

# Estudio de técnicas de Ingeniería de Tráfico basadas en SDN

## Study of SDN Traffic Engineering Techniques

---

Betegón García, Miguel<sup>1</sup>

`miguel.betegon@alumnos.unican.es`

Julio, 2018

<sup>1</sup>Grado en Ingeniería de Tecnologías de Telecomunicación



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

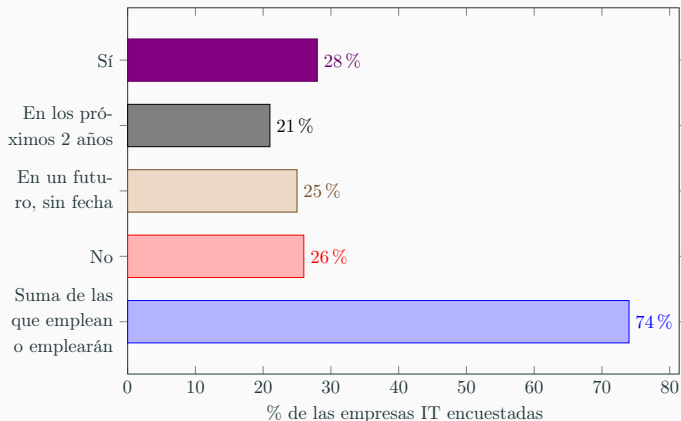
SDN se ha establecido como un producto conocido.

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



# Motivación y objetivos III

Adopción de las redes SDN en las empresas IT en 2016.



Fuente: Channel Insider Networking - Michael Vizard



# 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.



## Conceptos teóricos

---

# Redes Definidas por Software (SDN)

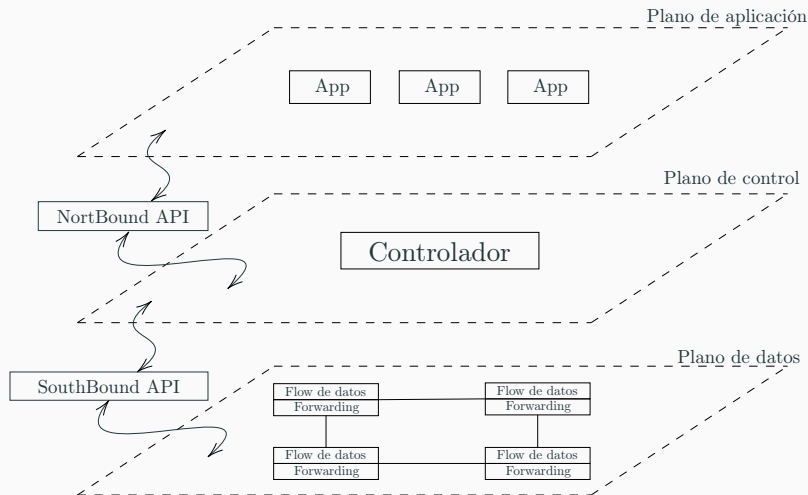


Figura 1: Arquitectura de alto nivel SDN.





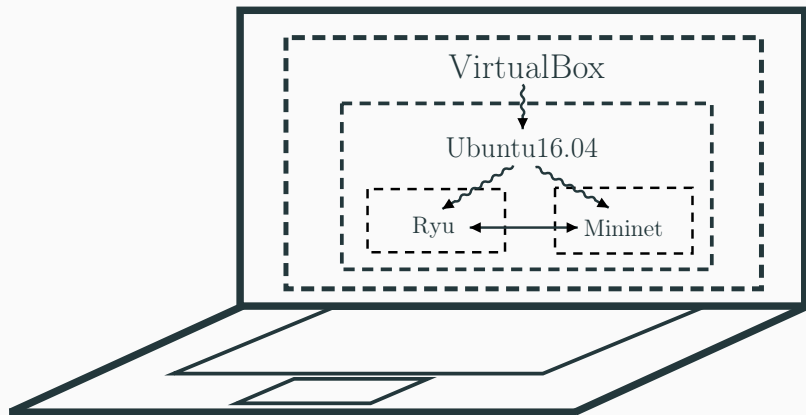


Figura 2: Esquema físico del proyecto.



