



Memoria P4

David Carrascal

Junio del 2019

Universidad de Alcalá

Índice general

| | |
|--|-----------|
| 1. Lenguaje P4 | 5 |
| 1.1. Enlaces útiles | 5 |
| 1.2. Papers de interés | 5 |
| 1.3. Introducción teórica | 5 |
| 1.3.1. ¿Qué es P4? | 5 |
| 1.4. Instalación | 6 |
| 1.5. Test P4 | 7 |
| 1.6. Test 1: Implementando el forwarding básico | 7 |
| 1.6.1. Comprobar el modelo suministrado | 7 |
| 1.6.2. Desarrollo del forwarding L3 | 8 |
| 1.7. Test 2: Implementando el tunelado básico | 12 |
| 1.7.1. Crear protocolo de tunelado | 12 |
| 1.7.2. Parte Avanzada Tunelado | 15 |
| 1.8. Test 3: Implementación del plano de control vía P4Runtime | 20 |
| 1.8.1. Escenario previo | 20 |
| Anexos | 22 |
| A. Vagrant | 25 |
| A.1. ¿Qué es? | 25 |
| A.2. Instalación | 25 |
| A.3. ¿Para que sirve? | 26 |
| A.4. ¿Cómo funciona? | 26 |
| A.5. Comandos útiles | 26 |
| B. Generar grafos desde programas p4 | 27 |
| B.1. p4c-graphs | 27 |
| B.2. Herramienta auto_graph.sh | 27 |
| B.3. Ejemplo | 28 |

Capítulo 1

Lenguaje P4

1.1. Enlaces útiles

- Tutoriales: <https://github.com/p4lang/tutorials/>
- Protobuf: <https://developers.google.com/protocol-buffers/>
- gRPC: <https://grpc.io/>
- gRPC ejemplo: <https://www.nocountryforgeeks.com/grpc-protocol-buffers/>
- Lista de reproducción sobre conferencias:
https://www.youtube.com/playlist?list=PLRffjASqEa5VMsUml2V34_xDTeVydeM_L
- Correo explicando BM2: http://lists.p4.org/pipermail/p4-dev_lists.p4.org/2016-June/002112.html

1.2. Papers de interés

- Paper sobre la agregación de tecnología p4 para parsear paquetes generados por sensores IoT a una WAN [3].
- Paper sobre como emplear Switches p4 frontera para servicios de IoT [4]
- Paper sobre el gran potencial que tiene p4 para interconectar el mundo SDN con la próxima generación de nodos finales que interactúan con clusters de TI innovadores, 5G fronthaul e internet de las cosas. [1].
- Paper que describe y mide el rendimiento de una plataforma que implementa un programa P4 en OVS [2].

1.3. Introducción teórica

En este punto se va a tratar de abordar todos los conceptos teóricos que nos vayan surgiendo según exploremos p4. Se intentará tratar de entender todas las tecnologías que emplea p4, cual es su uso, cuales son sus pros y sus posibles contras.

1.3.1. ¿Qué es P4?

1.4. Instalación

- Hemos instalado una máquina virtual de Ubuntu 16.04, con 20GB de almacenamiento ya que con 10 se nos queda corto. 4 GiB de ram y cuatro núcleos
- Hemos hecho un update y upgrade, e instalado git, vim, htop, openssh-server para conectarse vía ssh en remoto.
- Hemos hecho un git clone del repositorio con el tutorial: <https://github.com/p4lang/tutorials>
- Lanzamos los dos shell-scripts user-bootstrap.sh y root-bootstrap.sh.
- Hemos encontrado varios problemas en el momento de llevar a cabo la instalación. Por lo que se ha decidido modificar los shells-scripts de instalación ofrecidos en el repositorio.

Como se ha introducido antes, hemos tenido que crear nuestro propio script de instalación **install.sh**. Habrá que darle permisos de ejecución **chmod 777 install.sh** y lanzarlo con **sudo**.

La necesidad de crear otro método de instalación es la deficiencia de los métodos alternativos de instalación que tienen aparte de Vagrant. Ellos disponen de dos shell-scripts, donde crean un environment muy cercano a personas con poco conocimiento de Linux, facilitándoles user propio p4, generar iconos en el escritorio, configurar IDEs... Tareas propias de cada desarrollador en función de sus preferencias.

Por lo que se ha decidido limpiar el proceso de instalación, dejando únicamente lo estrictamente necesario para llevar a cabo el tutorial de p4. Dejando únicamente un shell-script, más optimizado y más limpio. Se ha tenido que hacer una doble llamada al shell-script **autogen.sh** debido a este issue:

- <https://github.com/protocolbuffers/protobuf/issues/149>

Como forma de agradecer a la organización de p4 el hecho de que tengan un buen repositorio de tutoriales dando un buen endpoint para los noobies que acaban de empezar con p4 (Yo) se ha ofrecido el script de instalación para que se incorpore al master.

Al hacer el pull-request, el mantenedor de los tutoriales le pareció una buena idea, pero el hecho de tener un método de instalación nuevo significa más trabajo para mantener el repositorio. Me pidió que tratara de incluir la instalación nativa de Vagrant para que así con un único script podamos tener la instalación sobre Vagrant y el método que habíamos propuesto. Para poder llevar esta tarea a cabo se tuvo que estudiar la herramienta de Vagrant. En los anexos se han puesto unas pinceladas de los aspectos más significativos de esta herramienta.

De momento no está incluido en el master pero se puede hacer de uso del nuevo método de instalación el fork creado de los tutoriales de p4.

- <https://github.com/davidcawork/testP4/tree/master/tutorial>

Se puede consultar el estado del pull-request aquí.

- <https://github.com/p4lang/tutorials/pull/261>

Para llevar a cabo la instalación del modo nativo sobre Vagrant se debe instalar Vagrant, instalar un proveedor de máquinas virtuales y cargar el VagrantFile. Para más información sobre el uso de Vagrant consulte la información referente a ello en los anexos.

1.5. Test P4

En esta sección vamos a seguir el tutorial y los test que proponen la organización de p4 en su repositorio oficial.

- <https://github.com/p4lang/tutorials>

La metodología de los tutoriales consiste en completar el esqueleto del código p4 dado por ellos para hacer funcionar las distintas pruebas. El escenario que se ha empleado para establecer un eviroment de pruebas es Mininet.

1.6. Test 1: Implementando el forwarding básico

El objetivo de este test es escribir un programa P4 que implemente el forwarding básico. Con el reenvío de IPv4, el conmutador debe realizar las siguientes acciones para cada paquete:

- Actualizar las direcciones MAC de origen y destino.
- Disminuir el campo de la cabecera IP (TTL).
- Reenviar el empaque al puerto apropiado.

Nuestro router tendrá una sola tabla, que el plano de control llenará con reglas estáticas. Cada regla asignará una dirección IP a la dirección MAC y al puerto de salida para el próximo salto. Ya se han definido las reglas del plano de control, por lo que solo necesitamos implementar la lógica del plano de datos de nuestro programa P4 (*basic.p4*).

