Taller de sistemas ciberfísicos

Curso 2023

Programación de placas inteligentes (SmartNic)

Autores:
Lucia de Oliveira
Raul Maglione

Supervisor:
Leonardo Alberro
Eduardo Grampín

22 de noviembre de 2023

Índice

A. Introducción **3**

B. Marco teórico **4**

B.1. Placas programables (SmartNics) 4

B.1.a. Agilio SmartNic 4

B.2. Programming Protocol-independent Packet Processors (P4) 5

B.2.a. Mininet 7

C. Resultados obtenidos **7**

C.1. Instalación de las placas 7

C.1.a. Requsitos previos 7

C.1.b. Paso 1 7

C.1.c. Paso 2 8

C.1.d. Paso 3 8

C.1.e. Paso 4 8

C.1.f. Corroborar instalación 9

C.1.g. Normal mode 9

C.1.h. Instalación y complilación 10

C.2. Generación de programas en P4 para mininet 10

C.2.a. Basic forwarding 10

C.2.b. Basic tunneling 11

C.2.c. Explicit congestion notification (ECN) 12

C.2.d. Multi-Hop Route Inspection (MRI) 13

C.2.e. Source routing 13

C.2.f. Calculator 14

C.2.g. Load balancing 14

C.2.h. Quality of service 14

C.2.i. Firewall 15

C.2.j. Link Monitoring 16

C.3. Adaptación a p4 behavioural mode 17

C.4. Ejecución de programas p4 en SmartNic 18

C.4.a. Basic forwarding 18

D. Conclusiones **19**

A. Introducción

En la era actual de la tecnología de la información, la eficiencia y la innovación en el procesamiento de datos se han convertido en piedras angulares para el avance en diversas áreas del conocimiento y la industria. Este informe, elaborado como parte del Taller de Sistemas Ciberfísicos del curso 2023, se enfoca en un componente crítico de esta revolución tecnológica: las placas programables inteligentes, conocidas como SmartNics. A través de este estudio, buscamos explorar las capacidades y aplicaciones de las SmartNics, particularmente en el contexto del lenguaje de programación Programming Protocol-independent Packet Processors (P4), una herramienta emergente y poderosa en el campo de las redes informáticas.

El objetivo principal de este informe es presentar una visión integral de la programación de las SmartNics utilizando el lenguaje P4, destacando tanto sus posibilidades técnicas como los desafíos prácticos. Se examina el rol de las SmartNics en la transformación de los centros de datos, donde estas placas asumen responsabilidades clave, liberando a los servidores para concentrarse en la ejecución de aplicaciones. Esta transición no solo implica un cambio en la arquitectura de hardware, sino también una evolución en las metodologías de programación y gestión de redes.

Nuestra investigación se centra en el uso específico de las placas AgilioCX 2x10GbE SmartNIC, analizando sus especificaciones, capacidad de programación y entorno de desarrollo. Adicionalmente, profundizamos en el lenguaje P4, abordando su aplicabilidad en el diseño y la implementación de protocolos de red personalizados que pueden adaptarse rápidamente a las necesidades cambiantes de los entornos de red modernos.

A través de este informe, presentamos los resultados obtenidos de nuestros experimentos y proyectos realizados con SmartNics y P4, incluyendo una serie de programas desarrollados para Mininet, una herramienta de emulación de redes. Estos programas ilustran la versatilidad y el potencial de las SmartNics en aplicaciones como el forwarding básico, tunelización, ECN, MRI y source routing. Además, discutimos los desafíos encontrados durante la instalación y programación de las placas, así como las soluciones implementadas para superarlos.

Con este informe, aspiramos a contribuir al conocimiento y entendimiento de las SmartNics y el lenguaje P4, resaltando su importancia en el futuro de la gestión

de redes y procesamiento de datos. Esperamos que este trabajo sirva como una base sólida para futuras investigaciones y desarrollos en este campo dinámico y en constante evolución.

B. Marco teórico

B.1. Placas programables (SmartNics)

Las tarjetas de red (nics) son componentes de hardware, usualmente una placa de circuitos o un chip, que son instaladas en una computadora para permitir la conexión a una red. [1]

Las placas programables (SmartNics) surgieron con el objetivo de ser las responsables de operar la infraestructura, ayudando eficientemente a los servidores en tareas de redes, almacenamiento y seguridad, entre otras. Esto permitirá que los data centers hagan la tarea para la cual fueron pensados: ejecutar aplicaciones, mientras que el resto de las tareas serán delegadas a las placas programables. [2]

Estas placas están compuestas por bloques que son unidades de hardware altamente especializadas, llamadas aceleradores, que ejecutan los trabajos de comunicación de una manera más eficiente que las CPUs (Central Processor Unit). Algunas de las unidades son unidades flexibles, las cuales permiten que los usuarios las programen para manejar sus necesidades cambiantes y mantenerse al día con los protocolos de red a medida que estos evolucionan. [3]

Actualmente existen diversas compañías que se dedican a la venta de placas programables, entre ellas podemos encontrar: Intel, Nvidia, Xilinx, Broadcom, Marvell, Netronome, entre otras.

