МИНИСТЕРСТВО ОБРАЗОВАНИЯ АЗЕРБАЙДЖАНСКОЙ РЕСПУБЛИКИ

АЗЕРБАЙДЖАНСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ

«ЦЕНТР МАГИСТРАТУРЫ»

На правах рукописи

МАМЕДЗАДЕ АГИЛЬ КАМИЛЬ

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

НА ТЕМУ:

«СОЗДАНИЕ И УПРАВЛЕНИЕ ВИРТУАЛЬНЫМИ СЕТЯМИ»

Наименование и шифр специальности: 060509 «Компьютерные науки»

Наименование и шифр специализации: İİM020004 «Информационные технологии управления»

Научный руководитель: **д.ф.м., РЗАЕВА У.**

Руководитель

магистерской программы: **к.ф.-м.н., доц. АЛИЕВА Т.А.**

Заведующий кафедрой: **акад. АББАСОВ А.М.**

БАКУ – 2018

Содержание

Введение

Актуальность темы. Виртуализация – это абстрагированные от физических (аппаратных) аппаратной реализации, комплексы вычислительных ресурсов, которые полностью изолированы между собой, находясь при этом в одной физической машине. Виртуальные сети, в свою очередь выступают тут в роли виртуальных коммуникационных ресурсов. Широко используются две реализации данной технологии: VPN (Virtual Private Network) и VLAN (Virtual Local Area Network). VPN – виртуальная частная сеть, это логическая сеть (одно или более соединений) реализованная по верх другой сети (сети Интернет в частности), используя шифрование. Данная технология часто используется для соединения нескольких удаленных локальных сетей. Как пример можно показать коммуникация офисом географически отдаленных друг от друга или же CYOD (Choose Your Own Device) или BYOD (Bring Your Own Device). VLAN – виртуальная локальная сеть в свою очередь, является логической сетью, разделенной на подсети физической сети. Как пример можно представить разделение физической сети здания на логические сети отделов. Данные сети могут быть полностью изолированы и работать по заранее настроенным правилам передачи данных. Данные технологии повсеместно используются везде, но существуют несколько технологий реализации и различные подрядчики данных технологий. В данной работе рассматривается именно прототипы Xen и OpenFlow как два более распространённых. И рассматриваются производительность, показатели, интерфейсы, возможности и т.д. Основываясь на выше перечисленном данная работа считается актуальной.

Предмет и объект исследования. Предметом исследования являются перспективы и недостатки прототипов виртуализации, а так же виртуализации в целом и методы реализации и повышения эффективности работы виртуальных сетей. Объектом исследования являются наиболее распространённые прототипы виртуализации.

Основная цель и задачи исследования. Целью данной работы является: выявление основных привилегий прототипов, методы их развертывания, и повышение эффективности работы таких сетей.

Научная новизна. Научная новизна исследования заключается в научном обосновании роли технологий виртуализации в развертывании полноценных сетей с расширенными возможностями, в обосновании теоретического и практического применения прототипов виртуальных сетей как основной шаг при выборе типа реализации.

Теоретико-методологической основой исследования является утилиты анализа виртуальных сетей, применение которого способно лучше оценить производительность прототипов виртуальных сетей в различных сценариях развертывания, для понимания слабых и сильных сторон данного метода. Что приводит к более эффективным и сбалансированным релизам сети.

Практическая значимость данного исследования состоит в том, что результаты работы могут быть использованы администраторами сети в процессе создания виртуальных сетей, и при его усовершенствовании. Полученные результаты позволяют сделать вывод о наличии перспектив применения тех или иных прототипов. Выдвинутые предложения и рекомендации могут быть полезны при использовании виртуальных технологий.

Объем и структура исследования. Работа состоит из введения, 3 глав, 8 параграфов, 32 схематических рисунков, предложения и рекомендация, списка использованной литературы, и заключения на азербайджанском и английских языках.

В I главе рассматривается виртуализация в общем, технологии виртуализации сетей, наиболее распространенные инструменты виртуализации и их положительные и отрицательные показатели, а так же оценивается эффективность применения прототипов виртуализации

используя различные сценарии и сопоставляя результаты показателей утилит на физической среде и виртуальной.

Во II главе рассматриваются интерфейсы виртуальных сетей, а так же их показатели и тенденции, в частности это внутренние и внешние интерфейсы прототипов Xen и OpenFlow. Их отличии и преимущества.

В III главе рассмотрены элементы управления виртуальными сетями и методы повышения эффективности работы виртуальных сетей, в частности прототипов Xen и OpenFlow.

I ГЛАВА. ВИРТУАЛИЗАЦИЯ

1.1. Технологии виртуализации

Для лучшего понимания средств виртуализации, важно определить различия между технологиями виртуализации. В этой главе диссертационной работы описаны концепции и методы, используемые Xen, VMware и OpenVZ. Существует множество различных определений виртуализации, но все они схожи в том, что данные технологии существуют для совместного использования вычислительных ресурсов и предоставления определенного уровня изоляции между виртуальными средами. Классическим определением, согласно Попеку и Голдбергу, является то, что виртуализация - это технология, которая обеспечивает виртуальные среды, являющиеся эффективными и изолированными подобно физическому оборудованию. Сегодня, это понятие может быть обобщено не только на аппаратных средствах, но и на любом вычислительном ресурсе, таком как ядро ОС или абстракция VM, при использовании языков программирования Java, C# и т.д. При этом в результате виртуализации возникают несколько проблем. Первая проблема, относящаяся ко все виртуальным средам, – это каким образом разделенные виртуальные образ получают доступ к основным вычислительным ресурсам. Для виртуализации аппаратных средств существуют общие ресурсы - это центральный процессор, RAM, долгосрочная память и сеть. Разделение по отношению к RAM-у может быть реализовано по-разному. Виртуальной среде можно предоставить доступ к пространству виртуальной памяти, которое имеет ссылки на блоки памяти физического RAM-а, тем же самым способом, которым пользуется ОС. Другой подход должен позволить VMs возможность получить знания о типе виртуализации и непосредственно получать доступ к памяти, выделенной гипервизору. Центральный процессор может быть представлен несколькими способами и осуществляться, используя такие механизмы, как круговая система, взвешенная круговая система, распределение по требованиям и др.

Ввод/вывод, в целом, может быть обработан общим способом, используя буфера для хранения переданных данных, которые являются мультиплексными и не мультиплексными между физической и виртуальной периферией. Сегодня на практике используют продукты с архитектурой x86, которые влекут дополнительные проблемы виртуализации. В начале развития виртуализации, а именно в 1970-х, аппаратные средства были разработаны для использования в виртуализации. В этой архитектуре центрального процессора виртуализация аппаратных средств могла быть достигнута с помощью использования техники, названной «de-privileging». По словам Попека и Голдберга, есть три требования для реализации гипервизора над физическими аппаратными средствами:

1. Эффективность, означающая, что многочисленная подгруппа — это набор команд от виртуального центрального процессора (vCPU), которые должны быть выполнены непосредственно в реальном центральном процессоре без любого вида вмешательства гипервизора;
2. Контроль за ресурсом, означающий, что гипервизор должен иметь полный контроль над всеми ресурсами;
3. Эквивалентность, означающая, что гипервизор должен предоставить виртуальный интерфейс виртуальной среде, эквивалентной оригинальному интерфейсу.

Полная виртуализация. Полная виртуализация - метод виртуализации, в котором виртуальны все оригинальные интерфейсы, причем, интерфейсы, выставленные в виртуальной среде, идентичны оригинальному интерфейсу. В этом подходе гостевой ОС, находящейся в VM, не должен быть изменен и непосредственно выполняется в VM.

Совместные усилия крупнейших производителей аппаратных средств направлены на оптимизацию процесса виртуализации. Консолидация сервера использует виртуализацию для замены нескольких серверов одним

единственным аппаратным средством с более высокими показателями, где выполняется работа нескольких виртуальных серверов. Консолидация сервера стала обычной практикой, направленной на сокращение оборудования и затрат на обслуживание в крупнейших компаниях. Поэтому и AMD, и Intel разработали технологии для более эффективной поддержки виртуализации в современных центральных процессорах. Intel Virtualization Technology (IVT) и виртуализация AMD (AMD-V) обеспечили лучшую работу для полной виртуализации, введя два новых режима функционирования: корневой и некорневой. Режим функционирования корня используется гипервизором и подобен регулярной операции в центральном процессоре, обеспечивая полный контроль за центральным процессором и реализуя традиционный метод четырех уровней привилегий. Некорневой способ предназначается для исполнения со стороны VM - ов. В этом способе центральный процессор также инициирует четыре уровня привилегии и гостя, в котором ОС больше не прекращает процесс на уровне «de-privileged», а совершает это действие на начальном уровне. Каждый раз, когда гостевой ОС выполняет проблематичную инструкцию, центральный процессор перехватывает его и возвращает контроль к гипервизору, чтобы использовать эту инструкцию.

Паравиртуализация. Паравиртуализация - это метод виртуализации, в котором гостевой ОС сотрудничает с гипервизором, для произведения лучшей работы. В паравиртуализации гостем называется тот ОС, который изменен, когда исполняется проблематичная инструкция. Чувствительная инструкция заменена осведомленной о виртуализации инструкцией и называется гипервизором. Поэтому гипервизор не должен контролировать выполнение VM для проблематичных инструкций. Компромиссом является то, что ОС должен быть изменен и повторно запущен, чтобы исполнить паравиртуализированные образы ОС. Паравиртуализированный образ ОС необходим, чтобы делать вызовы гипервизора. Это препятствует

виртуализации устаревшего ОС и требует сотрудничества с разработчиками ОС.

1.2. Инструменты виртуализации

В этом разделе диссертационной работы описываются главные методы виртуализации, и будут представлены результаты оценки деятельности для Xen, VMware и OpenVZ.

Xen - общедоступный гипервизор, предложенный для работы на товарных платформах аппаратных средств, которые используют метод паравиртуализации. Xen позволяет нам одновременно управлять многофункциональными VMs на единственной физической машине. Архитектура Xen состоит из одного гипервизора, расположенного выше физических аппаратных средств и нескольких VMs по гипервизору. У каждого VM может быть свой собственный ОС и приложения. Гипервизор управляет доступом к аппаратным средствам, а также имеющимися ресурсами, разделенными между VMs. Кроме того, драйверы устройств сохранены в изолированном VM, названном Доменом 0 (dom0) для обеспечения надежной и эффективной аппаратной поддержки. Поскольку у dom0 есть полный доступ к аппаратным средствам физической машины, у него существуют также специальные привилегии по сравнению с другими VMs, называемыми Пользовательскими Доменами (domUs). С другой стороны, у domUs есть виртуальные драйвера, названные драйверами фронтенда, которые общаются с драйверами бэкенда, расположенными в dom0 для получения доступа к физическим аппаратным средствам.

Далее, кратко поясним, как Xen виртуализирует каждый машинный ресурс для виртуального маршрутизатора: процессор, память и ввод/вывод устройства.

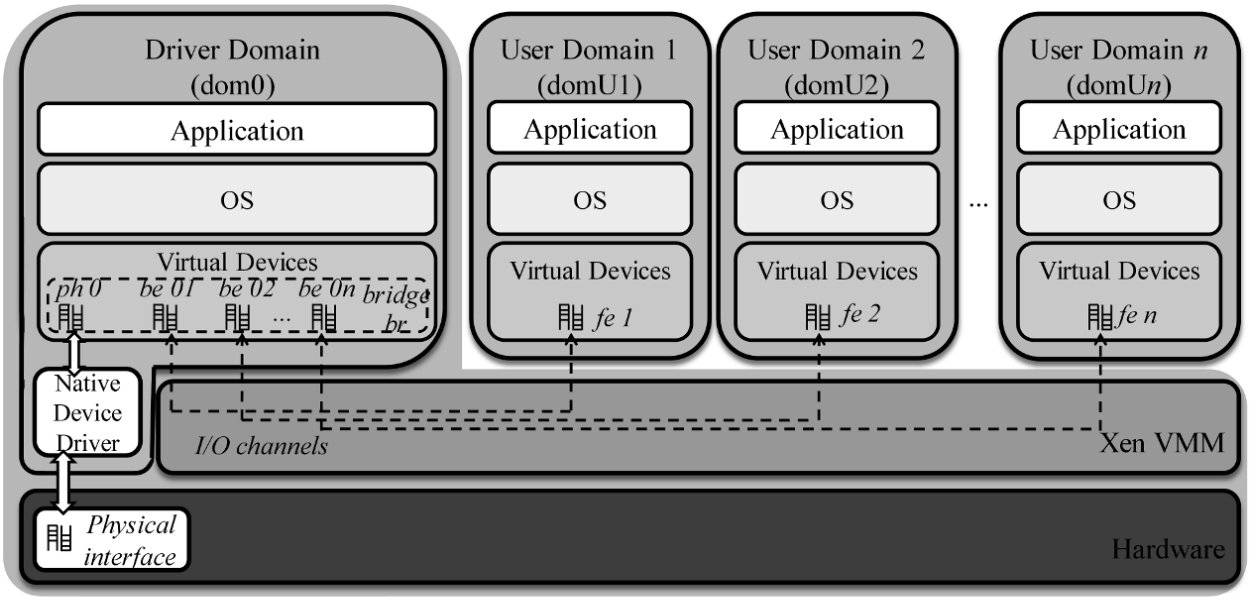


Рис.1. Название рисунка

Xen виртуализирует процессор, назначая vCPUs к VMs. vCPUs - это центральные процессоры, которые видят идущие процессы в каждой VM. Гипервизор Xen является планировщиком центрального процессора, который динамично распределяет процессорное время физического центрального процессора между каждым vCPU во время определенного периода. Планировщик по умолчанию используемой Xen - планировщик кредита. Планировщик кредита перераспределяет ресурсы центрального процессора каждому VM (или, более определенно, каждому vCPU) согласно параметрам, назначенным на VMs. Планировщик кредита может также быть сохранен для работы на Symmetric Multi Processing (SMP). Это означает, что планировщик разрешает физическим центральным процессорам достигать 100%, если какой-либо VM должен проделать работу. В сохраняющем работу планировщике нет никакого ограничения на сумму ресурсов центрального процессора, которые может использовать VM. Распределение памяти в Xen в настоящее время делается статически. Каждый VM получает установленный объем памяти, который определен во время его создания. В дополнение, чтобы потребовать минимального участия от гиперщитка, VMs ответственны

за распределение и управление соответствующей частью таблиц страниц аппаратных средств. Каждый раз, когда VM требует новые таблицы страниц, он ассигнует и инициализирует страницу от своего собственного места в памяти и регистрирует его в гиперщитке Xen, который ответствен за обеспечение изоляции. В Xen данные из устройств ввода/вывода переданы к каждому и от каждого VM, использующего совместно память асинхронного буферного дескрипторного уровня. Задача гиперщитка Xen состоит в том, чтобы выполнить ратификации проверки. Устройства ввода/вывода доступны Dom0 непосредственно при помощи его родных драйверов и также выполняют операции по вводу/выводу от имени domUs. С другой стороны, domUs используют свои драйвера бэкенда, чтобы запросить доступ устройства от dom0.

Особый случай виртуализации ввода/вывода - сетевая виртуализация, которая ответственна за демультиплексирование поступающих пакетов от физических интерфейсов до VMs и также для мультиплексирования коммуникационных пакетов, произведенных VMs. Рисунок 1.1. иллюстрирует используемую Xen архитектуру сети по умолчанию. Для каждого domU Xen создает интерфейсы виртуальной сети, требуемые этим domU. Эти интерфейсы называются интерфейсами фронтенда и используются domUs для всех его сетевых коммуникаций. Кроме того, внутренние интерфейсы создаются в dom0 и соответствуют каждому интерфейсу фронтенда в domU. Чтобы обмениваться данными между бэкендом и интерфейсами фронтенда, Xen использует канал ввода/вывода, который, в свою очередь, использует механизм нулевой копии, в котором внутренние интерфейсы вступают в роли полномочий для виртуальных интерфейсов в dom0. Фронтенд и внутренние интерфейсы связаны друг с другом через канал ввода/вывода. На рисунке 1.1. внутренние интерфейсы в dom0 связаны с физическими интерфейсами, а также друг с другом через мост виртуальной сети. Эту архитектуру по умолчанию, используемую Xen, называют соединенным способом. Таким образом и канал ввода/вывода и

сетевой мост устанавливают канал связи между виртуальными интерфейсами, созданными в domUs через физический интерфейс.

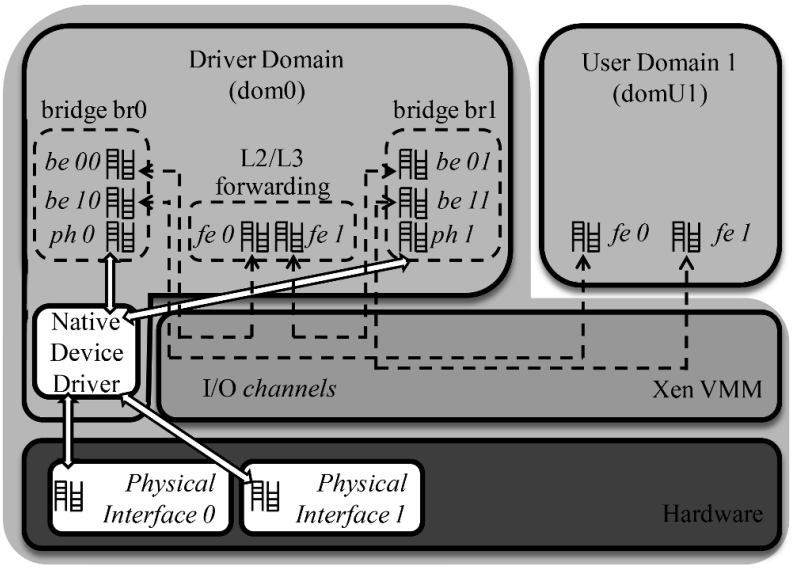


Рис. 1.1. Архитектура сети Xen

VMware - компания, которая предоставляет машинные платформы виртуализации клиентам центра обработки данных и конечному пользователю. Платформы виртуализации VMware основаны на понятии полной виртуализации. Оно оценивает уровень платформы виртуализации центра обработки данных VMware под названием VMware Сервер ESX. Изоляция VM и частота представления ресурса на основе политики распределения ресурсов установлены системным администратором. Разделяющий ресурс динамичен, потому как ресурсы могут определены и перераспределены к VMs по требованию. Архитектура VMware, как показано на рисунке 1.2. состоит из компонентов интерфейса аппаратных средств, монитора виртуальной машины (VMM), VMkernel, менеджера ресурсов и сервисного управления. Компоненты интерфейса аппаратных средств ответственны за осуществление определенных для аппаратных средств функций и создают предоставленную VMs абстракцию аппаратных средств. Это делает независимые аппаратные средства VM как VMM ответственными за центральный процессор виртуализации, предоставляя vCPU каждому VM. VMkernel управляет и следит за основными аппаратными средствами. VMM

и VMkernel вместе осуществляют слой виртуализации. Управление ресурсами осуществляется VMkernel. Он делит основные физические ресурсы между VM, перераспределяя ресурсы для каждого VM. Сервисный панель предоставляет множество услуг, таких как самонастройка, инициирование выполнения слоя виртуализации, управление ресурсов, запуск приложения, осуществляющего поддержку управления.

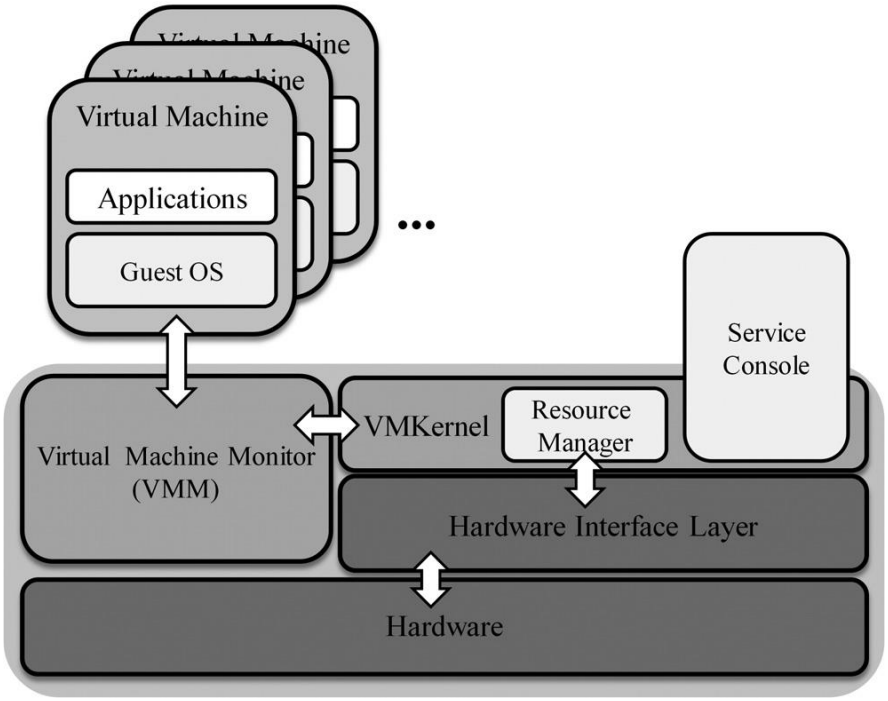


Рис. 1.2. Архитектура VMware

VMware Сервер ESX, как и другие инструменты виртуализации, виртуализирует четыре главных ресурса: центральный процессор, память, диск и сетевое устройство. Виртуализация центрального процессора осуществляется посредством установки vCPU для каждого VM. VM не понимает, что запускается через vCPU, потому что у vCPUs, должны быть свои собственные регистры и структуры контроля. У VM могут быть один или два vCPUs. Когда у него есть больше, чем один центральный процессор, он называется SMP VM. VMM ответственен за виртуализацию центрального процессора, устанавливает этапы реализации и выполняет инструкции. В виртуализированной окружающей среде гостевая ОС находится на более низком привилегированном уровне, чем тот, на котором она была запущена.

