

- Gestión de dispositivos IoT con acceso a la red mediante 802.11ax y SDN en el edge para orquestar los flujos de datos

Sergio Baza Alonso

UOC

Curso 2020/2021



1

INTRODUCCIÓN

1. Introducción

REDES DEFINIDAS POR SOFTWARE (SDN)

- Cambio de paradigma en el entorno de redes de comunicaciones.
- Red controlada por aplicaciones/programación (software).
- Beneficios a nivel de negocio, de operación y tecnológico.

ACCESO DE RED INALÁMBRICO (WiFi)

- Proporciona comodidad en el acceso a la red.
- Evolución del estándar hasta proporcionar rendimiento similar al del cableado tradicional.
- Reducción de costes en la capa de acceso a la red.

VIRTUALIZACIÓN DE FUNCIONES DE RED (NFV)

- Cambio de equipos físicos por equipos virtuales con la misma función.
- Incremento agilidad/flexibilidad en la red.
- Reducción de complejidad al implementar nuevos servicios.

DISPOSITIVOS IoT

- Cada vez mayor número de dispositivos necesitan conexión a internet.
- Gran variedad de entornos en los que encontrar estos dispositivos.
- Se busca la comunicación M2M.



2

OBJETIVOS

2. Objetivos

1. **Aplicación de SDN en un entorno empresarial de carácter industrial con un gran número de dispositivos IoT conectados a la red.**
 - a. Realización de un estudio de la situación tecnológica de la aplicación de SDN sobre una infraestructura de red inalámbrica (SD-WLAN).
 - b. Aplicación de NFV para sustituir los servicios que oferta una red inalámbrica tradicional por funciones de red virtualizadas.
2. **Realización de un estudio del nuevo estándar inalámbrico IEEE 802.11ax**
 - a. Comprobación de las ventajas que proporciona a dispositivos IoT.
 - b. Integrar el acceso mediante WiFi en una red SDN de la manera más adecuada.



● 2. Objetivos

3. Diseño de una arquitectura de red que permite orquestar los flujos de datos provenientes de una red de acceso inalámbrica.

4. Evaluación de la solución propuesta mediante una maqueta de red.



3

DESARROLLO DEL PROYECTO

● 3. Desarrollo del proyecto

○ Arquitectura de red WiFi empresarial tradicional

En una red WiFi tradicional existen los siguientes elementos de red:

- Equipo controlador WiFi (WLC)
 - Aprovechona puntos de acceso con configuración
 - Autentica usuarios para permitir el acceso a la red
 - Gestiona las comunicaciones entre los clientes conectados a la red WiFi
 - Gestiona la movilidad de los clientes de un AP a otro
- Puntos de acceso (APs)
 - Se conectan con la controladora WiFi
 - Reenvían el tráfico de los clientes a la controladora para su gestión

● 3. Desarrollo del proyecto

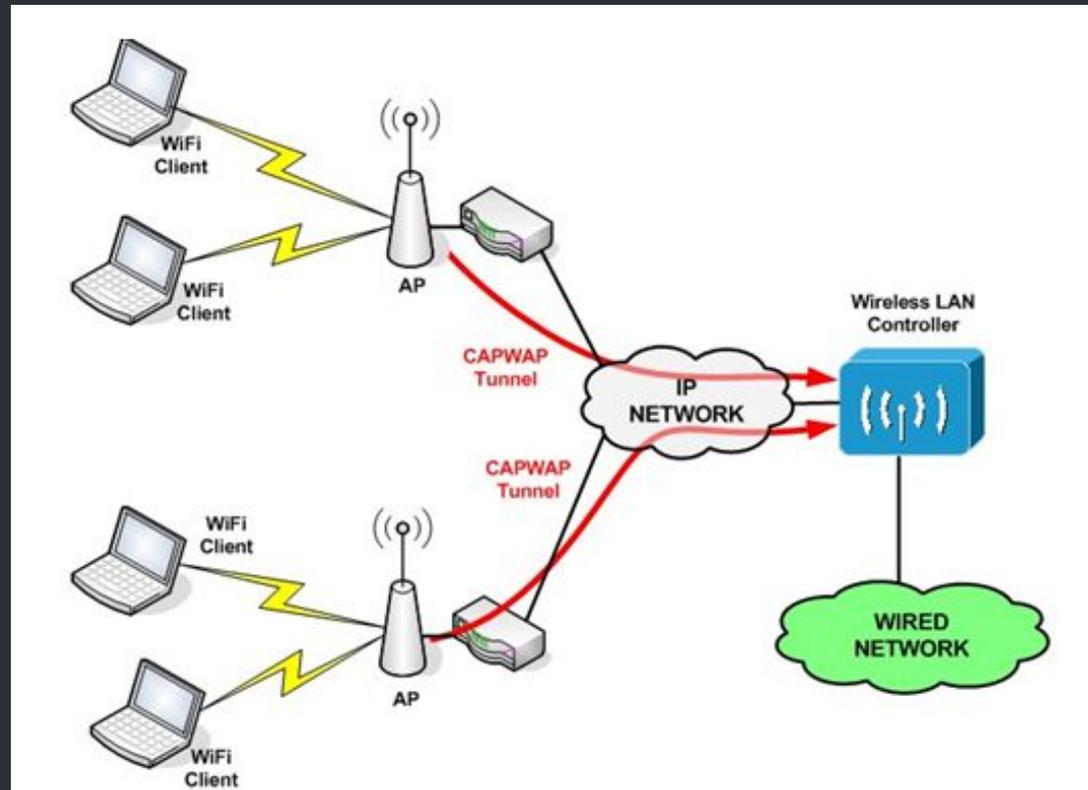
○ Arquitectura de red WiFi empresarial tradicional

