CONTROLADORAS REDUNDANTES SDN EN LA PLATAFORMA TEMONET FASE II – FCI 011 DE LA UNIVERSIDAD DE GUAYAQUIL

REDUNDANT SDN CONTROLLERS ON THE TEMONET EDGE COMPUTING PLATFORM PHASE II – UNIVERSITY OF GUAYAQUIL

Jenny Arizaga Gamboa [1*]

[1] Carrera Tecnologías de la Información, Facultad Ciencias Matemáticas y Físicas, Universidad de Guayaquil. Ecuador. ORCID: https://orcid.org/0000-0002-2098-9077. Correo: jenny.arizagag@ug.edu.ec

Eduardo Alvarado Unamuno [2]

[2] Carrera Tecnologías de la Información, Facultad Ciencias Matemáticas y Físicas, Universidad de Guayaquil. Ecuador. ORCID: https://orcid.org/0000-0001-6145-7926. Correo: eduardo.alvaradou@ug.edu.ec

*** Autor para correspondencia: jenny.arizagag@ug.edu.ec**

Resumen

La plataforma TEMONET, basada en Edge Computing y SDN, requiere que su elemento de control esté siempre operativo para mantener múltiples conexiones simultáneas y ofrecer un servicio eficiente a sus usuarios. Este proyecto tiene como objetivo diseñar e implementar controladores redundantes SDN bajo el esquema activo/activo utilizando herramientas de software libre como distribuciones de LINUX, Ryu Controller y hardware adicional de bajo costo como Raspberry Pi 3 B+. Se implementó un clúster de controladores que garantizaron una conexión estable para los usuarios finales mediante el protocolo OpenFlow y Open vSwitch. Los resultados demuestran que el prototipo cumple con la redundancia y la alta disponibilidad, proponiendo una solución económica y viable para implementaciones empresariales.

Palabras clave: Redes Definidas por Software (SDN); Controladoras SDN; Tolerancia a Fallas; Edge Computing

Abstract

The TEMONET platform, based on Edge Computing and SDN, requires its control element to be always operational to maintain multiple simultaneous connections and offer an efficient service to its users. This project aims to design and implement redundant SDN controllers under the active/active scheme using free software tools such as LINUX distributions, Ryu Controller and additional low cost hardware such as

Raspberry Pi 3 B+. A cluster of controllers was implemented to ensure a stable connection for end users using OpenFlow and Open vSwitch protocol. The results demonstrate that the prototype complies with redundancy and high availability, proposing an economical and viable solution for enterprise deployments.

Keywords: *Software-Defined Network (SDN); Edge Computing; Fault Tolerance; SDN Controllers*

Fecha de recibido: 21/09/2024
Fecha de aceptado: 11/11/2024
Fecha de publicado: 13/11/2024

Introducción

El avance tecnológico de las últimas décadas ha cambiado drásticamente la forma en que se prestan y consumen servicios de información, la creciente necesidad de ancho de banda y capacidad de procesamiento ha fomentado la adopción de tecnologías como el Cloud Computing y el Edge Computing. El Cloud Computing ha revolucionado el almacenamiento y procesamiento de datos, permitiendo a las organizaciones utilizar recursos de computación bajo demanda y reducir costos operativos. Sin embargo, la latencia causada por el envío de datos a centros de datos remotos puede ser un obstáculo para aplicaciones que requieren procesamiento en tiempo real. En este contexto, el Edge Computing surge como una solución complementaria, acercando el procesamiento de datos a la fuente de generación y reduciendo significativamente la latencia.

En el ámbito de la salud, estas tecnologías han abierto nuevas oportunidades para el desarrollo de plataformas que faciliten el acceso a terapias y tratamientos médicos (Rodríguez et al., 2023). TEMONET es una plataforma de terapias médicas basada en contenidos audiovisuales, diseñada para ofrecer opciones de tratamiento a pacientes con deficiencias cognitivas como la dislexia. La plataforma utiliza Edge Computing para proporcionar un servicio eficiente y accesible, permitiendo a los pacientes recibir tratamiento desde sus hogares.

Las Redes Definidas por Software (SDN) ofrecen una solución avanzada para la gestión de redes, separando el plano de control del plano de datos y permitiendo una administración centralizada y flexible. Los controladores SDN, que actúan como el cerebro de la red, son cruciales para el funcionamiento eficiente de la plataforma. Sin embargo, la falta de un sistema de administración automatizado para estos controladores puede convertirse en un cuello de botella, afectando la eficiencia operativa y la resiliencia del servicio.

En este contexto, el proyecto de controladoras redundantes SDN en la plataforma TEMONET Fase II - FCI 011 de la Universidad de Guayaquil se propone como una solución innovadora para mejorar la administración y sincronización de los controladores SDN, asegurando alta disponibilidad y un servicio ininterrumpido. Utilizando herramientas de programación de código abierto como Ryu, Raspbian, Ubuntu Server, Django y Open vSwitch, el proyecto busca desarrollar un software que permita gestionar las controladoras de manera eficiente, reducir la carga operativa y mejorar la experiencia del usuario.

Este proyecto tiene como objetivo diseñar e implementar controladores redundantes SDN bajo el esquema activo/activo utilizando herramientas de software libre como distribuciones de LINUX, Ryu Controller y hardware adicional de bajo costo como Raspberry Pi 3 B+.

Materiales y métodos

Definición y Conceptos Clave

El Edge Computing es una arquitectura informática emergente que acerca el procesamiento de datos y los servicios de almacenamiento a los puntos donde se generan los datos, en lugar de depender de centros de datos centralizados. Esto reduce la latencia, ahorra ancho de banda y mejora la eficiencia del procesamiento de datos en tiempo real. Edge Computing se ha convertido en una solución clave para aplicaciones que requieren respuestas inmediatas, como Internet de las Cosas (IoT) (Shi et al., 2016), automóviles autónomos, y plataformas de salud como TEMONET.

El Edge Computing se basa en el principio de distribuir recursos de computación y almacenamiento más cerca de los dispositivos finales (edge devices) que generan los datos. A diferencia del Cloud Computing tradicional, donde los datos deben viajar a través de la red hasta un centro de datos centralizado para ser procesados, Edge Computing realiza el procesamiento en el borde de la red (edge), es decir, cerca de la fuente de datos.

