Estudio de técnicas de Ingeniería de Tráfico basadas en SDN

Study of SDN Traffic Engineering techniques

Betegón García, Miguel[1]
miguel.betegon@alumnos.unican.es

Junio, 2018

[1]Grado en Ingeniería de Tecnologías de Telecomunicación

1. COMPROBAR QUE FUNCIONA BIEN CON UNA SEGUNDA PANTALLA, QUE SE MUESTRAN LAS NOTAS EN UNA Y LA PRESENTACIÓN EN OTRA.
2. COMPROBARLO CON LAS PANTALLAS DE LA PRESENTACIÓN (MI ORDE NO TIENE VGA, PREGUNTAR SI HAY HDMI Y LLEVAR EL MIO POR SI ACASO)
3. EN LA TERMINAL, $: pdfpc main.pdf - -notes=right
4. https://bugs.kde.org/show_bug.cgi?id=152585
5. ESTOY COMPILANDO DESDE OVERLEAF CON XUALATEX. CON XELATEX PUEDE FUNCIONAR CREO, PERO CON PDFlatex no. PDFlatex no entiende la fuente (fira) y utiliza las suyas
6. CUIDADO, SE ME PONE EL TEXTO DE COLOR BLANCO Y NO SE VE, ES CULPA DE USAR NOTAS (PFGPAGES) Y XELATEX
7. SI USO TODOs SE JOROBA LA FUENTE Y DA ERRORES. POR ESO HE PUESTO EL TEXTO QUE NO ES DE NOTAS SINO DE COSAS PARA CAMBIAR DE LAS DIAPOSITIVAS EN ROJO
8. ECHAR UN OJO A LA PORTADA, POR SI NO PRESENTARÁ EN JUNIO (Pone Junio, 2018)

Índice

1. Son los capítulos del trabajo, el índice hace de ”Estructura de proyecto”.

Introducción

Motivación y objetivos I

Las redes definidas por software (SDN) surgen a principios de 2010 por necesidad:

- La mayoría de las redes tradicionales fueron diseñadas para aplicaciones cliente-servidor que se ejecutan en una infraestructura no virtualizada.

SDN se ha establecido en la adultez temprana como un producto conocido.

Es una realidad que muchas de las empresas y proveedores de servicios de todo el mundo ya han adoptado.

1. NO son la solución a un problema sin resolver sino que resuelven de una forma mas eficiente que las soluciones tradicionales.

2. SDN ha crecido más allá de su adolescencia y euforia prematura...

3. NO es una próxima novedad en el horizonte de la creación de redes...

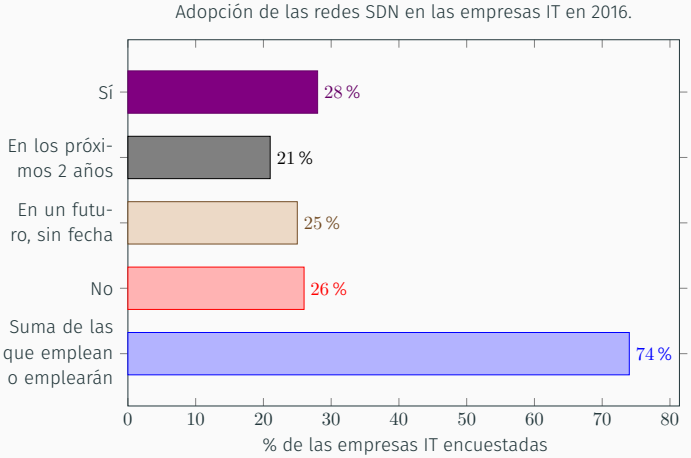
Motivación y objetivos II

Rohit Mehra y *Brad Casemore* en su previsión sobre SDN publicada en 2016:

> *Virtualization, cloud, mobility, and now the Internet of Things (IoT) have exposed the limitations of traditional network architectures and operational models.*

1. VER SI AÑADO ESTA DIAPO O NO, DECIDIRLO CUANDO TENGA TODAS LAS DIAPOS HECHAS, PARA VER SI SON DEMASIADAS. OCUPARÍA 15 SEGUNDOS EXPLICAR ESTA DIAPO, POR TIEMPO NO HABRÍA PROBLEMA

Motivación y objetivos III



Fuente: Channel Insider Networking - Michael Vizard

1. Debido a la creciente demanda en las redes, en estos años se ha visto una evolución en el mercado de SDN.

2. Es por esto y por el TFG DE RUBEN que surge el proyecto.

OBJETIVOS

» Exponer dos casos de uso real de las redes SDN.