B.1.a. Agilio SmartNic

Para este proyecto se emplearon dos placas AgilioCX 2x10GbE SmartNIC1. Estas placas estan diseñadas para mitigar los problemas asociados a los data center y el manejo de altos volumenes de datos, haciendo un uso eficiente de los recursos de CPU.[4]

La placa cuenta con las siguientes especificaciónes:

- **Interfaces:**2-port 10GbE, SFP+
- **Memoria:**2GB DDR3
- **Sistema operativo:**RHEL, CentOS, Ubuntu
- **Hipervision:L**inux KVM
- **Conectividad de puertos:**IEEE Std 802.3ae de 10 y 40 Gigabit Ethernet, IEEE Std 802.1Q.1p VLAN tags y prioridades y IEEE P802.1Qaz D0.2 ETS

Las placas AgilioCX requieren ser instaladas en una maquina con sistema operativo Linux, además dichas placas fueron diseñadas para programarse en los lenguajes P4 y C, brindando también un entrono de desarrollo personalizado para estas placas.

Figura 1: Placa AgilioCX2x10GbE.

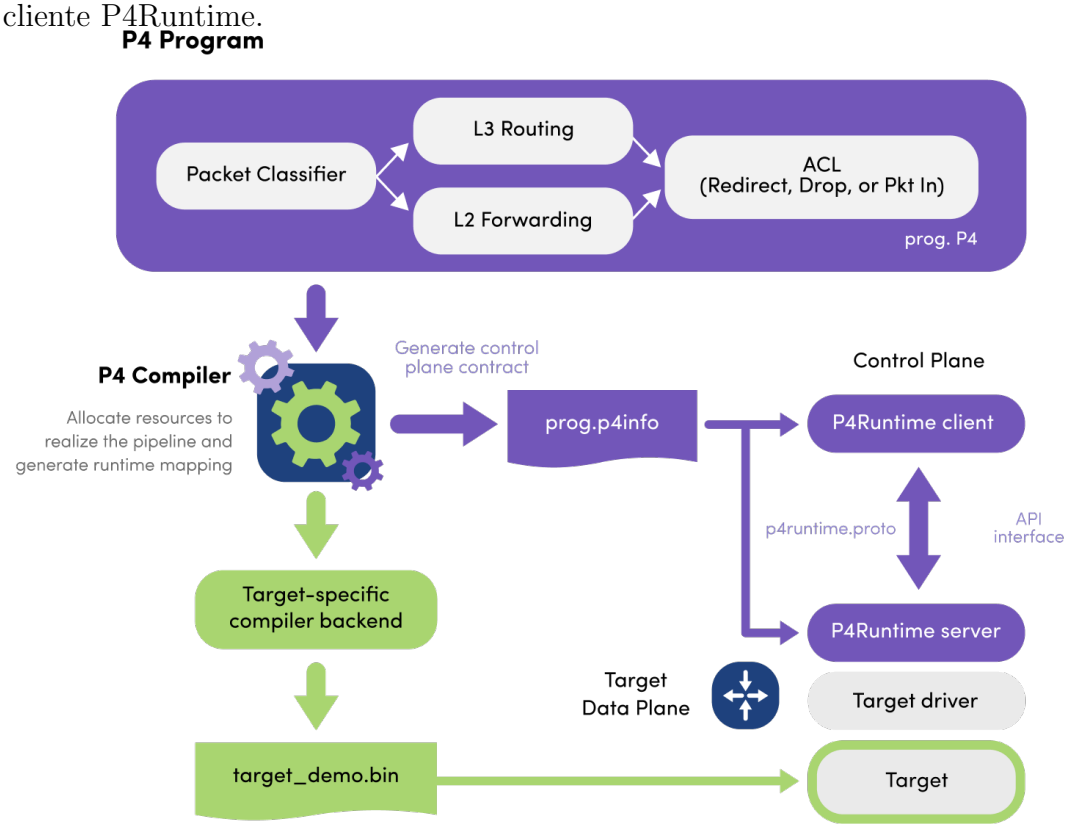
B.2. Programming Protocol-independent Packet Processors (P4)

Programming Protocol-independent Packet Processors (P4) es un lenguaje de dominio específico para dispositivos de red, que especifica cómo dispotivos como switches, NICs, routers, etcétera, procesan paquetes. Con P4, desarrolladores de

aplicaciones e ingenierios de redes pueden implementar comportamiento específico en la red y estos cambios pueden ser hechos en minutos.

