第一题：搭建基于SDN的网络架构

问题1. 简要描述网络环境的搭建方案，包括采用的软硬件及其在网络中的作用

- 软件描述及其作用
  - **mininet**：能够仿真由虚拟的主机，交换机以及控制器创建的网络。
  - **OpenDayLight**：SDN控制器并向用户提供北向接口。
- 环境搭建
  - 系统环境描述： **ubuntu16.04**
  - 安装**mininet**：
    `sudo apt-get install mininet`
  - 安装**OpenDayLight**
    1. 在官网下载OpenDayLight的BORON SR3版本
    2. 打开Linux终端切换至**root**权限。
    3. 输入以下命令安装OpenDayLight组件：

```
feature:install odl-restconf
feature:install odl-dlux-core
feature:install odl-dlux-node
feature:install odl-dlux-yangui
feature:install odl-l2switch-switch-ui
```

    4. 上述环境安装完毕后，在当前文件夹中的/bin目录下运行./**karaf**文件，效果如下：

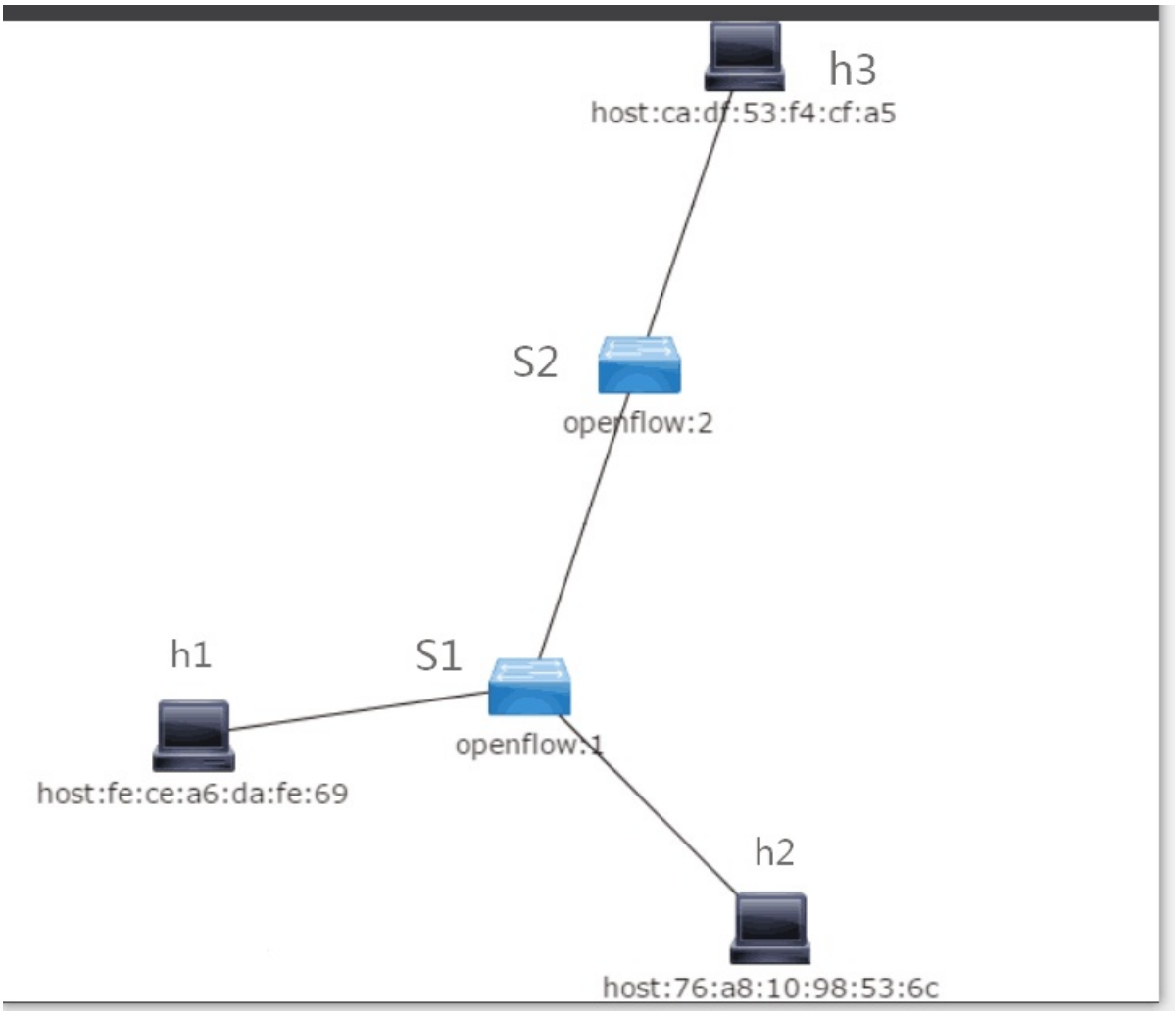
```
mininet> h1 ping -c5 h2
PING 2.2.3.1 (2.2.3.1) 56(84) bytes of data.
64 bytes from 2.2.3.1: icmp_seq=1 ttl=64 time=0.204 ms
64 bytes from 2.2.3.1: icmp_seq=2 ttl=64 time=0.038 ms
64 bytes from 2.2.3.1: icmp_seq=3 ttl=64 time=0.047 ms
64 bytes from 2.2.3.1: icmp_seq=4 ttl=64 time=0.104 ms
64 bytes from 2.2.3.1: icmp_seq=5 ttl=64 time=0.107 ms

--- 2.2.3.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 0.038/0.100/0.204/0.059 ms
```

