Содержание

[Введение 5](#_Toc451953211)

[Анализ проблемы 7](#_Toc451953212)

[Обзор предметной области 7](#_Toc451953213)

[VMWare 8](#_Toc451953214)

[Терминологические особенности предметной области и типы виртуальных сетей 8](#_Toc451953215)

[Open vSwitch 12](#_Toc451953216)

[Проблема и её актуальность 14](#_Toc451953217)

[Постановка задачи и первичный анализ проблемы 15](#_Toc451953218)

[Проектирование решения 18](#_Toc451953219)

[Проектирование и сборка топологий 23](#_Toc451953220)

[Реализация автоматического конфигуратора 38](#_Toc451953221)

[Первая топология 39](#_Toc451953222)

# Введение

За последние несколько лет, дата-центры плотно вошли в жизнь многих корпораций. Практически никому не нужно объяснять, какую роль они играют в современном мире. Центры обработки данных (ЦОД) выполняют массу задач, существенно облегчая организацию хранения, обработки, и распространения информации корпоративных клиентов. Эффективное использование технических средств за счёт объединения вычислительных ресурсов и средств хранения данных в ЦОД сокращает общую стоимость IT-инфраструктуры предприятия. Такое консолидирование способствует равномерному распределению нагрузок и сокращению затрат на администрирование.

Технология виртуализации решила массу проблем и стала существенным шагом вперёд. Аппаратная независимость, инкапсуляция необходимых сервисов, приложений и рабочих станций в виртуальные машины. Распределение виртуальных машин по хостам, экономия физического оборудования за счёт совместного использования этого оборудования виртуальными машинами. Миграция виртуальных машин с одного хоста на другой с нулевыми простоями. По мнению специалистов корпорации VMware, виртуализация ЦОД уменьшает затраты на 40%, за счёт:

* hardware
* электроэнергии
* охлаждения
* места в дата-центре[[1]](#footnote-1)

Современные ЦОД не стоят на месте, и основным трендом развития этого сегмента технологий являются не просто виртуализированные дата-центры, а, так называемые гибридные облака. Гибридное облако включает в себя дополнительно виртуализацию сетевых технологий, поверх физических, что делает такие решения максимально гибкими, легко управляемыми и экономными для бизнес задач.

В контексте подобного управления сетями и постепенного снижения затрат на поддержание одной виртуальной машины, возникает масса задач, связанных с управлением сетью внутри ЦОД.

# Анализ проблемы

## Обзор предметной области

Так как виртуальные машины внутри ЦОД изолированы друг от друга, а при наличии большого парка виртуальных машин, зачастую, контроль физического сетевого адаптера нужен далеко не каждой виртуальной машине, то управление полностью виртуальной сетью является растущей областью задач.

Как известно, существует несколько производителей программного обеспечения для ЦОД. Наиболее известным из них является компания VMWare, предоставляющая множество технологических решений в области виртуализации, оркестрации и администрирования ЦОД. Помимо коммерческих продуктов, необходимо так же упомянуть open source решение – OpenStack. Все они, так или иначе, предоставляют средства управления виртуальной сетью внутри созданного на их основе ЦОД.

В данной выпускной квалификационной работе использована преимущественно терминология, специфичная продуктам VMWare, так как оркестратором ЦОД, внутри которого ведётся разработка и тестирование, является продукт ESX от данного поставщика.

Виртуальные среды распространения сетевого траффика внутри ЦОД, чаще всего, представляют собой виртуальные switch-устройства, предоставляющие настройки различных сетевых политик для порт-групп. Масса преимуществ кроется в серверной виртуализацией сети. С появлением таких технологий компании могут строго разделять среды тестирования, разработки и размещения. Ранее создание и поддержка таких сред для компании была весьма дорогостоящим удовольствием, теперь же один специалист может легко менять, восстанавливать и разворачивать необходимые среды внутри дата-центра.

### VMWare

С момента своего появления и по сегодняшний день, компания VMWare занимает позицию крупнейшего игрока на рынке виртуализации. Основанная в 1998 году, эта компания первая успешно решила проблемы виртуализации x86 архитектуры, разработав технологию для выполнения 17 проблемных инструкций, ранее мешавших развитию виртуализации.

В настоящий момент компания предлагает множество программных решений для различных видов виртуализации. Наиболее интересны для целей данной выпускной квалификационной работы продукты ESX и ESXi. Это аппаратные гипервизоры архитектур Intel x86, также совместимые с x64. ESXi – это более компактная версия ESX, распространяемая бесплатно и не содержащая в себе сервисной консоли. Как аппаратные гиппервизоры, ESX и ESXi ставятся непосредственно на физическое оборудование и содержат в себе собственное ядро.

Основным инструментом администратора при работе с ESX-кластером становится vSphere client, доступный как в десктоп версии для платформы Windows, так и в web-варианте. С помощью этого инструмента администратор решает большинство задач работы с кластером: развёртывание машин, диагностика производительности, настройка сети, распределение ресурсов, администрирование ролей, резервное копирование и многое другое.

### Терминологические особенности предметной области и типы виртуальных сетей

Существует несколько различных парадигм виртуализации: полная виртуализация, паравиртуализация и виртуализация на уровне операционной системы. Для целей данной работы наиболее интересна полная виртуализация, так же называемая аппаратной, а иногда, на просторах сети, серверной, что не совсем корректно. Сам сервер можно виртуализировать по-разному, но поскольку полная виртуализация наиболее распространена, то название «серверная виртуализация» следует толковать в зависимости от контекста статьи.

Аппаратная виртуализация является наиболее распространённым типом. При таком подходе операционная система виртуальной машины не знает о том, что она выполняется внутри этой самой виртуальной машины. Тем самым никакого изменения в самих операционных системах производить не нужно, они могут использоваться как есть. Гипервизор здесь является единственной программой, исполняемой в привилегированном режиме непосредственно на физическом оборудовании. Такой тип виртуализации можно изобразить следующей схемой (см. рисунок 1).

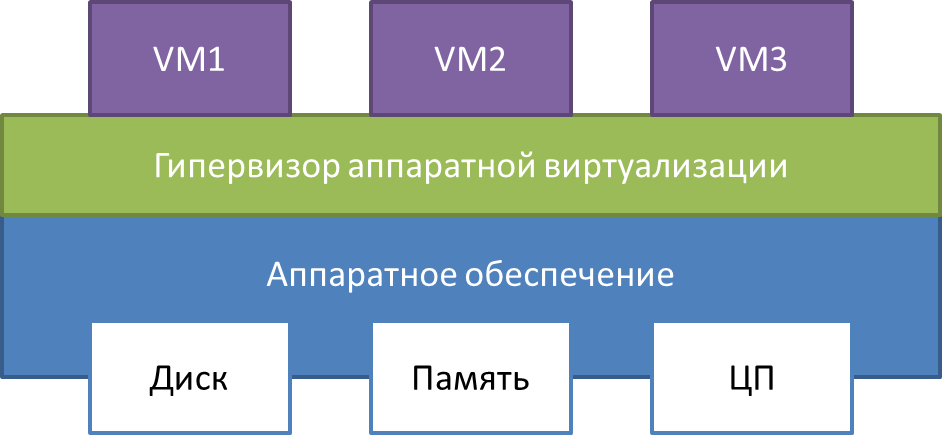


Рисунок 1 – Схема устройства аппаратной виртуализации

Поскольку между физическим оборудованием и виртуальной машиной существует некая прослойка гипервизора, обеспечивающая абстракцию между физическим и виртуальным оборудованием, то сами устройства, которые «видит» виртуальная машина так же являются абстракцией. Это послужило причиной введения несколько иных названий для этих устройств, во избежание путаницы с физическими и более чёткого понимания их места в системе.

Виртуальные сетевые устройства, которые виртуальные машины видят как сетевой адаптер, принято называть Network Interface Controller (NIC). Так же, общепонятным считается сокращение vSwitch, говорящее о том, что экземпляр сетевого switch-устройства является виртуальным. Однако, виртуальные сетевые switch-устройства, соединённые между собой не образуют vLAN. Это сокращение используется для наименования сетевой технологии L2-тоннелей, а так же для обозначения тегированного этой технологией траффика. Так же, поскольку аппаратная виртуализация от различных производителей гипервизоров зачастую предоставляет технологию динамической миграции (live migration) виртуальных машин между физическими хостами, то существует такая разновидность устройств, как distributed virtual switch, сокращённо DvSwitch. Основное его отличие от обычного виртуального switch-устройства в том, что он может находиться внутри не только лишь одного физического хоста, а присутствовать сразу в нескольких, тем самым упрощая виртуальную сетевую конфигурацию.

Виртуальные switch-устройства содержат в себе настройки порт-групп, позволяющие очень гибко и быстро конфигурировать различные топологии.

Порт-группы – это

Также в них зачастую присутствуют и другие опции конфигурирования, присущие основному функционалу физических аналогов этих устройств: тегированный траффик, различные виды сетевых политик, ограничения канала входного и выходного размера канала, failover функционал, мониторинг, аллокация ресурсов и т.д.