## 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.



# Protocolo OpenFlow II

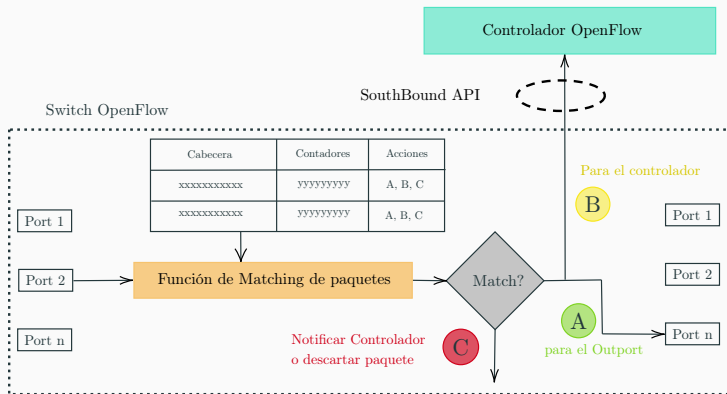


Figura 3: Switch OpenFlow. Operación básica.



## 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).



## 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.



- 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.



# 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".



## Routing multicamino con balanceador de carga

---



# Balanceador de carga con routing multicamino

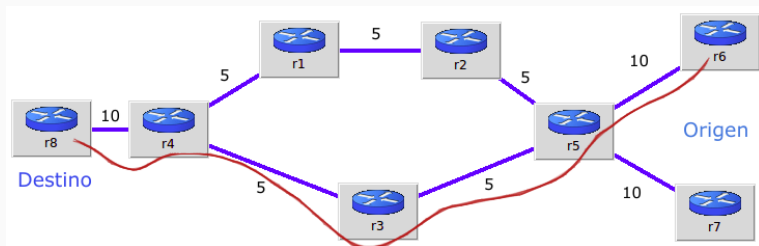


Figura 4: Problema del pez.

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`.



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).



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:  
 $O(N + E)$



## 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).



# Iteraciones DFS

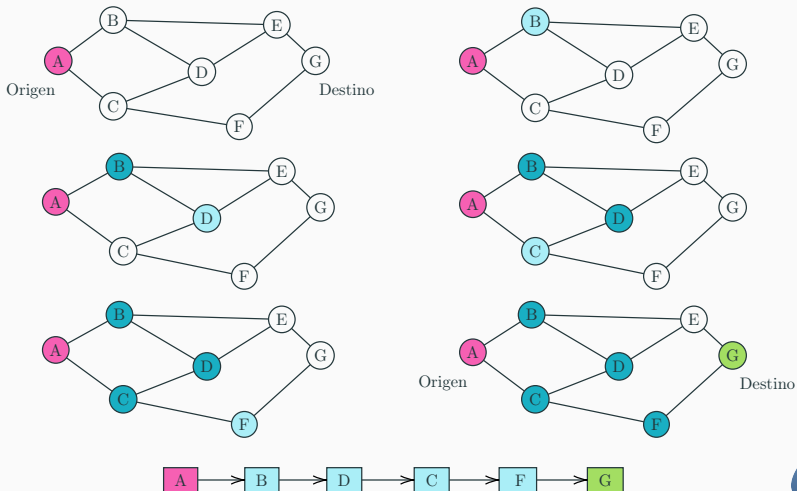


Figura 5: Iteraciones del algoritmo DFS.



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.

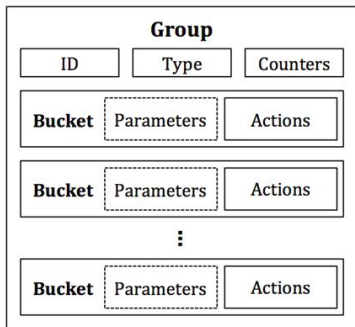
$$Cost(l) = \frac{BW_{Reference}}{BW(l)} \quad ;$$

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

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



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.



