Universidad Zaragoza

Trabajo Fin de Grado

Evaluación del uso de Open vSwitch en Proxmox para el análisis de tráfico telefónico

Evaluating the use of Open vSwitch in Proxmox for the analysis of telephone traffic

Autor

Carlos Irigoyen Álvarez

Director

Fernando Mayo Canuria

Ponente

Julián Fernández Navajas

Escuela de Ingeniería y Arquitectura

2023

Resumen

Dentro de la tecnología que conllevan las llamadas telefónicas, en particular, en la telefonía IP, una de las partes clave para que la experiencia de usuario sea la indicada es la iniciación correcta de estas llamadas. El encargado de realizar este proceso es el protocolo **SIP.** Es ampliamente importante para las empresas operadoras con telefonía VoIP, poder controlar este proceso de establecimiento, mantenimiento y finalización de las llamadas, para poder obtener estadísticas o tarificar el uso de sus sistemas, entre otras cosas. Es por ello que estas empresas buscan trabajar con las aplicaciones más óptimas a fin de minimizar los errores y hacer que los usuarios disfruten sin problemas de la telefonía.

Durante este Trabajo Fin de Grado se desarrolla una plataforma de pruebas capaz de realizar un análisis de tráfico telefónico mediante aplicaciones relacionadas con la telefonía IP en servidores virtualizados. Para ello, se va a utilizar el software llamado *Proxmox*, utilizado por la empresa *System One Noc & Development Solutions, S.A.* (en adelante SONOC, empresa con la que se realiza el proyecto) en casi la totalidad de sus servidores, y con multitud de posibilidades en lo que a virtualización y optimización de recursos se refiere. SONOC es una experta en tecnologías de voz sobre IP (VoIP), que nos permiten realizar y recibir llamadas de voz a través de la red, a diferencia de la red telefónica convencional, mucho menos práctica. El protocolo de uso más importante hoy en día para esta tecnología es SIP. Sobre todo ello se basará este proyecto.

Con *Proxmox*, es posible hacer uso de muchas tecnologías. Entre ellas, nuestro estudio se centra en analizar el funcionamiento de **Open vSwitch** (software de conmutación virtual), y las ventajas que nos puede aportar a la hora de analizar tráfico telefónico, en sustitución de los **Linux Bridge** convencionales (la opción por defecto de arquitectura de red en Proxmox). Open vSwitch es un software diseñado para utilizarse como bridge virtual en entornos de servidores virtualizados. Se encarga de reenviar tráfico entre máquinas virtuales, o entre máquinas virtuales y la red física. Contiene cantidad de funcionalidades que analizaremos, con la finalidad de comprobar si su uso por parte de la empresa pudiera ser beneficioso respecto al sistema actual. En concreto, se evaluará en detalle la posibilidad de realizar *port mirroring*, para concentrar el tráfico a analizar en una sola máquina virtual.

Después de analizar las tecnologías y los programas que se utilizan, pasamos a implementar una plataforma sobre la que hacer las pruebas, poniendo a funcionar un Open vSwitch desde Proxmox y haciendo diversas pruebas de funcionamiento con las máquinas virtuales que utilizamos, simulando una situación real de tráfico SIP. Una vez el escenario es correcto en términos de conectividad, se genera tráfico telefónico mediante una aplicación de simulación.

Finalmente se recogen resultados y conclusiones respecto al uso de Open vSwitch en el sistema de virtualización Proxmox, dejando en claro las ventajas o inconvenientes que nos aporta en comparación con la implementación actual que utiliza SONOC (Linux Bridges convencional) para realizar este tipo de análisis de tráfico VoIP.

Abstract

Within the technology involved in telephone calls, particularly in IP telephony, one of the key parts of the user experience is the correct initiation of these calls. The SIP protocol is responsible for this process. It is very important for VoIP telephony operators to be able to control this process of establishing, maintaining and terminating calls, in order to obtain statistics or rate the use of their systems, among other things. That is why these companies seek to work with the most optimal applications in order to minimize errors and make users enjoy telephony without problems.

During this Final Degree Project, a test platform capable of performing a telephone traffic analysis using applications related to IP telephony on virtualized servers is developed. For this, we will use the software called Proxmox, used by the company System One Noc & Development Solutions, S.A. (hereinafter SONOC, the company with which the project is carried out) in almost all of its servers, and with a multitude of possibilities in terms of virtualization and resource optimization. SONOC is an expert in voice over IP (VoIP) technologies, which allow us to make and receive voice calls over the network, unlike the conventional telephone network, which is much less practical. The most important protocol in use today for this technology is SIP. This project will be based on this.

With Proxmox, it is possible to make use of many technologies. Among them, our study focuses on analyzing the operation of Open vSwitch (virtual switching software), and the advantages it can bring us when analyzing telephone traffic, replacing conventional Linux Bridges (the default network architecture option in Proxmox). Open vSwitch is a software designed to be used as a virtual bridge in virtualized server environments. It is responsible for forwarding traffic between virtual machines, or between virtual machines and the physical network. It contains a number of functionalities that we will analyze, in order to check if its use by the company could be beneficial with respect to the current system. In particular, the possibility of *port mirroring* will be evaluated in detail, in order to concentrate the traffic to be analyzed on a single virtual machine.

After analyzing the technologies and software used, we will implement a platform on which to test, running an Open vSwitch from Proxmox and performing various performance tests with the virtual machines we use, simulating a real situation of SIP traffic. Once the scenario is correct in terms of connectivity, telephone traffic is generated using a simulation application.

Finally, results and conclusions regarding the use of Open vSwitch in the Proxmox virtualization system are gathered, making clear the advantages or disadvantages that it brings compared to the current implementation that uses SONOC (conventional Linux Bridges) to perform this type of VoIP traffic analysis.

Índice

Capítulo 1: Introducción

1.1. Introducción

Hoy en día, las tecnologías de comunicación están presentes en todos los aspectos de nuestras vidas, y no paran de desarrollarse y evolucionar. Una de las tecnologías que ha revolucionado la comunicación empresarial es la **Telefonía IP (ToIP)**. Con la telefonía IP, las empresas y usuarios pueden realizar videoconferencias, compartir archivos y datos, y mantener una comunicación constante y efectiva en tiempo real. En resumen, la Telefonía IP es una tecnología fundamental en la actualidad para los usuarios que desean mejorar su comunicación y aumentar su eficiencia.

Esta Telefonía IP es ampliamente utilizada, ya que conforma varias ventajas en comparación con la telefonía convencional. Entre ellas, unos costes reducidos, mayor flexibilidad y escalabilidad, funciones avanzadas como son las llamadas de conferencia, llamadas en espera, identificador de llamadas, grabación… Esto hace que hoy en día sea la tecnología que va sustituyendo a la telefonía convencional.

Es de vital importancia comprender perfectamente cómo funciona la empresa con la que se realiza el proyecto, de forma que así entendamos qué papel jugaría la implementación que se está estudiando. Además, queremos conseguir desarrollar una plataforma de pruebas que pueda asemejarse (en la medida de lo posible) a lo que pasaría en la realidad, por lo que el esquema tiene que ser consonante con el real.

La empresa SONOC es un proveedor de servicios de monitoreo y gestión de redes de tecnología de la información (TI) para empresas y organizaciones, y ofrece soluciones de **monitoreo** proactivo, **gestión de incidentes** y **soporte técnico.** Opera varias plataformas de intercambio de tráfico VoIP (Voz sobre IP, que es la base de ToIP) distribuidas por todo el mundo.

Dichas plataformas, en esencia, se componen de un switch VoIP y de una serie de aplicaciones de gestión y facturación del tráfico. La arquitectura del softswitch (software en una computadora que realiza la tarea de conmutación en los sistemas VoIP), es un desarrollo propio llamado "dedalus" y se compone de varios bloques, distribuidos en diferentes servidores.

Para las empresas encargadas de la gestión de comunicaciones telefónicas, es crucial que se realice un análisis detallado de todas estas llamadas, con dos finalidades principales: primero, para analizar la **calidad** de ellas, pudiendo hacerlo a través de diversos parámetros como el retardo, el jitter, la pérdida de paquetes, el nivel de ruido… por otro lado, será de crucial importancia saber detectar y gestionar distintos problemas que se puedan dar en la red. Por ejemplo, si una operadora no está siendo capaz de conectarse con otra, la empresa que “ofrece” este servicio de comunicación entre ellas, tiene que ser capaz de detectar en el menor tiempo posible qué problemas están sucediendo y cómo se puede actuar ante ellos, etc. Para este

propósito, se dispone de aplicaciones de monitoreo y detección de errores, así como encargadas de extraer estadísticas o tarificar correctamente estas comunicaciones.

Por todo ello, se trabaja con distintas aplicaciones que perfeccionan este análisis constantemente mientras se van desarrollando. Para analizar el tráfico telefónico y detectar posibles problemas, se utilizan entornos que permitan redirigir el tráfico hacia los equipos de monitoreo. Es muy común que para estos equipos de análisis se utilice virtualización con la finalidad de aprovechar al máximo los recursos hardware. Proxmox es un entorno de virtualización muy útil a la hora de sacar el máximo partido de los recursos y nos permite crear escenarios para probar aplicaciones que analizan el tráfico de llamadas telefónicas. Dentro de Proxmox, la arquitectura de red convencional, mostrada en la *Figura 1*, está basada en el uso de Linux Bridge (la más común y en uso en la actualidad), pero no permite redirigir el tráfico de un interfaz de red de un determinado host virtual, hacia otro host virtual en la misma máquina donde se puedan recibir todos los tráficos de interés. Pero también es posible instalar un Open vSwitch como bridge virtual, que sí que permite redirigir el tráfico. Esto es una gran ventaja, pues permite disponer de una máquina virtual de análisis/trouble shooting dentro de cada host Proxmox.

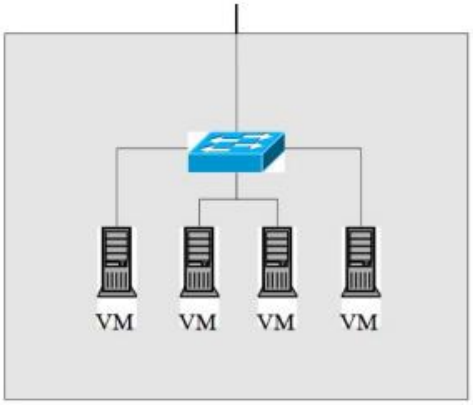


Figura 1. Bridge virtual.

En definitiva, para las compañías que ofrecen servicios de telefonía (como lo es SONOC), algo básico y fundamental es poder detectar qué está pasando (*trouble shooting*) si un cliente no obtiene el resultado que esperaba de las comunicaciones que ha contratado. Centrándonos en ello, puede ser interesante utilizar Open vSwitch (software abierto para la creación de switch virtuales) como bridge virtual, ya que nos podremos aprovechar de varias ventajas, como la posibilidad de hacer *port mirroring*, que hacen este análisis o *trouble shooting* más sencillo y efectivo. Por tanto, se analizará su uso en detalle, con el fin de comprobar si beneficiaría o no.

1.2. Motivación y objetivos

El objetivo principal de este Trabajo Fin de Grado es el estudio y evaluación de las funcionalidades que nos proporciona la tecnología **Open vSwitch** y que permiten el análisis de tráfico telefónico. Esta tecnología a estudiar es soportada por servidores virtualizados con el software Proxmox, con los que realizaremos este estudio.

La idea de utilizar Open vSwitch está motivada por las ventajas que esta tecnología puede ofrecernos en comparación con el uso del sistema de conmutación virtual utilizado hoy en día en los servidores Proxmox (Linux Bridge). Esto se basa, principalmente, en plantearse una solución más sencilla y cómoda a los problemas que supone hoy en día, para la empresa con la que se trabaja, el análisis correcto o detección de problemas de las comunicaciones telefónicas que se quieren establecer.

El funcionamiento actual ofrece posibilidades parciales o incompletas en cuanto a detectar fallos en las comunicaciones. Se realizan análisis incómodos o poco efectivos ya que es necesario acceder a equipos donde el tráfico telefónico a estudiar está mezclado con cualquier otro tipo de paquetes, además de ser equipos en producción en los que no es deseable manipular demasiado.

Esta problemática de no poder analizar el tráfico telefónico tratado por los servidores de una forma realmente eficaz, se basa en las limitaciones que tienen los Linux Bridge de los servidores Proxmox. Sería realmente interesante poder capturar el tráfico que manejan los propios servidores virtualizados, dentro de ellos mismos. Esto en la actualidad no es posible realizarlo, ya que los Linux Bridge no permiten redireccionar el tráfico de una interfaz de web de un host virtual, hacia otro host virtual en la misma máquina Proxmox.

Open vSwitch sí es capaz de llevar a cabo esta funcionalidad, ya que puede realizar *port mirroring* de los puertos que nos interesen analizar. Por tanto, el objetivo es conseguir implementar en el sistema actual esta tecnología, consiguiendo reunir el tráfico susceptible a ser analizado en una única máquina dentro del propio servidor, que haría las funciones de monitoreo, análisis, facturación, etc. De esta forma se conseguiría evitar el análisis incómodo en equipos físicos de conmutación, y se podrá elegir qué tráfico nos interesa recibir en esta máquina de análisis.

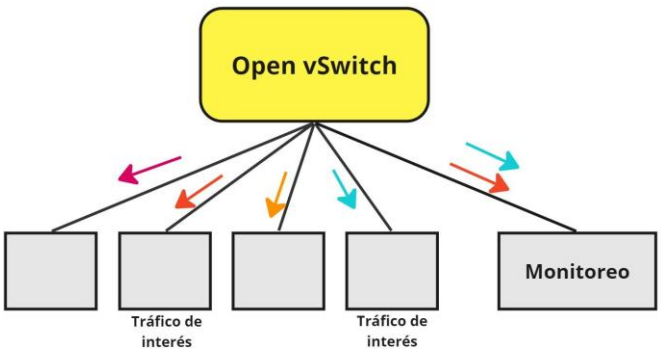


Figura 2. Esquema del funcionamiento deseado

1.3. Organización de la memoria

La memoria se presenta de acuerdo a la siguiente disposición:

- Capítulo 1: Se hace una introducción del trabajo, y se destacan las motivaciones y objetivos del mismo.
- Capítulo 2: Se describen los protocolos y herramientas utilizadas, se hace un análisis previo de la situación actual y del funcionamiento de la empresa, y se plantea una solución final.
- Capítulo 3: Se expone el sistema a desarrollar, el entorno de trabajo y las principales funcionalidades.
- Capítulo 4: Se detalla el despliegue realizado del sistema, desarrollando su instalación, funcionamiento y configuración.
- Capítulo 5: Se explican y presentan las pruebas realizadas durante el trabajo y se muestran los resultados de las mismas.
- Capítulo 6: Se exponen las conclusiones finales del trabajo y las posibles líneas futuras.

Finalmente, se incluyen varios anexos explicativos al final del documento:

- Anexo A: Desarrollo en detalle de protocolos y herramientas.
- Anexo B: Funcionamiento de la empresa con la que se trabaja.
- Anexo C: Entorno de trabajo, conectividad.
- Anexo D: Configuración de los servidores de trabajo.
- Anexo E: Explicación de la interfaz web de Proxmox.
- Anexo F: Configuración de red con Open vSwitch.
- Anexo G: Códigos y funcionamiento del *Port Mirroring*.
- Anexo H: Aplicación SIPp.
- Anexo I: Aplicaciones de monitoreo Heplify y Homer.
- Anexo J: Aplicación de análisis Grafana.

Capítulo 2: Análisis previo

2.1. Protocolos y herramientas utilizadas

En este apartado se van a describir los principales protocolos y herramientas que se han utilizado a lo largo de este proyecto. Además, se puede consultar más información respecto a ellas en el ANEXO A, donde también se dan a conocer algunas herramientas con menor relevancia, pero que se aconseja leer para una mejor compresión de la memoria.

Protocolo SIP

En el contexto del proyecto, basado en el análisis de tráfico telefónico, el protocolo SIP es de particular importancia, ya que se utiliza para establecer y controlar las llamadas telefónicas sobre redes IP. Al analizar el tráfico telefónico basado en SIP, se puede obtener información valiosa sobre las llamadas realizadas, como la duración, origen y destino, servicios utilizados, flujos de datos… y otros detalles. Por ello, para comprender los objetivos de este proyecto, es vital conocer cómo funciona este protocolo.

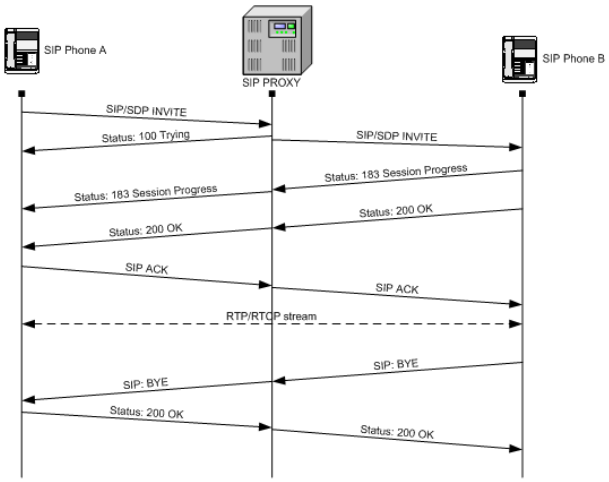


Figura 3. Esquema de funcionamiento del protocolo SIP

El Protocolo de Inicio de Sesión (SIP por sus siglas en inglés) es un protocolo de señalización utilizado en redes de comunicación de voz y vídeo a través de internet. Fue desarrollado por el IETF (grupo de trabajo de ingeniería de internet), y está definido en el estándar RFC 3261.

SIP es un protocolo basado en texto que facilita el establecimiento, modificación y finalización de sesiones de comunicación multimedia entre dos o más participantes. Estas comunicaciones pueden ser llamadas de voz, videoconferencias, mensajería instantánea o transferencia de archivos. Se utiliza, mayoritariamente, en el mundo de la telefonía IP. Cabe destacar cómo su mecanismo de petición-respuesta facilita la resolución de errores.

Los mensajes SIP describen la identidad de los participantes en una llamada y cómo los participantes pueden ser localizados sobre una red IP. Una vez que el intercambio de los mensajes de configuración es completado, la comunicación multimedia utiliza otro protocolo, típicamente RTP (Real-Time Transmission Protocol).

Se intercambian diferentes tipos de mensajes entre los participantes, de los cuales los más comunes quedan explicados en el ANEXO A.

SIP Proxy

Un Proxy SIP es un componente de las redes de comunicación SIP que se encarga de gestionar y enrutar las llamadas o sesiones de comunicación. Actúa como intermediario entre dispositivos SIP tomando decisiones de enrutamiento. Su objetivo es facilitar y controlar las comunicaciones SIP de manera eficiente y segura.

Protocolo LACP (IEEE 802.3ad)

LACP (Link Aggregation Control Protocol) es un protocolo de control utilizado para combinar dos o más enlaces físicos en un único enlace lógico de mayor capacidad. Su implementación en los sistemas sirve para aumentar el ancho de banda y mejorar la redundancia. Asegura una distribución equitativa del tráfico y proporciona conectividad en caso de fallos en los enlaces individuales.

Proxmox Virtual Environment

Proxmox VE (Virtual Environment) es una solución empresarial de código abierto para el despliegue de un entorno de virtualización. Su objetivo es ayudar a optimizar el uso de los recursos ya existentes, minimizar el costo por hardware y el tiempo empleado. Utiliza habitualmente Debian GNU/Linux como sistema operativo mediante un Kernel de Linux personalizado. Además, permite su instalación sobre un Debian ya existente.

A parte de la conexión mediante consola a los servidores virtualizados ubicados en Proxmox, éste ofrece también la posibilidad de conectarse a través de una **interfaz web,** que ayuda a administrar fácilmente máquinas virtuales y contenedores, recursos de almacenamiento y redes definidas por software, agrupación en clúster de alta disponibilidad y, además, ofrece otras herramientas adicionales, que analizaremos en detalle.

Cada servidor con Proxmox instalado se convierte en un NODO y puede trabajar de forma independiente o puede estar agrupado en un Cluster. El beneficio de definir un **Cluster** es tener la administración centralizada, poder mover máquinas entre cada nodo, activar "Alta Disponibilidad" y aprovechar al máximo los recursos de los equipos físicos para la virtualización. En nuestro caso hemos definido un Cluster con los dos servidores Proxmox que utilizamos. De esta manera conectándonos a uno, podremos configurar ambos, ganando en comodidad.

Open vSwitch (OvS)

Open vSwitch, abreviado OvS, es un software de código abierto diseñado para ser utilizado como un *switch* virtual. Es el encargado de reenviar el tráfico entre diferentes máquinas virtuales (VMs) en el mismo host físico y también reenviar el tráfico por los interfaces reales de una red física.

OvS es un software multicapa cuyo objetivo es la implementación de una plataforma de calidad que soporte interfaces de gestión estándar y que exponga las funciones de forwarding de forma programable. Está bien adaptado para trabajar como un switch virtual en ambientes implementados con máquinas virtuales. Además de exponer interfaces estándar de control y visibilidad con la capa de red virtual, fue diseñado para soportar una distribución a través de múltiples servidores físicos.

Open vSwitch tiene la capacidad de conectarse con un controlador SDN a través del protocolo OpenFLow.

Port Mirroring

Es una función en redes que permite la copia del tráfico de red de un puerto o puertos a otro. Es utilizado para monitorear y analizar tráfico sin interrumpir la operación normal de la red. Esta característica es útil para detectar errores en la red, hacer análisis de seguridad o pruebas de rendimiento.

SIPp

SIPp es una herramienta de código abierto de prueba de rendimiento mediante generación de tráfico para el **protocolo SIP**. Incluye algunos escenarios básicos de agentes de usuario (UAC y UAS), y establece y libera múltiples llamadas con los métodos INVITE y BYE. Además puede leer archivos *.xml* de escenarios personalizados que pueden describir flujos más complejos de llamadas.

También cuenta con la visualización dinámica de estadísticas sobre la ejecución de pruebas (tasa de llamadas, retrasos, estadísticas de mensajes…), volcados periódicos de estadísticas y un ajuste de la tasa de llamadas dinámico.

SIPp se puede utilizar para probar varios equipos SIP reales como proxies SIP, servidores de medios SIP, puertas de enlace SIP/x... También es muy útil para emular miles de agentes de usuario que llaman a su sistema SIP.

Heplify - Homer

Homer SIP capture server es muy probablemente el programa Open Source más utilizado de captura y análisis de la señalización SIP. Se basa en el protocolo HEP, que es un protocolo de encapsulación que permite trabajar con distintos protocolos de señalización y de flujo media; los paquetes HEP llegan al servidor de recogida donde se guardan en una base de datos que luego es posible consultar a través de una interfaz Web. Se pueden consultar ejemplos de lo que muestra en el ANEXO I.2**.**

Heplify es un único ejecutable que puede trabajar sobre Linux, ARM, MIPS o Windows para capturar paquetes IPv4 o IPv6 y enviarlos a HOMER, que puede estar instalado en otro equipo para realizar un análisis exhaustivo. Puede enviar SIP, correlated RTCP, RTCPXR, DNS y

Logs a HOMER. Esto supone un tráfico de paquetes, transportados mediante UDP hacia el equipo en el que se instala el servidor HOMER de análisis, cuya IP se debe especificar en los archivos de configuración del propio programa Heplify.

Grafana

Grafana es una plataforma interactiva Open Source de visualización de datos desarrollada por Grafana Labs. Permite a los usuarios ver datos de cualquier tipo a través de tablas y gráficos que se unifican en un panel de control (o en varios) para facilitar la interpretación y la comprensión.

Los paneles de control de Grafana otorgan un nuevo significado a los datos recolectados desde varias fuentes, ya que se pueden compartir con otros equipos o miembros, lo que permite que colaboren y realicen análisis más amplios de los datos y sus implicaciones. Es posible diseñar paneles de control específicos para el usuario y su equipo y personalizarlos para crear las visualizaciones que se deseen mediante las funciones de consulta y transformación avanzadas.

Figura 4. Interfaz web de Grafana

2.2. Sistema actual y planteamiento del problema

Como vemos en la *Figura 5*, en esencia, las llamadas VoIP vienen desde un cliente (origen del tráfico) por una conexión o "trunk" SIP de entrada hasta el equipo de interconexión encargado de proporcionar el control de llamada, procesamiento, y otros servicios sobre una red de conmutación de paquetes IP (Softswitch), que decide por qué proveedor conecta la llamada, a través de un trunk SIP de salida. La llamada, por así decirlo, involucra varios diálogos SIP, que utilizan los diferentes servidores de la plataforma. Tanto para el *trouble shooting* (detección de algún problema con una llamada o ruta determinada), como para que ciertas aplicaciones puedan analizar el tráfico con diversos propósitos (monitorización de la calidad, facturación, etc), es necesario tener la capacidad de capturar el tráfico, es decir, de realizar una captura de paquetes.

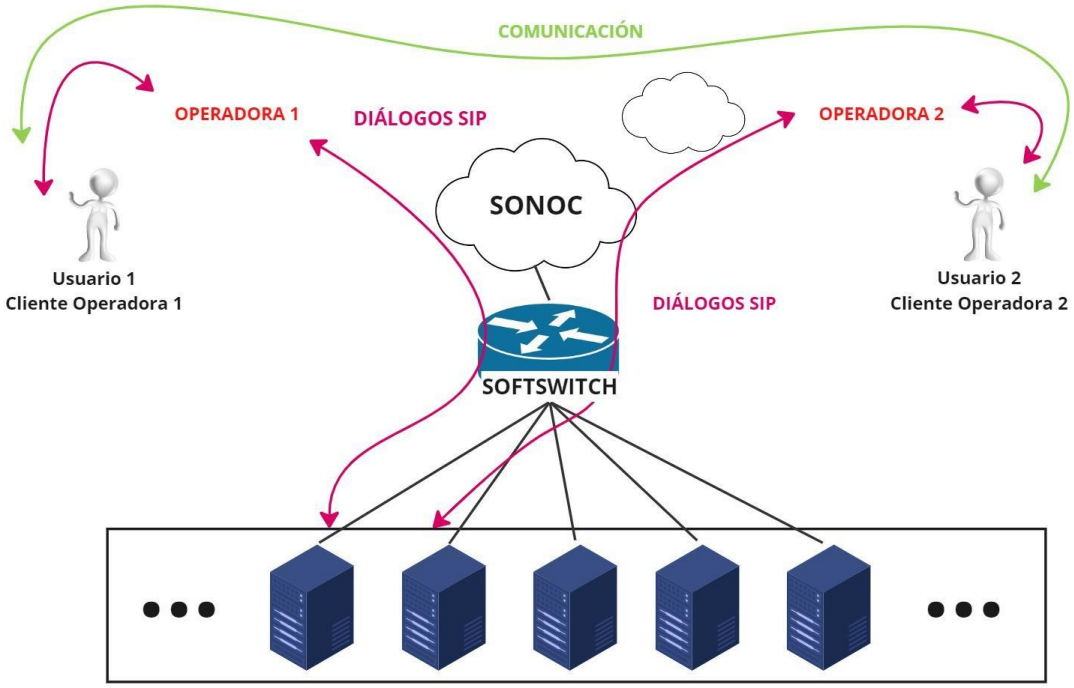


Figura 5. Diálogo llamadas VoIP

SONOC, que utiliza la tecnología de voz sobre IP (VoIP) y permite a los usuarios realizar llamadas a través de una conexión de internet de alta velocidad en lugar de una línea telefónica tradicional. Los clientes de SONOC (operadoras de cualquier país) se comunican con la compañía, para que les proporcione el soporte y conecte las llamadas con operadoras distintas o incluso de otros países. Es posible que SONOC no sea proveedor del destino de una comunicación, en cuyo caso pasará a ser cliente de otro proveedor, que sí pueda establecer comunicación con la operadora final, y poderse realizar la llamada o llamadas. Por tanto, funciona a la vez como proveedor de servicios a compañías que contactan con ellos (las llamamos **clientes**) y como cliente de otros proveedores telefónicos que consigan que se dé la comunicación con el destinatario (las llamamos **proveedores**).

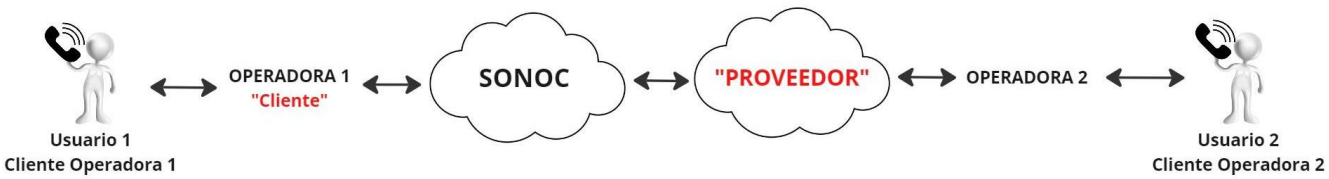


Figura 6. Comunicación cliente - proveedor

Es posible y se da el caso, de que determinadas comunicaciones no se puedan dar por diversos motivos (pérdida de información, fallos en el sistema de comunicación, incumplimiento de protocolos por parte de las compañías…). En estas situaciones, SONOC, que da soporte y soluciones ante problemas que puedan aparecer, entra en escena para detectarlos y solucionarlos.

