Aplicación para Control de Acceso a la Red para SDN

Morillo D.*; Bernal I.*; Mejía D.*

**Escuela Politécnica Nacional, Facultad de Ingeniería Eléctrica y Electrónica*
Quito, Ecuador (Tel: 593-2-2507-144; e-mail: gabrielamorillo17@gmail.com, {ivan.bernal, david.mejia}@epn.edu.ec)

Resumen: *Las Redes Definidas por Software definen una nueva alternativa de arquitectura de red. Esta alternativa está basada en un modelo lógico centralizado, cuya idea principal es que un programa o aplicación pueda tomar decisiones para reenviar los paquetes a los switches, permitiendo un control granular del tráfico de red. En este artículo se presenta un prototipo de SDN basado en software, el cual está conformado por switches virtuales, un controlador y PC clientes; adicionalmente se presenta una aplicación para NAC (Network Access Control), la cual se implementa en el controlador y permite controlar el acceso de los dispositivos a la red.*

Palabras clave: *SDN, OpenFlow, SVN, Floodlight, NAC.*

Abstract:

Software Defined Networks establish a new network architecture alternative. This alternative is based on a centralized logic model, which main idea is that a program or application can make decisions to forward packets to switches, allowing granular control of network traffic. This paper presents a SDN prototype based on software, which consists of virtual switches, a controller and PC clients; also an application for NAC (Network Access Control) is presented, which is implemented in the controller and allows controlling the access of devices to the network.

Keywords: *SDN, OpenFlow, SVN, Floodlight, NAC.*

1. INTRODUCCIÓN

La arquitectura tradicional de las redes actuales, presenta ciertas limitaciones ante los requerimientos de los usuarios, referentes a calidad de servicio, manejo de tráfico, seguridad, entre otras; además, ciertos aspectos como la capacidad de los equipos de conectividad, los diversos tipos de tráfico que se manejan en la red y el despliegue de aplicaciones que requieren múltiples servicios a la vez, han provocado que la industria del *networking* busque una nueva arquitectura que cubra todas estas necesidades, que tenga funcionalidades sofisticadas, pero que a la vez, sea menos costosa y más sencilla de manejar que las redes de hoy en día [1], [5].

Esta nueva arquitectura se denomina SDN (*Software Defined Network*) [6], cuya ideología fundamental es la separación del plano de datos y del plano de control, y que mediante el protocolo OpenFlow [10], se realice la comunicación entre estos dos planos, y que mediante un software especializado, denominado servidor controlador se lleve a cabo la gestión del plano de control.

Por otro lado, la virtualización de los distintos componentes de una red, permite realizar un uso más eficiente de los recursos de la misma, en particular, los switches virtuales pueden convertirse en una solución flexible para centros de datos, en los que se disponga de máquinas virtuales, y que permitan la comunicación entre estas y las máquinas físicas. Este switch virtual es capaz de ejecutar todas las funcionalidades de cualquier switch físico.

Usando las ventajas de la virtualización, es posible generar un prototipo como el que se presenta en este documento, sin la necesidad de una gran cantidad de equipos físicos. Mediante este prototipo se pueden presentar las ventajas de las SDN, así como también usarlo para desarrollar y realizar pruebas con aplicaciones que corran sobre el servidor controlador. Estas aplicaciones pueden ayudar en la gestión del plano de control y con las cuales se pueden tener soluciones referentes a seguridad, enrutamiento y otras, sin necesidad de recurrir a equipamiento adicional. En este artículo se presenta una aplicación desarrollada para SDN que permita el control de acceso a la red (NAC).

2. SOFTWARE DEFINED NETWORKS

En la Figura 1 se muestran los elementos de la arquitectura de una SDN.

Artículo recibido el 14 de abril de 2014. Este trabajo fue financiado por la Escuela Politécnica Nacional, en el marco del proyecto semilla sobre esta temática que se desarrolla en el DETRI (Departamento de Electrónica, Telecomunicaciones y Redes de Información) bajo la dirección del M.Sc. David Mejía y el Dr. Iván Bernal.
Autor de contacto: David Mejía, e-mail: david.mejia@epn.edu.ec, Tel: 593-2-2507-144 ext. 2348. Escuela Politécnica Nacional. Ladrón de Guevara E11 - 253. Quito, Ecuador.

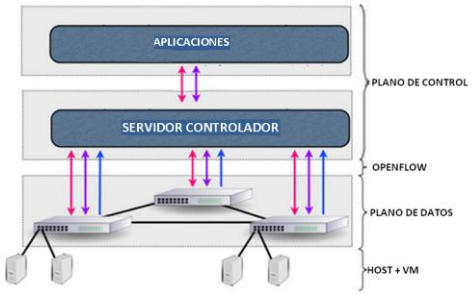

Figura. 1 *Arquitectura SDN [6]*

La infraestructura está conformada por todos los elementos físicos y virtuales que permiten el transporte de la información (medios de transmisión, equipos de conectividad y PC físicos o virtuales).

