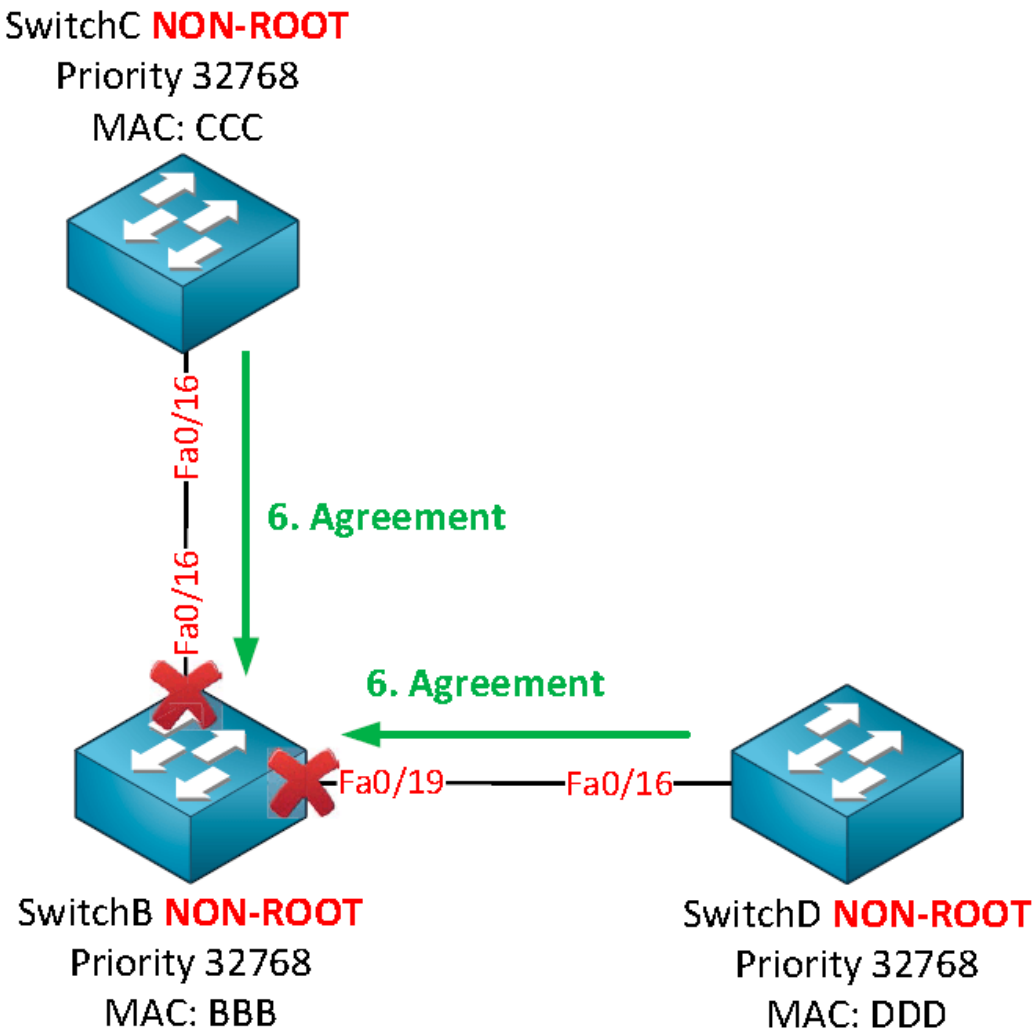
Это соглашение является копией proposal BPDU, где proposal bit, был **switched off**, a agreement bit - switched on. Интерфейс fa0/14 на коммутаторе B теперь перейдет в режим forwarding.



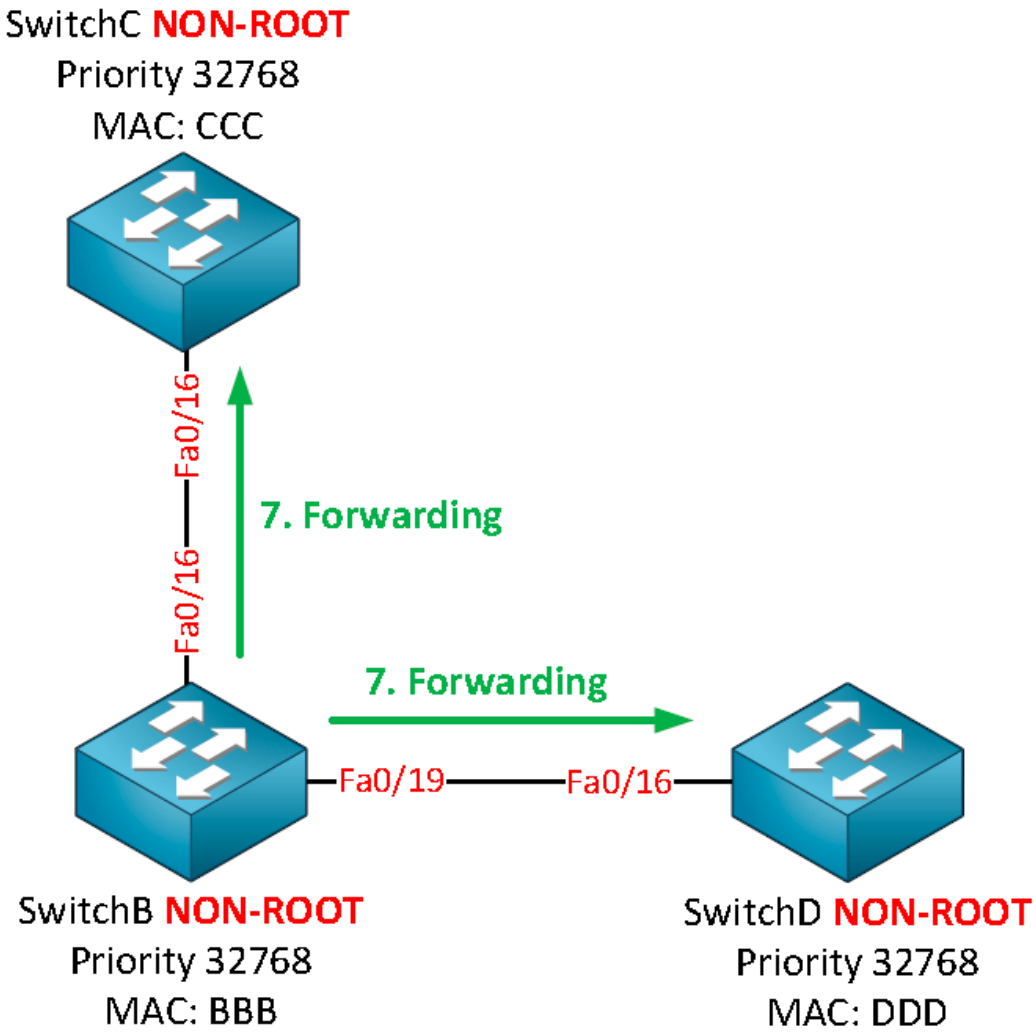
Как только коммутатор A получит соглашение от коммутатора B, он немедленно переведет свой интерфейс fa0/14 в режим пересылки. А как насчет интерфейса fa0 / 16 и fa0 / 19 на коммутаторе B?



Точно такой же механизм синхронизации будет иметь место и сейчас на этих интерфейсах. Коммутатор B направит предложение по своим интерфейсам fa0/16 и fa0/19 в сторону коммутатора C и коммутатора D.



Коммутатор C и **коммутатор D** не имеют никаких других интерфейсов, поэтому они отправят соглашение обратно на коммутатор B.



Коммутатор B переведет свои интерфейсы fa0/16 и fa0/19 в режим forwarding, и на этом мы закончим. Этот механизм синхронизации - всего лишь пара сообщений, летающих туда-сюда, и очень быстро, это намного быстрее, чем механизм на основе таймера classic spanning tree!

ЧТО ЕЩЕ НОВОГО В RAPID SPANNING TREE?

Есть еще три вещи:

- **UplinkFast**;
- **Механизм изменения топологии**;
- **Совместимость с классическим связующим деревом**.

Когда вы настраиваете classic spanning tree, вы должны включить **UplinkFast** самостоятельно. Rapid spanning tree использует UpLinkFast по умолчанию, вам не нужно настраивать его самостоятельно. Когда коммутатор теряет свой корневой порт, он немедленно переводит свой альтернативный порт в forwarding.

Разница заключается в том, что classic spanning tree нуждалось в **multicast** кадрах для обновления таблиц MAC-адресов всех коммутаторов.

Нам это больше не нужно, потому что механизм изменения топологии для rapid spanning tree отличается. Так что же изменилось в механизме изменения топологии?

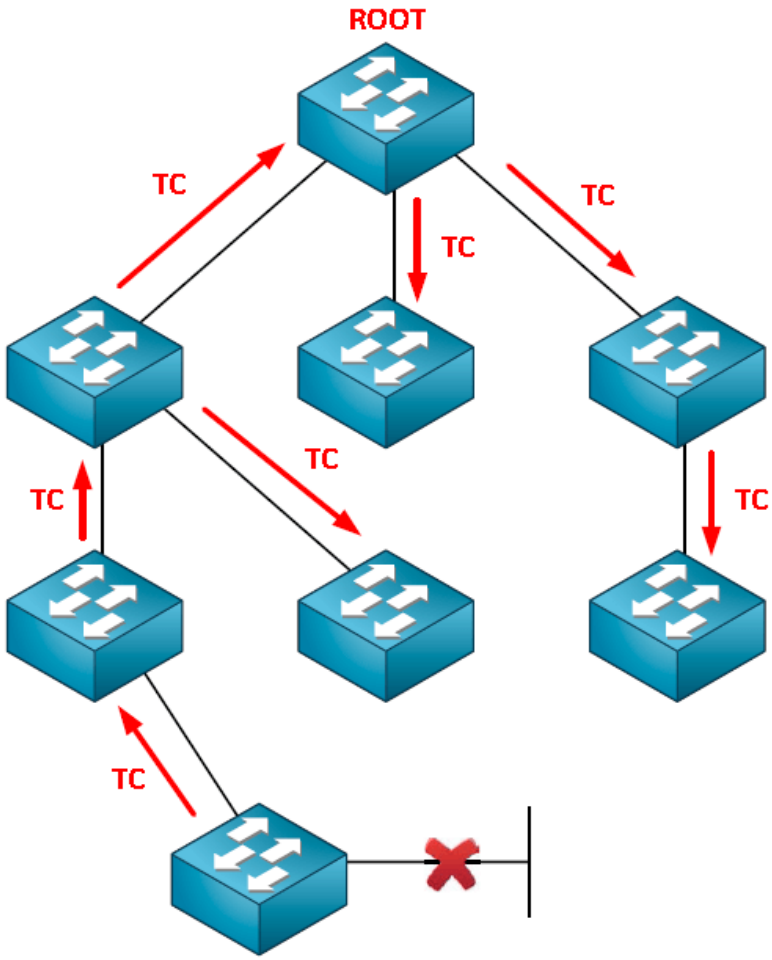
С classic spanning tree сбой связи вызвал бы изменение топологии. При использовании rapid spanning tree сбой связи не влияет на изменение топологии. Только non-edge интерфейсы (ведущие к другим коммутаторам), которые переходят в forwarding состояние, рассматриваются как изменение топологии. Как только коммутатор обнаружит изменение топологии это произойдет:

- Он начнет **изменение топологии** при значении таймера, которое в два раза превышает hello time. Это будет сделано для всех назначенных non-edge и корневых портов.;
- Он **будет очищать MAC-адреса**, которые изучаются на этих портах.;

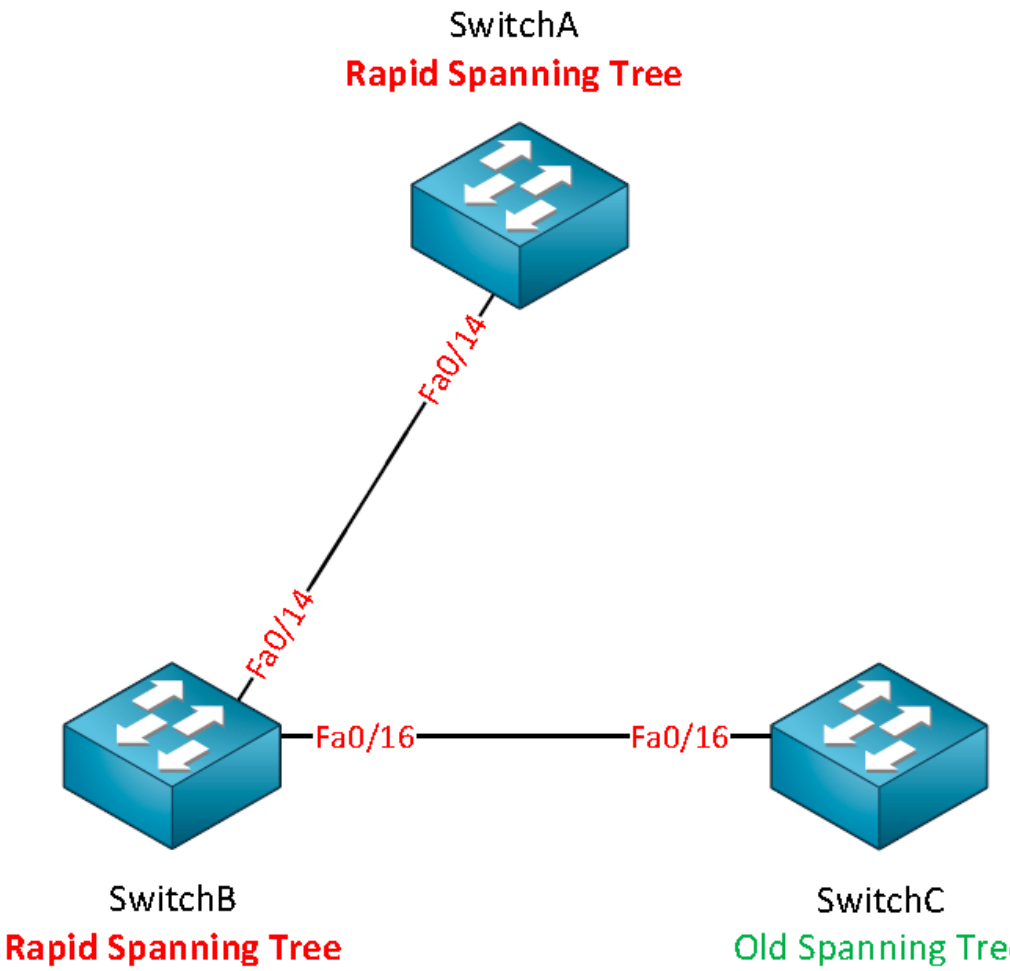
- До тех пор, пока происходит изменение топологии, во время активности таймера, он будет **устанавливать бит изменения топологии** в BPDU, которые отправляются из этих портов. BPDU также будет отправлен из своего корневого порта.;

Когда соседний коммутатор получит этот BPDU с установленным битом изменения топологии, произойдет следующее:

- Он **очистит все свои MAC-адреса** на всех интерфейсах, кроме того, на котором он получил BPDU с включенным изменением топологии.;
- Он **запустит изменение топологии** во время самого таймера и отправит BPDU на все назначенные порты и корневой порт, установив бит изменения топологии.;



Вместо того, чтобы отправлять изменения топологии вплоть до корневого моста, как это делает classic spanning tree, изменение топологии теперь быстро распространяется по всей сети.