1.6.1. Comprobar el modelo suministrado

En este punto siguiendo el tutorial, tenemos que comprobar como efectivamente el modelo suministrado no es capaz de establecer comunicaciones entre nodos finales. Esto se debe a que el plano de datos de los routers está incompleto, y será nuestra misión la de completar el plano de datos con nuestro programa en p4 que dotará de la capacidad de forwarding de paquetes a los routers.

Procedimiento a seguir para llevar a cabo el test:

- Hacemos uso del Makefile que trae el tutorial: **make run**, este target del Makefile automatizará las siguientes tareas:
 - Compilará el archivo *basic.p4* para el behavioral model 2 (bm2) el target simple switch
 - Lanzará Mininet con una topología de tres router conectados triangularmente y a cada router se conectará un host. Los host tendrán respectivamente las IPs 10.0.1.1, 10.0.2.2, 10.0.3.3.
- En el mismo directorio se encuentran dos herramientas escritas en Python, servidor y cliente, que nos ayudaran a generar tráfico desde un host a otro. Estas herramientas hacen uso de Scapy para poder visualizar el contenido el paquete generado. Herramientas *receive.py* y *send.py*.
- Levantaremos dos terminales por ejemplo, host 1 y host 2: **xterm h1 h2**
- Ejecutaremos las herramientas para generar tráfico entre ambos host.
- El mensaje no llegará, salimos de Mininet con **exit**, y limpiamos el escenario y Mininet con **make stop** (Este target llamará a **sudo mn -c** por nosotros).

Un programa P4 define una pipe-line de procesamiento de paquetes, pero las reglas dentro de cada tabla se insertan desde el plano de control. Cuando una regla coincide con un paquete (Hay un hit), su acción se invoca con los parámetros proporcionados por el plano de control como parte de la regla.

En este test, ya se ha implementado la lógica del plano de control, está suministrado por el equipo de p4. A la hora de levantar la instancia de Mininet, el comando **make run** instalará las reglas de procesamiento

```

- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.2.2', 32] => MyIngress.ipv4_f
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.3.3', 32] => MyIngress.ipv4_f
Configuring switch s1 using P4Runtime with file s1-runtime.json
- Using P4Info file build/basic.p4info.txt...
- Connecting to P4Runtime server on 127.0.0.1:50051 (bmw2)...
- Setting pipeline config (build/basic.json)...
- Inserting 4 table entries...
- MyIngress.ipv4_lpm: (default action) => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.1.1', 32] => MyIngress.ipv4_f
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.2.2', 32] => MyIngress.ipv4_f
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.3.3', 32] => MyIngress.ipv4_f

s1 -> gRPC port: 50051
s2 -> gRPC port: 50052
s3 -> gRPC port: 50053
*****
h1
default interface: h1-eth0      10.0.1.1      00:00:00:00:01:01
*****
h2
default interface: h2-eth0      10.0.2.2      00:00:00:00:02:02
*****
h3
default interface: h3-eth0      10.0.3.3      00:00:00:00:03:03
*****
Starting mininet CLI

Welcome to the BMW2 Mininet CLI!

Your P4 program is installed into the BMW2 software switch
and your initial runtime configuration is loaded. You can interact
with the network using the mininet CLI below.

To view a switch log, run this command from your host OS:
    tail -f /home/n0obie/repo_p4/tutorials/exercises/basic/logs/<switchname>.l

To view the switch output pcap, check the pcap files in /home/n0obie/repo_p4
for example run: sudo tcpdump -xxx -r s1-eth1.pcap

To view the P4Runtime requests sent to the switch, check the
corresponding txt file in /home/n0obie/repo_p4/tutorials/exercises/basic/log
for example run: cat /home/n0obie/repo_p4/tutorials/exercises/basic/logs/s

mininet> xterm h1 h2

```

```

root@n0obie-VirtualBox:~/repo_p4/tutorials/exercises/basic# ./receive.py
WARNING: No route found for IPv6 destination :: (no default route?)
sniffing on h2-eth0

```

```

id      = 1
flags   =
frag    = 0L
ttl     = 64
proto   = tcp
chksum  = 0x63c6
src     = 10.0.1.1
dst     = 10.0.2.2
\options
###[ TCP ]###
sport   = 65503
dport   = 1234
seq     = 0
ack     = 0
dataofs = 5L
reserved = 0L
flags   = S
window  = 8192
chksum  = 0x3319
urgptr  = 0
options = []
###[ Raw ]###
load    = 'P4 mola'

```

Figura 1.1: Test Mininet: No llega el mensaje.

de paquetes en las tablas de cada router. Estas se definen en los archivos sX-runtime.json, donde X corresponde al número de router.

Se hace uso de P4Runtime para instalar las reglas del plano de control. El contenido de los archivos sX-runtime.json se refiere a nombres específicos de tablas, claves y acciones, tal como se define en el archivo P4Info producido por el compilador (busque el archivo build / basic.p4info después de ejecutar make run). Cualquier cambio en el programa P4 que agregue o cambie el nombre de tablas, claves o acciones deberá reflejarse en estos archivos sX-runtime.json.

1.6.2. Desarrollo del forwarding L3

El archivo basic.p4 contiene un programa P4 esqueleto con piezas lógicas clave donde deberemos completar su cuerpo para el correcto funcionamiento del router. Las partes que debemos completar son las siguientes:

- Completar el Parser para extraer las cabeceras de ethernet e ipv4.
- Completar un action llamado forward_ipv4 que deberá:
 - Establecer el puerto de salida del paquete.
 - Establecer como dirección MAC origen la MAC destino de la trama recibida.
 - Establecer como dirección MAC destino del paquete la MAC del siguiente salto.
 - Decrementar el campo TTL de la cabecera ipv4.
- Completar el campo apply para decidir si aplicar la tabla.

Para completar el parser de entrada del router hemos definido tres estados, el primer estado, entry-point, llamado start. El segundo parse.ethernet, para extraer la cabecera de Ethernet, y decidir si si entramos a la ultima fase del parser, parse_ipv4, en función del campo etherType. La ultima fase del parser únicamente extrae la cabecera de Ipv4.

```

/*****
***** P A R S E R *****/
*****/

parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {
        /* TODO: add parser logic */
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType){
            0x800: parse_ipv4;
            default: accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition accept;
    }
}

```

Figura 1.2: Parser.

Hemos definido un action que hacer forwarding a los paquetes que les llega a los router, deberá especificar un puerto de salida, modificar las MAC para el siguiente hop, y decrementar en uno el campo TTL de la cabecera Ipv4.

```

action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
    /* TODO: fill out code in action body */

    /* Establecer puerto de salida del paquete */
    standard_metadata.egress_spec = port;

    /* Establecer la dirMAC origen la MAC destino de la trama recibida */
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;

    /* Establecer la dirMAC destino la dir MAC del proximo salto */
    hdr.ethernet.dstAddr = dstAddr;

    /* Decrementar el valro del ttl */
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1 ;
}

```

Figura 1.3: Action forwarding.