El plano de datos se encarga de la transmisión de los datos del usuario. Mientras que el plano de control usa la información de control para tomar las decisiones para el reenvío de los datos. El plano de control está formado por dos elementos: servidor controlador, encargado de centralizar de forma lógica toda la comunicación de la red; y aplicaciones de control, las cuales manejen el comportamiento de los flujos de tráfico en la red.

OpenFlow [1] es el protocolo encargado de trasladar el control de la red hacia el software de control lógicamente centralizado (servidor controlador) y permite la comunicación entre el plano de control y el plano de datos.

Las aplicaciones para gestionar el plano de control, son ejecutadas sobre el servidor controlador y son las encargadas de proporcionar las soluciones en la red referentes a enrutamiento, seguridad, calidad de servicio (QoS), ingeniería de tráfico, entre otras.

2.1 Switch OpenFlow

Son los dispositivos de conectividad que manejan el protocolo OpenFlow. La Figura 2 indica los componentes de este tipo de switches.

Tabla de Flujo: permite realizar el *matching* de los paquetes, mediante la definición de tres elementos: *Rule*, que establece los valores de los campos de los encabezados de Ethernet, IP y TCP/UDP para la definición de los flujos. *Action*, que permite establecer cómo se procesarán los paquetes de los diferentes flujos. Y *Stats*, encargado de almacenar las estadísticas relacionadas al procesamiento de los flujos.

Canal Seguro (*Secure Channel*): es la interfaz que permite la conexión entre el controlador y el switch,

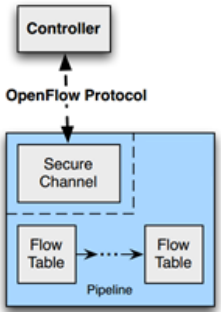

Figura 2*. Componentes de un switch OpenFlow*

2.2 Servidor Controlador

Existen diferentes alternativas de software para el servidor controlador, entre estas están: *Beacon* [2], que emplea el lenguaje de programación Java para la definición de las reglas de control, es un software estable, multiplataforma, de código abierto y provee una interfaz web de usuario; *FloodLight* [7], que nació como un proyecto alterno a Beacon, también emplea el lenguaje de programación Java para la definición de las reglas de control; *Trema* [11], que emplea los lenguajes de programación Ruby y C para la definición de las reglas de control, además, dispone de un emulador de red y una plataforma para realizar pruebas; *NOX* [4] y *POX* [9], son dos controladores que emplean el lenguaje de programación C++ y Python para la definición de las reglas de control.

2.3 Virtualización de los Funciones de la Red

NFV (*Network Functions Virtualisation*) [3] es una arquitectura de red relacionada a SDN, aunque cada una en su propio dominio. Mientras en las SDN se busca separar el plano de control del plano de datos, NFV se enfoca en portar las funciones de red a ambientes virtuales. Es decir permite desacoplar ciertas funciones de la red como NAT (*Network Address Traslation*), *firewall*, NAC (*Network Access Control*), DNS (*Domain Name System*), *catching*, detección de intrusos, entre otras, del hardware propietario, permitiendo a los administradores reducir costos respecto a los recursos usados.

3. PROTOTIPO DE RED SDN

El prototipo de red SDN implementado se presenta en la Figura 3. Este prototipo está constituido por dos PC. Se denominó PC01 a uno de estos y PC02 al otro. En el PC01 se instaló Ubuntu 12.04 LTS y sobre este el hipervisor KVM (*Kernel-based Virtual Machine*) y el gestor de máquinas virtuales Virtual Machine Manager (virt-manager), además se instaló Open vSwitch como switch virtual. Con la ayuda del hipervisor KVM se crearon tres máquinas virtuales, denominadas VM_11, VM_12 y VM_CONTROLADOR, de las cuales las dos primeras corresponden a equipos clientes y la última al servidor controlador. En la Figura 4 se puede apreciar la red virtualizada implementada en el PC01. En el PC02 se instaló Ubuntu, el hipervisor KVM, virt-manager y Open vSwitch, y con la ayuda del hipervisor se crearon dos máquinas virtuales, denominadas VM_21 y VM_22, las cuales corresponden a equipos clientes. La interconexión de ambos PC se realizó usando un switch físico.

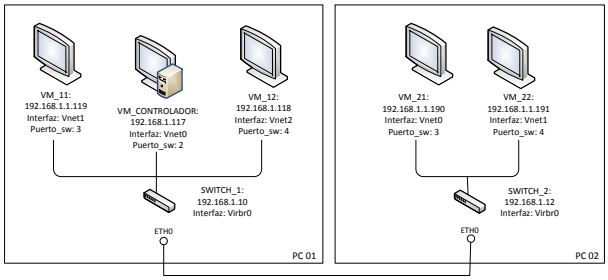

***Figura 3**. Prototipo de red SDN*

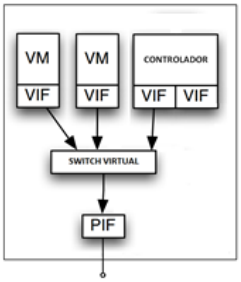

***Figura 4**. Red virtualizada en PC01*

La red virtualizada en el PC01, está implementada tomando en consideración que el equipo físico tiene una interfaz de red (PIF *Physical InterFace*), en este equipo se instalarán las máquinas virtuales (VM *Virtual Machine*) las mismas que tienen interfaces virtuales (VIF *Virtual InterFace*) para permitir la conectividad con el resto de equipos en la red, sobre el sistema operativo del equipo físico (gracias al hipervisor) y además se instala un switch virtual, el cual permitirá dicha conectividad.