И последнее, но не менее важное, давайте поговорим о совместимости. Rapid spanning tree и classic spanning tree совместимы. Однако, когда коммутатор, на котором работает Rapid spanning tree, связывается с коммутатором, на котором работает **classic spanning tree**, все функции скоростной передачи данных не будут работать!

В приведенном выше примере у меня есть три коммутатора. Между коммутатором A и коммутатором B мы запустим rapid spanning tree. Между коммутатором B и коммутатором C мы вернемся к classic spanning tree.

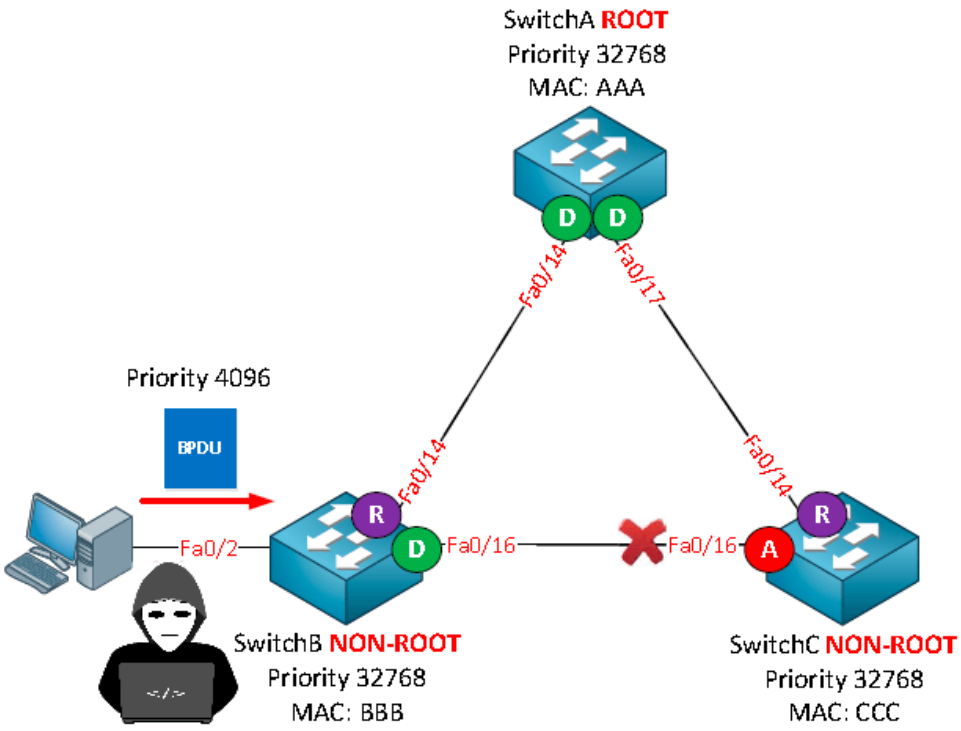
3. Продвинутое руководство по Spanning Tree: Toolkit

Выходим на новый уровень. Для изучения следующей темы вы уже должны хорошо понимать **связующее дерево**. Связующее дерево (Spanning Tree Protocol STP) — это важная тема. Есть много вещей, которые могут пойти не так, и в этой статье мы рассмотрим ряд инструментов, которые мы можем использовать для защиты нашей топологии связующего дерева.

Для профессионалов

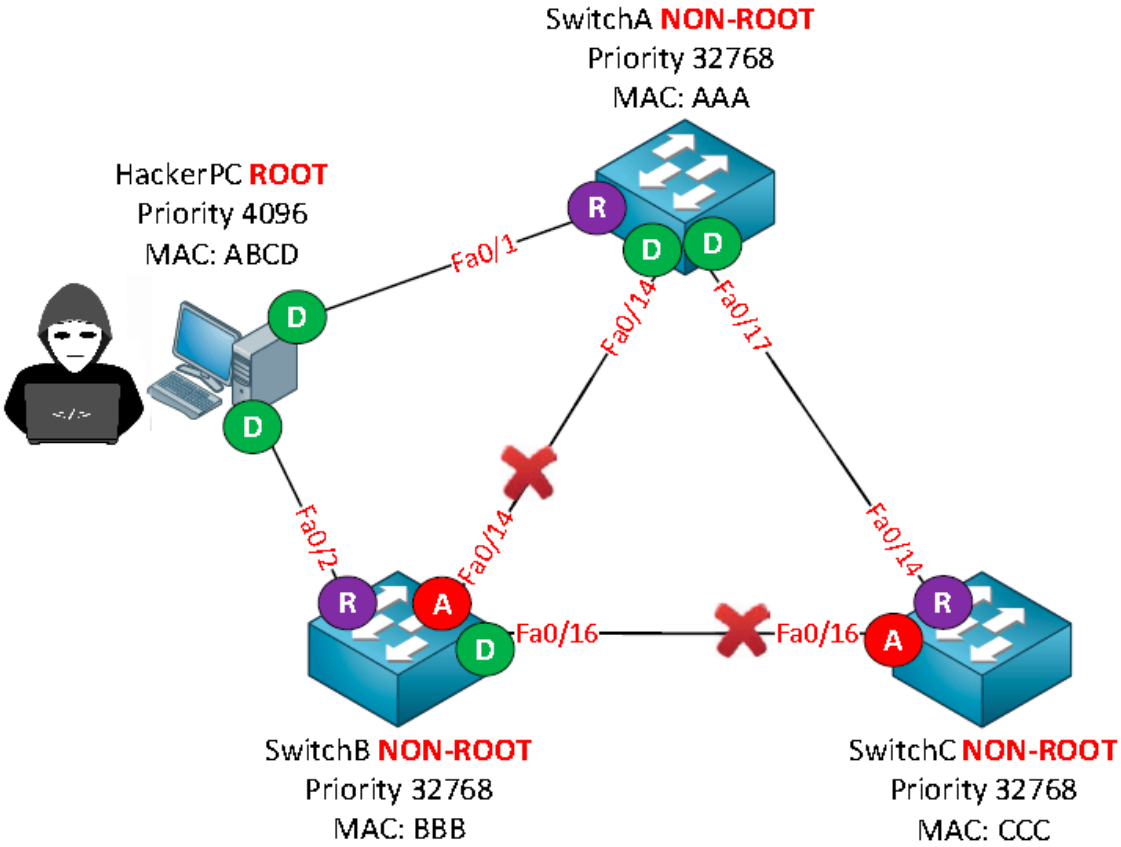
- **PortFast**: Он настроит порт доступа как пограничный порт, поэтому он переходит в режим **forwarding** немедленно.
- **BPDU Guard**: это отключит (**err-disable**) интерфейс, который имеет настроенный **PortFast**, если он получает **BPDU**.
- **BPDUFilter**: это будет подавлять **BPDU** на интерфейсах.
- **Root Guard**: это предотвратит превращение соседнего коммутатора в корневой мост, даже если он имеет лучший идентификатор моста.
- **UplinkFast**: мы видели это в статье о связующем дереве. Он улучшает время конвергенции.
- **BackboneFast**: мы также видели это в статье о связующем дереве. Оно улучшает время конвергенции, если у вас есть сбой косвенной связи.

UplinkFast и BackboneFast не требуются для rapid spanning tree, поскольку оно уже реализовано по умолчанию. Мы начнем с BPDUguard:

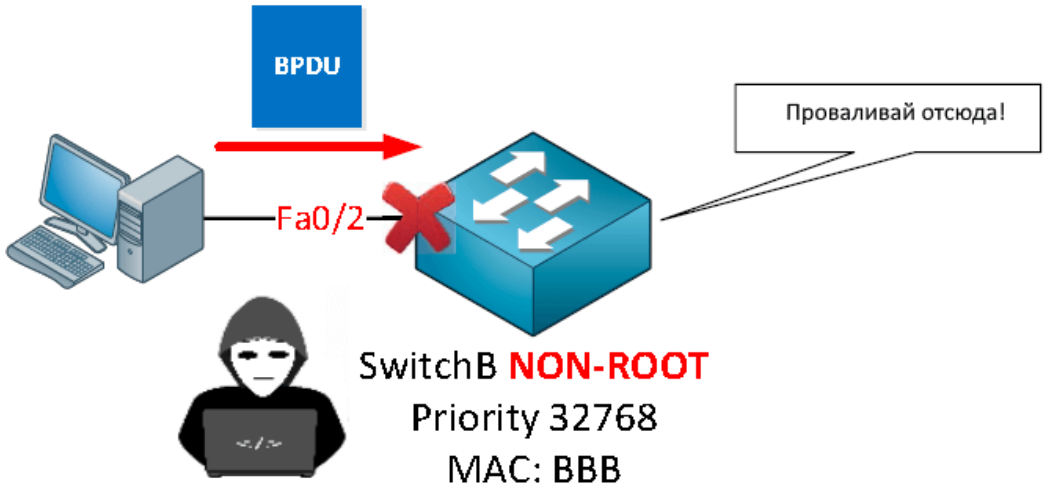


В топологии выше мы имеем идеально работающую топологию остовного дерева. По умолчанию связующее дерево будет отправлять и получать BPDU на всех интерфейсах. В нашем примере у нас есть компьютер, подключенный на интерфейсе `fa0/2` коммутатора B. Есть кто-то, кто с враждебными намерениями мог бы запустить инструмент, который

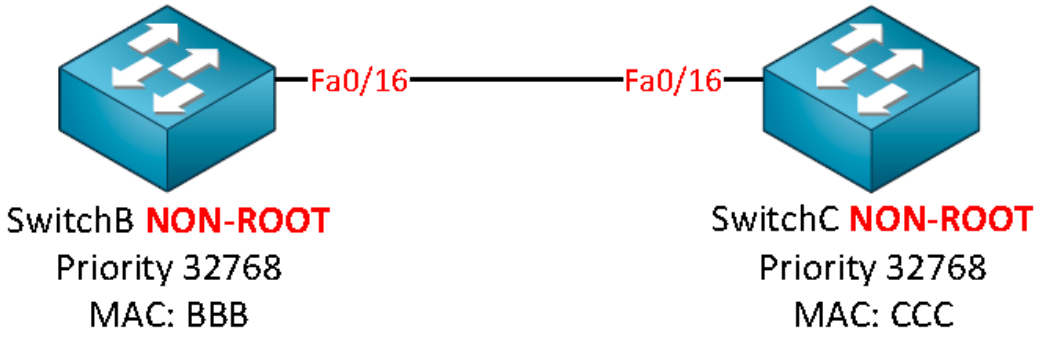
сгенерирует BPDU с превосходящим ID моста. Что же произойдет- так это то, что наши коммутаторы будут считать, что корневой мост теперь может быть достигнут через коммутатор B, и у нас будет повторный расчет связующего дерева. Звучит не очень хорошо, правда?



Можно поставить человека (хакера) в середине топологии для атаки так, чтобы никто не знал. Представьте себе, что хакер подключает свой компьютер к двум коммутаторам. Если хакер станет корневым мостом, то весь трафик от коммутатора A или коммутатора C к коммутатору B будет проходить через него. Он запустит **Wireshark** и подождет, пока произойдет чудо.



BPDUguard гарантирует, что, когда мы получаем BPDU на интерфейс, интерфейс перейдет в режим err-disable.



Чтобы продемонстрировать работу BPDUguard будем использовать два коммутатора. Настроем интерфейс `Fa0/16` коммутатора B так, что он перейдет в режим err-disable, если он получит BPDU от коммутатора C.

```
SwitchB(config)#interface fa0/16
SwitchB(config-if)#spanning-tree bpduguard enable
```

Вот как вы включаете его в интерфейсе. Имейте в виду, что обычно вы никогда не будете делать это между коммутаторами. Вы должны настроить это на интерфейсах в режиме доступа, которые подключаются к компьютерам.

```
SwitchB#
%SPANTREE-2-BLOCK_BPDUGUARD: Received BPDU on port Fa0/16 with BPDU Guard
enabled. Disabling port.
%PM-4-ERR_DISABLE: bpduguard error detected on Fa0/16, putting Fa0/16 in
errdisable
state
: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/16, changed
state to down
%LINEPROTO-5-UPDOWN: Line protocol on Interface Vlan1, changed state to down
*Mar 1 00:19:32.089: %LINK-3-UPDOWN: Interface FastEthernet0/16, changed
state to down
```

А-а... вот и наш интерфейс.

```
SwitchB(config-if)#no spanning-tree bpduguard
SwitchB(config-if)#shutdown
SwitchB(config-if)#no shutdown
```

Избавиться от BPDUguard можно используя команды `shut/no shut`, чтобы сделать интерфейс снова рабочим.

```
SwitchB(config)#spanning-tree portfast bpduguard
```

Вы также можете использовать команду `spanning-tree portfast bpduguard`. Это позволит глобально активировать BPDUguard на всех интерфейсах, которые имеют включенный portfast.

```
SwitchB(config)#spanning-tree portfast default
```

Portfast также может быть включен глобально для всех интерфейсов, работающих в режиме доступа.