Ниже представлена диаграмма внешней (external) виртуальной сети. В данной классификации, предложенной Джоном Говардом (John Howard), Senior Software Manager в команде Hyper-V компании Microsoft[[2]](#footnote-2), такими сетями называются сети, позволяющие коммуницировать:

* Виртуальной машине с виртуальной машиной на одном физическом сервере
* Виртуальной машине с родительским разделом и наоборот
* Виртуальной машине с внешними серверами и наоборот
* Родительскому разделу с внешним сервером

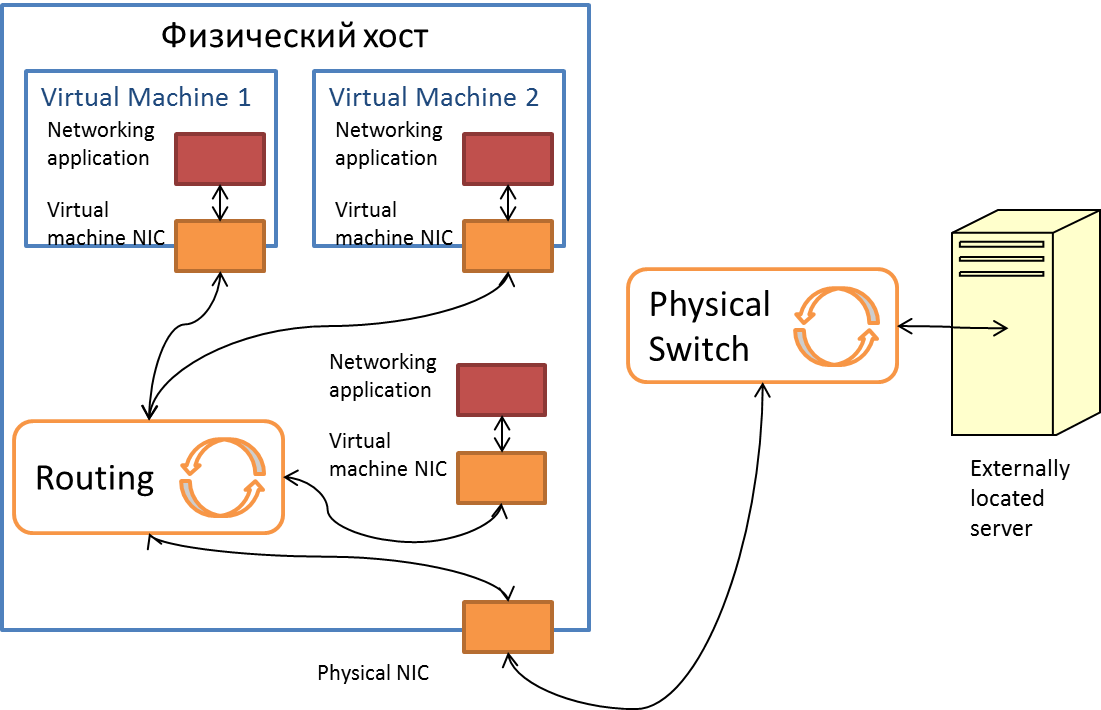


Рисунок 2 – Внешняя виртуальная сеть

Выделяют также и более урезанные типы виртуальных сетей. Частные (private) и выделенные (dedicated) сети в целом отличаются друг от друга отсутствием подключения к внешним серверам, то есть не имеют никаких коммуникаций с «внешним миром». На рисунке 3 отображены оба этих вида сети, а граница между ними проведена пунктирной линией.

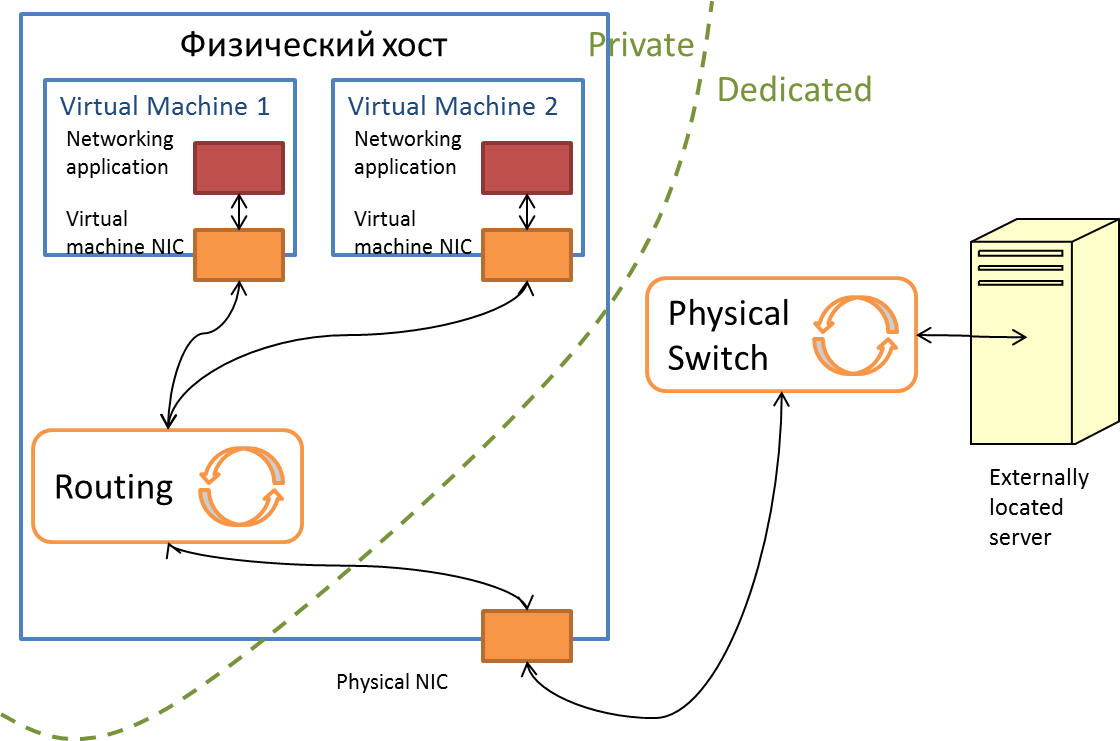


Рисунок 3 – Private and dedicated networks

На рисунке 3 эти два типа объединены в один, поскольку абстракция полной виртуализации нескольких физических серверов зачастую стирает границы между физическими хостами, объединяя их в один большой кластер. При таком состоянии дел, зачастую, администратору не нужно задумываться над тем, на каком хосте «живут» его виртуальные машины – dynamic resource scheduler [[3]](#footnote-3)(DRS) и DvSwitch позволяют забыть об этом на таком уровне абстракции. А потому с такой точки зрения приватные и выделенные сети теряют различия.

### Open vSwitch

Современное сетевое оборудование активно использует распределенные протоколы и алгоритмы. Однако, разработка этих алгоритмов крайне длительна и сложна. У исследователей компьютерных сетей возникла необходимость упростить процесс разработки новых сетевых технологий, работая при этом в реальном окружении, в уже развернутых компьютерных сетях. Для этого необходимо добавить возможность работы с таблицей переключения (packet switching/forwarding table) (неважно L2/L3) в существующее оборудование, чтобы важные узлы сети работали, как и сконфигурировал администратор, а экспериментальные использовали полный контроль над switching table. Контроль над этой таблицей переключения осуществляется с рабочей станции исследователя извне. Обычно этот контроль осуществляется программой, которая может принимать решения по значительно большему количеству условий. Программа (контроллер) может работать с несколькими сетевыми устройствами одновременно, за счет этого появляется возможность реализовывать локальные алгоритмы маршрутизации. Протокол, который лежит в основе этой коммуникации, называется OpenFlow, а сетевые устройства, поддерживающие этот протокол, называются OpenFlow-enabled.

Программной имплементацией OpenFlow является Open vSwitch. Он превращает обычную рабочую станцию в программируемый сетевой свитч L2/L3. При этом на одной рабочей станции может быть несколько логически независимых свитчей. При отсутствии реального сетевого оборудования, на основе нескольких рабочих станций появляется возможность разработать новый протокол, или «политику» маршрутизации. OpenFlow отвечает только за правила переключения пакетов, то есть является протоколом контролирующим переключение. Для конфигурации Open vSwitch используется управляющий протокол OVSDB или OF-CONFIG. Они позволяют создать виртуальные сущности, аналогичные тем которые присутствуют в реальных свитчах, например, добавить порт, агрегировать несколько портов, создать интерфейс и т.д.

Добавь предысторию про Ethane.

Добавь про Google’s move to SDN и SDN в целом.

Добавить про возможности

Добавить про архитектуру

<http://web.eecs.umich.edu/~sugih/courses/eecs589/f13/16-Ethane.pdf>

<http://jedelman.com/home/ethane-changed-everything-devops-for-networking-could-be-next/>

<http://queue.acm.org/detail.cfm?id=2856460>

<http://openvswitch.org/slides/ppf.pdf>

<http://benpfaff.org/papers/ext-networking.pdf>

Добавить про Open Flow

Добавить про VxLAN

## Проблема и её актуальность

Основной проблемой, рассматриваемой в данной выпускной квалификационной работе, является объединение двух локальных сетей. Вне зависимости от наличия виртуальных сетей, большого количества хостов и других сервисов в сети, с момента появления сетевых технологий этот вопрос вставал всё чаще и чаще. Как показывает практика, большинство системных администраторов, так или иначе, сталкиваются с этим вопросом. Развитие технологий, виртуальных сетей, больших виртуальных сетей выделенного типа (dedicated networks), а также снижение затрат на поддержку виртуальных машин стали основными предпосылками для написания данной работы.

Проблема объединения сетей стояла всегда, и масса решений её неукоснительно увеличивается с появлением специфических технологий и топологий. В данной выпускной квалификационной работе рассматривается специфика объединения сетей внутри одного или нескольких ЦОД. Поскольку число Software defined data centers (SDDC), гибридных облаков и прочих подобных ЦОД неукоснительно растёт, растёт и количество администраторов, сталкивающихся с этой проблемой.

### Постановка задачи и первичный анализ проблемы

Сама по себе эта проблема многогранна, и в процессе поиска пути её решения администраторам приходится решать сопутствующие ей подпроблемы. С учётом специфики решаемого вопроса, исходными данными чаще всего являются следующие аспекты:

* Уже существуют две раздельные IP-сети с некоторым количеством виртуальных машин в каждой
* Операционные системы на виртуальных машинах в разных сетях совершенно не однородны
* Конфигурацию виртуальных машин менять нельзя, т.е. крайне не желательна инсталляция каких бы то ни было приложений

Такие исходные данные можно изобразить простой схемой (см. Рисунок 4).