En los clientes (VM_11, VM_12, VM_21 y VM_22) y en el servidor controlador se instaló el sistema operativo Ubuntu; finalmente, se escogió como software para el servidor controlador a FloodLight, debido a que dispone de una API mediante el cual puede ser extendido, permite manejar flujos de tipo reactivo y proactivo, está bien documentado y dispone de una comunidad de soporte.

Virt-manager configura una interfaz VIF llamada `virbr0`, luego de ser instalado, la cual debe asociarse al switch virtual para permitir las conexiones con el equipo físico y las máquinas virtuales. Se puede asociar la VIF `virbr0` para que se comporte como un *bridge* virtual con la PIF del equipo físico (`eth0`). Los comandos para realizar esta configuración se presenta en la Figura 5, en la línea 1 se agrega al switch virtual el `virbr0`, luego se agrega el puerto en la línea 2, se elimina la dirección IP de la interfaz `eth0` en la línea 3, se configura la dirección IP de la interfaz `virbr0` en la línea 4, se agrega la ruta por defecto al interfaz `virbr0` en la línea 5 y finalmente en la línea 6 se eliminar la ruta por defecto de la interfaz `eth0`.

```
1 $ ovs-vsctl add-br virbr0
2 ovs-vsctl add-port virbr0
3 ifconfig eth0 0
4 ifconfig virbr0 192.168.1.10 netmask 255.255.255.0
5 route add default gw 192.168.1.1 virbr0
6 route del default gw 192.168.1.1 eth0
```

***Figura 5.** Comandos para configuración del switch virtual*

Las VIF de las máquinas virtuales se denominan `vnetX`, donde `X` corresponde a un identificador para cada una, por ejemplo 0 para VM_CONTROLADOR, 1 para VM_11 y 2 para VM_12. Estas VIF deben asociarse al switch virtual a través de puertos (estas asociaciones son los equivalentes de conectar un cable de red de la PIF del equipo físico con un puerto del switch físico). Para terminar la configuración del switch, se debe indicar la dirección IP del servidor controlador, así como el puerto, usando el comando que se presenta en la Figura 6. El resultado de la configuración del switch virtual (SWITCH_1) se puede apreciar en la Figura 8.

Para realizar pruebas en el prototipo implementado se usó el módulo de FloodLight denominado StaticFlowEntryPusher, el cual permite realizar la inserción de flujos estáticos. Los flujos que se insertarán en los switches son básicamente de dos tipos, el primero corresponde al tráfico ARP, el mismo que permite llenar las tablas ARP de los switches; y el segundo, que corresponde al flujo bidireccional que permitirá la conexión entre clientes.

Los flujos están especificados en StaticFlowEntryPusher por los siguientes campos: "Switch", que corresponde al DPID (*Datapath ID*) del switch; "name", que permite asignar un nombre al flujo; "ingress-port", que corresponde al puerto en el cual está conectado el equipo que envía el mensaje; "ether-type", permite indicar el tipo de trama Ethernet; "Protocol", que corresponde al protocolo de capas superiores; "vlan-id", que corresponde al ID de la VLAN, o en caso de que no se empleen VLAN corresponde al valor de -1; "Priority", que permite establecer la prioridad de cada flujo; "src-ip", que permite indicar la dirección IP del origen de la transmisión; "dst-ip", que permite establecer la dirección IP del destino de la transmisión; "Actions", mediante el cual se establece la acción que se aplicará al flujo, por ejemplo: inundamiento (*flood*), reenvío por un puerto en particular, etc.

En la Figura 8 se presentan los comandos para agregar el flujo estático que permita el paso del tráfico ARP. La línea 1 corresponde al flujo para el SWITCH_1, y la línea 2 corresponde al flujo para el SWITCH_2. El comando cURL permite transferir datos usando varios protocolos para lo cual se requiere especificar la URL del recurso, y en particular para usar el método POST de HTTP y enviar los datos en el cuerpo del mensaje se emplea la opción –d.

```
1 $ ovs-vsctl set-controller virbr0
  tcp:192.168.1.10:6633
```

***Figura 6**. Establecimiento de la dirección IP del servidor controlador*

```
$ ovs-vsctl show
b24e1b20-51cf-4f70-b87b-62aeed445ba7
    Bridge "virbr0"
        Controller "tcp:192.168.0.117:6633"
        Port "virbr0"
            Interface "virbr0"
                Type: internal
        Port "eth0"
            Interface "eth0"
        Port "vnet0"
            Interface "vnet0"
        Port "vnet1"
            Interface "vnet1"
        Port "vnet2"
            Interface "vnet2"
```