»» Aplicar técnicas de ingeniería de tráfico en dichos casos.

»»» Implementarlos en Mininet haciendo uso del controlador `Ryu` y un script en `Python`.

1. COMPROBAR QUE LOS TITULOS DE ESTAS DIAPOS ESTÁN BIEN: I, II, III, IV, V, ...

Conceptos teóricos

Estado del arte

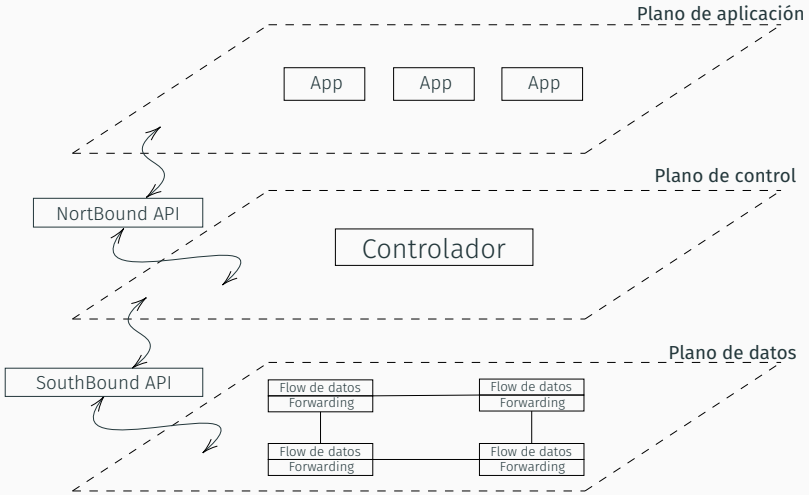


Figura 1: Arquitectura de alto nivel SDN.

1. La Open Networking Foundation define una arquitectura de alto nivel para SDN con tres capas o planos principales, como se muestra en la figura.
2. SDN separa Plano de Control - Plano de Datos
3. Simplifica la operación en el plano de datos -> dispositivos de red menos costosos.
4. Centraliza el control (toma de decisiones) en la red.
5. Programabilidad de la red, administración simplificada y autónoma.
6. Estimula la aplicacion $\Rightarrow$ abre mercados y oportunidades para todo el sector.
7. COMPROBAR SI EXISTEN MÁS DIAPOS CON EL MISMO TITULO PARA PONER: I, II, III, IV, V, ...

Configuración física

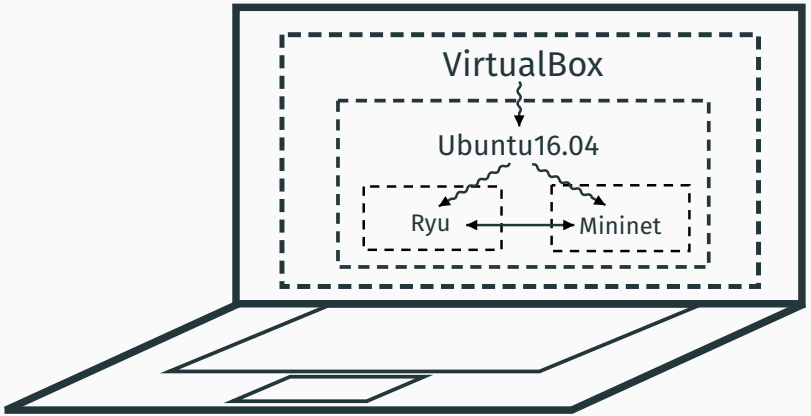


Figura 2: Esquema físico del proyecto.

1. Explicar de fuera a dentro PC->vBox->Ubuntu->Ryu,Mininet

1. PONER LA LETRA MAS GRANDE? (large) PARA LAS DIAPOS COMO ESTAS? CON TEXO. O SINO PARA EL TEXTO DE TODAS LAS DIAPOS. VERLO UNA VEZ ACABADO TODO.

OpenFlow

Protocolo estandarizado por Open Networking Foundation en 2013 que define la comunicación hacia el sur (Southbound) entre un controlador y un switch OpenFlow.

El tráfico se clasifica en flows en función de sus características.

