



Pervasive Computing

Final Course Report

Title: Simulation of A Simple Cloud Networking
Platform Based on Mininet

Student Name: _____
Student ID: _____
School/College: _____
Major: _____
Assignment: Assignment 1/Assignment 2
Date: 06/04/2016

Table of Contents

Problem Description:..... 3

 Task 1: Network Building..... 3

 Task 2: Firewall 4

 Task 3: Virtual Network Slicing 4

Runtime Environment Setup:..... 6

 Comments: 6

 Concise Instructions I concluded: 6

Solution to Task 1: 7

Solution to Task 2: 9

Solution to Task 3: 16

Academic Honesty Declaration 25

Problem Description:

In this assignment, you will be required to simulate a simple cloud networking platform. The Mininet network emulator will be used to build a virtual network as required. And you need to program the OpenFlow controller POX (of course, you can choose other preferred controllers) to implement two common applications in multi-tenants cloud environment: Firewall and Virtual Network Slicing.

The Mininet network emulation environment and POX controller should run on different machines (we recommend you to use two virtual machines) and they communicate via LAN (or a virtual LAN if you use two virtual machines in a single physical machine).

For both tasks, you may need to use iperf to generate some traffic to test your code.

For details of Mininet, POX and iperf, please refer to the document: “A short walk-through of Mininet and POX.pdf”

Task 1: Network Building

In this part, your task is to build a simple virtual network in the Mininet network emulation environment. The network consists of 4 servers, 4 switches and 8 links that connect them together. The topology of the network is shown in Figure 1.

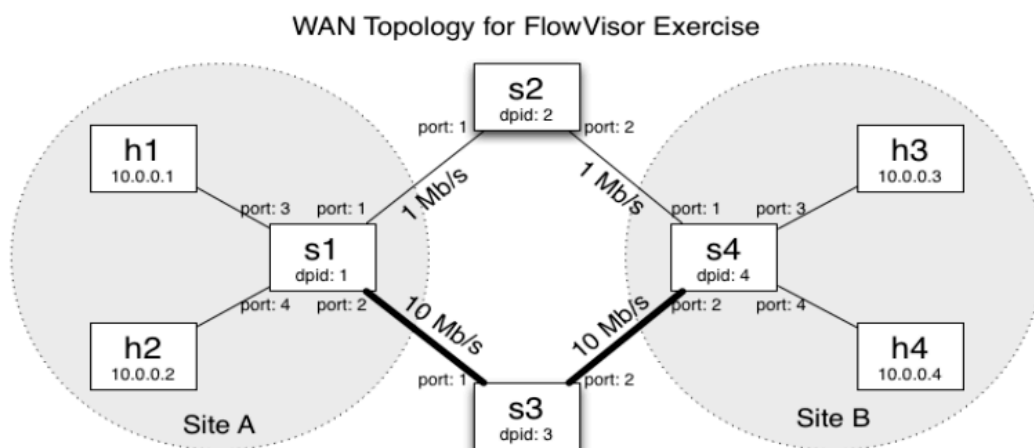


Figure 1: Network with 4 servers, 4 switches and 8 links.

You need to build a network whose topology is described in Figure 1, where h represents servers and s represents switches. Notice that

the configurations of the links that connect the 4 servers to their nearest switches are not specified. The only requirement is they should not be the bottleneck. It is recommended you write a python script to do the job.

You will be provided with a template of script, based on which you can add your own code to set up the network.

Task 2: Firewall

In this task, you need to implement a layer-2 firewall application using the POX controller. You are required to block the communication between h1 and h4.

The basic idea is when a connection between a switch and a controller is up, the application installs flow entries to disable the traffic between each pair of the MAC addresses in the list. To make things simple, you can install the firewall rules on all switches in the network.

Task 3: Virtual Network Slicing

In this part, your task is to slice the network based on the application that is sending the traffic. In principle, SDN networks can be sliced based on any attributes.

The topology of the network is the same as in Task 1. However, you need to treat the video and non-video traffic differently. Between site A and site B, all the video traffic will go through the high bandwidth path (video slice), and all other traffic will go through the low bandwidth path (non-video slice). To simplify this task, we assume that all the video traffic use TCP port 80.