VMware объединяет два метода виртуализации центрального процессора: прямое выполнение и эмуляция центрального процессора. Инструкции от пространства пользователя VM выполнены непосредственно на физическом центральном процессоре технологией, известной как прямое выполнение. Инструкции, которые чувствительны к уровню привилегии, отбираются VMM, который следует к примеру выполнения инструкции, представляя первоначальный процесс. Объединение обоих методов предоставляет центральному процессору интенсивное пространство. Потеря исполнительных показателей зависит от количества точных инструкций, которые должны были быть заменены. Планирование центрального процессора выполняется менеджером ресурсов и основано на действиях, предоставляющих данные о том, сколько времени дано каждому VM. Планирование центрального процессора - пропорциональное действие, означающее, что время центрального процессора, данное каждому VM, пропорционально. В SMP VM распределение центрального процессора отличается. Менеджер ресурсов назначает vCPUs непосредственно на физические центральные процессоры и пытается выполнить их в то же время. Планировщик центрального процессора VMware пытается держать равновесие между распределением центрального процессора VM. Когда VM будет остановлен или перестанет работать, менеджер ресурсов назначает его время использования центральным процессором к другим VMs. Подход виртуализации памяти состоит в том, что VMware должен создать новый уровень перевода адреса памяти. Это делается путем предоставления каждому гостевому ОС виртуальной таблицы страниц с помощью Memory Management Unit (MMU). В среде виртуализации VMware гостевой ОС получает доступ к пространству виртуальной памяти, предоставленной VM. Гостевая таблица страниц ОС поддерживает последовательность между гостевой виртуальной страницей и гостевой виртуальной “физической” страницей. Гостевая виртуальная страница является страницей виртуальной памяти как в VM, так и в родной виртуальной памяти ОС. Однако, гость

виртуального механизма не может получить доступ непосредственно к физической памяти, он получает доступ к гостевой виртуальной “физической” памяти. Гостевая виртуальная “физическая” память является абстракцией физической памяти. Когда гостевая ОС пытается выполнить инструкцию и получить доступ к физической памяти, выполняется инструкция VMM, и его адрес переводится по реальному физическому адресу. Разделяющая память VMware, повинуется политике администрации, которая, например, определяет минимальную и максимальную сумму физической памяти, к которой получит доступ VM. Также можно иметь VMs, потребляющие больше, чем общая сумма физической памяти, доступной на физической машине. Это возможно, потому что хост-система может также производить обмен, как и в традиционном механизме виртуальной памяти, используемом в современном ОС - ах. Подход виртуализации ввода/вывода VMware должен подражать критическим по отношению к работе устройствам, таким как диск и сетевые карты. Доступны устройства, эмулированные VMkernel. VMkernel - ем называется слой интерфейса аппаратных средств, который ответственен за доступ к драйверу устройства и выполнение операции на физическом устройстве. Для виртуализации хранения в VM представлен драйвер Small Computer System Interface (SCSI). Относительно виртуализации сети I/O VMware осуществляет vmxnet драйвер устройства, который является абстракцией основного физического устройства. Когда приложение хочет послать данные через сеть, гостевая ОС обрабатывает запрос и вызывает vmxnet драйвер устройства. Запрос ввода/вывода получает VMM, и контроль передается VMkernel. VMkernel независим от физического устройства. Он обрабатывает запрос, справляется с различными запросами VM и вызывает уровень интерфейса аппаратных средств, которые осуществляются определенным драйвером устройства. VMware Сервер ESX собирает группы из отправки или получения сетевых пакетов прежде, чем сделать передачу контекста. Этот механизм используется только тогда, когда приоритет пакета достаточно высок, чтобы

не увеличивать задержки пакета. В заключение, Сервер ESX VMware - инструмент виртуализации, который предоставляет обширное управление и администрирование. VMware Сервер ESX сосредоточен на виртуализации центра обработки данных. Он обеспечивает гибкую и высокоэффективную центральную виртуализацию памяти. Однако ввод/вывод виртуализации все еще остается проблемной, так как она происходит на основе физических устройств и включает изменения контекста.

OpenVZ - общедоступный инструмент виртуализации уровня ОС. OpenVZ позволяет многократную изолированную окружающую среду исполнения по единственному ядру ОС. Каждую изолированную окружающую среду исполнения называют Virtual Private Server (VPS). VPS похож на физический сервер, имея его собственные процессы, пользователей, файлы, адреса Internet Protocol (IP), системную конфигурацию и обеспечивая полный доступ к корню. Главное место использования этой технологии виртуализации является веб-хостинг, где предоставляется каждому клиенту полная окружающая среда Linux, а так же она используется в образовательных учреждениях информационных технологий (IT), предоставляя каждому студенту сервер Linux, который может проверяться и управляться удаленно. OpenVZ менее гибок, чем другие инструменты виртуализации, такие как VMware или Xen, потому что окружающая среда для OpenVZ должна быть Linux дистрибутивом, на основе того же самого ядра ОС физического сервера.

Архитектура OpenVZ, как показано в рисунке 1.3. состоит из измененного ядра Linux, которое находится выше аппаратных средств. OpenVZ-измененное ядро, осуществляющее виртуализацию и изоляцию нескольких подсистем, управление ресурсом и контрольно-пропускными пунктами. Кроме того, механизмы виртуализации ввода/вывода обеспечены OpenVZ-измененным ядром, у которого есть драйвер устройства для каждого устройства ввода/вывода. Это измененное ядро также осуществляет двухуровневый планировщик процесса, который ответственен за первый

уровень, определяя какой VPS будет исполнятся, и за решение кто будет управлять процессы VPS. Двухуровневый планировщик и некоторые особенности, которые обеспечивают изоляцию между VPSs, формируют слой виртуализации OpenVZ. VPS – ы исполняются выше слоя виртуализации OpenVZ. У каждого VPS есть свой собственный набор заявлений и пакетов, которые являются сегментациями определенных распределений Linux и содержат заявления или услуги. Поэтому у VPS могут быть свои собственные услуги и заявления, независимые друг от друга.

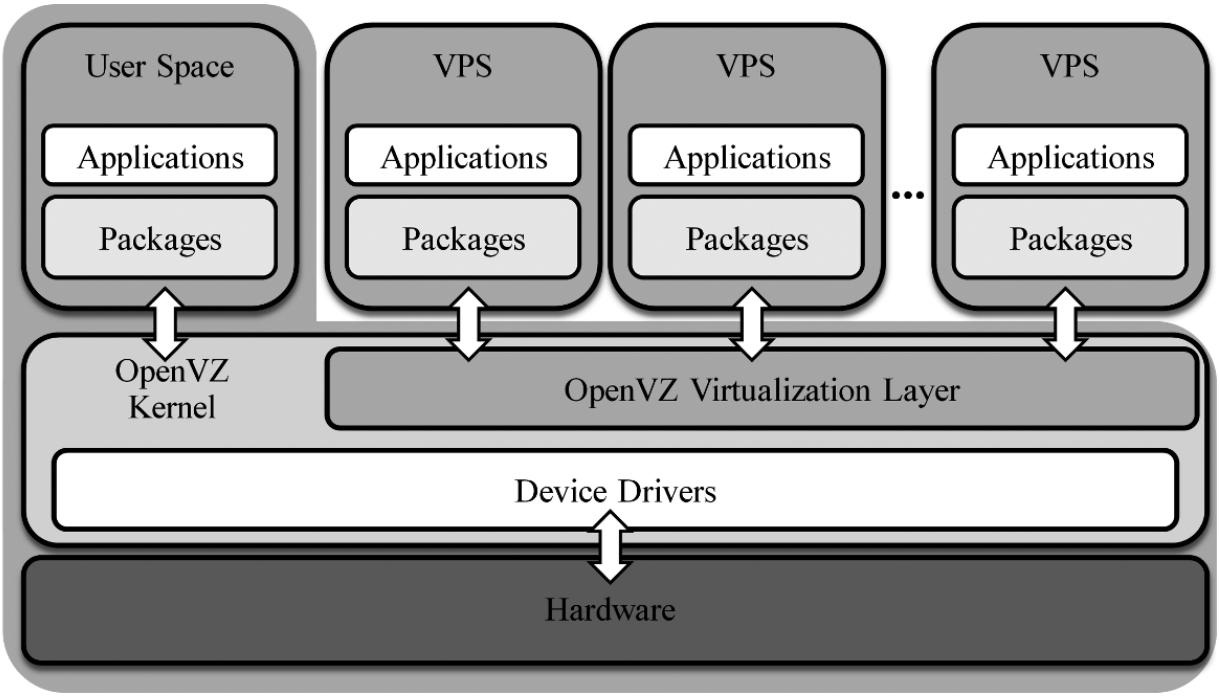


Рис. 1.3. Архитектура OpenVZ

Виртуализация ресурса в OpenVZ решает, позволив или запретив, что VPS получит доступ к ресурсу на физическом сервере. В целом ресурсы в OpenVZ не эмулированы, они разделены между VPS - ми. Определение значения каждого ресурса, который гарантируется для каждого VPS, находится в конфигурационном файле VPS. Для виртуализации процессора OpenVZ осуществляет двухуровневый планировщик центрального процессора. На первом уровне решает слой виртуализации, который VPS выполнит в течение каждого интервала времени, принимая во внимание приоритет центрального процессора VPS, измеренный в единицах CPU. На

втором уровне, который исполняется в VPS, стандартный планировщик Linux определяет, который процесс выполнит в течение каждого интервала времени, принимая во внимание стандартные приоритетные параметры процесса.

OpenVZ позволяет VPS - ам непосредственно получать доступ к памяти. Кроме того, это более гибко, чем другие технологии виртуализации. Во время выполнения VPS сумма памяти, посвященная одному VPS, может быть динамично изменена администратором. Ядро OpenVZ управляет местом в памяти VPS - ов, чтобы держать в физической памяти блок виртуальной памяти, соответствующей управлению VPS.

OpenVZ виртуальный диск является разделением файловой системы физической машины. Так же к планированию центрального процессора, дисковое использование OpenVZ определено двухуровневой дисковой квотой. На первом уровне слой виртуализации OpenVZ определяет дисковую квоту на каждый VPS, например, ограничивая максимальный размер папки в файловой системе. На втором уровне возможно определить дисковые квоты на пользователей и группы в VPS, используя стандартные механизмы квоты Linux. Сетевой слой виртуализации изолирует VPS-ы друг от друга и от физической сети. Механизм виртуализации сети по умолчанию OpenVZ создает интерфейс виртуальной сети для VPS и назначает IP-адрес на него в хост-системе. Когда пакет прибывает в хост-систему с IP-адресом назначения VPS, маршруты хост-системы пакета к соответствующему VPS. Этот подход сетевой виртуализации позволяет пакетам VPS быть полученными и посланными, используя модуль направления хост-системы. Это упрощает сетевую виртуализацию, но вводит дополнительный перевод в пакетах маршрута.

1.3. Оценка эффективности

Оценка результатов деятельности. В данной работе дан результат экспериментов с VM, который сравнивает работу, полученную с Xen, VMware и OpenVZ против родного Linux. Мы используем определенные оценки для центрального процессора, RAM, жесткого диска и сетевых ресурсов.

Производительность ЦП Тест Суперпи - интенсивная центральным процессором задача на основе алгоритма Гаусса-Лежандра, чтобы вычислить стоимость Пи. Алгоритм Гаусса-Лежандра повторяющийся и основанный на многих арифметических операциях. Для этого теста скрипт оболочки вычисляет стоимость Пи с 222 цифрами (4,194,304 цифры) для десяти раундов. Исполнительная метрика - время, потраченное, чтобы вычислить Пи в секундах.

Работа памяти. Распределение памяти, Набор и Рид (MASR) являются инструментом сопоставительного анализа памяти, разработанным Teleinformatic and Automation Group (GTA) из Федерального университета Рио-де-Жанейро. MASR определяет эффективность памяти, предназначая 2 ГБ памяти гипервизору, последовательно устанавливая все значения памяти в постоянное значение и последовательно читая все данные из памяти. MASR был развит для сопоставительного анализа памяти с детерминированным количеством операций, независимых от работы компьютера. Так как никакие инструменты сопоставительного анализа памяти Linux не были найдены с этой явной особенностью, был создан MASR. Для этого теста был разработан скрипт оболочки, в котором MASR выполняется за 10 шагов. Оцененный параметр требует время, чтобы выполнить критерий MASR в каждом раунде.

Бонни ++. Бонни ++ является общедоступной дисковой утилитой, разработанной, чтобы оценить работу файловой системы и жесткий диск. Главная программа Бонни ++ проверяет в 20 - ти виртуальных сетях один временный файл в 1 ГБ или много временных файлов общим объемом в 1 ГБ для больших объемов данных, с доступом в базы данных, моделируя операции такие, как создание, чтение и удаление больших и маленьких

временных файлов. Вторая программа проверяет исполнение различных областей жесткого диска, читая данные из блоков, расположенных в начале, в середине и в секторах конца жесткого диска. Для этого теста скрипт оболочки приостанавливает Бонни ++ за 10 раундов с параметром для использования 2 ГБ дискового пространства. Исполнительная метрика - время, потраченное, чтобы приостановить Бонни ++, определяя каждый раз эффективность.

ZFG Zero File Generator. ZFG является дисковым инструментом сопоставительного анализа, разработанным GTA/UFRJ. ZFG определяет эффективность диска непрерывной скоростью записи, создавая бинарный файл на 2 ГБ, заполненный нулями 10 раз в каждом раунде. ZFG был разработан для сопоставительного анализа диска с детерминированным количеством операций, независимый от работы компьютера, так как с таким явным признаком никакие дисковые инструменты сопоставительного анализа Linux не были найдены. Для этого теста был развит скрипт оболочки, в котором ZFG выполнен за 10 раундов. Оцененный параметр - время, потраченное, чтобы выполнить оценку ZFG на каждом раунде.

Производительность сети. Iperf - общедоступная сетевая оценка, которая позволяет определять эффективность сетей, используя и однонаправленные и двунаправленные потоки данных по протоколу Transmission Control Protocol (TCP) или User Datagram Protocol (UDP). У Iperf есть несколько параметров, которые могут формироваться, такие как размер пакета, пропускная способность и т.д. В данном тесте использовано Iperf, формируемый с однонаправленными потоками данных UDP, используя 1,472 байта полезного груза, чтобы избежать IP фрагментации. Для этого теста был развит скрипт оболочки, в котором однонаправленный поток данных UDP идет или с испытательного компьютера на компьютер тестирования сети или в противоположном направлении.

Суперпи. Результаты для оценки Суперпи, выполненной за 10 циклов, показывают в рисунке 1.4. В среднем время выполнения на раунда теста

показывает вертикальная ось, наименьший показатель наилучший. Инструменты виртуализации и пользовательские Linux распределены вдоль горизонтальной оси.

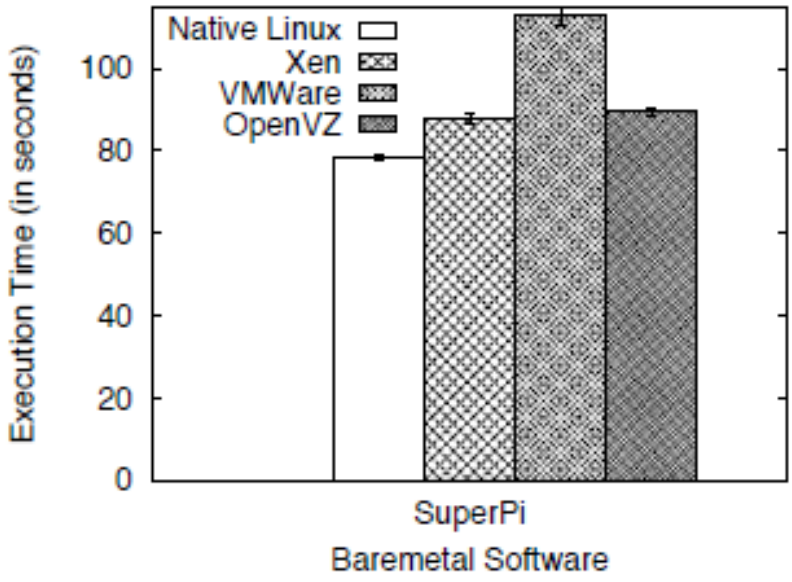


Рисунок 1.4. Тест суперпи

Как ожидалось невиртуальная система выигрывает у виртуальной системы. Это было ожидаемо, так как паравиртуализированный VM Xen может непосредственно получить доступ к памяти после одобрения гиперщитка для использования области памяти. У VMware худший показатель, последствие дополнительного перевода адреса между виртуальной предполагаемой RAM к VM и физической RAM, и также к планировщику центрального процессора VM. У OpenVZ немного выше показатели, чем Xen, из-за механизма разделения планирования между контейнерами центрального процессора.

MASR. Результаты для оценки MASR, выполненной за 10 циклов, показаны на рисунке 1.5. Среднее время выполнения для цикла теста показано на вертикальной оси, наименьшие показатели лучше. Инструменты виртуализации и Linux распространены вдоль горизонтальной оси.

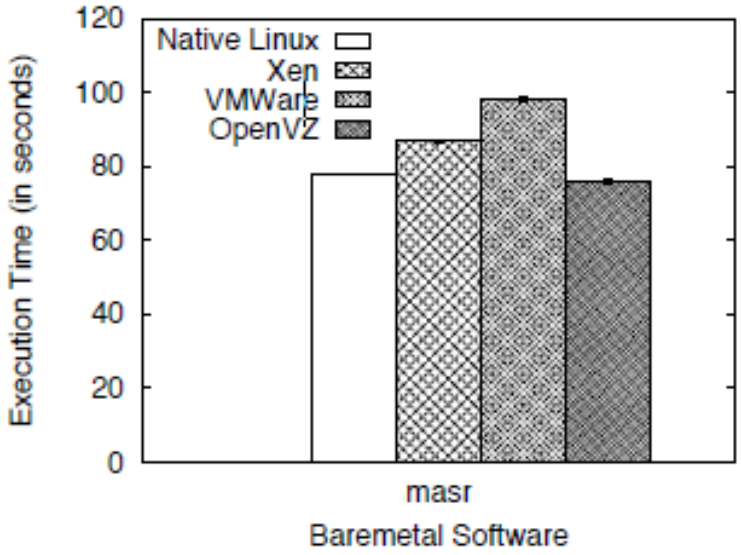


Рисунок 1.5. Тест MASR

Результаты показывают, что у Linux – а, развёрнутого над физической оболочкой, лучшие показатели в работе как и ожидалось. OpenVZ достигает подобными показателями работы к Linux, так как OpenVZ делит память динамично со всеми контейнерами с низким уровнем изоляции, который только гарантирует минимальную сумму частной памяти для каждого контейнера. Показатели Xen чуть лучше, чем VMware. VMware худше работает из-за дополнительного перевода адреса памяти и планировщика VM. Все инструменты виртуализации представили приемлемый пороговый показатель, который является очень важным результатом, так как операции по доступу памяти очень распространены в отправлении поиска по таблице и других задач направления.

Бонни ++. Результаты для Бонни ++ дисковая оценка, выполненная за 10 циклов, показаны на рисунке 1.6. Среднее время выполнения для цикла теста показано на вертикальной оси, наименьшие показатели лучше. Инструменты виртуализации и Linux распространены вдоль горизонтальной оси.

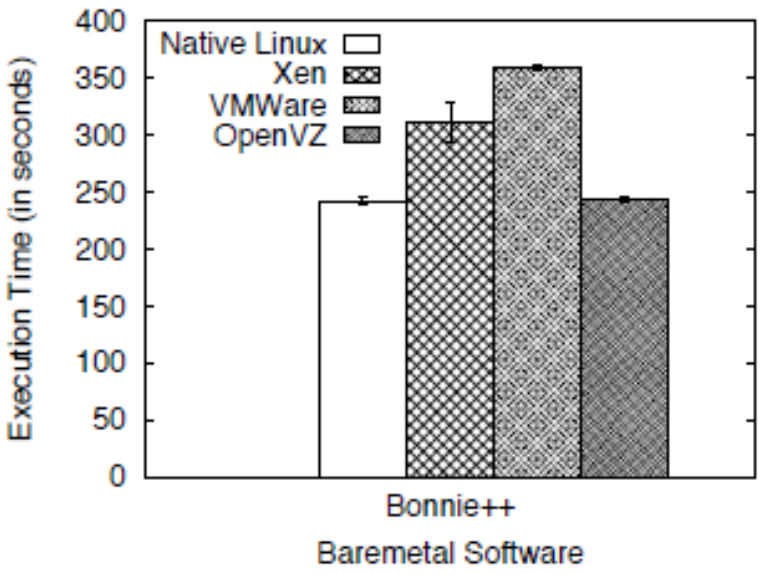


Рисунок 1.6. Бонни ++ тест

У Linux – а наилучший показатель, и OpenVZ достигает подобных результатов, так как дисковый доступ в OpenVZ очень похож на дисковый доступ в Linux. Xen представил некоторые задержки, поскольку он использует дополнительный буфер между dom0 и VM, уровень ввода / вывода, которое добавляет латентность при чтении с диска. VMware представляет худшие результаты, вероятно впоследствии дополнительных буферов в гиперщитке, которые используются, чтобы передать данные между физическим диском и виртуальным диском. Тем не менее, дисковая работа имеет незначительное значение, так как это не влияет на большую долю операций по маршрутизатору.