    5. 在浏览器中输入：localhost:8181/index.html，帐号和密码都是**admin**
    6. 重新打开终端，以**mininet**提供的Python接口设计拓扑结构 `sudo mn --custom pingall.py --topo mytopo`

```python
#!/usr/bin/env python3
# coding=utf-8

from mininet.topo import Topo

class MyTopo(Topo):
    def __init__(self):
        Topo.__init__(self)
        leftSwitch = self.addSwitch("s1")
        rightSwitch = self.addSwitch("s2")

        leftHost = self.addHost("h1")
        midHost = self.addHost("h2")
        rightHost = self.addHost("h3")

        self.addLink(leftHost,leftSwitch)
        self.addLink(midHost,leftSwitch)
        self.addLink(rightHost,rightSwitch)

        self.addLink(leftSwitch,rightSwitch)

topos = {"mytopo":(lambda:MyTopo())}
```

得到下面的拓扑结构：

问题2. 拓扑示意图



问题3. 上图中两两互通的截图示意图

```
mininet> pingall
*** Ping: testing ping reachability
h2 -> h3 h1
h3 -> h2 h1
h1 -> h2 h3
*** Results: 0% dropped (6/6 received)
mininet>
```

第二题：初步分析基于OpenFlow的SDN网络控制功能

系统环境：**ubuntu16.04**
模拟环境：**Mininet**
控制器：**POX**
交换机：**OpenVSwitch**

问题1. 下发流表项实现h1和h2，h2和h3不能互通，h1和h3可互通，当前查询到的流表信息和Ping测试结果

- 初始流表截图：

```
root@mininet-vm:~# ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x2a00000000000030, duration=8.804s, table=0, n_packets=2, n_bytes=140,
idle_timeout=600, hard_timeout=300, idle_age=3, priority=10,dl_src=fe:ce:a6:da:f
e:69,dl_dst=76:a8:10:98:53:6c actions=output:2
 cookie=0x2a00000000000031, duration=8.804s, table=0, n_packets=2, n_bytes=140,
idle_timeout=600, hard_timeout=300, idle_age=3, priority=10,dl_src=76:a8:10:98:5
3:6c,dl_dst=fe:ce:a6:da:fe:69 actions=output:1
 cookie=0x2b00000000000032, duration=23.744s, table=0, n_packets=8, n_bytes=560,
 idle_age=3, priority=2,in_port=3 actions=output:1,output:2
 cookie=0x2b00000000000033, duration=23.743s, table=0, n_packets=9, n_bytes=658,
 idle_age=3, priority=2,in_port=1 actions=output:3,output:2,CONTROLLER:65535
 cookie=0x2b00000000000034, duration=23.742s, table=0, n_packets=8, n_bytes=560,
 idle_age=3, priority=2,in_port=2 actions=output:3,output:1,CONTROLLER:65535
 cookie=0x2b00000000000015, duration=27.71s, table=0, n_packets=5, n_bytes=425,
idle_age=2, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b00000000000015, duration=27.71s, table=0, n_packets=0, n_bytes=0, id
le_age=27, priority=0 actions=drop
```

