Introducción a los Sistemas Distribuidos (75.43)
TP #3: Enlace

Esteban Carisimo[1] y AMD[1]

[1]Facultad de Ingeniería, Universidad de Buenos Aires

28 de mayo de 2018

Resumen

Las *Software-Defined Networks* han sido el tema dominante de la investigación y la innovación en Internet desde su aparición en 2008. Su irrupción en escena surgió de la necesidad de poder flexibilizar el uso del hardware para poder satisfacer las nuevas demandas de Internet. Su éxito fue casi inmediato, tal es así que Google en 2009 ya se encontraba utilizando esta tecnología para administrar el tráfico en su red global. Este trabajo práctico tiene como objetivo familiarizarse con los desafíos por los cuales surgen las SDNs, el protocolo *OpenFlow*, a través del cual se programan los dispositivos de red. Dado que ahora los dispositivos son programables, también se buscará aprender a controlar el funcionamiento de los switches a través de APIs en diferentes lenguajes de programación.

Palabras clave— OpenFlow, Software-Defined Networking, Traffic Engineering, Firewall

Conceptos previos específicos

Para abordar este trabajo no sólo se requiere el manejo conceptual de la esencia de los protocolos anteriormente vistos, sino que también se necesita conocer y dominar los siguientes temas puntuales

1. Concepto de flujo
2. IP blackholing

1. Introducción

Internet ha evolucionado a velocidades inimaginadas, y a pesar de que hoy el sistema de comunicaciones se sigue basando en un paradigma diseñado a fines de los 60s, cada 5 o 10 años Internet introduce innovaciones significativas que llevan al cambio de su fisonomía. La reciente masificación de los contenidos multimedia, sumados al cloud computing y a la exorbitante cantidad de nuevos dispositivos conectados luego del advenimiento de los smarthpones, ha planteado nuevos desafíos en Internet. Grandes volúmenes de tráfico, proveedores de contenidos que monopolizan el centro de la escena y monumentales datacenters donde se alojan casi todos los datos de los usuarios de Internet, han motivado a un uso más eficiente de los recursos y a re-adaptar las necesidades. A continuación se introducirán OpenFlow y las Software-Defined Networks (SDN), las cuales son el resultado de esta etapa de la historia de Internet donde se necesita más flexibilidad y dinamismo a la hora de la distribución de los paquetes.

1.1. Software-Defined Netowrking: Introducción a la problemática

En la actualidad existe una innumerable cantidad de routers y switches que conforman la red global de Internet. Más aún, existe una gran variedad de dispositivos dependiendo de su envergadura, sus requerimientos y su ubicación en la red, lo que eventualmente determina su costo y su consumo energético. A pesar que no existe una lista sumamente extensa de fabricantes, los dispositivos presentan diferencias de acuerdo a su manufacturante, más allá de que todos respeten los protocolos TCP/IP y eventualmente IEEE 802.3, IEEE 802.11 y etcétera. Estas diferencias se deben a prestaciones extra que los fabricantes introducen en los equipos para cumplir con *buenas prácticas*, satisfacer calidad de servicio o políticas de seguridad. Sin embargo, poco se conoce de cómo o cuáles son las funciones complementarias que introducen los equipos, sumado que existen nulas o limitadas formas de interactuar con estas prestaciones extra.

Un ejemplo de suma importancia dentro de las funcionalidades extra que ejecutan los routers, son las *buenas prácticas* sobre los flujos a la hora del ruteo (ver RFC2991 [1]). Suponiendo de que un router tiene más de un camino de igual costo para acceder a un destino, éste debe enviar los paquetes pertenecientes a flujo siempre por el mismo camino. Por más que no exista una definición estricta de que es un flujo, podemos interpretarlo como la cuaterna formada por $(srcIP, dstIP, srcport, dstport)$. Entonces, para poder hacer esto, los routers no sólo deben actuar de acuerdo a la dirección IP destino, sino que también a otros campos de las cabeceras IP y TCP. Sin embargo, llegado este punto, los usuarios (léase los administradores de la red), no pueden decidir que directivas ejecutar ante diferentes cuaternas.