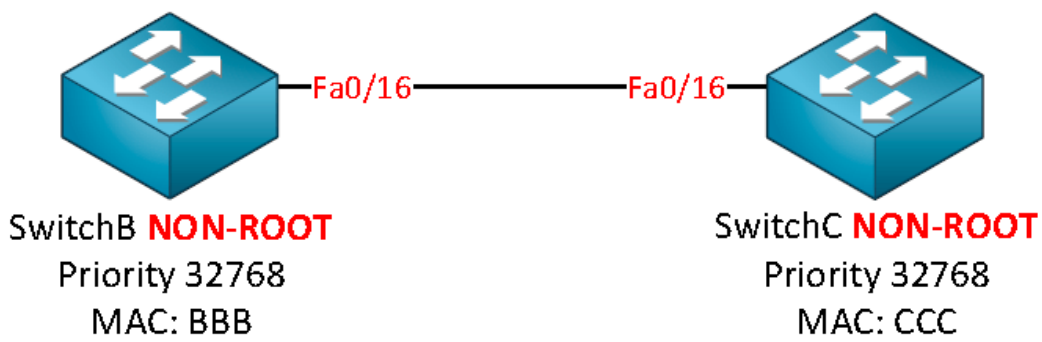
```
SwitchB#show spanning-tree summary
Switch is in pvst mode
Root bridge for: none
Extended system ID is enabled
Portfast Default is enabled
PortFast BPDU Guard Default is enabled
Portfast BPDU Filter Default is disabled
Loopguard Default is disabled
EtherChannel misconfig guard is enabled
UplinkFast is disabled
BackboneFast is disabled
Configured Pathcost method used is short
```

Это полезная команда, позволяющая проверить свою конфигурацию. Вы видите, что portfast и BPDUGuard были включены глобально.

BPDUGuard переведет интерфейс в режим err-disable. Кроме того, можно фильтровать сообщения BPDU с помощью BPDUfilter. BPDUfilter может быть настроен глобально или на уровне интерфейса и есть разница:

- **Глобальный**: если вы включите bpdufilter глобально, любой интерфейс с включенным portfast станет стандартным портом.
- **Интерфейс**: если вы включите BPDUfilter на интерфейсе, он будет игнорировать входящие BPDU и не будет отправлять никаких BPDU.

Вы должны быть осторожны, когда включаете BPDUfilter на интерфейсах. Вы можете использовать его на интерфейсах в режиме доступа, которые подключаются к компьютерам, но убедитесь, что вы никогда не настраиваете его на интерфейсах, подключенных к другим коммутаторам. Если вы это сделаете, вы можете получить цикл.



Для демонстрации работы BPDUfilter мы будем снова использовать коммутатор B и коммутатор C.

```
SwitchB(config)#interface fa0/16
SwitchB(config-if)#spanning-tree portfast trunk
SwitchB(config-if)#spanning-tree bpdufilter enable
```

Он перестанет посылать BPDU и будет игнорировать все, что будет получено.

```
SwitchB#debug spanning-tree bpdu
```

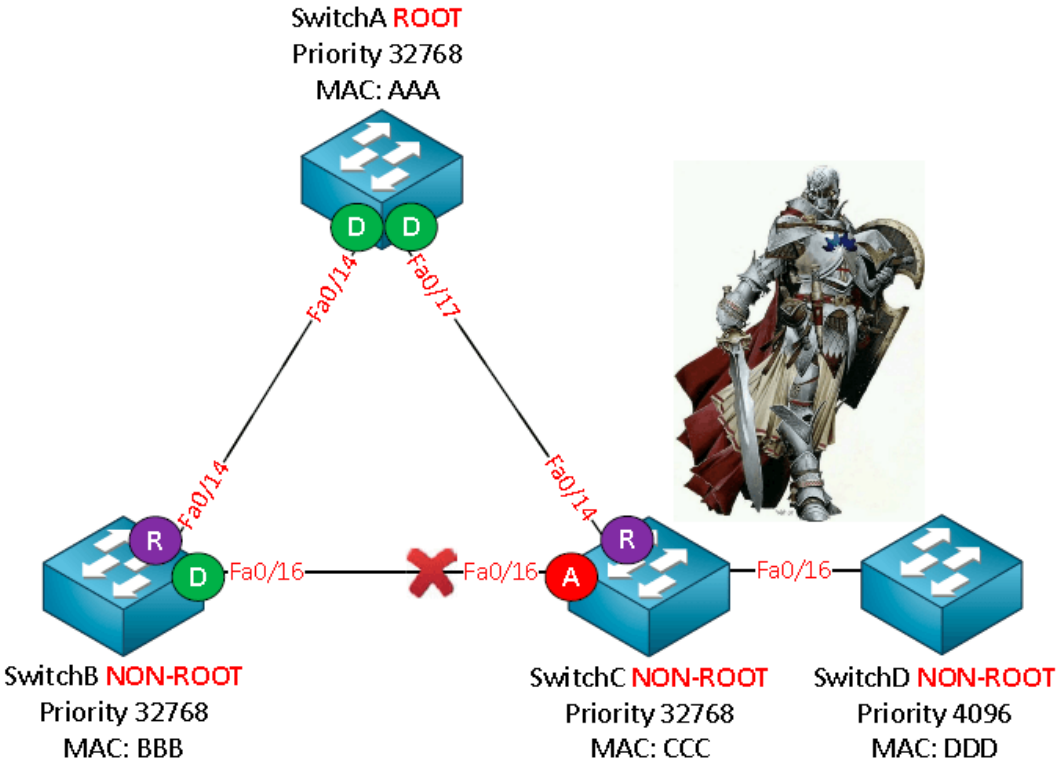
Вы не увидите никаких интересных сообщений, но если вы включите отладку BPDU, то заметите, что он больше не отправляет никаких BPDU. Если вы хотите, вы также можете включить отладку BPDU на коммутаторе C, и вы увидите, что нет ничего от коммутатора B.

```
SwitchB(config)#interface fa0/16
SwitchB(config-if)#no spanning-tree bpdufilter enable
```

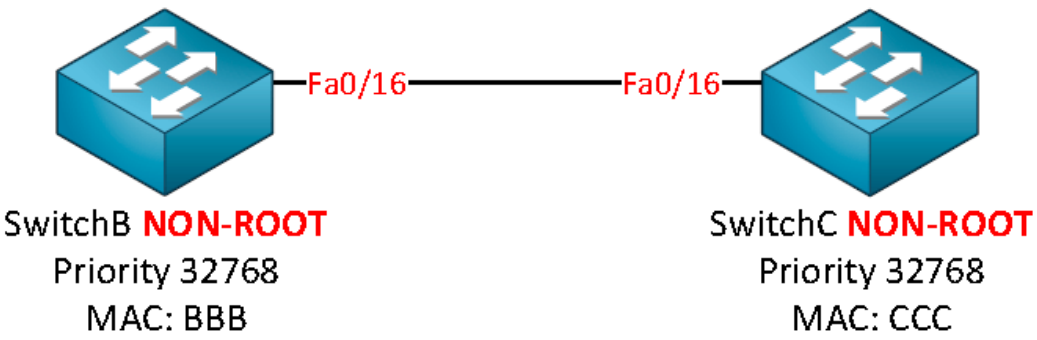
Давайте избавимся от команды BPDUfilter на уровне интерфейса.

```
SwitchB(config)#spanning-tree portfast bpdufilter default
```

Вы также можете использовать глобальную команду для BPDUfilter. Это позволит включить BPDUfilter на всех интерфейсах, которые имеют portfast.



Еще один вариант, с помощью которого мы можем защитить наше связующее дерево, - это использовать **RootGuard**. Проще говоря, RootGuard позаботится о том, чтобы вы не принимали определенный коммутатор в качестве корневого моста. BPDU отправляются и обрабатываются нормально, но, если коммутатор внезапно отправляет BPDU с идентификатором верхнего моста, вы не будете принимать его в качестве корневого моста. Обычно коммутатор D становится корневым мостом, потому что у него есть лучший идентификатор моста, к счастью, у нас есть RootGuard на коммутатое C, так что этого не произойдет!



Рассмотрим с вами конфигурацию с коммутатором B и коммутатором C.

```
SwitchB(config)#spanning-tree vlan 1 priority 4096
```

Давайте убедимся, что коммутатор C не является корневым мостом.

```
SwitchB(config)#interface fa0/16
SwitchB(config-if) #spanning-tree guard root
%SPANTREE-2-ROOTGUARD_CONFIG_CHANGE: Root guard enabled on port
FastEthernet0/16.
```

Вот как мы включаем RootGuard на интерфейсе.

```
SwitchB#debug spanning-tree events
Spanning Tree event debugging is on
```

Не забудьте включить отладку, если вы хотите увидеть события.

```
SwitchC(config)#spanning-tree vlan 1 priority 0
```

Давайте перенастроим коммутатор B, изменив приоритет на наименьшее возможное значение `0` на коммутаторе C. Он теперь должен стать корневым мостом.

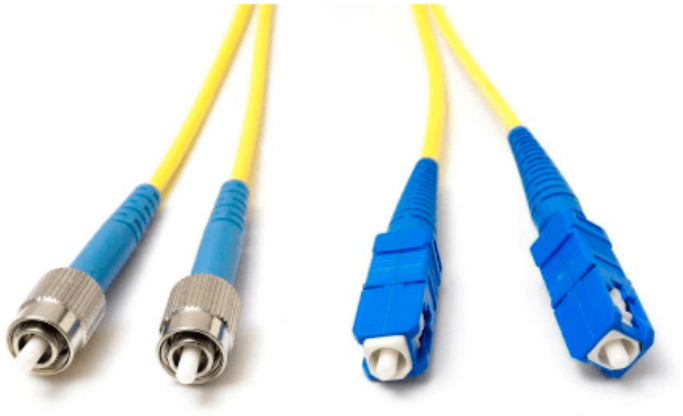
```
SwitchB#
STP: VLAN0001 heard root 1-000f.34ca.1000 on Fa0/16
supersedes 4097-0019.569d.5700
%SPANTREE-2-ROOTGUARD_BLOCK: Root guard blocking port FastEthernet0/16 on
VLAN0001.
```

Вот так коммутатор B не будет принимать коммутатор C в качестве корневого моста. Это заблокирует интерфейс для этой VLAN.

```
SwitchB#show spanning-tree inconsistentports
Name Interface Inconsistency
-------------------- ------------------------ ------------------
VLAN0001 FastEthernet0/16 Root Inconsistent
Number of inconsistent ports (segments) in the system : 1
```

Вот еще одна полезная команда, чтобы проверить, работает ли RootGuard.

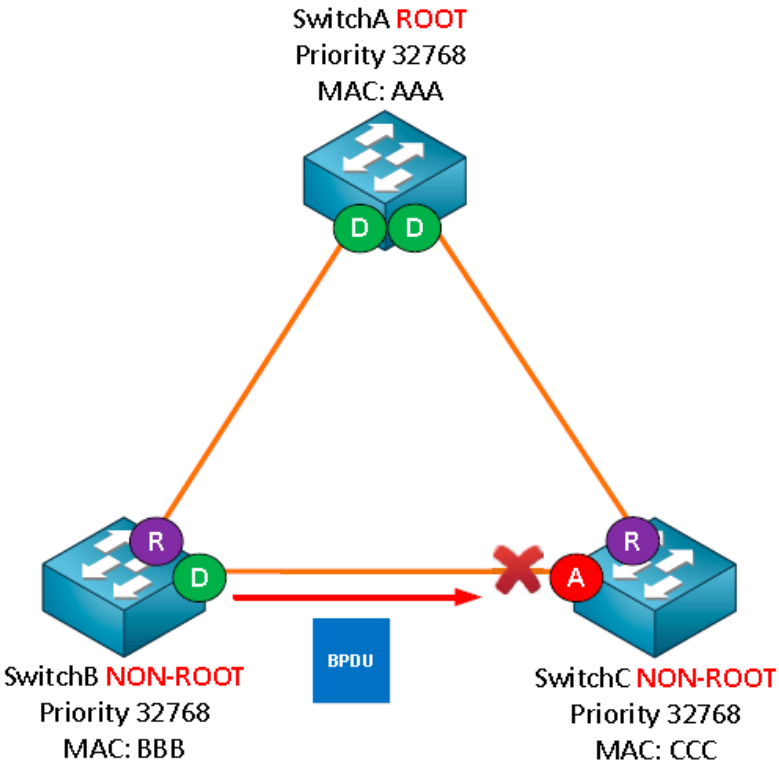
Связующее дерево становится все более безопасным с каждой минутой! Однако есть еще одна вещь, о которой мы должны подумать...



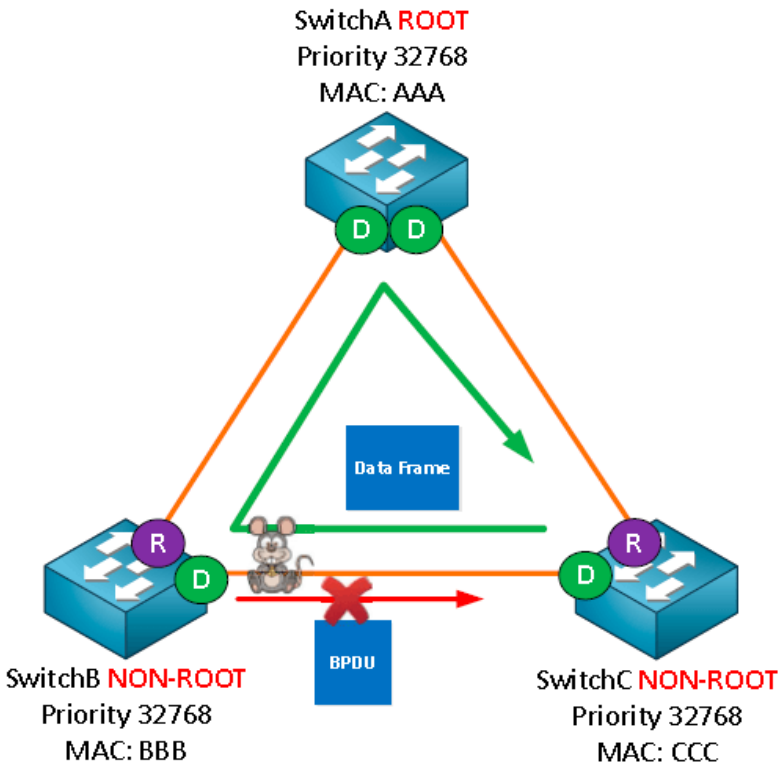
Если вы когда-либо использовали волоконные кабели, вы могли бы заметить, что существует другой разъем для передачи и приема трафика. Если один из кабелей (передающий или принимающий) выйдет из строя, мы получим однонаправленный сбой связи, и это может привести к петлям связующего дерева. Существует два протокола, которые могут решить эту проблему:

- LoopGuard
- UDLD

Давайте начнем с того, что внимательно рассмотрим, что произойдет, если у нас произойдет сбой однонаправленной связи.