- 下发流表**操作**截图：

```
le_age=27, priority=0 actions=drop
root@mininet-vm:~# ovs-ofctl add-flow s1 priority=2,in_port=2,actions=drop
 cookie=0x2b00000000000015, duration=216.848s, table=0, n_packets=0, n_bytes=0,
idle_age=216, priority=0 actions=drop
root@mininet-vm:~# ovs-ofctl add-flow s1 priority=2,in_port=1,actions=output:3
root@mininet-vm:~# ovs-ofctl add-flow s1 priority=2,in_port=3,actions=output:1
root@mininet-vm:~#
```

- 下发后流表截图：

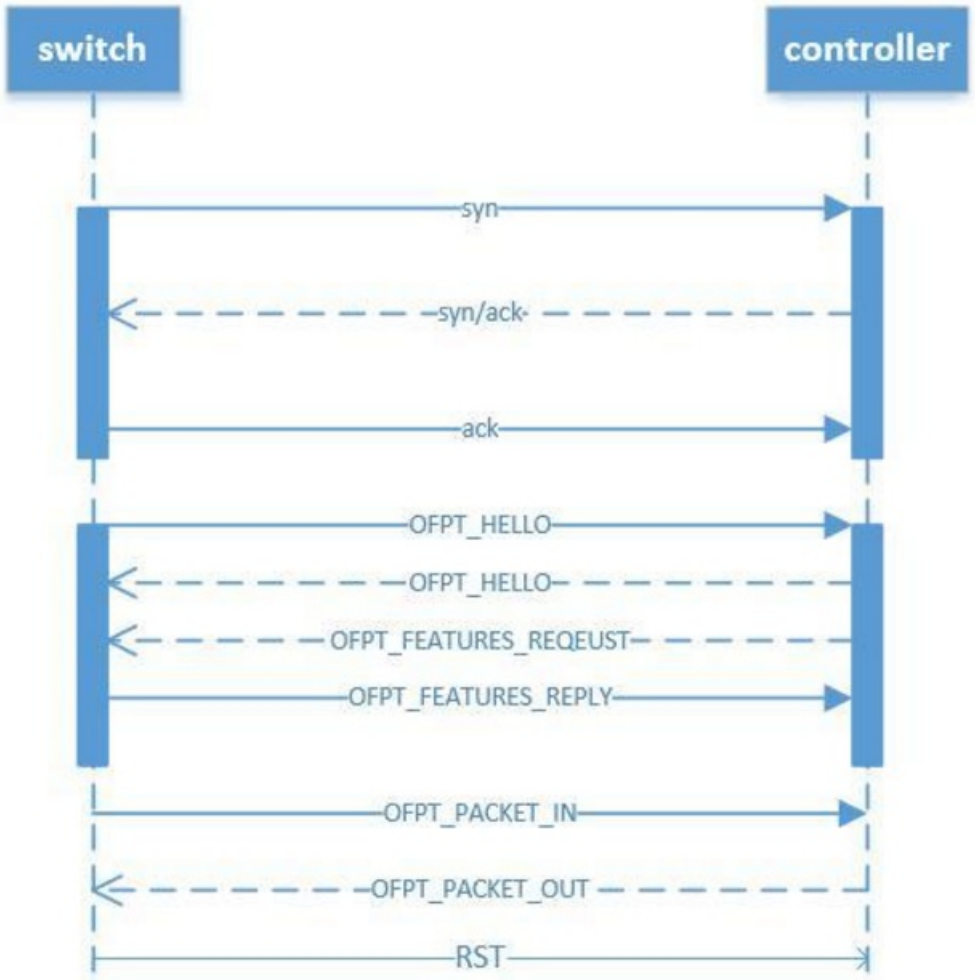
```
root@mininet-vm:~# ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=383.143s, table=0, n_packets=12, n_bytes=728, idle_age=42,
 priority=2,in_port=3 actions=output:1
 cookie=0x0, duration=428.86s, table=0, n_packets=16, n_bytes=1064, idle_age=42,
 priority=2,in_port=1 actions=output:3
 cookie=0x0, duration=527.237s, table=0, n_packets=27, n_bytes=1414, idle_age=39
, priority=2,in_port=2 actions=drop
 cookie=0x2b00000000000015, duration=713.933s, table=0, n_packets=142, n_bytes=1
2070, idle_age=4, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b00000000000015, duration=713.933s, table=0, n_packets=0, n_bytes=0,
idle_age=713, priority=0 actions=drop
root@mininet-vm:~#
```