1. PONER LA LETRA MAS GRANDE? (large) PARA LAS DIAPOS COMO ESTAS? CON TEXO. O SINO PARA EL TEXTO DE TODAS LAS DIAPOS. VERLO UNA VEZ ACABADO TODO.
2. COMPROBAR QUE LOS TITULOS DE ESTAS DIAPOS ESTÁN BIEN: I, II, III, IV, V, ...

Protocolo OpenFlow II

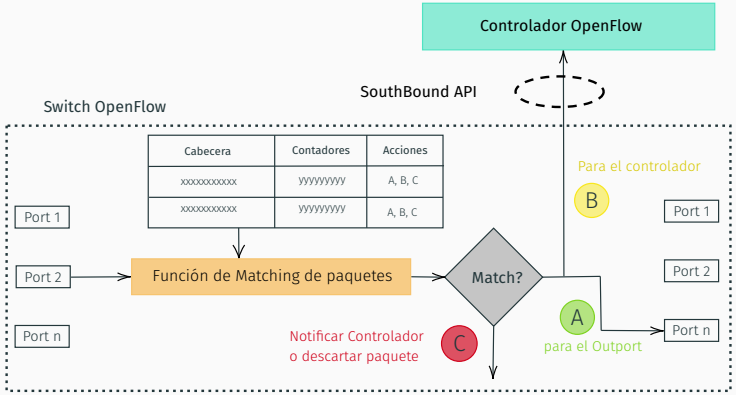


Figura 3: Switch OpenFlow. Operación básica.

1. COMPROBAR QUE LOS TITULOS DE ESTAS DIAPOS ESTÁN BIEN: I, II, III, IV, V, ...

Elementos

- Ryu
- VirtualBox
- Mininet
- iPerf

1. Contar para que se he usado cada uno.
2. COMPROBAR QUE LOS TITULOS DE ESTAS DIAPOS ESTÁN BIEN: I, II, III, IV, V, ...

Ingeniería de Tráfico

Es una aplicación de red importante que estudia la medición y gestión del tráfico.

Diseña mecanismos de enrutamiento para guiar el tráfico de red a fin de mejorar la utilización de los recursos y cumplir mejor los requisitos de QoS.

1. ESTAS NOTAS YA SE HABRÁN CONTANDO EN DIAPOS ANTERIORES, EN LO QUE ES SDN. PERO BUENO, NO PASA NADA POR REPETIR QUE ESAS CARACTERISTICAS QUE TIENE SDN VAN BIEN PARA LA ING. TRÁFICO.
2. PONER LA LETRA MAS GRANDE? (large) PARA LAS DIAPOS COMO ESTAS? CON TEXO. O SINO PARA EL TEXTO DE TODAS LAS DIAPOS. VERLO UNA VEZ ACABADO TODO.
3. En comparación con las redes tradicionales, SDN tiene muchas ventajas para ser compatible con TE debido a sus características distintivas, como el aislamiento de los planos de control y datos, el control centralizado global y la programabilidad del comportamiento de la red.

QoS

Conjunto de estándares y mecanismos que garantizan un rendimiento de alta calidad para aplicaciones críticas.

Los administradores de red pueden usar los recursos existentes de manera eficiente y garantizar el nivel de servicio requerido.

→ Sin expandir de forma reactiva ni aprovisionar en exceso o sobredimensionar sus redes.

1. El concepto de calidad de QoS es aquel en el que los requisitos de algunas aplicaciones y usuarios son más críticos que otros, lo que significa que parte del tráfico necesita un tratamiento preferencial.

2. OBJETIVO: proporcionar un servicio de entrega preferencial para las aplicaciones que lo necesitan asegurando un ancho de banda suficiente, controlando la latencia y reduciendo la pérdida de datos.

1. PONER LA LETRA MAS GRANDE? (large) PARA LAS DIAPOS COMO ESTAS? CON TEXO. O SINO PARA EL TEXTO DE TODAS LAS DIAPOS. VERLO UNA VEZ

Escenarios existentes

- Los balanceadores de carga usan hardware dedicado.
  - Costoso e inflexible.
  - Contienen pocos algoritmos.
  - No son programables (`Vendor-Locked`).
- QoS se implementa en los routers.
  - Es necesario configurar cada router para activar QoS.