ZFG. Результаты для теста ZFG показаны на рисунке 1.7. VMware, Linux и результаты OpenVZ были получены из 10 ступеней. Результаты Xen были получены из 100 ступеней, чтобы иметь приемлемое значение погрешности. В среднем время выполнения на раунда теста показывает вертикальная ось, наименьший показатель наилучший. Инструменты виртуализации и пользовательские Linux распределены вдоль горизонтальной оси.

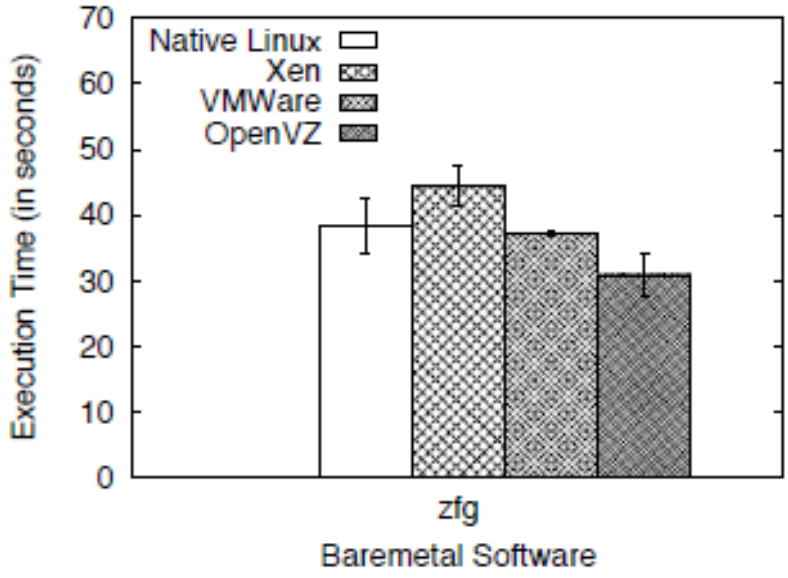


Рисунок 1.7. Тест ZFG

Как наблюдается на рисунке 1.7, VMware VM проявил немного лучше результат, чем Linux, и мы зачитываем это к буферу, который передает данные между виртуальным диском и реальным диском так, как это производит VM, представляя, что данные были уже записаны на диск, когда это на самом деле передается от буфера к физическому диску. OpenVZ неожиданно выиграл, показав наилучший результат. Xen VM показал наихудший результат.

Iperf: поток данных UDP с VM на компьютер. Тестирования сети. В этом тесте определяется эффективность пакета UDP от VM. Результаты для эталонного трафика генерирующего сети Iperf с VM на компьютер тестирования для 100 раундов показаны на рисунке 1.8. Средняя пропускная способность показана на вертикальной оси, наибольшие показатели лучше. Инструменты виртуализации и Linux распространены вдоль горизонтальной оси.

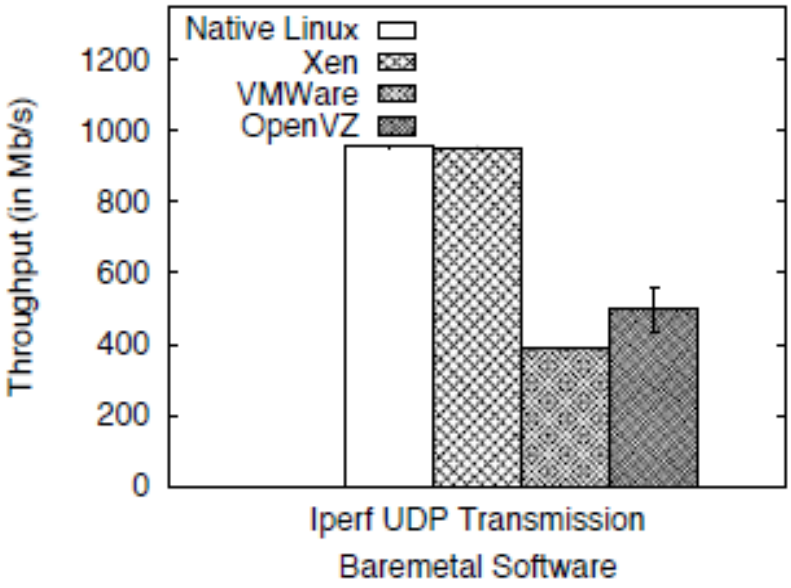


Рисунок 1.8. Тест Iperf пакета UDP

Как ожидалось Linux на физическом устройстве достигает лучших результатов около теоретического предела гигабита адаптера Ethernet. Xen достигает почти такой же результат работы, показывающей, что механизм ввода/вывода способен поставлять данные достаточно быстро. VMware представляет большую исполнительную деградацию, которая указывает на неэффективное внедрение ее дефолта сетевым механизмом. Сетевой механизм OpenVZ по дефолту на основе IP виртуализации слоя выступил плохо и не мог использовать полную пропускную способность интерфейса гигабита. Это - очень важный результат, так как пропускная способность маршрутизатора непосредственно зависит от его способности отправки пакетов, и результаты показывают, что через виртуализированную среду возможно послать большие пакеты на скорости почти равной скорости физического адаптера.

Iperf: поток данных UDP от компьютера тестирования сети до VM. В этом тесте определяется эффективность приема пакета UDP от VM. Результаты для эталонного трафика генерирующего сети Iperf от Компьютера Тестирования сети до VM для 100 раундов показываются на

рисунке 1.9. Средняя пропускная способность показана на вертикальной оси, наибольшие показатели лучше. Инструменты виртуализации и Linux распространены вдоль горизонтальной оси.

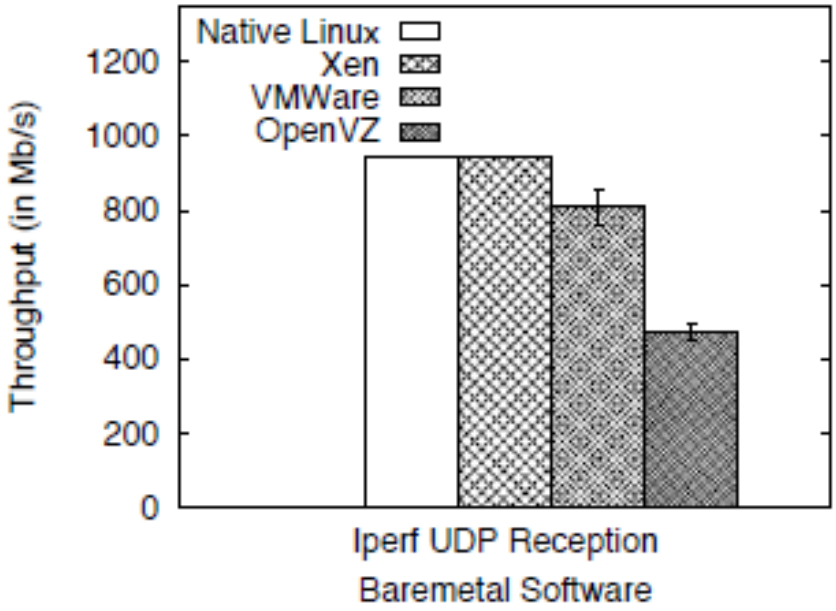


Рисунок 1.9. Тест Iperf UDP

Результаты показывают еще раз, что Linux близок к теоретической максимальной пропускной способности. Xen еще раз достигает почти физическим показателям. VMware выступил намного лучше в приеме, чем в передаче, но все еще хуже, чем Linux и Xen. Сетевой механизм виртуализации OpenVZ по дефолту проявил плохой результат. Результаты показывают, что Xen может обращаться с приемом больших пакетов в почти физической скорости.

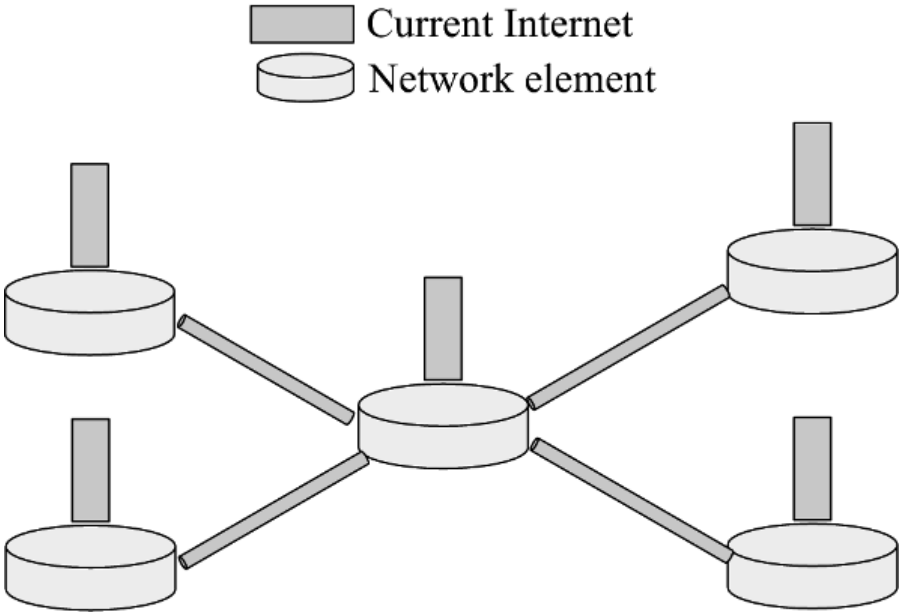
II ГЛАВА. ИНТЕРФЕЙСЫ ВИРТУАЛЬНОЙ СЕТИ

2.1. Виртуальные сети: производительность и тенденции

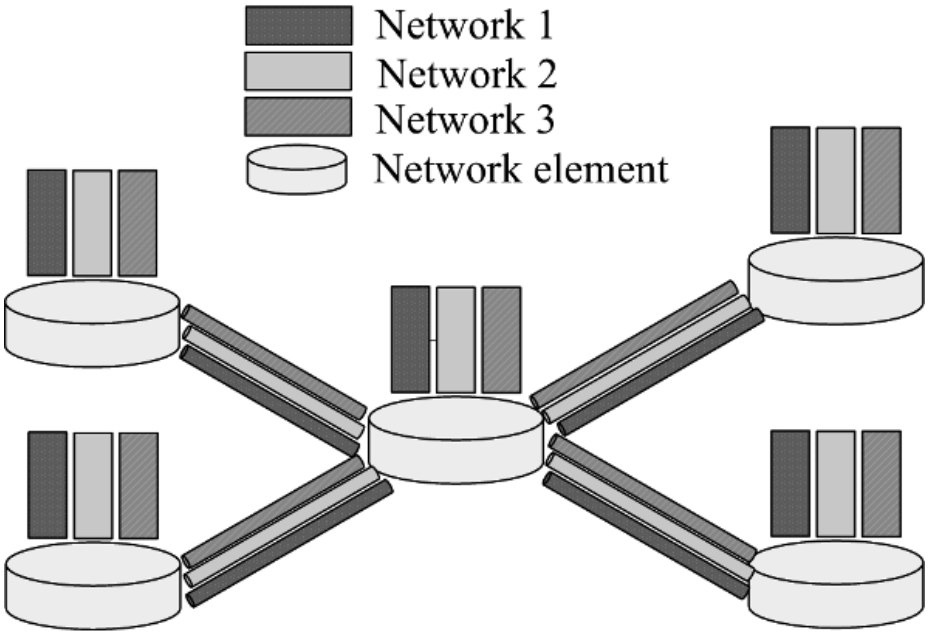
Виртуальные сетевые интерфейсы

Будущие предложения по архитектуре Интернета можно разделить на две основные модели – монистическую и плюралистическую. В монистической модели, показанной на рис. 2.1a), сеть имеет монолитную архитектуру, которая должна быть достаточно гибкой для поддержки всех видов приложений. При таком подходе только один стек протокола проходит по физической схеме за один раз. С другой стороны, плюралистический подход, показанный на рисунке 2.1b), основан на той идее, что Интернет должен одновременно поддерживать несколько сетей, использующих стек протоколов, которые соответствуют потребностям конкретного приложения. Создание специализированных сетей, предназначенных для конкретных служб, упрощает развертывание новых приложений, требующих выполнения таких функций, как безопасность, мобильность или качество обслуживания. В диссертационной работе раскрывается подход, который утверждает, что разработка нескольких сетей для обеспечения разных услуг происходит проще, чем проектирование уникальной сети для обработки всех услуг одновременно. Таким образом, плюралистический подход может быть определен как «зонтик», основанный на подходе «разделяй и властвуй». Если очень сложно найти единственное решение для покрытия всех возможных требований, такая жизнеспособная альтернатива может быть создана с несколькими различными сетями для поддержки уровней. Эта характеристика является главным аргументом для плюралистической архитектуры, потому что монистический подход не только должен решить все известные Интернет-проблемы, но также должен развиваться, чтобы постоянно функционировать в будущем. Кроме того, еще одним ключевым преимуществом модели плюрализма является ее внутренняя обратная

совместимость, поскольку в текущем режиме в Интернете должна быть поддержка сетей, работающих параллельно.



a) Монистическая модель



b) Плюралистическая модель

Рисунок 2.1. Сетевые архитектуры: монистическая модель с одним стек-протоколом и плюралистическая модель с несколькими стеками протоколов

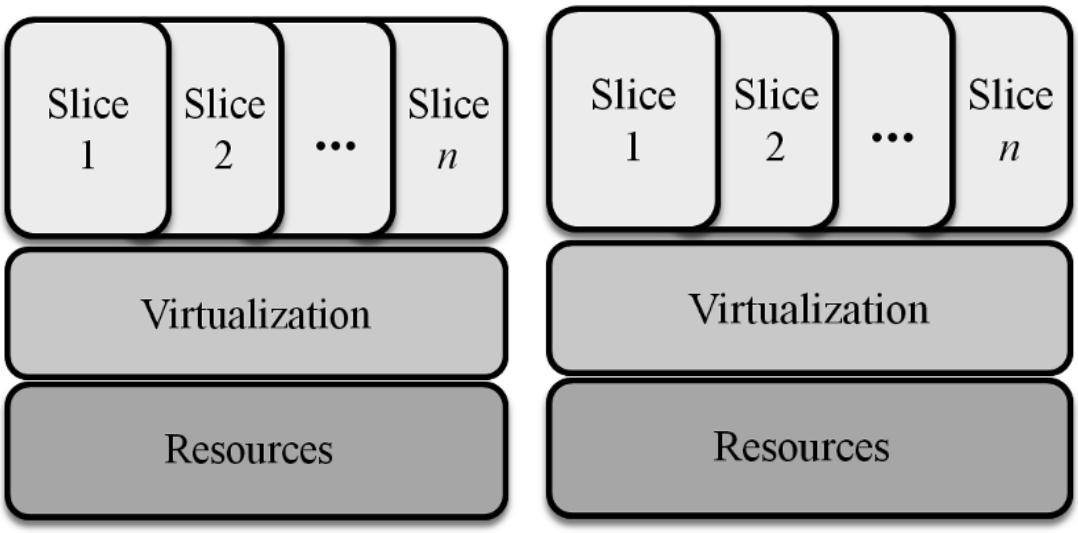
Плюралистическая модель, основанная на идее множественных сетей, использует одну и ту же инфраструктуру, причем эти сети могут различаться по форматам пакетов, схемам адресации и протоколам. Поскольку данные сети используют один и тот же физический сетевой маршрутизатор, множественный доступ к базовому средству должен быть организован монитором. Этот монитор является частью специального программного обеспечения, который виртуализирует общий носитель в несколько сетей. Следовательно, каждая сеть работает так, как если бы она использовала заданное количество физических ресурсов. Эти сети называются виртуальными сетями и решают проблему производительности в результате использования общего физического подхода. Это становится дополнительной задачей – появляются накладные расходы плюралистической архитектуры по сравнению с монистической, поскольку добавляется программный уровень. Основываясь на плюралистическом подходе, предыдущие работы, такие как исследования, проведенные командой проекта Horizon, предлагают использовать две различные платформы виртуализации для обеспечения виртуальной сети: Xen и OpenFlow. Эти платформы решают проблему разделения физических ресурсов между несколькими виртуальными сетями. В данном разделе описывается проблема совместного использования сетевого субстрата среди различных виртуальных сетей. В диссертационной работе проведен анализ двух типичных подходов к виртуализации физических сетей – Xen и OpenFlow, а также способы использования этих технологий для виртуальных сетей. Основная цель - раскрытие плюсов и минусов сетевой виртуализации, используемой в качестве основы плюралистической архитектуры для будущего Интернета. С этой целью проведены эксперименты для оценки Xen и производительности OpenFlow, выступающих в роли виртуализированного программного маршрутизатора. Основываясь на представленных результатах, в диссертационной работе обнаружен компромисс между гибкостью и эффективностью, что указывает на то, что использование

данных в общей плоскости может быть важным архитектурным выбором при разработке виртуальных сетей. Другим ключевым моментом является то, что, используя общие плоскости данных, Xen и OpenFlow могут мультиплексировать несколько виртуальных сетей без каких-либо измеримых потерь производительности, по сравнению со сценарием, в котором такая же скорость передачи пакетов обрабатывается одним элементом виртуальной сети.

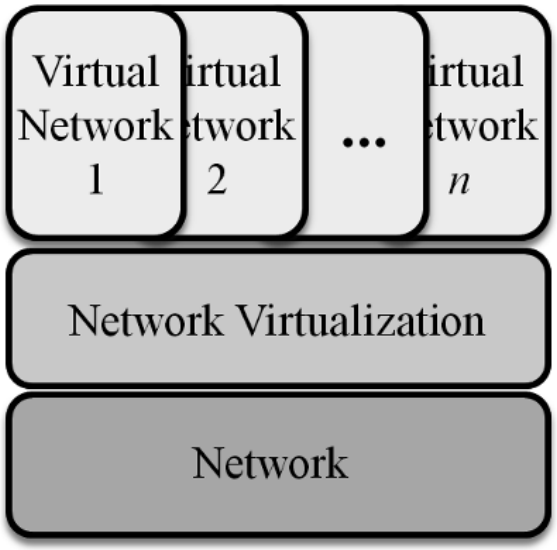
Подходы к виртуализации сети

В диссертационной работе виртуализация рассматривается как абстракция ресурса, которая позволяет разделять физические ресурсы на несколько виртуальных ресурсов того же типа, но с разными возможностями, как это показано на рисунке 2.2. Эта абстракция часто реализуема программным слоем, который обеспечивает «виртуальные разделенные интерфейсы», аналогичные реальным интерфейсам. Сосуществование нескольких виртуальных разделов над тем же ресурсом возможен, потому что уровень виртуализации отделяет реальный ресурс и вышеописанный уровень. На рис. 2.2 показаны два примера виртуализации: виртуализация компьютеров и виртуализация сети. Абстракция виртуализации компьютера реализуется монитором виртуальной машинный (VMM), который предоставляет виртуальным машинам (VM) интерфейс (то есть уровень абстрагирования аппаратного обеспечения), весьма похожий на компьютерно-аппаратный интерфейс. Этот интерфейс включает в себя процессор, память, вход/выход (I / O) и т.д. Таким образом, каждая виртуальная машина создает впечатление, что она работает непосредственно над физическим оборудованием, но на самом деле физическое оборудование – это несколько виртуальных машин. Это разделение ресурсов мы называем распределением, потому что виртуальные машины изолированы: одна виртуальная машина не может вмешиваться в работу другой. Виртуализация компьютеров широко используется в дата-центрах для запуска нескольких серверов на одной физической машине.

Помимо экономии энергии и снижения затрат на техническое обслуживание, самой важной особенностью этого метода является его гибкость, позволяющая каждой виртуальной машине иметь свои собственные операционные системы, приложения, правила конфигурации и административных процедур. Возможность запуска настраиваемого стека протоколов в каждой виртуальной доле является основной мотивацией применения виртуализации для сетей. Как показано на рисунке 2.2, виртуализация сети аналогично виртуализации компьютеров, в которой общим ресурсом является сеть. Эта концепция не нова, и она была использована в виртуальных частных сетях (VPN) и в виртуальных локальных сетях (VLAN). В настоящее время, однако, существуют новые методы, позволяющие даже виртуализацию маршрутизатора. В этом случае каждая виртуальная часть маршрутизатора может реализовать настраиваемый стек сетевых протоколов.



a) Общая виртуализация b) Виртуализация компьютеров



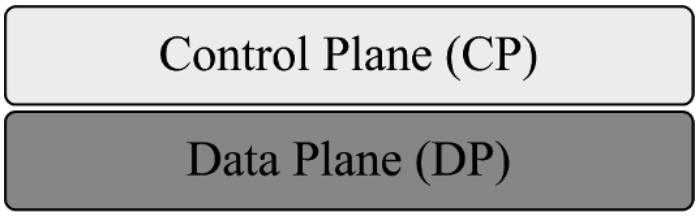
c) Сетевая виртуализация

Рисунок 2.2. Виртуализация: «разделенные» ресурсы, виртуальные машины и виртуальные маршрутизаторы

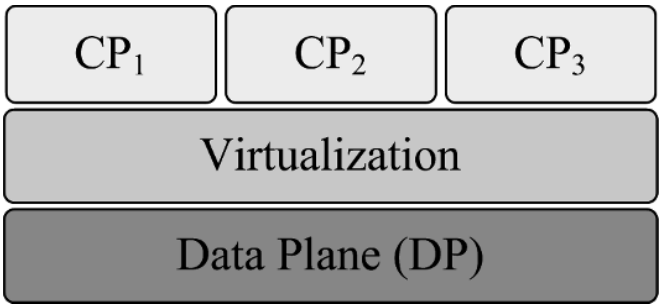
Подходы к сопоставлению виртуализации сети на уровне абстракции предоставляют, что можно проиллюстрировать положение слоя виртуализации в архитектурной модели. На рисунке 2.3 сравниваются два основных подхода для виртуализации элемента сети. На рис. 2.3 a) показана обычная архитектура сетевого элемента с единой панелью управления и данными. В маршрутизаторе плоскость управления отвечает за запуск программного обеспечения сетевого управления, таких как протоколы маршрутизации (например, Routing Information Protocol (RIP), Open Shortest Path First (OSPF) и Border Gateway Protocol (BGP)) и протоколы управления сети (например, Internet Control Message Protocol (ICMP)), тогда как плоскость данных отвечает за реализацию таблиц пересылки и аппаратные каналы передачи данных. Виртуализация процедуры маршрутизации означает вставку виртуализации на некоторый уровень архитектуры сетевых элементов, позволяющий сосуществование нескольких виртуальных сетевых элементов над одним физическим сетевым элементом. Предполагается наличие уровня виртуализации между контролем и плоскостью данных, где система управления виртуальна, как это показано на рисунке 2.3 b). В этом

случае плоскость данных используется всеми виртуальными сетями, и каждая виртуальная сеть запускает собственное программное обеспечение управления. По сравнению с традиционной архитектурой сети, этот подход значительно улучшает сетевую программируемость, поскольку он позволяет запускать несколько и настраиваемых протоколов вместо одного стека протокола по умолчанию. Например, можно программировать разные стеки протоколов для сети 1, сети 2 и сети 3, как показано на рис. 2.3 b). Во втором подходе сетевой виртуализации панели управления и данные виртуализированы, как показано на рисунке 2.3 c). В этом случае, каждый элемент виртуальной сети реализует свою собственную плоскость данных рядом с управляющей плоскостью, улучшая тем самым программируемость сети. Этот подход позволяет настраивать плоскости данных за счет пересылки пакетов, поскольку плоскость данных больше не предназначена для общих задач.

