

Estudio de técnicas de Ingeniería de Tráfico basadas en SDN

Study of SDN Traffic Engineering Techniques

Betegón García, Miguel¹

miguel.betegon@alumnos.unican.es

Julio, 2018

¹Grado en Ingeniería de Tecnologías de Telecomunicación



1. EN LA TERMINAL, \$: pdfpc main.pdf - -notes=right

1. Introducción
2. Conceptos teóricos
3. Routing multicamino con balanceador de carga
4. Implementación
5. Conclusiones y líneas futuras

1. Comenzaré con ... le seguirán los conceptos teóricos en los que se basa el proyecto, después se comentará la parte técnica antes de pasar a la implementación y por último acabaré con las conclusiones y líneas futuras.



Introducción

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

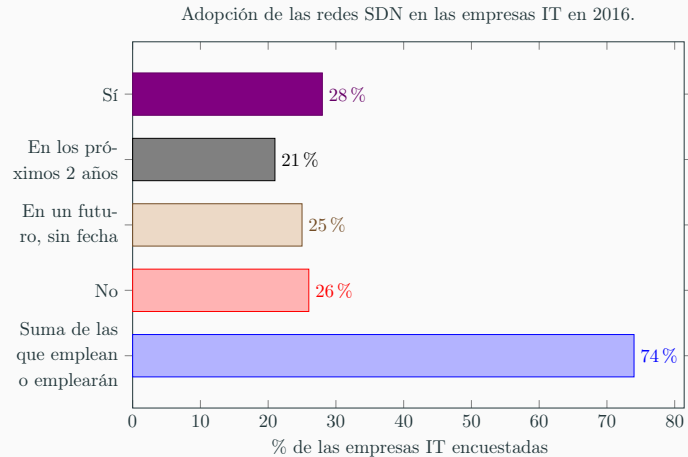
- La mayoría de las redes tradicionales fueron diseñadas como aplicaciones cliente-servidor que se ejecutan en una infraestructura no virtualizada.

SDN se ha establecido 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..
2. redes tradicionales fueron diseñadas como aplicaciones cliente-servidor que se ejecutan en una infraestructura no virtualizada.
3. Es una realidad que muchas de las empresas y proveedores de servicios de todo el mundo ya han adoptado. ENLAZAR CON SIGUIENTE DIAPO!!





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 las redes SDN.
- »» Aplicar técnicas de ingeniería de tráfico en dichos casos.
- »»» Implementarlos en un emulador con visión hacia la docencia.

1. 2 CASOS DE USO

2. APLICAR TE EN LOS CASOS

3. IMPLEMENTAR EN EMULADOR, USAR PARA DOCENCIA



Conceptos teóricos

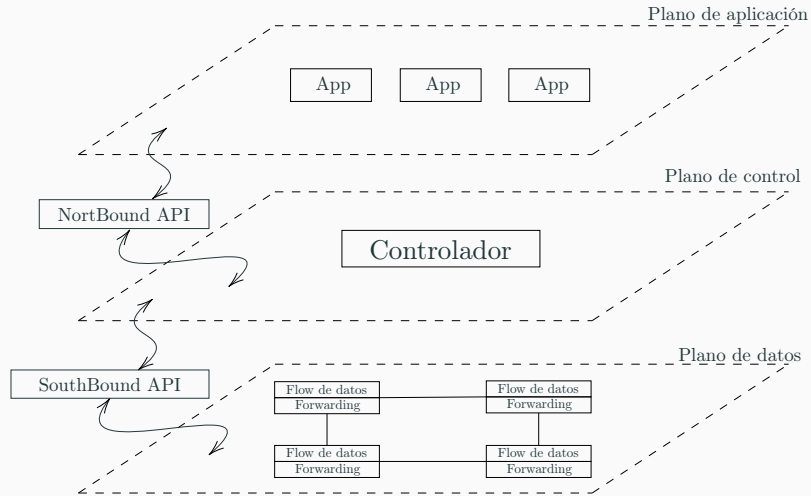


Figura 1: Arquitectura de alto nivel SDN.

1. Se divide en 3 CAPAS/PLANOS
2. separación 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. Estimula la aplicacion => abre mercados y oportunidades para todo el sector.