Figura 7. *Resultado de la configuración del switch virtual*

```
-   SWITCH_1
curl  -d '{"switch":"00:00:e8:03:9a:ab:ff:b2",
"name":"static-flow1", "ingress-port":"1", "ether-
type":"0x0806", "priority":"30000", "actions":
"output=flood"}'
http://localhost:8080/wm/staticflowentrypusher/json

-   SWITCH_2
curl  -d '{"switch":"00:00:d0:27:88:be:31:7c",
"name":"static-flow2", "ingress-port":"1", "ether-
type":"0x0806", "priority":"30000","actions":
"output=flood"}'
http://localhost:8080/wm/staticflowentrypusher/json
```

Figura 8*. Flujo estático para tráfico ARP*

```
-   SWTICH_1
curl  -d '{"switch":"00:00:e8:03:9a:ab:ff:b2",
"name":"static-flow3",  "ingress-port":"1",  "vlan-
id":"-1", "priority":"32000", "src-
ip":"192.168.1.119", "dst-ip": "192.168.1.190",
"actions": "output=3"}'
http://localhost:8080/wm/staticflowentrypusher/json

curl  -d '{"switch":"00:00:e8:03:9a:ab:ff:b2",
"name":"static-flow4",  "ingress-port":"3", "vlan-
id":"-1", "priority":"32001", "src-
ip":"192.168.1.190", "dst-ip": "192.168.1.119",
"actions": "output=1"}'
http://localhost:8080/wm/staticflowentrypusher/json

-   SWITCH_2
curl  -d '{"switch":"00:00:d0:27:88:be:31:7c",
"name":"static-flow5", "ingress-port":"1", "vlan-
id":"-1", "priority":"32002", "src-
ip":"192.168.1.190", "dst-ip": "192.168.1.119",
"actions": "output=2"}'
http://localhost:8080/wm/staticflowentrypusher/json

curl  -d '{"switch":"00:00:d0:27:88:be:31:7c",
"name":"static-flow6", "ingress-port":"2", "vlan-
id":"-1", "priority":"32003", "src-
ip":"192.168.1.119", "dst-ip": "192.168.1.190",
"actions": "output=1"}'
http://localhost:8080/wm/staticflowentrypusher/json
```

Figura 9*. Flujo estático de dos vías para la comunicación entre dos PC*

En la Figura 9 se presentan los comandos para generar los flujos estáticos para la comunicación entre dos equipos con direcciones IP 192.168.1.190 (VM_21) y 192.168.1.119 (VM_11), la línea 1 corresponde al flujo de que se origina en la dirección IP 192.168.1.190 y que tiene como destino la IP 192.168.1.119 para el SWITCH_1, mientras que la línea 2 corresponde al flujo que se origina en la dirección IP 192.168.1.119 y cuyo destino es la dirección IP 192.168.1.190. Las líneas 3 y 4 corresponden a los flujos para el SWITCH_2 en ambos sentidos. Para el resto de equipos se agregan flujos similares, solamente haciendo los cambios respectivos respecto al puerto de entrada, y las direcciones IP.

En la interfaz web de FloodLight puede apreciarse la topología del prototipo implementado, lo cual se presenta en la Figura 10. En la Figura 11 se presenta el resultado de la ejecución del comando Ping, lo que demuestra el correcto funcionamiento del prototipo.

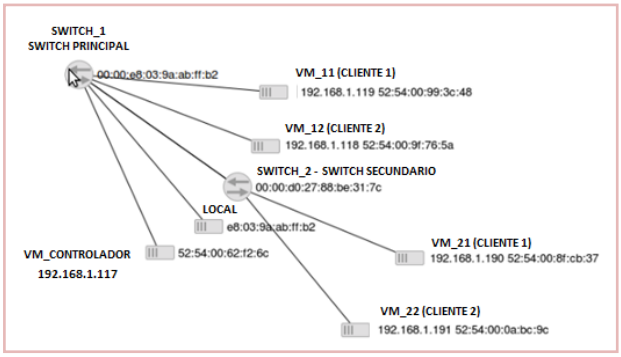

Figura 10. *Topología del Prototipo*

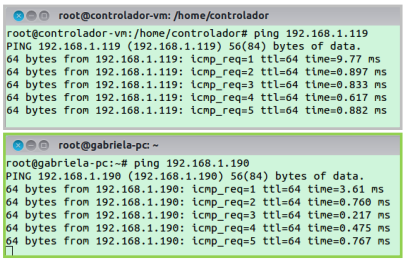

Figura 11*. Prueba de conectividad*

4. DESARROLLO DE LA APLICACIÓN NAC

La aplicación NAC permite controlar que equipos tienen acceso a la red. Para esto se definen dos acciones básicas: permitir y negar; la primera se aplica cuando el equipo del cliente tiene algún componente de seguridad, mientras que la segunda se aplica cuando carece del mismo.

Esta aplicación se basa en el modelo cliente-servidor. El servidor NAC escuchará por conexiones en el puerto 5000 y empleará TCP. El servidor NAC correrá en el controlador, mientras que el cliente NAC estará instalado en las máquinas de los clientes.

4.1 Cliente NAC

El cliente NAC implementa un mecanismo para la comunicación con el servidor, además, implementa un mecanismo que permite conocer si el equipo del cliente tiene un componente en particular, por simplicidad, únicamente se revisará la existencia de un archivo particular que indique si el equipo cumple con las políticas de seguridad. En función del contenido del archivo, el cliente NAC envía una

indicación al servidor NAC, esta indicación servirá para que se permita o se niegue el acceso del cliente a la red.

