PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

Desarrollo de una solución SDN/OpenFlow de ciberseguridad para slices HPC a alta velocidad sobre OpenStack usando SR-IOV

Tesis para obtener el título profesional de Ingeniero de Telecomunicaciones

AUTOR

Gabriel Omar Gamero Montenegro

ASESOR

César Augusto Santiváñez Guarniz

Lima, febrero, 2022

Resumen

El presente trabajo de tesis consiste en el diseño de un módulo que permita instalar, en el *switch* de datos, reglas de ciberseguridad para instancias con cargas de trabajo HPC con el fin de recuperar la seguridad pérdida en una nube privada OpenStack al evitar (baipasear) la capa de procesamiento de paquetes del hipervisor gracias al uso de SR-IOV en combinación con PCI *passthrough*.

En el primer capítulo se caracteriza el estado de las redes actualmente. Además, se ahonda en las necesidades de aplicaciones como HPC, así como la problemática que presenta el uso de HPC en un entorno *cloud*. También se define la justificación, relevancia y objetivos de la tesis.

En el segundo capítulo se detallan tecnologías relevantes al desarrollo de la tesis, como PCI *passthrough* y SR-IOV; OpenStack, especialmente el proyecto de Neutron y el *feature* de *security groups*; SDN y el controlador OpenDaylight.

En el tercer capítulo se realiza el diseño del módulo de ciberseguridad en sí. Partiendo por los requerimientos y consideraciones, primero se describe la arquitectura y luego se realiza el *high level design* del módulo describiendo cada uno de los componentes que lo conforman.

En el cuarto capítulo se realizan las pruebas y obtienen los resultados. Sobre el escenario de pruebas desplegado, se hace uso del módulo de ciberseguridad para comprobar su correcto funcionamiento, verificando que la seguridad en las instancias se garantice.

Finalmente, se presentan las conclusiones como análisis de los resultados obtenidos y verificación de que los requisitos planteados hayan sido cumplidos.

Índice

Lista de Figuras

Lista de Tablas

Introducción

Las cargas de trabajo HPC (High Performance Computing) y de Big Data se caracterizan por su alta demanda de recursos de procesamiento y velocidad de red (poca tolerancia a latencia y *jitter*). Su uso cada vez mayor en diferentes áreas de la actividad humana ha incrementado una presión por el uso compartido de los recursos siguiendo el modelo IaaS de cómputo en la nube. Sin embargo, tecnologías de cómputo en la nube, como OpenStack típicamente emplean una capa de hipervisor (p.ej. software switches) que introduce un *overhead* significativo y aumenta la latencia cuando se procesan decenas de millones de paquetes por segundo.

Actualmente OpenStack ofrece dos alternativas para operar cargas HPC eficientemente: aprovisionamiento Bare-metal con Ironic, y *bypaseo* de *kernel/hipervisor* con SR-IOV. Ambas opciones reducen el *overhead* de procesamiento, pero la segunda (SR-IOV) provee de un mayor ancho de banda de comunicación al *slice* HPC, permite un mejor uso compartido de los recursos, y provee máxima flexibilidad en la elección de la configuración de los recursos de cómputo. Por esta razón, el orquestador de cargas HPC desarrollado por el grupo GIRA (el HAST) utiliza SR-IOV al desplegar *slices* HPC.

Sin embargo, al evitar la capa de hipervisor se pierde la capacidad para ejecutar reglas de seguridad y control de acceso (*security groups* en OpenStack) para los puertos de comunicaciones de las instancias del slice HPC, dejándolos expuestos a ataques (lo mismo pasa con Ironic). Una opción para recuperar la seguridad perdida sin sacrificar la escalabilidad ganada, es ejecutar estas reglas en el *switch* ToR del *rack* (a *line rate*). Para un conjunto importante de reglas de interés, esto se puede realizar usando el protocolo SDN/OpenFlow para configurar reglas en el *switch* ToR.

El objetivo de la presente tesis es el desarrollo de un módulo de ciberseguridad para el orquestador IaaS HAST del GIRA que permita instalar reglas de

seguridad para los *slices* HPC creados por esta herramienta. El módulo configura el ToR *switch* usando la interfaz REST (Northbound API) provista por el controlador SDN parte del HAST, con el fin de recuperar características tradicionalmente implementadas en el *kernel* del hipervisor y que fueron perdidas al integrar la tecnología de SR-IOV a OpenStack.

El desarrollo de esta tesis implica: (i) la determinación de las reglas de seguridad apropiadas para entornos HPC haciendo un balance entre seguridad y *performance*, (ii) el mapeo de los requerimientos del usuario a reglas implementables mediante OpenFlow, (iii) el diseño del módulo de ciberseguridad que permita instalar reglas de seguridad para los *slices* HPC, (iv) la realización de pruebas de validación del diseño en un emulador y (v) la realización de pruebas de funcionamiento y rendimiento en un escenario físico (usando un *switch* físico de datos).

1. HPC en OpenStack y SR-IOV

Las necesidades de las redes actuales evidencian grandes carencias que las soluciones tradicionales no pueden llegar a cubrir. Se requieren de nuevas herramientas que se ajusten al dinamismo de los nuevos esquemas en las redes y los servicios que estas brindan. Nuevas tecnologías emergentes tratan de dar un nuevo enfoque al diseño, despliegue y administración de las redes, mientras que las empresas muestran su interés por su incorporación debido a la rentabilidad que puede dar este tipo de planteamiento.

Los requisitos de las empresas o inconformidad de usuarios finales son factores esenciales para el surgimiento de estas tecnologías nacientes. A continuación, se enuncian distintas razones que motivan la implementación de nuevas tecnologías de red o la migración a nuevos paradigmas, inclusive.

1.1 Características del escenario actual de las redes

1.1.1 Gastos elevados

Las compañías con redes WAN que unen varias redes locales se enfrentan a costos elevados debido a MPLS y demás servicios de red privados. Periódicamente estas empresas deben renovar contratos con operadores de *data centers* para realizar ajustes a los costos. Generando deudas cada vez más importantes debido a las demandas crecientes en todo el sector de redes y aplicaciones, por ejemplo, en el ancho de banda contratado. La causa de esto radica ya sea en el crecimiento progresivo normal del negocio o debido a los cambios radicales de la temporada que transcurre.

También se suele recurrir a adquirir más equipos de red (*switch*, *router*, *firewall*, etc) como medida de solución ante las necesidades insatisfechas. La inversión realizada en adquirir estos equipos se eleva pues el *hardware* propietario viene con sistema operativo y software también propietario que solo brinda compatibilidad con los equipos de la misma marca y restringe a la empresa a las limitaciones del *vendor*.

A lo largo de los años se ha tenido a las redes que estaban construidas con circuitos integrados de aplicación específica (ASICs) y hardware de función específica como aquellas con mayor disponibilidad y rendimiento, pues son capaces de dar un trato especial a diferentes tipos de paquetes o tráfico; sin embargo, suelen ser muy costosos. Esto junto con el factor de hacer despliegues *on-premise* (soluciones dentro de servidores e infraestructuras locales) incrementan aún más los gastos.

1.1.2 Cambios dinámicos

Las redes tradicionales carecen de flexibilidad y han sido definidas por las propiedades físicas y tangibles que las caracterizan, es decir, el *hardware* y cableado que conforman las conexiones físicas entre puntos.

En redes, el término de flexibilidad se refiere a la capacidad de adaptar los recursos de red disponibles, ya sea en la topología o flujos a los cambios de requerimiento de diseño; por ejemplo, menor latencia o distribuciones de tráfico distintas [1].

En los *data center* empresariales, los patrones de tráfico han cambiado significativamente volviéndolo en ciertos casos impredecibles. A diferencia de aplicaciones cliente-servidor donde la mayor parte de la comunicación se realiza entre un cliente y un servidor, las aplicaciones acceden a distintos servidores y bases de datos, generando tráfico *east-west* (de máquina a máquina), previo al retorno de datos al usuario final en el patrón de tráfico clásico *north-south*. Simultáneamente, los usuarios están cambiando el patrón de tráfico al acceder a contenidos y aplicaciones desde diversos dispositivos, desde cualquier lado, a todo momento [2].

1.1.3 Demandas crecientes

Las aplicaciones que los usuarios emplean se vuelven cada vez más inteligentes y piden recursos mayores con el fin de cumplir con *features* como el de interacción en tiempo real. Los usos de tiempo de procesamiento, memoria y tráfico de red son cada vez mayores debido a la aparición de aplicaciones *cloud* como *big data* o de HPC. Los negocios sufren de aumentos en aplicaciones multimedia, aplicaciones *cloud* y uso móvil, por el crecimiento usual de la empresa o algún evento especial desencadenante. Como resultado, las empresas urgen de contratar un mayor ancho de banda como método de solución al incremento en la demanda de tráfico.

Por otro lado, la adquisición de nuevos equipos de red también busca satisfacer las demandas, volviendo más compleja la red al tener que configurarlos y administrarlos. La complejidad de las redes aumenta al incluir nuevas soluciones, esto influye en la dificultad del *troubleshooting* de problemas que

puedan surgir en la red al considerar esquemas como equipos inalámbricos, túneles, BYOD, etc.

1.2 HPC

High Performance Computing (HPC) consiste en la habilidad de procesar datos y realizar cálculos complejos a altas velocidades. Las soluciones HPC consisten en un conjunto grande de nodos de cómputo que trabajan juntos para completar una tarea o un conjunto de tareas, a través del procesamiento en paralelo. Para la construcción de una infraestructura HPC, se conectan los servidores de cómputo en una red, formando un *cluster* (unión de múltiples servidores de cómputo). Sobre estos se ejecutan los programas o algoritmos en simultáneo para luego enviar los resultados a un almacenamiento de datos conectados en la misma red del *cluster* [3].

Al contrario que las computadoras tradicionales, cuyo rendimiento se mide en millones de instrucciones por segundo (MIPS), los servidores HPC usan cálculos de coma flotante, los cuales son operaciones únicas en cada instrucción de computación y su métrica de medición son *floating-point operations per second* (FLOPS). En cada nodo de un *cluster* HPC encontramos un procesador que usa múltiples núcleos en un CPU o GPU. La comunicación entre *cores* permite la división de las cargas de trabajo aprovechando el diseño *multi-core* [4].

Sin importar sobre qué entorno sea desplegado, las soluciones HPC se emplean actualmente para distintos casos, tomando ventaja de este funcionamiento de alto rendimiento. Algunas aplicaciones de interés son [3], [5]:

- Aplicaciones MPI (*Message Passing Interface*): estándar para escribir programas en paralelo que definen una *application programmer interface* (API) implementando el modelo de programación m*essage-passing*. Las computadoras paralelas más rápidas del mundo corren programas escritos en MPI [6].

- Aplicaciones GPU: La gran arquitectura de *hardware* y el alto rendimiento de operaciones de memoria y de coma flotante en los GPUs los vuelven adecuados para muchas de las cargas de trabajos de científicos e ingenieros que se encuentran en *clusters* HPC, llevando a su incorporación como aceleradores HPC [7]. Empresas como NVIDIA brindan GPUs programables, junto con librerías y un SDK especializado para HPC que permiten crear aplicaciones útiles para resolver desafíos de alta exigencia [8].
- Big Data: se ha acuñado el término High Performance Data Analytics (HPDA) para representar la unión de HPC y la analítica de Big Data desplegadas en configuraciones de estilo HPC. Big Data surgió debido a la necesidad de procesar largos volúmenes de datos, generalmente a velocidades de red bajas, en sistemas locales y nodos con discos. Obtenemos grandes ventajas de rendimiento al correr cargas de trabajo de Big Data, por ejemplo, de una plataforma Hadoop en una infraestructura HPC [9].
- *Artificial intelligence* y *Machine Learning*: se usa HPC para la detección de fraude de tarjetas de crédito, que vehículos autónomos (sin conductor) aprendan a conducir por sí mismos, mejorar técnicas de detección de cáncer, etc. [3].

1.2.1 Entornos de despliegue de HPC

HPC es desplegado en distintos escenarios: *on-premise*, *at the edge* (es decir, cerca del origen de los datos) o en la nube. Se prevé que HPC sea desplegado cada vez más en entornos *multi-cloud* e híbridos (público y privado) [4].

Diversas nubes públicas ofrecen opciones para implementar una infraestructura HPC sobre la nube, por ejemplo AWS [10], Azure [11] u Oracle Cloud [12], prometiendo una “capacidad virtualmente ilimitada” [10].

A diferencia de sistemas HPC *on-premises*, correr aplicaciones HPC sobre arquitecturas cloud brinda flexibilidad y escalabilidad ante la demanda. Es decir, los recursos con los que se disponen en la infraestructura son provistos dinámicamente y del mismo modo, cuando se tenga otras cargas de trabajo que deben ser procesadas se remueven las capacidades de cómputo dinámicamente. De esta forma, se brinda la capacidad al administrador de proveer de la forma que crea conveniente los recursos para llevar a cabo los procesamientos requeridos.

Además, otra ventaja de correr HPC sobre cloud es que los tiempos de espera relacionados con *clusters* HPC on-premises, por ejemplo para la liberación y reasignación de recursos, son acortados [10], [11]. En la siguiente figura observamos una de las formas de construcción de correr cargas de trabajo HPC nativas en Azure.

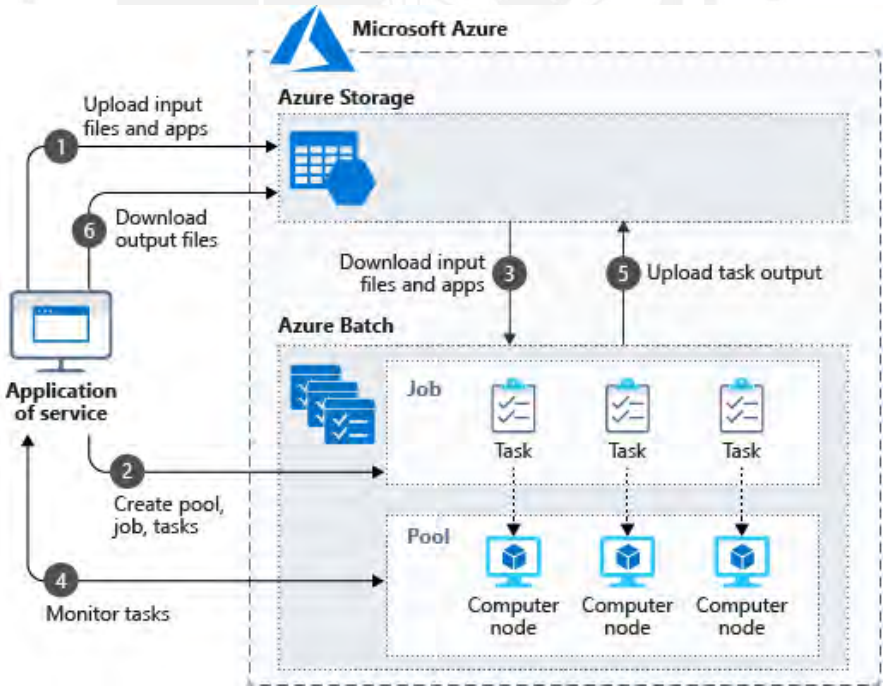

FIGURA 1-1: HPC *workloads* en Azure

FUENTE: [13]

Para cualquier tipo de nube sobre la que se haga el despliegue HPC, se debe considerar qué tipo de instancias conviene desplegar sobre los nodos de

cómputo, encargados de procesar la carga HPC. Entre las posibilidades existen las máquinas virtuales, *containers* y *bare-metal*.

Un sistema *cloud* que provee instancias de servidores *bare-metal* bajo demanda se diferencia de servidores *cloud* tradicionales, en cuanto que están libres de *overheads* de virtualización, propio de las instancias de máquinas virtuales. El uso de hipervisores como KVM o Xen, introducen estos *overheads* de rendimiento considerables. La eliminación de este *overhead* sería óptimo para aplicaciones HPC como se propone en [14], donde además se presenta un estudio de rendimiento y escalabilidad de servidores *cloud* basados en *bare metal*.

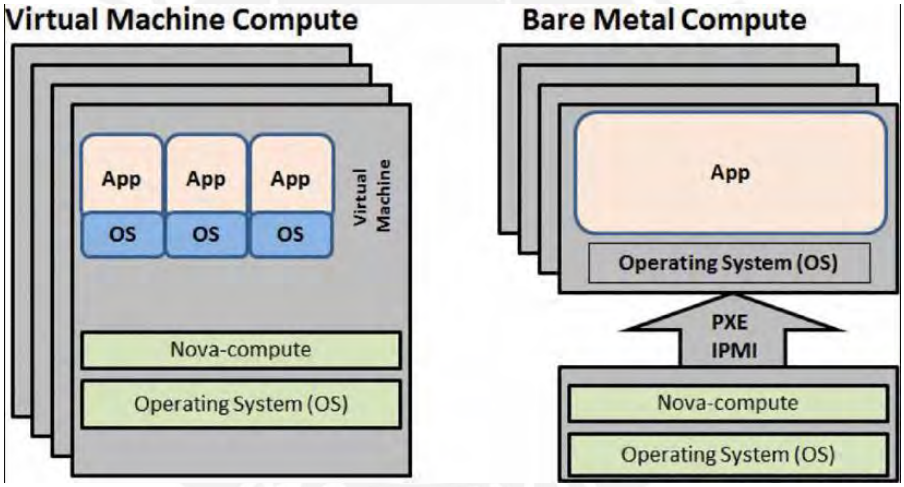

FIGURA 1-2: Bare metal vs Virtual Machine compute en OpenStack
FUENTE: [14]

OpenStack Ironic es el proyecto encargado de aprovisionar servidores *bare metal*, cuyo objetivo es otorgar a los usuarios finales un poder de procesamiento mayor de los recursos físicos. Sin perder los *features* propios de *cloud*, como la flexibilidad y alta disponibilidad, es posible deshacerse de la capa de virtualización. Ya que la premisa de usar los servidores *cloud bare metal* es dar la máxima potencia de procesamiento disponible al usuario, el servidor no puede compartirse entre varios *tenants*, es decir, se busca que nadie pueda interferir

con el rendimiento de las instancias. Por lo tanto, los servidores *bare-metal* son sistemas *single-tenant* [14].

1.2.2 Problemática del uso de HPC en cloud

Usar HPC en la nube provee varias ventajas mencionadas previamente, sin embargo, la forma tradicional de trabajo que tiene OpenStack, es decir, instancias con recursos virtualizados asignados por un hipervisor, generan un *overhead* perjudicial para las aplicaciones HPC.

El factor de latencia de mensajes de red es crítico para el rendimiento de aplicaciones HPC. Las aplicaciones sensibles a este factor poseen un bajo desempeño en entornos virtualizados convencionales. Las aplicaciones que dependen de parámetros como la velocidad de operaciones I/O verán un gran *overhead* en entornos completamente virtualizados, teniendo un impacto negativo. El *overhead* puede mitigarse mediante la paravirtualización, en la que un guest OS incluye soporte para correr dentro de un entorno virtualizado [15].

Además, los modelos tradicionales de despliegue de OpenStack incluyen *mechanism drivers* basados en agentes del tipo virtualizados los cuales implican la creación de equipos como Linux bridge y Open vSwitch, dentro de los nodos de cómputo.

En un entorno virtualizado, el tráfico de red de las VMs viaja a través dispositivos de red virtualizados a la infraestructura definida por *software* que corre encima del hipervisor. El procesamiento de paquetes en estos casos se vuelve mucho más complejo, que un despliegue HPC convencional. Debido a esto, cuando lo comparamos con un entorno *bare metal*, el rendimiento de *networking* puede ser menos eficiente y el de CPU más exigente en un entorno virtualizado [15].

Otro estudio más reciente como [16] menciona que la red tiende a limitar la escalabilidad y rendimiento de aplicaciones HPC, por lo cual se propone la

agregación de tarjetas de interfaz de red (NICs) con una nube usando Linux *containers*. Esto lleva a demostrar una mejora en términos de *throughput* y latencia de la red en la mayoría de aplicaciones HPC de distintos patrones de comportamiento. Los resultados concluyeron que para escenarios donde la infraestructura de red es escalable en términos de NICs, cables y puertos de switch, implementaciones como agregación de NICs para un entorno en la nube pueden brindar mejoras de rendimiento en escenarios HPC.

En trabajos relacionados como [17] se realizaron estudios del rendimiento de red usando CloudStack para desplegar nubes basadas en KVM y LXC. Se demostró que KVM logra tasas de *throughput* óptimas pero una degradación en el rendimiento de la latencia. Por otro lado, LXC presentó un mejor rendimiento en latencia. Con los resultados de *throughput* y latencia se indicó una alternativa para mejorar el rendimiento de red, consistente en el módulo *vhost-net*.

En [18] se toman en cuenta 2 optimizaciones investigadas que consisten en una virtualización ligera y afinidad de CPU. Se consideraron 2 técnicas para aligerar la virtualización:

- VMs ligeras configuradas con PCI *passthrough* para I/O.
- *Containers* como LXC, es decir, virtualización a nivel de sistema operativo. En este esquema se comparte la red física entre las instancias y el *host*. La exigencia es que sea sobre el sistema operativo, perdiendo un poco de flexibilidad.

Además, se realizó la comparación de 4 configuraciones: (i) máquinas físicas (*bare*), (ii) LXC *containers*, (iii) máquinas virtuales configuradas con emulación de red por defecto (*plain* VM) y (iv) VMs con *passthrough networking* (*thin* VM) con Input/Output Memory Management Unit (IOMMU) habilitado para proveer a las VMs acceso directo al hardware Ethernet. Las 2 últimas corriendo sobre un hipervisor KVM.

Los resultados de los estudios llevados a cabo en [18] confirman que la virtualización de la capa de red es el cuello de botella de la nube y que la virtualización ligera reduce el *overhead* de latencia de la virtualización al dar a las VMs un acceso directo a las interfaces de red. Además, se comprobó que las VMs ligeras y los contenedores presentan un *overhead* significativamente menor en la comunicación.

Sin importar el tipo de arquitectura desplegada para soportar cargas de trabajo HPC en OpenStack, ya sea HPC virtualizado en OpenStack (todos los componentes de HPC son virtualizados en OpenStack), *bare-metal* HPC en OpenStack (todos los componentes HPC se despliegan en servidores *bare-metal* usando OpenStack Ironic) o una arquitectura híbrida con *head nodes* virtualizados y *compute nodes bare-metal* [19]; debemos considerar el tipo de red por la cual pasará el tráfico de datos HPC, ya que, este será un factor importante para el desempeño de la solución.

En [19] se encuentran recomendaciones para la configuración de *networking* en OpenStack aplicado a cargas HPC:

- *Provider networks*: se emplean para relacionar las redes directamente a redes físicas existentes, creando una conexión a la infraestructura de switching L2 directa. Debido a esto, las redes *provider* no necesitar enrutar tráfico L3 usando el *control plane* de OpenStack, pues debe existir un *gateway* L3 en la topología de red del *data center*.
- *Single root input/output virtualization* (SR-IOV): se recomienda para cargas de trabajo HPC, pues permite a OpenStack extender las capacidades de NICs físicas directamente a las instancias conectadas, mediante el uso de SR-IOV NIC Virtual Functions (VF).

A continuación, se verá el impacto de la integración de SR-IOV en OpenStack, considerando los beneficios y desventajas relacionados.

1.2.3 Integración de SR-IOV en OpenStack para cargas de trabajo HPC

En un despliegue tradicional de Openstack, un puerto Neutron es un puerto virtual que va conectado a un *bridge* virtual (p.ej. Linux Bridge u Open vSwitch) dentro de un *compute node*. Al integrar SR-IOV *networking* a nuestro entorno OpenStack, podremos asociar el puerto Neutron a una Virtual Function (VF) que reside en el adaptador de red, dejando de necesitar un *virtual bridge* dentro del *compute node* [20].

Cada vez que un paquete entra al puerto físico de un adaptador de red, este es colocado dentro de un VF *pool* específico, basado en direcciones MAC o VLAN *tag* del paquete entrante. Ya que existe una transferencia de paquetes directo a la memoria hacia y desde la VM, se elimina la necesidad de usar el hipervisor para el movimiento de paquetes. Con esto obtenemos la ventaja de eliminar cuellos de botella en la virtualización de red [20].

La siguiente figura muestra el mecanismo de funcionamiento de SR-IOV relacionado con los guests, NIC virtuales, VFs, PF, hipervisor, y las extensiones de Intel o AMD.

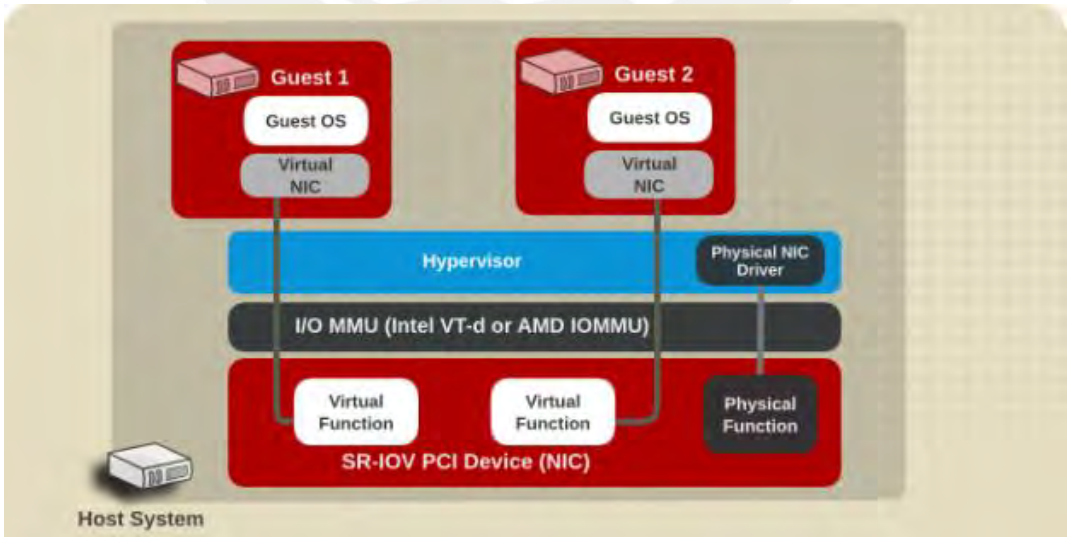

FIGURA 1-3: Funcionamiento de SR-IOV
FUENTE: [20]

Aún si usamos distintos esquemas en nuestra nube, es decir, instancias VM usando puertos SR-IOV e instancias VM con puertos regulares, conectados a *switches* virtuales, la comunicación entre ellas puede ser posible siempre que contemos con la configuración de las redes provider: *flat* o VLAN [20]. La documentación oficial de OpenStack en su versión Ussuri (2020-1) sobre SR-IOV [21] recomienda usar redes *provider* tipo VLAN para segmentación. De esta forma tendremos la capacidad de combinar instancias con y sin puertos tipo SR-IOV en una misma red.

A parte de SR-IOV se tienen diferentes esquemas para manejar las interfaces de red, con el fin de que la carga de trabajo de las instancias sea manejada de forma eficiente. En paravirtualización, un dispositivo de interfaz de red es creado en *software* y es diseñado para la interfaz de software más eficiente, en PCI *passthrough* un dispositivo físico es transferido exclusivamente desde el hipervisor a una VM *guest*. En SR-IOV, un dispositivo físico crea un número de funciones virtuales (VF), compartiendo los mismos recursos físicos. Las VFs pueden pasar a través del *hipervisor* hacia una VM *guest*. La siguiente imagen representa la comparación de tecnologías [15].

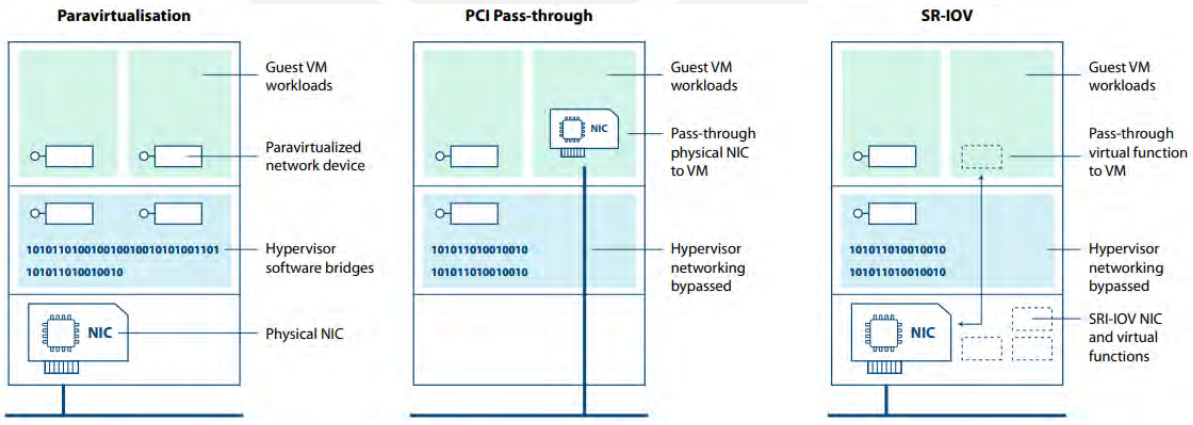

FIGURA 1-4: Comparación de Paravirtualización, PCI *passthrough* y SR-IOV

FUENTE: [15]

El beneficio de usar SR-IOV *networking* en OpenStack es que proveemos de un mayor rendimiento para las comunicaciones a través de la red de las instancias que corren sobre la infraestructura *cloud*. Gracias al baipás del hipervisor que lleva a cabo SR-IOV, el *delay* de los paquetes es menor y se obtiene una velocidad de comunicación cercana al *line-rate*. Sumado a que cada dispositivo virtual (VF) puede conectarse a una instancia de forma directa. Esto es lo que se busca para aplicaciones del tipo HPC, en donde a parte del procesamiento exigente, la comunicación entre instancias es importante para llevar a cabo tareas que se realizan en paralelo.

En [21] se listan las limitaciones conocidas de usar SR-IOV en OpenStack y que representan el *trade-off* de brindar una mayor velocidad en la comunicación de las instancias. Entre las limitantes encontramos:

- *Security groups* no son soportados al usar SR-IOV, por lo que el *firewall driver* debe deshabilitarse en la configuración de Neutron.
- Conectar puertos SR-IOV a servidores existentes previamente no es soportado.
- La función de Quality of Service (QoS), *burst* sobre max_kbps no es soportada.
- SR-IOV no se encuentra integrado en el OpenStack Dashboard, por lo que los usuarios deben usar la CLI o API para configurar interfaces SR-IOV.
- La capacidad de *live migration* ha sido agregada recientemente en la versión Train a Nova para instancias con puerto SR-IOV. Si bien la migración de interfaces SR-IOV de modo indirecto (macvtap o virtio-forwarder) son transparentes al *guest*, las interfaces SR-IOV de modo directo (*direct* o *direct-physical*) no son transparentes al *guest*, pues primero deben desconectarse y luego de la migración volverse a conectar.

Para poder desplegar SR-IOV en OpenStack, en [22] se da a conocer algunos requisitos para la habilitación de esta tecnología en los diferentes nodos de nuestra nube. El *compute node*, que puede correr un hipervisor como KVM,

necesita el soporte de SR-IOV en múltiples capas: BIOS, sistema operativo host y el adaptador físico de red. No hay necesidad de desplegar ningún tipo de *switch virtual* dentro de este nodo. Por otro lado, el *network node* puede seguir usando Open vSwitch para la conectividad a las redes de datos *tenant*, sobre todo si se requiere servir instancias con servicios de red como DHCP, L3 *routing* y NAT. En una topología simplificada de solo 2 nodos, un *compute node* y un *controller node*, tendríamos los elementos mostrados en la siguiente figura dentro de cada nodo.

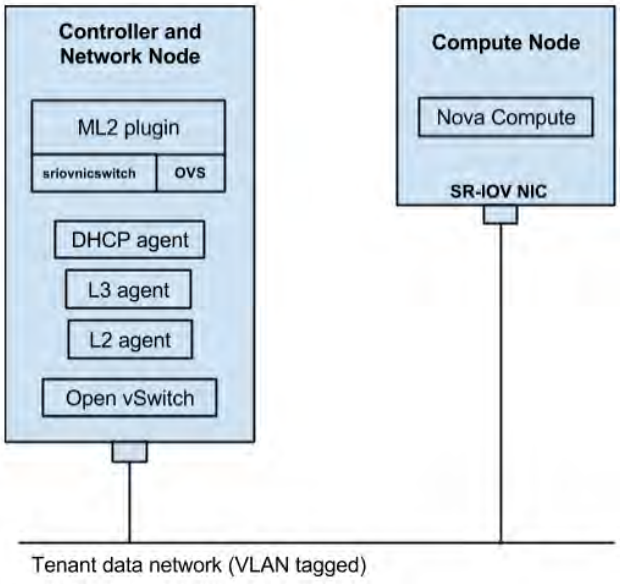

FIGURA 1-5: Comunicación entre un *controller/network node* y un *compute node* con capacidad SR-IOV
FUENTE: [22]

En [23] se establece que los *overheads* que introduce la virtualización se ven despreciables debido al baipás del hipervisor que lleva a cabo SR-IOV. En sí, las instancias de cómputo tienen un acceso directo a la red física subyacente mediante el uso de dispositivos VFs asignado a cada instancia. Este camino directo que se establece entre la instancia y la interfaz de red evita la implementación de *networking* de *software* que el hipervisor implementa.

> "Si bien este enfoque entrega altos niveles de rendimiento, también realiza el baipás de las reglas de *firewall* de *security groups* que

> OpenStack aplica a una instancia. Consecuentemente, SR-IOV *networking* solo debería usarse en redes internas (confiables) y debe configurarse en conjunto con configuraciones de red convencionales para administrar conectividad con redes desconfiables" [23].

En [15] se reafirma la idea de que SR-IOV genera una limitación en la infraestructura definida por *software*, ya que no es generalmente posible aplicar políticas de *security groups* a los puertos de instancias relacionados con una VF SR-IOV. Lo cual, puede generar problemas de seguridad para redes accesibles externamente. Sin embargo, esto no nos impide usar SR-IOV *networking* para una comunicación de gran rendimiento, especialmente para aplicaciones HPC, en la que la carga de trabajo paralela es procesada sobre OpenStack.

Además, usar SR-IOV para *networking* de alto rendimiento puede llegar a reducir el *overhead* de virtualización típicamente de 1-9% del rendimiento *bare metal* para cargas de trabajo HPC conectadas a la red *tenant* [15].

Como se menciona en [24], al mover cargas HPC a la nube se obtienen las ventajas propias del sistema, como facilidad de administración, rápido despliegue, aislamiento de rendimiento y compartición de recursos. A pesar de esto, la mayoría de aplicaciones HPC necesitan una conexión de red de baja latencia y alto *throughput*, características que no siempre se encuentran en una nube tradicional. La razón de esta necesidad es que la mayoría de este tipo de conexiones realizan un baipás del *kernel*. De darse el caso que no se provea soporte para interconexiones rápidas, muchas aplicaciones HPC se verán perjudicadas en cuanto al tiempo para finalizar las tareas. SR-IOV puede resolver estos problemas mediante la presentación de una Physical Function (PF) como múltiples Virtual Functions (VF) y el acceso de cada instancia a una de estas VF. SR-IOV promete lograr rendimientos I/O cercanos al nativo, representando una necesidad para aplicaciones HPC sobre la nube. Por ejemplo, los nodos de cómputo de Amazon EC2 especializados para este tipo de cargas pueden soportar tarjetas SR-IOV.

A pesar de las ventajas ya mencionadas, este trabajo tesis se centrará en una de los *trade-off* de integrar SR-IOV en nubes desplegadas con OpenStack, específicamente la pérdida del *feature* de *security groups* con el fin de lograr que exista un nivel de seguridad al correr cargas de trabajo HPC sobre OpenStack.

1.3 Orquestador IaaS HAST del GIRA

Hybrid Academic Scenario Testbed (HAST) es un orquestador con la capacidad de desplegar *clusters* HPC sobre OpenStack que responde a la necesidad de flexibilidad para la convivencia de múltiples soluciones HPC sobre una misma infraestructura que, por ejemplo, puede surgir en una universidad o centro de investigación. El objetivo es que las diversas cargas de trabajo HPC que puedan existir (por ejemplo, de OpenMPI más SLURM y Hadoop) puedan desplegarse sobre los mismos recursos físicos sin perder el rendimiento de ejecutar tareas en paralelo con alto rendimiento.

Las soluciones tradicionales de HPC suelen servir para un solo tipo de tarea en específico y presentan desventajas como la pérdida de rendimiento debido a la capa del hipervisor usada para la virtualización, generando una mayor latencia; la dificultad para relacionar los requerimientos de los usuarios a los recursos adecuados; la necesidad de una interfaz gráfica que simplifique la generación de clústeres; y la falta de un modelo PaaS personalizado que corra sobre la infraestructura IaaS.

HAST busca dar solución a todas las necesidades mencionadas previamente con la posibilidad de administrar una gran cantidad de recursos físicos como vCPUs, RAM y almacenamiento y de escalar según la necesidad. HAST aprovecha la API de OpenStack para el desarrollo de la solución y cubrir las necesidades del usuario. HAST tiene la capacidad de desplegar *slices* con diferentes niveles de complejidad, por ejemplo, servicios de soporte como

servidor LDAP, *firewall*, servidor NFS, servidor web y clústeres con servicios HPC como OpenMPI+SLURM y Hadoop.

Mediante una interfaz gráfica, se busca que el cliente no se preocupe en conocer la tecnología de *cloud computing* subyacente, sólo en la creación de sus escenarios de trabajo, a través de la personalización de parámetros como nombre del clúster, nivel de redundancia del nodo maestro, número de nodos de trabajo y la lista de usuarios del clúster. Si las plantillas con la cantidad de recursos recomendada no satisfacen los requerimientos del usuario, HAST da la opción de importar escenarios, así como imágenes de disco.

Se tiene planeado integrar la tecnología de SR-IOV al HAST con el fin de obtener una tasa de transferencia de mayor rendimiento y menor latencia que el escenario actual con las que cargas de trabajo HPC puedan beneficiarse, gracias el baipás del hipervisor y la eliminación del *overhead* asociado a la red. Sin embargo, como ya se explicó, emplear esta tecnología también simboliza una pérdida de características implementadas en el *kernel* tal como *security groups* (*iptables*).

