Contents

Chapter 1

Pruebas de escala

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, la arquitectura debe funcionar completamente en el entorno. Esto quiere decir que el entorno debe soportar las mismas herramientas de software que utiliza el prototipo, y que funcionen de la misma forma. 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.

- Soporte para Quagga y OpenVSwitch: La arquitectura del prototipo es dependiente en que los routers puedan usar estas herramientas, en-

tonces resulta crucial que el entorno de emulación pueda cumplir con dicho comportamiento.

- OpenFlow 1.3: La aplicación que implementa VPNs sobre el prototipo depende de que los switches OpenFlow tengan soporte para MPLS. OpenFlow ofrece MPLS a partir de la versión 1.3.

- Fácil 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.

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.

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.

Mininet

Mininet es un emulador de redes SDN que permite emular hosts, switches, controladores y enlaces. Mininet utiliza virtualizacion basada en procesos para ejecutar multiples instancias (hasta 4096) de hosts y switches en un unico kernel de sistema operativo. Tambien utiliza una capacidad de Linux denominada *network namespace* que permite crear "interfaces de red virtuales", y de esta manera dotar a los hosts 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 opcion de crear nodos con Linux containers o maquinas virtuales resuelve el problema de Quagga y OF 1.3, pero llevaria una gran cantidad de trabajo construir distintas topologias (sobre todo si son grandes), ya que casi todo

debe ser configurado manualmente por el usuario.

Máquinas virtuales

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, 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

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 un gran interés realizar pruebas sobre la misma que analicen su comportamiento cuando debe manejar 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. Desde el punto de vista técnico, lo que se busca es estudiar el impacto que causa la escalada de flujos en los routers sobre el tiempo de "lookup". En teoría, cuantos más

flujos en la tabla, más demora 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.

En esencia, lo que se debe hacer para lograr esto es lo siguiente: se debe crear un servicio que permita pasar el tráfico que se utilizará en la prueba, y, se debe crear una cierta cantidad de servicios "dummy", es decir, que no coincidirán con el tipo de tráfico que pasará por los routers. Esa cantidad dependrá de la magnitud de la prueba que se quiera hacer. Esto causará que el router pierda tiempo buscando en sus tablas el flujo que coincide con el tráfico.

En las pruebas los servicios que se utilizaron fueron de tipo VPN de capa 2. Una VPN está compuesta por 2 servicios, uno de ida y uno de vuelta. Un servicio de capa 2 implica 42 flujos (uno por cada ethertype posible) insertados en cada router por el que pasa la VPN. Por lo tanto, una VPN equivale a 84 flujos insertados en los routers involucrados. Para crear las VPNs "dummy" se usaron distintos valores de identificador de VLAN y prioridad de VLAN (protocolo 802.1Q). Dado que hay 8 valores posibles para la prioridad, y 4096 para el identificador, se pueden crear 32.768 posibles combinaciones, y por ende la misma cantidad de VPNs distintas. Como el tráfico utilizado no tenía estos campos, ninguno de esos flujos coincidirá. Se utilizó la herramienta 'iperfudp' para medir el throughput entre los nodos. Esta herramienta envía paquetes UDP entre un nodo y otro y mide el throughput promedio.

La primera prueba realizada fue: aplicar una gran cantidad de VPNs de capa 2 a una topología básica compuesta por 4 RauSwitch, 2 routers legados (funcionando como routers CE) y 2 hosts, cada uno conectado a uno de los routers legados (insertar imagen de topologia). Los resultados que arrojó fueron los siguientes:

3.500 VPNs - 7.000 servicios - 294.000 flujos - sin cambio en el throughput

15.000 VPNs - 30.000 servicios - 1.260.000 flujos - sin cambio en el throughput