En una red WiFi tradicional existen los siguientes elementos de red:

- Switches
 - Interconectan a nivel de red los APs con la WLC
- Clientes
 - Equipos que se conectan a la red a través de los APs

3. Desarrollo del proyecto

Arquitectura de red WiFi empresarial tradicional



● 3. Desarrollo del proyecto

○ Arquitectura de red WiFi empresarial tradicional → Arquitectura de red (SDN + NFV + WiFi)

1. Red IP tradicional → Red SDN
 - Instalación de un equipo controlador SDN.
2. Despliegue de VNFs
 - Equipo controlador WiFi
 - Servidor Radius
3. Instalación de infraestructura de virtualización donde desplegar las VNFs
4. Switches tradicionales → Switches integrables en red SDN
 - Comunicación con el controlador a través de Openflow
5. APs tradicionales → APs compatibles con Openflow
6. Aplicaciones para gestionar la red SDN

● 3. Desarrollo del proyecto

Arquitectura de red (SDN + NFV + WiFi)

- Red IP tradicional → Red SDN
 - Instalación de un controlador SDN (Ryu)
 - Se ejecuta como una aplicación de Python
 - Soporte de múltiples protocolos en sus interfaces:
 - Norte: REST
 - Sur: Openflow, Netconf, Netflow
 - Desarrollo de aplicaciones en Python
 - API bien definida



● 3. Desarrollo del proyecto

Arquitectura de red (SDN + NFV + WiFi)

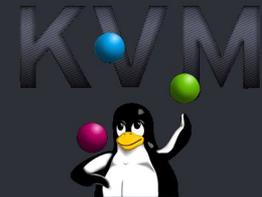
- Despliegue de VNFs
 - Equipo controlador WiFi - Mikrotik
 - Equipo de bajo coste
 - Permite múltiples opciones de configuración
 - Compatible con Openflow
 - Servidor Radius - Freeradius
 - Implementación opensource del servidor Radius
 - Compatibilidad con múltiples métodos de autenticación en una red inalámbrica

The Mikrotik logo features the word "Mikro" in a white, lowercase, sans-serif font, followed by "tik" in a bold, uppercase, sans-serif font. A white arc above the "i" in "Mikro" suggests a signal or antenna.The FreeRADIUS logo consists of the word "free" in a light blue, lowercase, cursive font, followed by "RADIUS" in a white, uppercase, sans-serif font.