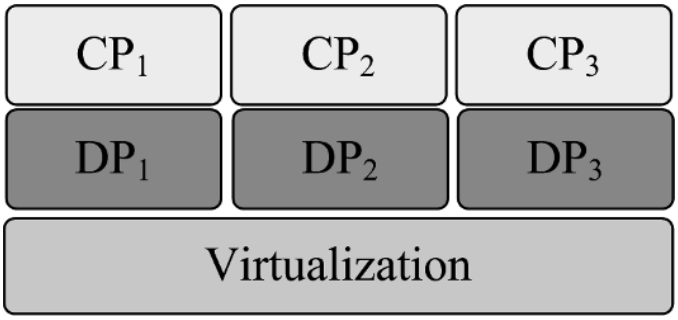
Стоит отметить, что только виртуализацией панели управления можно определить большее количество подходов в зависимости от уровня изоляции плоскости данных, разделяемых между виртуальной сетью и ее элементами. Если требуется сильная изоляция, то каждая виртуальная плоскость должна иметь доступ только к ее разделу плоскости данных и не может вмешиваться в другие. С другой стороны, если вся плоскость данных распределяется между виртуальными управляющими плоскостями, тогда возможно, что виртуальная панель управления будет интерферировать с другими виртуальными плоскостями управления. Например, одна виртуальная контрольная плоскость может заполнять всю таблицу пересылки своими собственными записями, что может привести к тому, что пакеты других виртуальных машин могут быть утеряны.



a) Обычная модель



b) Плюралистическая модель только с виртуализированной моделью виртуальной машины



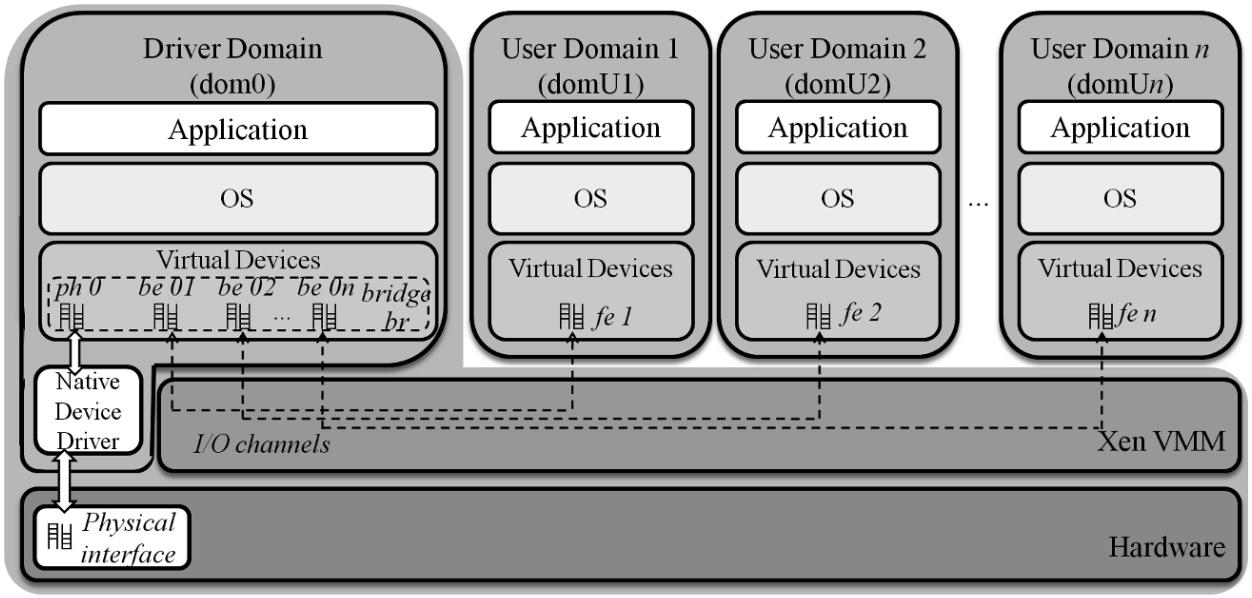
c) Плюралистическая модель с виртуализированной панелью управления и дата планами

Рисунок 2.3. Сетевые модели

Технологии сетевой виртуализации

В этом разделе мы подробно рассмотрим две технологии, которые могут быть использованы для виртуализация сети: Xen и OpenFlow. Xen - это VMM (монитор виртуальных машин) с открытым исходным кодом, также называемый гипервизором, который работает на основной аппаратной платформе. Архитектура Xen состоит из одного VMM, расположенного над физическим оборудованием и несколькими доменами, одновременно работая над гипервизором, называемым виртуальной машиной, как показано на рисунке

2.4. Каждая виртуальная машина может иметь свою собственную операционную систему и приложения. VMM контролирует доступ нескольких доменов к оборудованию, а также управляет распределением ресурсов между этими доменами. Следовательно, одной из основных задач VMM является изолирование различных виртуальных машин, то есть работа одного VM не должна влиять на производительность других. Кроме того, все драйвера устройств хранятся в изолированном домене драйверов, называемого Domain 0 (dom0) для обеспечения надежной и эффективной аппаратной поддержки. Домен 0 имеет особые привилегии по сравнению с другими доменами, называемыми непривилегированными доменами (domUs), поскольку он имеет полный доступ к физическому машинному оборудованию. Непривилегированные домены, также называемые пользовательскими доменами, имеют виртуальные драйвера и работают так, как если бы они могли напрямую обращаться к оборудованию. Тем не менее, эти виртуальные драйвера взаимодействуют с dom0, чтобы иметь доступ к физическим оборудованиям.



fe – front-end interface be – back-end interface ph – physical interface

Рисунок 2.4. Архитектура Xen: монитор виртуальной машины (VMM), домен драйвера (dom0) и доменов пользователей (domU)

Xen виртуализирует один физический сетевой интерфейс посредством демультиплексирования входящих пакетов от физического интерфейса к domUs и, наоборот, мультиплексирование исходящих пакетов, генерируемых этими domU. Эта процедура – так называемая сетевая виртуализация ввода-вывода – работает следующим образом. Домен 0 напрямую получает доступ к устройствам ввода-вывода, используя свои собственные драйвера устройств, а также выполняет операции ввода-вывода от имени domUs. С другой стороны, domUs используют виртуальные устройства ввода/вывода, управляемые виртуальными драйверами, для запроса dom0 доступа устройств, как показано на рисунке 2.4. Каждый domU имеет свой собственный виртуальный сетевой интерфейс, называемый внешним интерфейсом, требуемым для коммуникационной сети. Внутренние интерфейсы создаются в dom0, корреспондируют с каждым внешним интерфейсом в domU и действуют как прокси для виртуальных интерфейсов в dom0. Внутренние и внешние интерфейсы подключены к друг другу через канал ввода/вывода, который использует механизм нулевой копии для переназначения физической страницы, содержащий пакет в целевом домене. Таким образом, пакеты обмениваются между back-end и front-end интерфейсами. Внешние интерфейсы воспринимаются системой, работающих на domU в качестве реальных интерфейсов. Тем не менее, интерфейсы в dom0 подключены к физическому интерфейсу, а также к каждому через виртуальный сетевой мост. Это – стандартная архитектура, используемая Xen, и она называется «мостовым режимом». Таким образом, как канал ввода/вывода, так и сетевой мост устанавливают канал связи между виртуальными интерфейсами, созданными в domUs, и физическим интерфейсом. Различные элементы виртуальной сети могут быть реализованы с использованием Xen, поскольку эта технология позволяет

нескольким виртуальным машинам работать одновременно на одном и том же оборудовании, как показано на рисунке 2.5. В этом случае каждая виртуальная машина запускает виртуальный маршрутизатор, который имеет собственные панели управления и данных, поскольку уровень виртуализации Xen низкоуровневый.

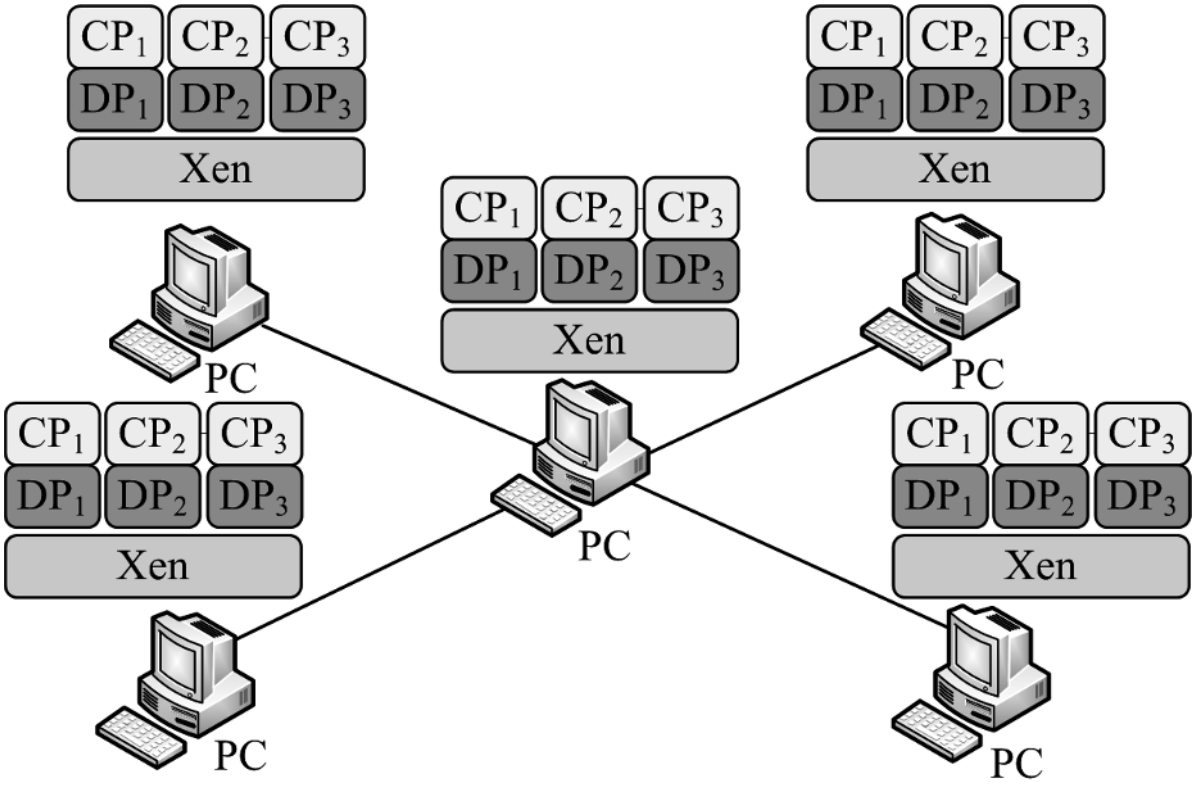


Рисунок 2.5. Виртуальная сеть Xen: одна панель данных и управления на виртуальный маршрутизатор

OpenFlow позволяет использование коммутационных помещений в кампусе университета не только для производственной сети, но и для экспериментальных сетей. Проект OpenFlow, предложенный Стэнфордским университетом, направлен на создание виртуальных сред для инноваций параллельно с производством сети с использованием сетевых элементов, таких как коммутаторы, маршрутизаторы, точки доступа и персональные компьютеры. OpenFlow представляет новую архитектуру сред для

виртуальной сети. Ключевой идеей является физическое разделение управления и дата планов. Поэтому различные сетевые элементы выполняют пересылку пакетов (панель данных) и процедуру управления сетью (панель управления).

Виртуализация элементов пересылки осуществляется посредством совместной таблицы потоков, которая представляет собой панель данных, тогда как все плоскости управления централизованы в узле, называемом контроллером, запуска приложений, которые управляют каждой виртуальной сетью. Пример работы OpenFlow показан на рисунке 2.6.

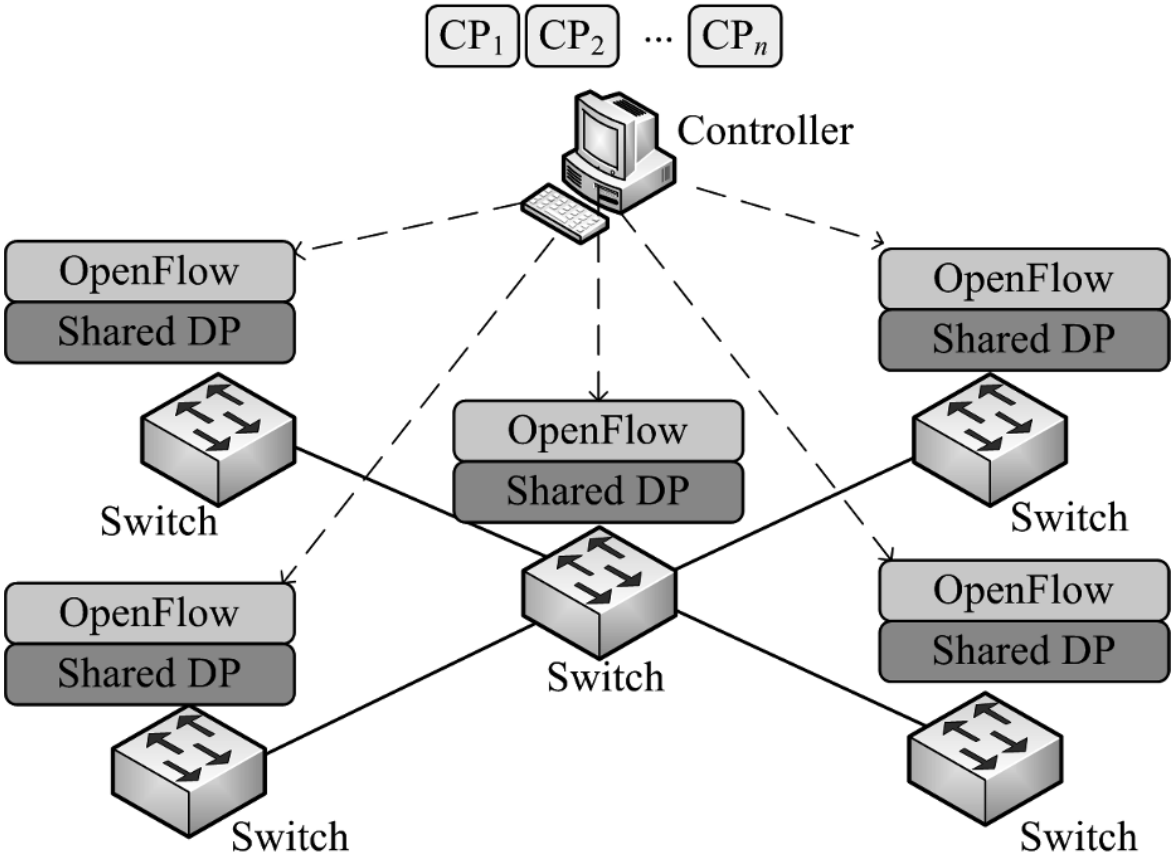


Рисунок 2.6. Виртуальная сеть OpenFlow: общая панель данных для каждого узла и все плоскости управления в одном узле

Протокол OpenFlow определяет связь между пересылкой узлов и сетевого контроллера. Он основан на создании безопасного канала между каждым узлом пересылки и контроллером, который использует этот канал

для мониторинга и настройки узлов пересылки. Каждый раз, когда новый пакет достигает элемента переадресации, первые биты пакета пересылаются в контроллер, который устанавливает путь для этого пакета в выбранных элементах пересылки. Контроллер также может устанавливать действия нормальной обработки для потока, подлежащего пересылке в соответствии с обычной маршрутизацией уровня 2 (L2) и уровня 3 (L3), как если бы OpenFlow не существовал. Именно по этой причине OpenFlow можно использовать параллельно с производственной сетью, не затрагивая текущий трафик. Панель данных в OpenFlow – это таблица потоков, описываемая полями заголовка, счетчиков и действий. Плоскость заголовка представляют собой структуру из 12 частей, которая описывает заголовок пакета, как показано на рисунке 2.7. Эти плоскости определяют поток, задавая значение для каждой плоскости или используя шаблон для установки только подмножества плоскостей. Таблица потоков также поддерживает использование масок подсети, если оборудование также поддерживает это. Эта структура из 12 частей дает высокую гибкость для пересылки пакетов, поскольку поток может быть перенаправлен на основе не только по адресу IP-адреса получателя, как в обычном протоколе управления передачей (TCP)/IP, но также по порту TCP, MAC-адресс и т. д. Поскольку потоки могут быть установлены на основе адресов уровня 2, элементы пересылки OpenFlow также называются коммутаторами OpenFlow. Это, однако, не обязательно подразумевает пакет пересылки в слое-2. Одной из будущих целей OpenFlow является обработка пользовательских полей заголовка, что означает, что заголовок пакета не будет описан фиксированными полями, но комбинацией полей, заданных администраторами виртуальных сетей. Это даст OpenFlow возможность пересылки пакетов, принадлежащих сетям, использующих любой стек протокола.

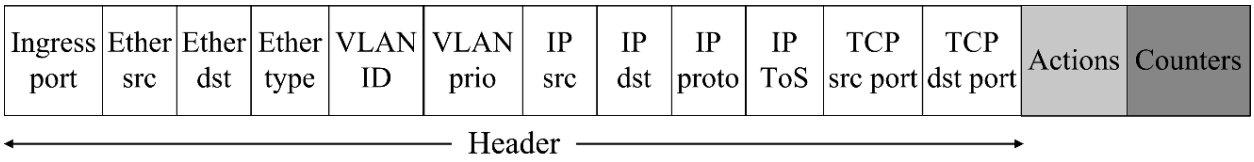


Рисунок 2.7. Запись потока в сети OpenFlow

После полей заголовка в описании потока следуют счетчики, которые используются для контроля узлов. Счетчики вычисляют данные, такие как длительность потока и количество пересылаемых байтов. Последние поля в описании потока действия, которые представляют собой набор инструкций, могут выполняться по каждому пакету определенного потока в пересылаемых элементах. Эти действия включают не только переключение входящего пакета на порт назначения, но также изменение заголовков, таких как идентификатор VLAN и адреса, и источника, и получателя. Контрольный узел является центральным элементом сети и может взаимодействовать со всеми узлами для настройки своих таблиц потоков. Контроллер запускает сетевую операционную систему, которая обеспечивает управление виртуальной сетью при помощи основных функций для настройки сети. Следовательно, контроллер в OpenFlow работает как интерфейс между сетью приложений и передачей элементов, обеспечивая основные функции доступа к пакетам из потока и для мониторинга узлов. OpenFlow работает с любым контроллером, совместимым с протоколом OpenFlow, например, такой как сеть операционной системы (NOX). В этом случае каждая плоскость управления состоит из набора приложений, работающих над NOX. Следовательно, виртуальная сеть в OpenFlow определяется своей плоскостью управления, которая представляет собой набор приложений, работающих над контроллером, и соответствующих им потоков как показано на рисунке 2.8.

Используя модель одного контроллера, можно создать множество виртуальных сетей. Обратим внимание на то, что разные приложения, работающие на одном и том же ОС, не изолированы. В результате, если в одном приложении есть ошибка, она может остановить контроллер, повредив

все другие виртуальные сети. FlowVisor – это инструмент, используемый с OpenFlow, чтобы позволить различным контроллерам работать на той же физической сети. FlowVisor работает как прокси-сервер между передающимися элементами и контроллерами, предполагая, например, один контроллер на сеть. Используя эту модель, можно гарантировать, что сбои в одной виртуальной сети не влияют на другие.