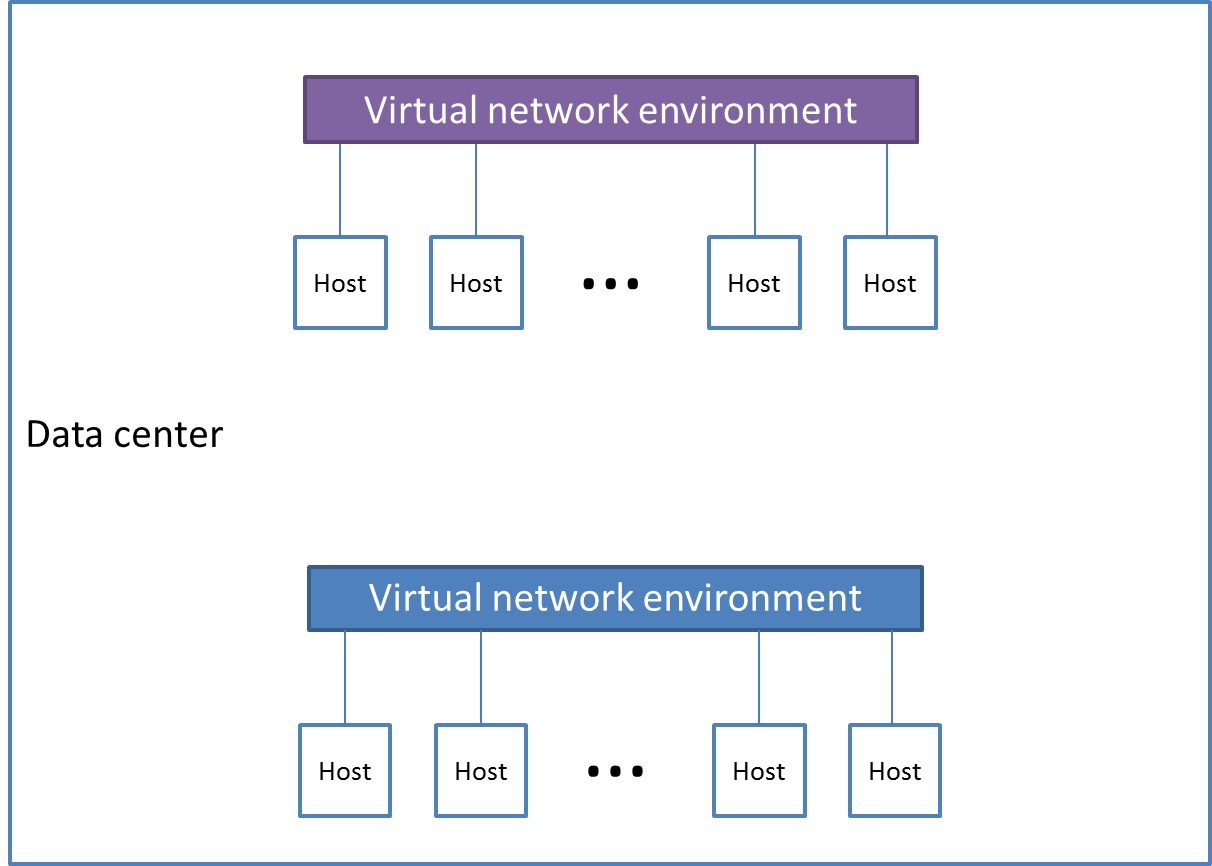


Рисунок 4 – Исходные данные проблемы соединения двух виртуальных сетей

Часто возникают и дополнительные трудности, такие как:

* Одна или обе сети принадлежат к внешнему (external) типу виртуальной сети
* Сети сильно разделены между собой, то есть находятся в разных ЦОД, не связанных между собой напрямую, но имеющих доступ в глобальный интернет

Изобразим такую ситуацию на рисунке 5. pNIC на нём означает физическую сетевую карту (physical network interface controller).

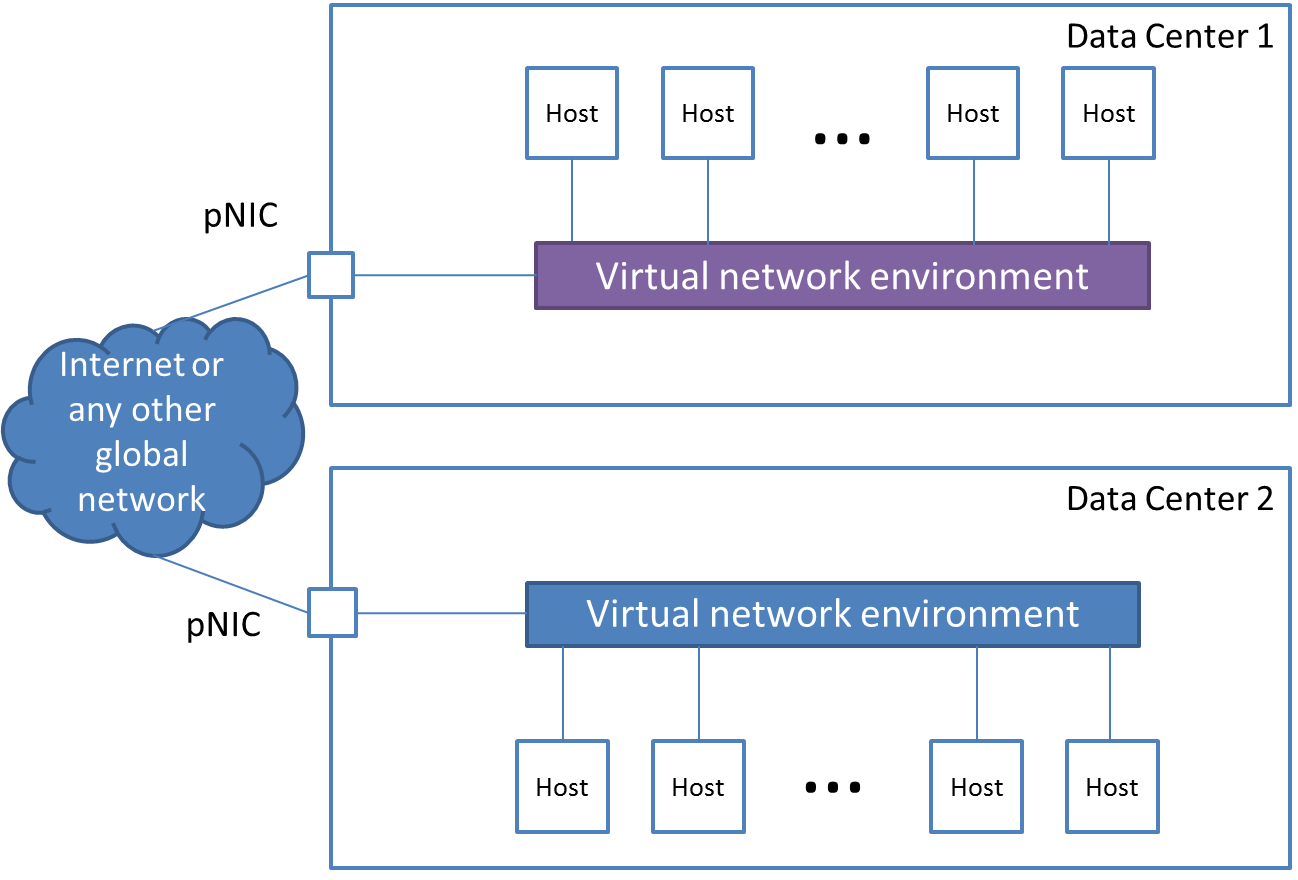


Рисунок 5 – Исходные данные проблемы соединения виртуальных сетей внешнего типа

Конечно, здесь может быть ещё масса других вариантов конфигурации, в том числе с различными конфигурациями доступа в Интернет, трансляцией адресов NAT, существующими VPN сетями и туннелями, а так же требованиями безопасности, такими как Firewall. Общее решение для столь многогранных частностей кажется изначально невозможным. Тем не менее, попытаемся подобрать одно решение для наиболее общего случая.

С учётом всего вышесказанного, решаемая проблема такова: разработка платформонезависимого решения проблемы соединения двух виртуальных сетей, не затрагивающего конфигурации виртуальных машин входящих в них, с возможностью использования этого решения поверх готовых IP-сетей.

# Проектирование решения

Первой, естественной идеей для решения задачи объединения двух сетей, является добавление одного промежуточного хоста, как показано на рисунке 6.