Todo el tráfico generado por las miles de comunicaciones que pasan por SONOC para ser conectadas, se dirige hacia servidores propios de la empresa. Originalmente, toda la arquitectura del Softswitch se basaba en servidores físicos, y la captura de tráfico se hacía con *port mirroring* en los LAN switch que duplicaban el tráfico de las conexiones de los servidores hacia un equipo de monitoreo. Actualmente, SONOC ha migrado toda su arquitectura de servidores físicos a servidores virtuales, por las ventajas comentadas, (principalmente, el ahorro de recursos y la eficiencia), usando el entorno de virtualización Proxmox. Estos servidores soportan toda esta carga y en ellos se crean numerosas máquinas virtuales. En ellas, por ejemplo, se instalan servidores Proxy, o aplicaciones para realizar actividades de monitoreo, análisis, facturación… Por tanto, parece natural migrar también los equipos de monitoreo a servidores virtuales.

El proceso que sigue el tráfico de señalización SIP a través de la red de la empresa se explica detalladamente en el ANEXO B.

Como se ha comentado, es imprescindible poder encontrar qué está fallando en el sistema, en caso de que las comunicaciones tengan problemas. Cuando se detectan problemas con las comunicaciones de algún servidor, existen diferentes puntos para capturar la información del tráfico en el sistema para detectarlo, lo que se muestra en la *Figura 7*:

- **(1) Router de backbone.** Solución utilizada hoy en día. Son los router superiores de interconexión, donde estamos viendo todo el tráfico que atraviesa la red, lo cual es incómodo por la cantidad inmensa de paquetes capturados y arriesgado ya que trabajamos con un equipo en producción con muchas máquinas o servidores conectados a él.
- **(2) LAN Switch** de interconexión con el host Proxmox, haciendo una sesión de monitoreo. Esta es otra solución utilizada por la empresa a día de hoy, ya que podemos buscar qué está fallando en cualquiera de los servidores. De todos modos, esta solución no es del todo eficaz ni práctica, ya que en estos equipos aparece de nuevo el tráfico mezclado y puede ser dificultosa la tarea de trouble shooting.

- **(3) Enlace LAN Switch-Servidor Proxmox.** Esta opción podría ser buena ya que estaríamos capturando sólo el tráfico de un servidor físico. Aún así, veríamos mezcladas las comunicaciones con todas las máquinas virtuales del propio servidor. No podríamos seleccionar qué máquinas queremos monitorizar. Además, en caso de preparar un *mirroring* en el *switch* físico, necesitaríamos preparar una máquina física para recibir este tráfico, lo que no es favorable en cuanto a gasto de recursos.
- **(4) Cada máquina virtual.** Solución poco práctica ya que necesitaríamos entrar a capturar en todas las máquinas por separado para intentar localizar el problema, ya que los *Linux Bridge* de los Proxmox no permiten realizar *mirroring* para agrupar el tráfico en una sóla máquina.

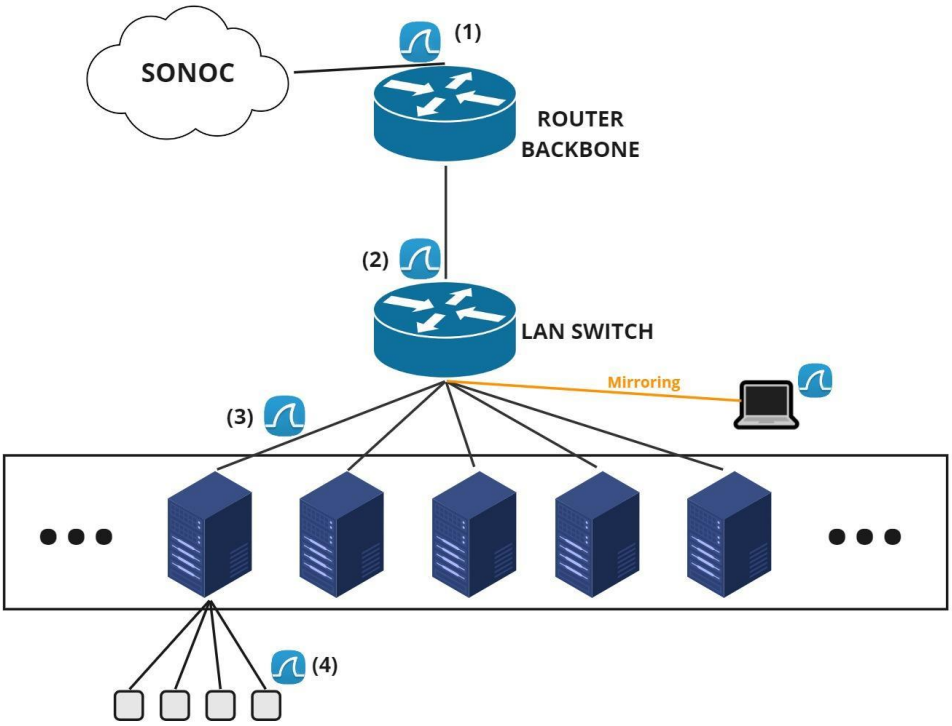


Figura 7. Posibles ubicaciones donde realizar el monitoreo.

En todos los casos, **el tráfico de interés viene mezclado** con el resto de tráfico, lo que plantea problemas de tratamiento (cantidad bruta de tráfico muy elevada) y de análisis (filtros de captura muy elaborados).

Analizando estas posibles soluciones, tal y como hemos comentado en la sección de *objetivos*, la empresa se vería ampliamente beneficiada en muchos aspectos si se consiguiera identificar y concentrar el tráfico deseado de los servidores para replicarlo en una única máquina que sólo realizara funciones de monitoreo y analizara sólo el tráfico de interés. Esta solución nos la puede proporcionar **Open vSwitch**, ya que nos da la posibilidad de realizar un *mirroring* que direcciona el tráfico deseado hacia la máquina de análisis. Así, ganaríamos en recursos utilizados y podríamos centralizarlo en una sóla máquina, que recibiría sólo el tráfico para analizar, a la hora de detectar fallos de comunicación. Esta disciplina, como se ha comentado anteriormente, es llamada **trouble shooting**, y es uno de los principales puntos hacia los que se enfoca el equipo de soporte de SONOC.

2.3. Solución final a implementar

La solución final elegida para conseguir los objetivos descritos es la de implementar la tecnología de Open vSwitch dentro de Proxmox. Como se ha comentado, una arquitectura de red basada en Open vSwitch nos va a ofrecer numerosas posibilidades que anteriormente no se podrían plantear, y que simplifican de manera importante las tareas de trouble shooting o similares a la hora de monitorear tráfico.

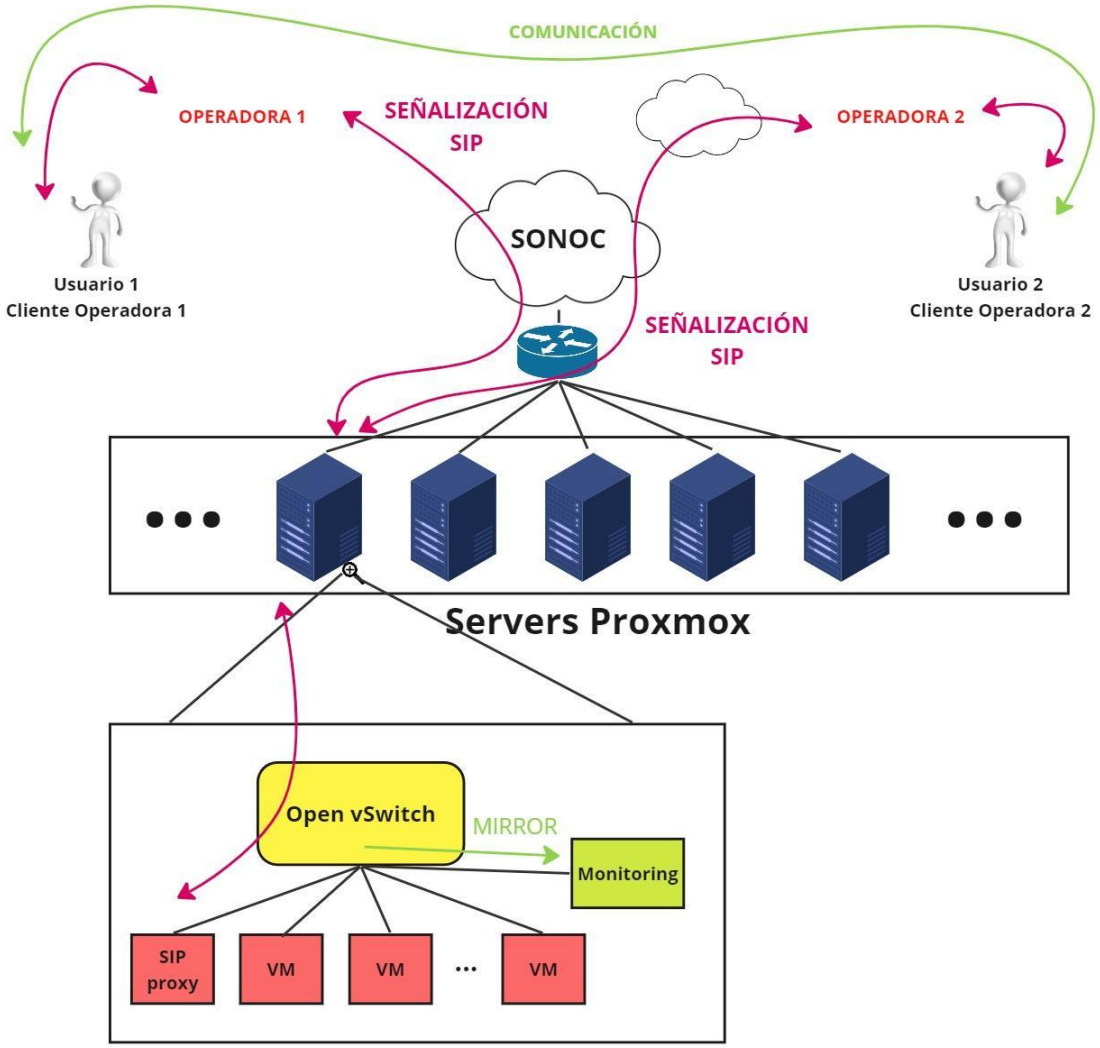


Figura 8. Funcionamiento del sistema con Open vSwitch

De la manera que se describe en la *Figura 8*, el sistema funciona con una máquina virtual adicional dentro de cada servidor Proxmox. Esta máquina recibirá el tráfico cursado por las máquinas virtuales del propio host Proxmox y se analizará mediante distintas aplicaciones. En el Open vSwitch se programa un *port mirroring*, que redirigirá el tráfico de las interfaces virtuales de interés hacia esta máquina adicional que sólo realizará funciones de monitoreo.

Mediante esta implementación, se va a mejorar el sistema de análisis de tráfico considerablemente, ya que contamos con las siguientes funcionalidades o ventajas respecto al sistema convencional utilizado, que son explicadas en detalle en el ANEXO F.2:

- Ahorro de recursos.
- Simplificación del trabajo.
- Efectividad.
- Posibilidades adicionales (flujos, controlador…).

Capítulo 3: Exposición del sistema a desarrollar

3.1. Descripción

El contexto en el que se va a realizar este proyecto es un escenario de pruebas controlado. El objetivo de este es implantar la solución a evaluar (comentada en el apartado *2.3. Solución final a implementar*) en un sistema de pruebas que simule el funcionamiento de los servidores en producción de la empresa SONOC.

De esta forma, podremos evaluar el comportamiento de la nueva tecnología que se estudia (Open vSwitch) ante situaciones realistas que se darían en caso de una implantación final en el sistema de producción. Seremos capaces de extraer estadísticas, datos y conclusiones respecto al uso de esta solución a partir de este escenario.

Para implementar este entorno controlado, se opta por utilizar dos servidores interconectados mediante un LAN Switch, en los que se instalará la plataforma de virtualización Proxmox, y los equipos o máquinas virtuales necesarias dentro de ellos.

El despliegue y configuración necesario para poner en funcionamiento este sistema de pruebas se explica paso a paso en los siguientes subapartados.

3.2. Entorno de trabajo y escenario en Proxmox

El entorno de pruebas en el que se realiza el estudio se describe en la *Figura 9*, y su configuración puede consultarse más extensamente en el ANEXO C.

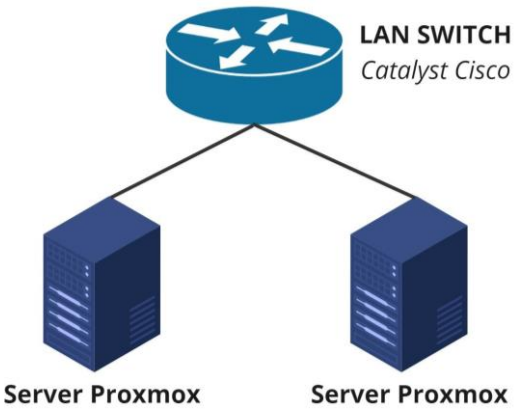


Figura 9. Entorno de trabajo

Centrándonos en la infraestructura, vamos a instalar dos servidores Proxmox (que identificaremos como Servidor pve1 y Servidor pve2) y dentro de cada servidor, se ubican diversos equipos virtuales. En el Servidor pve2 va a dejarse funcionando un *Linux Bridge* convencional. En el Servidor pve1, configuraremos un *Open vSwitch* para probar su funcionamiento y evaluar sus funcionalidades, en especial, el *Port Mirroring*. El diseño y funcionamiento de este tipo de bridge se explicará más adelante.

Estos equipos o máquinas son las necesarias para generar un escenario controlado de pruebas en el que podamos generar, recibir y monitorear tráfico SIP. Por tanto necesitaremos un equipo (*test1*) que genere tráfico desde el Servidor pve1 hacia el exterior de este servidor, mediante aplicaciones de simulación (utilizaremos SIPp). En el Servidor pve2, existirá otra máquina (*test2*) que responda a este tráfico SIP mediante la misma aplicación, simulando así tráfico entrante al servidor real en producción (Servidor pve1) en el que podría ubicarse un Proxy SIP tratándolo. De esa forma podremos analizar este tráfico SIP, que sería el existente en el entorno real de la empresa.

En nuestro escenario la máquina *test1* "actuará" como Proxy SIP, ya que la comunicación es directa entre los dos equipos al no disponer de un Proxy instalado, pero simulará su funcionamiento al estar trabajando con toda la señalización SIP derivada de las comunicaciones (como haría un servidor Proxy en esta situación). Por tanto, el tráfico cursado por la máquina *test1* (o por N máquinas del servidor) será el que nos interesaría analizar en la máquina adicional de monitoreo (*monitoring*). Esta máquina se instalará en el Servidor pve1 y estará dedicada únicamente a recibir una réplica del tráfico susceptible de analizar dentro del propio servidor, y a trabajar con distintas aplicaciones que monitorizan lo recibido. No generará tráfico de ningún tipo. Además, ha sido preparada con dos interfaces que la conectan al OvS: La primera (*net0*) es la interfaz de red propia de la máquina, la que tiene una IP asignada. La segunda interfaz (*net1*) se crea únicamente para recibir el tráfico procedente del *mirroring*, evitando así sobrecargar este enlace con otro tráfico proveniente de la red.

De esta forma, podemos simular lo que serían muchas máquinas virtuales recibiendo tráfico en el mismo servidor de trabajo. Así se podrá analizar el funcionamiento de OvS respecto a este tráfico SIP cursado por equipos de la empresa como pueden ser Proxy SIP a la hora de redireccionar este tráfico hacia otra interfaz del propio OvS donde se analice.

El funcionamiento, en resumen, será:

- **VM *test1* en pve1**
- **VM *monitoring* en pve1**
- **VM *test2* en pve2**

test1 y *test2* intercambiarán tráfico SIP, y *monitoring* debe ser capaz de verlo.

El escenario objetivo es el que podemos observar en el esquema de la *Figura 10.*

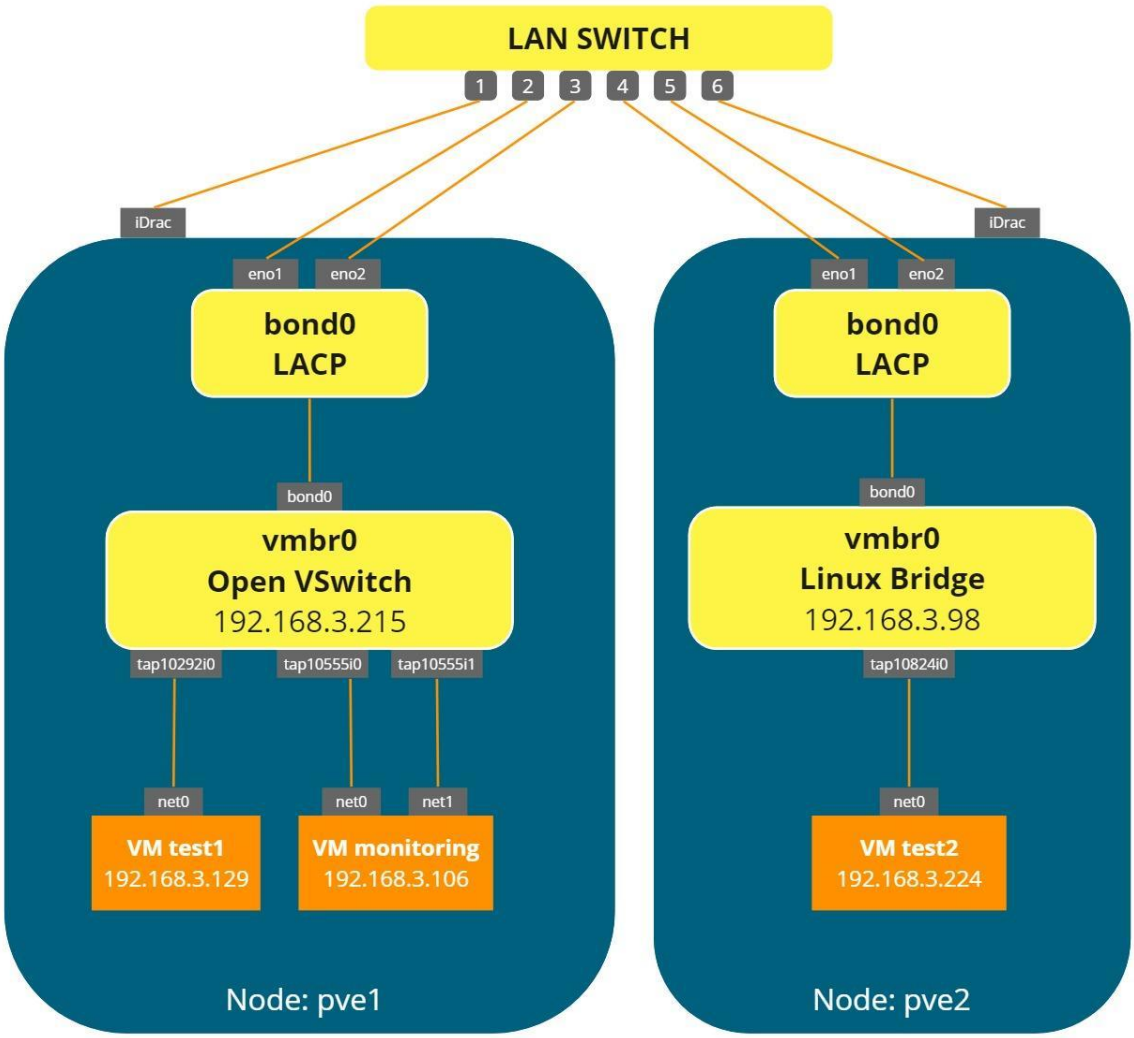


Figura 10: Escenario virtual en Proxmox.

Con las máquinas ya conectadas y preparadas para funcionar seremos capaces de conectarnos mediante conexión *ssh* a ellas y proceder con las pruebas que se preparen.

3.3. Sistema completo

Una vez comprendido el funcionamiento de los equipos virtuales y físicos, somos capaces de desarrollar el sistema completo (tanto física como virtualmente) sobre el que realizar las pruebas. Uniendo las dos partes que han sido explicadas, el esquema completo sería el siguiente:

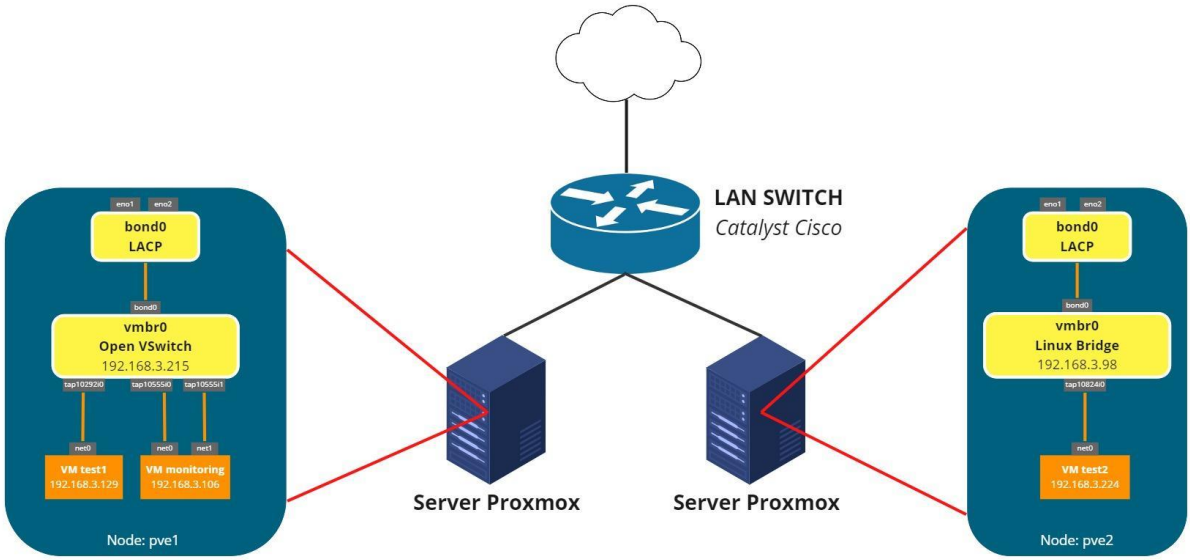


Figura 11. Esquema del sistema completo

Para que se pudiera llevar a cabo este proyecto, era necesario conseguir utilizar dos servidores que fueran compatibles con la versión de *Proxmox (Proxmox VE 7.4-3)* que utiliza la empresa en la actualidad. Finalmente se opta por utilizar unos servidores propios de la empresa y que no están en producción, en la sede de Madrid, en los que se instalará el entorno de virtualización. Por tanto se debe comprobar la correcta configuración de red de estos servidores.

Una vez los servidores tienen conectividad completa a la red de la empresa, se procede con el proceso de instalación, disponible en manuales que tiene preparados la propia empresa. A partir de ahí, seremos capaces de conectarnos y configurar el entorno virtual deseado para nuestras pruebas.

En los siguientes capítulos se pasa a describir con detalle el proceso de instalación y configuración que se ha seguido para la realización del proyecto.

Capítulo 4. Instalación y configuración del sistema de pruebas

4.1. Configuración y conectividad de los servidores de trabajo.

Para el objetivo que se busca con este proyecto, se instalan dos servidores con la plataforma de virtualización. El objetivo de realizarlo así es básicamente simular tráfico tanto entrante como saliente a la red externa. El primer servidor hará el papel de servidor en producción de la empresa, en el que se ubican servidores Proxy SIP, y el segundo simulará equipos ubicados en redes externas que direccionan tráfico hacia el primero. De esta manera estaremos simulando un tráfico real entrante del exterior, pero realmente estará fluyendo de un servidor a otro en nuestra plataforma de pruebas.

Al estar buscando realizar pruebas de tráfico entre los servidores, debemos realizar correctamente la configuración de red para que esto funcione. Así pues, para que la conectividad sea completa **servidor-servidor** y **servidor-exterior**, accedemos a ellos remotamente desde las oficinas de la empresa en Zaragoza a través del *switch* que interconecta estos servidores, según el siguiente esquema:

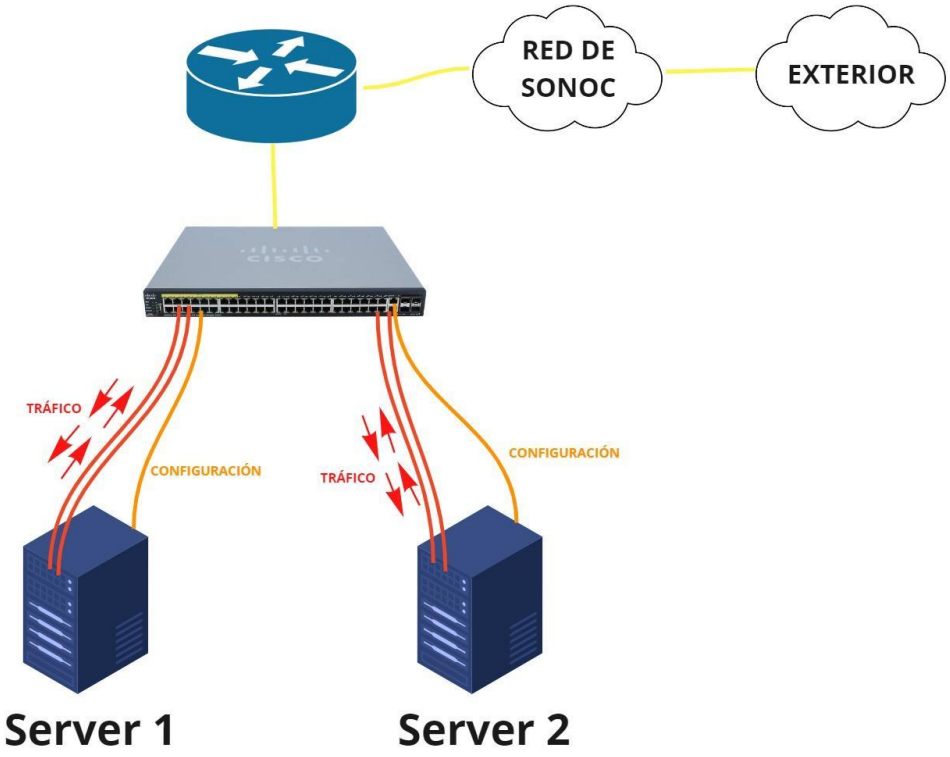


Figura 12. Conectividad y tráfico de los servidores

Centrándonos en cada uno de los servidores, debemos utilizar tres puertos de enlace. Dos funcionarán para comunicación, y el otro puerto es el conectado a la iDrac, una funcionalidad propia de los servidores Intel (puede consultarse más en profundidad en el ANEXO D), que

servirá para conectarse al servidor y configurarlo internamente. Las configuraciones que se explican en este apartado son análogas para el Servidor 1 y el Servidor 2.

Las primeras dos interfaces, conectadas al LAN Switch, sirven para comunicarse entre los servidores internos. La otra interfaz es la propia de la iDrac. La explicación extendida y configuración concreta de las interfaces se describe en el ANEXO D**.**

A través de la iDrac, se le configuran las rutas por defecto, tablas de encaminamiento, asignación dinámica/estática de direcciones… etc. En nuestro caso, se configuró la conectividad con el exterior, siendo accesible entonces desde la red de SONOC sin problema para realizar cualquier configuración del servidor.

4.1.1. Configuración de puertos mediante bond LACP

La conexión de los servidores que estamos utilizando con el LAN Switch propio de la empresa se ha realizado generando **"port bonding"** de manera que cada servidor se conecte con dos puertos de red al *switch*, pero estos se unifican. Es una técnica que combina múltiples conexiones de red en una sola interfaz virtual.

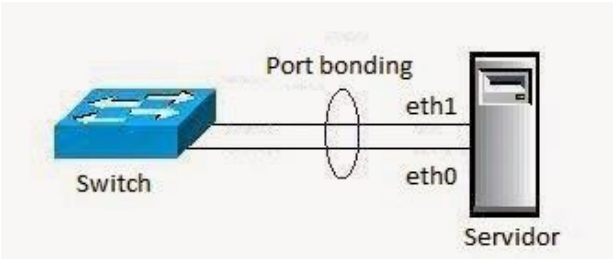


Figura 13. Esquema simple de port bonding.

La empresa realiza esta configuración en casi todos los servidores con los que se trabaja, por ello se ha optado por implementarlo también en nuestra plataforma de pruebas. Se puede consultar más información sobre la utilización de esta técnica en el ANEXO D.1.