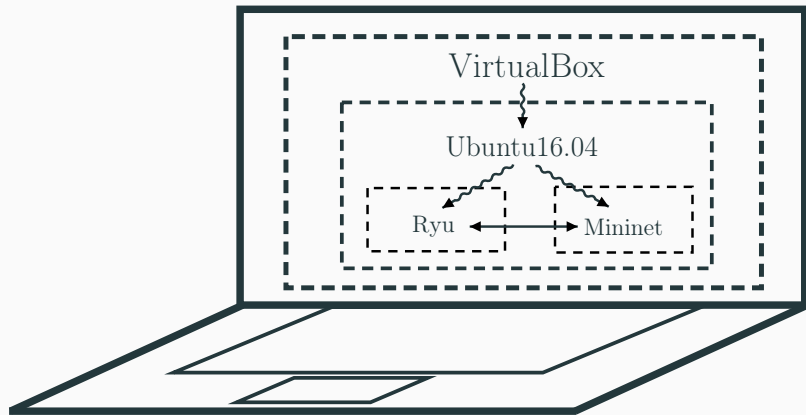


Figura 2: Esquema físico del proyecto.

1. MININET->RYU->Ubuntu->VBOX->PC



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. Protocolo que define la comunicación entre switch y controlador.
2. Tráfico se clasifica en flujos en función de sus características. -> ENLAZA CON LA SIGUIENTE DIAPO.



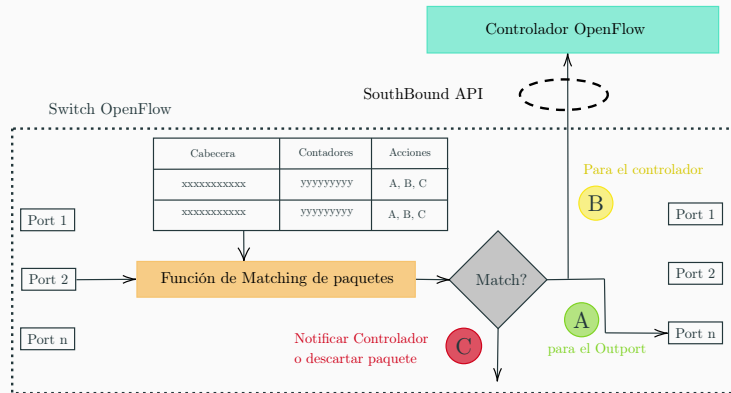


Figura 3: Switch OpenFlow. Operación básica.

1. Dependiendo de estas características, cuando un switch recibe un flow se pasa por una función de reconocimiento de paquetes y según su resultado, se envía bien al puerto de salida del switch para seguir con su reenvío, se informa al controlador o bien se descarta el paquete.-
2. ESTÁS FUNCIONES PUEDEN SER MÁS O MENOS COMPLEJAS EN FUNCIÓN DE LAS TÉCNICAS DE INGENIERÍA DE TRÁFICO QUE APLIQUEMOS -> SIG DIAPO.



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 calidad de servicio (QoS).



1. Diseña mecanismos de enrutamiento para guiar el tráfico con el fin de MEJORAR UTILIZACIÓN DE RECURSOS y cumplir requisitos QoS.
2. En comparación con las redes tradicionales, SDN tiene muchas ventajas para ser compatible con TE debido a sus características distintivas
3. islamamiento de los planos de control y datos, control centralizado y la PROGRAMABILIDAD.

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. Existen aplicaciones y usuarios son más críticos que otros, lo que significa que parte del tráfico necesita un tratamiento preferencial.
2. aseguran un ancho de banda , controlan la LATENCIA y reduciendo la pérdida de datos.
3. Evitar el sobredimensionamiento de la red.



- 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. Clientes envían al balanceador. Este reenvía a los servidores según su estrategia.
3. los LB son Costosos, Pocos algoritmos, no programables
4. QOS SE IMPLEMENTA EN ROUTERS(unos a unos), difícil en una red extensa.