Según los requerimientos dados, debemos comprobar con anterioridad si existe y es valida la cabecera Ipv4.

```

table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = NoAction();
}

apply {
    /* TODO: fix ingress control logic
     * - ipv4_lpm should be applied only when IPv4 header is valid
     */
    if(hdr.ipv4.isValid()){
        ipv4_lpm.apply();
    }
}

```

Figura 1.4: Tabla match-action de nuestro router.

Por último, únicamente debemos especificar en el deparser como queremos serializar las cabeceras, y en que orden. A continuación, se incluirá la carga útil y se conformará el paquete. Por carga útil entendemos toda información no procesada por el parser de entrada por el switch p4.

```

/*****
*****  D E P A R S E R  *****/
*****/

control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        /* TODO: add deparser logic */
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
    }
}

```

Figura 1.5: Deparser de nuestro router.

A continuación, se puede apreciar en la arquitectura conformada en bloques de nuestro switch. Donde, cada bloque va especificando la funcionalidad para la que ha sido programada. Este diseño es muy útil para ver de manera la operativa funcional de nuestro switch.

```

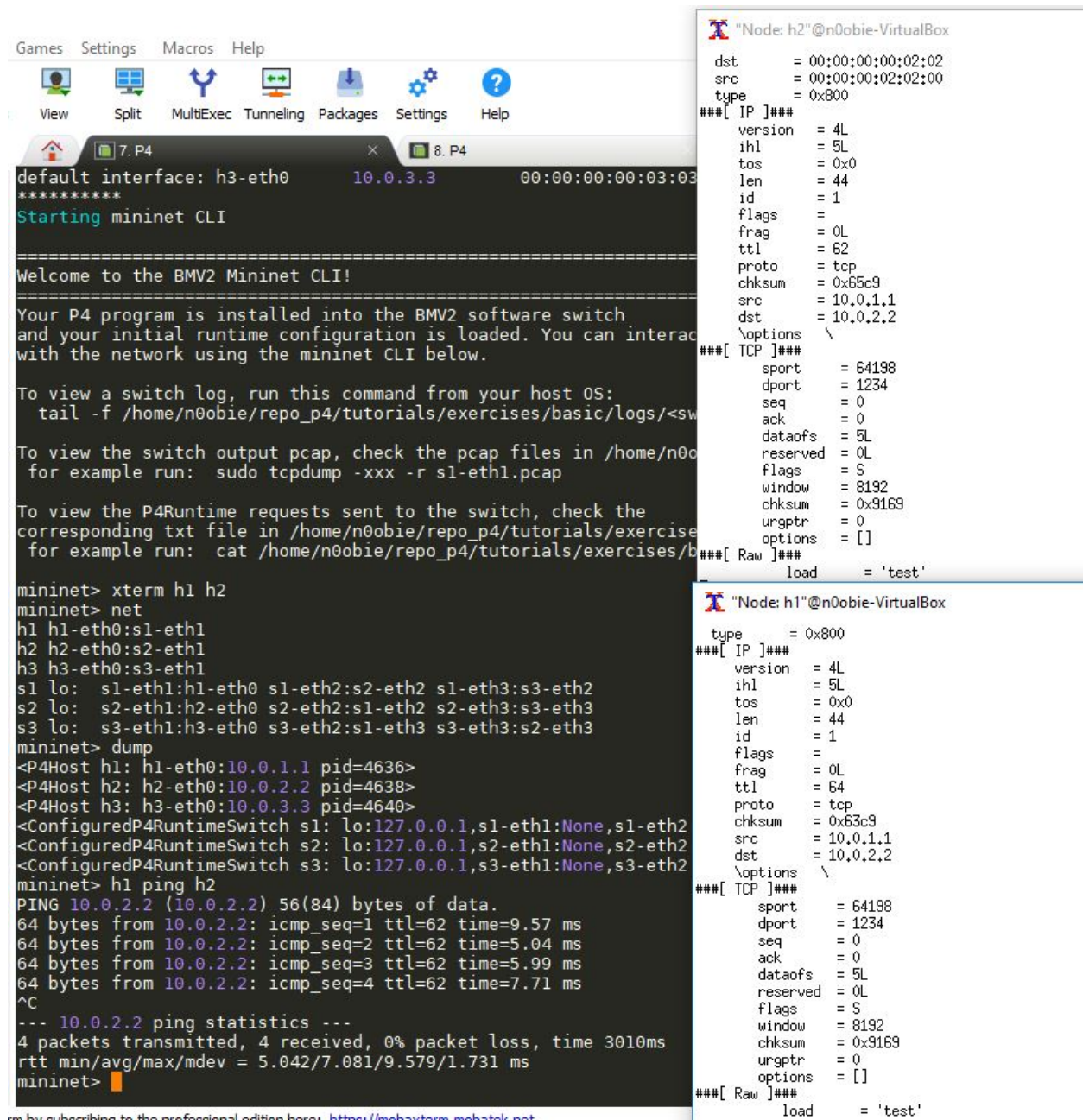
/*****
*****  S W I T C H  *****/
*****/

V1Switch
MyParser(),
MyVerifyChecksum(),
MyIngress(),
MyEgress(),
MyComputeChecksum(),
MyDeparser()
] main;

```

Figura 1.6: Topología funcional de nuestro switch.

A continuación, se expone como tras implementar los cambios existe conectividad en la topología entre todos los host. Para implementar los cambios antes hemos realizado un **sudo make stop** y un **sudo make clean** para limpiar los archivos residuales de los switch p4 y de Mininet.



```

Games Settings Macros Help
View Split MultiExec Tunneling Packages Settings Help

7. P4 8. P4
default interface: h3-eth0 10.0.3.3 00:00:00:00:03:03
*****
Starting mininet CLI

=====
Welcome to the BMV2 Mininet CLI!
=====

Your P4 program is installed into the BMV2 software switch
and your initial runtime configuration is loaded. You can interact
with the network using the mininet CLI below.

To view a switch log, run this command from your host OS:
    tail -f /home/n0obie/repo_p4/tutorials/exercises/basic/logs/<sw

To view the switch output pcap, check the pcap files in /home/n0obie/repo_p4/tutorials/exercises/basic/logs/<sw>
for example run: sudo tcpdump -xxx -r s1-eth1.pcap

To view the P4Runtime requests sent to the switch, check the corresponding txt file in /home/n0obie/repo_p4/tutorials/exercises/basic/logs/<sw>
for example run: cat /home/n0obie/repo_p4/tutorials/exercises/basic/logs/<sw>.txt

mininet> xterm h1 h2
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
s1 lo: s1-eth1:h1-eth0 s1-eth2:s2-eth2 s1-eth3:s3-eth2
s2 lo: s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth3
s3 lo: s3-eth1:h3-eth0 s3-eth2:s1-eth3 s3-eth3:s2-eth3
mininet> dump
<P4Host h1: h1-eth0:10.0.1.1 pid=4636>
<P4Host h2: h2-eth0:10.0.2.2 pid=4638>
<P4Host h3: h3-eth0:10.0.3.3 pid=4640>
<ConfiguredP4RuntimeSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None>
<ConfiguredP4RuntimeSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None>
<ConfiguredP4RuntimeSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None>
mininet> h1 ping h2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.2.2: icmp_seq=1 ttl=62 time=9.57 ms
64 bytes from 10.0.2.2: icmp_seq=2 ttl=62 time=5.04 ms
64 bytes from 10.0.2.2: icmp_seq=3 ttl=62 time=5.99 ms
64 bytes from 10.0.2.2: icmp_seq=4 ttl=62 time=7.71 ms
^C
--- 10.0.2.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3010ms
rtt min/avg/max/mdev = 5.042/7.081/9.579/1.731 ms
mininet>