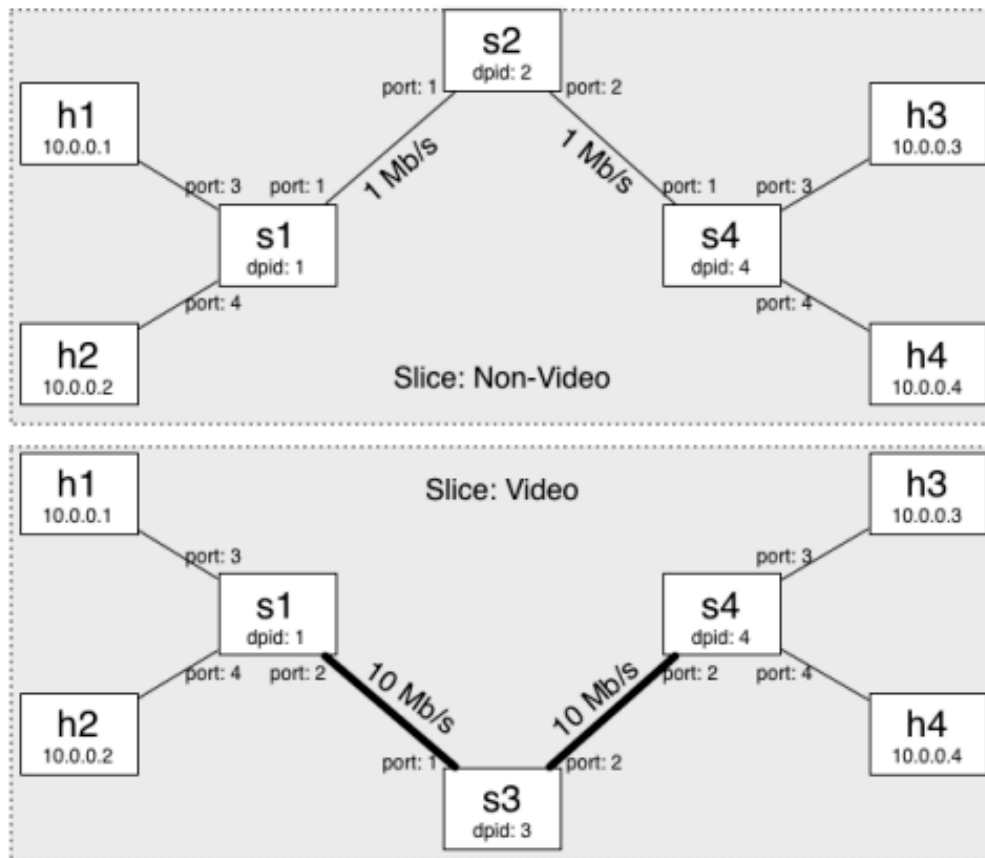


Figure 2: Sliced Network Traffic.

For illustration, Figure 2 depicts the sliced network, in which the video traffic flows through the path S1-S3-S4, and the rest traffic is forwarded to the path S1-S2-S4.

Runtime Environment Setup:

Comments:

Setting up runtime environment is always a long, boring and painful experience. You may encounter many kinds of traps and sometimes when you are not lucky, you just cannot get out of them.

My working operating system environment is Ubuntu 14.04 LTS.

Concise Instructions I concluded:

1. Google search Mininet and download prepackaged Mininet Virtual Machine. (As for me, this is the first obstacle as the download link happened to be down when I started.)
2. Google search VirtualBox, download it and install it.
3. Boot Mininet VM from your VirtualBox. Login credential(username: mininet, password: mininet)
4. Set up host-only network on VirtualBox.
5. Hands on Mininet:
 - A. ssh to Mininet and start two terminal
 - B. type “sudo mn –topo single,3 –mac –controller remote –switch ovsk” in one terminal
 - C. type “./pox.py log.level –DEBUG forwarding.hub” in the other terminal
 - D. type “h1 ping h2”
6. If all comes out to be good, you are ready to go.

Solution to Task 1:

Source Code:

mininet_modified.py:

```
#!/usr/bin/python

import inspect
import os
import atexit
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel, info
from mininet.cli import CLI
from mininet.topo import Topo
from mininet.link import TCLink
from mininet.topo import SingleSwitchTopo
from mininet.node import RemoteController

net = None

class FVTopo(Topo):
    # credit:
    https://github.com/onstutorial/onstutorial/blob/master/flowvisor\_scripts/flowvisor\_topo.py
    def __init__(self):
        # Initialize topology
        Topo.__init__(self)

        # Create template host, switch, and link
        hconfig = {'inNamespace': True}
        http_link_config = {'bw': 1}
        video_link_config = {'bw': 10}
        host_link_config = {}

        # Create switch nodes
        for i in range(4):
            sconfig = {'dpid': "%016x" % (i+1)}
            self.addSwitch('%s%d' % (i+1), **sconfig)

        # Create host nodes
        for i in range(4):
```

```

        self.addHost('%d' % (i+1), **hconfig)

    # Add switch links
    # Specified to the port numbers to avoid any port number consistency issue

    self.addLink('s2', 's1', port1=1, port2=1, **http_link_config)
    self.addLink('s3', 's1', port1=1, port2=2, **video_link_config)
    self.addLink('h1', 's1', port1=1, port2=3, **host_link_config)
    self.addLink('h2', 's1', port1=1, port2=4, **host_link_config)

    self.addLink('s2', 's4', port1=2, port2=1, **http_link_config)
    self.addLink('s3', 's4', port1=2, port2=2, **video_link_config)
    self.addLink('h3', 's4', port1=1, port2=3, **host_link_config)
    self.addLink('h4', 's4', port1=1, port2=4, **host_link_config)

    info( '\n*** printing and validating the ports running on each interface\n' )

def startNetwork():
    info('** Creating Overlay network topology\n')
    topo = FVTopo()
    global net
    net = Mininet(topo=topo, link = TCLink,
                  controller=lambda name: RemoteController(name, ip='127.0.0.1'),
                  listenPort=6633, autoSetMacs=True)

    info('** Starting the network\n')
    net.start()

    info('** Running CLI\n')
    CLI(net)

def stopNetwork():
    if net is not None:
        info('** Tearing down Overlay network\n')
        net.stop()

if __name__ == '__main__':
    # Force cleanup on exit by registering a cleanup function
    atexit.register(stopNetwork)

```



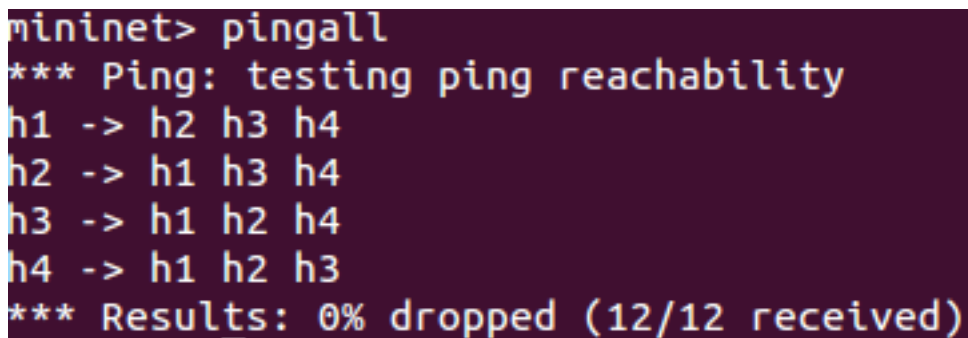
```
# Tell mininet to print useful information
setLogLevel('info')
startNetwork()
```

README:

Instructions:

1. Open one terminal and type `cd ~/pox`
2. Next, `sudo ./pox.py log.level -DEBUG forwarding.hub`
3. Open another terminal and type `cd ~/pox/pox/misc`
4. Next, `sudo python mininet_modified.py`
5. Finally, type `pingall` and you will see the firewall working.

Result:

A terminal window with a dark purple background and light blue text. The text shows the command 'mininet> pingall' followed by a series of ping results for hosts h1, h2, h3, and h4. The results show that all hosts are reachable, with 0% dropped and 12/12 received.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Solution to Task 2:

Source Code:

controller.py:

```
from pox.core import core
from collections import defaultdict

import pox.openflow.libopenflow_01 as of
import pox.openflow.discovery
import pox.openflow.spanning_tree

from pox.lib.revent import *
from pox.lib.util import dpid_to_str
from pox.lib.util import dpidToStr
from pox.lib.addresses import IPAddr, EthAddr
from collections import namedtuple
import os
```

```

import csv

log = core.getLogger()

class VideoSlice (EventMixin):

    def __init__(self):
        self.listenTo(core.openflow)
        core.openflow_discovery.addListeners(self)

        # Adjacency map. [sw1][sw2] -> port from sw1 to sw2
        self.adjacency = defaultdict(lambda:defaultdict(lambda:None))

        ...

        The structure of self.portmap is a four-tuple key and a string value.
        The type is:
        (dpid string, src MAC addr, dst MAC addr, port (int)) -> dpid of next switch
        ...

        self.portmap = {
        # h1 <-- port 80 --> h3
            ('00-00-00-00-00-01', EthAddr('00:00:00:00:00:01'),
             EthAddr('00:00:00:00:00:03'), 80): '00-00-00-00-00-03',

            # "" Add your mapping logic here""
            ('00-00-00-00-00-03', EthAddr('00:00:00:00:00:01'),
             EthAddr('00:00:00:00:00:03'), 80): '00-00-00-00-00-04',

            ('00-00-00-00-00-03', EthAddr('00:00:00:00:00:03'),
             EthAddr('00:00:00:00:00:01'), 80): '00-00-00-00-00-01',

            ('00-00-00-00-00-04', EthAddr('00:00:00:00:00:03'),
             EthAddr('00:00:00:00:00:01'), 80): '00-00-00-00-00-03',

        # h2 <-- port 22 --> h4
            ('00-00-00-00-00-01', EthAddr('00:00:00:00:00:02'),
             EthAddr('00:00:00:00:00:04'), 22): '00-00-00-00-00-02',

            ('00-00-00-00-00-02', EthAddr('00:00:00:00:00:02'),
             EthAddr('00:00:00:00:00:04'), 22): '00-00-00-00-00-04',

            ('00-00-00-00-00-02', EthAddr('00:00:00:00:00:04'),
             EthAddr('00:00:00:00:00:02'), 22): '00-00-00-00-00-01',