1.4 Justificación

El presente trabajo de tesis surge ante la necesidad de una herramienta específica llamada HAST, con el objetivo de mejorar su rendimiento, dándole un mejor uso a la infraestructura sobre la cual se piense desplegar. De esta forma, las cargas de trabajo HPC que se ejecuten en las instancias de OpenStack no se perjudiquen por la latencia de la red que las conecta, sino más bien, impedir que el cuello de botella sea la comunicación para el tráfico de datos.

Si bien se están tomando en cuenta los requerimientos y necesidades generados al integrar la tecnología SR-IOV al HAST, la solución que se plantea más adelante puede emplearse para otro tipo de proyectos, en especial aquellos que

requieran brindar un cierto nivel de seguridad a las instancias cuando no se cuenta con mecanismo tradicionales como security groups en OpenStack.

En [25], por ejemplo, nos señalan que debido a la adopción de SR-IOV como la solución I/O de mejor rendimiento, soluciones como Amazon EC2 han adoptado esta alternativa. Los efectos negativos que trae consigo la implementación de SR-IOV están sobre todo relacionados con la seguridad. Por ejemplo, una instancia VM puede explotar el acceso directo a un dispositivo SR-IOV mediante la realización de un ataque de *flooding* con paquetes PCIe. Esto desencadenaría una congestión en la interconexión PCIe, lo cual resultaría en problemas serios de rendimiento entre la VM atacante, las VMs conectadas a la misma red de datos y el *host*, inclusive.

Tal como el ataque de denegación de servicio (DoS), las instancias se encuentran expuestas a otras vulnerabilidades que podrían llegar a ser explotadas por usuarios maliciosos si es que no se implementa ningún tipo de seguridad. Las opciones para mitigar estos ataques no son variadas o solamente se restringen a tipos de ataques específicos.

Por ejemplo, en [26] se proponen 2 arquitecturas de *hardware* integrado que previenen completamente ataques DoS, tomando como base las extensiones de Quality-of-Service (QoS) de la especificación PCIe. Primero se determina qué extensiones QoS son útiles y se muestran cómo los vCPUs deben implementarlas e interconectarse para protegerse de DoS. Este mecanismo de QoS PCIe para prevenir DoS SR-IOV puede prevenir este tipo de ataques de forma correcta.

Mientras que en [25] la propuesta de mitigación de ataques DoS se afronta desde 2 perspectivas: en el lado del *hardware*, se proponen extensiones de monitoreo dentro de dispositivos SR-IOV que distinguen un uso permitido del dispositivo de un uso malicioso, mediante el análisis de la tasa de transacciones PCIe entrantes a nivel de VM. Aquellas VMs identificadas como maliciosas son reportadas al

host mediante interrupciones. Del lado del *software*, se mitigan los efectos de interferencia del rendimiento al ajustar dinámicamente la planificación del *host* de la VM maliciosa o apagándola. Se implementó un prototipo con un controlador Ethernet SR-IOV comercial y una placa FPGA. Finalmente se demostró que un *scheduling* apropiado de VMs maliciosas mitiga los efectos de interferencia.

Este trabajo de tesis busca dar otro tipo de solución enfocada más al *software* y tomando en cuenta un tipo de carga de trabajo específico (HPC), por tanto, un conjunto de reglas delimitadas al tráfico de red necesario para el desarrollo apropiado de las tareas. Con este fin, se propone el uso de un controlador SDN/OpenFlow que se encargue de manejar el tráfico entre instancias de distintos *compute nodes* de OpenStack. Mediante el protocolo OpenFlow se configurarán *flows* dentro del *switch* ToR (Top of Rack), según los requerimientos del usuario, basado en las cargas de trabajo HPC que piense correr sobre las VMs.

El motivo de la elección de la alternativa recién mencionada es que se con ella se puede lograr dar un balance entre seguridad y rendimiento con el que un escenario tradicional no cuenta. Además, se obtienen las ventajas que nos brinda el uso del paradigma SDN como el control programable mediante software y de forma centralizada a través de un controlador SDN.

1.5 Relevancia

El presente trabajo de tesis trata conceptos de redes, software y ciberseguridad. La problemática genera la necesidad de relacionar 2 conceptos; *cloud computing* y *software-defined networking*, que vienen siendo desarrollados en muchos escenarios actuales, donde la flexibilidad y administración centralizada son buscados, pero que no suelen tratarse de manera conjunta en entornos de red tradicionales.

De manera específica, en este trabajo se trata con cargas de trabajo HPC que corren sobre instancias VM desplegadas con OpenStack y donde se busca desplegar SR-IOV para mejorar el rendimiento y latencia de la comunicación de red entre instancias de distintos nodos de cómputo. La comprensión de estas tecnologías es necesaria para poder integrar un nivel de seguridad con el cual no se cuenta en un despliegue predeterminado que integra OpenStack con SR-IOV.
Este proyecto de tesis busca dar un nuevo enfoque de las redes a través de la integración de tecnologías crecientes como SDN y *cloud computing* relacionados a cargas de trabajo HPC.

Los beneficios económicos que representan este tipo de soluciones para las empresas es múltiple, junto con otras tecnologías que definen una nueva forma de diseñar, desplegar y administrar las redes y sus servicios, tales como Network Virtualization (NV), Network Function Virtualization (NFV) y White Boxes.

Equipos de red tradicionales son sobrevaluados debido a la licencia de sistema operativo que funciona exclusivamente con el *hardware* adquirido. A través de SDN se puede obtener equipos más económicos conocidos como *white boxes* que no poseen un sistema propietario, sino que se basan en *chipsets* genéricos con compatibilidad de sistemas operativos de red de código abierto tal como Open Network Linux (ONL). Además, un control centralizado reduce los costos de operación en las empresas.

Desplegar aplicaciones HPC sobre *bare-metal* podría representar la sub-utilización de los recursos físicos de la infraestructura, como los recursos de cómputo, ya que, solo podría correr un mismo tipo de carga. Mientras que un despliegue de aplicaciones HPC en la nube aprovecha la flexibilidad de estos entornos y permite desplegar escenarios HPC incompatibles entre sí. De esta forma incrementamos la utilización de los recursos físicos, lo cual finalmente se traduce en beneficios económicos.

A través de una solución cloud la reasignación de recursos se facilita, lo cual permite dar prioridad a ciertas cargas que se consideren de mayor importancia. En los casos que se necesite, se podrá remover vCPUs, RAM o disco de un *slice* de OpenStack y reasignarlos a otro *slice* al que se le ha colocado una prioridad mayor. Mientras que en una solución *bare-metal* se deberá reasignar el total de los recursos a la nueva tarea o comprar más equipos que puedan dejar ejecutar ambas tareas en escenarios aislados, y luego de la culminación pudiendo dejar una infraestructura subempleada.

Los puntos que hacen rentable la implementación de estas tecnologías repercuten en la sociedad, pues precios más rentables, ayudan a que empresas emergentes potencien su despliegue de redes y lo hagan con solvencia económica.

1.6 Objetivo de la tesis

Se establece que el objetivo de la presente tesis es el desarrollo de un módulo de ciberseguridad para el orquestador IaaS HAST del GIRA que permita instalar reglas de seguridad para los slices HPC creados por esta herramienta para reemplazar *security groups*.

El módulo desarrollado debe configurar el *switch* ToR del *rack* que conecta las instancias de distintos nodos de cómputo creadas por OpenStack, a través del protocolo SDN/OpenFlow, mediante *flows* que implementen las reglas de seguridad y control de acceso de interés. Para lograr esto, el módulo se comunicará usando la interfaz REST (Northbound API) provista por el controlador SDN que se encuentre administrando el *switch* ToR y lo configurará. De esta manera, se buscará recuperar las características que suelen ser implementadas en el *kernel* del hipervisor y se perdieron al integrar la tecnología de SR-IOV a OpenStack.

1.7 Objetivos específicos de la tesis

El desarrollo del módulo de ciberseguridad implica:

- Determinar qué reglas de seguridad son apropiadas para entornos HPC manteniendo un balance entre seguridad y rendimiento.
- Establecer la relación de los requerimientos del usuario a reglas que puedan implementarse mediante el uso de OpenFlow.
- Diseñar el módulo de ciberseguridad que permita instalar reglas de seguridad para los *slices* HPC.
- Realizar pruebas de validación que confirmen el funcionamiento de la solución, a través de escenarios virtualizados.
- Comparar el rendimiento de redes legacy versus redes implementadas mediante SR-IOV.

2. Tecnologías de computación en la nube y redes definidas por software

En este capítulo se detallan las tecnologías involucradas en la integración de SR-IOV en un entorno *cloud* sobre el cual corren cargas de trabajo HPC y la aplicación del concepto SDN/OpenFlow para recuperar el *feature* de *security groups* de OpenStack perdido al habilitar SR-IOV.

2.1 Tecnologías PCI passthrough y SR-IOV

PCI *passthrough* y SR-IOV son 2 tecnologías que permiten asignar una tarjeta de red física (un tipo de dispositivo PCI) a una VM. Mientras que SR-IOV permite compartir un solo adaptador de red físico (en el *host*) a múltiples VMs *guest*, PCI *passthrough* es la tecnología *legacy* y predecesora de SR-IOV con la que el adaptador físico es dedicado completamente a una sola VM *guest* [27].

2.1.1 PCI passthrough

Peripheral Component Interconnect (PCI) *passthrough* es una tecnología que permite a dispositivos PCI, como interfaces de red, parecer como si estuviesen físicamente conectados al sistema operativo *guest*, realizando un baipás del *hipervisor* (por ejemplo, KVM), que reside en el *kernel* (Ring 0 de Protection Rings). Esto, con el objetivo de obtener una tasa de transferencia de datos más elevada [28].

Un pre-requisito para que PCI p*assthrough* asigne un dispositivo PCI a un sistema operativo *guest* es que el hipervisor soporte una de las siguientes 2 extensiones: VT-d (de Intel) o IOMMU (de AMD) [20].

PCI *passthrough* permite a los *guests* tener un acceso exclusivo a los dispositivos PCI para una diversidad de tareas, dejando de estar disponible para el *host*. Los dispositivos PCI aparecerán y se comportarán como si estuvieran físicamente conectados a los *guest* OS. Específicamente, para *networking* podemos dedicar un dispositivo de red completo, es decir, un puerto físico de un adaptador de red a un *guest* OS corriendo en una VM [27].

2.1.2 SR-IOV

Single-root I/O virtualization (SR-IOV) es un estándar para un tipo de PCI *passthrough* que permite a una NIC física presentarse como múltiples vNICs o virtual functions (VFs) a las que una máquina virtual se puede adjuntar. SR-IOV puede combinarse con otras tecnologías de virtualización, como Intel VT-d, para mejorar el rendimiento I/O de una máquina virtual (VM). Con SR-IOV, cada VM puede tener acceso directo a paquetes encolados para las VFs conectadas a la VM. Para situaciones en las que se requiera un rendimiento que se aproxime al de interfaces *bare metal* físicas podemos emplear SR-IOV [28].

Las interfaces de red físicas que soportan SR-IOV pueden ser conectadas a VMs usando PCI *passthrough*. Con PCI *passthrough*, SR-IOV permite que una

función *root* (por ejemplo, un puerto Ethernet) sea visto como múltiples dispositivos físicos. Un dispositivo físico con capacidades SR-IOV se puede configurar para que aparezca en el espacio de configuración PCI como múltiples funciones virtuales (VFs) [28].

SR-IOV usa dos funciones PCI [29]:

- Physical Functions (PFs): son dispositivos *full* PCIe (PCI Express) que incluyen capacidades SR-IOV. Las funciones físicas son descubiertas, administradas y configuradas como dispositivos PCI normales. PFs configuran y administran la funcionalidad SR-IOV mediante la asignación de Virtual Functions.
- Virtual Functions (VFs): son funciones PCIe simples que solo procesan I/O. Cada Virtual Function es derivada de un PF. El número de VFs que un dispositivo puede tener está limitada por el *hardware* del dispositivo. Un único puerto Ethernet, el dispositivo físico, podría ser mapeado a varias VFs que pueden ser compartidas a *guests*.

Gracias a SR-IOV, dispositivos compatibles con esta tecnología pueden compartir un solo puerto físico con múltiples *guests*. VFs tienen un rendimiento cercano al nativo y proveen mejor rendimiento que *drivers* para-virtualizados y un acceso emulado. Además, VFs brindan protección de datos entre *guests* en el mismo servidor físico, ya que, los datos son administrados y controlados por el mismo hardware. Sin embargo SR-IOV no presenta *features* como live migration, pues con PCI *passthrough*, se necesita configuraciones de dispositivos idénticas para migraciones *live* u *offline*, Sin configuraciones de dispositivo idénticas, los *guests* no podrán acceder a los dispositivos *passed-through* luego de migrar [29].

Asignar una VF a una instancia VM permite que el tráfico que viaja haga un baipás de la capa de *software* del hipervisor y comunique directamente la VF y la VM. Debido a que la lógica para las operaciones I/O reside en el adaptador

mismo y las VMs piensan que están interactuando cada una con un dispositivo de red separado, podemos obtener un rendimiento cercano al *line-rate*. Esto nos puede ahorrar la necesidad de dedicar una NIC física separada para cada VM [20].

El baipás del hipervisor comprende también a los *switches* de *software* que se emplean generalmente en entornos de virtualización, como OvS y Linux bridges. Ahora, es el adaptador físico de red quien se responsabiliza de administrar los flujos de red, esto es, la separación y *bridging* adecuado. Por este motivo, el adaptador de red con capacidad SR-IOV debe dar soporte para implementar una forma de *hardware-base* Virtual Ethernet Bridge (VEB) [20].

La siguiente imagen muestra la arquitectura del *passthrough* del hipervisor que SR-IOV lleva a cabo, representando el mapeo de una VF a un sistema *guest*.

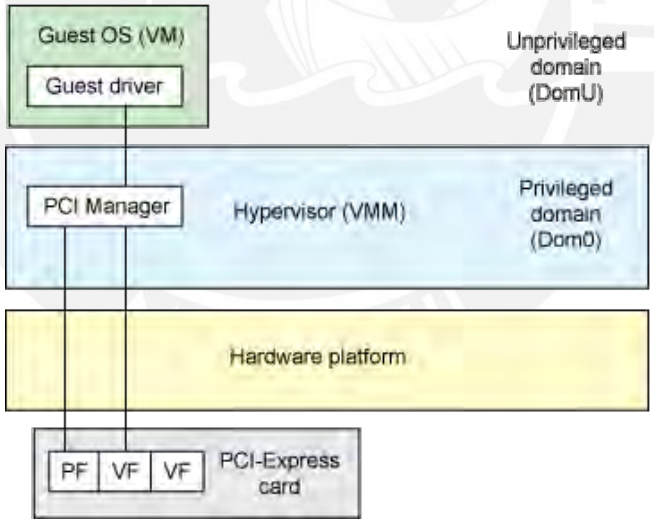

FIGURA 2-1: P*assthrough* con SR-IOV
FUENTE: [30]

Los hipervisores deben tener compatibilidad para este tipo de tecnología. Para los adaptadores Ethernet Intel, los hipervisores con soporte para SR-IOV son [31]:

- Microsoft Hyper-V
- VMware vSphere

- KVM (Kernel-based Virtual Machine)

2.1.3 NIC Intel 82599 y Drivers

El presente trabajo de tesis usará un modelo específico de NIC con capacidad SR-IOV con el fin de delimitar las funciones del diseño que se hará posteriormente y con la cual se cuenta para hacer las pruebas que se consideren pertinentes. El modelo de la NIC es Intel 82599 10 Gigabit Ethernet Controller.

El documento “Intel 82599 SR-IOV Driver Companion Guide” [32] presenta una guía para el manejo de los drivers de la NIC 82599. A continuación, se explicarán puntos clave que explican el funcionamiento de estas tarjetas de red para habilitar la capacidad de SR-IOV.

SR-IOV define un mecanismo estandarizado para crear dispositivos de red compartidos nativamente. Mediante este mecanismo, una Single Root Function (p.ej., un puerto Ethernet) puede aparecer como múltiples dispositivos físicos separados. El objetivo de SR-IOV es estandarizar la forma en que se comparte un dispositivo I/O en un entorno virtualizado. Esto se puede lograr con el baipás del hipervisor en la transferencia de datos y es el propio hipervisor quien asigna una o más VFs a una VM [32].

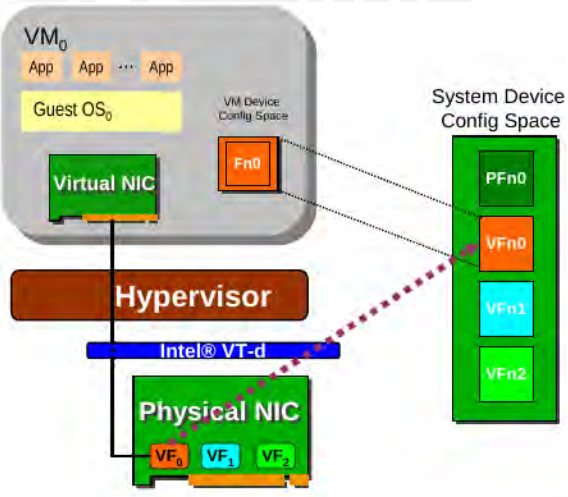

FIGURA 2-2: Asignación de VFs

FUENTE: [32]

2.1.3.1 Elementos de una VF

Los paquetes que entran a través de la NIC física serán divididos en colas, agrupándolos en pares de colas (1 cola de TX y 1 cola de RX). A su vez, se agruparán pares de colas RX/TX en un grupo de pares asignadas a la misma VF, llamados pools. En un Intel 82599 podemos tener hasta 64 *pools*, en las que se tienen 2 pares de colas por *pool* (2 colas de TX y RX). En la siguiente tabla se muestran las equivalencias de colas y *pools* para distintas configuraciones, tomando en cuenta que en modo SR-IOV pueden existir 16, 32 o 64 *pools* [32]:

Número de pools	Par de colas por Pool	Total de colas por Pool
16	8	16
32	4	8
64	2	4

TABLA 2-1: proporciones de *colas/pools*

FUENTE: [32]

Los datos entrantes se colocan en un *pool* determinado basados en los filtros de dirección MAC y VLAN *tag*, previo a la asignación de direcciones MAC y VLAN *tags* de los *pools*, que pueden ser compartidas. Este clasificador que distribuye los paquetes según MAC destino o VLAN *tag*, aplicando los filtros y asignándoles al *pool* correspondiente puede representarse como un *switch* L2. Cada VF tendrá asignado recursos dedicados. La siguiente imagen muestra la relación de las VFs con los componentes nombrados [32]:

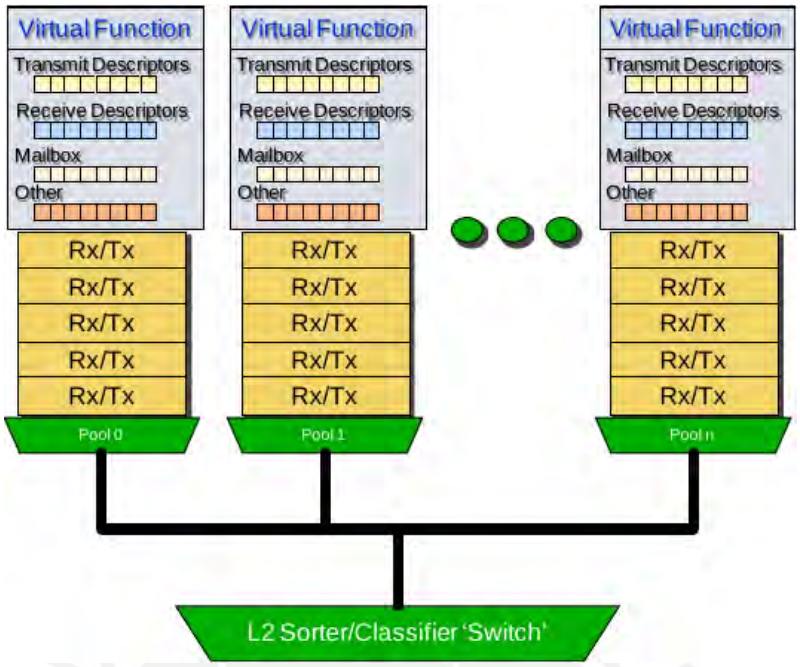

FIGURA 2-3: VFs, pools y clasificador L2

FUENTE: [32]

En la arquitectura de trabajo existen varios componentes que trabajan juntos para lograr la funcionalidad de SR-IOV en esta tarjeta de red. En la siguiente figura podemos ver algunos de estos elementos, tales como el VF Driver que reside en el *guest*, el PF Driver que reside en el *kernel space* del host y asigna las VFs a las VMs y el mecanismo de *mailbox* que sirve para habilitar la comunicación entre estos 2 tipos de *drivers*. A continuación, pasarán a desarrollarse.

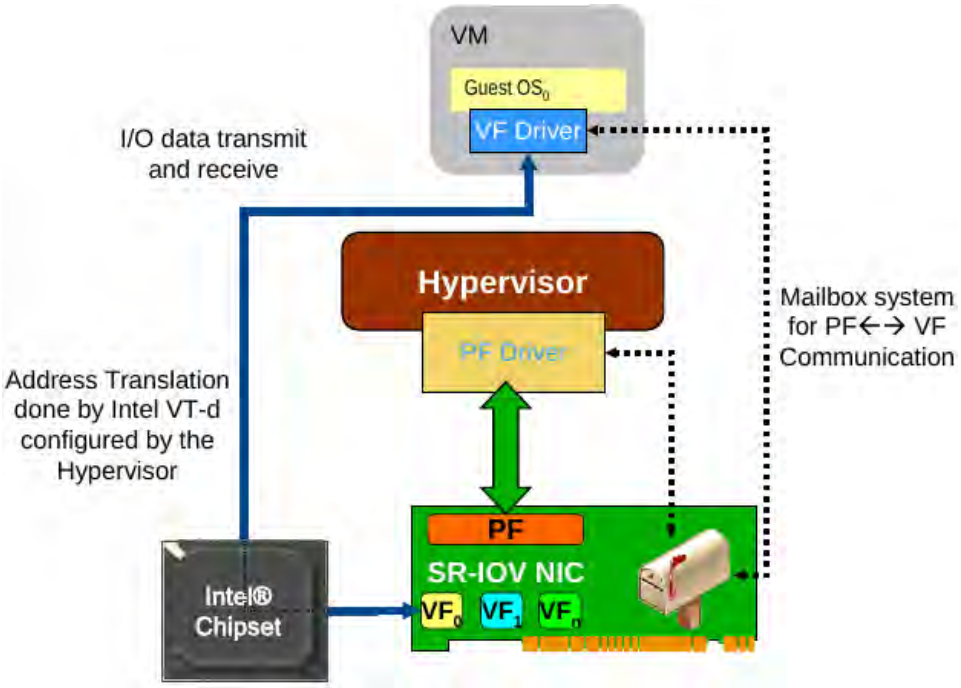

FIGURA 2-4: Componentes habilitadores de SR-IOV en Intel

FUENTE: [32]

2.1.3.2 Virtual Function Driver

El VF Driver no está al tanto del entorno virtualizado. En un entorno virtualizado, el driver se comunicará con una capa de *software* intermediario que se encarga de abstraer y emular el dispositivo físico que se encuentra debajo. El Intel VF Driver puede dividirse en 3 partes: (i) la interfaz OS - API expuesta al sistema operativo de la VM, (ii) operaciones I/O – usa las capacidades SR-IOV para I/O sin intervención del hipervisor, (iii) tareas de configuración – acciones que requieren la comunicación con el PF Driver (p.ej., VLAN *filtering*) [32].

El VF Driver usa el sistema Mailbox para pasar un mensaje al PF Driver, cuando los recursos PCI expuestos por el VF no puedan cumplir con las tareas requeridas. Entre las tareas que podemos realizar usando el mecanismo *mailbox* están [32]:

- Configurar la dirección MAC de la VF: el VF *driver* envía este mensaje cuando desea definir su propia dirección MAC, en lugar de la MAC que el PF define durante la iniciación de la VF.

- Configurar un filtro VLAN: el VF *driver* envía este mensaje cuando desea configurar un VLAN ID *Tag* para filtrar los paquetes entrantes. Cuando el PF *driver* configura el filtrado VLAN apropiado para la VF, también configura el VLAN *stripping* correspondiente. La inserción del VLAN *tag*, llevada a cabo por el VF Driver se realiza cuando se crea un paquete para transmitir.
- Resetear la VF
- Configurar una dirección *multicast*
- Configurar un tamaño máximo de paquete largo.

2.1.3.3 Physical Function Driver

El PF Driver se encarga de configurar los recursos compartidos y debe correr en un entorno más privilegiado que un *driver* de una VM. Su responsabilidad es proveer acceso a los recursos I/O del hipervisor y realizar operaciones que tengan impacto en todo el dispositivo. Este *driver* debe ser cargado previo a la carga de cualquier VF Driver. Además, contiene las capacidades de un Intel Ethernet Controller Driver tradicional, por ejemplo, VLAN *filtering*. Entre las funciones relacionadas al uso de SR-IOV el *driver* cuenta con las siguientes [32]:

- Generación de direcciones MAC para VFs
- Comunicación con el VF Driver a través del sistema Mailbox con el fin de realizar:
    - Configuración de VLAN *filter* (por el VF Driver)
    - Configuración de dirección Multicast (por el VF Driver)
    - Configuración del tamaño máximo de paquetes largos (por el VF Driver)
    - Reinicio de recursos de la VF

En un entorno virtualizado, la comunicación entre instancias se realiza a través de un *switch* virtual basado en *software* en el hipervisor. Los paquetes que van siendo transmitidos desde una VM son examinados para ver si la dirección MAC de destino coincide con otra VM corriendo en el sistema. Si hay un *match*, el

hipervisor no envía físicamente el paquete a través del ethernet controller, sino que lo envía a la VM destino a través del *switch* virtual. Ya que uno de los objetivos de SR-IOV es realizar el baipás del hipervisor, el "*switch*" interno de la NIC tiene la capacidad de replicar esta funcionalidad en *hardware* sin necesidad de un hipervisor y es nombrada *pool to pool* (o VF *to* VF) *bridging*. Esto se habilita desde el PF Driver [32].

El PF Driver se encarga además de quitar los VLAN Tags (VLAN *tag stripping*) para paquetes entrantes luego de pasar el mecanismo de L2 *filtering*. Además, se encarga de asignar dinámicamente a cada VF una dirección MAC. La gran mayoría de mensajes del PF Driver son mensajes de respuesta al VF driver mediante el sistema Mailbox (p.ej. configuración de dirección MAC y configuración de filtro VLAN) [32].

Contamos con otros *features* como *anti-spoofing* de dirección MAC y VLAN *tag*, y pueden habilitarse por cada VF desde el PF Driver. Detalles de los dos mecanismos de *anti-spoofing*: (i) dirección MAC: la dirección MAC origen de cada paquete saliente se compara a la dirección MAC que la VM transmisora usa para recepción de paquetes. Un paquete con una dirección MAC origen no coincidente es descartada. (ii) VLAN *tag*: se valida que la VF transmisora sea un miembro de la VLAN configurada en el paquete [32].

2.1.3.4 Sistema de comunicación Mailbox

Ya que las VFs son componentes limitados en cuanto a funcionalidad, requieren del PF para realizar una acción que no es provista dentro de los recursos PCI expuestos. Con este fin, debe darse una comunicación entre VF Driver y el PF Driver. Un ejemplo de estas peticiones que el VF Driver realiza al PF Driver es para aplicar filtros VLAN [32].

2.2 OpenStack

OpenStack es un sistema operativo *cloud* que controla múltiples *pools* de cómputo, almacenamiento, y recursos de red en un *data center*, todos estos administrados y aprovisionados mediante APIs con mecanismos comunes de autenticación. OpenStack provee principalmente funcionalidades de infraestructura como servicio (IaaS), sin embargo, también puede encargarse de la orquestación, administración de fallos, administración de servicios, así como otras funciones con capacidad de asegurar la alta disponibilidad de las aplicaciones de usuarios [33].

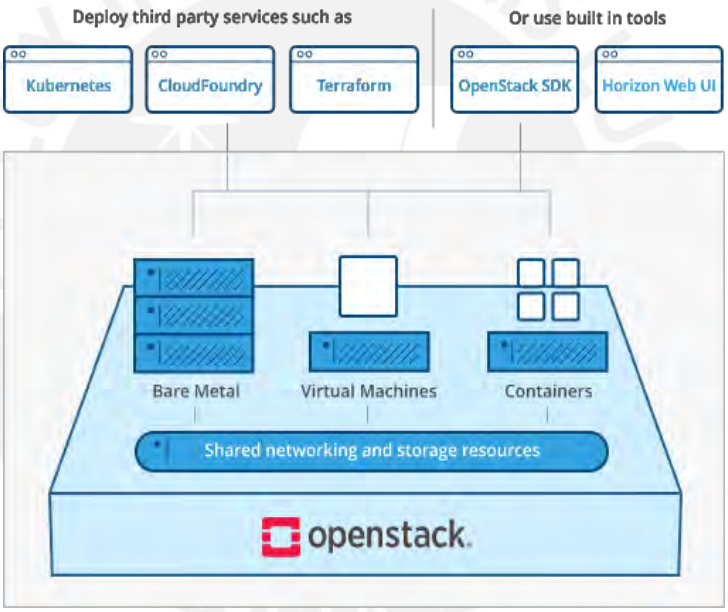

FIGURA 2-5: Esquema general de OpenStack

FUENTE: [33]

OpenStack está dividido en servicios con la finalidad de conectar componentes según las necesidades. En sí, OpenStack se conforma de varios comandos agrupados en scripts, los cuales juntos conforman “proyectos” de OpenStack que realizan tareas específicas en entornos de *cloud*. OpenStack depende además de virtualización y un sistema operativo base [34].

2.2.1 Arquitectura de OpenStack

La arquitectura de OpenStack se conforma de distintos proyectos [35], pero son principalmente seis los que habilitan desplegar una infraestructura base que permita aumentar otros servicios de mayor particularidad. Para manejar los procesos de informática, red, almacenamiento, identidad e imágenes los proyectos principales son [34]:

- Nova: herramienta para la gestión y acceso a recursos de cómputo de OpenStack, controlando la planificación, creación y eliminación de instancias.
- Neutron: encargado de la conexión de las redes de los proyectos.
- Keystone: se encarga de autenticar y autorizar todos los servicios que despleguemos en OpenStack. Además de servir como catálogo de servicios.
- Glance: sirve para almacenar y recuperar imágenes de disco de las máquinas virtuales.
- Cinder: brinda almacenamiento persistente de bloques.
- Swift: proporciona almacenamiento de objetos con alta tolerancia a errores.

2.2.2 Neutron

Neutron es el nombre del proyecto para OpenStack Networking que provee tecnologías de red que solemos encontrar dentro de *data centers*: *switching*, *routing*, NAT, *load balancing*, *firewalling*, VPN, entre otros. Neutron expone una API a los usuarios y pasa sus solicitudes a los *plugins* de red que se encuentra configurados para procesarlas. Tal como otros proyectos de OpenStack, se conecta a base de datos y se comunica con agentes a través de una mensajería de colas [36].

2.2.2.1 Switching

OpenStack emplea *switches* virtuales, basados en *software*, para conectar VMs a redes virtuales en L2 (capa de enlace de datos del modelo OSI). OpenStack soporta los siguientes *switches* virtuales [36]:

- Linux Bridge: provisto por el módulo de *kernel bridge* de Linux. Interfaz virtual que conecta múltiples interfaces de red. En Neutron, el *bridge* usualmente incluirá una interfaz física y una o más interfaces *tap* virtuales.
- Open vSwitch (OvS): switch virtual multicapa diseñado para habilitar la automatización de red a través de extensiones. Tiene *features* de seguridad (aislamiento VLAN, filtrado de tráfico), monitoreo (Netflow, sFlow, SPAN, RSPAN), QoS (*traffic queuing* y *traffic shapping*) y control automatizado (OpenFlow, OVSDB *management protocol*) [37].

Podemos configurar VLAN *tagging* o redes *overlay* en *software* con protocolos L2-in-L3 *tunneling*, como GRE o VXLAN.

2.2.2.2 Routing

Neutron provee capacidades de *routing* y NAT a través del uso de IP *forwarding*, *iptables* y *network namespaces*. Dentro de estos últimos podemos encontrar *sockets*, puertos *bound*, e interfaces virtuales, cada cual con su propia tabla de enrutamiento. Además, da la posibilidad de la superposición de subredes y la configuración de un *router* para que las instancias interactúen y se comuniquen con redes externas [36].

2.2.2.3 Tipos de redes en Neutron

En OpenStack se pueden definir 2 tipos de redes [36]:

- **Provider network**: red virtual que mapea una red física directamente. Es creada para habilitar acceso a recursos físicos fuera de la nube (p.ej. redes fuera de la nube) y está usualmente mapeada a una VLAN. Se

puede brindar acceso a uno o múltiples proyectos para acceder a este tipo de redes.

- **Project/tenant/self-service network**: red virtual creada por un proyecto (o por un administrador a nombre de un proyecto) para que las instancias accedan a los recursos dentro de esta. Los detalles físicos de la red no son expuestos al proyecto.

La diferencia entre una red *provider* y una red *project* puede verse durante el proceso de aprovisionamiento (creación) de la red [36]:

- **Provider network**: el administrador es el encargado de crear (y administrar) estas redes y puede hacerlo para un proyecto particular o que sea una red compartida entre proyectos. Además, al momento de la creación se pueden configurar detalles como el tipo de red, la interfaz física, el identificador de segmentación de red (VLAN ID o VXLAN VNI).
- **Project/tenant/self-service network**: a diferencia de una red *provider*, una red *project* no puede ser compartida con otros proyectos y la configuración de atributos es más limitada. Si bien las redes *project* tienen los mismos atributos que las redes *provider*, el usuario no tiene los permisos para especificarlos, Neutron los define automáticamente.

Cualquier red creada con Neutron, ya sea por un usuario regular o un usuario con rol *admin*, posee atributos provider que definen esa red, entre los que se identifican: (i) tipo de red (p.ej. flat, VLAN, GRE, VXLAN o local), (ii) interfaz física de red por donde atraviesa el tráfico e (iii) ID de segmentación de la red (p.ej. VLAN ID o VXLAN VNI). La diferencia entre una red *provider* y una red *tenant* consiste en quién o qué configura esos atributos y cómo son administrados dentro de OpenStack [36].

Las redes *provider* sólo pueden crearse y administrarse por un administrador de OpenStack (rol *admin* en Keystone) ya que se requiere conocimiento y configuración de la infraestructura de red física. Cuando una red *provider* es

configurada para que funcione como una red externa de los *routers* de Neutron, la red *provider* es conocida como una red *external provider*. Usualmente las redes *provider* se configuran como redes *flat* o *vlan* y usan un dispositivo de enrutamiento externo para comunicarse con el exterior de la nube [36].

Las redes *self-service* son creadas por usuarios y suelen aislarse de otras nubes en la red. Ya que no se puede configurar la infraestructura física para este tipo de red ni definir los atributos provider manualmente, los *tenants* deberán conectar sus redes a *routers* de Neutron cuando se necesite conectividad externa. Los atributos de la red *self-service* son pre-definidos por el administrador en los archivos de configuración de Neutron [36].

A pesar de que una red provider está típicamente asociada a una red física mediante una VLAN (o *flat*) en un *data center*, esto no es un requerimiento. Es posible crear una red *provider* con un protocolo *overlay* (GRE/VXLAN). Del mismo modo se pueden crear redes *tenant* usando tecnologías *underlay* (VLAN) u *overlay*. Por lo tanto, la selección de una tecnología *underlay* u *overlay* no influye en si se puede utilizar una red *project* o una red *provider* [38].

2.2.2.4 Otros recursos de red de Neutron

Otros recursos de importancia para el funcionamiento de *networking* en OpenStack son [36]:

- Subred: bloque de direcciones IP usadas para asignarlas a puertos creados en la red.
- Puerto: punto de conexión de un único dispositivo, como la tarjeta de interfaz de red virtual (vNIC) de la instancia a la red virtual. Un puerto cuenta con una dirección MAC y una dirección IP fija de la subred.
- Router: dispositivo virtual para proveer enrutamiento entre redes *provider* y *tenant*.
- Security group: conjunto de reglas de *firewall* virtual que se encargan de controlar el tráfico que ingresa y sale a través del puerto de una instancia.

Una interfaz de red virtual, denominada *tap* interface, se crea en el *compute node* donde se encuentra la instancia. Esta interfaz *tap* tiene una correspondencia directa con una interfaz de red dentro de la instancia *guest* y posee las propiedades del puerto creado en Neutron: direcciones MAC e IP. Neutron habilita la opción de especificar alternativas a la interfaz *tap* estándar tales como Macvtap y SR-IOV [36].

2.2.2.5 ML2 Plugin

Neutron provee una arquitectura basada en *plugins* y *drivers* desarrollados por la comunidad de OpenStack y por terceros. A través de estos componentes de *software*, se pueden usar tecnologías basadas en *hardware* y *software* para implementar la red. Existen principalmente 2 tipos de *plugin* que podemos conectar a la arquitectura Neutron [36]:

- Core plugin: se encarga de implementar el núcleo de Neutron API y tiene la tarea de adaptar la red lógica descrita por redes, puertos y subredes en algo que pueda implementarse por el agente L2. Además, se encarga de la administración de direcciones IPs que corren en el *host*.
- Service plugin: provee servicios de red adicionales como *routing*, *load balancing*, *firewalling*, etc.

