UNIVERSIDAD NACIONAL AGRARIA DE LA SELVA

FACULTAD DE INGENIERÍA EN INFORMÁTICA Y SISTEMAS
ESCUELA PROFESIONAL DE INGENIERÍA EN INFORMÁTICA Y SISTEMAS

INFORME DE TESIS

EVALUACIÓN DEL RENDIMIENTO DE UNA RED AVANZADA TRADICIONAL Y UNA RED AVANZADA SDN

Para obtener el Título Profesional de Ingeniero en Informática y Sistemas

Autor: Juan Ramos Estela

Asesor: Mg. William Rogelio Marchand Niño (UNAS-Perú)

Co-Asesor: M. en C. José Ignacio Castillo Velásquez (UACM-México)

Lugar de ejecución: Universidad Nacional Agraria de la Selva

Grupo de investigación: Redes Seguridad y Gestión de TI

Programa de Investigación: Tecnologías de Información (TI)

Línea de Investigación: Redes e Infraestructura de TI

Duración: Del 11 de Mayo de 2020 al 17 de Diciembre del 2021

Tingo María – Perú

Mayo, 2021

Universidad Nacional Agraria de la Selva Facultad de Ingeniería en Informática y Sistemas	**ACTA DE SUSTENTACIÓN DE TESIS N.º 02-2021**	**COMISION DE GRADOS Y TITULOS** **Fecha:** 05/mayo/2021

PARTE 1. FASE INICIAL

Siendo las **19:30 horas del día 05 de mayo de 2021**; en la Sala Virtual Cisco Webex de la FIIS, se instala el jurado calificador conformado por:

Jurado 1: Ing. Pedro Crisólogo Trujillo Natividad (Presidente)

Jurado 2: Mg. Christian García Villegas

Jurado 3: Mg. Gardyn Olivera Ruiz

Oficializado mediante **RESOLUCIÓN N° 016-2021-D-FIIS-UNAS** del 08 de marzo de 2021, para el proceso de sustentación del informe final de Tesis del bachiller **Juan RAMOS ESTELA,** titulado: **"EVALUACIÓN DEL RENDIMIENTO DE UNA RED AVANZADA TRADICIONAL Y UNA RED AVANZADA SDN".** ASESOR**: Mg. William Rogelio Marchand Niño,** COASESOR**: M. en C. José Ignacio Castillo Velázquez (UACM-México).**
Se manifiesta que el bachiller cumple con los requisitos exigidos de Ley y se le invita a disertar su Tesis por espacio de 30 minutos, asimismo se dispondrá de igual tiempo para la absolver preguntas y sugerencias.

PARTE 2. FASE DE PREGUNTAS Y RESULTADO

Culminada la exposición se inicia la fase de preguntas por parte del jurado calificador; también se invita a los asistentes a formular preguntas sobre el tema de Tesis.
Absueltas todas las peticiones, el jurado calificador procede a deliberar en privado la calificación y resultado.
Concluida la deliberación y en presencia del público, el jurado calificador anuncia que el resultado de la Sustentación de Tesis es: **APROBADO POR UNANIMIDAD**
(NOTA: consignar una de la siguientes: DESAPROBADO, APROBADO POR MAYORIA o APROBADO POR UNANIMIDAD)
Con calificativo de: EXCELENTE
(NOTA: consignar una de la siguientes: EXCELENTE, MUY BUENO, BUENO, DEFICIENTE, MUY DEFICIENTE)
Por lo que se comunicará a las instancias correspondientes para el trámite respectivo.

PARTE 3. CONFORMIDAD

De todo lo mencionado se firma al pie en señal de conformidad, siendo las 21:22 horas se da por finalizada la ceremonia de Sustentación de Tesis.

Firma: 	Firma: 	Firma: 
Jurado 1: Pedro Crisólogo Trujillo Natividad	**Jurado 2:** Christian Garcia Villegas	**Jurado 3:** Gardyn Olivera Ruiz
Firma: 	Firma: 	Firma: 
Sustentante: Juan Ramos Estela	**Asesor:** William Rogelio Marchand Niño	**Co-Asesor:** José Ignacio Castillo Velázquez

DEDICATORIA

A Dios, principalmente por permitirme la vida brindándome salud y sabiduría para lograr llegar hasta este momento muy importante de mi vida y formación profesional.

A mis queridos padres Segundo Manuel Ramos Campos y María Yolanda Estela Alarcón, quienes han sido el pilar fundamental de quien soy como persona y profesional, además de su apoyo incondicional constante, siempre estaré orgulloso de tenerlos.

A mis hermanos Leoncio Ramos Estela y Cristian Max Ramos Estela por estar siempre presente y brindarme su apoyo en diferentes sentidos y demás hermanos por alentarme a seguir adelante.

AGRADECIMIENTOS

A Dios por la virtud de vivir con su bendición y permitirme tener una familia unida y ejemplar durante toda mi vida.

A mi Asesor Mg. William Rogelio Marchand Niño un especial agradecimiento por todo el tiempo y todos sus conocimientos impartidos durante mi etapa universitaria y durante la elaboración de esta tesis, por su confianza, apoyo, consejos y orientaciones constantes; mis reconocimientos por ser un excelente profesional, calidad de persona y gran amigo.

Al M. en C. José Ignacio Castillo Velázquez, asesor de este trabajo de investigación, una excelente profesional, gran persona y amigo, quien a pesar de la distancia siempre se dio tiempo de estar pendiente del avance de esta investigación a través de sus orientaciones y conocimientos para lograr un mejor resultado.

A la Universidad Nacional Agraria de la Selva (UNAS), mi alma mater, y a todos los docentes de la Facultad de Ingeniería en Informática y Sistemas por haber sido parte de mi formación profesional y alentarme a cumplir mis objetivos.

Al Laboratorio de Redes y Seguridad de la Facultad por permitirme ser parte durante mucho tiempo donde pude desarrollar muchas habilidades profesionales, gracias al jefe del Laboratorio William Marchand Niño y las personas que forman parte de él.

A T.Y.M.C. por haberme acompañado siempre, y por estar presente en todo momento motivándome a seguir con mis objetivos.

A mis amigos, Gian Jara, Jhon Cántaro, Ana Terrones, Ángel Cerna, Fitzgerald Ortiz, Thalía Díaz y demás amigos y colegas de la promoción 2010 de la FIIS de la UNAS por su apoyo y buenos deseos para cumplir este objetivo.

ABREVIATURAS

SDN	Red Definida por Software
HDN	Red Definida por Hardware
OSPFv3	Abrir el camino más corto primero Versión 3
MB	MegaBytes
Mbps	Megabites por segundo
Ms	Milisegundos
HTTP	Protocolo de Transferencia de Hipertexto.
VLC	Video Red Cliente
CPU	Unidad Central de Procesamiento
RAM	Memoria de Acceso Aleatorio
VM	Máquina Virtual
ICMPv6	Protocolo de Mensajes de Control de Internet Versión 6
TCP	Protocolo de Control de Transmisión
UDP	Protocolo de Datagrama de Usuario

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Resumen

La presente investigación denominada **"Evaluación del rendimiento de una red avanzada tradicional y una red avanzada SDN"** se enfoca en el estudio de la arquitectura de red avanzada de Chile (REUNA) con el fin de determinar su rendimiento en base a dos tipos de implementaciones, una de manera tradicional "Red Definida por Hardware" (HDN) y otra de manera programable "Red Definida por Software" (SDN). Este estudio identifica cómo se relaciona dicha implementación con el rendimiento de la red, considerando el retardo, la pérdida de paquetes y el nivel de consumo de recursos computacionales. La primera fase consistió en emular la topología REUNA, donde, en HDN se implementó de manera tradicional mediante la herramienta GNS3 y en SDN se programó mediante las herramientas Mininet, Quagga y Ryu; Asimismo, se realizó diversas pruebas con el fin de obtener datos basados en el rendimiento de la red. La segunda fase consistió en probar la hipótesis "La relación que existe entre el rendimiento y el tipo de implementación de una arquitectura de red avanzada es inversamente proporcional"; los resultados entre HDN vs SDN indican que hay un retardo de transmisión (6 183.30700 vs 16.47689 ms), un retardo de procesamiento de (0.02157 vs 0.00006 ms) y una variación de retardo de (20.05345 vs 0.01110 ms) respectivamente, en la pérdida de paquetes tenemos una cantidad (231 vs 1 paquetes perdidos) respectivamente; y de igual manera tenemos que el nivel de consumo de recursos computacionales en procesador es de (42.69 vs 7.08 %), en memoria es de (3 876.29 vs 898.41 MB) y en ancho de banda es de (2.96 vs 13 977.98 Mbps) respectivamente; por tanto se aceptó la hipótesis, esto debido a que el rendimiento depende inversamente de la cantidad de instancias de control y la forma en que se ha implementado y al mismo tiempo es proporcional, esto debido a que con una sola instancia (SDN) la red es más eficiente ya que hay menor tiempo de retardo, menor pérdida paquetes y menos consumo de recursos computacionales que en una red con dieciocho instancias de control (HDN); esto se resume en que si se añade más instancias de control tendremos menor rendimiento de red. Los resultados indican que, en esta emulación, la implementación de forma programable (SDN) es más eficiente que la implementación de manera tradicional (HDN); en efecto el rendimiento se relaciona de manera inversamente proporcional con el tipo de implementación que se realice, considerando la cantidad de instancias de control.

Palabras Clave: OpenFlow, SDN, HDN, programable, tradicional, OSPFv3, evaluación, rendimiento, protocolo, arquitectura de red avanzada, Mininet, Ryu, Quagga, controlador.

Abstract

The present research, entitled, “The Evaluation of the Yield of a Traditional Advanced Network and an SDN Advanced Network,” focuses on the study of the architecture of Chile’s advanced network (REUNA – acronym in Spanish) with the purpose of determining its yield, based on two types of implementations: one of a traditional fashion, “hardware-defined networking,” (HDN), and the other, of a programable fashion, “software-defined networking,” (SDN). This study identifies how the said implementations are related to the network yield, considering the delay, the loss of packets, and the level of computational resource consumption. The first phase consisted of emulating the REUNA topology, where the HDN was implemented in a traditional fashion through the GNS3 tool and the SDN was programmed using the Mininet, Quagga and Ryu tools. At the same time, diverse testing was done with the goal of obtaining data based on the yield of the network. The second phase consisted in testing the hypothesis: “The relationship that exists between the yield and the type of implementation of the architecture of an advanced network is inversely proportional.” The results between the HDN vs the SDN indicate that there is a transmission delay of (6183.30700 vs 16.47689 ms), a processing delay of (0.02157 vs 0.00006 ms) and the delay varies by (20.05345 vs 0.01110 ms), respectively; for the loss of packets the quantity was (231 vs 1 packets lost), respectively. In the same manner, the level of computational resource consumption of the processor was (42.69 vs 7.08 %), for the memory it was (3 876.29 vs 898.41 MB) and for the bandwidth it was (2.96 vs 13 977.98 Mbps), respectively; thus, the hypothesis was accepted, due to the fact that the yield inversely depends on the number of control points and the form in which they have been implemented, and at the same time, it is proportional due to the fact that in just one instance (SDN) the network was more efficient, since it had less delay time, less packet loss and less consumption of computational resources than the red with eighteen control points (HDN). This can be resumed in that, if more control points are added, the network would have a lower yield. The results indicate that for this emulation, the implementation of a programable format (SDN) is more efficient than the implementation of a traditional fashion (HDN); effectively, the yield is related in an inversely proportional manner with the type of implementation that is used, considering the quantity of control points.

Keywords: OpenFlow, SDN, HDN, programmable, traditional, OSPFv3, evaluation, yield, protocol, advance network architecture, Mininet, Ryu, Quagga, controller.

Introducción

Las redes IP actualmente tienen contexto con base a la implementación de tecnologías de información que son desplegadas en entornos de red para dar soporte y brindar conectividad en los servicios que se brinda a los diferentes usuarios e investigaciones en el ámbito científico y educativo, donde surgen las redes avanzadas. A medida que las innovaciones tecnológicas se incrementan, lo ideal es que el funcionamiento de las redes IP evolucionen para dar una mejor conectividad y soporte, pero esto no se ha visto desarrollado de manera muy evidente ya que las redes siguen operando de forma tradicional y su funcionamiento no ha cambiado. Sin embargo, aproximadamente desde el año 1990 se ha incorporado un nuevo paradigma denominado redes definidas por software (SDN), el cual ha tomado mayor fuerza en el año 2011 cuando el consorcio industrial ONF (*Open Networking Foundation*) se creó con el fin de promover y liderar la evolución de las SDN estandarizando los elementos de esta arquitectura SDN, lo cual aún no se ha implementado de forma generalizada debido a que las infraestructuras de red en las distintas organizaciones del estado, empresas privadas y universidades operan de forma tradicional y los dispositivos de red con los que cuentan no soportan los protocolos que demandan las SDN.

El objeto de esta investigación es evaluar la relación que existe entre el rendimiento de una arquitectura de red avanzada y el tipo de implementación, para la elaboración de una guía de implementación eficiente, esto se realizará mediante una comparación en entornos emulados de dos implementaciones para la arquitectura de red avanzada que será una de manera tradicional (HDN) y otra de manera programable (SDN), para ello pueden existir diferentes factores a considerar, entre los cuales se evaluará el retardo de red, pérdida de paquetes en la red, y el consumo de recursos computacionales.

El contenido de esta tesis consta de cinco capítulos: En el capítulo I se describe la formulación del problema de donde se deriva tanto el problema general como los específicos, la justificación e importancia, alcance y limitaciones, los antecedentes como trabajos de investigación que guardan similitud con esta tesis y los objetivos tanto general como específicos en relación con el rendimiento de una arquitectura de red avanzada. El capítulo II hace referencia toda la revisión bibliográfica, las bases teóricas y el marco conceptual. El capítulo III engloba la emulación de la arquitectura de red avanzada REUNA implementada como HDN y SDN, previas instalaciones y configuraciones de las herramientas, por mencionar algunas: GNS3, Mininet, Quagga, Ryu. El capítulo IV se refiere a la metodología,

donde se describe el tipo y nivel de investigación, la población y muestra y nuestras hipótesis tanto general como específicas, también se describe nuestras variables de investigación y su operacionalización, así mismo se describe cuáles son las herramientas que permitirán la recolección de nuestros datos tales como Wireshark, Iperf, y otras propias del sistema operativo como Monitor del sistema, Ping, Tracert; así mismo mediante los cuales se da pase a la validación de nuestras hipótesis. En el capítulo V se realiza la interpretación de los resultados, así como la discusión en base a nuestros antecedentes.

I. Planteamiento del Estudio

1.1. Marco referencial del Problema: Problemática

Los conocimientos que engloba las redes tradicionales son abordados entre los años 1960 y 1970, donde se crea Internet por la Agencia de Proyectos Avanzados del Departamento de Defensa Norteamericano (DARPA); es aproximadamente en el año 1990 cuando Internet requiere una red casi exclusivamente académica, así lo manifiesta Serrano (2015); esto debido a que las diversas investigaciones requerían nuevas características en la red; según el artículo realizado por Feamster, Rexford & Zegura (2014), se indica que esto era un proceso lento de estandarización que conllevaba a probar diversos laboratorios para ser llevados a la IETF, y prueba de ello es IPv6.

Todos los conocimientos y habilidades de administración de red son parte fundamental en los profesionales relacionados a las tecnologías de información, los cuales generalmente son adquiridos durante el proceso de desarrollo profesional; también es preciso mencionar que las diversas plataformas de aplicaciones que son de utilidad para los usuarios operan sobre una infraestructura de red, y es un ingeniero o especialista en redes el que se encuentra con la necesidad de contar con redes más flexibles, con facilidad de administración y control; razón por la cual se debe adquirir conocimientos para dar funcionalidad a las redes definidas por software de manera que estas sean programables y permita tener una administración de red eficiente.

En consideración con lo mencionado es que nace el paradigma de las SDN cerca del año 1990, así lo indica García (2016), esto se da cuando se empieza a permitir la programación de la red, la cual fuese impulsada por DARPA (*Defense Advanced Research Projects Agency*). Las SDN aparecen como una solución para ejercer una mejor administración de las redes, tal es así que, Maistre (2012) menciona que, en el evento SDN & *OpenFlow World Congress*; *Google* se impresionó de los beneficios que puede tener la programación de la red como característica principal de las SDN, ya que influiría en el alcance internacional de la red de datos; en base a esto es que en esta investigación se plantea como pregunta clave, ¿una red programable tiene mejor rendimiento que una red tradicional?; teniendo en cuenta como referente esta pregunta nace el interés de estudiar una arquitectura de red avanzada que sea implementada de manera tradicional, el cual es definida como HDN en comparativa con otra que sea implementada mediante la programación, el cual es definida como SDN; esto con el objetivo de poder determinar cuál de estas

implementaciones es la mejor solución de funcionamiento considerando el rendimiento de una arquitectura de red avanzada.

En la actualidad las redes de comunicación funcionan bajo el modelo TCP/IP definidas en la RFC 793 (1981), por lo que las redes tradicionales no han sufrido cambios, según (Diego Kreutz, 2015) "En las redes de datos tradicionales las funcionalidades de los planos de datos y control siempre han estado integradas en el mismo dispositivo físico", el administrador de red reconoce las características de operación de un dispositivo por las funcionalidades que vienen del fabricante, y no es una preocupación diferenciar cuáles son las funcionalidades relacionadas al plano de datos o el plano de control.

El contexto actual de las redes de arquitectura avanzada tradicional tiene lugar en el sector científico, investigación y educación en todo el mundo, según REDCLARA (2020) "Las redes avanzadas se dedican exclusivamente a la educación, innovación e investigación"; por parte de las redes definidas por software existen organizaciones tecnológicas que ofrecen estas soluciones, entre ellas destacan *Google*, la O*NF*, la Universidad *Stanford, Cisco, Cyan, NetCracker, Comptel, Telco Systems, HP, Juniper, Nakina Systems.*

Las SDN básicamente están orientadas a optimizar el funcionamiento de la red, y nacen por la necesidad de facilitar la administración y un mejor control de estas a través del *software*, es por lo que esta investigación tiene como objetivo principal realizar una comparación del rendimiento entre las redes tradicionales y las redes programables, que permita a las redes avanzadas continuar con investigaciones provechosas para el desarrollo global.

Las redes avanzadas tradicionales operan con gran ancho de banda y deben ser fluidas, según la investigación realizada por Castillo Velázquez, Ramírez Díaz & Marchand Niño (2019) la monitorización de las redes avanzadas puede demandar el uso recursos entre los que considera el ancho de banda, CPU y memoria de acuerdo con los protocolos que se utiliza, además que actualmente la configuración de enrutamientos debe ser OSPF, lo cual es una característica a considerar en las redes avanzadas ya que tienen una conectividad aislada del internet comercial, lo cual las hace ser una red dedicada y es de gran importancia considerar el rendimiento de la red. SDN tiene como su objetivo principal ejercer de forma controlada la administración de la red, además de que ofrece un mejor control, por lo que el

rendimiento de la red es un punto muy importante para las SDN. Entonces, si en una red avanzada tradicional es de vital importancia el rendimiento, se podría considerar que las redes avanzadas funcionen como redes SDN, siempre con el objetivo de continuar con las investigaciones diversas para un mejor desarrollo, por ello es que en esta investigación se implementará entornos emulados de una red avanzada tradicional y una red avanzada definida por software para determinar el nivel de rendimiento que existe y a través de esta comparación dar a conocer qué tipo de funcionamiento puede tener mejor rendimiento para las redes avanzadas. Según Bolatti (2018) en las SDN se debe tener en cuenta particularmente la posibilidad que ofrecen la caracterización de parámetros de enlaces, tanto en lo que hace a las características del enlace como ancho de banda, retardo, paquetes enviados y recibidos y velocidades de transmisión.

1.2. Formulación del Problema

1.2.1. Problema General

¿Qué relación existe entre el rendimiento y el tipo de implementación de una arquitectura de red avanzada?

1.2.2. Problemas Específicos

1. ¿Qué relación existe entre el retardo y el tipo de implementación de una arquitectura de red avanzada?

2. ¿Qué relación existe entre la pérdida de paquetes y el tipo de implementación de una arquitectura de red avanzada?

3. ¿Qué relación existe entre el consumo de recursos computacionales y el tipo de implementación de una arquitectura de red avanzada?

1.3. Justificación e Importancia

La necesidad de realizar investigaciones científicas ha dado lugar a las redes avanzadas, considerando que SDN es una red mucho más ágil y mejor gestión e incluso es tratada como una alternativa de solución en las organizaciones, entonces lo usual sería que las redes de arquitectura avanzada tengan funcionamiento bajo las SDN.

La importancia de esta investigación se caracteriza en demostrar la diferencia que existe entre la implementación mediante la emulación de una arquitectura real de red avanzada tomada como referencia; en este caso REUNA como una red definida por hardware

(HDN) versus una red programable como red definida por software (SDN), se busca comparar el rendimiento de cada una de estas implementaciones y con ello determinar cuál es la mejor opción de funcionamiento para una red avanzada considerando el rendimiento de la red; por tanto consideramos responder a algunas preguntas de referencia en lo práctico ¿esta investigación ayuda a resolver algún problema real?; el rendimiento en una red es un factor primordial o más bien es una necesidad prioritaria y tiene mayor énfasis tratándose de una red de arquitectura avanzada, ya que se requiere conectividad estable para la producción científica que se desarrolla a través de múltiples investigaciones, mediante esta evaluación se puede considerar una solución para el funcionamiento más eficiente de una arquitectura de red avanzada basados en los resultados que esta investigación; en lo teórico ¿se llenará algún vacío de conocimiento?, esta investigación por ende valida conceptos que incluyen que una SDN es mucho más ágil que una HDN pero esto en redes que son implementadas bajo IPv4, el conocimiento que aportará esta investigación recae básicamente en la parte de contrastar este concepto pero en una arquitectura de red avanzada es de decir bajo IPv6; ¿se podrá generalizar los resultados a principios más amplios?, en esta investigación se obtendrá datos del rendimiento de una arquitectura de red avanzada basada en el tipo de implementación, basado en estos resultados podemos generalizar la implementación más optima de una arquitectura de red avanzada tomando como pilar el rendimiento; en lo metodológico ¿contribuye esta investigación a la definición de la relación entre las variables?, esta investigación contribuye a definir la relación que hay entre el rendimiento de una arquitectura de red avanzada y el tipo de implementación, ya que es muy distinto implementar una red de manera tradicional donde prácticamente se tiene tanto controladores preestablecidos en cada nodo o router que en una implementación de manera programable, donde podemos ejercer la administración de la red mediante un controlador centralizado, ¿esta investigación puede contribuir a crear un instrumento para recolectar datos?, el instrumento que nos permite recolectar datos es un recurso, por tanto en esta investigación puede contribuir a un instrumento de recolección de datos para investigaciones que consideren escenarios de implementación de red y se requiera analizar como recolectar datos basadas en los indicadores que engloba el rendimiento de una arquitectura de red avanzada; en lo social ¿cuál es su trascendencia para la sociedad?, ¿Quiénes se beneficiarán con los resultados de esta investigación?, esta investigación trascenderá al momento que las redes avanzadas migren de su implementación de manera tradicional HDN a una de manera programable SDN y que finalmente dará paso a otros factores de investigación que sean más

que solo el rendimiento; finalmente pueden beneficiarse distintas organizaciones así como la comunidad de redes avanzadas para realizar dicha migración; también se considera como aporte para los futuros profesionales que en algún momento requerirán de este tipo de conocimientos en el campo de investigación, laboral y empresarial, además bien es sabido que no solo de las redes avanzadas, sino que la con actividad global pasará a operar bajo IPv6; por tanto será un pilar de investigación para trabajos futuros.

1.4. Alcance y Limitaciones

Debido a los avances de investigación en el sector académico y empresarial deriva el interés de estudiar una arquitectura de red avanzada enfocada en determinar su rendimiento a manera de comparativa, para ello es necesario tener dos implementaciones distintas que son una de manera tradicional y otra de forma programable. Esta investigación tiene como alcance medir el rendimiento de una arquitectura de red avanzada para ambas implementaciones, las cuales se realizará en entornos emulados y durante el año 2020.

Las limitaciones de esta investigación están dadas con base en la recolección de datos, ya que al no disponer con equipos físicos compatibles con redes definidas por software (SDN) se realizará la implementación mediante el emulador Mininet; así mismo, al no contar con la disponibilidad de dispositivos físicos para una red definida por hardware (HDN) se realizará dicha implementación mediante el emulador GNS3. Para realizar la evaluación se empleará para ambas emulaciones, la topología más actualizada del backbone de la red avanzada de Chile (REUNA), bajo el protocolo IPv6.

1.5. Antecedentes

Existen diversas investigaciones referentes al rendimiento en redes, redes avanzadas y redes definidas por software, a continuación, se hará una descripción destallada de las investigaciones que guardan relación con la presenta investigación que se está realizando.

Enríquez (2015) en su artículo “Análisis de rendimiento en redes IPv6”, se enfoca en desarrollar una arquitectura IPv6/DissServ en un laboratorio de red. Con el fin de evaluar el comportamiento de la red de núcleo en IPv6, aquí considera que la voz y telefonía IP generan tráfico en tiempo real y requieren de bajo retardo, y en lo mínimo perdida de paquetes en la transmisión de extremo a extremo y los videos requieren mayor ancho de banda para una mejor calidad de servicio (QoS), para datos de voz y video el rendimiento es sensible al retardo, diferencia de retardo y pérdida de paquetes. Los videos son sensibles al *jitter* y

pérdida de paquetes, los videos en tiempo real son más sensibles al *jitter* y pérdida de paquetes, las transacciones interactivas medianamente sensibles al retardo y las aplicaciones de mejor esfuerzo son sensibles al retardo, ancho de banda, diferencia de retardo ni perdida de paquetes, para el retardo se considera como métrica el *delay*, para la diferencia de retardo el *jitter* y para cantidad de datos transmitidos throughput. En esta arquitectura al ser de red avanzada opera bajo IPv6, se da uso a OSPFv3 que utiliza como métrica el costo lo cual es asociado a la velocidad de la interfaz, esta prueba es realizada en dispositivos de capa 2 y 3 de las tecnologías Cisco. El autor en sus aportaciones manifiesta que para medir el rendimiento de una red, debemos identificar problemáticas de despliegue, evitar malas configuraciones o incluso interpretaciones erróneas, para dar conclusiones acerca del retardo, ancho de banda y pérdida de paquetes, el autor menciona algunas técnicas utilizadas como *class-map* para definir las clases de tráfico, *police-map QoS-Policy* para permitir crear políticas dependiendo del tráfico, *random-detect dscp-based* para aplicar algoritmo RED (*Random Early Detection*)al tráfico de video interactivo para descartar de forma selectiva paquetes cuando haya congestión, *policy-map Marcacion* para marcación de campo DS a paquetes IPv6, *policy-map Shaping-Police* para tráfico *best-effort* y comprimir la cabecera TCP. El autor menciona que OSPFv3 no considera congestión o retardo. Para algunas pruebas de carga se observa métricas de txload para transmisión y rxload para recepción de carga entre las que determina el ancho de banda máximo a utilizar. Para la conectividad de extremo a extremo hace pruebas de ping y *traceroute*, observa los menores tiempos de retardo de red, y desprende inyección de paquetes desde los extremos de diferentes bytes, donde a principio el autor reconoce que los paquetes de mayor byte deberían ser rechazados, pero no ocurrió. Algunas de sus otras pruebas es análisis de tráfico en periodos de segundos con paquetes de diferentes tamaños. En sus conclusiones indica que tuvo enlaces WAN de baja velocidad y no se congestionaron con facilidad. Algo que el autor destaca es que cuando ocurre una congestión, el retardo aumenta considerablemente; en una prueba inyección que realizó de paquetes de 50000 bytes, el retardo en momentos de congestión fue superior a 3800 mseg; el autor admite que el retardo y diferencia de retardo también afectan considerablemente la funcionalidad de estas aplicaciones, una observación clara es que OSPF utiliza como ruta entre 2 redes la ruta de menor costo y esto evidencia una congestión de red.

Muro (2016) en su trabajo de fin de Máster “Plataforma de pruebas para evaluar el desempeño de las redes definidas por software basadas en el protocolo *OpenFlow*”, realiza una plataforma de pruebas para el desempeño del protocolo *OpenFlow*, el cual es el estándar en las redes definidas por software, que al final termina influyendo en el desempeño de toda la SDN. Para esta investigación los controladores seleccionados para la evaluación fueron: *POX, RYU, OVS, ONOS, FLOODLIGHT y OPENDAYLIGHT*, y se utiliza el protocolo OpenFlow entre las versiones 1.0, la máquina virtual con el sistema operativo *Ubuntu* 14.04 es donde se le instaló el emulador *Mininet*, se utilizó el analizador de protocolos *WIRESHARK* con soporte para el protocolo *OpenFlow*. El autor en sus conclusiones indica que para las SDN basadas en el protocolo *OpenFlow* se requiere profundizar en su estudio y evaluación. Mediante la implementación de una plataforma de pruebas y el diseño de experimentos se comprueba que las SDN pueden reducir la latencia de la red, las pruebas de escalabilidad realizadas con las herramientas desarrolladas permitieron evaluar la capacidad de escalar grandes redes SDN con numerosos hosts y conmutadores. También manifiesta que existe una estrecha relación de dependencia entre la capacidad de la plataforma de cómputo, la topología de la red y el máximo tamaño de esta; la evaluación de controladores SDN requiere de un estudio profundo de cada sistema operativo de red, así como de las aplicaciones de red equivalentes para realizar una comparación justa.

Musante (2017) En su informe de tesis de Maestría “Diseño de una red industrial” desarrollada en el Instituto Tecnológico de Buenos Aires, tuvo como objetivo diseñar una red que cumpla con requerimiento de sistemas productivos y de áreas de tecnología de la información, donde se plantea una red tradicional y una red definida por software. Para ello el autor enfoca su inicio en que todas las redes considera el plano de control (protocolos), el de datos (tráfico de red) y el de gestión (transmisión de operaciones y administración de tráfico), esto en una red tradicional es independiente con visibilidad parcial de la topología, en una SDN la arquitectura el enfoque es control centralizado y un plano de datos distribuido de forma dinámica, flexible y adaptable. El autor menciona que otra de sus particularidades es que separan los planos de control de la red de las funciones de reenvío invocando una interfaz API entre los dos, esto permitirá la programabilidad del control y abstracción de la infraestructura. Una SDN debe estar formada por un controlador SDN que básicamente es el cerebro de la red y es quien retransmite la información de conmutación a los equipos de red mediante la API *southbound* (interfaz de control de red) y se comunica con las

aplicaciones del negocio mediante la API *northbound* la cual permite la innovación, automatización de tareas para desarrollar la programabilidad. El autor dentro de sus consideraciones para diseñar la red toma como prioridad la disponibilidad, baja latencia, nivel de servicios de red, procesamiento, consolidación de hardware, coexistencia de redes logísticas, enrutamiento y segregación lógica conectividad controlada. Con base en las conclusiones esperadas, el autor opta por diseñar una SDN para lo cual se realizaron las pruebas de concepto considerando los diferentes dispositivos de red de marca Cisco, tal que se planteó en un ambiente de laboratorio, que debió simular cada uno de los ambientes reales; significa, que se simuló un *core* de red distribuido utilizando grandes longitudes de fibra óptica, se estableció un nodo de distribución con sus accesos con el fin de emular una línea de producción, se configuraron servidores de aplicación, se conectaron a la red elementos de control industrial y se realizaron pruebas de acceso y gestión como se harían en la implementación final.

Vergel (2016) en el artículo científico "Evaluación de desempeño y configuraciones de las SDN mediante la simulación" se enfoca en determinar el desempeño de las SDN, e indica que para evaluar los elementos de las SDN es necesario conocer las métricas que brindan información relevante del desempeño de la red, relacionadas con los mensajes *OpenFlow* del protocolo, del controlador o de los *Switches*. También indica que existen muchas métricas definidas para las redes tradicionales que también son válidas para las SDN como es el caso de la razón de pérdidas de paquetes, la latencia y variación de la latencia y el tiempo de activación del servicio. Las pruebas que realiza son mediante el emulador *Mininet* y define dos categorías de pruebas; las de configuraciones y las de desempeño. El autor define medidas propias de estas pruebas; para configuración y conectividad entre nodos, se debe verificar el intercambio de mensajes *OpenFlow*, y la integración con dispositivos tradicionales. Para las pruebas de desempeño se debe medir la razón de pérdidas de paquetes, latencia y variación de la latencia (*throughput*); el autor concluye que en función de las pruebas que se realicen en cada simulación se podrá extraer determinada información relevante del escenario. La simulación de un escenario recreando un prototipo de una red de campus SDN, empleando *Mininet*, permitió obtener las métricas relevantes para las SDN y el desempeño del controlador *OpenDaylig*ht. Los resultados obtenidos permitieron analizar aspectos importantes como el rendimiento y la escalabilidad del

controlador y se pudo comprobar la importancia de la simulación para estudiar y evaluar el comportamiento de una red SDN.

Castillo Velázquez, Cobos Panduro y Marchand Niño (2018) en su artículo científico "*IPV6 Connectivity and Management Emulation for REUNA, the Chilean Advanced Network*" realiza una emulación de la Red Nacional Académica de Chile (REUNA) como parte de la Red CLARA (red que interconecta las redes académicas del continente americano). Para ello se usó IPv6 por tal razón se utilizó OSPFv3 como protocolo de enrutamiento en toda la red. Al realizar las pruebas de conectividad en todos los casos este proceso consumió un 45.0 % de un CPU Intel Core i7 y el 87.5 % de memoria de 16 GB, todo esto en el proceso del funcionamiento de la emulación de la red avanzada REUNA mediante GNS3. En las conclusiones menciona que hay poca información sobre las redes avanzadas y por ello la importancia de contribuir a ofrecer un enfoque para conocer el funcionamiento de REUNA.

1.6. Objetivos

1.6.1. Objetivo General

Evaluar la relación que existe entre el rendimiento y el tipo de implementación de una arquitectura de red avanzada para la elaboración de guía de implementación.

1.6.2. Objetivos Específicos

1. Evaluar la relación que existe entre el retardo y el tipo de implementación de una arquitectura de red avanzada.

2. Evaluar la relación que existe entre la pérdida de paquetes y el tipo de implementación de una arquitectura de red avanzada.

3. Evaluar la relación que existe entre el consumo de recursos computacionales y el tipo de implementación de una arquitectura de red avanzada.

II. Revisión de Literatura

2.1. Marco Teórico

2.1.1. Red Tradicional o Red Definida por Hardware (HDN)

La definición básica de una red es un conjunto de nodos interconectados, por lo que una red de computadoras es un conjunto de dispositivos interconectados que puede generar información.

Según (Tanenbaum, 2003) "es un conjunto de computadoras autónomas interconectadas que pueden intercambiar información".

Las redes tradicionales actualmente operan bajo el protocolo IPv4 de forma estándar, pero todo computador soporta protocolo IPv6 para cuando sea necesario migrar a este protocolo de red.

Las redes se pueden clasificar de acuerdo con su extensión de área física donde se encuentran sus componentes de manera física o lógica en diferentes tipos, esto quiere decir que el tamaño de la red es la que las clasifica y pueden ser desde personales hasta Internet que es la red de redes, según (Cisco, 2010) estas son:

Redes de área personal (PAN). Permiten a los dispositivos comunicarse dentro del rango de una persona.

Redes de área local (LAN). Infraestructura de la red que proporciona acceso a usuarios o terminales en un área geográfica pequeña, Son redes de propiedad privada que operan dentro de un solo edificio, como una casa, oficina o fábrica.

Red de área metropolitana (MAN). Infraestructuras de red que abarcan un área física mayor que la de una LAN, pero menor que la de una WAN Son las redes que cubren toda una ciudad como las redes de televisión por cable disponibles en muchas ciudades.

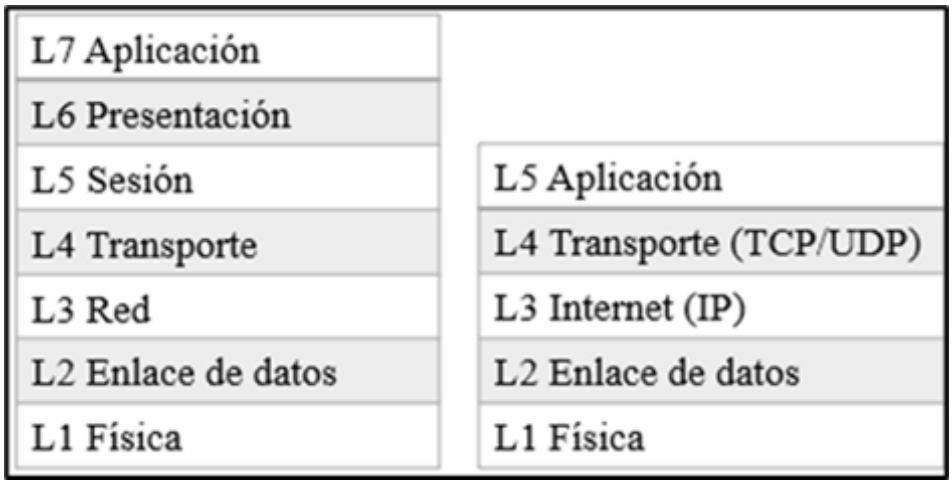
Red de área amplia (WAN). Infraestructura de la red que proporciona acceso a otras redes en un área geográfica extensa que suele ser propiedad de un proveedor de servicios, quien también la administran Estas redes comprenden una extensa área geográfica, por lo general un país o continente.

LAN inalámbrica (WLAN). Son similares a las LAN, solo que interconectan de forma inalámbrica a los usuarios y los extremos en un área geográfica pequeña.

Toda red tradicional o red definida por hardware se basa en el modelo de referencia OSI para determinar las funciones y servicios que se pueden presentar en cada capa, para que las comunicaciones se lleven a cabo correctamente se define el modelo de protocolos TCP/IP, el cual abarca 5 capas; esto se observa en la Figura 1.

Figura 1

Modelo de referencia OSI y TCP/IP



Nota. La Figura muestra las 7 capas del modelo OSI y las 5 capas de modelo TCP/IP. Tomado de (Castillo Velázquez J. I., 2019)

A continuación, se indica la función de cada una de las capas que son definidas por Stalings (2004).

Capa de Aplicación. Contiene protocolos utilizados para comunicaciones procesos a procesos, que representa datos para el usuario más el control de codificación y de dialogo

Capa de Presentación. Proporciona una representación común de los datos transferido entre los servicios de la capa de aplicación.

Capa de Sesión. Proporciona servicios a la capa de presentación para organizar dialogo y administrar intercambio de datos.

Capa de Transporte. Define los servicios para segmentar, transferir y reensamblar los datos para las comunicaciones individuales; transporte (origen-destino) que admite la comunicación entre distintos dispositivos a través de las diversas redes.

Capa de Red. Proporciona servicios para intercambiar los datos individuales en la red entre terminales identificados; la cual determina el mejor camino a través de una red**.**

Capa de Enlace de Datos. Describe los métodos para intercambiar tramas de datos entre dispositivos en un medio común; acceso a la red que controla los dispositivos del hardware y los medios que forman la red.

Capa Física. Describe los medios mecánicos, eléctricos, funcionales y de procedimiento para activar, mantener y desactivar conexiones físicas para la transmisión de bits hacia y desde un dispositivo de red, es decir especifica las características de hardware.

Para el funcionamiento de una red, en las distintas capas existen protocolos, los cuales son estándares de comunicación que existe en una red, estas funcionan al existir tráfico que es generada a través de peticiones entre cliente y servidor, u otros tipos de conexiones, en la Figura 2 podemos apreciar los diversos protocolos y otras consideraciones que funcionan en las redes tradicionales en base al modelo de referencia OSI.

Figura 2

Protocolos y funcionalidades del modelo OSI

Aplicación		HTML, http, telnet, FTP
Presentación		JPEG, MIDI, MPEG, ASCII
Sesión		Control de diálogo
Transporte		Control de flujo TCP UDP
Red		Enrutaiento IP, IPX, RIP, IGRP, OSPF
Enlace de datos	LLC	Ethernet, 802.2, 802.3, HDLC, Frame Relay
	MAC	
Física		Bits

Nota. La Figura muestra los protocolos y que operan en cada capa del modelo OSI. Tomado de (Ariganello, 2014).

A nivel de red encontramos los protocolos de enrutamiento, de los cuales según (Ariganello, 2014) pueden ser estáticos y dinámicos, estos protocolos funcionan tanto como para IPv4 como IPv6.

Enrutamiento Estático: Este enrutamiento permite conectividad a través de una configuración de ruta ya sea en ambas direcciones o como requiera la Red, las rutas estáticas requieren de una construcción manual de enrutamiento, podemos considerar también las rutas por defecto.

Enrutamiento Dinámico: Permite que los dispositivos de enrutamiento actualicen los conocimientos antes posibles cambios sin tener que recurrir a realizar nuevas configuraciones, para ello este tipo de enrutamiento mantiene una actualización de tablas routing, existen 2 núcleos de enrutamiento dinámico.

- ✓ **Protocolos de Gateway Interior (IGP).** Son los que se utilizan para intercambiar información de enrutamiento dentro de un sistema autónomo, entre ellos encontramos RIP, IGRP, EIGRP, OSPF, IS- IS.
- ✓ **Protocolos de Gateway Exterior (EGP).** Se utilizan para intercambiar información de enrutamiento entre sistemas autónomos, entre estos encontramos a BGP.