```

```

"Node: h2"@n0obie-VirtualBox
dst      = 00:00:00:00:02:02
src      = 00:00:00:00:02:00
type     = 0x800
#### IP ####
version  = 4L
ihl      = 5L
tos      = 0x0
len      = 44
id       = 1
flags    =
frag     = 0L
ttl      = 62
proto    = tcp
chksum   = 0x65c9
src      = 10.0.1.1
dst      = 10.0.2.2
\options \
#### TCP ####
sport    = 64198
dport    = 1234
seq      = 0
ack      = 0
dataofs  = 5L
reserved = 0L
flags    = S
window   = 8192
chksum   = 0x9169
urgptr   = 0
options  = []
#### Raw ####
load     = 'test'

"Node: h1"@n0obie-VirtualBox
type     = 0x800
#### IP ####
version  = 4L
ihl      = 5L
tos      = 0x0
len      = 44
id       = 1
flags    =
frag     = 0L
ttl      = 64
proto    = tcp
chksum   = 0x63c9
src      = 10.0.1.1
dst      = 10.0.2.2
\options \
#### TCP ####
sport    = 64198
dport    = 1234
seq      = 0
ack      = 0
dataofs  = 5L
reserved = 0L
flags    = S
window   = 8192
chksum   = 0x9169
urgptr   = 0
options  = []
#### Raw ####
load     = 'test'

```

Figura 1.7: Funcionamiento de router programado.

1.7. Test 2: Implementando el tunelado básico

En este test, agregaremos soporte para un protocolo de tunelado básico al router que hicimos en el test anterior. El router básico reenvía en función de la dirección IP de destino. Tendremos que definir un nuevo tipo de encabezado para encapsular el paquete IP y modificar el código del router, para que en su lugar decida el puerto de salida utilizando una nueva cabecera de túnel.

El nuevo tipo de cabecera contendrá una ID de protocolo, que indica el tipo de paquete que se está encapsulando, junto con una ID de destino que se utilizará para el enrutamiento.

1.7.1. Crear protocolo de tunelado

Siguiendo la guía debemos completar los siguientes items para conseguir el resultado del tunel entre los host.

- Se debe actualizar el parser para extraer la cabecera de myTunnel o la cabecera de ipv4 según el campo etherType en la cabecera de Ethernet. El etherType correspondiente al encabezado **myTunnel** es **0x1212**. El parser también debe extraer el encabezado ipv4 después del encabezado myTunnel si `proto_id == TYPE_IPV4`.
- Debemos definir una nueva acción llamada **myTunnel_forward** que deberá situar el puerto de salida dado por el plano de control.
- Definir una tabla llamada como **myTunnel_exact**. Esta tabla se encargará de asociar de forma exacta dado un `dst_id` e invocará la acción programada anteriormente para especificar un puerto de salida idoneo para el `dst_id` dado. Por defecto hará un drop.
- Hay que modificar el campo de **apply** del bloque de control de **MyIngress**. Este deberá aplicar en un primer caso la tabla **myTunnel_exact** siempre y cuando se encuentre la cabecera myTunnel. En caso contrario se deberá aplicar la tabla del test anterior de `ipv4_lpm`.
- Hay que actualizar el Deparser para que añada la cabecera nueva de myTunnel.

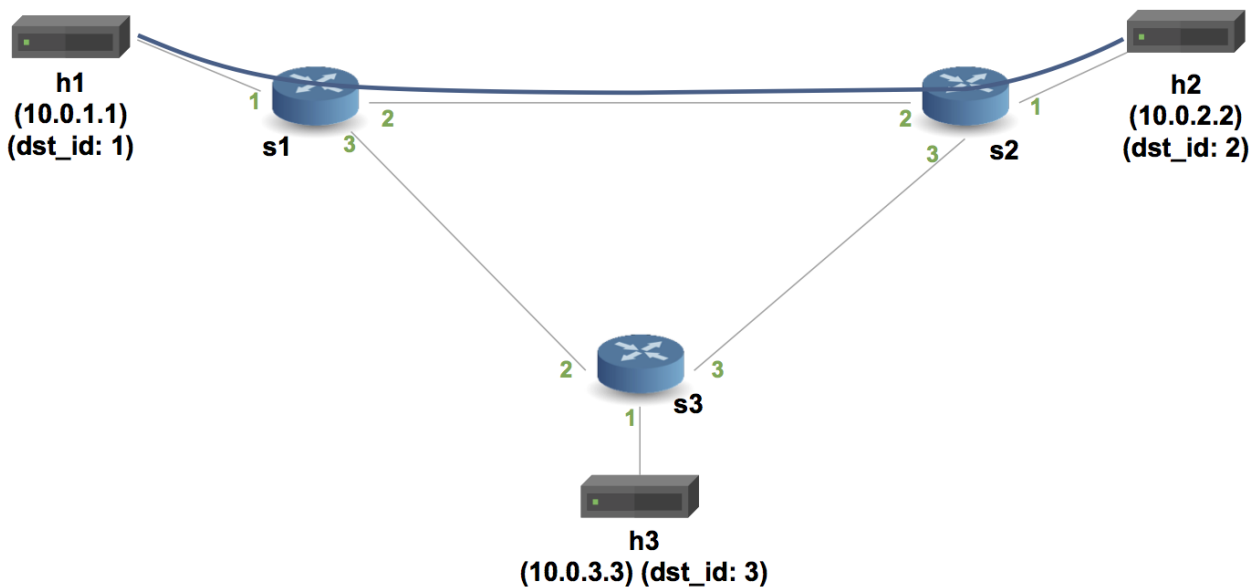


Figura 1.8: Escenario a recrear.

Para cumplir el primer requerimiento se ha procedido de forma análoga al Test 1. Hemos extraído la cabecera de Ethernet, acto seguido hemos hecho un select en función del valor del EtherType, para parsearla cabecera Ipv4 o myTunnel.

```

/*****
***** P A R S E R *****/
*****/

// TODO: Update the parser to parse the myTunnel header as well
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_IPV4 : parse_ipv4;
            TYPE_MYTUNNEL : parse_mytunnel;
            default : accept;
        }
    }

    state parse_mytunnel {
        packet.extract(hdr.myTunnel);
        transition select(hdr.ethernet.etherType) {
            TYPE_IPV4 : parse_ipv4;
            default : accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition accept;
    }
}

```

Figura 1.9: Parser actualizado.