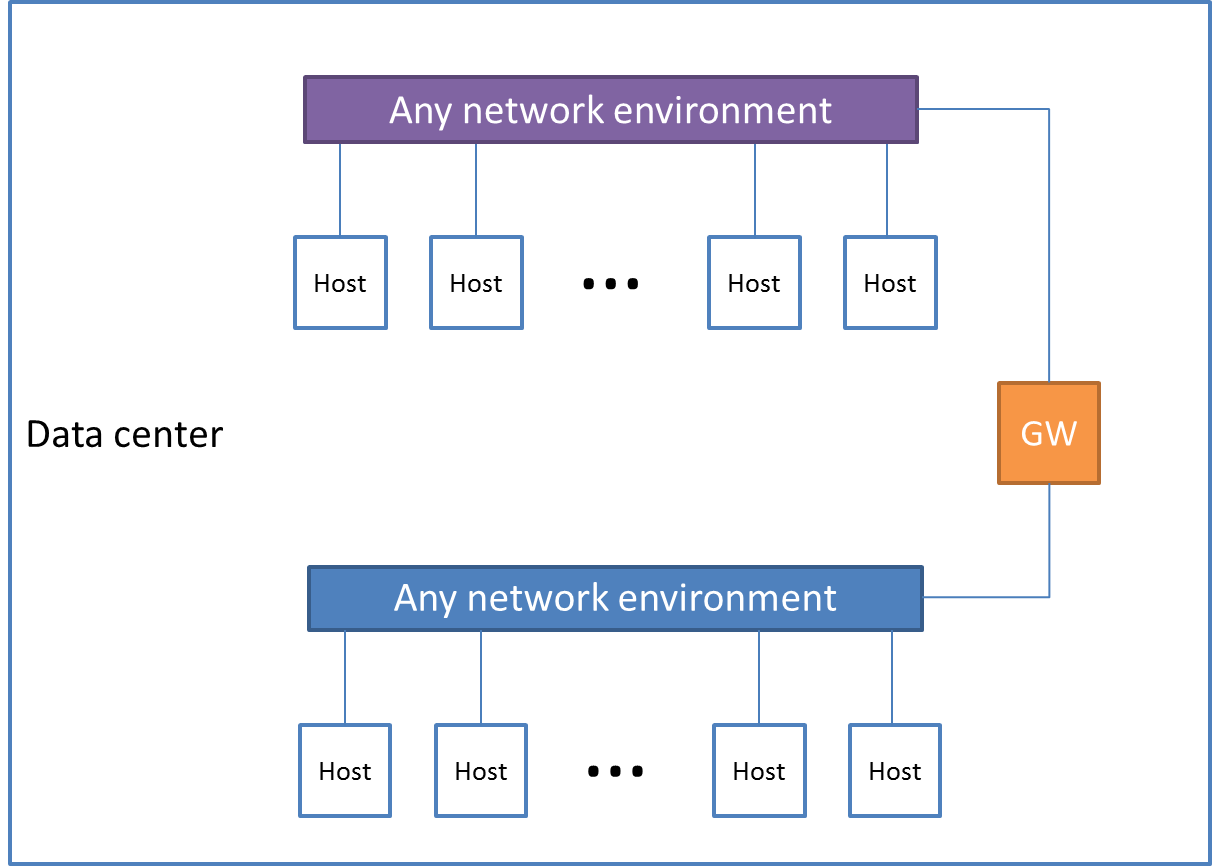


Рисунок 6 – простейшее решение для соединения сетей

Большинство системных администраторов прибегают именно к этому решению, но вынуждены вручную с нуля конфигурировать введённый хост, выбирать или изучать технологии, помогающие маршрутизировать траффик внутри хоста. Выбор довольно велик: GRE туннели, Linux Bridge & routing, Open vSwitch, VLAN\VxLAN. Это конфигурирование можно существенно упростить, имея готовый набор скриптов, но что если уже объединённую таким образом сеть понадобится соединить с ещё одной? Масштабирование такого решения вносит не меньшее количество проблем.

А что если эти сети расположены на разных кластерах? Одним промежуточным хостом в таком случае проблему не решить. Рисунок 7 иллюстрирует решение, родственное первому.

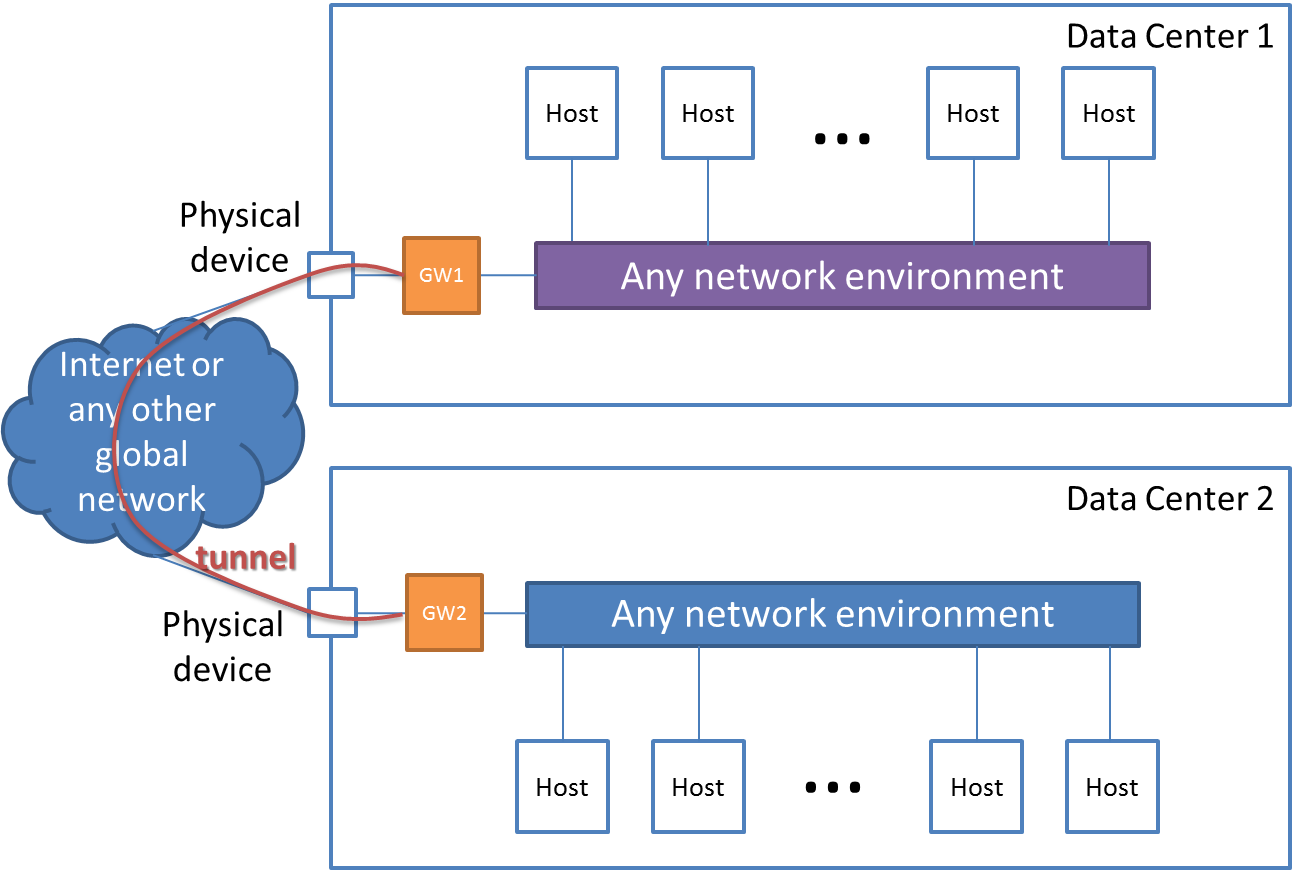


Рисунок 7

Такой подход заведомо усложнит конфигурирование вдвое. На рисунке 7 красным цветом изображен тегированный траффик между двумя соединяемыми сетями.

Как решить проблему сложного конфигурирования и масштабируемости? Было принято решение к введению следующей топологии (см. рисунок 8).

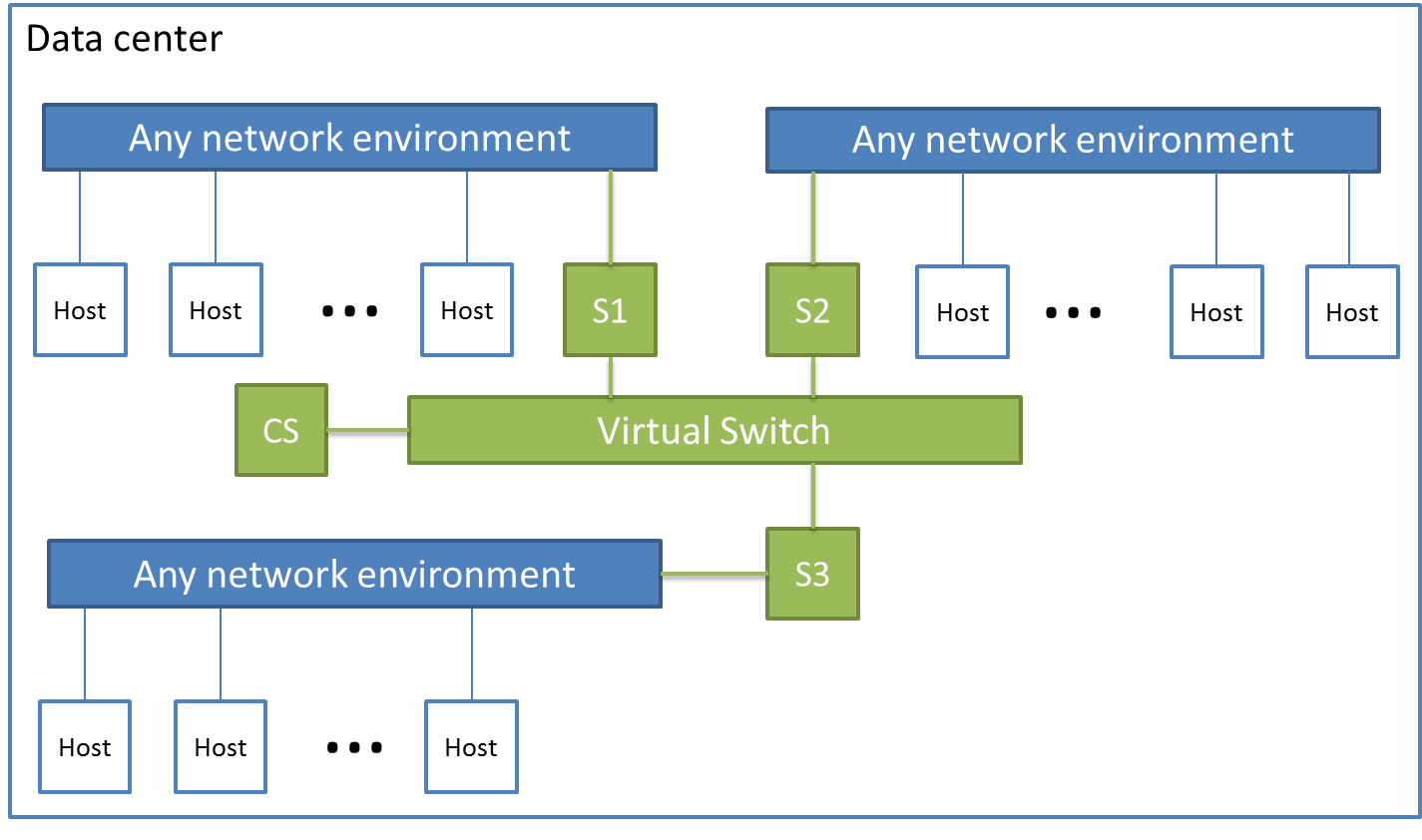


Рисунок 8 – топология решения

Зелёным помечены сущности, которые необходимо ввести. CS – контрольная станция (control station), выполняющая функции управления виртуальными машинами, обеспечивающими туннеллирование и маршрутизацию траффика, а именно машинами, названными Sn (station number n). На рисунке сразу же приведена топология решения для объединения трёх сетей. Несложно понять, как масштабировать такое на n сетей. Virtual Switch, так же помеченный зелёным, должен быть предоставлен средствами гипервизора.