Beneficios de Edge Computing

La computación de borde almacena y procesa datos en dispositivos de borde sin depender de plataformas en la nube. Este modelo presenta importantes ventajas en varios ámbitos clave (Cao et al., 2020):

- Reducción de Latencia: Al procesar los datos localmente, Edge Computing minimiza el tiempo necesario para enviar datos a un centro de datos centralizado y recibir respuestas, lo cual es crucial para aplicaciones en tiempo real.
- Ahorro de Ancho de Banda: Procesar datos en el borde reduce la cantidad de datos que necesitan ser transmitidos a través de la red, disminuyendo el consumo de ancho de banda.
- Mejora de la Seguridad y Privacidad: Los datos sensibles pueden ser procesados localmente, reduciendo el riesgo de exposición durante la transmisión y cumpliendo con regulaciones de privacidad.
- Escalabilidad y Flexibilidad: Permite una expansión más sencilla y flexible de la infraestructura, ya que los recursos pueden ser añadidos en ubicaciones específicas según la demanda.

Arquitectura de Edge Computing

A continuación se describe la arquitectura general actual de la computación de borde como se muestra en la Fig. 1. Describimos los componentes de la arquitectura e introducimos una red informática de borde heterogénea de tres niveles, en la que la primera capa es la capa de cosas, la segunda capa es la capa de borde y la tercera capa es la capa de nube, cuyos componentes son se describe en detalle a continuación(Liu et al., 2022).

Capa de usuario

También conocida como capa de usuario, está formada por diversos equipos terminales, dispositivos de realidad aumentada, cámaras de vigilancia, sensores de salud inteligentes, etc.

Capa de borde

La capa de borde es la parte media de esta arquitectura de tres capas, ubicada en el borde de la red y consta de una gran cantidad de nodos de borde. Por lo tanto, sus componentes de hardware generalmente incluyen enrutadores, puertas de enlace, conmutadores, puntos de acceso, estaciones base, servidores perimetrales específicos, etc. Desde el aspecto de la composición del software, se realizan principalmente las siguientes funciones:

1) El subsistema de enrutamiento puede realizar el reenvío de datos y la conexión de red y soportar la virtualización de redes.
2) El subsistema abierto de capacidades proporciona una interfaz API y middleware de plataforma para realizar la invocación de capacidades de red.
3) El subsistema de gestión de plataforma realiza principalmente control, planificación y programación de infraestructura, informes estadísticos de información de facturación y otras funciones.

Capa de nube

La capa de nube tiene poderosas capacidades de procesamiento y almacenamiento de datos. La tendencia actual de desarrollo de la capa de nube se basa en el núcleo de la tecnología de computación en la nube y la capacidad de la computación de borde, formando una plataforma de nube elástica integral con capacidades de computación, red, almacenamiento, seguridad y otras en el borde.

Edge Computing con SDN

Edge Computing implica acercar los recursos computacionales a los dispositivos finales, impulsado por el uso generalizado de IoT y dispositivos pequeños. La computación en nube tradicional no siempre es adecuada para estas necesidades. A pesar de su potencial, Edge Computing se enfrenta a varios retos técnicos que deben abordarse para su adopción generalizada. Se sugiere el uso de redes definidas por software (SDN) para superar estos retos. SDN es también una tendencia tecnológica creciente, aunque todavía en evolución. Entre las principales ventajas de integrar SDN con Edge Computing se incluyen un mayor control, flexibilidad, innovación, facilidad de implementación, adaptabilidad, interoperabilidad, ahorro de costes y una mayor movilidad de las máquinas virtuales (VM) (Baktir et al., 2017).

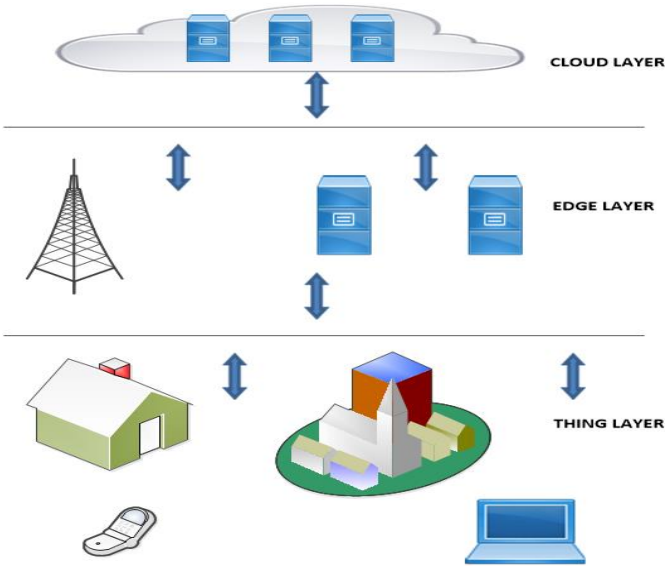

Figura 1. Arquitectura de Edge Computing.

Redes SDN

Últimamente SDN se ha convertido en uno de los temas más populares en el ámbito de las TIC. Sin embargo, al ser un concepto nuevo, aún no se ha llegado a un consenso sobre su definición exacta. De hecho, en los últimos años han surgido muchas definiciones diferentes, cada una de las cuales tiene sus propios méritos. En esta sección, primero presentamos una definición generalmente aceptada de SDN, luego describimos un conjunto de beneficios y desafíos clave de SDN y, finalmente, presentamos un modelo de referencia de SDN como base de este documento de estudio (Cox et al., 2017).

Definición de SDN

La Open Networking Foundation (ONF) es un consorcio sin fines de lucro dedicado al desarrollo, estandarización y comercialización de SDN. ONF ha proporcionado la definición más explícita y mejor recibida de SDN es de la siguiente manera:

Las redes definidas por software (SDN) son una arquitectura de red emergente donde el control de la red está desacoplado del reenvío y es directamente programable(Xia et al., 2015).