Hemos creado la nueva acción para hacer el forward hacia en puerto de salida dado por el plano de control. Esto lo especificamos indicándolo en los meta-datos asociados a ese paquete en el campo de egress port.

```

// TODO: declare a new action: myTunnel_forward(egressSpec_t port)
action myTunnel_forward (egressSpec_t port) {
    standard_metadata.egress_spec = port;
}

```

Figura 1.10: Nueva acción de forwarding.


```
// TODO: declare a new table: myTunnel_exact
// TODO: also remember to add table entries!
table myTunnel_exact {

    key = {
        hdr.myTunnel.dst_id: exact;
    }

    actions = {
        myTunnel_forward;
        drop;
    }
    size = 1024;
    default_action = drop();
}

apply {

    // TODO: Update control flow
    if(hdr.myTunnel.isValid()){
        myTunnel_exact.apply();
    }else{
        if (hdr.ipv4.isValid()) {
            ipv4_lpm.apply();
        }
    }
}
}
```

Figura 1.11: Nueva tabla de nuestro router.

Para definir la nueva tabla hemos aplicado un criterio de búsqueda exacto, y que lo haga en función del `dst_id` de la cabecera parseada. Si no hubiera ningún hit, por defecto se descarta el paquete. El campo `apply` también ha sido actualizado para que primero se pase por la tabla de `myTunnel_exact` en caso de que exista una cabecera válida de `myTunnel`, en caso contrario se sigue pasando por la tabla `ipv4_lpm`.

```
/******
***** DEPARSER *****
******/

control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        // TODO: emit myTunnel header as well
        packet.emit(hdr.myTunnel);
        packet.emit(hdr.ipv4);
    }
}
```

Figura 1.12: Deparser actualizado.

En cuanto al Deparser únicamente se ha añadido la sentencia `packet.emit(hdr.myTunnel)`. Esto hará que en caso de haber un paquete que contenga la cabecera `myTunnel` y que esta sea válida, la serializará. En caso contrario omitirá esta sentencia.

1.7.2. Parte Avanzada Tunnelado

Siguiendo la guía nos ofrecen una parte avanzada para hacer más interesante y realista del test realizado. Esto consistirá en añadir la cabecera del túnel a la entrada del túnel, y quitársela a la salida, antes de entregarle el paquete al host final.

Para llevar esta tarea a cabo, nos das dos pistas que nos serán de gran utilidad.

- El switch de ingreso deberá asignar en función de la dirección IP de destino el `dst_id` correspondiente para el encabezado myTunnel. Además, se debe establecer el bit de validez para el encabezado myTunnel para que pueda ser emitido por el Deparser.
- El switch de egreso deberá eliminar el encabezado myTunnel del paquete después de buscar el puerto de salida apropiado usando el campo `dst_id`.

Atendiendo a los requerimientos, a las pistas que nos han dado, hemos diseñado dos nuevas acciones que utilizaremos en el bloque de control de **MyIngress**, llamadas **myTunnel_ingress** y **myTunnel_egress**. Estas acciones se encargarán respectivamente de manejar la entrada y la salida del túnel. Pero, ¿Cuándo sabemos que un paquete está entrando o saliendo del túnel? Una forma para saber si está entrada o al túnel hubiera sido el hecho de que el paquete no llevara cabecera myTunnel, pero cabe la posibilidad de que la IP destino del paquete sea la misma del emisor, por lo que no entraría al túnel y no sería consistente. Y lo que se complica más, saber cuando el switch es ultimo antes de salir del túnel.

Si nos fijamos en el diagrama de la topología podemos apreciar como todos los switches están conectados por el puerto 1 a los host. Bingo! Para este caso nos servirá, mediante el uso de reglas estáticas (archivos `sX-runtime.json`) en las tablas creadas podremos manejar los paquetes a nuestro antojo sabiendo que entran por un puerto de host y que no llevan cabecera myTunnel, se la añadiremos. También cuando el valor del `dst_id` sea el correspondiente al host que tiene más próximo a el le quitaremos la cabecera del túnel ya que ya a llegado a su destino, y lo sacamos por el puerto 1.

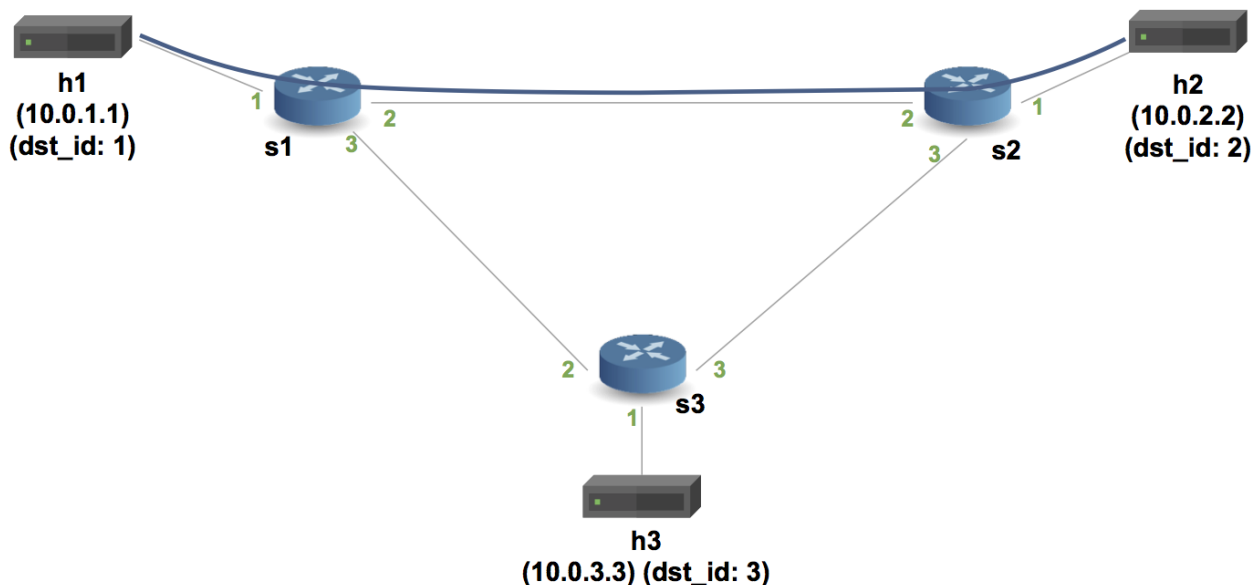


Figura 1.13: Escenario.

En este caso nos favorece la disposición del escenario, pero en caso contrario habría que darle más complejidad al procesamiento de paquetes. Sabiendo ya la estrategia a implementar, se dota de funcionalidad las acciones anteriormente descritas.