1. Hoy en día Redes -> mucho tráfico (miles clientes y cumplir requisitos)
2. 1 servidor no soporta esa carga -> Varios servidores con Balanceador.
3. Clientes envían al balanceador. Este reenvía a los servidores según su estrategia.
4. Los administradores de red no pueden usar su propios algoritmos de balanceo.
5. fácil si es un area local. Red extensa es una MOVIDA.

Mejora de los escenarios Existentes con SDN

- Los balanceadores de carga basados en SDN presentan ventajas:
  - No se necesita hardware dedicado (menos costoso).
  - Mejoran el rendimiento del balanceador
  - Reducen la complejidad de su implementación.
  - Son programables y permiten diseñar e implementar estrategias propias.
- Con SDN se puede tener el control de QoS centralizado:
  - Fácil de implementar en redes de mayor tamaño.
  - Con el controlador SDN se implementa QoS directamente en los switches.
  - Los routers no necesitan esa capacidad de cómputo ”menos inteligencia”.

1. Soluciona el problema de la red a mayor escala.
2. los routers que se encarguen de sus cometidos.
3. Se facilita la gestión de una forma eficaz de QoS en toda la red.
4. ¡¡¡¡¡Se puede cambiar en cualquier momento las normas QoS dinámicamente!!!!!!

Routing multicamino con balanceador de carga

Balanceador de carga con routing multicamino

El balanceador de carga con routing multicamino es uno de los casos de uso más comunes e implementados de SDN.

En nuestro caso, se desarrolla en un script en *Python* que hemos llamado `multicamino.py`.

1. No se profundiza del todo -> es uno de los más usados y documentados.

2.

3. Luego se verá como ejecutarlo y comprobar que funciona. ahora analizaremos la parte teórica.

Routing multicamino

Técnica que explota los recursos de la red mediante la propagación del tráfico desde un nodo de origen a un nodo de destino por medio de múltiples rutas a lo largo de la red.

- Balanceo de carga
- Agregación de ancho de banda.
- Minimización de retardo de extremo a extremo.
- Aumento de la tolerancia a fallos (mejorar fiabilidad).

1. Existen 3 partes: descubrimiento, distribución de trafico y mantenimiento de rutas

2. Nos centramos en el decubrimiento.La distribución del tráfico se hará de acuerdo a lo que queremos implementar, un balanceador de carga, así que repartiremos el tráfico por igual a través de las rutas encontradas y consideraremos que los caminos existentes se mantienen fijos durante la realización de este trabajo.

3. VER SI AÑADO UNA DIAPO MAS CON LO DE LAS 3 PARTES IMPORTANTE,SOLO SI NECESITO MAS DIAPOS (LO DUDO).

PathFinding Algorithms

Los algoritmos de Pathing son los encargados de obtener la ruta más corta entre dos puntos.

- DFS y BFS son dos algoritmos conocidos, que en la búsqueda agotan todas las posibilidades.
- Iteran sobre todos los caminos posibles hasta alcanzar el nodo de destino
- Se ejecutan en tiempo lineal, según la notación Big-$\mathcal{O}$ sería: $\mathcal{O}(N + E)$

1. se basan en Dijkstra->camino más corto en grafo con pesos.

2. A diferencia de Dijkstra, que no agota todas las posibilidades.
3. BFS-Búsqueda en Anchura, DFS-Búsqueda profundidad.

4. N=Nºnodos, E=Nºenlaces entre los nodos de la red.

DFS

- Búsqueda en profundidad del grafo.
- Explora todos los nodos en un grafo hasta encontrar el nodo más profundo y después retrocede con el propósito de encontrar otros posibles nodos.
- Hace uso de una pila (*stack*).

1. ELEGIDO PARA ESTE TRABAJO.

2. Útil por el uso de la pila, modificamos para encontrar las rutas posibles entre dos nodos.