Modelo de referencia SDN

La ONF ha sugerido un modelo de referencia para SDN, como se ilustra en la Fig. 2 , este modelo consta de tres capas, a saber, una capa de infraestructura, una capa de control y una capa de aplicación, apiladas unas sobre otras.

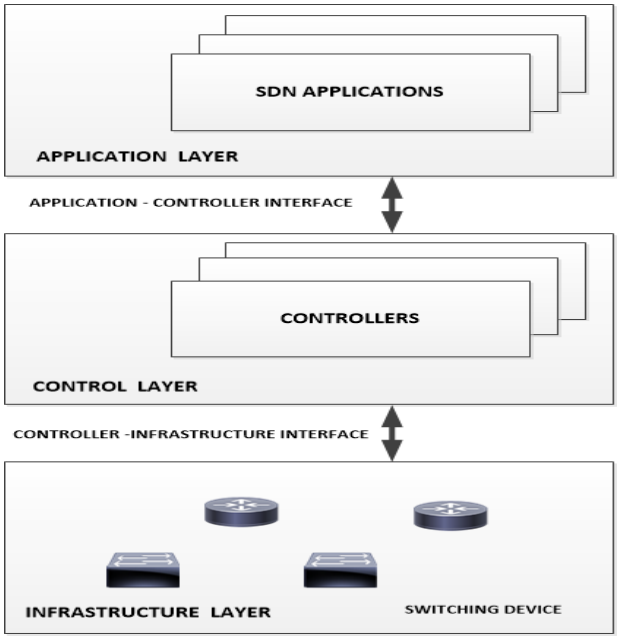

Figura 2. Arquitectura de SDN.

La **capa de infraestructura** consta de dispositivos de conmutación (por ejemplo, conmutadores, enrutadores, etc.) en el plano de datos. Las funciones de estos dispositivos de conmutación son principalmente dobles. En primer lugar, son responsables de recopilar el estado de la red, almacenarlo temporalmente en dispositivos locales y enviarlo a los controladores. El estado de la red puede incluir información como topología de la red, estadísticas de tráfico y usos de la red. En segundo lugar, son responsables de procesar paquetes según reglas proporcionadas por un controlador.

La **capa de control** une la capa de aplicación y la capa de infraestructura, a través de sus dos interfaces. Para interactuar hacia abajo con la capa de infraestructura (es decir, la interfaz con destino al sur), especifica funciones para que los controladores accedan a las funciones proporcionadas por los dispositivos de conmutación. Las funciones pueden incluir informar el estado de la red e importar reglas de reenvío de paquetes. Para interactuar hacia arriba con la capa de aplicación (es decir, la interfaz con destino al norte), proporciona puntos de acceso a servicios en varias formas, por ejemplo, una interfaz de programación de aplicaciones (API) (Cornelio et al., 2016). Las aplicaciones SDN pueden acceder a la información del estado de la red reportada desde los dispositivos de conmutación a través de esta API, tomar decisiones de ajuste del sistema basadas en esta información y llevar a cabo estas decisiones estableciendo reglas de reenvío de paquetes a los dispositivos de conmutación que utilizan esta API. Dado que existirán múltiples controladores para un gran dominio de red administrativa, también será necesaria una interfaz de comunicación "este-oeste" entre los controladores para que los controladores compartan información de la red y coordinen sus procesos de toma de decisiones.

La **capa de aplicación** contiene aplicaciones SDN diseñadas para cumplir con los requisitos del usuario. A través de la plataforma programable proporcionada por la capa de control, las aplicaciones SDN pueden acceder y controlar dispositivos de conmutación en la capa de infraestructura. Ejemplos de aplicaciones SDN podrían incluir control de acceso dinámico, movilidad y migración fluida, equilibrio de carga de servidores y virtualización de redes.

Dado que el enfoque del presente trabajo es la capa de control, explicaremos más a detalle la misma.

Capa de control

Como se ilustra en la Fig. 2, la capa de control une la capa de aplicación y la capa de infraestructura. Primero presentamos un diseño lógico para la capa de control SDN, que consta de cuatro componentes principales, a saber, un lenguaje de alto nivel, un proceso de actualización de reglas, un proceso de recopilación del estado de la red y un proceso de sincronización del estado de la red, como se ilustra en la Fig. 3.

Diseño del controlador

El controlador es el componente más importante de la arquitectura SDN, donde reside la complejidad (Xia et al., 2015), la arquitectura propuesta, como se muestra en la Fig.3, se basa en dos principios, que incluyen:

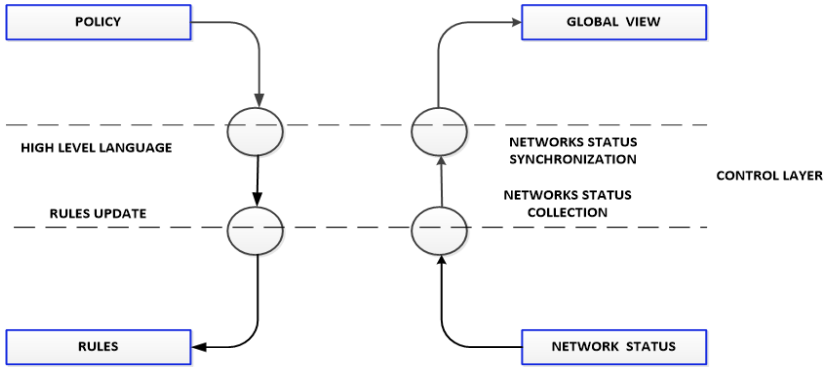

Fig. 3. Diseño Lógico de un controlador

Objetos: El controlador SDN trata con dos tipos de objetos. Uno se utiliza para el control de la red, incluidas las políticas impuestas por la capa de aplicación y las reglas de reenvío de paquetes para la infraestructura. El otro está relacionado con el monitoreo de la red, en formato de estado de la red local y global (Fonseca et al., 2019). De ello se deduce que la arquitectura lógica tiene dos flujos de información contra direccionales, como se ilustra en la Fig. 3. En el flujo descendente, el controlador traduce la política de la aplicación en reglas de reenvío de paquetes, con respecto al estado de la red. La principal preocupación de este proceso es garantizar la validez y coherencia de las reglas de reenvío. En el flujo ascendente, el controlador sincroniza el estado de la red recopilado de la infraestructura para la toma de decisiones sobre la red.

