Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической
кибернетики и компьютерных наук

ПРОГРАММНО УПРАВЛЯЕМЫЕ СЕТИ И ИХ ПРИМЕНЕНИЕ

КУРСОВАЯ РАБОТА

студента 2 курса 251 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Дергунова Дмитрия Витальевича

Научный руководитель
доцент, к. т. н. _____ В. М. Соловьев

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На данный момент традиционный способ управления сетями, в основе которых лежит стек протоколов TCP/IP является устаревшим, в связи с тем, что является недостаточно эффективым при передаче разнородного трафика [1]. В связи с этим появляются другие способы управления сетями, один из которых – программно управляемые сети (Software Defined Network). В основе программно управляемых сетей лежит разделение функций передачи и управления данными, что упрощает управление и контроль над ними по сравнению с традиционным способом управления, в котором эти функции совмещены. Также упрощается массштабирование, увеличивается надежность, что способствует ускорению развития сетей и приложений, работающих в них.

Также стоит упомянуть технологию виртуализации сетевых функций, необходимую для решения проблем, связанных со временем развертывания сетей и энергопотреблением сетевыми технологиями, в связи с тем, что текущая традиционная архитектура сетей недостаточна эффективна.

Программно управляемые сети и виртуализация сетевых функций способсвуют упрощению администрирования и массштабирования компьютерных сетей, помогают увеличить пропускную способность каналов связи, перераспределяя автоматически нагрузку на те, или иные узлы сети [2].

Благодаря этому снижаются затраты на создание и поддержание работы сети, что благоприятно влияет на интернет-провайдеров, телекоммуникационные компании и владельцев компьютерных сетей.

Обратим внимание на недостатки текущего подхода к построению компьютерных сетей, помочь решить которые стремятся выше упомянутые технологии программно управляемых сетей и виртуализации сетевых функций [3]:

1. Сложность и экономически высокая стоимость обслуживания, в связи с тем, что для управления сетью используются сетевые устройства, которые со временем становятся всё более сложными в производстве из-за неоходимости поддержи все большего числа сетевых протоколов, которых на данный момент насчитывается более 600. Это способствует ухудшению стабильности работы и подбору обслуживающего персонала. который должен быть высококвалифицированным
2. Истощение пропускной способности, ввиду того, что интернет-трафик показывает значителный рост, в связи с тем, что всё большее количество

людей становятся пользователями глобальной сети. Методы контроля трафика развиваются недостаточно быстро, в связи с чем, каналы связи становятся более нагруженными.

3. Препятствие для разработки новых сервисов, проведения экспериментов, тех или иных нововведений, ввиду того, что большинство сетевых интерфейсов имеют проприетарное программное обеспечение.
4. Постоянный рост количество сетевых протоколов для обеспечения безопасности связи и удовлетворения всех потребностей пользователей сети. При возникновении каких-либо проблем, стек протоколов TCP/IP дополняется новым протоколом, которых на данный момент, как было сказано выше, насчитывается более 600.
5. Увеличение количества облачных услуг, что подразумевает потребность пользователя к высокой скорости доступе к этим услугам.
6. Изменение моделей передвижения трафика.

К созданию архитектуры программно управляемых сетей привело несоответствие возможностей текущей архитектуры и требований рынка.

SDN – это сеть, функционирующая независимо от сетевых устройств. В этой архитектуре происходит разделение сетевых сервисов и приложений от физического оборудования.

Рассмотрим задачи, на которых фокусируется технология программно управляемых сетей:

1. Единое и полное управление сетью.
2. Разделение функций передачи данных и управления трафиком, исполюзую специальзированное программное обеспечение, способное функционировать на отдельном сети, и управляемое администратором сети.
3. Создание программно-управляемого интерфейса между сетевыми приложениями и средой транспортировки трафика. Между транспортной средой и сетевыми приложениями создать программно-управляемый интерфейс.

SDN не является улучшением работы текущего подхода к простроению сетей, это принципиально новая архитектура, направленная на устранение недостатков и качественному изменению принципов функционирования и управления компютерными сетями [4].