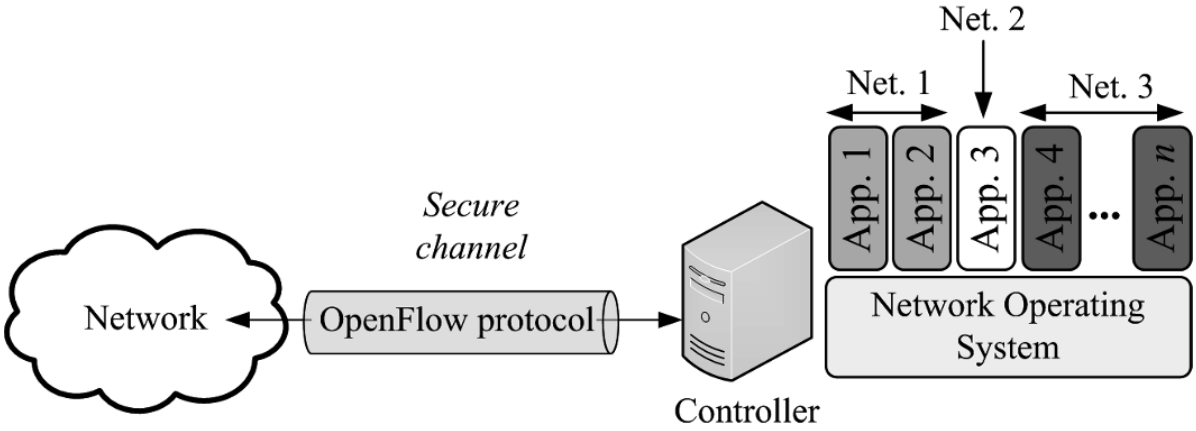


Рисунок 2.8. Централизованная модель контроллера OpenFlow

OpenFlow предоставляет гибкую инфраструктуру, основанную на идее распределенных элементов пересылки, предлагающих базовые функции для сети и централизованных панелей управления. Используя эту инфраструктуру, можно разрезать физическую сеть на несколько виртуальных сетей. В OpenFlow, создание сети – это просто создание некоторого набора приложений в контроллере. Новые сетевые потоки будут созданы в соответствии с пакетами, входящими в сеть. OpenFlow также предоставляет гибкую инфраструктуру для перераспределения сетевых ресурсов. Перераспределение сети в OpenFlow означает только перепрограммирование таблицы потоков каждого участвующего узла. Это простая операция для контроллера, потому что он знает, где находятся физические устройства и как они связаны.

2.2. Xen прототип

Прототип Xen разработан в лаборатории Grupo de Teleinformática e Automação (GTA) для экспериментов с новыми интерфейсами. В этом разделе диссертационной работы описываются интерфейсы, связанные с Xen прототипами, разработанными в соответствии с архитектурой, описанной на рисунке 2.9. Панель макетирования запрашивает службы на сервере виртуальной машины (VMS). Эти услуги могут быть связаны с распознаванием или действием на инфраструктуру. VMS выполняет требуемое действие и отправляет ответ на плоскости пилотирования. Для упрощения реализации плоскости макетирования интерфейсов, предлагаемых VMS, должны быть четко определены, а платформа независима.

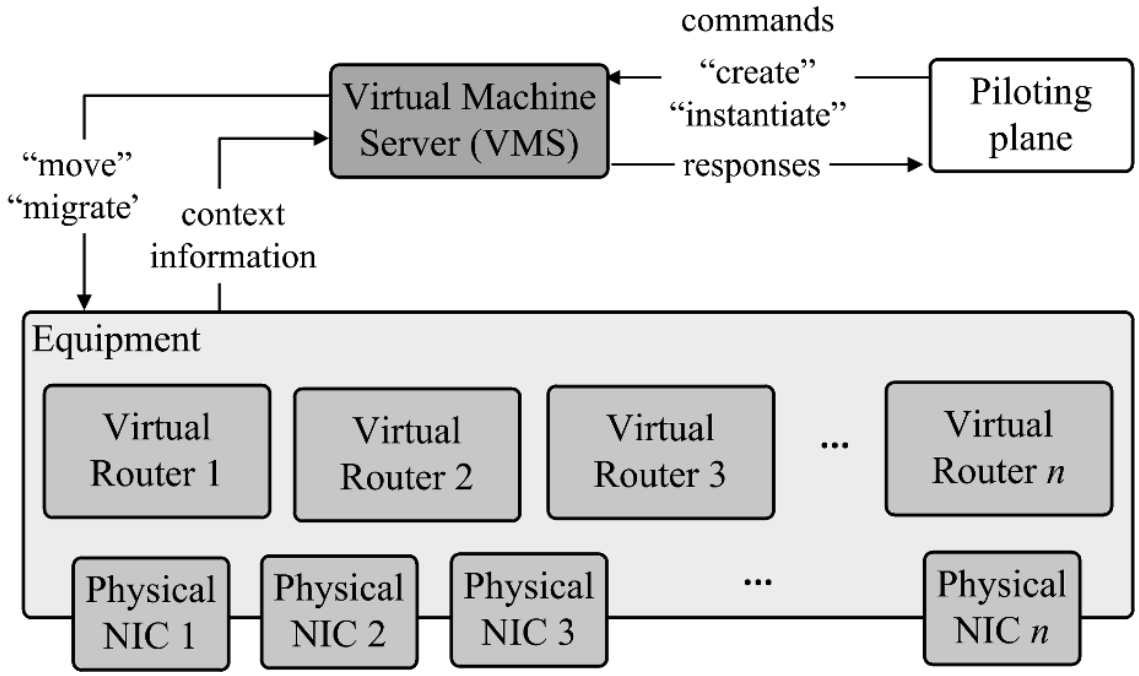


Рисунок 2.9. Архитектура прототипа Horizon Xen

VMS предоставляет набор сервисов для управления виртуальными сетями и виртуальными маршрутизаторами. Эта система обеспечивает виртуальные маршрутизаторы по запросу для соответствия определенным требованиям. Получив запросы на новые виртуальные сети, сервер создает

правильное количество виртуальных машин и развертывает их в определенных узлах физической сети. Кроме того, этот сервер может принимать участие в деятельности сетевой администрации.

Сервер может быть реализован с использованием веб-служб и может отвечать на запросы, используя протокол простого доступа к объектам (SOAP). Веб-сервер может быть представлен Apache Tomcat. Этот подход упрощает создание гетерогенных клиентов для VMS и уменьшает сложность добавления новых функций. Каждая услуга производится общедоступным методом, моделируемым как класс VirtualMachineServer. Панель макетирования может действовать в сети и самостоятельно выполнять некоторые изменения, как это предлагается в будущих интернет-проектах. Например, мы можем уменьшить нагрузку перегруженной физической машины методом переноса одной из машин на другой физический узел. В этом случае панель макетирования отправляет команду в VMS с использованием протокола SOAP для запроса миграции виртуальной машины. Затем VMS использует Libvirt – виртуализованные системные библиотеки управления для выполнения операции миграции. Среди доступных услуг в прототипе Xen есть те, с помощью которых можно управлять несколькими виртуальными сетями. В следующих, типичных услугах, предоставляемых в тестовом стенде на основе Xen, обобщены методы для плюралистической архитектуры вместе с описанием их основных целей.

CreateVirtualMachine. Эта служба должна вызываться всякий раз, когда новые виртуальные машины должны быть созданы на узле сети, чтобы помочь для обеспечения работы сети прототипа Xen посредством операций человека. Этот интерфейс может заменить плоскость макетирования и может иметь доступ ко всем сетевым задачам администрирования.

CreateVirtualNetwork. Эта служба создает набор виртуальных машин на некоторых физических узлах сети. Кроме того, сервер VM должен

отображать созданный виртуальный сетевой интерфейс на указанную физическую сеть.

DestroyVirtualMachine. Эта служба уничтожает виртуальную машину. Уничтоженная виртуальная машина не может быть повторно использована в будущем.

RegisterNodes. Эта служба может использоваться для регистрации существующих узлов в сети, то есть имени, открытого ключа и IP-адреса виртуальных машин, которые будут сохранены на сервере VM.

GetPhysicalServerStatus. Эта служба получает список основных информаций о физическом сервере. Текущий список содержит количество ЦП, количество ядер, объем оперативной памяти, объем свободной памяти, имя хоста, номера и имена активных виртуальных доменов.

GetRegisteredNodes. Эта служба возвращает список зарегистрированных узлов на сервере VM.

GetVirtualMachineStatus. Эта служба возвращает список основной информации о виртуальной машине. Данный список содержит имя виртуальной машины, текущий объем оперативной памяти, общую RAM-память, которая может быть использована, текущее количество виртуальных процессоров (VCPU), максимальное количество VCPU, что VM может использовать, время процессора и текущее состояние виртуальной машины.

MigrateVirtualMachine. Эта служба переносит виртуальную машину с одного физического хоста на другой в той же сети.

SanityTest. Эта услуга является проверкой работоспособности для сервера VM. Клиент отправляет строку на сервер, и сервер отправляет обратно ту же строку.

ShutdownVirtualMachine. Эта служба отключает виртуальную машину. В этом случае, VM можно использовать в будущем.

TopologyDiscover. Эта служба создает матрицу с соседними узлами на физических и виртуальных сетях. Существует одно ограничение на эту услугу: узел должен быть зарегистрирован на сервере, используя служебный

регистр Nodes, чтобы быть частью физической топологии, а также иметь виртуальные машины в виртуальных топологиях.

GetVirtualMachineSchedulerParameters. Эта служба запрашивает гипервизор для планировщика CPU,и параметры виртуальной машины.

Клиент сервера виртуальных машин

Для упрощения развития клиента класс может быть предложен с учетом каждой новой услуги, добавляемой на сервере. Для каждой новой услуги используется метод создания полезной нагрузки сообщений, добавляющийся в класс клиента. В следующем, возможные сообщения перечислены с их соответствующими полезными нагрузками, реализованными в классе клиента, для взаимодействия с VMS. Эти сообщения предлагаются в проекте Horizon.

CreateVirtualMachinePayload. Этот метод создает полезную нагрузку для запроса о создании виртуальной машины на основе имени физической машины, в котором *она* размещена.

IP-адрес и требуемый объем оперативной памяти

Метод возвращает Extensible Markup Language (XML), представленный объектом класса OMElement, с результатом операции, который может быть успешным или неудачным.

CreateVirtualNetworkPayload. Этот метод создает полезную нагрузку для запроса о создании виртуальной сети на основе списка физических имен машин, на которых будут размещаться новые виртуальные машины, списка с именами новых виртуальных машин, списка с желаемыми IP-адресами, а так же списка с нужными RAM-памятью и физических сетевых интерфейсов, которые будут отображены на новых виртуальных сетевых интерфейсах, созданных на виртуальных машинах. Метод, возвращающий XML-сообщения, представлен объектом класса OMElement, а результатом операции может быть успешным или неудачным.

DestroyVirtualMachinePayload. Этот метод создает полезную нагрузку для запроса об уничтожении виртуальной машины на основе имени

физической машины, которая содержит виртуальную машину и имя виртуальной машины. Метод возвращает XML сообщение, представленное объектом класса OMElement, а результат может быть успешным или неудачным.

GetPhysicalServerStatusPayload. Этот метод создает полезную нагрузку для получения статуса физического сервера, основанного на имени физической машины. Метод возвращает XML-сообщение, представленное объектом класса OMElement, с результатом, который может быть успешным или неудачным, количеством процессоров, количеством ядер, общим объемом оперативной памяти, количеством свободной оперативной памяти, именем хоста, а также номером и именем активных виртуальных доменов.

GetVirtualMachineStatusPayload. Этот метод создает полезную нагрузку для получения статуса виртуальной машины, основанной на имени физического компьютера, на котором размещена виртуальная машина, и имени виртуальной машины. Метод возвращает XML-сообщение, представленное объектом класса OMElement, с результатом операции, который может быть успешным или неудачным, именем виртуальной машины, текущим объемом оперативной памяти, общей RAM-памятью, которую можно использовать, текущим количеством VCPU, максимальным количеством VCPU, которые VM может использовать, используемым временем процессора и текущим состоянием виртуальной машины.

MigrateVirtualMachinePayload. Этот метод создает полезную нагрузку для переноса виртуальной машины на основе имени исходной физической машины, имени конечного физического компьютера, имени виртуальной машины и строки с указанием если операция в реальном времени, то есть, в случае, если произойдет миграция без прерывания программ, запущенной на виртуальной машине. Метод возвращает XML-сообщение, представленное объектом класса OMElement, с результатом операции, который может быть либо успешным, либо неудачным.

TopologyDiscoverPayload. Этот метод создает полезную нагрузку для обнаружения топологии. Сервис не имеет параметров. Метод возвращает XML сообщение, представленное объектом класса OMElement, с результатом физической и виртуальной топологий.

Графический интерфейс пользователя (GUI)

GUI прототипа Xen может получить доступ к VMS через запросы командной строки. Все службы, которые используют пользователи GUI, получают доступ через коммуникационный интерфейс Веб-сервиса. Клиент командной строки получает необходимые службы и параметры от GUI.

2.3. OpenFlow прототип

Прототип OpenFlow был развернут в лаборатории GTA для экспериментов с новыми интерфейсами. В этом разделе диссертационной работы описываются компоненты, связанные с прототипом OpenFlow. Следуя той же идее прототипа Xen, прототип OpenFlow также может основываться на веб-сервисах. Связь между ядром прототипа и внешними приложениями использует HTTP (HyperText Transfer Ptotocol) для обмена XML-сообщениями. Чтобы измерить сеть OpenFlow можно использовать датчики. Эти датчики в основном являются счетчиками, установленными на коммутаторах, доступных через протокол OpenFlow, или в виде информации на таблице OpenFlow, такой как количество записей потока и другие статистические данные.

Приложения NOX собирают информацию о датчиках и делают их доступными как Веб-сервис. На рис. 2.10 показана архитектура прототипа OpenFlow. NOX контроллер является базой для приложений OpenFlow. Контроллер NOX обеспечивает протокол OpenFlow и реализацию безопасного канала для приложения.

Список приложений, работающих на контроллере NOX, вместе с их основными целями описаны ниже.

Приложение статистики: это приложение собирает статистику о коммутаторах и преобразует их в XML-сообщение.

Приложение обнаружения: это приложение обнаруживает сетевую топологию и описывает это как XML-сообщение.

Приложение SpanningTree: это приложение реализует остовное дерево – алгоритм, который позволяет избежать возникновения сетевых циклов.

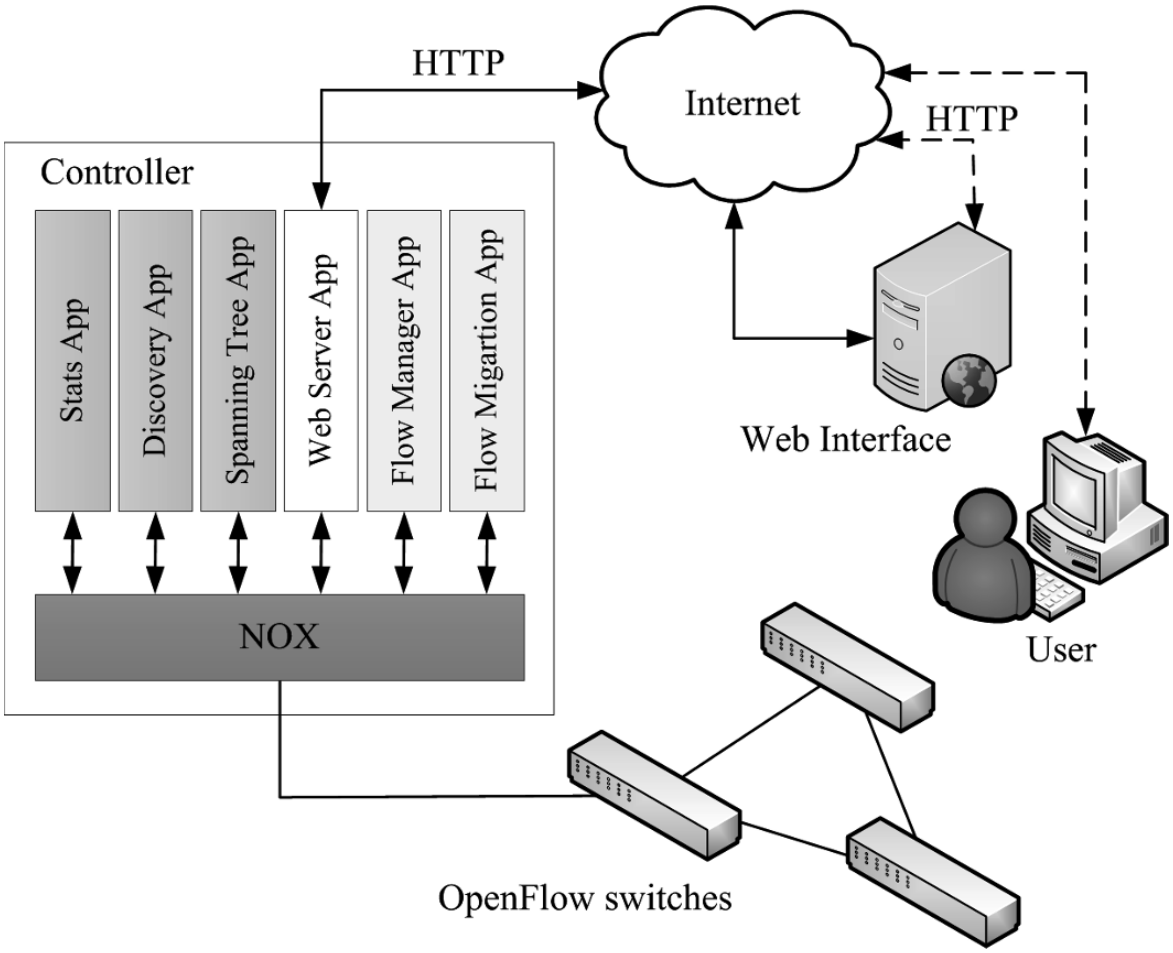


Рисунок 2.10. Приложения OpenFlow, NOX и взаимодействия между агентами

Приложение FlowManager: это приложение реализует изменения потока как добавление, модификация и удаление потоков.

Приложение FlowMigration: это приложение реализует изменения потока, перенос потока с одного пути на другой.

Приложение WebServer: это приложение обеспечивает интеграцию между функцией приложения и контроллера NOX. Приложение WebServer реализует протокол HTTP для обеспечения интерфейса между приложениями контроллера NOX и внешними приложениями. Следовательно, приложение контроллера NOX обрабатывает HTTP-запросы, преобразует их в вызов метода приложения и выполняет метод. Прототипом, используемый для экспериментов, является клиент для приложения WebServer. Клиент - это еще один веб-сервер, который позволяет администраторам управлять сетью OpenFlow.

Веб-сервер OpenFlow. Приложение WebServer - это приложение NOX, которое отвечает предоставлению веб-интерфейса для других приложений NOX. Приложение WebServer реализует концепцию веб-службы, в которой функциональность других приложения может быть доступна по HTTP-запросу, и возвращает XML сообщения. Реализация этого приложения основана на сети NOX, серверное приложение по умолчанию можно настроить для работы на порту 8080, возможно прослушивание HTTP-запросов. Все HTTP-запросы обрабатываются ресурсом OpenFlow, определенным в приложении mywebserver. Для каждого веб-сервиса, предоставляемого OpenFlow, существует метод, определенный в MyWebServerResource. Концепции, касающиеся каждого компонента приложения WebServer, описаны ниже.

Приложение NOX реализует платформу для развертывания Веб-сайтов, как аппликация NOX. Эта функция используется для реализации сети услуг как особый вид веб-сайтов. Класс mywebserver - это NOX приложение для реализации класса. Он запускает все приложения, которые должны запускаться одновременно с приложением WebServer. Он также запускает NOX по умолчанию и определяет его ресурс как объект класса MyWebServerResource. Класс MyWebServerResource определяет ресурс,

который является своего рода веб-сайтом, реализуемым стандартом NOX. Этот класс также реализует отображение URL-запросов для вызова функций, обеспечивающих интерфейс между пользователем и OpenFlow сетью. Есть некоторые услуги, уже реализованные на MyWebServerResource. К каждой службе можно обратиться по HTTP-запросу, используя определенный URL. Услуги описаны ниже:

GetStats: эта служба не принимает никаких параметров. Она вызывает приложение Stats и возвращает статистику и информацию о переключателе OpenFlow сети в XML-сообщении.

GetTopology: эта служба не принимает никаких параметров. Она вызывает Discovery App и возвращает топологию сети в XML-сообщении. Эта служба возвращает список всех сетевых ссылок.

GetNeighbor: эта служба не принимает никаких параметров. Она вызывает Discovery App и возвращает топологию сети в XML-сообщении. Эта служба возвращает список всех соседних узлов для каждого узла сети.

GetSpanningTree: эта служба не принимает никаких параметров. Она вызывает App Discovery и возвращает связующее дерево сети в XML-сообщении. Эта служба возвращает список соседних узлов, которые связаны с узлом по связующему дереву, для каждого узла в сети.

AddFlow: эта служба использует характеристики потока такие параметры, как поток, тайм-аут простоя, жесткий тайм-аут, приоритет и действие. Эта услуга добавляет новый поток, вызывающий приложение FlowManager, которое выполняет требуемое действие над сетью.

DelFlow: эта служба принимает в качестве параметров характеристики потока, такие как совпадение потока, тайм-аут простоя, жесткий тайм-аут и приоритет. Эта служба удаляет поток, вызывающий приложение FlowManager, которое выполняет требуемое действие по сети.

MigrateFlow: эта служба использует характеристики потока параметрами которых являются, например совпадение потока, тайм-аут простоя, жесткий тайм-аут, приоритет, действие, а также список которые

должны быть установлены. Эта услуга переносит вызов потока приложения FlowMigration, которое выполняет требуемые действия по сети.

Графический интерфейс пользователя

OpenFlow перенаправляет сетевой трафик в соответствии с таблицей потоков, содержащей активные потоки. Эта таблица содержит характеристики и правила потока, которые должны применяться, например, для определения очереди и выходного порта. Эта таблица может быть настроена локально или сетевым контроллером. Удобный интерфейс может быть так реализован, чтобы позволить пользователям изменять таблицы потоков, и облегчить конфигурации управлением сетью. Этот пользовательский интерфейс был разработан на основе веб-приложения, в котором, используя веб-браузер, пользователь имеет доступ к интерфейсу и может запускать команды и запросы для управления сети.