Modular Layer 2 (ML2) es un *core plugin* diseñado para ser escalable y soporta arquitecturas de red heterogéneas que pueden aprovechar varias tecnologías de red a la vez. ML2 plugin introdujo dos nuevos conceptos, Type Drivers y Mechanism Drivers para separar los tipos de redes que serán implementadas y los mecanismos para implementar redes de esos tipos [36]:

1. ML2 Type Driver: un Type Driver mantiene un estado de red de un tipo específico, valida los atributos de la red *provider* y describe segmentos de red usando atributos *provider*, entre los cuales tenemos etiquetas de interfaz de red, *segmentation* IDs y tipos de red. En la siguiente tabla se compara los tipos de redes soportados:

Tipo	Descripción
Local	Red aislada de otras redes y nodos. Las instancias conectadas a una misma red local pueden comunicarse con otras instancias dentro del mismo nodo de cómputo, pero no podrán hacerlo con instancias de otros *hosts*. Esta limitación hace que solo sea útil para propósitos de prueba.
Flat	En una red *flat* (*untagged*), no existe VLAN *tagging* u otra segregación de red. En varios entornos, una red *flat* corresponde a una VLAN de acceso o VLAN nativa en una troncal.
VLAN	Redes que usan 802.1Q *tagging* para segregar tráfico de red. Aquellas instancias que se encuentren en una misma VLAN son consideradas como parte de la misma red y se encuentran en el mismo dominio de *broadcast* L2. El enrutamiento entre VLANs solo es posible a través del uso de un *router* físico o virtual.
GRE	Redes que usan el protocolo de *tunneling* "*generic routing encapsulation*" para encapsular paquetes y enviarlos a través de redes punto a punto entre nodos. Para segregar tráfico se usa el campo KEY en el *header* GRE.
VXLAN	Redes que usan un ID de segmentación único, llamado VXLAN Network Identifier (VNI) para diferenciar tráfico entre redes VXLAN. El tráfico de una instancia a otra es encapsulado por el host usando el VNI y se envía a través de una red L3 existente usando UDP, donde es desencapsulado y reenviado a la instancia. Con VXLAN podemos resolver algunas limitaciones de redes VLANs.
GENEVE	Las redes GENEVE son parecidas a VXLAN, en cuanto a que se usa una ID para segmentación de tráfico (como VNI en VXLAN) y también usa UDP como mecanismo de transporte. Actualmente es usado por el *mechanism driver* Open Virtual Networking (OVN)

TABLA 2-2: Comparación de los tipos de red soportados por Type Drivers
FUENTE: [36]

En el presente trabajo de tesis se usará un Type Driver tipo VLAN, pues el uso de SR-IOV nos brinda la funcionalidad de filtrado de VLAN a nivel de la interfaz física de red, asignando una determinada VLAN a las VFs para la comunicación entre instancias de distintos nodos de cómputo. Por otro lado, no podríamos usar redes *overlay* como GRE, VXLAN o GENEVE porque los paquetes con identificadores del segmento de red no podrían ser identificados ni filtrados por las NIC SR-IOV.

2. ML2 Mechanism Driver: responsable de tomar la información que el Type Driver establece y asegurar que se implemente apropiadamente. ML2

permite configurar para operar múltiples *mechanism drivers* a la vez. Podemos categorizarlos en 3 tipos:

a. Basado en agente: Incluye Linux bridge, Open vSwitch, SR-IOV, etc
b. Basado en controlador: Juniper Contrail, Tungsten Fabric, OVN, Cisco ACI, VMWare NSX, etc.
c. Top of Rack (ToR): Cisco Nexus, Arista, Mellanox, etc

Como se indica en [21], para agregar la funcionalidad de SR-IOV a OpenStack debemos agregar *el mechanism driver* de este. Junto con otros parámetros de configuración tanto en los nodos *controller* como *compute* como se indica en la documentación, lograremos hacer el baipás del hipervisor y la capa de *switches* virtuales, con efectos como baja latencia y velocidad cercana al *line-rate*.

Otros *mechanism drivers* de interés son [36]:

- Linux bridge *mechanism driver*: soporta los tipos de red local, flat, vlan y vxlan. Cuando se usa este *mechanism driver*, el agente Neutron usa los módulos de *kernel bridge*, *8021q* y *vxlan* para conectar las instancias y otros recursos de red a *switches* virtuales. De esta forma, las instancias podrán comunicarse con otros recursos de red dentro o fuera de la nube.
  En una red basada en el *mechanism driver* de Linux bridge hay 5 tipos de interfaces administrados por OpenStack Networking: interfaces *tap*, interfaces físicas, interfaces VLAN, interfaces VXLAN y Linux Bridges.
  En la siguiente figura se muestra un escenario donde se requiere dos VLANs y por tanto dos Linux Bridges son creados para separar el tráfico de las instancias a nivel de VLAN:

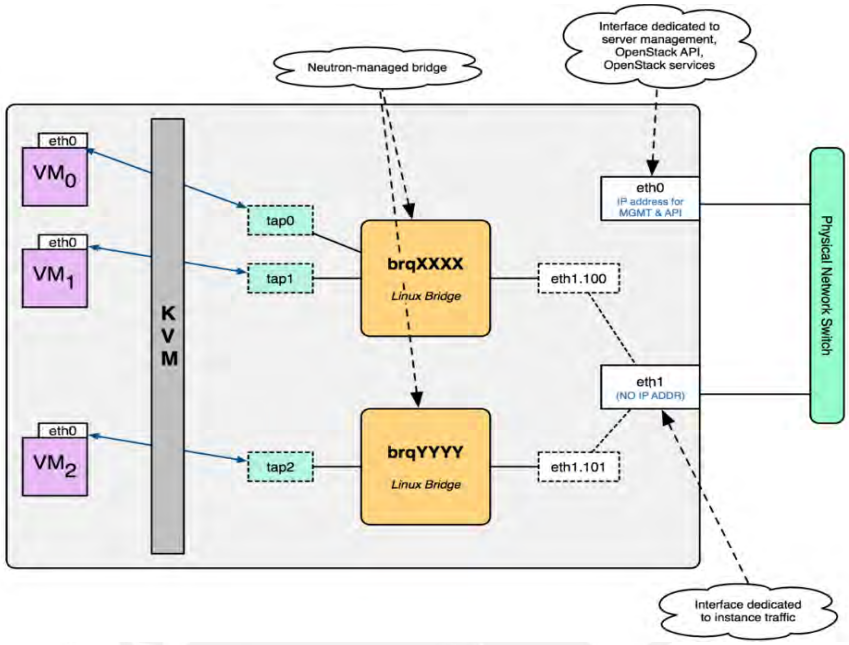

FIGURA 2-6: *Compute node* con una red *provider* tipo VLAN usando Linux Bridge como *mechanism driver*
FUENTE: [36]

- Open vSwitch *mechanism driver*: soporta los tipos de red local, flat, vlan, vxlan, gre y geneve. OpenStack Networking solo usa un conjunto de *features* de Open vSwitch, el cual se compone de 3 elementos principales: módulo de *kernel*, *vSwitch daemon* y el servidor de base de datos (OVSDB). En una red basada por el Open vSwitch *mechanism driver* se tienen 5 tipos de dispositivos de networking virtual: dispositivos tap, Linux bridges, cables virtual ethernet (Veth), OvS bridges y OvS patch ports.

  Los dispositivos Linux Bridges siguen teniendo el mismo uso que en para un escenario con Linux Bridge *mechanism driver*, solo que ahora los usamos para conectarlos a un OvS *bridge* de integración mediante *veth pairs*. A su vez, el *integration bridge* se conecta a un *provider bridge* que sirve para proveer conectividad a la red física. En la siguiente imagen se representan estos elementos:

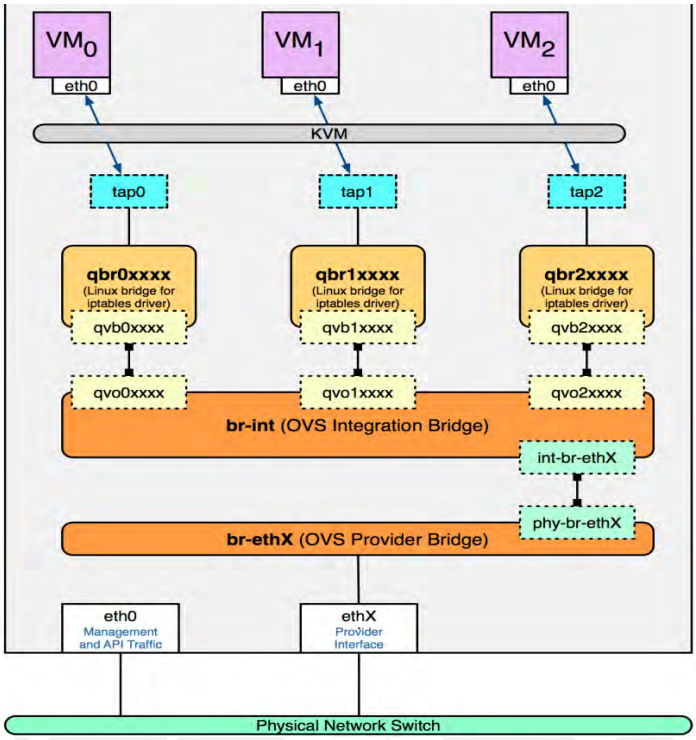

FIGURA 2-7: C*ompute node* con Open vSwitch *mechanism driver*

FUENTE: [36]

En lo que respecta a este trabajo de tesis se aprovechará la compatibilidad del plugin ML2 con múltiples mechanism drivers, empleando el driver de SR-IOV junto con otro driver convencional; ya sea el de Open vSwitch o el de Linux bridge. De esta manera se usará la PF de la interfaz de red física con el mecanismo *legacy* y las múltiples VFs definidas en el sistema para instancias con interfaces directas (mecanismo SR-IOV).

2.2.3 Seguridad en OpenStack

Neutron nos ofrece 2 APIs para implementar filtros de seguridad en el tráfico de red [36]:

1. Security group API: filtra el tráfico a nivel de puerto de la instancia. Las reglas que se configuren para un *security group* pueden implementarse (i) dentro de *iptables* o como (ii) reglas Open vSwitch *flows* dentro de un compute node. El filtrado se da al tráfico entrando y saliendo de los puertos conectados a las instancias.

2. Firewall as a Service (FWaaS) API: implementa reglas de filtrado tanto a nivel de puertos de instancias como a nivel de puerto de un *router* y de otros equipos.

2.2.3.1 Security groups implementado con iptables

Un *security group* es “una colección de reglas de acceso a la red conocidas como *security group rules* que limitan el tipo de tráfico que una instancia puede enviar o recibir” [36]. Al usar el *driver* basado en *iptables* las reglas de *security groups* son traducidas a reglas *iptables*.

Por defecto, el tráfico es implícitamente denegado y en las reglas de los security groups solo definimos el tráfico permitido que pueda pasar por los puertos de las instancias a las que se asigne el *security group*.

Iptables es un *firewall* construido para sistemas Linux que habilita a un administrador del sistema definir a través de tablas, que se componen de cadenas de reglas, cómo deben tratarse los paquetes entrantes o salientes. Los paquetes se procesan secuencialmente pasando por las reglas en cadena dentro de las tablas Raw, Filter, NAT y Mangle. Las 5 cadenas predeterminadas son PREROUTING, INPUT, FORWARD, OUTPUT y POSTROUTING. Cada vez que un paquete pasa por una cadena, se examina cada regla. Si una regla coincide con el paquete (*match* de los criterios de la regla), la regla toma la acción indicada por el veredicto configurado. Las acciones disponibles que se pueden aplicar a un paquete son ACCEPT, DROP, REJECT, LOG, DNAT, SNAT y RETURN [36].

Neutron logra abstraer la implementación de *security groups* para los usuarios. Con el objetivo que el número de reglas no crezca de forma desmedida, Neutron usa la herramienta *ipset* para reducir el número de reglas *iptables* requeridas. Esto tiene un impacto positivo en el rendimiento y confiabilidad del sistema [36].

OpenStack permite administrar los *security groups* y las reglas asociadas a ellos usando la REST API de Neutron, el CLI de OpenStack o desde el mismo Dashboard, con la capacidad de listar, crear, eliminar, modificar y ver detalles [36].

Cuando usamos un agente Linux Bridge, las interfaces *tap* de las instancias son conectadas a un *bridge* compartido. En cambio, cuando usamos un agente OvS, las interfaces *tap* se conectan a sus propios Linux Bridges. En cualquiera de estos 2 casos, las reglas de *security groups* deben aplicarse a los puertos conectados al Linux Bridge. Además, tenemos la posibilidad de aplicar más de un *security group* a una instancia. Si bien podemos especificar un *security group* a un puerto; en el caso donde una nstancia tenga más de un puerto, debido a que un puerto no puede ser especificado como parte de un comando, el *security group* se aplicará a todos los puertos que la instancia tenga conectados [36].

2.2.3.2 Security groups implementado con flows

A raíz de que Open vSwitch (OVS) no puede interactuar directamente con iptables para implementar security groups, el agente OVS y el servicio de Compute insertan un Linux bridge (que contiene las reglas de iptables de la instancia) entre cada máquina virtual y el bridge de integración (br-int) para la implementación de security groups [39].

Para evitar problemas de escalabilidad y rendimiento al agregar componentes adicionales entre instancias y la infraestructura de red física, el agente OVS incluye un firewall driver opcional. Este firewall driver implementa security groups como flows insertados en el OVS, en lugar de usar Linux bridges e iptables. Aumentando así, el nivel de escalabilidad y el rendimiento [39].

El OVS driver posee la misma API que la usada actualmente para el driver de iptables, manteniendo el estado de security groups y los puertos dentro del firewall. Se creó una clase para mantener la consistencia de estado, mapeando

de puertos a security groups y viceversa. Cada puerto y security group es representado por su propio objeto encapsulando la información necesaria [40].

En un primer momento, cada conexión se divide en procesos ingress o egress basado en el input o output port respectivamente. Cada puerto contiene los flows iniciales (hardcoded) para ARP, DHCP y conexiones establecida, que son aceptadas por defecto. Para detectar conexiones establecidas, un flow debe marcarse primero por conntrack con una reglas action=ct(). Un flow aceptado significa que los paquetes entrantes para la conexión son enviados directamente al puerto y los puertos salientes son dejados para que sean procesados por el integration bridge [40].

Las conexiones que no hayan hecho match con las reglas previas son enviadas a la tabla de filtrado ingress o egress, dependiendo de su dirección. La razón por la cual las reglas están basadas en reglas de security groups en tablas separadas es para hacer más fácil su detección durante la eliminación. Las reglas de security groups son tratadas de forma distinta según si tienen o no un remote group ID [40].

2.2.3.3 Allowed address pairs

Neutron aplica, por defecto, reglas anti-spoofing a todos sus puertos, para asegurarse que tráfico no deseado no se origine o pase a través de un puerto. Esto incluye reglas que prohíben a las instancias de funcionar como servidores DHCP. La extensión allowed-address-pairs puede usarse para permitir IPs, subredes y direcciones MAC adicionales a través de un puerto, además de la IP y dirección MAC fijas asociadas con el puerto [36].

Allowed address pairs también son de utilidad cuando se trata una instancia como un dispositivo de enrutamiento o un concentrador VPN, o cuando se implementa una alta disponibilidad entre múltiples instancias (clustering

instances) utilizando direcciones que necesitan “flotar” entre ellas, como una implementación haproxy y/o keepalived [36].

En la siguiente imagen podemos comparar el estado inicial del puerto (izquierda) donde solo deja pasar el par IP/MAC por defecto, comparado con un puerto con un par de direcciones IP/MAC extra agregado (derecha):

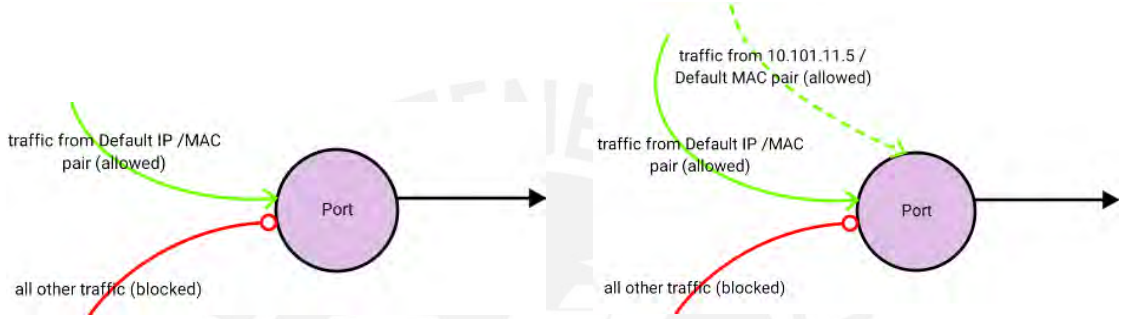

FIGURA 2-8: Estado del puerto inicial (izquierda) *versus* estado del puerto con un *allowed address pair* agregado (derecha)
FUENTE: [41]

2.2.3.4 Port security

La extensión port security perteneciente al plugin ML2 permite deshabilitar todo el filtrado de paquetes en un puerto; incluyendo reglas predeterminadas que previenen IP y MAC spoofing, así como la funcionalidad de security group. Esta extensión es especialmente útil cuando se despliegan instancias para ser usadas como un router, load balancer o firewall y no es posible proveer todas las combinaciones con “allowed-address-pairs” para cubrir todas las máquinas posibles que puedan enviar tráfico a través de esta instancia de networking [36].

Aún sin la extensión de port security agregada, Neutron implementa el mismo comportamiento predeterminado de la extensión mediante la implementación de reglas de spoofing de direcciones IP; MAC y DHCP en cada puerto. La extensión de port security permite a los usuarios con el rol de admin deshabilitar la seguridad de un puerto, ya sea en puertos individuales o en toda la red. Una vez que se haya deshabilitado port security en un puerto, la API no permitirá asociar

el puerto con ningún security group. Ya que no existen reglas de filtrado, todo el tráfico entrante y saliente a través del puerto es permitido [36]. La siguiente imagen representa cómo sería el flujo del tráfico, una vez que se ha agregado la extensión de port security y se ha deshabilitado port security para un puerto.

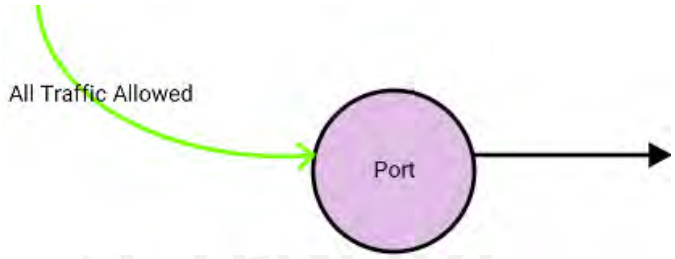

FIGURA 2-9: *Port security* deshabilitado para un puerto
FUENTE: [41]

2.3 SDN

Software-Defined Networking (SDN) representa un nuevo enfoque para las redes caracterizado por ser dinámico, adaptable, fácil de administrar y rentable, acomodándose mejor a los requerimientos actuales de gran ancho de banda y dinámicos [42]. El planteamiento de SDN permite la programabilidad de las redes, es decir, la capacidad de iniciar, controlar, cambiar y administrar los recursos de red dinámicamente a través de interfaces abiertas [43].

2.3.1 Arquitectura SDN

La arquitectura SDN se caracteriza principalmente por el desacoplamiento de los planos de datos y de control, permitiendo que el control sea programable de forma centralizada y la infraestructura sea abstraída para las aplicaciones y servicios de red.

De acuerdo al RFC7426 (Software-Defined Networking (SDN): Layers and Architecture Terminology) del Internet Research Task Force (IRTF), SDN comprende distintos planos, capas de abstracción e interfaces. En la siguiente figura se muestra la arquitectura en capas propuesta por el RFC7426.

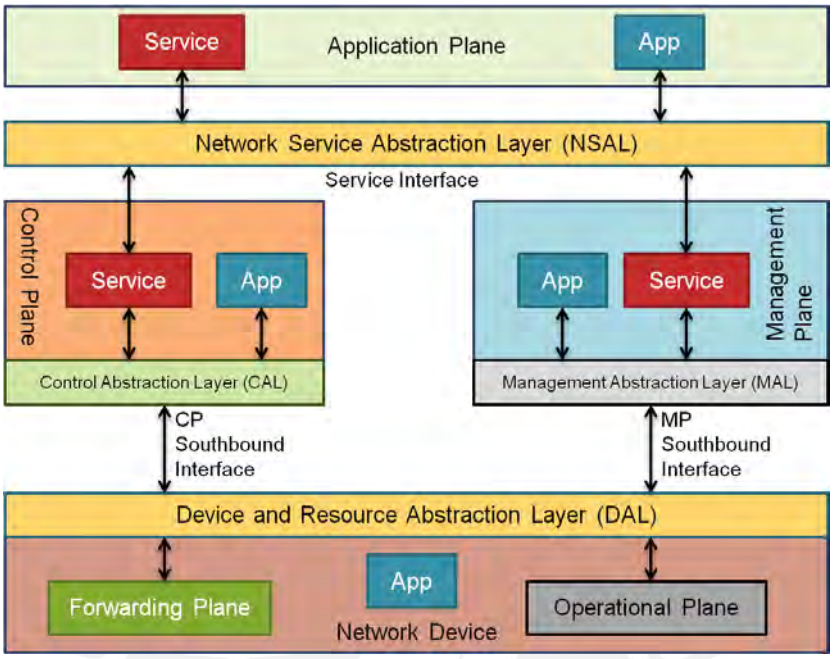

FIGURA 2-10: Arquitectura SDN en capas según el RFC7426

FUENTE: [44]

2.3.1.1 Planos

Los planos hacen referencia al conjunto de funciones y recursos que se relacionan a una misma funcionalidad. En redes tradicionales, los Internet *routers* y *switches* Ethernet han tenido al plano de control implementado junto con el dispositivo de red. Asimismo, el plano de administración se ha encontrado centralizado como un sistema de gestión de la red. Con el modelo SDN se puede centralizar el plano de control de los dispositivos de red y agruparlo con el plano de administración en un solo *software*. En las siguientes figuras se muestra una comparativa de una arquitectura de red tradicional y una SDN.

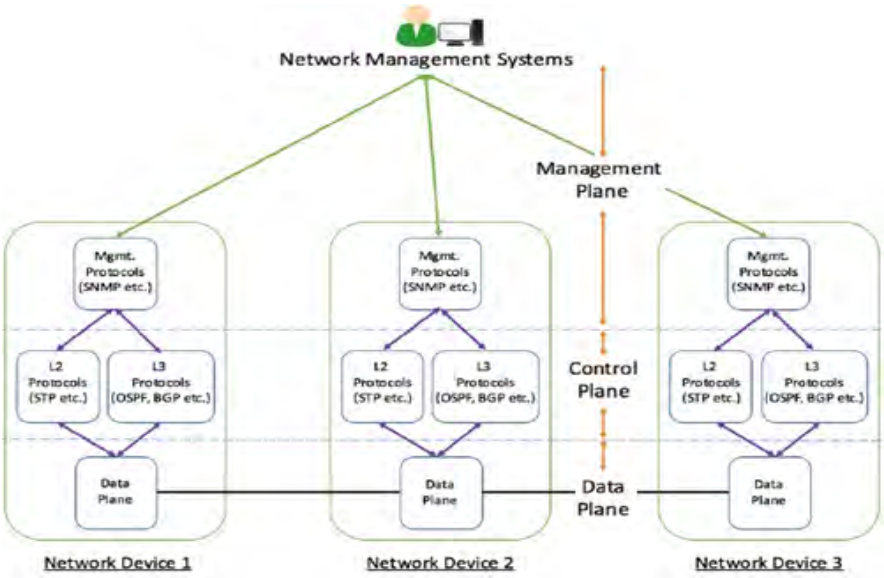

FIGURA 2-11: Arquitectura de red tradicional

FUENTE: [45]

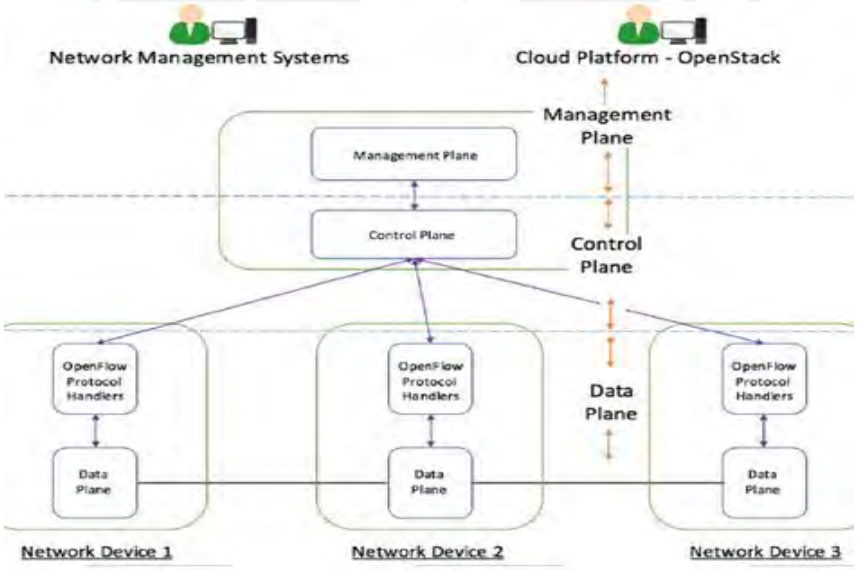

FIGURA 2-12: Arquitectura de red SDN

FUENTE: [45]

- Forwarding/Data/User Plane: plano responsable de manejar paquetes de datos según las instrucciones recibidas del plano de control. Toma las acciones de reenviar, descartar y modificar los paquetes [43].

- Operational Plane: administra el estado operacional de los dispositivos de red, es decir, si se encuentra activo, inactivo, disponibilidad y estado de puertos, entre otras responsabilidades. Es usualmente el punto final de las aplicaciones y servicios del plano de administración [43].

- Control Plane: es el plano que toma las decisiones sobre los paquetes, definiendo cómo deben ser reenviados por los dispositivos de red e ingresando dichas decisiones en ellos, es decir, decide cómo será configurado el plano de datos de cada equipo [43].

- Management Plane; su trabajo es monitorear, configurar y mantener los dispositivos de red; por ejemplo, realiza decisiones sobre el estado de un equipo en la red [43].

- Application Plane: en este plano se integran las aplicaciones y servicios que define el cómo se comporta la red [43].

2.3.1.2 Capas de abstracción

Las capas de abstracción se refieren a la abstracción de los recursos de un plano en particular:

- Device and Resource Abstraction Layer (DAL): abstrae los recursos del *forwarding* y *operational plane* del dispositivo de red a los control y *management plane* [43].

- Control Abstraction Layer (CAL): abstrae la Interfaz Southbound del *control plane* y el DAL para las aplicaciones y los servicios del plano de control [43].

- Management Abstraction Layer (MAL): abstrae la Interfaz Southbound del *management plane* y el DAL para las aplicaciones y los servicios del plano de administración [43].

- Network Service Abstraction Layer (NSAL): provee abstracción de servicios que pueden ser usados por aplicaciones y servicios [43].

2.3.1.3 Interfaces

Las interfaces SDN involucran a las APIs que existen entre los planos:

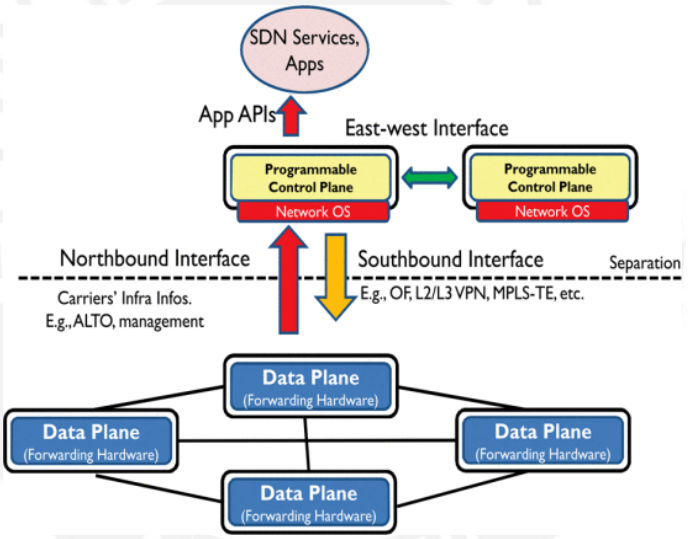

FIGURA 2-13: Arquitectura de referencia SDN y APIs
FUENTE: [46]

- Southbound Interface: interfaz entre el *control plane* (controlador) y *data plane*. Debe estar caracterizada por la programabilidad y rápida reconfiguración brindando flexibilidad al plano de control y la posibilidad de adoptar nuevos esquemas de control en la red rápidamente. Además, debe permitir compartir recursos a través de la abstracción de las características de los recursos físicos. También tiene la responsabilidad de aislar el tráfico, proveyendo un aislamiento entre redes virtuales y asegurando los aspectos de rendimiento y seguridad. Y será capaz de

abstraer la red, precisamente la información de los recursos de la red física [46].

- Northbound Interface: interfaz entre las aplicaciones/servicios y la infraestructura de red (controladores). Debe cumplir con requerimientos relacionados con enrutamiento (descubrir la topología, ingeniería de tráfico, retraso, *jitter*, QoS, etc.), administración (recursos, uso de energía, monitoreo, mantenimiento, etc.) y políticas (control de acceso, seguridad, etc.) [46].

- East-West Interface: interfaz entre controladores. Tienen los objetivos de brindar intra/inter dominios, escalabilidad, interoperabilidad, facilidad de despliegue, etc. [46].

2.3.2 OpenFlow

OpenFlow es una instancia de la arquitectura SDN conformada por un conjunto de especificaciones mantenidas por el Open Networking Forum (ONF) [47]. Define un protocolo de comunicación a través del cual un controlador centralizado (*control plane*) pueden controlar un *switch* OpenFlow (*data plane*). Cada uno de estos *switches* mantiene una o más *flow tables*, empleadas para realizar búsqueda de paquetes [43]. En síntesis, OpenFlow es el protocolo que determina la comunicación entre los *switches* OpenFlow y el controlador OpenFlow.

2.3.2.1 Switch OpenFlow

Un *switch* OpenFlow u OpenFlow Logical Switch es un conjunto de recursos que pueden ser administrados como una sola entidad, incluyendo un *datapath* y un canal de control [9]. El *switch* OpenFlow está conformado por una o más *flow tables*; una *group table*, la cual realiza búsqueda y reenvío de paquetes; y uno o más OpenFlow *channel* hacia un controlador externo [9]. En la siguiente figura se grafican los componentes de un *switch* OpenFlow.

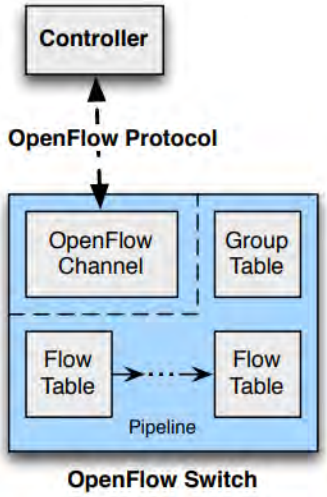

FIGURA 2-14: Componentes principales de un *switch* OpenFlow

FUENTE: [48]

2.3.2.2 OpenFlow ports

Los *switches* OpenFlow se encuentran conectados lógicamente uno a otro mediante puertos OpenFlow, interfaces con la capacidad de pasar paquetes entre el procesamiento OpenFlow y el resto de la red. Los paquetes OpenFlow se reciben a través de un *ingress port* y posteriormente se procesan por el OpenFlow *pipeline* que podría reenviarlos a través de un *output port*. El *packet ingress port* representa el puerto OpenFlow en el cual el paquete fue recibido en el *switch* OpenFlow y puede ser usado para el *matching* de paquetes [48].

En un *switch* OpenFlow deben soportarse 3 tipos de puerto OpenFlow: puerto físico, puerto lógico y puerto reservado [48].

- Puertos estándar: definidos como puertos: físico, lógico y el reservado LOCAL. Pueden emplearse como *ingress* y *output port*.
- Puertos físicos: puertos definidos por el *switch* que corresponden a una interfaz de *hardware* del *switch*.
- Puertos lógicos: puertos definidos por el *switch* que tienen una correspondencia directa con la interfaz de un *switch*, pudiendo ser definidos por métodos distintos a OpenFlow (*link aggregation*, túneles,

interfaces *loopback*). Tienen la posibilidad de encapsular paquetes y mapearse a varios puertos físicos.

- Puertos reservados: puertos obligatorios y opcionales que especifican acciones de reenvío genéricas: enviar un paquete al controlador, *flooding* o no ser procesados por OpenFlow, es decir, procesarlos como un *switch* convencional. Ejemplos de puertos reservados: ALL, CONTROLLER, TABLE, IN_PORT, ANY, LOCAL, NORMAL, FLOOD.

2.3.2.3 OpenFlow Tables

En los switches compatibles con OpenFlow, aquellos que solo soportan OpenFlow (OpenFlow-only) se procesan todos los paquetes exclusivamente a través del OpenFlow pipeline. Los *switches* OpenFlow-hybrid soportan operación OpenFlow y operación normal de *switching* Ethernet, por lo que deben tener un mecanismo de clasificación para procesar el tráfico por el pipeline de OpenFlow o el normal [48].

Un OpenFlow *pipeline* contiene varios flow tables, donde cada flow se conforma de flow entries, compuesto de componentes como Match Field, Prioridad, Contadores, Instrucciones, Timeouts, Cookie y Flags. El procesamiento del OpenFlow pipeline define cómo los paquetes interactúan con esas flow tables. En la siguiente figura se muestra el procesamiento del pipeline de OpenFlow [48].

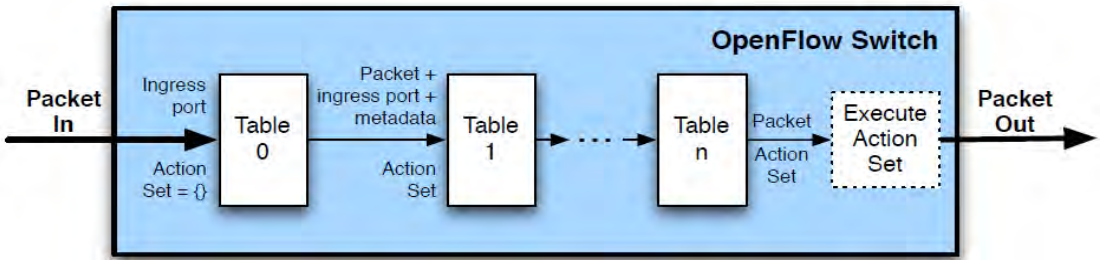

FIGURA 2-15: Procesamiento de OpenFlow Pipeline

FUENTE: [48]

Al recibir un paquete, un *switch* OpenFlow comienza realizando una búsqueda en la primera *flow table* y sigue un flujo según se detalla en la Figura siguiente. El procesamiento de un paquete por cada tabla que pasa es (i) encontrar la *matching flow entry* con mayor prioridad, (ii) aplicar instrucciones (p.ej. modificar el paquete y actualizar los campos de *match*, actualizar el conjunto de acciones, o actualizar la metada) y (iii) enviar los datos con *match* y el conjunto de acciones a la siguiente tabla. Un paquete hace *match* con una entrada de la *flow table* si los valores en el campo de *match* del paquete coinciden con los definidos en la entrada del *flow table*. Si el paquete hace *match*, se ejecutan instrucciones contenidas en el *flow entry*, pudiendo realizar cambios en el paquete, realizar una acción o procesarlo en el *pipeline* [48].

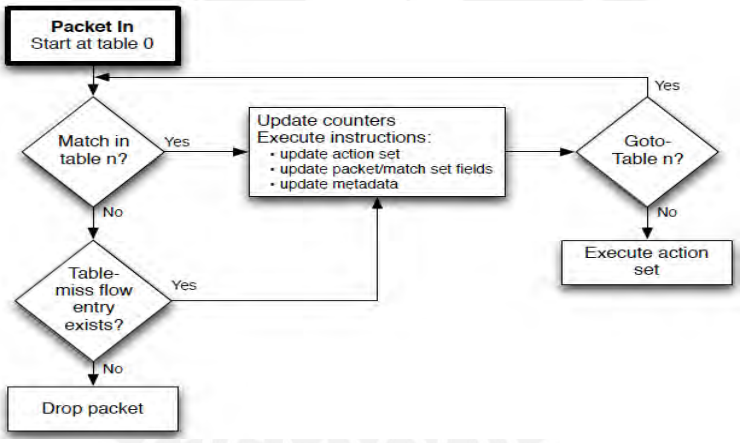

FIGURA 2-16: Diagrama de flujo de un paquete en un *switch* OpenFlow
FUENTE: [48]

2.3.2.4 OpenFlow Channel

El canal OpenFlow es la interfaz que se encarga de realizar la conexión entre cada *switch* OpenFlow con un controlador OpenFlow y a través de la cual, este controlador configura y administra el *switch*, recibe eventos del *switch* y envía paquetes al *switch*. El canal OpenFlow es usualmente encriptado a través de TLS [48].

2.3.2.4 Control Channel

El canal de control del *switch* podría soportar un solo canal OpenFlow con un único controlador o varios canales OpenFlow dándole la capacidad de administrar el *switch* mediante distintos controladores. El canal de control incluye un canal OpenFlow por controlador OpenFlow [48].

2.3.2.5 OpenFlow Switch Protocol

Mediante el Openflow switch protocol, el controlador puede añadir, actualizar y eliminar *flow entries* en una *flow table*, ya sea de forma reactiva o proactiva. Cada *flow* en la tabla está conformada por *flow entries*, y cada *flow entry* se compone de *match fields*, contadores y un conjunto de instrucciones que serán aplicados a paquetes coincidentes (*matching packets*).

El componente principal de la especificación del *switch* OpenFlow es el conjunto de estructuras usadas para mensajes del OpenFlow Switch Protocol. Este protocolo puede ser visto en 4 componentes que interactúan como se muestra en la siguiente figura.

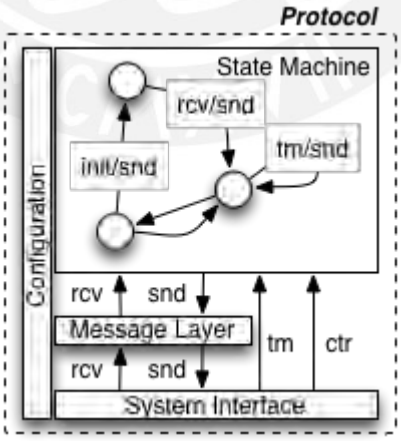

FIGURA 2-17: componentes e interacción del protocolo OpenFlow
FUENTE: [47]

- Message Layer: componente principal del protocolo que define una estructura y semántica válidas para todos los mensajes. La capa de mensaje soporta la capacidad de construir, copiar, comparar, imprimir y manipular mensajes [47].
- State Machine: define el comportamiento a bajo nivel del protocolo. Describe acciones como negociación, capacidad de descubrir, control de flujo, entrega, etc [47].
- System Interface: define cómo el protocolo interactúa con el exterior. Identifica interfaces necesarias y opcionales junto con su uso previsto, como TLS y TCP como canales de transporte [47].
- Configuration: casi todos los aspectos del protocolo tienen configuraciones o valores iniciales.