- 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. LB - hace de LB un switch (no se necesita hardware dedicado)
2. LB - Son PROGRAMABLES-> estrategias propias
3. QOS - fuera inteligencia en routers (se encarguen de reenviar solo).
4. QOS - desde un mismo punto se controla todo DINAMICAMENTE.
5. QOS - Se cambia DINAMICAMENTE.

Routing multicamino con
balanceador de carga

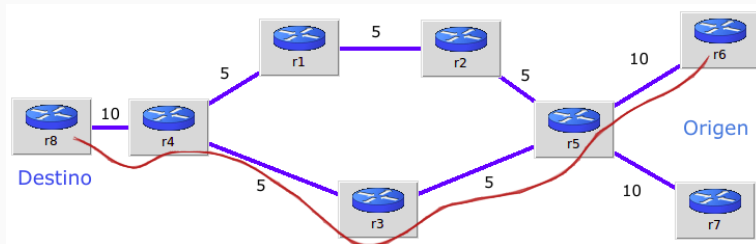


Figura 4: Problema del pez.

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. Uno de los casos más implementados en SDN.
2. 2 partes : BALANCEADOR - MULTICAMINO
3. SE NECESITAN LOS CAMINOS PARA APLICAR DESPUÉS EL BALANCEO ASÍ QUE EN EL ROUTING MULTICAMINO -> SIGUIENTE DIAPO.

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. SE BASA EN LA PROPAGACIÓN DEL TRÁFICO DE UN NODO A OTRO POR MEDIO DE MÚLTIPLES RUTAS.

2. E.G. balanceo, agregación de BW, Minimizar retardo, Aumentar tolerancia a fallos.



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- O sería:
 $O(N + E)$

1. Se basan en encontrar la ruta más corta entre dos puntos
2. DFS BFS agotan todas las posibilidades.
3. BFS-Búsqueda en Anchura, DFS-Búsqueda profundidad.
4. VENTAJA se EJECUTAN en TIEMPO LINEAL



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, por el USO DE PILA
2. BUSQUEDA EN PROFUNDIDAD -> como se observa en la siguiente DIAPO.



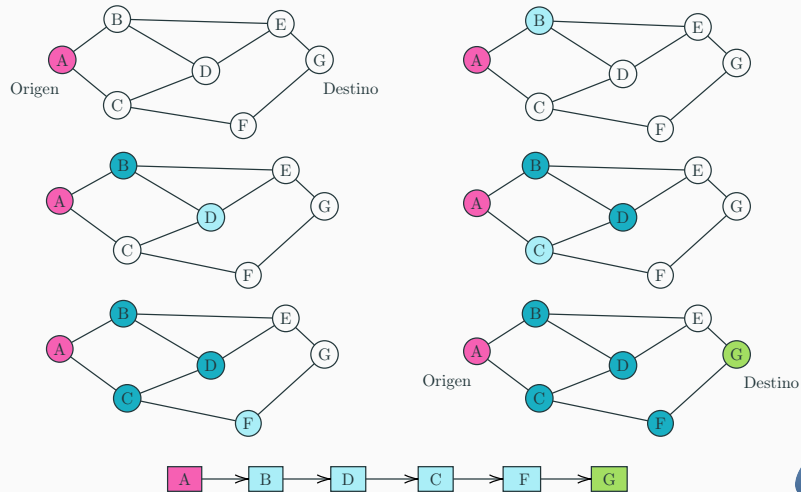


Figura 5: Iteraciones del algoritmo DFS.

1. ITERACIONES DFS hasta encontrar el destino.
2. En orden descendiente, se visita el vecino del origen, despues el vecino del vecino..
3. NO ACABA AQUÍ, se volvería atrás al ultimo nodo que queda en la pila (E).



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.

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

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

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



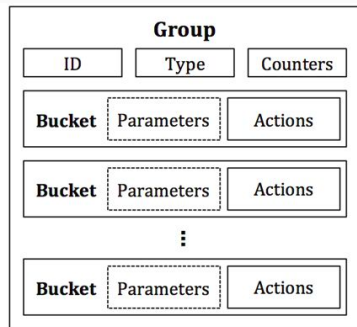
1. DFS DEVUELVE LAS RUTAS, PERO SIN PESOS.