# 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**



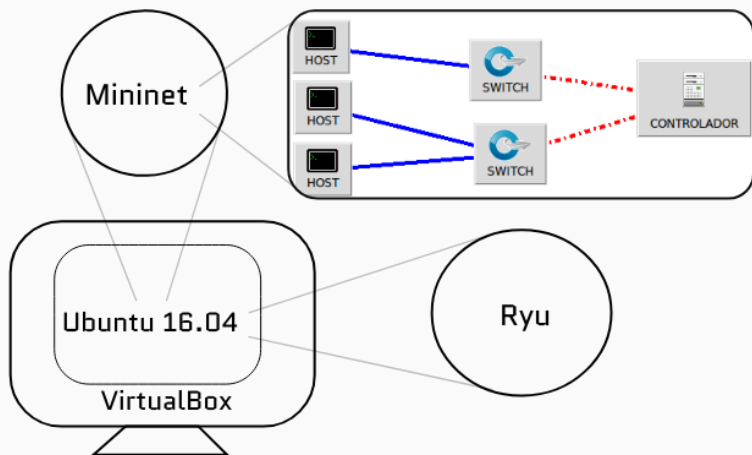
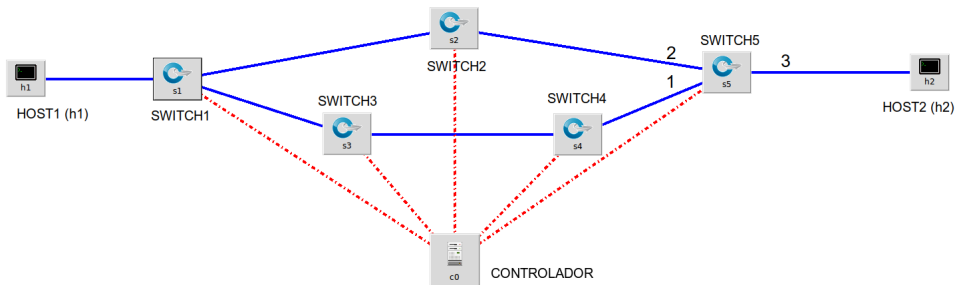


Figura 7: Entorno de desarrollo.



## Definición del escenario de aplicación



# 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
Camino disponible 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

Camino disponible 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
```

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

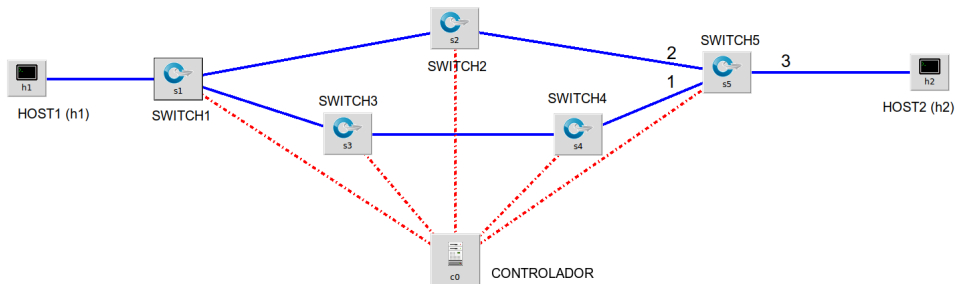
```
OF1.3 reply (OF1.3) (xid=0x2):
```

```
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
```



# 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
```

# Balanceador de carga multicamino V

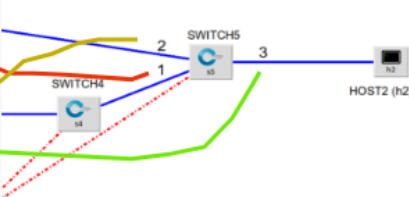
## Creación de tráfico TCP con iPerf.

```
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
```

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



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

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



# Monitorización del tráfico con cierta QoS I

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	$\infty$	2 Mbps	$\infty$	2 Mbps	$\infty$	2 Mbps

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



## 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



# Monitorización del tráfico con cierta QoS III

```
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"}}}}}]
```

# VÍDEO



## 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.



## 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.



¿Preguntas?