● 3. Desarrollo del proyecto

Arquitectura de red (SDN + NFV + WiFi)

- Instalación de infraestructura de virtualización donde desplegar las VNFs
 - Software opensource - Linux KVM
- Switches tradicionales → Switches integrables en red SDN
 - Software opensource - Open vSwitch
 - Compatible con openflow
- APs tradicionales → APs compatibles con Openflow
 - Equipos físicos Mikrotik



● 3. Desarrollo del proyecto

○ Arquitectura de red (SDN + NFV + WiFi)

- Aplicaciones para gestionar la red SDN
 - Aplicaciones desarrolladas en Python que interactúan con el controlador SDN
 - Aplicación de gestión de equipos como switches
 - Permite interactuar con la tabla de MACs del equipo
 - Aplicación de gestión de equipos como firewalls
 - Permite interactuar con un switch como si se tratase de un firewall
 - Habilitar/Deshabilitar el firewall
 - Listar/introducir/eliminar reglas



4

ESCENARIO DE PRUEBAS

● 4. Escenario de pruebas

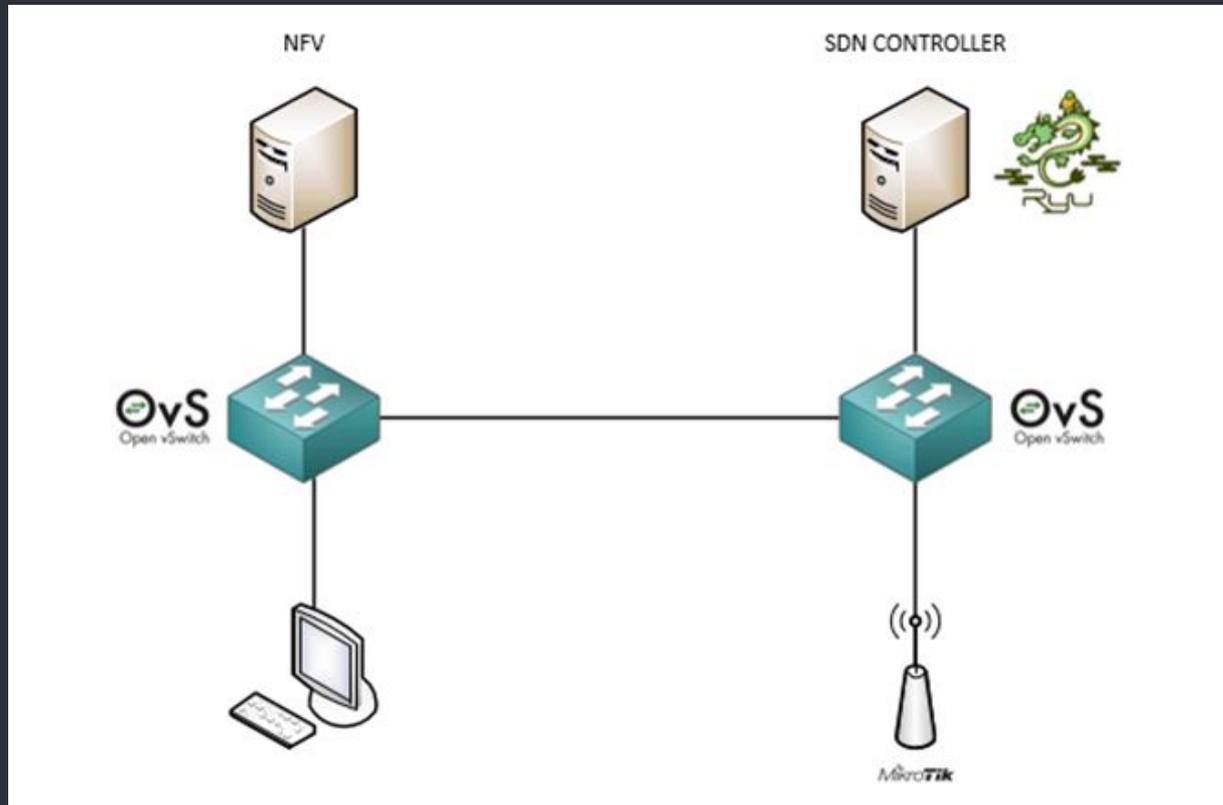
○ Arquitectura de red propuesta

La arquitectura de red propuesta está compuesta por:

- 1 servidor controlador SDN
- 1 servidor de virtualización de infraestructura
- 2 switches Open vSwitch
- 1 equipo virtual conectado por cable
- 1 AP físico

4. Escenario de pruebas

Arquitectura de red propuesta



● 4. Escenario de pruebas

○ Pruebas y métricas

- Se han definido 3 escenarios de pruebas:
 - Realizar el conexionado de dos dispositivos IoT por WiFi y comprobar la conectividad entre ambos mediante ICMP.
 - Realizar el conexionado de un dispositivo IoT por WiFi y comprobar la conectividad con el dispositivo conectado por cable mediante ICMP.
 - Realizar el conexionado de un dispositivo IoT por WiFi y uso del firewall para impedir la comunicación con el dispositivo conectado por cable. La comprobación de la conectividad se hace mediante ICMP.