Este último ejemplo mencionado es de vital importancia en todo el campo de Internet, ya que el protocolo IP fue concebido para que el ruteo sólo implique la dirección destino, y consecuentemente a una entrada en la tabla de ruteo. Aquí es necesario rememorar que la redes de conmutación de paquetes y en especial Internet surgieron con el objetivo de interconectar computadoras a través de sistemas intermediarios (routers) sumamente sencillos. Sin embargo, la evolución de la red sumando capacidades multimedia e interactivas, ha llevado a que el ruteo basado simplemente en dirección destino sea insuficiente para proveer calidad de experiencia.

Todo este problema de inflexibilidad que existe sobre los switches y routers naturalmente lleva a un problema en la innovación, y en la capacidad de adaptar las redes a las necesidades específicas de una organización. Una alternativa que se puede lograr es generar un switch o un router por medio de funciones en software en algún sistema operativo, por ejemplo Linux, donde ya existen algunas distribuciones para esta finalidad. Sin embargo, implementar funciones de routeo y forwarding en software es varios ordenes de magnitud más lento que ejecutarlo en hardware, por medio de memorias de rápido acceso como lo pueden ser las TCAM. Más aún, también aparecerían problemas de escala cuando este swtich virtual deba manejar múltiples interfaces en simultaneo, tal como lo hace un dispositivo de red.

1.2. OpenFlow

Ante los mencionados problemas de flexibilidad y las necesidades de introducir políticas de ruteo más específicas surgió OpenFlow [2], como un protocolo para poder interactuar directamente con las tablas de ruteo. Si analizamos un swtich, este posee una tabla CAM [1], la cual determina un puerto de salida en función de la dirección MAC destino. Estas memorias son de relativo bajo costo y de muy alta velocidad, entonces OpenFlow se propone poder reutilizar las memorias de los switches de manera inteligente. Como se mencionó anteriormente, hoy el ruteo en base a direcciones destino es insuficiente, entonces la idea es poder generar políticas en base a flujos, por lo cual cada una de las entradas en la tabla correspondería a un flujo. Entonces, la propuesta de OpenFlow consiste en 3 partes:

1. Una tabla de flujos
2. Un canal encriptado de comunicaciones hacia un controlador
3. El protocolo OpenFlow

Tanto routers como switches tienen su arquitectura dividida en dos: el plano de control y el plano de datos. En el plano de control operan las decisiones de como administrar el dispositivo, las cuales corren en software, cambio en el plano de datos, se administra la recepción y envío de paquetes y opera en el plano de hardware. La idea de OpenFlow es crear un protocolo de comunicación para poder controlar el plano de control, que eventualmente gestiona las entradas en las tablas de ruteo del plano de datos.

Sumado al concepto de una tabla de ruteo manipulable surge la idea de un controlador, un elemento externo capaz de configurar la tabla de flujos, tal como se lo muestra en la Figura 1. La relación entre el switch y el controlador puede ofrecer dos dinámicas: Configuración inicial, donde el controlador asigna funciones una única vez cuando se instala el switch, o dinámica, en donde constantemente se definen nuevas políticas. Es importante destaca que estos switches no aprenden, por lo cual sin interacción con el controlador, nunca enviarán un paquete.

Dado que OpenFlow se enfoca a poder controlar los flujos, los switches que manejan este protocolo pueden ejecutar múltiples acciones según si los paquetes coinciden con una entrada en la tabla de flujos. Es necesario destacar, que las tablas de flujos admiten *wildcards*. Por ejemplo en las acciones por flujo se pueden tomar políticas de ingeniería de tráfico, reescribir campos de la cabecera si fuera necesario (NAT) o descartar paquetes en caso de un ataque (Blackholing). Para esto los switches cuentan con hardware específico para la lectura de 10 campos, los cuales aparecen en la Tabla 1, por medio de los cuales se pueden individualizar un flujo.