Interfaces: El controlador SDN tiene dos interfaces. La interfaz con destino al sur, que está marcada como interfaz controlador-infraestructura en la Fig. 2, se ocupa de las transacciones con la capa de infraestructura, es decir, recopila el estado de la red y actualiza las reglas de reenvío de paquetes a los dispositivos de conmutación en la capa de infraestructura en consecuencia. La interfaz con destino al norte, que está marcada como interfaz de controlador de aplicación en la Fig. 2, maneja transacciones con la capa de aplicación, es decir, recibe políticas descritas en lenguajes de alto nivel desde aplicaciones SDN y proporciona una vista global sincronizada.

Aprovechando estos principios arquitectónicos, el diseño lógico de los controladores SDN se puede desacoplar en cuatro componentes (Xia et al., 2015): un lenguaje de alto nivel, un proceso de actualización de reglas, un proceso de recopilación del estado de la red y un proceso de sincronización del estado de la red. En esta subsección, adoptamos esta estructura para explicar los esfuerzos de investigación existentes en el diseño de controladores en los grupos antes mencionados.

a) Lenguaje de Alto Nivel

Una de las funciones clave del controlador es traducir los requisitos de la aplicación en reglas de reenvío de paquetes. Esta función dicta un protocolo de comunicación (por ejemplo, un lenguaje de programación) entre la capa de aplicación y la capa de control. Un enfoque sencillo es adoptar algunos lenguajes de configuración comunes, por ejemplo, la interfaz de línea de comandos (CLI) para el sistema operativo Cisco Internetwork (IOS). Sin embargo, estos lenguajes de configuración comunes sólo ofrecen abstracciones primitivas derivadas de las capacidades del hardware subyacente. Dado que están diseñados para configuraciones de hardware, normalmente son inadecuados para adaptarse al estado de la red dinámica y con estado. Además, son propensos a errores y exigen un esfuerzo adicional en el proceso de programación. Por lo tanto, es imperativo proporcionar un lenguaje de alto nivel para que las aplicaciones SDN interactúen con los controladores. Un lenguaje de alto nivel de este tipo debería adoptar una sintaxis expresiva y completa para que las aplicaciones SDN describan fácilmente sus requisitos y estrategias de gestión de red.

b) Actualización de reglas

Un controlador SDN también es responsable de generar reglas de reenvío de paquetes que describen las políticas e instalarlas en los dispositivos de conmutación adecuados para su funcionamiento. Al mismo tiempo, las reglas de reenvío en los dispositivos de conmutación deben actualizarse debido a cambios de configuración y control dinámico, como dirigir el tráfico de una réplica a otra para el equilibrio de carga dinámico, la migración de máquinas virtuales y Recuperación de la red después de un fallo inesperado. En presencia de dinámica de red, la coherencia es una característica básica que la actualización de reglas debe preservar para garantizar operaciones de red adecuadas y propiedades de red preferidas, como ausencia de bucles, agujeros negros y seguridad.

c) Colección de estado de la red

En el flujo ascendente, los controladores recopilan el estado de la red para crear una vista global de una red completa y proporcionan a la capa de aplicación la información necesaria, por ejemplo, un gráfico de topología de la red, para decisiones de operación de la red. Un estado principal de la red son las estadísticas

de tráfico, como el tiempo de duración, el número de paquetes, el tamaño de los datos y el ancho de banda compartido de un flujo. Normalmente, la recopilación del estado de la red funciona de la siguiente manera. Cada dispositivo de conmutación recopila y almacena estadísticas de tráfico local dentro de su propio almacenamiento. Estas estadísticas de tráfico local pueden ser recuperadas por los controladores (es decir, un modo “pull”) o reportadas proactivamente a los controladores (es decir, un modo “push”). Diferentes modos y estrategias tienen diferentes características en términos de precisión y gastos generales de medición. De esta manera, un objetivo clave de la investigación es encontrar un "punto óptimo" (es decir, un punto óptimo) con una precisión adecuada pero manteniendo una baja sobrecarga de medición.

d) Sincronización del estado de la red

Delegar el control a un controlador centralizado puede provocar un cuello de botella en el rendimiento del controlador centralizado. Una solución común para superar este cuello de botella es implementar múltiples controladores que actúen como controladores iguales, de respaldo o replicados. Mantener una visión global consistente entre todos los controladores es esencial para garantizar las operaciones adecuadas de la red. Los estados inconsistentes o obsoletos pueden provocar que la capa de aplicación tome decisiones incorrectas, lo que luego conduce a operaciones inapropiadas o subóptimas de la red.

Tolerancia a fallos en el plano de control

La tolerancia a fallos es crucial para la resiliencia de la red, garantizando una alta disponibilidad y fiabilidad en los sistemas. Las redes definidas por software (SDN) presentan nuevos retos y oportunidades para desarrollar estrategias y arquitecturas de tolerancia a fallos (Rehman et al., 2019). La resiliencia del plano de control es fundamental, ya que el controlador debe ser capaz de procesar todos los comandos de tráfico necesarios en cualquier situación. Se proponen varios enfoques, como replicar el controlador en una red diferente o integrar mecanismos de autorreparación frente a ataques dirigidos.

Las arquitecturas multicontrolador en SDN pueden ser planas/horizontales, en las que todos los controladores comparten las mismas responsabilidades, o jerárquicas/verticales, en las que los controladores tienen diferentes responsabilidades en función de su capa. Las arquitecturas planas ofrecen más resistencia pero son más difíciles de gestionar, mientras que las arquitecturas jerárquicas simplifican la gestión. Ambas arquitecturas pretenden mejorar la latencia de la conmutación por error y deben responder eficientemente a las peticiones de conmutación por error sin afectar al rendimiento (Blial et al., 2016)(Pashkov et al., 2014).