4. Escenario de pruebas

Escenario de ejemplo - Firewall

Configuración de red para los dos dispositivos involucrados

```
PC1> show ip IP 192.168.58.196/24
```

```
NAME      : PC1[1]
IP/MASK   : 192.168.58.196/24
GATEWAY   : 0.0.0.0
DNS       :
DHCP SERVER : 192.168.58.250
DHCP LEASE  : 595, 600/300/525
MAC       : 00:50:79:66:68:00
LPORT     : 10027
RHOST:PORT : 127.0.0.1:10028
MTU       : 1500
```

Cableado

```
pi@raspberrypi: ~
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.58.197 | netmask 255.255.255.0 broadcast 192.168.58.255
    inet6 fe80::c915:ccd3:2344:e640 prefixlen 64 scopeid 0x20<link>
    ether b8:27:eb:a9:31:f1 txqueuelen 1000 (Ethernet)
    RX packets 383 bytes 74489 (72.7 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 137 bytes 21309 (20.8 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

WiFi

4. Escenario de pruebas

Escenario de ejemplo - Firewall

Ejecutar el controlador con la aplicación de firewall

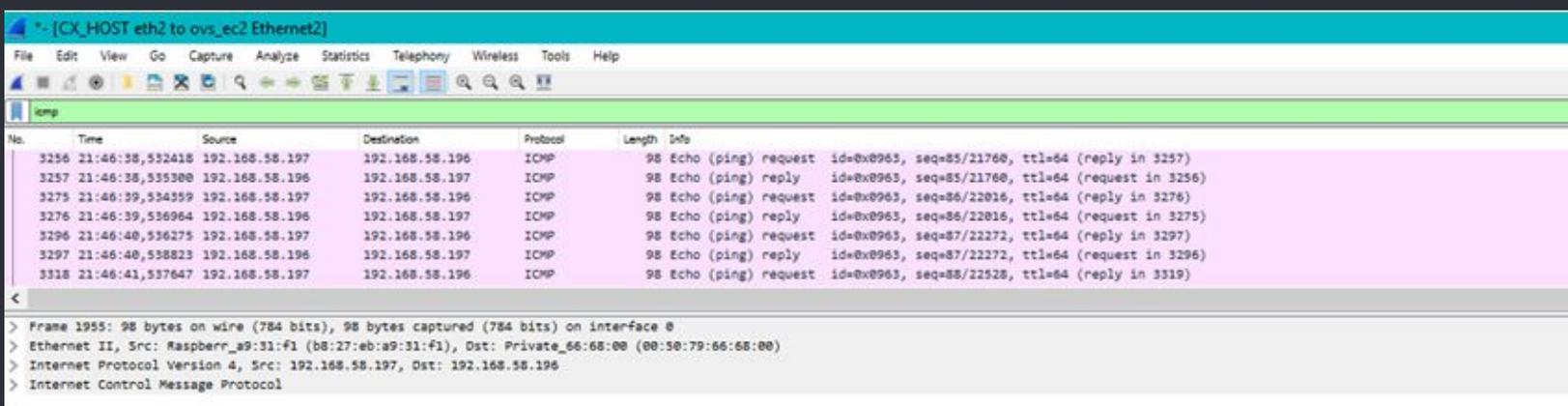
```
root@SDN:/usr/local/lib/python3.6/dist-packages/ryu/app# ryu run rest_firewall.py
loading app rest_firewall.py
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app rest_firewall.py of RestFirewallAPI
instantiating app ryu.controller.ofp_handler of OFPHandler
(1497) wsgi starting up on http://0.0.0.0:8080
[FW][INFO] dpid=00000800270055dd: Join as firewall.
[FW][INFO] dpid=00000800270b6287: Join as firewall.
```

4. Escenario de pruebas

Escenario de ejemplo - Firewall

Comprobación de conectividad entre los dos dispositivos