- ping测试截图：

```
mininet> pingall
*** Ping: testing ping reachability
h3 -> h1 X
h1 -> h3 X
h2 -> X X
*** Results: 66% dropped (2/6 received)
mininet>
```

问题2. 给出报文结构图，简要分析协议的作用；结合捕获的SDN相关协议，分析其报文结构并简要描述该类报文的作用。

- 报文结构图：



- 简要描述协议作用：
  OpemFlow协议支三种类型结构：**Controller-to-Switch**,**asynchronous**和**symmetric**。
  OpenFlow通过这三种消息来保持控制器和交换机之间的通信。常用的消息主要是：**Hello**,**Packet_in**,**Packet_out**和**Flow_mod**等。其中**Hello**,**Feature**,**Echo**消息分别包含**REQUEST**与**REPLY**头部，每一个消息头部**REQUEST**与**REPLY**的**Transaction ID**相同。
  OpenFlow协议可以通过**Hello**消息让控制器与交换机建立连接，建立连接以后，通过**Packet_in**和**Packet_out**消息实现交换机与控制器信息的相互转发，再通过**Flow_mod**消息添加，删除，修改**OpenFlow**交换机的流表信息，实现互相连通。
- 捕获的报文描述：
  在本次仿真实验中，控制器使用Opendaylight，控制器IP为192.168.184.128，Mininet的Ip为192.168.184.129。Openfliw协议包括许多的消息。分析openflow报文结构如下：
  Openflow定义了三种主要消息：Controller-to-switch,asynchronous和symmertric，三种消息中包含了许多子消息，举例如下：
  Packet_in消息
  当交换机碰到新数据包不知道要如何处理的时候，或者action要求发送给控制器，那么交换机就会用packet_in消息发送给控制器。一般将数据包缓存在交换机中，将有效的数据包信息和缓存id发送给控制器。如果交换机不支持缓存或者内存耗尽了，那么就把整个数据包放在数据部分发给控制器，并且缓存id为-1。
  抓到的Packet_in消息格式如下：

```
▼ OpenFlow 1.3
    Version: 1.3 (0x04)
    Type: OFPT_PACKET_IN (10)
    Length: 127
    Transaction ID: 0
    Buffer ID: OFP_NO_BUFFER (0xffffffff)
    Total length: 85
    Reason: OFPR_ACTION (1)
    Table ID: 0
    Cookie: 0x2b0000000000000f
  ▶ Match
    Pad: 0000
  ▶ Data
```

  Version表示的是当前所使用的的openflow协议的版本，这里使用的是openflow1.3。Type表示消息类型，还有有Packet_in，Packet_out，OFPT_STATS_REQUEST等。Transaction ID为用户序列号。
  Buffer ID为packet_in事件所携带的数据包在交换机中的缓存区ID。Total length为data端总长度。Reason为packet_in事件产生的原因。Match为流表匹配规则。Data为报文具体数据内容地址。

问题3. 简要叙述在传统分布式网络中实现具体要求1的步骤，结合前面的操作分析与SDN实现之间的差别。

在传统的分布式网络中，要想实现上面的要求，需要用到访问控制列表（ACL），实现要求一的步骤。在本次实验中，h1的ip地址为1.1.1.1，接在S1的e1口上。h2的ip地址为1.1.2.1，接在S1的e2口上。h3的ip地址为1.1.3.1，接在S2的e1口上。S1通过e3口与S2的e2口相连。

```
S1: vlan 10
Int vlan 10
Ip address 1.1.1.0 255.255.255.0
Interface e1
Switch access vlan10
Vlan 20
Int vlan 20
Ip address 1.1.2.0 255.255.255.0
Interface e2
Switch access vlan 20
Interface e3
Switch mode trunk
S2: vlan 30
Int vlan 30
Ip address 1.1.3.0 255.255.255.0
Interface e1
Switch access vlan 30
Interface e2
Switch mode trunk
```