1 Теоретические сведения

1.1 Архитектура SDN

Архитектура программно управляемых сетей состоит из трех уровней абстракции

1. Уровень приложений, включающий в себя набор программ контроллера прикладного назначения, необходимых для осуществления управления сетью с наибольшей эффективностью.
2. Уровень управления, включающий в себя сетевую операционная система (Network Operation System). С помощью которой происходит проверка текущего статуса и работоспособности сети, а также сетевых устройств. NOS предоставляет необходимый интерфейс для управления потоками данных в компьютерной сети и сетевой инфраструктуре уровню, находящемуся выше [5].
3. Уровень инфраструктуры, который состоит из сетевых устройств и каналов передачи данных, которые представляют собой топологию сети 1.

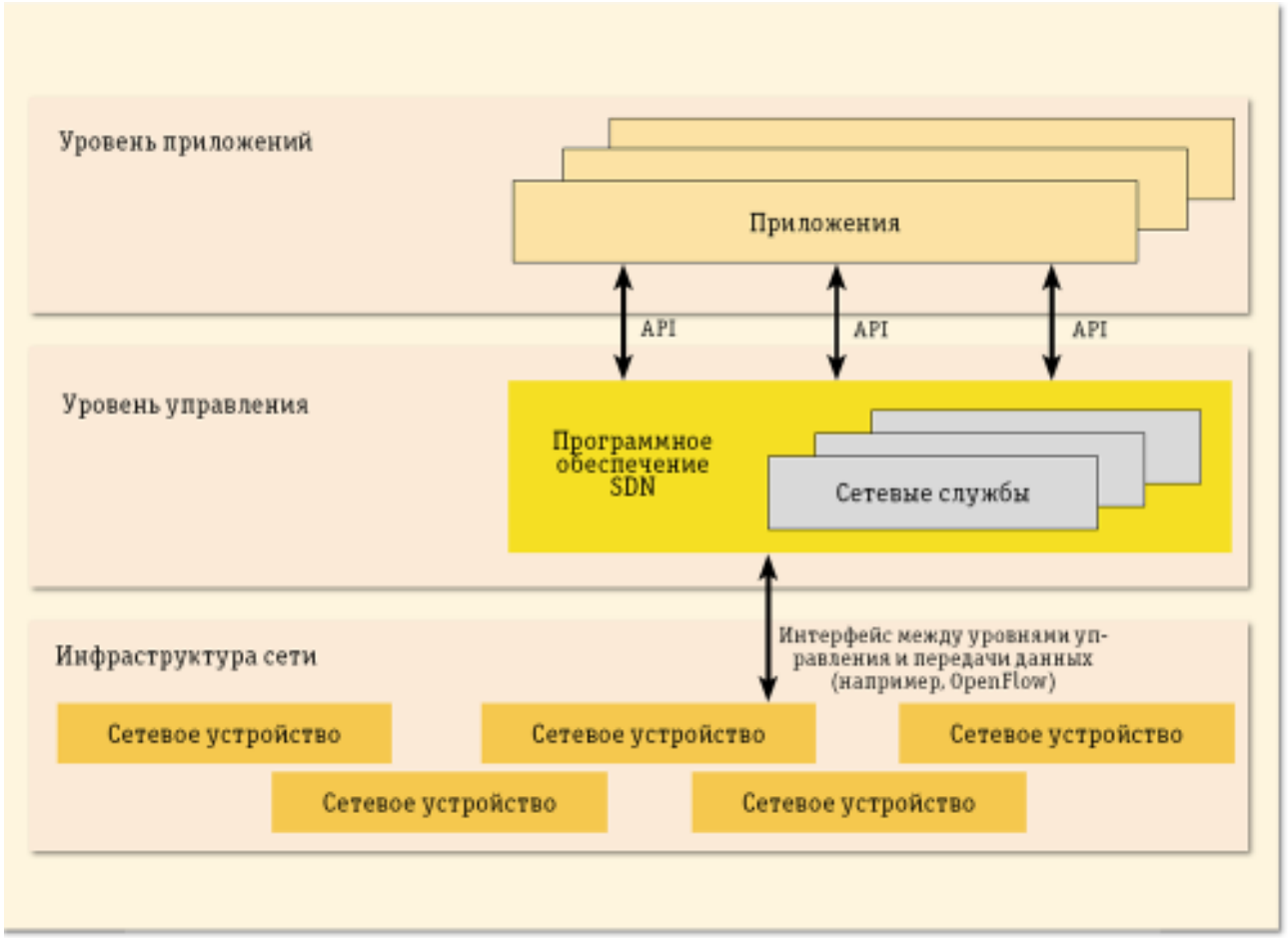

Рисунок 1 – Архитектура SDN

1.2 Протокол OpenFlow

Наиболее популярным способом унификации управления программно управляемой сетью вне зависимости от сетевого оборудования является протокол OpenFlow [6], предоставляющий возможности пользователям сети самостоятельно определять и контроллировать трафик и взаимосвязи между участниками компьютерной сети. А также управлять пропускной способностьюи качеством связи между узлами сети. Данный протокол поддерживает три типа сообщений:

1. Сообщения, имеющие тип контроллер-коммутатор необходимы для непосредственного управления коммутатором посредством контроллера. Используются для конфигурирования коммутатора, модификации, добавления и удаления записей из таблицы потоков. Инициируется контроллером.
2. Асинхронные сообщения служат для оповещения контроллера о тех или иных событиях, произошедших в сети. Инициируются коммутатором при прибытии пакета данных или изменения состояния какого-либо узла сети.
3. Симметричные сообщения инициируются коммутатором или контроллером без необходимости запроса. Используются при установлении соединения, изерения пропускной способности, качества связи и задержек в соединении.

1.3 Сетевая операционная система (NOS)

Централизованное управление сетевыми данными предполагает вынесения всего функционала управления сетью на отдельный узел сети, который принято называть контроллером. Контроллер находится под управлением сетевого администратора и представляет из себя отдельный физический сервер. Управление контроллером происходит посредством одного или более коммутаторов, обычно работающих на протоколе OpenFlow, и содержищих сетевую операционнью систему, задачей которой является предоставление интерфейса для низкоуровневого управления компьютерной сетью, её сегментами и состояниями её узлов, посредством сетевых сервисов, а также высокоуровневое управление сетью и поток данных посредством приложений [7].