2.3.3 Controlador SDN

El controlador SDN es la entidad que centraliza el plano de control de los dispositivos de red (*switch*, *router*), con la función de programar el plano de datos de estos equipos y administrarlos.

Existen diversos controladores *open source* como ONOS, Ryu, Floodlight y OpenDayLight, cada cual con su conjunto de funciones propio. En la siguiente tabla se muestra un resumen comparativo de distintos controladores en términos sus propiedades y *features* [49].

	Ryu	Floodlight	OpenDayLight	ONOS
Southbound Interfaces	OF 1.0, 1.2, 1.3, 1.4, NETCONF, OFCONFIG, OVSDB	OF1.0,1.1,1.2, 1.3, 1.4,1.5	OF1.0, 1.3,1.4,1.5 NETCONF/YANG,OVSDB, PCEP, BGP/LS, LISP, SNMP, OFCONFIG	OF1.0, 1.3,1.4, 1.5 NETCONF
REST API	Yes (For SB only)	Yes	Yes	Yes
GUI	Yes (Initial phase)	Web/ Java- based	Web-based	Web-based
Modularity	Medium	Medium	High	High
Orchestrator Support	Yes	Yes	Yes	No
OS Support	Most supported on Linux	Linux, Windows , and MAC	Linux, Windows , and MAC	Linux, Windows , and MAC
Partner	Nippo Telegraph And Telephone Corporation (NTT)	Big Switch Networks	Linux Foundation With Memberships Covering Over 40 Companies, like Cisco, IBM,	ON.LAB, Sk Telecom, Cisco, Ericsson, Fujitsu, Huawei, Intel, AT&T, Nec, Ciena, Nsf.Ntt Comunnication
Documentation	Medium	Good	Very good	Good
Programming Language	Python	Java	Java	Java
Multi-threading support	Yes	Yes	Yes	Yes
TLS Support	Yes	Yes	Yes	Yes
Virtualization	Mininet and OVS	Mininet and OVS	Mininet and OVS	Mininet and OVS
Application Domain	Campus	Campus	Data center and Transport-SDN WAN	Data center and Transport-SDN WAN
Distributed/Centralized	Centralized	Centralized	Distributed	Distributed

TABLA 2-3: Comparación de controladores SDN

FUENTE: [49]

La arquitectura SDN no especifica un diseño o implementación específica de un controlador SDN. Se puede pensar en el controlador SDN como una caja negra, definida por su comportamiento observable externamente que debe tener algunos componentes funcionales como mínimo: *data plane control function* (DPCF), coordinador, virtualizador y agente [50]. La siguiente figura muestra la arquitectura y lógica de un controlador SDN.

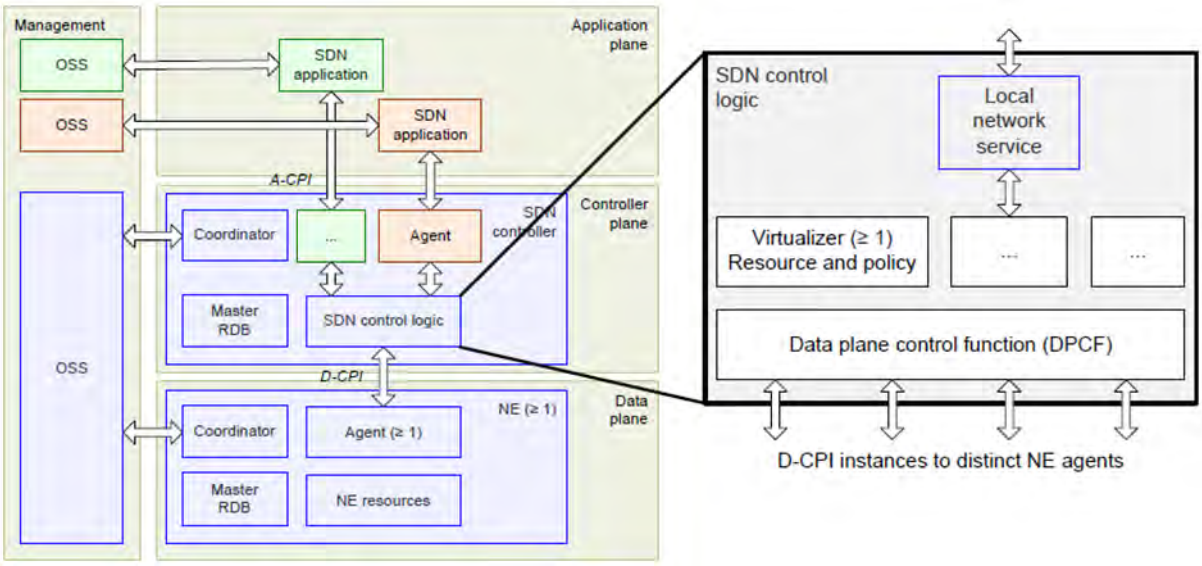

FIGURA 2-18: Arquitectura y lógica de un controlador SDN

FUENTE: [50]

- Data Plane Control Function: posee los recursos subordinados disponible a él y hace uso de ellos según lo indique el coordinador o virtualizados que los controla [50].
- Coordinador: es el componente del controlador SDN que actúa en nombre del administrador, estableciendo el entorno del cliente y el servidor [50].
- Virtualizador: asigna recursos abstraídos a clientes o aplicaciones [50].
- Agente: componente funcional que representa los recursos y capacidades del cliente en el entorno del servidor [50].

2.3.3.1 Elección de un Controlador SDN

La implementación del paradigma SDN permitirá solucionar las dificultades de administración de *networking*. La elección de un controlador que se integre a un *entorno cloud*, brindando los beneficios de un control centralizado, es básica para cumplir con los objetivos de la tesis.

Un controlador debe ser capaz de desempeñarse correctamente en distintas cargas o aplicaciones para diversos entornos. Pero no se puede obtener rendimiento a cambio de modularidad. La arquitectura del controlador debe garantizar escalamiento horizontal en ambientes de *cloud* [51].

Se ha optado por elegir a OpenDaylight (ODL) como controlador SDN de trabajo, debido a su enfoque para espacios de integración con *cloud* y SD-LAN. OpenDaylight es una plataforma abierta modular para personalizar y automatizar redes de todo tamaño y escala que surgió del movimiento SDN con un enfoque en la programabilidad de la red. El proyecto ODL ha sido implementado en código Java bajo la responsabilidad de Linux Foundation Networking [52]. A continuación, se detalla el funcionamiento de ODL y su arquitectura.

2.3.3.2 OpenDaylight

OpenDaylight presenta una arquitectura en capas con dos interfaces [45]:

- Interfaz *southbound*: asegura que tecnologías de *networking* y *hardware* de diversos *vendor* puedan ser usados por ODL.
- Interfaz *northbound*: brinda APIs para los usuarios finales y otras tecnologías como OpenStack.

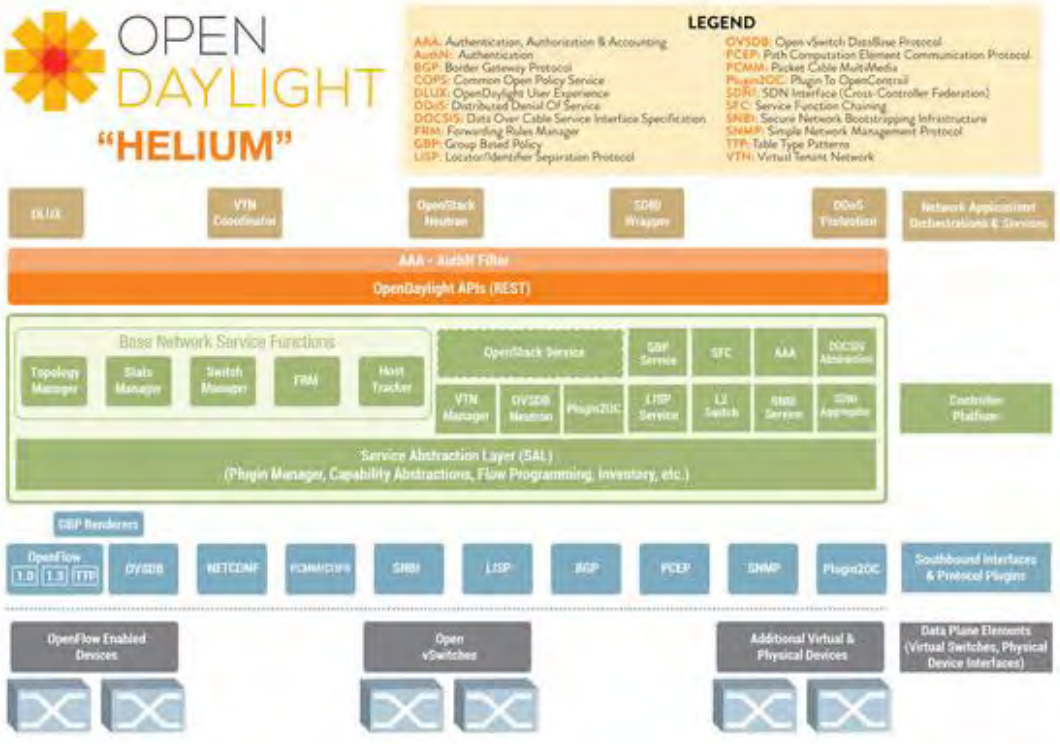

FIGURA 2-19: Arquitectura OpenDaylight

FUENTE: [53]

Como se muestra en la figura previa la arquitectura de ODL se representa por 3 capas que juntas conforman el plano de control: aplicaciones y servicios de red, plataforma del controlador y las interfaces Southbound junto con los protocolos [54].

La Northbound API de OpenDaylight, llamada RESTCONF, es un protocolo basado en HTTP que provee REST APIs para datos modelados por YANG e invoca RPCs (*remote procedure call*) modelados por YANG, usando formatos XML o JSON como *payload*. RESTCONF permite acceso a *datastores* en el controlador, soportando operaciones de OPTIONS, GET, PUT, POST y DELETE. Junto con RESTCONF, podemos enviar *flows* OpenFlow mediante la REST API provista [55].

El *plugin* de OpenFlow para OpenDaylight habilita la comunicación del controlador con un *switch* OpenFlow aplicando las reglas insertadas con RESTCONF. El proyecto de OpenFlow *plugin* de OpenDaylight busca desarrollar un *plugin* que soporte implementaciones de la especificación OpenFlow basado en la arquitectura Model Driven Service Abstraction Layer (MD-SAL) [56].

Un *plugin* prescindible de OpenDaylight llamado DLUX habilita una interfaz web que ayuda a representar visualmente la topología con los *switches* OpenFlow conectados al controlador SDN y los *hosts* conectados al *switch*. Además, se puede ver un detalle de los nodos, ids, tráfico y *flows* de los mismos.

3. Diseño de un módulo de ciberseguridad basado en SDN

En el capítulo previo se revisaron las tecnologías que nos permitirán realizar el diseño de la solución de ciberseguridad que se tiene planteado como objetivo de la tesis.

En el presente capítulo se analizarán los requerimientos de seguridad que se necesitan para las redes de un *slice* HPC, con el fin de construir reglas que protejan la comunicación entre instancias de máquinas virtuales permitidas. Esto se realizará teniendo en cuenta la escalabilidad de la cantidad de reglas y TCAMs del *switch* de datos que emplean. Posteriormente, se trabajará con la arquitectura de la solución como un subconjunto de componentes de la aplicación HAST que conversa con sistemas de *software* externos como OpenStack y OpenDaylight. Finalmente, se propondrá el diseño de la solución de seguridad conformada por los componentes de los módulos de ciberseguridad y *slice manager*; así como la definición de las APIs que permite la interacción entre ambas partes.

3.1 Slice HPC

Se denomina *slice* HPC al elemento que agrupa una serie de componentes virtualizados y que se pretende resguardar ante ataques mediante reglas de seguridad. Un *slice* HPC es un concepto que emplea distintos tipos de tecnologías relacionadas con la virtualización de redes y de cómputo, con capacidad de correr cargas HPC.

Cada *slice* HPC está compuesto de elementos virtuales como redes, subredes, puertos, *switches*, *routers*, e instancias de máquinas virtuales (VM) creados con la API de OpenStack. Todas estas piezas se encuentran conectadas y organizadas de forma que la comunicación entre instancias sea posible y las cargas de trabajo HPC estén divididas. De esta manera, la seguridad puede implementarse específicamente a los distintos tipos de tráfico de red.

Las instancias VM que pertenecen a un *slice* HPC se pueden diferenciar según el rol que desempeñan en el *slice* y las redes a las que están conectadas; pudiendo ser del tipo *master* o *worker*. Se precisan las funciones de cada tipo de VM:

- **Instancias VM *master***: máquinas virtuales usadas para correr servicios específicos (SSH, HTTP, etc.) a los que usuarios externos puedan acceder para interactuar con el clúster. También son usadas para administrar las VMs *workers* y comunicarse a altas velocidades con estas VMs.
- **Instancias VM *workers***: máquinas virtuales pensadas para correr cargas de trabajo demandantes en cómputo y enviar datos a altas velocidades entre ellas. También se emplean para comunicaciones exigentes con los *masters*.

Se considera que un *slice* HPC debe contar con 3 tipos de redes distintas para segmentar los distintos tipos de tráfico que las instancias VM pueden generar:

red de acceso, red de administración y red de datos. A continuación, se detallan las funciones de cada tipo de red del slice HPC:

- **Red de acceso (*access network*)**: red creada para que los usuarios externos puedan acceder al clúster HPC (conectándose al nodo *master*) y ejecutar tareas o acceder a algún servicio específico (SSH, HTTP, etc.). Solo las VMs *master* poseen un puerto conectado a este tipo de red. Esta red no requiere características de red demandantes (en velocidad ni latencia).
- **Red de administración (*management network*)**: red usada para gestión centralizada de todas las VMs *workers* desde la instancia *master*: configuración de archivos, procesos, instalación de paquetes, etc. Tanto las VMs master como *worker* poseen un puerto conectado a este tipo de red. Esta red presenta propiedades de red exigentes (alta velocidad y baja latencia).
- **Red de datos (*data network*)**: red usada para la comunicación entre instancias *workers* con usos específicos de aplicaciones HPC que requieren altas velocidades de red y baja latencia. Solo las VMs *worker* poseen un puerto conectado a este tipo de red.

Las instancias *masters* se pueden conectar a redes externas a través de su conexión en la red de acceso. Cada red de acceso de un *slice* HPC tiene un *router* virtual de OpenStack conectado a una red *external provider*, la cual tiene asociada una interfaz física del servidor con capacidad de conexión al exterior.

En la siguiente imagen se muestra un ejemplo de un *slice* HPC con los componentes virtuales que lo conforman y han sido creados mediante OpenStack API.

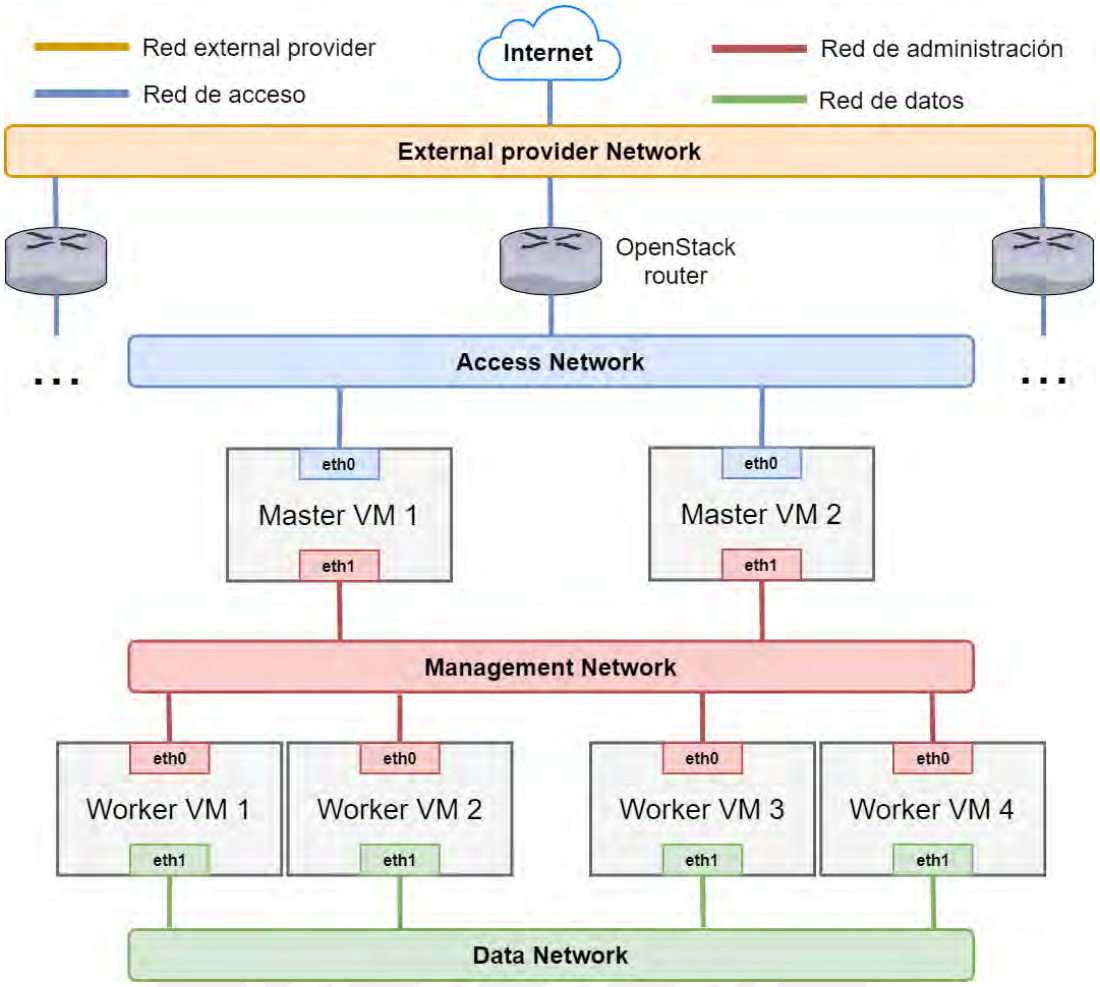

FIGURA 3-1: Ejemplo de un *slice* HPC con las redes e instancias identificadas

La selección de un ML2 *mechanism driver* para cada tipo de red juego un rol importante, en cuanto a que habilita cumplir con los requerimientos del tipo de tráfico que generarán las instancias. La red tipo (i) acceso usa el mechanism driver de Open vSwitch aunque la solución también es compatible con el Linux bridge *mechanism driver*. El motivo de elegir este *mechanism driver* es que el tráfico de red no es demandante. Además, se puede tener reglas/políticas de acceso complejas que se pueden beneficiar de la flexibilidad que provee este mecanismo al implementar las reglas en software (p.ej., IP Tables). Las redes tipo (ii) administración y (iii) datos usan el *mechanism driver* de SR-IOV ya que las comunicaciones en red son exigentes en velocidad y latencia.

Debido a que las redes tipo acceso pueden trabajar con los *mechanism driver* de Open vSwitch o Linux bridge, la seguridad en la red puede implementarse usando *security groups*. Por otro lado, las redes de administración y datos no cuentan con el *feature* de *security group*, debido a la incompatibilidad del *mechanism driver* de SR-IOV con esta funcionalidad. Por este motivo, para las redes de administración y acceso, se buscará aplicar otro mecanismo de seguridad, como lo es insertar *flows* en el *switch* de datos que interconecta los nodos de cómputo de OpenStack.

Otra restricción del uso de SR-IOV *mechanism driver* actual es que para la elección del ML2 *type driver* se restringe al uso de VLANs como método de segmentación (VLAN *type driver*), ya que, no posee compatibilidad para túneles actualmente. Por este motivo, todas las redes creadas para un *slice* HPC tendrán asociadas un VLAN *tag*.

3.2 Requerimientos y consideraciones de seguridad

Los *slices* HPC implementados sin ningún mecanismo de filtración de datos son vulnerables a ataques a través de la red. Principalmente se pueden identificar 2 tipos de atacantes: (i) atacante *outsider* y (ii) atacante *insider*. El atacante *outsider* es un tipo de atacante que no tiene asignado ningún recurso dentro del sistema ni administra nada dentro del mismo. El atacante *insider* se caracteriza porque forma parte del entorno desplegado, desempeñando un rol específico en este.

Atacantes externos u *outsiders* se conectan a través de una red exterior con la capacidad de enrutar tráfico hasta el clúster de servidores. Por tanto, se requiere bloquear cualquier comunicación del exterior que no sea un tráfico permitido explícitamente en las redes de acceso de cada *slice* HPC. Según el requerimiento del cliente, las reglas de acceso varían pues los servicios abiertos de cara al cliente son distintos.

Tanto a los clientes a los que se les asigna un *slice* para la realización de procesos requeridos, como trabajadores responsables de administrar la infraestructura y asignar *slices* a usuarios regulares, se les considera como potenciales atacantes tipo *insider*. Sin embargo, los permisos que posee un trabajador son mucho mayores que los de un cliente; por tanto, se considera que los requerimientos de protección contra ataques de un cliente regular forman un subconjunto de los requerimientos de protección para ataques de trabajadores.

Los atacantes *insiders* comprenden un rango más amplio de preocupaciones al momento de diseñar un mecanismo de seguridad. El motivo es que, a diferencia de los atacantes externos, no solo pueden acceder hasta las redes de acceso de los slices HPC, sino también las redes de administración y datos son vulnerables a sus ataques.

La principal vulnerabilidad sobre la cual se consideran los requerimientos de seguridad para las redes de acceso, administración y datos es la posibilidad de que un atacante *insider* ingrese con permisos de superusuario a un servidor que no forme parte del despliegue de OpenStack, pero que tenga conexión con las interfaces *provider* de los demás nodos conectadas a través de un *switch*.

El atacante puede explotar la vulnerabilidad del *slice* para realizar ataques como *sniffing*, interceptando los datos que atraviesan las redes donde se encuentran conectadas instancias VM. Con esa información recolectada, el atacante puede filtrar el flujo de datos para obtener data sensible de la víctima o comprender un patrón de tráfico y realizar un ataque dirigido.

En la siguiente tabla se resumen los requerimientos de seguridad para cada red que conforma un *slice* HPC. Para cada una de ellas se toma en cuenta el tráfico entrante, saliente e interno a la red.

	Requerimientos de seguridad de red
Red de acceso	**Tráfico entrante**: - Permitir acceso a puertos TCP desde rangos IPs determinados. (Req. 1) - Bloquear todo el tráfico entrante restante. (Req. 2)
	Tráfico saliente: permitir todo el tráfico saliente de cualquier dirección de red perteneciente a una instancia master. (Req. 3)
	Tráfico dentro de la red: permitir la comunicación entre instancias masters pertenecientes al mismo slice HPC. (Req. 4)
Red de administración	**Tráfico entrante**: bloqueado. (Req.5)
	Tráfico saliente: bloqueado. (Req.6)
	Tráfico unicast dentro de la red ● **master** → **worker**: permitido para VMs verificadas. (Req. 7) ● **master** → **master**: permitido para VMs verificadas. (Req. 8) ● **worker** → **master**: permitido para VMs verificadas. (Req. 9) ● **worker** → **worker**: bloqueado. (Req. 10)
	Tráfico multicast dentro de la red ● **master** → **worker**: permitido para VMs verificadas. (Req. 11) ● **master** → **master**: permitido para VMs verificadas. (Req. 12) ● **worker** → **master**: permitido para VMs verificadas. (Req. 13) ● **worker** → **worker**: bloqueado. (Req. 14)
Red de datos	**Tráfico entrante**: bloqueado. (Req. 15)
	Tráfico saliente: bloqueado. (Req. 16)
	Tráfico unicast dentro de la red: permitido para VMs verificadas. (Req. 17)
	Tráfico multicast dentro de la red: permitido para VMs verificadas. (Req. 18)

TABLA 3-1: Requerimientos de seguridad de red

Si el atacante *insider* posee acceso a ejecutar comandos como superusuario en un nodo de cómputo sobre el cual hemos implementado alguna regla o residen máquinas virtuales, no se puede garantizar la seguridad en la red. El atacante tiene la capacidad de sobrescribir el comportamiento de seguridad deseado. Por lo que el presente trabajo de tesis no busca lograr la protección contra ataques de *insiders* con permisos de superusuario en nodos computes donde existan máquinas virtuales de slices HPC.

3.3 Implementación de las reglas de seguridad

3.3.1 Reglas de seguridad en la red de acceso

Para la implementación de la seguridad en la red de acceso de un *slice* HPC se propone la creación de un *security group* propio del *slice* donde se insertarán las reglas. Cabe notar que las reglas del *security group* variarán según los requerimientos del cliente al cual se le asignará el *slice*. Es decir, el cliente puede requerir distintas configuraciones de puertos TCP abiertos y rangos IP permitidos a los cuales necesite acceder.

A continuación, se lista la implementación de las reglas de seguridad, en base al cumplimiento de los requerimientos para la red de acceso:

- Req. 1: implementado agregando una regla de *security group* que permita el tráfico entrante para cada puerto TCP permitido o todos los puertos TCP con origen desde un rango IP específico.
- Req. 2: implementado usando un *security group*. El comportamiento por defecto de los *security groups* es bloquear todo el tráfico entrante.
- Req. 3: implementado agregando una regla de *security group* que permita todo el tráfico saliente (0.0.0.0/0).
- Req. 4: implementado asignando al puerto de acceso de las instancias de un *slice* HPC el mismo *security group* y agregando una regla que permita el tráfico *ingress* de aquellas instancias que tengan ese *security group* asignado (*remote security group*).

En la siguiente tabla se muestra un ejemplo con reglas de *security group* que se verían en el dashboard de OpenStack luego de aplicar la seguridad para la red de acceso del *slice* HPC.

Requerimiento	Dirección	EtherType	IP Protocol	Rango de puertos	Prefijo IP remoto	Security group remoto
Req. 1	Ingress	IPV4	TCP	8080	192.168.1.0/24	-
Req. 1	Ingress	IPV4	TCP	Any	10.1.1.0/24	-
Req. 3	Egress	IPV4	Any	Any	0.0.0.0/0	-
Req. 4	Ingress	IPV4	Any	Any	-	SG actual

TABLA 3-2: Ejemplo de implementación de seguridad con *security groups* en la red de acceso

3.3.2 Reglas de seguridad en las redes de administración y datos

Para implementar la seguridad en las redes de administración y datos de un *slice* HPC se toma en cuenta la imposibilidad de usar un método propio o nativo de OpenStack para aplicar seguridad (como *security groups*), debido al *mechanism driver* elegido para la red (SR-IOV). Por tanto, se plantea el uso de *flows* insertados con un controlador SDN en el *switch* de datos. De esta manera, cualquier tráfico entre instancias en nodos de cómputo diferentes y perteneciente a este tipo de red será filtrado conforme a los requerimientos establecidos.

El conjunto de posibilidades según el tipo de máquina virtual (*master/worker*) de origen/destino y el tipo de comunicación (*unicast/multicast*) en la red (identificada mediante el VLAN *tag*) se traducen a un conjunto de *flows* que puedan ser ingresados en el *switch*, con una acción que cumpla con las políticas de seguridad.

Como parte del análisis de la construcción de flows para la implementación de la seguridad, se consideran factores como la cantidad de *flow entries* (y por tanto el uso de TCAMs en el *switch*), la cantidad de OpenFlow *tables* y el soporte del *switch* y controlador SDN para los *matches* y *actions* de *flows*. Tomando en cuenta estas múltiples variables, se han construido tres versiones distintas como opciones de implementación de *flows* (tanto para la red de administración como la red de datos) que apliquen la seguridad a la red. Todas estas opciones han

sido construidas usando OpenFlow versión 1.3. A continuación se lista, de forma general, cada una de las opciones:

- Opción 1: *flows* usando *unicast* y *broadcast* distinguidos (1 OpenFlow *table*)
- Opción 2: optimización del número de *flow entries* usando
- multiple *tables*
- Opción 3: *flows* para la implementación en el *switch* hibrido de la nube privada de la sección Ingeniería de las Telecomunicaciones de la PUCP, con soporte OpenFlow limitado.

La **opción 1** de implementación de *flows* emplea una única tabla OpenFlow y no toma en consideración la cantidad de *flows* que se vayan a implementar. Es decir, esta opción, no optimizada en el consumo de TCAMs, solo busca agrupar con la menor cantidad de tipos de *flows* el tráfico que será permitido o bloqueado para la red en cuestión. Esta opción requiere que el *switch* y en controlador SDN tengan la capacidad de hacer *match* a las MACs con el *multicast* MAC *bit* configurado (bit número 41, dst_mac[40] == 1)

La **opción 2** de implementación de flows busca una optimización en la cantidad de *flow entries*, en cuanto a que se emplean múltiples tablas para reducir la cantidad de *flows*. La forma de implementar los flows será permitiendo la comunicación inicial de máquinas virtuales permitidas en una primera tabla y redirigir los paquetes a otras tablas, según el origen del paquete (*master* o *worker*). Esta opción, a parte del requerimiento de la opción 1 para el *match* con las MACs *multicast*, requiere que el *switch* y el controlador SDN tengan la capacidad de soportar múltiples tablas OpenFlow y aplicar el *action* de redirigir los paquetes a otras tablas (Goto *table*).

La **opción 3** de implementación de *flows* comprende los *flows* que serán usados en el *switch* hibrido de la nube privada de la sección Ingeniería de las Telecomunicaciones de la PUCP. En el entorno de implentación se usará el *switch* Brocade VDX 6740 (con sistema operativo Network OS 7.0.2) y el

controlador SDN OpenDaylight Oxygen (versión 0.8.4). De la documentación del sistema operativo del *switch* referente a SDN [57] se precisa que el *switch* solo soporta una única *flow table* para instrucciones OpenFlow (y, por tanto, el *action* Goto-Table tampoco es soportado). Además, la interfaz RESTCONF de OpenDaylight no permite ingresar *flows* con *match* a una dirección MAC usando una máscara; por tanto, el *match* para tráfico *multicast* no se hará con un *flow* que coincida explícitamente con este tipo de tráfico, sino que se segmentará con el uso de prioridades.

A continuación, se detallarán las tres opciones de implementación de *flows*, tanto para la red de administración, como para la red de datos. Para cada opción se tomará en cuenta las propiedades de cada tipo de *flow*: la función que cumple, la tabla OpenFlow en la que reside, la prioridad, la cantidad de *flows* a implementarse, los requerimientos de seguridad con los que cumple, el *match* y el *action*,

Se lista la leyenda de abreviaciones usada en las tablas de opciones de *flows*:

- m: número de VMs con rol *master* del *slice* HPC
- w: número de VMs con rol *worker* del *slice* HPC
- OF_SRC_PORT_master / OF_SRC_PORT_worker: ID del puerto OpenFlow (Node Connector Id) al que se conecta el nodo *compute* donde reside la VM *master/worker* origen. P.ej.: openflow:431646109239664:8
- OF_DST_PORT_master / OF_DST_PORT_worker: ID del puerto OpenFlow (Node Connector Id) al que se conecta el nodo *compute* donde reside la VM *master/worker* destino. P.ej.: openflow:431646109239664:8
- G_masters: *group table* (*type* ALL) conformado por un *bucket* para cada ID de puerto OpenFlow (Node Connector Id) al que se conecta el nodo compute donde reside cada VM *master* del *slice* HPC.
- G_workers: *group table* (*type* ALL) conformado por un *bucket* para cada ID de puerto OpenFlow (Node Connector Id) al que se conecta el nodo *compute* donde reside cada VM *worker* del *slice* HPC.

- G_all: *group table* (*type* ALL) conformado por un *bucket* para cada ID de puerto OpenFlow (Node Connector Id) al que se conecta el nodo *compute* donde reside cada VM *master* y *worker* del *slice* HPC.
- [BROAD]: cuando el paquete tiene el *bit* de *multicast* habilitado, en una red *legacy* significa que debe ser enviado a todos los nodos de la subred. El caso más típico es de una trama *broadcast*.
- [?]: resto de direcciones MAC que no han hecho *match* hasta el momento. Por ejemplo, MAC de VM desconocidas y direcciones *broadcast* cuando ya se han procesado todas las reglas para direcciones *unicast* conocidas

3.3.2.1 Implementación de las reglas de seguridad en la red de administración

En todas las opciones de *flows* para la red de administración, se han ordenado los tipos de *flows* por fuente; primero las VMs tipo *master* como origen y luego las VMs tipo *worker* como origen, ya sea tráfico *unicast* o *multicast*.

Para las tres opciones de implementación de seguridad en la red de administración, se debe considerar que ninguno de los *flows* excluye que paquetes *broadcast* originados en un *worker* le lleguen a otros *workers* co-localizados en el mismo nodo compute donde hay un *master* del mismo *slice* HPC. El motivo es que este tráfico no atraviesa el *switch* de datos donde se aplican los *flows*. Este hecho no pone en riesgo la seguridad pues los *masters/workers* son confiables entre sí.

La opción 1 de implementación de *flows* en la red de administración suma un total de reglas igual a: 2mw + w + m^2

Flow	**Match**	**Action**
TABLE: 0 (default rule: DROP); **Req.**: 5,6,10		
1: aceptar tráfico master → worker master → master **priority**: 4; **# flows**: (m*w) + m(m-1) **Req.**: 7,8	**- in-port**: OF_SRC_PORT_master **- dl_src**: MAC de la VM master origen **- dl_dst**: MAC de la VM destino **- dl_vlan**: VLAN ID de la red de administración	**Output**: Port ID = OF_DST_PORT_master/ OF_DST_PORT_worker
2: aceptar tráfico master → [BROAD] **priority**: 3; **# flows**: (m); **Req.**: 11,12	**- in-port**: OF_SRC_PORT_master **- dl_src**: MAC de la VM master origen **- dl_dst**: multicast MAC bit set, dst_mac[40] == 1 **- dl_vlan**: VLAN ID de la red de administración	**Group**: Group ID = G_all
3: aceptar tráfico worker → master **priority**: 2; **# flows**: (w*m) **Req.**: 9	**- in-port**: OF_SRC_PORT_worker **- dl_src**: MAC de la VM worker origen **- dl_dst**: MAC de la VM master destino **- dl_vlan**: VLAN ID de la red de administración	**Output**: Port ID = OF_DST_PORT_master
4: aceptar tráfico worker → [BROAD] **priority**: 1; **# flows**: (w) **Req.**: 13,14	**- in-port**: OF_SRC_PORT_worker **- dl_src**: MAC de la VM worker origen **- dl_dst**: multicast MAC bit set, dst_mac[40] == 1 **- dl_vlan**: VLAN ID de la red de administración	**Group**: Group ID = G_masters

TABLA 3-3: Opción 1 – Implementación de *flows* en la red de administración

La opción 2 para la implementación de *flows* en la red de administración reduce la cantidad de *flows* total a: 2w + 3m + 2

Flow	**Match**	**Action**
TABLE: 0 (default rule: DROP); **Req.**: 5,6		
1: permitir comunicación inicial (unicast/multicast) de masters confiables **table**: 0; **priority**: 2; **# flows**: (m); **Req.**: 7,8,11,12	**- in-port**: OF_SRC_PORT_master **- dl_src**: MAC de la VM master origen **- dl_vlan**: VLAN ID de la red de administración	**Goto table**: Table ID = MASTER_SRC
2: permitir comunicación inicial (unicast/multicast) de workers confiables **table**: 0; **priority**: 1; **# flows**: (w); **Req.**: 9,10,13,14	**- in-port**: OF_SRC_PORT_worker **- dl_src**: MAC de la VM worker origen **- dl_vlan**: VLAN ID de la red de administración	**Goto table**: Table ID = WORKER_SRC
TABLE: MASTER_SRC (default rule: DROP)		
3: aceptar tráfico master → master/worker **table**: MASTER_SRC; **priority**: 2; **# flows**: (m+w); **Req.**: 7,8	- **ethernet-destination**: MAC de la VM destino - **vlan-id**: VLAN ID de la red de administración	**Output**: Port ID = OF_DST_PORT_master/ OF_DST_PORT_worker
4: aceptar tráfico master → [BROAD] **table**: MASTER_SRC; **priority**: 1; **# flows**: 1; **Req.**: 11,12	**- dl_dst**: multicast MAC bit set, dst_mac[40] == 1 **- dl_vlan**: VLAN ID de la red de administración	**Group**: Group ID = G_all
TABLE: WORKER_SRC (default rule: DROP); **Req.**: 10		
5: aceptar tráfico worker → master **table**: WORKER_SRC; **priority**: 2; **# flows**: (m); **Req.**: 9	- **ethernet-destination**: MAC de la VM master destino - **vlan-id**: VLAN ID de la red de administración	**Output**: Port ID = OF_DST_PORT_master
6: aceptar tráfico worker → [BROAD] **table**: WORKER_SRC; **priority**: 1; **# flows**: 1; **Req.**: 13,14	**- dl_dst**: multicast MAC bit set, dst_mac[40] == 1 **- dl_vlan**: VLAN ID de la red de administración	**Group**: Group ID = G_masters

TABLA 3-4: Opción 2 – Implementación de *flows* en la red de administración

La opción 3 de implementación de *flows* en la red de administración da el mayor valor total respecto a las opciones previas con una cantidad de *flows* igual a: 2mw + 2w + m^2

Flow	**Match**	**Action**
TABLE: 0 (default rule: DROP); **Req.**: 5,6		
1: aceptar tráfico master → worker master → master **priority**: 5; **# flows**: (m*w) + m(m-1) **Req.**: 7,8	**- in-port**: OF_SRC_PORT_master **- dl_src**: MAC de la VM master origen **- dl_dst**: MAC de la VM destino **- dl_vlan**: VLAN ID de la red de administración	**Output**: Port ID = OF_DST_PORT_master/ OF_DST_PORT_worker
2: aceptar tráfico master → [?] **priority**: 4; **# flows**: (m); **Req.**: 11,12	**- in-port**: OF_SRC_PORT_master **- dl_src**: MAC de la VM master origen **- dl_vlan**: VLAN ID de la red de administración	**Group**: Group ID = G_all
3: aceptar tráfico worker → master **priority**: 3; **# flows**: (w*m) **Req.**: 9	**- in-port**: OF_SRC_PORT_worker **- dl_src**: MAC de la VM worker origen **- dl_dst**: MAC de la VM master destino **- dl_vlan**: VLAN ID de la red de administración	**Output**: Port ID = OF_DST_PORT_master
4: bloquear tráfico [?] → worker **priority**: 2; **# flows**: (w) **Req.**: 10	**- dl_dst**: MAC de la VM worker destino **- dl_vlan**: VLAN ID de la red de administración	**DROP**
5: aceptar tráfico worker → [?] **priority**: 1; **# flows**: (w) **Req.**: 13,14	**- in-port**: OF_SRC_PORT_worker **- dl_src**: MAC de la VM worker origen **- dl_vlan**: VLAN ID de la red de administración	**Group**: Group ID = G_masters

TABLA 3-5: Opción 3 – Implementación de *flows* en la red de administración

3.3.2.2 Implementación de las reglas de seguridad en la red de datos

La opción 1 para la implementación de *flows* en la red de datos da una suma total de *flows* igual a: w^2

Flow	**Match**	**Action**
TABLE: 0 (default rule: DROP); **Req.**: 15,16		
1: aceptar tráfico worker → worker **priority**: 2; **# flows**: w(w-1) **Req.**: 17	**- in-port**: OF_SRC_PORT_worker **- dl_src**: MAC de la VM worker origen **- dl_dst**: MAC de la VM worker destino **- dl_vlan**: VLAN ID de la red de datos	**Output**: Port ID = OF_DST_PORT_worker
2: aceptar tráfico worker → [BROAD] **priority**: 1; **# flows**: (w) **Req.**: 18	**- in-port**: OF_SRC_PORT_worker **- dl_src**: MAC de la VM worker origen **- dl_dst**: multicast MAC bit set, dst_mac[40] == 1 **- dl_vlan**: VLAN ID de la red de datos	**Group**: Group ID = G_workers

TABLA 3-6: Opción 1 – Implementación de *flows* en la red de datos

La opción 2 de implementación de *flows* para la red de datos logra reducir la cantidad total de *flows* a: 2w + 1. Emplear múltiples tablas favorece el hecho de que la cantidad de reglas no incremente de forma cuadrática (w^2), sino de forma lineal (2w).