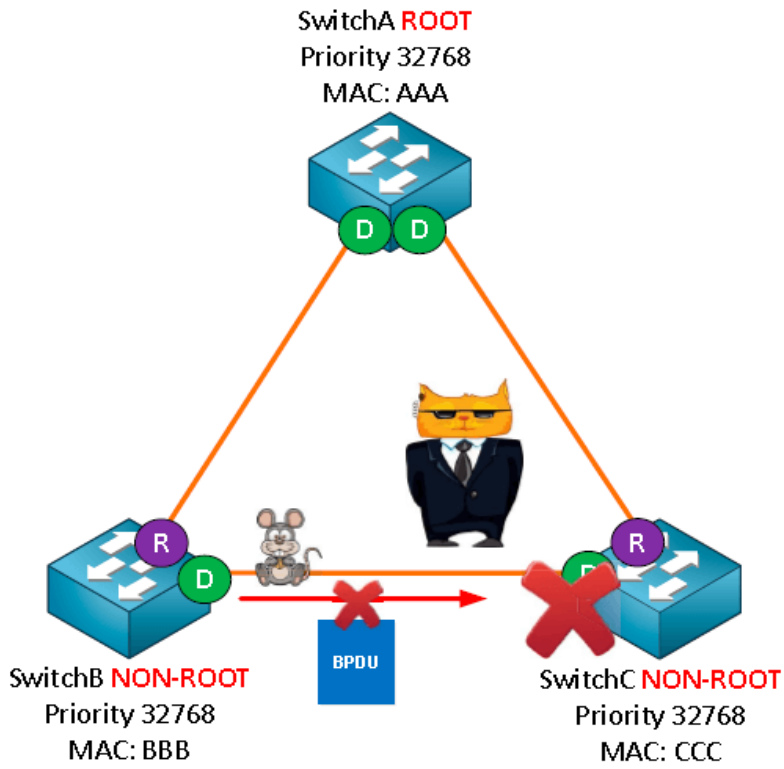


Представьте себе, что между коммутаторами волоконно-оптические соединения. На самом деле имеется другой разъем для передачи и приема. Коммутатор C получает BPDU от коммутатора B, и в результате интерфейс стал альтернативным портом и находится в режиме блокировки.

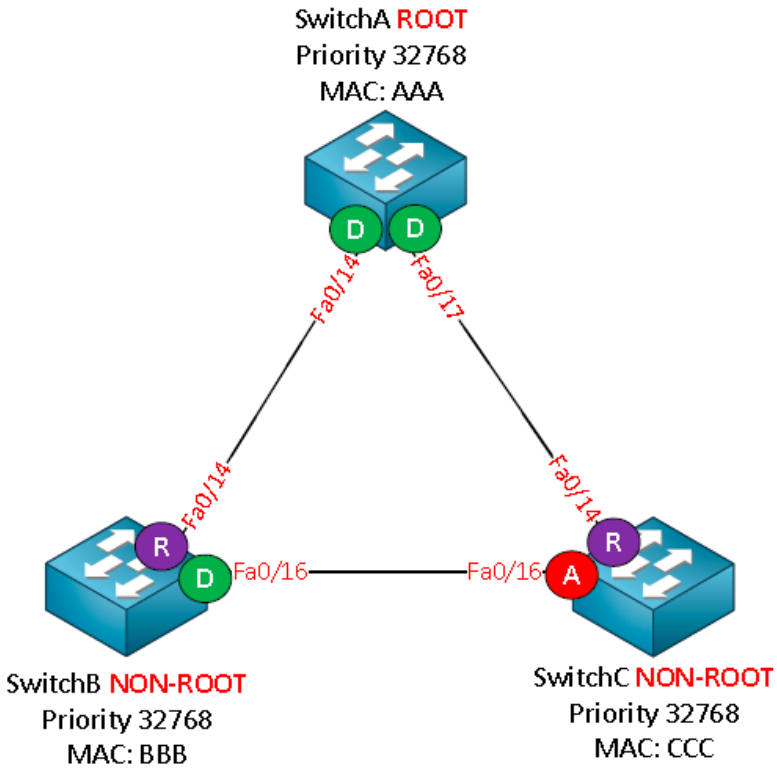


Теперь что-то идет не так... **transmit** коннектор на коммутаторе B к коммутатору C был съеден мышами. В результате коммутатор C не получает никаких BPDU от коммутатора B, но он все еще может отправлять трафик для переключения между ними.

Поскольку коммутатор C больше не получает BPDU на свой альтернативный порт, он перейдет в forwarding режим. Теперь у нас есть **one way loop** (петля в один конец), как указано зеленой стрелкой.



Один из методов, который мы можем использовать для решения нашего однонаправленного сбоя связи — это настройка **LoopGuard**. Когда коммутатор отправляет, но не получает BPDU на интерфейсе, LoopGuard поместит интерфейс в состояние несогласованности цикла и заблокирует весь трафик!



Мы снова будем использовать эту топологию для демонстрации LoopGuard.

```
SwitchA(config)#spanning-tree loopguard default
SwitchB(config)#spanning-tree loopguard default
SwitchC(config)#spanning-tree loopguard default
```

Используйте команду `spanning-tree loopguard` по умолчанию, чтобы включить LoopGuard глобально

```
SwitchB(config)#interface fa0/16
SwitchB(config-if)#spanning-tree portfast trunk
SwitchB(config-if)#spanning-tree bpdufilter enable
```

В примере у нас нет никаких волоконных разъемов, поэтому мы не сможем создать однонаправленный сбой связи. Однако мы можем смоделировать его с помощью BPDUfilter на интерфейсе SwitchB `fa0/16`. Коммутатор C больше не будет получать никаких BPDU на свой альтернативный порт, что заставит его перейти в режим переадресации.

```
SwitchC#
*Mar 1 00:17:14.431: %SPANTREE-2-LOOPGUARD_BLOCK: Loop guard blocking port
FastEthernet0/16 on VLAN0001.
```

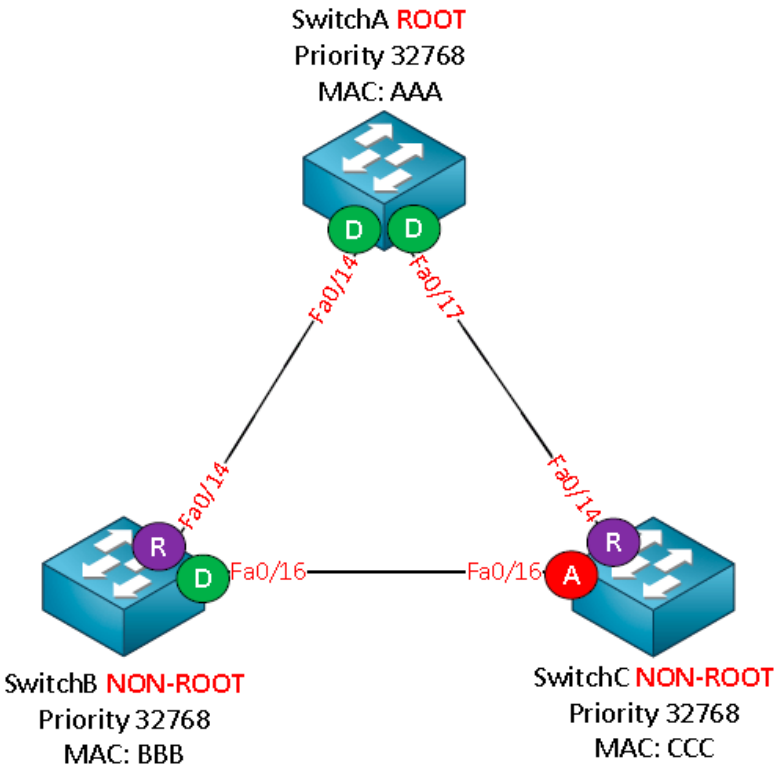
Обычно это вызвало бы петлю, но, к счастью, у нас есть настроенный LoopGuard. Вы можете увидеть это сообщение об ошибке, появляющееся в вашей консоли. Проблема решена!

```
SwitchC(config-if)#spanning-tree guard loop
```

Если вы не хотите настраивать LoopGuard глобально, вы т можете сделать это на уровне интерфейса.

Другой протокол, который мы можем использовать для борьбы с однонаправленными сбоями связи, называется **UDLD (UniDirectional Link Detection)**. Этот протокол не является частью инструментария связующего дерева, но он помогает нам предотвратить циклы.

Проще говоря, UDLD — это протокол второго уровня, который работает как механизм **keepalive**. Вы посылаете приветственные сообщения, вы их получаете, и все прекрасно. Как только вы все еще посылаете приветственные сообщения, но больше их не получаете, вы понимаете, что что-то не так, и мы блокируем интерфейс.



Убедитесь, что вы отключили LoopGuard перед работой с UDLD. Мы будем использовать ту же топологию для демонстрации UDLD.

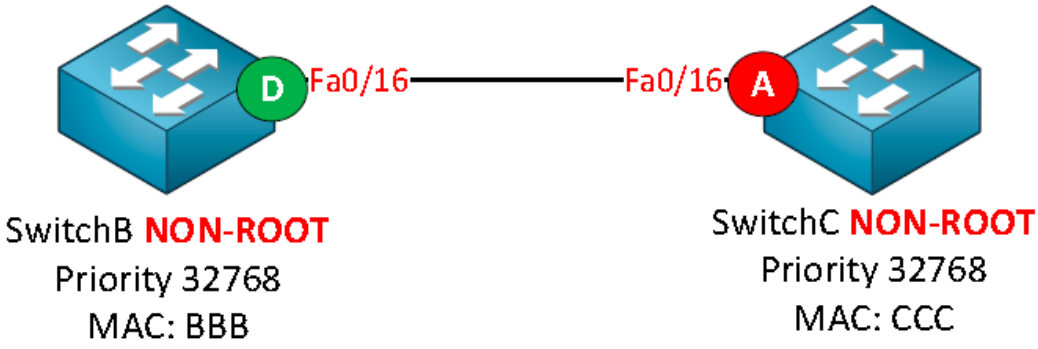
```
SwitchA(config)#udld ?
aggressive Enable UDLD protocol in aggressive mode on fiber ports except
where locally configured
enable Enable UDLD protocol on fiber ports except where locally
configured
message Set UDLD message parameters
```

Существует несколько способов настройки UDLD. Вы можете сделать это глобально с помощью команды udld, но это активирует только UDLD для оптоволоконных линий связи! Существует два варианта для UDLD:

- Normal (default)
- Aggressive

Когда вы устанавливаете UDLD в нормальное состояние, он помечает порт как неопределенный, но не закрывает интерфейс, когда что-то не так. Это используется только для того, чтобы «информировать» вас, но это не предотвратит циклы.

Агрессивный - это лучшее решение, когда пропадает связь с соседом. Он будет посылать кадр UDLD 8 раз в секунду. Если сосед не отвечает, интерфейс будет переведен в режим errdisable.




```
SwitchB(config)#interface fa0/16
SwitchB(config-if)#udld port aggressive
SwitchC(config)#interface fa0/16
SwitchC(config-if)#udld port aggressive
```

Мы будем использовать коммутатор B и C, чтобы продемонстрировать UDLD. Будем использовать агрессивный режим, чтобы мы могли видеть, что интерфейс отключается, когда что-то не так.

```
SwitchB#debug udld events
UDLD events debugging is on
SwitchC#
New entry = 34422DC (Fa0/16)
Found an entry from same device (Fa0/16)
Cached entries = 2 (Fa0/16)
Entry (0x242BB9C) deleted: 1 entry cached
Cached entries = 1 (Fa0/16)
Checking if multiple neighbors (Fa0/16)
Single neighbor detected (Fa0/16)
Checking if link is bidirectional (Fa0/16)
Found my own ID pair in 2way conn list (Fa0/16)
```

Если вы хотите увидеть, что UDLD работает, вы можете попробовать выполнить отладку.

Теперь самое сложное будет имитировать однонаправленный сбой связи. LoopGuard был проще, потому что он был основан на BPDUs. UDLD запускает свой собственный протокол уровня 2, используя собственный MAC-адрес `0100.0ccc.cccc`.

```
SwitchC(config)#mac access-list extended UDLD-FILTER
SwitchC(config-ext-macl)#deny any host 0100.0ccc.cccc
SwitchC(config-ext-macl)#permit any any
SwitchC(config-ext-macl)#exit
SwitchC(config)#interface fa0/16
SwitchC(config-if)#mac access-group UDLD-FILTER in
```

Это творческий способ создавать проблемы. При фильтрации MAC-адреса UDLD он будет думать, что существует сбой однонаправленной связи!

```
SwitchB#
UDLD FSM updated port, bi-flag udld_empty_echo, phase udld_detection (Fa0/16)
timeout timer = 0 (Fa0/16)
Phase set to EXT. (Fa0/16)
New_entry = 370CED0 (Fa0/16)
Found an entry from same device (Fa0/16)
Cached entries = 2 (Fa0/16)
Entry (0x3792BE0) deleted: 1 entries cached
Cached entries = 1 (Fa0/16)
Zero IDs in 2way conn list (Fa0/16)
Zero IDs in 2way conn list (Fa0/16)
UDLD disabled port, packet received in extended detection (Fa0/16)
%UDLD-4-UDLD_PORT_DISABLED: UDLD disabled interface Fa0/16, unidirectional
link detected
%PM-4-ERR_DISABLE: udld error detected on Fa0/16, putting Fa0/16 in errdisable
state
```

Вы увидите много отладочной информации, но конечным результатом будет то, что порт теперь находится в состоянии err-disable.

```
SwitchB#show udld fastEthernet 0/16
Interface Fa0/16
---
Port enable administrative configuration setting: Enabled / in aggressive
mode
Port enable operational state: Enabled / in aggressive mode
Current bidirectional state: Unidirectional
Current operational state: Disabled port
```

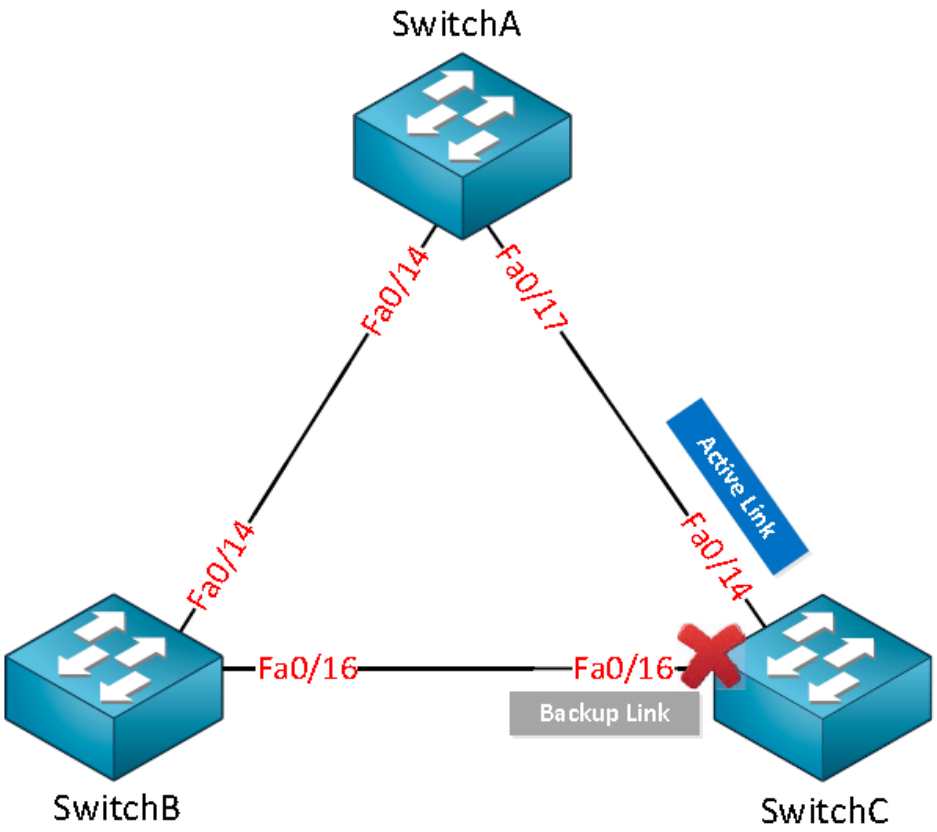
Вы можете проверить это с помощью команды `show udld`.