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

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

4. COSTE DE ENLACE = 1 -> COSTE DE LA RUTA ES EL N° DE ENLACES (FÁCIL PARA ENTENDERLO).

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.

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



Figura 6: Grupo OpenFlow.

1. Openflow no entiende costes, sino BUCKET WEIGHTS, CONTRARIOS AL COSTE.
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. Una vez ajustado el cambio, toca la implementación → SIGUIENTE DIAPO.

Implementación

Requisitos

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

1. Requisitos necesarios para realizar el experimento.
2. Que tenga miniedit.
3. Actualizaciones en la API.



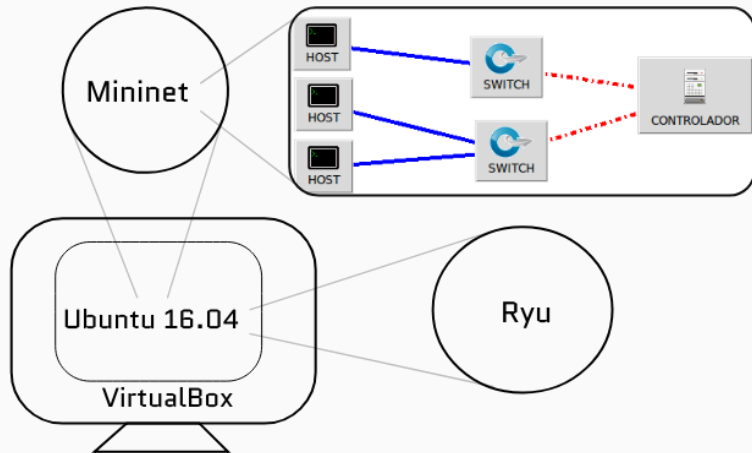
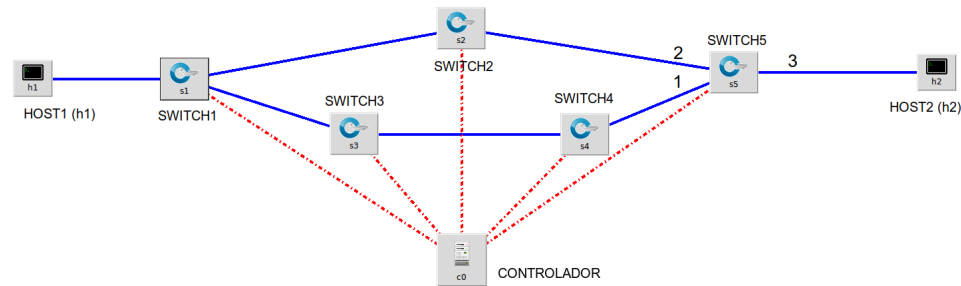


Figura 7: Entorno de desarrollo.

1. De dentro a fuera
(MININET->RYU->UBUNTU->VBOX..)
2. igual para los dos casos de uso



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



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. Una vez diseñada la topología, iniciamos el controlador para que descubra las rutas.
2. Las rutas se descubren hasta que no se realiza un ping H1-H2
3. ir a la diapo anterior y que las rutas concuerdan.
4. Salen los caminos de ida y vuelta porque suponíamos que eran bidireccionales

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

```
OFPPST_FLOW reply (OF1.3) (xid=0x2):
```

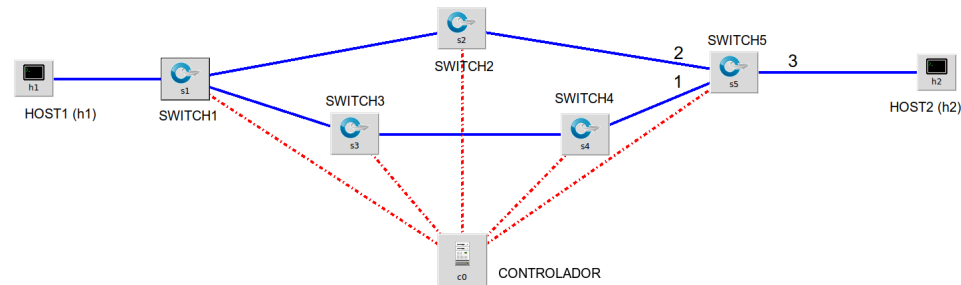
```
cookie=0x0, duration=4345.812s, table=0, n_packets=4,  
n_bytes=392, ip, nw_src=10.0.0.2,  
nw_dst=10.0.0.1 actions=group:2742512190
```

```
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 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 redirigen los dos al mismo grupo:-> Siguiente
DIAPO

Balanceador de carga multcamino IV



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