Protocolo de Internet Versión 6 (IPv6)

El protocolo de internet versión 6, conocido también como IPv6 es una versión mejorada del IPv4, trae consigo bastantes mejoras, según (RFC-2460, 1998), algunos de los cambios que propicia IPv6 son los siguientes:

- ✓ **Capacidades de direccionamiento expandidas.** Debido al problema de agotamiento de direcciones IP, a diferencia de IPv4 que tiene tamaño de 32 bits, IPv6 trae consigo un mayor tamaño, ya que tiene 128 bits, un numero bastante grande para satisfacer las necesidades de asignación de direcciones IP, además de ello, IPv6 define un nuevo tipo de dirección llamada "dirección *anycast*", utilizada para enviar un paquete a cualquiera de un grupo de nodos**.**
- ✓ **Simplificación de formato de encabezado.** El encabezado IPv6 es parecido al encabezado IPv4, a excepción de algunos campos de encabezado IPV4 que se han eliminado o se han hecho opcionales, para reducir el coste de procesamiento de casos comunes del tratamiento de paquetes y para limitar el coste de ancho de banda del encabezado IPv6.
- ✓ **Autenticación y privacidad.** Se agregaron extensiones para admitir autenticación, integridad de datos y la confidencialidad de los datos se especifica para IPv6.

- ✓ **Direcciones IPv6.** Según (RFC-1884, 1995), existen tres tipos de direcciones para IPv6, entre los cuales se encuentran los siguientes:
  1. ***Unicast*:** Este tipo de direcciones IPv6 tienen identificador para una sola interfaz. Un paquete enviado a una dirección de unidifusión se entrega a la interfaz identificada por esa dirección.
  2. ***Anycast:*** Este tipo de dirección utiliza un identificador para un conjunto de interfaces que normalmente pertenecen a diferentes nodos. Un paquete enviado a una dirección *anycast* se entrega a una de las interfaces identificadas por esa dirección.
  3. ***Multicast:*** Este tipo de dirección utiliza un identificador para un conjunto de interfaces, el paquete se entrega a todas las interfaces identificadas por esa dirección. No hay direcciones *broadcast* en IPv6 su función esta reemplazada por direcciones de *multicast*.

OSPF para IPv4 (OSPFv2)

El protocolo OSPF según (RFC-1247, 1991), es un protocolo de enrutamiento dinámico basado en un estado de enlace, está diseñado para ser ejecutado internamente a un único Sistema Autónomo (SA), cada router OSPF mantiene una base de datos idéntica que describe la topología del sistema autónomo, a partir de esa base de datos, una tabla de enrutamiento se calcula construyendo un árbol de ruta de acceso más corto. Esto quiere decir que OSPF puede percibir rápidamente cambios en la topología y después de un pequeño periodo de convergencia, calcular nuevas rutas. Este protocolo de enrutamiento dinámico se basa solamente en la dirección de destino; es decir, no encapsula los paquetes IP.

OSPF para IPv6 (OSPFv3)

Al existir las direcciones IPV6 y la necesidad de utilizar el protocolo de enrutamiento OSPF se creó OSPF para IPv6, conocida también como OSPFv3, y es la versión mejorada de OSPFv2 que es para direcciones IPv4, según (RFC-5340, 2008), los mecanismos fundamentales OSPF (inundación, elección del enrutador designado (DR), ayuda del área, camino más corto, cálculos SPF, etc.) permanecen sin cambios. Los algoritmos de OSPFv2 para IPv4 han sido conservados para IPv6, pero con algunos cambios necesarios como:

- ✓ **El protocolo de procesamiento ya no es por subred.** OSPF para IPv6 se ejecuta por enlace en lugar de IPv4 comportamiento de la subred por IP.

- ✓ **Eliminación de la semántica de direccionamiento.** Lo que deja un núcleo independiente del protocolo de red.
- ✓ **Adición de alcance de inundación (LSA).** Ahora existe alcance local de enlace, alcance del área, alcance AS.
- ✓ **Soporta explícito para múltiples instancias de enlace.** Admite la capacidad la ejecutar varias instancias de protocolo OSPF en un único enlace.
- ✓ **Uso de direcciones locales de enlace.** Las direcciones locales de enlace IPv6 se usan en un solo enlace, para fines de descubrimiento vecino, configuración automática, etc.
- ✓ **Cambios de autenticación.** La autenticación se ha eliminado de OSPF protocolo
- ✓ **Cambio de formato de paquete.** El número de versión de OSPF se ha incrementado de 2 a 3, el campo de opciones *hello* y descripción se ha ampliado a 24 bits, se agregaron dos bits de opciones "R-bit" y "v6-bit", el paquete *hello* no contiene información de dirección, los campos de autenticación y *autype* se han eliminado, el encabezado del paquete ahora contiene un ID para múltiples instancias.
- ✓ **Cambio de formato LSA.** Todas las semánticas de direccionamiento se han eliminado del encabezado LSA, router-LSAs y *network*-LSAs, ahora describen el enrutamiento de la topología del dominio de una manera independiente del protocolo de red.
- ✓ **Manejo de tipos de LSA desconocidos.** Los tipos LSA desconocidos se tratan como si tuvieran alcance de inundación local de enlace, o se almacenan e inundan como si fueran entendido.
- ✓ **Soporte de área Stub /NSSA.** De los tipos de LSA obligatorios, las áreas de código auxiliar solo llevan LSA de enrutador, LSA de red, LSA entre prefijos de área, LSA de enlace e LSA de prefijo dentro del área.
- ✓ **La identificación de vecinos por Router ID en OSPF para IPv6**. Los routers vecinos en un determinado enlace siempre son identificados por su ID de router OSPF.

2.1.2. Redes Avanzadas

Según REDCLARA (2016) "Las redes de investigación y educación (o redes avanzadas) permiten a científicos, investigadores, académicos, profesores y estudiantes colaborar, compartiendo información y herramientas mediante una serie de interconexiones de redes". *GÉANT* es el proyecto de conectividad global que apoya la investigación de toda la comunidad académica y científica, en la Figura 3, se muestra el mapa global de conectividad de redes avanzadas dada en 8 regiones.

Figura 3

Mapa de conectividad global para redes avanzadas

Nota. Mapa de conectividad global de las redes avanzadas. Tomado de (GÉANT, 2020).

Según (Cabezas Bullemore & Bravo Marchant, 2010), las redes avanzadas son clave debido a que:

- ✓ Sobre ellas se desarrollan nuevos servicios y aplicaciones que permiten el crecimiento y fortalecimiento de Internet.
- ✓ La comunidad científica más amplia requiere una infraestructura para la colaboración, educación y acceso a instrumental que el sector privado no puede ofrecerle.
- ✓ La red en sí misma es un laboratorio para probar nuevos protocolos, mejorar la calidad de servicio y velocidades que no existen en internet comercial.
- ✓ La infraestructura permite un espacio de colaboración para afrontar temas que son desafíos mundiales y requieren de muchos grupos de investigadores trabajando sobre grandes volúmenes de datos.
- ✓ La negociación en conjunto permite rebajas considerables de los costos de conexión a las instituciones de investigación.

Las redes avanzadas ofrecen servicios que incluye:

Redes:

- ✓ Tecnología DWDM en sus troncales que permite navegar a altas velocidades.
- ✓ Circuitos virtuales para proyectos de investigación.
- ✓ Adquisición conjunta o asesoría en enlaces de acuerdo con los requerimientos de la comunidad.
- ✓ IP tradicional (IPv4) e IPv6
- ✓ Red con *multicast* (multidifusión)
- ✓ Monitoreo permanente del estado de la red.

Servicios:

- ✓ Respuesta rápida a incidentes
- ✓ Servicios de videoconferencia
- ✓ Certificación: canales seguros de comunicación en los servidores de Internet (Web y de otro tipo), lo que ofrece la posibilidad de emplear certificados automáticamente reconocidos por los clientes (navegadores, lectores de correo, etc.)
- ✓ Autenticación: Infraestructura de clave pública
- ✓ Movilidad: Acceso a la red para usuarios móviles (en Europa)
- ✓ Telefonía IP

Colaboración

- ✓ Cooperación con investigadores para determinar la viabilidad técnica de sus proyectos, lo que permite que los colaboradores interactúen para obtener proyectos eficientes.
- ✓ Captación de recursos vía fondos concursables para proyectos que usan la red.
- ✓ Conectividad internacional.

2.1.3. Red Definida por Software (SDN)

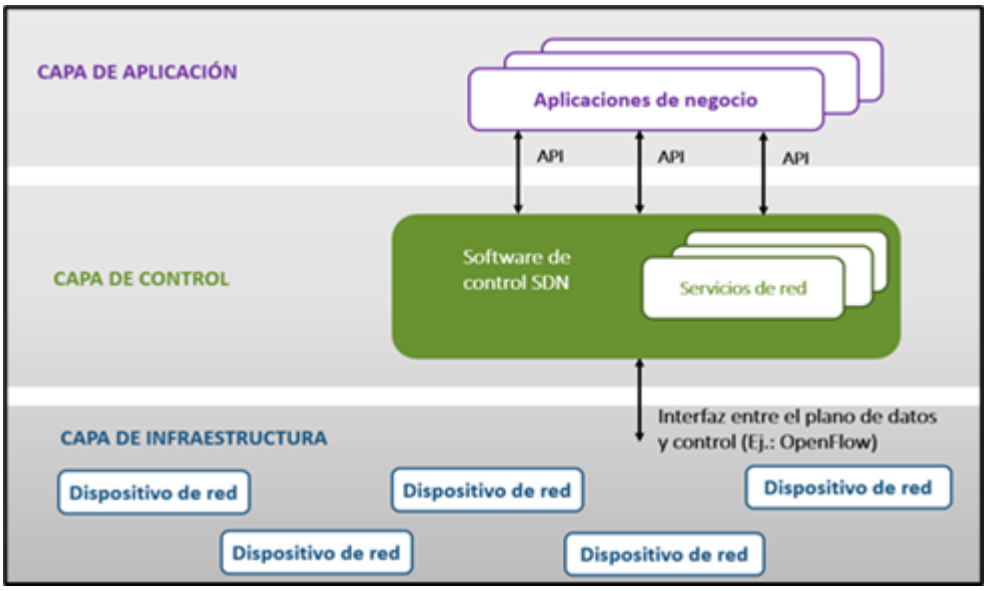
Las redes definidas por software permiten una mejor administración y control de los dispositivos de red, según (Cai, 2011) "una Red Definida por Software es un paradigma, donde un software central, llamado controlador, el cual manda el comportamiento de toda una red", a través de un controlador que separa el plano de datos y el de control, según (Albert Greenberg, 2005) "los dispositivos de red se convierten en dispositivos de reenvió de paquetes, obedeciendo al controlador, la configuración se realiza de forma centralizada, para realizar cambios en una red SDN, no es necesario configurar cada dispositivo de red, sino que toda la red haga un reenvió de tráfico hacia una ubicación lógica centralizada, llamado controlador quien tiene conocimiento global del estado de la red".

Según la (Open Networking Foundation, 2019) SDN "es una arquitectura de red dinámica, gestionable, adaptable de costo eficiente, separando el control de red y la funcionalidad de reenvío de información, a fin de que el control sea programable, haciendo que las aplicaciones servicios de red se abstraigan. Las redes definidas por software (SDN) son una arquitectura emergente que es dinámica, manejable, rentable y adaptable, haciéndola ideal para la naturaleza de banda ancha dinámica de las aplicaciones actuales, la arquitectura de las redes definidas por software debe ser programables, ágil, centralizadas, la configuración debe ser mediante la programación, y debe estar basada en estándares abiertos y neutrales". Lo que hace que el plano de datos y el plano de control estén separados, con intenciones de controlar la red de forma programable.

La arquitectura SDN básicamente está formado por el plano de control, y el plano de datos, que distribuye la capa de aplicación, la capa de control y la capa de infraestructura como se aprecia en la Figura 4. La base fundamental de una SDN se encuentra en el controlador, también llamado cerebro, lo particular es que este controlador es basado en software y mantiene una vista completa de la topología de red.

Figura 4

Arquitectura de las redes definidas por software (SDN)



Nota. La imagen muestra la arquitectura de una red definida por software, en base a 3 capas. Tomado de (Open Networking Foundation, 2019)

Dentro de los elementos de las Redes Definidas por Software podemos encontrar:

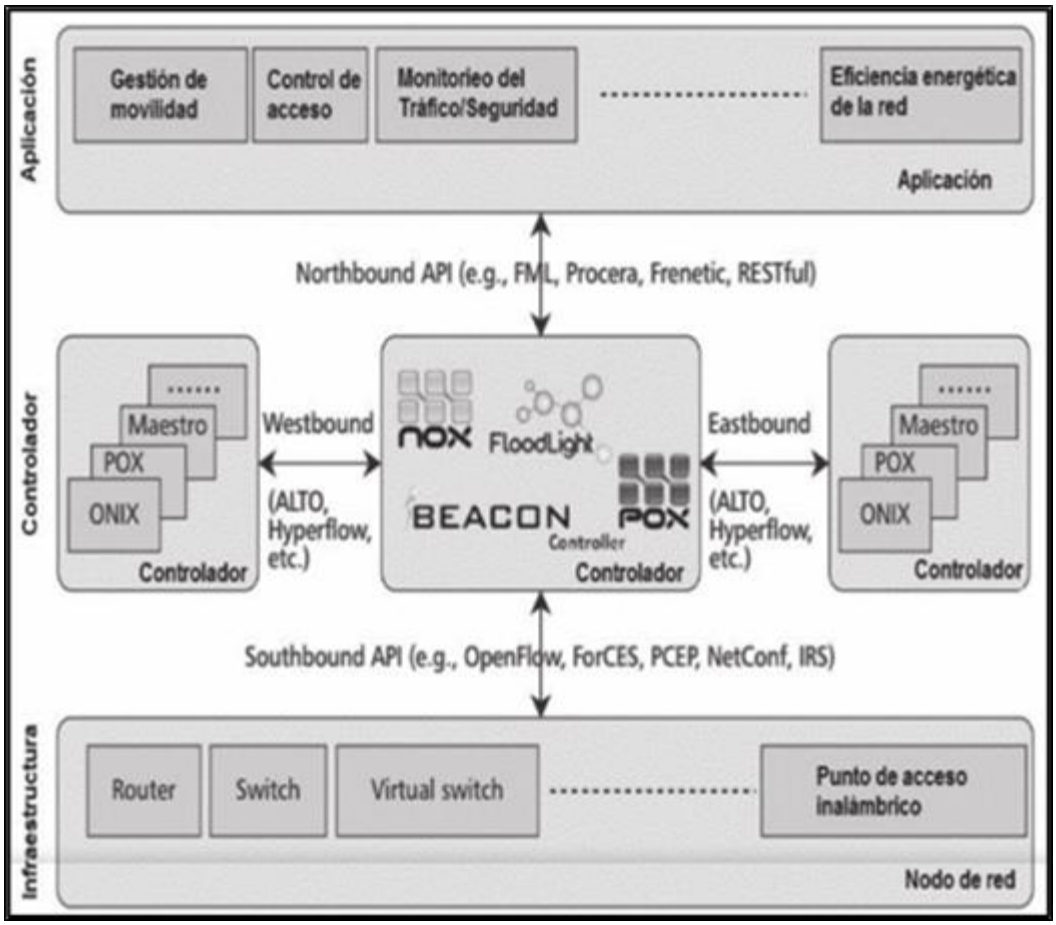
1. **Capa de control**

Básicamente esta capa crea una vista general y centralizada de la topología de red, para gestionar el flujo de datos en la capa de infraestructura a través de un protocolo, que a la actualidad el estandarizado es *OpenFlow*, esta capa opera en el plano de control.

Según (Abdullah, Kemal, & Selcuk, 2015) "esta capa proporciona un control de red programable, y sumamente ágil en comparación a una red tradicional, una red inteligente permite a los administradores de red configurar, administrar y optimizar los recursos de red a través de los scripts", por otra parte, esta capa es responsable de establecer el trato de los flujos de los datos, con la ayuda del controlador SDN. Aunque existen varios protocolos, se utiliza *OpenFlow*, Pinilla (2015). En la Figura 5 se observa algunos de los controladores más referenciados que pueden operar en esta capa.

Figura 5

Controladores en arquitecturas SDN



Nota. La Figura hace referencia a los diferentes controladores controladores dentro de la arquitectura SDN. Tomada de (Albert Greenberg, 2005)

2. Capa de Aplicación

Esta capa permite crear aplicaciones que ayuden a optimizar el proceso de configuración de nuevos servicios de red, se comunica con el controlador a través de una *API* (*Appliction Programming Interface*) con el fin de tener una visión del estado actual de la red, para mejorar su transmisión de datos, esta capa está enlazada con el plano de control. También puede ser llamada como la capa de negocio ya que aquí se crea y se componen servicios que serán utilizados en la red, Minh & Doan B (2016)

Según (Hidalgo, 2014) "Admite establecer aplicaciones para de forma automática ejecutar las configuraciones, abastecimiento y extender los nuevos servicios en la red, las capas de aplicación y control se comunican mediante una *API*, para conocer el estado general

de la red. Esto ayuda a mejorar la forma de transmitir los datos porque los nodos podrán administrar el tráfico de flujos de las redes individuales para aplicaciones específicas, como sus plataformas de control serán distribuidas, la capa de aplicaciones permite relacionar las funciones de los centros de datos para obtener seguridad en la red mientras se está trabajando a la vez que mejora de forma constante la automatización".

La capa de aplicación funciona dentro del plano de control y es donde operan las aplicaciones de negocio de los usuarios quienes son los que finalmente utilizarán los servicios de una red definida por software a través de las *API NorthBound*.

3. Capa de infraestructura

Esta capa se encuentra formada por los elementos de red, que son gestionados por el controlador a través del protocolo *OpenFlow*, esta capa opera en el plano de datos; según (Barona, 2013) "se encuentra conformada por enrutadores y conmutadores físicos, encargados de la administración de las tablas de flujo mediante el controlador, encargada de cambiar o agregar dispositivos, esta capa enlazada con el plano de datos", según (Icaza, 2016) "las solicitudes que se ejecutan entre la capa de infraestructura y la capa de control son realizadas mediante la interfaz *SouthBound*, la comunicación entre la capa de control y las aplicaciones es realizada con la interfaz *NorthBound*".

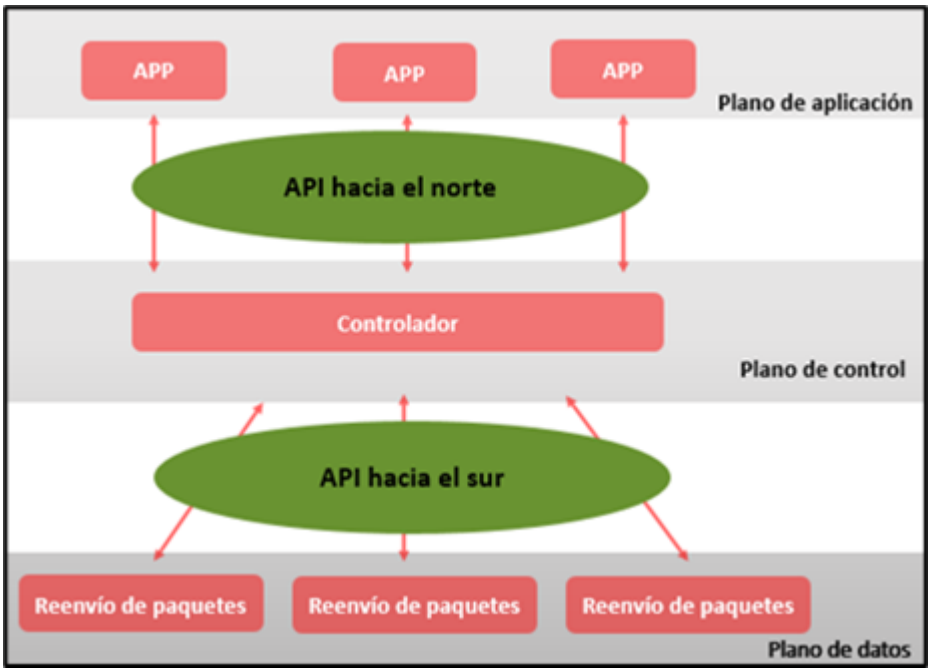
La capa de infraestructura básicamente es aquella donde encontramos a los conmutadores que deben realizar el reenvío de datos dentro de la red, los cuales serán utilizados a petición de los usuarios de la capa de aplicación a través del controlador.

4. Interfaces

Básicamente son las que conectan el controlador con la capa de aplicaciones y la capa de infraestructura, estas interfaces son las *NorthBound* ó interfaz hacia el norte que son las que hacen posible comunicación entre el controlador y la capa de aplicaciones, mediante una API *Rest*, la Figura 6 nos muestra algunas conocidas como *Frenétic*, Procera; y las *SoutBound* o interfaz hacia el sur que son las que dan conectividad entre el controlador y la capa de infraestructura a través de un protocolo que actualmente es *OpenFflow* el estandarizado.

Figura 6

Interfaces de redes definidas por software



Nota. La Figura muestra las interfaces entre el plano de control con el plano de aplicación y el plano de datos. Adaptada de (Jianqing, Yan , Jiaming, Qin, & Xin, 2015).

5. *Northbound* API

Así se ha definido a la interfaz que conecta el controlador con la capa de aplicaciones. La vista del controlador de la red es proporcionada a través de la interfaz en dirección norte Tatang, Quinkert, Frank, Ropke, & Holz (2017), según (Moreno, 2015) "se trata de una interfaz que va hacia la parte externa. Es heterogénea: *REST, RPC, OSGI,* etc. Mediante a estas interfaces se pueden comunicar entre aplicaciones y controladores, para realizar configuraciones de red con las solicitudes de las aplicaciones. Si se desea tener un nivel de seguridad alto, el administrador puede controlar las peticiones que se realicen, mediante reglas sobre el uso del tráfico o sobre permisos de accesos".

6. *SouthBound* API

Es el término que se le ha dado a la interfaz que conecta el controlador con la capa de infraestructura. El estado real de la red proporcionada por los dispositivos de red a través de la interfaz en dirección sur, Tatang, Quinkert, Frank, Ropke, & Holz (2017). En el plano de control, es el lugar en el que se realizan las configuraciones de los controladores, mediante *SB APIS*, son enviadas las reglas de flujos de tráficos a todos los dispositivos de la red;

también es posible la colocación de capas intermedias a manera de *proxy*, entre el controlador y los dispositivos de la red, para poder ser usados para la administración en distintos controladores o *NB APIS*, Moreno (2015).

7. Plano de Control

Aquí es donde encontramos al controlador de la red, según (De Cusatis, 2013) "El plano de control proporciona una capa agregada de control en el envío del estado de la red. Aunque el modelo del plano de control es centralizado, la implementación puede ser distribuida para la escalabilidad y redundancia. El plano de control presenta una abstracción de la red hacia las aplicaciones. La aplicación no tiene que preocuparse acerca de cómo se está creando el mapa y la forma en que se mantiene sincronizada y actualizada. Todo lo que sabe es que tiene el mapa como entrada, lleva a cabo su función de control y entrega una decisión". Este plano de control es quien debe ser configurado y programado para que nuestra red sea administrada a través de un software que se encuentre dentro de un controlador.

8. Plano de Datos

El plano de datos es aquel donde el flujo de datos a través del mando del controlador debe reenviarse a través de los conmutadores y enrutadores, según (De Cusatis, 2013) "El plano de datos proporciona el servicio fundamental de reenvío de datos, está optimizado para mover datos entre los puertos y se compone de uno o más elementos de reenvío interconectados por enlaces. Cada conmutador presenta un modelo de reenvío local para el plano de control, este modelo de reenvío proporciona una cierta capacidad de programación", el plano de datos es quien mediante los dispositivos de la capa de infraestructura hará él envió de datos según como lo designe el controlador que pueden ser a petición de los usuarios.

9. Controlador

Es un sistema operativo de red, que como su nombre lo indica será quien debe tener el control de la red a través de un software; los controladores SDN, tienen control de la red y se comunican con los conmutadores mediante un canal que puede ser cifrado y da lugar a la conmutación de paquetes y enrutamiento mediante las aplicaciones SDN, Tatang, Quinkert, Frank, Ropke, & Holz (2017); según (Centeno, 2014) "El controlador es quien gestiona los flujos, el controlador equivale al sistema operativo de la red que controla todas las

comunicaciones entre las aplicaciones y los dispositivos. El controlador SDN se encarga de traducir las necesidades o requisitos de la capa Aplicación a los elementos de red, y de proporcionar información relevante a las aplicaciones SDN".

Un controlador debe ser programable para poder gestionar la red, según (Kayssi, 2018) "un controlador adhiere programabilidad en el plano de control para que el administrador pueda implementar cambios a la red, normaliza las interfaces de programación de aplicaciones (*API*) mediante una *API* hacia el sur para comunicarse con la infraestructura y otra hacia el norte para comunicarse con las aplicaciones".

El controlador es el responsable de que el plano de control y el plano de datos tenga una comunicación adecuada a través de una interfaz que puede ser gestionada dentro del controlador y configurado en los dispositivos de la capa de infraestructura.

10. Protocolos de las Redes Definidas por Software

Según (EcuRed, 2015) "protocolo de red es un conjunto de normas standard que especifican el método para enviar y recibir datos entre varios ordenadores. Es una convención que controla o permite la conexión, comunicación, y transferencia de datos entre dos puntos finales".

En las redes definidas por software los protocolos deben ser flexibles que faciliten la función del controlador, según actualmente existen varios protocolos, entre los cuales se destacan *OpenFlow, OpenStack y NETCONF*.

11. OpenFlow

Según la *Open Networking Foundation* (2019) indica que las SDN requieren de este protocolo para la comunicación entre los enrutadores y los conmutadores de red, esto incluye que *OpenFlow* es un protocolo que puede gestionar una red de forma general y no como una serie de dispositivos. Asimismo, es el protocolo SBI (salida hacia elementos de red) a utilizar, la razón es que ofrece una interfaz flexible dinámica y programable para administrar los elementos de red desde un controlador centralizado SDN quien interactúa con las aplicaciones (Manzanares López, Muñoz Gea, Malgosa Sanahuja, & Flores de la Cruz (2019)

Según (Fernández, 2015) "*OpenFlow* es una tecnología de conmutación abierta que se inició en el año 2008 como un proyecto de investigación de la Universidad de *Stanford* para

no afectar el tráfico normal, bajo esta tecnología, una red puede ser gestionada como un todo y no como una serie de dispositivos individuales, es un protocolo emergente abierto que permite al controlador determinar el camino de los paquetes que debe seguir en una red".

En una SDN la comunicación que existe entre la capa de control y la capa de infraestructura es denominada *Soutbound* o *API* hacia el sur, *OpenFlow* es el protocolo de comunicación estandarizado para esta comunicación, según (Subharthi, 2017) ". la principal *API* hacia el sur es *OpenFlow* que es estandarizada por la ONF", además Mamamta (2017) menciona que mediante *OpenFlow* se separa la complejidad del protocolo de red y lo lleva al software de control de red de forma dinámica.

Entornos de emulación

La emulación de hardware consiste en dar uso a un dispositivo para imitar el funcionamiento de otro dispositivo de hardware. Según (Rouse, 2013) "Un emulador de hardware está diseñado para simular el funcionamiento de una plataforma de hardware totalmente diferente a la que se ejecuta. La emulación de hardware se utiliza generalmente para depurar y verificar un sistema en fase de diseño."; como entornos de emulación hay distintas opciones para poder realizarlo.

GNS3

Es un software gratuito de código abierto, y puede soportar dispositivos emulados y simulados, según (GNS3, 2020) "GNS3 imita o emula el hardware de un dispositivo y ejecuta imágenes reales en el dispositivo virtual"; GNS3 tiene funcionalidades de simulación y emulación de una red, además tiene múltiples opciones de utilidad que permiten conectar dispositivos físicos con máquinas virtuales, utiliza enlaces cloud y provee funcionalidades reales de los IOS.

Mininet

Según (Mininet, 2018) "*Mininet* es un emulador de red que crea una red de hosts virtuales, conmutadores, controladores y enlaces. Los hosts *Mininet* ejecutan a través del software de red *Linux* estándar y sus conmutadores admiten *OpenFlow* para un enrutamiento personalizado altamente flexible y una red definida por *software*".

Por lo que un entorno emulado es la experimentación para lograr resultados de comparación entre las redes tradicionales y las redes definidas por software, para los objetivos de esta investigación es sobre el rendimiento de red.

2.2. Marco Conceptual

2.2.1. Red Académica mundial

El esquema general de la red académica mundial está formado por ocho regiones que se interconectan, así lo indica GEANT (2020); a continuación, se describe las regiones con sus puntos de conectividad.

EUMEDCONNECT2, La Red Académica de la Región Mediterráneo. Es la única Red del Mediterráneo

Redes Académicas Avanzadas de la Región Asia-Pacífico (APAN), formada por 2 redes avanzadas.

- ✓ APAN, Es la Red Académica Avanzada de la región Asia-Pacifico
- ✓ TEIN3, *Trans-Euroasia Information Network.*

Redes Académicas del Continente Africano. Es la única Red africana

CANARIE, La Red Nacional Académica de Canadá. Única red de Canadá

INTERNET2, La Red Nacional Académica de los Estados Unidos. Única red de EE. UU.

GEANT2 (Red Europea), es la región con mayor número de redes académicas, está formada por 27 redes avanzadas las cuales se muestran en la Tabla 1, donde claramente vemos las redes académicas de los países de Europa.

Tabla 1

Redes avanzadas GEANT2

Red Académica Avanzada	
Red GEANT2	Conexiones entre redes académicas europeas
ACConet	Red Nacional de Austria)
BELNET	Red Nacional Académica de Bélgica
ISTF	Red Académica de Bulgaria
CARNet	Red Nacional Académica de Croacia
SANET	Red Nacional Académica de Eslovaquia
ARNES	Red Nacional Académica de Eslovenia
RedIRIS	Red Nacional Académica de España
EENet	Red Nacional Académica de Estonia
RENATER	Red Nacional Académica de Francia
HUNGARNET	Red Nacional Académica de Hungría
HEAnet	Red Nacional Académica de Irlanda
IUCC	Red Nacional Académica de Israel
GARR	Red Nacional Académica de Italia
LATNET	Red Nacional Académica de Latvia
LITNET	Red Nacional Académica de Lituania
RESTENA	Red Nacional Académica de Luxemburgo
SURFnet	Red Académica de los Países Bajos
NORDUnet	Red Académica de los Países Nórdicos
PSNC	Red Nacional Académica de Polonia
FCCN	Red Nacional Académica de Portugal
JANET	Red Nacional Académica del Reino Unido
CESNET	Red Nacional Académica de la República Checa
RoEduNet	Red Nacional Académica de Rumania
JSCC	Red Nacional Académica de Rusia
U LAKBIM	Red Nacional Académica de Turquía
SWITCH,	Red Nacional Académica de Suiza

Nota. La Tabla muestra todas las redes avanzadas pertenecientes a GEANT

Redes Académicas Interconectadas por la Red CLARA. Formada por 14 redes avanzadas que muestran en la Tabla 2.

Tabla 2

Redes avanzadas CLARA

Red Académica Avanzada	
CLARA	Red que Interconecta las Redes Académicas del Continente Americano
CUDI	Red Académica de México
RENIA	Red Académica Nacional de Nicaragua
RAGIE	Red Avanzada Guatemalteca
RAICES	Red Avanzada Salvadoreña
RNP	Red Académica y de Investigación de Brasil
INNOVARED	Red Académica de la Argentina
REUNA	Red Nacional Académica de Chile
RAU	Red Académica Nacional de Uruguay
ARANDU	Red Académica Nacional de Paraguay
RAAP	Red Académica Nacional del Perú
CEDIA	Red Nacional Académica del Ecuador
REACCIUN	Red Nacional Académica de Venezuela
RENATA	Red Nacional Académica de Colombia

Nota. La Tabla muestra todas las redes avanzadas pertenecientes a la red CLARA.

2.2.2. Red Universitaria Nacional de Chile (REUNA)

Según (REUNA, 2020) es la Red Universitaria Nacional, REUNA, es una corporación integrada por universidades, centros de investigación de excelencia y grupos astronómicos internacionales. Es la Red Nacional para Investigación y Educación Chilena (NREN por su sigla en inglés), y actualmente, está conformada por 37 instituciones.

La red de REUNA se extiende a lo largo de 12 regiones, desde Arica a Puerto Montt, y aspira a sumar a todas las regiones del país. Además, se encuentra interconectada a sus pares Internacionales: en América Latina (RedCLARA), América del Norte (Internet2 y Canarie), Europa (GÉANT), Asia (APAN) y Oceanía (AARNET).

La corporación de REUNA conecta la mayoría de las instituciones de Chile, se encuentra integrada por instituciones de educación superior, CONICYT y el observatorio AURA. Cada socio cuenta con un representante institucional y uno técnico, que canalizan las inquietudes de sus respectivas instituciones hacia el interior de REUNA, a través de su participación en Asambleas, reuniones o simplemente por contacto con la dirección ejecutiva.

Las instituciones socias, organizadas entre Arica y Osorno son:

- ✓ Universidad de Tarapacá
- ✓ Universidad Arturo Prat
- ✓ Universidad Católica del Norte
- ✓ Universidad de Antofagasta
- ✓ *Association of Universities for Research in Astronomy (AURA)*
- ✓ Universidad de La Serena
- ✓ Universidad Federico Santa María
- ✓ Universidad de Valparaíso
- ✓ Universidad de Chile
- ✓ Pontificia Universidad Católica de Chile
- ✓ Universidad de Santiago de Chile
- ✓ Universidad Metropolitana de Ciencias de la Educación
- ✓ Universidad Tecnológica Metropolitana
- ✓ Agencia Nacional de Investigación y Desarrollo (ANID)
- ✓ Universidad de O'Higgins
- ✓ Universidad de Talca
- ✓ Universidad de Concepción
- ✓ Universidad del Bío-Bío
- ✓ Universidad de La Frontera
- ✓ Universidad Austral de Chile
- ✓ Universidad de Los Lagos

En la Figura 7 se puede apreciar la topología general actual de la red avanzada de Chile (REUNA).

Figura 7

Topología de REUNA

Nota. La Figura muestra la Topología de red de la arquitectura REUNA, cuenta con 18 puntos de conectividad. Tomada de (REUNA, 2019).

2.2.3. Tipo de Red

El tipo de red puede darse por diversos factores, Cantli (1997), manifiesta que pueden ser según su extensión, topología, ancho de banda, estándar, protocolo. Para esta investigación nos estamos enfocando por los protocolos, y estructura de funcionamiento de la red y como parte de ello encontramos las redes tradicionales y las redes definidas por software, por tanto, para ello se considera como dimensión para esta investigación:

- ✓ **Arquitectura de red.**

Es la forma de funcionamiento de la red en base a reglas o protocolos, Cai (2011) indica que las redes tradicionales mantienen unido el plano de datos y el de control, mientras que las redes definidas por software separan estos planos con el fin de tener una mejor gestión de red, un indicador que ayudará a medir es:

Instancia de control. Se refiere al plano de control que operan en los dispositivos de red; por lo que el control de red se refiere al funcionamiento de la red que es ejecutada a través del plano de control.

2.2.4. Rendimiento de Red

El rendimiento de una red engloba el funcionamiento en general de la red; para IBM (2020), los recursos de hardware y la presión del tráfico suelen producir un efecto significativo en el rendimiento de las aplicaciones de negocio electrónico. Puede utilizar este

tema para obtener información sobre cómo optimizar el rendimiento de la red y ajustar los recursos de comunicaciones del servidor.

El impacto que causan los componentes de la red en el rendimiento suele ser difícil y complejo de medir, esto debido a que el tráfico de la red y el ancho de banda disponibles pueden cambiar frecuentemente y se ven afectados por influencias sobre las que el administrador del sistema no tiene un control directo, por tanto, entre estos factores consideraremos los que puedan ser medibles basadas en las herramientas que se pueda tener a disposición y en base al escenario que se plantea considerando que no se tiene tantos dispositivos para la implementación completa de manera tradicional HDN y menos con los dispositivos que soporten los protocolos de la implementación de manera programable SDN de la arquitectura de red avanzada; tomaremos los que sean viables, ya que si bien es cierto en las redes implementadas de manera tradicional tenemos opciones definidas ya para poder recolectar datos, sin embargo en una red programable SDN debemos construir o buscar la forma de que estas herramientas puedan ser soportadas, ya que los mismos dispositivos no soportan aun los protocolos SDN, por tanto se a continuación se definirá tres dimensiones para esta investigación con sus indicadores:

- ✓ **Retardo de red**

Es el resultado de variaciones que pueden ocurrir en la red, según González (1999), el retardo de un sistema de comunicaciones de datos está compuesto por 3 factores principales que son:

Retardo de transmisión. Es el tiempo que la red tarda desde el transmisor hasta el receptor; en transportar un bloque de información; depende del tamaño del paquete y de la velocidad de transmisión.

Retardo de propagación. Tiempo que un dispositivo de red tarda en efectuar una operación sobre la información del paquete, desde que le llega hasta ser enviado.

Variación de retardo (jitter). Es la diferencia, de valores de retardos de los paquetes en la red, que pueden ser de extremo a extremo.

- ✓ **Pérdida de paquetes**

Según (Mejía Fajardo, 2004) “Es un fenómeno común en todas las redes conmutadas por paquetes como las redes IP. En particular, no se establecen circuitos físicos entre

extremos y los paquetes provenientes de diferentes fuentes se almacenan en colas en espera de ser transmitidos por el enlace de salida de cada enrutador. Un paquete que llega se pierde en la red si no encuentra espacio en la cola. Mientras más personas accedan la red, los enrutadores más se congestionan y se produce la pérdida de paquetes", para ello el indicado que puede ayudar en la pérdida de paquetes en la red es:

Cantidad de Pérdida de paquetes. Se considera el envío de paquetes desde un punto a otro donde no llegan todos o la caída de una conexión. Por tanto, la pérdida de paquetes se refiere a la cantidad de paquetes que cayeron entre un origen y un destino, de todos los paquetes que fueron enviados dentro de la red de datos en mención.

- ✓ **Consumo de recursos computacionales**

Según, (Mazza, 2014)"Se consideran recursos computacionales a todos aquellos que intervienen en el desarrollo y/o despliegue de soluciones basadas en hardware, software, telecomunicaciones, servicios, etc". Por lo que el consumo de recursos computacionales se refiere a la cantidad que se utiliza de la disponibilidad de nuestros recursos de software, hardware y entre otros en el proceso de la implementación de los entornos emulados, para ello se debe considerar los indicadores:

Nivel de consumo de CPU. Determina e consumo de CPU durante la implementación y funcionamiento de los entornos emulados, de las redes avanzadas tradicionales y SDN.

Nivel de consumo de memoria. Determina el consumo de RAM durante la implementación y funcionamiento de los entornos emulados.

Nivel de consumo de ancho de banda. Según (Mejía Fajardo, 2004) "el ancho de banda es uno de los factores claves que afectan la calidad de servicio (QoS) en una red; cuanto más ancho de banda es mejor la calidad de servicio". En tanto el ancho de banda implica de manera directa el rendimiento de una red, que al final terminará reflejándose en la calidad de servicio, según Ramos de Santiago (2010) el ancho de banda desde un usuario final es la velocidad que percibe al navegar por Internet o hacer uso de un servicio, por ello el consumo de ancho de banda se refiere al grado que se utilizará de la capacidad de ancho de banda de la red, para medir el ancho de banda podemos considerar el envío o descarga de archivos. donde a través del envío de un archivo o la descarga de este, podemos identificar el consumo de ancho de banda en la red.

III. Emulación de Arquitectura de Red avanzada

Para lograr la emulación de una arquitectura de red avanzada, se realizará en dos partes, una correspondiente a la HDN y la otra correspondiente a la SDN de la Red Nacional Académica Avanzada de Chile (REUNA), debemos considerar algunos puntos que serán estándar para utilizar en ambas emulaciones.

3.1. Especificaciones técnicas de Hardware y Software

Para realizar la emulación de la red avanzada tanto como HDN y SDN, se ha trabajado con 2 equipos físicos que se tiene a disposición; se ha considerado utilizar estos 2 equipos físicos debido a que en la HDN emularemos mediante un IOS de *Cisco* y por tanto un solo equipo no soportaría ejecutar 18 routers considerando adicional las 2 máquinas virtuales que se requiere instalar como host para realizar las pruebas, de igual manera en la SDN en un equipo pondremos a disposición para que se ejecute el script mediante Mininet y en el otro equipo un controlador; en la Tabla 3 se muestra estas especificaciones.

Tabla 3

Especificaciones técnicas de equipos físicos y hardware para la emulación

N°	Equipos	Procesador	RAM	Sistema Operativos	Herramientas
1	Laptop Toshiba	Intel® Core™ i3-3110M CPU @ 2.40 GHz	16 GB	Ubuntu Server 18.04 LTS	GNS3 Mininet, quagga
2	PC Desktop Advance	Intel® Core™ i7-4790 CPU @ 3.60 GHz	16 GB	Ubuntu Server 18.04 LTS	GNS3, Ryu

Nota. La Tabla muestra las especificaciones de software y hardware de los 2 equipos físicos utilizados en la investigación.

3.2. Distribución de parámetros para configuraciones

- ✓ **Distribución de Router-id**

Para los dos escenarios, es decir, tanto como HDN y SDN, se trabajará con los router ID que se muestran en la Tabla 4, estos router-id son identificadores para cada router que además es un parámetro que se requiere en las configuraciones para OSPFv3.