Сетевая операционная система также способствует обеспечениюю до-

ступа к управлению сетью с помощью приложений и отслеживанию конфигурацию средств сети в режиме реального времени.

Аналогично операционной системе, в традиционном ее понятии, NOS обеспечивает программный интерфейс (API) для управления сетью, а также реализует механизмы управления таблицами коммутаторов: добавление, удаление, изменение правил и сбор статистики.

Ввиду этого, решение задач управления сетью выполняется фактически с помощью приложений, реализованных на основе программного интерфейса NOS, позволяющего оперировать высокоуровневыми понятиями, такими как имя пользователя или узда, а не низкоуровневыми параметрами конфигурациями, такими как IP-адрес или MAC-адрес. Это способствует более высокой абстракции над топологией компьютерной сети, однако необходимым условием является то, что сетевая операционная система должна поддерживать отображения между низкоуровневой конфигурацией и её высокоуровневой абстракцией.

2 Эмуляция компьютерной сети

2.1 Mininet

Mininet – это эмулятор компьютерной сети, включающей в себя не только хосты и коммутаторы, а также контроллеры. Это программное обеспечение позволяет разворачивать сети из произвольного количества хостов, коммутаторов в различных топологиях в рамках одной виртуальной среды. На всех узлах допускается изменение сетевой конфигурации, а также возможность использования стандартных средств, таких как ipconfig и ping. Помимо этого имеется возможность доступа к терминалу каждого узла сети. Предоставляется возможность управления трафиком через коммутаторы путем задания различных правил маршрутизации и обработки данных.

Ядром Linux, начиная с версии 2.6.24, поддерживаются механизм виртуализации – Cgroups [8], который обеспечивает процессы сетевыми интерфейсами и таблицами маршрутизации в рамках одной операционной системы. Данный механизм виртуализации позвляет запустить несколько процессов в рамказ операционной системы в изолированном окружении, что позволяет Mininet создавать коммутаторы, OpenFlow-контроллеры и хосты, а также взаимодействовать в рамках сети, смоделированной в изолированной среде. Исходый код Mininet написан на Python, за исключением некоторых системных утилит. В рамках этого эмулятора можно описать любую топологию на язке Python. Вся работа с Mininet происходит в рамках командного эмулятора mn 2.