LoopGuard и UDLD решают одну и ту же проблему: однонаправленные сбои связи.

Они частично пересекаются, но есть ряд различий, вот общий обзор:

	LoopGuard	**UDLD**
Настройки	Глобально/на порту	Глобально (для оптики)/на порту
VLAN?	Да	Нет, на порту
Автосохранение	Да	Да, но вам нужно настроить errdisable timeout.
Защита от сбоев STP из-за однонаправленных связей	Да - нужно включить его на всех корневых и альтернативных портах	Да - нужно включить его на всех интерфейсах.
Защита от сбоев STP из-за сбоев программного обеспечения (нет отправки BPDU)	Да	Нет
Защита от неправильного подключения (коммутационный оптический приемопередающий разъем)	Нет	Да

Есть еще одна последняя тема, которую хотелось бы объяснить, это не протокол связующего дерева, но речь идет о избыточных ссылках, поэтому я оставлю ее здесь. Это называется **FlexLinks**.



Вот в чем дело: при настройке FlexLinks у вас будет активный и резервный интерфейс. Мы настроим это на коммутаторе C:

- `Fa0/14` будет активным интерфейсом.
- `Fa0/16` будет интерфейс резервного копирования (этот блокируется!).

При настройке интерфейсов в качестве FlexLinks они не будут отправлять BPDU. Нет никакого способа обнаружить петли, потому что мы не запускаем на них связующее дерево. Всякий раз, когда наш активный интерфейс выходит из строя, резервный интерфейс заменяет его.

```
SwitchC(config)#interface fa0/14
SwitchC(config-if)#switchport backup interface fa0/16
```

Именно так мы делаем интерфейс `fa0/16` резервной копией интерфейса `fa0/14`.

```
SwitchC#
%SPANTREE-6-PORTDEL_ALL_VLANS: FastEthernet0/14 deleted from all Vlans
%SPANTREE-6-PORTDEL_ALL_VLANS: FastEthernet0/16 deleted from all Vlans
```

Вы можете видеть, что связующее дерево отключается для этих интерфейсов.

```
SwitchC#show interfaces switchport backup
Switch Backup Interface Pairs:
Active Interface Backup Interface State
------------------------------------------------------------------------
FastEthernet0/14 FastEthernet0/16 Active Up/Backup Standby
```

Проверьте нашу конфигурацию с помощью команды `show interfaces switchport backup`. Вот и все, что нужно было сделать. Это интересное решение, потому что нам больше не нужно связующее дерево. Ведь в любой момент времени активен только один интерфейс.

```
SwitchC(config)#interface f0/14
SwitchC(config-if)#shutdown
```

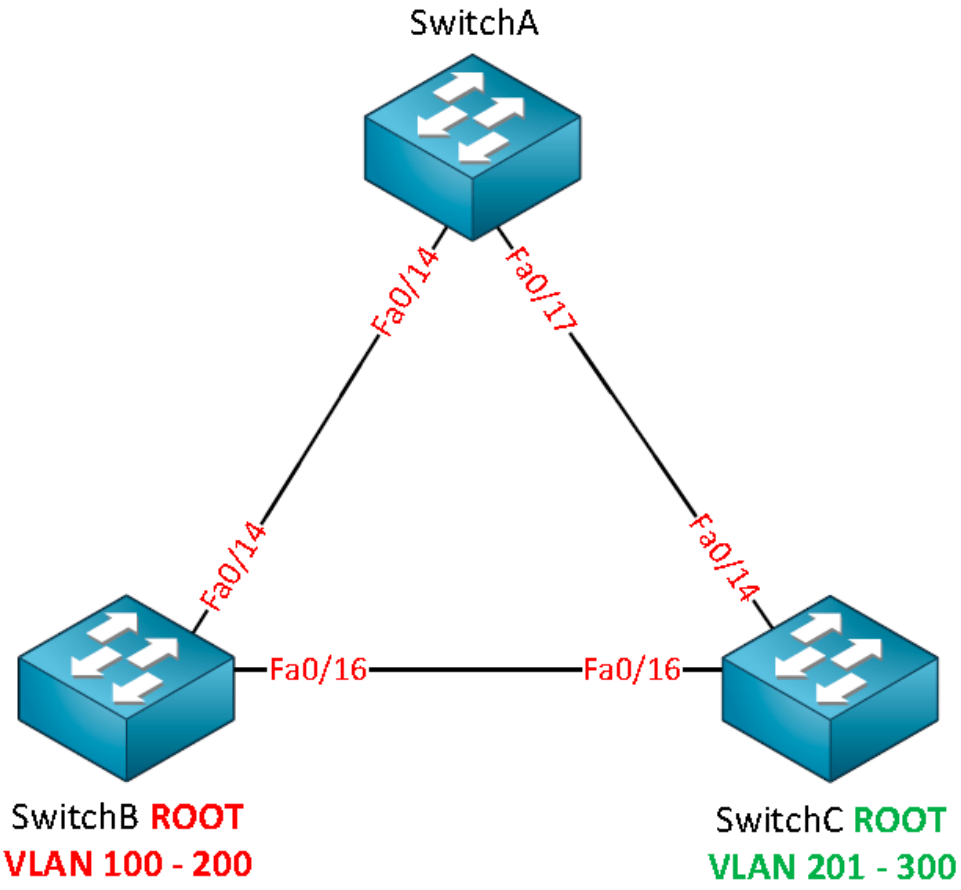
Давайте закроем активный интерфейс.

```
SwitchC#show interfaces switchport backup
Switch Backup Interface Pairs:
Active Interface Backup Interface State
------------------------------------------------------------------------
FastEthernet0/14 FastEthernet0/16 Active Down/Backup Up
```

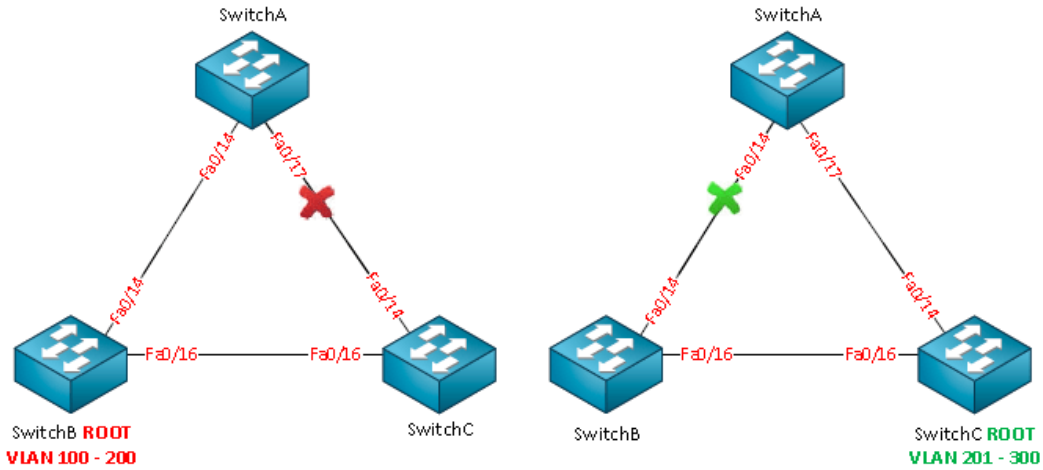
Вы можете видеть, что **fa0/16** стал активным. Вот и все.

4. Протокол MST: Multiple Spanning Tree

MULTIPLE SPANNING TREE



Взгляните на топологию выше. У нас есть три коммутатора и много VLAN. Всего существует **199** VLAN. Если мы запускаем **PVST** или **Rapid PVST**, это означает, что у нас имеется 199 различных вычислений для каждой VLAN. Это требует большой мощности процессора и памяти.



Коммутатор B является корневым мостом для сети от VLAN 100 до VLAN 200. Это, означает, что интерфейс `Fa0/17` коммутатора A будет заблокирован. Мы будем иметь 100 вычислений связующего дерева, но все они выглядят одинаково для этих VLAN.

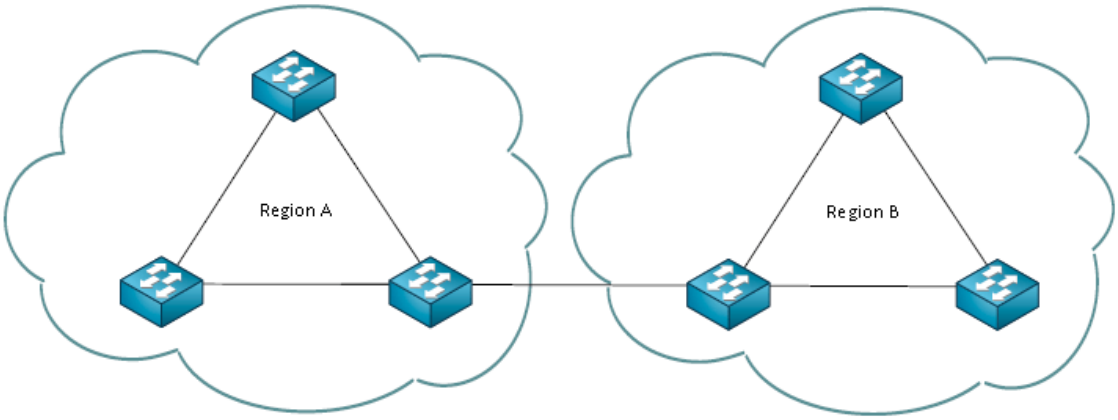
То же самое относится и к VLAN 201 – 300. Коммутатор C является корневым мостом для VLAN от 201 до 300. Интерфейс `Fa0/14` на коммутаторе A, вероятно, будет заблокирован для всех этих VLAN.

Два разных результата, но мы все еще имеем 199 различных вариантов исполнения связующего дерева. Это пустая трата мощности процессора и памяти, верно?

MST (Multiple Spanning Tree) сделает это за нас. Вместо вычисления связующего дерева для каждой VLAN, мы можем использовать **instance** и карту VLAN для каждого instance. Для сети выше мы могли бы сделать что-то вроде этого:

- instance 1: VLAN 100-200
- instance 2: VLAN 201-300

Логично, не так ли? Для всех этих VLAN требуется только два вычисления связующего дерева (instance).

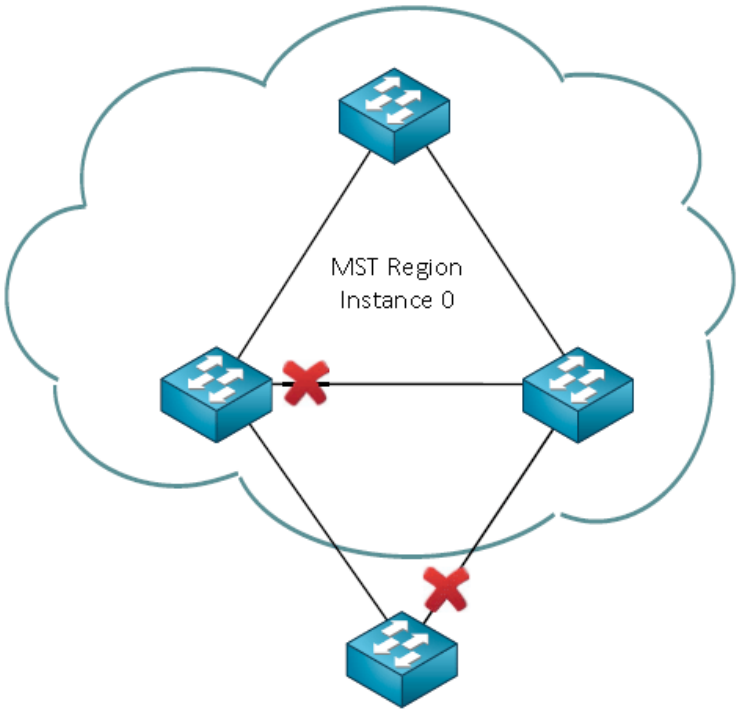


MST работает с концепцией регионов. Коммутаторы, настроенные для использования MST, должны выяснить, работают ли их соседи под управлением MST. Если коммутаторы имеют одинаковые атрибуты, они будут находиться в одном регионе. Это необходимо, чтобы была возможность разделения сети на один или несколько регионов. А вот атрибуты, которые должны соответствовать:

1. MST имя конфигурации
2. MST номер редакции конфигурации
3. MST экземпляр в таблице сопоставления VLAN

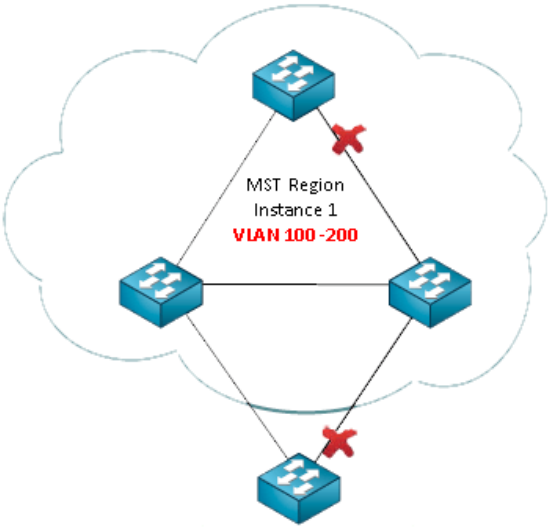
Если коммутаторы имеют одинаковые настроенные атрибуты, они будут находиться в одном регионе. Если атрибуты не совпадают, то коммутатор рассматривается как находящийся на границе области. Он может быть подключен к другому региону MST, но также разговаривать с коммутатором, работающим под управлением другой версии связующего дерева.