Tabla 4

Distribución de router-id para la emulación

Ciudad	Router - ID
Arica	1.1.1.1
Iquique	2.2.2.2
Antofagasta	3.3.3.3
ESO	4.4.4.4
Calama	5.5.5.5
ALMA	6.6.6.6
La Serena	7.7.7.7
AURA	8.8.8.8
Santiago	9.9.9.9
Valparaiso	10.10.10.10
Rancagua	11.11.11.11
Talca	12.12.12.12
Chillán	13.13.13.13
Concepción	14.14.14.14
Temuco	15.15.15.15
Valdivia	16.16.16.16
Osorno	17.17.17.17
Pto Montt	18.18.18.18

Nota. La Tabla muestra todos los router-id distribuidos para cada router en referencia las ciudades de la arquitectura de red avanzada REUNA, para las dos implementaciones realizadas.

- ✓ **Distribución de Direcciones IPv6**

En la Tabla 5 se especifica las direcciones de red IPv6 y máscara de red para cada enlace entre los router que representa cada ciudad, también es necesario añadir 2 host a los extremos para la recolección de datos que se realizará para nuestros indicadores de cada dimensión de nuestras variables de investigación que permitirán evaluar el rendimiento de la red de arquitectura avanzada para ambas implementaciones una como HDN y otra como SDN.

Tabla 5

Distribución de direcciones IPv6 en los enlaces

Enlace	Red
Arica – Iquique	2001:1234:5678:400::/64
Iquique - Antofagasta	2001:1234:5678:800::/64
Antofagasta – ESO	2001:1234:5678:c00::/64
Antofagasta - Calama	2001:1234:5678:1000::/64
Antofagasta – La Serena	2001:1234:5678:1400::/64
Calama – ALMA	2001:1234:5678:1800::/64
La Serena – AURA	2001:1234:5678:1c00::/64
La Serena – Santiago	2001:1234:5678:2000::/64
Santiago - Valparaiso	2001:1234:5678:2400::/64
Santiago - Rancagua	2001:1234:5678:2800::/64
Rancagua – Talca	2001:1234:5678:2c00::/64
Talca – Chillán	2001:1234:5678:3000::/64
Chillán - Concepción	2001:1234:5678:3800::/64
Concepción - Temuco	2001:1234:5678:3c00::/64
Temuco – Valdivia	2001:1234:5678:4000::/64
Valdivia – Osorno	2001:1234:5678:5000::/64
Osorno – Pto Montt	2001:1234:5678:5400::/64
Host1 – Arica	2001:1234:5678:A400::/64
Host2 – Pto Montt	2001:1234:5678:A800::/64

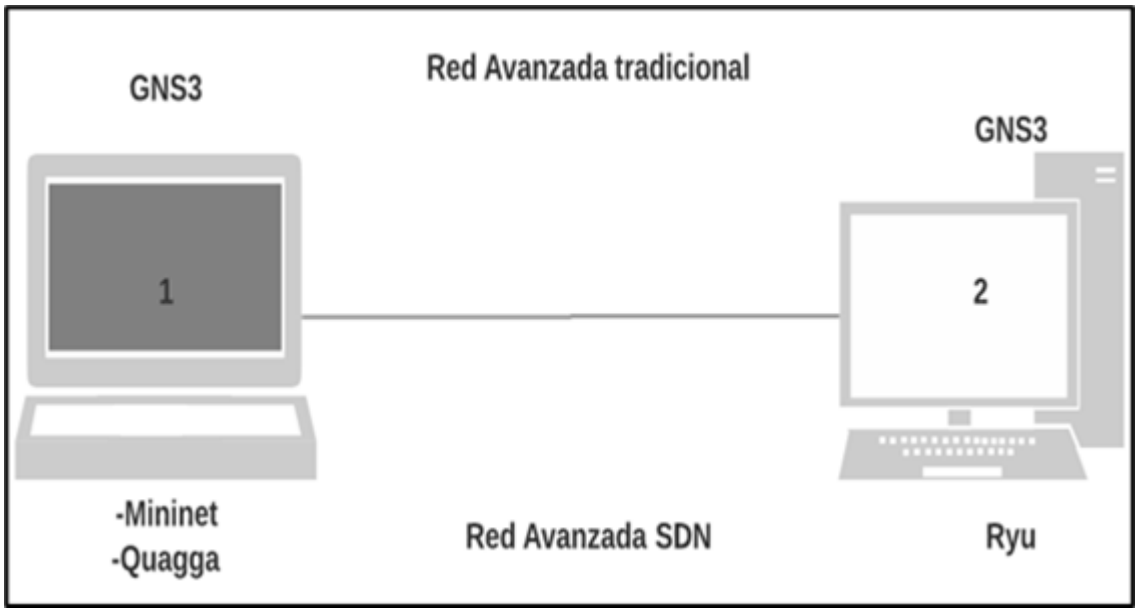
Nota. La Tabla muestra todas las direcciones IPv6 asignadas en todos los enlaces de cada router referente a las ciudades de la arquitectura REUNA.

3.3. Topología Física de red

La topología física de los dos equipos físicos que se implementará para la emulación de la red de arquitectura avanzada con HDN y SDN y las herramientas de emulación que se instalará se muestra en la Figura 8; donde para la HDN la herramienta de emulación es únicamente GNS3 y deben tener conexión directa para poder poner en funcionamiento la topología completa de REUNA; en la SDN se ha distribuido que en el primer equipo es donde se ejecutará el script mediante Mininet con el funcionamiento de quagga y con conexión directa al equipo 2 donde se ejecutará la instancia de control mediante el controlador *Ryu*.

Figura 8

Topología Física de red



Nota. La Figura muestra la topología física de red implementada para el desarrollo de esta investigación.

3.4. Emulación de Arquitectura de Red Avanzada tradicional HDN

3.4.1. Instalación de Herramienta GNS3

Para la emulación de la red avanzada tradicional se instaló en los 2 equipos físicos la versión estable de la herramienta GNS3, para instalar GNS3 se debe seguir los siguientes pasos desde la terminal:

- ✓ **Agregar repositorio**

  add-apt-repository ppa://gns3/ppa

- ✓ **Actualizar las fuentes de repositorio**

  apt-get update

- ✓ **Instalar GNS3 con GUI**

  apt-get install gns3-gui

- ✓ **Instalar dynamips**

  apt-get install dynamyps

- ✓ **Configurar permisos para el usuario servidor**

  user mod -aG sudo servidor.

3.4.2. Emulación de REUNA con HDN

Para realizar la emulación debemos considerar las especificaciones de los equipos físicos de la Tabla N° 3, lo usual sería que de los 18 routers se distribuya equitativamente 9 routers en cada equipo; sin embargo para esta distribución se ha considerado que el equipo 1 es una laptop con un procesador de 2.4 GHZ y el segundo equipo es una PC que tiene un procesador más alto de 3.60 GHZ; por tanto de acuerdo a los recursos de procesador al ser 18 routers se ha considerado que en el equipo físico 1 se implementará 6 routers pertenecientes a Arica, Iquique, Antofagasta, ESO, Calama, ALMA y el host1 que se conectará al router Arica, y en el dispositivo físico 2 se implementará los 12 routers pertenecientes a La Serena, AURA, Santiago, Valparaiso, Rancagua, Talca, Chillán, Concepción, Temuco Valdivia, Osorno, Pto Montt y el host2;.

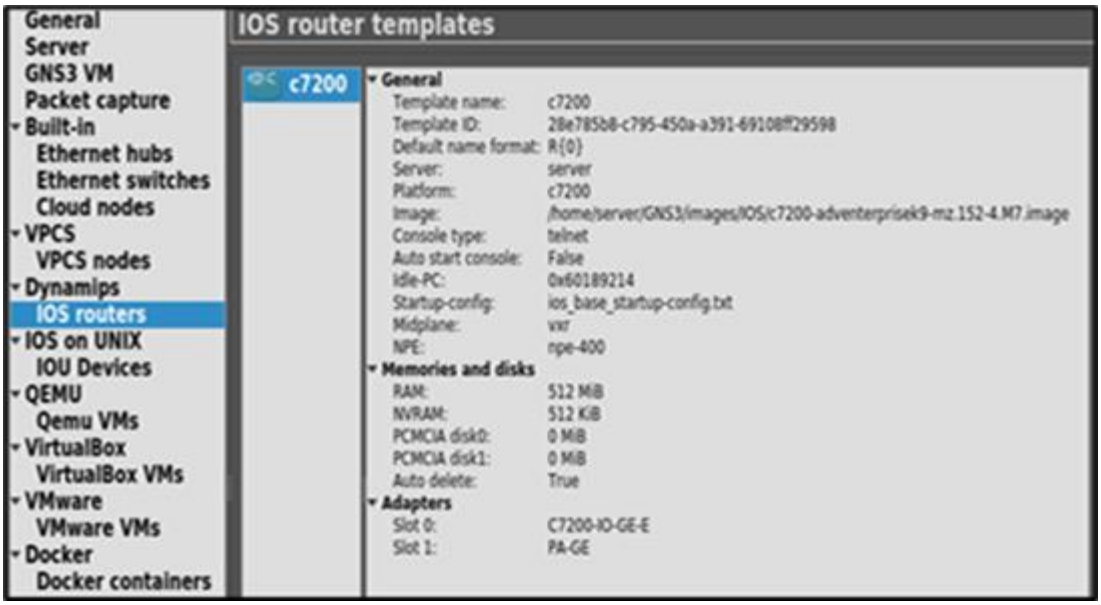
Debido a que ya se cuenta con conocimientos de estas configuraciones, se procedió a realizar la emulación directamente de la topología REUNA, en base a la Tabla N° 4 y Tabla N° 5, y para realizar este escenario utilizaremos el IOS Cisco C7200, para esto añadimos el IOS a GNS3.

- ✓ **Añadir IOS a GNS3**

Iniciamos la herramienta GNS3, abrimos la pestaña de *preferences* y nos dirigimos a la sección *Dynamyps* en la opción IOS routers y agregamos el IOS, en la Figura 9 se muestra la descripción del IOS r*outer* C7200 que se añadió, también se configura inicialmente los slots los cuales sirven para añadir las interfaces del router, los cuales son en este caso son GigabitEthernet.

Figura 9

Especificaciones de IOS router C7200



Nota. La Figura muestra las características del IOS C7200 de Cisco que se ha utilizado para la implementación HDN.

- ✓ **Construcción de la topología**

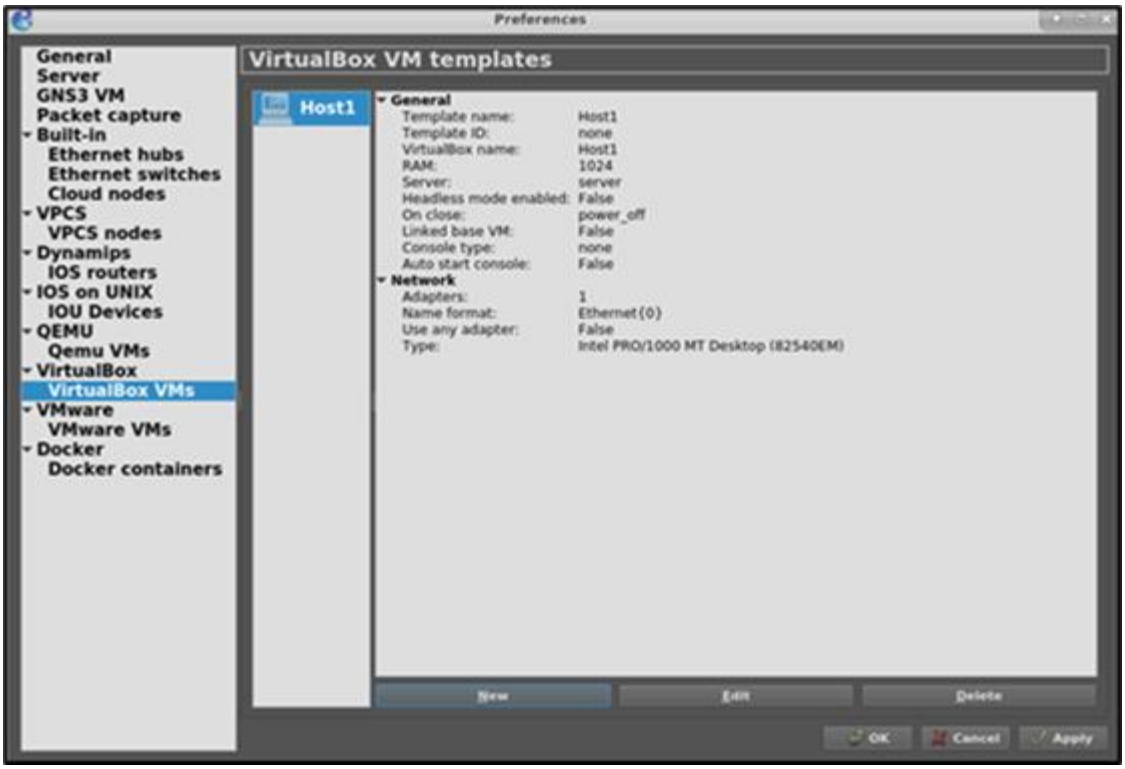
Para construir la topología solo se arrastra el router de nuestro IOS C7200 hacia el área de trabajo y los conectamos con los enlaces, para nuestro caso se ha utilizado interfaces GigabitEthernet, en el **Anexo N° 2** se muestra todo el proceso de configuración de GNS3 para la emulación de la Red Avanzada Tradicional.

Después de distribuir los 18 routers en ambos equipos se procedió a añadir los host virtuales, para esto se ha creado 2 máquinas virtuales (VM) una en cada equipo; el primero tiene como nombre host1 el cual se encuentra en el equipo 1 y el segundo es host2 que se encuentra en el equipo 2; se les ha asignado 1024 MB de RAM y 10 GB de espacio; el host1 y host2 son máquinas virtuales con sistema operativo Ubuntu Desktop 18.04 que se ha instalado a través de VirtualBox donde solo se ha configurado IPv6; GNS3 tiene la capacidad de añadir estas máquinas virtuales en la pestaña *preferences* sección *VirtualBox* en la opción *VirtualBox* VMs nos aparecerá la lista de las máquinas virtuales que tenemos disponibles previa instalación, dentro del equipo 1 que corresponde a la primera parte de la topología de REUNA que se está emulando, en la Figura 10 observamos la VM host1 que se encuentra

dentro del equipo 1 y que se ha añadido en el GNS3; debemos tener en cuenta que las interfaces de las máquinas virtuales deben encontrase como *Generic Driver*.

Figura 10

VM Host1 en GNS3

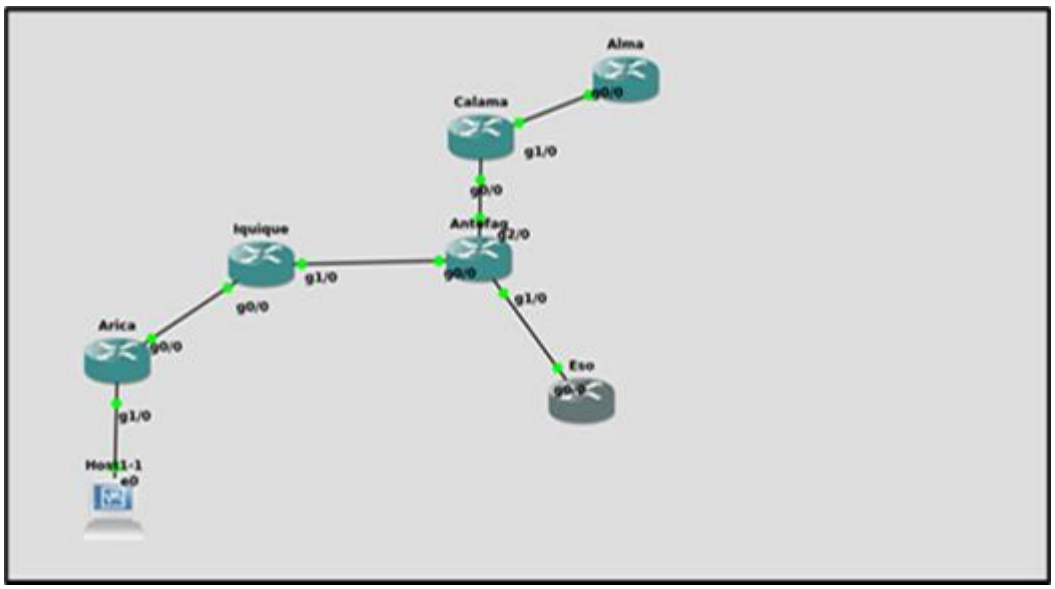


Nota. La Figura muestra la Máquina Virtual correspondiente al extremo inicial Host1 de la topología REUN en HDN.

Seguidamente de añadir todos los routers respectivos a cada ciudad y los hosts al proyecto creado en GNS3; la Figura 11 nos muestra la primera parte de nuestra la topología de la arquitectura de red avanzada REUNA, según se ha designado que se implementará dentro del equipo 1.

Figura 11

Primera parte de topología REUNA en GNS3

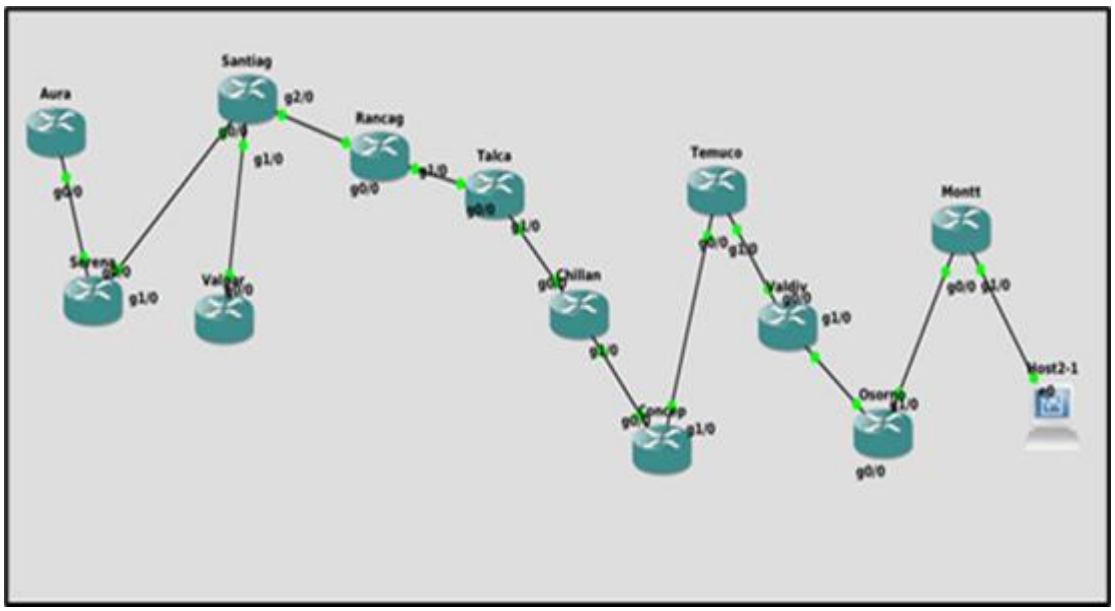


Nota. La Figura muestra la primera parte de la Topología REUNA en GNS3.

En la siguiente Figura 12, podemos observar la otra parte de la construcción de la topología de nuestra arquitectura de red avanzada REUNA, el cual es implementado dentro del equipo 2.

Figura 12

Segunda parte de topología REUNA en GNS3



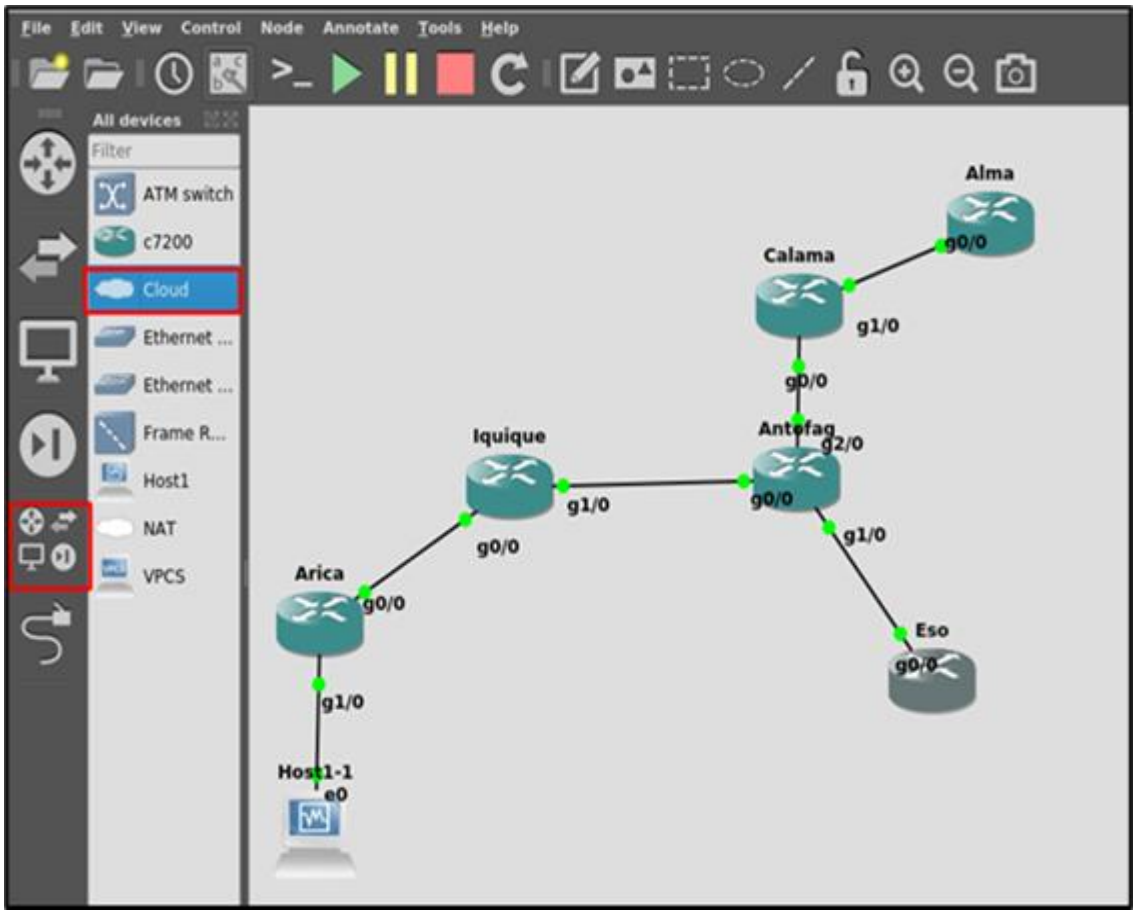
Nota. La Figura muestra la segunda parte de la topología REUNA en GNS3.

Debido a que la emulación de la red de arquitectura avanzada se ha realizado en 2 equipos tiene que existir conexión entre estos2 equipos; para esto el equipo 1 tiene configurada la dirección IPv6 2001:1234:5678:1400::10 y el equipo 2 la dirección IPv6 2001:1234:5678:1400::20; y la conexión para que GNS3 tenga conectividad en los 2 equipos se realiza mediante un enlace *cloud*; para esto consideramos que el enlace que une estas 2 partes de la topología REUNA es entre el router perteneciente a Antofagasta y el router Serena; para ello debemos añadir dicho enlace *cloud* en cada equipo en el enlace mencionado.

La Figura 13 nos muestra la sección en donde encontramos el enlace *cloud*; para lo cual debemos arrastrar hacia cualquier punto de la topología que requerimos hacer conexión y realizamos la configuración.

Figura 13

Enlace Cloud para unir las 2 partes de REUNA en GNS3

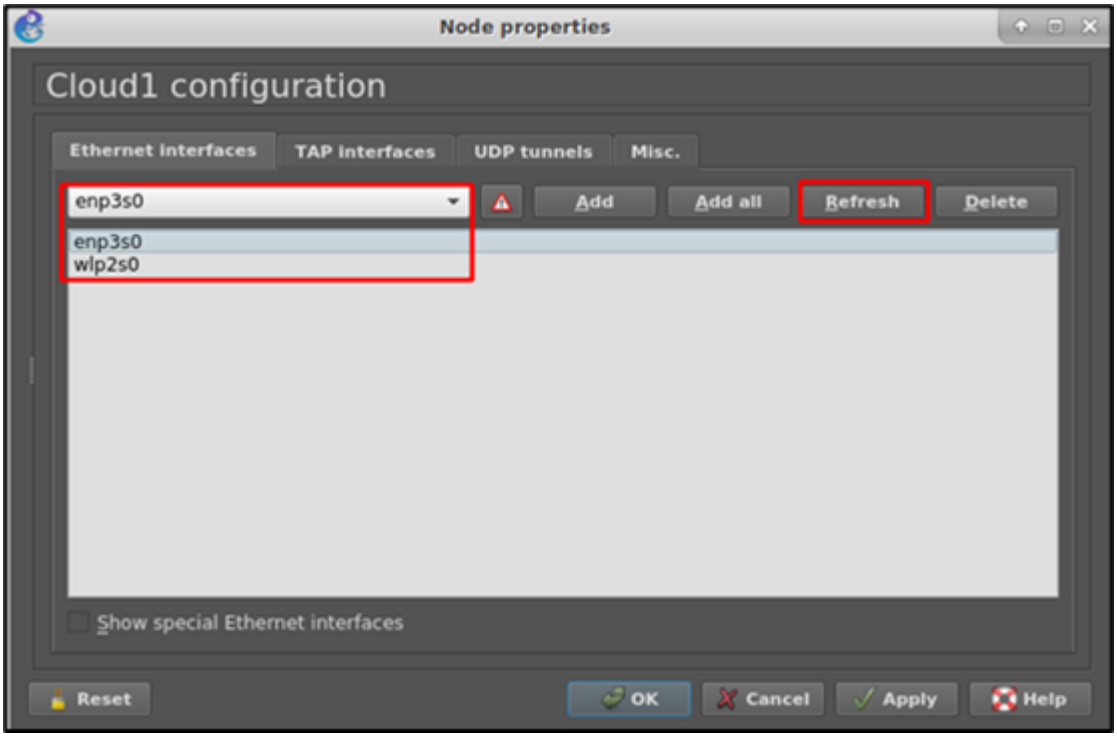


Nota. La Figura muestra el enlace cloud utilizado para unir las implementaciones de parte y parte 2 de REUNA a finde que exista conectividad.

En la Figura 14 se observa la configuración del enlace *cloud* realizado en el equipo 1 el cual será enlazado con el router Antofagasta, para acceder hacemos *click* derecho sobre el *cloud* y seleccionamos la opción configure; se aprecia que el equipo 1 tiene 2 interfaces; seleccionamos la interfaz enp3s0 el cual corresponde a la interface física para cableado, la otra interfaz que muestra es de Wi-Fi el cual no es recomendado para trabajar en esta conexión dentro de distribuciones GNU/Linux.

Figura 14

Configuración de interface para Cloud

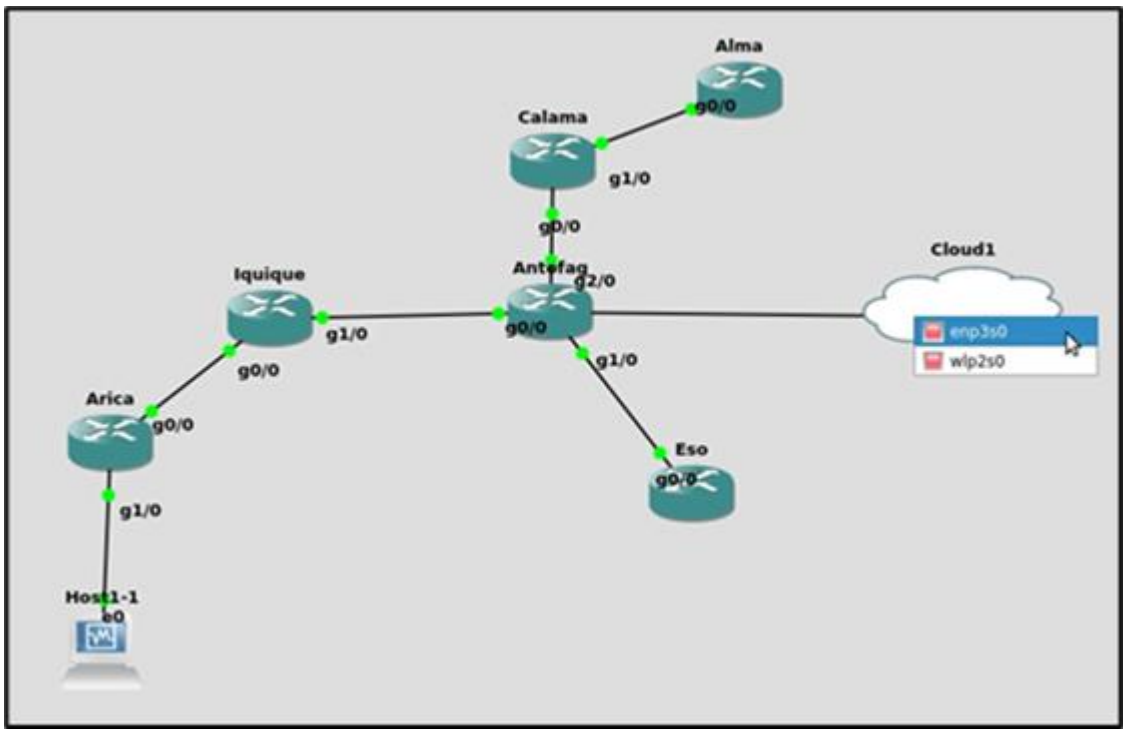


Nota. La Figura muestra que al momento de configurar el enlace Cloud aparecerán las interfaces físicas del equipo.

Después de configurar adecuadamente el enlace *cloud*; debemos enlazarlo con el router Antofagasta, ya que este es el router donde ocurre la conexión cloud que llegará al otro equipo con el router Serena; para esto seleccionamos un enlace y unimos el router Antofagasta mediante una interfaz GigabitEthernet que se ha añadido y conectamos al enlace cloud en la interface enp3s0 que hemos detectado y configurado previamente; la Figura 15 nos muestra este proceso que existe.

Figura 15

Enlace Cloud en primera parte de la topología REUNA

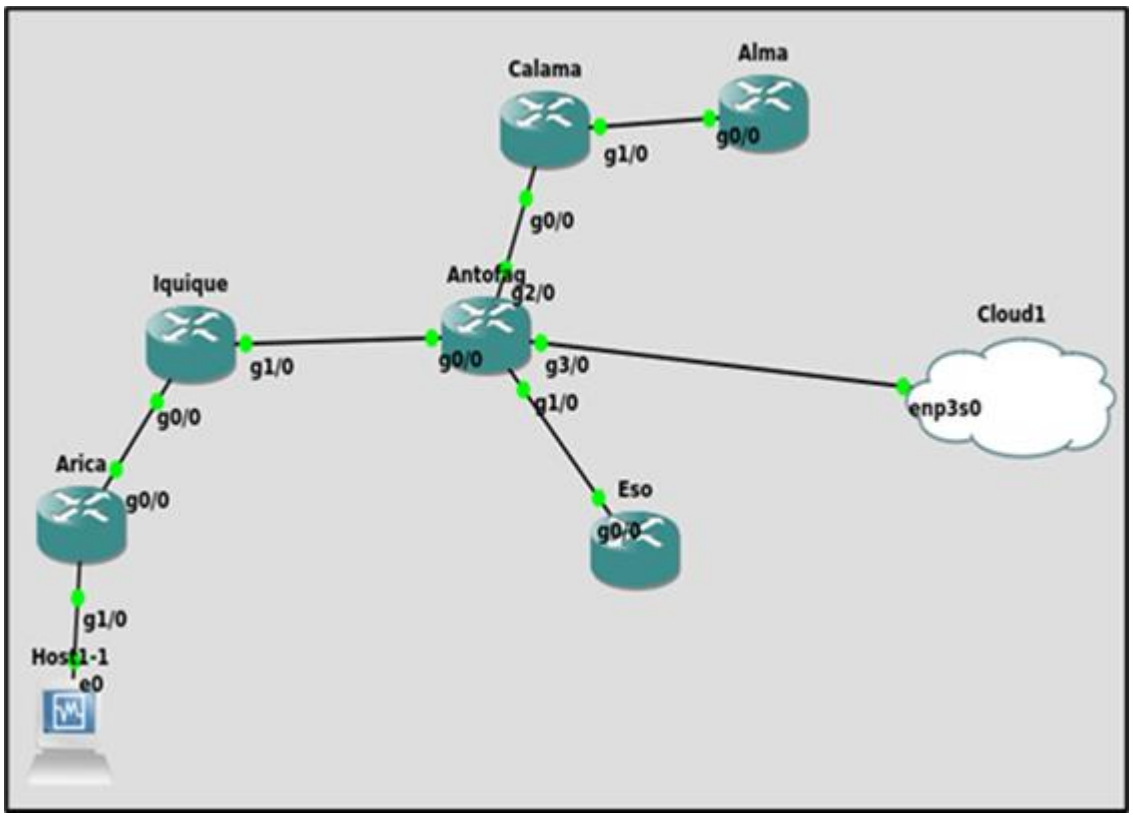


Nota. La Figura muestra el enlace cloud a seleccionar para hacer puente con el otro enlace cloud.

Después de tener añadido todos los dispositivos; en la Figura 16 se observa la construcción de la primera parte de topología en el equipo 1 con los 6 routers respectivos, y en la Figura 17 se aprecia la segunda parte de la topología construida en el equipo 2 que contiene los 12 routers correspondientes, donde también se añadió el enlace cloud para poder establecer comunicación, la interfaz que se tiene aquí es la enx00e04c534458.

Figura 16

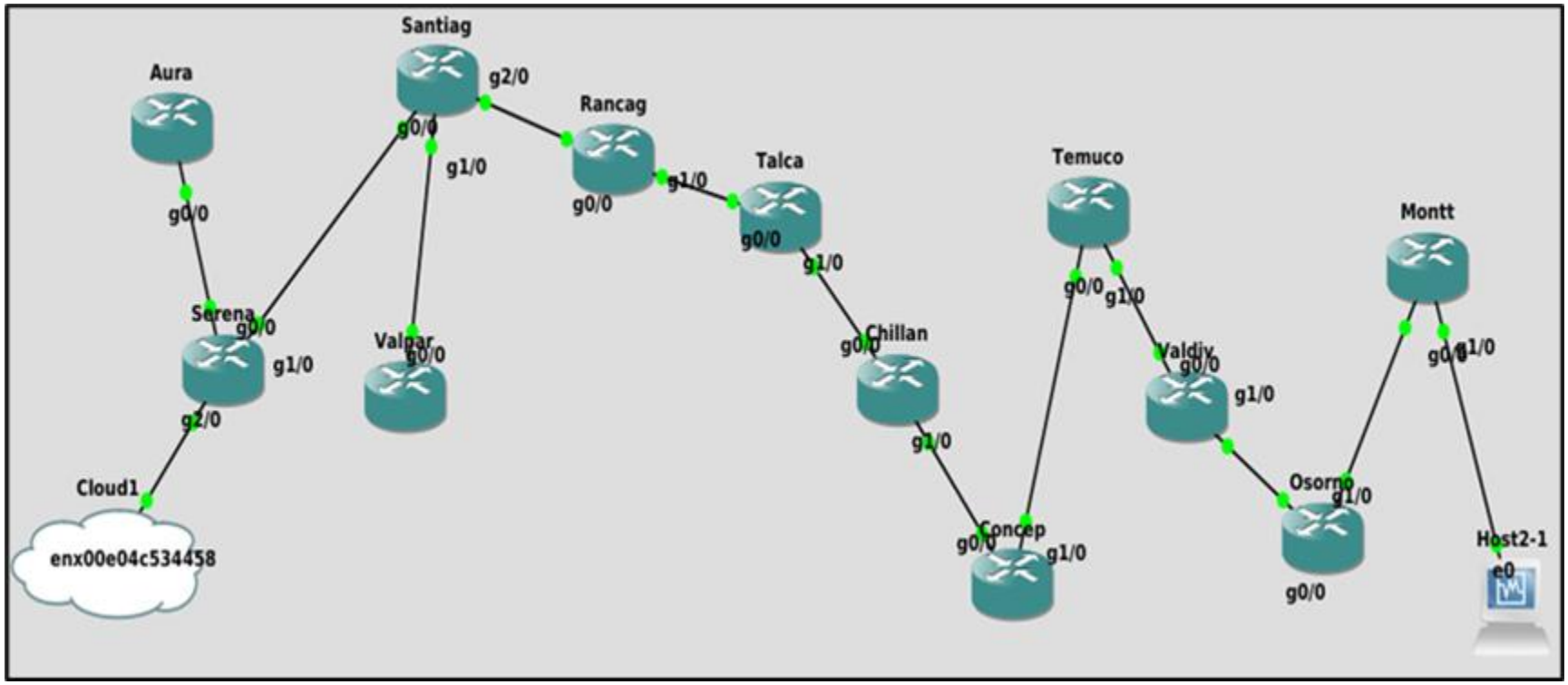
Primera parte de topología REUNA en GNS3 con enlace cloud



Nota. La Figura muestra la primera parte de la topología REUNA funcionando, la cual se ha implementado de manera tradicional HDN y el enlace cloud que conectará a la segunda parte.

Figura 17

Segunda parte de topología REUNA en GNS3 con enlace cloud



Nota. La Figura muestra la segunda parte de la topología REUNA funcionando, la cual se ha implementado de manera tradicional HDN y el enlace cloud que conectará a la primera parte.

Si realizamos la comparación de la topología real de arquitectura de red avanzada REUNA con la topología de red emulada que hemos implementado en GNS3, se tiene las siguientes observaciones:

- ✓ La Emulación ha sido desarrollada en 2 equipo físicos que se conectan a través de un *cloud*.
- ✓ En a la topología real hay varias conexiones entre ciudades que contienen más de un enlace, en la emulación todos los enlaces entre routers se ha considerado solo uno, esto debido a los recursos que contamos, y teniendo en cuenta que al conectar más de un enlace entre nodos lo que estamos cubriendo es calidad de servicio (QoS), y balanceo de carga; para evaluar el rendimiento de nuestra topología no se considera necesario ya que es una emulación y estamos haciendo redundancia.
- ✓ Todos los enlaces entre todos los router C7200 son interfaces Gigabit Ethernet.
- ✓ Los nombres de algunos routers se han abreviado, esto para tener un estándar con SDN que no permite agregar más de siete caracteres a un dispositivo.

Para conectarse el dispositivo físico 1 y el dispositivo físico 2 y tener la topología completa de la arquitectura de red avanzada REUNA lo hacemos a través de un enlace *cloud*, el cual a través de las interfaces físicas de los dispositivos permite hacer un puente entre ambos equipos, en la conexión que existe entre el router Antofagasta y La Serena es donde se realiza este puente y los dispositivos físicos deben estar conectados en la misma red y con conexión directa cableada, ya que por red inalámbrica GNS3 en GNU/Linux tiene problemas.