Рисунок 2 – Командный эмулятор mn

По-умолчанию, без дополнительных параметров программное обеспечение переходит в режим интерпретатора команд. Эмулируется компьютерная сеть, состоящая из двух хостов, коммутатора и и контроллера, если в параметрах запускане указано иное.

Mininet предоставляет некоторые собственные команды для управления виртуальной компьютерной сетью. Ниже приведены некоторые из возможностей [9].

Команда nodes повзялет вывести список узлов всех типов, зарегистрированных в данной сети 3.

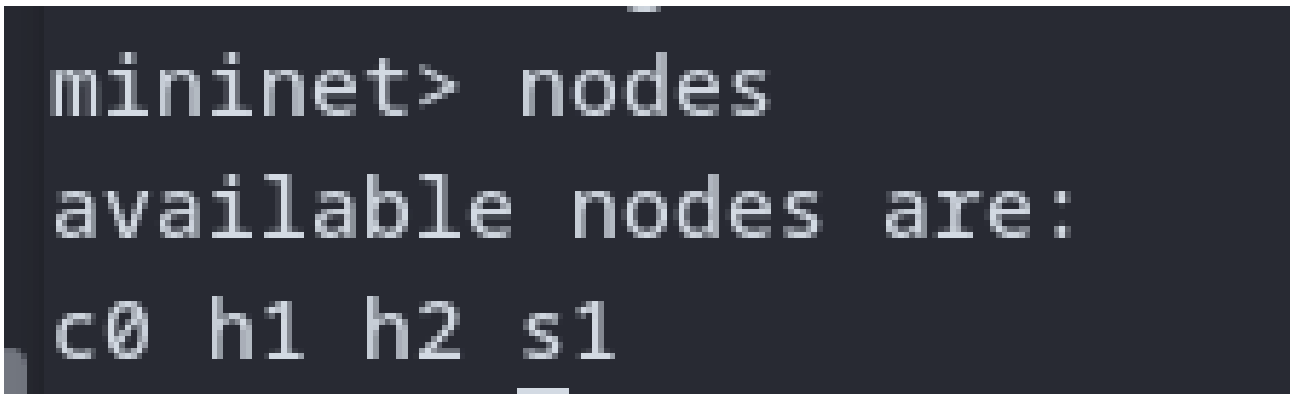

Рисунок 3 – Список хостов

Команда net позмоляет посмотреть текущую топологию сети и сопо-

ставление портов коммутатора и хостов 4.

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
```

Рисунок 4 – Топология сети

Команда ifconfig, по аналогии со стоим системным аналогом предоставляет возможность вывода конфигурации сетевого интерфейса конкретного узла сети 5.

```
mininet> h1 ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::583f:dfff:fee3:4e31  prefixlen 64  scopeid 0x20<link>
        ether 5a:3f:df:e3:4e:31  txqueuelen 1000  (Ethernet)
        RX packets 9  bytes 706 (706.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 796 (796.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Рисунок 5 – Конфигурация сетевого интерфейса

С помощью команды link можно управлять состоянием любого из портов 6.

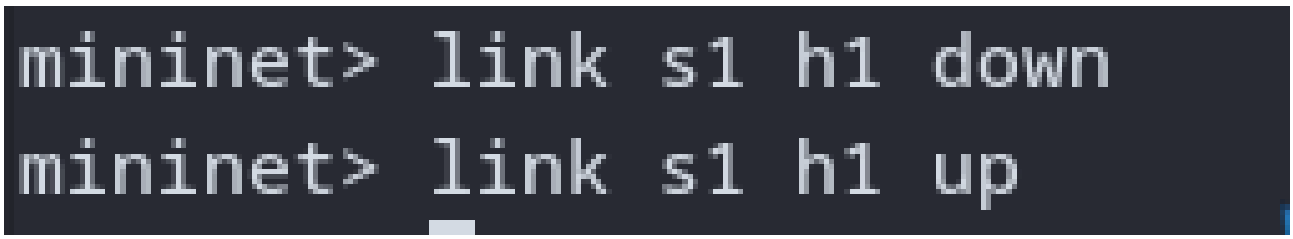

Рисунок 6 – Управление портами коммутатора

Команда route, аналогично системной команде ядра Linux позволяет просмотреть таблицу маршрутизации для узла созданной виртуальной компьютерной сети 7.

```
mininet> h1 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.0.0.0       U     0      0        0 h1-eth0
```

Рисунок 7 – Таблица маршрутизации

Также предоставляется привычная команда ping 8.

```
mininet> h1 ping -c4 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=7.97 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=1.05 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.170 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.231 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 0.170/2.358/7.973/3.260 ms
```

Рисунок 8 – Пинг

А также команда pingall 9.