```

```

        ('00-00-00-00-00-04', EthAddr('00:00:00:00:00:04'),
         EthAddr('00:00:00:00:00:02'), 22): '00-00-00-00-00-02',
    }

log.debug('Enabling Firewall Module __init__')
self.mac_pair = []
policyFile = "%s/pox/pox/misc/firewall-policies.csv" % os.environ[ 'HOME' ]
file = open(policyFile,'rb')
reader = csv.DictReader(file)
for row in reader:
    self.mac_pair.append((row['mac_0'], row['mac_1']))
    self.mac_pair.append((row['mac_1'], row['mac_0']))

log.debug('Firewall policies loaded successfully')

def _handle_LinkEvent (self, event):
    l = event.link
    sw1 = dpid_to_str(l.dpid1)
    sw2 = dpid_to_str(l.dpid2)

    log.debug ("link %s[%d] <-> %s[%d]",
               sw1, l.port1,
               sw2, l.port2)

    self.adjacency[sw1][sw2] = l.port1
    self.adjacency[sw2][sw1] = l.port2

def _handle_PacketIn (self, event):
    """
    Handle packet in messages from the switch to implement above algorithm.
    """
    packet = event.parsed
    tcp = event.parsed.find('tcp')

    def install_fwdrule(event,packet,output):
        msg = of.ofp_flow_mod()
        msg.idle_timeout = 10
        msg.hard_timeout = 30
        msg.match = of.ofp_match.from_packet(packet, event.port)
        msg.actions.append(of.ofp_action_output(port = output))
        msg.data = event.ofp
        msg.in_port = event.port
        event.connection.send(msg)

```

```

def forward (message = None):
    this_dpid = dpid_to_str(event.dpid)

    if packet.dst.is_multicast:
        flood()
        return
    else:
        log.debug("Got unicast packet for %s at %s (input port %d):",
            packet.dst, dpid_to_str(event.dpid), event.port)

        try:
            # """ Add your logic here"""
            k = (this_dpid, packet.src, packet.dst, packet.find('tcp').dstport)
            if not self.portmap.get(k):
                k = (this_dpid, packet.src, packet.dst,
packet.find('tcp').srcport)
            if not self.portmap.get(k):
                raise AttributeError

            ndpid = self.portmap[k]
            log.debug("install: %s output %d" % (str(k),
self.adjacency[this_dpid][ndpid]))
            install_fwdrule(event,packet,self.adjacency[this_dpid][ndpid])

        except AttributeError:
            log.debug("packet type has no transport ports, flooding")

            # flood and install the flow table entry for the flood
            install_fwdrule(event,packet,of.OFPP_FLOOD)

# flood, but don't install the rule
def flood (message = None):
    """ Floods the packet """
    msg = of.ofp_packet_out()
    msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
    msg.data = event.ofp
    msg.in_port = event.port
    event.connection.send(msg)

forward()

def _handle_ConnectionUp(self, event):

```

```

        dpid = dpidToStr(event.dpid)
        log.debug("Switch %s has come up.", dpid)

        for (mac_src, mac_dst) in self.mac_pair:
            match = of.ofp_match() # obj describing packet header fields & input port
            to match on
            match.dl_src = EthAddr(mac_src)
            match.dl_dst = EthAddr(mac_dst)
            msg = of.ofp_flow_mod() # create packet out message
            msg.match = match
            msg.priority = msg_mirror.priority = 42
            msg.actions.append(of.ofp_action_output(port = of.OFPP_NONE))
            event.connection.send(msg)

        log.debug("Firewall rules installed on %s", dpidToStr(event.dpid))

def launch():
    # Run spanning tree so that we can deal with topologies with loops
    pox.openflow.discovery.launch()
    pox.openflow.spanning_tree.launch()

    ...

    Starting the Video Slicing module
    ...

    core.registerNew(VideoSlice)