```
root@raspberrypi:/home/pi# ping 192.168.58.196
PING 192.168.58.196 (192.168.58.196) 56(84) bytes of data.
64 bytes from 192.168.58.196: icmp_seq=1 ttl=64 time=9.53 ms
64 bytes from 192.168.58.196: icmp_seq=2 ttl=64 time=3.99 ms
64 bytes from 192.168.58.196: icmp_seq=3 ttl=64 time=4.30 ms
64 bytes from 192.168.58.196: icmp_seq=4 ttl=64 time=4.75 ms
64 bytes from 192.168.58.196: icmp_seq=5 ttl=64 time=4.33 ms
```



The screenshot shows a Wireshark capture window titled "[CX_HOST eth2 to ovs_ec2 Ethernet2]". The interface includes a menu bar (File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help) and a toolbar. The main display area shows a list of captured packets, with the first six packets selected. The packet list table is as follows:

| No. | Time | Source | Destination | Protocol | Length | Info |
|------|-----------------|----------------|----------------|----------|--------|---|
| 3256 | 21:46:38,532418 | 192.168.58.197 | 192.168.58.196 | ICMP | 98 | Echo (ping) request id=0x0965, seq=85/21760, ttl=64 (reply in 3257) |
| 3257 | 21:46:38,535300 | 192.168.58.196 | 192.168.58.197 | ICMP | 98 | Echo (ping) reply id=0x0963, seq=85/21760, ttl=64 (request in 3256) |
| 3275 | 21:46:39,534359 | 192.168.58.197 | 192.168.58.196 | ICMP | 98 | Echo (ping) request id=0x0963, seq=86/22016, ttl=64 (reply in 3276) |
| 3276 | 21:46:39,536964 | 192.168.58.196 | 192.168.58.197 | ICMP | 98 | Echo (ping) reply id=0x0963, seq=86/22016, ttl=64 (request in 3275) |
| 3296 | 21:46:40,536275 | 192.168.58.197 | 192.168.58.196 | ICMP | 98 | Echo (ping) request id=0x0965, seq=87/22272, ttl=64 (reply in 3297) |
| 3297 | 21:46:40,538823 | 192.168.58.196 | 192.168.58.197 | ICMP | 98 | Echo (ping) reply id=0x0963, seq=87/22272, ttl=64 (request in 3296) |
| 3318 | 21:46:41,537647 | 192.168.58.197 | 192.168.58.196 | ICMP | 98 | Echo (ping) request id=0x0963, seq=88/22528, ttl=64 (reply in 3319) |

Below the packet list, the details pane shows the structure of the selected packet (No. 1955):

- > Frame 1955: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface e
- > Ethernet II, Src: Raspberr_a9:31:f1 (b8:27:eb:a9:31:f1), Dst: Private_66:68:00 (00:50:79:66:68:00)
- > Internet Protocol Version 4, Src: 192.168.58.197, Dst: 192.168.58.196
- > Internet Control Message Protocol

4. Escenario de pruebas

Escenario de ejemplo - Firewall

Introducción y comprobación de reglas en los equipos mediante la aplicación

```
root@SDN: /home/sergio
Gestion del firewall
 1 - Ver estado del Firewall en los switches
 2 - Habilitar Firewall en switch
 3 - Deshabilitar Firewall en switch
 4 - Obtener las reglas de un switch
 5 - Introducir una nueva regla en un switch
 6 - Borrar una regla de un switch
 7 - Configurar IP del controlador
 8 - Salir
```

```
Elige una opcion: 5
Introduce el id del switch: 00000800270b6287
Introduce la red origen: 192.168.58.197/32
Introduce la red destino: 192.168.58.196/32
Introduce el protocolo: ICMP
Introduce la accion: DENY
Introduce la prioridad: 30
Pulsar enter para continuar
```

```
root@SDN: /home/sergio
Gestion del firewall
 1 - Ver estado del Firewall en los switches
 2 - Habilitar Firewall en switch
 3 - Deshabilitar Firewall en switch
 4 - Obtener las reglas de un switch
 5 - Introducir una nueva regla en un switch
 6 - Borrar una regla de un switch
 7 - Configurar IP del controlador
 8 - Salir
```