[1] CAM: Content Addressable Memory

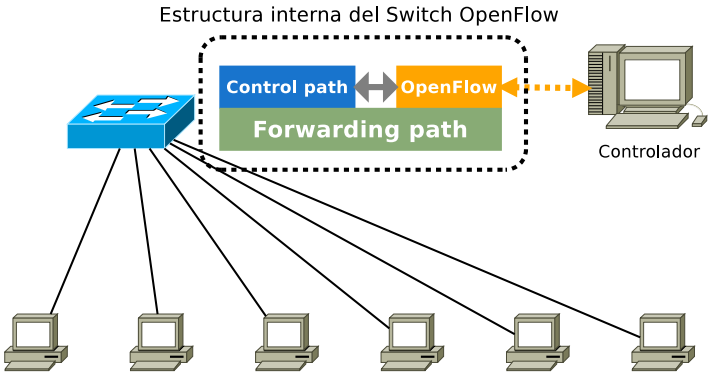

Figura 1: .

port IN	VLAN tag	eth			IP			TCP	
		src	dst	Type	src	dst	Proto	src	dst

Cuadro 1: Los 10 campos que se pueden utilizar para individualizar un flujo en OpenFlow

2. Propuesta de trabajo

El objetivo del trabajo será construir una topología dinámica, donde se pedirá utilizar OpenFlow para poder implementar un Firewall a nivel de capa de enlace. Para poder plantear este escenario se emulará el comportamiento de la topología a través de mininet. El trabajo cuenta, con múltiples pasos y requisitos y a continuación iremos introduciendo uno por uno. Además, la cátedra proveerá una máquina virtual (VM) donde ya estarán instalados los programas, como también incluidos los archivos accesorios.

2.1. Base conceptual de la simulación

En la introducción hemos incorporado nuevos conceptos y tecnologías, por lo cual, vale la pena hacer una pequeñas enumeración de estos conceptos, para luego ver como se incorporan a la propuesta de trabajo. Entonces, hasta ahora hemos mencionado

- **OpenFlow** es un protocolo que surge como respuesta a necesidades más específicas a la hora del ruteo. Este protocolo permite que a través de un controlador central se definan políticas de como se deben enviar y clasificar los paquetes. La ventaja radical que ofrece OpenFlow es la independencia del hardware para realizar estas acciones.
- **Firewall** es un dispositivo de seguridad de red que monitorea el tráfico de red entrante y saliente y decide si permite el paso o bloquea un tráfico específico en función de un conjunto definido de reglas de seguridad. Los firewalls han sido una primera línea de defensa en seguridad de red durante más de 25 años. Establecen una barrera entre las redes internas seguras y controladas que pueden ser de confianza y las que no son de confianza fuera de las redes, como Internet. Un firewall puede ser hardware, software o ambos.[2]

2.2. Mininet

Mininet [3] es un conocido simulador de redes, cuya aparición se debe a la necesidad de contar con un simulador capaz de operar con switches OpenFlow. El ingreso revolucionario de las SDN y la posibilidad de correr simulaciones, llevo a un gran crecimiento de mininet, por lo cual hoy es muy sencillo encontrar gran cantidad de información disponible. Para poder familiarizarnos nos mininet y su uso, hay un tutorial en el GitHub [3] oficial del proyecto. **A su vez, la cátedra presenta un breve tutorial de mininet en SDN, inspirado en el tutorial oficial**.

Mininet se utiliza de una forma radicalmente diferente al resto de los simuladores de red, ya que los dispositivos incluidos y su interconexión se definen a través de un script en Python.

- Abrir una nueva terminal e iniciar mininet con una topología simple, con 3 hosts conectados a un switch.

[2] Què es un firewall?: `https://www.cisco.com/c/en/us/products/security/firewalls/what-is-a-firewall.html`
[3] Tutorial mininet con OpenFlow: `https://github.com/mininet/openflow-tutorial`

```
$ sudo mn --topo single,3 --mac --arp --switch ovsk --controller remote
```

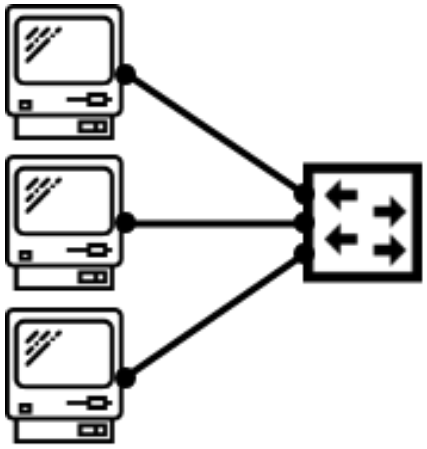

Figura 2: Topología Simple

- También se pueden crear topologías mas complejas, por ejemplo, se puede crear una topología *FatTree* con el siguiente comando.

```
$ sudo mn --custom ~/mininet/custom/fattree.py --topo fattree --mac --arp --
    switch ovsk --controller remote
```