Имя конфигурации MST — это то, что вы можете придумать, оно используется для идентификации региона MST. Номер версии конфигурации MST — это также то, что вы можете придумать, и идея этого номера заключается в том, что вы можете изменить номер всякий раз, когда вы меняете свою конфигурацию. VLAN будут сопоставлены с экземпляром с помощью таблицы сопоставления MST instance to VLAN. Это то, что мы должны сделать сами.

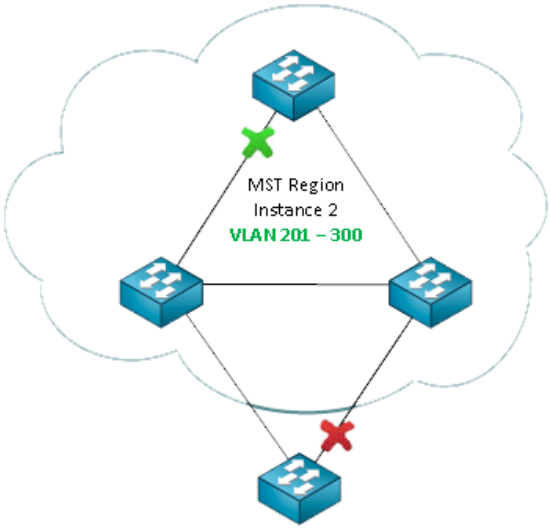


ДРУГИЕ ВЕРСИИ STP

В пределах области MST у нас будет один instance связующего дерева, который создаст свободную от цикла топологию внутри области. При настройке MST всегда существует один instance по умолчанию, используемый для вычисления топологии в пределах региона. Мы называем это **IST** (внутреннее связующее дерево). По умолчанию Cisco будет использовать `instance 0` для запуска IST. На случай, если вам интересно, это rapid spanning tree, которое мы запускаем в пределах MST.

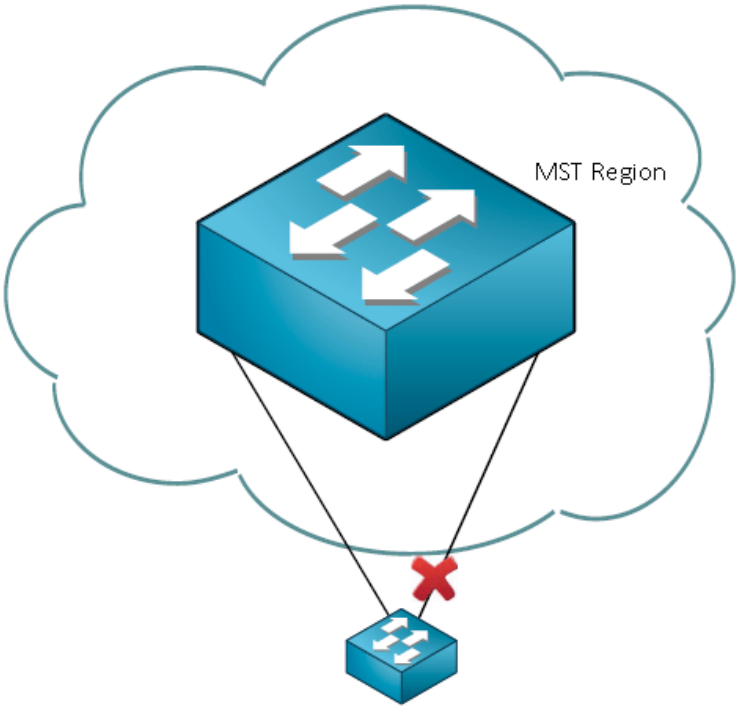


Другая версия STP



Другая версия STP

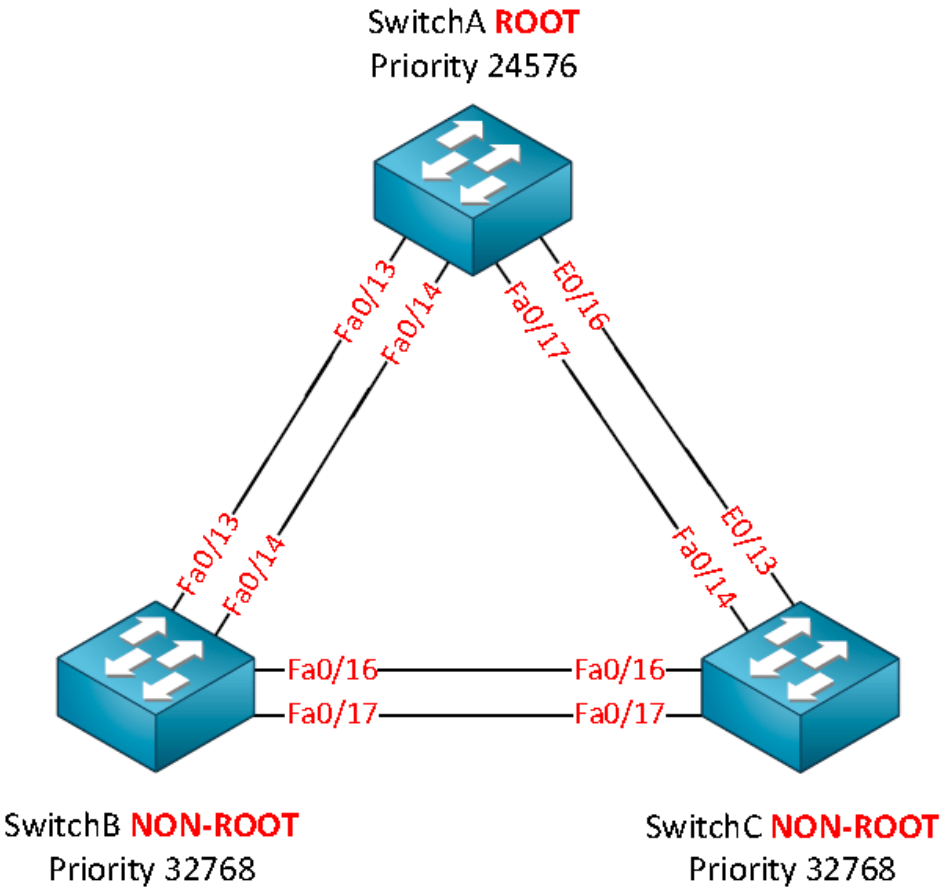
Мы могли бы создать `instance 1` для VLAN 100-200 и `instance 2` для VLAN 201-300. В зависимости от того, какой коммутатор станет корневым мостом для каждого instance, будет заблокирован различный порт.



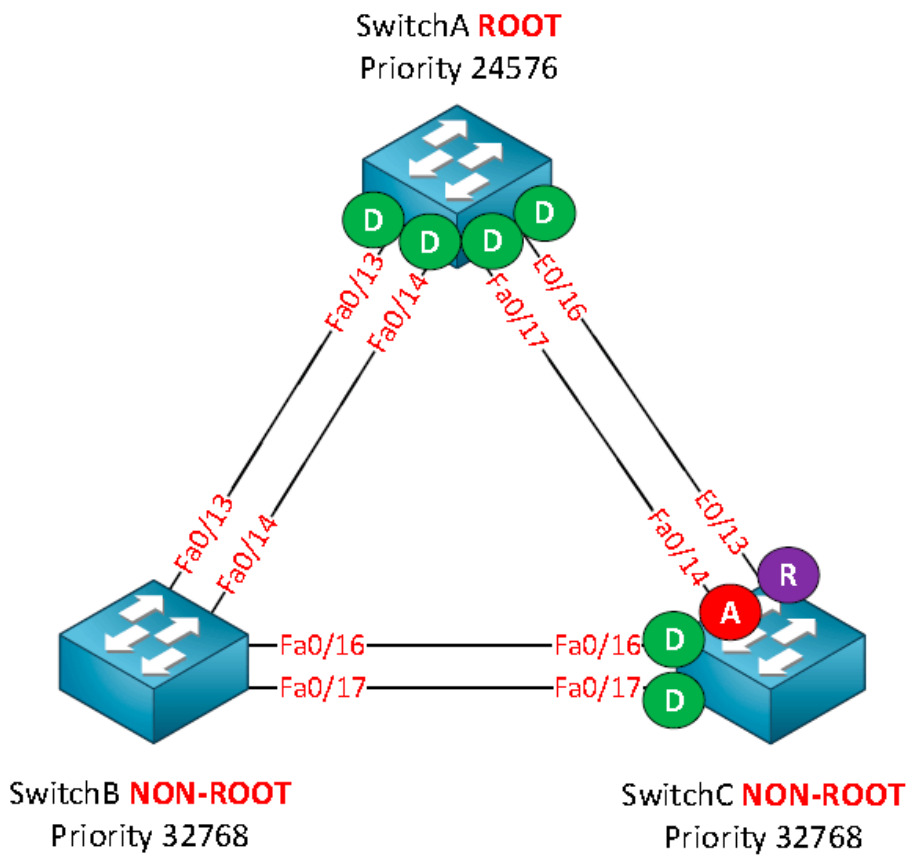
Коммутатор за пределами области MST не видит, как выглядит область MST. Для этого коммутатора все равно, что говорить с одним большим коммутатором или «черным ящиком».

5. Траблшутинг STP (Spanning tree protocol)

CASE #1



На рисунке представлена топология, состоящая из трех коммутаторов, и между коммутаторами у нас есть два канала связи для резервирования. Коммутатор A был выбран в качестве корневого моста для VLAN 1. Когда вы имеете дело со связующим деревом, лучше всего нарисовать небольшую схему сети и записать роли интерфейса для каждого коммутатора (назначенного, не назначенного/альтернативного или заблокированного). Обратите внимание, что одним из каналов связи между коммутатором A и коммутатором C является интерфейс Ethernet (10 Мбит). Все остальные каналы — это **FastEthernet**.



Мы используем команду `show spanning-tree` для проверки ролей интерфейса для коммутатора A и коммутатора C. Вы видите, коммутатор C выбрал свой интерфейс Ethernet 0/13 как корневой порт, а интерфейс FastEthernet 0/14 выбран в качестве альтернативного порта. Это не очень хорошая идея. Это означает, что мы будем отправлять весь трафик вниз по линии 10 Мбит, в то время как 100 Мбит не используется вообще. Когда коммутатор должен выбрать корневой порт он выберет его следующим образом:

1. Выбирается интерфейс, который имеет самую низкую стоимость для корневого моста.
2. Если стоимость равная, выбирается наименьший номер интерфейса. Обычно стоимость интерфейса Ethernet выше, чем Fast Ethernet, поэтому он должен выбрать интерфейс FastEthernet.

Почему коммутатор выбрал интерфейс Ethernet 0/13?

```
SwitchC#show spanning-tree vlan 1
VLAN0001
Spanning tree enabled protocol ieee
Root ID     Priority    24577
            Address     0011.bb0b.3600
            Cost        19
            Port        13 (FastEthernet0/13)
            Hello Time  2 sec Max Age 20 sec Forward Delay 15 sec

Bridge ID   Priority    32769 (priority 32768 sys-id-ext 1)
            Address     000f.34ca.1000
            Hello Time  2 sec Max Age 20 sec Forward Delay 15 sec
            Aging Time 15
Interface           Role Sts Cost         Prio.Nbr Type
------------------- ---- --- ---------    -------- ----------------------------
-----
E0/13               Root FWD 19           128.13   P2p
Fa0/14              Altn BLK 19           128.14   P2p
Fa0/16              Desg FWD 19           128.16   P2p
Fa0/17              Desg FWD 19           128.17   P2p
```

Мы видим, что интерфейс Ethernet 0/13 и FastEthernet0/14 имеют одинаковую стоимость. Затем коммутатор C выберет самый низкий номер интерфейса, который является interface Ethernet 0/13.

```
SwitchC#show run interface fa0/13
Building configuration...
Current configuration : 102 bytes!
Interface Ethernet0/13
      switchport mode dynamic desirable
      spanning-tree cost 19
```

После проверки конфигурации интерфейса, видно, что кто-то изменил стоимость интерфейса на 19 (по умолчанию для интерфейсов FastEthernet).

```
SwitchC(config)#interface Ethernet 0/13
SwitchC(config-if)#no spanning-tree cost 19
```