Теперь рассмотрим подробнее архитектуру этого решения. Она изображена на рисунке 9.

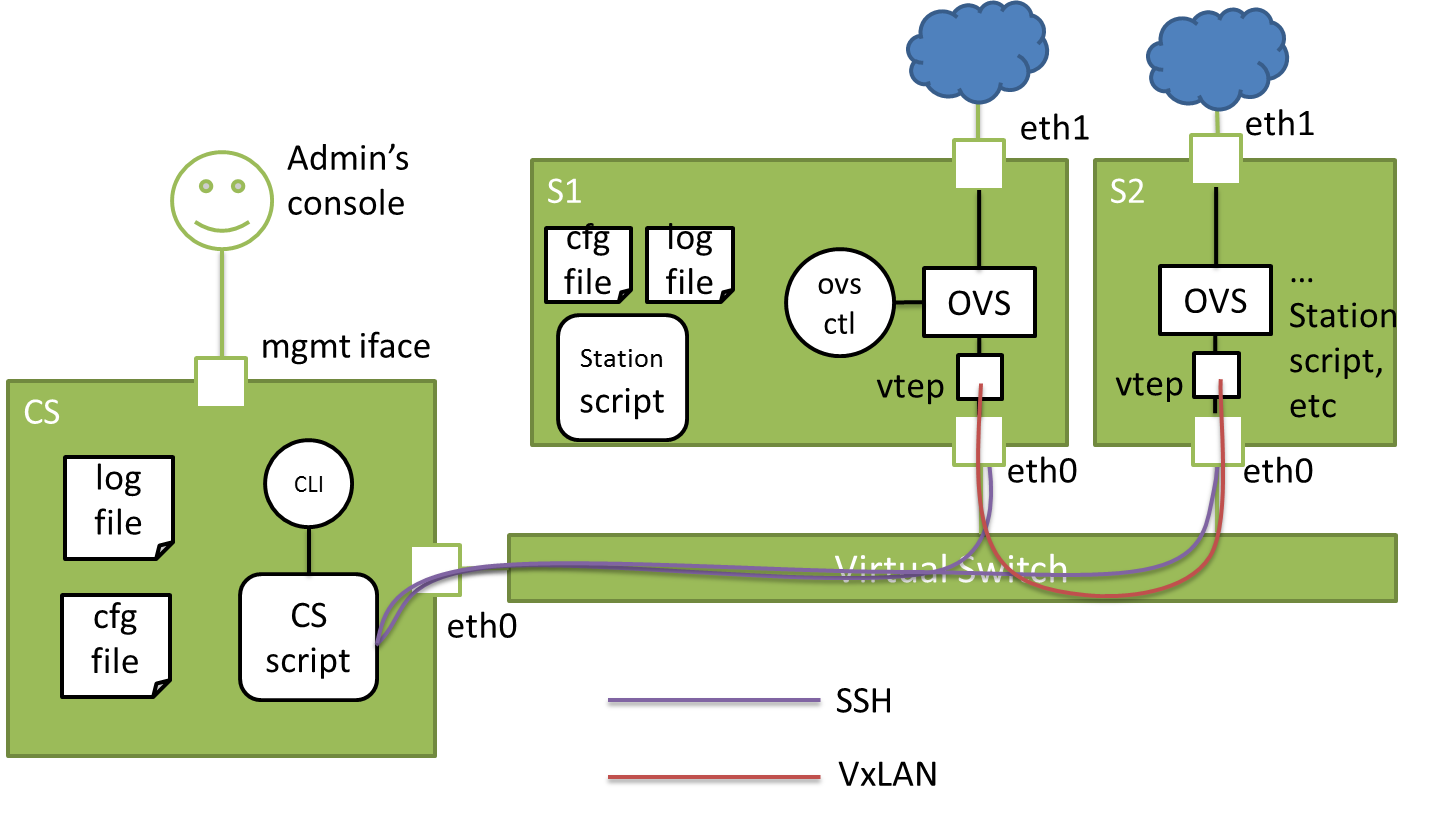


Рисунок 9 – подробная архитектура решения

В качестве рабочей ОС для всех вводимых машин был выбран дистрибутив Linux Ubuntu Server 14.04 LTS. Выбор был сделан за счёт популярности системы, довольно компактных размеров за счёт отсутствия GUI в серверной версии, хорошей документированности и массы информации на форумах, в т.ч. и русскоязычных, а так же LTS (long time support) релиза.

Интерфейс, который не изображён на рисунке 8 и носит название mgmt iface (management interface, менеджмент интерфейс) служит для подключения администратора к консоли машины. Оттуда ему доступны команды управления для CS script’а. Интерфейсом этого взаимодействия является командная строка (command line interface, CLI).

Управляющий скрипт контрольной станции хранит и поддерживает в актуальном состоянии лог своей работы (log file), а так же файл конфигурации сети (cfg file). Основная его задача, это сконфигурировать машины-станции по команде пользователя. Коммуникация между любой из станций и контрольной станцией осуществляется при помощи протокола SSH. При создании соединения между сетями, контрольная станция подключается к станциям и запускает их внутренние скрипты с определённым набором параметров. Эти скрипты должны

* Очистить результаты предыдущих запусков
* Создать экземпляр Open vSwitch
* Создать экземпляр VTEP (vxlan tunnel endpoint) для VxLAN туннеля
* Включить eth и vtep интерфейсы в Open vSwitch
* Сконфигурировать все эти объекты для корректного создания туннеля
* Проверить доступность и работоспособность созданной конфигурации
* Сохранить записи о действиях в лог

## Проектирование и сборка топологий

В процессе разработки решения и конфигурационных скриптов важна первая тестовая топология. Поскольку множество реальных топологий довольно велико, важно протестировать технологические и количественные ограничения, создаваемые множеством выбранных технологий.

Начнём с максимально простых IP-сетей: двух не пересекающихся сетей C-класса, соединённых между собой только одной станцией. Изобразим такую топологию на рисунке 10.

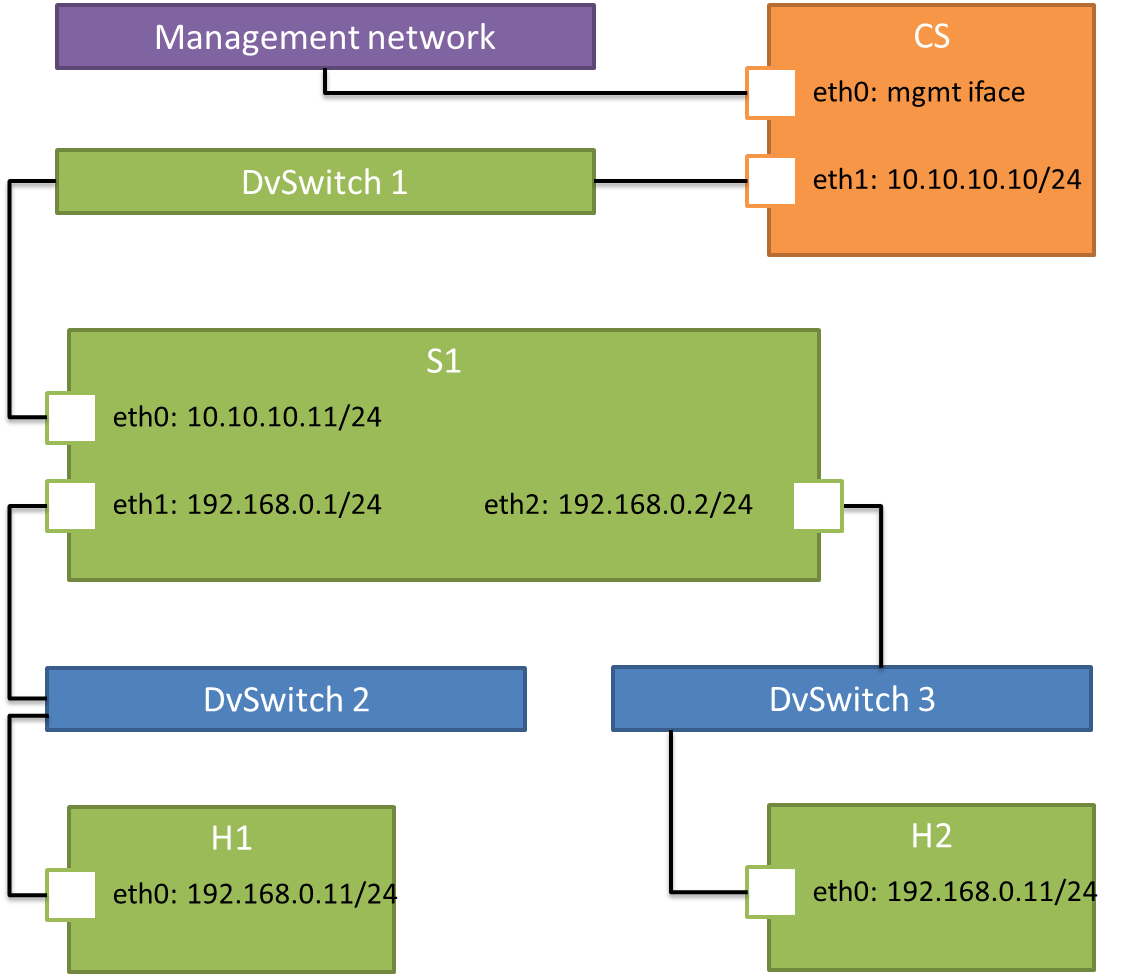


Рисунок 10 – первая тестовая топология

Причиной создания такой топологии послужила потребность в получении оценки сложности конфигурирования такой сети при помощи выбранных инструментов. Так же, наличие такой топологии позволит оценить ширину канала и скорость передачи, которую в последствие можно сравнивать с результатами в более сложных топологиях.