```
Elige una opcion: 5
Introduce el id del switch: 00000800270b6287
Introduce la red origen: 192.168.58.196/32
Introduce la red destino: 192.168.58.197/32
Introduce el protocolo: ICMP
Introduce la accion: DENY
Introduce la prioridad: 30
Pulsar enter para continuar
```

```
root@SDN: /home/sergio
```

- 4 - Obtener las reglas de un switch
- 5 - Introducir una nueva regla en un switch
- 6 - Borrar una regla de un switch
- 7 - Configurar IP del controlador
- 8 - Salir

```
Elige una opcion: 4
Introduce el id del switch: 00000800270b6287
```

```
[
 {
  "switch_id": "00000800270b6287",
  "access_control_list": [
    {
      "rules": [
        {
          "rule_id": 1,
          "priority": 20,
          "dl_type": "IPv4",
          "actions": "ALLOW"
        },
        {
          "rule_id": 9,
          "priority": 30,
          "dl_type": "IPv4",
          "nw_src": "192.168.58.197",
          "nw_dst": "192.168.58.196",
          "nw_proto": "ICMP",
          "actions": "DENY"
        },
        {
          "rule_id": 10,
          "priority": 30,
          "dl_type": "IPv4",
          "nw_src": "192.168.58.196",
          "nw_dst": "192.168.58.197",
          "nw_proto": "ICMP",
          "actions": "DENY"
        }
      ]
    }
  ]
}
```

4. Escenario de pruebas

Escenario de ejemplo - Firewall

Introducción y comprobación de reglas en los equipos mediante la aplicación

| No. | Time | Source | Destination | Protocol | Length | Info |
|------|---------------|----------------|----------------|----------|--------|----------------------|
| 1391 | 21:46:46,3... | 192.168.58.253 | 192.168.58.239 | OpenFlow | 178 | Type: OFPT_FLOW_MOD |
| 1394 | 21:46:46,5... | 192.168.58.239 | 192.168.58.253 | OpenFlow | 206 | Type: OFPT_PACKET_IN |

> Frame 1391: 178 bytes on wire (1424 bits), 178 bytes captured (1424 bits) on interface 0
> Ethernet II, Src: PcsCompu_e1:ed:3f (08:00:27:e1:ed:3f), Dst: f6:5f:de:a6:06:04 (f6:5f:de:a6:06:04)
> Internet Protocol Version 4, Src: 192.168.58.253, Dst: 192.168.58.239
> Transmission Control Protocol, Src Port: 6653, Dst Port: 51726, Seq: 577, Ack: 15901, Len: 112
v OpenFlow 1.3
 Version: 1.3 (0x04)
 Type: OFPT_FLOW_MOD (14)
 Length: 112
 Transaction ID: 492588471
 Cookie: 0x0000000000000009
 Cookie mask: 0x0000000000000000
 Table ID: 0
 Command: OFPFC_ADD (0)
 Idle timeout: 0
 Hard timeout: 0
 Priority: 30
 Buffer ID: OFP_NO_BUFFER (4294967295)
 Out port: OFPP_ANY (4294967295)
 Out group: OFPG_ANY (4294967295)
 Flags: 0x0000
 Pad: 0000
 Match
 v Instruction
 Type: OFPIT_APPLY_ACTIONS (4)
 Length: 24
 Pad: 00000000
 v Action
 Type: OFPAT_OUTPUT (0)
 Length: 16
 Port: OFPP_CONTROLLER (4294967293)
 Max length: 128
 Pad: 000000000000

4. Escenario de pruebas

Escenario de ejemplo - Firewall

Comprobación de los flujos instalados en los switches:

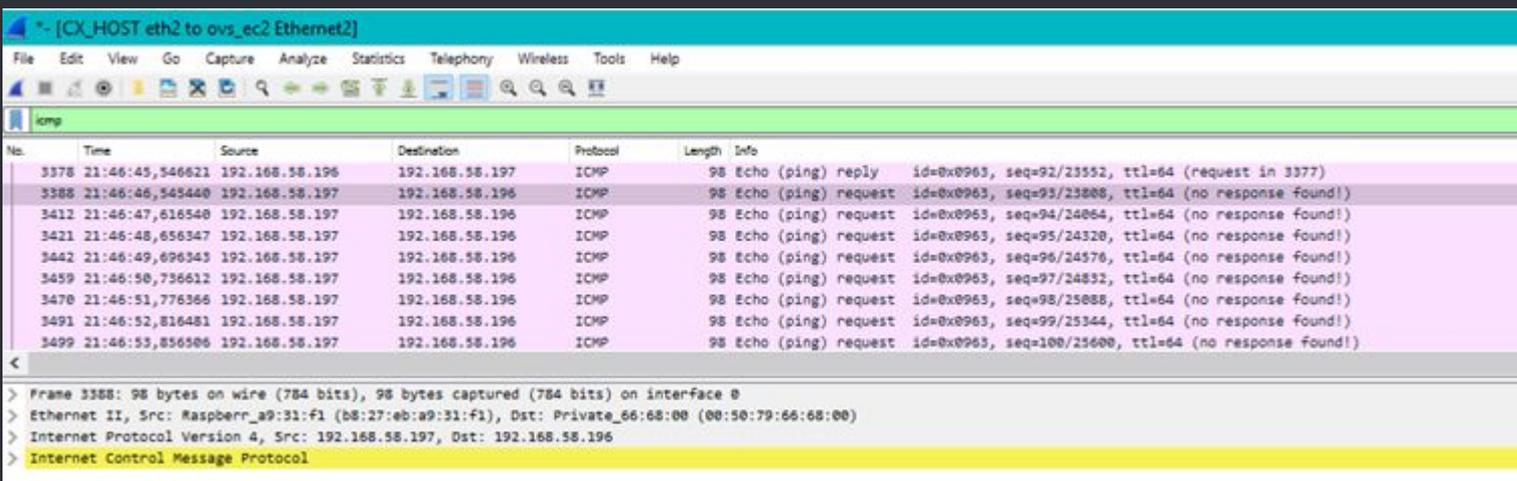
```
root@ovs_ec2:/home/sergio# ovs-ofctl dump-flows sw0
cookie=0x0, duration=4684.313s, table=0, n_packets=1349, n_bytes=80898, priority=65534,arp actions=NORMAL
cookie=0x1, duration=4487.681s, table=0, n_packets=52067, n_bytes=16451308, priority=20,in actions=NORMAL
cookie=0xb, duration=98.043s, table=0, n_packets=0, n_bytes=0, priority=30,icmp,nw_src=192.168.58.197,nw_dst=192.168.58.196 actions=CONTROLLER:128
cookie=0xc, duration=73.690s, table=0, n_packets=0, n_bytes=0, priority=30,icmp,nw_src=192.168.58.196,nw_dst=192.168.58.197 actions=CONTROLLER:128
cookie=0x0, duration=4684.314s, table=0, n_packets=3330, n_bytes=453180, priority=0 actions=CONTROLLER:128
```

4. Escenario de pruebas

Escenario de ejemplo - Firewall

Comprobación de la conectividad tras haber insertado las reglas