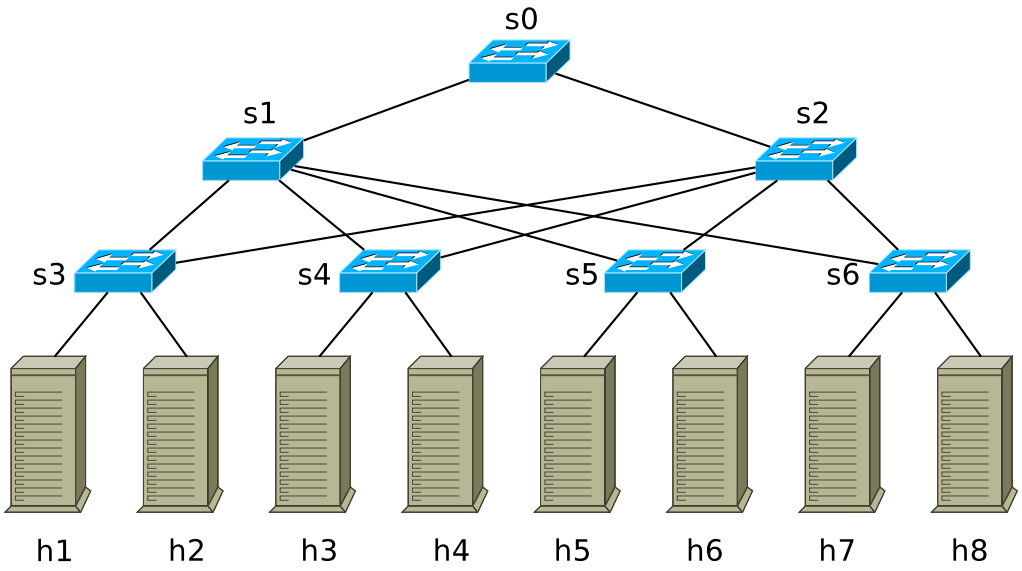

Figura 3: Topología de *Fattree*

Para visualizar las topologías se puede usar la siguiente herramienta online

- `http://demo.spear.narmox.com/app/?apiurl=demo#!/mininet`

2.3. Controladores de OpenFlow

Hemos hablando de OpenFlow y del controlador, pero no hemos mencionado aún como hacer uso del controlador, de manera tal de ejecutar rutinas. Aunque OpenFlow es el protocolo, por lo general los controladores corren aplicaciones las

cuales les permiten hacer uso del protocolo. Estas aplicaciones cuentan con API, las cuales se encuentran disponibles en Java, Python y C, aunque ésta última ha sido dejada de lado paulatinamente durante los últimos años.

En la cátedra proponemos hacer uso de un controlador POX [4], el cual tiene una API en Python, aunque esta la libertad de usar otros controladores ya sea Frenetic (Python) o Beacon (Java).

- Abrir una terminal e iniciar el controlador, indicándole que los switches correrán STP (verificar que se este utilizando la versión **betta** de pox, de no ser asi, se puede cambiar la versión cambiando al branch correcto 'git checkout betta')

```
$ pox/pox.py samples.spanning_tree
```

3. Preguntas a responder

1. ¿Cuál es la diferencia entre un Switch y un router? ¿Qué tienen en común?
2. ¿Cuál es la diferencia entre un Switch convencional y un Switch OpenFlow?
3. ¿Se pueden reemplazar todos los routers de la Intenet por Swithces OpenFlow? Piense en el escenario interASes para elaborar su respuesta

4. Ejercicios

Los grupos (no más de 3 personas), deberán elaborar un código capaz de generar las topologías indicadas sobre la cual se realizaran diferentes ensayos. El día de la entrega se deberá ejecutar el controlador y la simulación en clase, mostrando su funcionamiento y dando evidencias de que se conoce el uso de las herramientas dadas.