Приложение, которое предоставляет графический интерфейс, делится на три уровня.

Первый уровень - это уровень данных, где выполняются пользовательские команды и где сбор данных выполняется в виде ответов на запросы.

Второй уровень - уровень обработки данных, который обрабатывает всю полученную информацию, отправляя его на другие уровни.

Третий уровень - это уровень презентации, который организует и показывает данные пользователю.

Изоляция, обеспечивающаяся слоем структур, позволяет модифицировать определенный слой без изменения других. На рисунке 2.11 показаны уровни приложения и используемые протоколы для обмена сообщениями.

Уровень данных: уровень данных состоит из контроллера NOX и его приложения. Приложение WebServer App отвечает за предоставление коммуникационного интерфейса между слоем данных и слоем обработки

данных. Это приложение взаимодействует с уровнем обработки данных через HTTP-протокол.

Уровень обработки данных: уровень обработки данных состоит из Веб сервера, который обрабатывает запросы и команды от пользователей. Проектирование собственного сервера, вместо использования существующего сервера, такого как Apache, может быть полезным. Предложение администраторам полного контроля над предоставленными услугами способствует развитию системы.

Уровень представления данных: уровень презентации состоит из метки и скриптов, которые интерпретируются веб-браузером. Они могут быть в виде HTML, JavaScript (JS), CSS, XML и SVG. HTML-файлы имеют веб-интерфейс который веб-браузер интерпретирует, чтобы показать эту веб-страницу.

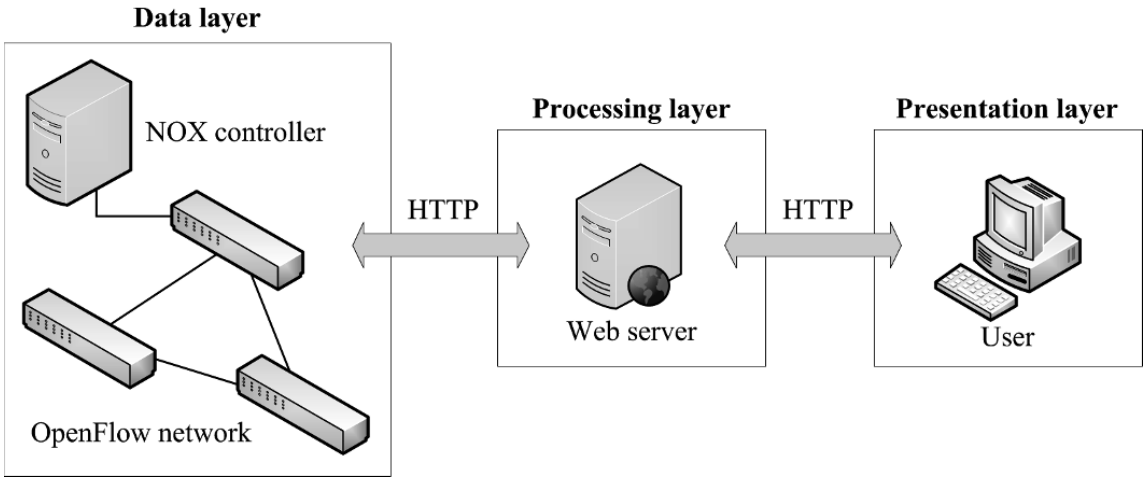


Рисунок 2.11. Прикладные уровни

CSS-файлы имеют разметку стиля, чтобы улучшить вид, предоставленного HTML. Файл CSS интерпретируется веб-браузером и применяет стиль к разметке HTML. Файлы JS имеют функции скрипта для создания динамичных и интерактивных веб-страниц. XML-файлы имеют информацию о веб-браузере или JS, чтобы показывать или обмениваться данными. Сообщения, которые предоставлены в качестве ответов команд пользователя, представляют собой XML-сообщения. SVG-файлы имеют

XML-описание топологии сети для предоставления графической визуализации. Комбинация между SVG и JS позволяет создавать анимацию и способствует взаимодействию с изображениями, созданными с помощью SVG. На этом уровне есть ресурсы, которые позволяют пользователям получать сетевую информацию и выполнять команды.

III ГЛАВА. ПОВЫШЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ И УПРАВЛЕНИЕ ЭЛЕМЕНТАМИ ВИРТУАЛЬНОЙ СЕТИ

3.1. Xen-based прототип

Во второй главе диссертационной работы определены пять примитивов (создание, удаление, миграция, мониторинг и настройка), которые инфраструктура виртуализации сети должна обеспечивать для панели управления, с помощью которого, осуществляется контроль и управление элементами виртуальной сети. На рисунке 3.1 показана взаимосвязь между панелью управления и общим виртуализированным сетевым элементом.

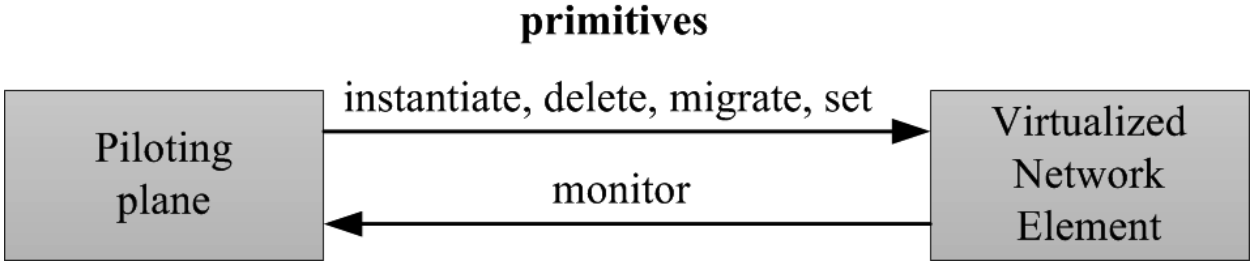


Рисунок 3.1. Управляющие примитивы и связь с панели управления и виртуализированного сетевого элемента

В принципе, панель управления выполняет интеллектуальные алгоритмы: автоматическое создание / удаление виртуальных сетей, а также миграция сетевых элементов и задание параметров распределения ресурсов. Следовательно, для панели управления необходимо получить информацию и использовать примитив монитора. Этот примитив выполняет вызовы инструментов мониторинга, необходимых для измерения интересующих переменных, таких как доступная пропускная способность, процессор и память, связь и сквозная задержка. После мониторинга, панель управления

может действовать в сети с использованием четырех примитивов. Параметры, доступные для мониторинга и настройки, включают в себя низкоуровневые и аппаратные параметры, такие как количество виртуальных процессоров, предоставляемых виртуальному маршрутизатору, и приоритет использования процессора в конфликтном сценарии. Таким образом, панель управления динамически адаптирует ресурсы, выделенные для каждой виртуальной сети, в соответствии с текущим состоянием сети, количеством пользователей, приоритетом каждой виртуальной сети, соглашениями об уровне обслуживания (SLA) и т. д. Мы считаем, что элемент виртуальной сети имеет две основные панели: панель виртуализации и управления. Платформа виртуализации обеспечивает субстрат для работы логических сетевых элементов в одной общей физической сети. Панель управления обеспечивает основу для оптимизации производительности сети на основе SLA для каждой виртуальной сети. На рисунке 3.2 показана архитектура общего элемента виртуальной сети. Ядром узла является система виртуализации, которая может быть реализована различными инструментами виртуализации, такими как Xen или OpenFlow. Датчики и приводы также считаются составляющими элементами системы виртуализации. Датчики собирают информацию, позволяющую описать контекст, к которому они принадлежат, а исполнительные механизмы - это программные компоненты, которые применяют действия, требуемые панелью управления на сетевом уровне. На рисунке 3.2 контроллер представляет собой панель управления, который принимает и агрегирует информацию, отправленную узлами, и затем отправляет команды для установки параметров и выполнения управляющих действий на узлах. Все узлы должны обеспечивать интерфейс с панелью управления и иметь возможность обмениваться с ним сообщениями для управления в формате Extensible Markup Language (XML).
В этой главе рассматриваются конкретные изменения, виртуализации для реализации пяти примитивов, используемых администраторами, а также для повышения производительности виртуализованных сетевых элементов.

Каждый инструмент виртуализации вводит другой набор датчиков, а также требует различные механизмы управления сетью и реализации примитивов панели управления. Например, Xen предоставляет встроенные приводы для создания и уничтожения виртуальных машины (VM), но собственный механизм миграции VM недостаточно адаптирован к приложениям виртуального маршрутизатора, потому что он не избегает потери пакетов. Поэтому должен быть обеспечен эффективный механизм миграции. Подобно Xen, OpenFlow предоставляет инструменты для создания и уничтожения потоков, но он также не имеет механизма миграции родного потока. Таким образом, пути по внедрению подробно описаны для двух конкретных прототипов на основе инструментов виртуализации Xen и OpenFlow.

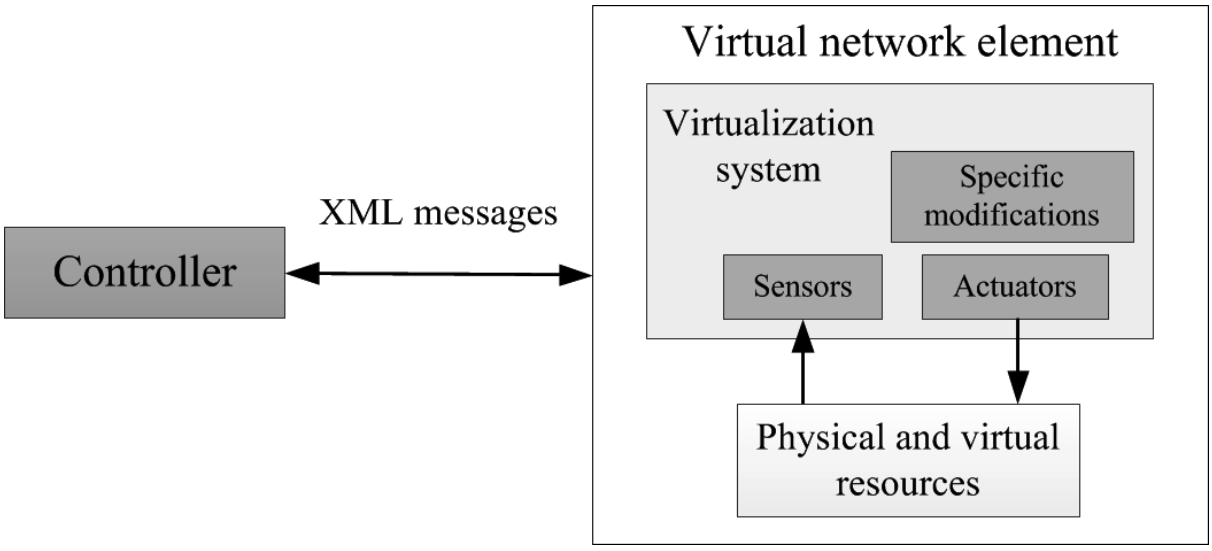


Рисунок 3.2. Архитектура общего элемента виртуальной сети: контроллер принимает данные, полученные датчиками, и отправляет команды для управления физическими и виртуальными ресурсами сетевых элементов

Прототип сети на основе Xen состоит из нескольких модулей, которые выполняются на разных типах узлов. В принципе, каждый узел играет некую роль: контроллера, маршрутизатора или же клиента. Контроллер - это

специальный узел, который принимает и консолидирует данные, полученные от всех сетевых узлов. Узел контроллера получает данные и отправляет команды физическим и виртуальным маршрутизаторам. Маршрутизатор узла - это сетевые субстраты. Один физический маршрутизатор запускает один или несколько виртуальных маршрутизаторов. Как физические, так и виртуальные маршрутизаторы запускают несколько модулей, которые контролируют и оперативно действуют при приеме команд. Наконец, клиентский узел позволяет пользователям взаимодействовать с контроллером с помощью графического пользователя Интерфейс (GUI). Этот графический интерфейс представляет информацию о прототипе для мониторинга пользователей, а также простой интерфейс управления. На рис. 3.3. приведены основные модули Xen-прототипа и его интерфейсы. Контроллер имеет два модуля: сервера виртуальной машины (VMS) и связи с клиентом. VMS является ядром контроллера. Который имеет интерфейс протокола простого доступа к объектам (SOAP) для взаимодействия как с панелью управления, так и с клиентским узлом. VMS объединяет все виды информации и выполняет все алгоритмы управления и обслуживания. Клиент коммуникационного модуля используется VMS – ом для связи с другими модулями маршрутизатора.

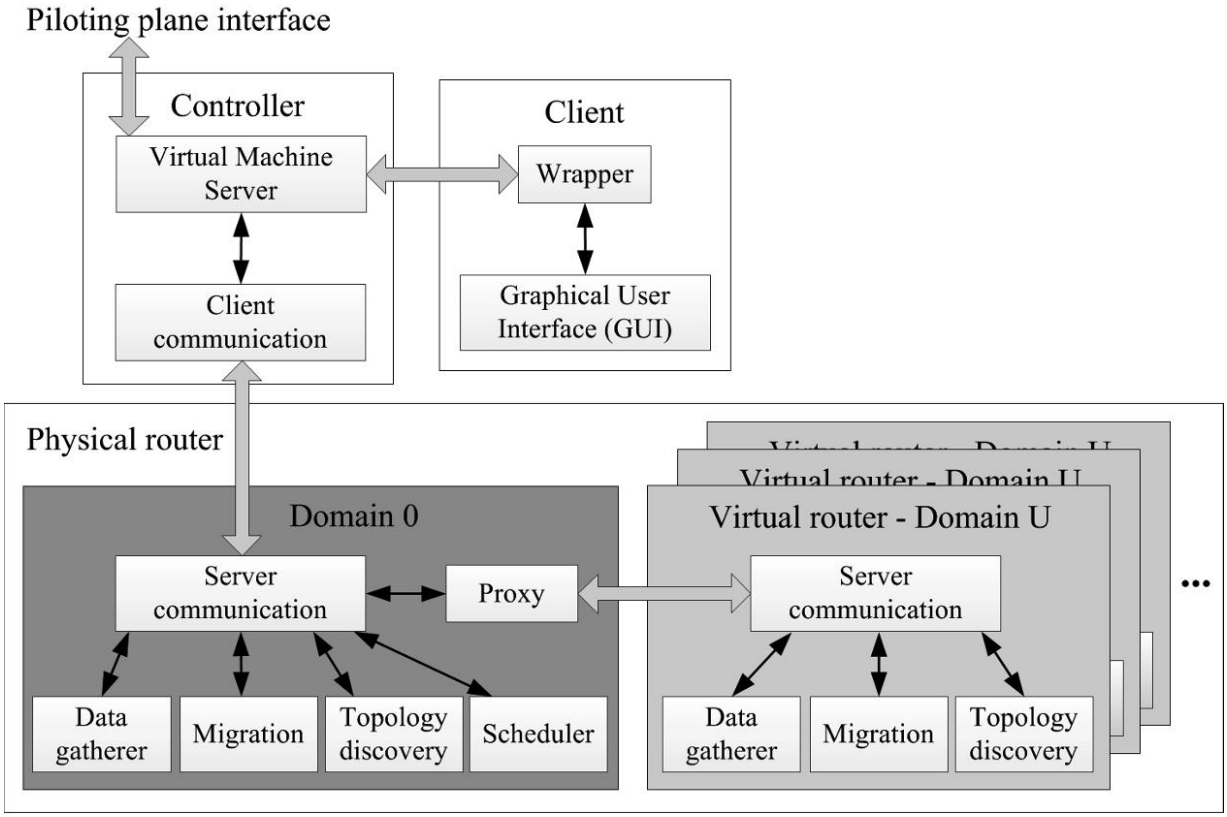


Рисунок 3.3. Архитектура прототипа Xen: Основные модули и отношения

Физические маршрутизаторы обеспечивают базу, используемую виртуальными сетями. Каждый физический маршрутизатор выполняет модуль связи с сервером, который принимает запросы от контроллера и перенаправляет эти запросы на определенные модули. Если запрос адресован данному виртуальному маршрутизатору, он перенаправляется на конкретный виртуальный маршрутизатор с помощью модуля Proxy. В противном случае он отправляется к одному из других модулей, работающих на физическом маршрутизаторе. Например, если полученный запрос рассматривает задачу мониторинга, он пересылается в Data Gatherer модуль для получения параметров с физических и виртуальных маршрутизаторов. Если полученный запрос соответствует действию, связанному с совместным использованием ресурсов среди виртуальных маршрутизаторов модуль Scheduler обрабатывает этот запрос. Модуль Topology Discovery обрабатывает запросы,

связанные с физическим или виртуальным обнаружением топологий сетей. Запросы на миграцию обрабатываются модулями миграции, которые обеспечивают миграцию виртуального маршрутизатора без потерей пакетов. Клиентский узел позволяет пользователям мониторить и контролировать прототип сети. Данный узел имеет графический интерфейс, который показывает физические, и виртуальные сетевые топологии, а так же дополнительно позволяет контролировать мелкий материал, демонстрируя подробную информацию о выбранном физическом или виртуальном маршрутизаторе. Клиентский узел также позволяет пользователям отправлять команды прототипу. Например, пользователи могут выполнить миграцию виртуального маршрутизатора с одного физического маршрутизатора на другой физический маршрутизатор с помощью клика компьютерной мышкой. GUI, запущенный на клиенте взаимодействует с контроллером с помощью модуля Wrapper. Этот модуль преобразует команды из GUI в SOAP, а затем вызывает контроллер.

Миграция - важный контрольный примитив, поскольку позволяет панели управления динамически перестраивать логическую топологию сети, без нарушения работы служб и без потери пакетов. Xen имеет собственный механизм миграции, разработанный для перемещения виртуальных машин. Этот механизм основан на двух предположениях:

(1) миграция происходит внутри локальной сети;

(2) жесткий диск VM совместно используется в сети.

Таким образом, миграция виртуальной машины состоит из двух основных процедур:

(1) копия виртуальной машинной памяти к новой физической локации;

(2) реконфигурация сети ссылки без разрыва соединений.

Существует несколько вариантов выполнения копии памяти. Простейший альтернативой является приостановка виртуальной машины, перенос всех блоков памяти на новый физический узел, а затем возобновить работу виртуальной машины. Несмотря на простоту, эта альтернатива

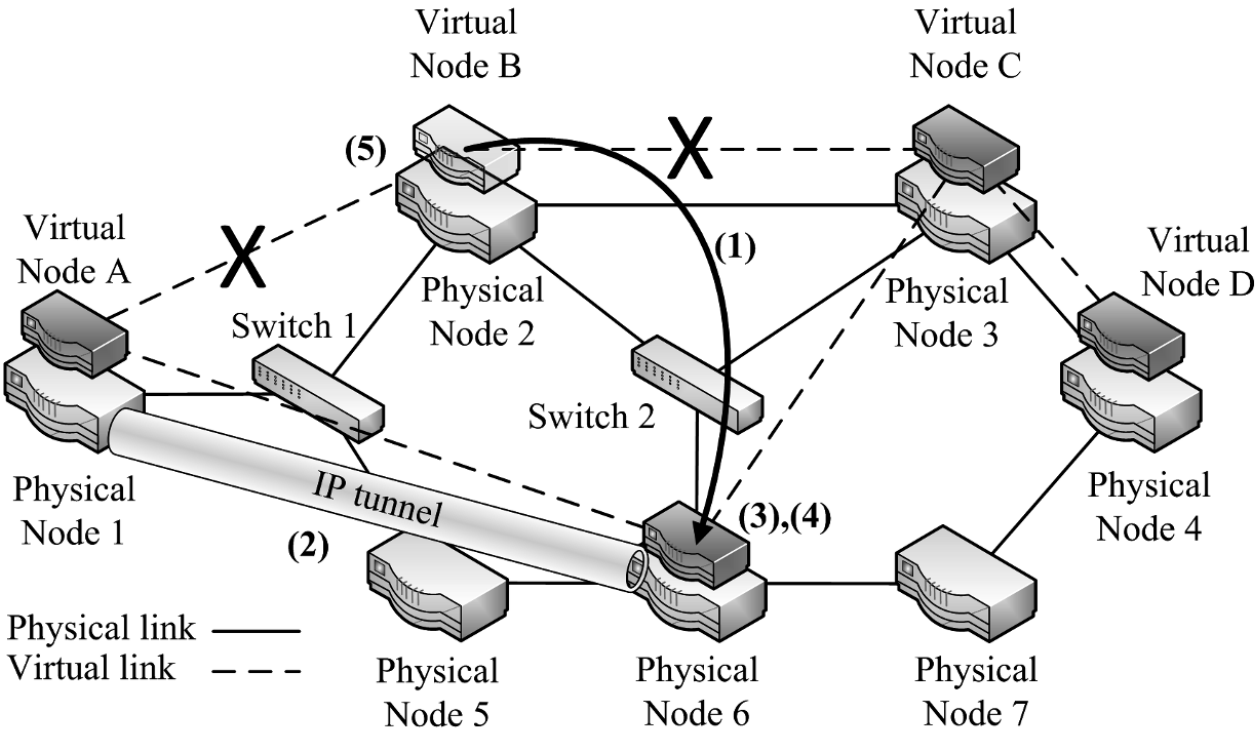
страдает высоким временем простоя, то есть время, когда виртуальная машина недоступна во время миграции. Чтобы сократить время простоя, процедура приостановки-передачи-возобновления pre-copy migration, которая состоит из двух этапов:

(1) итеративное предварительное копирование;

(2) копирование.