Configuración del router

Una vez armada nuestra topología debemos realizar los siguientes pasos:

- ✓ Encender haciendo *click* derecho en el router y ejecutando la opción *start*.
- ✓ Una vez iniciado, todas nuestras interfaces conectadas al router se encenderán, debemos hacer *click* derecho sobre el router y ejecutar la opción *console*, que es la interfaz consola donde se realizará las configuraciones.
- ✓ Finalmente se abre la interfaz consola y aquí realizaremos la configuración que se muestra a continuación, para este caso el router Iquique

Habilitamos IPv6

Iquique(config)# ipv6 unicast-routing

Asignamos un id al protocolo OPF

Iquique(config)# ipv6 router ospf 10

Asignamos el router ID correspondiente

Iquique(config-rtr)# router-id 2.2.2.2

Configuramos IPv6 y OSPF con área única en las interfaces

Iquique(config)# interface gigabitEthernet 0/0

Iquique(config-if)# ipv6 add 2001:1234:5678:400::2/64

Iquique(config-if)# ipv6 ospf 10 area 0

Iquique(config)# interface gigabitEthernet 1/0

Iquique(config-if)# ipv6 add 2001:1234:5678:800::1/64

Iquique(config-if)#¿ ipv6 ospf 10 area 0

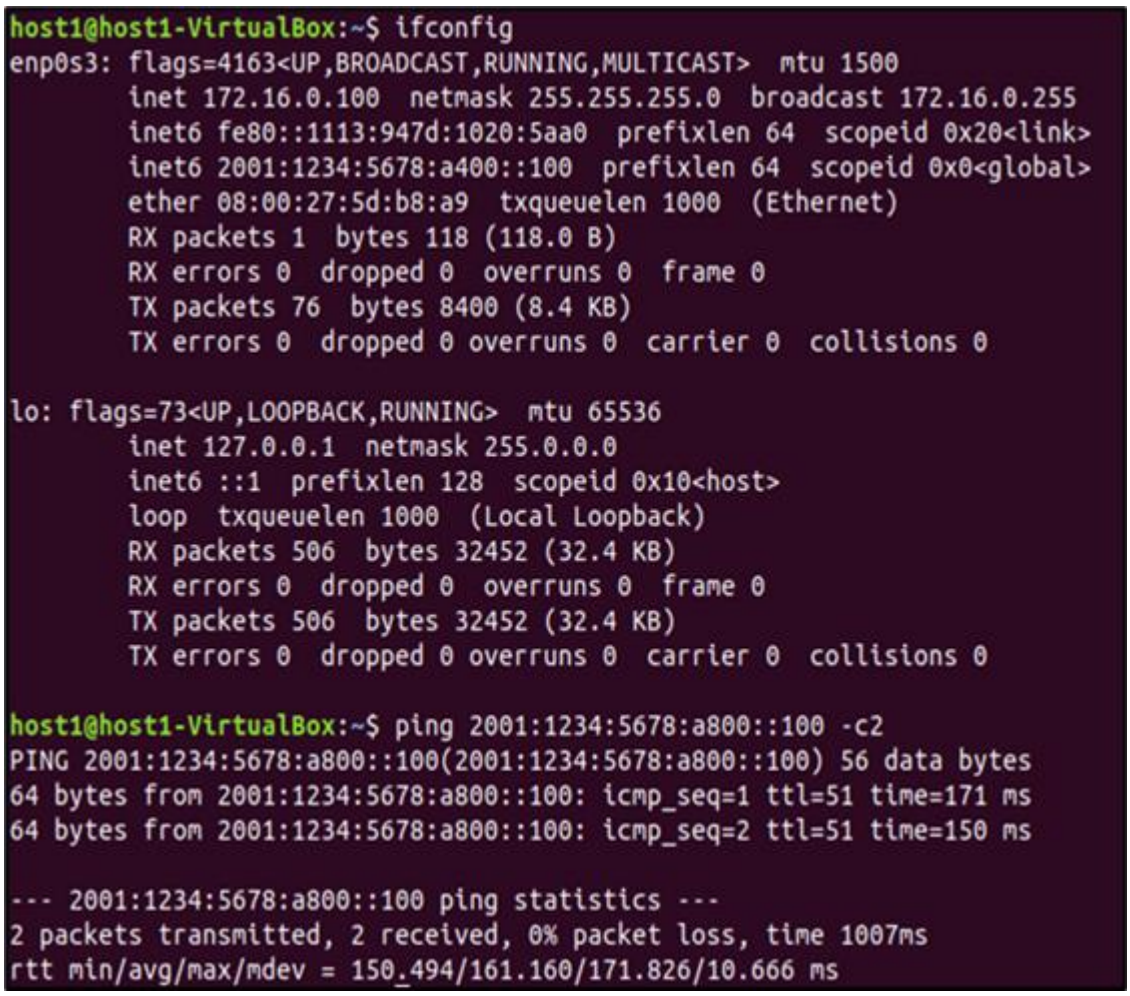
Después de configurar adecuadamente las interfaces de cada router, podemos limpiar los procesos y verificar automáticamente las redes adyacentes con el comando *clear opsfv3 process*, hasta este punto ya debemos tener conectividad extrema.

3.4.3. Prueba de Conectividad

Se realizó pruebas de conectividad entre todos los routers mientras se realizaba la configuración, en una parte final se empezó a realizar ping desde el host1 hacia todos los demás dispositivos; en la Figura 18 se observa el ping de conectividad extrema desde el host1 hasta el host2.

Figura 18

Prueba de conectividad entre h1 y h2 de red Tradicional REUNA



Nota. La Figura muestra la prueba de conectividad ejecutada de extremo a extremo entre el host 1 y el host 2 mediante ping.

3.5. Emulación de Arquitectura de Red Avanzada Programable SDN

Para emular la red de arquitectura avanzada con SDN, se tomará en cuenta la topología física que se muestra en la Figura 8, donde se instalará las herramientas mencionadas para lograr la dicha implementación.

Una vez creado nuestro script dentro del equipo 1 debemos mandarlo hacia el controlador que se encuentra en el quipo 2.

3.5.1. Instalación de Mininet

Mininet es la herramienta que nos permitirá crear nuestra topología SDN a través de un script que se ejecutará en el dispositivo físico 1.

Para instalar *Mininet*, se ha tomado la guía de instalación desde la página oficial de *Mininet* que se encuentra en http://mininet.org/download/, para ello primero clonamos el repositorio y luego instalamos la herramienta; la secuencia de comandos a seguir en la terminal es:

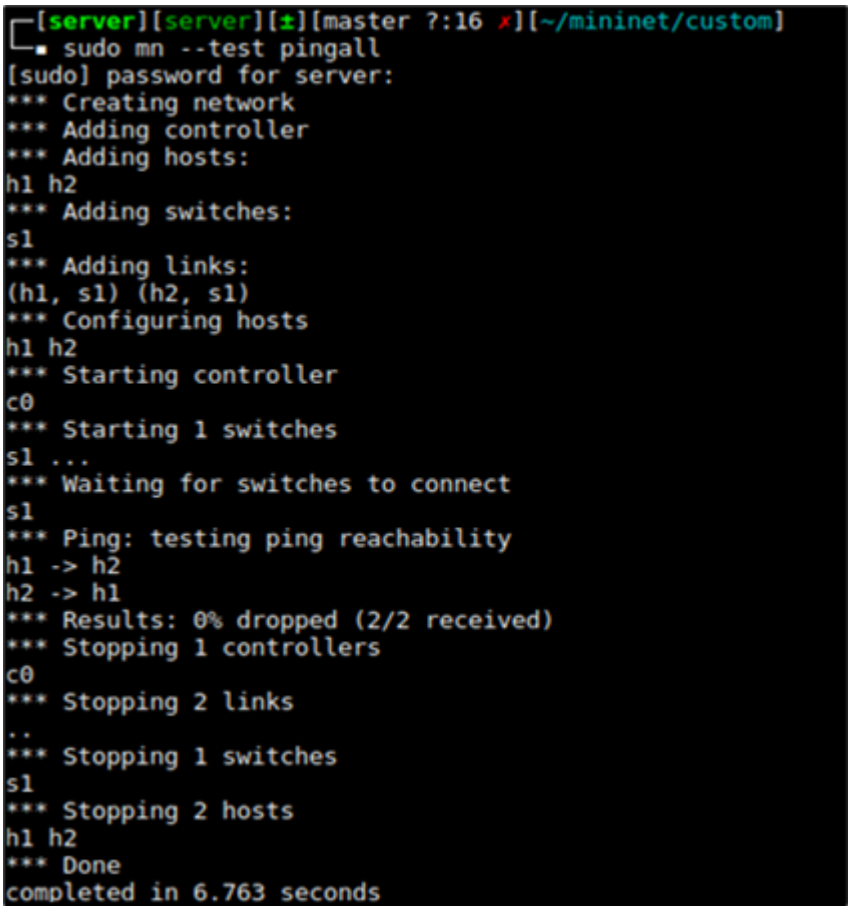
- ✓ **Clonar el repositorio de Mininet**
  git://github.com/mininet/Mininet

- ✓ **Instalar Mininet**
  mininet/util/install.sh -a

La versión de *Mininet* instalada es 2.3, debemos tener en cuenta que es necesario tener instalado el lenguaje Python dentro del equipo; en los sistemas operativos bajo GNU/Linux viene instalado por defecto, preciso es indicar que la versión adecuada que utilizamos para toda esta investigación es la 2.7.17 de Python, después de instalar *Mininet* se procedió a verificar que esta herramienta esté funcionando adecuadamente; para ello utilizamos un test que nos trae *Mininet* el cual conecta un switch con dos hosts, tal como se observa en la Figura 19 la prueba de conectividad que fue ejecutada mediante el comando sudo mn –test pingall es satisfactoria, la cual significa que el switch con los dos host tienen conectividad; ahora ya podemos crear nuestro propio script para la implementación de la red de arquitectura avanzada como SDN.

Figura 19

Test de funcionamiento de Mininet



Nota. La Figura muestra la prueba de funcionamiento de Mininet ejecutada.

3.5.2. Instalación de Quagga

Para lograr conectividad completa se requiere realizar enrutamiento, para esto se ha determinado que se trabajará con OSPF, teniendo en cuenta que para nuestra investigación al ser una red de arquitectura avanzada debemos utilizar OSPFv3 para IPv6, por ello después de realizar una recopilación de herramientas; se determinó utilizar *Quagga* que es un proyecto *OpenSource* que contiene paquetes para la implementación de protocolos de enrutamiento dinámicos que funcionan a través de un demonio principal llamado *zebra* que es el que hace la labor de reenvío; se puede configurar a través de una interfaz consola *VTY*, pero para nuestro caso debido a que son 18 routers, crearemos los archivos de configuración que serán llamados y utilizados en nuestro *script* que se creará y ejecutará en el dispositivo físico 1.

Quagga tiene varias versiones, las cuales hemos ido probando instalar, para nuestra investigación se ha realizado la instalación de *quagga* en su versión 1.2.1, la cual funciona correctamente, para ello descargamos quagga desde savannah.gnu.org y descomprimimos dentro de nuestro servidor, seguidamente entramos en la carpeta de *quagga* para previamente instalar algunas bibliotecas que requiere dicha implementación; posteriormente debemos configurar nuestro usuario y el grupo al cual pertenece este usuario para que pueda realizar las funciones de *quagga*, se podría crear otro usuario específico con su grupo, pero para evitarnos cualquier incidencia dentro del proceso se ha decidido utilizar el usuario root; después de estos pasos realizados se procedió a instalar el paquete que nos trae *quagga* a través de la herramienta *make* como se aprecia a continuación.

- ✓ **Descomprimir el archivo quagga**

  tar -xvzf quagga-1.2.1.tar.gz

- ✓ **Instalar bibliotecas dentro de la carpeta quagga**

  apt-get install gawk

  apt-get install libreadline6 libreadline6-dev

  apt-get install libc-ares-dev

- ✓ **Configuración del usuario y grupo**

  ./configure --enable-vtysh

  --enable-user=root

  --enable-group=root

  --enable-vty-group=root

- ✓ **Instalar quagga dentro de la carpeta quagga**

  make

  make install

- ✓ **Copiar las bibliotecas de zebra y ospf**

  server@server:/usr/local/lib$ sudo cp libzebra.* /lib

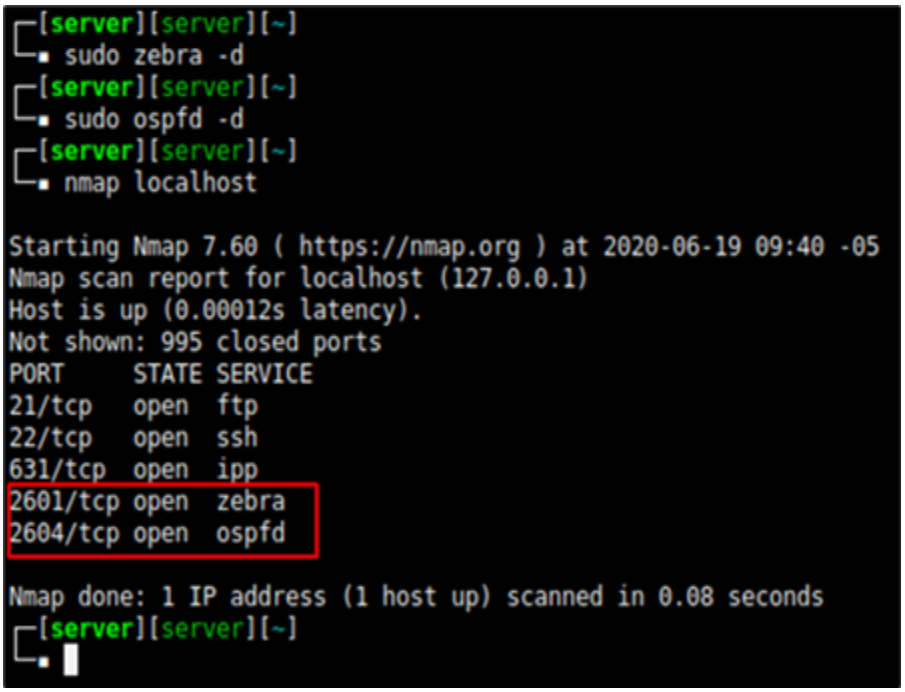
  server@server:/usr/local/lib$ sudo cp libospf.* /lib

Después de haber instalado, debemos utilizar los archivos que nos proporciona quagga, nos dirigimos a la ruta /usr/local/etc y encontraremos un archivo zebra.conf.sample el cual

es el archivo que ejecuta el demonio zebra para proporcionar reenvío, la extensión .sample indica que es una muestra, por tanto a fin de conservar este archivo original debemos copiar este archivo a un nuevo archivo zebra.conf, ya que el demonio zebra utiliza los archivos con extensión .conf; asimismo copiamos el archivo ospfd.conf.sample a otro archivo ospfd.conf con la misma intención de conservar el principal y que zebra utilice este nuevo achivo .conf, finalmente iniciamos los servicios digitando el servicio -d y verificamos que estos se encuentren ejecutándose, esto se aprecia en la Figura 20.

Figura 20

Ejecución de servicios zebra y ospfd



Nota. La Figura muestra la prueba de funcionamiento de protocolos Zebra y OSPF.

3.5.3. Instalación de Ryu

En esta emulación se ha determinado utilizar *ryu* que es un proyecto *OpenSource* el cual realiza las funciones de controlador SDN ya que contiene componentes que admiten protocolos como *OpenFlow*, *Netconf*, *Of-config*. Para nuestro caso la implementación de nuestra red de arquitectura SDN es mediante el protocolo *OpenFlow*.

En el equipo 2 para tener el funcionamiento de *ryu* es necesario clonar el repositorio, e instalar algunas bibliotecas que requiere *ryu*; a continuación, se muestra la instalación adecuada de *ryu*; donde es necesario tener instalado Python el cual para nuestro caso viene instalado por defecto la versión 2.7.17.

- ✓ **Clonar el repositorio**

  git clone https://github.com/faucetsdn/ryu.git

- ✓ **Instalar bibliotecas dentro de la carpeta Ryu**

  python ./setup.py install

  pip install six –upgrade

  pip install oslo.config msgpack-python

  pip install eventlet --upgrade

- ✓ **Actualizar e Instalar Ryu**

  apt-get update -y

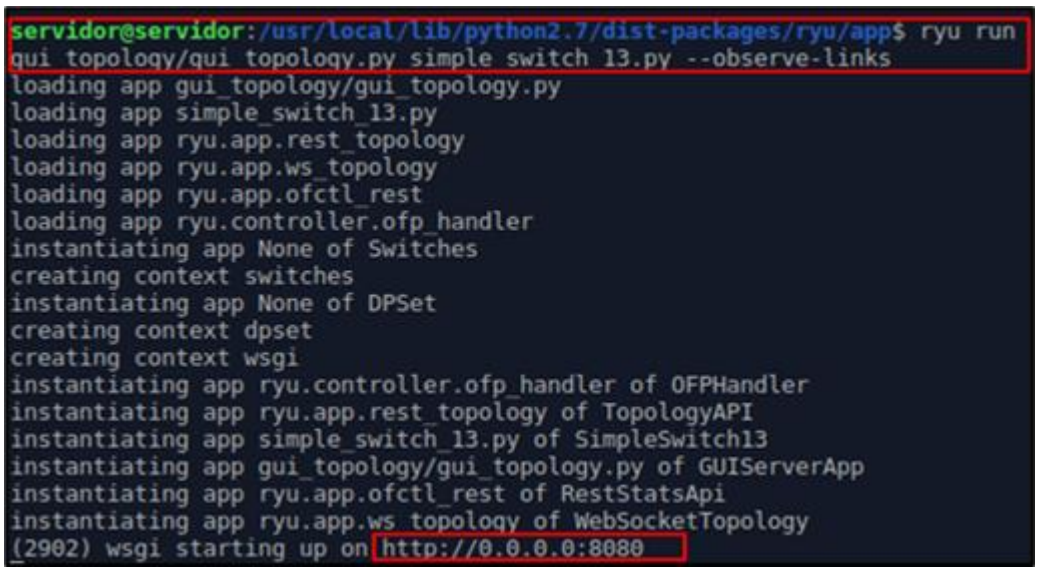
  apt-get install -y Python-ryu

  pip install.

La versión instalada de ryu es la 4.34; para verificar que el controlador se encuentra funcionando correctamente, ejecutamos el comando con el protocolo OpenFlow 1.3 tal como se aprecia en la Figura N° 21, donde después de ejecutarlo nos indica que debemos acceder a nuestro localhost en el puerto 8080 para poder tener una visualización de la topología en nuestro controlador.

Figura 21

Inicio de ejecución de Ryu



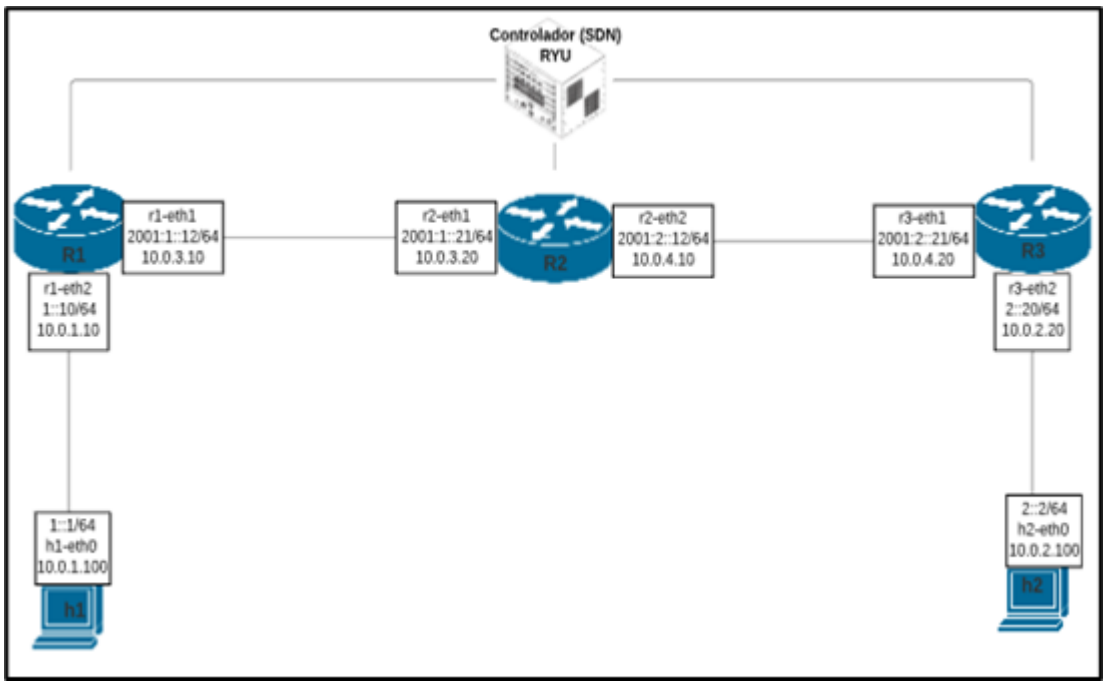
Nota. La Figura muestra la prueba de funcionamiento del controlador Ryu.

3.5.4. Emulación de Prueba OSPFv3

Para poder realizar la emulación de REUNA en arquitectura de red SDN, inicialmente se ha requerido de investigar sobre las clases, métodos y parámetros que proporciona Mininet con el objetivo de poder programar la red; debido a que no se contaba con conocimientos suficientes para realizar esta programación, se procedió a realizar algunas pruebas verificando los scripts que generan algunas topologías creadas en Mininet a través de la funcionalidad que proporciona miniedit.py; donde solo se puede utilizar dispositivos de capa 2 como switchs, sin embargo, se requiere del funcionamiento de routers en nuestra topología; por ello se realizó distintas pruebas; en el **Anexo 3** se muestra una prueba final donde se logró hacer funcionar adecuadamente una topología básica de prueba con IPv4 y OSPFv2. Sin embargo nuestra red requiere de direcciones IPv6 y para ello implementar OSPFv3; para esto se procedió a programar topologías de prueba para OSPFv3, en la Figura 22 se observa una topología de prueba final que se ha diseñado para dar continuidad a la emulación de la topología de la red avanzada académica de Chile REUNA, en la cual a raíz de complicaciones de que Mininet aún no soporta programar IPv6 de la misma manera que IPv4, es decir al crear una interfaz de red Mininet no permite asignarle directamente IPv6, sino que se debe realizar ingresando a la interfaz como si estuviéramos accediendo a nuestra terminal de un host a cambiar IPv6, solo que el enfoque es programado; por ello es que en esta prueba se requirió implementar tanto IPv4 como IPv6 al mismo tiempo.

Figura 22

Topología de prueba SDN con OSPFv3



Nota. La Figura muestra la topología de Prueba para IPv6 con OSPFv3.

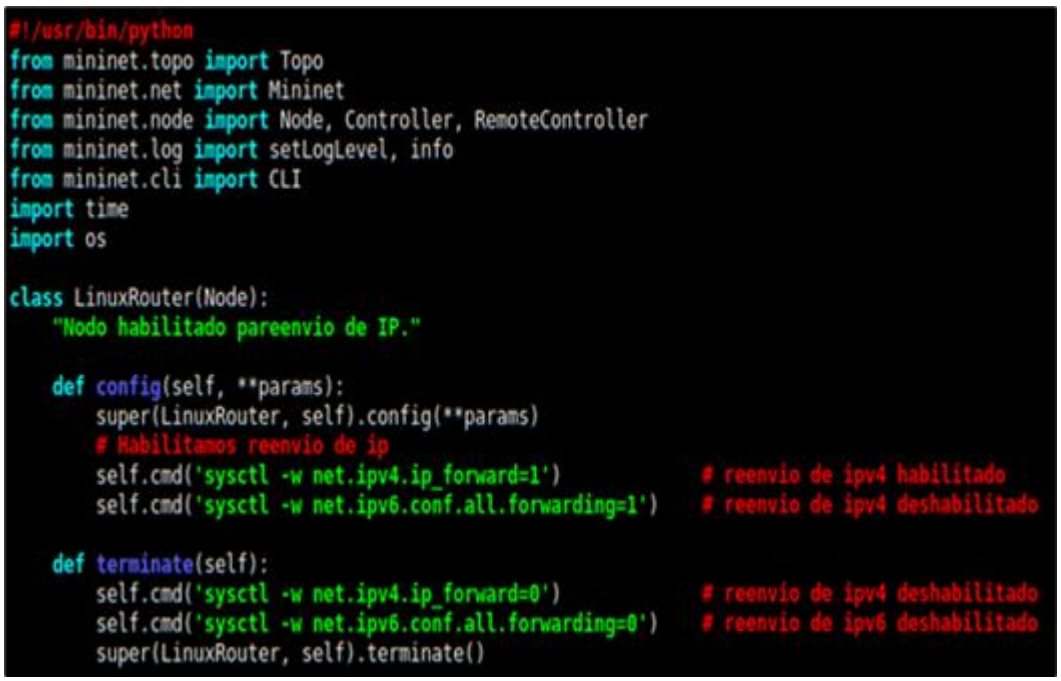
Consideramos que el script y todos los archivos que requiere quagga funciona en el equipo 1 y el controlador ryu en el equipo 2; para esto se ha creado un script en Python con el nombre SDNv6.py, inicialmente se trató de hacer funcionar netamente IPv6 con OSPFv3 teniendo muchas complicaciones ya que reiterando lo mencionado, Mininet no permite agregar directamente IPv6 como si se ha realizado para IPv4. En las siguientes Figuras se explica los segmentos de código y las dificultades que se ha tenido y como se ha dado solución.

En la Figura 23 se aprecia la primera parte del segmento de código implementado; donde se realiza la importación de bibliotecas y objetos que nos proporciona Mininet el cual permitirá crear nuestra topología de red, donde el objeto *Topo* contiene información de la estructura de la red que estamos programando, el objeto *Mininet* es donde se ejecutarán las acciones de la red que estemos programando, el objeto *Node* es el que permite definir un nodo en la implementación de nuestra red, otro objeto que se ha importado es *RemoteController* que nos permite que toda la topología creada sea ejecutada a través de un controlador ya sea local o remoto, otros objetos muy importante que se ha utilizado son *setLogLevel* que nos permite crear logs de la ejecución de nuestro script, el objeto *info* nos

permite visualizar en consola lo que el emulador está realizando durante el inicio y finalización de la emulación, el objeto *CLI* es el que permite la comunicación con los nodos y además nos permite utilizar la interfaz *CLI*, la funcionalidad *time* es quien nos permite dar tiempo de ejecución al *api* y la funcionalidad *os* nos proporciona utilidad de matar procesos, también utilizamos una clase llamada *LinuxRouter* que permitirá hacer el reenvío de IP, para ello se debe utilizar los métodos *config* donde podemos expresar el renvío de IP como estado 1 habilitado y en el método *terminate* le indicamos 0 como deshabilitado, uno de los problemas que se tenía era el reenvío para IPv4 es a través de *sysctl net.ipv4.ip_forward=1*, lo lógico es que para ipv6 simplemente se habilitara el reenvío a través de *sysctl net.ipv6.ip_forward=1*, sin embargo eso no es correcto, pero revisando se intentó realizar de forma manual en el archivo *sysctl.conf* de nuestro sistema operativo Linux en general, según (Ferguson, 2019) la línea adecuada para poder hacer reenvío de *ipv6* es *sysctl -w net.ipv6.conf.all.forwarding=1*, todo esto se realiza dentro de un método *config()* y para deshabilitar el estado se cambia a 0 en un método *terminate()*, estos métodos son capaces de leer la clase *LinuxRouter* que trae *Mininet*.

Figura 23

Segmento 1 de código SDN con OSPFv3



```
#!/usr/bin/python
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Node, Controller, RemoteController
from mininet.log import setLogLevel, info
from mininet.cli import CLI
import time
import os

class LinuxRouter(Node):
    "Nodo habilitado pareenvio de IP."

    def config(self, **params):
        super(LinuxRouter, self).config(**params)
        # Habilitamos reenvio de ip
        self.cmd('sysctl -w net.ipv4.ip_forward=1')            # reenvio de ipv4 habilitado
        self.cmd('sysctl -w net.ipv6.conf.all.forwarding=1')   # reenvio de ipv4 deshabilitado

    def terminate(self):
        self.cmd('sysctl -w net.ipv4.ip_forward=0')            # reenvio de ipv4 deshabilitado
        self.cmd('sysctl -w net.ipv6.conf.all.forwarding=0')   # reenvio de ipv6 deshabilitado
        super(LinuxRouter, self).terminate()
```

Nota. La Figura muestra la primera parte del código SDN para la prueba con OSPFv3.

En la segunda parte del código que se observa en la Figura 24, donde se ha creado la clase con nombre SDNv6 el cual recibe como parámetro Topo; y a través de método build nos permitirá construir la emulación de nuestra topología hasta que se termine de ejecutar, se ha asignado una dirección IPv4 por defecto a los router esto solo a fin de darle un parámetro inicial a las interfaces, creando los enlaces y definiendo el Gateway que permitirá conectividad entre redes, los switch son definidos en Mininet por ello para agregarlos se realiza a través de la propiedad self.addSwitch, y se ingresa al menos un parámetro, que sería el nombre del Switch, aunque también se puede agregar un identificador pid, también para un host se debe agregar a través de la propiedad selfaddHost con parámetros igual al Switch; sin embargo, no existe formas de crear router, por ello la necesidad de integrar quagga, que a través de los protocolos le dará la funcionalidad de router a un nodo, por ello se agrega como self.addNode que es una propiedad de Mininet que tiene como funcionalidad crear nodos, y a estos nodos los podemos dar la funcionalidad que se requiere, incluso como host o switch, pero para nuestro caso se requiere darle la funcionalidad de router, esta propiedad también recibe como parámetro el nombre y a la vez se le asigna la IP por defecto que tendrá, para construir los enlaces como es lógico deben ser entre 2 dispositivos, es decir, entre host-router o entre router-switch y entre switch-host, lo lógico entonces es pasar dos parámetros

asociados a los 2 dispositivos, donde el valor del primero se designa como intfName1 y el segundo como intfName2, la propiedad intfName nos permite darle el nombre al enlace y también añadir IPv4 al dispositivo que le asignaremos inicialmente sin necesidad de configurar enrutamientos, solo para que se creen los enlaces y luego poder añadir IPv6, esto a través de los parámetros params1 y params2 que para algunos casos de la topología lo requiere y los otros vienen dados por el defaultIP que se ha designado, otra función que podemos apreciar es la de defaultRoute que permite indicar cual es la dirección Gateway para un segmento de red; se aprecia que primero se ha definido los enlaces de cada nodo que posteriormente serán utilizarlos, esto debido a que no se puede generar directamente las interfaces para ipv6, por ello se decidió trabajar en el mismo script de red para las configuraciones IPv4 e IPv6, se observa la parte de la asignación de ipv4, también se ha creado los enlaces y se define los Gateway para la salida de los host, una manera lógica sería agregar aquí todo la parte de configuración IPv6 en cada enlace y es lo que se hizo en primera instancia, pero esto no funciona, y la razón es porque Mininet no ofrece configuración directa de IPv6, es decir se intentó pasar como parámetros las direcciones IPv4 supliendo a las IPv4, sin embargo esto no es correcto, y para ello debemos programar estas asignaciones de IPv6, para esto debemos abstraer lo que normalmente se haría para cambiar la IPv6 de un terminal, pero lo tenemos que realizar dentro del script

Figura 24

Segmento 2 de código SDN con OSPFv3

```
class SDNv6(Topo):
    '''
    h1 h1-eth0:r1-eth2
    h2 h2-eth0:r3-eth2
    r1 r1-eth1:r2-eth1 r1-eth2:h1-eth0
    r2 r2-eth1:r1-eth1 r2-eth2:r3-eth1
    r3 r3-eth1:r2-eth2 r3-eth2:h2-eth0
    '''

    def build(self, **opts):
        defaultIP1 = '10.0.3.10/24'  # Add IP default para router
        defaultIP2 = '10.0.3.20/24'
        defaultIP3 = '10.0.4.20/24'
        router1 = self.addNode('r1', cls=LinuxRouter, ip=defaultIP1)
        router2 = self.addNode('r2', cls=LinuxRouter, ip=defaultIP2)
        router3 = self.addNode('r3', cls=LinuxRouter, ip=defaultIP3)

        h1 = self.addHost('h1', ip='10.0.1.100/24', defaultRoute='via 10.0.1.10')  # Gateway
        h2 = self.addHost('h2', ip='10.0.2.100/24', defaultRoute='via 10.0.2.20')

        self.addLink(router1, router2, intfName1='r1-eth1', intfName2='r2-eth1')   # Enlaces
        self.addLink(router2, router3, intfName1='r2-eth2',params1={ 'ip' : '10.0.4.10/24' },intfName2='r3-eth1')
        self.addLink(h1, router1, intfName2='r1-eth2', params2={'ip': '10.0.1.10/24'})
        self.addLink(h2, router3, intfName2='r3-eth2', params2={'ip': '10.0.2.20/24'})
```

Nota. La Figura muestra la segunda parte del código SDN para la prueba con OSPFv3.

En la siguiente Figura 25 que corresponde a la tercera parte del código es donde se programa la ejecución de la topología a través de la clase *run*; esta topología creada es la que incluye la clase SDNv6 y que es llamada mediante propiedad topo; también es aquí donde se envía toda la funcionalidad hacia el controlador a través de la propiedad RemoteController; nuestro controlador ryu se encuentra en la IP 2001:1234:5678:1400::20 es enviado a través del puerto 6633, además se le indica que inicie con la propiedad start; debemos tener en cuenta que para los routers necesitamos llamarlos en esta sección con la propiedad getNodeByName haciéndole pasar como parámetro el nombre que se le ha declarado al nodo para que pueda utilizar la configuración OPFv3; y es aquí donde recién realizamos todas las asignaciones de IPv6.

Inicialmente se probó ejecutando la topología y ejecutando el comando *ifconfig* para agregar ipv6 logrando las asignaciones de IPv6, sin embargo, esto sería algo que se debería realizar cada vez que se ejecuta el *script*, lo cual no tendría sentido de ser una red programada.

La razón de no poder implementar IPv6 de la misma forma que IPv4 es porque *Mininet* no soporta directamente una configuración ipv6, sin embargo, al crear los nodos es posible ingresar y asignar a los enlaces y asignar las direcciones IPv6 a cada interface de cada nodo a través con la propiedad cmd que permite insertar escribir y leer archivos, para esto se realizó a través de la propiedad ifconfig y es aquí se aprovecha en asignar las direcciones IPv6, así como definir el *Gateway* para los host esto a través de la propiedad route, previamente se debe reconocer los nodos que han sido definidos en la clase SDNv6 a través de la función *net.getNodebyName()* que pasa como parámetro el nombre asignado tal como la prueba anterior realizada.

Figura 25

Segmento 3 de código SDN con OSPFv3

```
def run():
    "Test linux router"
    topo = SDNv6()
    net = Mininet(controller=RemoteController, topo=topo)
    c1 = net.addController('c1', ip='192.168.8.150', port=6633)
    net.start()
    info('*** Routing Table on Router:\n')

    r1 = net.getNodeByName('r1')
    r2 = net.getNodeByName('r2')
    r3 = net.getNodeByName('r3')
    h1 = net.getNodeByName('h1')
    h2 = net.getNodeByName('h2')

    '''
    Configure ipv6 for interface.
    '''
    r1.cmd('ifconfig r1-eth1 inet6 add 2001:1::12/96') # Add IP default en routers
    r2.cmd('ifconfig r2-eth1 inet6 add 2001:1::21/96')
    r2.cmd('ifconfig r2-eth2 inet6 add 2001:2::12/96')
    r3.cmd('ifconfig r3-eth1 inet6 add 2001:2::21/96')

    r1.cmd('ifconfig r1-eth2 inet6 add 1::10/96') # Add IP las demas interfaces
    h1.cmd('ifconfig h1-eth0 inet6 add 1::1/96')
    h1.cmd('route -6 add default gw 1::10 dev h1-eth0')  # Gateway para h1
    r3.cmd('ifconfig r3-eth2 inet6 add 2::20/96')
    h2.cmd('ifconfig h2-eth0 inet6 add 2::2/96')
    h2.cmd('route -6 add default gw 2::20 dev h2-eth0')  # Gateway para h2
```

Nota. La Figura muestra la tercera parte del código SDN para la prueba con OSPFv3.

En la Figura 26, se aprecia el último segmento de código, donde se crea los socket para OSPFv3 a través de *ospf6d*, debemos tener en cuenta que los servicios *zebra* y *ospf6d* deben estar ejecutándose esto mediante los comandos servicio -d, un detalle es que al realizar el escaneo de puertos deben aparecer ejecutándose el 2601 referente a *zebra*, el 2604 para *ospfd* y el 2606 que es para *ospf6d* que no aparece en el escaneo, sin embargo al realizar una

conexión remota a través del protocolo *telnet* a este puerto se puede acceder, lo que significa que *ospf6d* si se está ejecutando, entonces es momento de revisar los logs, lo que finalmente ocurre es que zebra se encarga de acceder a *ospf6d* para utilizar los archivos rxosp6d.conf, lo mismo que ocurre para ospfd solo que el servicio se muestra como tal en el escaneo de puertos que se realizó a través de la herramienta *Nmap* que para tenerlo disponible basta con instalarlo a través del comando *apt-get install Nmap*; entonces lo que hace es el llamado a zebra, el cual es el demonio principal que proporciona reenvío para que pueda crear los sockets que son las interfaces de programación de aplicación para los protocolos TCP/IP, esto se debe hacer para cada router con la propiedad cmd que es la que lee y crea archivos; una vez creado el socket entonces zebra puede utilizar la configuración de los archivos de cada router, es de utiliza los "rxzebra.conf", donde x define el número del router; una vez que el script reconoce a zebra, se crea un socket que consiste en una extensión .api y otra .interface que son los que finalmente harán que ospf6d lea la configuración de los archivos rxospf6d.conf, donde x representa el número del router.

Figura 26

Segmento 4 de código SDN con OSPFv3

```
    info('Iniciando zebra ospfd ospf6d service:\n')

    r1.cmd('zebra -f /usr/local/etc/r1zebra.conf -d -z ~/Desktop/Sockets/r1zebra.api -i ~/Desktop/Sockets/r1zebra.interface')
    time.sleep(1)  # Aqui zebra crea los api socket
    r2.cmd('zebra -f /usr/local/etc/r2zebra.conf -d -z ~/Desktop/Sockets/r2zebra.api -i ~/Desktop/Sockets/r2zebra.interface')
    r3.cmd('zebra -f /usr/local/etc/r3zebra.conf -d -z ~/Desktop/Sockets/r3zebra.api -i ~/Desktop/Sockets/r3zebra.interface')
    # ospfd - ospfv2
    r1.cmd('ospfd -f /usr/local/etc/r1ospfd.conf -d -z ~/Desktop/Sockets/r1zebra.api -i ~/Desktop/Sockets/r1ospfd.interface')
    r2.cmd('ospfd -f /usr/local/etc/r2ospfd.conf -d -z ~/Desktop/Sockets/r2zebra.api -i ~/Desktop/Sockets/r2ospfd.interface')
    r3.cmd('ospfd -f /usr/local/etc/r3ospfd.conf -d -z ~/Desktop/Sockets/r3zebra.api -i ~/Desktop/Sockets/r3ospfd.interface')
    # ospf6d - ospfv3
    r1.cmd('ospf6d -f /usr/local/etc/r1ospf6d.conf -d -z ~/Desktop/Sockets/r1zebra.api -i ~/Desktop/Sockets/r1ospf6d.interface')
    r2.cmd('ospf6d -f /usr/local/etc/r2ospf6d.conf -d -z ~/Desktop/Sockets/r2zebra.api -i ~/Desktop/Sockets/r2ospf6d.interface')
    r3.cmd('ospf6d -f /usr/local/etc/r3ospf6d.conf -d -z ~/Desktop/Sockets/r3zebra.api -i ~/Desktop/Sockets/r3ospf6d.interface')

    CLI(net)
    net.stop()
    os.system("killall -9 ospfd ospf6d zebra")
    os.system("rm -f *api*")
    os.system("rm -f *interface*")


if __name__ == '__main__':
    setLogLevel('info')
    run()
```

Nota. La Figura muestra la cuarta parte del código SDN para la prueba con OSPFv3.

En las siguientes Figuras se aprecia el archivo de configuración de *r2ospf6d.conf*, donde a diferencia de ospfv2, debemos configurar varios parámetros para poder tener un funcionamiento adecuado.

La Figura 27, muestra la primera parte de la configuración OSPFv3, en primera instancia se realizó una configuración propia en base a conocimientos de redes tradicionales el cual no funcionó por razones específicas de parámetros que requiere zebra, sin embargo quagga nos ofrece un archivo *ospf6d.conf.sample* el cual claramente es una muestra de cómo configurar, por ello es que debemos regirnos a los parámetros de esta configuración, la cual consiste en darle una identificación con nombre que para este caso es r2, una contraseña de acceso vty y después de varias pruebas se le dio los parámetros que requiere por cada interfaz de nodo (*router*), entre ellos el costo de 1, el intervalo del paquete *hello* que será cada 10 segundos y cuando se da por muerto en intervalo de 40 segundos, el de tiempo de transmisión de 1 y el intervalo de retransmisión de 4, también la prioridad de 1 y un id de instancia del proceso que será de 0; es necesario declarar la interfaz *loopback* de nuestro equipo, ya que en pruebas anteriores se obvió esto creyendo innecesario pero no funcionó; todos estos parámetros lo indica el archivo ospf6d de muestra que nos trae quagga al momento de instalar.

Figura 27

Segmento 1 de configuración OSPFv3 en r2ospf6.conf

Nota. La Figura muestra la primera parte de la configuración OSPFv3 para el router 2.

En la segunda parte de la configuración OSPFv3 que se observa en la Figura 28, realizamos la configuración de área única en cada interfaz, como también sus métricas, un detalle importante es conservar esta configuración, ya que inicialmente se ignoró varios segmentos teniendo muchas complicaciones, sin embargo, es necesario permitir loopback, Multicast, y la red 2001:1::/ que es la general de todas nuestros segmentos de red, también es necesario dejar la configuración VTY, una herramienta necesaria es los logs, estos nos ha permitido identificar que el protocolo si funciona por medio de zebra.

Figura 28

Segmento 2 de configuración OSPFv3 en r2ospf6.conf

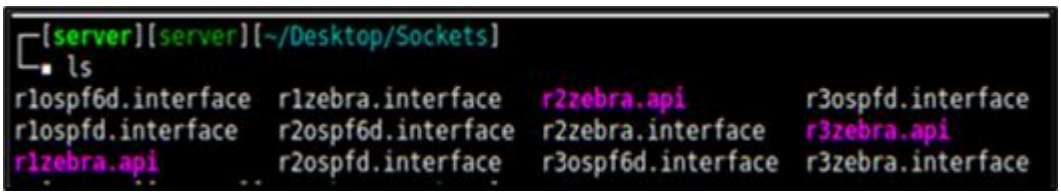
```
router ospf6
 router-id 2.2.2.2
! redistribute static route-map static-ospf6
 interface r2-eth1 area 0.0.0.0
 interface r2-eth2 area 0.0.0.0
!
access-list access4 permit 127.0.0.1/32
!
ipv6 access-list access6 permit 3ffe:501::/32
ipv6 access-list access6 permit 2001:1::/48
ipv6 access-list access6 permit ::1/128
!
ipv6 prefix-list test-prefix seq 1000 deny any
!
route-map static-ospf6 permit 10
 match ipv6 address prefix-list test-prefix
 set metric-type type-2
 set metric 2000
!
line vty
 access-class access4
 ipv6 access-class access6
 exec-timeout 0 0
!
log file /usr/local/etc/r2ospf6d.log
```

Nota. La Figura muestra la segunda parte de la configuración OSPFv3 del router 2.