Como se puede apreciar en la acción `myTunnel_ingress` nos encargaremos, de establecer un puerto de salida apropiado (forwarding), marcar la cabecera como válida, guardar en el campo `proto_id` de la cabecera `myTunnel` el protocolo superior que va sobre él (Copiamos el valor que tenía el `etherType` antes), indicar en la trama de Ethernet el campo `etherType`, que el protocolo superior es `myTunnel`, y por último establecer un `dst_id` dado por el plano de control.

```
//PARTE AVANZADA TUNNELING

//Entrada
action myTunnel_ingress(bit<16> id, egressSpec_t port){

    myTunnel_forward(port);
    hdr.myTunnel.setValid();
    hdr.myTunnel.proto_id = hdr.ethernet.etherType;
    hdr.ethernet.etherType = TYPE_MYTUNNEL;
    hdr.myTunnel.dst_id = id;
}

//Salida
action myTunnel_egress(macAddr_t dstAddr, egressSpec_t port){
    myTunnel_forward(port);
    hdr.ethernet.dstAddr = dstAddr;
    hdr.ethernet.etherType = hdr.myTunnel.proto_id;
    hdr.myTunnel.setInvalid();
}
}
```

Figura 1.14: Acciones `myTunnel_ingress` y `myTunnel_egress`.

En cuanto a la acción `myTunnel_egress`, se encargará de hacer forwarding de la cabecera `myTunnel`, modificar las cabeceras de Ethernet para establecer el destino y modificar el `etherType` con el valor del `proto_id`. Además, invalidamos la cabecera `myTunnel` para sacarla del paquete a la hora de hacer el des-parseo.

De forma adicional, hemos tenido que añadir estas dos nuevas acciones a las tablas ya creadas. Para así poder hacer uso de ellas cuando se estime oportuno.

- `ipv4_lpm`
- `myTunnel_exact`

Con la tabla `ipv4_lpm` controlaremos el acceso al túnel y con la tabla `myTunnel_exact` controlaremos la salida al túnel.


```

table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        myTunnel_ingress;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = drop();
}

// TODO: declare a new table: myTunnel_exact
// TODO: also remember to add table entries!
table myTunnel_exact {

    key = {
        hdr.myTunnel.dst_id: exact;
    }

    actions = {
        myTunnel_forward;
        drop;
        myTunnel_egress;
    }
    size = 1024;
    default_action = drop();
}

```

Figura 1.15: Tablas actualizadas.

La lógica que se ha seguido para aplicar estas tablas ha sido la siguiente, si la cabecera Ipv4 es valida y la cabecera de myTunnel es invalida (o no está presente), aplicamos la tabla ipv4_lpm. En caso contrario, la cabecera myTunnel será valida, por lo que aplicaremos la tabla de myTunnel_exact para poder gestionar la salidas del túnel.

```

apply {
    // TODO: Update control flow
    if(hdr.ipv4.isValid() && !hdr.myTunnel.isValid()){
        ipv4_lpm.apply();
    }else{
        myTunnel_exact.apply();
    }
}

```

Figura 1.16: Lógica para aplicar las tablas.

Por último, solo quedaría definir como van a ser las entradas de las tablas `ipv4_lpm` y `myTunnel_exact`. Al igual que en el test de tunelado básico, las entradas serán estáticas. Estarán definidas por archivos json, cada switch tendrá su fichero con las distintas entradas que registrarán cada tabla de cada switch. Por así decirlo, estos ficheros actuarán como la parte de control de nuestra topología.

```
"table_entries": [
  {
    "table": "MyIngress.ipv4_lpm",
    "match": {
      "hdr.ipv4.dstAddr": ["10.0.1.1", 32]
    },
    "action_name": "MyIngress.ipv4_forward",
    "action_params": {
      "dstAddr": "00:00:00:00:01:01",
      "port": 1
    }
  },
  {
    "table": "MyIngress.ipv4_lpm",
    "match": {
      "hdr.ipv4.dstAddr": ["10.0.2.2", 32]
    },
    "action_name": "MyIngress.myTunnel_ingress",
    "action_params": {
      "id": 2,
      "port": 2
    }
  },
  {
    "table": "MyIngress.ipv4_lpm",
    "match": {
      "hdr.ipv4.dstAddr": ["10.0.3.3", 32]
    },
    "action_name": "MyIngress.myTunnel_ingress",
    "action_params": {
      "id": 3,
      "port": 3
    }
  },
]
```

Figura 1.17: Definición de entradas a las tablas del switch 1.

Se puede apreciar como cada entrada tiene definidos un match y una acción. De forma adicional, en el caso de que la acción lo requiera se deberá suministrarle los parámetros apropiados. Por ejemplo en tabla que gestionará la entrada del switch 1, aplicaremos la acción de `myTunnel_ingress` siempre y cuando el paquete tenga como destino IPv4 distintas al host 1 que tiene directamente conectado al switch.

En función, de la dirección destino ipv4, deberemos pasarle unos parámetros adecuados a la acción `myTunnel_ingress`, identificador de tunel y el puerto por el cual el switch debe sacar el paquete. De esta manera se procedería de forma análoga, en la tabla `myTunnel_exact` y en los otros switch, adecuando y corrigiendo las entradas para conseguir la funcionalidad deseada. A continuación, dejamos dos grafos describen la operativa funcional de los switches programados en este test. Los grafos harán referencia a los dos bloques más simbólicos del switch, el `MyParser` y `MyIngress` donde se lleva a cabo la lógica del switch.