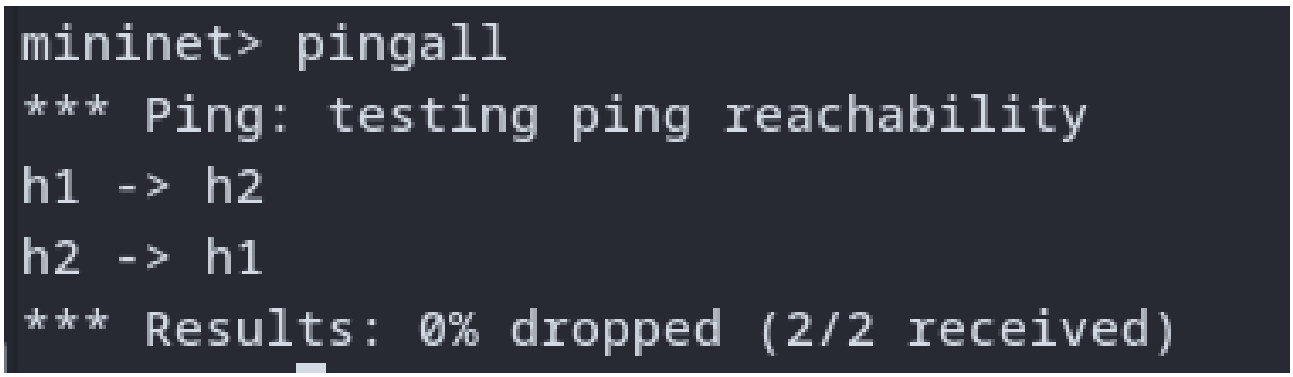

Рисунок 9 – Пинг каждого с каждым

Доступна возможность тестирования пропускной способности сети с помощьюкоманды iperf ??.

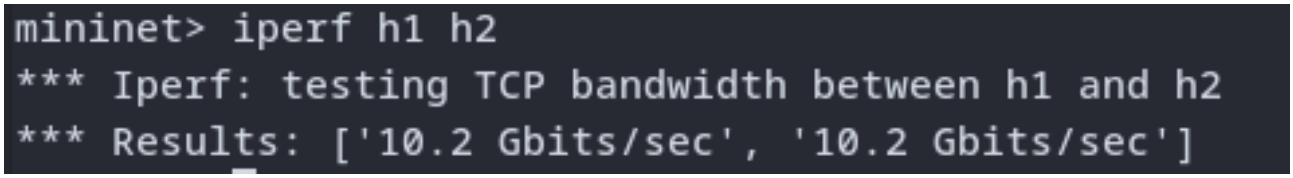

Рисунок 10 – Пропускная способность между узлами

Топология сети определяет количество узлов того или иного типа в виртуальной сети. С этого начинается эмуляция компьютерной сети посредством Mininet

Из коробки Mininet предлагает четыре варианта топологии создаваемой сети.

1. minimal – конфигурация по-умолчанию. Создается два хоста, коммутатор и OpenFlow-контроллер 11.

```
root@eab24e402068:~/sdn-course-work/docker/ryu# mn
*** Error setting resource limits. Mininet's performance may be affected.
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Рисунок 11 – Minimal топология

2. single – аналогично конфигурации minimal, но предоставляется возможность указать количество хостов 12.

```
root@eab24e402068:~/sdn-course-work/docker/ryu# mn --topo single,8
*** Error setting resource limits. Mininet's performance may be affected.
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1) (h7, s1) (h8, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Рисунок 12 – Single топология

3. linear – топология, при которой каждый хост подключен к собственному коммутатору, которые в свою очередь соединены между собой 13.