Dentro de P4 se tienen dos versions $P4_{14}$ y $P4_{16}$, siendo esta la última versión del lenguaje. Actualmente la versión más utilizada es $P4_{16}$.

Un programa P4 clasifica los paquetes por su cabezal y decide las acciones que se deben tomar para paquetes entrantes (por ejemplo, forward, drop). Un compilador P4 genera un mapeo de la metada en ejecución para permitir que el plano de control y el plano de datos se comuniquen usando P4Runtime. Un compilador P4 también genera un ejecutable para el plano de datos, especificando el formato de los cabezales y las acciones correspondientes para el dispositivo objetivo. A su vez se genera un contrato para el plano de control, que es el entregado al cliente P4Runtime.



Dentro de P4 tenemos conceptos como BMv2 y V1Model. BMv2 es un modelo de switch abstracto (framework) que permite a los usuarios implementar una tarjeta por encima de él, mientras que V1Model es la arquitectura que es comúnmente usada con BMv2, utilizada para definir la estructura programable del pipeline.

Actualmente tanto los vendedores de placas que trabajan con P4 como las

plataformas de emulación y simulación de redes proporcionan su propio compilador de P4 e implementación de las librerías como V1Model con soporte específico para ese hardware o simulador. [5]

B.2.a. Mininet

Mininet es una herramienta de emulación de redes que permite simular una red de computadoras compleja utilizando una sola máquina, facilitando así el desarrollo, las pruebas y la investigación en el ámbito de las redes informáticas. Esta herramienta propociona una consola en la cual se puede interactuar con los distintos elementos que componen nuestra red.

Mininet es una manera de desarrollar, compartir y experimentar con sistemas de redes definidas por software (en inglés software defined networking, SDN), usando OpenFlow y P4. [6]

C. Resultados obtenidos

En esta sección, se detallarán los diversos resultados alcanzados, así como los desafíos enfrentados y las estrategias empleadas para superarlos.

C.1. Instalación de las placas

C.1.a. Requsitos previos

1. Linux 18.04
2. Kernel 4.xxx
3. Puertos compatibles
4. Intel Virtualization Technology for Directed I/O (VT-d) [7]
5. SDK proporcionado por la empresa

C.1.b. Paso 1

Descargar y descomprimir el SDK el cual contiene los archivos necesarios, cersiórando que se obtenga una carpeta con el nombre Agilio que dentro contenga

otra carpeta llamada install donde se alojarán todos los archivos. De no succeder crée manualmente dichas carpetas y mueva los archivos a la carpeta install.

C.1.c. Paso 2

Abra la carpeta install en la terminal e introduzca los siguientes comandos:
sudo dpkg -i nfp-sdk_6.1.0.1-preview-3243-2_amd64.deb
tar xvf nfp-sdk-p4-rte-6.1.0.1-preview-3214.ubuntu.x86_64.tar
cd nfp-sdk-6-rte-v6.1.0.1-preview-Ubuntu-Release-r2750-2018-10-10-ubuntu.binary/
sudo ./sdk6_rte_install.sh install
Luego reinicie su sistema.

Si el último comando de este paso ocacionó algun tipo de error siga estos pasos:
wget https://deb.netronome.com/gpg/NetronomePublic.key
apt-key add NetronomePublic.key
add-apt-repository "deb https://deb.netronome.com/apt stable main"
apt-get update
apt-get install agilio-nfp-driver-dkms

C.1.d. Paso 3

Introduzca el siguiente comando:
sudo update-initramfs -u

Ante cualquier error introduzca el siguiente comando:
blacklist nfp_netvf
 # Disable netdev mode; implies cpp mode is enabled
options nfp nfp_pf_netdev=0

C.1.e. Paso 4

Abra la terminal en la siguiente dirección opt/netronome/p4/bin e introduzca el siguiente comando:

sudo cp p4c-bm2-ss /opt/netronome/p4/libexec

C.1.f. Corroborar instalación

Para esté punto la placa debería estar correctamente instalada, proseguiremos a comprobar que funcione.
Introduzca los siguientes comandos si quiere acceder al modo debug:
sudo systemctl disable nfp-sdk6-rte
sudo systemctl stop nfp-sdk6-rte
sudo systemctl enable nfp-sdk6-rte-debug
sudo systemctl enable nfp-hwdbg-srv
sudo systemctl start nfp-sdk6-rte-debug
sudo systemctl start nfp-hwdbg-srv

Con estó la placa deberia estar encendida y funcionando, para verificarlo introduzca los siguientes comandos:
sudo systemctl status nfp-sdk6-rte-debug
sudo systemctl status nfp-hwdbg-srv
La salida esperada de ambos comandos debería contener en la seccion *ACTIVE* que está activa y corriendo(active(running)).