Después de tener nuestro script completado y los archivos de configuración, ejecutamos el controlador y ejecutamos el script SDNv6.py en *Mininet*, es necesario verificar que se creen todos los socket, esto quiere decir que los archivos rxzebra.api, rxzebra.interface, rxospf6d.interface deben crearse dentro de la ruta que se ha designado /Desktop/Sockets/ después de ejecutar el script, ya que estos permiten el funcionamiento de los protocolos de enrutamientos, en nuestro caso OSPFv3 mediante el demonio zebra como se puede observar en la Figura 29, sin embargo este era un problema permanente que se presentó al desarrollar esta emulación de OSPFv3 ya que no se lograban crear los sockets, al parecer ospf6d no realizaba la funcionalidad, sin embargo como se mencionó era porque se ignoró algunas configuraciones que en base a nuestros conocimientos de redes tradicionales eran innecesarios, en la Figura 29 se aprecia el *socket* creado a partir del demonio *zebra* y las configuraciones OSPF.

Figura 29

Creación de sockets con OSPFv3

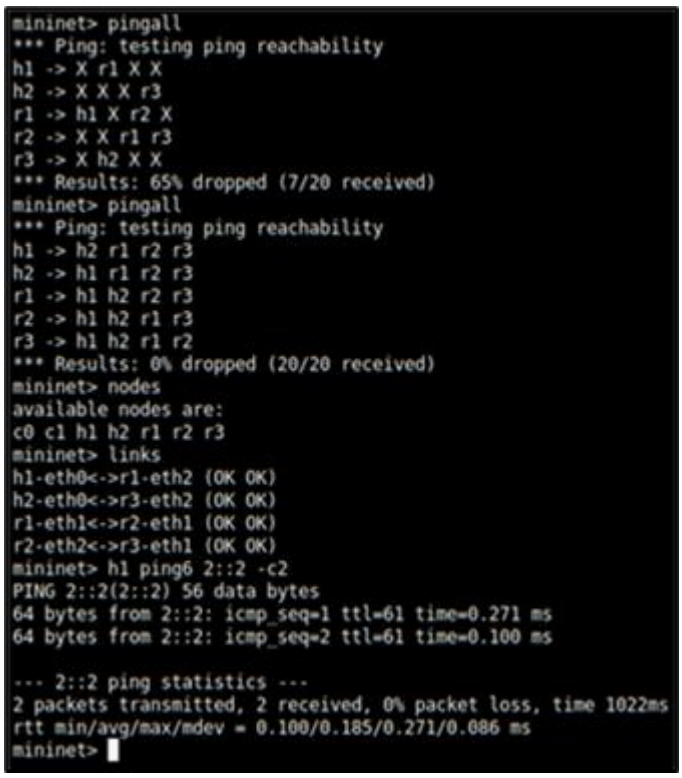


Nota. La Figura muestra la topología de los sockets creados en la topología de prueba.

Una vez ejecutando nuestro *script*, se verificó los nodos creados mediante el comando nodes, asimismo se verificó el estado de todos los enlaces de red que se ha programado a través del comando links; para verificar la conectividad entre los nodos en IPv4 mediante se realiza mediante *pingall con el cual* estamos verificando conectividad entre todos los nodos, sin embargo para verificar conectividad total en ipv6 es necesario realizar un ping6, en nuestra prueba se realizó de extremo a extremo, es decir desde h1 hasta h2, en la Figura 30 se aprecia que se envía dos paquetes de IPv6 desde h1 hasta h2 que tiene como ipv6 asignada la 2::2, el cual llega con éxito.

Figura 30

Prueba de conectividad SDN con OSPFv3



Nota. La Figura muestra la topología de Prueba de conectividad en IPv6 con OSPFv3.

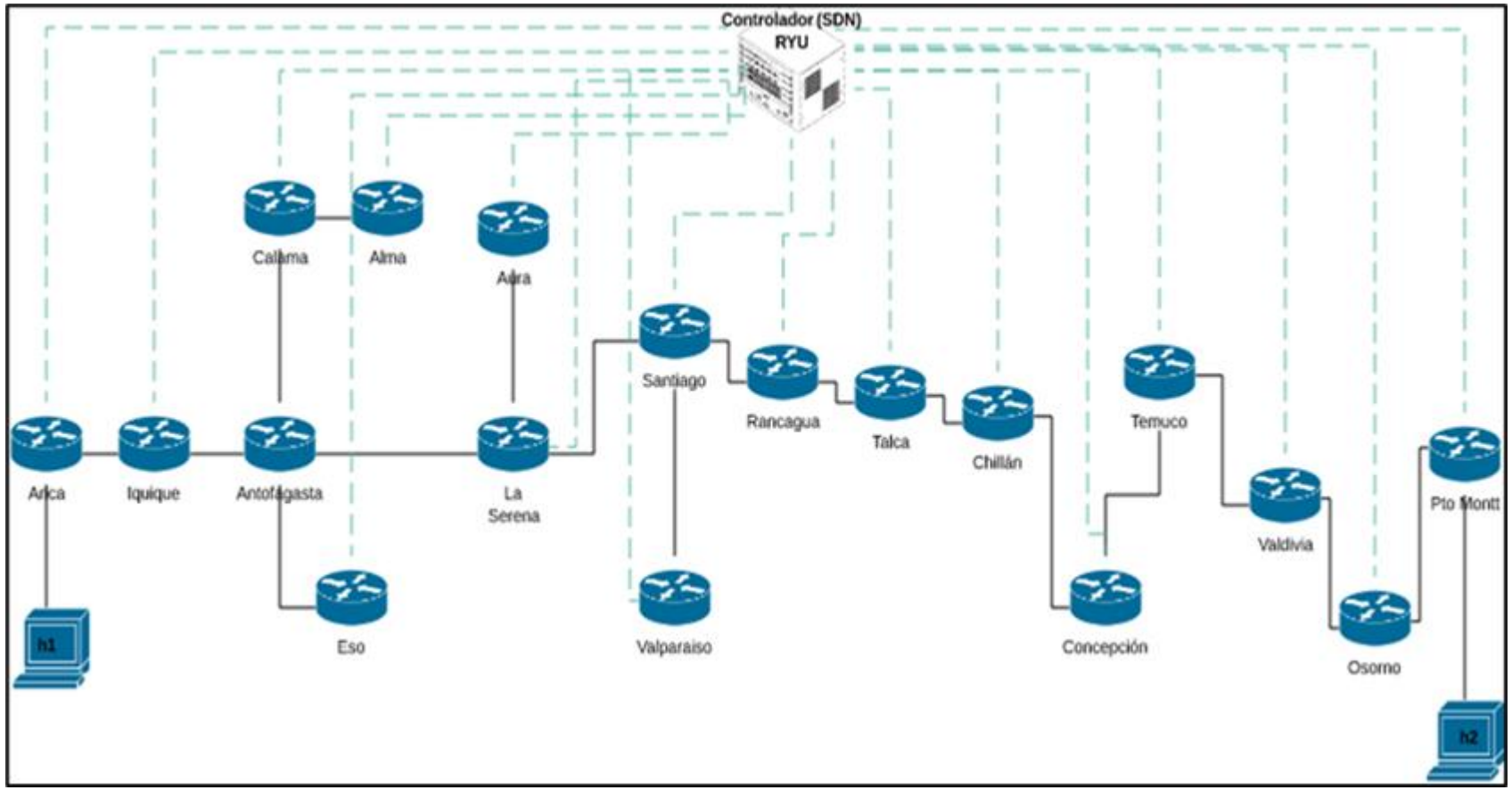
3.5.5. Emulación de REUNA con SDN

Después de haber realizado algunos escenarios de prueba, donde se ha realizado la programación de topologías de red para la implementación del funcionamiento tanto para OSPFv2 que soporta IPv4 y OPFv3 para IPv6 a través de las herramientas que se han instalado; procedemos a emular la red avanzada académica de Chile (REUNA), teniendo en cuenta que para programar la red y la configuración de OSPFv3 debemos considerar la Tabla N° 4 y Tabla N° 5 tal como se realizó en la red tradicional.

En la Figura 31 se aprecia el diseño de la topología de la red avanzada académica e Chile (REUNA) que se ha realizado en base a la topología original de la Figura 4, pero como SDN, el cual incorpora el controlador *Ryu.*

Figura 31

Topología REUNA SDN



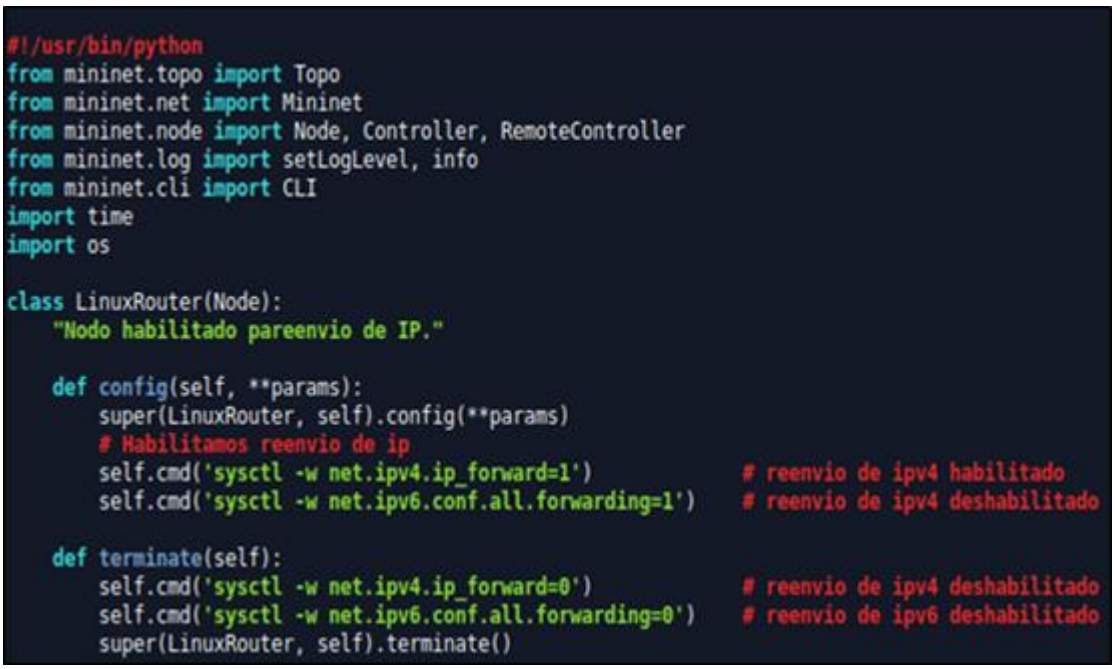
Nota. La Figura muestra la topología REUNA en implementación SDN.

Se ha creado el archivo en *Python* con nombre AvanzadaSDN.py que es el *script* que ejecutaremos para la emulación de nuestra topología de red avanzada, debemos tener en cuenta que este archivo debe crearse y ejecutarse dentro de la ruta *mininet/custom*.

En la Figura 32 se aprecia el primer segmento del código, en donde de acuerdo con lo explicado en los escenarios de prueba anteriores, se realiza la importación de las bibliotecas que permitirán crear nuestra red y se hace el renvío de IPv4 e IPv6.

Figura 32

Segmento 1 de código SDN en REUNA



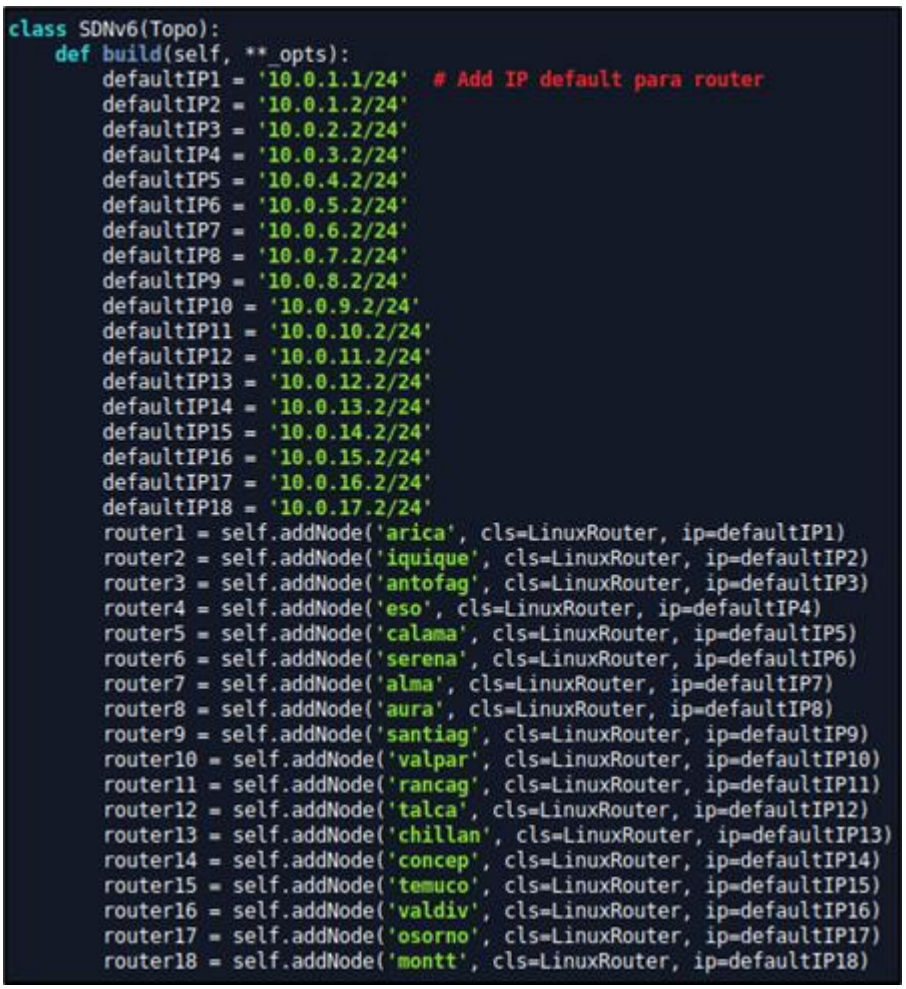
```
#!/usr/bin/python
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Node, Controller, RemoteController
from mininet.log import setLogLevel, info
from mininet.cli import CLI
import time
import os

class LinuxRouter(Node):
    "Nodo habilitado pareenvio de IP."

    def config(self, **params):
        super(LinuxRouter, self).config(**params)
        # Habilitamos reenvio de ip
        self.cmd('sysctl -w net.ipv4.ip_forward=1')            # reenvio de ipv4 habilitado
        self.cmd('sysctl -w net.ipv6.conf.all.forwarding=1')   # reenvio de ipv4 deshabilitado

    def terminate(self):
        self.cmd('sysctl -w net.ipv4.ip_forward=0')            # reenvio de ipv4 deshabilitado
        self.cmd('sysctl -w net.ipv6.conf.all.forwarding=0')   # reenvio de ipv6 deshabilitado
        super(LinuxRouter, self).terminate()
```

Nota. La Figura muestra la primera parte del código SDN de REUNA.

Como se ha explicado en lo relacionado a IPv6 y OSPFv3 que es la segunda prueba, necesitamos implementar IPv4 que finalmente no serán utilizadas, aunque como se explicó se puede hacer funcionar IPv4 e IPv6 al mismo tiempo, en la Figura 33 se aprecia las direcciones IPv4 asignadas como *defaultIP*, se agregó los 18 nodos que a través de lo programado se convertirán en routers que identifican a cada ciudad de nuestra topología de la red avanzada académica de Chile, debemos considerar que se tuvo la necesidad de abreviar los nombres de algunos routers, debido a que el nombre de los nodos soporta hasta un máximo de siete caracteres que se explicó también en las observaciones de la red tradicional para que sea estandarizado los nombres en ambas redes.

Figura 33

Segmento 2 de código SDN en REUNA



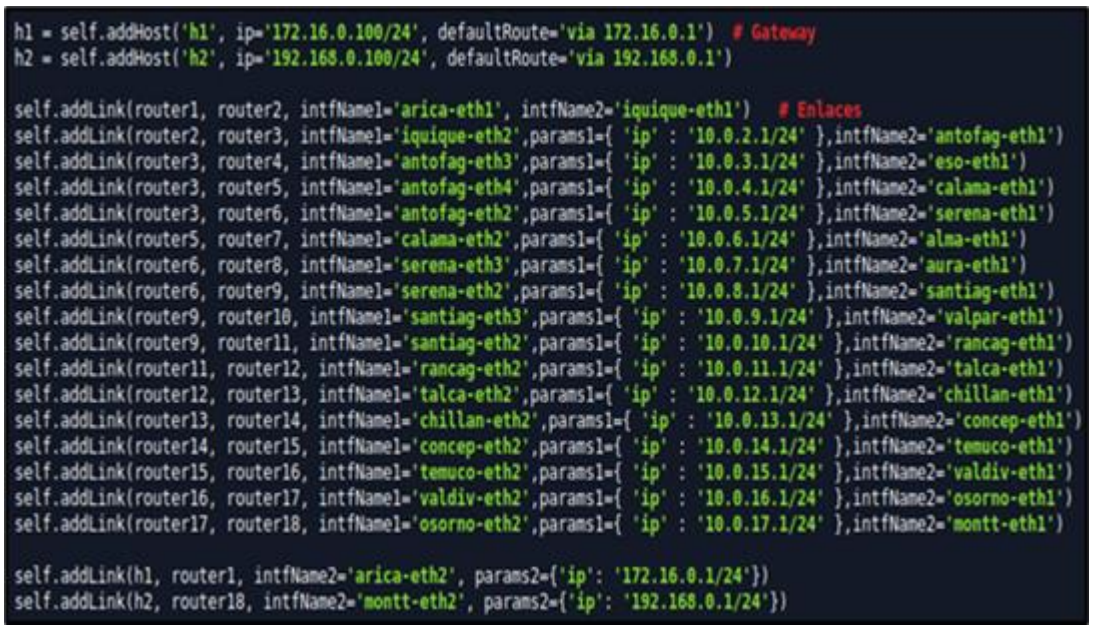
```
class SDNv6(Topo):
    def build(self, **_opts):
        defaultIP1 = '10.0.1.1/24'  # Add IP default para router
        defaultIP2 = '10.0.1.2/24'
        defaultIP3 = '10.0.2.2/24'
        defaultIP4 = '10.0.3.2/24'
        defaultIP5 = '10.0.4.2/24'
        defaultIP6 = '10.0.5.2/24'
        defaultIP7 = '10.0.6.2/24'
        defaultIP8 = '10.0.7.2/24'
        defaultIP9 = '10.0.8.2/24'
        defaultIP10 = '10.0.9.2/24'
        defaultIP11 = '10.0.10.2/24'
        defaultIP12 = '10.0.11.2/24'
        defaultIP13 = '10.0.12.2/24'
        defaultIP14 = '10.0.13.2/24'
        defaultIP15 = '10.0.14.2/24'
        defaultIP16 = '10.0.15.2/24'
        defaultIP17 = '10.0.16.2/24'
        defaultIP18 = '10.0.17.2/24'
        router1 = self.addNode('arica', cls=LinuxRouter, ip=defaultIP1)
        router2 = self.addNode('iquique', cls=LinuxRouter, ip=defaultIP2)
        router3 = self.addNode('antofag', cls=LinuxRouter, ip=defaultIP3)
        router4 = self.addNode('eso', cls=LinuxRouter, ip=defaultIP4)
        router5 = self.addNode('calama', cls=LinuxRouter, ip=defaultIP5)
        router6 = self.addNode('serena', cls=LinuxRouter, ip=defaultIP6)
        router7 = self.addNode('alma', cls=LinuxRouter, ip=defaultIP7)
        router8 = self.addNode('aura', cls=LinuxRouter, ip=defaultIP8)
        router9 = self.addNode('santiag', cls=LinuxRouter, ip=defaultIP9)
        router10 = self.addNode('valpar', cls=LinuxRouter, ip=defaultIP10)
        router11 = self.addNode('rancag', cls=LinuxRouter, ip=defaultIP11)
        router12 = self.addNode('talca', cls=LinuxRouter, ip=defaultIP12)
        router13 = self.addNode('chillan', cls=LinuxRouter, ip=defaultIP13)
        router14 = self.addNode('concep', cls=LinuxRouter, ip=defaultIP14)
        router15 = self.addNode('temuco', cls=LinuxRouter, ip=defaultIP15)
        router16 = self.addNode('valdiv', cls=LinuxRouter, ip=defaultIP16)
        router17 = self.addNode('osorno', cls=LinuxRouter, ip=defaultIP17)
        router18 = self.addNode('montt', cls=LinuxRouter, ip=defaultIP18)
```

Nota. La Figura muestra la segunda parte del código SDN de REUNA.

En la Figura 34 se observa la tercera parte de nuestro código que es correspondiente solo a IPv4 se programó de forma idéntica a las anteriores pruebas los enlaces entre routers y de router a host con sus respectivos Gateway.

Figura 34

Segmento 3 de código SDN en REUNA



```
h1 = self.addHost('h1', ip='172.16.0.100/24', defaultRoute='via 172.16.0.1')  # Gateway
h2 = self.addHost('h2', ip='192.168.0.100/24', defaultRoute='via 192.168.0.1')

self.addLink(router1, router2, intfName1='arica-eth1', intfName2='iquique-eth1')   # Enlaces
self.addLink(router2, router3, intfName1='iquique-eth2',params1={ 'ip' : '10.0.2.1/24' },intfName2='antofag-eth1')
self.addLink(router3, router4, intfName1='antofag-eth3',params1={ 'ip' : '10.0.3.1/24' },intfName2='eso-eth1')
self.addLink(router3, router5, intfName1='antofag-eth4',params1={ 'ip' : '10.0.4.1/24' },intfName2='calama-eth1')
self.addLink(router3, router6, intfName1='antofag-eth2',params1={ 'ip' : '10.0.5.1/24' },intfName2='serena-eth1')
self.addLink(router5, router7, intfName1='calama-eth2',params1={ 'ip' : '10.0.6.1/24' },intfName2='alma-eth1')
self.addLink(router6, router8, intfName1='serena-eth3',params1={ 'ip' : '10.0.7.1/24' },intfName2='aura-eth1')
self.addLink(router6, router9, intfName1='serena-eth2',params1={ 'ip' : '10.0.8.1/24' },intfName2='santiag-eth1')
self.addLink(router9, router10, intfName1='santiag-eth3',params1={ 'ip' : '10.0.9.1/24' },intfName2='valpar-eth1')
self.addLink(router9, router11, intfName1='santiag-eth2',params1={ 'ip' : '10.0.10.1/24' },intfName2='rancag-eth1')
self.addLink(router11, router12, intfName1='rancag-eth2',params1={ 'ip' : '10.0.11.1/24' },intfName2='talca-eth1')
self.addLink(router12, router13, intfName1='talca-eth2',params1={ 'ip' : '10.0.12.1/24' },intfName2='chillan-eth1')
self.addLink(router13, router14, intfName1='chillan-eth2',params1={ 'ip' : '10.0.13.1/24' },intfName2='concep-eth1')
self.addLink(router14, router15, intfName1='concep-eth2',params1={ 'ip' : '10.0.14.1/24' },intfName2='temuco-eth1')
self.addLink(router15, router16, intfName1='temuco-eth2',params1={ 'ip' : '10.0.15.1/24' },intfName2='valdiv-eth1')
self.addLink(router16, router17, intfName1='valdiv-eth2',params1={ 'ip' : '10.0.16.1/24' },intfName2='osorno-eth1')
self.addLink(router17, router18, intfName1='osorno-eth2',params1={ 'ip' : '10.0.17.1/24' },intfName2='montt-eth1')

self.addLink(h1, router1, intfName2='arica-eth2', params2={'ip': '172.16.0.1/24'})
self.addLink(h2, router18, intfName2='montt-eth2', params2={'ip': '192.168.0.1/24'})
```

Nota. La Figura muestra la tercera parte del código SDN de REUNA.

En la cuarta parte de nuestro código que se observa en la Figura 35, donde se ha programado la misma secuencia y pasos que las anteriores pruebas, lo primero que realizamos es el llamado a la clase creada SDNv6, enviamos a nuestro controlador y se inicia, también en esta parte se realiza el llamado a nuestros dispositivos que son nuestros routers y los host a través del objeto net con la propiedad que nos ofrece getNodeByName que pasa el nombre de los dispositivos declarados en la clase SDNv6.

Figura 35

Segmento 4 de código SDN en REUNA

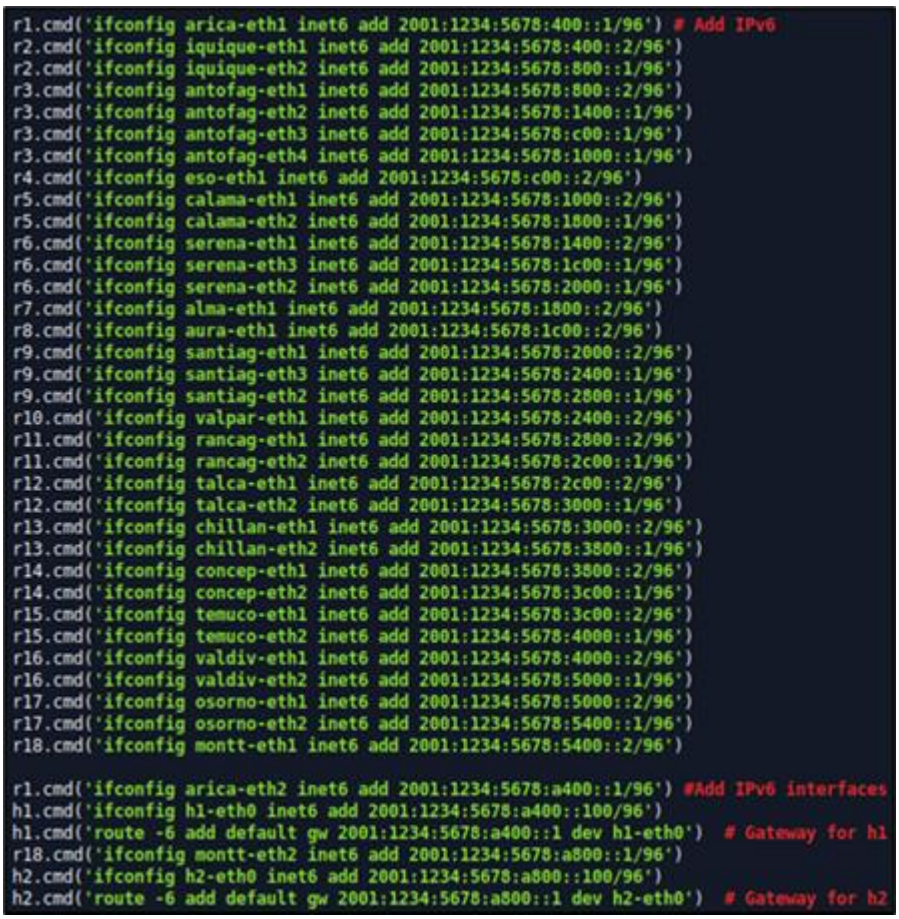
```
def run():
    "Test linux router"
    topo = SDNv6()
    net = Mininet(controller=RemoteController, topo=topo)
    c1 = net.addController('c1', ipv6='2001:1234:5678:1400::20', port=6633)
    net.start()
    info('*** Routing Table on Router:\n')

    r1 = net.getNodeByName('arica')
    r2 = net.getNodeByName('iquique')
    r3 = net.getNodeByName('antofag')
    r4 = net.getNodeByName('eso')
    r5 = net.getNodeByName('calama')
    r6 = net.getNodeByName('serena')
    r7 = net.getNodeByName('alma')
    r8 = net.getNodeByName('aura')
    r9 = net.getNodeByName('santiag')
    r10 = net.getNodeByName('valpar')
    r11 = net.getNodeByName('rancag')
    r12 = net.getNodeByName('talca')
    r13 = net.getNodeByName('chillan')
    r14 = net.getNodeByName('concep')
    r15 = net.getNodeByName('temuco')
    r16 = net.getNodeByName('valdiv')
    r17 = net.getNodeByName('osorno')
    r18 = net.getNodeByName('montt')
    h1 = net.getNodeByName('h1')
    h2 = net.getNodeByName('h2')
```

Nota. La Figura muestra la curta parte con propiedad getNodeByName que se asigna a la variable de los routers mediante su nombre del router.

En la quinta parte de la programación realizada que se observa en la Figura 36 y en base a lo investigado y las pruebas iniciales desarrolladas se realiza la configuración IPv6 a través de la propiedad cmd, accedemos a las interfaces de red y con la propiedad *ifconfig* añadimos IPv6 a cada enlace creado a través de la función *net6 add* seguido de la IPv6, así mismo en la parte del *host* definimos el *Gateway* definido por el router al que se encuentra conectado.

Figura 36

Segmento 5 de código SDN en REUNA



```
r1.cmd('ifconfig arica-eth1 inet6 add 2001:1234:5678:400::1/96') # Add IPv6
r2.cmd('ifconfig iquique-eth1 inet6 add 2001:1234:5678:400::2/96')
r2.cmd('ifconfig iquique-eth2 inet6 add 2001:1234:5678:800::1/96')
r3.cmd('ifconfig antofag-eth1 inet6 add 2001:1234:5678:800::2/96')
r3.cmd('ifconfig antofag-eth2 inet6 add 2001:1234:5678:1400::1/96')
r3.cmd('ifconfig antofag-eth3 inet6 add 2001:1234:5678:c00::1/96')
r3.cmd('ifconfig antofag-eth4 inet6 add 2001:1234:5678:1000::1/96')
r4.cmd('ifconfig eso-eth1 inet6 add 2001:1234:5678:c00::2/96')
r5.cmd('ifconfig calama-eth1 inet6 add 2001:1234:5678:1000::2/96')
r5.cmd('ifconfig calama-eth2 inet6 add 2001:1234:5678:1800::1/96')
r6.cmd('ifconfig serena-eth1 inet6 add 2001:1234:5678:1400::2/96')
r6.cmd('ifconfig serena-eth3 inet6 add 2001:1234:5678:1c00::1/96')
r6.cmd('ifconfig serena-eth2 inet6 add 2001:1234:5678:2000::1/96')
r7.cmd('ifconfig alma-eth1 inet6 add 2001:1234:5678:1800::2/96')
r8.cmd('ifconfig aura-eth1 inet6 add 2001:1234:5678:1c00::2/96')
r9.cmd('ifconfig santiag-eth1 inet6 add 2001:1234:5678:2000::2/96')
r9.cmd('ifconfig santiag-eth3 inet6 add 2001:1234:5678:2400::1/96')
r9.cmd('ifconfig santiag-eth2 inet6 add 2001:1234:5678:2800::1/96')
r10.cmd('ifconfig valpar-eth1 inet6 add 2001:1234:5678:2400::2/96')
r11.cmd('ifconfig rancag-eth1 inet6 add 2001:1234:5678:2800::2/96')
r11.cmd('ifconfig rancag-eth2 inet6 add 2001:1234:5678:2c00::1/96')
r12.cmd('ifconfig talca-eth1 inet6 add 2001:1234:5678:2c00::2/96')
r12.cmd('ifconfig talca-eth2 inet6 add 2001:1234:5678:3000::1/96')
r13.cmd('ifconfig chillan-eth1 inet6 add 2001:1234:5678:3000::2/96')
r13.cmd('ifconfig chillan-eth2 inet6 add 2001:1234:5678:3800::1/96')
r14.cmd('ifconfig concep-eth1 inet6 add 2001:1234:5678:3800::2/96')
r14.cmd('ifconfig concep-eth2 inet6 add 2001:1234:5678:3c00::1/96')
r15.cmd('ifconfig temuco-eth1 inet6 add 2001:1234:5678:3c00::2/96')
r15.cmd('ifconfig temuco-eth2 inet6 add 2001:1234:5678:4000::1/96')
r16.cmd('ifconfig valdiv-eth1 inet6 add 2001:1234:5678:4000::2/96')
r16.cmd('ifconfig valdiv-eth2 inet6 add 2001:1234:5678:5000::1/96')
r17.cmd('ifconfig osorno-eth1 inet6 add 2001:1234:5678:5000::2/96')
r17.cmd('ifconfig osorno-eth2 inet6 add 2001:1234:5678:5400::1/96')
r18.cmd('ifconfig montt-eth1 inet6 add 2001:1234:5678:5400::2/96')

r1.cmd('ifconfig arica-eth2 inet6 add 2001:1234:5678:a400::1/96') #Add IPv6 interfaces
h1.cmd('ifconfig h1-eth0 inet6 add 2001:1234:5678:a400::100/96')
h1.cmd('route -6 add default gw 2001:1234:5678:a400::1 dev h1-eth0')  # Gateway for h1
r18.cmd('ifconfig montt-eth2 inet6 add 2001:1234:5678:a800::1/96')
h2.cmd('ifconfig h2-eth0 inet6 add 2001:1234:5678:a800::100/96')
h2.cmd('route -6 add default gw 2001:1234:5678:a800::1 dev h2-eth0')  # Gateway for h2
```

Nota. La Figura muestra la quinta parte del código SDN de REUNA para la asignación de direcciones IPv6 en cada enlace de los router.

En la Figura 37 correspondiente a la sexta parte de nuestro código observamos la creación del *api* de *zebra* para cada uno de los 18 *routers* a través de la propiedad cmd que posteriormente hará utilidad de las configuraciones OSPF, también se observa en nuestro código de la red programada que creamos los *sockets* para OSPF a través del *api* creadas por *zebra*, se aprecia tanto para OSPFv3, esto también a través de la propiedad cmd; debemos considerar que la ruta donde se están creando todos los sockets es en */Desktop/Sockets*, estás líneas de código también se realizó de la misma forma que en las pruebas realizadas anteriormente.

Figura 37

Segmento 6 de código SDN en REUNA

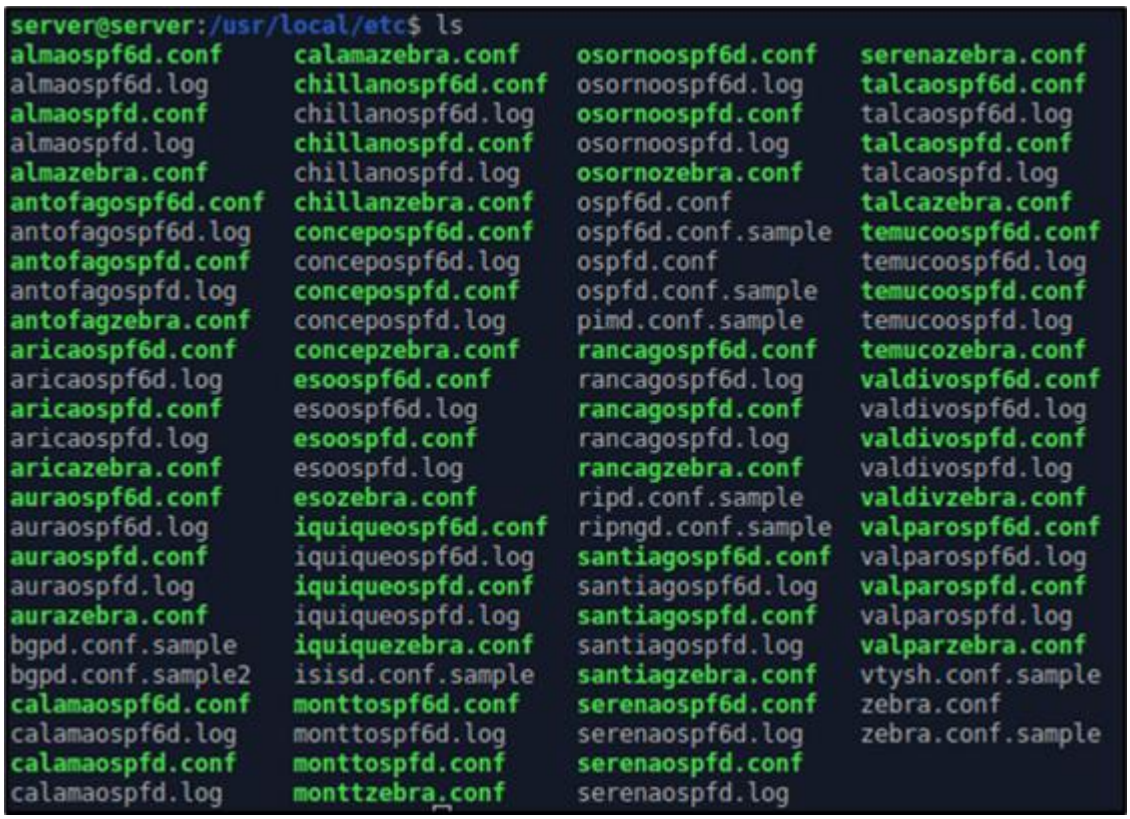
```
info('Iniciando zebra ospfd ospf6d service:\n')

r1.cmd('zebra -f /usr/local/etc/aricazebra.conf -d -z ~/Desktop/Sockets/aricazebra.api -i ~/Desktop/Sockets/aricazebra.interface')
time.sleep(1)  # Aqui zebra crea los api socket
r2.cmd('zebra -f /usr/local/etc/iquiquezebra.conf -d -z ~/Desktop/Sockets/iquiquezebra.api -i ~/Desktop/Sockets/iquiquezebra.interface')
r3.cmd('zebra -f /usr/local/etc/antofagzebra.conf -d -z ~/Desktop/Sockets/antofagzebra.api -i ~/Desktop/Sockets/antofagzebra.interface')
r4.cmd('zebra -f /usr/local/etc/esozebra.conf -d -z ~/Desktop/Sockets/esozebra.api -i ~/Desktop/Sockets/esozebra.interface')
r5.cmd('zebra -f /usr/local/etc/calamazebra.conf -d -z ~/Desktop/Sockets/calamazebra.api -i ~/Desktop/Sockets/calamazebra.interface')
r6.cmd('zebra -f /usr/local/etc/serenazebra.conf -d -z ~/Desktop/Sockets/serenazebra.api -i ~/Desktop/Sockets/serenazebra.interface')
r7.cmd('zebra -f /usr/local/etc/almazebra.conf -d -z ~/Desktop/Sockets/almazebra.api -i ~/Desktop/Sockets/almazebra.interface')
r8.cmd('zebra -f /usr/local/etc/aurazebra.conf -d -z ~/Desktop/Sockets/aurazebra.api -i ~/Desktop/Sockets/aurazebra.interface')
r9.cmd('zebra -f /usr/local/etc/santiagzebra.conf -d -z ~/Desktop/Sockets/santiagzebra.api -i ~/Desktop/Sockets/santiagzebra.interface')
r10.cmd('zebra -f /usr/local/etc/valparzebra.conf -d -z ~/Desktop/Sockets/valparzebra.api -i ~/Desktop/Sockets/valparzebra.interface')
r11.cmd('zebra -f /usr/local/etc/rancagzebra.conf -d -z ~/Desktop/Sockets/rancagzebra.api -i ~/Desktop/Sockets/rancagzebra.interface')
r12.cmd('zebra -f /usr/local/etc/talcazebra.conf -d -z ~/Desktop/Sockets/talcazebra.api -i ~/Desktop/Sockets/talcazebra.interface')
r13.cmd('zebra -f /usr/local/etc/chillanzebra.conf -d -z ~/Desktop/Sockets/chillanzebra.api -i ~/Desktop/Sockets/chillanzebra.interface')
r14.cmd('zebra -f /usr/local/etc/concepzebra.conf -d -z ~/Desktop/Sockets/concepzebra.api -i ~/Desktop/Sockets/concepzebra.interface')
r15.cmd('zebra -f /usr/local/etc/temucozebra.conf -d -z ~/Desktop/Sockets/temucozebra.api -i ~/Desktop/Sockets/temucozebra.interface')
r16.cmd('zebra -f /usr/local/etc/valdivzebra.conf -d -z ~/Desktop/Sockets/valdivzebra.api -i ~/Desktop/Sockets/valdivzebra.interface')
r17.cmd('zebra -f /usr/local/etc/osornozebra.conf -d -z ~/Desktop/Sockets/osornozebra.api -i ~/Desktop/Sockets/osornozebra.interface')
r18.cmd('zebra -f /usr/local/etc/monttzebra.conf -d -z ~/Desktop/Sockets/monttzebra.api -i ~/Desktop/Sockets/monttzebra.interface')

# ospf6d - ospfv3
r1.cmd('ospf6d -f /usr/local/etc/aricaospf6d.conf -d -z ~/Desktop/Sockets/aricazebra.api -i ~/Desktop/Sockets/aricaospf6d.interface')
r2.cmd('ospf6d -f /usr/local/etc/iquiqueospf6d.conf -d -z ~/Desktop/Sockets/iquiquezebra.api -i ~/Desktop/Sockets/iquiqueospf6d.interface')
r3.cmd('ospf6d -f /usr/local/etc/antofagospf6d.conf -d -z ~/Desktop/Sockets/antofagzebra.api -i ~/Desktop/Sockets/antofagospf6d.interface')
r4.cmd('ospf6d -f /usr/local/etc/esoospf6d.conf -d -z ~/Desktop/Sockets/esozebra.api -i ~/Desktop/Sockets/esoospf6d.interface')
r5.cmd('ospf6d -f /usr/local/etc/calamaospf6d.conf -d -z ~/Desktop/Sockets/calamazebra.api -i ~/Desktop/Sockets/calamaospf6d.interface')
r6.cmd('ospf6d -f /usr/local/etc/serenaospf6d.conf -d -z ~/Desktop/Sockets/serenazebra.api -i ~/Desktop/Sockets/serenaospf6d.interface')
r7.cmd('ospf6d -f /usr/local/etc/almaospf6d.conf -d -z ~/Desktop/Sockets/almazebra.api -i ~/Desktop/Sockets/almaospf6d.interface')
r8.cmd('ospf6d -f /usr/local/etc/auraospf6d.conf -d -z ~/Desktop/Sockets/aurazebra.api -i ~/Desktop/Sockets/auraospf6d.interface')
r9.cmd('ospf6d -f /usr/local/etc/santiagospf6d.conf -d -z ~/Desktop/Sockets/santiagzebra.api -i ~/Desktop/Sockets/santiagospf6d.interface')
r10.cmd('ospf6d -f /usr/local/etc/valparospf6d.conf -d -z ~/Desktop/Sockets/valparzebra.api -i ~/Desktop/Sockets/valparospf6d.interface')
r11.cmd('ospf6d -f /usr/local/etc/rancagospf6d.conf -d -z ~/Desktop/Sockets/rancagzebra.api -i ~/Desktop/Sockets/rancagospf6d.interface')
r12.cmd('ospf6d -f /usr/local/etc/talcaospf6d.conf -d -z ~/Desktop/Sockets/talcazebra.api -i ~/Desktop/Sockets/talcaospf6d.interface')
r13.cmd('ospf6d -f /usr/local/etc/chillanospf6d.conf -d -z ~/Desktop/Sockets/chillanzebra.api -i ~/Desktop/Sockets/chillanospf6d.interface')
r14.cmd('ospf6d -f /usr/local/etc/concepospf6d.conf -d -z ~/Desktop/Sockets/concepzebra.api -i ~/Desktop/Sockets/concepospf6d.interface')
r15.cmd('ospf6d -f /usr/local/etc/temucoospf6d.conf -d -z ~/Desktop/Sockets/temucozebra.api -i ~/Desktop/Sockets/temucoospf6d.interface')
r16.cmd('ospf6d -f /usr/local/etc/valdivospf6d.conf -d -z ~/Desktop/Sockets/valdivzebra.api -i ~/Desktop/Sockets/valdivospf6d.interface')
r17.cmd('ospf6d -f /usr/local/etc/osornoospf6d.conf -d -z ~/Desktop/Sockets/osornozebra.api -i ~/Desktop/Sockets/osornoospf6d.interface')
r18.cmd('ospf6d -f /usr/local/etc/monttospf6d.conf -d -z ~/Desktop/Sockets/monttzebra.api -i ~/Desktop/Sockets/monttospf6d.interface')
```

Nota. La Figura muestra la sexta parte del código SDN de REUNA referente a la lectura de los archivos zebra y OSPF6d y la creación de los sockets.