Flow	**Match**	**Action**
TABLE: 0 (default rule: DROP); **Req.**: 15,16		
1: permitir comunicación inicial (unicast/multicast) de workers confiables **table**: 0; **priority**: 1; **# flows**: (w); **Req.**: 17,18	**- in-port**: OF_SRC_PORT_worker **- dl_src**: MAC de la VM worker origen **- dl_vlan**: VLAN ID de la red de administración	**Goto table**: Table ID = WORKER_SRC
TABLE: WORKER_SRC (default rule: DROP)		
2: aceptar tráfico worker → worker **table**: WORKER_SRC; **priority**: 2; **# flows**: (w); **Req.**: 17	- **dl_dst**: MAC de la VM worker destino - **vlan-id**: VLAN ID de la red de datos	**Output**: Port ID = OF_DST_PORT_worker
3: aceptar tráfico worker → [BROAD] **table**: WORKER_SRC; **priority**: 1; **# flows**: 1; **Req.**: 18	**- dl_dst**: multicast MAC bit set, dst_mac[40] == 1 **- dl_vlan**: VLAN ID de la red de datos	**Group**: Group ID = G_ workers

TABLA 3-7: Opción 2 – Implementación de *flows* en la red de datos

La opción 3 de implementación de *flows* para la red de datos suma la misma cantidad total de *flows* que la opción 1: w^2. El comportamiento para tráfico *unicast* es parecido al de los *switch legacy* cuando una dirección MAC destino es desconocida pues el paquete se envía por todos los puertos. Sin embargo, el *source* (*worker*) sigue siendo confiable.

Flow	**Match**	**Action**
TABLE: 0 (default rule: DROP); **Req.**: 15,16		
1: aceptar tráfico worker → worker **priority**: 2; **# flows**: w(w-1) **Req.**: 17	**- in-port**: OF_SRC_PORT_worker **- dl_src**: MAC de la VM worker origen **- dl_dst**: MAC de la VM worker destino **- dl_vlan**: VLAN ID de la red de datos	**Output**: Port ID = OF_DST_PORT_worker
2: aceptar tráfico worker → [?] **priority**: 1; **# flows**: (w) **Req.**: 18	**- in-port**: OF_SRC_PORT_worker **- dl_src**: MAC de la VM worker origen **- dl_vlan**: VLAN ID de la red de datos	**Group**: Group ID = G_workers

TABLA 3-8: Opción 3 – Implementación de *flows* en la red de datos

3.3.3 Resumen de la implementación de las reglas de seguridad

En la siguiente tabla, para cada red, se relacionan las reglas de seguridad y el (o los) requerimiento(s) que satisface(n) cada una de ellas. Las reglas usadas para la red de administración y la red de datos empleadas en esta tabla se basan en la opción de *flows* para la implementación en el entorno de pruebas (opción 3).

Red	**Requerimiento**	**Regla de seguridad**
Red de acceso	1	Regla de security group que permita el tráfico entrante para cada puerto TCP permitido o todos los puertos TCP con origen desde un rango IP específico.
	2	Usando security groups (bloquea todo el tráfico entrante)
	3	Regla de security group que permita todo el tráfico saliente.
	4	Regla de security group que permita el tráfico ingress de aquellas instancias que tengan ese security group asignado.
Red de administración	5, 6	Default rule (table 0): descartar tráfico
	7, 8	Flow 1: aceptar tráfico master → worker, master → master
	9	Flow 3: aceptar tráfico worker → master
	10	Flow 4: bloquear tráfico [?] → worker
	11,12	Flow 2: aceptar tráfico master → [?]
	13, 14	Flow 5: aceptar tráfico worker → [?]
Red de datos	15, 16	Default rule (table 0): descartar tráfico
	17	Flow 1: aceptar tráfico worker → worker
	18	Flow 2: aceptar tráfico worker → [?]

TABLA 3-9: Resumen de las reglas de seguridad de red en base a requerimientos

3.4 Arquitectura de la solución

La arquitectura de la solución propuesta encaja en un sistema mayor llamado HAST. El presente trabajo de tesis propone una solución de ciberseguridad del HAST para *slices* HPC, cuya arquitectura es un subconjunto de la arquitectura del HAST. Esta nueva aplicación escrita en lenguaje Python (con el *framework* Django) interactúa con otros sistemas de *software* de código abierto: OpenStack y OpenDaylight. En la siguiente imagen se presenta un diagrama de contexto del sistema que se ha modificado para soportar seguridad en *slices* HPC.

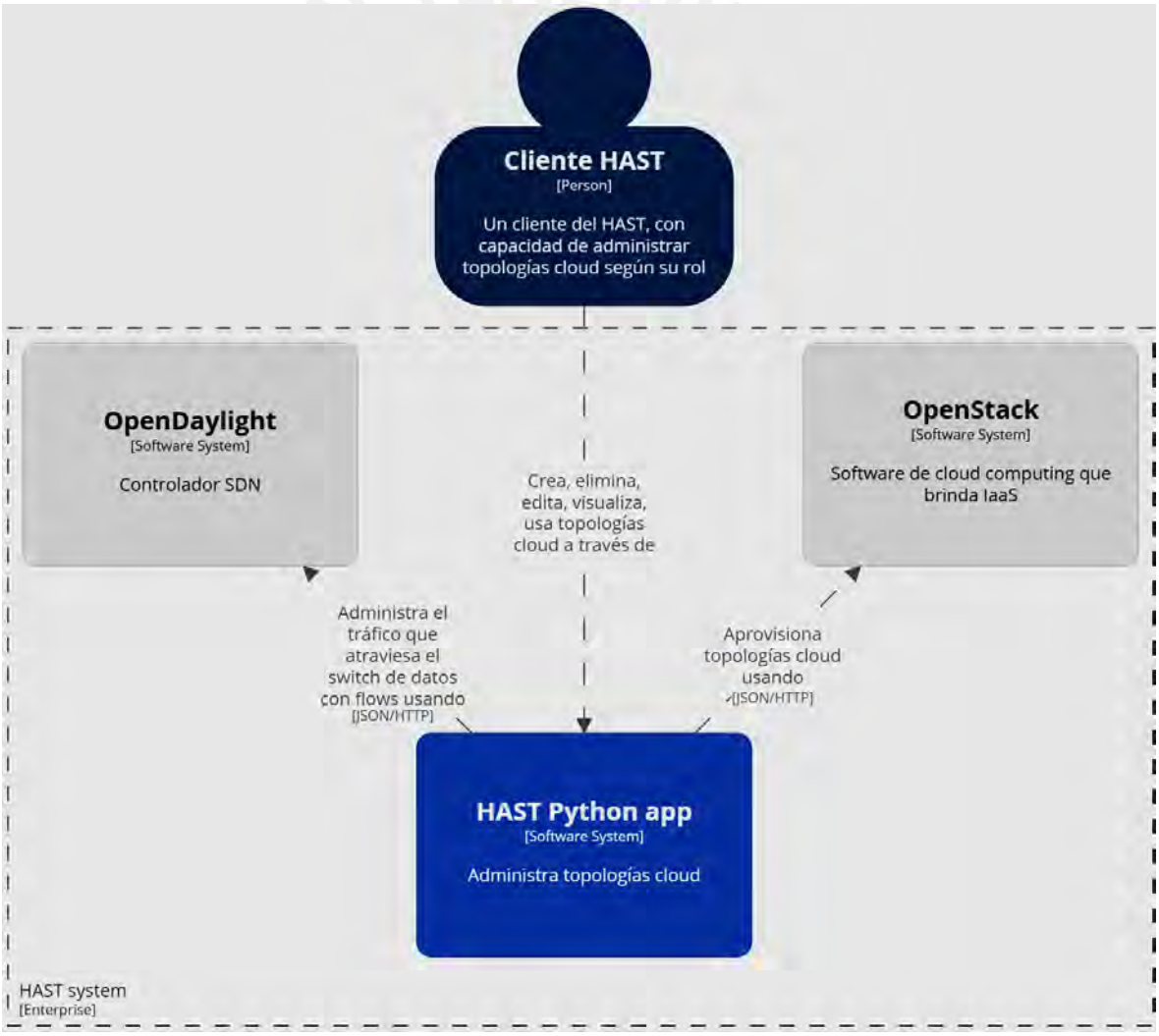

FIGURA 3-2: Diagrama de contexto del sistema

HAST es capaz de recibir solicitudes de clientes para la gestión de topologías *cloud* aprovisionadas con soluciones virtuales a través de OpenStack. Cuando esto ocurre, HAST establece comunicaciones con OpenStack y OpenDaylight mediante *requests* a sus respectivas APIs. Se ha agregado al sistema el controlador SDN de OpenDaylight para habilitar un manejo del tráfico de datos a través de *flows* insertados en un *switch* OpenFlow que conecta los nodos de cómputo. A continuación, se detallan las funciones y características de todos los elementos de la solución.

3.4.1 Cliente HAST

El cliente HAST interactúa con la aplicación, a través de una interfaz gráfica o consumiendo la API que tiene disponible. Si el rol del cliente es de “jefe de práctica” (equivalente a un administrador) este puede crear, eliminar, visualizar y usar topologías cloud; en cambio, si su rol es de “alumno” (equivalente a un usuario regular) tendrá la posibilidad solo de visualizar y usar las topologías asignadas.

Básicamente, el cliente HAST se comunica con la aplicación para el aprovisionamiento y administración de infraestructura *cloud* y la aplicación HAST, a su vez, se comunica con OpenStack para llevar a cabo las solicitudes si cumple con los permisos.

3.4.2 OpenStack

En el contexto del sistema desarrollado OpenStack ofrece sus capacidades como sistema operativo *cloud* para manejar todos los elementos correspondientes a una topología *cloud*. HAST es un cliente de OpenStack que consume sus APIs: OpenStack Identity para la generación de *tokens*, OpenStack Networking para la administración de elementos de red y OpenStack Compute para el manejo de instancias de máquinas virtuales.

3.4.2.1 Configuración de OpenStack

Las herramientas básicas para el funcionamiento de OpenStack que deben agregarse al sistema son un servidor Network Time Protocol, una base de datos SQL como MariaDB, un *software* de negociación de mensajes como RabbitMQ que implementa el estándar Advanced Message Queuing Protocol (AMQP), un sistema distribuido para caché basado en memoria como Memcached y un *software* de almacenamiento de configuraciones como etcd. Luego, los proyectos de OpenStack mínimos a ser desplegados son Keystone, Glance, Nova y Neutron.

Para Nova, que implementa la API de Compute de OpenStack, es importante asegurarse que el nodo de cómputo soporte aceleración de *hardware*, lo cual no requiere ninguna configuración adicional; de no ser el caso, debemos configurar Libvirt para que use QEMU en lugar de KVM [58]. Para la habilitación de SR-IOV en OpenStack, debemos configurar una *whitelist* con los dispositivos PCI a los cuales se les permitirá intercambiar tráfico en cada nodo de cómputo. Esto permitirá que todas las VFs que pertenezcan a un interfaz de red específica tengan permitido hacer un *passthrough* a las instancias directamente. Finalmente, en el nodo *controller*, es necesario configurar el nova-scheduler agregando el filtro PCI a la lista de filtros habilitados [21].

La configuración realizada para el servicio de Neutron, que implementa la API de Networking de OpenStack, consta de configurar un *mechanism driver* típicamente usado con ML2, como los son linuxbridge u openvswitch. Para este *mechanism driver* se habilita *security groups* tanto en el nodo *controller* como en los nodos de cómputo. Junto con el *mechanism driver* ya existente, se aumenta el *mechanism driver* de sriovnicswitch para que sea posible crear redes compatibles con SR-IOV. Luego, se debe configurar el rango de VLANs asignadas para cada *mechanism driver*.

Siguiendo con la configuración de Neutron, se debe instalar y configurar el agente SR-IOV (neutron-sriov-nic-agent) en cada nodo de cómputo, agregando

el mapeo entre la red física y la NIC SR-IOV. En el mismo archivo de configuración de SR-IOV es necesario agregar una línea que indique que se deshabilita el *feature* de OpenStack *firewall*, es decir, no se usará *security groups* con redes SR-IOV. Finalmente se habilita el servicio del agente SR-IOV [58].

3.4.3 OpenDaylight

Dentro del sistema, OpenDaylight cumplirá el rol de controlador SDN para configurar *flows* en el *switch* de datos compatible con OpenFlow. Con esta adición, los datos ya no atravesarán un *switch* de datos convencional (Ethernet switching), sino que, para las redes dispuestas tendrán un *match* de *flow* que conozca cómo administrar el tráfico entre instancias de distintos nodos de cómputo. De esta manera, las VMs que usen redes tipo SR-IOV no estarán desprovistas totalmente de protección.

3.4.3.1 Configuración de OpenDaylight

La configuración de OpenDaylight no requerirá un gran nivel de personalización, pero será necesario contar con ciertos módulos y *plugins* para que el sistema funcione como se prevé. Se requerirá del módulo RESTCONF para realizar peticiones HTTP al controlador OpenDaylight que permitan insertar nuevos *flows* representando reglas de seguridad para las instancias. Además, se necesita tener agregado el *plugin* de OpenFlow para OpenDaylight, para que el controlador pueda intercambiar mensajes con el *switch* OpenFlow.

El *switch* de datos OpenFlow debe configurarse para que apunte hacia el controlador desplegado en un servidor. De forma predeterminada, el servicio de OpenDaylight está corriendo en el puerto TCP 6633, al cual debe apuntar el *switch*, junto con la dirección IP del controlador. Además, es necesario especificar la versión OpenFlow que será empleada para evitar problemas de compatibilidad; en este caso OpenFlow 1.3.

3.4.4 Hast Python app

La aplicación HAST es el orquestador con capacidades de administrar topologías *cloud* mediante solicitudes a OpenStack. HAST recibe las solicitudes de los clientes de la aplicación y las traduce en nuevas llamadas a OpenStack a través de su REST API.

HAST está dividido en diferentes módulos que comprenden la solución, con el fin de separar funciones y hacer que se puedan realizar modificaciones en cada uno de los módulos de forma separada y en paralelo. En el siguiente diagrama se descompone el HAST en subsistemas desplegables de forma independiente, solo considerando aquellos relevantes para el desarrollo de la solución propuesta en el presente trabajo.

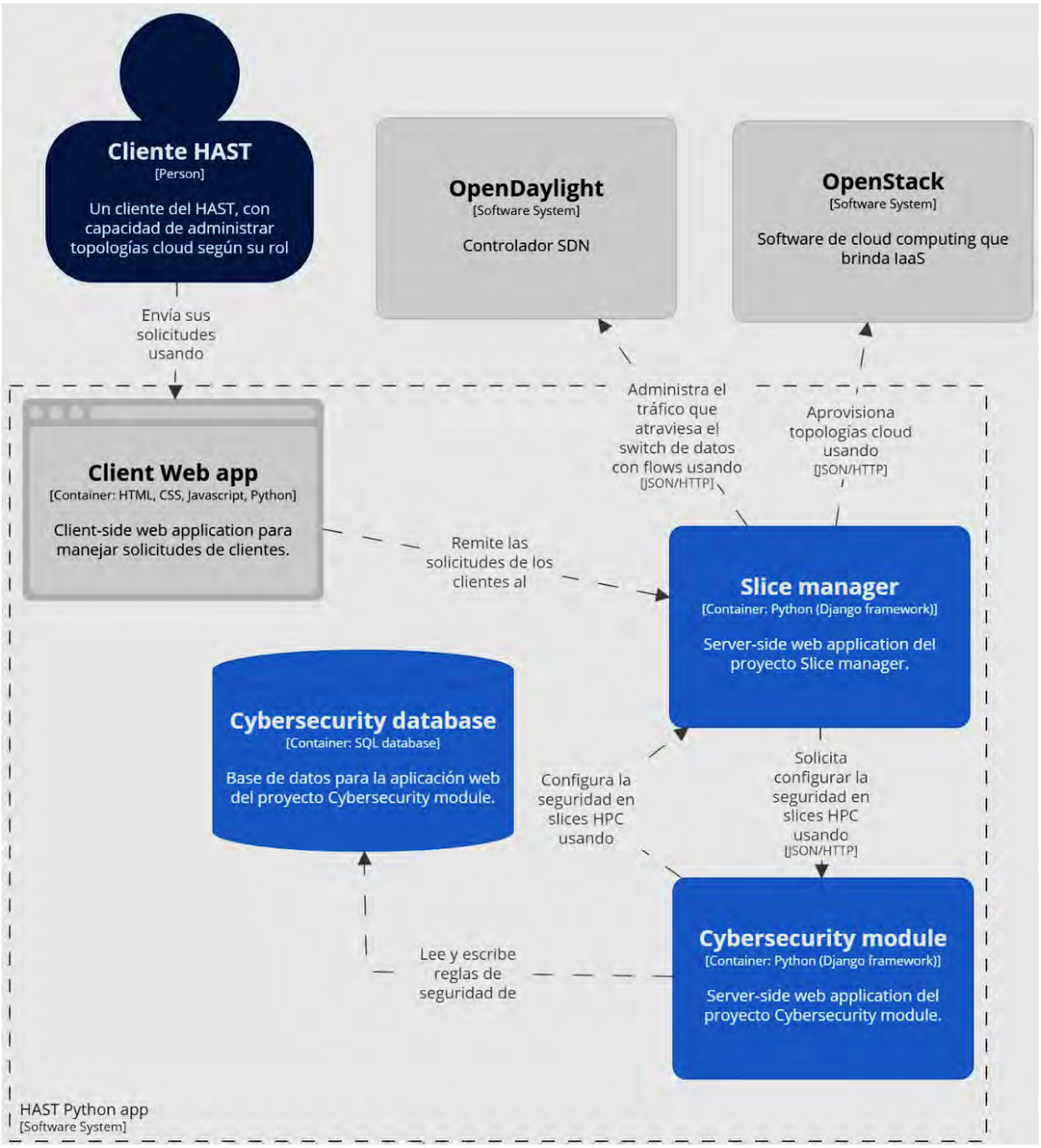

FIGURA 3-3: Diagrama a nivel de contenedor de la solución

3.4.4.1 Client web app

El módulo de aplicación *web* del cliente es un módulo existente, que forma parte del HAST, se encarga de cargar código HTML, CSS y Javascript del lado del cliente para que a través de este se puedan comunicar y enviar solicitudes. Mediante esta interfaz, los clientes también pueden ver de forma gráfica la topología del *slice* que se ha desplegado. Las solicitudes que se reciban de los clientes mediante formularios en la interfaz web se remiten al módulo de *slice manager*.

3.4.4.2 Slice manager

El módulo de *slice* manager es el módulo central encargado de procesar todas las solicitudes que llegan de los clientes y contactarse con otros módulos de la aplicación HAST para llevar a cabo tareas propias de la interacción con topologías *cloud*.

Se propone hacer modificaciones al *slice* manager agregando nuevas funcionalidades dentro del código existente con el fin de soportar la implementación de seguridad para *slices* HPC. El motivo de la elección de este módulo como parte del flujo de la aplicación de seguridad en slices HPC es que, al ser el módulo núcleo del sistema, es el elemento ideal para ejecutar tareas relevantes a la ciberseguridad y comunicarse con otros módulos tanto internos como externos al sistema.

El *slice manager*, como componente que participa en configurar la seguridad, se comunica con el módulo de ciberseguridad para enviar solicitudes de configuración de reglas de seguridad en *slices* HPC. También se responsabiliza de comunicarse con sistemas externos al HAST. Envía solicitudes a OpenStack para el manejo de *security group* y *security group rules*; así como solicitudes a OpenDaylight para insertar *flows* que eviten ataques contra las instancias VM que se encuentran en los nodos de cómputo. Para obtener un grado de escalabilidad y orden en el código, se opta por un componente central como lo

es el *slice manager*, para que conozca la sintaxis de las reglas de seguridad que se están insertando en ambos sistemas.

3.4.4.3 Módulo de ciberseguridad (Cybersecurity module)

El módulo de ciberseguridad tiene la función de procesar las solicitudes del *slice manager* para administrar (ver, aplicar, remover) la seguridad de un *slice* HPC y retransmitir esas solicitudes al *slice manager* y a la base de datos de ciberseguridad.

El módulo de ciberseguridad se encarga de ingresar las solicitudes correspondientes al *slice manager* para que se comunique con los módulos externos (OpenStack y OpenDaylight) que son los componentes finales responsables de aplicar la seguridad. La razón de separar las funciones de solicitud del *slice manager* es que se requiere de un componente distinto que entienda del estado de la seguridad en los slices HPC.

Asimismo, el módulo de ciberseguridad entabla un intercambio de peticiones, leyendo y escribiendo, en la base de datos del módulo de ciberseguridad para poder configurar las propiedades de seguridad que existen sobre un *slice* HPC.

3.4.4.4 Base de datos de ciberseguridad (Cybersecurity database)

La base de datos de ciberseguridad es un componente del sistema que recibe solicitudes para conocer y modificar el estado de la seguridad de un *slice* HPC. Ya que se requiere tener un registro de la seguridad configurada en los *slices* HPC, se opta por manejar una base de datos simple, de una sola tabla, que almacene esta información.

La estructura de una entrada de base de datos se compone de 5 propiedades:

- **Slice ID**: identificador único del *slice* HPC
- **Estado de la seguridad**: habilitado/deshabilitado
- **Request body**: objeto JSON con información requerida por el *slice* manager para aplicar la seguridad en el *slice* HPC
- **Datos de seguridad en OpenStack**: datos sobre el *security group* y reglas ingresadas para la red de acceso del *slice* HPC.
- **Datos de seguridad en OpenDaylight**: *flows* ingresados en el *switch* de datos para la seguridad en la red de administración y datos del *slice* HPC.

3.5 Diseño de la solución (High level design)

Para el diseño de la solución se ha considerado hacer modificaciones y agregar nuevas funcionalidades sobre el código de HAST existente. Al *slice manager*, módulo que ya existe en el HAST, se le han agregado funciones para soportar la comunicación con los componentes externos con el fin de implementar la seguridad en slices HPC. El módulo de ciberseguridad es una nueva aplicación web desarrollada con el fin de mandar las órdenes de configuración de seguridad, así como para comunicarse con la base de datos de ciberseguridad.

3.5.1 Diseño del slice manager

El módulo de *slice manager* está compuesto por otros submódulos encargados de centralizar la implementación de *slices* y proveer la función de administrarlos. Para el caso de slices HPC, se pretende agregar nuevos submódulos con el objetivo de que este tipo de *slices* tengan un nivel de protección frente ataques cibernéticos por red. En el siguiente diagrama se muestran los componentes del *slice manager* y la interacción entre ellos, así como con sistemas externos.

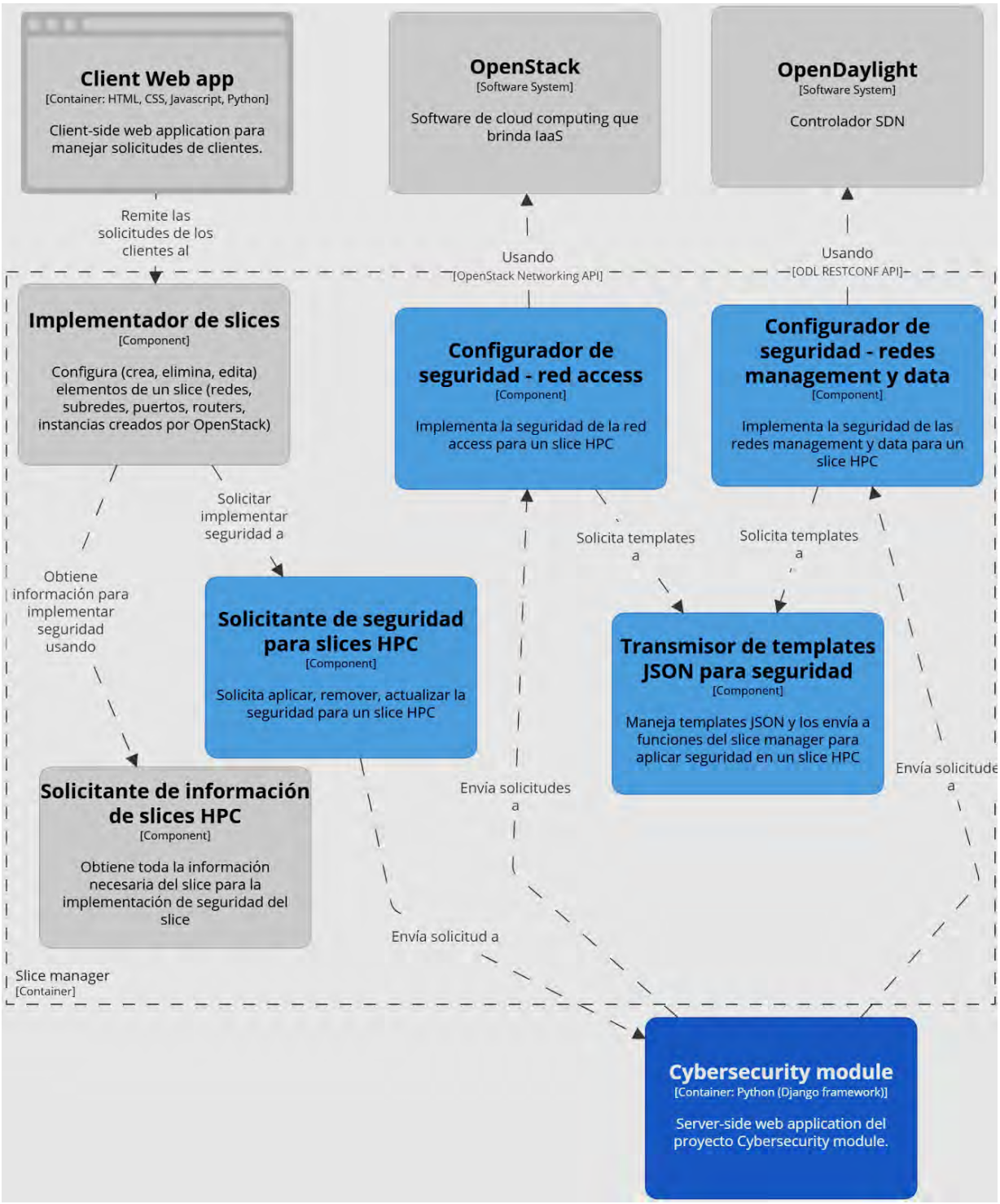

FIGURA 3-4: Diagrama de componentes del contenedor *slice manager*

3.5.1.1 Implementador de slices

El implementador de *slices* es un componente del *slice manager* que posee la capacidad de configurar (crear, eliminar, editar) elementos de un *slice* (redes, subredes, puertos, *routers*, instancias) mediante OpenStack. Este submódulo puede entregar al cliente una variedad de topologías según los requerimientos insertados como entrada. El componente no sufrirá modificaciones en el diseño planteado, pero forma parte del flujo en la configuración de seguridad de slices HPC.

El implementador de *slices* debe recoger la solicitud del cliente, ingresada a través de una interfaz API. En ella se podrá especificar como mínimo, para crear slices HPC con seguridad, la cantidad de VMs *masters*, *workers* y los puertos TCP abiertos y rangos IP permitidos para la red de acceso. Por otro lado, si lo que se desea es eliminar un *slice* HPC y sus reglas de seguridad correspondientes, solo bastará el ID del *slice*, así como la cantidad de VMs *masters* y *workers*.

3.5.1.2 Solicitante de información de slices HPC

El submódulo solicitante de información de slices HPC tiene la responsabilidad de obtener todos los datos necesarios en OpenStack para configurar la seguridad, posterior al aprovisionamiento del *slice*. Este submódulo no hace consultas sobre la red de acceso, sino que las delega al Configurador de seguridad para la red de acceso. El motivo de este diseño es que ya existe una comunicación con OpenStack pensada para la configuración de puertos *access*.

La respuesta que retorna al implementador de slices cuando este solicita obtener información para configurar la seguridad contiene características de las redes de administración y datos del *slice* HPC a configurar. La información comprende las VLANs de ambas redes, MACs y puertos OpenFlow (relacionados a los nodos de cómputo donde están las VMs) de cada instancia VM *master* y *worker* en estas dos redes.

Si bien los datos recolectados por el presente submódulo podrían obtenerse usando el submódulo Configurador de seguridad de las redes de administración y datos, se prefiere separar funciones para obtener los datos de las redes que usarán SR-IOV. De esta forma, el submódulo no hará consultas adicionales a OpenStack previo a aplicar la seguridad con OpenDaylight.

3.5.1.3 Solicitante de seguridad para slices HPC

El submódulo solicitante de seguridad para *slices* HPC se encarga de recibir las peticiones del implementador de *slices* cuando se pasa a configurar la seguridad en un *slice* HPC, ya sea para aplicarla o removerla. El presente submódulo se encarga de transferir las peticiones al módulo de ciberseguridad mediante un *request* HTTP para configurar la seguridad del *slice* en cuestión

3.5.1.4 Transmisor de templates JSON para seguridad

El submódulo Transmisor de *templates* JSON para seguridad maneja formatos para implementar reglas de seguridad específicas; ya sea a través de reglas de *security groups* para la red de acceso o *flows* para las redes de administración y datos. Se emplea el formato de texto JSON para el intercambio de datos gracias a la compatibilidad que ofrecen las APIs de OpenStack y OpenDaylight de trabajar con este formato.

La razón de que estas plantillas residan dentro del *slice manager* se debe a que el diseño de la solución HAST busca centralizar en un solo módulo la sintaxis JSON que se usa para comunicarse tanto con OpenStack como con OpenDaylight para realizar configuraciones. De esta forma, si existe algún cambio en las APIs de los sistemas de *software* externos en una nueva versión de OpenStack/OpenDaylight, los otros módulos no tengan que ser editados.

Para el caso de OpenStack, se tienen plantillas con formatos para la creación de security groups y todas las reglas definidas en la implementación de los

requerimientos para la red de acceso. Para OpenDaylight solo se manejan plantillas con los *flows* que se ingresarán en las redes tipo SR-IOV, donde deben configurarse los campos de id, prioridad, *match* y acción. Todos los valores que requieran una configuración en los *templates* JSON serán especificados por los dos submódulos configuradores de seguridad de sus respectivas redes al llamar las funciones del presente submódulo.

3.5.1.5 Configurador de seguridad - red de acceso

El submódulo Configurador de seguridad para la red de acceso se encarga de recibir las solicitudes del módulo de ciberseguridad y hacer la configuración respectiva para la red de acceso del *slice* HPC que se intenta configurar. El submódulo logra su objetivo enviando solicitudes a OpenStack Networking API.

Si se va a aplicar la seguridad en la red de acceso el flujo que seguirá el configurador luego de recibir la solicitud será crear un *security group*, crear las reglas de *security group* y finalmente configurar ese *security group* en los puertos de acceso de las VMs *masters*. Por otro lado, si se va a remover la seguridad en la red de acceso, el flujo será más simple; primero se desconfigurará el *security group* creado en todos los puertos de acceso de los *masters* y finalmente se eliminará el *security group*.

3.5.1.6 Configurador de seguridad - redes de administración y datos

El submódulo Configurador de seguridad para las redes de administración y datos se ocupa de aceptar las solicitudes que envía el módulo de ciberseguridad para configurar seguridad en las redes con puertos SR-IOV. El submódulo se comunica con OpenDaylight para cumplir con los requerimientos de seguridad de ambas redes.

Para cualquier caso, ya sea que se aplique o remueva seguridad en las redes de administración y datos; primero, se generará un arreglo de *flows* con todas las reglas de seguridad identificadas por un id. Luego, dependiendo de si se va a

aplicar o eliminar los *flows* en el switch OpenFlow, se hará una solicitud HTTP distinta.

3.5.2 Diseño del módulo de ciberseguridad

El módulo de ciberseguridad se encuentra formado por otros submódulos con los objetivos de recepcionar las solicitudes del *slice manager* para aplicar seguridad en *slices* HPC y reenviar mensajes tanto a la base de datos de ciberseguridad como al *slice manager*. En el siguiente diagrama se muestra cómo está compuesto el módulo de ciberseguridad y la interacción con agentes externos.

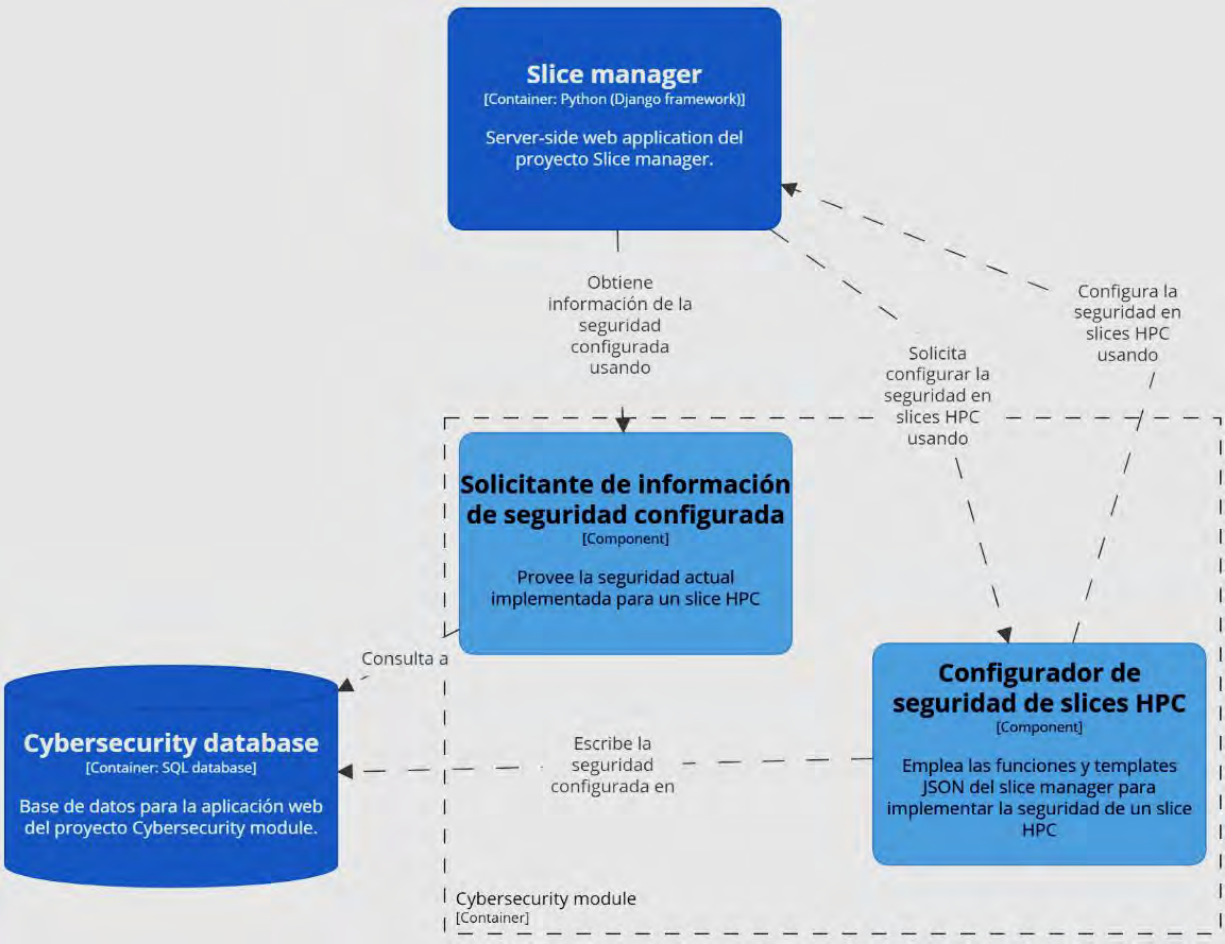

FIGURA 3-5: Diagrama de componentes del contenedor módulo de ciberseguridad

3.5.2.1 Configurador de seguridad de slices HPC

El configurador de seguridad de slices HPC emplea las funciones y *templates* JSON que provee el *slice manager* para implementar la seguridad de un *slice* HPC. Eso lo logra a través de solicitudes hacia el módulo *slice manager*, luego de recibir peticiones del mismo.