4.2 Servidor NAC

El servidor NAC implementa un mecanismo para escuchar por peticiones de los clientes NAC, además implementa un mecanismo que permita la comunicación con el controlador. También, implementa un método que le permita diferenciar a los switches de los PC. Es importante también, que permita establecer qué switch actuará como switch principal y cuáles serán los switches secundarios, cabe mencionar que el switch principal es el equipo al cual se conecta el controlador directamente, y los switches segundarios son el resto de posibles switches que no tienen conexión directa con el controlador. Tiene que verificar que no se generen reglas repetidas, eliminar las reglas de los dispositivos que eventualmente se van desconectando, así como también ensamblarlas reglas para enviarlas al controlador. Inicialmente debe instalar reglas que permitan el tráfico ARP, para que los switches puedan llenar sus tablas ARP y también definir un flujo bidireccional que permita la comunicación entre el cliente NAC y el servidor NAC.

4.3 Controlador FloodLight

Se emplea la API REST del controlador FloodLight, en particular los módulos Firewall y Forwarding. Estos dos módulos proporcionan las funcionalidades requeridas para el manejo de los paquetes y su reenvío. El módulo Forwarding se encarga de reenviar paquetes entre dos dispositivos. El módulo Firewall permite cumplir reglas ACL (*Access Control List*). El servidor controlador debe enviar el tráfico al servidor NAC para que la aplicación genere las reglas para permitir o denegar el tráfico, y una vez que tenga las reglas adecuadas, las instale en los switches. Además, se aclara que se hizo uso de Avior, una aplicación que proporciona a los administradores de red una interfaz gráfica para el manejo de ciertos módulos de la API REST de Floodlight.
En la Figura 12 se presenta un diagrama de secuencias que indica las interacciones que se realizan entre el cliente NAC, el servidor NAC y el controlador.

4. Código de la aplicación NAC

Se desarrollaron varias clases para implementar el servidor NAC y el cliente NAC. Además, fue necesario hacer algunos cambios en varias clases de Avior, debido a que estas no estaban finalizadas. Las clases empleadas se resumen a continuación: `Ruler.java`, define los parámetros de las reglas de tráfico (dirección IP, dirección MAC, puertos, acciones, entre otras) así como su serialización; `FloodLightProvider.java`, permite obtener la lista de los switches conectados y las reglas definidas; `FirewallPusher.java`, define las funciones `push()` y `remove()` para agregar y eliminar los flujos; `Firewall.java`, define todo el funcionamiento del módulo en el controlador; `Switches.java`, define las características de los equipos de conectividad (DPID, puertos, especificaciones de hardware, entre otras); `DevicesSummary.java`, define las características de los equipos de los clientes (dirección IP, dirección MAC, puerto del switch al cual están conectados, entre otras).
Para el servidor NAC se establecieron clases y métodos para realizar sus tareas, como: `loadSwitchData()`, carga los datos de los switches, en este método es necesario establecer cuál switch es el principal y cuáles son los secundarios, pues cuando se generen las reglas de reenvío se deberá especificar porque puerto enviar el tráfico; `loadDeviceData()`, carga los datos de los PC de los clientes y también elimina las reglas de aquellos equipos que se han desconectado; `Filtro()`, contiene a los argumentos que se emplean para la inserción de las reglas; `conexionCliente()`, establece la conexión con los clientes y posteriormente a partir de la información que recibe del mismo, envía al controlador las reglas para permitir o denegar su ingreso a la red. Los dos primeros métodos se ejecutan en un hilo. Se dispone de un segundo hilo el cual escucha por las peticiones de los clientes, el cual ejecuta el método `conexionCliente()`, una vez que se reciba la solicitud de conexión de un cliente NAC se crea un nuevo hilo para manejar las comunicaciones de forma independiente.
Para el cliente NAC, se emplea un socket para establecer la conexión con el servidor NAC, usando TCP y el puerto 5000.
En la Figura 13 se presenta el diagrama de clases del cliente NAC y en la Figura 14 se presenta el diagrama de clases del servidor NAC.