```
root@raspberrypi:/home/pi# ping 192.168.58.196
PING 192.168.58.196 (192.168.58.196) 56(84) bytes of data.
^C
--- 192.168.58.196 ping statistics ---
11 packets transmitted, 0 received, 100% packet loss, time 410ms
```



The screenshot shows a Wireshark capture of ICMP traffic on the interface eth2. The capture shows a series of ping requests and replies between 192.168.58.196 and 192.168.58.196. The traffic is filtered by 'icmp'. The packet list shows 11 packets, all of which are ping requests (seq=92 to seq=100) that did not receive a response. The packet details pane for frame 3388 shows the following structure:

- Frame 3388: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
- Ethernet II, Src: Raspberr_a9:31:f1 (b8:27:eb:a9:31:f1), Dst: Private_66:68:00 (00:50:79:66:68:00)
- Internet Protocol Version 4, Src: 192.168.58.197, Dst: 192.168.58.196
- Internet Control Message Protocol

4. Escenario de pruebas

Escenario de ejemplo - Firewall

Comprobación de las acciones de los switches ante el tráfico

The screenshot shows a Wireshark capture of network traffic on the interface [SDN-1 Ethernet0 to ovs_ec2 Ethernet1]. The capture filter is 'openflow_v4'. The main display area shows a list of captured packets, with the selected packet (No. 1394) expanded to show its details.

| No. | Time | Source | Destination | Protocol | Length | Info |
|------|-----------------|----------------|----------------|----------|--------|-------------------------|
| 1391 | 21:46:46,357668 | 192.168.58.253 | 192.168.58.239 | OpenFlow | 178 | Type: OFPT_FLOW_MOD |
| 1394 | 21:46:46,504579 | 192.168.58.239 | 192.168.58.253 | OpenFlow | 206 | Type: OFPT_PACKET_IN |
| 1396 | 21:46:46,902622 | 192.168.58.239 | 192.168.58.253 | OpenFlow | 168 | Type: OFPT_PACKET_IN |
| 1398 | 21:46:46,937377 | 192.168.58.240 | 192.168.58.253 | OpenFlow | 168 | Type: OFPT_PACKET_IN |
| 1403 | 21:46:47,414808 | 192.168.58.240 | 192.168.58.253 | OpenFlow | 74 | Type: OFPT_ECHO_REQUEST |
| 1405 | 21:46:47,415721 | 192.168.58.253 | 192.168.58.240 | OpenFlow | 74 | Type: OFPT_ECHO_REPLY |
| 1407 | 21:46:47,575481 | 192.168.58.239 | 192.168.58.253 | OpenFlow | 206 | Type: OFPT_PACKET_IN |
| 1410 | 21:46:48,615142 | 192.168.58.239 | 192.168.58.253 | OpenFlow | 206 | Type: OFPT_PACKET_IN |
| 1412 | 21:46:48,898172 | 192.168.58.239 | 192.168.58.253 | OpenFlow | 168 | Type: OFPT_PACKET_IN |

Packet 1394 details:

- Frame 1394: 206 bytes on wire (1648 bits), 206 bytes captured (1648 bits) on interface 0
- Ethernet II, Src: f6:5f:de:a6:06:04 (f6:5f:de:a6:06:04), Dst: PcsCompu_e1:ed:3f (08:00:27:e1:ed:3f)
- Internet Protocol Version 4, Src: 192.168.58.239, Dst: 192.168.58.253
- Transmission Control Protocol, Src Port: 51726, Dst Port: 6653, Seq: 15901, Ack: 689, Len: 140
- OpenFlow 1.3
 - Version: 1.3 (0x04)
 - Type: OFPT_PACKET_IN (10)
 - Length: 140
 - Transaction ID: 0
 - Buffer ID: OFF_NO_BUFFER (4294967295)
 - Total length: 98
 - Reason: OFFR_ACTION (1)
 - Table ID: 0
 - Cookie: 0x0000000000000000
 - Match
 - Pad: 0000
 - Data
 - Ethernet II, Src: Raspberr_e8:31:f1 (b8:27:eb:a8:31:f1), Dst: Private_66:68:00 (08:50:79:66:68:00)
 - Internet Protocol Version 4, Src: 192.168.58.197, Dst: 192.168.58.196
 - Internet Control Message Protocol

● 4. Escenario de pruebas

○ Escenario de ejemplo - Firewall

Comprobación de las acciones del controlador ante el tráfico