Por otro lado, el configurador de seguridad de slices HPC se encarga de ingresar en base de datos, información sobre la seguridad configurada, ya sea que se hayan agregado o eliminado reglas. Además, se guarda el estado de la seguridad (habilitada/deshabilitada) en la entrada correspondiente al *slice* HPC que se está configurando.

3.5.2.2 Solicitante de información de seguridad configurada

El submódulo solicitante de información de seguridad configurada se encarga de recepcionar solicitudes del *slice manager* para mostrar datos de seguridad de todos los *slices* HPC o uno en específico. Esto lo logra a través de consultas a la base de datos de ciberseguridad. Mediante el ID relacionado al *slice* HPC se devuelven los datos del estado de seguridad, reglas de seguridad de OpenStack y reglas de seguridad de OpenDaylight configuradas.

3.5.3 APIs de la solución de ciberseguridad

La solución de ciberseguridad conformada por componentes en el *slice manager* y el módulo de ciberseguridad soporta solicitudes HTTP como parte de la API, usada para la comunicación entre subsistemas y sistemas de *software* exteriores. Mediante este intercambio de mensajes, se logra completar el flujo de configuración de la seguridad en un *slice* HPC.

3.5.3.1 API del slice manager

En la siguiente tabla se resumen los métodos HTTP, las rutas para realizar las solicitudes y la descripción de cada método de la API del *slice manager*.

	Ruta	**Descripción**
POST	*/slicemanager/handle-client-create-request*	Manejar solicitud de clientes para crear un slice HPC con seguridad. ● **Request body**: cantidad de masters, cantidad de workers, red access
POST	*/slicemanager/handle-client-delete-request*	Manejar solicitud de clientes para eliminar un slice HPC y su seguridad. ● **Request body**: slice ID, cantidad de masters y workers
POST	*/slicemanager/request-apply-security*	Solicitar al módulo de ciberseguridad aplicar seguridad en el slice ● **Request body**: slice ID, red access, red management, red data
POST	*/slicemanager/request-remove-security*	Solicitar al módulo de ciberseguridad remover seguridad en el slice ● **Request body**: slice ID, red management, red data
POST	*/slicemanager/apply-openstack-security*	Aplicar seguridad al slice en OpenStack (red access) ● **Request body**: slice ID, red access, red management
POST	*/slicemanager/remove-openstack-security*	Remover seguridad al slice en OpenStack (red access) ● **Request body**: slice ID, red access, red management
POST	*/slicemanager/apply-opendaylight-security*	Aplicar seguridad al slice en OpenDaylight (red management y data) ● **Request body**: slice ID, red management, red data
POST	*/slicemanager/remove-opendaylight-security*	Remover seguridad al slice en OpenDaylight (red management y data) ● **Request body**: slice ID, red management, red data
GET	*/slicemanager/get-slice-security-info?sliceid=*	Solicitar al módulo de ciberseguridad que realice un query a base de datos para obtener información de seguridad de un slice HPC. ● **Parámetro GET**: *sliceid* - request con ID del slice
GET	*/slicemanager/get-all-slices-security-info*	Solicitar al módulo de ciberseguridad que realice un query a base de datos para obtener información de seguridad de todos los slices HPC (MÉTODO 1: imprimir list)
GET	*/slicemanager/get-all-slices-security-info2*	Solicitar al módulo de ciberseguridad que realice un query a base de datos para obtener información de seguridad de todos los slices HPC (MÉTODO 2: render HTML template)

TABLA 3-10: Métodos HTTP de la API del módulo *slice manager*

3.5.3.2 API del módulo de ciberseguridad

En la siguiente tabla se resumen los métodos HTTP, las rutas para realizar las solicitudes y la descripción de cada método de la API del módulo de ciberseguridad.

	Ruta	Descripción
POST	*/cybersecurity/apply-security*	Solicitar al slice manager aplicar la seguridad en el slice con OpenStack API y OpenDaylight RESTCONF. ● **Request body**: información completa del slice HPC
POST	*/cybersecurity/remove-security*	Solicitar al slice manager remover la seguridad en el slice con OpenStack API y OpenDaylight RESTCONF. ● **Request body**: slice ID, red management, red data
GET	*/cybersecurity/get-slice-security-info?sliceid=*	Query a base de datos para obtener información de seguridad de un slice HPC. ● **Parámetro GET**: *sliceid* - request con ID del slice
GET	*/cybersecurity/get-all-slices-security-info*	Query a base de datos para obtener información de seguridad de todos los slices HPC (MÉTODO 1: imprimir list)
GET	*/cybersecurity/get-all-slices-security-info2*	Query a base de datos para obtener información de seguridad de todos los slices HPC (MÉTODO 2: render HTML template)

TABLA 3-11: Métodos HTTP de la API del módulo de ciberseguridad

4. Pruebas y resultados

4.1 Entorno de desarrollo

El entorno de desarrollo es el escenario sobre el cual se ha programado la solución de ciberseguridad para slices HPC del HAST y se han realizado pruebas de funcionamiento del código. Este entorno de desarrollo es del tipo virtual y se ha usado un clúster de servidores con OpenStack para desplegar las siguientes máquinas virtuales:

- **OvS**: VM con un *switch* ovs creado y con múltiples puertos que se conectan con las máquinas virtuales *controller* y *computes*. A través de esta VM fluye el tráfico de la red *provider*.
- **Controller**: dentro de esta VM se ejecutan servicios básicos (base de datos SQL, mensajería en colas, NTP, sistemas de memoria caché, entre otras) y proyectos de OpenStack como Keystone, Glance, Nova, Neutron y Horizon.
- **Computes**: VMs que ejecutan los servicios de cómputo de OpenStack (Nova) y algunos servicios de red (Neutron). En estos nodos residen las instancias VM que se crearán con OpenStack.
- **OpenDaylight**: contiene el servicio del controlador SDN OpenDaylight corriendo en un puerto TCP específico para administrar al OvS de la VM del mismo nombre. De esta forma es posible manejar el flujo de datos en la red *provider*.
- **App server**: esta VM tiene desplegados servidores en Python donde corren los proyectos que conforman la aplicación HAST.

Los nodos *computes* y *controller* en realidad forman un despliegue de OpenStack para el entorno de desarrollo en VMs, es decir, se trata de un despliegue de OpenStack sobre OpenStack (OpenStack on OpenStack). Por otro lado, las redes que se han creado para conectar estas máquinas virtuales son dos:

1. **Red de administración** para la comunicación entre servicios de OpenStack, tráfico del plano de control entre la VM con el OvS y el controlador SDN y comunicación entre la solución de ciberseguridad y las otras VMs.
2. **Red provider** para el flujo de datos entre los nodos de cómputo y el nodo controller, es decir, el tráfico de máquinas virtuales.

En la siguiente imagen se muestran las máquinas virtuales provistas por OpenStack y las redes a las que se encuentran conectadas.

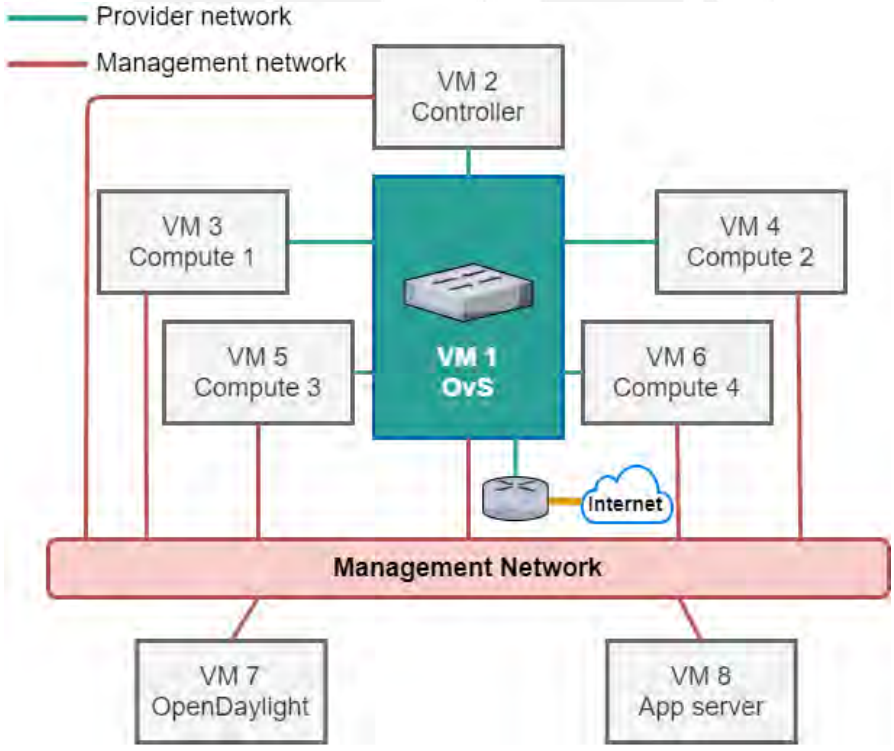

FIGURA 4-1: Entorno de desarrollo

Al ser un entorno de desarrollo del tipo virtual, no será posible usar el *mechanism driver* de SR-IOV. De forma que el *mechanism driver* de SR-IOV se reemplaza por el de Linux bridge. Entonces, las redes de acceso de *slices* HPC son creadas

con el *mechanism driver* de Open vSwitch y, solo en entorno, se usará el *mechanism driver* de Linux bridge para crear las redes de administración y datos de los *slices* HPC.

Debido a que se busca la mayor fidelidad a un entorno físico, donde una NIC con capacidades de SR-IOV puede ser usada tanto por un *mechanism driver* común (como Open vSwitch o Linux bridge) a través de su PF, como por el *mechanism driver* de SR-IOV usando las VFs, se decide hacer un despliegue acorde en cada nodo de cómputo. Además, se toma en consideración el *switch* interno con el que cuentan las NICs al habilitar SR-IOV y que permite la comunicación entre VMs que tienen asignada VFs distintas en la misma NIC.

Con el fin de simular el funcionamiento del *mechanism driver* de SR-IOV en los nodos de cómputo de OpenStack se ha creado un OvS extra (br-sriov) al cual se conecta la interfaz del nodo de cómputo para la red provider (eth3). Luego se crea una interfaz interna (phy2) en este OvS que es vista por el sistema y será configurada para el Linux bridge *mechanism driver*. Cuando se creen nuevas redes con el *mechanism driver* de Linux bridge, se usará la interfaz interna para crear nuevas subinterfaces con VLANs distintas. Por otro lado, con el objetivo de que el OvS del *mechanism driver* de Open vSwitch para redes VLAN (br-provider) tenga salida a la red provider, se crea manualmente un *patch cable* entre los OvS br-sriov y br-provider. En la siguiente imagen se representa un ejemplo de los tipos de componentes que se podrían encontrar en un nodo de cómputo luego de crear *slices* HPC.

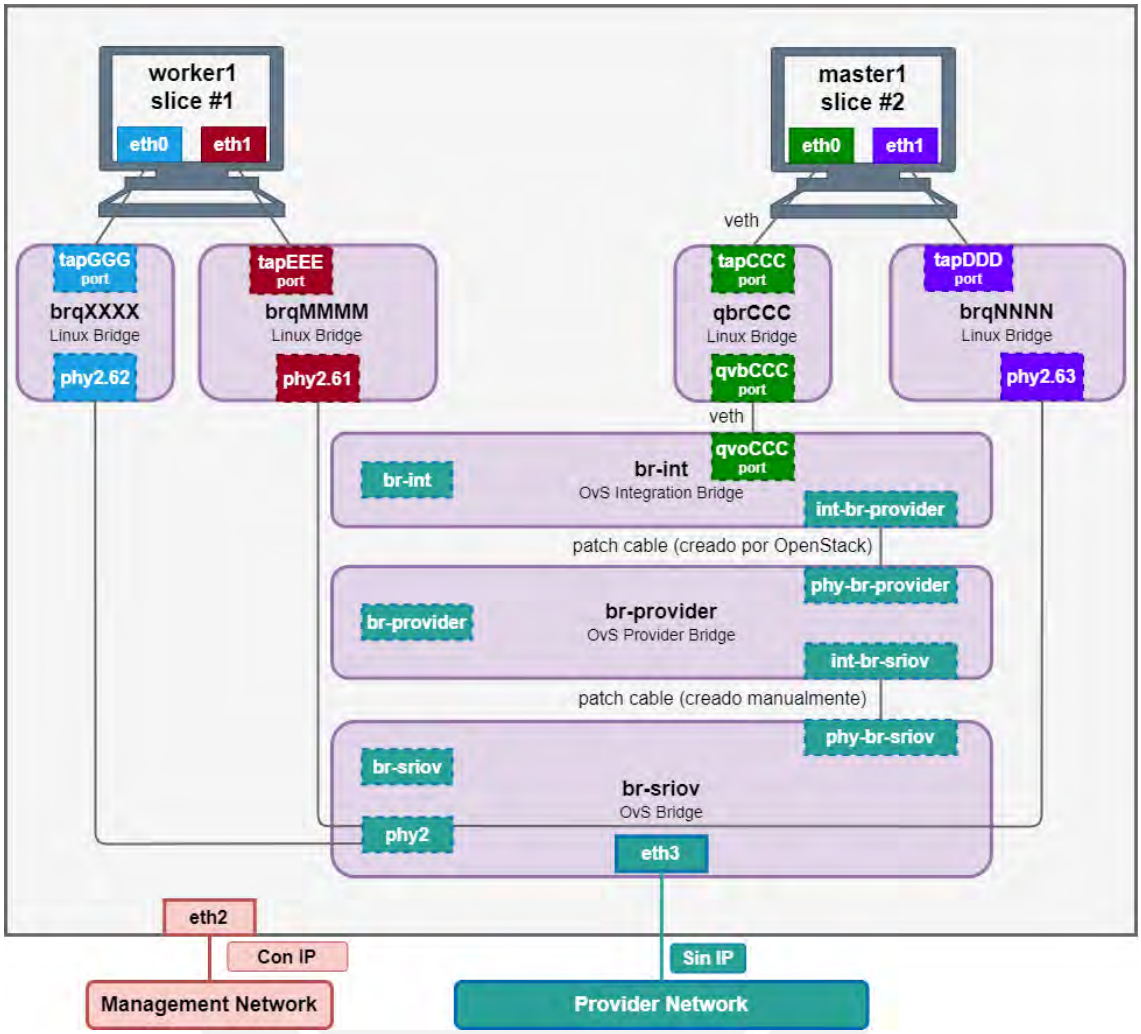

FIGURA 4-2: Ejemplo de nodo de cómputo del entorno de desarrollo

4.2 Entorno de pruebas

El entorno de pruebas es el escenario donde se demostrará el funcionamiento de las reglas de seguridad. Para este fin se emplearán tres servidores físicos: un nodo *controller* de OpenStack y dos nodos de cómputo de OpenStack. El nodo *controller* además tendrá los servicios del controlador OpenDaylight y la aplicación en lenguaje Python para implementar la seguridad en slices HPC. En la siguiente imagen se muestran los componentes que forman parte del entorno de pruebas.

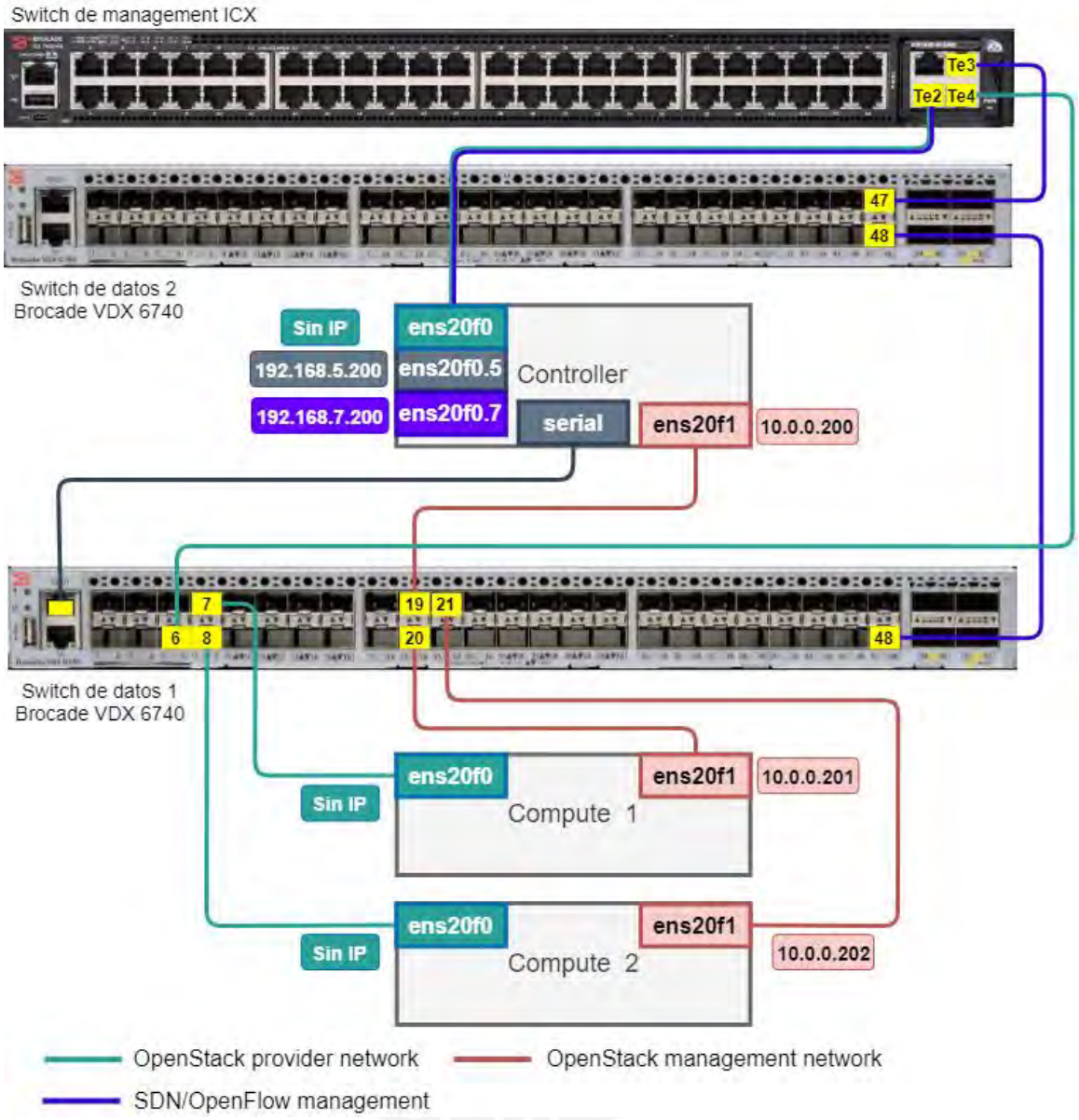

FIGURA 4-3: Entorno de pruebas

Todos los servidores cuentan con dos interfaces físicas Intel 82599 con capacidad SR-IOV. El nodo controller usa una interfaz (ens20f1) para la red de *management* de OpenStack, es decir, por la interfaz atravesará tráfico de administración. La otra interfaz (ens20f0) del nodo *controller* es usada para múltiples propósitos: la interfaz misma (ens20f0) es usada para la red de datos de OpenStack (sin IP), una subinterfaz (ens20f0.5) para el acceso a internet del nodo *controller* (VLAN tagged) y otra subinterfaz (ens20f0.7) para la administración SDN/OpenFlow (VLAN tagged). Los nodos de cómputo usan una

interfaz para la red de *management* de OpenStack (ens20f1) y la otra interfaz para la red de datos de OpenStack (ens20f0).

Para el tráfico de datos entre servidores, el entorno de pruebas cuenta con tres *switches*. El motivo de uso de múltiples *switches* se debe a la disponibilidad de puertos de alta velocidad (10 Gbps) de los que se dispone en cada *switch*. El *switch* de datos principal (*switch* de datos 1), se encargará del flujo de tráfico, tanto para la red de administración, como para la red *provider* (datos) de OpenStack. Tres puertos del *switch* (puertos 19, 20, 21) configurados en modo L2 (untagged) son usados para la comunicación en la red de OpenStack *management*, a través de una VLAN específica.

El *switch* de datos principal es un *switch* Brocade VDX 6740 que tiene, como ya se mencionó, soporte limitado de OpenFlow versión 1.3 y con una lista determinada de *matches* y *actions* para la inserción de *flows*. Además, en la guía de configuración del sistema operativo del *switch* [57] se señala que el *switch* tiene soporte de una sola *flow table* para instrucciones OpenFlow. Los 3 puertos del *switch* (puertos 6, 7, 8) para la red *provider*, destinados para la comunicación entre máquinas virtuales son configurados como puertos OpenFlow. Debido a que este *switch* trabaja como un *switch* OpenFlow-only, no será posible ingresar un comportamiento *legacy* (NORMAL) para el tráfico que no se quiera procesar bajo el OpenFlow *pipeline*. Para simular el comportamiento de un *switch legacy*, se configuró un *flow* para que todo el tráfico perteneciente a una VLAN en la red de acceso sea reenviado por todos los puertos menos el del origen del tráfico.

Se usa un puerto (*untagged*) distinto del *switch* de datos principal (puerto 48) para la administración SDN/OpenFlow, es decir, un puerto que sirve para la comunicación con el controlador OpenDaylight, en una VLAN específica. Mediante este puerto se podrá insertar o eliminar *flows* y *group tables* al *switch* con el fin de manejar el tráfico de datos.

El nodo *controller*, desde el punto de vista de OpenStack, se conforma de una estructura de OvS *bridges*, *namespaces* e interfaces para brindar conectividad a las máquinas virtuales que residen en los nodos de cómputo. El *bridge* de integración (br-int) se encarga de centralizar los *namespaces* de 2 tipos. El primer tipo de *namespace* es el de servidor DHCP, creado para cada red de acceso, con el fin de configurar la dirección IP a las VMs masters. El segundo tipo de *namespace* es el *router*, creado para conectar la red de acceso de cada *slice* HPC con la red *external provider* y que, de esta forma, las VMs *masters* tengan salida a Internet y se puedan comunicar con otros equipos permitidos en la red *external provider* cuando se asigne una IP flotante a la VM. El *bridge* br-int se conecta con el OvS *bridge provider* (br-provider) que tiene una interfaz física para salida de datos (ens20f0). En la siguiente imagen se presenta un ejemplo de la topología interior que encontraríamos en el nodo *controller*.

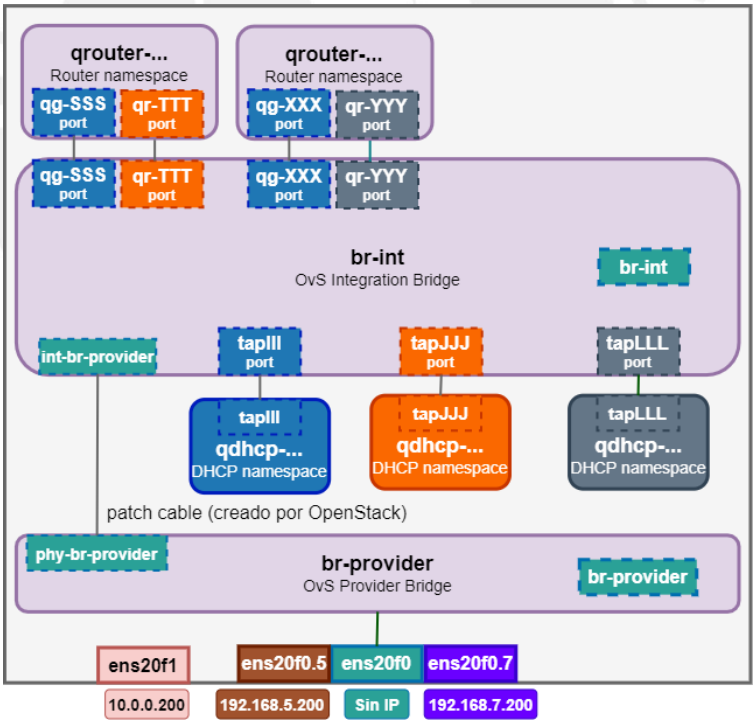

FIGURA 4-4: Ejemplo de nodo *controller* del entorno de pruebas

Un nodo *compute* presentará otra organización de componentes internos, respecto al nodo *controller*. La interfaz física dedicada a datos estará dividida en múltiples funciones virtuales (VFs) y una sola función física (PF). La PF (ens20f0)

se conectará como interfaz del OvS asignado para el *mechanism driver* de Open vSwitch (br-provider), *bridge* que estará conectado directamente al *ovs* creado por OpenStack (br-int). En el ovs br-int se tendrá conectado un puerto *tap* enlazado a la interfaz de la red de acceso de cada VM *master*. Por otro lado, las VFs creadas desde la NIC con capacidades de SR-IOV se encuentran asignadas directamente a las interfaces de las máquinas virtuales para las redes de administración y datos. En br-int se insertan *flows* para la implementación de *security groups* (usando Native Open vSwitch firewall driver), que configurará la seguridad para las redes de acceso.

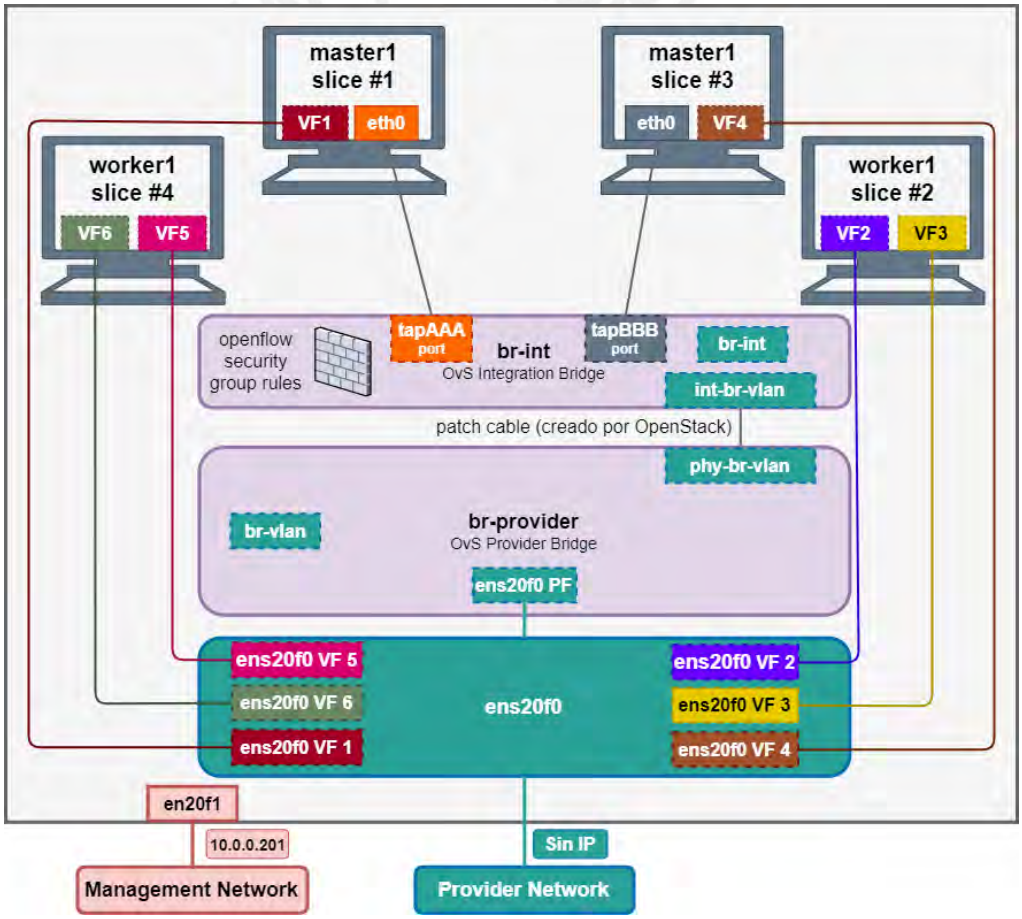

FIGURA 4-5: Ejemplo de nodo de cómputo 1 del entorno de pruebas

4.3 Pruebas y resultados

4.3.1 Pruebas de funcionamiento

Sobre la infraestructura desplegada, usando el programa desarrollado en Python se ha creado un *slice* HPC conformado por dos instancias *masters* y dos instancias *workers* que residen en los dos nodos de cómputo del entorno de prueba. También se crea un *router namespace* que conecta la red de acceso con la red *external provider* para la comunicación de las instancias *masters* con equipos externos. La aplicación de Python se ha encargado de crear todas las reglas de seguridad (tanto de *security groups* como de OpenFlow) para resguardar a las VMs según los requerimientos de seguridad y la implementación de las reglas para este entorno. En la siguiente imagen se muestra el diagrama de conexiones que se encontraría en el *dashboard* de OpenStack.

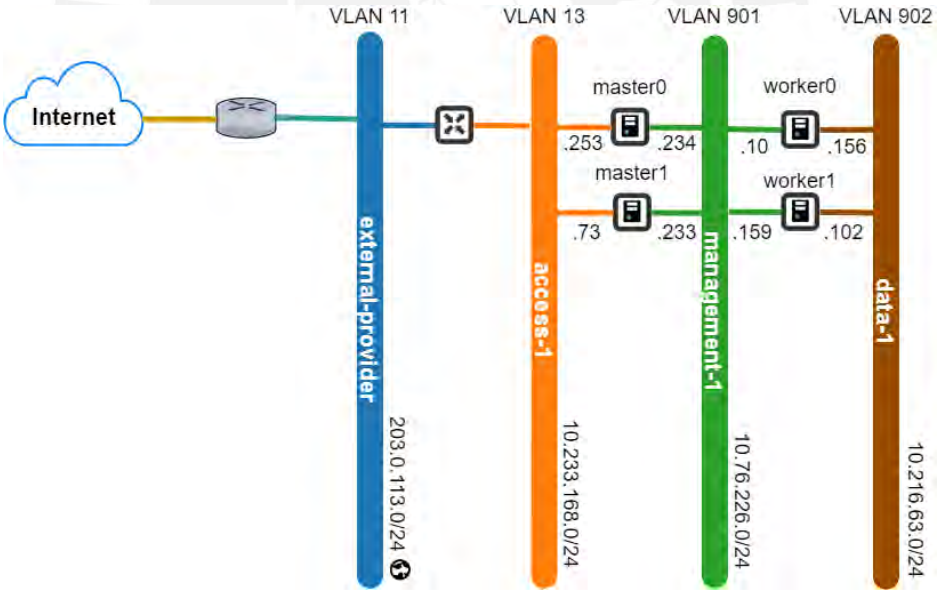

FIGURA 4-6: *Slice* HPC usado para las pruebas

En la siguiente tabla se resumen las propiedades de las cuatro instancias VM creadas para el *slice* HPC. Para cada instancia *master* y *worker* se especifica el nodo cómputo donde la instancia ha sido desplegada y, por tanto, el puerto Openflow asociado a donde se conecta el servidor físicamente en el *switch*.

Además, se detallan los parámetros de MAC e IP de cada puerto con el que cuenta la VM.

VM	Propiedad	Valor
master0	in-port (compute)	port 7 (compute1)
	access iface (ens3)	MAC: fa:16:3e:6e:d9:67 - IP: 10.233.168.253
	management iface (ens5)	MAC: fa:16:3e:14:ac:45 - IP: 10.76.226.234
master1	in-port (compute)	port 8 (compute2)
	access iface (ens3)	MAC: fa:16:3e:45:67:a5 - IP: 10.233.168.73
	management iface (ens5)	MAC: fa:16:3e:cb:67:11 - IP: 10.76.226.233
worker0	in-port (compute)	port 7 (compute1)
	management iface (ens4)	MAC: fa:16:3e:6d:ed:56 - IP: 10.76.226.10
	data iface (ens5)	MAC: fa:16:3e:4b:ab:05 - IP: 10.216.63.156
worker1	in-port (compute)	port 8 (compute2)
	management iface (ens4)	MAC: fa:16:3e:96:81:f6 - IP: 10.76.226.159
	data iface (ens5)	MAC: fa:16:3e:fc:e6:dd - IP: 10.216.63.102

TABLA 4-1: Propiedades de red de las VMs del *slice* HPC

A continuación, se comprobará la correcta implementación de seguridad para cada tipo de red en el *slice* HPC creado. Con este objetivo, se usarán las herramientas *ping* y *arping* de Linux para la generación de tráfico entre VMs. Además, para el caso de las redes de administración y datos se verán los *flows* insertados en el *switch* a los que se hace *match*, así como su incremento en la cantidad de paquetes, según las estadísticas del *flow*.

4.3.1.1 Red de acceso

Dentro del *slice* HPC, se espera que la comunicación entre VMs *masters* verificadas, usando la red de acceso, esté permitida. Como se muestra en la siguiente imagen, los ARP *requests* mandados a través de la herramienta *arping* desde la interfaz de acceso de la VM master0 son respondidos por la VM master1.

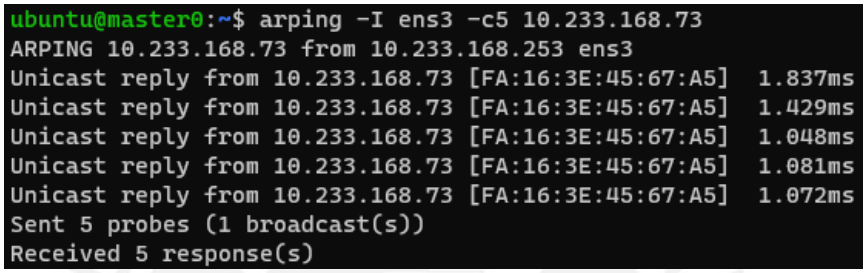

FIGURA 4-7: Comunicación master0 con master1 (red de acceso)

Además, el tráfico saliente de las VMs masters hacia cualquier red externa y los paquetes de respuesta también deben ser permitidos en la red de acceso. Como se muestra en las siguientes imágenes, los mensajes ARP *request* hacia una IP en la red *external provider* generados con *arping* y con destino *broadcast* no obtienen una respuesta de regreso. Por otro lado, los paquetes ICMP transmitidos con la herramienta de *ping* hacia la misma IP de la red *external provider* sí tienen un mensaje de retorno debido a la presencia de un *router namespace* que conecta las redes de acceso y administración.

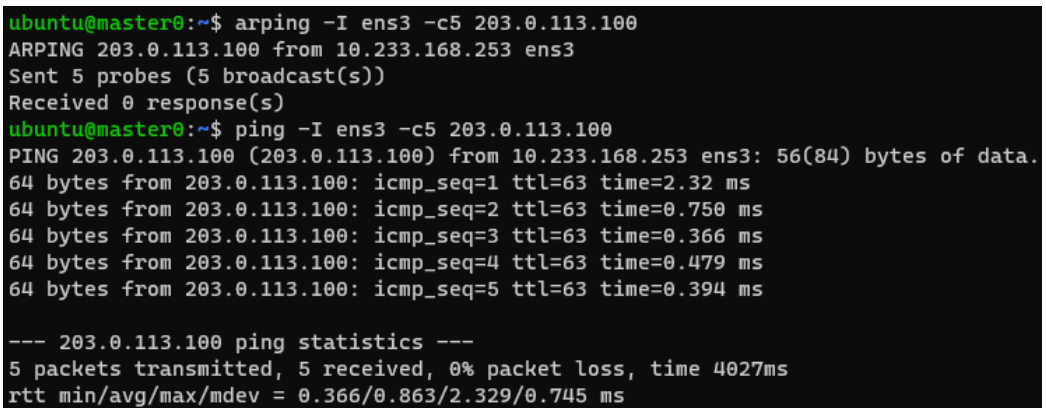

FIGURA 4-8: Comunicación master0 con red externa usando *arping* y *ping*

4.3.1.2 Red de administración

- Comunicación entre *master* y *worker* (iniciada desde un *master*)

En la red de administración, la comunicación entre *masters* y *workers* está habilitada. En la siguiente imagen se comprueba el intercambio de mensajes exitoso entre una instancia *master* (master0) y una instancia *worker* (worker1) en distintos nodos de cómputo a través de la red de administración.

```
ubuntu@master0:~$ arping -I ens5 -c5 10.76.226.159
ARPING 10.76.226.159 from 10.76.226.234 ens5
Unicast reply from 10.76.226.159 [FA:16:3E:96:81:F6]  0.848ms
Unicast reply from 10.76.226.159 [FA:16:3E:96:81:F6]  0.732ms
Unicast reply from 10.76.226.159 [FA:16:3E:96:81:F6]  0.720ms
Unicast reply from 10.76.226.159 [FA:16:3E:96:81:F6]  0.779ms
Unicast reply from 10.76.226.159 [FA:16:3E:96:81:F6]  0.735ms
Sent 5 probes (1 broadcast(s))
Received 5 response(s)
```

FIGURA 4-9: Comunicación master0 con worker1 (red de administración)

En la imagen previa, la comunicación entre ambas VMs comienza con un único mensaje ARP *request* en *broadcast* enviado por master0. El paquete hace *match* con un *flow* en el *switch* cuyo *action* es enviar el paquete al *group table* con *out ports* OpenFlow que tengan conectados nodos de cómputo con VMs *masters* y *workers* del *slice* HPC. En la siguiente imagen se muestra el *flow entry* y el incremento de un solo paquete en las estadísticas.

```
VDX6740-01# show openflow flow flowid 16
Flow ID: 16 Priority: 4001 Status: ACTIVE
        Rule:
           In Port:                    Te 1/0/7
           In Vlan:                    Tagged[901]
           Source Mac:                 fa16.3e14.ac45
        Instructions: Apply-Actions
           Action: FORWARD
                       Out Group: 292577406
                       Out Port: None


Statistics:
        Total Pkts:  19
        Total Bytes: NA
```

```
VDX6740-01# show openflow flow flowid 16
Flow ID: 16 Priority: 4001 Status: ACTIVE
        Rule:
           In Port:                    Te 1/0/7
           In Vlan:                    Tagged[901]
           Source Mac:                 fa16.3e14.ac45
        Instructions: Apply-Actions
           Action: FORWARD
                       Out Group: 292577406
                       Out Port: None


Statistics:
        Total Pkts:  20
        Total Bytes: NA
```

FIGURA 4-10: *Flow* con *match* para tráfico master0 → [?]

Al llegar el mensaje ARP *request* al destino indicado (worker1), esta VM responde con un mensaje ARP *reply* en *unicast* hacia master0. Al pasar por el *switch*, el paquete hará match con un *flow* que se encarga de hacer *forward* hacia el puerto OpenFlow asociado con la VM master destino (master0). En la siguiente imagen se muestra el *flow entry* y el incremento de 5 paquetes en las estadísticas, debido a los 5 ARP *replies* del comando *arping*.