Teniendo nuestro *script* listo para ejecutar con todo el código que se ha mostrado, debemos considerar que todos los archivos a utilizar se encuentren y estén configurados adecuadamente, en la Figura 38 se aprecia todos los archivos *zebra* y *OSPF* que son llamados para crear los *sockets*.

Figura 38

Archivos de configuración SDN de REUNA

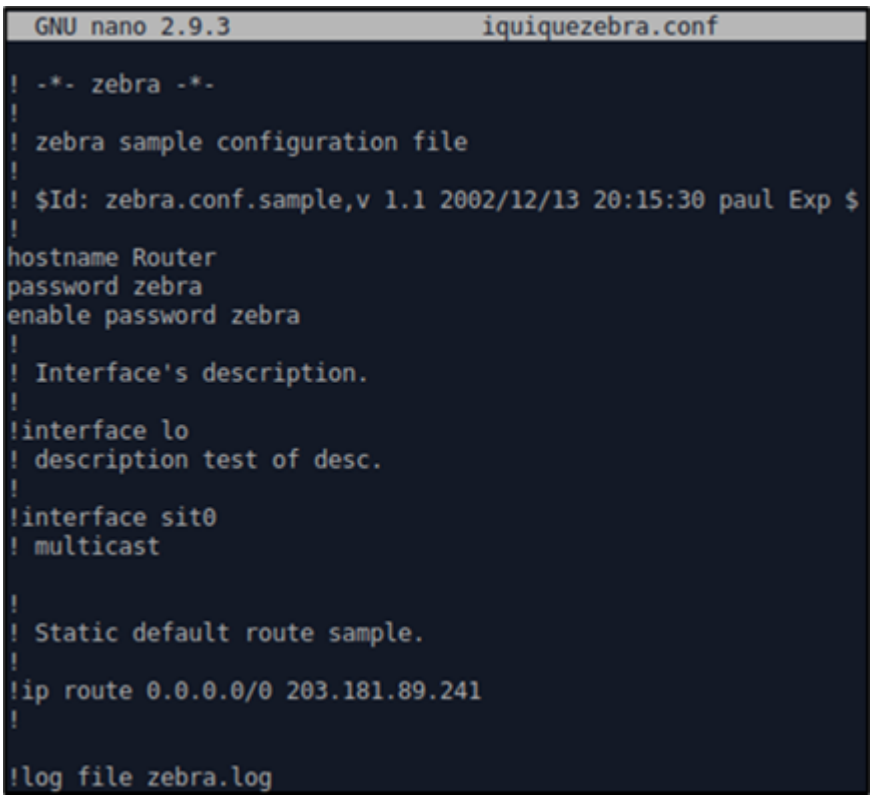


Nota. La Figura muestra la topología todos los archivos que requiere Quagga para enrutamiento.

En la Figura 39 se aprecia el archivo zebra que se ha configurado para el router que corresponde a Iquique que tiene como nombre iquizebra con la extensión .conf (iquiquezebra.conf), que como se explicó en las anteriores pruebas es solo una copia del archivo zebra que nos trae *quagga*.

Figura 39

Archivo de configuración iquiquezebra.conf

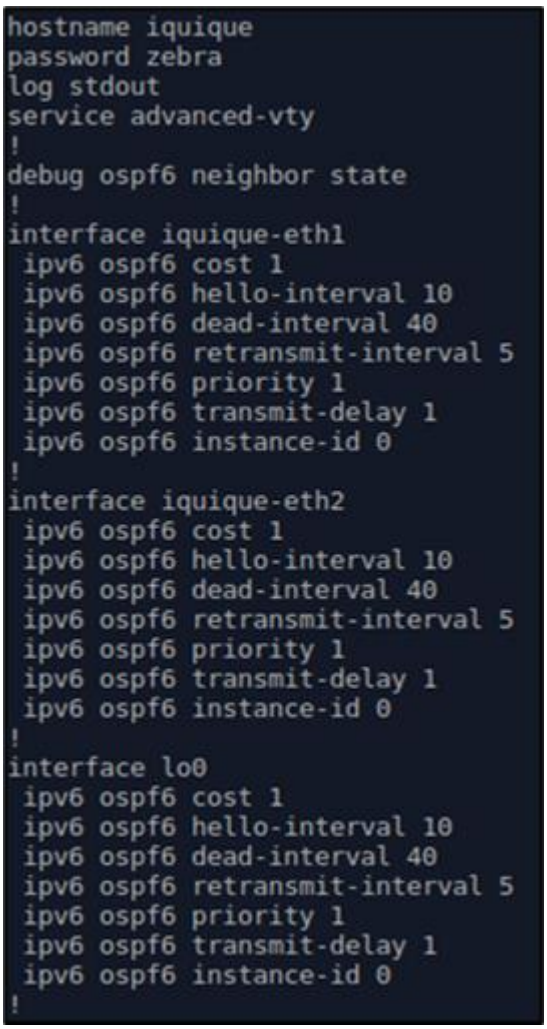


Nota. La Figura muestra la configuración del archivo zebra para el router Iquique.

En la Figura 40 se observa la primera parte de la configuración realizada al archivo iquiqueospf6d.conf, donde se aprecia que existe inicialmente una configuración de identidad para el router con su nombre, además se hace un debug de rutas vecinas, para que se creen las actuales del router según la configuración, también se declara las interfaces del router incluyendo la Loopback con los parámetros que se mencionó en la anterior prueba.

Figura 40

Segmento 1 de configuración iquiqueospfd.conf



```
hostname iquique
password zebra
log stdout
service advanced-vty
!
debug ospf6 neighbor state
!
interface iquique-eth1
 ipv6 ospf6 cost 1
 ipv6 ospf6 hello-interval 10
 ipv6 ospf6 dead-interval 40
 ipv6 ospf6 retransmit-interval 5
 ipv6 ospf6 priority 1
 ipv6 ospf6 transmit-delay 1
 ipv6 ospf6 instance-id 0
!
interface iquique-eth2
 ipv6 ospf6 cost 1
 ipv6 ospf6 hello-interval 10
 ipv6 ospf6 dead-interval 40
 ipv6 ospf6 retransmit-interval 5
 ipv6 ospf6 priority 1
 ipv6 ospf6 transmit-delay 1
 ipv6 ospf6 instance-id 0
!
interface lo0
 ipv6 ospf6 cost 1
 ipv6 ospf6 hello-interval 10
 ipv6 ospf6 dead-interval 40
 ipv6 ospf6 retransmit-interval 5
 ipv6 ospf6 priority 1
 ipv6 ospf6 transmit-delay 1
 ipv6 ospf6 instance-id 0
!
```

Nota. La Figura muestra la topología la primera parte del archivo de configuración ospfv3 para el router Iquique.

En la Figura 41 se observa la segunda parte de la configuración OSPFv3 del router correspondiente a la ciudad de Iquique, donde se indica que se dará utilidad a ospf6 para las dos interfaces se conectan a Iquique, además de la ACL para permitir difusión, la red general 2001:1::/ y la loopback, que son necesarias y el mapa de ruta que se da a través de los *neighbor.*

Figura 41

Segmento 2 de configuración iquiqueospfd.conf

```
router ospf6
 router-id 2.2.2.2
! redistribute static route-map static-ospf6
 interface iquique-eth1 area 0.0.0.0
 interface iquique-eth2 area 0.0.0.0
!
access-list access4 permit 127.0.0.1/32
!
ipv6 access-list access6 permit 3ffe:501::/32
ipv6 access-list access6 permit 2001:1::/48
ipv6 access-list access6 permit ::1/128
!
ipv6 prefix-list test-prefix seq 1000 deny any
!
route-map static-ospf6 permit 10
 match ipv6 address prefix-list test-prefix
 set metric-type type-2
 set metric 2000
!
line vty
 access-class access4
 ipv6 access-class access6
 exec-timeout 0 0
!
log file /usr/local/etc/iquiqueospf6d.log
```

Nota. La Figura muestra la segunda parte de la configuración ospfv3 para el router Iquique.

Teniendo configurado adecuadamente todos nuestros archivos *zebra* y OSPF, debemos iniciar nuestro controlador *Ryu* ya que toda la funcionalidad es enviada para que se pueda crear el flujo de datos, en la Figura 42 se aprecia el inicio de nuestro controlador que se encuentra en el dispositivo físico 2.

Figura 42

Inicio de controlador Ryu en SDN para REUNA

```
servidor@servidor:~$ ryu-manager --observe-links ~/flowmanager/flowmanager.py
 ryu.app.simple_switch_13
loading app /home/servidor/flowmanager/flowmanager.py
You are using Python v2.7.17.final.0
loading app ryu.app.simple_switch_13
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu.app.simple_switch_13 of SimpleSwitch13
instantiating app ryu.topology.switches of Switches
instantiating app /home/servidor/flowmanager/flowmanager.py of FlowManager
instantiating app ryu.controller.ofp_handler of OFPHandler
(5791) wsgi starting up on http://0.0.0.0:8080
```

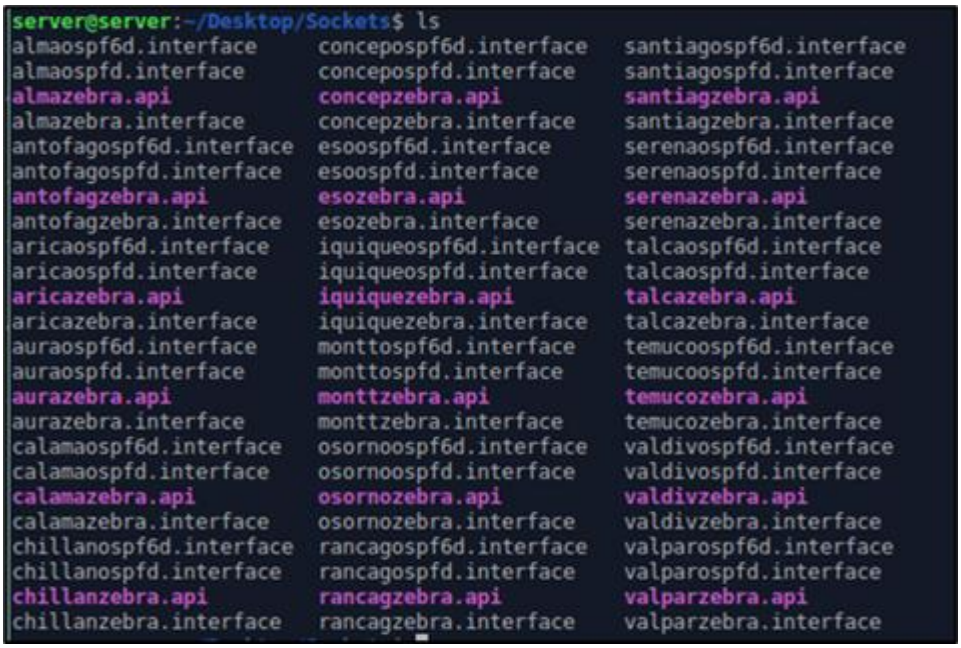
Nota. La Figura muestra la ejecución del controlador Ryu con OpenFlow.

Después de iniciar nuestro controlador, debemos ejecutar nuestro *script* llamado AvanzadaSDN.py que se encuentra en el dispositivo físico 1, esto simplemente con la funcionalidad que nos proporciona *Python*, debemos ejecutar el comando *python AvanzadaSDN.py* en la misma ruta *mininet/custom*.

Como se ha mencionado en las pruebas realizadas anteriormente, después de ejecutar el *script* debemos verificar que los sockets hayan sido creados, en la Figura 43 se observa que se han creado todos los *sockets* correspondientes a cada router que son para los archivos *zebra* y *OSPF*.

Figura 43

Creación de sockets SDN de REUNA



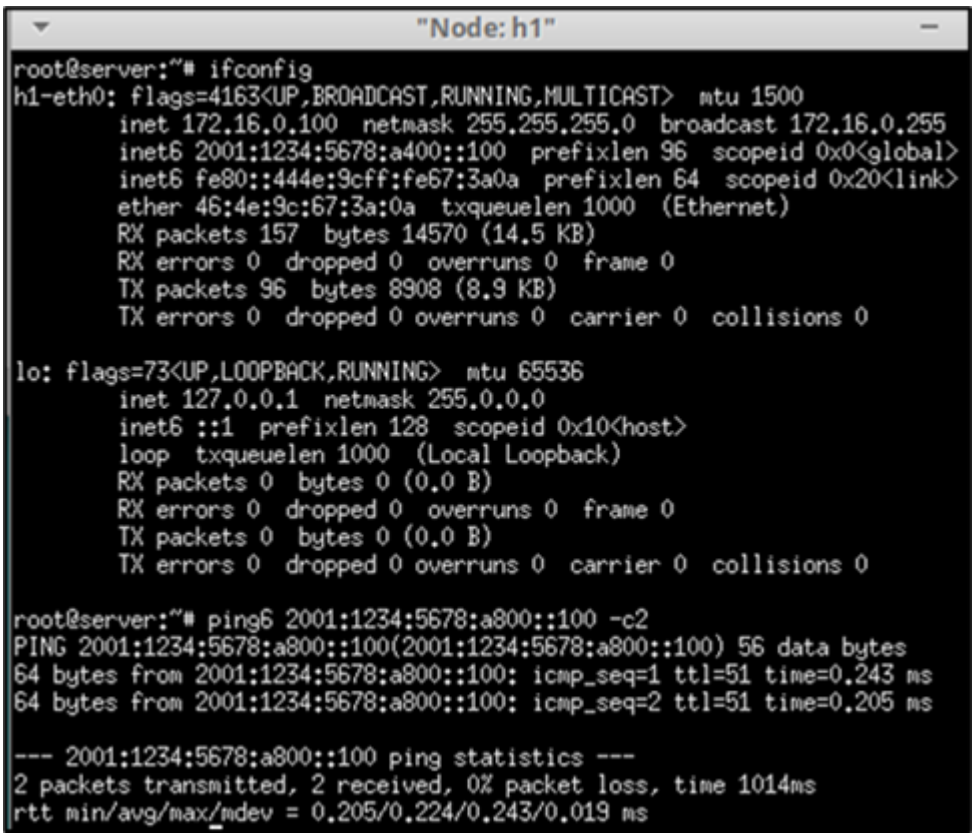
Nota. La Figura muestra los sockets creados después de ejecutar el script de REUNA.

3.5.6. Prueba de Conectividad

Habiendo verificado que los *sockets* han sido creados, procedemos a realizar la prueba de conectividad, inicialmente se ha probado conectividad entre todos los nodos, en la Figura 44 se observa la prueba de conectividad extremo a extremo, es decir desde h1 hasta h2, se aprecia el ping6 exitoso realizado desde h1 enviando dos paquetes hacia h2, debemos tener en cuenta que para acceder a la *cli* de los dispositivos se realiza a través del comando *xterm* que nos ofrece *Mininet* y seguido del nombre del nodo, por ejemplo *xterm h1* para abrir la *cli* de h1.

Figura 44

Prueba de conectividad entre h1 y h2 en SDN de REUNA



Nota. La Figura muestra la prueba de conectividad extremo a extremos en SDN.

IV. Metodología

4.1. Tipo de Investigación

El tipo de investigación es aplicada, según (Sampieri, 2014) “este tipo de investigación tiene como propósito resolver problemas”, en relación con el autor, esta investigación busca evaluar el rendimiento de una arquitectura de red avanzada en base al tipo de implementación a través de entornos emulados, para solucionar posibles problemas funcionamiento basadas en el rendimiento de red de dicha implementación de manera tradicional o programable.

4.2. Enfoque de Investigación

El enfoque de esta investigación es cuantitativo, ya que permite la evaluación de variaciones e identificar diferencias y medir resultados; como su característica principal incluye las mediciones y precisión Sampieri (2014); por lo que esta investigación mantiene este enfoque ya que los indicadores tienen términos de cantidad.

4.3. Nivel de Investigación

El nivel de esta investigación es explicativo ya que estos estudios están orientados a responder por las causas de los eventos y explicar porque ocurre un fenómeno y como se relacionan dos o más variables, Sampieri (2014); en esta investigación se terminará explicando cómo se relaciona el tipo de implementación de una red avanzada con el rendimiento de la red.

4.4. Diseño de Investigación

El diseño de esta investigación es cuasi-experimental, que según (Sampieri, 2014) “este tipo de investigación identifica la manipulación de la variable independiente para ver el efecto con la variable dependiente y se aplica grupos constituidos, se busca considerar el grado de características que puedan estar relacionadas con la variable de un objeto de investigación; además, consiste en recolectar conjunto de datos que no se asignan al azar”, por tanto esta investigación busca variar dos tipos de implementación de una red avanzada (X), para lo cual se define una de manera tradicional (HDN) y otra de manera programable SDN; con el propósito de comparar el rendimiento de red (Y), considerando el retardo, pérdida de paquetes y consumo de recursos computacionales de la red.

Según (Sampieri, 2014) “el diseño es el plan o estrategia concebida para obtener la información que se deesa con el fin de responder al planteamiento del problema; el investigador debe visualizar la manera práctica y concreta de cumplir los objetivos fijados,

lo que implica seleccionar o desarrollar el diseño y aplicarlo al contexto particular del estudio"; por tanto, no existiendo un diseño en particular para evaluar el rendimiento de una red se ha optado por definir las siguientes fases.

A. Planificación

En esta fase se establece los principios que corresponde a obtener la información alineada a nuestro objetivo de estudio, para ello respondemos a algunas preguntas, ¿cuáles son los elementos que integran este estudio?; contestando a esta interrogante, dentro de esta investigación en el capítulo III encontramos las especificaciones técnicas de los equipos que se utilizan con sus capacidades los cuales conforman nuestra topología física, asimismo dentro de lo cual se define las direcciones IPv6 y el protocolo OSPFv3 como elementos principales para la implementación de la arquitectura de red avanzada; también respondemos a la pregunta ¿qué herramientas se utilizará para construir el escenario (arquitectura de red avanzada)?, dentro del mismo capítulo III, encontramos definidas las herramientas que se utilizan y su instalación y configuración adecuada para próximamente realizar las pruebas correspondientes, por mencionar tenemos dentro de la implementación HDN a GNS3, y VirtualBox, en la implementación SDN encontramos Mininet, Ryu y Quagga.

B. Diseño

En esta fase se concreta el escenario de emulación, el cual es la arquitectura de red avanzada REUNA; dentro del capítulo III se encuentra la emulación de REUNA de manera tradicional (HDN) y de manera programable (SDN); además dentro de esta topología como tal observamos los 18 routers referentes a las ciudades, pero se añaden dos hosts extremos para efectos de pruebas dentro del escenario emulado.

C. Pruebas

Las pruebas corresponden al proceso de obtener la información que permita evaluar el rendimiento de la red en base al tipo de implementación, para ello se ejecutan pruebas que se definen como adecuadas, la siguiente Tabla muestra las pruebas definidas que se realizada para la validación de nuestra hipótesis dentro del capítulo IV de este informe.

Tabla 6

Pruebas para recolectar datos

Indicador		**Pruebas**	
Retardo de transmisión	Ping	Traceroute	Archivos
Retardo de procesamiento	Ping	Traceroute	
Variación de retardo	Iperf		
Pérdida de paquetes	Ping	Stream	
Consumo de CPU	Ping	Stream	
Consumo de Memoria	Ping	Stream	
Consumo de ancho de banda	Iperf		

Nota. La Tabla muestra todas las pruebas ejecutadas para recolectar datos.

D. Análisis de resultados

Esta fase corresponde a describir los resultados mediante resúmenes en gráficos o tablas donde se podrá diferenciar el rendimiento de la arquitectura de red avanzada en base al tipo de implementación, considerando cada indicador de nuestras dimensiones definidas, esto corresponde al capítulo V de este informe.

E. Conclusiones

Esta fase corresponde a las conclusiones generales de esta investigación en relación con nuestros objetivos que están alineados a nuestro problema e hipótesis; esto se describe en base a nuestros resultados obtenidos previo análisis de estos que terminarán validando nuestra hipótesis mediante una evidencia a favor o en contra.

F. Recomendaciones

Esta fase describe las recomendaciones propias del investigador en base a la experiencia obtenida al realizar dicho estudio; las recomendaciones van enfocadas al tipo de implementación en una arquitectura de red avanzada considerando claramente el rendimiento de la red.

4.5. Población y Muestra

La población se encuentra definida ya que según (GÉANT, 2020) existen 47 redes avanzadas académicas divididas en 8 regiones, las cuales existen actualmente dentro del mapa de conectividad global; de las cuales todas tienen como característica principal que su funcionamiento es bajo el protocolo IPv6. La muestra se encuentra definida y es no probabilística, según (Sampieri, 2014) “es un subgrupo que representa la población, que

consiste en la elección por casusas relacionadas del investigador y no depende de la probabilidad" por ello se considerará únicamente la muestra de 1, siendo la red académica de Chile (REUNA), la que se utilizará para realizar la evaluación del rendimiento de una arquitectura de red avanzada.

4.6. Hipótesis

4.6.1. Hipótesis General

La relación que existe entre el rendimiento y el tipo de implementación de una arquitectura de red avanzada es inversamente proporcional.

4.6.2. Hipótesis Específicas

1. La relación que existe entre el retardo y el tipo de implementación de una arquitectura de red avanzada es inversamente proporcional.

2. La relación que existe entre la pérdida de paquetes y el tipo de implementación de una arquitectura de red avanzada es inversamente proporcional.

3. La relación que existe entre el consumo de recursos computacionales y el tipo de implementación de una arquitectura de red avanzada es inversamente proporcional.

4.7. Variables de Investigación

4.7.1 Variable Independiente

Tipo de Red

Con base en los antecedentes, bases teóricas y demás información revisada, para esta variable se considera una dimensión.

- ✓ Arquitectura de red

4.7.2 Variable Dependiente

Rendimiento de red.

Con base a las bases teóricas, antecedentes y demás información revisada, para esta variable se considera las siguientes tres dimensiones

- ✓ Retardo de red
- ✓ Pérdida de paquetes
- ✓ Consumo de recursos computacionales

4.7.3 Operacionalización de Variables

Tabla 7

Matriz de operacionalización de variables

Variable	Tipo de Variable	Dimensión	Definición Conceptual	Definición Operacional	Indicador	Nivel de medición	Valor
Tipo de red	Cuantitativa	Arquitectura de red	Según (Cai, 2011) las redes tradicionales mantienen unido el plano de datos y el de control, mientras que las redes definidas por software separan estos planos con el fin de tener una mejor gestión de red	Se refiere a la forma de hacer funcionar la red, a través de la configuración de los protocolos y dispositivos	• Cantidad de instancias de plano de control	Nominal	• 1 (SDN) • >1 (tradicional)
Rendimiento de red	Cuantitativa	Retardo de red	Según (Gonzales, 1999) el retardo es el resultado de variaciones que pueden ocurrir en la red	Se refiere al tiempo que tarda en llegar un paquete desde su origen hasta su destino	Retardo de transmisión	Razón	• Milisegundos
					Retardo de procesamiento	Razón	• Milisegundos
					Variación de retardo	Razón	• Milisegundos
		Pérdida de paquetes	"Es un fenómeno común en todas las redes conmutadas por paquetes como las redes IP. Un paquete que llega se pierde en la red si no encuentra espacio en la cola. Mientras más usuarios accedan la red, los enrutadores más se congestionan y se produce la pérdida de paquetes	Se refiere a la cantidad de paquetes que cayeron entre un origen y un destino, de todos los paquetes que fueron enviados dentro de la red de datos en mención	Cantidad de paquetes perdidos	Razón	• N° de paquetes perdidos

		Consumo de recursos computacionales	Según, (Mazza, 2014) “Se consideran recursos computacionales a todos aquellos que intervienen en el desarrollo y/o despliegue de soluciones basadas en hardware, software, telecomunicaciones, servicios, etc”.	Se refiere a la cantidad que se utiliza de la disponibilidad de nuestros recursos de software, hardware y entre otros en el proceso de la implementación del laboratorio virtual	• Nivel de consumo de CPU (%)	Intervalo	• (0 – 100) %
					• Nivel de consumo de memoria	Razón	• MegaBytes
					• Nivel de consumo de ancho de banda	Razón	• Mbps

Nota. La Tabla muestra todo el proceso de operacionalización de las variables de la investigación.

4.8 Instrumentos de medición y recolección de datos

El instrumento de medición es el recurso que utiliza el investigador para registrar información o datos sobre las variables; la recolección de datos se basa en estos instrumentos y podemos obtenerlos por medición, observación y documentación para proveer un mayor entendimiento de los significados y experiencias; los instrumentos pueden ser tipo cuestionarios, escalas de medición y otros donde se puede hacer pruebas y procedimientos propios de la disciplina Sampieri (2014); en base a esto nuestro instrumento se compone de las siguientes herramientas:

4.8.1 Herramientas de Monitoreo de Red

Esta herramienta se utilizará para medir el estado en que se encuentra el equipo, el consumo de recursos computacionales de los entornos emulado, las herramientas identificadas a utilizar son:

Monitor de Sistema

Esta herramienta está disponible en cualquier sistema operativo, a través de esta herramienta podemos identificar, el consumo de Memoria RAM y Procesador, según (Gnome, 2014) “muestra qué programas están en ejecución y cuánto procesador, tiempo, memoria y espacio en disco están usando”.

Iperf

Es una herramienta de red multiplataforma, que permite monitorizar el ancho de banda en tiempo real, según (Amoedo, 2020) “Es una herramienta de monitorización de ancho de banda basada en consola que funciona en tiempo real”, esta herramienta consiste en determinar el ancho de banda en la red donde un host actúa como cliente y el otro como servidor.

Traceroute

Esta herramienta es muy útil ya que nos permite determinar la ruta que toma un paquete determinado desde su origen hasta su destino, esta herramienta mostrará el tiempo que le toma al paquete llegar gasta cada router, cuando el paquete alcanza un router, es considerado como un salto.

4.8.2 Captura de Paquetes y Logs

Se utilizará para encontrar tiempos, número de paquetes, saltos y entre otros para determinar el retardo y la convergencia de red, así como también servirá para verificación y revisión de *logs*, algunas de las herramientas identificadas a utilizar son:

Ping

La utilidad de ping es muy amplia, en primera instancia para probar conectividad entre los diferentes puntos de la red, por tanto esta es una de las principales herramientas que se utilizará, a través de las propiedades -C para indicarle la cantidad de intentos, -s para indicarle mostrar las estadísticas y -b para indicarle el tamaño del paquete, esta funcionalidad nos permitirá obtener resultados para poder calcular el retardo de propagación, retardo de transmisión, retardo de procesamiento, retardo de extremo a extremo, variación de retardo y la pérdida de paquetes, según (Ramírez González, 2016) "Por medio del tiempo de espera de la respuesta a ese envío de información se determina el retraso o no de esa respuesta, lo que también se conoce como latencia". Ping está disponible tanto como par aipv4 como ping y para ipv6 como ping6.

Wireshark

Esta herramienta es considerada un *sniffer* (escucha), es un capturador de paquetes que también nos será de gran utilidad, ya que nos permitirá verificar datos para calcular los indicadores tanto de retardo de red y convergencia de red, según (Wireshark, 2020) "es el analizador de protocolos de red más importante y utilizado del mundo".

4.9 Validación de Hipótesis

La validación de Hipótesis se realizó en base a la recolección de datos a través de las pruebas ejecutadas de acuerdo con nuestros indicadores para cada una de nuestras dimensiones de las variables, que determinarán del rendimiento de red; para validar la hipótesis a través del instrumento se ha utilizado como prueba estadística la media ya que no es necesario utilizar otras pruebas para demostrar que la relación entre el rendimiento de una arquitectura de red avanzada y el tipo de implementación es inversamente proporcional.

En la HDN contamos con 18 *routers* pertenecientes a las 18 ciudades, en cada router tenemos el plano de control, es decir se ha configurado, lo que representa que en cada router se tiene un plano de control, al momento que se encuentra en funcionamiento la red tradicional tenemos 18 instancias de plano de control ejecutándose que hacen posible la comunicación en toda la topología de red.

En la SDN tenemos 18 archivos de configuración que son llamados a nuestro *script* que envía toda la ejecución hacia un solo controlador *Ryu*; al momento que se ejecuta la topología, el controlador ejecuta una sola instancia de plano de control que a través de los *sockets* puede ejercer la comunicación, la Tabla 8 nos muestra el resumen general de las instancias de control que se tiene en los dos tipos de implementaciones.

Tabla 8

Instancias de control en arquitectura de red avanzada y tipo de implementación

Tipo de implementación	N° de Instancias de Control
HDN	18
SDN	1

Nota. La Tabla muestra la cantidad de instancias de control para cada tipo de implementación

4.9.1. Retardo de Transmisión

Este retardo es asociado con el tiempo que se utiliza para transmitir los paquetes dentro de la red, este tiempo es el que se requiere para que el paquete sea puesto en el canal y sea transmitido en la red, si el paquete es de mayor tamaño o es de diferente tipo, ya sea que se utiliza otros protocolos para transmitir tendremos un tiempo diferente, por ello es que se ha considerado realizar distintas pruebas para poder tener mayor precisión con la recolección de datos; todas estas pruebas fueron realizadas directamente entre el host1 y el host2; cabe indicar que para tener una mayor precisión de tiempo se ha determinado por conveniente realizar 40 veces dichas pruebas; ya que según (Sampieri, 2014) se puede considerar cronológico, sucesión de eventos, ubicación, en este caso al ser tiempos que varían se trata de tener precisión, la Tabla 9 muestra los tipos de pruebas realizadas.

Tabla 9

Distribución de pruebas para retardo de transmisión

Prueba	Detalle	Casos de Prueba	
Ping	ICMPv6	56 Bytes	1500 Bytes
Traceroute	UDP	80 Bytes	1500 Bytes
Archivos	FTP	1.4 MB (.pdf)	14 MB (.mp4)

Nota. La Tabla muestra las tres pruebas ejecutadas con sus 2 casos de prueba para obtener datos del retardo de transmisión.

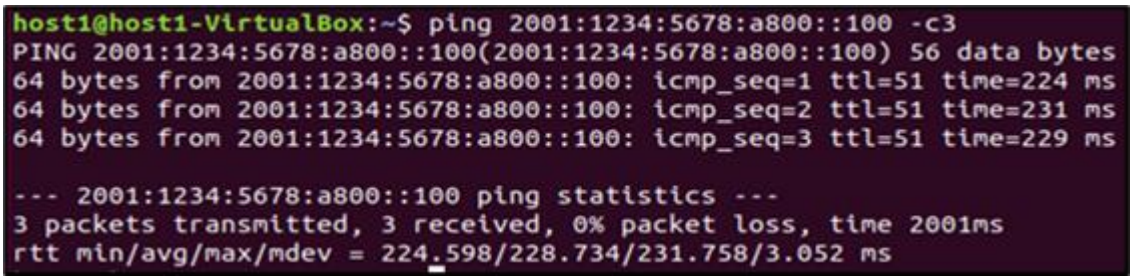
Retardo de transmisión en red avanzada tradicional HDN

- ✓ **Prueba Ping**

Para determinar el retardo de transmisión se procede a realizar la primera prueba de la Tabla 8; donde se ha determinado tener dos casos de prueba; el primer caso de prueba consiste en enviar el paquete ICMPv6 por defecto de 56 Bytes mediante la herramienta nativa ping para 3 paquetes, para ello basta con acceder a la terminal del host1 y hacer un ping hacia el host2 e ingresar el parámetro -c3 para el envío de 3 paquetes ICMPv6; en la Figura 45 se muestra el caso de prueba 1 realizado para la prueba de ping; donde se aprecia que los tiempos rtt, donde el que es de nuestro interés es el tiempo avg que es de 228.734 milisegundos; este caso de prueba se ha realizado 40 veces para obtener una mayor precisión.

Figura 45

Caso de prueba 56 Bytes con Ping en HDN - Retardo de Transmisión



Nota. La Figura muestra el tiempo que de trasmisión de un paquete ICMPv6 por defecto de 56 bytes desde el host1 hacia el host2.

En la Figura 46 observamos la captura de los datos para el caso de prueba de un paquete de tamaño de 1500 bytes el cual es enviado desde el host 1 hacia el host2; de igual manera se ha enviado 3 paquetes ICMPv6 y la media del tiempo de transmisión es de 234.184; también se ha realizado 40 veces.

Figura 46

Caso de prueba 1500 Bytes con Ping en HDN - Retardo de Transmisión

Nota. La Figura muestra el tiempo que de trasmisión de un paquete ICMPv6 por configurado de 1500 bytes desde el host1 hacia el host2.

En la Tabla 10 se aprecia el resumen de los tiempos de transmisión requeridos en la prueba de ping realizada.

Tabla 10

Retardos de transmisión de prueba Ping en HDN

Prueba (ping)	56 Bytes (Milisegundos)	1500 Bytes (Milisegundos)
1	228.7340	234.1840
2	227.9780	238.5060
3	227.9360	234.1420
4	237.5810	235.2520
5	232.3620	248.3530
6	229.0820	243.1490
7	246.5490	233.2240
8	227.8530	236.1770
9	231.9100	237.2190
10	238.4930	238.0990
11	238.1130	234.9420
12	237.6290	245.4490
13	227.9300	237.4350
14	229.5040	254.9040
15	233.8430	235.1790
16	233.7070	237.7440
17	225.5210	240.0230
18	227.8320	232.5700
19	231.6770	238.6890
20	229.7790	225.6230
21	226.3240	238.5850
22	226.7190	240.0040
23	226.8050	233.8060
24	229.1400	243.6990
25	233.6910	237.7460
26	236.6030	232.7680
27	237.6920	241.3760
28	234.9440	237.3390
29	234.7860	238.7840
30	237.4850	237.8960
31	233.4240	229.7560
32	231.2750	228.9650
33	233.4120	238.9790
34	231.2220	236.6280
35	231.2590	245.8260
36	227.3420	240.6570
37	228.1920	238.3940
38	239.2280	234.4550
39	227.3180	235.8660
40	229.0730	234.9870

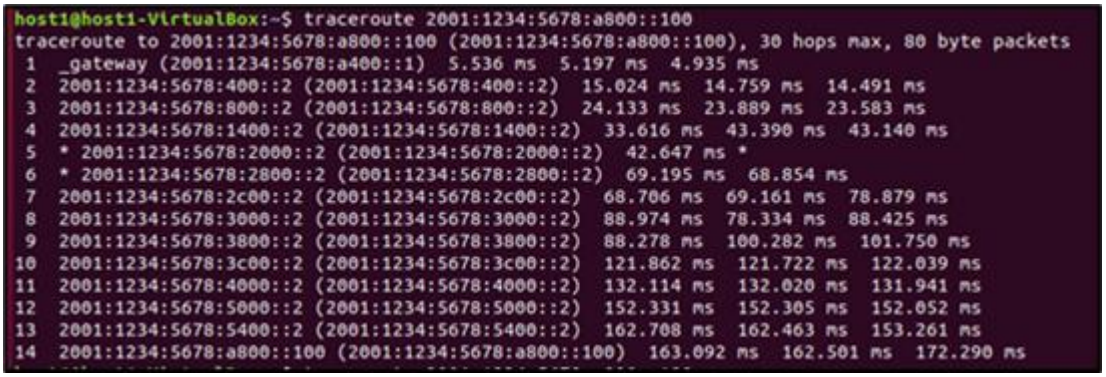
Con estos datos obtenidos se procedió a calcular la media del tiempo de retardo de transmisión, donde para el paquete de 56 Bytes se tiene un promedio de **231.9987 milisegundos** y en el paquete de 1500 Bytes el tiempo de transmisión es de **237.6845 milisegundos**; para tener un retardo de transmisión general de esta prueba sacamos su promedio en general el cual es de **234.8416 milisegundos**.

- ✓ **Prueba Traceroute**

Para esta prueba se ha utilizado la herramienta traceroute nativa del sistema operativo GNU/Linux; donde también se realizaron 2 casos de prueba; para el primero se considerará lo que la herramienta envía por defecto 3 paquetes UDP, para esto se realiza mediante la terminal; en la Figura 47 se muestra la primera prueba ejecutada para este caso de prueba realizado; traceroute muestra los tiempos de transmisión de los 3 paquetes por cada salto que ha realizado desde su origen hasta el destino; el que es de nuestro interés es el del último salto ya que este es el host2; los tiempos de transmisión de los 3 paquetes son de 163.092, 162.501 y 172.290 milisegundos respectivamente; para ello debemos calcular la media del tiempo de estos 3 paquetes que fueron enviados el cual es de **165.9610 milisegundos.**

Figura 47

Caso de prueba 80 Bytes con Tracert en HDN - Retardo de Transmisión



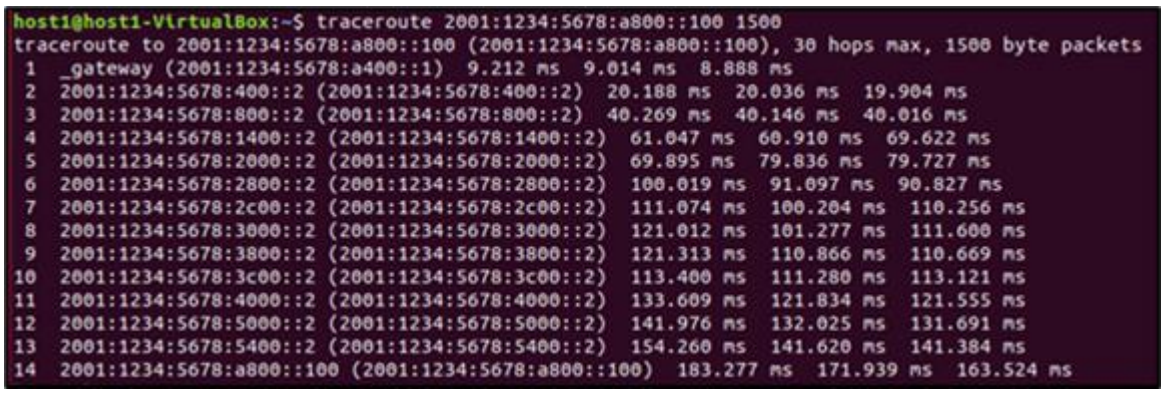
Nota. La Figura muestra la prueba Tracert de un paquete UDP por defecto de 80 bytes entre host1 y host2

Para el segundo caso de prueba mediante la herramienta traceroute se ha realizado el envío por defecto de los 3 paquetes UDP, pero con un tamaño de 1500 Bytes; la Figura 48 muestra que los tiempos de transmisión son de 183.277, 171.939 y 163.524 milisegundos respectivamente; seguidamente se procedió a calcular la media del retardo de transmisión de

los 3 paquetes enviados, el cual es de **172.9133 milisegundos**; también se ha realizado 40 veces este caso de prueba.