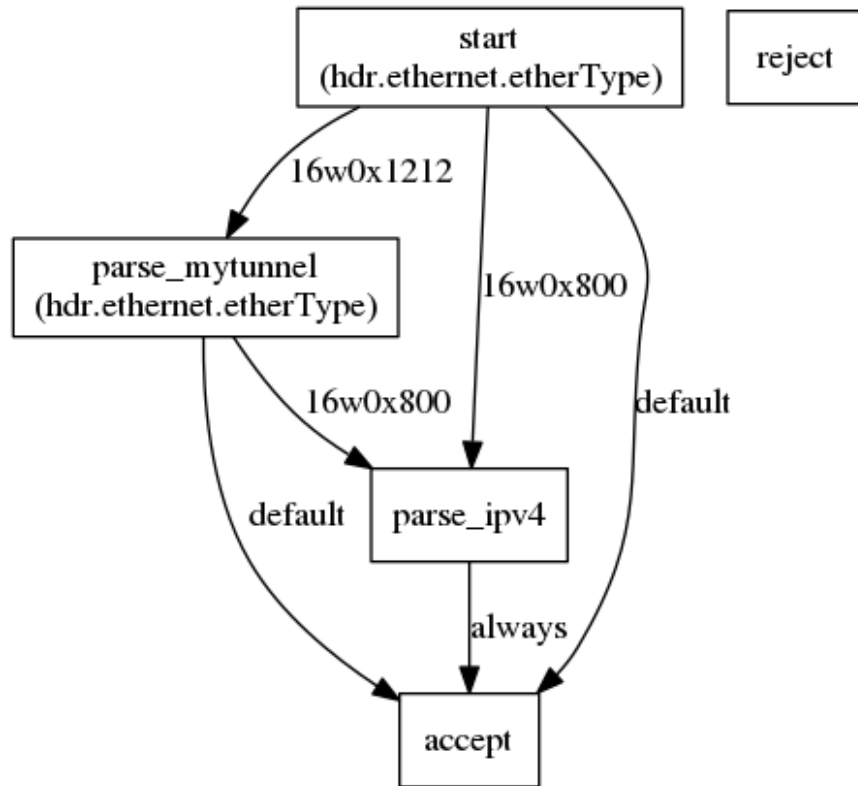


Figura 1.18: Bloque funcional MyParser

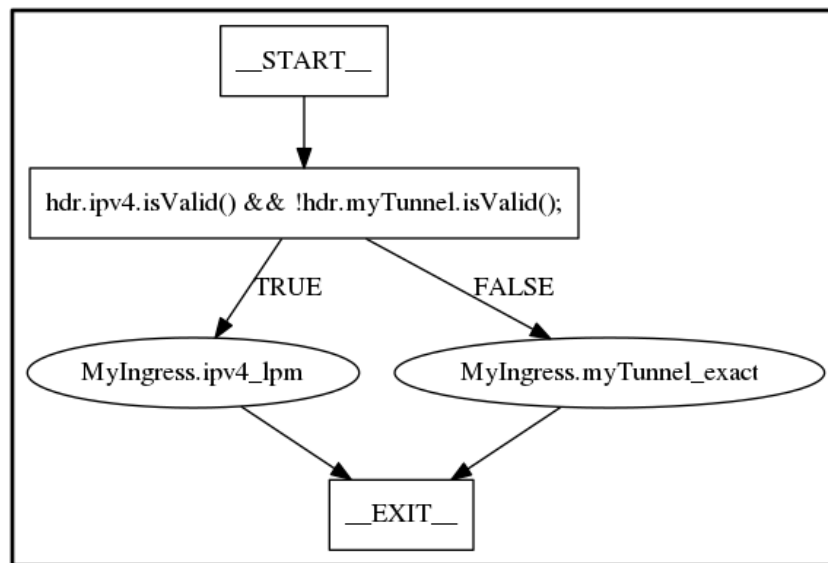


Figura 1.19: Bloque funcional MyIngress

Más información de como generar estos grafos a partir de programas p4 en el *apartado B de los anexos*.

1.8. Test 3: Implementación del plano de control vía P4Runtime

En este test, se va a usar P4Runtime para enviar entradas a las tablas del switch en lugar de utilizar la CLI del switch. El programa es el mismo programa P4 que se utilizó en la parte avanzada de tunneling. De forma adicional se han añadido dos contadores (**ingressTunnelCounter**, **egressTunnelCounter**). Se utilizarán para llevar la contabilidad de paquetes procesados.

Para implementar el plano de control se hará uso del script **mycontroller.py**, y algunas bibliotecas auxiliares en el directorio `p4runtime_lib` para crear las entradas de la tabla necesarias para encaminar el tráfico entre el host 1 y 2.

1.8.1. Escenario previo

El código para esta asignación de entradas a las tablas de los switch se encuentra en un archivo llamado **mycontroller.py**, y solo instalará algunas de las reglas que necesita para canalizar el tráfico entre dos hosts (h1, h2).

Si compilamos el programa p4 y lanzamos Mininet podremos comprobar como no hay conectividad entre los host, eso se debe a que los paquetes que le llegan pase la fase de parseo, pero cuando llegan a la fase MyIngress, y ya han pasado por la acción de myTunnel_ingress, al no haber entradas en la tabla myTunnel_exact correspondientes al forwarding de los paquetes que ya tienen cabecera myTunnel, estos no consiguen salir por el puerto correspondiente y alcanzar al siguiente switch, ya que por defecto la acción instalada es dropear los paquetes.

Bibliografía

- [1] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi. P4 edge node enabling stateful traffic engineering and cyber security. *IEEE/OSA Journal of Optical Communications and Networking*, 11(1):A84–A95, Jan 2019.
- [2] Muhammad Shahbaz, Sean Choi, Ben Pfaff, Changhoon Kim, Nick Feamster, Nick McKeown, and Jennifer Rexford. Pisces: A programmable, protocol-independent software switch. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 525–538, New York, NY, USA, 2016. ACM.
- [3] Yi-Bing Linn Shie-Yuan Wang, Chia-Ming Wu and Ching-Chun Huang. High-speed data-plane packet aggregation and disaggregation by P4 switches. *Journal of Network and Computer Applications*, 2019.
- [4] M. Uddin, S. Mukherjee, H. Chang, and T. V. Lakshman. Sdn-based multi-protocol edge switching for iot service automation. *IEEE Journal on Selected Areas in Communications*, 36(12):2775–2786, Dec 2018.

Anexos

Apéndice A

Vagrant



Figura A.1: Logo de Vagrant.

A.1. ¿Qué es?

Vagrant es una herramienta que nos ayuda a crear y manejar máquinas virtuales con un mismo entorno de trabajo. Nos permite definir los servicios a instalar así como también sus configuraciones. Está pensado para trabajar en entornos locales y lo podemos utilizar con shell-scripts, Chef, Puppet o Ansible como métodos de provisión de los recursos necesarios en la máquina virtual a desplegar.

Cabe destacar que vagrant no tiene la capacidad para correr una maquina virtual por si mismo, sino que simplemente se encarga de las características con las que debe crearse esa maquina virtual y los complementos a instalar en ella o recursos de aprovisionamiento. Para poder trabajar con las máquinas virtuales es necesario que también tengamos un proveedor de maquinas virtuales, como pueden ser VirtualBox , Wmware.

A.2. Instalación

En mi caso hice uso de Vagrant sobre Windows por lo que el método de instalación era descargar un gestor de instalación de ventanas y seguir los pasos establecidos.

- <https://www.vagrantup.com/downloads.html>

Podemos comprobar que la instalación se ha concluido satisfactoriamente haciendo uso de la shell o de la powershell. Escribimos `vagrant -v` debería mostrarnos la versión de vagrant actual que tenemos montada.

Una vez instalado Vagrant y predispuesto el fichero Vagrantfile únicamente debemos abrir la shell, powershell en Windows. Ir al directorio en cuestión donde tenemos el fichero VagrantFile y hacer **vagrant up**. Este proceso puede tardar más o menos en función de si lo hacemos la primera vez o ya hemos hecho uso de la *box* con la que se va a montar la máquina virtual.

A.3. ¿Para que sirve?

Como hemos venido mencionado vagrant sirve para ayudarnos a crear y configurar máquinas virtuales con determinadas características y componentes. La gran ventaja de vagrant es que posee un archivo de configuración **Vagrantfile** donde se centraliza toda la configuración de la VM que creamos además de añadir los métodos de aprovisionamiento de la maquina virtual.