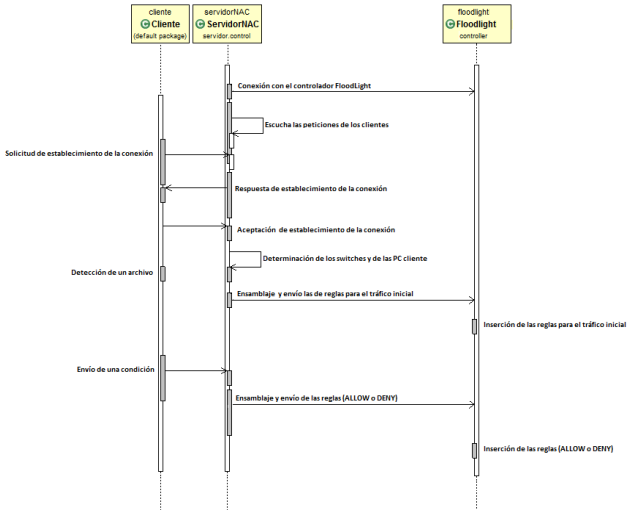

Figura 12. *Diagrama de secuencia*

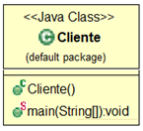

Figura 13. *Diagrama de clases de cliente NAC*

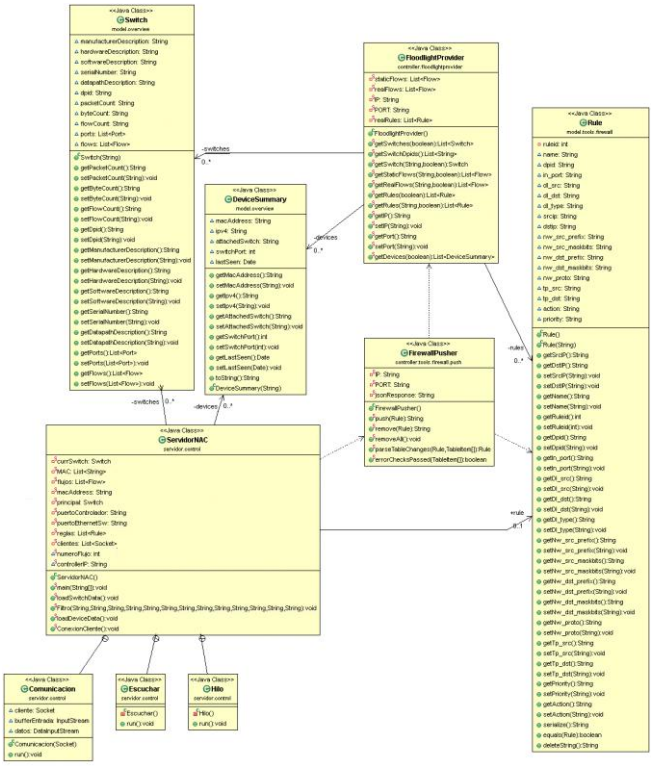

Figura 14. *Diagrama de clases de servidor NAC*

5. Pruebas y Resultados

Las pruebas se realizaron haciendo uso del prototipo de red SDN, se instalaron los clientes NAC en las máquinas virtuales, con lo cual es posible controlar el acceso a la red en función de lo que dicho cliente indique, es decir, permitir o negar su acceso a la red.

Para la primera prueba se configuro al cliente NAC para que indique que cumple con la política de seguridad, por lo que debería permitírsele el acceso a la red. En la Figura 15 se aprecia que el PC con dirección IP 192.168.1.190 tiene acceso a la red, y que puede comunicarse con otro PC de la red (192.168.1.118). En la Figura 16, se presentan las reglas insertadas en el switch principal (SWITCH_1), la primera regla permite el tráfico ARP, necesaria para que los switches llenen sus tablas ARP (línea 1); y las dos reglas restantes corresponden a un flujo bidireccional, que permiten la comunicación entre el cliente NAC y el servidor NAC usando el puerto 5000 de TCP (líneas 2 y 3). Adicionalmente, se tiene la regla que indica que el tráfico está permitido para ese cliente (línea 4).

