Contents

1 Pruebas de escala **2**
1.1 Motivación 2
1.2 Entorno virtual 2
1.2.1 Requerimientos del entorno virtual 2
1.2.2 Herramientas 3
1.2.3 Diseño e implementación del entorno 4
1.3 Pruebas realizadas 4
1.3.1 Stress de servicios y flujos 4

Chapter 1

Pruebas

1.1 Motivación

Uno de los principales objetivos del proyecto es realizar pruebas funcionales y de escala sobre la arquitectura del prototipo. Es de interés generar realidades distintas, y así detectar puntos de falla o variables clave en la performance de la arquitectura. Para esto se puede utilizar dos parámetros: topología y servicios. Es importante poder aplicar topologías complejas y relativamente grandes a la arquitectura, así como cantidades grandes de servicios para probar qué niveles de tráfico o flujos soporta el sistema antes de fallar o ver su performance reducida drásticamente. Dado que no es realista hacer este tipo de pruebas con un prototipo físico, por temas económicos y prácticos, se observó la necesidad de un entorno virtual capaz de simular las características del prototipo.

1.2 Entorno virtual

1.2.1 Requerimientos del entorno virtual

En primer lugar, el objetivo es que el entorno virtual se comporte de una forma lo más cercana posible al prototipo físico. Esto no quiere decir que deba usar las mismas herramientas, pero es deseable que así sea. A estos requerimientos, se agregan los requisitos inherentes de un entorno de virtualización como el que se pretende. A continuación se detallan los principales requerimientos a tener en cuenta.

- OSPF. El prototipo es dependiente en que los routers puedan utilizar este protocolo de enrutamiento, entonces resulta crucial que los nodos virtuales también puedan.

- OpenFlow 1.3: La aplicación que implementa VPNs sobre el prototipo depende de que los switches OpenFlow tengan soporte para MPLS, y OpenFlow ofrece MPLS a partir de la versión 1.3 (???).
-
- Facilidad de configurar: Es importante que el entorno pueda generar distintas topologias y escenarios sin demasiado esfuerzo de configuración.
- Escalabilidad: Dado que uno de los objetivos de las pruebas es realizar pruebas de carga, el entorno debería ofrecer buena escalabilidad. Esto se traduce a que una computadora promedio de uso personal pueda levantar una decena de nodos virtuales como mínimo.

1.2.2 Herramientas

Se estudió el estado del arte en lo que respecta a opciones de emulación o simulación para SDN. A continuación se detallan las principales herramientas analizadas al momento de hacer esta investigación.

NS-3

ns-3 fue descartado debido a que no ofrece soporte para Quagga ni OpenFlow 1.3 al momento de realizar esta investigacion.

Estinet

Estinet requiere licencias pagas, y se opto por elegir herramientas open source. Debido a la falta de documentación de libre acceso, no se sabe que tipo de capacidades ofrece.

Mininet

Mininet es un emulador de redes SDN que permite emular hosts, switches, controladores y enlaces. Utiliza virtualización basada en procesos para ejecutar múltiples instancias (hasta 4096) de hosts y switches en un unico kernel de sistema operativo. También utiliza una capacidad de Linux denominada *network namespace* que permite crear "interfaces de red virtuales", y de esta manera dotar a los nodos con sus propias interfaces, tablas de ruteo y tablas ARP. Lo que en realidad hace Mininet es utilizar la arquitectura *Linux container*, que tiene la capacidad de proveer virtualización completa, pero de un modo reducido ya que no requiere de todas sus capacidades. Mininet también utiliza *virtual ethernet (veth)* para crear los enlaces virtuales entre los nodos.

LXC
La opción de crear nodos con Linux containers resuelve el problema de Quagga y OpenFlow 1.3, pero llevaría una gran cantidad de trabajo construir distintas topologias (sobre todo si son grandes), ya que casi todo debe ser configurado manualmente por el usuario. Es una opción similar a Mininet, solo que sin gozar de todas las facilidades que ofrece esta última.

Máquinas virtuales
Es una opción similar a LXC (Linux Containers), sólo que menos escalable.

1.2.3 Diseño e implementación del entorno

La herramienta que se eligió fue Mininet. *Out of the box*, Mininet ya cumple tres de los cuatro requerimientos explicados anteriormente. Está diseñada para ser escalable, ya que usa containers reducidos, tiene soporte para OpenFlow 1.3 mediante OpenVSwitch, y es muy fácil de usar. El aspecto en el que falla es en el soporte para Quagga. Dado que Mininet es una herramienta de prototipado para SDN puro, no está pensado para un esquema híbrido como el que se propone. Los switches compatibles con OpenVSwitch que ofrece no pueden tener su propio network namespace, por lo tanto, no pueden tener su propia tabla de ruteo ni interfaces de red aisladas, así que no es posible que utilicen Quagga.