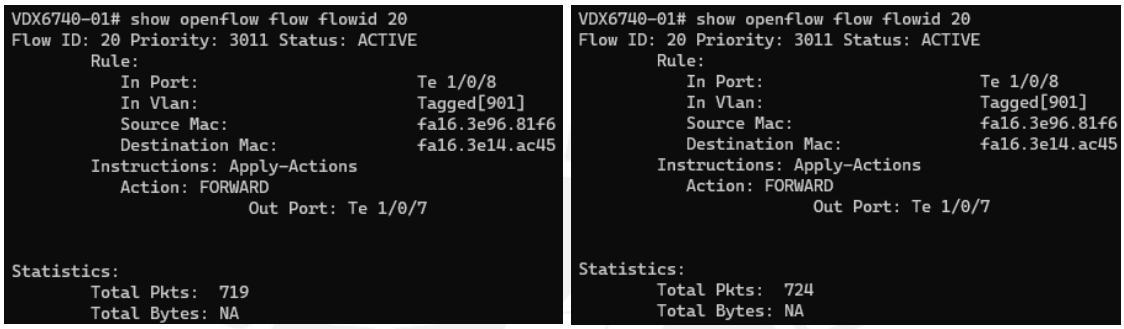

FIGURA 4-11: *Flow* con *match* para tráfico worker1 → master0

Cuando le llega el mensaje ARP *reply* a la VM master0, esta VM conocerá la MAC destino de worker1, respondiendo con mensajes ARP *request* en *unicast*. Estos paquetes hacen *match* en el *switch* con un *flow* que se encarga de la comunicación *unicast* entre master0 y worker1, enviando el paquete por el puerto OpenFlow al que se conecta el nodo de cómputo de la VM worker1 únicamente. En la siguiente imagen se muestra el *flow entry* y el incremento de 4 paquetes en las estadísticas, debido a los 4 ARP *requests* enviados en *unicast* con *arping*.

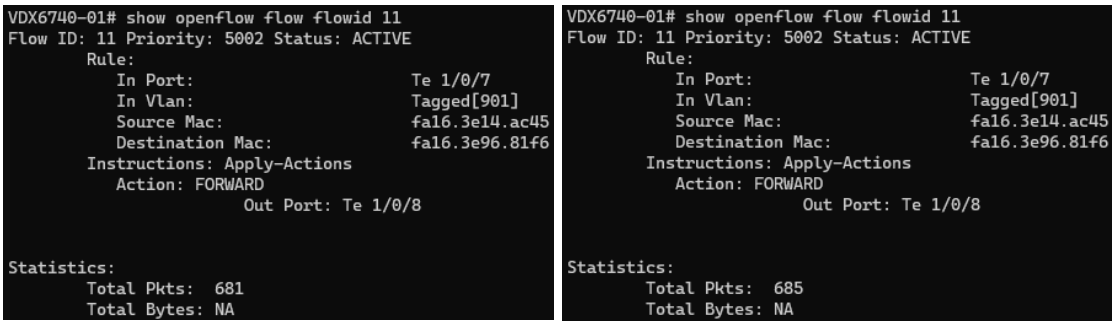

FIGURA 4-12: *Flow* con *match* para tráfico master0 → worker1

- Comunicación entre *master* y *worker* (iniciada desde un *worker*)

El intercambio de mensajes entre una VM *worker* y una VM *master*, iniciada desde la VM *worker* a través de la herramienta *arping*, presentará diferencias en el tipo de *flows* implementados respecto a la prueba anterior. En este caso, el primer paquete hará *match* con un tipo de *flow* distinto que se encarga de filtrar el tráfico *broadcast* que procede del *worker*. En la siguiente imagen se muestra el correcto intercambio de mensajes entre la VM worker0 y la VM master1, ejecutando *arping* desde worker0.

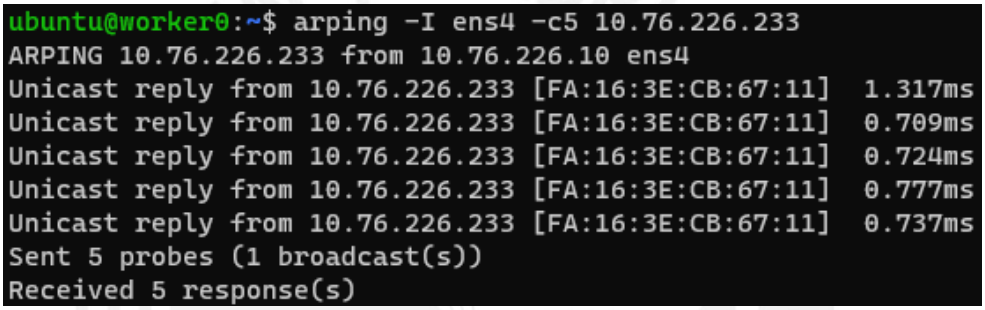

FIGURA 4-13: Comunicación worker0 con master1 (red de administración)

El primer paquete enviado desde worker0 es un ARP *request* enviado a la dirección MAC *broadcast*. Al atravesar el *switch*, el paquete hará *match* con un *flow* que se encarga de manejar el tráfico cuyo origen es worker0 y el destino es cualquier cosa menos un *master* u otro *worker*. La acción del *flow* será enviarlo al *group table* conformado por los puertos OpenFlows relacionados a todas las VMs *masters* del *slice*. En la siguiente imagen se muestra el *flow entry* y el incremento de 1 paquete en las estadísticas, debido al ARP *request* enviado en *broadcast*.

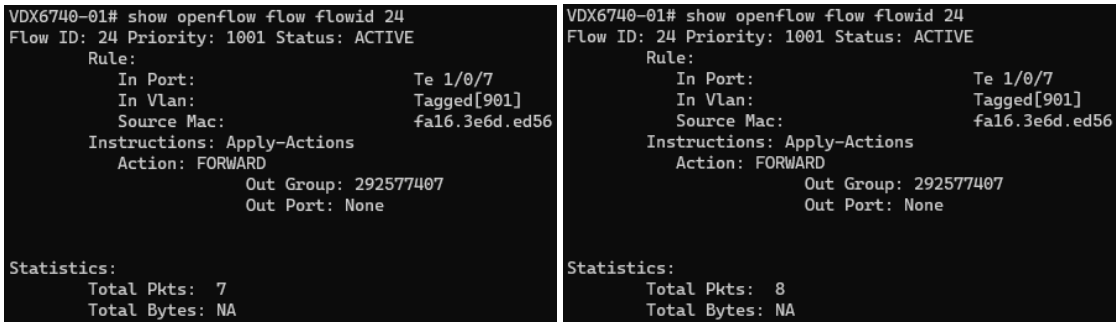

FIGURA 4-14: *Flow* con *match* para tráfico worker0 → [?]

Cuando el ARP *request* enviado en *broadcast* llegue a la VM master1, esta responderá con un ARP *reply* que ingresará al *switch* y hará match con un tipo de *flow* visto en la prueba anterior, donde la comunicación era iniciada desde un *master*. La acción del *flow* será enviar el paquete por el puerto OpenFlow asociado al *compute* donde reside la VM worker0. En la siguiente imagen se muestra el *flow entry* y el incremento de 5 paquetes en las estadísticas, debido a los 5 ARP *replies* generados por *arping*.

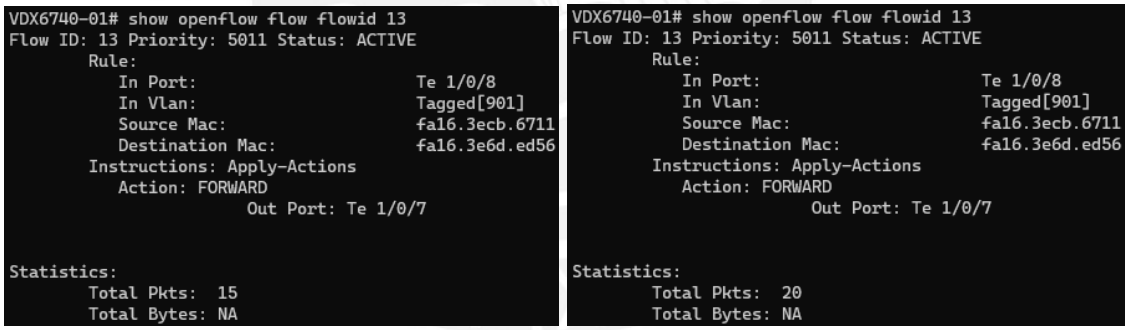

FIGURA 4-15: *Flow* con *match* para tráfico master1 → worker0

El paquete ARP *reply* enviado en *unicast* desde master1 a worker0 llega a esta VM que responderá con otro ARP *request* pero ahora en *unicast* hacia master1. El tipo de *flow*, que hace *match* con estos paquetes *unicast* cuyo origen es un *worker* y el destino es un *master*, también fue empleado en la prueba anterior. La acción del *flow* será enviar el paquete hacia el puerto OpenFlow asociado al nodo de cómputo donde reside la VM master1. En la siguiente imagen se muestra el *flow entry* y el incremento de 4 paquetes en las estadísticas por los 4 ARP *request unicast* enviados desde worker0 con *arping*.

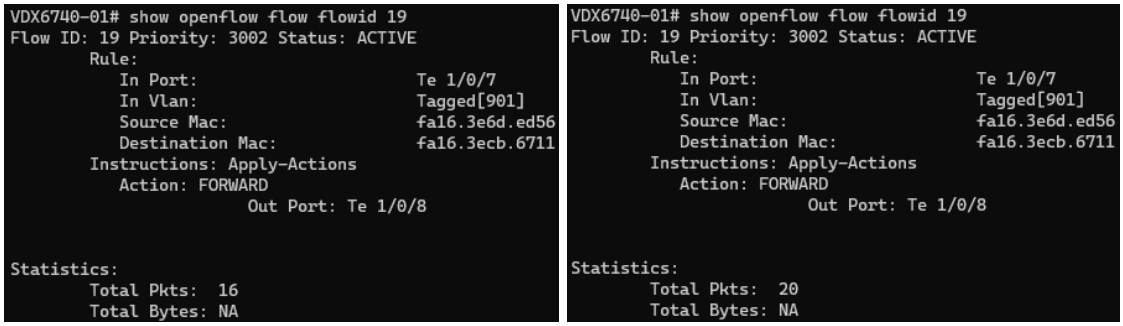

FIGURA 4-16: *Flow* con *match* para tráfico worker0 → master1

- Comunicación entre *masters*

La comunicación entre VMs *masters* es permitida en la red de administración. En la siguiente imagen se muestra la respuesta de mensajes ARP de la VM master1 a solicitudes ARP con origen en master0 usando *arping*.

```
ubuntu@master0:~$ arping -I ens5 -c5 10.76.226.233
ARPING 10.76.226.233 from 10.76.226.234 ens5
Unicast reply from 10.76.226.233 [FA:16:3E:CB:67:11]  0.756ms
Unicast reply from 10.76.226.233 [FA:16:3E:CB:67:11]  0.714ms
Unicast reply from 10.76.226.233 [FA:16:3E:CB:67:11]  0.684ms
Unicast reply from 10.76.226.233 [FA:16:3E:CB:67:11]  0.690ms
Unicast reply from 10.76.226.233 [FA:16:3E:CB:67:11]  0.691ms
Sent 5 probes (1 broadcast(s))
Received 5 response(s)
```

FIGURA 4-17: Comunicación master0 con master1 (red de administración)

El primer paquete es un ARP *request* enviado en *broadcast* desde la VM master0 que hace en *match* con un *flow* cuyo *action* es enviar el paquete al *group table* conformado por los puertos OpenFlow asociados a todas las VMs *masters* y *workers* del *slice*. En la siguiente imagen se muestra el *flow entry* y su incremento en un solo paquete en las estadísticas del *flow*, debido al primer ARP *request* enviado.

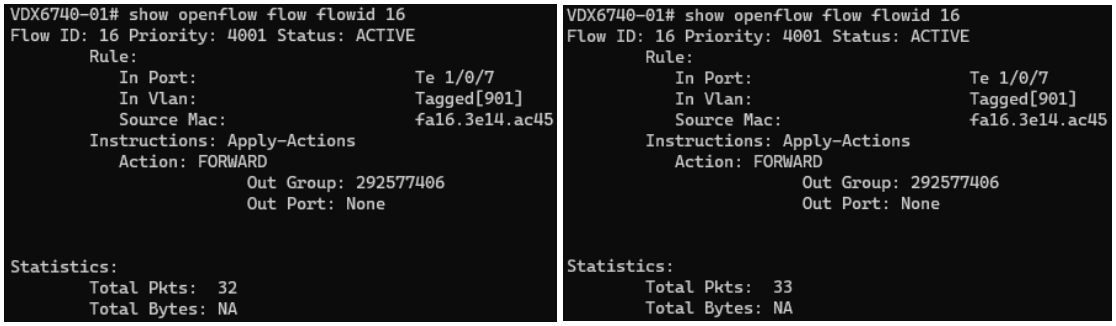

FIGURA 4-18: *Flow* con *match* para tráfico master0 → [?]

Cuando la VM master0 recibe el ARP *request* enviado en *broadcast*, la VM master1 genera un ARP *reply* en *unicast* hacia master0. Cuando el paquete ingrese al *switch*, se hace match con un *flow* usado para enviar el paquete al puerto OpenFlow de la VM master0. En la siguiente imagen se muestra el *flow entry* y su incremento del total de paquetes en 5, debido a los 5 ARP *replies* que envía master1 en respuesta a los *request* de master0.

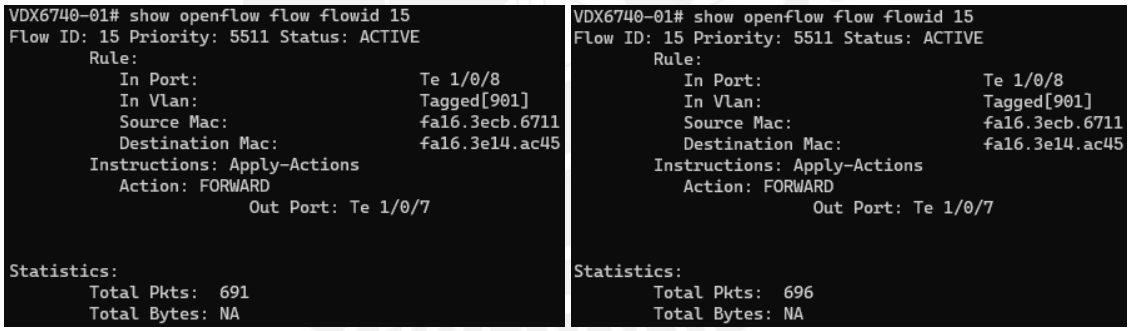

FIGURA 4-19: *Flow* con *match* para tráfico master1 → master0

Los ARP *requests* posteriores que se envían desde master0 ahora serán en *unicast* porque ahora esta VM conoce la MAC destino de master1. El resultado será que cuando el paquete ingrese al *switch* hará *match* con un *flow* cuyo *action* es enviar el paquete al puerto OpenFlow asociado a master1. En la siguiente imagen se muestra el *flow entry* y el aumento de 4 paquetes del total, debido a los 4 ARP *requests* enviados en *unicast* desde master0.

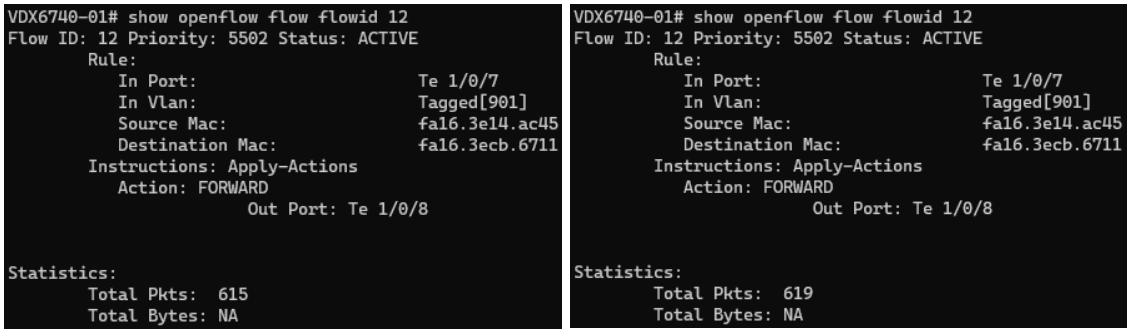

FIGURA 4-20: *Flow* con *match* para tráfico master0 → master1

- Comunicación entre *workers*

Por requerimientos de seguridad de la red de administración, la comunicación entre VMs *workers* debe estar bloqueada. Como ejemplo, en la siguiente imagen se muestran ARP *requests* enviados desde la interfaz de administración de la VM worker0 hacia otra VM worker1. El resultado es que no existe una respuesta de vuelta a las solicitudes ARP enviadas.

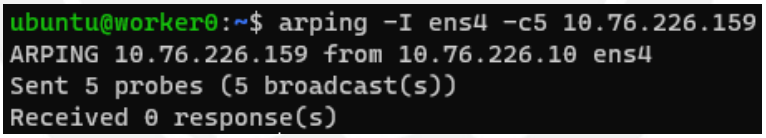

FIGURA 4-21: Comunicación worker0 con worker1 (red de administración)

Los paquetes enviados desde un *worker* en *multicast* harán *match* con un *flow* que enviará el paquete al *group table* conformado por los puertos OpenFlow con nodos de cómputo que tengan VMs *masters* del *slice* HPC. Este *flow* no excluye que los paquetes *multicast* originados en un *worker* le lleguen a otros *workers* colocalizados en el mismo nodo *compute* donde hay un *master* del mismo *slice* HPC.

El primer paquete ARP *request* del ejemplo previo es enviado en *broadcast* por worker0, haciendo *match* con el *flow* que reenvía el tráfico del *worker* hacia el *group table* de puertos OpenFlow relacionados a las VMs *masters*. En la

siguiente imagen se muestra el *flow entry* y el incremento de 5 paquetes en las estadísticas, debido a los 5 ARP *requests* enviados en *broadcast* desde worker0 con *arping*.

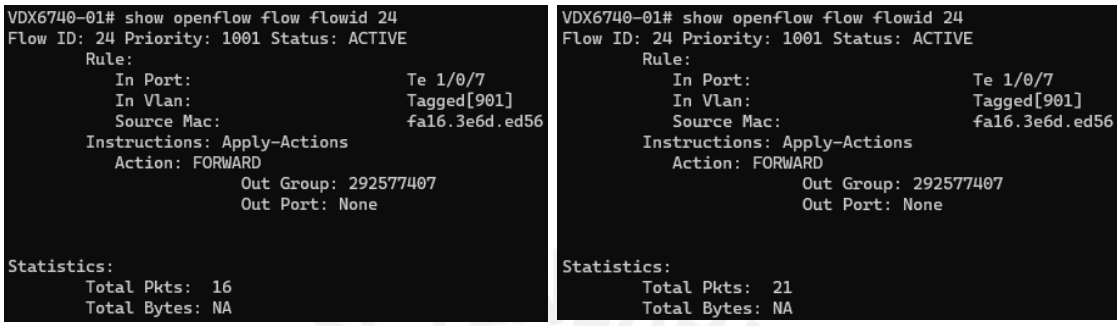

FIGURA 4-22: *Flow* con *match* para tráfico worker0 → [?]

Luego, el ARP *reply* de la VM worker1 será bloqueado por un *flow* que tiene como acción descartar el tráfico cuya MAC destino sea la de un *worker* (en este caso worker0). En la siguiente imagen se muestra el *flow entry* y el aumento del total de paquetes en 5, debido a los ARP *replies* enviados de worker1 a worker0 con arping.

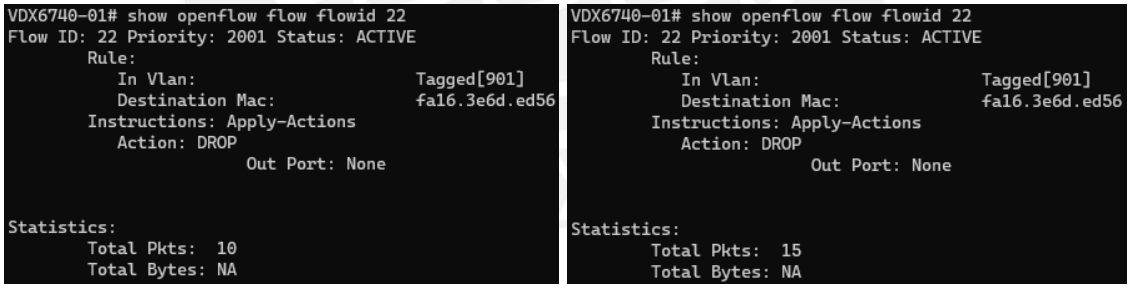

FIGURA 4-23: *Flow* con *match* para tráfico [?] → worker0

4.3.1.3 Red de datos

En la red de datos, solo existe comunicación entre un mismo tipo de instancia VM. En la siguiente imagen se comprueba que la comunicación entre worker0 y worker1 es permitida.

```
ubuntu@worker0:~$ arping -I ens5 -c5 10.216.63.102
ARPING 10.216.63.102 from 10.216.63.156 ens5
Unicast reply from 10.216.63.102 [FA:16:3E:FC:E6:DD]  0.773ms
Unicast reply from 10.216.63.102 [FA:16:3E:FC:E6:DD]  0.697ms
Unicast reply from 10.216.63.102 [FA:16:3E:FC:E6:DD]  0.704ms
Unicast reply from 10.216.63.102 [FA:16:3E:FC:E6:DD]  0.718ms
Unicast reply from 10.216.63.102 [FA:16:3E:FC:E6:DD]  0.701ms
Sent 5 probes (1 broadcast(s))
Received 5 response(s)
```

FIGURA 4-24: Comunicación worker0 con worker1 (red de datos)

En el ejemplo de la imagen anterior, la VM worker0 envía un mensaje ARP *request* en *broadcast*. Al ingresar al *switch*, el paquete hará match con un *flow* encargado de enviarlo por todos los puertos OpenFlow conectados a nodos de cómputo con VMs *workers* del mismo *slice* HPC. En la siguiente imagen se muestra el *flow entry* y el incremento en 1 paquete del total de paquetes en las estadísticas, debido al ARP *request* enviado en *broadcast* con *arping*.

```
VDX6740-01# show openflow flow flowid 28
Flow ID: 28 Priority: 8001 Status: ACTIVE
        Rule:
           In Port:                    Te 1/0/7
           In Vlan:                    Tagged[902]
           Source Mac:                 fa16.3e4b.ab05
        Instructions: Apply-Actions
           Action: FORWARD
                       Out Group: 292577408
                       Out Port: None


Statistics:
        Total Pkts:  12
        Total Bytes: NA
```

```
VDX6740-01# show openflow flow flowid 28
Flow ID: 28 Priority: 8001 Status: ACTIVE
        Rule:
           In Port:                    Te 1/0/7
           In Vlan:                    Tagged[902]
           Source Mac:                 fa16.3e4b.ab05
        Instructions: Apply-Actions
           Action: FORWARD
                       Out Group: 292577408
                       Out Port: None


Statistics:
        Total Pkts:  13
        Total Bytes: NA
```

FIGURA 4-25: Flow con match para tráfico worker0 → [?]

Posteriormente, worker1 responderá con un ARP *reply* en *unicast* hacia worker0. Al hacer *match* con el *flow* del *switch*, la acción será enviar el paquete por el puerto OpenFlow conectado al nodo de cómputo donde reside worker0. En la siguiente imagen se muestra el *flow entry* y el aumento en 5 paquetes en las estadísticas del *flow*, debido a los 5 ARP *replies* enviados desde worker1.

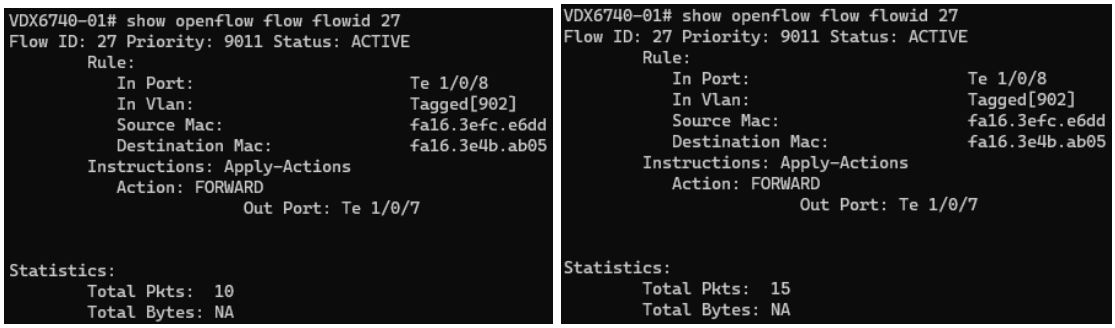

FIGURA 4-26: *Flow* con *match* para tráfico worker1 → worker0

Finalmente, los siguientes ARP *request* enviados desde worker0 serán en *unicast* y harán *match* con un *flow* distinto que enviará los paquetes hacia el puerto OpenFlow relacionado al nodo de cómputo donde está worker1. En la siguiente imagen se muestra el *flow entry* y el incremento de 4 paquetes del total en las estadísticas, debido a los 4 ARP *requests* enviados en *unicast* desde worker0.

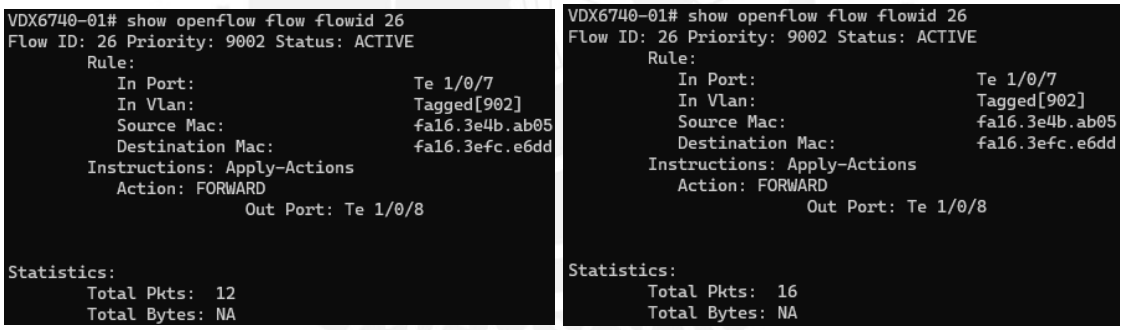

FIGURA 4-27: *Flow* con *match* para tráfico worker0 → worker1

4.3.2 Pruebas de protección contra atacantes externos

Partiendo de un *slice* HPC con dos VMs *masters* y dos VMs *workers*, se crean dos nuevas VMs, cada una con una interfaz conectada a la red *external provider*: inst1 e inst2. La VM inst1 (203.0.113.160/24) representa a una VM atacante, a la cual no se le permitirá acceder a la red de acceso del *slice*; mientras que la inst2 (203.0.113.170/24) tendrá permisos para acceder a puertos TCP específicos de las VMs *master*. Además, se asigna una IP flotante (203.0.113.150/24) perteneciente a la red *external provider* a la VM master0,

para que sea posible la comunicación con la red externa. El escenario se representa en la siguiente imagen.

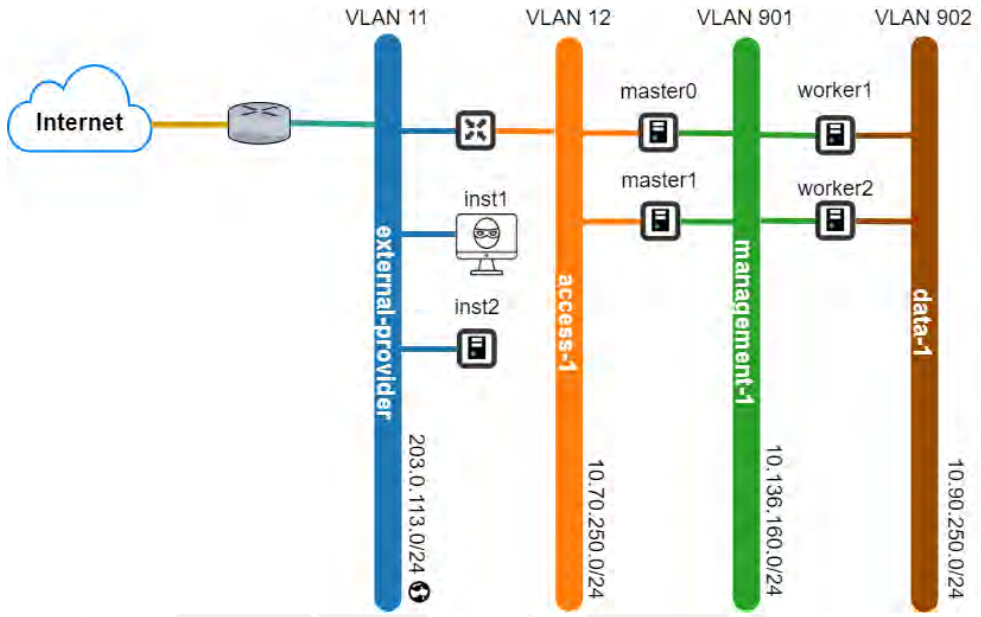

FIGURA 4-28: Escenario de pruebas de ataques externos

Al crear el *slice* HPC, el usuario puede ingresar como input los permisos que desea dar para la red de acceso, es decir, a qué puertos de las VMs master (en la red de acceso) pueden acceder ciertas IPs o rango de IPs. En este caso, se ha habilitado el acceso a los puertos 22 y 8080 desde la IP 203.0.113.170 (inst2), tal como se demuestra en el siguiente formato JSON enviado para la creación del slice HPC.

```
{
    "cant_masters": "2",
    "cant_workers": "2",
    "access_network": {
        "allowed_ranges": [
            {
                "allowed_ip_range": "203.0.113.170/32",
                "all_tcp_enabled": "false",
                "enabled_ports": [
                    22,
                    8080
                ]
            }
        ]
    }
}
```

La aplicación encargada de la implementación de ciberseguridad obtendrá el *input* del usuario y transformará los requisitos en reglas de *security group* para resguardar la red de acceso del *slice* HPC. Para el ejemplo en cuestión, las reglas que encontraremos en el *security group* asignado exclusivamente para el *slice* serán las que se muestran en la siguiente imagen.

Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group
Egress	IPv4	Any	Any	0.0.0.0/0	-
Egress	IPv6	Any	Any	::/0	-
Ingress	IPv4	Any	Any	-	40c38058-4237-4735-8a89-5b865e49f28a_cluster_access_sg
Ingress	IPv4	TCP	22 (SSH)	203.0.113.170/32	-
Ingress	IPv4	TCP	8080	203.0.113.170/32	-

Displaying 5 items

FIGURA 4-29: *Security group* para la red de acceso del *slice* HPC creado

Si intentamos acceder por SSH desde una VM no permitida o atacante (inst1) a la IP pública de la VM master0, comprobaremos que no es posible; mientras que el acceso desde la instancia permitida (inst2) es permitida. El motivo es que, a través de *security groups*, se están permitiendo las conexiones al puerto TCP 22

de las VMs conectadas a la red de acceso desde una instancia con IP 203.0.113.170. En la siguiente imagen se compara el intento de conexión entre la instancia atacante (inst1) y la instancia permitida (inst2).

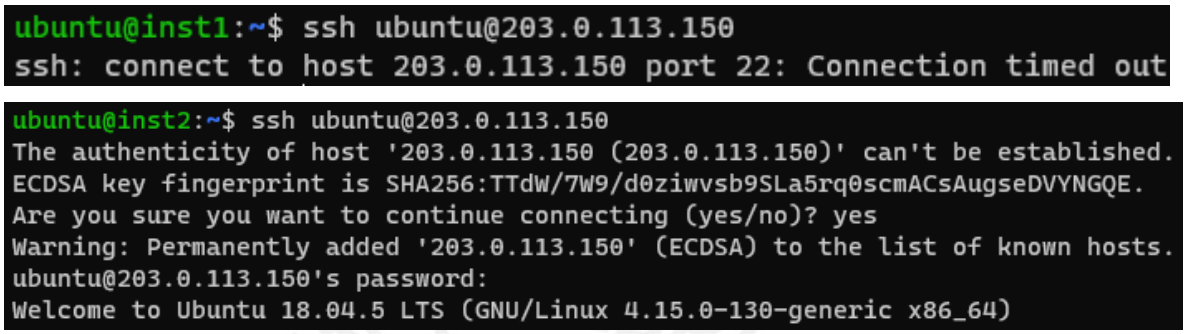

FIGURA 4-30: Prueba de conexión por SSH desde una instancia atacante (inst1) y desde una instancia permitida (inst2)

Usando la aplicación *netcat* se abre el puerto 8080 en modo escucha de la VM master0. Desde la instancia atacante (inst1) y la instancia permitida (inst2) se intenta acceder al puerto e inscribir un mensaje personalizado que se vea reflejado en el *prompt* de master0. Debido a que se ha agregado una regla de *security group* permitiendo que la IP de la VM inst2 pueda conectarse al puerto 8080 de las VMs en la red de acceso, solo se logra imprimir en master0 el mensaje enviado por inst2.

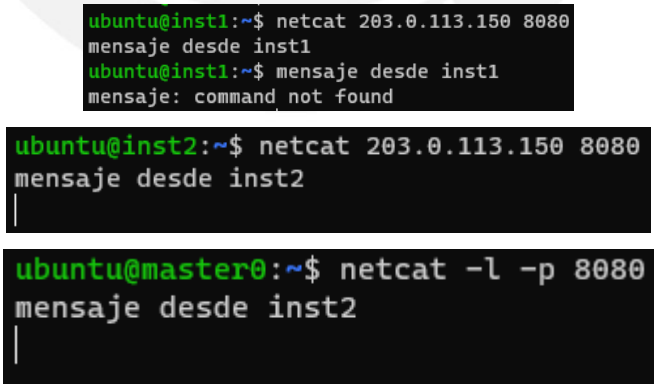

FIGURA 4-31: Prueba de conexión al puerto 8080 usando *netcat* desde una instancia atacante (inst1) y una instancia permitida (inst2)

Con la herramienta *netstat* podemos comprobar los puertos TCP de la VM master0 con conexiones establecidas. Para este caso, se ven los dos puertos

abiertos usados para las pruebas previas con SSH y *netcat* (22 y 8080) con la dirección IP local perteneciente a la red de acceso (10.70.250.89). La dirección remota en ambos casos pertenece a la instancia habilitada en *security groups* inst2, con puertos aleatorios abiertos usados para la comunicación.

```
ubuntu@master0:~$ netstat -antp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN      2878/netcat
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0     36 10.70.250.89:22         203.0.113.170:48306     ESTABLISHED -
tcp        0      1 10.70.250.89:58076      8.8.8.4:53              SYN_SENT    -
tcp        0      0 10.70.250.89:8080       203.0.113.170:52466     ESTABLISHED 2878/netcat
tcp        0      1 10.70.250.89:45964      8.8.8.8:53              SYN_SENT    -
tcp6       0      0 :::22                   :::*                    LISTEN      -
```

FIGURA 4-32: Puertos TCP con conexiones establecidas de la VM master0

4.3.3 Pruebas de protección contra atacantes insiders

Para simular ataques que un *insider* pueda realizar desde un nodo que tenga conexión a un puerto de datos del *switch*, distinto a los nodos compute, se emplea el nodo controller. Aquí, se crea una máquina virtual conectada al OvS br-provider para contar con salida a la interfaz física que está conectada al *switch* de datos y puede comunicarse con los otros servidores de cómputo. En la siguiente imagen se ilustra la conexión interna del nodo *controller* para habilitar al atacante.

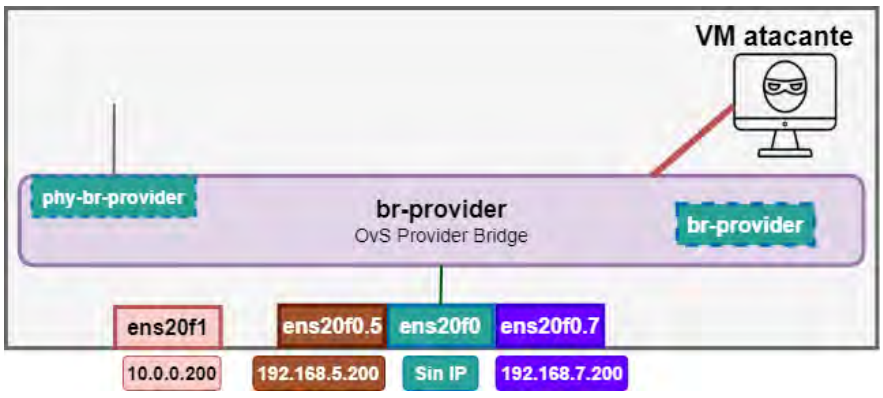

FIGURA 4-33: VM atacante conectada al nodo controller

Dentro de la VM atacante se crean 3 subinterfaces a partir de la única interfaz conectada al OvS *bridge*. Cada subinterfaz posee una VLAN y dirección IP dentro de la subred de las redes pertenecientes al *slice* HPC que se usó para las pruebas previas. Esto es, sin ningún mecanismo de seguridad, la VM atacante podría tener conexión a las VMs dentro de los nodos de cómputo a través de las redes acceso, administración y datos del *slice* HPC. En la siguiente figura se hace una representación de la conexión de la VM atacante con las redes del *slice* HPC objetivo de ataque.