Iteraciones DFS

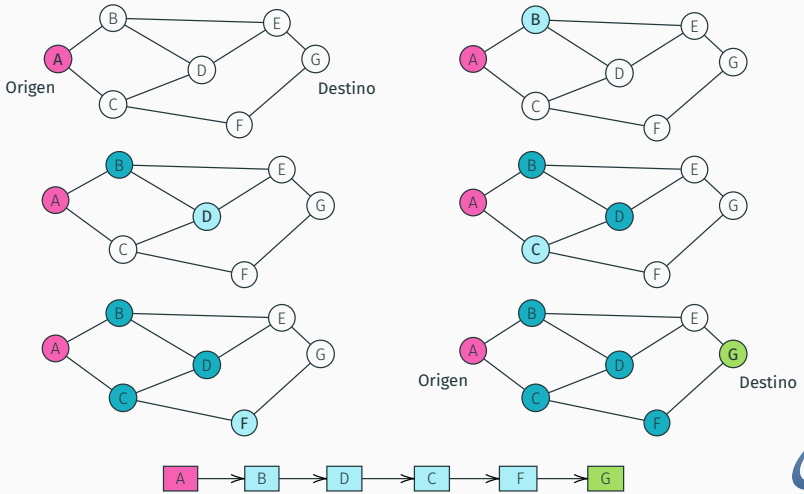


Figura 4: Iteraciones del algoritmo DFS.

1. Para grafos dirigidos el proceso es análogo, sólo cambia el significado de ”adyacencia”

2. Se van visitando los nodos y se marcan como visitados.

Cálculo del coste por camino

DFS devuelve una lista con las rutas, pero sin pesos.

Tenemos que medir el coste de los caminos o rutas:

1. Calcular todos los costes de enlaces que haya en la ruta.

   $Cost(l) = \frac{BW_{Reference}}{BW(l)}$;

   $BW(l) = \text{mín } (BW_{Switch1}, BW_{Switch2})$

2. Calcular el coste total de la ruta (sumar los costes de enlaces).

1. Calculamos el coste como en OSPF (protocolo de red)

2. BW enlace = BW de la interfaz de un switch.

3. COSTE DE ENLACE = 1 -> COSTE DE LA RUTA ES EL Nº DE ENLACES (FÁCIL PARA ENTENDERLO).

Bucket weight en OpenFlow

Group→Group table→buckets (bucket weight)→acciones

Los *Grupos* en OpenFlow representan una serie de puertos como una entidad única para el envío de paquetes. Existen varios tipos de grupos, interesándonos los *Select* para el multicamino.

Cada grupo consta de una *Group table* formada de entradas, *buckets*, y cada *bucket* contiene una serie de acciones que se aplican antes de enviar el paquete por el puerto de salida del *bucket* seleccionado, que será el de mayor *bucket weight*.

$$bw(p) = \left(1 - \frac{Cost(p)}{\sum_{i=0}^{i<n} Cost(i)}\right) \times 10$$

1. Contrario al coste, dónde menor coste es mejor, aquí es al revés.
2. Tenemos por tanto que solucionar el problema que nos surge, ajustar el criterio de los bucket weights con los costes de las ruta
3. Select solo se ejecuta un bucket.
4. p=path,ruta. bw(p)=entre 0 y 10
5. si un switch tuviera dos puertos con enlace, pondríamos cada uno de esos puertos como un bucket en la group table

Implementación

Requirements

1. Máquina corriendo `Ubuntu 16.04.3` o superior.
2. `Mininet v2.2.2` o superior.
3. `Ryu v4.0` o superior.
4. `iPerf` e `iPerf3`

1. 18.04 ha salido hace poco (compatible).
2. Que tenga miniedit.
3. Actualizaciones en la API.
4. uno para cada caso.
5. CUIDADO, ESTA DIAPO ES IGUAL QUE UNA DE LAS PRIMERAS (PENSAR SI QUITO LA PRIMERA Y DEJO ESTA O QUE...)