4.2. Entorno de virtualización Proxmox

La configuración realizada en los dos servidores Proxmox, explicada en los siguientes apartados, tiene como objetivo construir el escenario representado en el apartado *3.2. Descripción del escenario en Proxmox.*

4.2.1. Funcionalidades

Las características que nos ofrece Proxmox son numerosas y nos permiten trabajar con comodidad y efectividad. Se nombran las principales y más representativas, cuya explicación completa se recomienda consultar en el ANEXO A.

- **Virtualización para la mayoría de Sistemas Operativos.**
- **Backup & Restore de "Máquinas Virtuales".**
- **Snapshot Live.** Permite hacer copias instantáneas de máquinas virtuales.

Figura 14. Esquema de copia de VMs

- **"Migración en caliente".** Movilizar máquinas virtuales sin apagarlas.
- **Administración centralizada.**
- **Puentes de red**.
- **Generación de clúster.**
- **Alta disponibilidad.**

4.2.2. Instalación en los servidores

Para la instalación de Proxmox en los servidores de trabajo, al estar ubicados en Madrid, se opta por realizar la instalación remota desde la iDrac, característica de los *Servers Intel* ya explicada en el apartado *4.1. Configuración y conectividad de los servidores de trabajo*.

Se genera un **CD virtual** con la imagen ISO (tipo de archivo que almacena una copia exacta de un sistema de ficheros de una unidad óptica) de Proxmox VE 7.2.

Durante el proceso de instalación se sigue un manual que SONOC tiene documentado para el desarrollo paso a paso. Es un proceso sencillo en el que se nos pedirá asignar una IP al propio Proxmox, crear un usuario y contraseña de administración, se puede configurar el gateway, servidor DNS… y alguna otra opción básica.

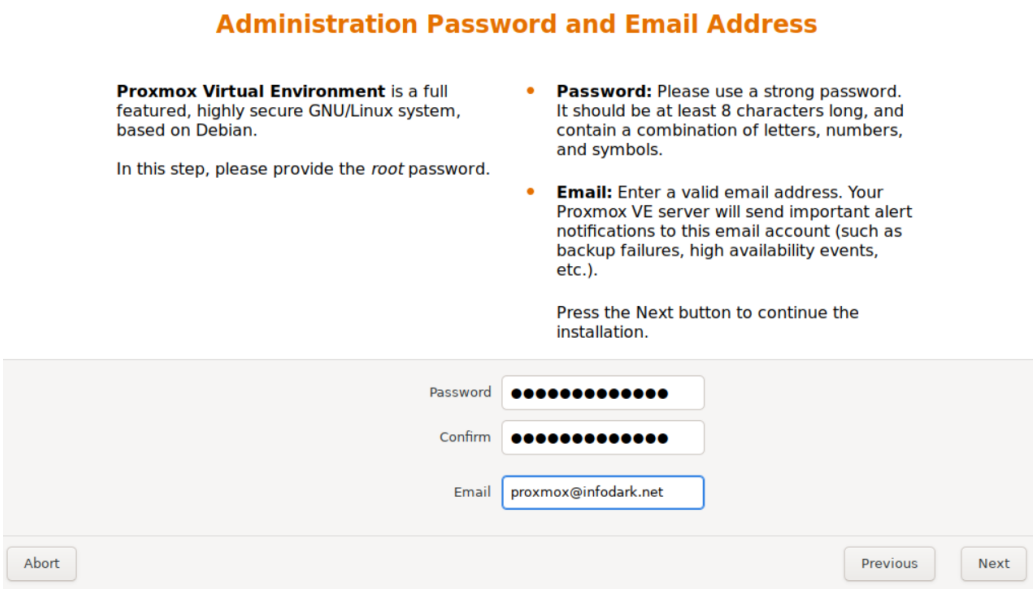


Figura 15. Proceso de instalación de Proxmox

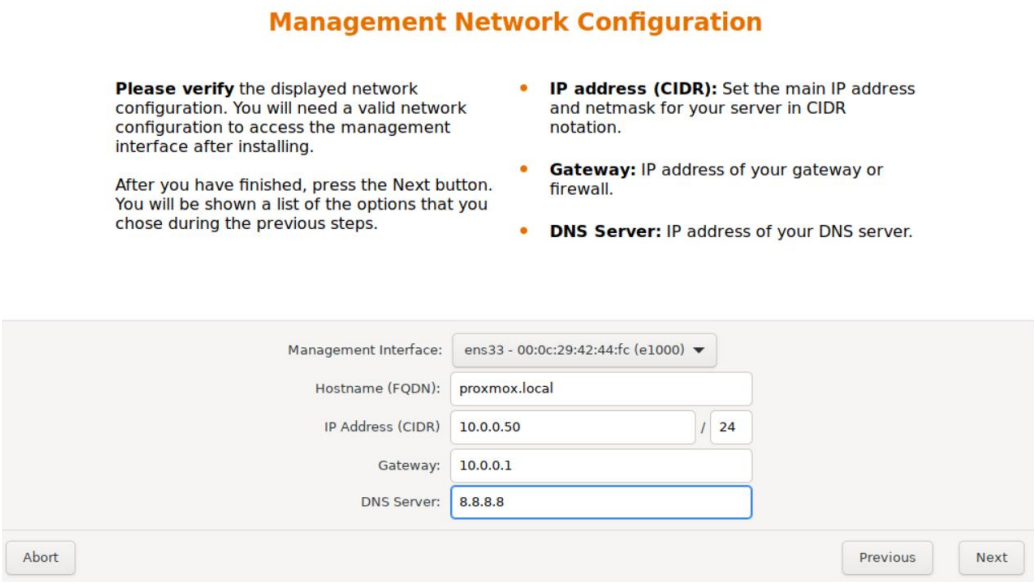


Figura 16. Proceso de instalación de Proxmox

Una vez se han escogido estos parámetros, se instalan y actualizan los paquetes necesarios para el funcionamiento completo. Seguidamente se procede a arrancar el sistema. Cuando arranca, aparece una ventana que indica la página web desde la que vamos a trabajar con Proxmox. Desde un ordenador IP remoto, podemos conectarnos desde un navegador a esta dirección web y acceder a la **interfaz web** de nuestro servidor Proxmox.

4.2.3. Administración desde interfaz web

El acceso a través del navegador web a una interfaz que permite la administración sencilla de los servidores Proxmox es una de las características fundamentales que ofrece este entorno de virtualización. Es intuitiva, y no se necesita instalar una herramienta de administración por separado. Todo puede ser modificado desde la propia web. Se utiliza una consola HTML5

integrada para acceder a la consola de usuario invitado. Como alternativa se puede utilizar *SPICE* (solución de acceso remoto a máquinas virtuales).

Como hemos definido un *cluster* con los dos servidores, podemos conectarnos a cualquiera de los dos nodos para configurar ambos. Cada nodo puede trabajar con los demás nodos, no hace falta dedicar un nodo específico para ello. La interfaz web es accedida mediante *https://youripaddress:8006*. El usuario por defecto es *root*, y la contraseña se habrá escogido en el proceso de instalación.

Las principales características a destacar de la interfaz son:

- Integración y gestión simple de clústeres.
- Acceso seguro a todas las Máquinas Virtuales y Contenedores mediante cifrado SSL (https).
- Interfaz gráfica rápida, capaz de manejar cientos o miles de Máquinas Virtuales.
- Consola HTML5 segura o SPICE.
- Gestión de permisos basada en roles.

Estas características y otras funciones de configuración que nos ofrece se pueden consultar de forma más extensa en el ANEXO E.1.

4.2.4. Instalación de las máquinas virtuales

Para la instalación de máquinas virtuales dentro de los servidores Proxmox, se sigue el proceso habitual de la empresa, ya que se tiene una metodología concreta para todas las máquinas con las que trabaja.

Esta creación de máquinas virtuales se realiza **desde la interfaz web**, ya que proporciona una herramienta sencilla de instalación. Aunque Proxmox también permite la instalación de contenedores LXC, no se hace uso de ello para el caso de nuestro proyecto.

El primer paso del proceso es cargar la imagen de disco (ISO) que queremos instalar en las máquinas. En nuestro caso se opta por instalar CentOS 7 en todas las máquinas virtuales, ya que su funcionamiento es óptimo para el proyecto y es el que más se conoce en la empresa.

Una vez tenemos la imagen en Proxmox, procedemos a crear la máquina mediante el botón *create VM* de la esquina superior derecha. Se nos abre una ventana con opciones a configurar, como *node:* nodo donde la queremos crear, *name:* nombre de la máquina (y hostname), *start at boot:* arrancar la máquina cuando Proxmox se inicie…

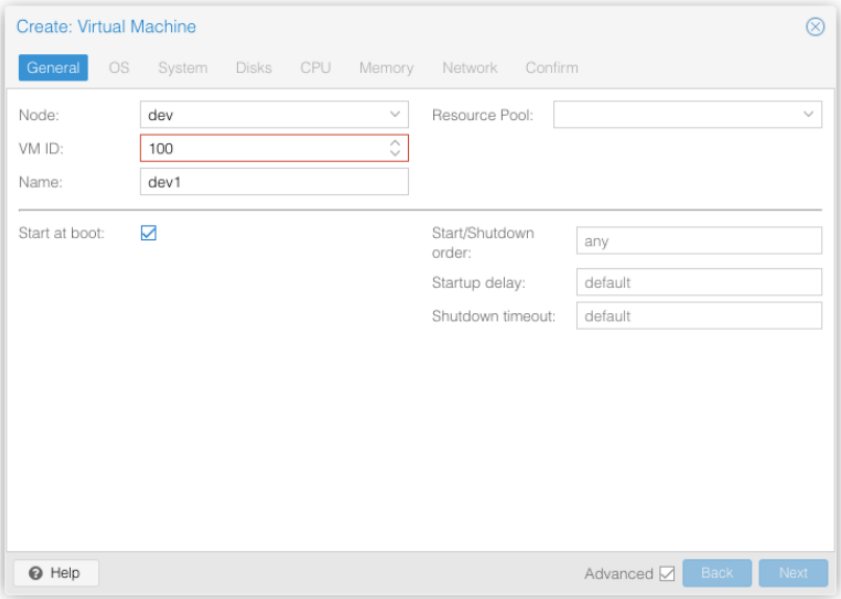


Figura 17. Ventana de generación de VMs

En la ventana *OS* indicaremos dónde está la imagen ISO que queremos utilizar y la seleccionamos. En *System* se puede seleccionar la BIOS a usar, controlador SCSI… La pestaña *Disks* permite crear el almacenamiento para la VM.

La configuración general que toman nuestras máquinas virtuales de trabajo se puede consultar desde la web, representado en la *Figura 18.* Para más detalle acerca de su configuración concreta en este proyecto se puede consultar el ANEXO E.2.

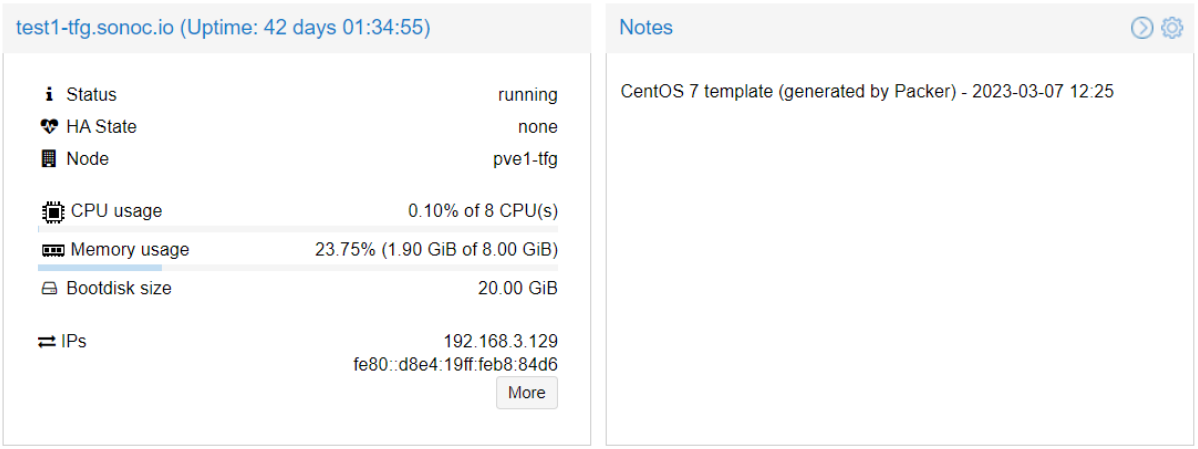


Figura 18. Ejemplo de información sobre la máquina test1.

4.3. Open vSwitch

4.3.1. Funcionalidades

Open vSwitch proporciona una alternativa flexible y escalable a los conmutadores Ethernet tradicionales, permitiendo crear y gestionar redes virtuales de manera eficiente. Es una solución versátil y potente para la virtualización de redes y la implementación de redes definidas por software.

Proporciona funcionalidades avanzadas de conmutación, segmentación de red y soporte para controladores SDN, lo que lo convierte en una herramienta muy útil en entornos de virtualización y cloud computing. Entre ellas, cabe destacar algunas que podrían ser de gran utilidad en el entorno de nuestro proyecto, cuya descripción completa se recomienda leer en el Anexo A:

- **Balanceo de carga**: **LACP, Bonding.** Distribución del tráfico de red.
- **OpenFlow - Controlador SDN.** Gestionado por controladores mediante el protocolo OpenFlow.
- **SPAN, RSPAN.** Permite hacer *port mirroring.*

4.3.2. Instalación en Proxmox

Para la instalación en Proxmox del software del Open vSwitch se procederá desde la consola del propio servidor Proxmox en el que se quiere establecer el conmutador OvS.

Por tanto, en nuestro Servidor 1 (*pve1*), debemos actualizar los paquetes ya instalados y posteriormente instalar los paquetes de Open vSwitch ejecutando:

```
$apt update
$apt install openvswitch-switch
```

4.3.3. Configuración del *OvS*

Los componentes fundamentales de la arquitectura de Open vSwitch son:

- **OVS kernel module.** Implementan datapaths que tienen asociada una tabla de flujos que definen el comportamiento de los paquetes.
- **ovs-vswitchd:** Componente central que se ejecuta en espacio de usuario. Procesa los mensajes OpenFlow.
- **ovsdb-server:** La configuración del switch es persistente y se guarda en la base de datos ovsdb

La gestión y configuración específica mediante comandos de nuestro Open vSwitch de trabajo se especifica en el ANEXO F, donde se pueden consultar, por ejemplo, todas las interfaces propias del OvS:

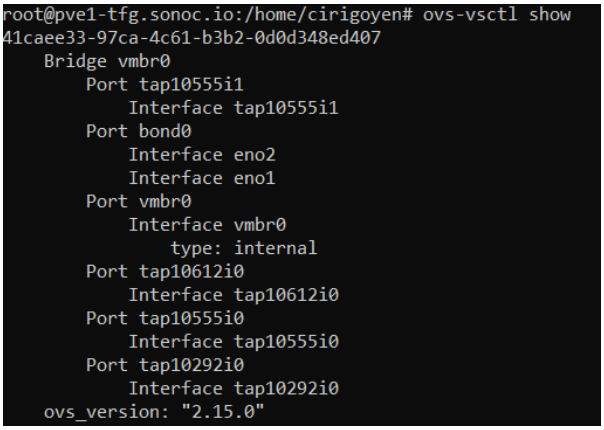


Figura 19. Descripción del contenido del OvS.

4.3.4. Configuración del *Port Mirroring*

Para configurar el *Port Mirroring* del Open vSwitch instalado, se definirán por consola varios comandos que especifiquen de qué interfaz/interfaces nos interesa recibir el tráfico replicado en la máquina de monitoreo, y cuál es el puerto destino en el que se encuentra esta máquina (interfaz por la que se sacará este tráfico replicado).

La estructura de este comando para configurar el *Port Mirroring* de un OvS es la siguiente:

```
$ovs-vsctl -- set bridge vmbr0 mirrors=@m -- \
--id=@port1 get Port eth1 -- \
--id=@port2 get Port eth2 -- \
--id=@m create mirror name=mirror1 \
select-dst-port=@port1 \
select-src-port=@port1 \
output-port=@port2 \
-- set bridge vmbr0 mirrors=@m
```

En este código, el tráfico entrante (`select-dst-port`) y saliente (`select-src-port`) de eth1 (`port1`) se redirigirá hacia el puerto destino del *mirror* (`output-port`), que es eth2 (`port2`). La última línea activa este *mirror*, llamado `mirror1`.

Para las pruebas de funcionamiento del sistema (Capítulo 5 de la memoria), se desarrolla más de un *mirroring.* Los códigos específicos desarrollados para la configuración del *Port Mirroring* para nuestro proyecto se pueden consultar en el ANEXO G.

4.4. Aplicación de simulación: SIPp

4.4.1. Instalación en las máquinas

La aplicación de simulación SIPp, ya explicada de forma general en el apartado *2.1. Protocolos y herramientas utilizadas*, está disponible para *Linux* y *Cygwin*.

En nuestro caso, estamos utilizando *Linux*. En *Linux*, SIPp se proporciona en forma de código fuente, por lo que deberemos compilarlo para usarlo. La aplicación ofrece cuatro opciones para compilar. Para la funcionalidad de nuestro sistema, elegimos la opción con compatibilidad con TLS (Seguridad de capa de transporte). Además, se puede instalar sin ella, con soporte *PCAP Play* (RTP Streams) o con soporte *SCTP* (Stream Transmission Control Protocol).

El código concreto que se ha utilizado para instalar la aplicación en nuestras máquinas virtuales se puede consultar en el ANEXO H.1.

Este proceso de instalación se repetirá tantas veces como usuarios necesitemos para simular comunicaciones. En nuestro caso, instalaremos SIPp en las máquinas *test1* (en el servidor pve1) y *test2* (en el servidor pve2), que son las que van a intercambiar tráfico SIP.

4.4.2. Funcionamiento y creación de escenarios

Como se ha explicado brevemente en el apartado *2.1. Protocolos y herramientas utilizadas*, en la sección dedicada a SIPp, esta aplicación nos va a permitir simular señalización de comunicaciones SIP entre equipos en los que esté instalada. Se pueden generar una o muchas llamadas SIP a un equipo remoto. La herramienta se inicia desde la línea de comandos.

Existen escenarios básicos incorporados con el ejecutable de SIPp, aunque también se pueden generar personalizados. Estos escenarios se ejecutan en las máquinas, y conectándose con otra en la que también esté corriendo SIPp, y comienza el intercambio de señalización SIP entre ellas.
Los principales escenarios que realizan comunicaciones sencillas y sus esquemas de funcionamiento se detallan en el ANEXO H.2:

La ejecución de estos escenarios predefinidos se realiza desde línea de comandos, por ejemplo:

```
# ./sipp -sn uas
# ./sipp -sn uac 127.0.0.1
*Siendo 127.0.0.1 la IP de la máquina (uas) con la que nos conectamos*
```

Podemos controlar SIPp de forma interactiva mediante el teclado o con un socket UDP. SIPp es compatible con las teclas "acceso directo" que se pueden ingresar en cualquier momento y también con un modo de comando simple. Desarrollado en el ANEXO H.2.

En el ANEXO H.2 también se especifican en detalle con ejemplos los comandos que se pueden modificar en cuanto al control del tráfico generado. SIPp genera tráfico SIP según el escenario

de funcionamiento escogido, pero se puede controlar el número de llamadas que se inician por segundo directamente al ejecutarlos mediante parámetros de inicio ("-r" tasa de llamadas, y "-rp" periodo de tasa).

SIPp nos ofrece la posibilidad de construir escenarios personalizados, con una amplia gama de opciones a modificar que dan gran libertad. Los ficheros de escenarios de SIPp están escritos en *xml*. Dentro del código, se configuran los parámetros de los mensajes SIP a enviar o recibir, en orden. La explicación concreta de estos parámetros se puede consultar en el ANEXO H.3.

4.5. Aplicaciones de monitoreo: Heplify, Homer

4.5.1. Instalación en la máquina de monitoreo

Partiendo de lo comentado en el apartado *2.1. Protocolos y herramientas utilizadas*, Heplify es un ejecutable que se instala en las máquinas virtuales y que envía los paquetes recibidos por la propia máquina a un servidor en el que se puedan analizar (HOMER).

En nuestro caso, la máquina de *monitoring* está preparada para recibir todo (o algo de) el tráfico SIP cursado dentro del servidor en el que se instala. Por ello, debemos instalar Heplify únicamente en esta máquina, ya que recibe el tráfico susceptible de analizar y por tanto sobre el que realizaremos el monitoreo desde HOMER.

Gracias al uso de Open vSwitch, sólo tenemos que instalar la aplicación en una máquina. Sin embargo, con el funcionamiento habitual de la empresa, deberíamos instalarla para cada una de las máquinas que quisiéramos monitorear (por ejemplo, en cada Proxy SIP que trabaje con señalización que queremos analizar, para que el programa la envíe a HOMER), lo que conlleva una carga y trabajo extra para estas máquinas que nos podemos ahorrar con OvS.

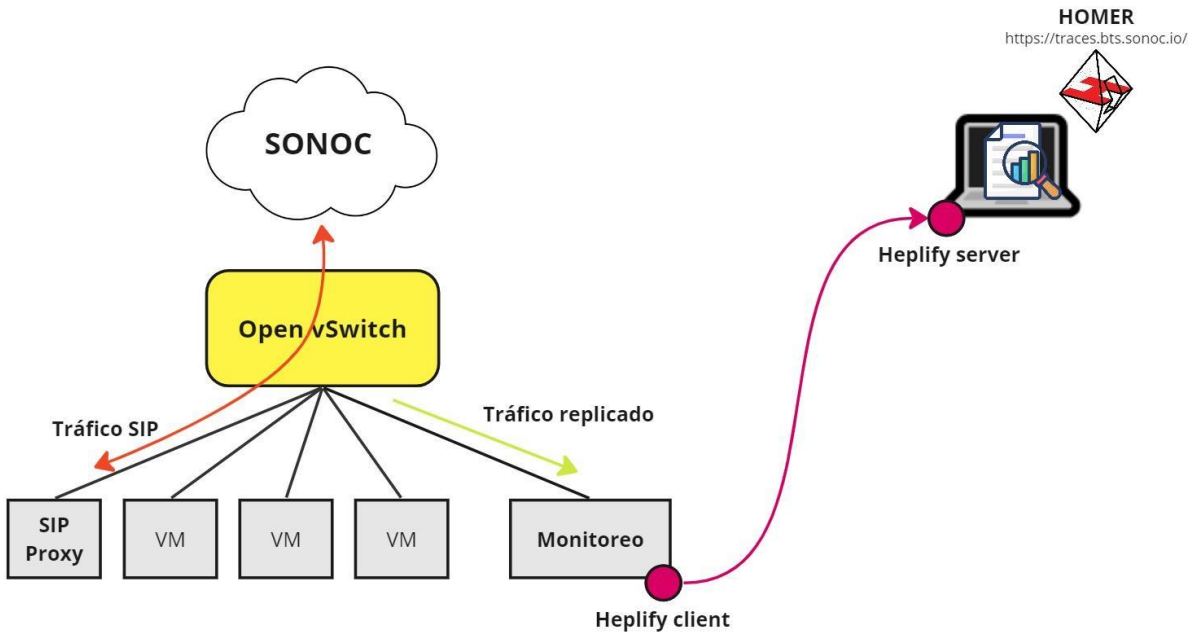


Figura 20. Esquema de funcionamiento Heplify - HOMER

Heplify se puede instalar como "**heplify client**" y como "**heplify server**". Para nuestra plataforma sólo vamos a instalar el *client* en la máquina de monitoreo, ya que la empresa facilita una máquina que tiene instalado el *server* y donde se encuentra la base de datos del HOMER. Por tanto, nuestro *heplify client* se conectará a este *heplify server* y le enviará los paquetes. Allí, se analizarán mediante HOMER.

Heplify cuenta con un manual de instalación paso a paso que se detalla en el ANEXO I.1.

Por otro lado, HOMER no necesita ninguna instalación por nuestra parte, ya que su base de datos se encuentra en el equipo con el heplify server ya instalado. Nosotros, para acceder a él, basta con conectarnos desde un buscador web al servidor, llamado *https://traces.bts.sonoc.io/*

4.5.2. Funcionamiento

La comprensión del uso de HOMER es determinante ya que es una de las herramientas principales de análisis que utiliza SONOC, por lo que se instalaría en los servidores en producción que apliquen la solución de nuestro proyecto con OvS (en la máquina de monitoreo). El funcionamiento en detalle de la herramienta HOMER se puede consultar en el ANEXO I.2.

4.6. Aplicación de análisis: Grafana

Grafana es una aplicación de análisis de datos muy útil en lo que a análisis de tráfico en una red se refiere. No es necesaria ninguna instalación para el proyecto, ya que la empresa ya cuenta con este servicio instalado en el que se dan de alta todas las máquinas virtuales en funcionamiento de sus servidores. Por tanto, podemos acceder desde el servidor web de Grafana a nuestras máquinas, como se presenta en la *Figura 21:*

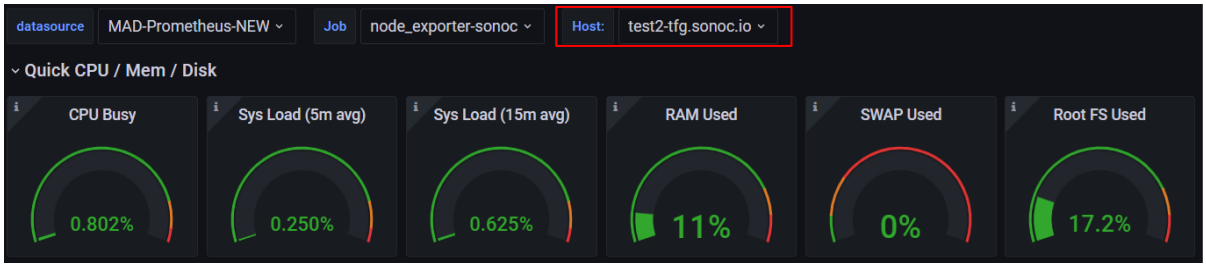


Figura 21. Características de nuestra máquina test2 desde Grafana

Desde esta interfaz, se puede acceder a cantidad de datos, métricas y gráficas acerca del tráfico que cursa la máquina que estamos analizando: Carga de la CPU, RAM utilizada, memoria, tráfico en la red cursado por cada una de sus interfaces, paquetes por segundo transmitidos, recibidos, errores de red… Estas y más funcionalidades se pueden consultar de forma extensa en el ANEXO J.

Figura 22. Ejemplo de métricas capturadas con Grafana

Capítulo 5: Pruebas de funcionamiento y resultados

5.1. Tráfico SIP cursado por una máquina del servidor

Inicialmente se realizan pruebas básicas de funcionamiento. Se va a generar tráfico SIP mediante la aplicación SIPp desde *test2* hacia test1, y se capturará el intercambio de señalización en la máquina de análisis (*monitoring*) para comprobar el correcto funcionamiento de la aplicación de simulación y del *mirroring* (el código necesario del *Port Mirroring* para esta situación se ha desarrollado en el ANEXO G.2). Con esto, estamos simulando el intercambio de mensajes SIP que resultaría de comunicaciones telefónicas que pasan por SONOC, en este caso por un **SIP Proxy** instalado en una máquina del propio servidor 1. Este tráfico sería el susceptible de analizar en la máquina preparada para ello (*monitoring*).

Por tanto, la situación a simular sería la del siguiente esquema:

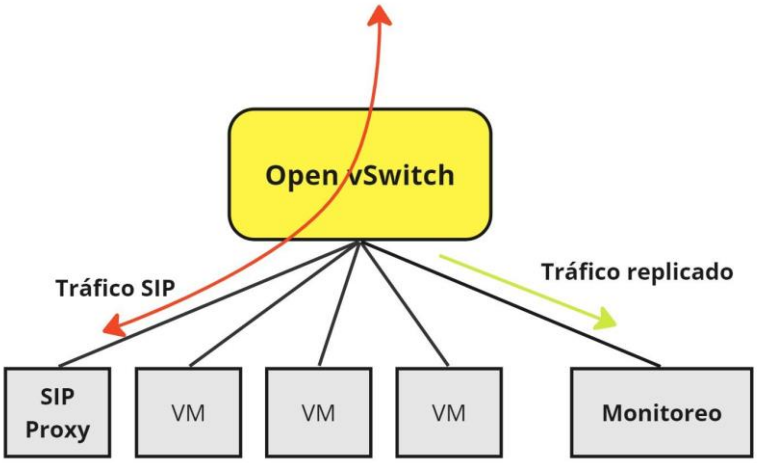


Figura 23. Primera situación simulada

Para los primeros análisis se van a ejecutar los archivos instalados por defecto con la aplicación SIPp: *uas.xml* y *uac.xml*. Estos escenarios, de "servidor" y "cliente" son los más básicos que pueden simularse. Su funcionamiento queda explicado en el ANEXO H.2.