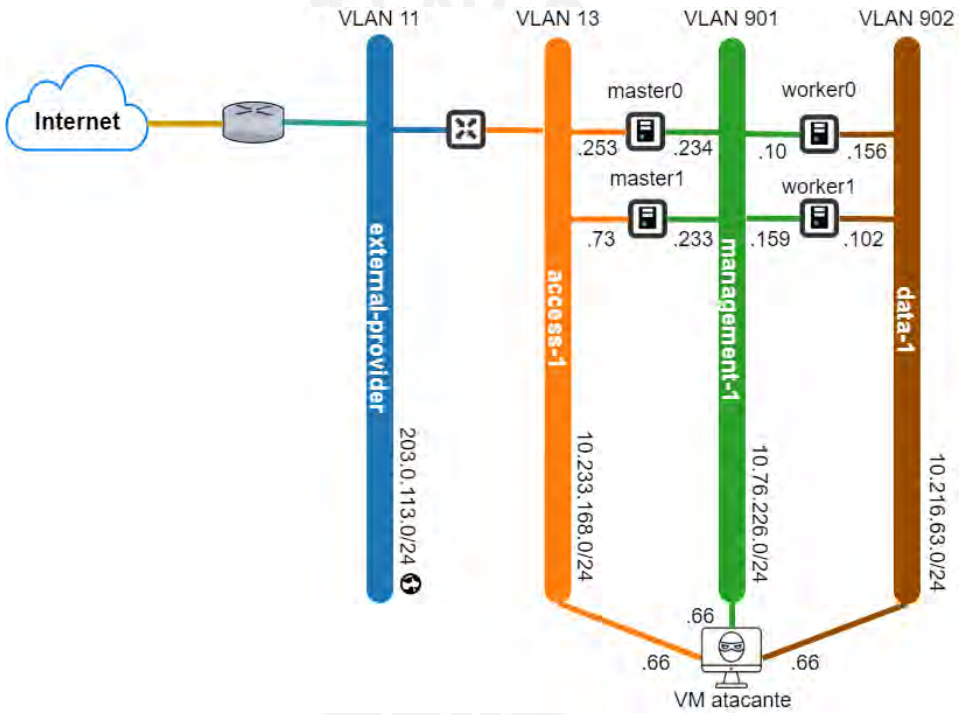

FIGURA 4-34: VM atacante conectada a las VLANs y subredes del *slice* HPC objetivo

En la red de acceso se aplica la seguridad con security groups, pues se emplea el *mechanism driver* de OpenvSwitch que es compatible con esta tecnología. Al implementar *security groups*, los mensajes ARP no podrán ser bloqueados, ya que, *security groups* aplica la seguridad desde la capa IP. Como se observa en la siguiente imagen, la máquina virtual atacante puede intercambiar mensajes ARP con la instancia master0, a través de la red de acceso.

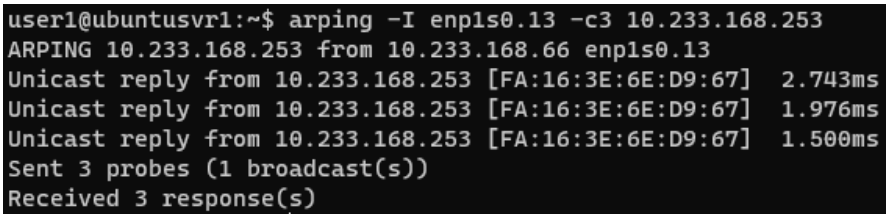

FIGURA 4-35: Prueba *arping* de VM atacante (red de acceso)

Como se mencionó previamente, el filtrado del tráfico de *security groups* en la red de acceso comenzará desde la capa IP. Por este motivo, el intercambio de paquetes ICMP entre la VM atacante y la VM master0 no será exitoso, tal como se muestra en la siguiente imagen.

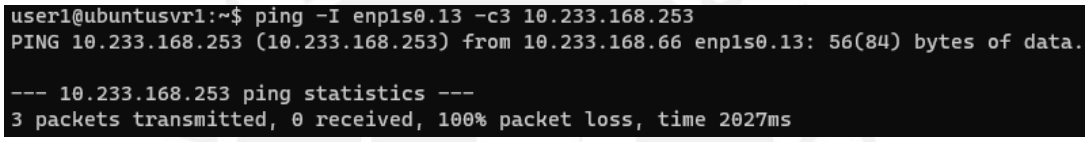

FIGURA 4-36: Prueba *ping* de VM atacante (red de acceso)

La forma de implementar seguridad en la red de administración de los slices HPC es a través de *flows* insertados en el *switch* SDN/OpenFlow. Como se muestra en la siguiente imagen, los mensajes ARP que la VM atacante envía a la VM master0 a través de la red de administración no obtienen respuesta, verificando la protección contra máquinas virtuales no pertenecientes a los nodos de cómputo permitidos.

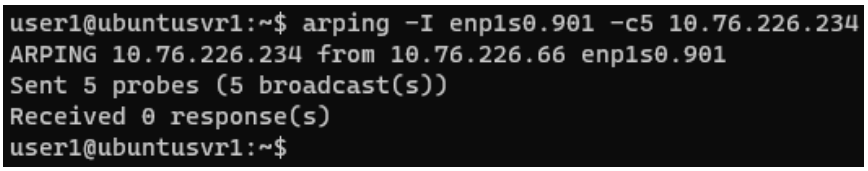

FIGURA 4-37: Prueba *arping* hacia master0 (red de administración)

La protección contra la máquina virtual atacante se garantiza por un *flow* con la menor prioridad respecto a los demás flows insertados en el *switch* para la VLAN en cuestión, cuya acción es la de descartar todos los paquetes entrantes. En la siguiente imagen se muestra el *flow entry* y el incremento de 5 paquetes en las

estadísticas, debido a los 5 ARP *requests* enviados en *broadcast* desde la VM atacante con *arping*.

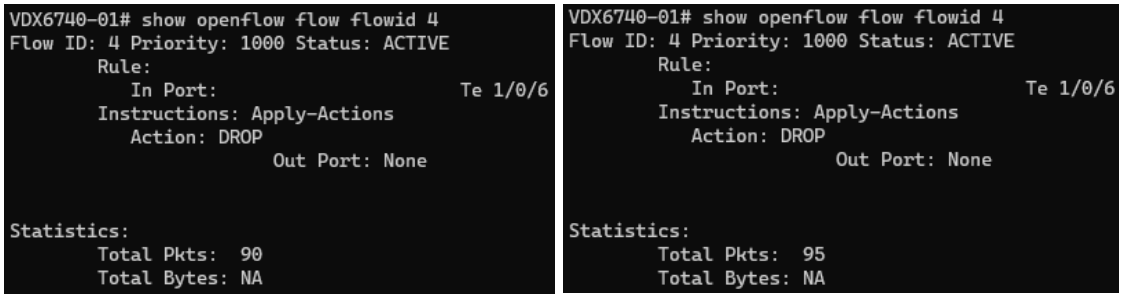

FIGURA 4-38: *Flow* con *match* para tráfico no permitido

Del mismo modo que se ejecutó el comando *arping* con destino a la IP de la interfaz de master0 en la red de administración, también se realizó la prueba con la interfaz de worker0 en la misma red. En la siguiente imagen se ve que los mensajes ARP *requests* enviados en *broadcast* desde la VM atacante no obtienen respuesta, pues la comunicación está siendo filtrada.

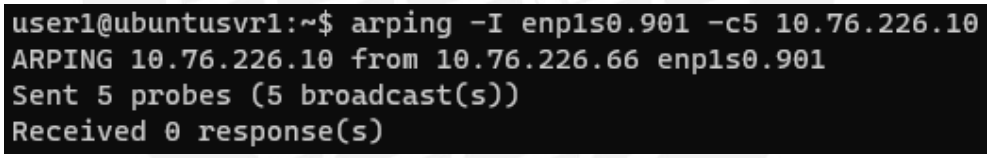

FIGURA 4-39: Prueba *arping* hacia worker0 (red de administración)

En este caso, para la protección de la VM *worker* se usa el mismo *flow* con prioridad mínima y acción de descartar que se empleó para prohibir la comunicación de la VM atacante hacia master0 en la red de administración. En la siguiente imagen se muestra el *flow entry* al que se está haciendo match y el incremento en 5 paquetes en las estadísticas, debido a los 5 ARP *requests* rechazados.

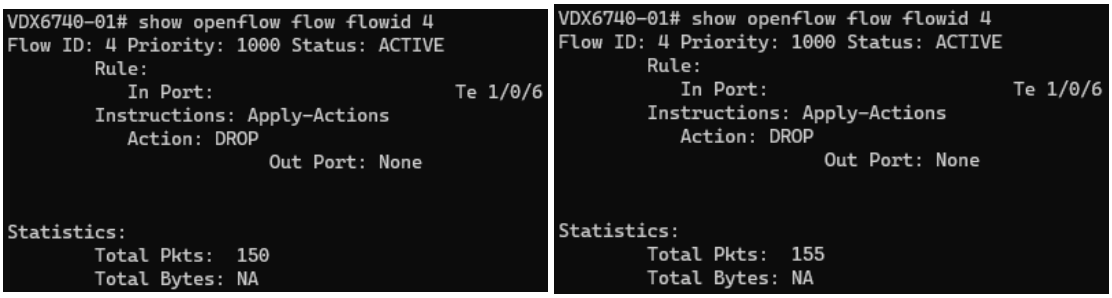

FIGURA 4-40: *Flow* con *match* para tráfico no permitido

Como se muestra en la siguiente imagen, la VM atacante no podrá establecer comunicación con la VM worker0 a través de la red de datos. La VM atacante ejecuta el comando *arping* con la IP de la interfaz de la VM worker0 perteneciente a la red de datos sin obtener respuesta.

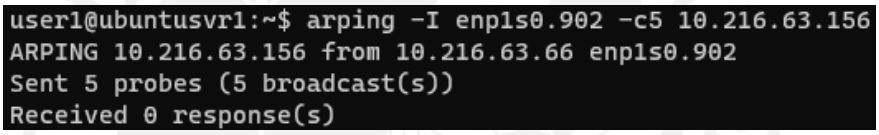

FIGURA 4-41: Prueba *arping* hacia worker0 (red de datos)

Aún para la red de datos se emplea el mismo *flow* usado para proteger a las instancias en la red de administración. Este *flow* con mínima prioridad tiene la acción de descartar el tráfico entrante. En la siguiente imagen se muestra el *flow entry* y el aumento en 5 paquetes del total en las estadísticas del *flow*, debido a los 5 ARP *requests* enviados en *broadcast* desde la VM atacante.

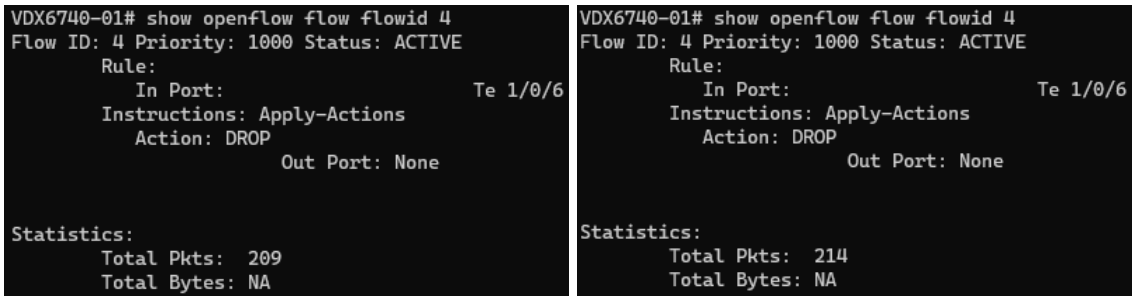

FIGURA 4-42: *Flow* con *match* para tráfico no permitido

4.3.4 Pruebas de protección contra otros tenants

Para la siguiente prueba se emplean 2 slices HPC, con el fin de comprobar la seguridad que se tiene frente a otros *tenants*, es decir, VMs que residen en la misma infraestructura, pero son parte de un *slice* perteneciente a otro cliente. Se probará que existe algún mecanismo de filtración para la comunicación entre una VM (master1) conectada a la red de acceso del *slice* HPC atacante (*slice* 1) con una VM (master0) conectada a la red de acceso del *slice* HPC objetivo (*slice* 0). En la siguiente imagen se muestra la conexión lógica que existe para los 2 *slices* HPC.

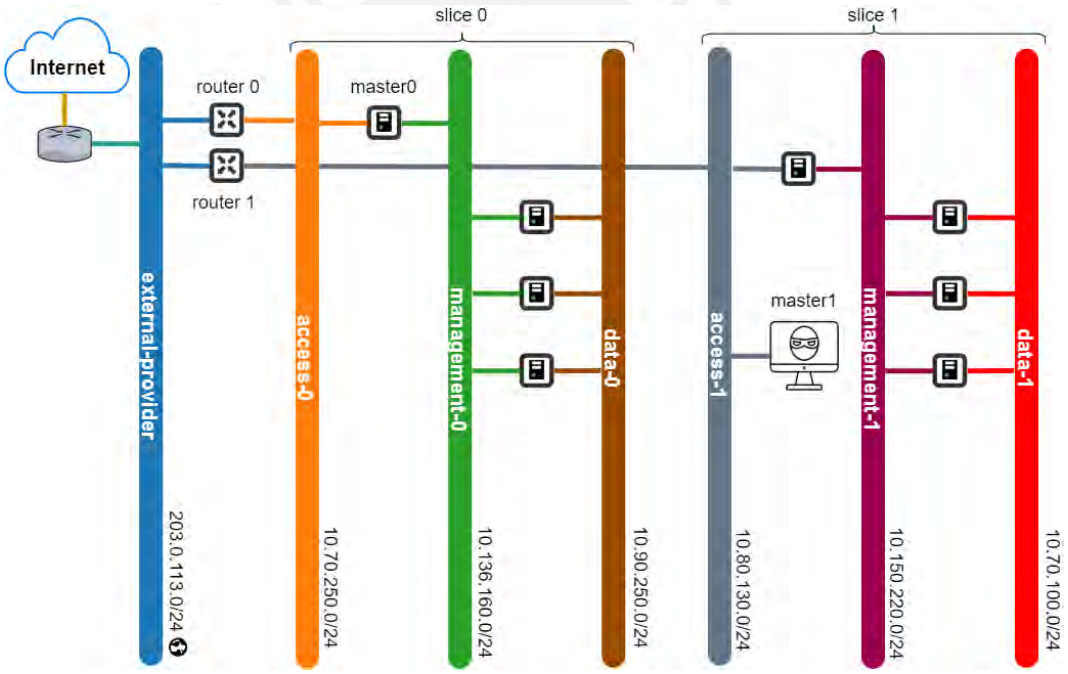

FIGURA 4-43: VM atacante desde otro *slice* HPC

Para que pueda llegar a existir una comunicación entre VMs masters de distintos *slices*, se debe asignar una IP flotante a cada VM *master* perteneciente a la subred asociada a la red *external provider*. Para la comunicación entre master1 (*slice* 1) hacia master0 (slice 0), el tráfico será enrutado por el *router* de *slice* 1 (*router* 1) para salir a la red *external provider* y luego será enrutado nuevamente por el *router* del *slice* 0 (*router* 0) para entrar a la red de acceso (del *slice* 0) y finalmente comunicarse con el destino (master0).

Una vez que ambas VMs cuentan con una IP flotante cada una, el tráfico podrá llegar de una VM a otra; sin embargo, al llegar a la red de acceso de la VM *master* objetivo (master0), el tráfico será filtrado por las reglas existentes en *security groups*, previo a ingresar a la VM. Debido a que la acción por defecto es descartar todo el tráfico entrante a la red, y ya que no se tiene la IP flotante de la VM master1 como permitida en ninguna regla, no será posible una comunicación exitosa entre VMs. Como se muestra en la siguiente imagen, la conexión por SSH desde master1 a master0 no es posible.

```
ubuntu@master1:~$ ssh -i mykeypair.key ubuntu@203.0.113.150
ssh: connect to host 203.0.113.150 port 22: Connection timed out
```

FIGURA 4-44: Prueba de conexión fallida usando SSH desde master1 master0

Para permitir el acceso de la VM master1 a master0 conectándose por SSH (puerto 22), se debe agregar una regla de *security groups* en la que se indique la IP flotante de la VM origen (master1). En la siguiente imagen se muestra lo que se vería en el *dashboard* de OpenStack luego de ingresar la reglas de *security groups* para permitir este tráfico.

☐	Direction	Ether Type	IP Protocol	Port Range ▲	Remote IP Prefix	Remote Security Group
☐	Ingress	IPv4	TCP	22 (SSH)	203.0.113.166/32	-

FIGURA 4-45: Regla de *security group* insertada para permitir conexiones SSH

Luego de ingresada la regla de *security groups*, la VM master1 (del slice 1) se podrá conectar a la VM master0 (del slice 0) por SSH a través de la IP flotante del destino, perteneciente a la red *external provider*. Como se aprecia en la siguiente imagen, la conexión por SSH es exitosa.

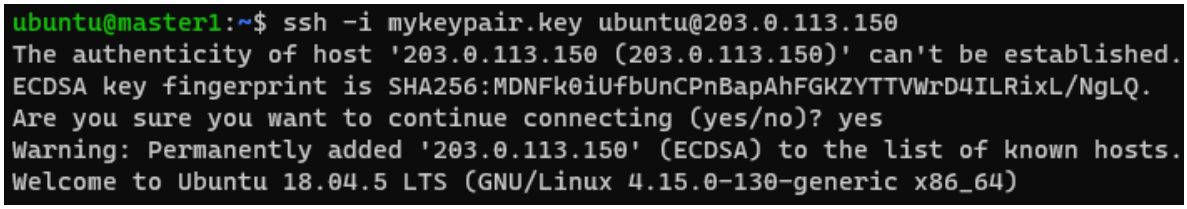

FIGURA 4-46: Prueba de conexión exitosa usando SSH desde master1 master0

Una vez conectados en la VM objetivo (master0), podemos comprobar desde qué IP y puerto (de master1) se ha establecido una conexión con el puerto 22 e IP local de la red de acceso. En la siguiente imagen, se usa la herramienta *netstat* para verificar que existe una conexión establecida con la IP flotante perteneciente a la red *external provider* de la VM master1 y un puerto aleatorio.

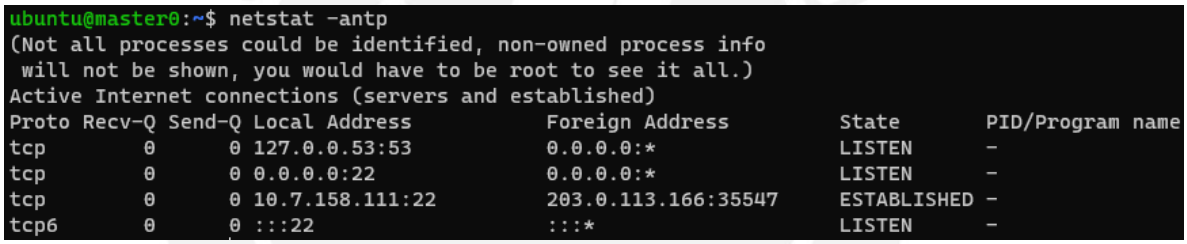

FIGURA 4-47: Conexión SSH establecida en VM master0

4.3.5 Pruebas de rendimiento

Con el fin de comparar el rendimiento que se tiene en una red SR-IOV respecto a una red convencional o *legacy* (implementada mediante OvS *mechanism driver*), en cuanto al uso de ancho de banda, se realizará la presente prueba.

La prueba consiste en realizar múltiples *tests* con la herramienta *iperf3* que nos demuestre la ventaja del ancho de banda disponible al usar una red SR-IOV para la comunicación de máquinas virtuales ubicadas en nodos de cómputo diferentes. Esto se basa en el hecho de que el *passthrough* del *hypervisor*, a través del uso de VFs para la comunicación en red, no se ve afectado por el cuello de botella insertado por la virtualización; tal como es el caso de redes convencionales (OvS).

Las máquinas virtuales usadas en estos *tests* son sistemas Ubuntu Server 20.04 LTS con 2 vCPUs y 2 GB de memoria RAM ubicados en nodos de cómputo distintos. Estos nodos de cómputo están conectados a través del *switch* físico donde se han insertados los *flows* que permiten la comunicación entre ambas VMs. Cada una de estas VMs tiene una interfaz conectada a una red OvS y otra interfaz conectada a una red SR-IOV.

Se usa el protocolo TCP como medio de transporte de paquetes entre cliente y servidor. Cada *test* se replica tanto en la red SR-IOV como en la red OvS de las máquinas virtuales, midiendo qué *bitrate* se logra en la sesión TCP.

Se realizan 4 *tests* distintos, tanto a través de la red SR-IOV como en la red OvS que consisten en lo siguiente:

- **Test 1**: 1 sesión TCP (cliente-servidor), con los parámetros por defecto de *iperf3*, es decir, un *buffer size* de 128KB.
- **Test 2**: 2 sesiones TCP concurrentes (cliente-servidor), con los parámetros por defecto de *iperf3*, es decir, un *buffer size* de 128KB en cada sesión.
- **Test 3**: 1 sesión TCP (cliente-servidor), con un *buffer size* de 1KB.
- **Test 4**: 2 sesiones TCP concurrentes (cliente-servidor), con un *buffer size* de 1KB.

Para todos los *tests*, cada sesión tiene una duración de 100 segundos. En los tests 2 y 4, que emplean 2 sesiones concurrentes, se inicia la segunda sesión 20 segundos después de haber iniciado la primera sesión.

En la siguiente tabla se muestran los valores de *bitrate* obtenidos para cada *test* realizado.

	SR-IOV Bitrate	OvS Bitrate
Test 1: Default buffer size 128KB (1 session)	9.26 Gbits/sec	1.65 Gbits/sec
Test 2: Default buffer size 128KB (2 concurrent sessions)	Session 1: 4.5 Gbits/sec	Session 1: 1.25 Gbits/sec
	Session 2: 4.6 Gbits/sec	Session 2: 758 Mbits/sec
Test 3: Buffer size 1KB (1 session)	2.19 Gbits/sec	1.59 Gbits/sec
Test 4: Buffer size 1KB (2 concurrent sessions)	Session 1: 2.07 Gbits/sec	Session 1: 1.03 Gbits/sec
	Session 2: 2.10 Gbits/sec	Session 2: 892 Mbits/sec

TABLA 4-2: *Bitrate* obtenido para cada *test*

De los resultados obtenidos en Test 1, se puede ver que el *bitrate* de SR-IOV trabaja casi a *line-rate* (10 Gbps) con un valor promedio de 9.26 Gbps; mientras que para OvS el *bitrate* de la sesión alcanza un promedio de 1.65 Gbps. Esto demuestra la mayor eficiencia de ancho de banda que puede lograrse con SR-IOV, así como el *overhead* que introduce usar redes virtuales como OvS.

Respecto a los resultados del Test 2, donde se emplean 2 sesiones concurrentes, se observa una reducción significativa en el *bitrate* de cada sesión para ambos casos (SR-IOV y OvS), respecto al Test 1. Para el caso de SR-IOV, al iniciar la primera sesión TCP, esta mantiene un promedio aproximado de 9.2 Gbps inicialmente. Luego de iniciar la segunda sesión, esta inicia con un promedio de 4.5 Gbps y la primera sesión se reduce a la mitad de su valor inicial aproximadamente (4.6 Gbps). Para el caso de OvS, la primera sesión inicia con un bitrate de 1.6 Gbps y se reduce a 1.2 Gbps al iniciar la segunda sesión. La segunda sesión de OvS inicia con un promedio de 500 Mbps y se mantiene así hasta que la primera sesión finaliza, subiendo el valor de su bitrate a 1.6 Gbps.

En el Test 3, se emplea un *buffer size* de 1KB, y los resultados varían respecto al Test 1. La única sesión TCP obtiene un valor de *bitrate* de 2.19 Gbps en la red SR-IOV y de 1.59 Gbps en la red OvS.

Los resultados obtenidos en Test 4 muestran que las sesiones paralelas de SR-IOV tienen una reducción mínima al comparar el *bitrate* de cada sesión al *bitrate* obtenido en Test 3. Por otro lado, para el caso de SR-IOV, el *bitrate* de cada sesión se degrada respecto al *bitrate* de la única sesión en Test 3. En la red SR-IOV, la primera sesión inicia con un promedio de 2.2 Gbps y se reduce a 2 Gbps aproximadamente, desde que inicia la segunda sesión. Por otro lado, la segunda sesión establecida 20 segundos luego de haber iniciado la primera, tiene un promedio constante de 2.1 Gbps en toda su duración. Para la red OvS, la primera sesión inicia con un *bitrate* de 1.5 Gbps aproximadamente y se reduce a 800 Mbps luego de haber iniciado la segunda sesión. Esta última sesión inicia con un promedio de 800 Mbps e incrementa a 1.5 Gbps aproximadamente cuando la primera sesión finaliza.

Comparando los resultados del Test 3 y Test 4, se puede observar que las sesiones concurrentes sobre la red SR-IOV no se ven tan afectadas en el *bitrate*, como el caso de las sesiones sobre OvS. Para el caso de SR-IOV, el *bitrate* de cada sesión concurrente es aproximadamente igual al *bitrate* obtenido con una única sesión. Mientras que, para el caso de OvS, el bitrate de cada sesión concurrente es casi la mitad del valor del *bitrate* obtenido con una sola sesión. Esto nos demuestra que el *overhead* que ingresa la virtualización de la red, en el caso de OvS *mechanism driver*, tiene un impacto negativo en el ancho de banda disponible.

Conclusiones

- Se logró cumplir el objetivo principal de la tesis al haber desarrollado el módulo de ciberseguridad para el orquestador IaaS HAST, capaz de insertar reglas de seguridad para slices HPC, reemplazando la funcionalidad de *security groups* de OpenStack.
- Se valida que, para los slices HPC, se tiene seguridad y control del tráfico en la red de acceso mediante *security groups*. Asimismo, se verifica la seguridad y control de tráfico para las redes de administración y datos mediante los *flows* insertados en el *switch* de datos.
- Se demuestra que a través de los *flow*s insertados por el módulo de ciberseguridad en el *switch* de datos, se tiene protección contra atacantes *insiders* que cuentan con acceso a un equipo conectado a un puerto del *switch*.
- Se concluye que la implementación del módulo de ciberseguridad es dependiente de la versión implementada de OpenFlow para la comunicación del controlador con el *switch* de datos. Además, para la implementación de los *flows* de seguridad a insertar, se debe considerar las modificaciones realizadas por el *vendor* a la especificación de OpenFlow.
- Se concluye que el rendimiento de *bitrate* para redes *legacy* (por ejemplo, implementadas mediante OvS *mechanism driver*) es inferior a redes implementadas mediante SR-IOV. Al tener múltiples sesiones concurrentes sobre estas redes, el impacto negativo, en cuanto a reducción del *bitrate*, es mucho más notable para las redes *legacy*.

Bibliografía

Todas las FIGURAS y TABLAS sin referencias son de elaboración propia.

[1] W. Kellerer, A. Basta, and A. Blenk, “Flexibility of Networks: a new measure for network design space analysis?,” 2015.

[2] Open Networking Foundation, “Software-Defined Networking: The New Norm for Networks ONF White Paper,” 2012.

[3] NetApp, “What Is High-Performance Computing (HPC)? | How It Works | NetApp,” 2019. [Online]. Available: https://www.netapp.com/us/info/what-is-high-performance-computing.aspx. [Accessed: 22-Jun-2020].

[4] AMAX Engineering, “The Future of High-Performance Computing,” 2019.

[5] Aspen Systems Inc., “HPC Applications for High Performance Computing | Apsen Systems,” 2016. [Online]. Available: https://www.aspsys.com/solutions/hpc-applications/. [Accessed: 22-Jun-2020].

[6] D. Padua *et al.*, “MPI (Message Passing Interface),” in *Encyclopedia of Parallel Computing*, Springer US, 2011, pp. 1184–1190.

[7] V. V. Kindratenko *et al.*, “GPU clusters for high-performance computing,” *Proc. - IEEE Int. Conf. Clust. Comput. ICCC*, 2009.

[8] NVIDIA, “HPC Developer | NVIDIA Developer,” 2015. [Online]. Available: https://developer.nvidia.com/hpc. [Accessed: 22-Jun-2020].

[9] Intel, “Big Data Meets High Performance Computing,” 2014.

[10] I. Amazon Web Services, “High Performance Computing (HPC) | AWS,” 2020. [Online]. Available: https://aws.amazon.com/hpc/. [Accessed: 22-Jun-2020].

[11] Microsoft, “High Performance Computing (HPC) on Azure - Azure Architecture Center | Microsoft Docs,” 2019. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/topics/high-performance-computing. [Accessed: 22-Jun-2020].

[12] Oracle, “High Performance Computing Cloud Infrastructure - HPC | Oracle,” 2020. [Online]. Available: https://www.oracle.com/cloud/solutions/hpc.html. [Accessed: 22-Jun-

2020].

[13] Microsoft, “3D video rendering - Azure Example Scenarios | Microsoft Docs,” 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/example-scenario/infrastructure/video-rendering. [Accessed: 22-Jun-2020].

[14] P. Rad, A. T. Chronopoulos, P. Lama, P. Madduri, and C. Loader, “Benchmarking Bare Metal Cloud Servers for HPC Applications,” in *Proceedings - 2015 IEEE International Conference on Cloud Computing in Emerging Markets, CCEM 2015*, 2016, pp. 153–159.

[15] S. Telfer, “The Crossroads of Cloud and HPC: OpenStack for Scientific Research.”

[16] A. M. Maliszewski, A. Vogel, D. Griebler, E. Roloff, L. G. Fernandes, and N. O. A. Philippe, “Minimizing Communication Overheads in Container-based Clouds for HPC Applications,” in *Proceedings - International Symposium on Computers and Communications*, 2019, vol. 2019-June.

[17] A. Vogel, D. Griebler, C. Schepke, and L. G. Fernandes, “An Intra-Cloud Networking Performance Evaluation on CloudStack Environment,” in *Proceedings - 2017 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2017*, 2017, pp. 468–472.

[18] A. Gupta *et al.*, “The who, what, why, and how of high performance computing in the cloud,” in *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, 2013, vol. 1, pp. 306–314.

[19] J. V. Pelegrino, “Recommendations for high-performance computing on OpenStack - Superuser,” 2018. [Online]. Available: https://superuser.openstack.org/articles/hpc-openstack-recommendations/. [Accessed: 23-Jun-2020].

[20] Nir Yechiel, “Red Hat Enterprise Linux OpenStack Platform 6: SR-IOV Networking - Part I: Understanding the Basics,” 2015. [Online]. Available: https://www.redhat.com/en/blog/red-hat-enterprise-linux-openstack-platform-6-sr-iov-networking-part-i-understanding-basics. [Accessed: 23-

Jun-2020].

[21] OpenStack, “OpenStack Docs: SR-IOV,” 2019. [Online]. Available: https://docs.openstack.org/neutron/ussuri/admin/config-sriov.html. [Accessed: 23-Jun-2020].

[22] “Red Hat Enterprise Linux OpenStack Platform 6: SR-IOV Networking - Part II: Walking Through the Implementation,” 2015. [Online]. Available: https://www.redhat.com/en/blog/red-hat-enterprise-linux-openstack-platform-6-sr-iov-networking-part-ii-walking-through-implementation. [Accessed: 23-Jun-2020].

[23] I. Foster and D. B. Gannon, *Cloud Computing for Science and Engineering*. Massachusetts Institute of Technology: The MIT Press, 2017.

[24] J. Zhang, X. Lu, and D. K. Panda, “High-Performance Virtual Machine Migration Framework for MPI Applications on SR-IOV Enabled InfiniBand Clusters,” in *Proceedings - 2017 IEEE 31st International Parallel and Distributed Processing Symposium, IPDPS 2017*, 2017, pp. 143–152.

[25] A. Richter, C. Herber, S. Wallentowitz, T. Wild, and A. Herkersdorf, “A Hardware/Software Approach for Mitigating Performance Interference Effects in Virtualized Environments Using SR-IOV,” in *Proceedings - 2015 IEEE 8th International Conference on Cloud Computing, CLOUD 2015*, 2015, pp. 950–957.

[26] A. Richter, C. Herber, T. Wild, and A. Herkersdorf, “Resolving Performance Interference in SR-IOV Setups with PCIe Quality-of-Service Extensions,” in *Proceedings - 19th Euromicro Conference on Digital System Design, DSD 2016*, 2016, pp. 454–462.

[27] Drew Jackman, “SR-IOV – The way to share real virtualized host devices with a virtual machine - SUSE Communities,” 2018. [Online]. Available: https://suse.com/c/sr-iov-way-share-real-virtualized-host-devices-virtual-machine/. [Accessed: 23-Jun-2020].

[28] Juniper Networks, “Configuring SR-IOV and PCI Passthrough on KVM - TechLibrary - Juniper Networks,” 2019. [Online]. Available: https://www.juniper.net/documentation/en_US/vsrx/topics/task/multi-

task/security-vsrx-kvm-sr-iov-pci-configuring.html. [Accessed: 09-Dec-2019].

[29] Red Hat, “Chapter 16. SR-IOV Red Hat Enterprise Linux 5 | Red Hat Customer Portal.” [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/virtualization/chap-para-virtualized_windows_drivers_guide-sr_iov. [Accessed: 09-Dec-2019].

[30] M. Tim Jones, “Virtualización Linux y PCI passthrough,” 2009. [Online]. Available: https://www.ibm.com/developerworks/ssa/library/l-pci-passthrough/index.html. [Accessed: 23-Jun-2020].

[31] Intel, “Frequently Asked Questions for SR-IOV on Intel® Ethernet Server...,” 2017. [Online]. Available: https://www.intel.com/content/www/us/en/support/articles/000005722/network-and-i-o/ethernet-products.html. [Accessed: 23-Jun-2020].

[32] I. Corporation, “Intel ® 82599 SR-IOV Driver Companion Guide Overview of the Intel ixgbe SR-IOV Driver Implementation,” no. May, pp. 1–44, 2010.

[33] OpenStack, “What is OpenStack?” [Online]. Available: https://www.openstack.org/software/. [Accessed: 09-Dec-2019].

[34] Red Hat, “El concepto de OpenStack.” [Online]. Available: https://www.redhat.com/es/topics/openstack. [Accessed: 09-Dec-2019].

[35] “What is OpenStack?” [Online]. Available: https://www.openstack.org/software/project-navigator/openstack-components#openstack-services. [Accessed: 24-Jun-2020].

[36] J. Denton, “Learning OpenStack Networking.” Packt Publishing, Birmingham, 2018.

[37] Linux Foundation Collaborative Project, “Open vSwitch.” [Online]. Available: http://www.openvswitch.org/. [Accessed: 24-Jun-2020].

[38] A. Swanson, “Tenant networks vs. provider networks in the private cloud context - Superuser,” 2016. [Online]. Available: https://superuser.openstack.org/articles/tenant-networks-vs-provider-networks-in-the-private-cloud-context/. [Accessed: 03-Jan-2021].

[39] “OpenStack Docs: Native Open vSwitch firewall driver.” [Online]. Available: https://docs.openstack.org/neutron/victoria/admin/config-ovsfwdriver.html. [Accessed: 12-Feb-2021].

[40] “OpenStack Docs: Open vSwitch Firewall Driver.” [Online]. Available: https://docs.openstack.org/neutron/victoria/contributor/internals/openvswitch_firewall.html. [Accessed: 12-Feb-2021].

[41] N. Abbas, “Managing port level security in OpenStack - Superuser,” 2017. [Online]. Available: https://superuser.openstack.org/articles/managing-port-level-security-openstack/. [Accessed: 03-Jan-2021].

[42] Open Networking Foundation, “Software-Defined Networking (SDN) Definition - Open Networking Foundation.” [Online]. Available: https://www.opennetworking.org/sdn-definition/?nab=1. [Accessed: 09-Dec-2019].

[43] IETF, “RFC 7426 - Software-Defined Networking (SDN): Layers and Architecture Terminology.” [Online]. Available: https://tools.ietf.org/html/rfc7426. [Accessed: 09-Dec-2019].

[44] E. Haleplidis, “Overview of RFC7426: SDN Layers and Architecture Terminology - IEEE Software Defined Networks.” [Online]. Available: https://sdn.ieee.org/newsletter/september-2017/overview-of-rfc7426-sdn-layers-and-architecture-terminology. [Accessed: 09-Dec-2019].

[45] S. Voruganti and S. Subramanian, *Software-Defined Networking (SDN) with OpenStack*. .

[46] M. K. Shin, K. H. Nam, and H. J. Kim, “Software-defined networking (SDN): A reference architecture and open APIs,” *Int. Conf. ICT Converg.*, pp. 360–361, 2012.

[47] Flowgrammable, “SDN / OpenFlow.” [Online]. Available: http://flowgrammable.org/sdn/openflow/. [Accessed: 09-Dec-2019].

[48] Open Networking Foundation, “OpenFlow Switch Specification,” 2013.

[49] L. Mamushiane, A. Lysko, and S. Dlamini, “A comparative evaluation of the performance of popular SDN controllers,” *IFIP Wirel. Days*, vol. 2018-April, pp. 54–59, 2018.

[50] Open Networking Foundation, “SDN architecture,” 2014.

[51] J. Medved, R. Varga, A. Tkacik, and K. Gray, “OpenDaylight: Towards a model-driven SDN controller architecture,” *Proceeding IEEE Int. Symp. a World Wireless, Mob. Multimed. Networks 2014, WoWMoM 2014*, pp. 1–6, 2014.

[52] F. Pakzad, “Comparison of Software Defined Networking (SDN) Controllers. Part 3: OpenDayLight (ODL) - Aptira.” [Online]. Available: https://aptira.com/comparison-of-software-defined-networking-sdn-controllers-part-3-opendaylight-odl/. [Accessed: 09-Dec-2019].

[53] “OpenDaylight Helium Architecture.” [Online]. Available: https://cdn.mirantis.com/wp-content/uploads/2015/04/opendalightfigure1.png. [Accessed: 24-Jun-2020].

[54] P. Berde *et al.*, “Facilitation of the OpenDaylight Architecture,” *Proc. third Work. Hot Top. Softw. Defin. Netw. - HotSDN ’14*, no. May, pp. 1–6, 2014.

[55] “Controller — OpenDaylight Documentation Oxygen documentation.” [Online]. Available: https://docs.opendaylight.org/en/stable-oxygen/developer-guide/controller.html. [Accessed: 27-Jun-2020].

[56] “OpenFlow Plugin Project User Guide — OpenDaylight Documentation Neon documentation.” [Online]. Available: https://docs.opendaylight.org/en/stable-neon/user-guide/openflow-plugin-project-user-guide.html. [Accessed: 27-Jun-2020].

[57] Brocade Communications Systems Inc, “Network OS Software Defined Networking (SDN),” no. February, pp. 1–30, 2016.

[58] “OpenStack Docs: Install and configure a compute node for Ubuntu.” [Online]. Available: https://docs.openstack.org/nova/ussuri/install/compute-install-ubuntu.html. [Accessed: 30-Jul-2020].