C.1.g. Normal mode

Si desea pasar del modo debug al modo normal (Normal mode) siga los siguientes pasos: *sudo systemctl stop nfp-sdk6-rte-debug*
sudo systemctl stop nfp-hwdbg-srv
sudo systemctl enable nfp-sdk6-rte
sudo systemctl start nfp-sdk6-rte

Luego confirme que la placa esta corriendo en el modo normal con el siguiente comando:
sudo systemctl status nfp-sdk6-rte

C.1.h. Instalación y complilación

Cuando se desee complilar un programa para subir a la placa se debera correr el siguiente comando, donde sorce_code.p4 representa el nombre de nuestro programa:
nfp4build –nfp4c_p4_version 16 –no-debug-info -p out -o firmware.nffw -l lithium -4 source_code.p4
(Si no detecta el comando nfp4build agregar al inicio /opt/netronome/p4/bin/)

Luego para subir el programa a la placa debera de intruducir el siguiente comando, donde design.json representa la topologia de la red:
rtecli design-load -f firmware.nffw -p out/pif_design.json

(Si no detecta el comando rtecli agregar al inicio /opt/netronome/p4/bin/)

Finalmente, para comprobar que el el programa se subió correctamente en la placa ingrese al archivo RTE que se encuentra en la siguiente dirección /var/log/nfp-sdk6-rte.log y corrobore que mencione en *loaded successfully*

C.2. Generación de programas en P4 para mininet

Para esta parte se siguieron los tutoriales de GitHub para p4 [8] donde se plantean distintos programas que se pueden implementar en p4 y luego se indica cómo ejecutar las pruebas de los mismos en mininet.

A cotinuación se presentarán los distintos programas generados y se explicará brevemente su funcionamiento. El código fuente de estos programas se puede encontrar en [9]

C.2.a. Basic forwarding

Este programa consiste en implementar un forwarding para IPv4. Para esto el switch tiene que realizar las siguientes acciones para cada paquete:

- Actualizar la MAC de origen y de destino
- Decrementar el TTL (time-to-live) en el cabezal IP
- Reenviar el paquete por el puerto apropiado

Para este programa se utilizó la topología de la figura 2

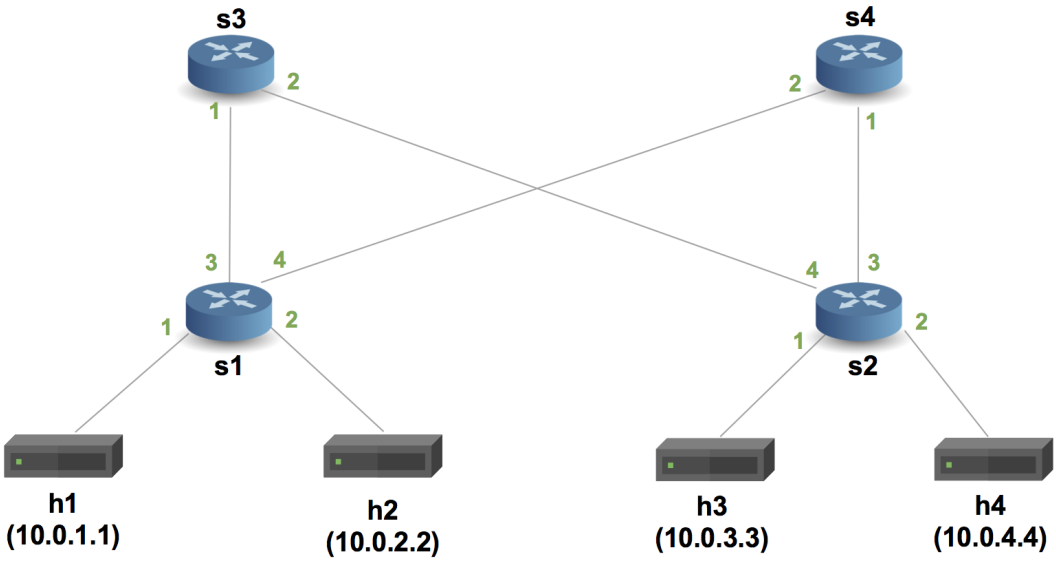


Figura 2: Descripción de la topología.