El proceso que se sigue para que se pueda dar la comunicación es el siguiente:

- Ejecutar el escenario *uas.xml* en la máquina virtual *test2*, la cual actuará como receptor de las llamadas que se simulen. Ejecutamos para ello:

```
./sipp -sn uas
```

- Ejecutar el escenario *uac.xml* en la máquina virtual *test1*, situada en el otro servidor. Ejecutamos para ello:

```
./sipp -sn uac -r 1 -rp 5000 -m 4 192.168.3.224
```

Mediante esta prueba sencilla, se podrá confirmar que en la máquina *monitoring* se puede capturar sin problema el total del tráfico entrante y saliente de las máquinas del servidor 1 (en este caso el tráfico relativo a la máquina *test1*).

Las estadísticas mostradas por la aplicación tras la ejecución del escenario son las siguientes:

```
---------------------------- Scenario Screen -------- [1-9]: Change Screen --
Call-rate(length)   Port   Total-time  Total-calls  Remote-host
 1.0(0 ms)/5.000s   5060       20.00 s            4  192.168.3.224:5060(UDP)

Call limit reached (-m 4), 0.000 s period  0 ms scheduler resolution
0 calls (limit 1)                     Peak was 1 calls, after 5 s
0 Running, 6 Paused, 0 Woken up
0 dead call msg (discarded)           0 out-of-call msg (discarded)
1 open sockets

                                 Messages  Retrans   Timeout   Unexpected-Msg
     INVITE ---------->          4         0         0
        100 <----------          0         0         0         0
        180 <----------          4         0         0         0
        183 <----------          0         0         0         0
        200 <----------  E-RTD1 4         0         0         0
        ACK ---------->          4         0
      Pause [      0ms]          4                             0
        BYE ---------->          4         0         0
        200 <----------          4         0         0         0

------------------------------ Test Terminated --------------------------------
```

```
---------------------------- Statistics Screen ------- [1-9]: Change Screen --
  Start Time              | 2023-06-07   11:18:50.410366 1686136730.410366
  Last Reset Time         | 2023-06-07   11:19:10.418353 1686136750.418353
  Current Time            | 2023-06-07   11:19:10.418508 1686136750.418508
--------------------------+---------------------------+--------------------------
  Counter Name            | Periodic value            | Cumulative value
--------------------------+---------------------------+--------------------------
  Elapsed Time            | 00:00:00:000000           | 00:00:20:008000
  Call Rate               |    0.000 cps              |    0.200 cps
--------------------------+---------------------------+--------------------------
  Incoming call created   |       0                   |       0
  OutGoing call created   |       0                   |       4
  Total Call created      |                           |       4
  Current Call            |       0                   |
--------------------------+---------------------------+--------------------------
  Successful call         |       0                   |       4
  Failed call             |       0                   |       0
--------------------------+---------------------------+--------------------------
  Response Time 1         | 00:00:00:000000           | 00:00:00:001000
  Call Length             | 00:00:00:000000           | 00:00:00:003000
------------------------------ Test Terminated --------------------------------
```

Figura 24. Estadísticas mostradas por SIPp

Podemos comprobar que la comunicación SIP entre ambos equipos se da sin problema. Obtenemos un call-rate de de 0.2 llamadas por segundo. En la primera captura se puede observar que no se pierde ningún mensaje de señalización, observamos 4 transmitidos/recibidos por cada tipo de ellos.

Ahora, capturando en la máquina *monitoring* (en el puerto configurado como destino al hacer el *mirroring*), utilizando la herramienta ***tcpdump*** y escogiendo la interfaz de captura *eth1*, deberíamos ser capaces de ver toda esta comunicación entre las dos máquinas. Filtramos escogiendo lo recibido por el puerto 5060 (SIP). Se muestra lo obtenido por pantalla:

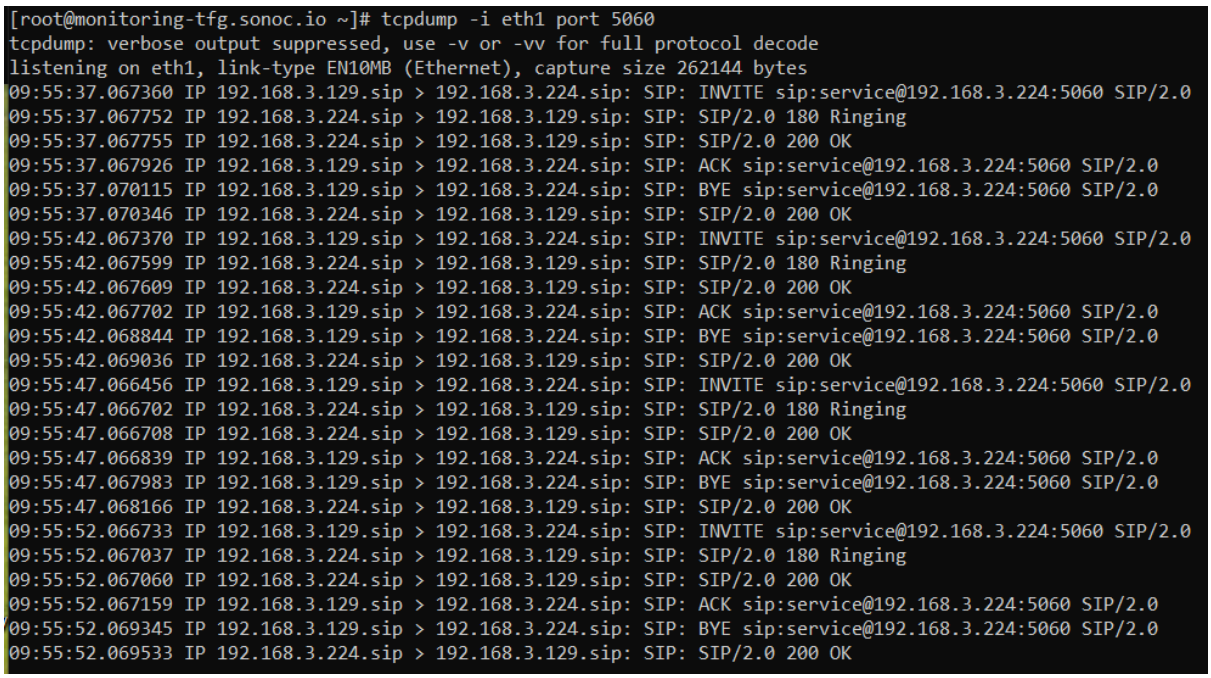


Figura 25. Captura en interfaz eth1 de la máquina monitoring.

Como podemos comprobar, toda la **señalización SIP** intercambiada entre *test1* y *test2* llega replicada por el interfaz destino del *mirroring* configurado: *eth1*.

Por tanto, estamos consiguiendo el objetivo principal: somos capaces de preparar aplicaciones de análisis en esta máquina y desde ella poder trabajar con señalización entrante a otras máquinas del propio servidor Proxmox.

Además, se puede comprobar el correcto funcionamiento de Open vSwitch comparando la cantidad de paquetes que se capturan en la interfaz de una de las máquinas de la comunicación con la cantidad recibida en *monitoring*. Esto se realiza capturando con *tcpdump* en una de las máquinas cliente/servidor. Si coincide, **no estamos perdiendo paquetes en el proceso de *mirroring*:**

Call-rate	Tiempo activo	Llamadas totales creadas	Llamadas exitosas	Paquetes capturados en cliente	Paquetes capturados en VM *monitoring*
0.2 cps	20s	4	4	24	24

Tabla de estadísticas de tráfico: Prueba 1

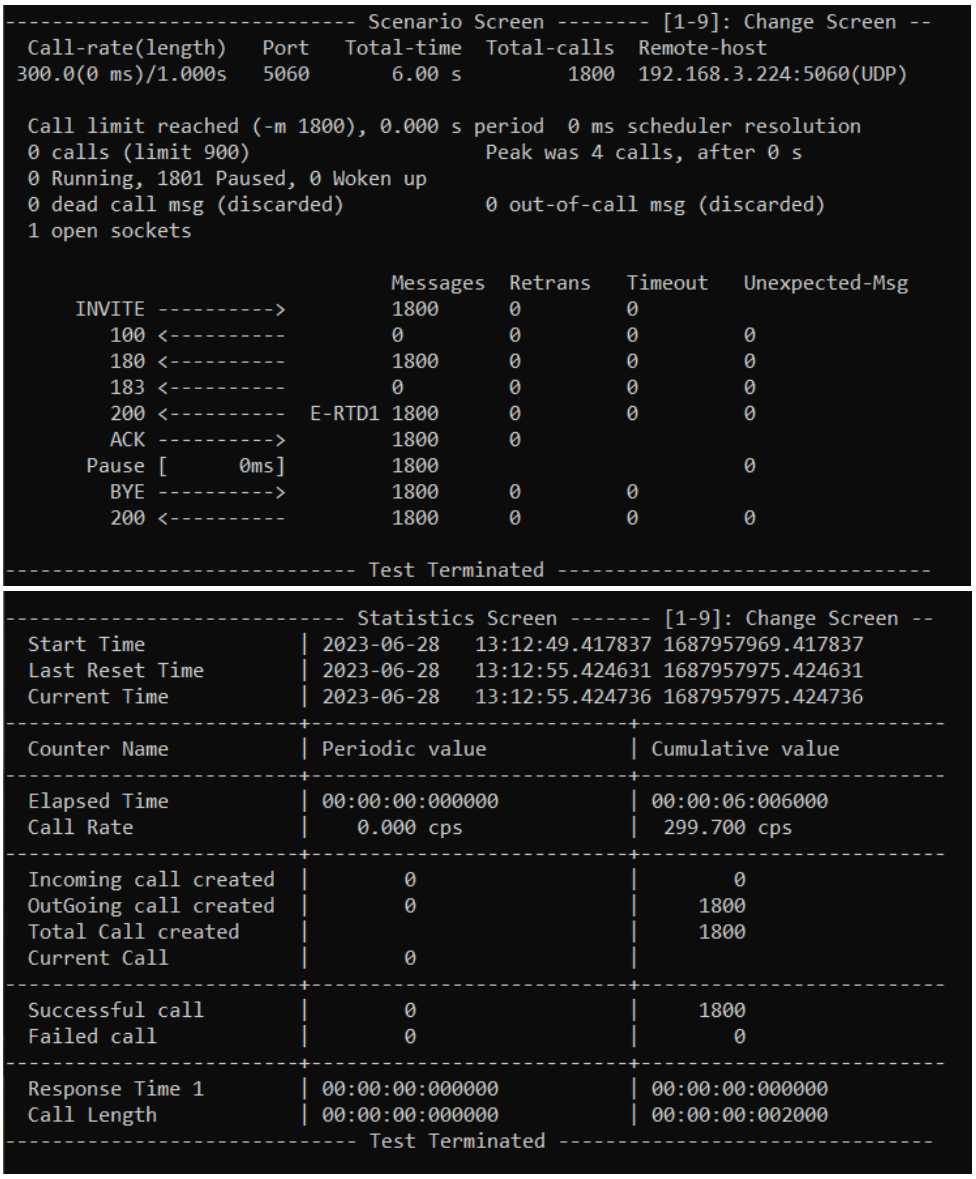
Con estos mismos escenarios básicos de cliente y servidor, podemos hacer pruebas con una carga de paquetes mayor (*call rate* alto). Así podemos comprobar que con una carga aproximada a lo que podría ser en la realidad, el sistema con Open vSwitch sigue funcionando sin sufrir pérdidas de cualquier tipo.

Ejecutamos de la misma forma que en la situación anterior (*uas* en *test2* y *uac* en *test1*), pero en este caso, modificamos la cantidad de llamadas emitidas. Vamos a simular un tráfico entrante de 300 llamadas por segundo durante 6 segundos de actividad, con una duración de llamadas de 0 segundos.

Para ello ejecutamos:

```
./sipp -sn uas
./sipp -sn uac -r 300 -rp 1000 -m 1800 192.168.3.224
```

Las estadísticas brindadas por la propia aplicación SIPp son las siguientes:



```
------------------------------ Scenario Screen -------- [1-9]: Change Screen --
 Call-rate(length)   Port   Total-time  Total-calls  Remote-host
300.0(0 ms)/1.000s   5060       6.00 s         1800  192.168.3.224:5060(UDP)

 Call limit reached (-m 1800), 0.000 s period  0 ms scheduler resolution
 0 calls (limit 900)                  Peak was 4 calls, after 0 s
 0 Running, 1801 Paused, 0 Woken up
 0 dead call msg (discarded)          0 out-of-call msg (discarded)
 1 open sockets

                           Messages  Retrans   Timeout   Unexpected-Msg
     INVITE ---------->         1800      0         0
        100 <----------         0         0         0         0
        180 <----------         1800      0         0         0
        183 <----------         0         0         0         0
        200 <----------  E-RTD1 1800      0         0         0
        ACK ---------->         1800      0
      Pause [      0ms]         1800                          0
        BYE ---------->         1800      0         0
        200 <----------         1800      0         0         0

------------------------------ Test Terminated --------------------------------
```

```
------------------------------ Statistics Screen ------- [1-9]: Change Screen --
 Start Time              | 2023-06-28   13:12:49.417837 1687957969.417837
 Last Reset Time         | 2023-06-28   13:12:55.424631 1687957975.424631
 Current Time            | 2023-06-28   13:12:55.424736 1687957975.424736
-------------------------+---------------------------+--------------------------
 Counter Name            | Periodic value            | Cumulative value
-------------------------+---------------------------+--------------------------
 Elapsed Time            | 00:00:00:000000           | 00:00:06:006000
 Call Rate               |    0.000 cps              |  299.700 cps
-------------------------+---------------------------+--------------------------
 Incoming call created   |        0                  |        0
 OutGoing call created   |        0                  |     1800
 Total Call created      |                           |     1800
 Current Call            |        0                  |
-------------------------+---------------------------+--------------------------
 Successful call         |        0                  |     1800
 Failed call             |        0                  |        0
-------------------------+---------------------------+--------------------------
 Response Time 1         | 00:00:00:000000           | 00:00:00:000000
 Call Length             | 00:00:00:000000           | 00:00:00:002000
------------------------------ Test Terminated --------------------------------
```

Figura 26. Estadísticas mostradas por SIPp

Observamos que la totalidad de llamadas se están realizando. Tenemos un total de 1800 mensajes de cada tipo de señalización SIP. Por tanto, se establecen 1800 comunicaciones a lo largo de 6 segundos. El *call rate* mostrado por el cliente de la comunicación es 299.7 llamadas por segundo (cps).

También se comprueba de igual manera que no perdemos paquetes en el proceso de *mirroring*:

Call-rate	Tiempo activo	Llamadas totales creadas	Llamadas exitosas	Paquetes capturados en cliente	Paquetes capturados en VM *monitoring*
300 cps	6s	1800	1800	29886	29886

Tabla de estadísticas de tráfico: Prueba 2

5.1.1. Análisis de métricas con Grafana

Esta segunda prueba se ha utilizado para obtener **métricas desde la aplicación Grafana**, con el objetivo de presentar una de las aplicaciones que utiliza habitualmente SONOC para analizar el tráfico en sus redes. Nos ofrece métricas acerca del tráfico cursado por las máquinas virtuales de los servidores. Esta se instalaría en nuestra máquina de análisis *monitoring* para obtener gráficas y estadísticas del tráfico que estamos analizando.

Se ha escogido esta prueba de carga para presentar resultados con Grafana, ya que las demás pruebas, realizadas únicamente para comprobar el correcto funcionamiento de OvS, no implican una gran carga en la red por lo que es difícil obtener gráficas de las que extraer información en claro.

Varios ejemplos de lo presentado por Grafana ante esta prueba se pueden consultar en el ANEXO J.2.

5.2. Tráfico SIP cursado por N máquinas del servidor

Para este segundo escenario de pruebas se va a simular la situación en la que más de una máquina virtual está trabajando con tráfico SIP que podría ser interesante para analizar, como podemos ver en la *Figura 27*. Para estas pruebas concretas, las máquinas recibiendo señalización serán dos, simulando N máquinas que funcionen así. En este caso, si queremos recibir en la máquina de monitoreo todo este tráfico, será necesario configurar el *Port Mirroring* de manera que redirija el cursado por dos de sus interfaces, no solamente por una. El código concreto para esta situación se desarrolla en el ANEXO G.3.

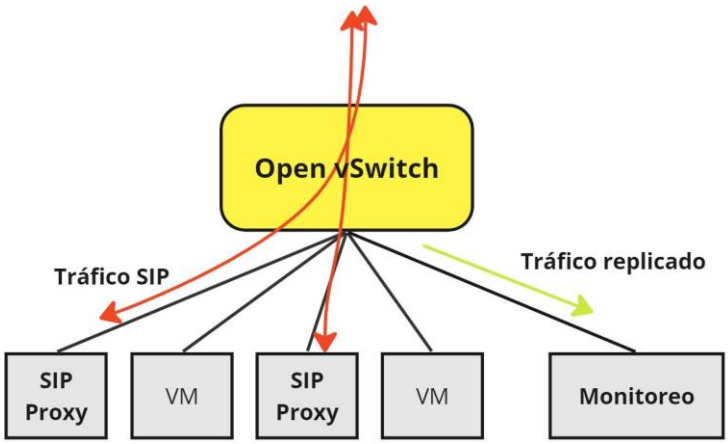


Figura 27. Segunda situación simulada

Para este escenario de pruebas, se ha instalado otra máquina virtual (*test3*) dentro del Servidor 1 (*pve1*) que realizará las mismas funciones que *test1*, cursando tráfico SIP generado mediante la aplicación *SIPp*. Así, conseguimos tener dos máquinas que hagan de "Servidor SIP Proxy" por el que pasa tráfico SIP dentro del mismo servidor de la máquina de monitoreo (*monitoring*):

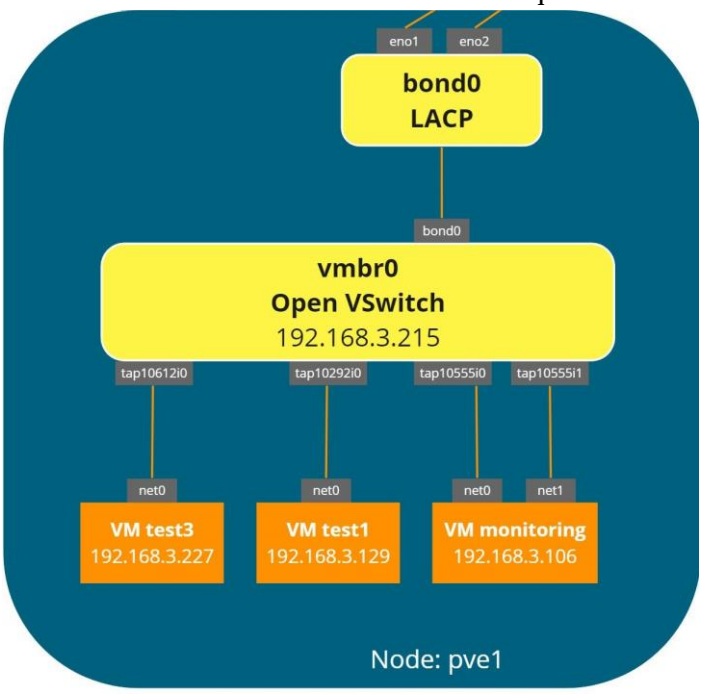


Figura 28. Nuevo escenario en Proxmox.

Trabajando con esta situación, nos planteamos otro escenario que podría ser posible dentro de un servidor Proxmox de la empresa, en el que se podrá elegir entre:

- Monitorizar la señalización SIP entrante a una única máquina virtual, ya que se ha podido detectar, por ejemplo, algún problema o error en una concreta. Se seleccionará su interfaz para el *Port Mirroring*.
- Monitorizar la señalización SIP entrante a más de una máquina, seleccionando las que pueden contener el tráfico interesante de analizar. Se hará *Port Mirroring* de varias interfaces concretas.
- Monitorizar toda la señalización SIP cursada por el servidor. En este caso se replicará el tráfico cursado por todas las interfaces.

De este modo, se demuestra la gran utilidad y comodidad en la utilización de Open vSwitch ya que simplemente modificando el *mirroring* que realiza, podemos analizar la máquina que nos interesa entre varias.

Vamos a preparar un escenario personalizado con los archivos de ejecución *uas_esc2.xml* y *uac_esc2.xml*, desarrollados en el ANEXO H.6.

Se ejecutará *uac_esc2.xml* en las máquinas *test1 y test3*, y *uas_esc2.xml* en la VM *test2*.

Obtenemos la siguiente captura en la VM *monitoring*, de los dos flujos de comunicación SIP entrando a dos máquinas distintas del Servidor 1.

```
[root@monitoring-tfg.sonoc.io ~]# tcpdump -i eth1 port 5060
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
10:28:54.267431 IP 192.168.3.129.sip > 192.168.3.224.sip: SIP: INVITE sip:service@192.168.3.224:5060 SIP/2.0
10:28:54.267888 IP 192.168.3.224.sip > 192.168.3.129.sip: SIP: SIP/2.0 180 Ringing
10:28:54.267890 IP 192.168.3.224.sip > 192.168.3.129.sip: SIP: SIP/2.0 200 OK
10:28:54.268040 IP 192.168.3.129.sip > 192.168.3.224.sip: SIP: ACK sip:service@192.168.3.224:5060 SIP/2.0
10:28:55.268442 IP 192.168.3.129.sip > 192.168.3.224.sip: SIP: BYE sip:service@192.168.3.224:5060 SIP/2.0
10:28:55.268666 IP 192.168.3.224.sip > 192.168.3.129.sip: SIP: SIP/2.0 200 OK
10:28:58.659271 IP 192.168.3.227.sip > 192.168.3.224.sip: SIP: INVITE sip:service@192.168.3.224:5060 SIP/2.0
10:28:58.659596 IP 192.168.3.224.sip > 192.168.3.227.sip: SIP: SIP/2.0 180 Ringing
10:28:58.659616 IP 192.168.3.224.sip > 192.168.3.227.sip: SIP: SIP/2.0 200 OK
10:28:58.659807 IP 192.168.3.227.sip > 192.168.3.224.sip: SIP: ACK sip:service@192.168.3.224:5060 SIP/2.0
10:29:03.661197 IP 192.168.3.227.sip > 192.168.3.224.sip: SIP: BYE sip:service@192.168.3.224:5060 SIP/2.0
10:29:03.661492 IP 192.168.3.224.sip > 192.168.3.227.sip: SIP: SIP/2.0 200 OK
```

Figura 29. Captura en la interfaz eth1 de la máquina monitoring

Observamos la señalización SIP:
test1(192.168.3.129) ←SIP→ test2(192.168.3.224).
test3(192.168.3.227) ←SIP→ test2(192.168.3.224)

5.2.1. Análisis de las comunicaciones con Heplify-Homer

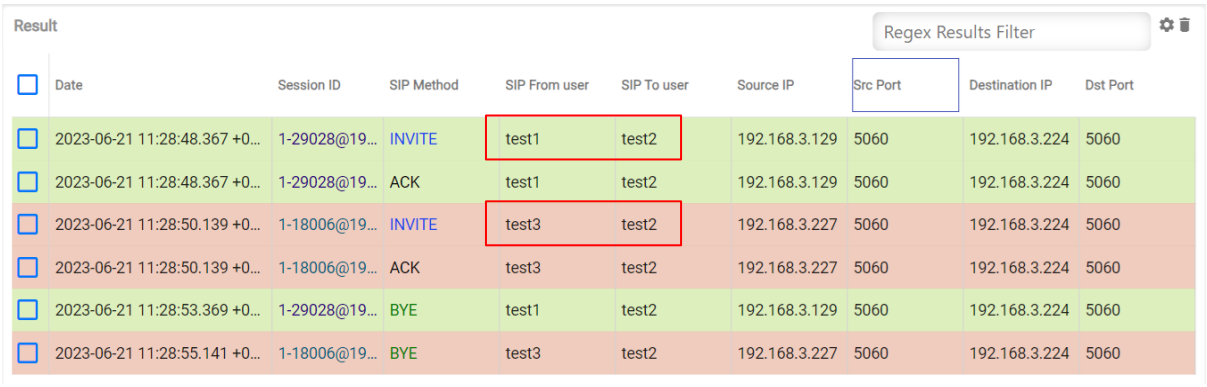
Para este segundo escenario de pruebas, se van a poner en marcha las aplicaciones de análisis ya explicadas en sus apartados concretos: **Heplify-Homer.**

Haciendo esto, nos ponemos en la situación real de funcionamiento, en la que una máquina recibe el tráfico SIP susceptible a analizar, y en ella se instalan diversas aplicaciones de

monitoreo, análisis, facturación… para trabajar con él. En nuestro caso vamos a utilizar las dos mencionadas, para obtener estadísticas del tráfico generado, gráficas, características de las llamadas…

Se extraen resultados realistas, que podrían ser los resultantes de aplicar este sistema al funcionamiento de la empresa.
Realizando en este escenario las pruebas descritas, podemos observar datos acerca de ambos flujos de comunicación (el de la máquina *test1* y el de la máquina *test3*).

En HOMER se ha recibido toda esta señalización SIP desde la máquina de *monitoring*, de manera que podremos analizarla y extraer datos de las llamadas a analizar. Simulando una llamada desde cada equipo, ambas de duración 5 segundos:



Result

Regex Results Filter

Date	Session ID	SIP Method	SIP From user	SIP To user	Source IP	Src Port	Destination IP	Dst Port
2023-06-21 11:28:48.367 +0...	1-29028@19...	INVITE	test1	test2	192.168.3.129	5060	192.168.3.224	5060
2023-06-21 11:28:48.367 +0...	1-29028@19...	ACK	test1	test2	192.168.3.129	5060	192.168.3.224	5060
2023-06-21 11:28:50.139 +0...	1-18006@19...	INVITE	test3	test2	192.168.3.227	5060	192.168.3.224	5060
2023-06-21 11:28:50.139 +0...	1-18006@19...	ACK	test3	test2	192.168.3.227	5060	192.168.3.224	5060
2023-06-21 11:28:53.369 +0...	1-29028@19...	BYE	test1	test2	192.168.3.129	5060	192.168.3.224	5060
2023-06-21 11:28:55.141 +0...	1-18006@19...	BYE	test3	test2	192.168.3.227	5060	192.168.3.224	5060

Figura 30. Mensajes SIP vistos desde HOMER.

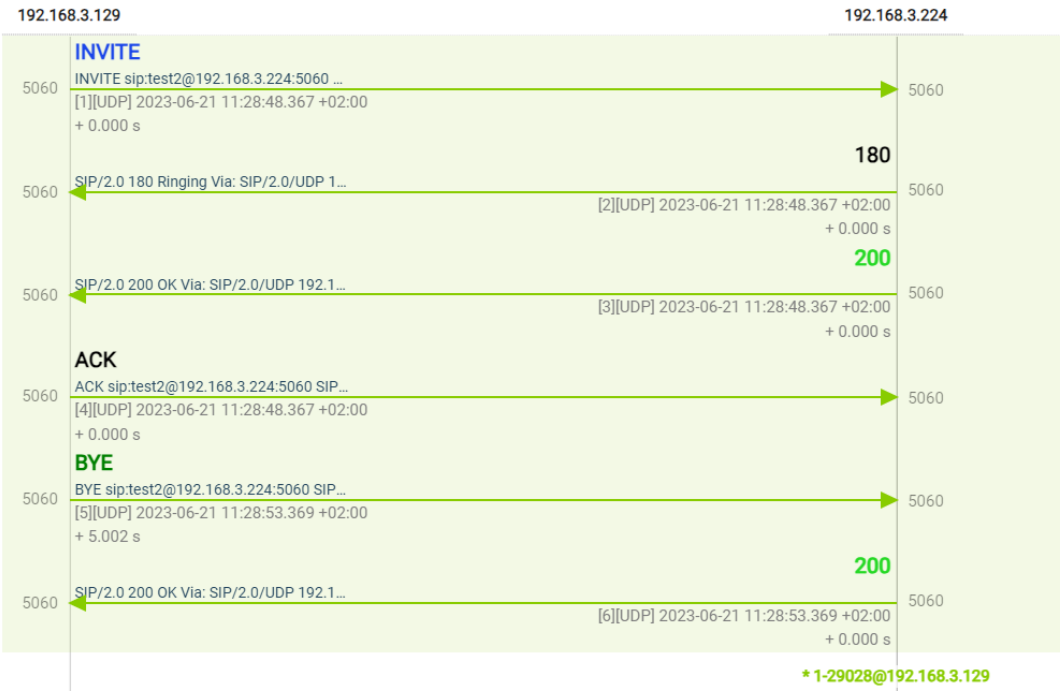


Figura 31. Flujo de mensajes entrante a test1.