То, как настроен Open vSwitch и какие связи имеют компоненты в данной топологии можно изобразить на схеме следующим образом (см. рисунок 11).

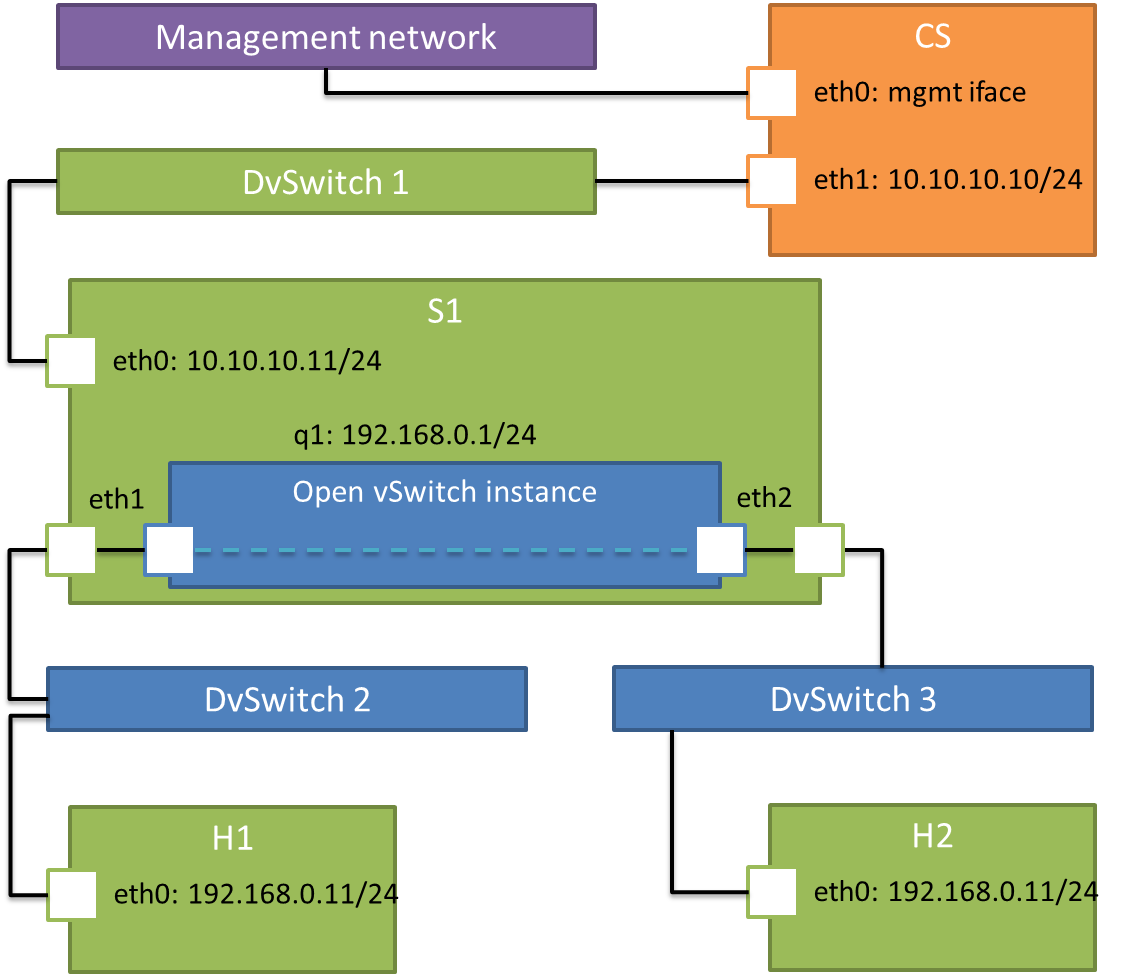


Рисунок 11

Ниже приведены шаги и пояснения для конфигурирования такой топологии. Этап сборки тестового стенда представляет не столь большой интерес для целей данной выпускной квалификационной работы, поскольку представляет собой лишь опыт пользования гипервизором. Тем не менее, стоит упомянуть, что на DvSwitch 2 и 3 в гипервизоре VMWare ESX необходимо выставить неразборчивый режим (promiscuous mode).

Установка Open vSwitch

$ sudo apt-get install openvswitch-switch

Для проверки можно использовать команду sudo service openvswitch-switch status. Вот её вывод после установки Open vSwitch версии 2.0.2.

$ sudo service openvswitch-switch status

openvswitch-switch start/running

Теперь, после проверки работоспособности служб, можно начать настройку. Создадим экземпляр виртуального бриджа с именем q1. Имя выбрано лишь для удобства набора команд, поскольку первое тестовое конфигурирование не предполагает наличие сразу же готового скрипта настройки. Его предстоит создать, анализируя последующие шаги.

$ sudo ovs-vsctl add-br q1

Сбросим eth1 и eth2 и добавим их к бриджу q1.

$ sudo ifconfig eth1 0

$ sudo ifconfig eth2 0

$ sudo ovs-vsctl add-port q1 eth1

$ sudo ovs-vsctl add-port q1 eth2

Текущее состояние Open vSwitch можно посмотреть при помощи команды sudo ovs-vsctl show. Вывод после этих действий будет следующим:

$ sudo ovs-vsctl show

8313bddd-1665-47a1-be41-1729e56aa95c

Bridge "q1"

Port "eth1"

Interface "eth1"

Port "q1"

Interface "q1"

type: internal

Port "eth2"

Interface "eth2"

ovs\_version: "2.0.2"

Теперь необходимо настроить q1 на перенаправление траффика между соединёнными в нём сетями. Команда ovs-ofctl применяется для ручной установки правил OpenFlow на q1. Структура этих правил определена в стандарте, а автоматизация процесса передачи и установки правил будет переложена на OpenFlow контроллер SDN позднее. Для начала, нужно выяснить номера портов, уже созданных в q1 для корректного назначения правил. Сделать это можно с помощью команды sudo ovs-ofctl show q1. Её вывод был следующим:

$ sudo ovs-ofctl show q1

OFPT\_FEATURES\_REPLY (xid=0x2): dpid:0000005056ae1ca5

n\_tables:254, n\_buffers:256

capabilities: FLOW\_STATS TABLE\_STATS PORT\_STATS QUEUE\_STATS ARP\_MATCH\_IP

actions: OUTPUT SET\_VLAN\_VID SET\_VLAN\_PCP STRIP\_VLAN SET\_DL\_SRC SET\_DL\_DST SET\_NW\_SRC SET\_NW\_DST SET\_NW\_TOS SET\_TP\_SRC SET\_TP\_DST ENQUEUE

**3**(eth1): addr:00:50:56:ae:1c:a5

config: 0

state: 0

current: 1GB-FD COPPER AUTO\_NEG

advertised: 10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO\_NEG

supported: 10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO\_NEG

speed: 1000 Mbps now, 1000 Mbps max

**4**(eth2): addr:00:50:56:ae:75:2e

config: 0

state: 0

current: 1GB-FD COPPER AUTO\_NEG

advertised: 10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO\_NEG

supported: 10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO\_NEG

speed: 1000 Mbps now, 1000 Mbps max

LOCAL(q1): addr:00:50:56:ae:1c:a5

config: 0

state: 0

speed: 0 Mbps now, 0 Mbps max

OFPT\_GET\_CONFIG\_REPLY (xid=0x4): frags=normal miss\_send\_len=0

В этом выводе искомые номера портов выделены жирным. Поскольку выполнение вышеперечисленных команд было частью ручного конфигурирования, то номера портов могут отличаться при последующем повторении. Этот процесс предстоит автоматизировать позднее, а пока просто приведём команды, назначающие нужные правила для достижения целей создания этой топологии.

Удалим все существующие правила и добавим новые.