Por otro lado, los hosts de Mininet sí tienen su propio network namespace, y gracias a su capacidad de aislar procesos, podemos ejecutar una instancia de Quagga y OpenVSwitch para cada host. De esta forma logramos un router como el requerido por la arquitectura. Esta extensión de las funcionalidades de los hosts es posible ya que Mininet está programado con orientación a objetos y permite al usuario crear subclases propias de sus clases.

Insertar imagen de las clases****

Insertar imagen de arquitectura de archivos de Mininet****

1.3 Pruebas realizadas

1.3.1 Stress de servicios y flujos

Es probable que si un arquitectura de este estilo fuera desplegada en la Red Académica Uruguaya, sería sujeta a grandes cantidades de tráfico y, en particular, grandes cantidades de servicios. Por eso es de gran interés realizar pruebas sobre la misma que analicen su comportamiento cuando debe mane-

jar decenas de miles, o incluso millones de flujos distintos. De esta forma se podrían identificar posibles puntos de falla, o umbrales bajo los cuales debe mantenerse la red para funcionar con buen rendimiento. En particular, interesa realizar estas pruebas estudiando dos aspectos clave: el comportamiento de los routers por sí mismos, y el de la arquitectura de la red en su conjunto.

Para estas pruebas se utilizó una topología básica de 4 routers, 2 routers CE y dos hosts como muestra la imagen (INSERTAR). Se crearon 15.000 VPNs de capa 2 entre los nodos 2 y 4, variando los cabezales OpenFlow para que toda VPN sea distinta de las demás. Además, se creó una VPN de capa 3 entre los mismos nodos, para permitir pasar el tráfico IP entre los hosts, que será el que se utilizará en las pruebas.

El comportamiento de la arquitectura fue consistente, apesar de la gran cantidad de servicios. Sólo se observó un error, que consiste en que luego de insertar algunas decenas de miles de flujos, puede haber fallas en la comunicación entre el controlador y los nodos de la red, llevando a que estos indiquen que abandonan la red. No se pudo determinar la causa exacta de este error, pero posiblemente se trate de un error en el ambiente de prueba, y no en la arquitectura. Sin considerar esto, es posible afirmar que la arquitectura de la red no tiene limitaciones con respecto a la cantidad de servicios. Sin ser una limitación, pero sí un factor importante, hay que recordar que los datos que maneja el controlador (entre ellos, los servicios) están en memoria. Por lo tanto se podrá agregar servicios mientras la computadora subyacente tenga suficiente memoria. La creación de 15.000 servicios aumentó el consumo de memoria del controlador en 205 Mb, por lo que un servicio ocupa alrededor de 14 Kb. Si extrapolamos ese número a un computador que puede dedicar 4 Gb de RAM al controlador, llegamos a que dicho controlador podrá almacenar alrededor de 300.000 servicios, un número más que suficiente.

El otro enfoque, el estudio de los routers de forma aislada, arrojó resultados interesantes. En teoría, cuantos más flujos en la tabla, más debería demorar el router en encontrar el flujo que corresponde con el tráfico que está analizando, y por lo tanto el paquete demora más en ser forwardeado. Esto debería tener un impacto directo en el throughput. Como ya fue explicado, se crearon 15.000 VPNs de capa 2 entre los nodos 2 y 4. Esto implica alrededor de 1.260.000 flujos en ambos nodos. Con la herramienta 'iperfudp', se creó tráfico UDP entre los hosts h1 y h2 y se midió el throughput promedio sin las VPNs y con ellas. Los resultados fueron exactamente los mismos. El impacto de más de un millón de flujos fue nulo. La explicación de este resultado se encuentra en la especificación de la herramienta OpenvSwitch, que utiliza la arquitectura para implementar OpenFlow. OpenvSwitch incluye un módulo en el kernel que actúa como caché para los últimos flujos utilizados. Cuando

el paquete de un determinado flujo llega por primera vez a un router, este paquete es enviado al pipeline de OpenFlow para determinar que acción se debe tomar. Luego de realizada, esta acción es escrita en la caché, y tiene un tiempo de vida de entre 5 y 10 segundos. Si en ese período de tiempo llega otro paquete del mismo flujo, no hay necesidad de consultar las tablas de OpenFlow, ya que ya se sabe que se debe hacer con ese paquete. Por lo tanto, si un flujo de datos es constante y rápido, no importa cuántos flujos OpenFlow tenga el nodo, ya que sólo el primer paquete de ese flujo deberá pasar por el pipeline, y por ende, solo él se verá demorado por la existencia de muchos flujos en el nodo.