De controlador único a controlador múltiple

En la etapa inicial del diseño de SDN, un único controlador gestiona toda la red, en la figura4, el controlador (c1) gestiona cuatro conmutadores de la red. Cuando el host de origen envía un nuevo paquete al conmutador (s1), el conmutador no puede realizar la función de reenvío debido a la falta de información de enrutamiento del nuevo paquete. Entonces, el conmutador (s1) envía mensajes (Packet-in) al controlador (c1) para obtener el enrutamiento del nuevo paquete, después de recibir el mensaje de respuesta del controlador, el conmutador reenvía el paquete al siguiente dispositivo. Finalmente, el paquete llega al host de destino con éxito. El controlador desempeña un papel importante en el proceso de transmisión de tráfico. Desafortunadamente, a medida que el tráfico de la red aumenta rápidamente, un único controlador no puede manejar una gran

cantidad de solicitudes de flujo enviadas desde los conmutadores debido a la capacidad limitada del controlador. Mientras tanto, una vez que el controlador único falla, los conmutadores no pueden planificar el enrutamiento para el paquete recién llegado, lo que afecta la comunicación y las aplicaciones de la red. En consecuencia, es necesario proponer un diseño de controlador moderno (Hu et al., 2018).

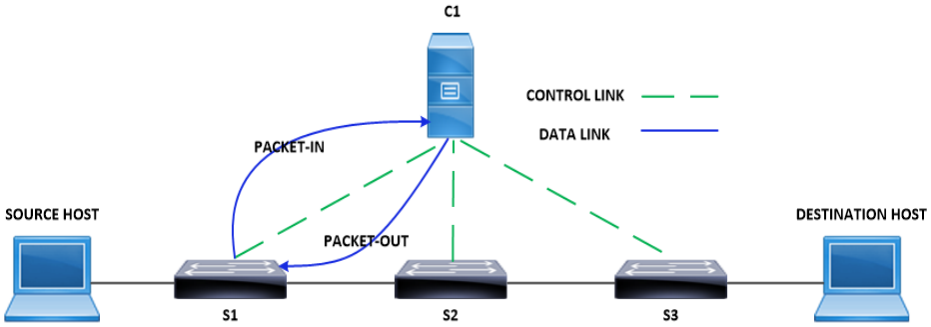

Figura 4. Red SDN con controlador único.

El desarrollo de OpenFlow, en particular la versión 1.2, introdujo controladores maestros, esclavos e iguales, permitiendo que un conmutador se conectara a varios controladores. Este avance llevó a la adopción de múltiples controladores en el diseño de SDN para abordar las limitaciones de un único controlador. En la topología de red mostrada en la Fig. 5, dos controladores (c1 y c2) gestionan diferentes segmentos de la red. Comparten la misma lógica de forma centralizada, lo que permite a ambos controladores instalar directamente rutas de reenvío en todos los conmutadores cuando llegan nuevos paquetes. Esto reduce la carga de procesamiento de flujos en un único controlador y proporciona redundancia, resolviendo el problema de un único punto de fallo.

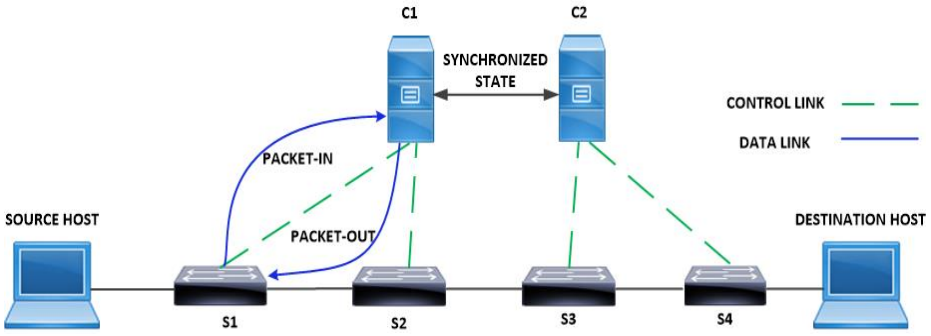

Figura 5. Red SDN con múltiples controladores

Con base en el análisis anterior, podemos descubrir que, en comparación con un solo controlador, un diseño con múltiples controladores puede mejorar de manera efectiva el rendimiento de la red SDN. Por lo tanto, el diseño con múltiples controladores se ha convertido gradualmente en una investigación popular en los últimos años.

Controladores SDN

Los controladores SDN son componentes críticos en la arquitectura de SDN, actuando como el cerebro de la red. Varios controladores populares incluyen:

- Ryu: Un controlador SDN basado en componentes, escrito en Python, que soporta múltiples versiones de OpenFlow y ofrece una amplia gama de APIs para la creación de aplicaciones de red.
- OpenDaylight: Una plataforma de controladores SDN basada en Java, ampliamente utilizada en la industria, que soporta diversos protocolos de red y ofrece una gran flexibilidad.
- ONOS (Open Network Operating System): Un controlador SDN de alto rendimiento y escalabilidad, diseñado para redes de operadores y proveedores de servicios.

Carga del controlador

Para la carga del controlador se debe sumar las contribuciones de todos los switches para todos los destinos (Cuba Espinoza & Becerra Ávila, 2016). Sea ARR(d, c) el arrival rate de paquetes al controlador con respecto al destino d generado por el switch c. Entonces, el arrival rate total al controlador es igual a la suma:

$$ARR_{TOT} = \sum_{C=1}^{C} \sum_{d=1}^{N} ARR(d,c) \ (1)$$

Es decir, el arrival rate total es la suma de N · C arrival process independientes, cada uno de baja intensidad. Entonces, el agregado de todos estos procesos puede aproximarse (con gran precisión) a un proceso de Poisson con media:

$$E\{ARR_{TOT}\} = N.C.E_d\{ARR(d)\} = \left(\frac{N^2}{S}\right) \cdot E_d\{ARR(d)\}(2)$$

Denominemos ARR_{MAX} al número máximo de arrivals por segundo que el controlador puede procesar. El tiempo de servicio del controlador es entonces $1 / \llbracket ARR \rrbracket \_MAX$. La distribución (pdf) de este tiempo de servicio depende de los procesos siendo ejecutados, y es probablemente cercano a un valor constante. Sin embargo, consideraremos que este tiempo de servicio tiene una distribución exponencial para obtener una aproximación conservadora del comportamiento (retardo, perdida) de los paquetes que llegan al controlador con una restricción de tiempo de respuesta.