C.2.b. Basic tunneling

En este programa se agregó soporte para un protocolo báisco de tunelización al programa descrito en C.2.a. El programa consiste en definir un nuevo tipo de cabezal para encapsular el paquete IP y modificar el código del switch para que decida el puerto destino usando el nuevo cabezal definido.

El nuevo cabezal contiene un identificador del protocolo, que indica el tipo de paquete que está siendo encapsulado y un identificador del destino, el cual será usando para el reenvío.

Para este programa se utilizó la topología de la figura 3

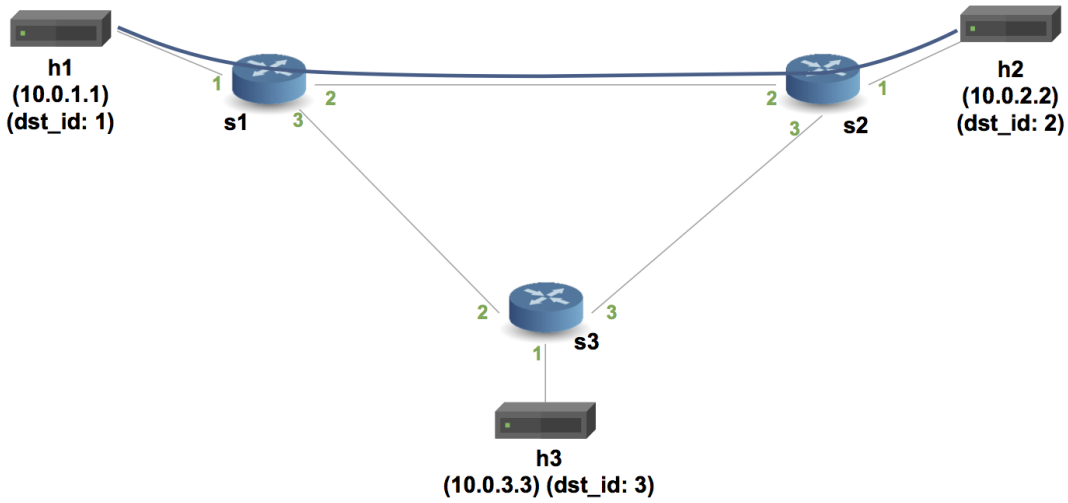


Figura 3: Descripción de la topología.

C.2.c. Explicit congestion notification (ECN)

En este programa se extendió el programa descrito en C.2.a para que implemente ECN. Esto permite la notificación de extremo a extremo de congestión en la red sin descartar paquetes. Si un host soporta ECN pone el valor 1 o 2 en el campo ecn del cabezal IPv4. Para estos paquetes, cada switch debe cambiar el valor del ecn a 3 si el largo de la cola es mayor a un umbral previamente definido, esto permite que el reemitente de los mensajes disminuya la frecuencia de los mismos para evitar saturar el enlace.

Para este programa se utilizó la topología de la figura 4

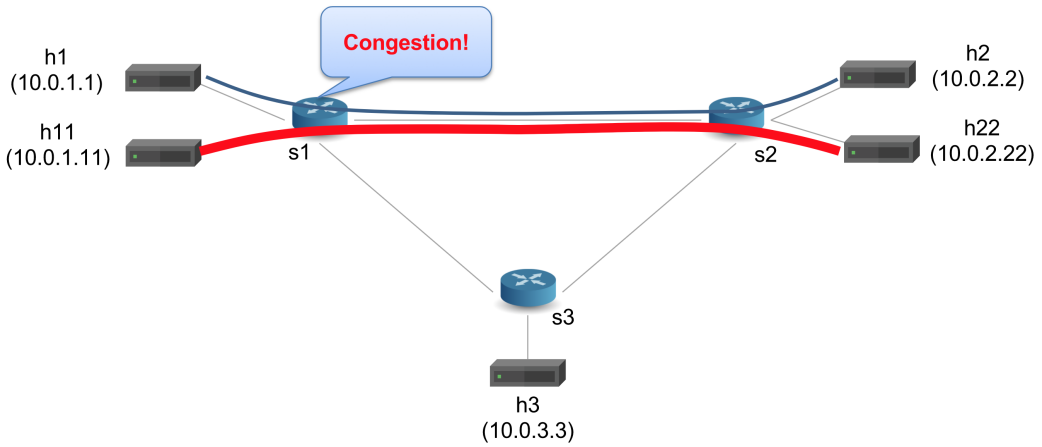


Figura 4: Descripción de la topología.