$ sudo ovs-ofctl del-flows q1

$ sudo ovs-ofctl add-flow q1 priority=100,in\_port=3,idle\_timeout=0,action=output:4

$ sudo ovs-ofctl add-flow q1 priority=100,in\_port=4,idle\_timeout=0,action=output:3

$ sudo ovs-ofctl add-flow q1 priority=0,action=normal

Как видно, эти правила задают маршрутизацию для пакетов пришедших на 3 и на 4 порт, а так же стандартное правило с самым низким приоритетом. Установленные правила на q1 можно посмотреть, выполнив следующую команду:

$ sudo ovs-ofctl dump-flows q1

Ниже приведён её вывод

NXST\_FLOW reply (xid=0x4):

cookie=0x0, duration=106.3s, table=0, n\_packets=90, n\_bytes=53016, idle\_age=1, priority=100,in\_port=3 actions=output:4

cookie=0x0, duration=83.398s, table=0, n\_packets=0, n\_bytes=0, idle\_age=83, priority=100,in\_port=4 actions=output:3

cookie=0x0, duration=28.367s, table=0, n\_packets=26, n\_bytes=17416, idle\_age=0, priority=0 actions=NORMAL

Для корректной работы сети, стоит установить IP-адрес для q1:

$ sudo ip addr add 192.168.0.1/24 dev q1

А так же, назначить его как default gateway на виртуальных машинах h1 и h2.

$ sudo route add default gw 192.168.0.1

После этого, между h1 и h2 будет проходить ping в обе стороны. Пример ping с h2 до h1:

$ ping 192.168.0.11

PING 192.168.0.11 (192.168.0.11) 56(84) bytes of data.

64 bytes from 192.168.0.11: icmp\_seq=1 ttl=64 time=0.674 ms

64 bytes from 192.168.0.11: icmp\_seq=2 ttl=64 time=0.484 ms

^C

--- 192.168.0.11 ping statistics ---

2 packets transmitted, 2 received, 0% packet loss, time 999ms

rtt min/avg/max/mdev = 0.484/0.579/0.674/0.095 ms

Теперь, соберём статистику для качественного анализа. Установим пакет iperf и зависимости для него. Опустим шаги установки, поскольку она не представляет особого интереса для целей данной работы, но опишем запуск и результаты. Запустим на h1 iperf в режиме сервера:

$ iperf3 –s

Старт iperf в режиме клиента на h2 осуществляется следующей командой:

$ iperf3 -c 192.168.0.11

Не трудно заметить, что IP-адрес соответствует адресу h1. Вывод данной утилиты был следующим:

Connecting to host 192.168.0.11, port 5201

[ 4] local 192.168.0.12 port 45244 connected to 192.168.0.11 port 5201

[ ID] Interval Transfer Bandwidth Retr Cwnd

[ 4] 0.00-1.00 sec 111 MBytes 934 Mbits/sec 0 416 KBytes

[ 4] 1.00-2.00 sec 112 MBytes 941 Mbits/sec 0 434 KBytes

[ 4] 2.00-3.00 sec 111 MBytes 928 Mbits/sec 0 454 KBytes

[ 4] 3.00-4.00 sec 112 MBytes 940 Mbits/sec 0 454 KBytes

[ 4] 4.00-5.00 sec 111 MBytes 930 Mbits/sec 0 475 KBytes

[ 4] 5.00-6.00 sec 109 MBytes 911 Mbits/sec 24 379 KBytes

[ 4] 6.00-7.00 sec 111 MBytes 928 Mbits/sec 0 409 KBytes

[ 4] 7.00-8.00 sec 112 MBytes 936 Mbits/sec 0 471 KBytes

[ 4] 8.00-9.00 sec 111 MBytes 935 Mbits/sec 0 471 KBytes

[ 4] 9.00-10.00 sec 112 MBytes 942 Mbits/sec 0 471 KBytes

- - - - - - - - - - - - - - - - - - - - - - - - -

[ ID] Interval Transfer Bandwidth Retr

[ 4] 0.00-10.00 sec 1.09 GBytes 933 Mbits/sec 24 sender

[ 4] 0.00-10.00 sec 1.08 GBytes 930 Mbits/sec receiver

iperf Done.

Количественные результаты получены, а оценка их будет получена позже, в результате сравнений. Теперь же, усложним задачу, приблизив её к нужной и начнём со следующей схемы сети (см. рисунок 12).

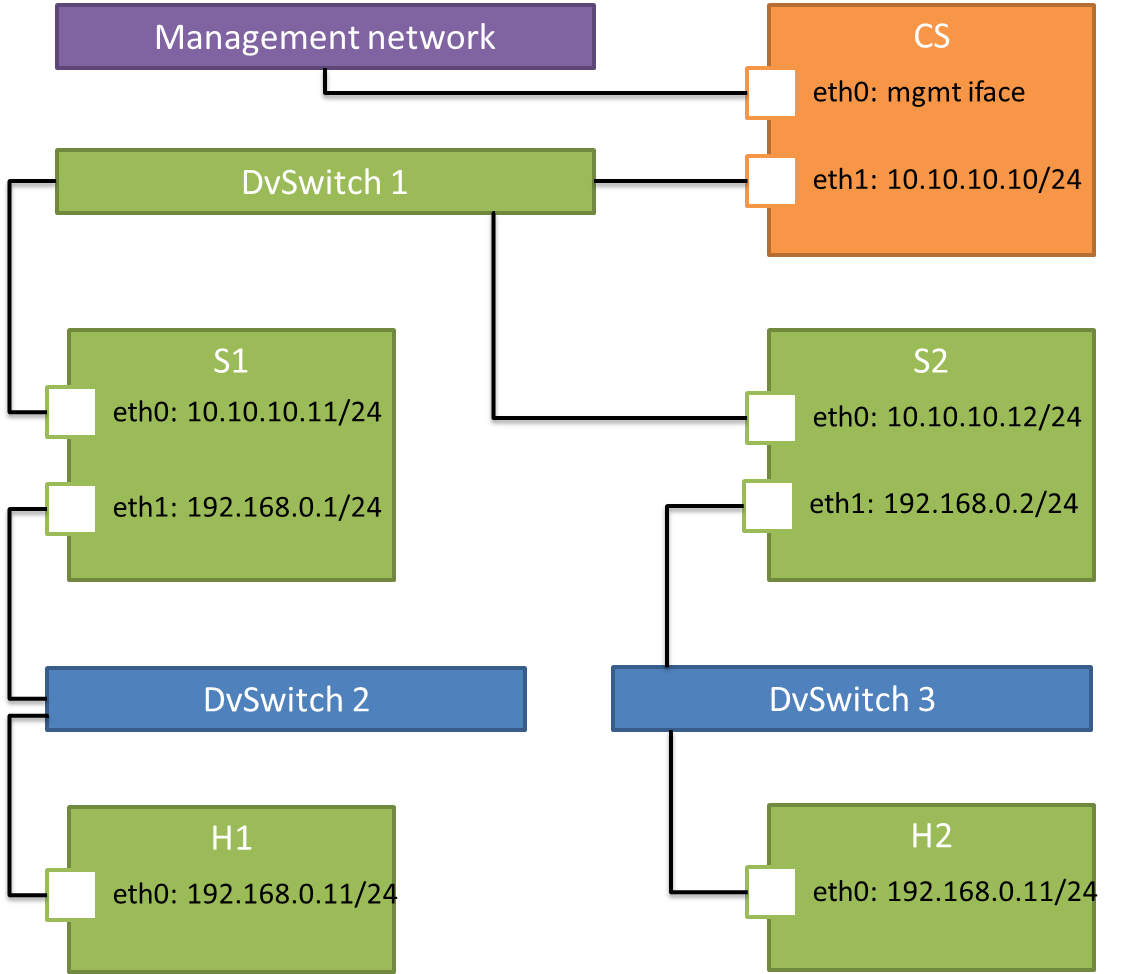


Рисунок 12 – вторая тестовая топология

Перед началом перестроения уже собранной топологии, Очистим S1 перед отключением её от DvSwitch 3:

$ sudo ovs-vsctl del-br q1

Далее выполним на обеих станциях уже известные шаги по созданию бриджа и добавлению в него порта eth1, а так же установки IP-адреса на bridge интерфейс. Выполнение этих команд было записано в одну строку для сокращения. Символы «--» позволяют передать в ovs-vsctl сразу несколько команд. Так же, все команды были выполнены от имени супер пользователя для краткости.

S1:

# ifconfig eth1 0

# ovs-vsctl add-br q1 -- add-port q1 eth1

# ip addr add 192.168.0.1/24 dev q1

S2:

# ifconfig eth1 0

# ovs-vsctl add-br w1 -- add-port w1 eth1

# ip addr add 192.168.0.2/24 dev w1

Создадим VxLAN-туннель между двумя бриджами, выполнив связку команд по добавлению порта, а если быть технически точным, то виртуального интерфейса к существующим бриджам. Команды на S1 и S2 будут отличаться лишь именем бриджа и опцией remote\_ip, которая указывает на IP-адрес другого конца туннеля. Присвоим имя «vtep» этому виртуальному интерфейсу. Это не значит, что в этой команде создаётся именно экземпляр объекта VxLAN VTEP, который принято понимать как виртуальное устройство упаковки\распаковки тегированного траффика. Тем не менее, смысловая нагрузка и аббревиатура хорошо согласуются с именованием объекта VxLAN VTEP (vxlan tunnel endpoint).