```
[FW][INFO] dpid=00000800270b6287: Blocked packet = ethernet(dst='00:50:79:66:68:00', ethertype=2048, src='b8:27:eb:a9:31:f1'), ipv4(csum=21550, dst='192.168.58.196', flags=2, header_length=5, identification=61344, offset=0, option=None, proto=1, src='192.168.58.197', tos=0, total_length=84, ttl=64, version=4), icmp(code=0, csum=38465, data=echo(data=b'\xc7\x06\xd5_\xc3\x8d\x0c\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&\'()*+,-./01234567', id=2745, seq=14), type=8)
```



5

CONCLUSIONES

● 5. Conclusiones

○ Se ha integrado SDN y NFV en una red con tecnología de acceso inalámbrica.

○ Se ha profundizado en las ventajas del nuevo estándar inalámbrico IEEE 802.11ax para los dispositivos IoT

○ Se han realizado con éxito pruebas sobre una maqueta de red que integra SDN, NFV y acceso por WiFi para dar servicio a dispositivos IoT

Gracias por la atención

Preguntas

sbaza@uoc.edu

[linkedin.com/in/sergiobazaalonso/](https://www.linkedin.com/in/sergiobazaalonso/)

- Gestión de dispositivos IoT con acceso a la red mediante 802.11ax y SDN en el edge para orquestar los flujos de datos

Sergio Baza Alonso

UOC

Curso 2020/2021