Sea T_{REPLY} el tiempo máximo de espera de la respuesta de un controlador. Es decir, el tiempo de espera para que el Switch obtenga la información necesaria para empezar a transmitir los paquetes de una nueva sesión. La probabilidad que una consulta al controlador tome más de este tiempo, puede ser hallada considerando al controlador como una cola M/M/1/m donde m (el tamaño del buffer) es igual a $T_{REPLY} * ARR_{MAX}$. En este caso, la probabilidad buscada es equivalente a la probabilidad de pérdida de paquete:

$$\rho = \frac{(1-\rho) \cdot \rho^m}{(1-\rho^m)} (3)$$

Donde:

$$\rho = \frac{E\{ARR_{TOT}\}}{ARR_{MAX}} (4)$$

Resultados y discusión

Propuesta Tecnológica

En la figura 6 se muestra el esquema en el que se basa el proyecto TEMONET, en la nube se encuentra las dos controladoras redundantes SDN, los equipos Open vSwitch se conectan a ambas controladoras con la finalidad de garantizar alta disponibilidad del servicio. De la misma manera el servidor web implementado en la nube aloja el software, encargado de la administración de las controladoras redundantes, permitiendo gestionar controladoras, aplicaciones y usuarios, incluyendo las solicitudes de los equipos de borde (Raspberry pi4).

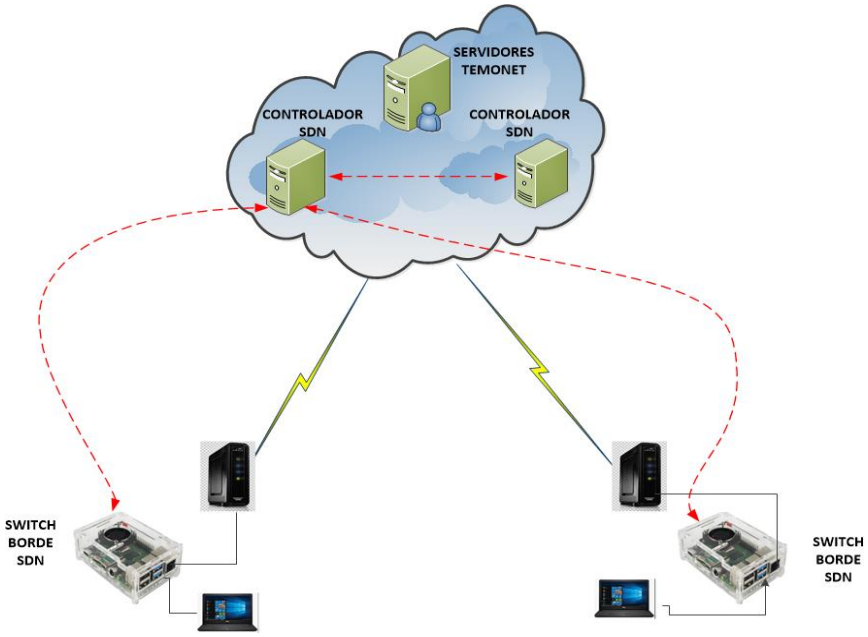

Figura 6. Modelo propuesto para TEMONET

Un Raspberry PI4 fue utilizado como un Switch virtual el cual se encuentra apuntando a las controladoras para brindar conexión a los usuarios finales hacia los servicios del proyecto TEMONET.

Controlador SDN: Ryu

Ryu es un controlador SDN modular y flexible, ideal para entornos de desarrollo e investigación. Ofrece compatibilidad con varias versiones de OpenFlow y proporciona APIs bien definidas para facilitar la integración y el desarrollo de aplicaciones de red (Asadollahi & Sameer, n.d.).

- Compatibilidad con OpenFlow: Soporta versiones 1.0, 1.2, 1.3, 1.4 y 1.5 de OpenFlow.
- APIs Southbound: Incluye NETCONF, OF-CONFIG, OVSDB y OpenFlow.

- Integración con Sistemas de Gestión de Nube: Incluye un plugin para OpenStack Neutron, permitiendo la gestión de redes virtualizadas.
- Soporte para Linux: Basado en Python y utilizando APIs REST, facilita la integración y despliegue en entornos Linux.

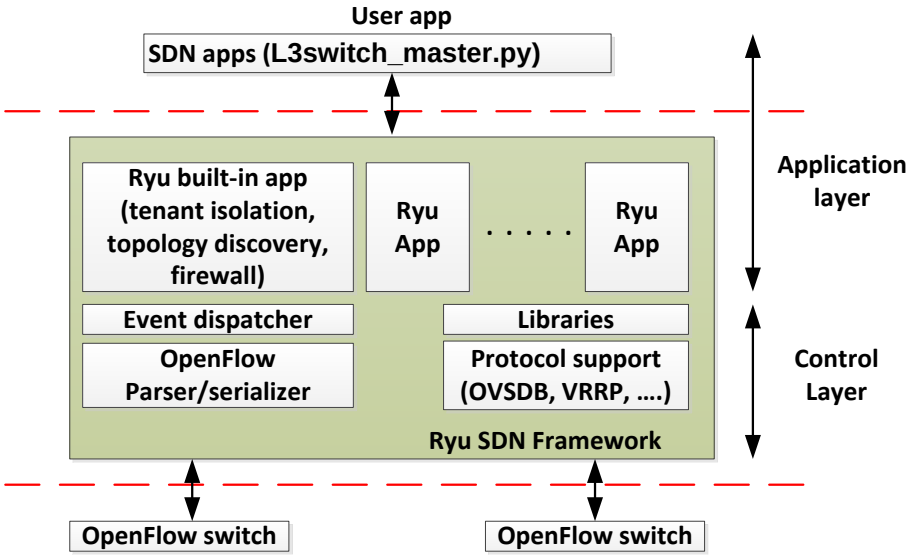

Figura 7. Arquitectura del controlador RYU para TEMONET

Para la comunicación entre el sistema gestor y las controladoras se hizo uso del protocolo SSH, por medio del cual se ejecutan las aplicaciones. Como se muestra en la figura 8.