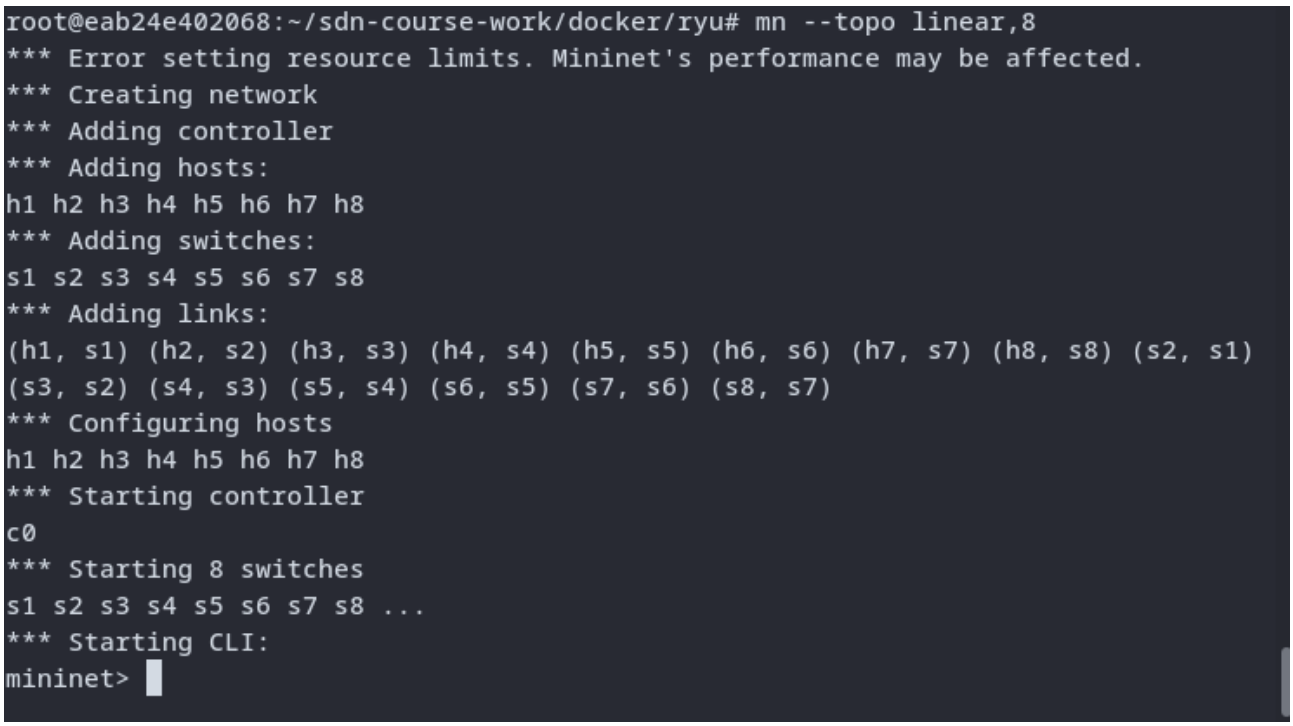

Рисунок 13 – Линейная топология

4. tree – Древовидная топология, дополнительные параметры указывают глубину иерархии коммутаторов и число подключенных хостов 14.

```
root@eab24e402068:~/sdn-course-work/docker/ryu# mn --topo tree,depth=3,fanout=4
*** Error setting resource limits. Mininet's performance may be affected.
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h2
3 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43
 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63
h64
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21
*** Adding links:
(s1, s2) (s1, s7) (s1, s12) (s1, s17) (s2, s3) (s2, s4) (s2, s5) (s2, s6) (s3, h1
) (s3, h2) (s3, h3) (s3, h4) (s4, h5) (s4, h6) (s4, h7) (s4, h8) (s5, h9) (s5, h1
0) (s5, h11) (s5, h12) (s6, h13) (s6, h14) (s6, h15) (s6, h16) (s7, s8) (s7, s9)
(s7, s10) (s7, s11) (s8, h17) (s8, h18) (s8, h19) (s8, h20) (s9, h21) (s9, h22) (
s9, h23) (s9, h24) (s10, h25) (s10, h26) (s10, h27) (s10, h28) (s11, h29) (s11, h
30) (s11, h31) (s11, h32) (s12, s13) (s12, s14) (s12, s15) (s12, s16) (s13, h33)
(s13, h34) (s13, h35) (s13, h36) (s14, h37) (s14, h38) (s14, h39) (s14, h40) (s15
, h41) (s15, h42) (s15, h43) (s15, h44) (s16, h45) (s16, h46) (s16, h47) (s16, h4
8) (s17, s18) (s17, s19) (s17, s20) (s17, s21) (s18, h49) (s18, h50) (s18, h51) (
s18, h52) (s19, h53) (s19, h54) (s19, h55) (s19, h56) (s20, h57) (s20, h58) (s20,
 h59) (s20, h60) (s21, h61) (s21, h62) (s21, h63) (s21, h64)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h2
3 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43
 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63
h64
*** Starting controller
c0
*** Starting 21 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21 ...
*** Starting CLI:
mininet>
```