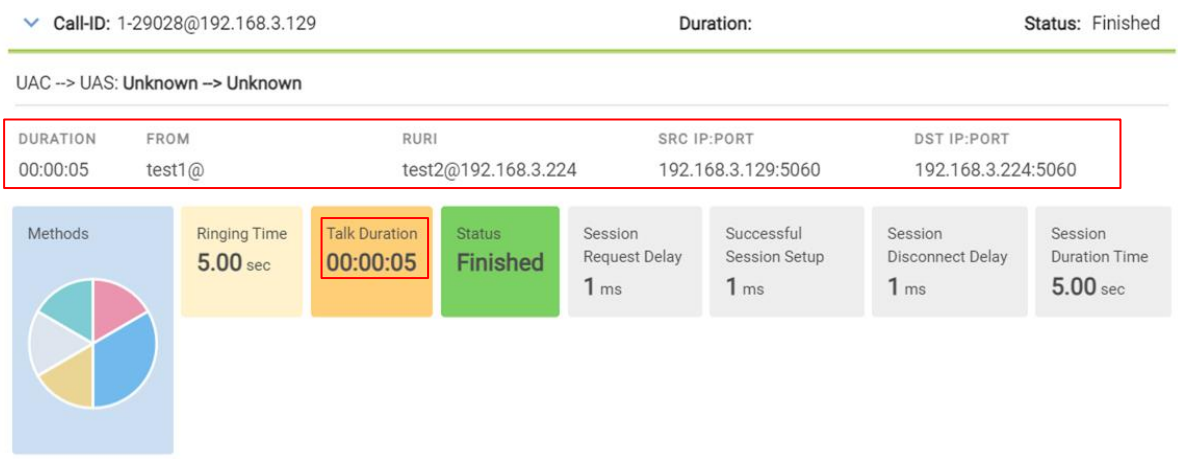


Figura 32. Información de sesión de la llamada entrante a test1.

Capítulo 6: Conclusiones y líneas futuras

6.1. Conclusiones

A lo largo de este Trabajo Fin de Grado, se ha desarrollado un sistema controlado de pruebas que simule el funcionamiento de los servidores en producción de la empresa SONOC y sea capaz de evaluar el funcionamiento de Open vSwitch en una red dedicada al análisis de tráfico telefónico, especialmente su funcionalidad de realizar *Port Mirroring* para concentrar el tráfico cursado por varias máquinas de un servidor en una sola máquina, que únicamente lo analiza.

Se ha desarrollado una plataforma constituida por un LAN Switch y dos servidores conectados. Tras configurar la conectividad de los equipos de trabajo, se ha instalado la plataforma de virtualización Proxmox en ambos servidores, con el objetivo de poder generar tantas máquinas virtuales en ellos como sean necesarias para realizar las pruebas pertinentes. De esta forma, se ha podido evaluar el comportamiento de la nueva tecnología que se estudia (Open vSwitch) ante situaciones realistas que se darían en caso de una implantación final en el sistema de producción.

El primer servidor de trabajo ha hecho las funciones de servidor real en producción de la empresa. En él, se ha instalado el Open vSwitch como bridge virtual para evaluar su funcionamiento. Además, a lo largo del proyecto, se han llegado a instalar tres máquinas virtuales en este Proxmox: dos máquinas que trabajan con señalización SIP, simulando servidores SIP Proxy en funcionamiento, y otra máquina que recibirá el tráfico que se desee analizar dentro del servidor en funcionamiento, y en la que se han instalado diversas aplicaciones de monitoreo o análisis de tráfico SIP.

En el segundo servidor, se ha dejado por defecto la configuración de *Linux Bridge* como *bridge* virtual. En él se ha instalado una única máquina que se ha utilizado para simular señalización SIP entrante del exterior a los servidores de la empresa (como sería el primero), en los que se encuentran los equipos que trabajan con este tráfico, y en los que, por ejemplo, se instalan Proxys SIP.

Se ha instalado la aplicación de simulación SIPp en las máquinas virtuales que intercambian señalización. Así, se han simulado comunicaciones SIP entre equipos virtuales. Este tráfico generado por la aplicación, es el que se ha redirigido a la máquina de análisis o monitoreo, mediante la configuración de un *Port Mirroring* de las interfaces deseadas hacia esta máquina.

En la máquina que recibe este tráfico SIP a analizar, se han instalado aplicaciones de monitoreo utilizadas en el día a día por SONOC para analizar el tráfico y, por ejemplo, detectar problemas con las comunicaciones que atraviesan su red. Se han presentado estadísticas de las llamadas simuladas entre nuestras máquinas virtuales mediante las aplicaciones Heplify y Homer. También se han obtenido gráficas descriptivas del tráfico existente en la red mediante la aplicación Grafana.

Todo ello, se ha realizado a modo de representar una situación real en la que nuestro servidor con OvS funcionando estuviera en producción y cursando tráfico telefónico real.

Por tanto, se ha cumplido con los objetivos iniciales, ya que se ha comprobado el correcto funcionamiento de la tecnología a estudiar: Open vSwitch, y de sus funcionalidades principales. El sistema de pruebas ha respondido bien a situaciones que podrían darse en el momento que la solución se implementase, por lo que los resultados obtenidos se han calificado como satisfactorios, considerando que mejorarían el funcionamiento del sistema actual.

6.2. Líneas futuras

Respecto a las líneas de trabajo futuras que servirían como posibles mejoras al trabajo realizado, se van a proponer algunos desarrollos o ampliaciones:

Inicialmente, la principal aplicación sería implementar la solución estudiada con OvS en servidores en producción que trabajen con gran cantidad de comunicaciones telefónicas. A partir de ahí, si se identifica el servidor en el que se quiere detectar algún problema, podríamos acceder a él directamente y modificando el *mirroring* configurado, recibiremos directamente el tráfico que nos interesa analizar, evitando así los filtrados adicionales en máquinas en producción que ralentizarían o entorpecerían sus funciones.

Además, podría considerarse la definición de flujos OpenFlow en el OvS, ya que esta configuración específica de flujos dinámicos tiene mayor potencial para el filtrado deseado que si funciona por defecto como *Learning Switch* y sólo establecemos un *mirroring* general. Nos permitiría una configuración mucho más flexible y adaptada al escenario concreto, que además no estuviera sujeta a imprecisiones en el comportamiento por defecto de los OvS, diferencias entre fabricantes… etc. Estos flujos podrían ser interesantes si queremos, por ejemplo, descartar paquetes de algún tipo para así simplificar aún más el análisis del tráfico que llega a la máquina de monitoreo.

Para hacer esto aún más efectivo, Open vSwitch nos brinda la posibilidad de utilizar un controlador que centralice la gestión de flujos de más de un OvS, de acuerdo a la configuración que nosotros le indiquemos. Básicamente, se podría hacer una configuración remota de los *switch* dejando el aprendizaje y la configuración de los flujos en manos del controlador, con la simplicidad extra y ahorro de trabajo que esto supondría al equipo de infraestructura.

Como se puede observar, la implementación de OvS en servidores virtualizados nos abre un abanico de posibilidades que hasta ahora el sistema de funcionamiento convencional no tenía, y nos otorga una gran simplificación del trabajo que sería interesante conseguir.

Capítulo 7: Anexos

ANEXO A. Protocolos y herramientas.

En este anexo se dan más detalles sobre las herramientas comentadas en el capítulo 2, además de describir algunas menos importantes.

Protocolo SIP

Se intercambian diferentes tipos de mensajes entre los participantes, entre los que cabe destacar los más comunes:

- **INVITE** (invitación): Se utiliza para iniciar una sesión. El remitente lo envía indicando la dirección del destinatario y las características de la sesión a establecer.
- **180 RINGING** (timbrado): Es enviado por el destinatario para indicar que la llamada está siendo procesada. El teléfono está sonando.
- **200 OK** (confirmación): Es enviado por el destinatario para indicar que ha aceptado la invitación. Expresa que la sesión ha sido establecida correctamente y contiene la información necesaria para que la comunicación entre remitente y destinatario se efectúe.
- **ACK** (confirmación de recepción): Lo envía el remitente para confirmar que ha recibido el mensaje 200 OK. Se utiliza para completar la fase de establecimiento.
- **BYE** (terminación): Se utiliza para finalizar la sesión de manera ordenada. Puede ser enviado tanto por el remitente como por el receptor de la llamada. Indica la intención de terminar la sesión.
- **CANCEL** (cancelación): Sirve para cancelar una invitación en curso. Si un usuario envía un mensaje INVITE y decide cancelar la llamada antes de que se establezca la sesión, se envía este mensaje.

Protocolo OpenFlow

OpenFlow es un protocolo de redes definidas por software (SDN) que separa en redes diferenciadas el control y el flujo de datos de los equipos integrantes de la red (como por ejemplo los conmutadores). Se utiliza para comunicar los equipos con el controlador (o controladores) y permite que éste tome decisiones centralizadas de enrutamiento y configure el comportamiento de los equipos de red. El controlador y los equipos de red se comunican a través de este protocolo, mediante mensajes para establecer y modificar reglas de flujo, lo que brinda flexibilidad y control sobre el tráfico de la red.

SSH (Secure shell)

SSH es un protocolo de red seguro para la conexión y administración remota de sistemas. Proporciona autenticación, cifrado y privacidad en la transferencia de datos. Utilizado ampliamente para acceder de forma remota a servidores y dispositivos, SSH garantiza la confidencialidad e integridad de la comunicación, así como la autenticación del usuario.

OpenVPN

Es una aplicación utilizada para establecer conexiones seguras y privadas a través de redes públicas. Permite la creación de túneles VPN utilizando protocolos de cifrado robustos, ofreciendo una capa adicional de seguridad para la transferencia de datos. OpenVPN es altamente configurable y compatible con muchos sistemas operativos y dispositivos.

Máquinas Virtuales (VM)

Una máquina virtual es un entorno de software que simula un sistema operativo completo y sus recursos en una computadora física. Permite ejecutar múltiples sistemas operativos o aplicaciones en un solo hardware, proporcionando aislamiento y flexibilidad.

Tcpdump

Es una herramienta de línea de comandos que captura y analiza el tráfico de red en tiempo real. Permite analizar paquetes de datos transmitidos en una red, mostrando información detallada de direcciones IP, puertos, protocolos, o contenido. Se utiliza para el diagnóstico y resolución de problemas en las redes telemáticas.

CentOS

Distribución de Linux de código abierto y gratuita. Conocida por su estabilidad, seguridad y soporte a largo plazo. Se utiliza comúnmente en servidores y entornos empresariales para implementar aplicaciones o servicios de manera confiable.

Proxmox Virtual Environment

Siguiendo con lo comentado en el *Capítulo 2*, este software coordina dos tecnologías de virtualización: *KVM/QEMU (kernel-based VM)* para máquinas virtuales y *LXC (Linux containers)* para contenedores, En el caso de nuestro proyecto, utilizaremos únicamente máquinas virtuales:

- **Kernel-based VM (KVM)** permite ejecutar tanto Windows como Linux en máquinas virtuales, y cada VM puede tener un hardware virtualizado privado, es decir, tarjetas de red, disco, adaptador de gráficos, etc. Es posible ejecutar varias aplicaciones en máquinas virtuales en un sólo hardware, lo que permite reducir los costes y tener una alta flexibilidad para construir un centro de datos definido por un software.

Las características que nos ofrece Proxmox son numerosas y nos permiten trabajar con comodidad y efectividad. Se describen las más representativas:

- **Virtualización para la mayoría de Sistemas Operativos.** En sus versiones 32/64 bits: Linux en todas sus versiones, Microsoft Windows 10 / 2016 / 2012 / 7 / 8/ 2003 / xp, Solaris, AIX, entre otros.
- **Backup & Restore de "Máquinas Virtuales".** Estas tareas con Proxmox son muy sencillas y se administran a través de su interfaz web. Se pueden hacer backups de forma inmediata o programadas. La restauración es muy sencilla y se realiza simplemente seleccionando el backup a restaurar.

- **Snapshot Live.** Permite hacer copias instantáneas de máquinas virtuales, incluyendo el contenido de la RAM, configuraciones y estado de sus discos virtuales. También se puede retroceder en tiempo las VM restaurando estos snapshots.

Figura 33. Esquema de copia de VMs

- **"Migración en caliente".** Esto nos permite movilizar máquinas virtuales entre cada NODO (servidor físico) sin necesidad de apagar la máquina.
- **Administración centralizada.** En un "**Cluster** Proxmox" se debe definir una de los Nodos como "Orquestador" con el objetivo de centralizar el trabajo, sin embargo cada nodo cuenta con su propio administrador Web. Sin embargo, si el nodo "Orquestador" llega a fallar, los demás nodos tienen replicada la información de éste, y se puede tomar el control.
- **Puentes de red**. Proxmox administra las tarjetas físicas a través de "Bridges" que comparte a las "Máquinas Virtuales". Es muy sencillo asociar una o varias tarjetas a un "Bridge" haciendo un balanceo automático del tráfico de datos.
- **Generación de clúster.** Generar un clúster en Proxmox nos va a aportar muchos beneficios, como administrar todas las instancias de forma centralizada, migrar máquinas virtuales/contenedores de un host a otro y **configurar fácilmente la alta disponibilidad (HA)**.
- **Alta disponibilidad.** La alta disponibilidad garantiza que una máquina virtual siga funcionando incluso si se apaga un nodo individual. Cuando se habilita, los nodos se comunican entre sí para monitorear constantemente el estado y disponibilidad de las VM. Si uno de ellos falla o se apaga, las máquinas virtuales que estaban trabajando en ese nodo migrarán automáticamente a otro que funcione correctamente, sin interrupciones o paradas que sean percibidas por el usuario final.
  Además de esto, la alta disponibilidad nos ofrece distintas funciones, como la detección de errores en cualquier punto del clúster, la recuperación de servicios caídos, una monitorización total de las máquinas virtuales y su estado, y el reinicio de máquinas que lo necesiten por haber sufrido algún fallo de cualquier tipo.

Open vSwitch (OvS)

La tecnología Open vSwitch permite más capacidades que los módulos regulares de kernel Linux, aun cuando la ruta de datos está dentro del propio kernel Linux, lo que lo hace ideal para la construcción de esquemas de redes virtuales para nubes o para investigación de nuevos protocolos de red.

OvS dispone de un diseño de mayor complejidad que los "bridges" estándar. Éstos sólo se ejecutan en el espacio del kernel del Host, mientras el Open vSwitch, aparte de hacer uso de este espacio, también se ejecuta en el "user space", y así puede tomar decisiones sobre cómo procesar los "nuevos" paquetes. Cuando un paquete de un nuevo flujo llega al OvS, este realizará una decisión bajo el user space, pero los sucesivos paquetes de ese flujo serán encaminados directamente por el kernel space, así es mucho más rápido y por lo tanto tiene mayor rendimiento.

Proporciona funcionalidades avanzadas de conmutación, segmentación de red y soporte para controladores SDN, lo que lo convierte en una herramienta muy útil en entornos de virtualización y cloud computing. Entre ellas, cabe destacar algunas que podrían ser de gran utilidad en el entorno de nuestro proyecto:

- **Balanceo de carga**: **LACP, Bonding.** Open vSwitch puede distribuir el tráfico de red de manera equitativa entre varios enlaces o interfaces, mejorando así el rendimiento y la disponibilidad de la red. LACP es a nivel de switch exterior (trunking) y el bonding a nivel de máquina virtual.

- **OpenFlow - Controlador SDN.** Open vSwitch es compatible con el protocolo OpenFlow, lo que significa que puede ser gestionado por controladores SDN. En una red definida por software (SDN) se puede conseguir que un controlador SDN, como *Ryu*, interactúe con los conmutadores compatibles con OpenFlow, como Open vSwitch, para tomar decisiones de reenvío de paquetes y configurar el comportamiento de la red. Esto, aplicado a nuestro proyecto, sería interesante en el momento en que el sistema con Open vSwitch se instalara en más de un servidor Proxmox, ya que seremos capaces de controlar de manera centralizada más de un OvS.
    1. El controlador SDN se conecta al conmutador OvS mediante el protocolo OpenFlow. El controlador envía las reglas de flujo al OvS. Definen cómo debe tratarse el tráfico que le llega.
    2. Cuando llega un paquete al OvS, se consulta si hay una regla definida para él en la propia tabla de flujos. Si no, se envía al controlador SDN a través del canal OpenFlow, para que tome una decisión.
    3. Actualización dinámica: El controlador puede enviar instrucciones de actualización o modificación de reglas de flujo al conmutador OpenvSwitch en cualquier momento. Esto permite una configuración dinámica de la red en función de las condiciones cambiantes o de las políticas definidas.

- **SPAN, RSPAN.** Son las características propias que nos permiten hacer ***mirroring*** de un puerto o incluso del bridge entero. Es decir, mandar una copia de los paquetes vistos en un puerto del switch a una interfaz para la monitorización de red en ese punto.

ANEXO B. Funcionamiento de la empresa SONOC.

Como hemos explicado, SONOC virtualiza sus servidores con el entorno de virtualización *Proxmox*. En caso de que ocurriera cualquier fallo en las comunicaciones, como los que hemos mencionado, se debe ser capaz de detectar qué es lo que está fallando y dónde. En estos servidores Proxmox, se pueden instalar cantidad de máquinas virtuales. La mayoría de servidores, contienen máquinas en las que se instalan servidores SIP Proxy que hagan posible esta señalización SIP necesaria para las conexiones telefónicas.
Así, SONOC recibe tráfico entrante de señalización SIP por parte de la operadora cliente, y éste entrará a los Proxys instalados en los servidores virtualizados:

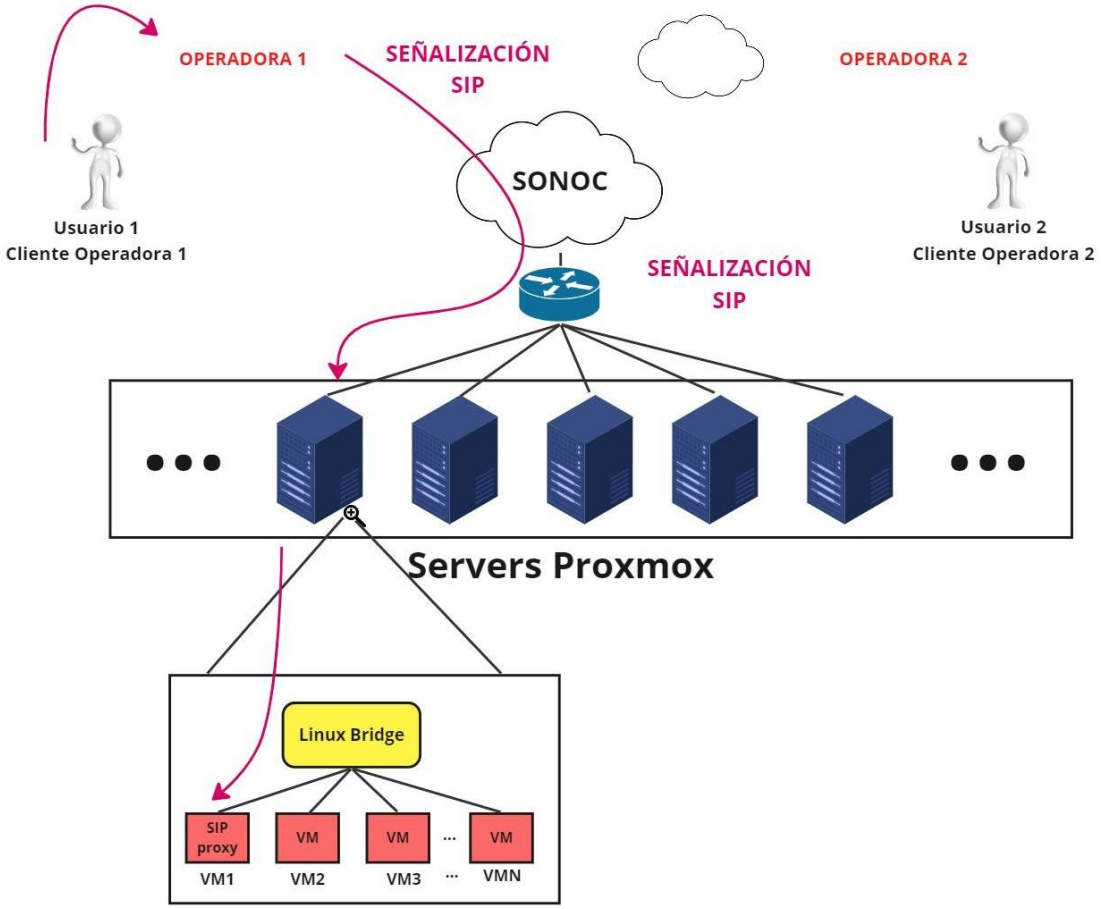


Figura 34. Entrada de señalización SIP a la empresa.

Posteriormente, como hemos explicado, es posible que la señalización podamos redirigirla directamente a la operadora del destinatario, o tenga que pasar por otro proveedor. Esta señalización llega a la operadora y se realiza el intercambio de mensajes SIP:

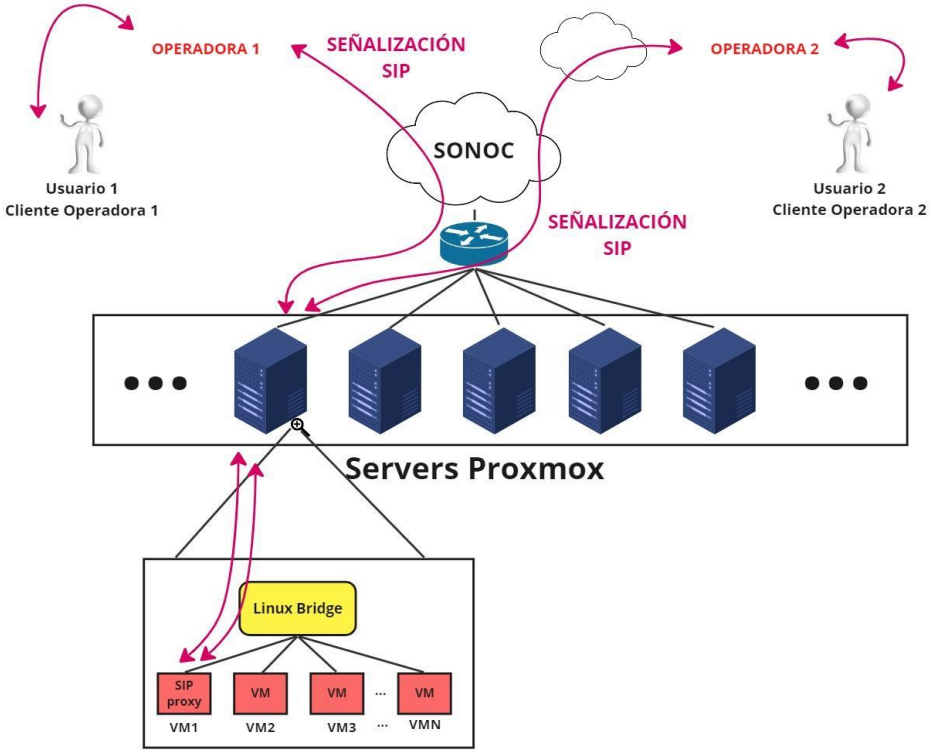


Figura 35. Intercambio señalización SIP

Una vez establecida la conexión SIP, los extremos ya pueden comunicarse entre sí, sin necesidad de los intermediarios:

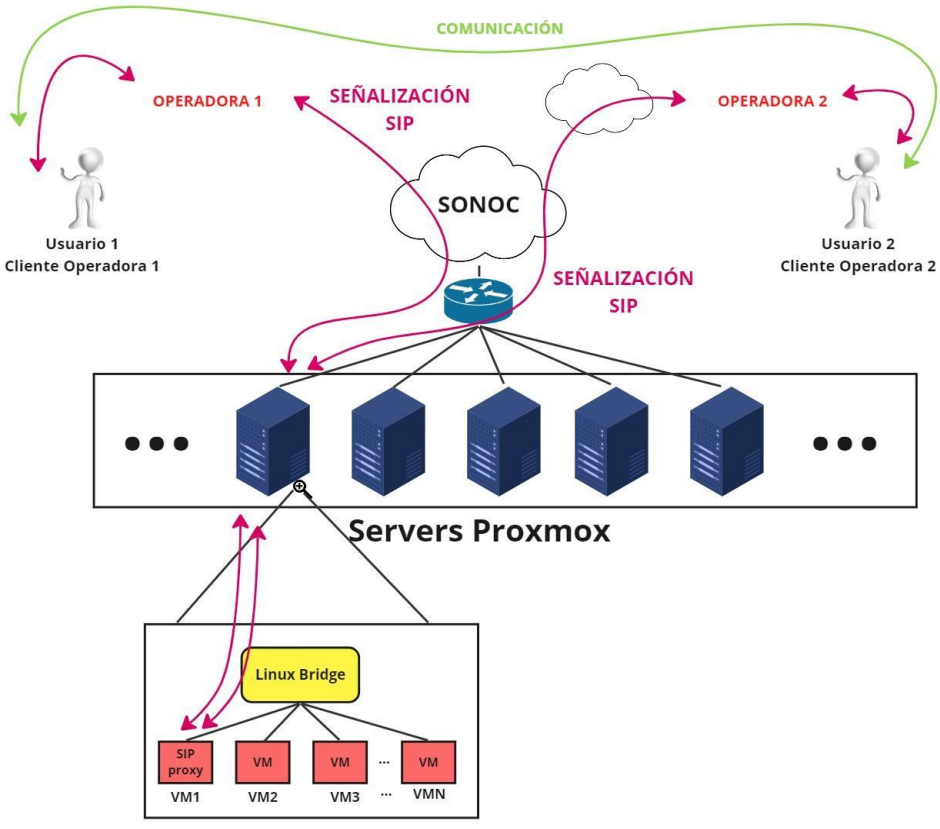


Figura 36. Comunicación final y papel de Sonoc en ella

ANEXO C. Entorno de trabajo.

El despliegue con el que se realiza este estudio es el necesario para poder conectarse a los servidores con los que se van a realizar las pruebas de funcionamiento. Al querer conectarnos remotamente, el despliegue consta de una conexión VPN mediante la herramienta *OpenVPN* que proporciona acceso a la red de la empresa SONOC, pudiendo así conectarse desde cualquier red a los servidores en cuestión. También se prepara el encaminamiento de forma que con la IP que se me asigne pueda acceder sin problema a las máquinas necesarias.

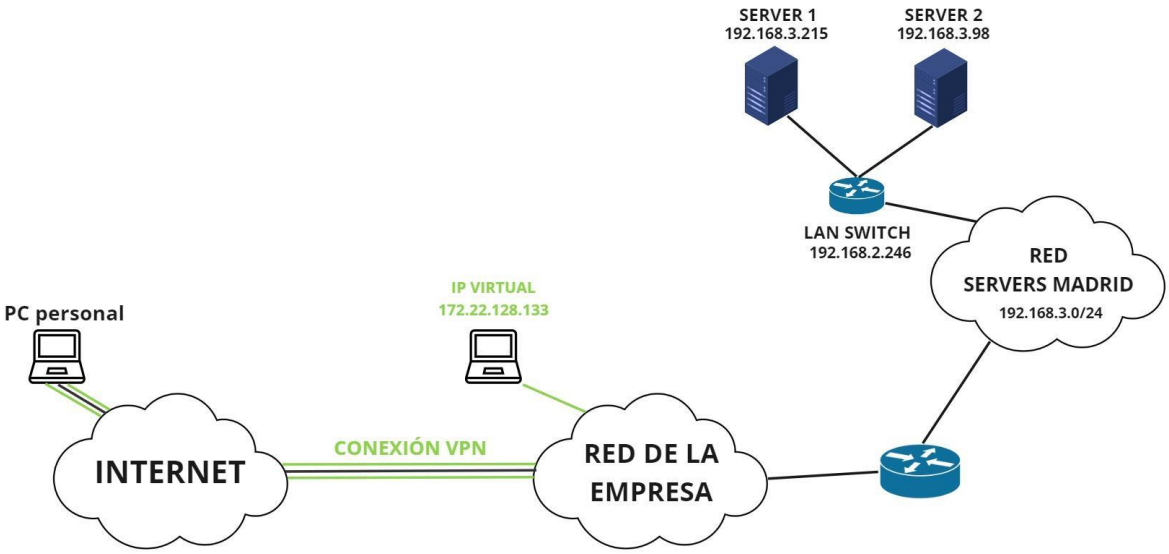


Figura 37: Despliegue completo inicial.

El entorno de trabajo final con el que se trabaja se muestra en la *Figura 35.* Consta de dos servidores propios de la empresa, y un LAN Switch que realiza las tareas de interconexión. Se trata, en concreto, de dos servidores *Intel* y un *LAN Switch Catalyst de Cisco*.

Estos equipos son con los que se interactúa a lo largo del proyecto. Por un lado se dota de conexión con el exterior al sistema por medio de la configuración de los puertos del LAN Switch. Por otro lado, en los servidores se instala la plataforma de virtualización elegida para el proyecto (Proxmox VE), y se crearán en ellos los escenarios necesarios para el desarrollo.

El primer paso para poder dar la conectividad deseada, es crear una conexión VPN desde el ordenador desde el que se harán las configuraciones (en este caso el del estudiante). De este modo, será posible acceder a los servidores, lo que no se podría hacer directamente como es obvio. La empresa SONOC dispone de cuatro servidores VPN dependiendo de cuál sea el objetivo de su uso. La VPN a la que me conecto, da conectividad directa sin restricción a la red en la que se sitúan los servidores a manipular. Para conseguir la conectividad, utilizamos el programa *OpenVPN.* Se me asigna una IP dentro de la red de la empresa. Dicha dirección es, como podemos observar en el esquema, la *172.22.128.133*.