C.2.d. Multi-Hop Route Inspection (MRI)

En este programa se extendió el programa descrito en C.2.a con una versión simplificada de In-Band Network Telemetry (INT), llamada Multi-Hop Route Inspection (MRI).

MRI le permite a los usuarios rastrear el camino y el largo de las colas por las que cada paquete viaja. Para esto se escribió un programa en P4 que agrega un conjutno de identificadores y largos de cola al cabezal de cada paquete. Cuando el paquete llega a su destino, la secuencia de identificadores corresponde al camino que el paquete tomó y cada identificador es asociado con el largo de la cola correspondiente.

Para este programa se utilizó la topología de la figura 4

C.2.e. Source routing

El objetivo de este programa es implementar source routing. Con source routing el host origen guía cada switch en la red para que envíe el paquete por un puerto específico. El host agrega al paquete una pila de puertos de salida. Cada switch toma un puerto de salida de la pila y reenvía el paquete de acuerdo al número de puerto obtenido.

Para este programa se utilizó la topología de la figura 3

C.2.f. Calculator

El objetivo de este programa es implementar una calculadora básica utilizando un cabezal de protocolo personalizado. Este cabezal contendrá una operación a realizar y los dos operandos. Cuando un switch recibe un cabezal de cálculo, ejecutará la operación para los operandos dados y retornará el resultado a quién le envió el paquete.

Para este programa se utilizó una topología simple, con dos host conectados a un switch.

C.2.g. Load balancing

El objetivo de este programa es implementar un tipo de balanceo de carga basado en una versión simplificada de Equal-Cost Multipath Forwarding. El switch utilizará dos tablas para reenvíar paquetes a uno de dos host destino, elegido al azar. La primera tabla utilizará una función de hash aplicada a las direcciones IP origen y destino, el protocolo IP y los puertos TCP origen y destino para seleccionar uno de los dos hosts. La segunda tabla utilizará el valor que computó el hash para reenviar el paquete al host seleccionado.

Para este programa se utilizó la topología de la figura 3

C.2.h. Quality of service

El objetivo de este programa se extender el programa descrito en C.2.a con una implementación de Quality of Service (QOS) usando servicios diferenciados.

Los servicios diferenciados son simples y escalables, estos son utilizados para calificar y gesitionar tráfico de red y proveen QOS en redes IP modernas.

Para este programa se utilizó la topología de la figura 5

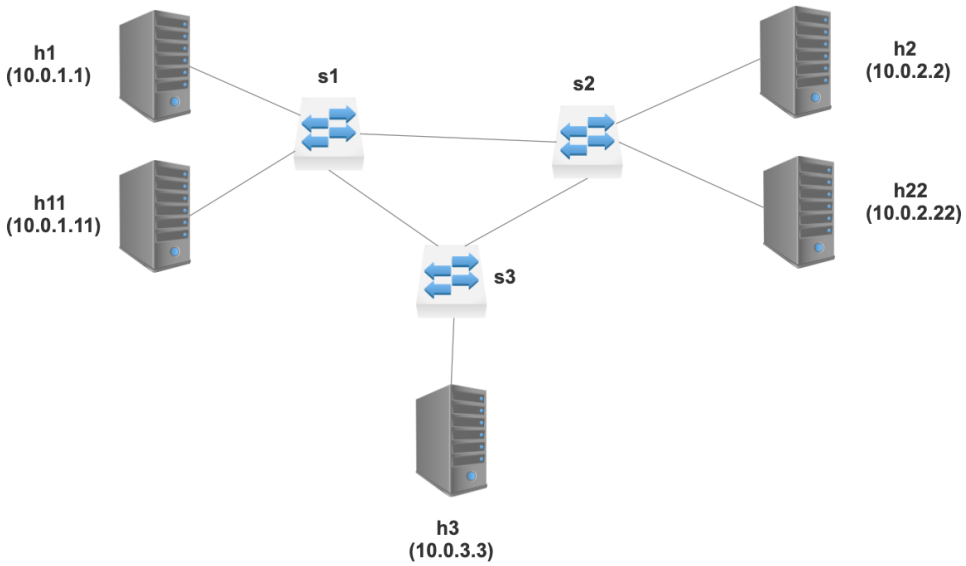


Figura 5: Descripción de la topología.

C.2.i. Firewall

El objetivo de este programa es implementar firewall sencillo con estados. Para realizar esto se utilizará el filtro de Bloom. Este ejercicio se basa en el programa descrito en C.2.a

Para este programa se utilizó la topología de la figura 2 donde el switch s1 será configurado con un programa P4 que implementa el firewall, mientras que el resto de los switches implementarán el programa descrito en C.2.a.