```
OFPT_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. Este es el grupo, que tiene 2 buckets en la Group Table.
2. bucket weight se corresponde con el coste: $2/3=4/6$
3. UNA VEZ COMPROBADO TODO \rightarrow CREAMOS TRÁFICO (SIGUIENTE DIAPO).

Creación de tráfico TCP con iPerf.

```
root@tfgVM: ~  
[ 52] local 10.0.0.1 port 5001 connected with 10.0.0.1  
[ ID] Interval      Transfer      Bandwidth  
[ 30] 0.0-10.3 sec   375 MBytes    306 Mbits/sec  
[ 29] 0.0-10.5 sec   364 MBytes    292 Mbits/sec  
[ 36] 0.0-10.3 sec   385 MBytes    314 Mbits/sec  
[ 37] 0.0-10.3 sec   375 MBytes    305 Mbits/sec  
[ 38] 0.0-10.3 sec   372 MBytes    303 Mbits/sec  
[ 41] 0.0-10.3 sec   376 MBytes    305 Mbits/sec  
[ 40] 0.0-10.4 sec   384 MBytes    310 Mbits/sec  
[ 39] 0.0-10.4 sec   373 MBytes    300 Mbits/sec  
[ 42] 0.0-10.3 sec   374 MBytes    304 Mbits/sec  
[ 43] 0.0-10.3 sec   382 MBytes    310 Mbits/sec  
[ 44] 0.0-10.4 sec   361 MBytes    291 Mbits/sec  
[ 28] 0.0-10.4 sec   375 MBytes    304 Mbits/sec  
[ 27] 0.0-10.4 sec   364 MBytes    293 Mbits/sec  
[ 26] 0.0-10.4 sec   370 MBytes    298 Mbits/sec  
[ 25] 0.0-10.5 sec   373 MBytes    299 Mbits/sec  
[ 24] 0.0-10.4 sec   368 MBytes    297 Mbits/sec  
root@tfgVM:~# iperf -c 10.0.0.1 -P 50  
Client connecting to 10.0.0.1, TCP port 5001  
TCP window size: 85.3 KByte (default)  
[ 64] local 10.0.0.2 port 56300 connected with 10.0.0.1 port 5001  
[ 26] local 10.0.0.2 port 56224 connected with 10.0.0.1 port 5001  
[ 23] local 10.0.0.2 port 56218 connected with 10.0.0.1 port 5001  
[ 24] local 10.0.0.2 port 56220 connected with 10.0.0.1 port 5001  
[ 27] local 10.0.0.2 port 56226 connected with 10.0.0.1 port 5001  
[ 6] local 10.0.0.2 port 56216 connected with 10.0.0.1 port 5001
```

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

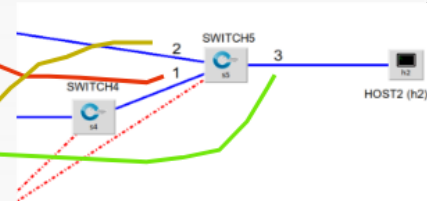
2. si solo usamos un cliente, se usaría una sola ruta y no podríamos balancear.