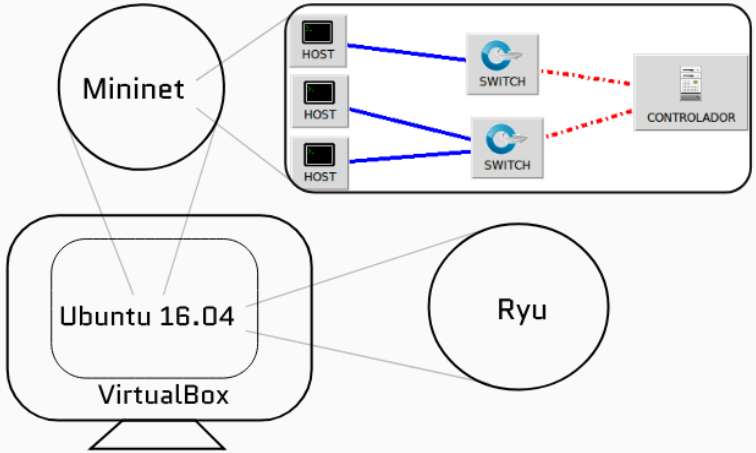
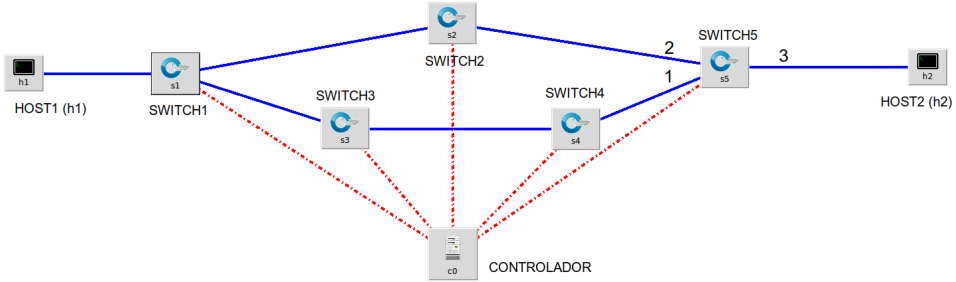


Figura 5: Entorno de desarrollo.

1. Como el mostrado anteriormente pero más detallado
2. igual para los dos casos de uso, solo cambia que en uno se usa Miniedit.
3. SUPONEMOS QUE YA TENEMOS CONFIGURADO/INSTALADO TODO LO DE LA FIGURA.

Balanceador de carga multicamino I

Definición del escenario de aplicación



1. SOLUCIONAR PROBLEMA CON EL FONDO BLANCO DE LA CAPTURA DE MINIEDIT.
2. Primer caso de uso.
3. Diseñamos en Miniedit y configuramos el controlador. IP loopback, puerto 6633, remote controller y OpenFlow 1.3
4. Ping H1-H2 para ver que funciona.

Balanceador de carga multicamino II

Inicio del controlador y descubrimiento de rutas

```
tfg@tfgVM
tfg@tfgVM:~$ryu-manager --observe-links multicamino.py
loading app multicamino.py
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app ryu.topology.switches of Switches
instantiating app multicamino.py of ProjectController
instantiating app ryu.controller.ofp_handler of OFPHandler
Se ha llamado a switch_features_handler
Se ha llamado a switch_features_handler
Se ha llamado a switch_features_handler
Se ha llamado a switch_features_handler
Se ha llamado a switch_features_handler
Caminos disponibles de  1  a  5  :  [[1, 3, 4, 5], [1, 2, 5]]
[1, 2, 5] coste =  2
[1, 3, 4, 5] coste =  3
Camino instalado en  0.00248599052429

Caminos disponibles de  5  a  1  :  [[5, 4, 3, 1], [5, 2, 1]]
[5, 2, 1] coste =  2
[5, 4, 3, 1] coste =  3
Camino instalado en  0.00211501121521
```


1. Se vería solo hasta se ha llamado. para los swtiches. Despues hacemos un ping para que se tenga constancia de los hosts.

2. ir a la diapo anterior y que las rutas concuerdan.

3. Costes (se ha puesto que el ancho de banda del enlace sea el mismo que el de referencia, así el coste de enlace es 1, y coste total=nº enlaces).

4. Salen los caminos de ida y vuelta porque suponíamos que eran bidireccionales

Balanceador de carga multicamino III

```
tfg@tfgVM:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s5

OFPST_FLOW reply (OF1.3) (xid=0x2):

cookie=0x0, duration=4345.812s, table=0, n_packets=4,
n_bytes=392, ip, nw_src=10.0.0.2,
nw_dst=10.0.0.1 actions=group:2742512190

cookie=0x0, duration=4345.812s, table=0, n_packets=2,
n_bytes=84, priority=1, arp, arp_spa=10.0.0.2,
arp_tpa=10.0.0.1 actions=group:2742512190
```