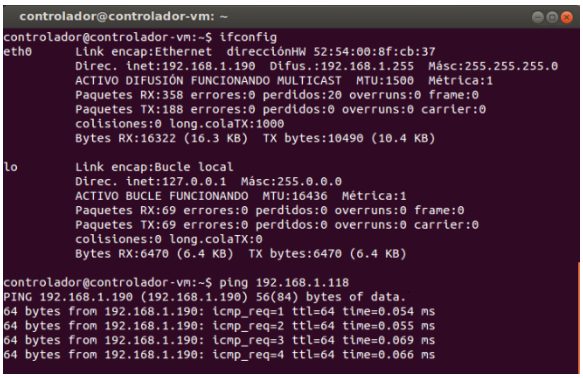

Figura 15. *Conectividad exitosa de 192.168.1.190 con 192.168.1.118*

```
1 {"ruleid":1641349361,"dpid":"00:00:e8:03:9a:ab:ff:b2","in_p
  ort":0,"dl_src":"00:00:00:00:00:00","dl_dst":"00:00:00:00:0
  0:00","dl_type":2054,"nw_src_prefix":"0.0.0.0","nw_src_mask
  bits":0,"nw_dst_prefix":"0.0.0.0","nw_dst_maskbits":0,"nw_p
  roto":0,"tp_src":0,"tp_dst":0,"wildcard_dpid":false,"wildca
  rd_in_port":true,"wildcard_dl_src":true,"wildcard_dl_dst":t
  rue,"wildcard_dl_type":false,"wildcard_nw_src":true,"wildca
  rd_nw_dst":true,"wildcard_nw_proto":true,"wildcard_tp_src":
  true,"wildcard_tp_dst":true,"priority":0,"action":"ALLOW"}]

2 {"ruleid":1952039499,"dpid":"00:00:e8:03:9a:ab:ff:b2","in_p
  ort":0,"dl_src":"00:00:00:00:00:00","dl_dst":"00:00:00:00:0
  0:00","dl_type":2048,"nw_src_prefix":"0.0.0.0","nw_src_mask
  bits":0,"nw_dst_prefix":"0.0.0.0","nw_dst_maskbits":0,"nw_p
  roto":6,"tp_src":0,"tp_dst":5000,"wildcard_dpid":false,"wil
  dcard_in_port":true,"wildcard_dl_src":true,"wildcard_dl_dst
  ":true,"wildcard_dl_type":false,"wildcard_nw_src":true,"wil
  dcard_nw_dst":true,"wildcard_nw_proto":false,"wildcard_tp_s
  rc":true,"wildcard_tp_dst":false,"priority":0,"action":"ALL
  OW"},

3 {"ruleid":1739665584,"dpid":"00:00:e8:03:9a:ab:ff:b2","in_p
  ort":0,"dl_src":"00:00:00:00:00:00","dl_dst":"00:00:00:00:0
  0:00","dl_type":2048,"nw_src_prefix":"0.0.0.0","nw_src_mask
  bits":0,"nw_dst_prefix":"0.0.0.0","nw_dst_maskbits":0,"nw_p
  roto":6,"tp_src":5000,"tp_dst":0,"wildcard_dpid":false,"wil
  dcard_in_port":true,"wildcard_dl_src":true,"wildcard_dl_dst
  ":true,"wildcard_dl_type":false,"wildcard_nw_src":true,"wil
  dcard_nw_dst":true,"wildcard_nw_proto":false,"wildcard_tp_s
  rc":false,"wildcard_tp_dst":true,"priority":0,"action":"ALL
  OW"},

4 {"ruleid":983025134,"dpid":"00:00:e8:03:9a:ab:ff:b2","in_po
  rt":0,"dl_src":"52:54:00:8F:CB:37","dl_dst":"00:00:00:00:00
  :00","dl_type":0,"nw_src_prefix":"0.0.0.0","nw_src_maskbits
  ":0,"nw_dst_prefix":"0.0.0.0","nw_dst_maskbits":0,"nw_proto
  ":0,"tp_src":0,"tp_dst":0,"wildcard_dpid":false,"wildcard_i
  n_port":true,"wildcard_dl_src":false,"wildcard_dl_dst":true
  ,"wildcard_dl_type":true,"wildcard_nw_src":true,"wildcard_n
  w_dst":true,"wildcard_nw_proto":true,"wildcard_tp_src":true
  ,"wildcard_tp_dst":true,"priority":0,"action":"ALLOW"},
```

Figura 16. *Reglas insertadas en el switch principal*

Para la segunda prueba, se configuro al cliente NAC para que indique que no cumple con la política de seguridad, por lo que no debería permitírsele el acceso a la red. En la Figura 17 se muestra la conexión fallida del PC con dirección IP 192.168.1.191 al tratar de comunicarse con otro equipo (192.168.1.118), y en la Figura 18 se aprecian las reglas insertadas en el switch principal.

5. CONCLUSIONES

En el prototipo de SDN se empleó virtualización para las máquinas de los clientes, el controlador y para los switches virtuales, con los cuales mediante el manejo de flujos estáticos y las pruebas realizadas se demostró que un administrador puede controlar de forma adecuada el flujo de tráfico en la red según las necesidades de la misma.