Se pedirá entregar el código y se exige no distribuirlo. Facilitar la distribución de código, como así también el uso de código desarrollado por otros, van en contra de la conducta que la Facultad de Ingeniería pretende de los alumnos. Más aún, nuestro objetivo en la cátedra es fomentar el aprendizaje y el interés por temas actuales e innovadores, por lo cual pretendemos que el desarrollo del trabajo sea motivador y enriquecedor para los alumnos.

4.1. Topología 1

Se propone desarrollar una topología parametrizable sobre la cual probaremos diferentes funcionalidades que nos brinda la tecnología OpenFlow.

La topología debe recibir por parámetro la cantidad de hosts, y los niveles de switches. Los hosts se deben conectar uniformemente a los switches externos de la topología. Es decir, si se eligen 3 hosts, debe haber dos conectados a un extremo de la topología, y el restante al otro extremo.

Los niveles de switches se definen, según: el nivel 1, tendrá un único switch, el nivel 2, tendrá dos switches conectados al switch del nivel anterior, el nivel 3, va estar compuesto por 4 switches, dos conectados a uno de los switches del nivel anterior, y los dos restantes conectados al otro. Una vez alcanzado el nivel indicado, la topología debe ser análoga de ambos lados, formando un rombo.

Por ejemplo, el siguiente comando (switch levels = 3 y host number = 4) debe dar como resultado, una topología como la mostrada en la figura 4.

```
$ sudo mn --custom ~/mininet/custom/custom_topo.py --topo custom,3,4 --mac --arp --
   switch ovsk --controller remote
```

[4] POX: http://noxrepo.github.io/pox-doc/html/

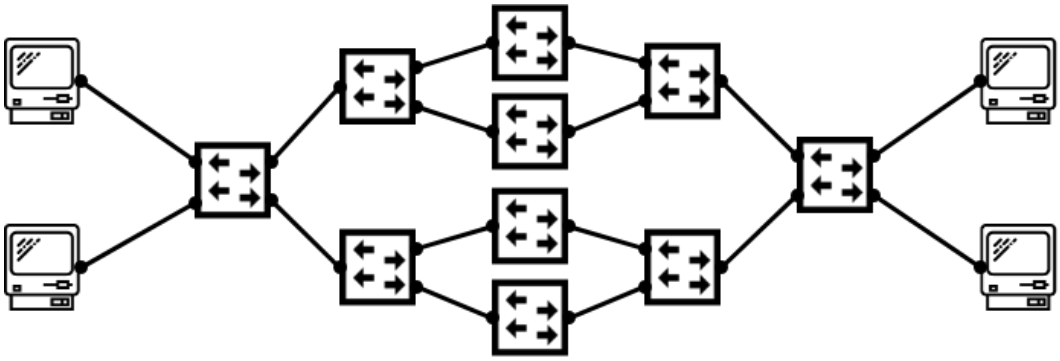

Figura 4: Topología 1 - 3 niveles y 4 hosts

Por una cuestión técnica, vamos a agregar como limitación, que los niveles de switches no van a ser mayores que 3 y la cantidad de hosts, no va a ser mayor que 5.

Para obtener la cantidad total de switches que va a tener la topología, se puede utilizar la siguiente función recursiva.

```
import math

def get_switch_amount(switch_level):
    sum = math.pow(2,switch_level-1)
    while (0 < switch_level-1):
        sum = sum + math.pow(2,switch_level-1)
        switch_level -= 1

    return sum
```

[5]

Se debe considerar que el controlador a utilizar para esta simulación, si bien no posee una gran complejidad, debe presentar las siguientes características:

- Los switches, deben aprender automáticamente la topología (l2_learning).
- Dado que la topología planteada, tiene bucles, se debe utilizar un mecanismo que los resuelva (openflow.spanning_tree).
- Se deben agregar los logs necesarios al controlador para poder verificar su funcionamiento, y poder contrastarlo con wireshark.

4.1.1. pingall

Una vez lograda la topología, se debe verificar el correcto funcionamiento de la red, y un mecanismo para hacerlo es mediante el comando *pingall*. Se debe correr el comando pingall en el entorno de mininet, y poder registrar el envio y recepcion de los mensajes, tanto en *Wireshark* como en los registros del controlador remoto.

```
mininet> pingall
```

4.2. Topología 2

Para la segunda topología, centraremos nuestros esfuerzos en el manejo del controlador, por lo que desarrollaremos una topología mas sencilla. Se tendrá una cantidad de switches variable, formando una cadena, en cuyos extremos se tienen dos hosts 5.