S1交换机中创建一个访问列表，实现如下配置：

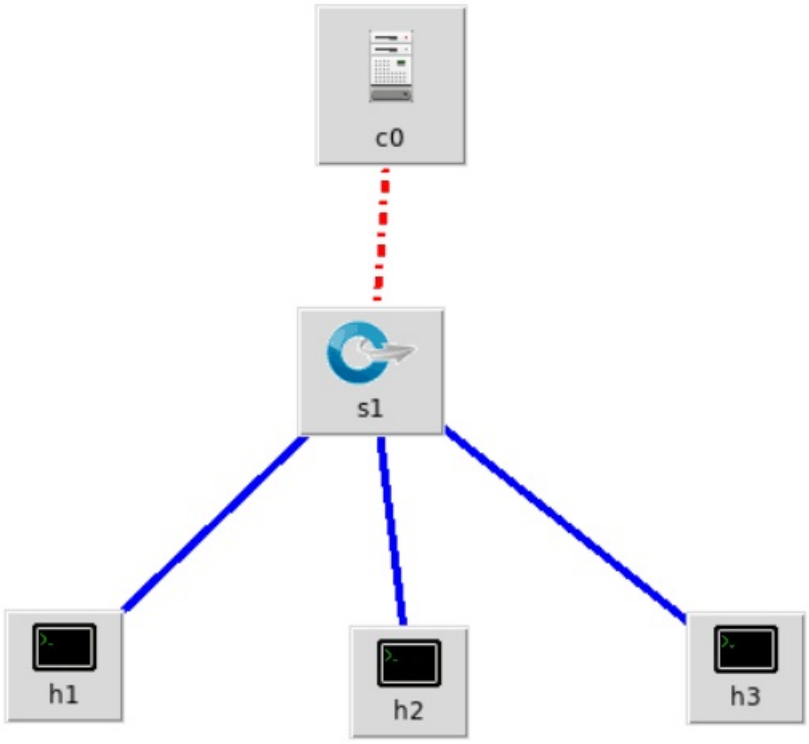
```
S1: Ip access-list standard r1  //创建一个名为r1的标准ip访问列表
  Permit 1.1.1.0 0.0.0.255
  Deny any
  Interface e3
  Ip access-group r1 out
```

以上步骤可以实现要求一。
传统分布式网络与SDN的区别是传统分布式网络对于不同网段需要划分VLAN，并且要在交换机上设置相应的访问列表。没有中心的控制节点，所需要的操作基本都基于交换机并在交换机上布置实现。而SDN则有一个控制器负责收集整个网络的拓扑、流量等信息，计算流量转发路径，通过OpenFlow协议将转发表项下发流表给交换机，交换机按照表项执行转发动作，十分的便捷。

第三题：传输层数据报控制应用的开发

系统环境：**ubuntu16.04**
模拟器：**Mininet**
控制器：**POX**
交换机：**OpenVSwitch**

问题1. 下发流表项实现h1和h2，h2和h3不能互通，h1和h3可互通，当前查询到的流表信息和Ping测试结果



问题2. 列出使用的北向API并简要介绍其功能

- `def _handle_PacketIn(self,event)`
  功能：采用事件驱动的模式处理每一个到达交换机的数据包
- `packet = event.parsed`
  功能：将发生的事件解析为具体的数据包
- `ipv4 = packet.find('ipv4')`
  功能：获得解析的数据包中的IPV4头部信息，TCP，UDP等的信息使用相同的方式获得
- `msg = of.ofp_flow_mod()`
  功能：修改当前的交换机的流表信息，且根据OpenFlow协议的具体信息对获得流表进行更新，具体的修改行为见下述代码
- `self.connection.send(msg)`
  功能：将更新后的流表信息发送给交换机

问题3. 结合图文说明程序能实现的具体要求，描述图文必须能说明实现方式为SDN而非其他的本地拦截方式。

- 首先，如下代码可以捕获数据包的信息

```
ipp = packet.find("ipv4")
tcpp = packet.find("tcp")
udpp = packet.find("udpp")
```