S1:

ovs-vsctl add-port q1 vtep -- set interface vtep type=vxlan option:remote\_ip=10.10.10.12 option:key=flow ofport\_request=10

S2:

ovs-vsctl add-port w1 vtep -- set interface vtep type=vxlan option:remote\_ip=10.10.10.11 option:key=flow ofport\_request=10

Опция key=flow указывает Open vSwitch на то, что созданный интерфейс может являться конечной точкой для любого тэга траффика. С каким бы VNI ни пришёл пакет, он будет принят через этот виртуальный интерфейс и обработан внутри бриджа, а не отброшен. Опция ofport\_request=10 избавляет от неопределённости номера Open Flow порта в бридже, т.е. в выводе команды ovs-ofctl show q1 будет видно, что порт vtep имеет десятый номер:

# ovs-ofctl show w1

OFPT\_FEATURES\_REPLY (xid=0x2): dpid:0000005056ae79fe

n\_tables:254, n\_buffers:256

capabilities: FLOW\_STATS TABLE\_STATS PORT\_STATS QUEUE\_STATS ARP\_MATCH\_IP

actions: OUTPUT SET\_VLAN\_VID SET\_VLAN\_PCP STRIP\_VLAN SET\_DL\_SRC SET\_DL\_DST SET\_NW\_SRC SET\_NW\_DST SET\_NW\_TOS SET\_TP\_SRC SET\_TP\_DST ENQUEUE

4(eth1): addr:00:50:56:ae:79:fe

config: 0

state: 0

current: 1GB-FD COPPER AUTO\_NEG

advertised: 10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO\_NEG

supported: 10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO\_NEG

speed: 1000 Mbps now, 1000 Mbps max

**10**(vtep): addr:36:61:64:05:56:77

config: 0

state: 0

speed: 0 Mbps now, 0 Mbps max

LOCAL(w1): addr:00:50:56:ae:79:fe

config: 0

state: 0

speed: 0 Mbps now, 0 Mbps max

OFPT\_GET\_CONFIG\_REPLY (xid=0x4): frags=normal miss\_send\_len=0

Зафиксируем так же и вывод команды ovs-vsctl show:

# ovs-vsctl show

8313bddd-1665-47a1-be41-1729e56aa95c

Bridge "w1"

Port "w1"

Interface "w1"

type: internal

Port "eth1"

Interface "eth1"

Port vtep

Interface vtep

type: vxlan

options: {key=flow, remote\_ip="10.10.10.11"}

ovs\_version: "2.0.2"

Приведём вывод ip a на S2 для полной ясности настройки. Описание интерфейса lo было удалено, поскольку не представляет показательного интереса.

# ip a

1: lo: …

2: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 00:50:56:ae:57:25 brd ff:ff:ff:ff:ff:ff

inet 10.10.10.12/24 brd 10.10.10.255 scope global eth0

valid\_lft forever preferred\_lft forever

3: eth1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast master ovs-system state UP group default qlen 1000

link/ether 00:50:56:ae:79:fe brd ff:ff:ff:ff:ff:ff

4: ovs-system: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default

link/ether 1a:e0:12:8f:01:bc brd ff:ff:ff:ff:ff:ff

5: w1: <BROADCAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UNKNOWN group default

link/ether 00:50:56:ae:79:fe brd ff:ff:ff:ff:ff:ff

inet 192.168.0.2/24 scope global w1

valid\_lft forever preferred\_lft forever

После этого, необходимо задать правила маршрутизации. Как и раньше, зададим их при помощи ovs-ofctl, но на сей раз используем возможность передачи текстового файла этой утилите для ускорения процесса. Команду необходимо выполнить на обеих станциях. Правила будут идентичны для обоих бриджей.

# ovs-ofctl add-flows q1 ofrule2.txt

Ниже приведено содержание файла ofrule2.txt.

# cat ofrule2.txt

table=0,in\_port=4,actions=set\_field:100->tun\_id,resubmit(,1)

table=0,actions=resubmit(,1)

table=1,tun\_id=100,arp,nw\_dst=10.10.10.11,actions=output:4

table=1,tun\_id=100,arp,nw\_dst=10.10.10.12,actions=output:10

table=1,in\_port=4,action=output:10

table=1,in\_port=10,action=output:4

Первое правило, созданное в таблице 0, говорит о том, что всё, что пришло через порт 4 (eth1), необходимо протегировать с VNI=100 и перенаправить для маршрутизации в таблицу 1. Второе правило просто перенаправляет любой пакет на маршрутизацию в таблицу 1.

Далее идёт описание маршрута ARP-запроса и перенаправление пакетов между портами eth1 и vtep аналогичное тому, что было задано в предыдущей топологии.

И так, проверим ping с h1 до h2:

$ ping 192.168.0.12 -c 3

PING 192.168.0.12 (192.168.0.12) 56(84) bytes of data.

64 bytes from 192.168.0.12: icmp\_seq=1 ttl=64 time=0.938 ms

64 bytes from 192.168.0.12: icmp\_seq=2 ttl=64 time=1.05 ms

64 bytes from 192.168.0.12: icmp\_seq=3 ttl=64 time=0.623 ms

--- 192.168.0.12 ping statistics ---

3 packets transmitted, 3 received, 0% packet loss, time 2002ms

rtt min/avg/max/mdev = 0.623/0.870/1.050/0.182 ms

Аналогично работает и в другую сторону.

Что творится в это время на S1 на интерфейсе eth0 (10.10.10.11)? Оставим ping работать, а результат посмотрим с помощью утилиты tcpdump.

# tcpdump -i eth0

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes

06:59:19.265640 IP 10.10.10.11.ssh > 10.10.10.10.53607: Flags [P.], seq 1640662855:1640663051, ack 359018905, win 303, options [nop,nop,TS val 1259407 ecr 920458294], length 196

06:59:19.265914 IP 10.10.10.10.53607 > 10.10.10.11.ssh: Flags [.], ack 196, win 322, options [nop,nop,TS val 920458305 ecr 1259407], length 0

06:59:20.056697 IP 10.10.10.11.56647 > 10.10.10.12.4789: **VXLAN**, flags [I] (0x08), vni 0

IP 192.168.0.11 > 192.168.0.12: **ICMP echo request**, id 16078, seq 29, length 64

06:59:20.056998 IP 10.10.10.12.33174 > 10.10.10.11.4789: **VXLAN**, flags [I] (0x08), vni 0

IP 192.168.0.12 > 192.168.0.11: **ICMP echo reply**, id 16078, seq 29, length 64

06:59:20.057507 IP 10.10.10.11.ssh > 10.10.10.10.53606: Flags [P.], seq 2874236419:2874236519, ack 2481302211, win 303, options [nop,nop,TS val 1259605 ecr 920458253], length 100

Жирным шрифтом выделены интересующие нас типы пакета: VXLAN и ICMP echo. Дамп содержит не только их, но и ssh пакеты, поскольку подключение к S1 выполнено по этому протоколу с контрольной станции (10.10.10.10).

Изобразим полученную топологию внутри станций диаграммой для лучшего понимания настройки (см. рисунок 13).

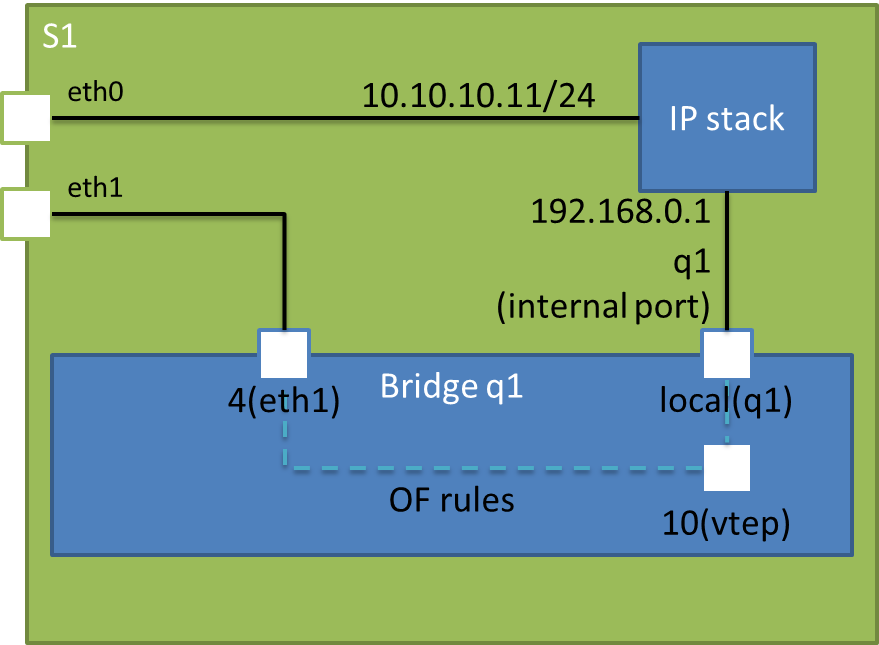


Рисунок 13

## Реализация автоматического конфигуратора

Для начала убедимся, что контрольная станция (CS) может получить права супер пользователя на станциях. По умолчанию в Ubuntu 14.04 server эта возможность выключена. Более того, скриптам контрольной станции понадобится возможность исполнения от супер пользователя. Включим эти возможности. Для начала, сгенерируем пару RSA ключей на контрольной станции:

ssh-keygen -t rsa

По умолчанию, сгенерированный ключ будет храниться по такому пути /root/.ssh/id\_rsa.pub. Но прежде чем пытаться установить его на S1 и S2 придётся включить супер пользователя, установить ему пароль и разрешить удалённый доступ с его правами.

Для этого на S1 и S2 выполним следующую последовательность команд:

$ sudo passwd -u root

$ sudo passwd root

$ su

# vi /etc/ssh/sshd\_config

Исправим в нём строку «PermitRootLogin without-password» на «PermitRootLogin yes»

# service ssh restart

Теперь можно копировать сгенерированный ранее ключ для супер пользователя. Выполним с контрольной станции две команды, добавляющие возможность выполнения привилегированных команд:

# ssh-copy-id -i /root/.ssh/id\_rsa.pub root@10.10.10.11

# ssh-copy-id -i /root/.ssh/id\_rsa.pub root@10.10.10.12

### Первая топология

1. <http://www.amt.ru/content/rus/art_text_pict/343/1.pdf> [↑](#footnote-ref-1)
2. <https://blogs.technet.microsoft.com/jhoward/2008/06/17/hyper-v-what-are-the-uses-for-different-types-of-virtual-networks/> [↑](#footnote-ref-2)
3. Система, использующая функции живой миграции (live migration), чтобы динамически перераспределять нагрузку, перенося машины с хоста на хост и с хранилища данных на хранилище данных. [↑](#footnote-ref-3)