[5] Correción: Sebastian Campoamor

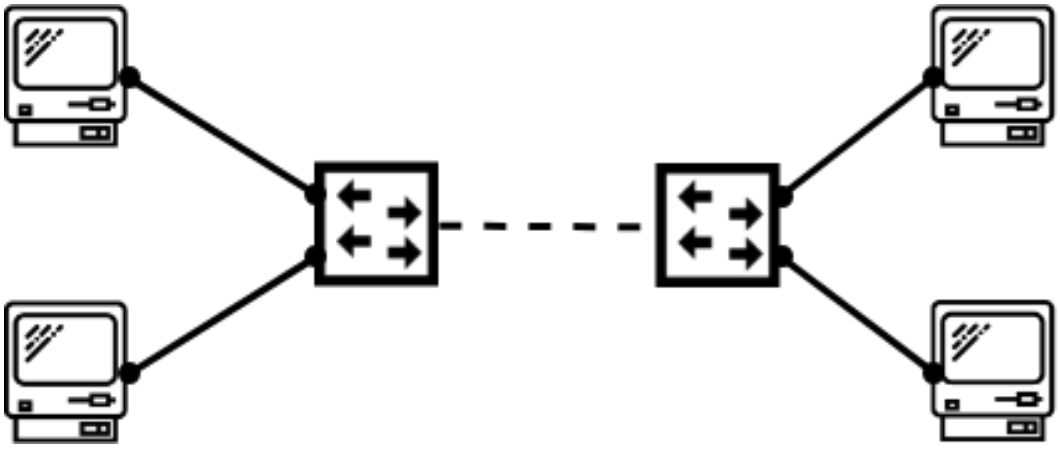

Figura 5: Topología 2

4.2.1. Firewall

Una vez verificado el correcto funcionamiento de la red, vamos a modifcar el controlador, para que funcione como un *Firewall*, con una serie de reglas definidas a continuacioón:

1. Se deben descartar todos los mensajes cuyo puerto destino sea 80.
2. Se deben descartar todos los mensajes que provengan del host 1, tengan como puerto destino el 5001, y esten utilizando el protocolo UDP.
3. Se debe elegir dos hosts cualquiera, y los mismos no deben poder comunicarse de ninguna forma.

Para la verificación del correcto funcionamiento de las reglas, se utilizará *iperf*[6]. Iperf es una herramienta para realizar pruebas de rendimiento en redes. Para realizar una prueba, se deben establecer tanto un servidor para recibir tráfico, como un cliente para generarlo.

En el directorio */pox/pox/samples*, se deja un archivo extraído del curso de SDN de Coursera [7] que se puede usar como base para la implementación del firewall.

```
'''
Coursera:
- Software Defined Networking (SDN) course
-- Programming Assignment: Layer-2 Firewall Application
Professor: Nick Feamster
Teaching Assistant: Arpit Gupta
'''

from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.revent import *
from pox.lib.util import dpidToStr
from pox.lib.addresses import EthAddr
from collections import namedtuple
import os
''' Add your imports here ... '''

log = core.getLogger()

''' Add your global variables here ... '''
```

[6] iperf:https://github.com/esnet/iperf

[7] Coursera-SDN - https://github.com/PrincetonUniversity/Coursera-SDN/blob/master/assignments/simple-controller/firewall.py

```
class Firewall (EventMixin):

    def __init__ (self):
        self.listenTo(core.openflow)
        log.debug("Enabling Firewall Module")

    def _handle_ConnectionUp (self, event):
        ''' Add your logic here ... '''

def launch ():
    '''
    Starting the Firewall module
    '''
    core.registerNew(Firewall)
```