且捕获到的数据信息如下：

```
[UDP 59433>5001 l:1478 c:598b]    //udp
[TCP 53238>5001 seq:2329530726 ack:0 f:S]    //tcp
[IP+UDP 10.0.0.3>10.0.0.1 (cs:7db5 v:4 hl:5 l:1498 t:64)]    //ipv4
[IP+TCP 10.0.0.3>10.0.0.1 (cs:26b9 v:4 hl:5 l:60 t:64)]    //ipv6
```

- 根据相应的解析代码：

```
if ipp != None:
        strIpp = str(ipp)
        if strIpp.split(' ')[0][1:].split('+')[1] == 'TCP' and protocol == 'TCP':
            tcpp = packet.find('tcp')
        elif strIpp.split(' ')[0][1:].split('+')[1] == 'UDP' and protocol == 'UDP':
            udpp = packet.find('udp')
        else:
            return

        if(tcpp != None):
            strTcpp = str(tcpp)
            listTcp = strTcpp.split(' ')[1].split('>')
            if(strTcpp.split(' ')[1].split('>')[1] == clientPort[1] and listIp[1].split('>')[0] == clientAddr and listIp[1].split('>')[1] == serverAddr):
                drop()
                return

        if udpp != None:
            strUdpp = str(udpp)
            listUdp = strUdpp.split(' ')[1].split('>')
            if(strUdpp.split(' ')[1].split('>')[1] == clientPort[1] and listIp[1].split('>')[0] == clientAddr and listIp[1].split('>')[1] == serverAddr):
                drop()
                return
```

对上述的从数据包中获得的头部信息以及从用户处获得丢弃信息进行对比，从而得到想要丢弃的数据包，即可以对OpenFlow交换机下发流表。

- 对OpenFlow交换机下发流表的代码如下：

```
msg = of.ofp_flow_mod()
msg.match = of.ofp_match.from_packet(packet, event.port)
msg.idle_timeout = 10
msg.hard_timeout = 30
msg.actions.append(of.ofp_action_output(port = port))
msg.data = event.ofp # 6a
self.connection.send(msg)
```