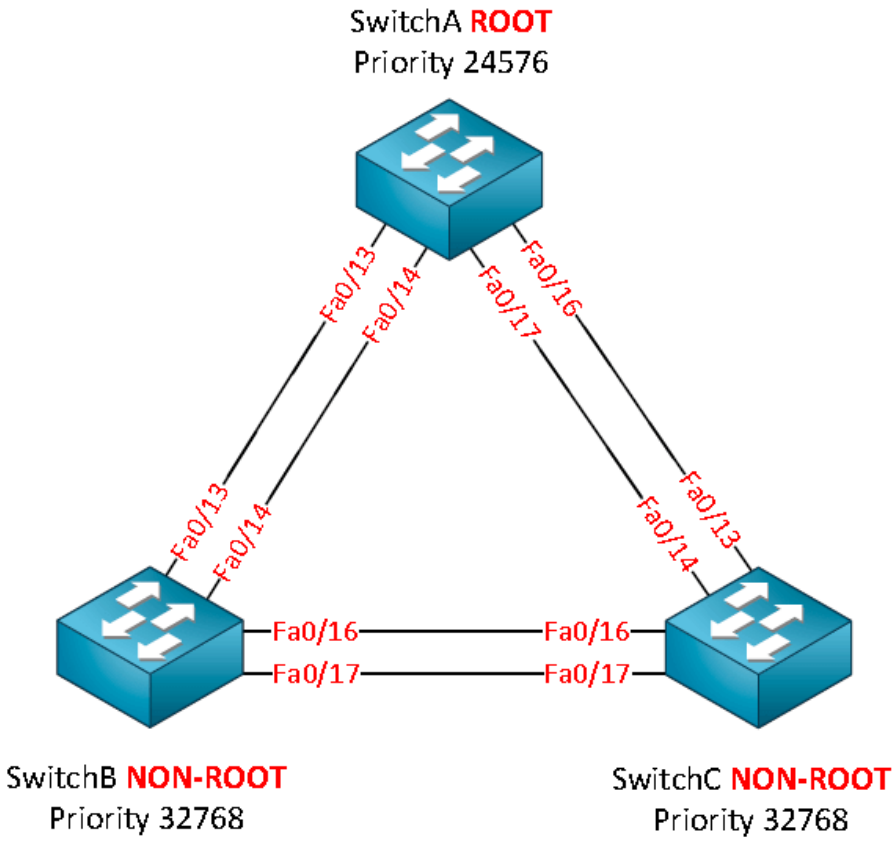
Уберем настройки команды cost.

```
SwitchC#show spanning-tree vlan 1
VLAN0001
      Spanning tree enabled protocol ieee
      Root ID     Priority    24577
                  Address     0011.bb0b.3600
                  Cost        19
                  Port        14 (FastEthernet0/14)
                  Hello Time  2 sec Max Age 20 sec Forward Delay 15 sec
      Bridge ID   Priority    32769 (priority 32768 sys-id-ext 1)
                  Address     000f.34ca.1000
                  Hello Time  2 sec Max Age 20 sec Forward Delay 15 sec
                  Aging Time 15
Interface         Role Sts Cost          Prio.Nbr Type
----------------- ---- --- --------- -------- --------------------------------
E0/13             Altn BLK 100           128.13   P2p
Fa0/14            Root FWD 19            128.14   P2p
Fa0/16            Desg FWD 19            128.16   P2p
Fa0/17            Desg FWD 19            128.17   P2p
```

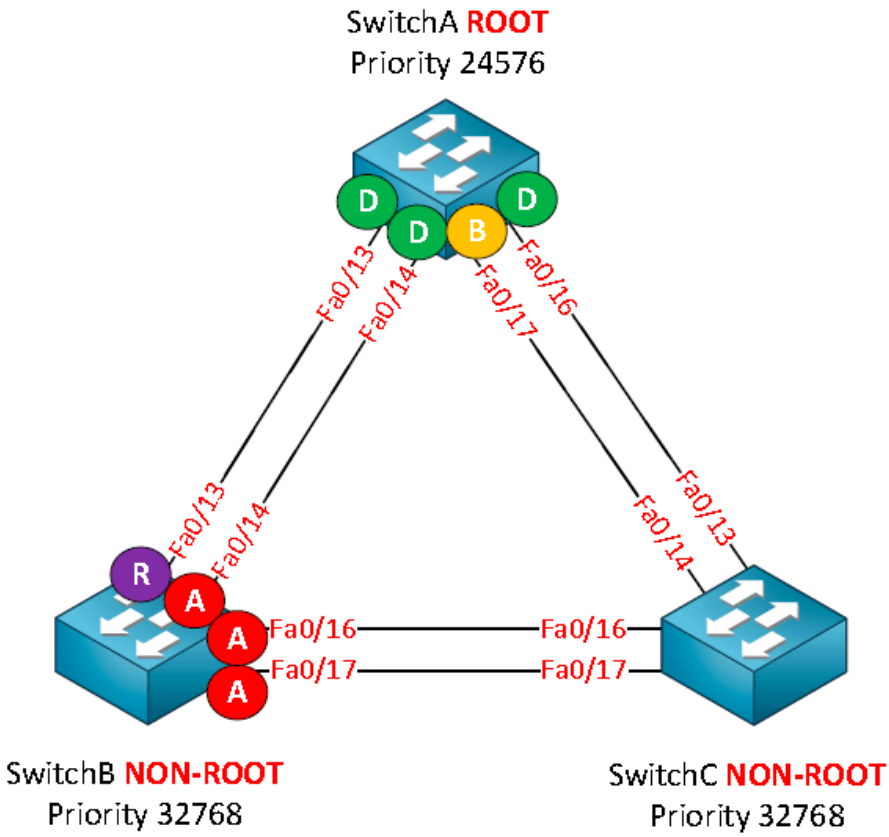
После того, как мы убрали настройки команды cost, видно, что состояние порта изменилось. FastEthernet 0/14 теперь является корневым портом, а стоимость интерфейса Ethernet 0/13 равна 100 (это значение по умолчанию для интерфейсов Ethernet). Задача решена!

Извлеченный урок: убедитесь, что интерфейс, которым вы хотите сделать в качестве корневого порта, имеет наименьшую стоимость пути.

CASE #2



Итак, новый сценарий. Все интерфейсы равны (FastEthernet). Коммутатор A является корневым мостом для VLAN 10, и после проверки ролей интерфейса мы находим следующее:



Хм, интересно... Коммутатор A является корневым мостом, а FastEthernet 0/17 был выбран в качестве резервного порта. Это то, что вы видите каждый день. Коммутатор B выбрал корневой порт, а все остальные интерфейсы являются альтернативными портами. Мы ничего не видим на коммутаторе C.

```
SwitchA#show spanning-tree vlan 10

      VLAN0010
      Spanning tree enabled protocol ieee

SwitchB#show spanning-tree vlan 10

      VLAN0010
      Spanning tree enabled protocol ieee

SwitchC#show spanning-tree vlan 10

      Spanning tree instance(s) for vlan 10 does not exist.
```

Мы видим, что Коммутатор A и Коммутатор B используют связующее дерево для VLAN 10. Коммутатор C, однако, не использует связующее дерево для VLAN 10. В чем может быть проблема?

```
SwitchC#show interfaces fa0/13 | include line protocol
      FastEthernet0/13 is up, line protocol is up (connected)

SwitchC#show interfaces fa0/14 | include line protocol
      FastEthernet0/14 is up, line protocol is up (connected)

SwitchC#show interfaces fa0/16 | include line protocol
      FastEthernet0/16 is up, line protocol is up (connected)

SwitchC#show interfaces fa0/17 | include line protocol
      FastEthernet0/16 is up, line protocol is up (connected)
```

Конечно, неплохо проверить, работают ли интерфейсы на коммутаторе C или нет (но, конечно, это то, что вы уже изучили и сделали в первой статье).

```
SwitchC#show interfaces trunk
Port        Mode         Encapsulation          Status           Native vlan
Fa0/13      desirable    n-isl                  trunking         1
Fa0/14      desirable    n-isl                  trunking         1
Fa0/16      desirable    n-isl                  trunking         1
Fa0/17      desirable    n-isl                  trunking         1
Port        Vlans allowed on trunk
Fa0/13      1-4094
Fa0/14      1-4094
Fa0/16      1-4094
Fa0/17      1-4094
Port        Vlans allowed and active in management domain
Fa0/13      1,10
Fa0/14      1,10
Fa0/16      1,10
Fa0/17      1,10
```

Интерфейсы выглядят хорошо. VLAN 10 активна на всех интерфейсах коммутатора C. Это означает, что остовное дерево должно быть активным для VLAN 10.

```
SwitchC#show spanning-tree vlan 10
Spanning tree instance(s) for vlan 10 does not exist.
```

Давайте еще раз посмотрим на это сообщение. Это говорит о том, что остовное дерево для VLAN 10 не существует. Есть две причины, по которым можно увидеть это сообщение:

- Для VLAN 10 нет активных интерфейсов.
- Spanning-дерево было отключено для VLAN 10.

Мы подтвердили, что VLAN 10 активна на всех интерфейсах коммутатора C, поэтому, может быть, связующее дерево было отключено глобально? `SwitchC(config)#spanning-tree vlan 10`

```
SwitchC#show spanning-tree vlan 10
VLAN0010
      Spanning tree enabled protocol ieee
      Root ID           Priority          24586
                        Address           0011.bb0b.3600
                        Cost              19
                        Port 13 (FastEthernet0/13)
                        Hello Time 2 sec Max Age 20 sec Forward Delay 15 sec
      Bridge ID         Priority          32778 (priority 32768 sys-id-ext 10)
                        Address           000f.34ca.1000
                        Hello Time 2 sec Max Age 20 sec Forward Delay 15 sec
                        Aging Time 300
Interface         Role Sts Cost          Prio.Nbr    Type
------------------- ---- --- --------- -------- --------------------------

Fa0/13            Root FWD 19            128.13      P2p
Fa0/14            Altn BLK 19            128.14      P2p
Fa0/16            Desg FWD 19            128.16      P2p
Fa0/17            Desg FWD 19            128.17      P2p
```

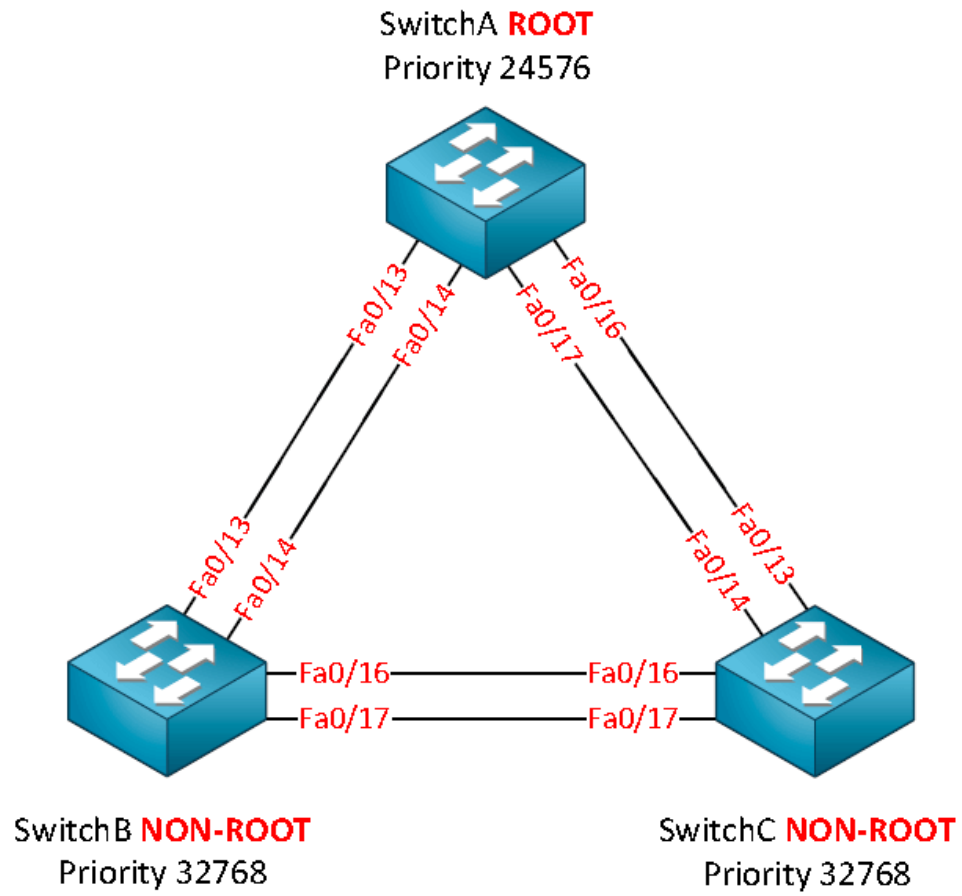
Вот так выглядит лучше! Теперь связующее дерево включено для VLAN 10 и работает ... проблема решена! Эта проблема может показаться немного странной, но она появляется ее время от времени в реальном мире. Сценарий, который мы рассмотрели раньше, - это событие из реальной жизни, где клиент, которому поставщик беспроводной связи отключил остовное дерево для интерфейсов, которые подключаются к точке беспроводного доступа. Ниже то, что клиент ввел на коммутаторе:

```
SwitchC(config)#interface fa0/1
SwitchC(config-if)#no spanning-tree vlan 10
SwitchC(config)#
```

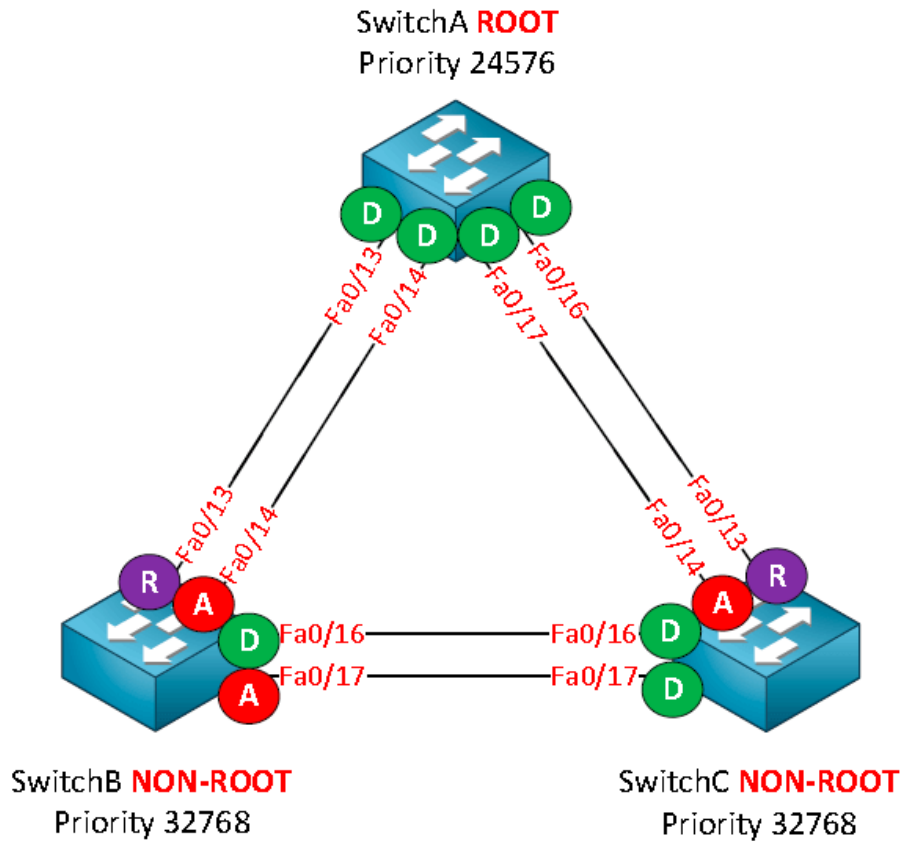
В интерфейсе они набрали no `spanning-tree vlan 10`, но как вы видите, что они оказались в режиме глобальной конфигурации. Нет команды для отключения остовного дерева на интерфейсе, подобного этой, поэтому коммутатор думает, что вы ввели глобальную команду для отключения остовного дерева. Коммутатор принимает команду отключения остовного дерева для **VLAN 10** и возвращает вас в режим глобальной конфигурации... проблема решена!