5. Código

5.1. mininet

- Crear un switch

```
self.addSwitch('switch_name')
```

- Crear un host

```
self.addHost('host_name')
```

- Crear un enlace entre dispositivos

```
self.addLink(dev1, dev2)
```

5.1.1. Ejemplo

A continuación se muestra una clase con una topología que cuenta con dos switches conectados a tres hosts como la mostrada en la figura 6.

```
from mininet.topo import Topo

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

    # Create switch
    s1 = self.addSwitch('switch_1');
    s2 = self.addSwitch('switch_2');

    # Create hosts
    h1 = self.addHost('host_1'))
    h2 = self.addHost('host_2'))
    h3 = self.addHost('host_3'))

    # Add links between switches and hosts
    self.addLink(s1, s2)
    self.addLink(s1, h1)
    self.addLink(s1, h2)
```

```
    self.addLink(s2, h3)

topos = { 'customTopo': Topo }
```

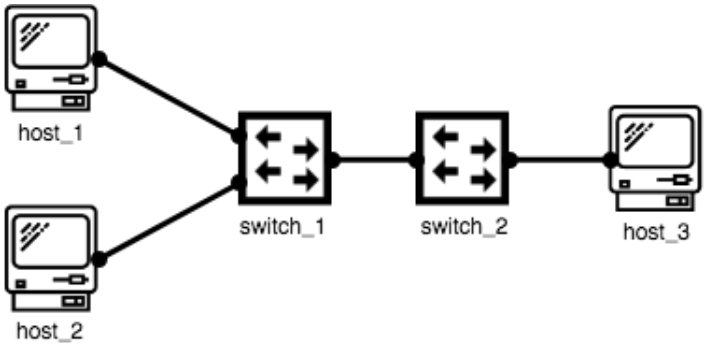

Figura 6: Topología Ejemplo

5.2. Pox

- Escuchar un evento

```
core.openflow.addListenerByName("PacketIn",_handle_PacketIn)
```

- Eventos

```
ConnectionUp
ConnectionDown
PortStatus
FlowRemoved
Statistics Events
PacketIn
ErrorIn
BarrierIn
```

- Enviar un mensaje

```
msg = of.ofp_packet_out()
msg.actions.append(of.ofp_action_output(port = outport))
msg.buffer_id = <some buffer id, if any>
connection.send(msg)
```

6. Entrega

La entrega debe contar con un informe donde se demuestre conocimiento en todas las herramientas utilizadas (mininet, pox, wireshark, iperf), así como también los resultados de las simulaciones (capturas de wireshark y logs del controlador).

También se pide entregar los archivos fuentes utilizados para ambas tecnologías y el controlador de firewall. El código fuente se analizará con una herramienta de detección de similaridad, y se almacenará para futuros cuatrimestres.

Para probar las simulaciones se deben indicar los comandos a ejecutar, el orden de los mismos y los parámetros que reciben. Se sugiere que junto con la entrega se adjunten scripts encargados de realizar las simulaciones. En el informe debe figurar un README de como correr los diferentes scripts y que se espera luego de la ejecución de cada uno.

La entrega consta de dos partes, la primer parte se hará presencial con una serie de pruebas a realizar en clase. Se pedirá que se corran las pruebas que verifiquen el correcto funcionamiento de la topología 1, así como también el firewall de la topología 2. También se pedirán algunos cambios durante la entrega, para poder corroborar el conocimiento de las

herramientas presentadas. La segunda parte se hace de forma virtual, y se sube el informa, los archivos de código fuente y los scripts necesarios al campus.

La fecha de entrega por campus esta pautada para el día martes **12 de Junio de 2018** a las **19:00 hs**. Esa misma clase se dividirá en dos, en la primera mitad se dictará clase normalmente (Clase #14: WAN), y en la segunda se realizarán las pruebas mencionadas previamente. No se aceptarán entregas ni se realizarán pruebas fuera de término.

7. Links Útiles

- Mininet: `http://mininet.org/walkthrough/`
- pox: `https://noxrepo.github.io/pox-doc/html/`
- Openflow: `https://www.opennetworking.org/technical-communities/areas/specification/open-datapath/`
- Open vSwitch: `http://www.openvswitch.org/`
- Visualizador de topologías: `http://demo.spear.narmox.com/app/?apiurl=demo#!/mininet`

Referencias

[1] D. Thaler and C. Hopps. Multipath Issues in Unicast and Multicast Next-Hop Selection. RFC 2991 (Informational), November 2000.

[2] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[3] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.