El firewall tendrá el siguiente comportamiento:

- Los hosts h1 y h2 están en la red interna y siempre se pueden conectar entre ellos.
- Los hosts h1 y h2 pueden conectarse libremento a h2 y h3 en la red externa.
- Los hosts h2 y h3 pueden únicamente responder a conexiones que hayan sido establecidas desde h1 o h2, pero no pueden iniciar conexiones nuevas con los hosts de la red interna.

C.2.j. Link Monitoring

El objetivo de este programa es escribir un programa P4 que permita que un host monitoree la utilización de todos los links de la red. Este ejercicio se construye sobre el programa descrito en C.2.a. Se modificará ese programa para procesar un paquete de sonda con enrutamiento de origen para que sea capaz de obtener la utilización del enlace de egreso en cada salto y devolverlo a un host para su monitoreo.

El paquete de sonda contendrá tres cabezales, uno indicando la cuenta de los saltos, otro que incluya la información que se agregará en cada salto y otro que indique el puerto por el cual el switch debería enviar el paquete, habrá un cabezal de este tipo por cada salto.

Para este programa se utilizó la topología de la figura 6

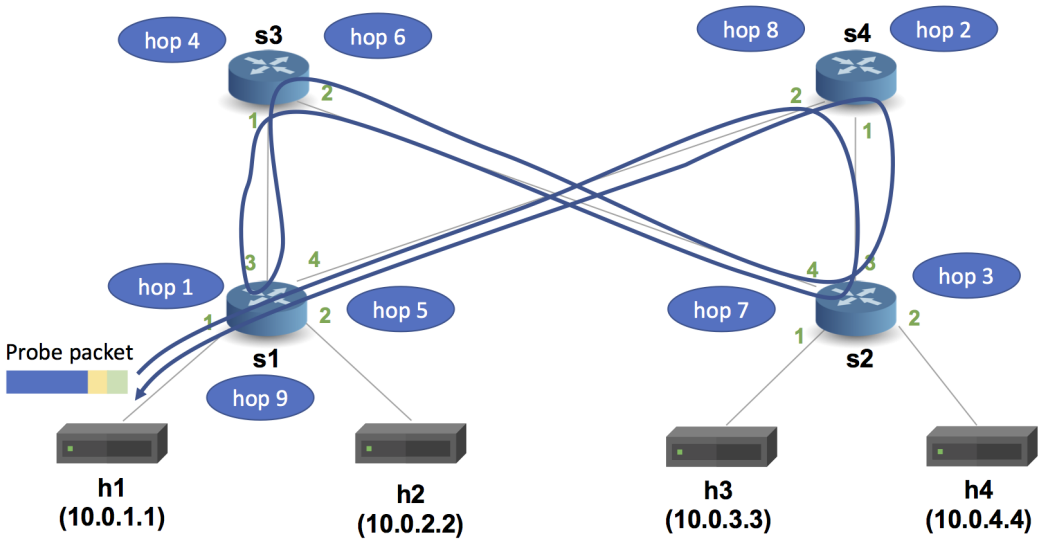


Figura 6: Descripción de la topología.

Para monitorear la utilización del enlace el switch va a mantener dos arreglos:

- `byte_cnt_reg` el cual cuenta el número de bytes transmitidos por cada puerto desde que el último paquete de sonda fue transimitido desde ese puerto.
- `last_time_reg` el cual almacena la última vez que un paquete de sonda fue transmitido por el puerto.

C.3. Adaptación a p4 behavioural mode

Aunque P4 y Mininet proporcionaron un entorno de desarrollo útil, es importante señalar que no todos los programas desarrollados son directamente compatibles con las placas. En su mayoría, se requieren algunas modificaciones para lograr una compilación correcta.

Los programas que no fue posible realizar refieren a temas como congestión, debido a que el v1model.p4 que utiliza el SDK de la placa difiere con el que emplea P4, por lo quen posee algunas limitaciones, en específico, no contamos con campos para manejar el tamaños colas (enq_qdepth), tampoco se cuenta con ninguna herramienta tipo timestamp para manejar tiempos. Otras diferencias encontradas entre el v1model de la placa y el de P4 [10] son:

- En un enumerado match_kind, que indica los tipos de matching que se pueden realizar, no existe la opción "optional" que aparentemente funciona para realizar un matching que puede ser exacto o como un comodín, que realiza matching con cualquier valor.
- Dentro del tipo standard_metadata_t no existe el campo bit<3> priority, utilizado para indicar prioridad en los paquetes.
- Falta la función extern void clone(in CloneType type, in bit<32>session) que se utiliza para clonar un paquete.
- Falta la función extern void resubmit<T>(in T data), que está deprecada.
- Falta la función void resubmit_preserving_field_list(bit<8> index), que reenvía paquetes y mantiene la metadata en el reenvío.
- Falta la función void recirculate<T>(in T data) que está deprecada.
- Falta la función void recirculate_preserving_field_list(bit<8> index), esto genera que el paquete sea procesado de nuevo por los parser, de la misma manera en la cual ingresó.
- Falta la función clone_preserving_field_list(in CloneType type, in bit<32> session, bit<8> index) que clona el paquete, creando un paquete independiente.

- Falta la función void assert(in bool check).
- Falta la función void assume(in bool check).
- Falta la función void log_msg(string msg) que permite al usuario definir mensajes de log.
- Falta la función void log_msg<T>(string msg, in T data) que permite al usuario definir mensajes de log.

C.4. Ejecución de programas p4 en SmartNic

Para la prueba de programas se utilizó el IDE Programmer Studio que está disponible para sistema operativo Windows, que permite conectarse remotamente a la placa así como compilar y subirle programas.

Teniendo en cuenta las restricciones mencionadas anteriormente se pudieron realizar los siguientes programas comptaibles con la placa:

- Basic forwarding
- Basic tunnel
- Source routing
- Calculadora
- Load balance
- Quality of service

El código adaptado de estos programas se encuentra en [9]
El resto de los programas no pudo ser compilado con éxito.

C.4.a. Basic forwarding

Este programa fue subido a la placa y se intentó probar su funcionamiento. Este programa simplemente hace forwarding de un paquete, pero no se pudo lograr una respuesta adecuada de la placa, dado que cuando se mandaban mensajes a las interfaces virtuales de la placa no se obtenía respuesta o el contador de paquetes no incrementaba.

D. Conclusiones

En este trabajo se analizó la viabilidad de emplear placas SmartNic y el lenguaje P4 para el control y análisis del flujo de paquetes de datos dentro de una red.

En primer lugar, se comprobó la eficiencia del lenguaje P4 para el desarrollo de protocolos de transporte de paquetes, esto se logró mediante el estudio del lenguaje y el desarrollo de diversos protocolos propios. Además, esto contribuyó a la comprensión de la motivación detrás de la creación de protocolos alternativos para el manejo de paquetes de red, ofreciendo una posible solución a problemas que un *datacenter* puede enfrentar.

En cuanto a los protocolos desarrollados, su implementación inicial dentro del ambiente de Mininet logró cumplir con las funcionalidades esperadas, lo que propició el desarrollo de protocolos más complejos.

Por otra parte, el empleo de las placas SmartNic no estuvo excento de complicaciones, algunas propias de la configuración, como lo son la instalación y la compatibilidad con los sistemas disponibles cuando se cuenta con escasa documentación. A pesar de los problemas, se cumplió con el objetivo de instalar las placas, además se logró compilar y subir los programas desarrollados previamente de manera exitosa.

Como trabajo futuro, existen dos lineas a profundizar. En primer lugar, ahondar más en el funcionamiento de las placas con el objetivo de completar su debida instalación, si bien se logró que los programas desarrollados corrieran en la placa, estos programas no funcionaban adecuadamente en las placas que se disponía, por lo que se recurrió al fabricante de las placas para corroborar el funcionamiento de los programas en la placa. El segundo punto sería el desarrollo de protocolos más complejos, explotando aún más el potencial que ofrece el lenguaje P4 al ser integrado con C.

Como conclusión final, se logró comprender el funcionamiento del lenguaje P4 y su potencial tanto en el desarrollo de protocolos de tranporte de paquetes como

en su versatilidad para el uso dentro de una red. Así mismo se logró verificar los objetivos planteados, así como establecer futuras lineas de investigación.

Referencias

[1] https://www.techtarget.com/searchnetworking/definition/network-interface-card

[2] https://ubuntu.com/blog/what-is-a-smartnic-and-how-is-the-technology-shaping-m

[3] https://blogs.nvidia.com/blog/what-is-a-smartnic/

[4] https://d3ncevyc0dfnh8.cloudfront.net/media/documents/PB_Agilio_CX_2x10GbE-7-20.pdf

[5] https://p4.org/

[6] https://mininet.org/

[7] https://www.intel.com/content/www/us/en/homepage.html

[8] https://github.com/p4lang/tutorials/tree/master/exercises

[9] https://gitlab.fing.edu.uy/lucia.de.oliveira/tscf

[10] https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4