```

mininet_modified.py:

```

from pox.core import core
from collections import defaultdict

import pox.openflow.libopenflow_01 as of
import pox.openflow.discovery
import pox.openflow.spanning_tree

from pox.lib.revent import *
from pox.lib.util import dpid_to_str
from pox.lib.util import dpidToStr
from pox.lib.addresses import IPAddr, EthAddr
from collections import namedtuple
import os

log = core.getLogger()

```

```

class VideoSlice (EventMixin):

    def __init__(self):
        self.listenTo(core.openflow)
        core.openflow_discovery.addListeners(self)

    def _handle_PacketIn (self, event):
        """
        Handle packet in messages from the switch to implement above algorithm.
        """
        packet = event.parsed
        tcpp = event.parsed.find('tcp')

    def install_fwdrule(event, packet, outport):
        msg = of.ofp_flow_mod() #install a flow table entry
        msg.idle_timeout = 10
        msg.hard_timeout = 30
        msg.match = of.ofp_match.from_packet(packet, event.port)
        msg.actions.append(of.ofp_action_output(port = outport))
        msg.data = event.ofp
        msg.in_port = event.port
        event.connection.send(msg)

    def forward (message = None):
        this_dpid = dpid_to_str(event.dpid)

        if packet.dst.is_multicast:
            flood()
            return
        else:
            log.debug("Got unicast packet for %s at %s (input port %d):",
                    packet.dst, dpid_to_str(event.dpid), event.port)
            ...

            Add your logic here to slice the network

            ...

# flood, but don't install the rule
def flood (message = None):
    """ Floods the packet """
    msg = of.ofp_packet_out()

```

```

        msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
        msg.data = event.ofp
        msg.in_port = event.port
        event.connection.send(msg)

    forward()

def _handle_ConnectionUp(self, event):
    dpid = dpidToStr(event.dpid)
    log.debug("Switch %s has come up.", dpid)
    ...

    Add your logic here for firewall application
    ...

    for (mac_src, mac_dst) in self.mac_pair:
        match = of.ofp_match() # obj describing packet header fields & input port
to match on
        match.dl_src = EthAddr(mac_src)
        match.dl_dst = EthAddr(mac_dst)
        msg = of.ofp_flow_mod() # create packet out message
        msg.priority = msg_mirror.priority = 42
        msg.match = match
        msg.actions.append(of.ofp_action_output(port = of.OFPP_NONE))
        event.connection.send(msg)

    log.debug("Firewall rules installed on %s", dpidToStr(event.dpid))

def launch():
    # Run spanning tree so that we can deal with topologies with loops
    pox.openflow.discovery.launch()
    pox.openflow.spanning_tree.launch()

    ...

    Starting the Video Slicing module
    ...

    core.registerNew(VideoSlice)

```

README:

Instructions:

6. Open one terminal and type `cd ~/pox`
7. Next, `sudo ./pox.py pox.misc.controller`
8. Open another terminal and type `cd ~/pox/pox/misc`
9. Next, `sudo python mininet_modified.py`
10. Finally, type `pingall` and you will see the firewall working.

Result:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 X
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> X h2 h3
*** Results: 16% dropped (10/12 received)
```

Solution to Task 3:

Source Code:

mininet_modified.py:

```
#!/usr/bin/python

import inspect
import os
import atexit
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel, info
from mininet.cli import CLI
from mininet.topo import Topo
from mininet.link import TCLink
from mininet.topo import SingleSwitchTopo
from mininet.node import RemoteController

net = None
```



```

class FVTopo(Topo):
def __init__(self):
    # Initialize topology
    Topo.__init__(self)

    # Create template host, switch, and link
    hconfig = {'inNamespace': True}
    http_link_config = {'bw': 1}
    video_link_config = {'bw': 10}
    host_link_config = {}

    # Create switch nodes
    for i in range(4):
        sconfig = {'dpid': "%016x" % (i+1)}
        self.addSwitch('s%d' % (i+1), **sconfig)