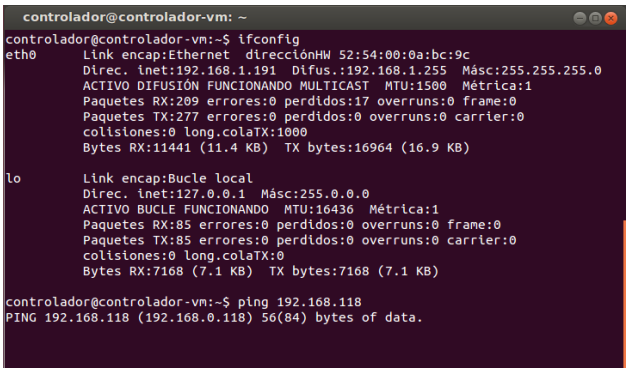

Figura 17. *Conectividad fallida de 192.168.1.191 con 192.168.1.118*


```
1 {"ruleid":1641349361,"dpid":"00:00:e8:03:9a:ab:ff:b2","in_p
ort":0,"dl_src":"00:00:00:00:00:00","dl_dst":"00:00:00:00:0
0:00","dl_type":2054,"nw_src_prefix":"0.0.0.0","nw_src_mask
bits":0,"nw_dst_prefix":"0.0.0.0","nw_dst_maskbits":0,"nw_p
roto":0,"tp_src":0,"tp_dst":0,"wildcard_dpid":false,"wildca
rd_in_port":true,"wildcard_dl_src":true,"wildcard_dl_dst":t
rue,"wildcard_dl_type":false,"wildcard_nw_src":true,"wildca
rd_nw_dst":true,"wildcard_nw_proto":true,"wildcard_tp_src":
true,"wildcard_tp_dst":true,"priority":0,"action":"ALLOW"}]

2 {"ruleid":1952039499,"dpid":"00:00:e8:03:9a:ab:ff:b2","in_p
ort":0,"dl_src":"00:00:00:00:00:00","dl_dst":"00:00:00:00:0
0:00","dl_type":2048,"nw_src_prefix":"0.0.0.0","nw_src_mask
bits":0,"nw_dst_prefix":"0.0.0.0","nw_dst_maskbits":0,"nw_p
roto":6,"tp_src":0,"tp_dst":5000,"wildcard_dpid":false,"wil
dcard_in_port":true,"wildcard_dl_src":true,"wildcard_dl_dst
":true,"wildcard_dl_type":false,"wildcard_nw_src":true,"wil
dcard_nw_dst":true,"wildcard_nw_proto":false,"wildcard_tp_s
rc":true,"wildcard_tp_dst":false,"priority":0,"action":"ALL
OW"},

3 {"ruleid":1739665584,"dpid":"00:00:e8:03:9a:ab:ff:b2","in_p
ort":0,"dl_src":"00:00:00:00:00:00","dl_dst":"00:00:00:00:0
0:00","dl_type":2048,"nw_src_prefix":"0.0.0.0","nw_src_mask
bits":0,"nw_dst_prefix":"0.0.0.0","nw_dst_maskbits":0,"nw_p
roto":6,"tp_src":5000,"tp_dst":0,"wildcard_dpid":false,"wil
dcard_in_port":true,"wildcard_dl_src":true,"wildcard_dl_dst
":true,"wildcard_dl_type":false,"wildcard_nw_src":true,"wil
dcard_nw_dst":true,"wildcard_nw_proto":false,"wildcard_tp_s
rc":false,"wildcard_tp_dst":true,"priority":0,"action":"ALL
OW"},

4 {"ruleid":983025134,"dpid":"00:00:e8:03:9a:ab:ff:b2","in_po
rt":0,"dl_src":"52:54:00:8F:CB:37","dl_dst":"00:00:00:00:00
:00","dl_type":0,"nw_src_prefix":"0.0.0.0","nw_src_maskbits
":0,"nw_dst_prefix":"0.0.0.0","nw_dst_maskbits":0,"nw_proto
":0,"tp_src":0,"tp_dst":0,"wildcard_dpid":false,"wildcard_i
n_port":true,"wildcard_dl_src":false,"wildcard_dl_dst":true
,"wildcard_dl_type":true,"wildcard_nw_src":true,"wildcard_n
w_dst":true,"wildcard_nw_proto":true,"wildcard_tp_src":true
,"wildcard_tp_dst":true,"priority":0,"action":"DENY"},
```


Figura 18. *Reglas insertadas en el switch principal*

Por otro lado la aplicación NAC desarrollada permite tener el control de acceso a la red mediante la creación y generación de reglas adecuadas, con la cual se demostró que es posible implantar cualquier tipo de aplicación en una red SDN.

El trabajo futuro incluye mejorar el cliente NAC para disponer de un mecanismo que detecte, por ejemplo la falta de antivirus en el PC cliente, antivirus desactualizado, falta de actualizaciones de seguridad del sistema operativo, entre otras. También, se puede mejorar el servidor NAC para que no solamente se empleen dos acciones: permitir o denegar, sino que, en caso de que el cliente no cumpla con las políticas de seguridad, este sea ubicado en una VLAN para evitar que pueda contagiar con un virus a otros equipos, o se instale el antivirus en caso de requerirlo.

REFERENCIAS

[1] ANDERSON Tom; MCKEOWN Nick, "OpenFlow: Enabling Innovation in Campus Networks", Marzo, 2008 http://www.openflow.org/documents/openflow-wp-latest.pdf (Consultado el 6 de enero 2013)

[2] Beacon Stanford. https://openflow.stanford.edu/display/Beacon/Configuration (Consultado el 12 de junio de 2013)

[3] ETSI, "Network Functions Virtualisation – Introductory White Paper", http://portal.etsi.org/NFV/NFV_White_Paper.pdf (Consultado el 20 de junio de 2013)

[4] NOX. http://www.noxrepo.org/nox (Consultado el 9 de agosto de 2013)

[5] ONF, "The New Norm for Networks", Abril 2012, http://www.bigswitch.com/sites/default/files/sdn_resources/onf-whitepaper.pdf (Consultado el 30 de marzo de 2013)

[6] OPEN NETWORKING FUNDATION, "Software Defined Network", https://www.opennetworking.org/images/stories/downloads/white-papers/wp- sdn- newnorm.pdf (Consultado el 2 de marzo de 2012)

[7] OPENFLOWHUB, "Project Floodlight", 2011, http://floodlight.openflowhub.org/ (Consultado el 11 de enero de 2013)

[8] PARRAGA, Jason, Código fuente de Avior, Marzo, 2013, https://github.com/Sovietaced/Avior/tree/master/src (Consultado el 10 de agosto 2013)

[9] POX. http://www.noxrepo.org/pox (Consultado el 15 de septiembre de 2013)

[10] SDN Central, "What is OpenFlow?", http://www.sdncentral.com/what-is-openflow/ (Consultado el 15 de marzo de 2013)

[11] Trema, http://trema.github.io/trema/ (Consultado el 7 de julio de 2013)