Рисунок 14 – Древовидная топология

2.2 Задание топологии сети

Также доступна возможность создание собственной топологии, описанной на языке Python с помощью ключе –custom Сделать это можно используя следующую команду запуска 15.

```
root@eab24e402068:~/sdn-course-work/tex/source# mn --topo mytopo --custom topo.py
 --link=tc
```

Рисунок 15 – Собственная топология

В файле topo.py описывается структура виртуальной компьютерной сети на языке Python.

```
class MyTopo( Topo ):
    "Network topology"

    def __init__( self ):
        Topo.__init__( self )

        # Add hosts
        host1 = self.addHost( 'h1' )
        host2 = self.addHost( 'h2' )
        host3 = self.addHost( 'h3' )
        host4 = self.addHost( 'h4' )
        host5 = self.addHost( 'h5' )
        host6 = self.addHost( 'h6' )
        host7 = self.addHost( 'h7' )
        host8 = self.addHost( 'h8' )
        host9 = self.addHost( 'h9' )

        # Add switches
        switch1 = self.addSwitch( 's1' )
        switch2 = self.addSwitch( 's2' )
        switch3 = self.addSwitch( 's3' )
        mainSwitch = self.addSwitch( 's4' )

        # Connect 4 hosts to first switch
        self.addLink( host1, switch1 )
        self.addLink( host2, switch1 )
        # Limit bandwidth to 40Mb/s
        self.addLink( host3, switch1, bw=40 )
        self.addLink( host4, switch1 )

        # Connect 4-7 hosts to second switch
        # Limit bandwidth to 20Mb/s
        self.addLink( host4, switch2, bw=20 )
        self.addLink( host5, switch2 )
        self.addLink( host6, switch2 )
        self.addLink( host7, switch2 )

        # Connect 8-9 hosts to third switch
        self.addLink( host8, switch3 )
        self.addLink( host9, switch3 )

```

```
42          # Connect all switches to main switch
43          # Limit bandwidth to 100Mb/s
44          self.addLink( switch1, mainSwitch, bw=100 )
45          self.addLink( switch2, mainSwitch, bw=100 )
46          self.addLink( switch3, mainSwitch, bw=100 )
```

После выполнения данного скрипта создается сетьсо следующей топологией 16

```
root@eab24e402068:~/sdn-course-work/tex/source# mn --topo mytopo --custom topo.py
 --link=tc
*** Error setting resource limits. Mininet's performance may be affected.
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s1) (40.00Mbit) (40.00Mbit) (h3, s1) (h4, s1) (20.00Mbit) (20.00Mbi
t) (h4, s2) (h5, s2) (h6, s2) (h7, s2) (s1, s3) (s2, s3)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...(40.00Mbit) (20.00Mbit)
*** Starting CLI:
```

Рисунок 16 – Топология сети

Полный код программы приведен в приложении А.

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы были описаны недостатки текущей сетевой архитектуры и рассмотрен новый подход к управлению компютерными сетями с помощью программно управляемых сетей и средств виртуализации сетевых функций. Также было рассмотрено программное обеспечение для эмуляции сети с различной топологией и возможностью поледующего управления этой сетью выше описанной технологией SDN.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Программно-конфигурируемые сети [Электронный ресурс]. — URL: http://arccn.ru/sdn (Дата обращения 18.05.2018). Загл. с экр. Яз. рус.