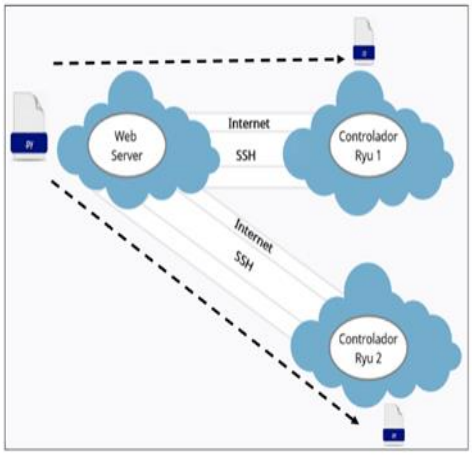

Figura 8. Comunicación entre sistema gestor y las controladoras.

El rol de los controladores ryu para este proyecto es MASTER/SLAVE, este tipo de rol de controlador permite el acceso completo al switch OpenFlow, significa que el controlador MASTER será responsable de administrar el plano de datos de flujo abierto del conmutador. El switch enviará los mensajes de control solo al controlador MASTER.

Cuando se ejecuta la API l3switch_master.py desarrollada para este proyecto, en ambos controladores se puede verificar el flujo de paquetes (direcciones físicas o MAC de los equipos conectados y prioridad) como se muestra en la figura 9.

```
root@controller1:/home/ryu1/multicontroller# ryumanager l3switch_master.py
Registered vcs backend: git
Registered vcs backend: hg
Registered vcs backend: svn
Registered vcs backend: bzr
loading app l3switch_master.py
loading app ryu.controller.ofp.handler
instantiating app l3switch.master.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
This Controller is master
packet in 963351283619 14:dd:a9:bd:b0:90 ff:ff:ff:ff:ff:ff 2
packet in 963351283619 14:dd:a9:bd:b0:90 33:33:ff:00:00:01 2
packet in 963351283619 44:82:e5:64:11:c2 14:dd:a9:bd:b0:90 1
```

Figura 9. Ejecución de la aplicación l3switch_master.py en el controlador Ryu

En el switch OVS, ejecutando el comando ovs-vsctl list controller se puede observar las características de los dos controladores, en el parámetro is_connected: TRUE se verifica la conexión exitosa y en role se evidencia el primero como master y el segundo como slave, como se observa en la figura 10.

```
root@raspberrypi:/home/pi# ovs-vsctl list controller
_uuid                         : a805224c-5572-4bca-9453-b36b20b747cd
connection_mode               : []
controller_burst_limit        : []
controller_rate_limit         : []
enable_async_messages         : []
external_ids                  : {}
inactivity_probe              : []
is_connected                  : true
local_gateway                 : []
local_ip                      : []
local_netmask                 : []
role                          : master
status                        : {last_error="Connection refused",
sec_since_connect="120", sec_since_disconnect="128", status=ACTIVE}
targed                        : "tcp:52.177.2.248:6633"

_uuid                         : 33e1b9f1-dd2c-418f-83da-853ca23ffcec
connection_mode               : []
controller_burst_limit        : []
controller_rate_limit         : []
enable_async_messages         : []
external_ids                  : {}
inactivity_probe              : []
is_connected                  : true
local_gateway                 : []
local_ip                      : []
local_netmask                 : []
role                          : slave
status                        : {last_error="Connection refused",
sec_since_connect="127", sec_since_disconnect="135", status=ACTIVE}
targed                        : "tcp:20.81.224.226:6633"
```

Figura 10. Rol de controladoras Master/Slave.

Para comprobar la redundancia entre los controladores se procede a cancelar la ejecución de la API en un controlador (fallo del controlador) y se verifica que el switch OVS siga conectado al otro controlador, además que el equipo cliente mantiene conectividad. Se valida los parámetros de acuerdo a la figura 11.

ryu 1 (controlador 1):

- parámetro is_connected: FALSE

- role: other

ryu 2 (controlador 2):

- parámetro is_connected: TRUE
- role: slave

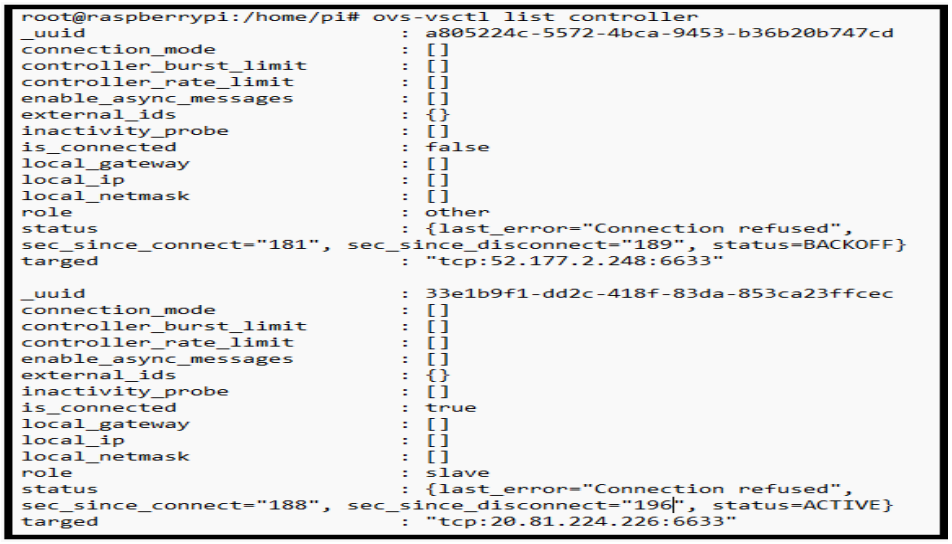

Figura 11. Rol de los controladores en caso de fall.