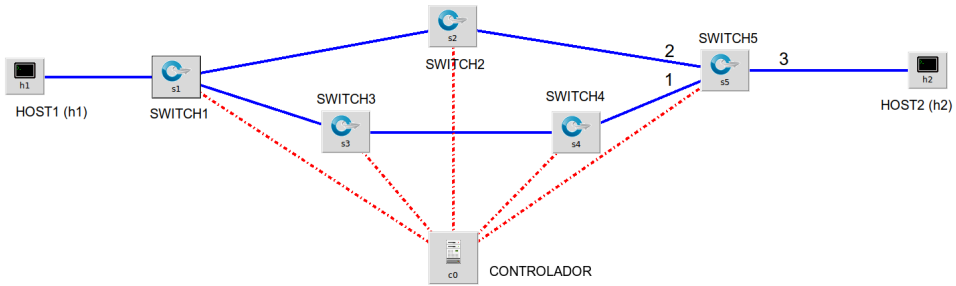
1. Veamos si las rutas encontradas han sido instaladas en s5. Este es el primer paso. haremos flows->grupos-> buckets->actions

2. Cada flow se encarga del matching de pkts IP y ARP (uno de cada)

3. Se ridirigen los dos al mismo group:...

4. YO QUITABA ESTA DIAPO Y PONIA DIRECTAMENTE LA SIGUIENTE, LA DE LOS GROUPS (SON DEMASIADAS DIAPOS SI NO).

Balanceador de carga multicamino IV




```
tfg@tfgVM:~$ sudo ovs-ofctl -O OpenFlow13  dump-groups s5

OFPST_GROUP_DESC reply (OF1.3) (xid=0x2):

group_id=2742512190,type=select,
bucket=weight:6,watch_port:2, actions=output:2,
bucket=weight:4,watch_port:1,actions=output:1
```

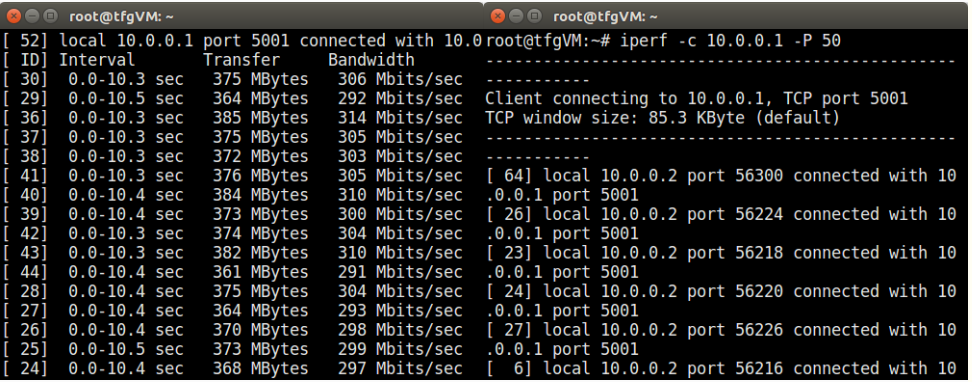

1. Hay un solo grupo con 2 buckets en la Group Table

2. bucket weight se corresponde con el coste: 2/3=4/6=> 2/(2+3)=4/(4+6)=0.4

3.

Balanceador de carga multicamino V

Creación de tráfico TCP con iPerf.



1. Creamos tráfico TCP con iPerf. 50 clientes en paralelo.

2. en Open vSwitch ultimas versiones para trafico TCP se realiza un hash simetrico (igual ambos sentidos) con los puertos de origen y destino y se elige el bucket teniendo en cuenta el hash (hash1->rutaX siempre...)

3. mirando la figura de la derecha (CLIENTE) se ve como se conectan los clientes cada uno con su puerto (port 56300, port 56224,...)

Balanceador de carga multicamino VI

Comprobación del balanceo de carga.

```
tfg@tfgVM:~$sudo ovs-ofctl -O OpenFlow13 dump-ports s5
OFPST_PORT reply (OF1.3) (xid=0x2): 4 ports
port  1: rx pkts=5376164,  bytes=355469919,
         tx pkts=8304407,  bytes=280097226367
port  2: rx pkts=8157000,  bytes=540397987,
         tx pkts=13448402, bytes=465062254229
port  3: rx pkts=21746866, bytes=745159117812,
         tx pkts=13530184, bytes=895685867
```

Ruta 1 $Traffic(1) = \frac{Tx2}{Rx3} \times 100 = \frac{13448402}{21746866} \times 100 = 61,83\,\%$

Ruta 2 $Traffic(2) = \frac{Tx1}{Rx3} \times 100 = \frac{8304407}{21746866} \times 100 = 38,18\,\%$

1.

2.

3.

AA

1.
2.

1.

2.

3.

AA

1.
2.

1.
2.
3.

Conclusiones y líneas futuras