На первом этапе все страницы памяти переносятся на новую физическую машину, за исключением тех, которые называются «горячими страницами», которые являются наиболее часто измененными страницами. Таким образом, время простоя сокращается, поскольку только несколько страниц, а именно горячие страницы, передаются, пока виртуальная машина отключена. Собственный механизм VM-миграции Xen хорошо работает для приложений консолидации серверов, но он не эффективен для миграции виртуального маршрутизатора. Из-за использования механизма предварительной копии, время простоя находится в порядке сотни миллисекунд, и за это время пакеты теряются. В зависимости от скорости передачи данных, виртуальные маршрутизаторы испытывают потери большого количества пакетов во время простоя. Также важно минимизировать общее время миграции, чтобы гарантировать, что мы можем быстро освободить ресурсы физической машины. Еще одна проблема собственного механизма Xen для переноса виртуальных маршрутизаторов - это то, что он предполагает, что миграция всегда выполняется с одного физического маршрутизатора к другому физическому маршрутизатору в той же локальной сети (LAN). В Интернете, мы не можем предположить, что физические узлы всегда принадлежат к одной и той же локальной сети. Чтобы перенести виртуальные маршрутизаторы без потери пакетов, альтернативой является реализация метода разделения плоскостей в Xen. Механизм миграции с плоской сепарацией гарантирует отсутствие потери пакетов в плоскости данных во время миграции виртуальной машины. Также отсутствует потеря пакетов управления. Механизм вставляет только

задержку доставки пакета управления. Модифицированный механизм, однако, основан на миграции по умолчанию Xen, т. е. он также требует, чтобы маршрутизаторы находились в одной локальной сети. Кроме того, отображение виртуальной связи по нескольким физическим ссылкам остается открытой проблемой, которая зависит от туннели Интернет-протокола (IP) или создание новых виртуальных маршрутизаторов в сети. Например, на рисунке 3.4 мы переносим Виртуальный Узел от физического узла 2 к физическому узлу 6. Физический узел 6, однако, не является соседом с физическим узлом 1. Следовательно, для завершения мы должны создать туннель от физического узла 6 до физического узла 1, для имитации окрестности с одним хопом. Другое решение создать новый виртуальный роутер для замены туннеля. Это решение, однако, изменяет виртуальную топологию и влияет на работу протокола маршрутизации. Есть также другой компонент, который используется во время процесса миграции, Миграция нужного компонента. Этот компонент работает как по источнику, так и по назначению и сообщают об успехе или провале процесса миграции.



(1) Migrate control plane from Physical Node 2 to 6 maintaining data and control planes
(2) Do interface dynamic binding and create tunnel between Physical Nodes 1 and 6
(3) Create a new forwarding table in Domain 0 of Physical Node 6
(4) Reconfigure links with ARP reply
(5) Delete data plane on Physical Node 2

Рисунок 3.4. Пример миграции маршрутизатора на основе Xen, когда виртуальная ссылка отображается в много интервальный путь в физической сети

Xen-топология. Модуль топологии обнаруживает топологии как физических, так и виртуальных сетей. Этот модуль проверяет всех соседей каждого сетевого элемента, используя Nmap Security Scanner. Модуль топологии имеет три компонента: сканирующие соседи, консолидация топологии и узел объединения.

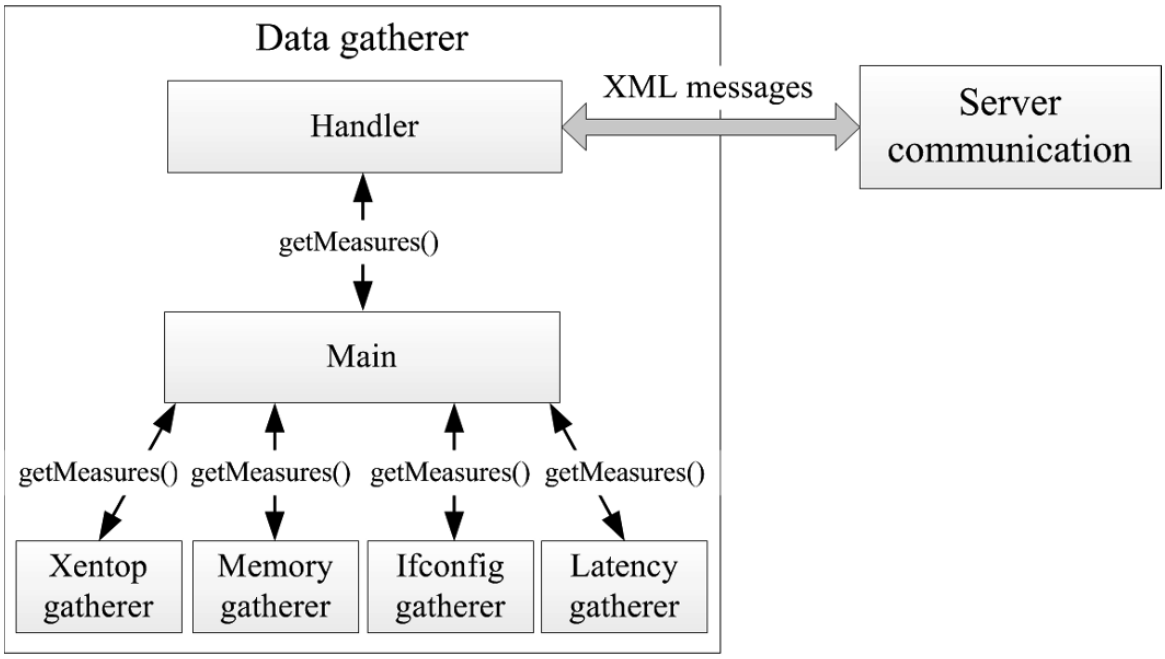


Рисунок 3.5. Архитектура Data Gatherer: основной модуль отправляет запросы на конкретные модули мониторинга, а затем пересылает полученные данные обработчику, который обрабатывает эти данные.

Компонент сканирования отвечает за обнаружение всех соседей определенного сетевого элемента. Выполняются как физические, так и виртуальные сетевые элементы этого компонента. Поиск соседа использует инструмент Nmap, который проверяет все диапазон, IP каждого сетевого интерфейса, элементов сети. Мы добавляем к списку адресов все IP-адреса, которые отвечают стандартам. После этого у нас есть все соседи сетевого элемента, которые связаны всеми сетевыми интерфейсами. Информация о соседстве это владение информацией об IP-адресе и MAC-адресе соседа, и о латентности ссылки. После обнаружения соседей всех сетевых интерфейсов, компонент создает список окружения и передает его компоненту узла консолидации через XML-сообщения, как на Рисунке 3.6. Компонент «Узел Консолидации» выполняется на каждом физическом элементе сети. Этот компонент направлен на получение информации о соседей этого элемента как в физических, так и в виртуальных сетях. Топология консолидации

компонента вызывает компонент «Консолидация узла», который имеет три задачи. Первая задача - обнаружить физических соседей элемента сети. Вторая задача - обнаружить всех соседей каждого элемента виртуальной сети, работающих над этим физическим сетевым элементом. И те и другие процедуры обнаружения используют компонент «Сканирование». Первые две задачи выполняются в одно и то же время, потому что между ними нет зависимости. Последняя задача компонент «Консолидация узла» предназначен для консолидации информации. Консолидация состоит из информации о соседстве физического элемента сети и списка элементов виртуальной сети со своими соседями. Модуль возвращает консолидированную информацию контроллеру, который запускает топологию консолидации компонента с использованием сообщений в формате XML.

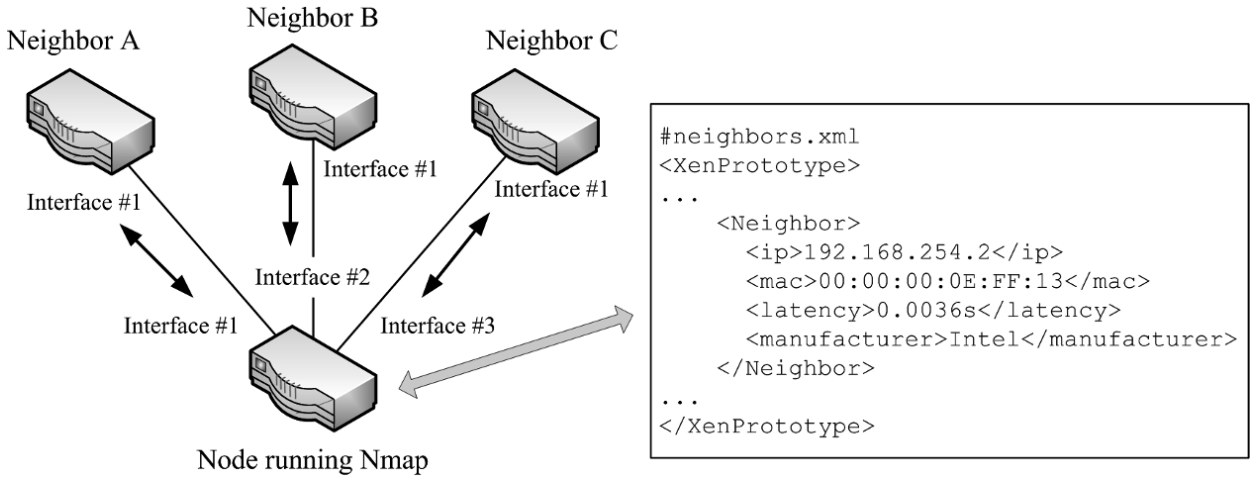


Рисунок 3.6. Компонент «Сканирующие соседи»: инструмент nmap используется для зондирования всех интерфейсов сетевого элемента и, следовательно, узлы, которые реагируют на эти зонды добавляются в список окрестностей.

VMS имеет список зарегистрированных узлов физической сети, но нет информации о взаимосвязях между этими узлами. Модуль топологии предоставляет информацию о подключении. Этот модуль запрашивает

каждый физический узел в списке зарегистрированных узлов о информации об соединений, о физических и виртуальных элементах. После получения всей информации о соединении, компонент топологии консолидации вычисляет топологию каждой сети. Мы моделируем топологию сети как график. Поэтому компонент возвращает график представления в VMS. График представлен матрицей смежности каждой сети. Таким образом, модуль топологии позволяет VMS предоставлять текущую топологию физических и виртуальных сетей.

3.2. OpenFlow прототип

Прототип OpenFlow основан на наборе приложений, запущенных над NOX контроллере. NOX - это контроллер OpenFlow который настраивает и управляет потоками в коммутаторах OpenFlow. Разработанное приложение NOX реализует пять необходимых примитивов: экземпляр потока (экземпляр примитива), удаление потока (удалить примитив), миграция потока (мигрировать примитив), управление и изменение потоков (набор примитивов), мониторинг сети и обнаружение топологии (примитив монитора), а также функции, чтобы избежать циклов в сети. Существует также интерфейс веб-сервиса для предоставления пяти примитивов системе управления. Архитектура прототипа OpenFlow проиллюстрирована на рисунке 3.7. Прототип выполняет роль как датчиков, так и исполнительных механизмов, определенные примитивами системы управления. Приводы представляют собой диспетчеров потоков и потоков приложений для миграции. Приложение Flow Manager предлагает создавать экземпляры / удалять / устанавливать примитивы. Приложение Flow Manager реализует интерфейс между другими приложениями NOX и командами OpenFlow. Это приложение отвечает за добавление, изменение и удаление потоков. Приложение Flow Manager получает запрос операции потока и переводит его

в команду OpenFlow. Другим исполнительным механизмом является миграция потока приложения, которое реализует примитив миграции. Миграция потока переносит поток от одного пути к другому без потери пакетов. Статистика и приложения обнаружения внедряют датчики в прототипе. Оба приложения удовлетворяют примитиву монитора. Приложение «Статистика» измеряет сети и собирает статистику о коммутаторах. Приложение Discovery обнаруживает сетевую топологию и строит график для представления сети. Существует также приложение Spanning Tree, которое позволяет избежать появления петли в сети и ненужные повторные передачи в эфир.

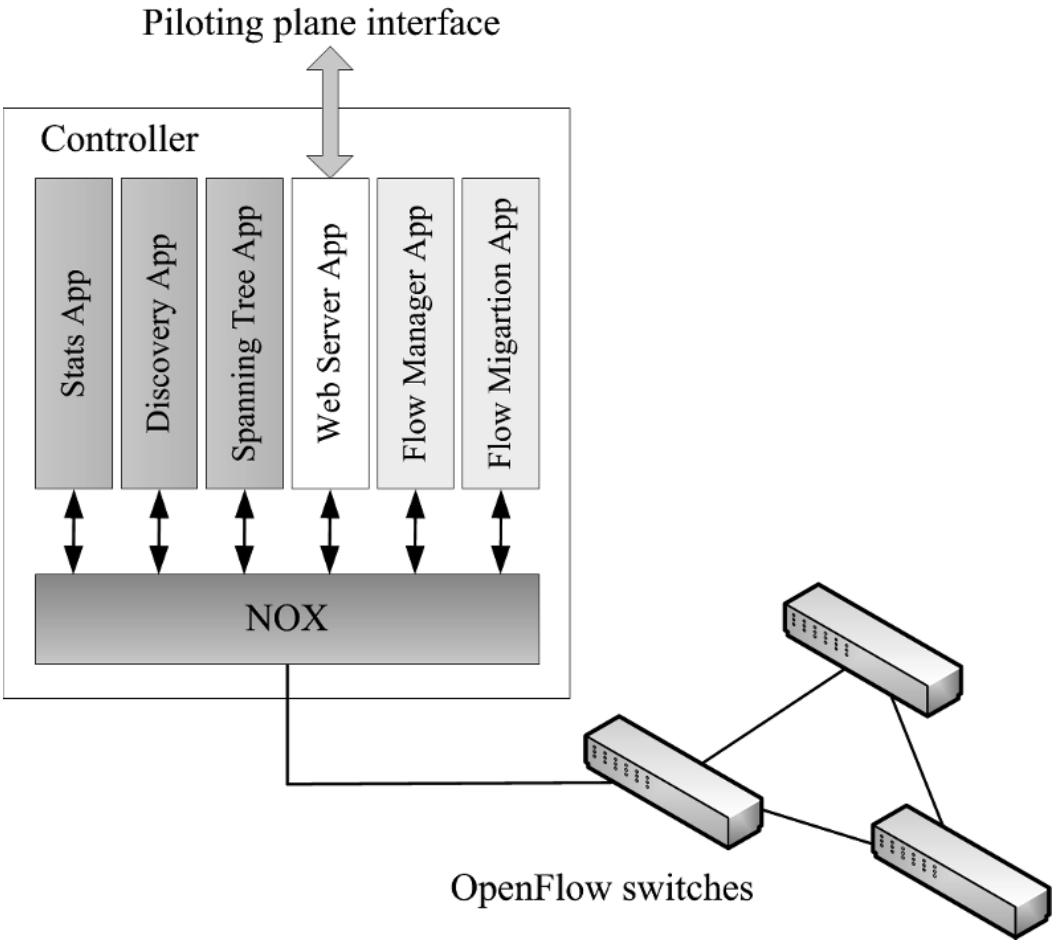


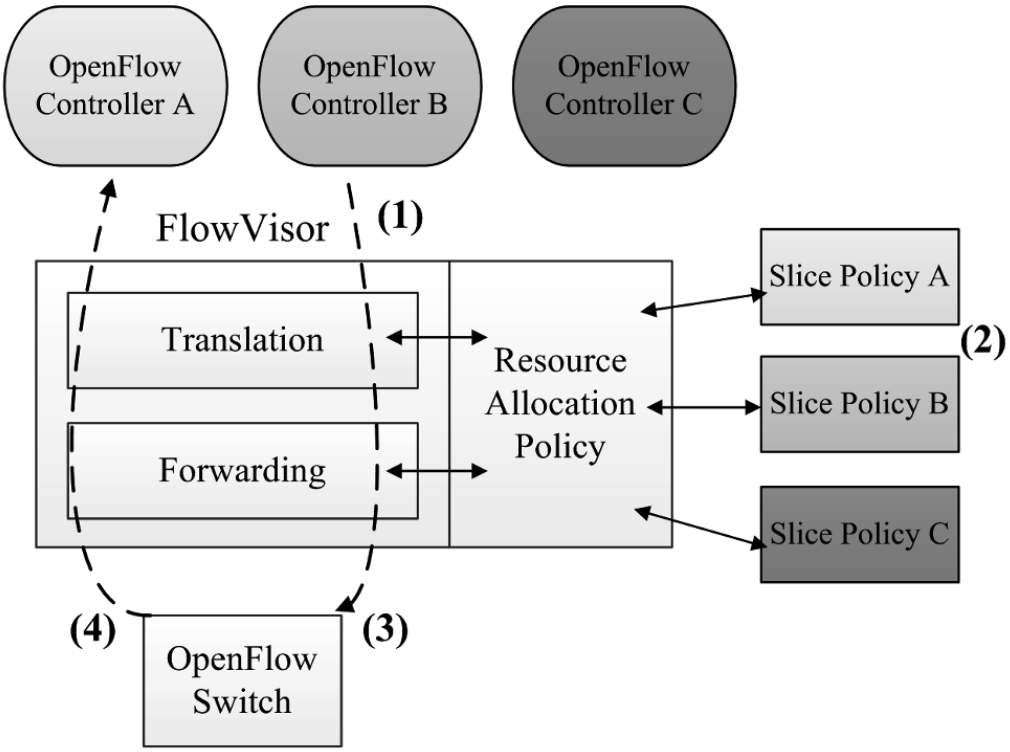
Рисунок 3.7. Архитектура прототипа на основе OpenFlow

Кроме того, используется инструмент, который позволяет нескольким контроллерам NOX работать параллельно для того, чтобы разделить сеть на

несколько виртуальных. Этот инструмент, который называется FlowVisor, управляет совместным использованием сетевых ресурсов, таких как пропускная способность, топология, наблюдение за контроллерами, трафик каждой активности, переключает использование ЦП и таблицу потоков. FlowVisor реализует заданный примитив для настройки параметров каждой виртуальной сети.

FlowVisor. FlowVisor является специальным типом контроллера OpenFlow. Оно работает как прозрачный прокси между сетевыми устройствами и другими контроллерами, такими как контроллеры NOX. Основной особенностью FlowVisor является возможность разделять сети и обмениваться сетевыми ресурсами контролируемыми и изолированными образом. Как показано на рисунке 3.8 (адаптировано из Sherwood et al.), FlowVisor перехватывает сообщения OpenFlow, отправленные гостевыми контроллерами. После FlowVisor, основанный на политике отделения пользователей (2), прозрачно изменяет сообщение, чтобы разделить элемент управления по различным сетям. Только FlowVisor пересылает сообщения от коммутаторов к гостям, если сообщения соответствуют заданной политике. FlowVisor разделяет сеть, сохраняя разделы, изолированные друг от друга.

FlowVisor может виртуализировать сеть, разделяя ресурсы коммутаторов среди нескольких контроллеров. Кроме того, FlowVisor позволяет создавать иерархии FlowVisor - ов, изменяющие сетевую архитектуру, как показано на рисунке 3.9. В этом случае один экземпляр FlowVisor (FlowVisor 2) виртуализирует несколько коммутаторов параллельно (Переключатели с 1 по 4), а FlowVisor 1 рекурсивно разделяет виртуальные разделы, определенные FlowVisors 2 и 3. Чтобы разделить сеть между контроллерами, FlowVisor фокусируется на совместном использовании пяти сетевых ресурсов: изоляция полосы пропускания, топология обнаружения, управление трафиком, таблицы мониторинга и контроля пересылки процессоров устройств.



FlowVisor:
(1) Intercepts the OpenFlow messages sent by guest controllers.
(2) Verifies the user slicing policy.
(3) Modifies the message to delimit the control to a slice of the network.
(4) Forwards messages from switches to guests, if the messages match the slice policy.

Рисунок 3.7. Операции, исполняемые FlowVisor

Например, изоляция полосы пропускания обеспечивается за счет использования различных очередей приоритетов на коммутаторах. Одна очередь изолирована от остальных. Все пакеты в потоке отмечены одним и тем же идентификатором, а затем отображаются на одну из восьми очередей приоритетов. Потоки можно классифицировать на основе трех разных идентификаторов: точка приоритета VLAN (PCP), тип IP Сервиса (ToS) и качество обслуживания OpenFlow (QoS). По умолчанию используется VLAN PCP. Важно отметить, что гарантировано минимальная пропускная способность.