Tras conseguir conectividad con la VPN, se sigue un proceso de comprobación de condiciones en los diversos Firewall que atravesará mi PC hasta llegar a la red de los servidores. De este modo, aseguramos que no se nos corte el acceso en ningún punto de la infraestructura de la empresa, ya que como es obvio hay un gran control de acceso a los equipos de la empresa. Tras permitir completamente mi acceso, ya podemos trabajar desde cualquier lugar con los servidores de la sede de Madrid.

En este momento, nuestro PC con el que configuraremos todo lo necesario, obtiene una IP de la red de la empresa, y tiene acceso a la red de SONOC en Madrid. Dentro de esta red, los equipos que intervienen en el proyecto son un *LANSwitch Catalyst de Cisco*, y dos *servers Intel*. Es necesario configurar la red de los equipos, asignándoles IPs, configurando los puertos correctamente y dándoles conectividad entre ellos.

La configuración del Switch debe ser cuidadosa ya que es un equipo en producción con unos diez servidores de la empresa conectados a él. Desde este Switch, se buscan seis puertos libres. De este modo, vamos a conectar dos a cada server para tráfico, más uno a cada uno de ellos que irá conectado a la *iDrac*, puerto de configuración de los *Intel*. La IP y nombre que asignamos al *LAN Switch* es: **ES-MAD-IXI-SNC-LSW-004-new: 192.168.2.246**

Respecto a los dos puertos conectados a los servidores para tráfico, se decide realizar un bond entre ambos, siguiendo el protocolo *LACP (802.3ad).*

Los servidores con los que vamos a trabajar están conectados al Switch por medio de tres enlaces, como he comentado anteriormente. Lo primero que debemos hacer, es configurar las interfaces de forma coherente con las del Switch, para que se comuniquen correctamente. Por ejemplo, debemos configurar la VLAN que tomamos como nativa, el *bond LACP* de igual manera que en el LAN Switch, los MTU, y diversas opciones específicas. De esta forma y tras configurar la red correctamente, tendremos conectividad total entre los tres dispositivos, pudiendo comunicarse entre ellos. Las IPs y nombres asignados a cada servidor son los siguientes:

- **pve1-tfg.sonoc.io: 192.168.3.215**
- **pve2-tfg.sonoc.io: 192.168.3.98**

Una vez tengan el aspecto de conectividad completamente configurado, se procede a la instalación de *Proxmox* en cada uno de los dos servidores. En este caso, la empresa tiene un manual documentado para simplificar la instalación, así que se procede según éste. La instalación se realiza de forma remota desde un equipo en Zaragoza, conectándonos a la *iDrac* de cada servidor. Generamos un CD virtual con la *ISO* (Proxmox VE 7.2) y a partir de él, se instala. Cuando ya está instalado, se puede acceder al entorno de virtualización *Proxmox* por medio de conexión remota *ssh* o a través de su *interfaz web* para trabajar con los servidores.

ANEXO D. Configuración de los servidores de trabajo.

Partiendo de lo descrito en el capítulo *4.1. Configuración y conectividad de los servidores de trabajo,* las primeras dos interfaces conectadas al Switch y que servirán para comunicarse con el otro servidor, son eno1 y eno2. Éstos son enlaces ethernet que configuramos desde consola a través del fichero de configuración */etc/network/interfaces* que se presentará seguidamente. Hemos configurado los enlaces modificando su MTU, que por defecto era de 1500, poniéndola a 9000, con el objetivo de ganar ancho de banda en los enlaces y evitar problemas de carga. Además se opta por realizar un bond de estos dos puertos con fines que se han descrito en su capítulo correspondiente.

La otra interfaz, es la propia de la iDrac del servidor. Este puerto tendrá una IP desde la que conectarse remotamente para realizar la configuración deseada. La controladora de acceso remoto integrada de Dell (iDRAC) está integrada en todos los servidores Dell PowerEdge. Proporciona la funcionalidad que ayuda a implementar, actualizar, supervisar y mantener los servidores Dell PowerEdge con o sin un agente de software de administración de sistemas. No requiere de un sistema operativo o un hipervisor para trabajar ya que está integrada desde fábrica. Lo que facilita es implementar un nuevo sistema operativo o aceptar una nueva configuración sin tener que instalar software adicional.

Esta herramienta proporciona tanto una interfaz web como una interfaz de línea de comandos que permiten a los administradores realizar estas tareas de manera remota. La interfaz web desde la que se configura es similar a la siguiente figura:

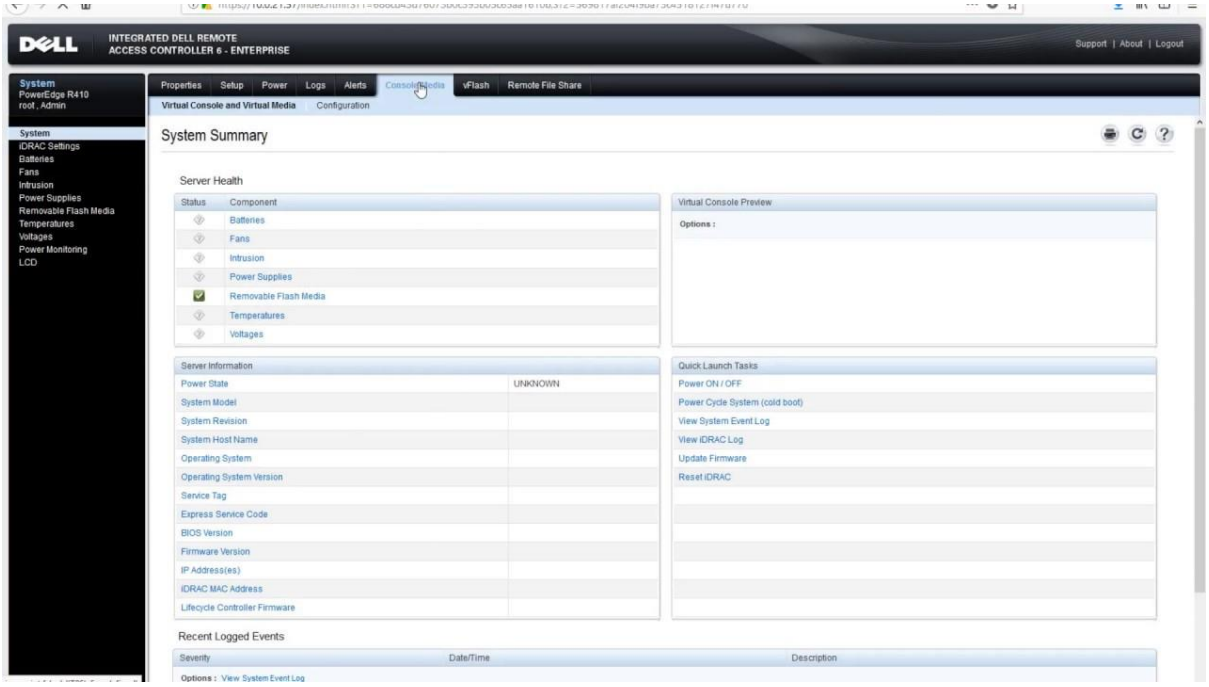


Figura 38. Interfaz web del acceso a la iDrac

Respecto a la configuración de red propia de cada servidor, se van a presentar los archivos de configuración de cada uno de ellos, ubicados en el directorio */etc/network/* y con nombre *interfaces*:

SERVIDOR 1

```
# Configuración Servidor 1 (Open vSwitch) en /etc/network/interfaces

auto lo
iface lo inet loopback

# Configuración interfaces físicos
auto eno1
iface eno1 inet manual
ovs_mtu 9000

auto eno2
iface eno2 inet manual
ovs_mtu 9000

#Configuración BOND LACP 802.3ad
auto bond0
iface bond0 inet manual
    ovs_mtu 9000
    ovs_bridge vmbr0
    ovs_type OVSBond
    ovs_bonds eno1 eno2
    ovs_options bond_mode=balance-tcp lacp=active
    stp_enable=false

#Configuracion OvS bridge
auto vmbr0
iface vmbr0 inet manual
    address 192.168.3.215/24
    gateway 192.168.3.1
    ovs_type OVSBridge
    ovs_ports bond0
    ovs_mtu 9000

post-up ip route add 79.170.64.0/21 via 192.168.3.254 dev vmbr0
post-up ip route add 87.237.84.0/24 via 192.168.3.254 dev vmbr0
post-up ip route add 87.237.86.0/24 via 192.168.3.254 dev vmbr0
post-up ip route add 192.168.0.0/16 via 192.168.3.254 dev vmbr0
post-up ip route add 172.16.0.0/12 via 192.168.3.254 dev vmbr0
```

SERVIDOR 2

```
# Configuración Servidor 2 (Linux Bridge) en /etc/network/interfaces

auto lo
iface lo inet loopback

# Configuracion interfaces físicos
auto eno1
iface eno1 inet manual
mtu 9000

iface eno2 inet manual
mtu 9000

# Configuracion BOND LACP 802.3ad
auto bond0
iface bond0 inet manual
  mtu 9000
```

```
  bond-slaves eno1 eno2
  bond-miimon 100
  bond-mode 802.3ad
  bond-xmit-hash-policy layer2+3

# Linux bridge configuration
auto vmbr0
iface vmbr0 inet static
  address 192.168.3.98/24
  gateway 192.168.3.1
  mtu 9000
  bridge-ports bond0
  bridge-stp off
  bridge-fd 0
  bridge-pvid 12

post-up ip route add 79.170.64.0/21 via 192.168.3.254 dev vmbr0
post-up ip route add 87.237.84.0/24 via 192.168.3.254 dev vmbr0
post-up ip route add 87.237.86.0/24 via 192.168.3.254 dev vmbr0
post-up ip route add 192.168.0.0/16 via 192.168.3.254 dev vmbr0
post-up ip route add 172.16.0.0/12 via 192.168.3.254 dev vmbr0
```

Como podemos ver, inicialmente se define la interfaz de loopback y las interfaces físicas se definen con un MTU de 9000, mediante los comandos `ovs_mtu 9000` y `mtu 9000` dependiendo de si estamos configurando el servidor con OvS instalado o no.
Después se define el Bond LACP, que involucra a las dos interfaces físicas *eno1* y *eno2*. Para el caso del Servidor 1 los comandos son diferentes ya que es necesario el uso de los comandos propios de configuración de OvS. Tras ello, se configura la propia interfaz de red o bridge virtual. Para el servidor 2, le asignamos una IP (con la que se conectarse a la red), definimos su gateway y tomamos como puerto del bridge el Bond que acabamos de crear. La configuración del Servidor 1 (Open vSwitch) se detalla en el ANEXO F.

Finalmente, para ambos servidores, se definen varias reglas de encaminamiento añadiendo rutas necesarias para la conectividad al exterior desde SONOC, y que implementan todos los servidores que están en funcionamiento.

1. Bond LACP

Los servidores suelen ser los más perjudicados en cuanto a la capacidad de las tarjetas de red, ya que se pueden ver con un tráfico mayor de lo normal debido a distintos factores (una alta demanda de servicios, nuevo software que consume más ancho de banda, etc…) y puede darse el caso de que aunque el servidor es capaz de soportar la carga a nivel de RAM y CPU, la tarjeta de red no es capaz de soportar tal tráfico, haciendo que los datos pasen por un cuello de botella que haría que no se pudiese sacar todo el partido que se desearía al servidor o incluso perder paquetes.

Las ventajas principales que nos aporta realizar un *bond* entre los puertos son las siguientes:

- **Mayor ancho de banda.** Al combinar múltiples enlaces en un solo enlace lógico, el port bonding permite aumentar el ancho de banda disponible. Esto es especialmente útil

para aplicaciones que requieren una alta capacidad de transferencia de datos, la transferencia de archivos grandes o el uso intensivo de la red en entornos empresariales.

- **Mayor disponibilidad y tolerancia a fallos.** Si uno de los enlaces falla, el port bonding permite que el tráfico de red se redirija automáticamente a los enlaces restantes sin interrupciones. Esto mejora la disponibilidad de la red y reduce el tiempo de inactividad.
- **Balanceo de carga.** El port bonding permite distribuir el tráfico de red entre los enlaces disponibles, lo que resulta en un balanceo de carga efectivo. Esto evita la congestión en enlaces individuales y optimiza la utilización de la capacidad de red disponible.

Por todo ello, es beneficioso para SONOC ya que requiere de una red confiable y de alto rendimiento para trabajar con tráfico de llamadas.

Este *bonding* se ha realizado siguiendo el **protocolo LACP**, ya explicado en el apartado *2.1. Protocolos y herramientas utilizadas.* En la práctica, el protocolo LACP cumple el principio general de la agregación de enlaces: el esfuerzo de establecer estructuras de red paralelas para proporcionar redundancia, o para mejorar el rendimiento.

Con todo ello se persigue no tener ningún tipo de problema en cuanto a ancho de banda se refiere, a la hora de recibir toda la información relativa a llamadas que recoge la empresa en sus servidores.

A la hora de implementarlo en nuestro escenario, debemos modificar el archivo de configuración de red especificando la configuración del Bond de cada uno de los servidores, como hemos visto al principio de este anexo.

ANEXO E. Interfaz web de Proxmox

1. Funcionamiento

Proceso de login

Cuando nos conectamos al servidor Proxmox a través de la interfaz web, lo primero que se ve es la ventana de login. Soporta varios servidores de autenticación, y se puede seleccionar el idioma.

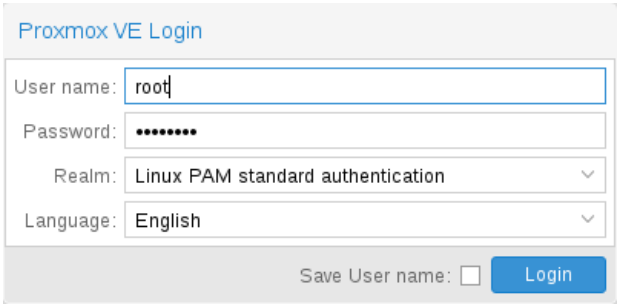


Figura 39. Ventana de login de Proxmox

Interfaz gráfica de usuario (GUI)

La interfaz de usuario consta de cuatro regiones principales: *Header*, *Árbol de recursos*, *Panel de contenido y Panel de registro*. En nuestro Proxmox se ve de la siguiente forma:

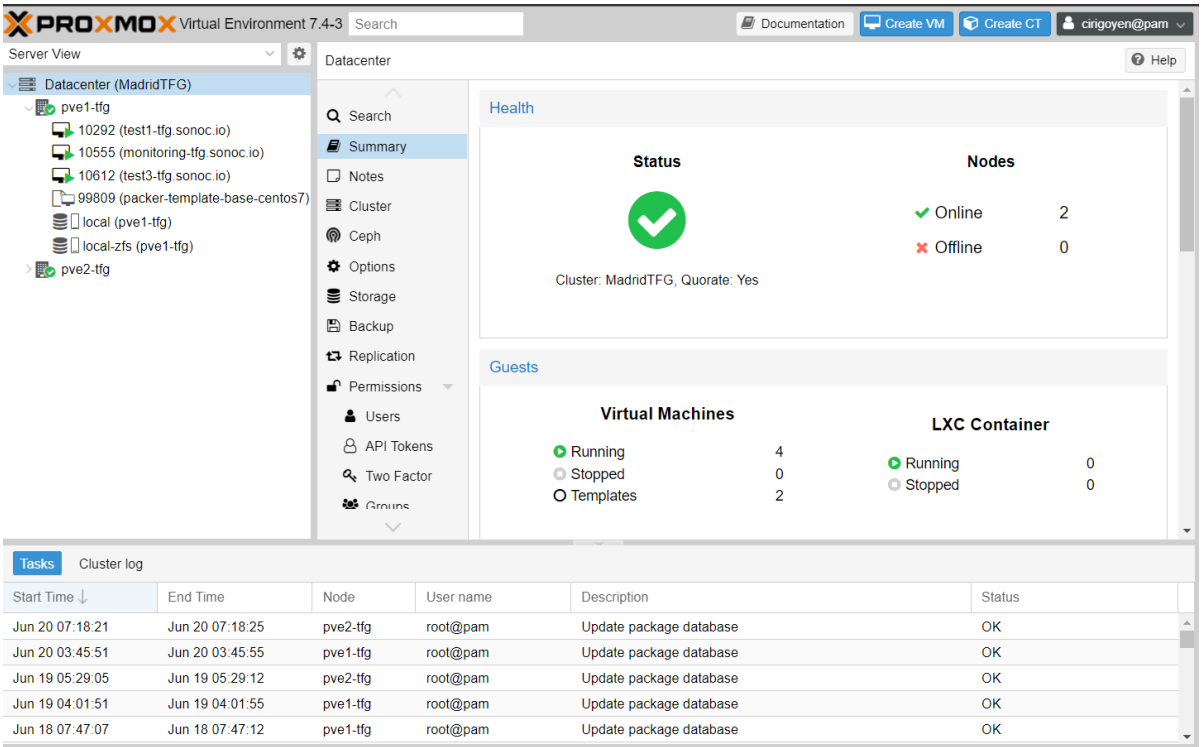


Figura 40. Interfaz gráfica de usuario en Proxmox

El **header** muestra información de estado y contiene botones para las acciones más importantes. Podemos ver el logo, la versión de Proxmox, un buscador para objetos específicos a encontrar (VMs, contenedores, nodos…). A la derecha entrando en nuestro nombre de usuario, podemos acceder al apartado *My settings*. Allí podemos ajustar parámetros almacenados localmente. Permite habilitar o deshabilitar almacenamientos específicos.

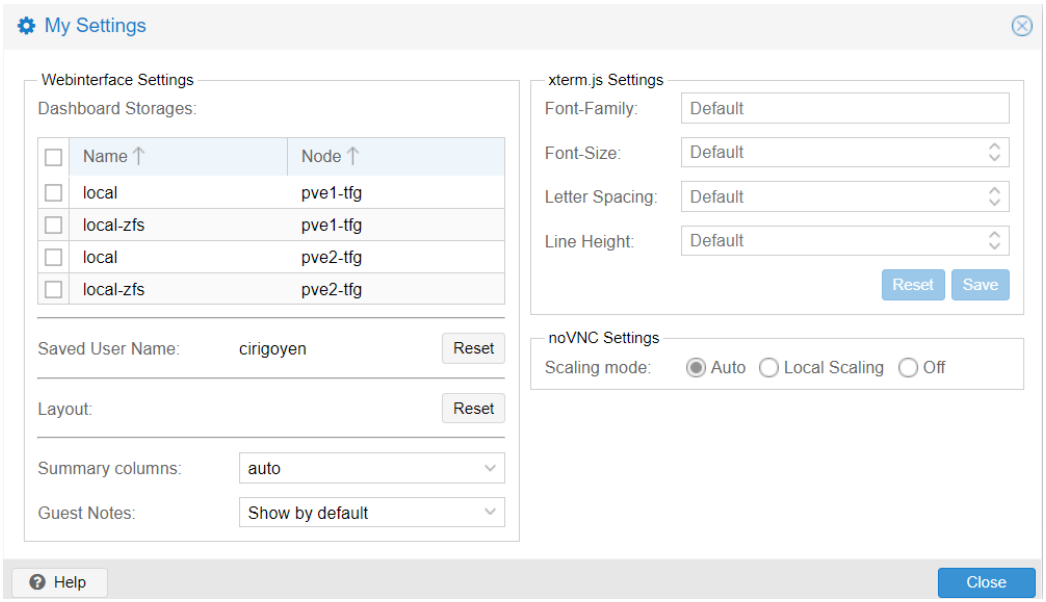


Figura 41. Ventana de "My Settings".

Además se puede acceder a *ayuda*, crear Máquinas Virtuales y Contenedores directamente.

El **árbol de recursos** es el árbol de navegación principal. Se puede cambiar la vista, pero la predeterminada es *Vista del servidor*. Muestra el centro de datos (configuración del clúster), los nodos del clúster, los *guests* (VMs, contenedores, plantillas), el almacenamiento de datos, y posibles *pool* de guests.

En nuestro caso, al haber generado un clúster con los dos servidores (*pve1-tfg* y *pve2-tfg*) observamos lo siguiente:

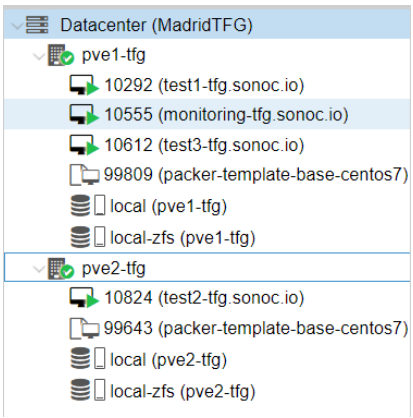


Figura 42. Árbol de recursos

El **panel de registros** (Log Panel) se encarga de mostrar lo que está sucediendo actualmente en el clúster. Se muestran registros referidos a todas las acciones que se realizan, como puede ser generar máquinas virtuales. Podemos ver si alguien más está trabajando sobre el mismo clúster.

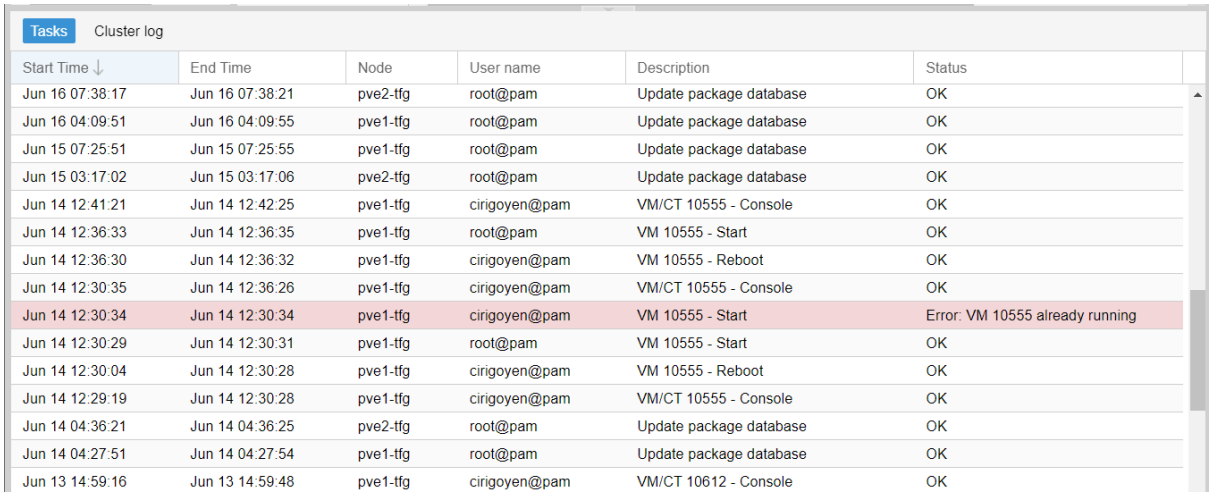


Figura 43. Log Panel del clúster

Paneles de contenido

Cuando selecciona un elemento del árbol de recursos, el objeto correspondiente muestra información de configuración y estado en el panel de contenido. Vamos a dar una breve descripción de esta funcionalidad y de sus apartados más importantes.

En el **centro de datos** se accede a la información de todo el clúster:

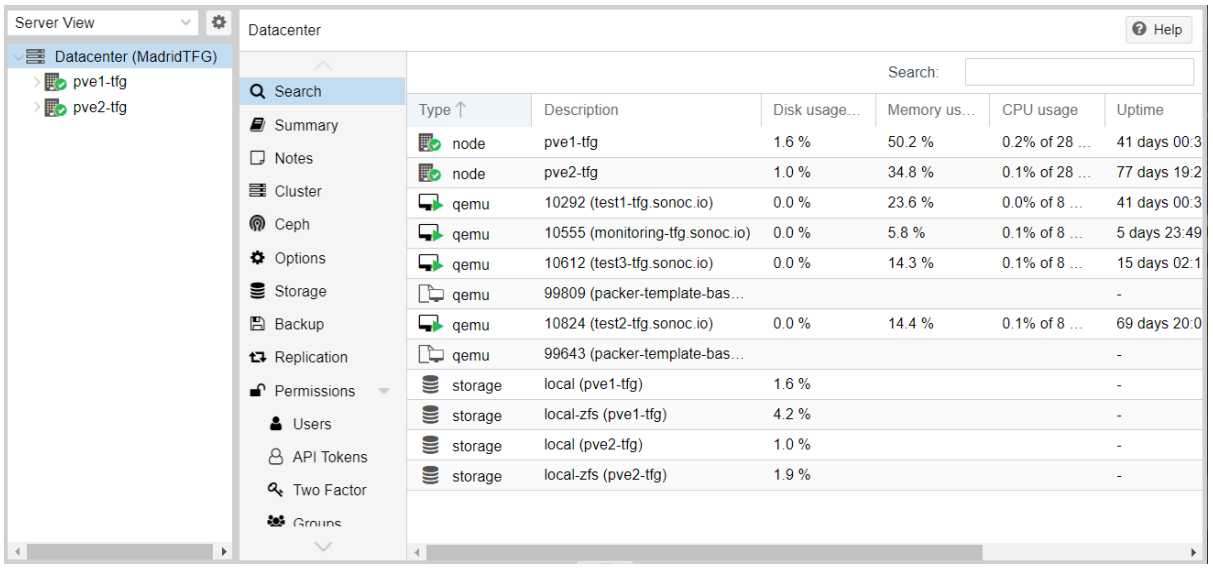


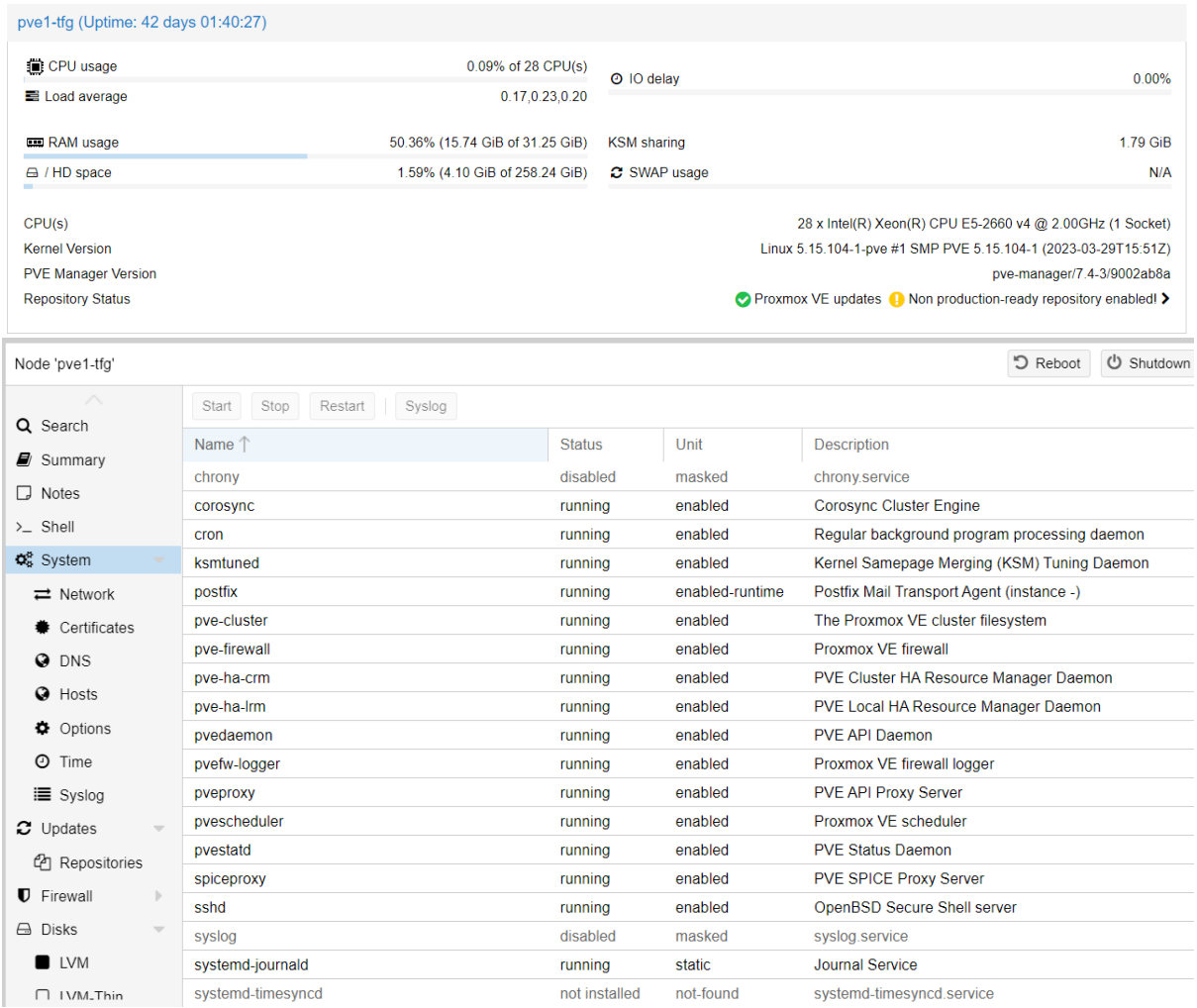
Figura 44. Información del Datacenter.