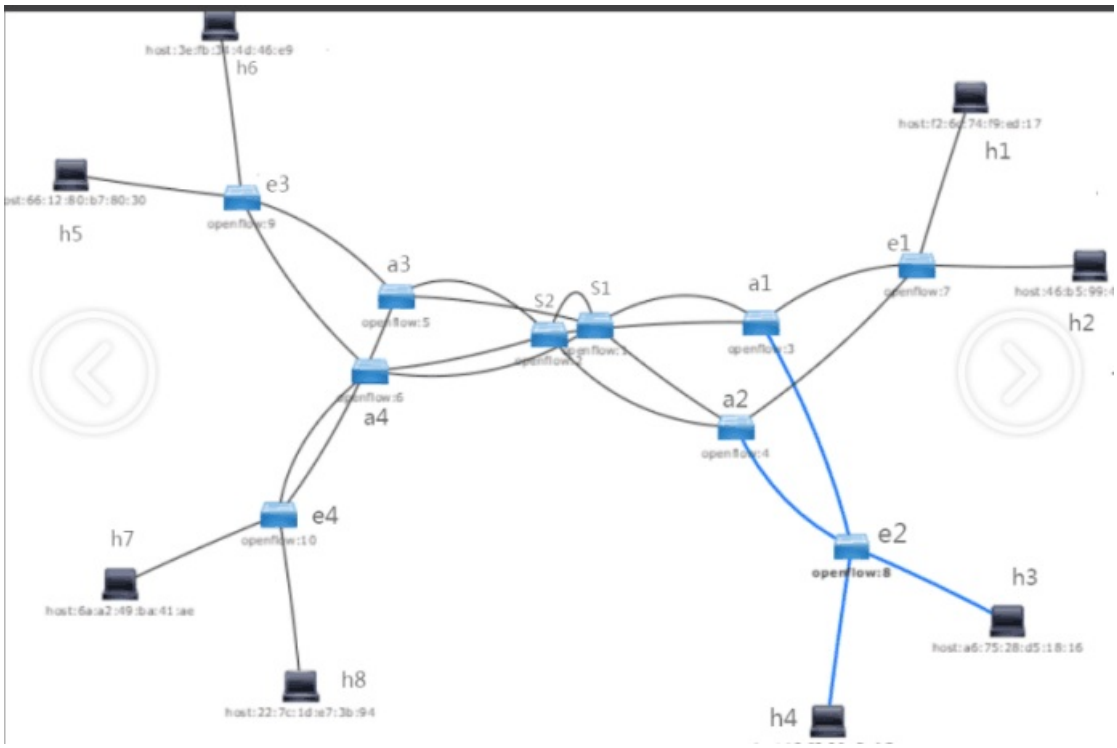
具体将上述丢弃操作封装为drop函数，在必要处对数据包进行丢弃操作。

第四题：创建基于2个数据中心的网络拓扑

系统环境：**ubuntu16.04**
模拟环境：**Mininet**
控制器：**POX**
交换机：**OpenVSwitch**
性能测试工具：**Iperf**

问题1. 给出具体的网络拓扑视图截图，Ping测试结果截图，并将节点标明在图中。

- 网络拓扑视图截图如下：



- Ping结果测试如下：
  - `h1 ping h2`

    如下，经验证，两主机间可以通信：

```
mininet> h1 ping -c5 h2
PING 2.2.3.1 (2.2.3.1) 56(84) bytes of data.
64 bytes from 2.2.3.1: icmp_seq=1 ttl=64 time=0.204 ms
64 bytes from 2.2.3.1: icmp_seq=2 ttl=64 time=0.038 ms
64 bytes from 2.2.3.1: icmp_seq=3 ttl=64 time=0.047 ms
64 bytes from 2.2.3.1: icmp_seq=4 ttl=64 time=0.104 ms
64 bytes from 2.2.3.1: icmp_seq=5 ttl=64 time=0.107 ms

--- 2.2.3.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 0.038/0.100/0.204/0.059 ms
```

  - `h1 ping h3`

    如下，经验证，两主机间可以通信：

```
mininet> h1 ping -c5 h3
PING 2.2.4.1 (2.2.4.1) 56(84) bytes of data.
64 bytes from 2.2.4.1: icmp_seq=1 ttl=64 time=0.314 ms
64 bytes from 2.2.4.1: icmp_seq=2 ttl=64 time=0.334 ms
64 bytes from 2.2.4.1: icmp_seq=3 ttl=64 time=0.200 ms
64 bytes from 2.2.4.1: icmp_seq=4 ttl=64 time=0.217 ms
64 bytes from 2.2.4.1: icmp_seq=5 ttl=64 time=0.262 ms

--- 2.2.4.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.200/0.265/0.334/0.054 ms
```

  - `h1 ping h5`

    如下，经验证，两主机间可以通信：

```
mininet> h1 ping -c5 h5
PING 2.2.6.1 (2.2.6.1) 56(84) bytes of data.
64 bytes from 2.2.6.1: icmp_seq=1 ttl=64 time=0.414 ms
64 bytes from 2.2.6.1: icmp_seq=2 ttl=64 time=0.271 ms
64 bytes from 2.2.6.1: icmp_seq=3 ttl=64 time=0.218 ms
64 bytes from 2.2.6.1: icmp_seq=4 ttl=64 time=0.533 ms
64 bytes from 2.2.6.1: icmp_seq=5 ttl=64 time=0.228 ms

--- 2.2.6.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 0.218/0.332/0.533/0.124 ms
```

问题2. 通过iperf对两主机间的带宽性能进行分析：

- `iperf h1 h2`

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
.*** Results: ['16.3 Gbits/sec', '16.3 Gbits/sec']
```

  测试TCP带宽发送数据的速率为**16.3 Gbit/s**，接收数据的速率为**16.3 Gbit/s**。

- `iperf h1 h3`

```
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['203 Mbits/sec', '208 Mbits/sec']
```

测试TCP带宽发送数据的速率为**203 Mbit/s**，接收数据的速率为**208 Mbit/s**。

- `iperf h1 h5`

```
mininet> iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['189 Mbits/sec', '195 Mbits/sec']
```

测试TCP带宽发送数据的速率为**189 Mbit/s**，接收数据的速率为**195 Mbit/s**。

性能测试结果表格如下：

Iperf测试	h1 and h2	h1 and h3	h1 and h5
发送速率	16.3 Gbit/s	203 Mbit/s	189 Mbit/s
接收速率	16.3 Gbit/s	208 Mbit/s	195 Mbit/s