Извлеченный урок: проверьте, включено ли связующее дерево.

CASE #3



Давайте продолжим по другому сценарию! Та же топология... наш клиент жалуется на плохую работу. Начнем с проверки ролей интерфейсов:



Посмотрите на картинку выше. Видите ли вы, что интерфейс FastEthernet 0/16 на коммутаторе B и коммутаторе C обозначены? На Коммутаторе A все интерфейсы обозначены. Как вы думаете, что произойдет, когда один из наших коммутаторов переадресует трансляцию или должен передать кадр? Правильно! У нас будет цикл ... Обычно в этой топологии интерфейсы FastEthernet 0/16 и 0/17 на коммутаторе C должны быть альтернативными портами, поскольку коммутатор C имеет худший ID моста. Так как они оба обозначены, мы предполагаем, что Коммутатор C не получает BPDU на этих интерфейсах.

Так почему же остовное дерево провалилось здесь? Здесь важно помнить, что связующему дереву требуются блоки BPDU, передаваемые между коммутаторами для создания топологии без петель. BPDU могут быть отфильтрованы из-за MAC access-lists, VLAN access-maps или из-за spanning-tree toolkit?

```
SwitchA#show vlan access-map
SwitchB#show vlan access-map
SwitchC#show vlan access-map
```

Ни на одном из коммутаторов нет VLAN **access maps**.

```
SwitchA#show access-lists
SwitchB#show access-lists
SwitchC#show access-lists
```

Нет списков доступа...

```
SwitchA#show port-security
Secure Port MaxSecureAddr CurrentAddr SecurityViolation Security Action
(Count) (Count) (Count)
---------------------------------------------------------------------------
---------------------------------------------------------------------------
Total Addresses in System (excluding one mac per port) : 0
Max Addresses limit in System (excluding one mac per port) : 6144


SwitchB#show port-security
Secure Port MaxSecureAddr CurrentAddr SecurityViolation Security Action
(Count) (Count) (Count)
---------------------------------------------------------------------------
---------------------------------------------------------------------------
Total Addresses in System (excluding one mac per port) : 0
Max Addresses limit in System (excluding one mac per port) : 6144


SwitchC#show port-security
Secure Port MaxSecureAddr CurrentAddr SecurityViolation Security Action
(Count) (Count) (Count)
---------------------------------------------------------------------------
---------------------------------------------------------------------------
Total Addresses in System (excluding one mac per port) : 0
Max Addresses limit in System (excluding one mac per port) : 6144
```

Нет **port security**... как насчет команд, связанных с остовным деревом?

```
SwitchB#show spanning-tree interface fa0/16 detail | include filter
      Bpdu filter is enabled

SwitchB#show spanning-tree interface fa0/17 detail | include filter
      Bpdu filter is enabled
```

Вот что-то есть!Фильтр **BPDU** был включен на интерфейсах FastEthernet 0/16 и 0/17 коммутатора B. Из-за этого коммутатор C не получает BPDU от коммутатора B.

```
SwitchB(config)#interface fa0/16
SwitchB(config-if)#no spanning-tree bpdufilter enable
SwitchB(config-if)#interface fa0/17
SwitchB(config-if)#no spanning-tree bpdufilter enable
```

Удалим настройки фильтра BPDU.

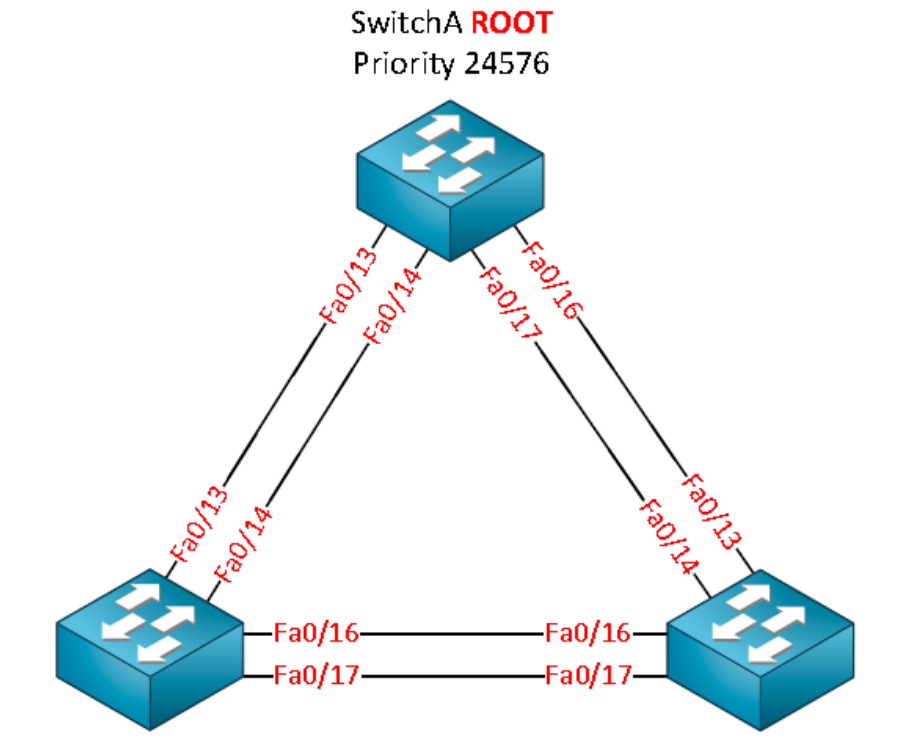
```
SwitchC#show spanning-tree vlan 10 | begin Interface
Interface         Role Sts          Cost Prio.Nbr     Type
------------------- ---- --- --------- -------- --------------------------

Fa0/13            Root FWD 19       128.13           P2p
Fa0/14            Altn BLK 19       128.14           P2p
Fa0/16            Altn BLK 19       128.16           P2p
Fa0/17            Altn BLK 19       128.17           P2p
```

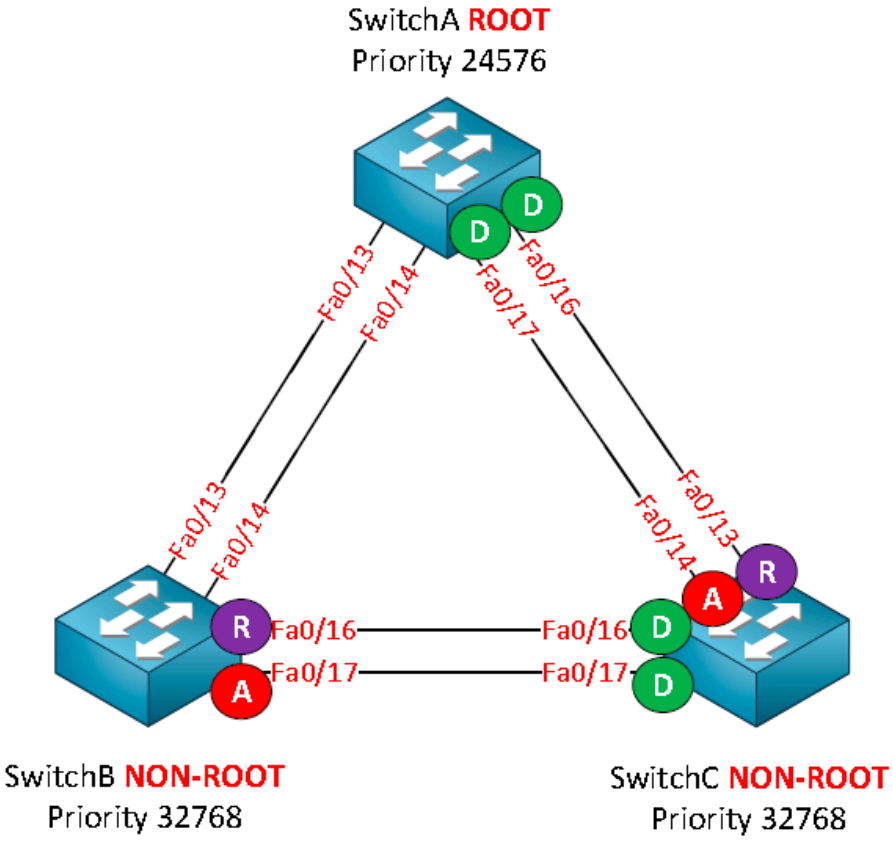
Теперь вы видите, что FastEthernet 0/16 и 0/17 являются альтернативными портами и блокируют трафик. Наша топология теперь без петель... проблема решена!

Извлеченный урок: убедитесь, что блоки BPDU не заблокированы и не отфильтрованы между коммутаторами.

CASE #4



Новая топология. Коммутатор A был выбран в качестве корневого моста для VLAN 10. Все интерфейсы являются FastEthernet каналами.



После использования команды `show spanning-tree vlan 10` вот, что мы видим. Все интерфейсы одинаковы, но по какой-то причине коммутатор B решил выбрать FastEthernet 0/16 в качестве корневого порта. Разве вы не согласны с тем, что FastEthernet 0/13 должен быть корневым портом? Стоимость доступа к корневому мосту ниже, чем у FastEthernet 0/16.

```
SwitchB#show spanning-tree interface fa0/13

Vlan             Role Sts Cost      Prio.Nbr Type
---------------- ---- --- --------- -------- --------------------------------

VLAN0001         Root FWD 19        128.15   P2p
SwitchB#show spanning-tree interface fa0/14

Vlan             Role Sts Cost      Prio.Nbr Type
---------------- ---- --- --------- -------- --------------------------------

VLAN0001         Altn BLK 19        128.16   P2p
```

Используем команду `show spanning-tree interface`, чтобы проверить информацию о spanning-tree для каждого интерфейса. Как вы можете видеть, существует только связующее дерево для VLAN 1, активное на интерфейсе FastEthernet 0/13 и 0/14.

Есть несколько вещей, которые мы могли бы проверить, чтобы увидеть, что происходит:

```
SwitchB#show spanning-tree vlan 10
VLAN0010
      Spanning tree enabled protocol ieee
      Root ID     Priority    24586
                  Address     0011.bb0b.3600
                  Cost        38
                  Port 18 (FastEthernet0/16)
                  Hello Time 2 sec Max Age 20 sec Forward Delay 15 sec

      Bridge ID   Priority    32778 (priority 32768 sys-id-ext 10)
                  Address     0019.569d.5700
                  Hello Time 2 sec Max Age 20 sec Forward Delay 15 sec
                  Aging Time 300

Interface        Role Sts Cost      Prio.Nbr Type
---------------- ---- --- --------- -------- --------------------------------

Fa0/16           Root FWD 19        128.18   P2p
Fa0/17           Altn BLK 19        128.19   P2p
```

Во-первых, всегда полезно проверить, активно ли связующее дерево для определенной VLAN. Можно отключить spanning-tree с помощью команды no spanning-tree vlan X. В этом сценарии связующее дерево активно для VLAN 10, потому что мы можем видеть на FastEthernet 0/16 и 0/17.

```
SwitchB#show interfaces fa0/13 switchport
Name: Fa0/13
Switchport: Enabled
Administrative Mode: dynamic auto
Operational Mode: static access
Administrative Trunking Encapsulation: negotiate
Operational Trunking Encapsulation: native
Negotiation of Trunking: On
Access Mode VLAN: 1 (default)


SwitchB#show interfaces fa0/14 switchport
Name: Fa0/14
Switchport: Enabled
Administrative Mode: dynamic auto
Operational Mode: static access
Administrative Trunking Encapsulation: negotiate
Operational Trunking Encapsulation: native
Negotiation of Trunking: On
Access Mode VLAN: 1 (default)
```

Мы знаем, что остовное дерево активно глобально для VLAN 10, но это не значит, что оно активно на всех интерфейсах. Мы можем использовать команду show interfaces switchport, чтобы проверить, работает ли VLAN 10 на интерфейсе FastEthernet 0/13 и 0/14. Это отобразит нам некоторую интересную информацию. Вы видите, что эти интерфейсы оказались в режиме **доступа**, и они находятся в VLAN 1.

```
SwitchB(config)#interface fa0/13
SwitchB(config-if)#switchport mode trunk

SwitchB(config-if)#interface fa0/14
SwitchB(config-if)#switchport mode trunk
```

Давайте изменим режим интерфейсов на **магистральный**, чтобы трафик VLAN 10 мог проходить через эти интерфейсы.

```
SwitchB#show spanning-tree vlan 10
VLAN0010
      Spanning tree enabled protocol ieee
      Root ID     Priority          24586
                  Address           0011.bb0b.3600
                  Cost              19
                  Port              15 (FastEthernet0/13)
                  Hello Time 2 sec Max Age 20 sec Forward Delay 15 sec

      Bridge ID   Priority          32778 (priority 32768 sys-id-ext 10)
                  Address 0019.569d.5700
                  Hello Time 2 sec Max Age 20 sec Forward Delay 15 sec
                  Aging Time 300
Interface         Role Sts Cost           Prio.Nbr    Type
------------------- ---- --- --------- -------- --------------------------

Fa0/13            Root FWD 19             128.15      P2p
Fa0/14            Altn BLK 19             128.16      P2p
Fa0/16            Altn BLK 19             128.18      P2p
Fa0/17            Altn BLK 19             128.19      P2p
```

Ну вот, теперь все намного лучше выглядит. Трафик VLAN 10 теперь передается по интерфейсу FastEthernet 0/13 и 0/14, и вы видите, что интерфейс FastEthernet 0/13 теперь выбран в качестве корневого порта. Задача решена!

Извлеченный урок: убедитесь, что VLAN активна на интерфейсе, прежде чем рассматривать проблемы связующего дерева.