    # Create host nodes
    for i in range(4):
        self.addHost('h%d' % (i+1), **hconfig)

    # Add switch links
    # Specified to the port numbers to avoid any port number consistency issue

    self.addLink('s2', 's1', port1=1, port2=1, **http_link_config)
    self.addLink('s3', 's1', port1=1, port2=2, **video_link_config)
    self.addLink('h1', 's1', port1=1, port2=3, **host_link_config)
    self.addLink('h2', 's1', port1=1, port2=4, **host_link_config)

    self.addLink('s2', 's4', port1=2, port2=1, **http_link_config)
    self.addLink('s3', 's4', port1=2, port2=2, **video_link_config)
    self.addLink('h3', 's4', port1=1, port2=3, **host_link_config)
    self.addLink('h4', 's4', port1=1, port2=4, **host_link_config)

    info( '\n*** printing and validating the ports running on each interface\n' )

def startNetwork():
    info('** Creating Overlay network topology\n')
    topo = FVTopo()
    global net
    net = Mininet(topo=topo, link = TCLink,
                  controller=lambda name: RemoteController(name, ip='127.0.0.1'),

```

```

        listenPort=6633, autoSetMacs=True)

    info('** Starting the network\n')
    net.start()

    info('** Running CLI\n')
    CLI(net)

def stopNetwork():
    if net is not None:
        info('** Tearing down Overlay network\n')
        net.stop()

if __name__ == '__main__':
    # Force cleanup on exit by registering a cleanup function
    atexit.register(stopNetwork)

    # Tell mininet to print useful information
    setLogLevel('info')
    startNetwork()

```

videoSlice_modified.py:

```

from pox.core import core
from collections import defaultdict

import pox.openflow.libopenflow_01 as of
import pox.openflow.discovery
import pox.openflow.spanning_tree

from pox.lib.revent import *
from pox.lib.util import dpid_to_str
from pox.lib.util import dpidToStr
from pox.lib.addresses import IPAddr, EthAddr
from collections import namedtuple
import os

log = core.getLogger()

class VideoSlice (EventMixin):

```

```

def __init__(self):
    self.listenTo(core.openflow)
    core.openflow_discovery.addListeners(self)

    # Adjacency map. [sw1][sw2] -> port from sw1 to sw2
    self.adjacency = defaultdict(lambda:defaultdict(lambda:None))

    ...

    The structure of self.portmap is a four-tuple key and a string value.
    The type is:
    (dpid string, src MAC addr, dst MAC addr, port (int)) -> dpid of next switch
    ...

    self.portmap = {
        # Video from h1 <----> h3
        ('00-00-00-00-00-01', EthAddr('00:00:00:00:00:01'),
         EthAddr('00:00:00:00:00:03'), 80): '00-00-00-00-00-03',
        ('00-00-00-00-00-03', EthAddr('00:00:00:00:00:01'),
         EthAddr('00:00:00:00:00:03'), 80): '00-00-00-00-00-04',
        ('00-00-00-00-00-04', EthAddr('00:00:00:00:00:03'),
         EthAddr('00:00:00:00:00:01'), 80): '00-00-00-00-00-03',
        ('00-00-00-00-00-03', EthAddr('00:00:00:00:00:03'),
         EthAddr('00:00:00:00:00:01'), 80): '00-00-00-00-00-01',

        # Video from h2 <---> h3
        ('00-00-00-00-00-01', EthAddr('00:00:00:00:00:02'),
         EthAddr('00:00:00:00:00:03'), 80): '00-00-00-00-00-03',
        ('00-00-00-00-00-03', EthAddr('00:00:00:00:00:02'),
         EthAddr('00:00:00:00:00:03'), 80): '00-00-00-00-00-04',
        ('00-00-00-00-00-04', EthAddr('00:00:00:00:00:03'),
         EthAddr('00:00:00:00:00:02'), 80): '00-00-00-00-00-03',
        ('00-00-00-00-00-03', EthAddr('00:00:00:00:00:03'),
         EthAddr('00:00:00:00:00:02'), 80): '00-00-00-00-00-01',

        # Video from h1 <---> h4
        ('00-00-00-00-00-01', EthAddr('00:00:00:00:00:01'),
         EthAddr('00:00:00:00:00:04'), 80): '00-00-00-00-00-03',
        ('00-00-00-00-00-03', EthAddr('00:00:00:00:00:01'),
         EthAddr('00:00:00:00:00:04'), 80): '00-00-00-00-00-04',
        ('00-00-00-00-00-04', EthAddr('00:00:00:00:00:04'),
         EthAddr('00:00:00:00:00:01'), 80): '00-00-00-00-00-03',
        ('00-00-00-00-00-03', EthAddr('00:00:00:00:00:04'),
         EthAddr('00:00:00:00:00:01'), 80): '00-00-00-00-00-01',
    }

```