Esto lo puede hacer vía shell-scripts que carga en la maquina virtual a gestionar o con la API que tienen establecida dándonos la posibilidad de describir en el mismo Vagrantfile los scripts deseados. El punto positivo de Vagrant es el hecho de que se puede compartir de una forma relativamente sencilla una maquina virtual con una serie de configuraciones y recursos montados. Esto se puede hacer únicamente compartiendo el archivo de VagrantFile.

A.4. ¿Cómo funciona?

Antes introducimos el término de *Box*. Vagrant cuando lee el fichero de VagrantFile lee la imagen sobre la que se va a construir la máquina virtual. Esta imagen por así llamarla, es una imagen de un SO con ciertas configuraciones ya predispuestas. Esto ahorrará tiempo en el momento de hacer el boot de la misma. Aquí puede encontrar numerosas Boxes creadas por los usuarios o por propias empresas mantenedoras de ciertos SO's.

- <https://app.vagrantup.com/boxes/search>

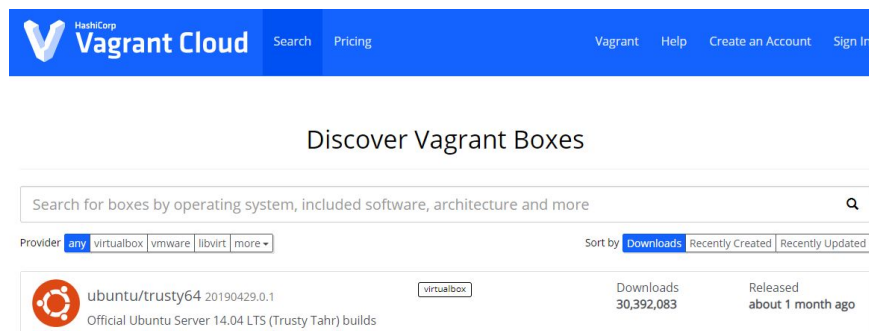


Figura A.2: Repositorio de Boxes.

A.5. Comandos útiles

```
# Arrancar una maquina virtual / Parar una maquina virtual
vagrant up / vagrant halt
# Conectarse vía ssh
vagrant ssh
# Listar maquinas creadas
vagrant global-status
# Eliminar maquina
vagrant destroy [id]
```

Apéndice B

Generar grafos desde programas p4

B.1. p4c-graphs

Muchas veces, cuando se está programando en p4 se nos escapa el flujo de trabajo que seguirá nuestro switch. P4 al tener unas características de diseño tan modulares, permite tras un análisis sintáctico generar un diagrama de bloques ó una representación simbólica de los elementos constituidos del switch mediante un esquema gráfico. Esta *feature* es un *backend* del compilador *p4c*, que por defecto se instala a la hora de hacer un build del compilador (Siempre y cuando las dependencias para este estén instaladas previamente).

Si se tiene instalado p4c, pero no encuentra p4c-graphs, siga estos pasos:

- Lo primero que vamos hacer es instalar las dependencias necesarias para poder llevar a cabo la instalación de p4c-graphs.
 - **sudo apt-get install libboost-graph-dev**
- Una vez las dependencias están instaladas, el proceso automático de build de p4c al volverlo a lanzar debería hacer un build de p4c-graphs.
 - **cd ./testP4/tutorial_p4/vm/p4c/build/**
 - Preconfiguramos el build: **sudo cmake ..**
 - Se puede acelerar el proceso de compilación, indicando número de cores que dispongamos en la máquina: **sudo make -jNUM_CORES**
 - Para tener un acceso global a herramienta solo quedaría: **sudo make install.**
 - Creará softlinks, enlaces simbólicos, a los ejecutables, en la ruta **/usr/local/bin** que está en path, para ser accesible por todos.

B.2. Herramienta auto_graph.sh

Normalmente trabajar con p4c-graphs se hace pesado ya que nos crea muchos ficheros *.dot que debemos parsear a formato png para poder ver la figuras gráficamente.

Por esto, se ha creado una herramienta que automatiza todo el proceso. Se llama auto_graph.sh, es un shellscript que te creará un directorio llamado graph, donde podrás encontrar todos los grafos en formato png de tu programa p4. Adicionalmente, te creará un subdirectorio llamado dot, donde te almacenará también todos los ficheros *.dot generados por p4c-graphs.

- Donde encontrarla: https://github.com/davidcawork/testP4/blob/master/tools/auto_graph.sh
- Uso: **./auto_graph.sh [programa p4]**

B.3. Ejemplo

```

parser MyParser(packet_in packet, out headers hdr, inout metadata meta,
                inout standard_metadata_t standard_metadata) {
  state start {
    transition parse_ethernet;
  }

  state parse_ethernet {
    packet.extract(hdr.ethernet);
    transition select(hdr.ethernet.etherType) {
      TYPE_IPV4 : parse_ipv4;
      TYPE_MYTUNNEL : parse_mytunnel;
      default : accept;
    }
  }

  state parse_mytunnel {
    packet.extract(hdr.myTunnel);
    transition select(hdr.ethernet.etherType) {
      TYPE_IPV4 : parse_ipv4;
      default : accept;
    }
  }

  state parse_ipv4 {
    packet.extract(hdr.ipv4);
    transition accept;
  }
}

```

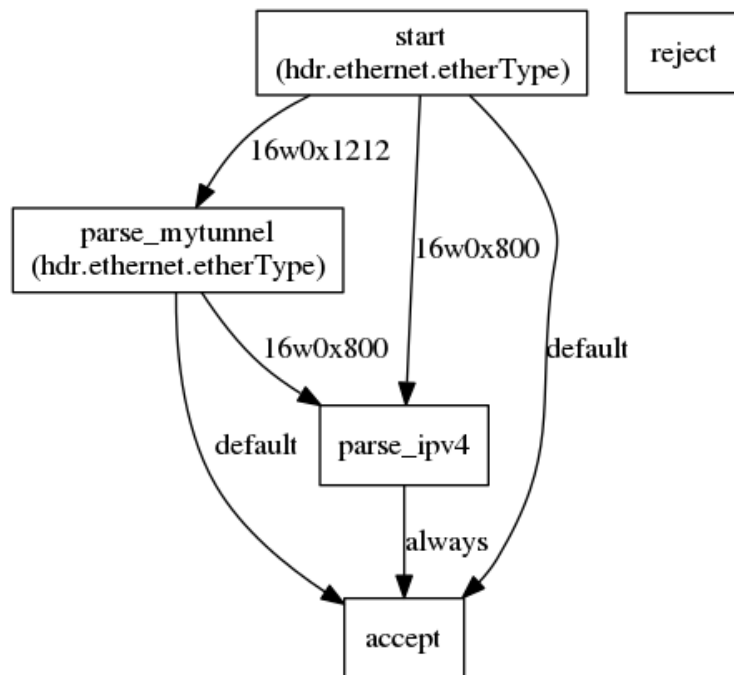


Figura B.1: Bloque funcional MyParser