Conclusiones

Los avances tecnológicos han transformado significativamente la prestación de servicios de información. La computación en nube ofrece un almacenamiento y procesamiento de datos flexible, pero adolece de problemas de latencia que la computación de borde mitiga procesando los datos más cerca de su origen, lo que mejora el rendimiento en tiempo real. En el sector sanitario, estas tecnologías permiten el desarrollo de plataformas innovadoras como TEMONET, que proporciona terapia cognitiva a través de contenidos audiovisuales, aprovechando la computación de borde para una mayor eficiencia y accesibilidad.

La integración de SDN con edge computing ofrece numerosas ventajas, como un mejor control, flexibilidad, innovación y ahorro de costes. Las redes definidas por software (SDN) mejoran la gestión de la red al desacoplar los planos de control y datos, lo que permite una administración centralizada. Sin embargo, sin una gestión automatizada, los controladores SDN pueden convertirse en cuellos de botella, afectando a la eficiencia operativa y a la capacidad de recuperación. Para hacer frente a estas limitaciones, los diseños multicontrolador mejoran el rendimiento y proporcionan redundancia. El proyecto TEMONET utiliza una configuración maestro/esclavo con controladores Ryu para gestionar los conmutadores OpenFlow, lo que garantiza una alta disponibilidad del servicio.

En general, la integración de edge computing con controladores SDN mejora la eficiencia, resistencia y experiencia de usuario de TEMONET, proporcionando una solución robusta para aplicaciones sanitarias en tiempo real. Este trabajo también ofrece una perspectiva actual sobre la adopción de SDN en redes empresariales, industriales y académicas. Además, puede servir de referencia para orientar los futuros

esfuerzos de investigadores y profesionales de la industria encaminados a hacer avanzar las SDN, especialmente en lo que respecta a sus numerosas aplicaciones, tanto actuales como futuras, en el contexto de las tecnologías emergentes. Además, este estudio aborda las necesidades inmediatas de investigación para mejorar las redes tradicionales mediante el uso de SDN.

Referencias

Asadollahi, S., & Sameer, M. (n.d.). *Ryu Controller ' s Scalability Experiment on Software Defined Networks*.

Baktir, A. C., Ozgovde, A., & Ersoy, C. (2017). How Can Edge Computing Benefit from Software-Defined Networking: A Survey, Use Cases, and Future Directions. *IEEE Communications Surveys and Tutorials*, *19*(4), 2359–2391. https://doi.org/10.1109/COMST.2017.2717482

Blial, O., Ben Mamoun, M., & Benaini, R. (2016). An Overview on SDN Architectures with Multiple Controllers. *Journal of Computer Networks and Communications*, *2016*. https://doi.org/10.1155/2016/9396525

Cao, K., Liu, Y., Meng, G., & Sun, Q. (2020). An Overview on Edge Computing Research. *IEEE Access*, *8*, 85714–85728. https://doi.org/10.1109/ACCESS.2020.2991734

Cornelio, O. M., Gulín, J. G., Fonseca, B. B., & Ching, I. S. (2016). Experiencia en la evaluación de competencias en un sistema de laboratorios a distancia. Anais do Encontro Virtual de Documentação em Software Livre e Congresso Internacional de Linguagem e Tecnologia Online,

Cox, J. H., Chung, J., Donovan, S., Ivey, J., Clark, R. J., Riley, G., & Owen, H. L. (2017). Advancing software-defined networks: A survey. *IEEE Access*, *5*, 25487–25526. https://doi.org/10.1109/ACCESS.2017.2762291

Cuba Espinoza, G. J., & Becerra Ávila, J. M. A. (2016). Diseño e implementación de un controlador SDN/openflow para una red de campus académica. *Pontificia Universidad Católica Del Perú*.

Fonseca, B. B., Benitez, L. C. M., & Oliva, Á. M. H. (2019). La estructura de desglose del trabajo como mecanismo viable para la generación de proyectos exitosos. Serie Científica de la Universidad de las Ciencias Informáticas, 12(5), 63-75. https://dialnet.unirioja.es/servlet/articulo?codigo=8590151

Hu, T., Guo, Z., Yi, P., Baker, T., & Lan, J. (2018). Multi-controller Based Software-Defined Networking: A Survey. *IEEE Access*, *6*, 15980–15996. https://doi.org/10.1109/ACCESS.2018.2814738

Liu, B., Luo, Z., Chen, H., & Li, C. (2022). A Survey of State-of-the-art on Edge Computing: Theoretical Models, Technologies, Directions, and Development Paths. *IEEE Access*, *10*, 54038–54063. https://doi.org/10.1109/ACCESS.2022.3176106

Pashkov, V., Shalimov, A., & Smeliansky, R. (2014). Controller failover for SDN enterprise networks. *SDN and NFV: Next Generation of Computational Infrastructure - 2014 International Science and Technology Conference - Modern Networking Technologies, MoNeTec 2014, Proceedings*, *1*. https://doi.org/10.1109/MoNeTeC.2014.6995594

Rehman, A. U., Aguiar, R. L., & Barraca, J. P. (2019). Fault-tolerance in the scope of Software-Defined

Networking (SDN). *IEEE Access*, *7*, 124474–124490. https://doi.org/10.1109/ACCESS.2019.2939115

Rodríguez , E., Mar , O., García , A. L., & Bron, B. (2023). Herramientas computacionales para el apoyo al diagnóstico de pacientes con Parkinson: una revisión sistemática. Revista Cubana de Ciencias Informáticas, 17(3). https://search.ebscohost.com/login.aspx?direct=true&profile=ehost&scope=site&authtype=crawler&jrnl=19941536&asa=N&AN=174090841&h=oWb7fa3MZdlxf5NmmLO7dDMFsVm4ewat%2BUVzDn2WlWcua7KGCklPBh8kaQVP1a%2FFy%2FqcC4UfWLvPrn%2Bg1XK2Qw%3D%3D&crl=c

Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, *3*(5), 637–646. https://doi.org/10.1109/JIOT.2016.2579198

Xia, W., Wen, Y., Foh, C. H., Niyato, D., & Xie, H. (2015). A Survey on Software-Defined Networking. *IEEE Communications Surveys and Tutorials*, *17*(1), 27–51. https://doi.org/10.1109/COMST.2014.2330903