2 Программно-конфигурируемые сети [Электронный ресурс]. — URL: http://www.osp.ru/os/2012/09/13032491 (дата обращения 19.05.2018). Загл. с экр. Яз. рус.

3 Технологии программно-конфигурируемых сетей и виртуализации сетевых функций [Электронный ресурс]. — URL: http://www.mctrewards.ru/texnologii/net (дата обращения 19.05.2018). Загл. с экр. Яз. рус.

4 Что такое программно-конфигурируемые сети и чем они не являются [Электронный ресурс]. — URL: http://shalaginov.com/2014/07/13/sdn/ (Дата обращения 19.05.2018). Загл. с экр. Яз. рус.

5 Программно определяемые сети (Software Defined Networks): настоящее и будущее [Электронный ресурс]. — URL: https://habr.com/company/hpe/blog/160531/ (дата обращения 25.07.2018). Загл. с экр. Яз. рус.

6 SDN / OpenFlow / Flowgrammable [Электронный ресурс]. — URL: http://flowgrammable.org/sdn/openflow/ (дата обращения 20.05.2018). Загл. с экр. Яз. рус.

7 Программно-определяемые сети [Электронный ресурс]. — URL: http://www.tadviser.ru/index.php/ (дата обращения 22.07.2018). Загл. с экр. Яз. рус.

8 cgroups [Электронный ресурс]. — URL: https://ru.wikipedia.org/wiki/Cgroups (дата обращения 27.07.2018). Загл. с экр. Яз. рус.

9 Mininet Walkthrough [Электронный ресурс]. — URL: http://mininet.org/walkthrough/#interact-with-hosts-and-switches (дата обращения 27.07.2018). Загл. с экр. Яз. рус.

ПРИЛОЖЕНИЕ А

Декларативное описание топологии сети

```
1  from mininet.topo import Topo
2
3  class MyTopo( Topo ):
4      "Network topology"
5
6      def __init__( self ):
7          Topo.__init__( self )
8
9          # Add hosts
10         host1 = self.addHost( 'h1' )
11         host2 = self.addHost( 'h2' )
12         host3 = self.addHost( 'h3' )
13         host4 = self.addHost( 'h4' )
14         host5 = self.addHost( 'h5' )
15         host6 = self.addHost( 'h6' )
16         host7 = self.addHost( 'h7' )
17         host8 = self.addHost( 'h8' )
18         host9 = self.addHost( 'h9' )
19
20         # Add switches
21         switch1 = self.addSwitch( 's1' )
22         switch2 = self.addSwitch( 's2' )
23         switch3 = self.addSwitch( 's3' )
24         mainSwitch = self.addSwitch( 's4' )
25
26         # Connect 4 hosts to first switch
27         self.addLink( host1, switch1 )
28         self.addLink( host2, switch1 )
29         # Limit bandwidth to 40Mb/s
30         self.addLink( host3, switch1, bw=40 )
31         self.addLink( host4, switch1 )
32
33         # Connect 4-7 hosts to second switch
34         # Limit bandwidth to 20Mb/s
35         self.addLink( host4, switch2, bw=20 )
36         self.addLink( host5, switch2 )
37         self.addLink( host6, switch2 )
38         self.addLink( host7, switch2 )
39
```

```
40          # Connect 8-9 hosts to third switch
41          self.addLink( host8, switch3 )
42          self.addLink( host9, switch3 )
43
44          # Connect all switches to main switch
45          # Limit bandwidth to 100Mb/s
46          self.addLink( switch1, mainSwitch, bw=100 )
47          self.addLink( switch2, mainSwitch, bw=100 )
48          self.addLink( switch3, mainSwitch, bw=100 )
49
50
51  topos = { 'mytopo': ( lambda: MyTopo() ) }
```

ПРИЛОЖЕНИЕ Б

USB-накопитель с отчетом о выполненной работе

На приложенном USB-накопителе можно ознакомиться со следующими файлами:

Папка `tex` — LaTeX- вариант курсовой работы;

`SoftwareDefinedNetworks.pdf` — курсовая работа.

`typo.py` — описание топологии сети на языке Python.