```
# Video from h2 <---> h4
('00-00-00-00-00-01', EthAddr('00:00:00:00:00:02'),
 EthAddr('00:00:00:00:00:04'), 80): '00-00-00-00-00-03',
('00-00-00-00-00-03', EthAddr('00:00:00:00:00:02'),
 EthAddr('00:00:00:00:00:04'), 80): '00-00-00-00-00-04',
('00-00-00-00-00-04', EthAddr('00:00:00:00:00:04'),
 EthAddr('00:00:00:00:00:02'), 80): '00-00-00-00-00-03',
('00-00-00-00-00-03', EthAddr('00:00:00:00:00:04'),
 EthAddr('00:00:00:00:00:02'), 80): '00-00-00-00-00-01',

# FTP from h1 <----> h3
('00-00-00-00-00-01', EthAddr('00:00:00:00:00:01'),
 EthAddr('00:00:00:00:00:03'), 22): '00-00-00-00-00-02',
('00-00-00-00-00-02', EthAddr('00:00:00:00:00:01'),
 EthAddr('00:00:00:00:00:03'), 22): '00-00-00-00-00-04',
('00-00-00-00-00-04', EthAddr('00:00:00:00:00:03'),
 EthAddr('00:00:00:00:00:01'), 22): '00-00-00-00-00-02',
('00-00-00-00-00-02', EthAddr('00:00:00:00:00:03'),
 EthAddr('00:00:00:00:00:01'), 22): '00-00-00-00-00-01',

# FTP from h2 <---> h3
('00-00-00-00-00-01', EthAddr('00:00:00:00:00:02'),
 EthAddr('00:00:00:00:00:03'), 22): '00-00-00-00-00-02',
('00-00-00-00-00-02', EthAddr('00:00:00:00:00:02'),
 EthAddr('00:00:00:00:00:03'), 22): '00-00-00-00-00-04',
('00-00-00-00-00-04', EthAddr('00:00:00:00:00:03'),
 EthAddr('00:00:00:00:00:02'), 22): '00-00-00-00-00-02',
('00-00-00-00-00-02', EthAddr('00:00:00:00:00:03'),
 EthAddr('00:00:00:00:00:02'), 22): '00-00-00-00-00-01',

# FTP from h1 <---> h4
('00-00-00-00-00-01', EthAddr('00:00:00:00:00:01'),
 EthAddr('00:00:00:00:00:04'), 22): '00-00-00-00-00-02',
('00-00-00-00-00-02', EthAddr('00:00:00:00:00:01'),
 EthAddr('00:00:00:00:00:04'), 22): '00-00-00-00-00-04',
('00-00-00-00-00-04', EthAddr('00:00:00:00:00:04'),
 EthAddr('00:00:00:00:00:01'), 22): '00-00-00-00-00-02',
('00-00-00-00-00-02', EthAddr('00:00:00:00:00:04'),
 EthAddr('00:00:00:00:00:01'), 22): '00-00-00-00-00-01',

# FTP from h2 <---> h4
('00-00-00-00-00-01', EthAddr('00:00:00:00:00:02'),
 EthAddr('00:00:00:00:00:04'), 22): '00-00-00-00-00-02',
```

```

        ('00-00-00-00-00-02', EthAddr('00:00:00:00:00:02'),
         EthAddr('00:00:00:00:00:04'), 22): '00-00-00-00-00-04',
        ('00-00-00-00-00-04', EthAddr('00:00:00:00:00:04'),
         EthAddr('00:00:00:00:00:02'), 22): '00-00-00-00-00-02',
        ('00-00-00-00-00-02', EthAddr('00:00:00:00:00:04'),
         EthAddr('00:00:00:00:00:02'), 22): '00-00-00-00-00-01',
    }

```

```

def _handle_LinkEvent (self, event):

```

```

    l = event.link

```

```

    sw1 = dpid_to_str(l.dpid1)

```

```

    sw2 = dpid_to_str(l.dpid2)

```

```

    log.debug ("link %s[%d] <-> %s[%d]",

```

```

               sw1, l.port1,

```

```

               sw2, l.port2)

```

```

    self.adjacency[sw1][sw2] = l.port1

```

```

    self.adjacency[sw2][sw1] = l.port2

```

```

def _handle_PacketIn (self, event):

```

```

    """

```

```

    Handle packet in messages from the switch to implement above algorithm.

```

```

    """

```

```

    packet = event.parsed

```

```

    tcpp = event.parsed.find('tcp')

```

```

def install_fwdrule(event,packet,outport):

```

```

    msg = of.ofp_flow_mod()

```

```

    msg.idle_timeout = 10

```

```

    msg.hard_timeout = 30

```

```

    msg.match = of.ofp_match.from_packet(packet, event.port)

```

```

    msg.actions.append(of.ofp_action_output(port = outport))

```

```

    msg.data = event.ofp

```

```

    msg.in_port = event.port

```

```

    event.connection.send(msg)

```

```

def forward (message = None):

```

```

    this_dpid = dpid_to_str(event.dpid)

```

```

    if packet.dst.is_multicast:

```

```

        flood()

```

```

    return

```

```

else:
    log.debug("Got unicast packet for %s at %s (input port %d):",
              packet.dst, dpid_to_str(event.dpid), event.port)