Figura 48

Caso de prueba 1500 Bytes con Tracert en HDN - Retardo de Transmisión



Nota. La Figura muestra la prueba Tracert de un paquete UDP configurado de 80 bytes entre host1 y host2

En la Tabla 11 se aprecia todos los tiempos de retardo de transmisión que se ha calculado por cada envío de los 3 paquetes UDP para los 2 casos de prueba de las 40 pruebas ejecutadas.

Tabla 11

Retardos de transmisión con Tracert en HDN

Prueba (Tracert)	80 Bytes (Milisegundos)	1500 Bytes (Milisegundos)
1	165.9610	172.9133
2	141.8820	182.5183
3	159.6687	180.9783
4	158.2227	185.9173
5	167.3533	180.4847
6	165.2263	189.0157
7	165.9070	202.6700
8	181.8307	212.3147
9	159.7053	206.6137
10	176.9020	199.0357
11	168.5437	178.0617
12	159.5103	181.3410
13	173.0380	171.7220
14	161.6443	165.8643
15	192.0297	173.9913
16	161.8140	186.8043
17	181.5067	181.7547
18	174.8947	169.6547
19	181.7597	185.4173
20	171.4893	172.4990
21	183.0720	165.0737
22	181.9937	181.8080
23	175.6243	172.0417
24	180.8303	172.5583
25	152.0067	172.0357
26	181.0600	184.7080
27	187.3897	179.4627
28	181.6853	182.0523
29	183.1407	175.7673
30	178.8863	195.7607
31	171.9203	208.7317
32	172.8620	175.1713
33	158.7740	169.2010
34	176.5053	168.7043
35	171.9947	171.1270
36	178.1413	172.5003
37	177.1853	172.8137
38	174.4327	188.2597
39	168.2093	179.3333
40	163.0960	171.9597

Asimismo, con estos datos obtenidos se procedió a calcular la media del tiempo de retardo de transmisión para esta prueba, donde para el paquete de 80 Bytes se tiene una media de **171.6925 milisegundos** y en el paquete de 1500 Bytes el tiempo de transmisión es de **180.9661 milisegundos**; para tener un retardo de transmisión general de esta prueba sacamos su media, el cual es de **176.3293 milisegundos**.

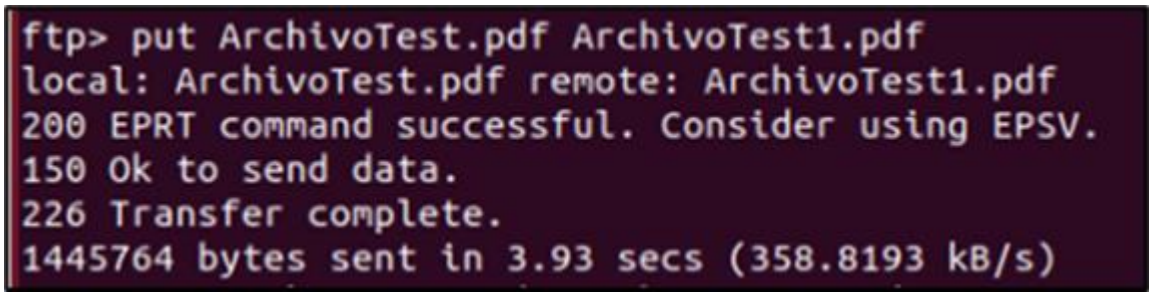
- ✓ **Prueba de archivos**

Para esta prueba se ha considerado enviar archivos desde el host 1 hacia el host 2; donde en el primer caso de prueba consiste en enviar un archivo en formato .pdf de 1.4 MB; en el segundo caso de prueba se ha considerado enviar un video en formato .mp4 de 18.4 MB; para ello se ha configurado un servidor FTP; en el Anexo se muestra la configuración de este servicio que nos permitirá el envío de los archivos.

En la Figura 49 se aprecia la primera prueba realizada en el caso de prueba 1, donde se aprecia que el tiempo que el tiempo de transmisión es de **3 930.0000 milisegundos**; esta prueba se ha realizado también 40 veces con fines de tener mayor precisión.

Figura 49

Caso de prueba de Archivos pdf de 1.4 MB en HDN-Retardo de Transmisión

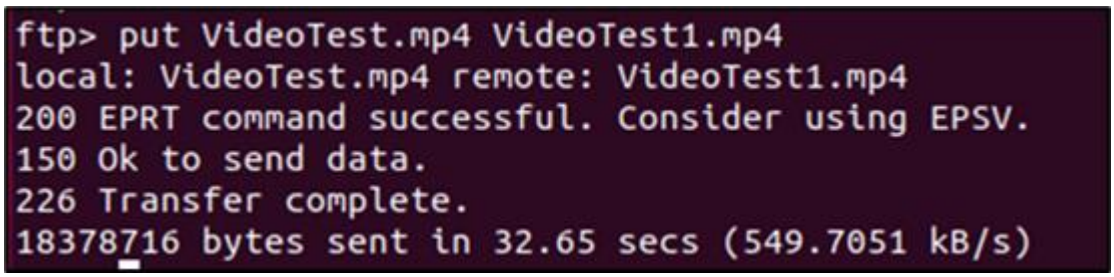


Nota. La Figura muestra la prueba del envío de un archivo .pdf de 1.4 MB entre host1 y host2

Para el caso de prueba 2 se ha enviado el video; en la Figura 50 se aprecia que el tiempo de transmisión requerido es de 32650.0000 milisegundos; de igual manera se ha tomado los datos en 40 ocasiones.

Figura 50

Caso de prueba de Archivos mp4 de 14 MB en HDN-Retardo de Transmisión



Nota. La Figura muestra la prueba del envío de un archivo .mp4 de 14 MB entre host1 y host2

La Tabla 12 nos muestra el resumen completo de los tiempos de retardo de transmisión de los 2 casos de prueba realizados, es decir para el envío de archivos .pdf y .mp4 especificados anteriormente; teniendo en cuenta que para obtener mayor precisión la prueba es ejecutada 40 veces.

Tabla 12

Retardos de transmisión con Archivos en HDN

Prueba (Archivos)	1.4 MB (.pdf-Milisegundos)	14 MB (.mp4-Milisegundos)
1	3 930.0000	3 2650.0000
2	3 280.0000	3 0120.0000
3	3 990.0000	3 2420.0000
4	5 020.0000	3 5010.0000
5	6 040.0000	2 8580.0000
6	4 850.0000	3 7170.0000
7	5 200.0000	3 2340.0000
8	4 180.0000	2 9490.0000
9	2 350.0000	2 9750.0000
10	6 330.0000	2 9060.0000
11	7 350.0000	3 3510.0000
12	4 480.0000	3 1320.0000
13	2 510.0000	2 8160.0000
14	2 410.0000	3 0540.0000
15	3 180.0000	3 2630.0000
16	5 060.0000	3 0640.0000
17	6 800.0000	3 4890.0000
18	3 960.0000	3 2270.0000
19	3 270.0000	3 6120.0000
20	4 700.0000	3 1080.0000
21	4 710.0000	3 1190.0000
22	7 020.0000	3 2870.0000
23	5 060.0000	3 0500.0000
24	3 800.0000	2 8780.0000
25	3 740.0000	3 2020.0000
26	5 370.0000	3 3120.0000
27	5 620.0000	3 0560.0000
28	5 400.0000	3 2020.0000
29	4 340.0000	3 3230.0000
30	2 880.0000	3 3900.0000
31	4 470.0000	3 3620.0000
32	3 810.0000	2 7950.0000
33	4 200.0000	3 1740.0000
34	4 330.0000	3 5190.0000
35	3 510.0000	3 2740.0000
36	2 690.0000	3 2910.0000
37	3 860.0000	3 4300.0000
38	4 030.0000	3 1530.0000
39	3 180.0000	2 9280.0000
40	2 570.0000	3 2420.0000

En base a esta Tabla 12, podemos calcular que el retardo de transmisión para el envío de archivos .pdf es de **4 337.0000 milisegundos** y para el envío de archivos .mp4 es de **3 1940.5000 milisegundos**; si calculamos el promedio general de retardo de transmisión para el envío de archivos es de **18 138.7500 milisegundos.**

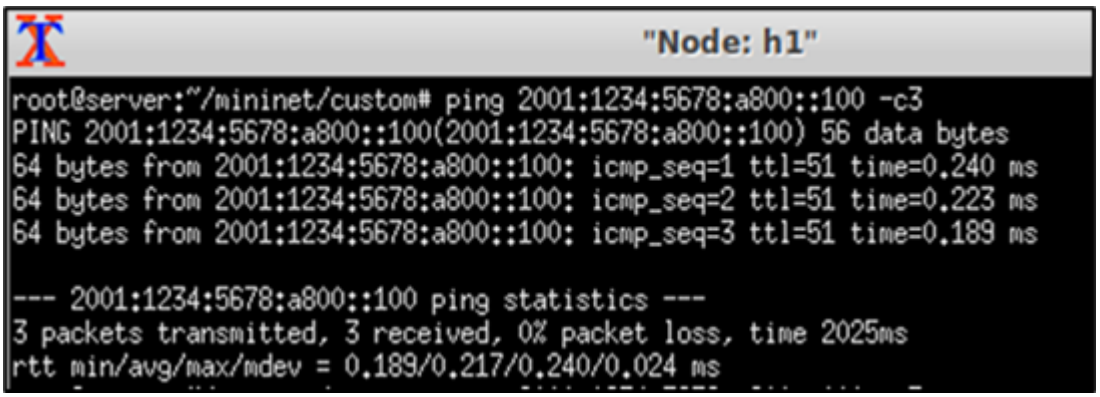
Retardo de transmisión en red avanzada SDN

- ✓ **Prueba de Ping**

De la misma manera que en la HDN, para determinar el retardo de transmisión se procedió a realizar la primera prueba de la Tabla 9; donde se ha determinado tener dos casos de prueba; el primer caso de prueba consiste en enviar el paquete ICMPv6 por defecto de 56 Bytes mediante la herramienta nativa ping para 3 paquetes a través del parámetro -c3; en la Figura 51 se muestra el caso de prueba 1 realizado para la prueba de ping; donde se aprecia que el tiempo avg es de **0.2170 milisegundos** el cual es el que tendremos en consideración para esta prueba.

Figura 51

Caso de prueba 56 Bytes con Ping en SDN - Retardo de Transmisión

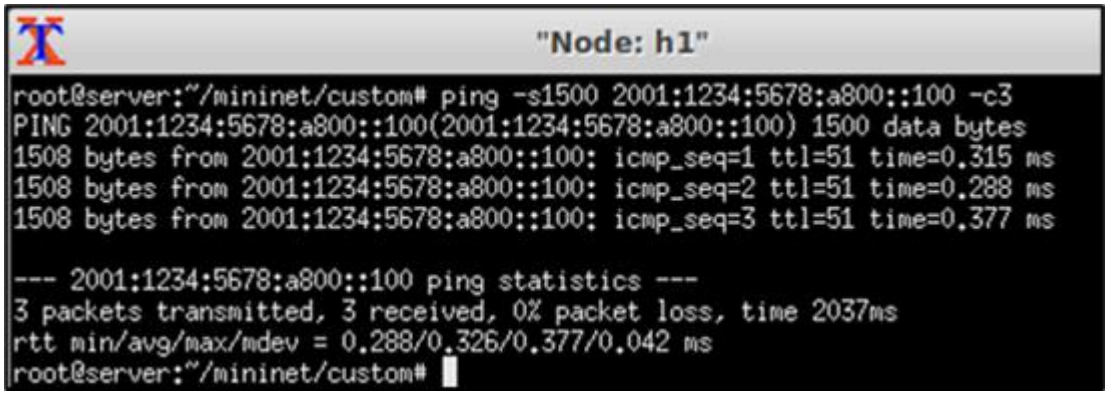


Nota. La Figura muestra el tiempo de retardo de transmisión de un paquete ICPMv6 por defecto de 56 Bytes entre h1 y h2 en la implementación SDN.

Para el caso de prueba 2, observamos la captura en la Figura 52 donde se ha enviado un paquete de tamaño de 1500 bytes el cual es enviado desde el host 1 hacia el host2; de igual manera se ha enviado 3 paquetes ICMPv6 y la media del tiempo de transmisión es de **0.3260 milisegundos**.

Figura 52

Caso de prueba 1500 Bytes con Ping en SDN - Retardo de Transmisión



Nota. La Figura muestra el tiempo de retardo de transmisión de un paquete ICPMv6 configurado de 1500 Bytes entre h1 y h2 en la implementación SDN.

La Tabla 13 muestra el resumen de estos tiempos de retardo de transmisión para la prueba de ping ejecutada, considerando ambos casos de prueba uno de 56 bytes y el otro de 1500 bytes; asimismo esta prueba se ha ejecutado 40 veces con el mismo objetivo de tener mayor precisión de dicha prueba.

Tabla 13

Retardos de transmisión de prueba Ping en SDN

Prueba (ping)	56 Bytes (Milisegundos)	1500 Bytes (Milisegundos)
1	0.2170	0.3260
2	0.2290	0.3010
3	0.2120	0.3130
4	0.2060	0.2680
5	0.1590	0.3140
6	0.2150	0.2580
7	0.2050	0.2750
8	0.2560	0.2830
9	0.2030	0.2540
10	0.2170	0.2930
11	0.2450	0.2840
12	0.1890	0.2540
13	0.2840	0.2650
14	0.1890	0.2870
15	0.2180	0.2860
16	0.2200	0.2550
17	0.2260	0.2900
18	0.2200	0.2750
19	0.2010	0.2660
20	0.2370	0.2650
21	0.2020	0.0279
22	0.2040	0.2550
23	0.2070	0.2650
24	0.1780	0.3190
25	0.2270	0.2700
26	0.1950	0.2690
27	0.2150	0.3710
28	0.1960	0.2850
29	0.210	0.2420
30	0.2270	0.3640
31	0.2200	0.2760
32	0.2090	0.2920
33	0.2190	0.2640
34	0.1870	0.2600
35	0.1760	0.2540
36	0.2180	0.2990
37	0.2270	0.2560
38	0.2060	0.2990
39	0.2330	0.2680
40	0.1830	0.2770

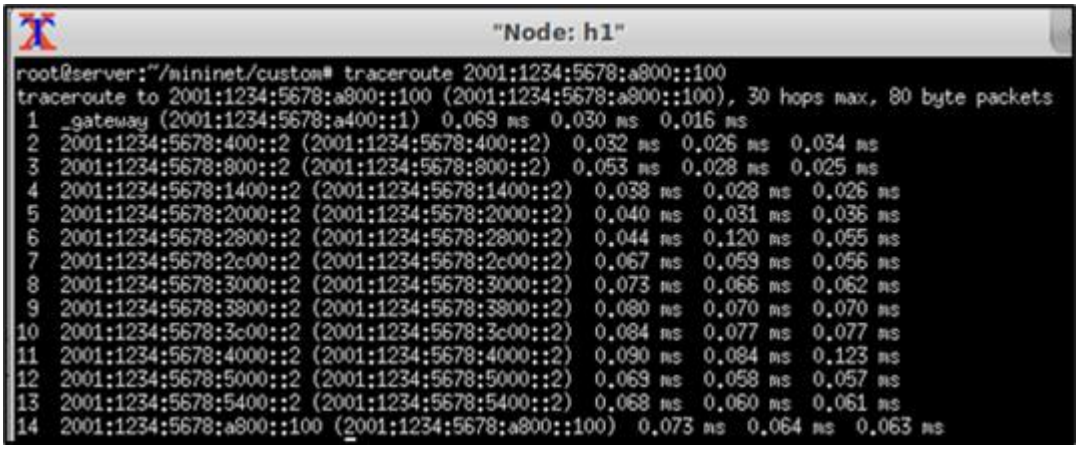
Con estos datos obtenidos se procedió a calcular la media del tiempo de retardo de transmisión, donde para el paquete de 56 Bytes se tiene un promedio de **0.2122 milisegundos** y en el paquete de 1500 Bytes el tiempo de transmisión es de **0.2756 milisegundos**; para tener un retardo de transmisión general de esta prueba sacamos su promedio el cual es **de 0.2439 milisegundos**.

- ✓ **Prueba Traceroute**

De igual manera que en la red avanzada tradicional para esta prueba se ha utilizado la herramienta *traceroute* nativa del sistema operativo; donde también se realizarán 2 casos de prueba; para el primero se considerará lo que la herramienta envía por defecto 3 paquetes UDP, para esto se realiza mediante la terminal; en la Figura 53 se muestra el primer caso de prueba realizado; *traceroute* muestra los tiempos de transmisión por cada salto; el que es de nuestro interés es el del último salto ya que este es host2; los tiempos de transmisión de los 3 paquetes son 0.0730, 0.0640, 0.0630 milisegundos respectivamente; para ello debemos calcular la media del tiempo de estos tres paquetes enviados el cual es de **0.0667 milisegundos.**

Figura 53

Caso de prueba:80 Bytes con Tracert en SDN-Retardo de Transmisión



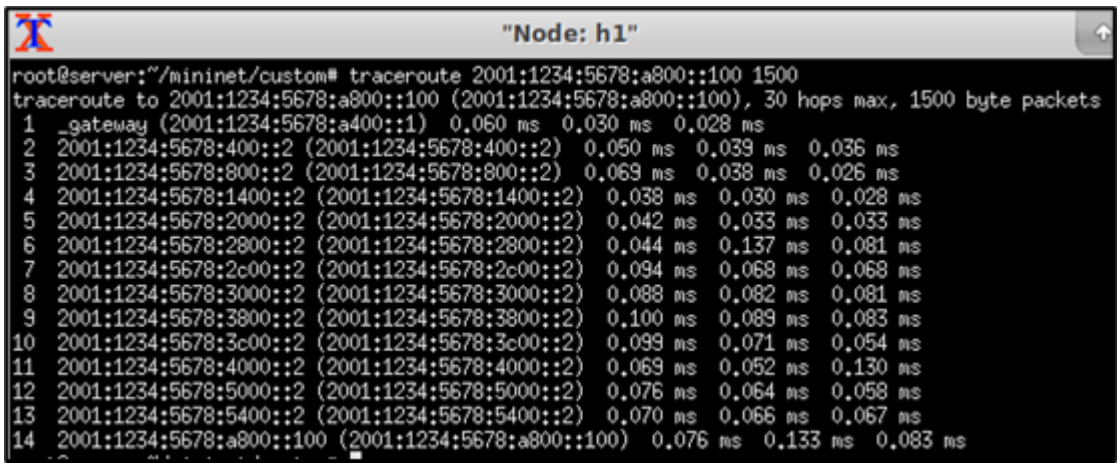
Nota. La Figura muestra el tiempo de retardo de transmisión de la prueba Tracert con paquete UDP por defecto de 80 Bytes entre h1 y h2 en la implementación SDN.

Para el segundo caso de prueba mediante la herramienta *traceroute* se ha realizado el envío por defecto de los 3 paquetes UDP, pero con un tamaño de 1500 Bytes; la Figura 54 muestra que los tiempos de transmisión son de 0.0760, 0.1330, 0.0830 milisegundos respectivamente; seguidamente se procedió a calcular la media del retardo de transmisión de

los 3 paquetes enviados, el cual es de **0.0973 milisegundos**; también se ha realizado 40 veces este caso de prueba.

Figura 54

Caso de prueba:1500 Bytes con Tracert-SDN - Retardo de Transmisión



Nota. La Figura muestra el tiempo de retardo de transmisión de la prueba Tracert con paquete UDP configurado de 1500 Bytes entre h1 y h2 en la implementación SDN.

En la siguiente Tabla 14 se muestra el resumen de los tiempos calculados para los 2 casos de prueba realizados en la prueba de *traceroute*. Con estos datos se calculó que la media del tiempo de transmisión es de **0.0608 milisegundos** en el tracert de 80 Bytes y para el de 1500 Bytes **0.0627 milisegundos**, asimismo la media general de transmisión de tracert es de **0.06177 milisegundos**.

Tabla 14

Retardos de transmisión de prueba Tracert en SDN

Prueba (Tracert)	80 Bytes (Milisegundos)	1500 Bytes (Milisegundos)
1	0.0667	0.0973
2	0.1000	0.0703
3	0.0563	0.0673
4	0.0530	0.0773
5	0.0490	0.0357
6	0.0733	0.0967
7	0.0457	0.0480
8	0.0600	0.0747
9	0.0603	0.0450
10	0.0473	0.0730
11	0.0817	0.0363
12	0.0750	0.0797
13	0.0547	0.0487
14	0.0497	0.0863
15	0.0700	0.0687
16	0.0697	0.0340
17	0.0530	0.0467
18	0.0787	0.0437
19	0.0557	0.0363
20	0.0587	0.0823
21	0.0353	0.0860
22	0.0447	0.0693
23	0.0553	0.0876
24	0.0737	0.0470
25	0.0670	0.0680
26	0.0720	0.0563
27	0.0730	0.1093
28	0.0603	0.0763
29	0.0373	0.0407
30	0.0410	0.0343
31	0.0663	0.0740
32	0.0707	0.0400
33	0.0513	0.04800
34	0.0663	0.0603
35	0.0347	0.0583
36	0.0743	0.0400
37	0.0697	0.0423
38	0.0663	0.0770
39	0.0580	0.0670
40	0.0570	0.0790

Asimismo, con estos datos obtenidos se procedió a calcular la media del tiempo de retardo de transmisión para esta prueba, donde para el paquete de 80 Bytes se tiene una media de **0.0608 milisegundos** y en el paquete de 1500 Bytes el tiempo de transmisión es de **0.0627 milisegundos**; para tener un retardo de transmisión general de esta prueba sacamos su media, el cual es de **0.0618 milisegundos**.

- ✓ **Prueba de archivos**

Coincidiendo con las pruebas realizadas en la HDN, para esta prueba se ha considerado enviar archivos desde el host 1 hacia el host 2; donde en el primer caso de prueba consiste en enviar un archivo en formato .pdf de 1.4 MB; en el segundo caso de prueba se ha considerado enviar un video en formato .mp4 de 18.4 MB; para ello se ha configurado un servidor FTP; en el Anexo se muestra la configuración de este servicio que nos permitirá el envío de los archivos.

En la Figura 55 se aprecia la primera prueba realizada en el caso de prueba 1, donde se aprecia que el tiempo de transmisión de este archivo de extensión .pdf es de 10 milisegundos.

Figura 55

Caso de prueba Archivos pdf de 1.4 MB-SDN-Retardo de Transmisión

Nota. La Figura muestra el tiempo de retardo de transmisión de la prueba de envío de un archivo pdf de 1.4 MB entre h1 y h2 en la implementación SDN.

Similarmente para el caso de prueba 2 de esta prueba de archivos, se ha enviado el archivo de formato .mp4; en la Figura 56 se aprecia que el tiempo de transmisión que se ha requerido, el cual es de 70.0000 milisegundos.

Figura 56

Caso de prueba Archivos mp4 de 14 MB-SDN-Retardo de Transmisión

Nota. La Figura muestra el tiempo de retardo de transmisión de la prueba de envío de un archivo mp4 de 14 MB entre h1 y h2 en la implementación SDN.

Una vez capturado todos los datos, se construyó la Tabla 15 la cual nos muestra el resumen de los tiempos de transmisión que se ha capturado para ambos casos de prueba que son él envió de un archivo .pdf y el otro un archivo .mp4. Asimismo, como en las anteriores pruebas se ha realizado 40 veces la prueba.

Tabla 15

Retardos de transmisión de prueba de archivos en SDN

Prueba (Archivos)	1.4 MB (pdf-Milisegundos)	14 MB (mp4-Milisegundos)
1	10.0000	70.0000
2	20.0000	100.0000
3	20.0000	110.0000
4	10.0000	110.0000
5	10.0000	80.0000
6	10.0000	70.0000
7	10.0000	90.0000
8	10.0000	90.0000
9	10.0000	80.0000
10	10.0000	80.0000
11	10.0000	120.0000
12	10.0000	70.0000
13	10.0000	70.0000
14	10.0000	80.0000
15	10.0000	110.0000
16	10.0000	70.0000
17	10.0000	100.0000
18	10.0000	80.0000
19	10.0000	70.0000
20	10.0000	70.0000
21	10.0000	70.0000
22	10.0000	110.0000
23	10.0000	60.0000
24	10.0000	80.0000
25	10.0000	80.0000
26	10.0000	80.0000
27	10.0000	90.0000
28	10.0000	80.0000
29	20.0000	100.0000
30	10.0000	100.0000
31	10.0000	90.0000
32	10.0000	80.0000
33	10.0000	110.0000
34	10.0000	90.0000
35	10.0000	120.0000
36	10.0000	80.0000
37	10.0000	90.0000
38	20.0000	70.0000
39	10.0000	80.0000
40	10.0000	110.0000

Con estos datos obtenidos se procedió a calcular la media del tiempo de retardo de transmisión, donde para el archivo .pdf de 1.4 MB se tiene una media de **11.0000 milisegundos** y en el archivo .mp4 de 14.8 MB el tiempo de transmisión es de **87.2500 milisegundos**; para tener un retardo de transmisión general de esta prueba sacamos su media el cual es de **49.1250 milisegundos**. Para evaluar este retardo de transmisión en ambas implementaciones de la arquitectura de red avanzada, se ha elaborado la Tabla 16 mediante todos los datos calculados.

Tabla 16

Comparación de retardo de transmisión HDN vs SDN

Prueba		**HDN (Milisegundos)**		**SDN (Milisegundos)**	
Ping	56 Bytes	231.9987	**234.8416**	0.2122	**0.2439**
	1500 Bytes	237.6845		0.2756	
Traceroute	80 Bytes	171.6925	**176.3293**	0.0608	**0.0618**
	1500 Bytes	180.9661		0.0627	
Archivos	1.4 MB pdf	4 337.0000	**18 138.7500**	11.0000	**49.1250**
	14 MB mp4	31 940.5000		87.2500	

Nota. La Tabla muestra el resumen del retardo de transmisión entre HDN vs SDN.

La Tabla 16 nos muestra una comparativa del retardo de transmisión que tiene la red avanzada REUNA en ambas implementaciones; donde podemos observar que en la prueba Ping de forma general tenemos que en la implementación HDN es de **234.8416 milisegundos**, sin embargo en la implementación SDN tenemos **0.24389875 milisegundos** en esta prueba; asimismo podemos apreciar que en la prueba de traceroute en la HDN tenemos **176.3293 milisegundos** frente a los **0.0618 milisegundos** y por último en la prueba de envió de archivos observamos que en la HDN es de **18 138.7500 milisegundos** comparados con los **49.1250 milisegundos** de retardo de transmisión en la SDN; lo cual nos indica claramente que el retardo de transmisión en la arquitectura de red avanzada REUNA en las implementaciones HDN es mucho mayor que en la SDN.

4.9.2. Retardo de Procesamiento

Este retardo se refiere a los tiempos que le toma al dispositivo para procesar la información de la cabecera de los paquetes y poder enviarlo, si un router recibe un paquete procesará la información de la cabecera para decidir a donde enviarlo o que realizar de acuerdo con su tabla de enrutamiento. Para esta prueba se ha hecho una elección de 2 routers que se encuentran en la red, el criterio que se ha elegido para seleccionar es tomar estos datos en los routers que tienen mayores enlaces, por ello es por lo que se realizará las pruebas en

el router Antofagasta y en el router Santiago; también se ha estimado tomar 40 veces la prueba para una mayor precisión de tiempos; la Tabla 17 muestra las pruebas con los puntos de red que se ha considerado para determinar los tiempos de retardo de procesamiento.

Tabla 17

Distribución de pruebas para retardo de procesamiento

Prueba	Detalle	Caso de prueba	
Ping	ICMPv6 56 Bytes	Antofagasta	Santiago
Traceroute	UDP 80 Bytes	Antofagasta	Santiago

Nota. La Tabla muestra las pruebas a ejecutar para el retardo de procesamiento.

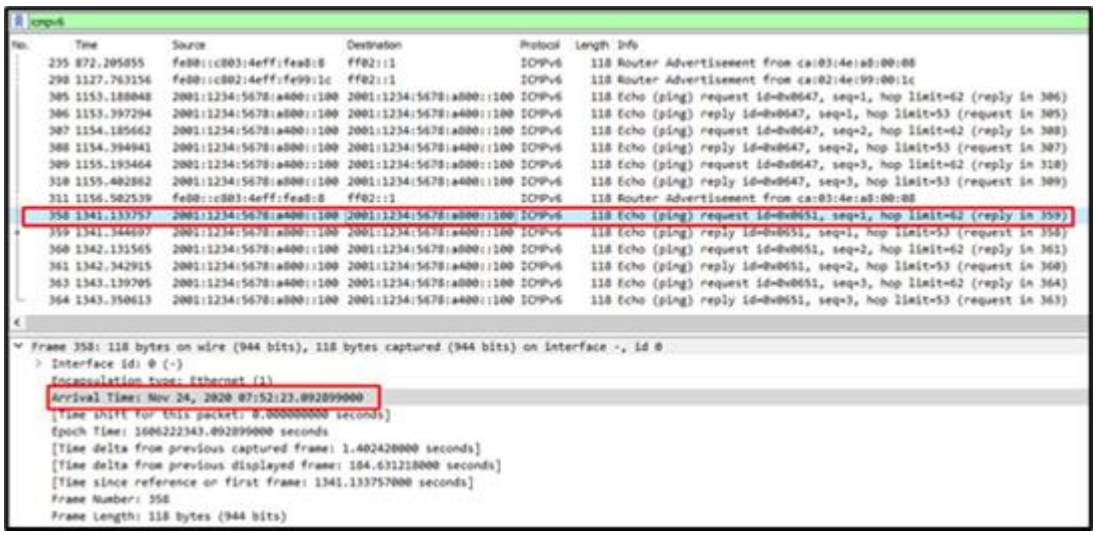
Retardo de procesamiento red avanzada tradicional HDN

- ✓ **Prueba de Ping en router Antofagasta**

Para realizar la toma de estos datos en el primer caso de prueba se ha utilizado la ping, que por defecto envía 3 paquetes icmpv6; previamente se ha inicializado la captura de datos mediante el sniffer wireshark en las interfaces Antofagasta-Iquique y Antofagasta-Serena; una vez enviado los paquetes icmpv6, se procedió a verificar los tiempos; en la Figura 57 se observa la captura del primer paquete icmpv6 de tipo Request; el cual ha sido capturado en la interfaz Antofagasta-iquique del router Iquique a las 07:52:23.092899000.

Figura 57

Interface Antofagasta-Iquique-prueba Ping-Retardo Procesamiento-HDN

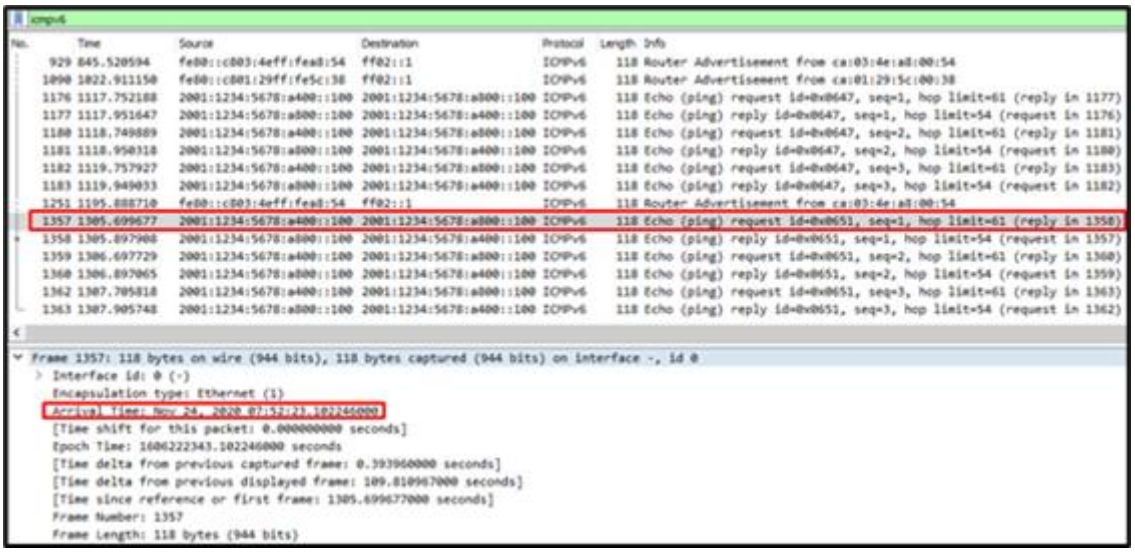


Nota. La Figura muestra la hora que fue capturado el paquete icmpv6 de ping en el enlace Antofagasta-Iquique mediante Wireshark en implementación HDN.

De igual manera la captura se ha realizado en la interfaz Antofagasta-serena que primero pasa por el enlace *cloud*; en la Figura 58 se aprecia que el paquete fue capturado a las 07:52:23.10224600.

Figura 58

Interface Antofagasta-Serena de prueba Ping-Retardo Procesamiento-HDN



Nota. La Figura muestra la hora que fue capturado el paquete icmpv6 de ping en el enlace Antofagasta-Serena mediante Wireshark en implementación HDN.

Para determinar este tiempo de procesamiento; calculamos la diferencia de tiempo el cual será el retardo de procesamiento que le ha tomado al router Antofagasta desde recibir el paquete y enviarlo; el cual es de **0.009347 milisegundos**; este cálculo es para el primer paquete; con los datos capturados de los otros dos paquetes se realizó el mismo procedimiento; en el cual se obtuvo tiempos de retardo de 0.009591 y 0.009540 milisegundos respectivamente; seguidamente se procedió a calcular la media de estos 3 tiempos de procesamiento el cual es para la primera prueba realizada que da como retardo de procesamiento **0.009493 milisegundos**.

- ✓ **Prueba Ping en router Santiago**

El caso de prueba 2 realizado es los mismos paquetes enviados de la prueba 1 pero en esta ocasión fueron capturado en el router Santiago; en la interfaz Santiago con serena fue capturado a las 07:52:23.052048 y en la interfaz que va de Santiago a Rancagua fue capturado a las 07:52:23.062163; el retardo de procesamiento del router para el paquete ICMPv6 es la diferencia el cual es de 0.010115 milisegundos; asimismo se calculó el retardo

de los otros 2 paquetes los cuales fueron de 0.010081 y 0.010169 milisegundos respectivamente; se ha calcula la media de este tiempo de los 3 paquetes el cual es de **0.010121667 milisegundos**.

En la Tabla 18 se muestra el resumen de los tiempos de retardo de procesamiento que se ha calculado para este caso de prueba de ping que se ha ejecutado 40 veces en base a los datos fueron capturados en cada interfaz de los dos casos de prueba que son del router Antofagasta y el router Santiago; en el desarrollo de esta prueba se tiene que la prueba ping en el router Antofagasta tiene una media de retardo de procesamiento de **0.006417 milisegundos,** en el router Santiago el retardo de procesamiento calculado es de **0.010085 milisegundos**; asimismo el retardo de procesamiento en general para la prueba Ping realizada tiene una media de **0.008251 milisegundos**.

Tabla 18

Prueba Ping HDN retardo procesamiento

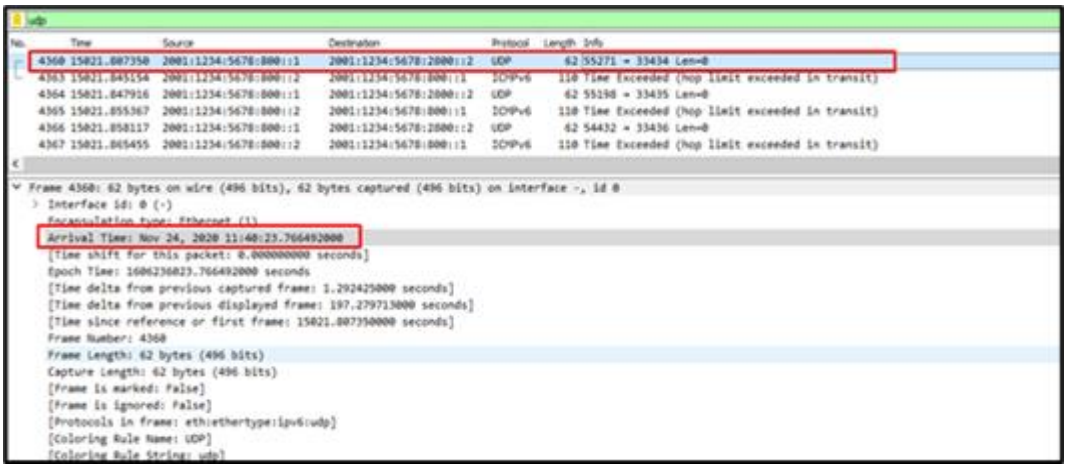
Prueba Ping	Antofagasta (Milisegundos)	Santiago (Milisegundos)
1	0.009493	0.010122
2	0.004817	0.010109
3	0.009127	0.010043
4	0.003934	0.010095
5	0.002463	0.010131
6	0.003170	0.010040
7	0.006600	0.010080
8	0.007621	0.010082
9	0.010153	0.010089
10	0.003045	0.010090
11	0.008914	0.010095
12	0.008149	0.010087
13	0.004746	0.010084
14	0.002220	0.010071
15	0.007038	0.010037
16	0.003588	0.010081
17	0.010319	0.010133
18	0.010044	0.010119
19	0.006145	0.010059
20	0.002872	0.010079
21	0.010191	0.010074
22	0.010138	0.010079
23	0.004831	0.010082
24	0.010285	0.009990
25	0.005901	0.010124
26	0.003777	0.010096
27	0.002880	0.010063
28	0.003640	0.010093
29	0.007590	0.010137
30	0.005381	0.010061
31	0.004175	0.010095
32	0.004508	0.010128
33	0.005695	0.010119
34	0.002449	0.010060
35	0.010374	0.010033
36	0.010238	0.010081
37	0.003887	0.010077
38	0.005938	0.010064
39	0.010136	0.010100
40	0.010213	0.010098

- ✓ **Prueba Traceroute en router Antofagasta**

Debido a que se requiere calcular los tiempos de procesamiento en un router específico de la red, se ha optado por utilizar la herramienta *traceroute* el cual envía 3 paquetes UDP por defecto; la Figura 59 y la Figura 60 muestran la captura del primer paquete UDP enviado; el cual muestra que fue a las 11:40:23.76649200 y a las 11:40:23.834642 respectivamente.

Figura 59

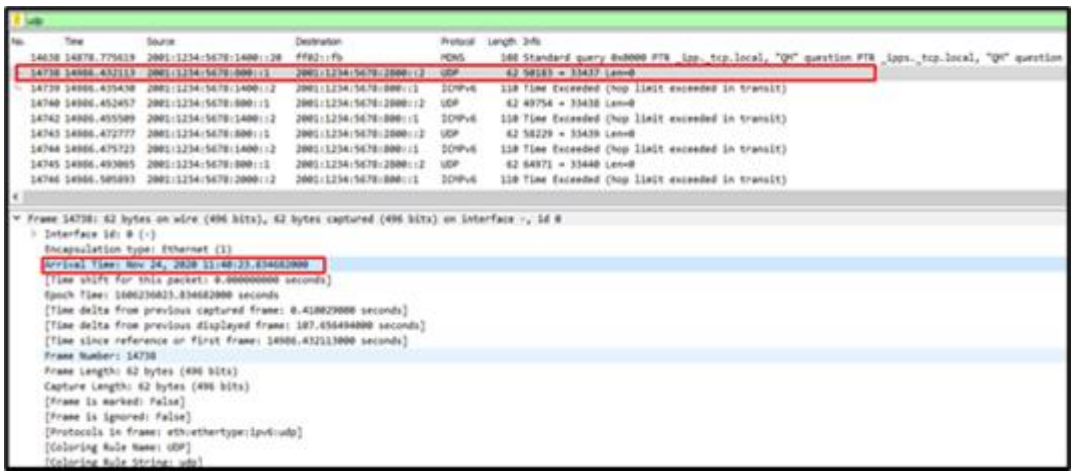
Interface Antofagasta-Iquique de prueba Tracert-Retardo Procesamiento-HDN



Nota. La Figura muestra la hora que fue capturado el paquete UDP de la prueba Tracert en la interfaz Antofagasta-Iquique en la implementación HDN.

Figura 60

Interface Antofagasta-Serena de prueba Tracert-Retardo Procesamiento-HDN



Nota. La Figura muestra la hora que fue capturado el paquete UDP de la prueba Tracert en la interfaz Antofagasta-Serena en la implementación HDN.

Se procedió a calcular el tiempo de procesamiento que le tomo al router para procesar el encabezado del paquete UDP y enviarlo el cual es la diferencia de estos registros de horas que es de 0.068190 milisegundos, los mismo se ha realizado con los otros paquetes, los cuales tienen retardo de 0.047968 y 0.058087 milisegundos respectivamente; posterior a esto se procedió a calcular la media del tiempo de los 3 paquetes el cual es de **0.058082 milisegundos.**

- ✓ **Prueba Traceroute en router Santiago**

De la misma manera se capturó la hora en que fue capturado este paquete; en la interfaz Santiago serena fue a las 11:40:24.094521 y en la interfaz Santiago Rancagua a las 11:40:24.134378; el retardo de procesamiento del router para este primer paquete UDP es de 0.039857 milisegundos; también se realizó este proceso para los otros 2 paquetes; los cuales después de realizar los cálculos dan como tiempos de retardo de 0.009615 y 0.009465; seguidamente se procedió a calcular la media del tiempo de estos 3 paquetes UDP el cual es de **0.019646 milisegundos**.