- ➢ Search: Búsqueda de nodos, VMs, dispositivos de almacenamiento…
- ➢ Summary: Breve descripción general del estado y uso de recursos.
- ➢ Cluster: Funcionalidad e información para crear o unirse a un clúster.
- ➢ Storage: Administra el almacenamiento del clúster.
- ➢ Permissions: Administra permisos de token de usuario, grupo y API, y autenticación.
- ➢ HA: Administra la alta disponibilidad de Proxmox VE.

2. Información de Nodos y Máquinas Virtuales

En este subapartado se muestra la configuración concreta de algún servidor o máquina virtual de nuestro proyecto, presentada por la interfaz de Proxmox, con el fin de comprender cómo son los equipos con los que se trabaja.

Nodos:



Figuras 45 y 46. Información general del nodo "pve1"

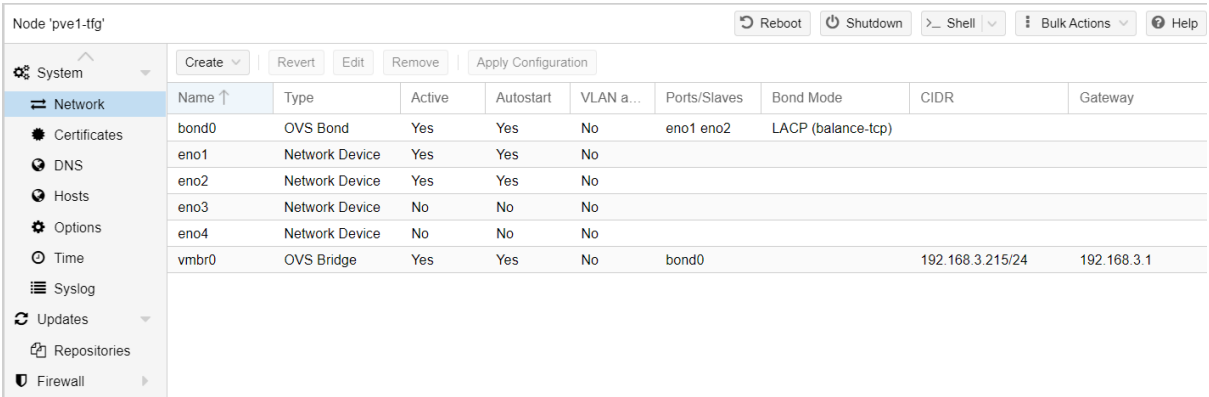


Figura 47. Información de red del nodo "pve1"

Máquinas virtuales:

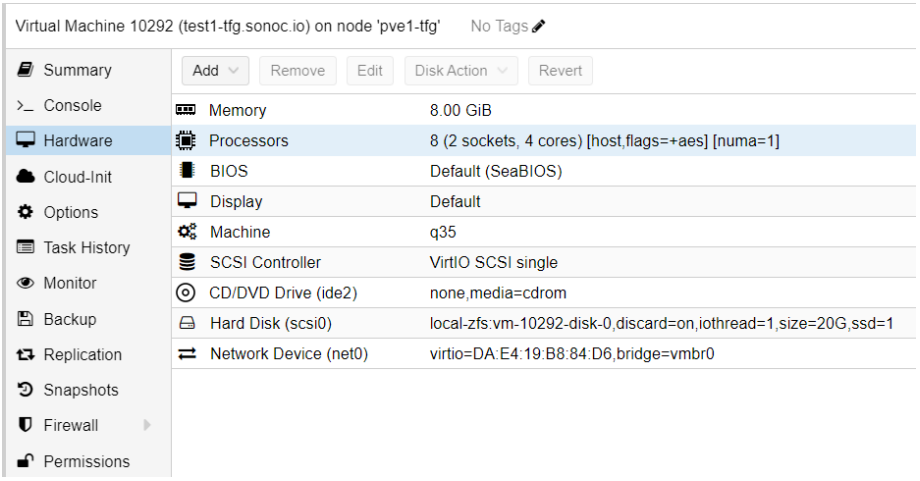


Figura 48. Información de hardware

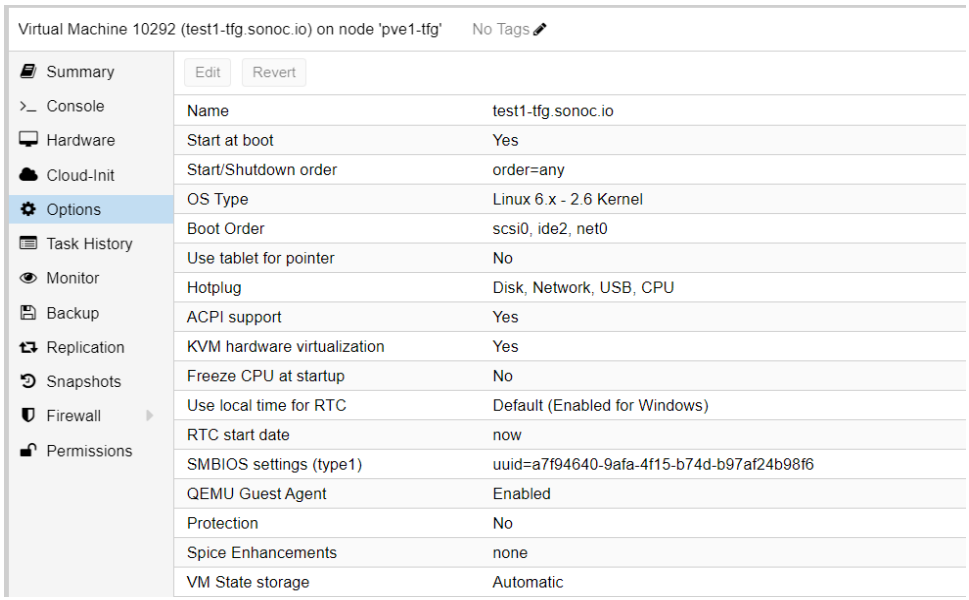


Figura 49. Opciones de configuración

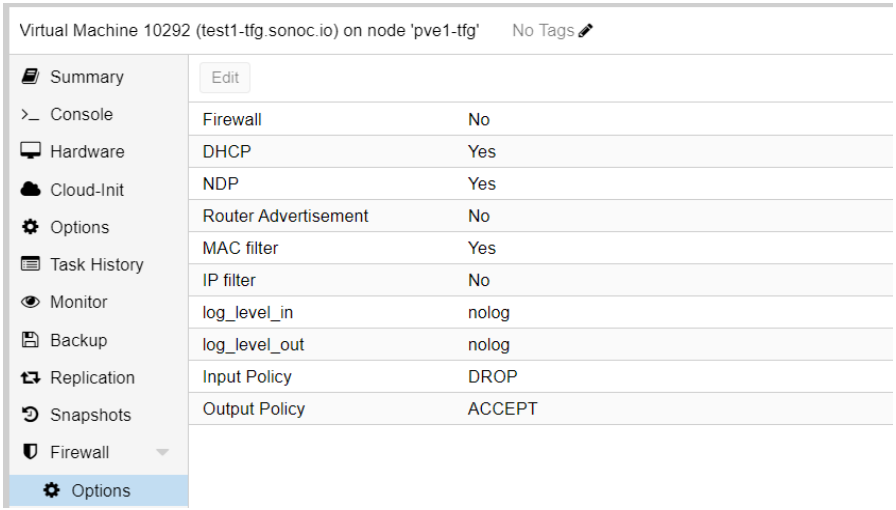


Figura 50. Opciones de Firewall

ANEXO F. Configuración de red con Open vSwitch.

1. Configuración de red

Siguiendo con lo comentado en el ANEXO D, se va a describir y comentar la configuración de red necesaria para poner en funcionamiento el OvS dentro de nuestro Servidor 1. Recuperando el código ya presentado de las interfaces de red, centrándonos en la parte en la que se define el bridge virtual como Open vSwitch:

```
#Configuracion OvS bridge
auto vmbr0
iface vmbr0 inet manual
    address 192.168.3.215/24
    gateway 192.168.3.1
    ovs_type OVSBridge
    ovs_ports bond0
    ovs_mtu 9000
```

Definimos primero el bridge como *vmbr0*.
Le damos una IP de conectividad al servidor.
Definimos su puerta de enlace.
El `ovs_type` se declara como `OVSBridge` ya que queremos construir un *Bridge* virtual.
El puerto conectado al *bridge* es el Bond "bond0" (unión de los dos enlaces físicos).
El MTU se fija a 9000 como en los demás casos.

Centrándonos en la gestión del propio OvS, se utilizan distintos comandos:

- **`ovs-dpctl`**: gestiona datapath
- **`ovs-ofctl`**: gestiona flujos OpenFlow
- **`ovs-vsctl`:** gestiona la DB ovsdb y los *switches* Open vSwitch (el comando utilizado mayoritariamente en este proyecto)

En nuestro escenario, vamos a configurar Open vSwitch como ***Learning Switch.*** Sólo maneja una única entrada en la tabla de flujos, es el funcionamiento NORMAL (es como un *switch* L2 con aprendizaje, directamente usa el datapath). No definiremos flujos, aunque es una posibilidad que se podría plantear por sus ventajas de cara al futuro.

Los conmutadores OvS vienen por defecto con una regla creada que los lleva a funcionar como un *learning switch*. Se puede ver mostrando toda la programación de flujos en el switch mediante el comando:

```
# ovs-ofctl dump-flows <NombreSwitchOvS>
```

Que nos devuelve lo siguiente, donde se puede ver la única regla de flujos por defecto en la que se especifica un comportamiento de aprendizaje de un switch normal.

```
root@pve1-tfg.sonoc.io:/home/cirigoyen# ovs-ofctl dump-flows vmbr0
 cookie=0x0, duration=4232071.461s, table=0, n_packets=688072013, n_bytes=105784209187, priority=0 actions=NORMAL
```

Podemos visualizar los conmutadores OvS presentes en el servidor y una breve descripción general del contenido de la base de datos de ellos mediante el comando:

```
# ovs-vsctl show
```

Que nos devuelve lo siguiente:

```
root@pve1-tfg.sonoc.io:/home/cirigoyen# ovs-vsctl show
41caee33-97ca-4c61-b3b2-0d0d348ed407
    Bridge vmbr0
        Port tap10555i1
            Interface tap10555i1
        Port bond0
            Interface eno2
            Interface eno1
        Port vmbr0
            Interface vmbr0
                type: internal
        Port tap10612i0
            Interface tap10612i0
        Port tap10555i0
            Interface tap10555i0
        Port tap10292i0
            Interface tap10292i0
    ovs_version: "2.15.0"
```

Y podemos observar las interfaces principales del Open vSwitch. Los puertos *tap1…* son las interfaces conectadas a las máquinas virtuales. En este caso tenemos tres máquinas conectadas. Por ejemplo, la máquina virtual con ID 10555, utiliza los puertos *tap10555i0* (eth0) y *tap10555i1* (eth1).
El puerto vmbr0 es la propia interfaz de red del *bridge*. También se define el Bond *bond0*, con las interfaces físicas que agrupa.

2. Ventajas de la solución final

- **Ahorro de recursos.** Como hemos descrito en las posibles soluciones alternativas, para capturar todo el tráfico entrante a un servidor determinado sería necesario instalar una máquina física adicional, que recibiera el tráfico destino tras programar un *port mirroring* en un LAN Switch físico. Con Open vSwitch, sólo será necesario instalar una máquina virtual dentro del propio servidor, que reciba el tráfico programado en el *port mirroring* del propio bridge virtual Open vSwitch.
- **Simplificación del trabajo.** En el sistema actual de trabajo, el tráfico telefónico se analiza en su totalidad, todos los flujos mezclados, sin posibilidad de elegir qué tráfico es el de interés. Además, cuando se quiere detectar un problema y para ello se monitorea en los equipos de interconexión (como puede ser un Router Backbone), se está estudiando una cantidad de tráfico inmensa, ya que por estos equipos discurre tráfico de cualquier tipo, en gran cantidad. Esto requiere de un trabajo de filtrado extra, que con Open vSwitch nos ahorramos.
- **Efectividad.** La implementación de Open vSwitch es muy efectiva, ya que somos capaces de actuar en cualquier momento eligiendo qué tráficos son los que queremos analizar, de una forma sencilla, simplemente modificando el *mirror* a realizar. Esto, por ejemplo, nos permite actuar con rapidez ante problemas que pasan por varias máquinas virtuales concretas. Simplemente deberemos elegir sus interfaces y redirigir su tráfico hacia el equipo de monitoreo dentro del servidor. Así analizamos directamente y sin necesidad de filtrar el tráfico entrante a las máquinas virtuales donde queremos detectar alguna anomalía.

- **Posibilidades adicionales.** Además de la gran ventaja que nos supone poder redirigir el tráfico (*port mirroring*) hacia una máquina que sólo recibirá el susceptible a analizar, Open vSwitch es una tecnología que nos ofrece muchas más funcionalidades de las que nos podemos aprovechar para mejorar el análisis de tráfico SIP (telefónico). Podemos remarcar varias de ellas, de las que se hablará más en profundidad en otros apartados. Primero, se pueden programar **flujos**, que seleccionen el tipo de comunicaciones que se desea analizar, pudiendo (por ejemplo) descartar el resto. Por otro lado, existe la posibilidad de utilizar un **controlador** externo, desde el que controlar y configurar más de un Open vSwitch ubicados en diferentes servidores Proxmox, para realizar un control centralizado y escalar la solución.

ANEXO G. Códigos para implementar el *Port Mirroring*.

1. Funcionamiento del código

Como se ha descrito en el apartado *4.3.4. Configuración del Port Mirroring*, el comando utilizado para declarar un *mirror* sigue la siguiente estructura:

```
$ovs-vsctl -- set bridge vmbr0 mirrors=@m -- \
--id=@port1 get Port eth1 -- \
--id=@port2 get Port eth2 -- \
--id=@m create mirror name=mirror1 \
select-dst-port=@port1 \
select-src-port=@port1 \
output-port=@port2 \
-- set bridge vmbr0 mirrors=@m
```

Primero, se declara el Bridge al que se le aplicará el *Port Mirroring*.
Mediante la instrucción `--id=@port1 get Port eth1 --` definimos los puertos con los que vamos a trabajar y les asignamos un "id". En este caso, estaríamos llamando *port1* a la interfaz *eth1*. Posteriormente al indicar los puertos a los que hacer *mirror*, utilizaremos este *id* para referirnos a ellos.
Con `--id=@m create mirror name=mirror1` nombramos el *mirror* a realizar como *mirror1*.

Mediante la instrucción `select-dst-port=@port1` declaramos que queremos copiar los paquetes que tienen como puerto destino el puerto con ID "*port1"*. Es decir, en este caso, se haría *mirroring* de los paquetes que entren al OvS por la interfaz *eth1*.
De la misma forma, `select-src-port=@port1` indica que se van a copiar los paquetes que tengan como puerto origen *eth1*, por tanto, los paquetes que salgan por este interfaz del OvS.

La instrucción `output-port=@port2` sirve para declarar el puerto destino al que se enviarán todos los paquetes replicados en el proceso de *mirroring.* Por tanto, los paquetes que cumplan lo especificado en las líneas anteriores serán enviados por el interfaz *eth2*.

En resumen, ejecutando esta instrucción en el servidor con Open vSwitch instalado, estaremos haciendo *mirroring* de todo el tráfico que entre o salga del interfaz *eth1* hacia el puerto destino *eth2*.

Ahora, se van a presentar los comandos programados para las pruebas concretas de funcionamiento que se realizan en el estudio:

2. Escenario 1

Como se ha descrito en el capítulo *5.1. Primer escenario: pruebas y resultados*, estamos simulando la situación de que sólo disponemos de una máquina tratando tráfico de señalización SIP que queremos monitorizar o analizar. Se quiere realizar un *Port Mirroring* de manera que el tráfico de señalización con el que trabaja la máquina *test1* se reciba en la otra máquina virtual dentro del servidor: *monitoring*. Desde esta máquina se analizará este tráfico mediante aplicaciones varias dedicadas para ello.

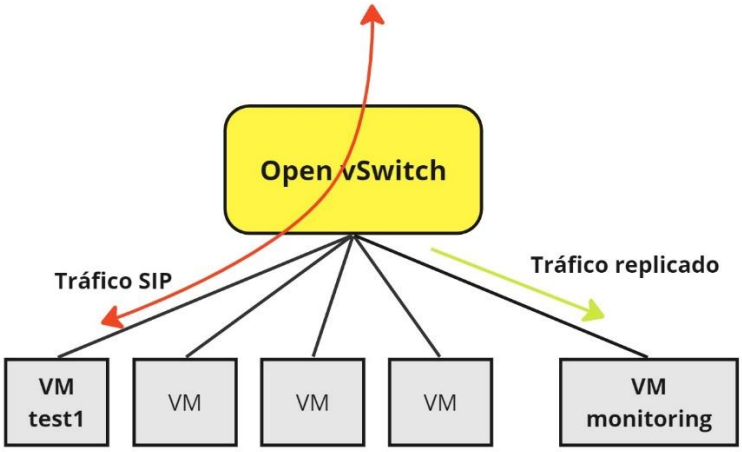


Figura 51. Situación del escenario de pruebas 1.

El código necesario para que esto suceda correctamente, siguiendo el esquema planteado al principio del anexo, es el siguiente:

```
$ovs-vsctl -- set bridge vmbr0 mirrors=@m – \
--id=@monitoring get Port tap10555i1 – \
--id=@test1 get Port tap10292i0 – \
--id=@m create mirror name=mirror1 \
select-dst-port=@test1 \
select-src-port=@test1 \
output-port=@monitoring\
--set bridge vmbr0 mirrors=@m
```

Las interfaces del OvS en juego son:

- tap10555i1: Interfaz conectada a *net1* de la VM *monitoring* (La destinada a recibir el tráfico copiado). Con ID "@monitoring"
- tap10292i0: Interfaz conectada a *net0* de la VM *test1*, por donde intercambia el tráfico con el exterior. Con ID "@test1"

3. Escenario 2

Atendiendo a la situación descrita en el capítulo *5.2. Segundo escenario: pruebas y resultados*, se simula un escenario en el que más de una máquina está recibiendo tráfico de señalización SIP. Por tanto, podemos seleccionar cuál de las N máquinas (en este caso 2) es la que queremos monitorizar, o decidirnos por analizar el tráfico de todas ellas. Así, tenemos varias posibilidades para el comando del *Port Mirroring* dependiendo de qué tráfico nos interesa recibir en la máquina de monitoreo.

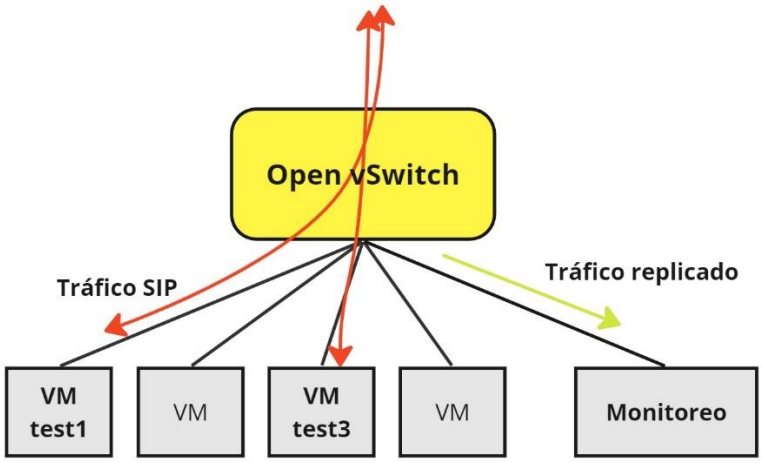


Figura 52. Situación del escenario de pruebas 2.

Si sólo queremos recibir el tráfico cursado por una VM en la máquina de análisis, el código sería el mismo que en el Escenario 1, seleccionando las interfaces deseadas.

Se va a presentar el código para el caso en el que interese analizar el tráfico de N (dos) máquinas del servidor:

```
$ovs-vsctl -- set bridge vmbr0 mirrors=@m – \
--id=@monitoring get Port tap10555i1 – \
--id=@test1 get Port tap10292i0 – \
--id=@test3 get Port tap10612i0 – \
--id=@m create mirror name=mirror1 \
select-dst-port=@test1,@test3 \
select-src-port=@test1,@test3 \
output-port=@monitoring \
-- set bridge vmbr0 mirrors=@m
```

Como podemos observar, el código es análogo, símplemente se enumeran las interfaces de interés en los comandos `select-dst-port y select-src-port` mediante comas, indicando sus ID de igual manera.

Las interfaces del OvS en juego para este caso son:

- tap10555i1: Interfaz conectada a *net1* de la VM *monitoring* (La utilizada para recibir el tráfico copiado). Con ID "@monitoring"
- tap10292i0: Interfaz conectada a *net0* de la VM *test1*, por donde intercambia el tráfico con el exterior. Con ID "@test1"
- tap10612i0: Interfaz conectada a *net0* de la VM *test3*, por donde intercambia el tráfico con el exterior. Con ID "@test3"

ANEXO H. SIPp: Proceso de instalación y escenarios.

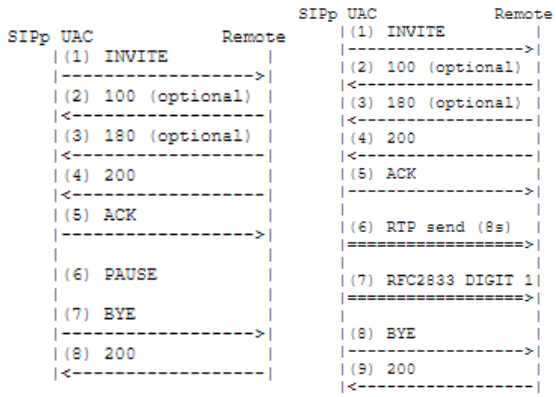
1. Instalación.

Siguiendo con lo comentado en el capítulo *4.4.1. Instalación en las máquinas*, sabemos que SIPp se proporciona en forma de código fuente. Al instalar desde consola, obtendremos un archivo comprimido con la extensión *.tar.gz* que deberemos descomprimir. Posteriormente, nos situamos en la carpeta de la propia aplicación que se ha generado al descomprimir. Allí ejecutamos el archivo *configure*, que contiene código que "parcheará" y localizará la distribución de la fuente para que se compile y se cargue en el sistema Linux local. Utilizaremos después la herramienta *make* para la creación de un fichero ejecutable o programa, para su instalación, limpieza de archivos temporales en la creación del fichero.

Código utilizado para la instalación en las VM de SIPp versión 3.4.1:

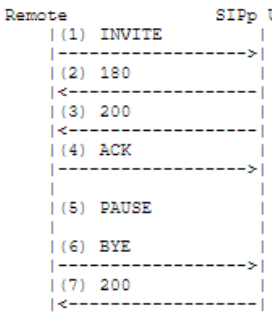
```
# wgehttps://github.com/SIPp/sipp/archive/v3.4.1.tar.gz
# tar -xvzf v3.4.1.tar.gz
# cd sipp-3.4.1
# ./configure --with-openssl
# make
# ./sipp -sn uas
```

2. Funcionamiento de escenarios básicos: uas, uac. Control de la aplicación.

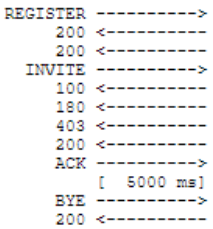


uac.xml

uac_pcap.xml



uas.xml



branchc/brachs.xml

Estos son los cuatro escenarios más básicos que nos proporciona SIPp. Todos funcionan con dos "usuarios": Servidor (*uas*) y Cliente (*uac*).

La máquina que funciona como Servidor queda en escucha de llamadas entrantes para responder con la señalización SIP correspondiente para el establecimiento de ellas.

Como podemos ver, nuestro *uas* espera recibir un mensaje INVITE que provenga de un remoto, y pasar a establecer la llamada. Enviará un mensaje *180* y *200 OK*. Posteriormente, se puede configurar una pausa, que simula la duración de las llamadas. Finalmente espera recibir un *BYE*, respondiendo con un *200 OK* para terminar la llamada.

Adicionalmente, el escenario *brachs.xml* incorpora una función más. Espera recibir un REGISTER, mensaje que se enviaría para registrarse en un servidor Proxy SIP.

Por otro lado, el Cliente (nuestro *uac*) funcionará como llamante. Será el encargado de iniciar la señalización SIP que dará lugar a las llamadas. En este caso el *uac* envía los mensajes esperados por el *uas*, y recibe la señalización procedente de él. Será el encargado de enviar el *BYE* y por tanto finalizar la llamada.

La herramienta SIPp se ejecuta desde línea de comandos, escogiendo el escenario que se desea. La máquina que actúa como Cliente (*uac*), deberá indicar la IP del Servidor (*uas*) con el que sedea comunicarse. El funcionamiento básico para que se ponga en marcha la comunicación entre cliente y servidor es el siguiente:

En el cliente: `# ./sipp -sn uac 127.0.0.1`
En el servidor: `# ./sipp -sn uas`

Podemos controlar el funcionamiento de SIPp de forma interactiva, mediante teclas (‘hot’ keys) que modifican el tráfico generado en cualquier momento. Las hot keys son:

Key	Action
+	Increase the call rate by 1 * rate_scale
*	Increase the call rate by 10 * rate_scale
-	Decrease the call rate by 1 * rate_scale
/	Decrease the call rate by 10 * rate_scale
c	Enter command mode
q	Quit SIPp (after all calls complete, enter a second time to quit immediately)
Q	Quit SIPp immediately
s	Dump screens to the log file (if -trace_screen is passed)
p	Pause traffic
1	Display the scenario screen
2	Display the statistics screen
3	Display the repartition screen
4	Display the variable screen
5	Display the TDM screen
6-9	Display the second through fifth repartition screen.

También disponemos del modo comandos, con el que se puede escribir un comando de una sola línea que indique a SIPp que realice alguna acción. El modo de comando es más versátil que las teclas de acceso rápido, pero toma más tiempo ingresar algunas acciones comunes. Están disponibles los siguientes comandos:

Command	Description
dump tasks	Prints a list of active tasks (most tasks are calls) to the error log.
set rate X	Sets the call rate.
set rate-scale X	Sets the rate scale, which adjusts the speed of '+', '-', '*', and '/'.
set users X	Sets the number of users (only valid when -users is specified).
set limit X	Sets the open call limit (equivalent to -l option)
set hide <true|false>	Should the hide XML attribute be respected?
set index <true|false>	Display message indexes in the scenario screen.
set display <main|ooc>	Changes the scenario that is displayed to either the main or the out-of-call scenario.
trace <log> <on|off>	Turns log on or off at run time. Valid values for log are "error", "logs", "messages", and "shortmessages".

Además para controlar el tráfico se pueden determinar parámetros de inicio, que se especifican a la hora de ejecutar los escenarios. Algunos de los más útiles son:

`-sn:` Usa un escenario por defecto (incluido con el ejecutable de SIPp). Si se omite esta función, el escenario estándar SipStone UAC se ejecuta.

`-sf:` Carga un archivo de escenario XML alternativo a los existentes por defecto. Para aprender más sobre la sintaxis de los escenarios XML se puede utilizar la opción -sd para mostrar escenarios incluidos. Contienen toda la ayuda necesaria.

`-i:` Establece la IP local para las cabeceras 'Contact:','Via:', y 'From:'. Por defecto es la IP primaria del host.

`-d:` Controla la duración de las llamadas. Más precisamente, controla la duración de las instrucciones de ‘pausa’ del escenario. Su valor por defecto es 0 y su unidad es milisegundos.

`-s:` Establece el username del URI de petición. Por defecto es ‘service’.

`-r:` Establece la tasa de llamadas (en llamadas por segundo). El valor por defecto es 10.

`-rp:` Establece el periodo de la tasa de llamadas. Por defecto es 1 segundo y su unidad es milisegundos. Permite realizar n llamadas cada m milisegundos (usando -r n -rp m).

`-m:` Para el test y sale de la ejecución cuando ‘calls’ llamadas son procesadas.

Por ejemplo, estaremos generando con SIPp 7 llamadas cada 2 segundos hacia un *uas* situado en la dirección IP 192.168.1.1 si ejecutamos:

```
# ./sipp -sn uac -r 7 -rp 2000 192.168.1.1
```

Para pausar el tráfico, pulsamos la tecla “p”, y para finalizarlo, la tecla “q”.

3. Escenarios personalizados.

Aunque los escenarios incluidos por defecto son útiles, se puede sacar mucho partido al configurar escenarios personalizados dependiendo de nuestras necesidades.