    try:
        # """ Add your logic here"""
        k = (this_dpid, packet.src, packet.dst, packet.find('tcp').dstport)
        if not self.portmap.get(k):
            k = (this_dpid, packet.src, packet.dst,
packet.find('tcp').srcport)
            if not self.portmap.get(k):
                raise AttributeError

        ndpid = self.portmap[k]
        log.debug("install: %s output %d" % (str(k),
self.adjacency[this_dpid][ndpid]))
        install_fwdrule(event, packet, self.adjacency[this_dpid][ndpid])

    except AttributeError:
        log.debug("packet type has no transport ports, flooding")

        # flood and install the flow table entry for the flood
        install_fwdrule(event, packet, of.OFPP_FLOOD)

# flood, but don't install the rule
def flood (message = None):
    """ Floods the packet """
    msg = of.ofp_packet_out()
    msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
    msg.data = event.ofp
    msg.in_port = event.port
    event.connection.send(msg)

forward()

def _handle_ConnectionUp(self, event):
    dpid = dpidToStr(event.dpid)
    log.debug("Switch %s has come up.", dpid)

def launch():
    # Run spanning tree so that we can deal with topologies with loops
    pox.openflow.discovery.launch()

```

```
pox.openflow.spanning_tree.launch()

...

Starting the Video Slicing module
...

core.registerNew(VideoSlice)
```

README:

Instructions:

1. Open one terminal and type `cd ~/pox`
2. Next, `sudo ./pox.py pox.misc.videoSlice_modified`
3. Open another terminal and type `cd ~/pox/pox/misc`
4. Next, `sudo python mininet_modified.py`
5. Finally, type `pingall` and you will see the network setup working.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

6. In mininet, type:
 - i) `h4 iperf -s -p 80 &`
 - ii) `h4 iperf -s -p 22 &`
 - iii) `h2 iperf -c h4 -p 80 -t 6 -i 1`
 - iv) `h2 iperf -c h4 -p 22 -t 6 -i 1`

Result:

```
mininet> h2 iperf -c h4 -p 80 -t 6 -i 1
-----
Client connecting to 10.0.0.4, TCP port 80
TCP window size: 85.3 KByte (default)
-----
[  3] local 10.0.0.2 port 60081 connected with 10.0.0.4 port 80
[ ID] Interval           Transfer     Bandwidth
[  3]  0.0- 1.0 sec      1.50 MBytes  12.6 Mbits/sec
[  3]  1.0- 2.0 sec      1.38 MBytes  11.5 Mbits/sec
[  3]  2.0- 3.0 sec      1.50 MBytes  12.6 Mbits/sec
[  3]  3.0- 4.0 sec      1.38 MBytes  11.5 Mbits/sec
[  3]  4.0- 5.0 sec      1.38 MBytes  11.5 Mbits/sec
[  3]  5.0- 6.0 sec      1.12 MBytes   9.44 Mbits/sec
[  3]  0.0- 6.2 sec      8.38 MBytes  11.4 Mbits/sec
```

```
mininet> h2 iperf -c h4 -p 22 -t 6 -i 1
```

```
-----  
Client connecting to 10.0.0.4, TCP port 22  
TCP window size: 85.3 KByte (default)  
-----
```

```
[ 3] local 10.0.0.2 port 52126 connected with 10.0.0.4 port 22  
[ ID] Interval          Transfer      Bandwidth  
[ 3] 0.0- 1.0 sec      384 KBytes   3.15 Mbits/sec  
[ 3] 1.0- 2.0 sec      128 KBytes   1.05 Mbits/sec  
[ 3] 2.0- 3.0 sec      128 KBytes   1.05 Mbits/sec  
[ 3] 3.0- 4.0 sec      256 KBytes   2.10 Mbits/sec  
[ 3] 4.0- 5.0 sec      128 KBytes   1.05 Mbits/sec  
[ 3] 5.0- 6.0 sec      128 KBytes   1.05 Mbits/sec  
[ 3] 0.0- 6.2 sec      1.25 MBytes  1.70 Mbits/sec
```


Academic Honesty Declaration

I declare that the material presented in this report is original except explicitly acknowledged.

Signature:

Date: 06/17/2016