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

4. Una vez creado trafico -> COMPROBAMOS BALANCEO (SIGUIENTE DIAPO)

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,
tx pkts=8304407,
port 2: rx pkts=8157000,
tx pkts=13448402,
port 3: rx pkts=21746866,
tx pkts=13530184,
```



Ruta 1 $Traffic(1) = \frac{T_{x2}}{R_{x3}} \times 100 = 62 \%$

Ruta 2 $Traffic(2) = \frac{T_{x1}}{R_{x3}} \times 100 = 38 \%$

1. se reciben por el puerto3 y se envian por los otros dos
($p3=p1+p2$)
2. El resultado se asemeja al teórico 4/10,6/10. sabiendo que una pequeña fracción del tráfico es creado por las comunicaciones entres switches.
3. CONCLUSION: Se ha implementado correctamente el LB.

Hoy en día la mayoría de las redes hacen uso de la calidad de servicio.

→ SDN debe ofrecer al menos los mismos servicios y aplicaciones que las redes tradicionales.

Tarifa	Fase 1		Fase 2		Fase 3	
	Datos	Tasa	Datos	Tasa	Datos	Tasa
Estándar	2 MB	1 Mbps	1 MB	0.2 Mbps	–	–
Premium	∞	2 Mbps	∞	2 Mbps	∞	2 Mbps

Cuadro 1: Tarifas móviles y sus fases en función del consumo de datos.



1. Caso de uso incorpora QoS para ver que fácil se implementa en SDN.
2. se pretende mostrar como se pueden transferir los datos de una red en función de la prioridad basada en el origen de los datos (usuarios).
3. además reservar ancho de banda para una comunicación con el objetivo de unir dos puntos con un constante ancho de banda.
4. Se implementa la tarifa ESTANDAR ya que es la más compleja.

Uso de Colas en OpenFlow

Cada cola se corresponde con una fase de la tarifa, por lo que se crean tres colas, cada una con un id diferente.

→ Se establece el cambio entre las fases mediante programación.

Prioridades de las colas:

Cola 0 Prioridad 0

Cola 1 Prioridad 1

Cola 2 Prioridad 2



1. Para llevarlo a la práctica, se hace uso de las colas o queues en OpenFlow.
2. En el script se supone que ya existen y están configuradas dichas colas (las creamos antes de ejecutar el script).
3. Las prioridades hacen que por eso se empiece en la cola 0 y se vaya cambiando a la 1 y a la 2.
4. **SE INICIA EL CONTROLADOR IGUAL QUE EN EL OTRO CASO PERO CON TARIFAS.PY**

```
tfg@tfgVM:~$ sudo mn --mac \  
--switch ovsk,protocols=OpenFlow13 \  
--controller remote,ip=127.0.0.1,port=6633
```

```
tfg@tfgVM:~$ curl -X POST -d '{"port_name": "s1-eth1",\  
"type": "linux-htb", "max_rate": "1000000", "queues": [ \  
{"max_rate": "1000000"}, \  
{"min_rate": "200000", "max_rate": "200000"}, \  
{"min_rate": "0", "max_rate": "0"}]}' \  
http://localhost:8080/qos/queue/000000000000000001  
  
[{"switch_id": "000000000000000001",  
"command_result": {"result": "success", "details": {  
"0": {"config": {"max-rate": "1000000"}},  
"1": {"config": {"max-rate": "200000", "min-rate": "200000"}},  
"2": {"config": {"max-rate": "0", "min-rate": "0"}}}}}]
```

1. Toca por tanto, arrancar Mininet y configurar las colas
2. 3 colas, 1Mbps, 200ks, 0. (tarifa standar)
3. Output del comando nos devuelve SUCCESS (exito).

VÍDEO

1. Min 02:57



Conclusiones y líneas futuras

SDN es una clara alternativa que demandan los operadores de red.

- Las técnicas de ingeniería de tráfico pueden hacer uso de SDN.
- Se ha demostrado el uso de SDN para implementar un balanceador de carga y calidad de servicio.
- Se ha desarrollado un entorno de simulación que puede ser utilizado con fines docentes.



1. Técnicas TE pueden hacer uso de SDN
2. Se han abordado dos aplicaciones que usan técnicas TE en SDN y se ha visto reflejada su rápida implementación y fácil aprendizaje.
3. Entorno de simulación con fines docentes.

Líneas futuras destacadas

- Otros tipos de casos de uso que incorporen técnicas de ingeniería de tráfico en SDN.
- Investigación de nuevas técnicas de ingeniería de tráfico.
- Nuevas aplicaciones SDN
- Seguridad en SDN.

1. Otros casos de uso con TE en SDN
2. Investigación sobre nuevas Tec. TE.
3. Nuevas aplicaciones SDN
4. Seguridad (unico punto de fallo, el controlador).



¿Preguntas?