Los escenarios de SIPp se escriben en xml. Siempre empiezan con:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="Basic Sipstone UAC">
```

Y acaban con:

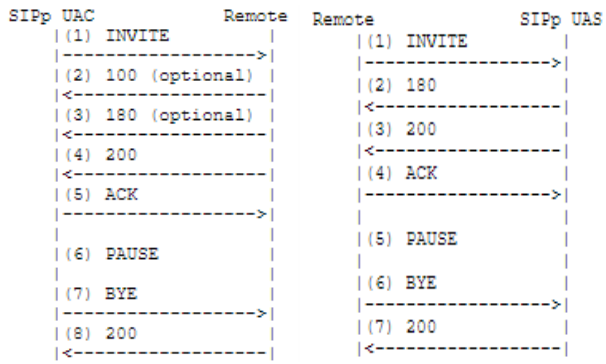
```
</scenario>
```

A conttinuación podemos ver un ejemplo de la estructura del código para enviar un mensaje INVITE (enviado por el *uac*):

```
<send>
   <![CDATA[

     INVITE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
     Via: SIP/2.0/[transport] [local_ip]:[local_port]
     From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
     To: sut <sip:[service]@[remote_ip]:[remote_port]>
     Call-ID: [call_id]
     Cseq: 1 INVITE
     Contact: sip:sipp@[local_ip]:[local_port]
     Max-Forwards: 70
     Subject: Performance Test
     Content-Type: application/sdp
     Content-Length: [len]

     v=0
     o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
     s=-
     t=0 0
     c=IN IP[media_ip_type] [media_ip]
     m=audio [media_port] RTP/AVP 0
     a=rtpmap:0 PCMU/8000

   ]]>
 </send>
```

4. Escenario 1: Prueba 1

Para las primeras pruebas con el Escenario 1, como se ha comentado en el apartado *5.1. Primer escenario: pruebas y resultados*, se utilizan los escenarios por defecto más sencillos: *uas.xml* y *uac.xml*. Por tanto, el intercambio de mensajes sigue el siguiente esquema:



El código xml de dichos escenarios no ha sido modificado para esta prueba, por lo que se utilizan los códigos básicos de SIPp, que pueden encontrarse en su página web.

Para ejecutarlos, se decide generar una llamada cada 5 segundos, con un límite de 4 llamadas efectuadas. Por ello el comando que se ejecuta es el siguiente:

```
# ./sipp -sn uas
# ./sipp -sn uac -r 1 -rp 5000 -m 4 192.168.3.224
```

5. **Escenario 1: Prueba 2**

Para las segundas pruebas de este escenario ejecutamos los mismos archivos, pero modificando los parámetros de inicio para realizar la prueba de carga pertinente. Simularemos un total de 300 llamadas cada segundo, con un límite de 1800. Ejecutamos:

```
# ./sipp -sn uas
# ./sipp -sn uac -r 300 -rp 1000 -m 1800 192.168.3.224
```

6. **Escenario 2**

En el escenario 2, la situación es diferente. Se va a intercambiar tráfico con el exterior desde dos máquinas que utilizan SIPp. Además, como queremos analizar los resultados de estas comunicaciones en los programas de monitoreo que se han explicado, se va a generar un escenario personalizado en el que se den nombres identificativos a los usuarios que se comunican en las llamadas, para simplificar y darle un sentido más realista a las pruebas. También se configurará una duración de llamadas distinta de cero, para más realismo.

Se van a ejecutar dos escenarios de Cliente en las máquinas *test1* y *test3* de manera que intercambien señalización con la máquina *test2*, en la que se ejecutará un Servidor, situada en el exterior.

Se va a presentar el código del escenario *uac_esc2.xml* que se ejecutará en la máquina *test1*, a modo de ejemplo. El código para la máquina *test3* sigue la misma forma.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">

<scenario name="uac - Second Scenario - TFG">
  <send retrans="500">
    <![CDATA[
      INVITE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
     From:test1<sip:test1@[local_ip]:[local_port]>;tag=[pid]SIPpTag00[call_number]
      To: <sip:[service]@[remote_ip]:[remote_port]>
      Call-ID: [call_id]
      CSeq: 1 INVITE
      Contact: sip:test1@[local_ip]:[local_port]
      Max-Forwards: 70
```

```
    Subject: Performance Test
    Content-Type: application/sdp
    Content-Length: [len]
    P-Asserted-Identity: <sip:test1@[local_ip]>

    v=0
    o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
    s=-
    c=IN IP[media_ip_type] [media_ip]
    t=0 0
    m=audio [media_port] RTP/AVP 0
    a=rtpmap:0 PCMU/8000

  ]]>
</send>

<recv response="100"
      optional="true">
</recv>

<recv response="180" optional="true">
</recv>

<recv response="183" optional="true">
</recv>

<recv response="200" rtd="true">
</recv>

<send>
  <![CDATA[

    ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
   From:test1<sip:test1@[local_ip]:[local_port]>;tag=[pid]SIPpTag00[call_number]
    To: <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
    Call-ID: [call_id]
    CSeq: 1 ACK
    Contact: sip:test1@[local_ip]:[local_port]
    Max-Forwards: 70
    Subject: Performance Test
    Content-Length: 0

  ]]>
</send>

<pause milliseconds="5000" />

<send retrans="500">
  <![CDATA[
```

```
      BYE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
    From:test1 <sip:test1@[local_ip]:[local_port]>;tag=[pid]SIPpTag00[call_number]
      To: <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
      Call-ID: [call_id]
      CSeq: 2 BYE
      Contact: sip:test1@[local_ip]:[local_port]
      Max-Forwards: 70
      Subject: Performance Test
      Content-Length: 0

    ]]>
  </send>

  <recv response="200" crlf="true">
  </recv>

  <!-- definition of the response time repartition table (unit is ms)   -->
  <ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200"/>

  <!-- definition of the call length repartition table (unit is ms)     -->
  <CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000"/>

</scenario>
```

Fijándonos en lo que se ha remarcado en rojo, podemos observar como estamos dando al usuario SIP el username *test1*, ya que es la máquina en la que lo ejecutamos. Además, hemos preparado una pausa de 5000 milisegundos, lo que sería la duración de las llamadas realizadas.

Y lo que se ejecutará desde las máquinas *test1* y *test3* será:

```
# ./sipp -sf uac_esc2.xml -r 1 -rp 2000 -m 1 -s test2 192.168.3.224
```

Generando una llamada cada 2 segundos, con límite de una llamada destinada al usuario *test2* situado en la IP 192.168.3.224.

Por otro lado, en la máquina *test2* se ejecutará el Servidor (*uas_esc2.xml*). No se ha modificado el *uas.xml* original ya que no es necesario para la comunicación.

ANEXO I. Heplify-Homer

1. Instalación y activación de Heplify.

Para desplegar Heplify junto con el Homer, existe una guía de instalación que se ha seguido para poner en marcha la aplicación y vincularla al servidor Homer. Este proceso se realiza dese consola en la máquina que recibe el tráfico que queremos analizar, en nuestro caso, la VM *monitoring*.
Solo es necesario instalar el Heplify Client, de la siguiente forma:

1. Obtenemos la última versión del binario Heplify client desde la página oficial del Git:

```
# cd /tmp/
# wget https://github.com/sipcapture/heplify/releases/download/1.63/heplify
```

2. Damos permisos al binario y lo copiamos en el directorio "/bin":

```
# chmod +x heplify
# cp heplify /bin/
```

3. Creamos el servicio "/etc/systemd/system/heplify.service" configurando la IP del Heplify server y la interfaz donde se va a capturar:

```
[Unit]
Description=Heplify
Requires=network.target remote-fs.target
After=network.target remote-fs.target
[Service]
Type=simple
User=root
Group=root
ExecStart=/bin/heplify -i eth1 -hs homer-db.bts.io:9061
Restart=on-failure
[Install]
WantedBy=multi-user.target
```

4. Activamos el servicio y lo lanzamos o paramos mediante los comandos:
Heplify puede estar activado (enviando paquetes al servidor) o desactivado (no envía nada).

```
# systemctl enable heplify
# systemctl start heplify
# systemctl stop heplify
```

2. Funcionamiento de HOMER.

En cuanto a HOMER, el análisis de llamadas se realiza desde su interfaz web que presenta la siguiente forma:

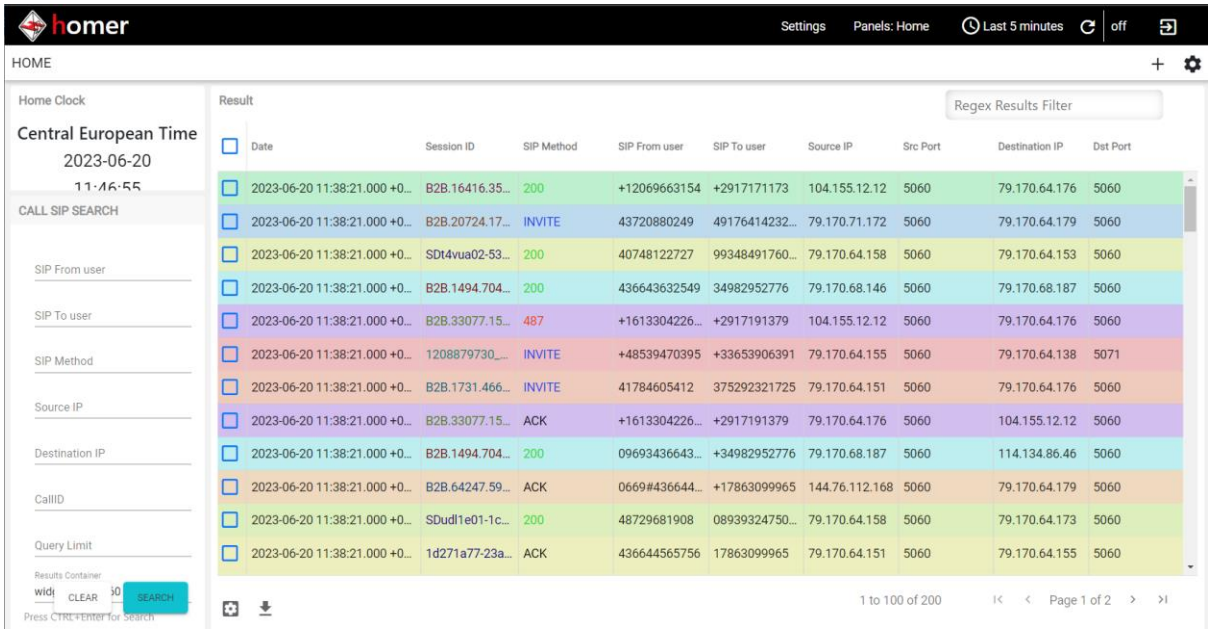


Figura 53. Vista principal de HOMER

En esta interfaz se nos presentarán la señalización proveniente de los equipos que nos interesa analizar, y que están conectados a este servidor mediante Heplify. La aplicación nos da la posibilidad de filtrar mediante varios parámetros, como son los "SIP ID" de los usuarios de la llamada, dirección IP fuente o destino, ID de la llamada, tipo de mensaje SIP capturado… y también por intervalo de tiempo: últimos 5 minutos, última hora…

Dentro de la ventana en la que se capturan los paquetes en sí, se nos presentan los IDs de los usuarios que intervienen, puerto fuente y destino, direcciones IP fuente y destino de la llamada, tipo de mensaje SIP, ID de la llamada y la fecha y hora en la que se realizó.

Además, haciendo click en algunos de estos parámetros, podemos analizar el contenido de cada mensaje SIP por dentro, ver el flujo de una llamada en concreto… y también nos permite consultar datos de la sesión, como la duración de la llamada, tiempo de timbrado o razón de finalización.

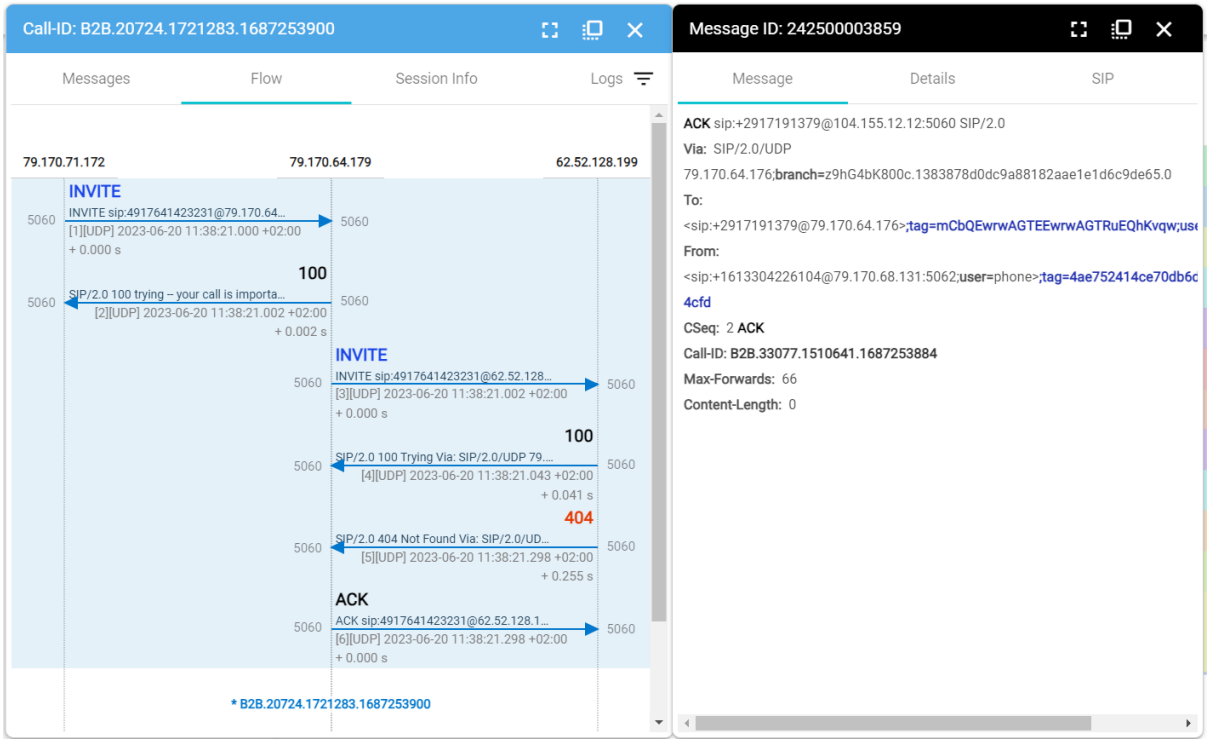


Figura 54. Ejemplo de flujo y mensaje SIP ACK

ANEXO J. Grafana: Análisis de datos.

1. Funcionamiento

Grafana es una herramienta con la que, a partir de una serie de datos recolectados, obtendremos un panorama gráfico de la situación de una máquina, empresa u organización.

Nos ofrece gran cantidad de posibilidades para visualizar las métricas que más nos interese analizar en una red. La pagina principal presenta la siguiente forma:

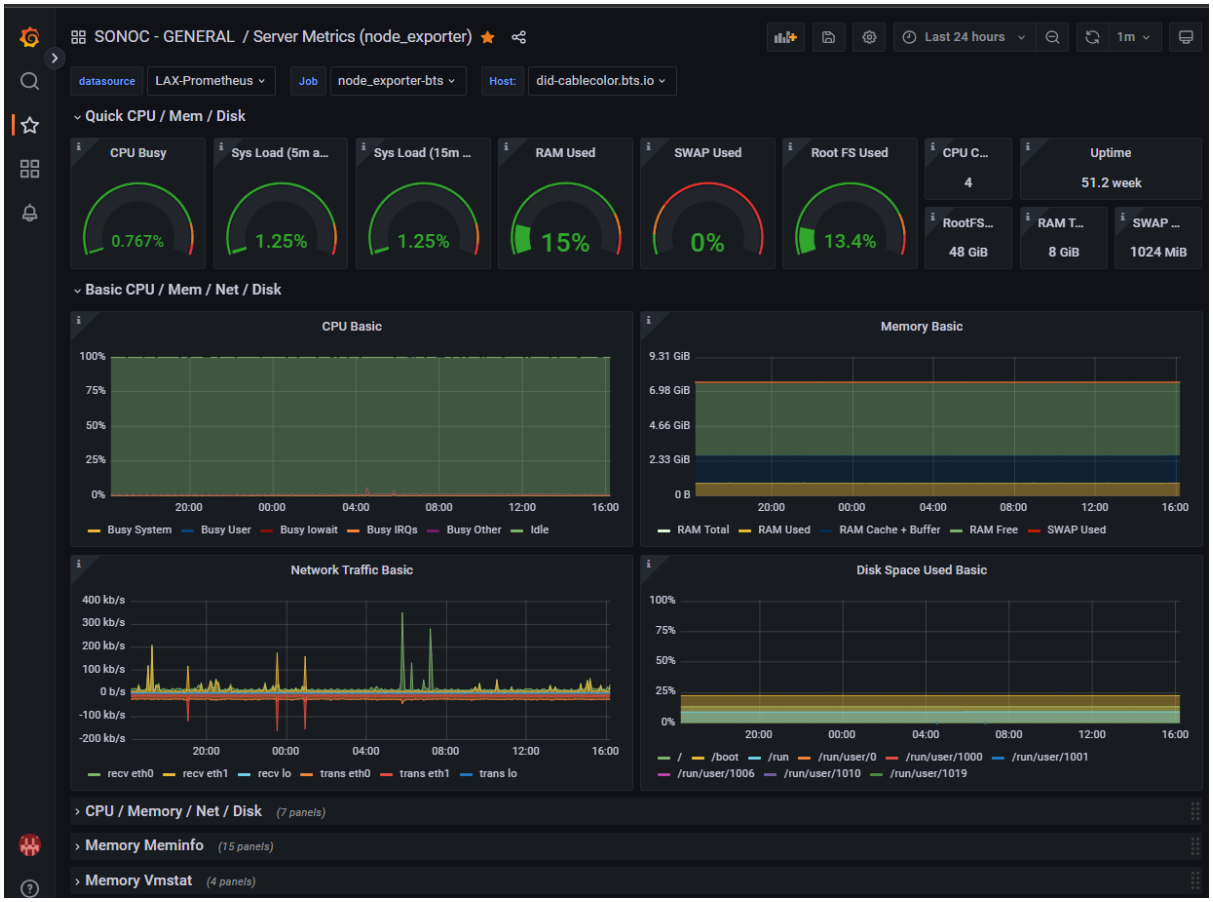
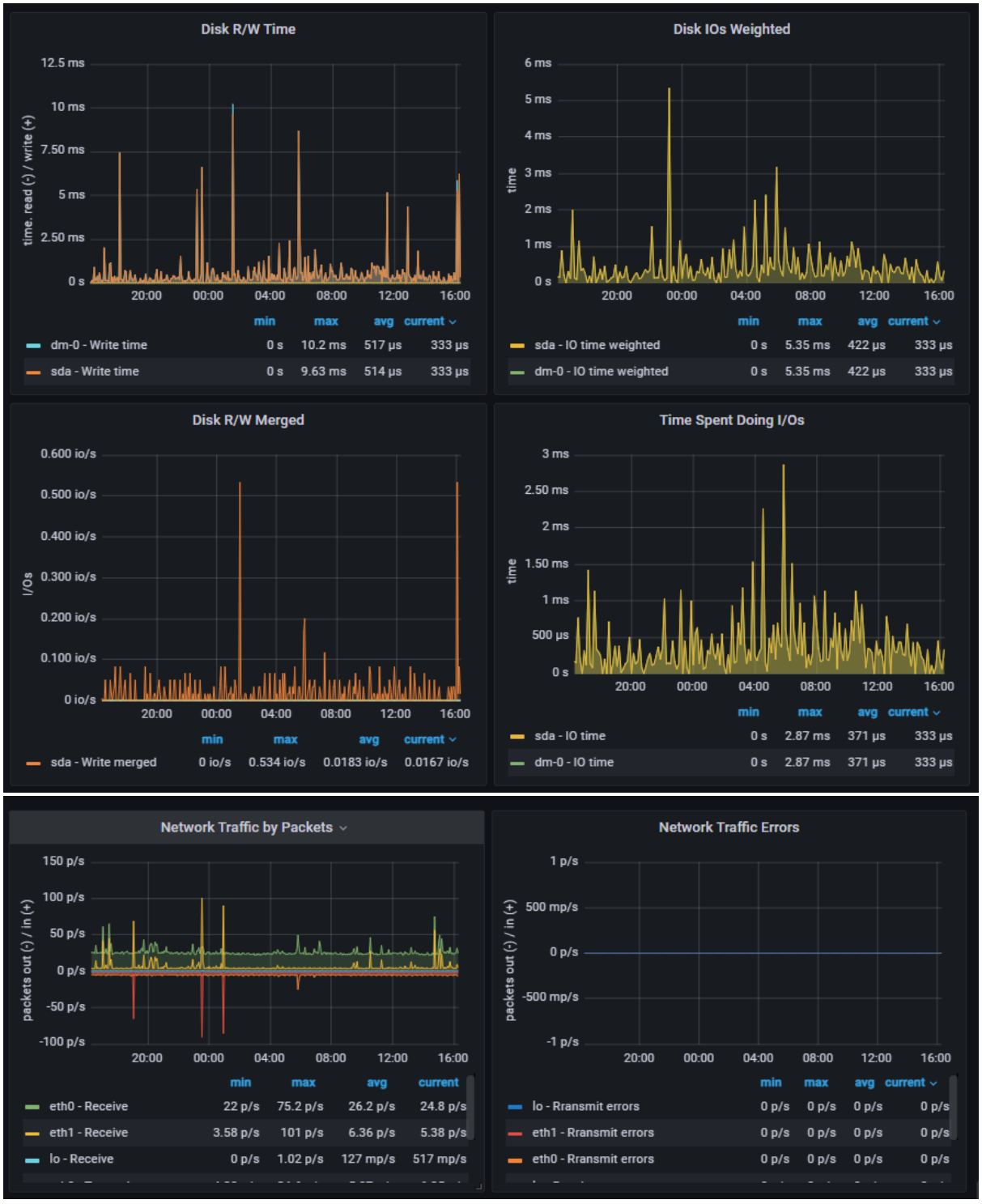


Figura 55. Interfaz de Grafana

En la parte superior, se puede escoger la máquina o host concreto que queremos analizar, que normalmente estarán organizadas por secciones o carpetas. De esta forma es muy sencillo e intuitivo para las empresas acceder a las estadísticas de los equipos que interesan.

A partir de esta página principal, se puede navegar por sus numerosos desplegables que presentan estadísticas de cualquier tipo, como pueden ser el gasto de la CPU, espacio del disco utilizado, tráfico básico en la red, procesos en el sistema, tráfico por paquetes, bytes, estado de la red en el tiempo…

Se presenta alguna gráfica interesante presentada por Grafana respecto a todo ello:



Figuras 56 y 57. Ejemplos de gráficas mostradas

2. Resultados de las pruebas realizadas

Trabajando con el escenario del A*partado 5.1*, se analiza mediante Grafana la máquina *test2*, tras hacer la prueba de carga, y podemos extraer algunas estadísticas por la gran carga de tráfico que recibe:

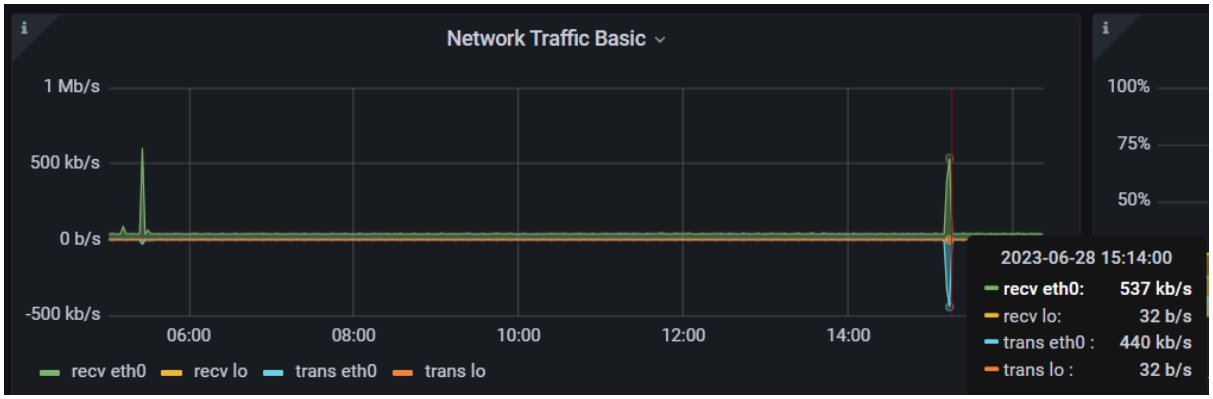


Figura 58. Tráfico en la red

Como observamos en esta gráfica, en el momento que se realizan las pruebas de carga sube el tráfico existente en la red, llegando a trabajar con 537 kb/s por el interfaz de conexión de esta máquina *test2*. Podemos ver que recibe más información de la que envía.

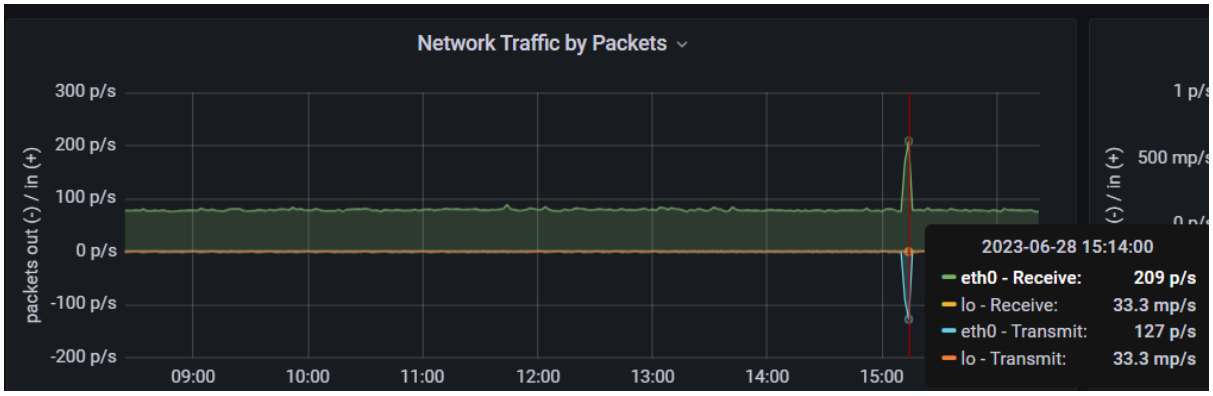


Figura 59. Tráfico en la red medido en paquetes.

Si analizamos según paquetes por segundo, podemos ver que en el momento de mayor carga se procesan 209 paquetes por segundo.

Capítulo 8: Bibliografía

[1] SONOC

Página web: https://www.sonoc.io/

[2] Protocolo SIP

https://es.wikipedia.org/wiki/Protocolo_de_iniciaci%C3%B3n_de_sesi%C3%B3n

[3] Proxmox:

Página oficial: https://www.proxmox.com/en/

Linux Bridges: https://developers.redhat.com/blog/2018/10/22/introduction-to-linux-interfaces-for-virtual-networking#bonded_interface

Máquinas virtuales: https://cutt.ly/RwuRDbgs

[4] Open vSwitch

Página oficial: https://www.openvswitch.org/

Manual para Proxmox: https://pve.proxmox.com/wiki/Open_vSwitch

Configuración: https://docs.openvswitch.org/en/latest/faq/configuration/

Port Mirroring: https://arthurchiao.art/blog/traffic-mirror-with-ovs/

[5] SIPp

Página oficial: https://sipp.sourceforge.net/

Instalación/documentación (html): https://sipp.sourceforge.net/doc/reference.html

[6] Heplify – Homer

Documentación Heplify: https://github.com/sipcapture/heplify

Documentación HOMER: https://github.com/sipcapture/homer

Instalación:https://drive.google.com/file/d/1Y_XakDwJHb5O1sUV9mEaORfD34kKbumu/view?usp=sharing

[7] Grafana

Página oficial: https://grafana.com/

Capítulo 9: Glosario de términos y siglas.

VoIP: Voz sobre IP.
ToIP: Telefonía sobre IP.
SIP: Session Initiation Protocol.
OvS: Open vSwitch.
RTP: Real-Time Transport Protocol.
VM: Máquina virtual.
ARM: Acorn RISC Machines.
MIPS: Microprocessor without Interlocked Pipeline Stages.
RTCP: RTP Control Protocol.
RTCPXR: RTP Control Protocol Extended Reports.
DNS: Sistema de nombres de dominio.
Trunk SIP: Intermediario entre los sistemas telefónicos y un servicio de telefonía por Internet.
LXC: Linux containers.
SCSI: Small Computer System Interface.
SDN: Software defined network.
SPAN: Switched port analyzer.
RSPAN: Remote switched port analyzer.