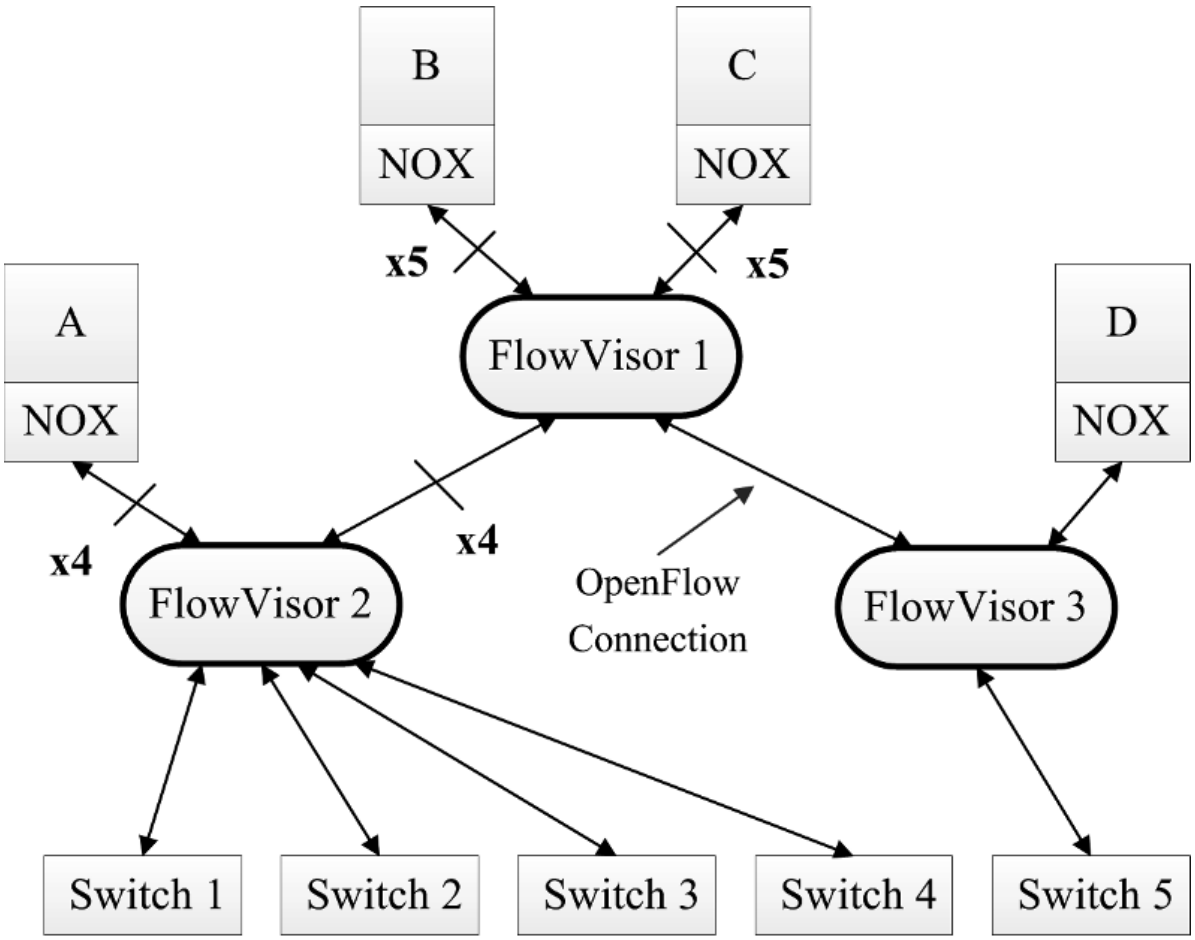


Рисунок 3.9. Иерархия FlowVisor: FlowVisor 1 рекурсивно разделяет виртуальные разделы, определенные FlowVisors 2 и 3

Миграция OpenFlow. Миграция потока - это приложение NOX, разработанное для прототипа, который определяет новый путь между источником и пунктом назначения OpenFlow, который переключает и изменяет текущий путь потока на новый путь без потерий пакетов, во время этой процедуры. Чтобы перенести поток, миграция потока приложения принимает в качестве параметров список коммутаторов. Этот список определяет упорядоченную группу переключателей, через которые должен пройти новый поток в пути. Вычислять кратчайший путь от источника до пункта назначения, включая выбранные коммутаторы, он использует обобщение алгоритма Дейкстры. Приложение Flow Migration имеет два

интерфейса: один из которых, является серверное приложение с веб-интерфейсом, с помощью которого изменяются параметры управления, и коммутатор, где приложение отправляет конфигурацию потока команды. Рисунок 3.10 иллюстрирует пример миграции потока в процессе. Первоначально источник отправляет пакеты в пункт назначения через Путь 1, составленный узлами A-F-E-D. Новый поток от коммутатора от A до D с требованием, чтобы он проходил через коммутаторы A, B и D, темно-серые узлы A, B и D не являются полным путем от A до D. Таким образом, приложение Flow Migration вычисляет полный путь от ключей источника до адресата, используя алгоритм Dijkstra, чтобы минимизировать количество переходов на новом пути. Согласно алгоритму, узел C должен быть включен в путь. После определения всех узлов, составляющих весь путь, приложение настраивает поток в переключателях в обратном направлении, то есть он настраивает поток от ближайшего коммутатора к месту назначения, до самого дальнего переключателя. Поскольку все пути от места назначения до источника уже настроены, но ссылка исходного компьютера на коммутатор A, которая может быть изменена, позволяет избежать потерий пакетов.

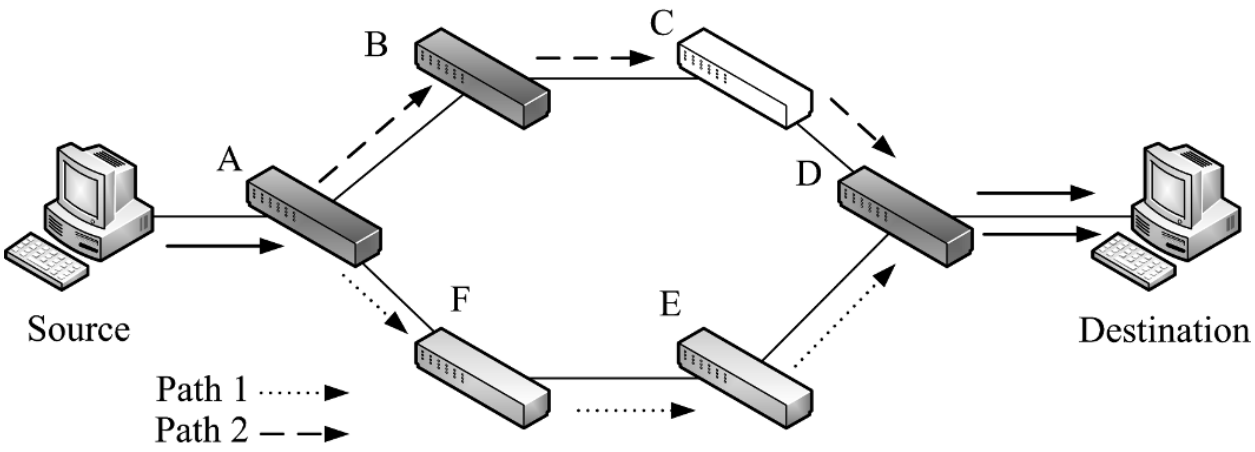


Рисунок 3.10. Работа приложения переноса потока

Открытие OpenFlow. Приложение Discovery представляет собой модифицированную версию стандартного NOX заявления. Во-первых, данное приложение совместимо с любым OpenFlow, и также разработан интерфейс для веб-сервера. Приложение Discovery также получает топологию как физических, так и виртуальных сетей. В основном, приложение Discovery реализует обнаружение Link Layer Discovery Протокол (LLDP). Этот протокол позволяет узлам передавать информацию о возможностях и текущем состоянии сетевых устройств. Реализация LLDP является необязательной в стеке протоколов станций IEEE 802 LAN. Формат релиза LLDP показан на рисунке 3.11. LLDP релиз имеет информацию о устройстве, хранящуюся в TLV структуре. Затем приложение Discovery создает LLDP-фрейм для всех портов данного коммутатора. Коммутатор передает эти релизы через свои порты. После получения первого релиза LLDP коммутатор не знает правил пересылки для этого фрейма. Переключатель переводит фрейм контроллера для анализа его содержимого. После получения релиза, контроллер запрашивает исполняемый экземпляр приложения Discovery для обработки этого фрейма. Таким образом, приложение анализирует, какой коммутатор получил релиз, из которого порта был получен фрейм, и через который порт был отправлен релиз. С помощью этой информации приложение создает данные структуры с переключателем источника, порт источника, переключатель назначения и назначение порта. Эта структура данных идентифицирует ссылку и агрегирует все ссылки сети характеризующей топологию сети.

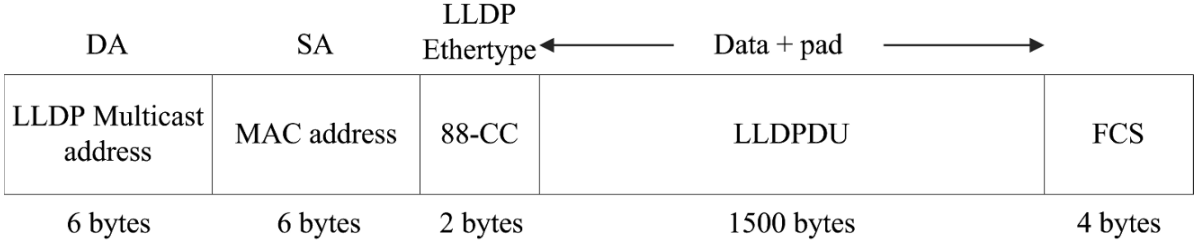
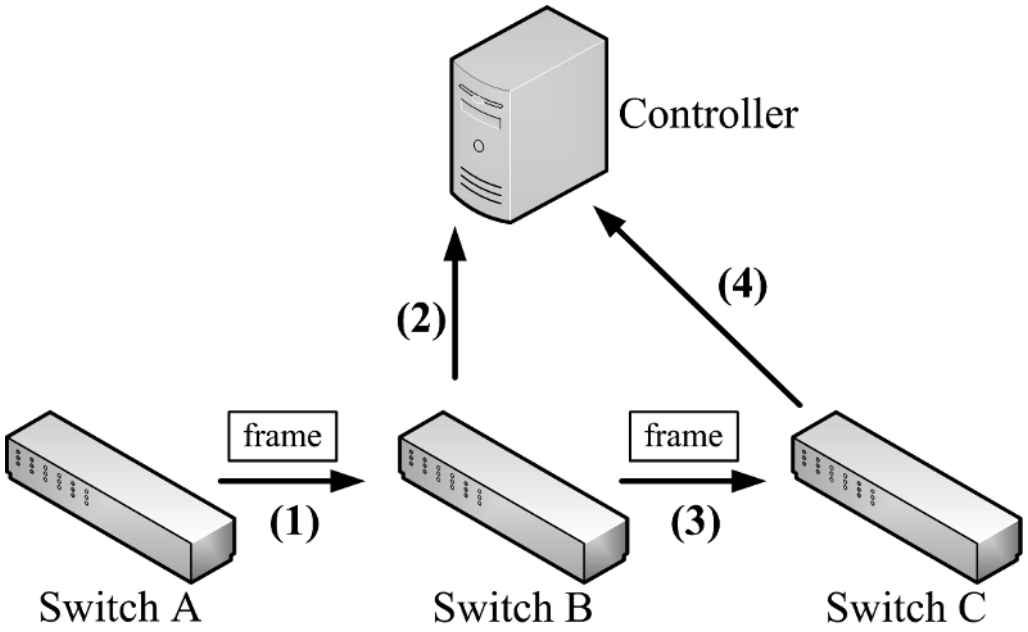
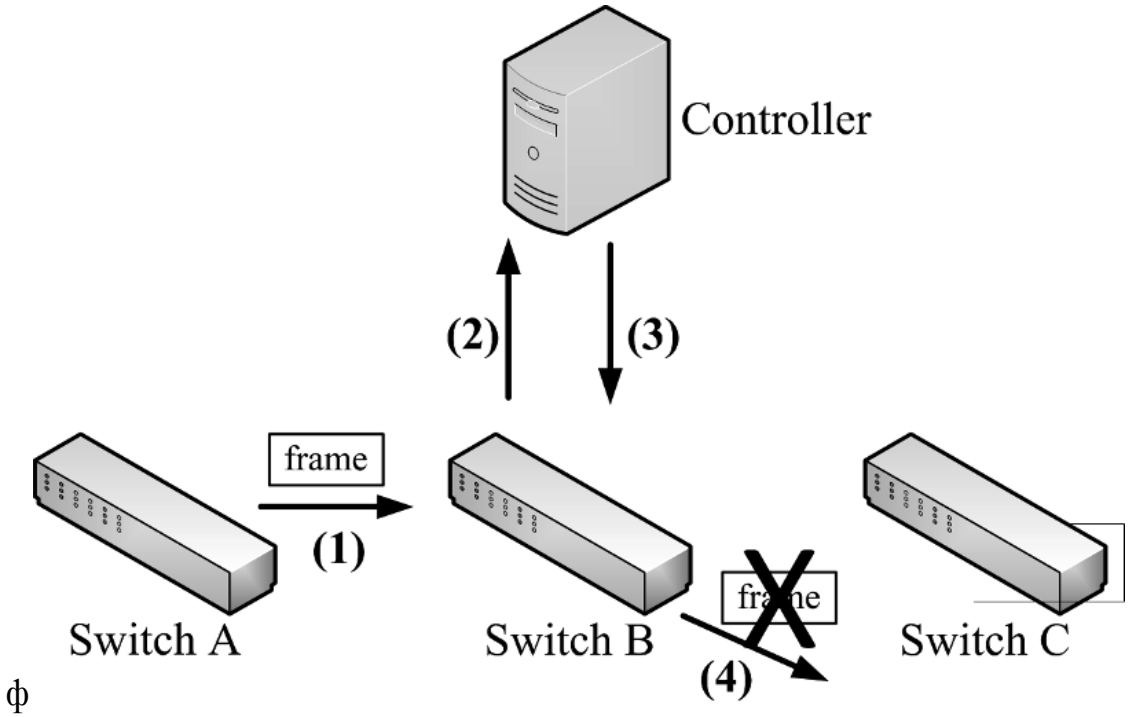


Рисунок 3.11. Формат кадра LLDP для IEEE 802.3

Алгоритм приложения Discovery NOX по умолчанию изменен на гарантирующую правильность процедуры обнаружения топологии на типе-1, среди переключателей OpenFlow. В коммутаторах типа 0 простейшая модель OpenFlow коммутатора, фрейм пересылается только в том случае, если контроллер уже настроил поток для этого фрейма. Если нет настроенного потока, фрейм будет сброшен. Однако коммутаторы Type-1 обрабатывают трафик потока, даже если нет специфического правила, настроенное контроллером, то есть по умолчанию исходные пакеты не отбрасываются автоматически. Этот факт приводит к неправильному представлению топологии сети. Предположим, что LLDP отправлен с коммутатора A на коммутатор B. С настройкой по умолчанию алгоритма, контроллер не отправит команду, чтобы удалить этот фрейм после его обработки. В этом случае B может неправильно перенаправить этот фрейм LLDP на другие переключатели, подключенные к нему. Предположим, что коммутаторы B и C напрямую подключены, но C не связано напрямую с A. В этом случае B переводит LLDP к C, следовательно контроллер ошибочно идентифицирует связь между A и C. Как следствие, предоставляется неправильная топология. Чтобы устранить эту проблему, мы изменим алгоритм по умолчанию, чтобы позволить ему всегда отправлять части команды к переключателям. Различия между двумя алгоритмами представленны на рисунке 3.12.



а) Алгоритм по умолчанию



ф

б) модифицированный алгоритм

Рисунок 3.12. Пример поведения по умолчанию и измененного алгоритма обнаружения: первые два шага обоих механизмов одинаковы, но

модифицированный алгоритм, контроллера отправляет команду drop на Switch B, которая сбрасывается

Рисунок 3.12 a) представляет алгоритм обнаружения по умолчанию. Все переключатели Типа 1. В этом примере (1) коммутатор A отправляет LLDP-кадр для переключения B, а после получая этот кадр, (2) B отправляет его на контроллер. Затем контроллер указывает, что существует связь между A и B. Контроллер не (3) переводит фрейм LLDP в C. Переключатель C получает фрейм, и, следовательно, (4) отправляет его на контроллер. Затем контроллер неправильно предполагает, что существует связь между A и C. Модифицированная версия алгоритма проиллюстрирована на рисунке 3.12b). Первые два шага такие же, как и с алгоритмом по умолчанию. Разница в том, что после этого, (3) контроллер отправляет команду сброса в B и, следовательно, (4) B отбрасывает LLDP вместо пересылки на C. Приложение по умолчанию предоставляет только физическую топологию сети, и нам также нужна информация о виртуальных сетях в прототипе. Для достижения этой цели, модифицированное приложение Discovery получает определение виртуальной сети, например идентификатор VLAN или диапазон IP, а затем он предоставляет структуру с топологией виртуальной сети, которая включает в себя все коммутаторы, которые перенаправляют трафик для этой виртуальной сети.

ПРЕДЛОЖЕНИЯ И РЕКОМЕНДАЦИИ

Технологии виртуализации как виртуальные частные сети и виртуальные локальные сети часто используются совместно. И подходы к развертыванию виртуальных сетей не всегда одинаков, порой данный выбор иногда даже не оставляется при выборе физического оборудования приобретённой от некоторых производителей, которые поддерживают крайне узкий список прототипов, подходов виртуальных сетей. Виртуальные сети так же отличаются своими топологиями, и при комбинировании многих показателей, таких как и топология сети, как прототип сети, технологии, заранее предписанные правила адресации в сети, и т.д., которые широко влияют на продуктивность данной сети. А в роль продуктивности сети выступает его пропускная способность.

В данной диссертационной работе в ходе исследования выявилось что в различных результатах каждый прототип показывает хорошие результаты в тех или иных тестах, как пример Xen прототип отличился результатами во время тестирования, который был лучше не только чем OpenFlow но так и от VMWare и ОС на аппаратном ресурсе, без использования технологий виртуализации. Но при этом OpenFlow отличался различаем в методе передачи и с расширенными возможностей. Оба прототипа имеют свои утилиты для создания и управления виртуальной средой, а так же для их миграции, контроля и установления. Которые являются основными для обеих прототипов, но так же в прототипе Xen есть модуль Xentop Gatherer который собирает данные сети, а так же Ifconfig Gatherer, The Latency Gatherer и Memory Gatherer. Данные модули являются датчиками используемые в Xen прототипе. Существует так же модуль топологий, который позволяет определять виртуальные и физические сети. Данные утилиты являются привилегиями прототипа Xen.

Но в прототипе OpenFlow, так же существуют NOX приложения, которые являются привилегиями данного прототипа. Как пример можно привести приложение «Статистика» которое предоставляет информацию о всех виртуальных коммутаторах, такую как состояние потоков, количество передаваемых пакетов и т.д. Приложение «Обнаружение» выявляет топологию сети. Но главным преимуществом OpenFlow является приложение «Spanning tree» которое позволяет избегать ненужного широковещательного трафика в сети, ненужной повторной передачи пакетов и сетевых петель передачи. В обеих прототипах поддерживается динамическая миграция без потерей пакетов и динамического реконфигурирования сетевых топологий.

Оба прототипа имеют поддержку веб интерфейса для удобной конфигурации и управления виртуальными сетями. Дополнительные сервисы могут быть добавлены в платформу виртуализации.

Как рекомендация можно добавить, что любая сеть считается эффективной когда она оптимально и эффективно исполняет свои задачи, поставленные тем или иным предприятием. Достижения эффективности можно достичь полностью проанализировав требования в сети, и использовав тот прототип, который полностью соответствует требованиям задачи. Но так и следует рассмотреть все возможности прототипов и максимально использовать их для создания оптимизированной виртуальной сети с соответствующим набором возможностей.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Джейсон Ньюман Книга GNS3: Создание Виртуальных сетей 2015
2. Отто Карлос Виртуальные Сети 2013
3. Лингжи Дуан, Джанвей Хуанг, Бийинг Шоу Когнетивные Виртуальные Сети 2013
4. Дамиан Флинн, Нигел Кейн, Элвин Моралес Создание Виртуализированных Решений 2014
5. Деннис Фоулер Частные Виртуальные Сети: Создание Правильных Соединений 1999
6. Браун С. Виртуальные Сети 2001
7. Александр Росляков Виртуальные Сети: Модели и Методы 2011
8. Девид Чиснал Гид по Xen Гипервизорам 2008
9. Джеан Метьюс, Эли Дау, Тодд Дешане Запуск Xen 2008
10. Педро Писа, Наталия Фернандес, Хьюго Карвадо Сетевая Миграция Xen и OpenFlow
11. Сиамак Азодолмолкий Программные Сети OpenFlow 2013

XÜLASƏ

İşdə virtual şəbəkələrin qurulması texnoloqiyaları araşdırılmışdır. Virtual şəbəkələrin yaradılmasında istifadə olunan əsas iki prototip (Xen və OpenFlow) üzərindən araşdırmalar və müxtəlif proqram təminatı ilə testlər aparılaraq nəticələr əldə edilmişdir. Hər bir prototipin interfeysləri, göstəriciləri və proqram təminatları araşdırılmışdır, həmçinin araşdırma nəticəsində onların bir biri üzərində olan üstünlükləri qeyd edilmişdir. Bu araşdırmaların nəticəsi olaraq demək olar ki, hər iki virtuallaşdırma prototipin fərqli istiqamətlərdə istifadədə olunan özünə məxsus proqram təminatı vardır, lakin Xen prototipin testlərinin digər prototiplərlə müqayisədə daha yaxşı nəticələr göstərməsi sübut edilmişdir. Nəticə etibarı ilə hər iki prototipin öz üstün və mənfi tərəfləri vardır, və onların istifadəsi şəbəkəyə qarşı qoyulan tələblərə əsasən qərar verilməlidir. Araşdırmaların nəticələrinə əsaslanaraq bu tələblərə uyğun olan prototip və texnologiyaların istifadəsi şəbəkə administratorları tərəfindən həyata keçirilə bilər, həmçinin araşdırma nəticələrindən bu virtual şəbəkələrin daha effektiv idarə olunması üçün və şəbəkənin optimallarşdırması üçün istifadə oluna bilər.

SUMMARY

The work reveals the technologies of installing virtual networks. The two main prototype prototypes (Xen and OpenFlow) used to create virtual networks were tested and tested with different software and the results were reached. Each prototype's interfaces, indicators, and software support have been investigated, and as a result of the research, the advantages over them have been revealed. As a result of these studies, almost all of the virtualization prototypes have their own software, which is used in different directions, but the Xen prototype's test results exceeded other prototypes. Consequently, both prototypes have their own advantages and disadvantages, and their use should be based on the requirements of the network. Based on the results of the research, the prototype and the technologies that meet these requirements can be implemented by network administrators. And it can also be used to more effectively manage these virtual networks and optimize networking.