Después de realizar estos cálculos; la Tabla 19 nos muestra el resumen de los tiempos de procesamiento que ha requerido ambas pruebas realizadas para esta prueba de tracert que se ha ejecutado, si calculamos dichas medias tenemos que para esta prueba en el router Antofagasta el retardo de procesamiento es de **0.059366 milisegundos** y en el router Santiago es de **0.010197 milisegundos**; asimismo la media general del retardo de procesamiento para esta prueba es de **0.034782 milisegundos**.

Tabla 19

Retardo de procesamiento Tracert en HDN

Prueba Tracert	Antofagasta (Milisegundos)	Santiago (Milisegundos)
1	0.058082	0.019646
2	0.087478	0.010006
3	0.056857	0.010059
4	0.045838	0.009885
5	0.080907	0.009479
6	0.080960	0.010067
7	0.081012	0.010089
8	0.046601	0.009978
9	0.044301	0.010063
10	0.090948	0.010083
11	0.043459	0.010056
12	0.080920	0.009810
13	0.045740	0.010070
14	0.042190	0.010049
15	0.046572	0.009889
16	0.053096	0.009911
17	0.045650	0.010059
18	0.045063	0.010026
19	0.047080	0.010068
20	0.045725	0.009704
21	0.058921	0.009989
22	0.042641	0.009949
23	0.079951	0.009752
24	0.045148	0.010009
25	0.049908	0.010024
26	0.062326	0.009811
27	0.065768	0.010039
28	0.081050	0.010067
29	0.042524	0.010092
30	0.061917	0.009975
31	0.081250	0.009878
32	0.045702	0.009883
33	0.052553	0.010057
34	0.080303	0.009822
35	0.071636	0.009966
36	0.044604	0.010038
37	0.091145	0.010078
38	0.047686	0.009708
39	0.046425	0.009638
40	0.054722	0.010092

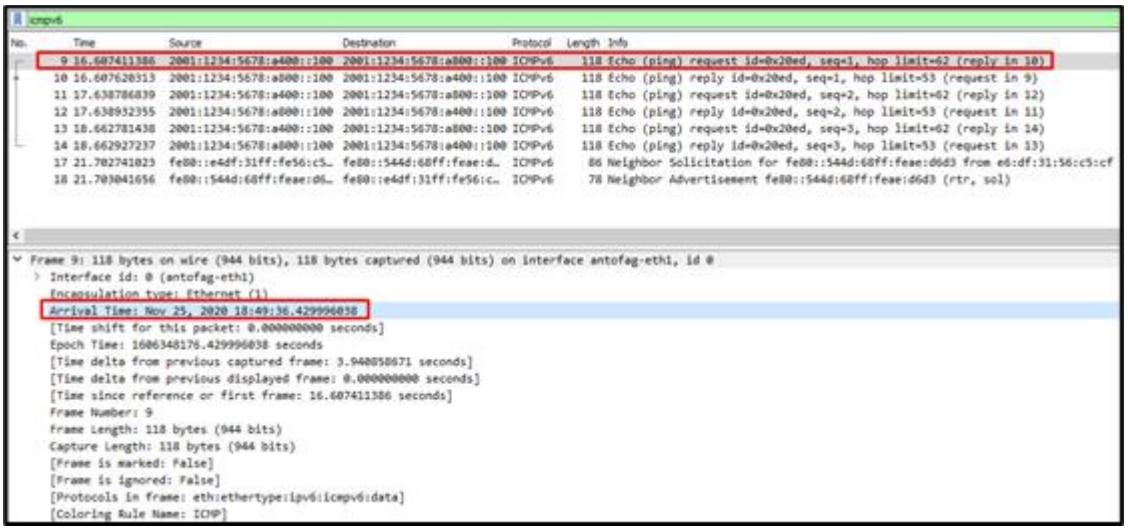
Retardo de procesamiento red SDN

- ✓ **Prueba Ping en router Antofagasta**

Para realizar la toma de estos datos en el primer caso de prueba se ha utilizado la herramienta *ping*, que por defecto envía 3 paquetes icmpv6; previamente se ha inicializado la captura de datos mediante el sniffer wireshark en las interfaces Antofagasta-Iquique y Antofagasta-Serena; una vez enviado los paquetes icmpv6, se procedió a verificar los tiempos; en la Figura 61 se observa la captura del primer paquete icmpv6 de tipo Request; el cual ha sido capturado en la interfaz Antofagasta.iquique del router Iquique a las 18:49:36.429996038.

Figura 61

Interface Antofagasta-Iquique de prueba Ping-SDN-Retardo Procesamiento

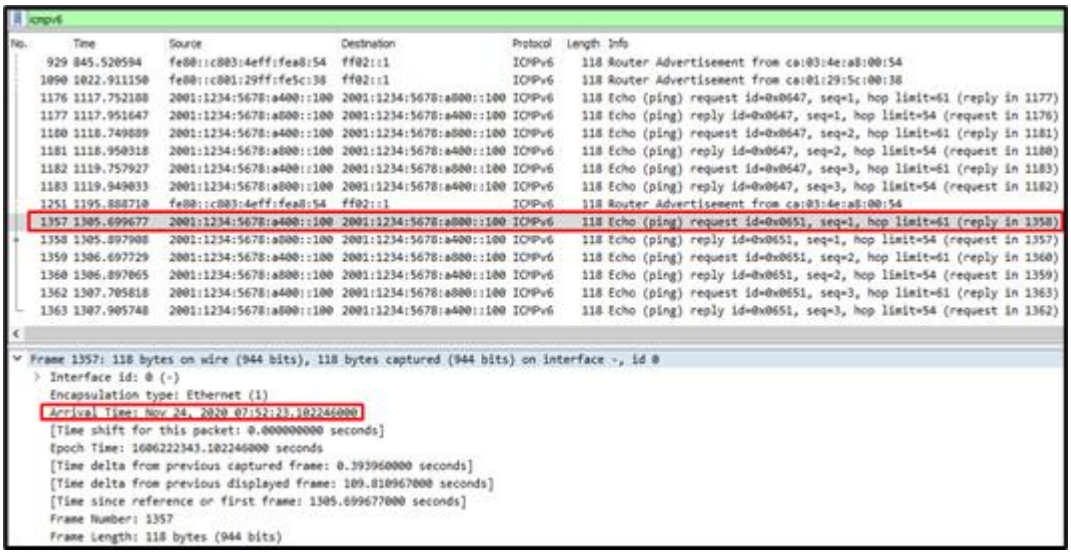


Nota. La Figura muestra la hora que fue capturado el paquete ICPMv6 de la prueba Ping en la interfaz Antofagasta-Iquique en la implementación SDN.

De igual manera la captura se ha realizado en la interfaz Antofagasta-serena que primero pasa por el enlace *cloud*; en la Figura 62 se aprecia que el paquete fue capturado a las 18:49:36.430009585.

Figura 62

Interface Antofagasta-Serena de prueba Ping-SDN-Retardo Procesamiento



Nota. La Figura muestra la hora que fue capturado el paquete ICPMv6 de la prueba Ping en la interfaz Antofagasta-Serena en la implementación SDN.

Para determinar este tiempo de procesamiento; calculamos la diferencia de tiempo el cual será el retardo de procesamiento que le ha tomado al router Antofagasta desde recibir el paquete y enviarlo; el cual es de 0.000014; este cálculo es para el primer paquete; con los datos capturados de los otros dos paquetes se realizó el mismo procedimiento; en el cual se obtuvo tiempos de retardo de 0.000013 y 0.000013 Ms respectivamente; seguidamente se procedió a calcular la media de estos 3 tiempos de procesamiento el cual es para la primera prueba realizada que da como retardo de procesamiento **0.000013002 milisegundos**.

- ✓ **Prueba Ping en router Santiago**

El caso de prueba 2 realizado es los mismos paquetes enviados de la prueba 1 pero en esta ocasión fueron capturado en el router Santiago; en la interfaz Santiago con serena fue capturado a las 18:49:36.430023510 y en la interfaz que va de Santiago a Rancagua fue capturado a las 18:49:36.430034562; el retardo de procesamiento del router para el paquete ICMPv6 es la diferencia el cual es de 0.000011052 Ms; asimismo se calculó el retardo de los otros 2 paquetes los cuales fueron de 0.000007 y 0.000008 Ms respectivamente; se ha calculado la media de este tiempo de los 3 paquetes el cual es de 0.000009 milisegundos.

La Tabla 20 muestra el resumen de todos estos tiempos de procesamiento obtenidos para esta primera prueba el cual fue realizado 40 veces.

Tabla 20

Retardo de procesamiento Ping SDN

Prueba Ping	Antofagasta (Milisegundos)	Santiago (Milisegundos)
1	0.000013	0.000009
2	0.000012	0.000008
3	0.000011	0.000008
4	0.000013	0.000009
5	0.000012	0.000008
6	0.000011	0.000008
7	0.000012	0.000008
8	0.000013	0.000009
9	0.000013	0.000014
10	0.000012	0.000009
11	0.000012	0.000008
12	0.000014	0.000010
13	0.000013	0.000009
14	0.000014	0.000009
15	0.000012	0.000008
16	0.000012	0.000009
17	0.000012	0.000009
18	0.000013	0.000008
19	0.000012	0.000008
20	0.000012	0.000008
21	0.000010	0.000007
22	0.000012	0.000008
23	0.000011	0.000008
24	0.000009	0.000006
25	0.000012	0.000007
26	0.000014	0.000009
27	0.000014	0.000010
28	0.000017	0.000010
29	0.000012	0.000008
30	0.000013	0.000010
31	0.000012	0.000010
32	0.000012	0.000008
33	0.000013	0.000011
34	0.000012	0.000008
35	0.000012	0.000008
36	0.000012	0.000008
37	0.000012	0.000009
38	0.000012	0.000008
39	0.000014	0.000010
40	0.000013	0.000009

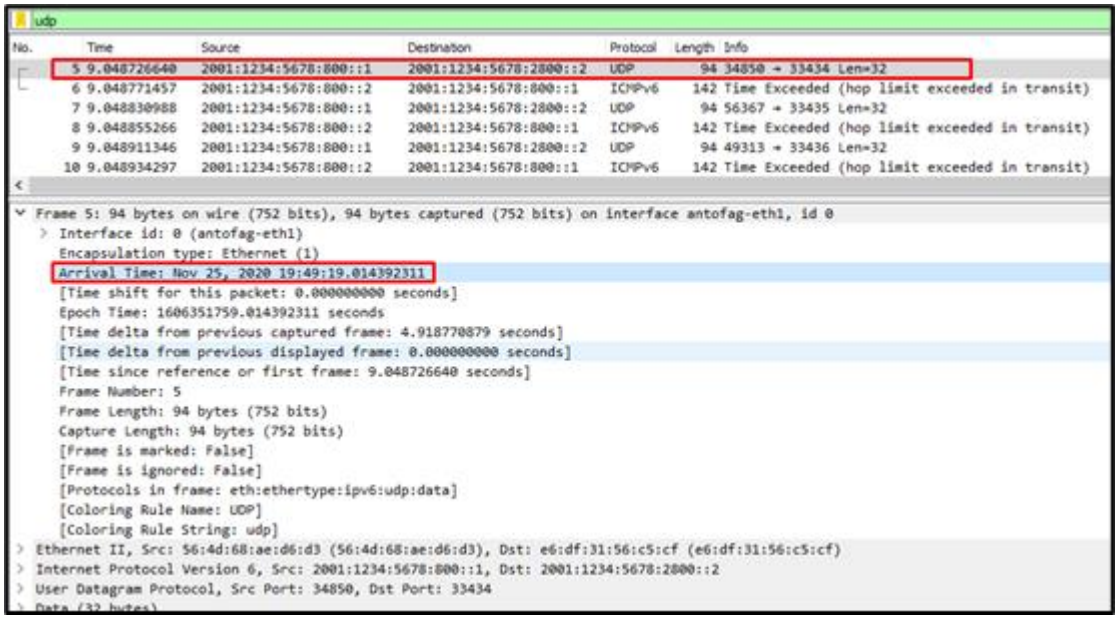
Basado en los datos de la Tabla 20, se procedió a calcular la media de retardo de procesamiento en el router Antofagasta el cual es de **0.000012 milisegundos**, de la misma manera en el router Santiago es de **0.000009 milisegundos**; finalmente podemos obtener que el retardo de procesamiento en general para esta prueba de Ping es de **0.000010 milisegundos**.

- ✓ **Prueba Traceroute en router Antofagasta**

Debido a que se requiere calcular los tiempos de procesamiento en un router específico de la red, se ha optado por utilizar la herramienta *traceroute* el cual envía 3 paquetes UDP por defecto; la Figura 63 muestra la captura del primer paquete UDP enviado; el cual muestra que fue a las 19:49:19.014392311.

Figura 63

Interface Antofagasta-Iquique de prueba Tracert-SDN-Retardo Procesamiento

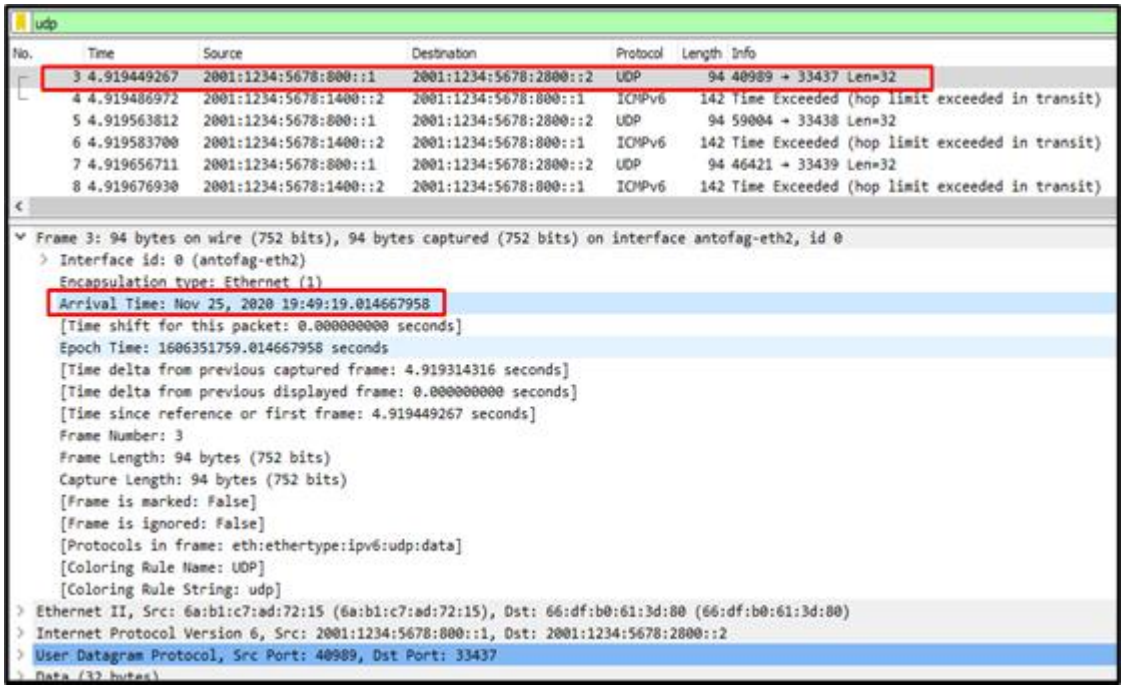


Nota. La Figura muestra la hora que fue capturado el paquete UDP de la prueba Tracert en la interfaz Antofagasta-Iquique en la implementación SDN.

De igual manera se capturo el paquete UDP en la siguiente interfaz, la Figura 64 nos muestra que fue a las 19:49:19.014667958.

Figura 64

Interface Antofagasta-Serena de prueba Tracert-SDN-Retardo Procesamiento



Nota. La Figura muestra la hora que fue capturado el paquete UDP de la prueba Tracert en la interfaz Antofagasta-Serena en la implementación SDN.

Se procedió a calcular el tiempo de procesamiento que le tomo al router para procesar el encabezado del paquete UDP y enviarlo el cual es la diferencia de estos registros de horas que es de 0.000276 milisegundos; de la misma manera se realizó para el segundo y tercer paquetes los cuales se calculó el retardo de 0.000286 y 0.000298 milisegundos respectivamente; seguidamente se procedió a calcular la media del tiempo de los 3 paquetes el cual es de **0.000287 milisegundos**.

- ✓ **Prueba Traceroute en router Santiago**

De la misma manera se capturó la hora en que fue capturado este paquete; en la interfaz Santiago serena fue a las 19:49:19.015342071 y en la interfaz Santiago Rancagua a las 19:49:19.015344503; el retardo de procesamiento del router para este primer paquete UDP es de 0.000002 milisegundos; también se realizó este proceso para los otros 2 paquetes; los cuales después de realizar los cálculos dan como tiempos de retardo de 0.000002 y 0.000002; seguidamente se procedió a calcular la media del tiempo de estos 3 paquetes UDP el cual es de **0.000002 milisegundos**.

Después de realizar estos cálculos por cada uno de los 3 paquetes enviados; la Tabla 21 nos muestra el resumen de los tiempos de procesamiento obtenidos en ambos casos de pruebas realizadas de traceroute.

Seguidamente se procedió a calcular la media de esta prueba por cada caso de prueba, es decir por cada router especificado, esto lo podemos apreciar en la Tabla 22, donde, para el router Antofagasta se tiene un tiempo de retardo de procesamiento de **0.000223 milisegundos**, y en el caso del router Santiago el retardo de procesamiento calculado fue de **0.000003 milisegundos**; así que dicha prueba realizada tiene una media de retardo de procesamiento de **0.000113 milisegundos**.

Tabla 21

Retardo de procesamiento en Router Santiago SDN

Prueba Tracert	Antofagasta (Milisegundos)	Santiago (Milisegundos)
1	0.000009	0.000002
2	0.000008	0.000002
3	0.000008	0.000002
4	0.000009	0.000003
5	0.000008	0.000002
6	0.000008	0.000002
7	0.000008	0.000003
8	0.000009	0.000002
9	0.000014	0.000003
10	0.000009	0.000002
11	0.000008	0.000002
12	0.000010	0.000002
13	0.000009	0.000004
14	0.000009	0.000003
15	0.000008	0.000002
16	0.000009	0.000002
17	0.000009	0.000002
18	0.000008	0.000003
19	0.000008	0.000002
20	0.000008	0.000002
21	0.000007	0.000003
22	0.000008	0.000003
23	0.000008	0.000002
24	0.000006	0.000002
25	0.000007	0.000003
26	0.000009	0.000002
27	0.000010	0.000003
28	0.000010	0.000004
29	0.000008	0.000002
30	0.000010	0.000002
31	0.000010	0.000002
32	0.000008	0.000003
33	0.000011	0.000003
34	0.000008	0.000002
35	0.000008	0.000003
36	0.000008	0.000002
37	0.000009	0.000003
38	0.000008	0.000003
39	0.000010	0.000002
40	0.000009	0.000002

La Tabla 22 muestra la comparación de retardo de procesamiento entre ambas arquitecturas de red.

Tabla 22

Comparación de retardo de procesamiento HDN vs SDN

Prueba	Router	HDN (Milisegundos)		SDN (Milisegundos)	
Ping	Antofagasta	0.006417	**0.008251**	0.000012	**0.000010**
	Santiago	0.010085		0.000009	
Tracert	Antofagasta	0.059366	**0.034782**	0.000223	**0.000113**
	Santiago	0.010197		0.000003	

Nota. La Tabla muestra el resumen de los tiempos de retardo de procesamiento de HDN vs SDN.

4.9.3. Variación de Retardo

La variación de retardo se refiere a cuanto varían estos retardos, que también es llamado jitter, para esta prueba se ha utilizado la herramienta iperf que por defecto un paquete de por un periodo de diez paquetes y nos indica esta variación que existe (jitter). Para instalar la herramienta solo basta con ingresar el comando *apt-get install iperf3,* esta versión es la que se encuentra disponible para IPv6, aunque Mininet ya lo trae instalado por defecto, solo se instaló en las máquinas virtuales para el caso de la red tradicional; debemos tener en cuenta iperf funciona en puertos desconocidos y por defecto utiliza el 5201.

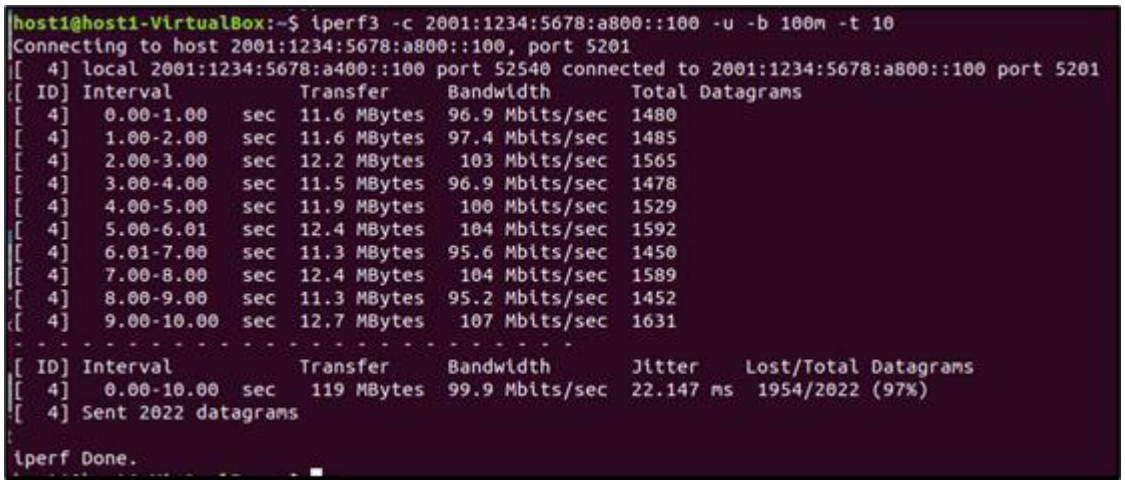
Para realizar esta prueba se realiza entre el host1 y el host2, debemos hacer que uno de ellos se encuentre en modo escucha que actúa como servidor y el otro host como cliente, esto se ha realizado en 2 casos de prueba, el cual consiste en enviar un paquete de 100 y 5000 Megabytes; asimismo al igual que en los anteriores indicadoes se realiza 40 veces dichas pruebas para obtener mayor precisión.

Variación de retardo Red Avanzada Tradicional

Para proceder a realizar la prueba se ha determinado que h1 será el cliente y h2 será el servidor, en la Figura 65, se aprecia el envío por un periodo de 10 segundos a través de parámetro -t, el paquete de tamaño de 100 MegaBytes, el cual es ejecutado través de iperf3 que es la versión de iperf disponible para IPv6, que son enviados desde el host1 que actúa como cliente hacia el host2 que actuará como servidor. Se observa que el jitter es de **22.1470 milisegundos**.

Figura 65

Variación de retardo prueba 100 MB en cliente host1 HDN

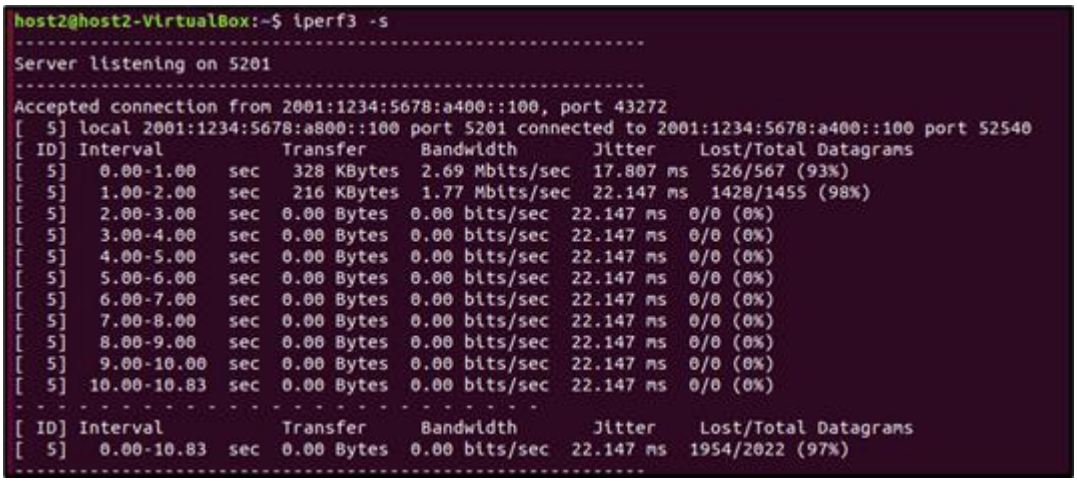


Nota. La Figura muestra al Host1 actuando como cliente de la prueba Iperf para las variaciones de retardo con un paquete UDP de 100 MB en la implementación HDN.

En la Figura 66 se observa que el host2 actúa como servidor y que recibe los paquetes y coincide el jitter con el que muestra host1 siendo **22.1470 milisegundos.**

Figura 66

Variación de retardo prueba de 100 MB en server host2 HDN

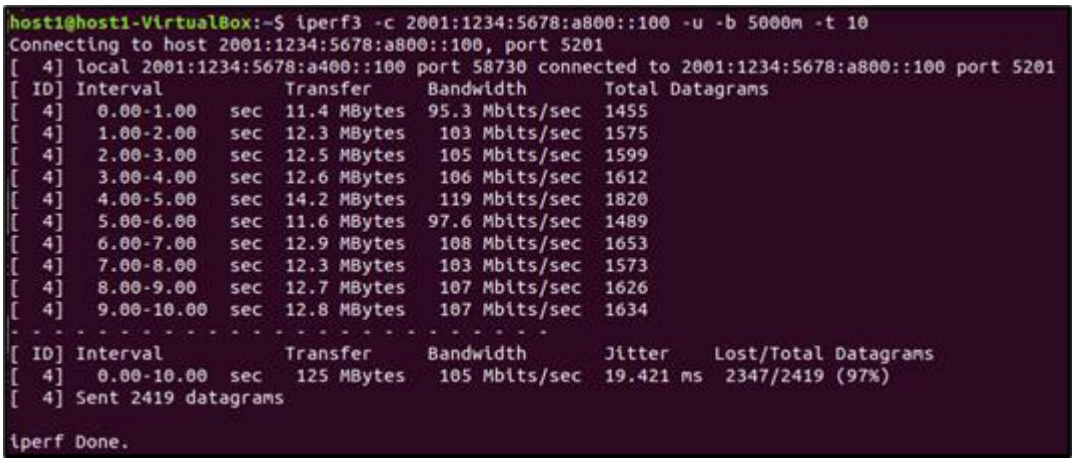


Nota. La Figura muestra al Host2 actuando como servidor de la prueba Iperf para las variaciones de retardo con un paquete UDP de 100 MB en la implementación HDN.

Para el caso de prueba 2 de la misma manera se ha ejecutado la prueba; la Figura 67 muestra esta variación de retardo que es **de 19.4210 milisegundos**.

Figura 67

Variación de retardo prueba de 5000 MB en server host1 HDN



Nota. La Figura muestra al Host1 actuando como cliente de la prueba Iperf para las variaciones de retardo con un paquete UDP de 5000 MB en la implementación HDN.

Una vez identificado estos datos, se construyó la Tabla 23; la cual nos muestra el resumen de las variaciones de retardo (jitter) de las 40 pruebas realizadas.

Asimismo, se procedió a calcular la media de ambas pruebas realizadas, donde se obtuvo que para la prueba de 100 MB las variaciones de retardo acumulan un promedio de **21.1718 milisegundos** y para la prueba de 500 MB la variación de retardo es de **18.9351 milisegundos**; si calculamos la media en general entre ambos casos de prueba tenemos que la variación de retardo es de **20.0535 milisegundos**.

Tabla 23

Variación de retardo en HDN

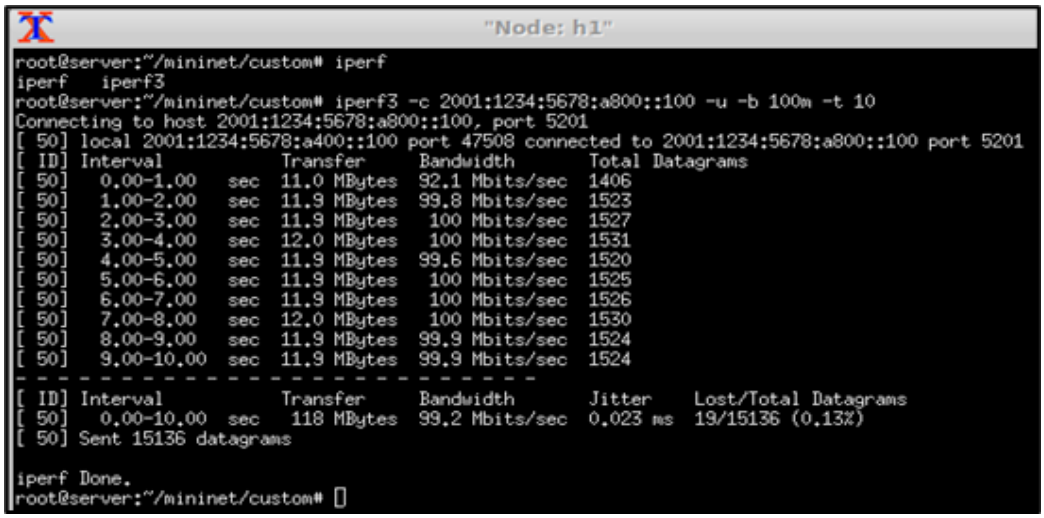
Prueba	100 MB (Milisegundos)	5000 MB (Milisegundos)
1	22.1470	19.4210
2	21.1710	20.1600
3	25.2780	15.3510
4	22.7470	16.3910
5	18.3520	21.8420
6	18.2550	18.2250
7	18.2600	16.3400
8	29.0180	24.6070
9	22.8890	19.1960
10	16.1770	18.9840
11	31.6500	18.6490
12	24.8380	15.1760
13	17.2780	22.0650
14	22.8960	20.8600
15	19.4720	19.9330
16	28.1660	18.9240
17	17.7820	17.4690
18	17.1180	20.3660
19	22.2290	16.8690
20	23.1680	21.6710
21	19.6780	20.7930
22	19.5950	18.8050
23	24.8530	18.6770
24	24.8450	19.7880
25	19.7860	17.8780
26	17.1390	13.7460
27	17.8720	18.8450
28	20.8020	18.5470
29	17.8220	17.6860
30	23.7370	18.7670
31	18.4240	16.6120
32	18.1580	19.2000
33	17.9370	16.1480
34	22.2360	21.0190
35	15.9850	26.0820
36	20.2910	18.1040
37	26.9960	17.7880
38	26.0570	20.2060
39	17.1270	17.0130
40	18.6420	19.2000

Variación de Retardo Red Avanzada SDN

Asimismo, se procedió a designar el host1 como cliente y el host2 como servidor, en la Figura 68 se observa al host1 enviando 10 paquetes de 100 MB hacia el host2, y tiene una variación de retardo de 0.0230 milisegundos; y en la Figura 69 el host 2 que actúa como servidor el cual está escuchándose confirma los 0.0230 milisegundos de jitter.

Figura 68

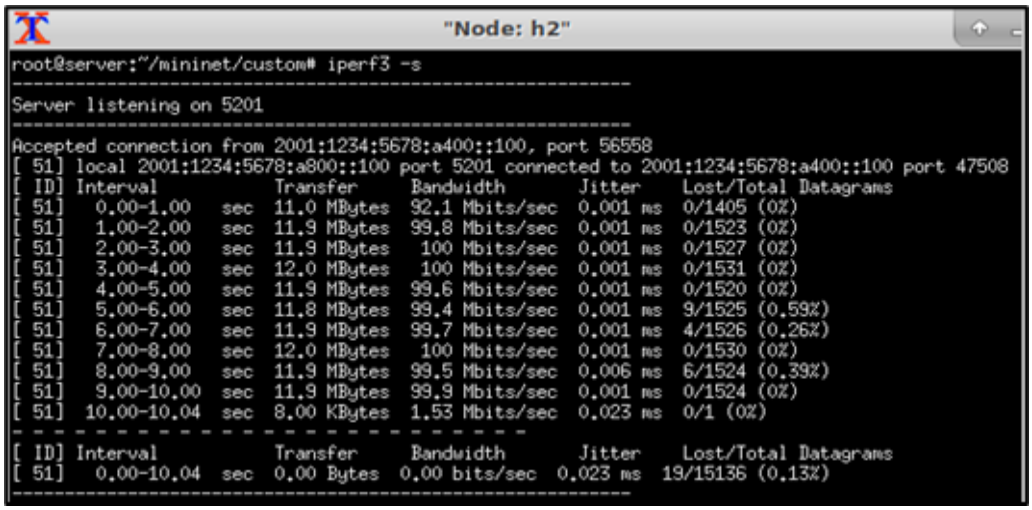
Variación de retardo prueba 100 MB en cliente host1 SDN



Nota. La Figura muestra al Host1 actuando como cliente de la prueba Iperf para las variaciones de retardo con un paquete UDP de 100 MB en la implementación HDN.

Figura 69

Variación de retardo prueba 100 MB en server host2 SDN

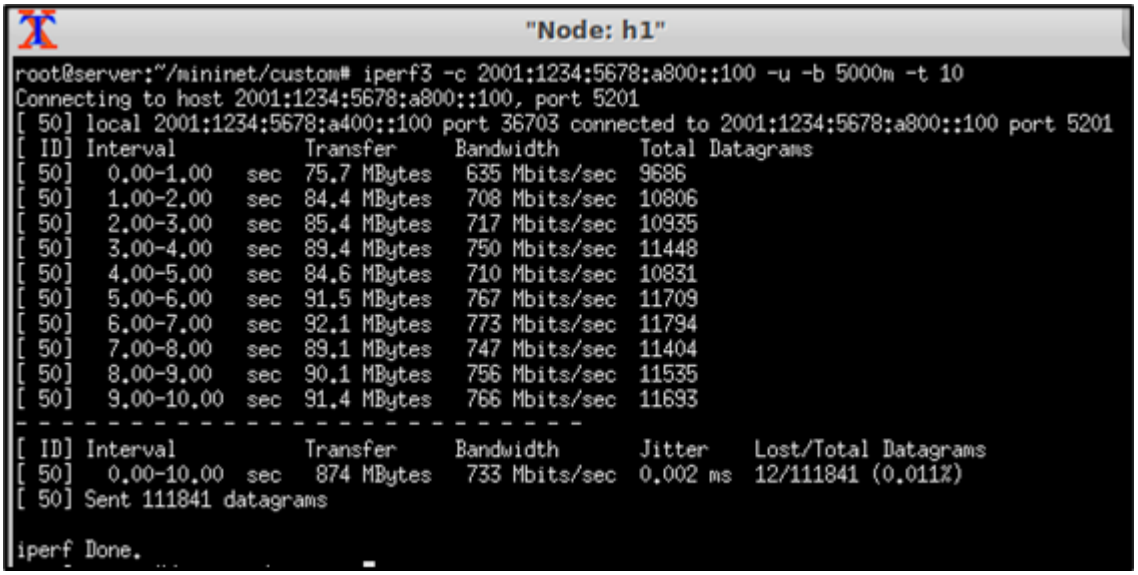


Nota. La Figura muestra al Host1 actuando como cliente de la prueba Iperf para las variaciones de retardo con un paquete UDP de 100 MB en la implementación HDN.

De la misma manera el caso de prueba 2 se ha realizado; la Figura 70 nos muestra que la variación de retardo es de 0.0020 milisegundos.

Figura 70

Variación de retardo prueba 5000 MB en cliente host1 SDN



Nota. La Figura muestra al Host2 actuando como servidor de la prueba Iperf para las variaciones de retardo con un paquete UDP de 5000 MB en la implementación HDN.

La Tabla 24 muestra el resumen de los tiempos de variación de retardo que es definido como jitter de las 10 pruebas realizadas.

Tabla 24

Variación de retardo SDN

Prueba	100 MB (Milisegundos)	5000 MB (Milisegundos)
1	0.0230	0.0020
2	0.0140	0.0020
3	0.0170	0.0020
4	0.0170	0.0020
5	0.0210	0.0020
6	0.0160	0.0020
7	0.0240	0.0020
8	0.0220	0.0020
9	0.0120	0.0020
10	0.0210	0.0010
11	0.0240	0.0020
12	0.0220	0.0010
13	0.0220	0.0020
14	0.0210	0.0020
15	0.0160	0.0030
16	0.0220	0.0020
17	0.0230	0.0020
18	0.0130	0.0010
19	0.0240	0.0020
20	0.0230	0.0020
21	0.0140	0.0020
22	0.0230	0.0020
23	0.0200	0.0020
24	0.0210	0.0030
25	0.0160	0.0020
26	0.0150	0.0020
27	0.0210	0.0010
28	0.0220	0.0010
29	0.0230	0.0040
30	0.0150	0.0030
31	0.0250	0.0020
32	0.0140	0.0020
33	0.0210	0.0020
34	0.0270	0.0020
35	0.0210	0.0020
36	0.0220	0.0020
37	0.0200	0.0020
38	0.0220	0.0020
39	0.0220	0.0050
40	0.0240	0.0020

Mediante los datos obtenidos en la Tabla 23, se procedió a calcular la media de ambas pruebas realizadas, donde se obtuvo que para la prueba de 100 MB las variaciones de retardo acumulan un promedio de **0.0201250 milisegundos** y para la prueba de 500 MB la variación de retardo es de **0.0020750 milisegundos**; si calculamos la media en general entre ambos casos de prueba tenemos que la variación de retardo es de **0.0111000 milisegundos**.

En la Tabla 25 se realiza la comparación de variación de retardo que es conocido como jitter en ambas arquitecturas.

Tabla 25

Comparación de variación de retardo entre HDN y SDN

Prueba	Prueba	HDN (Milisegundos)		SDN (Milisegundos)	
Iperf	100 MB	21.1718	**20.0535**	0.0201	**0.0111**
	5000 MB	18.9351		0.0021	

Nota. La Tabla muestra el resumen de la variación de retardo de las pruebas ejecutadas para HDN vs SDN.

4.9.4. Pérdida de Paquetes

Para esta prueba se considera dos casos de prueba, el primero consiste en enviar 10 paquetes ICMPv6 a través de ping desde el host1 hacia el host2, en el segundo caso de prueba se ha considerado realizar un Stream donde el host1 actua como servidor y el host2 como cliente, asimismo se ha considerado tomar 40 veces la prueba para tener una mayor precisión; en la Tabla 26 se especifica las pruebas a realizar.

Tabla 26

Distribución de pruebas para pérdida de paquetes

Prueba	Detalle
Ping	Paquetes ICMPv6
Stream	Paquetes UDP

Nota. La Tabla muestra las pruebas a ejecutarse para la pérdida de paquetes.

Perdida de paquetes Red Avanzada Tradicional

- ✓ **Prueba de Ping**

Para determinar la respuesta a la pérdida de paquetes en la red, se ha considerado la primera realizar dos casos de prueba; el primer caso consiste en enviar 10 paquetes ICMPv6, desde el host1 hacia el host2; para el desarrollo de la prueba se ha configurado la perdida de

paquetes de un 20 % a través del parámetro tc qdisc, en la Figura 71 se observa la configuración para que exista una pérdida de paquetes del 20 % a través de la interfaz del host1, el cual si bien le estamos indicando que tenga una pérdida de paquetes del 20 %, lo que ocurre es que al realizar este cambio ocurre la de pérdidas de paquetes acercándose a este porcentaje de 20 %; sin embargo esto puede variar de acuerdo con la arquitectura de red implementada.

Figura 71

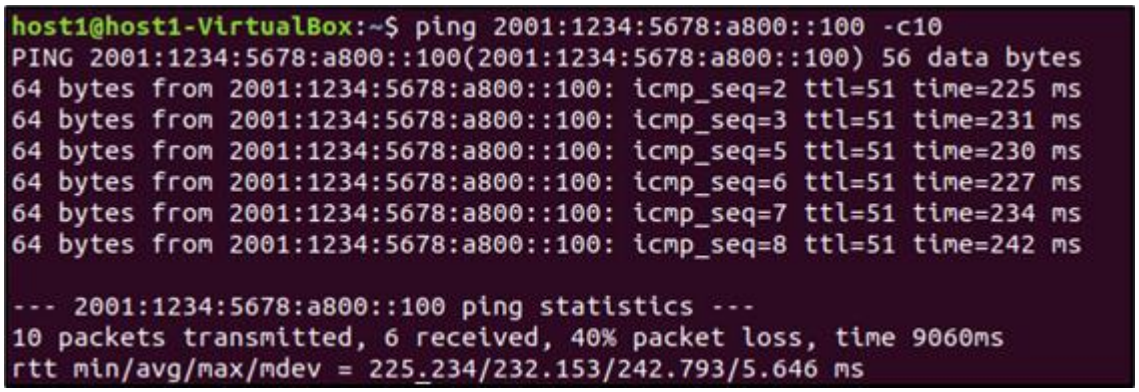
Configuración para pérdida de paquetes en HDN

Nota. La Figura muestra la configuración forzada realizada en la interface del host1 en HDN.

Después de configurar, se procedió a enviar los 10 paquetes ICMPv6 mediante ping, en la Figura 72 se observa los datos capturados, se aprecia que efectivamente existe pérdida de paquetes; de los 10 paquetes enviados 6 de ellos han sido recibidos y 4 de ellos fueron perdidos, se observa que los paquetes con secuencia 1, 4, 9 y 10 son los que se han perdido.

Figura 72

Pérdida de paquetes con ping en HDN



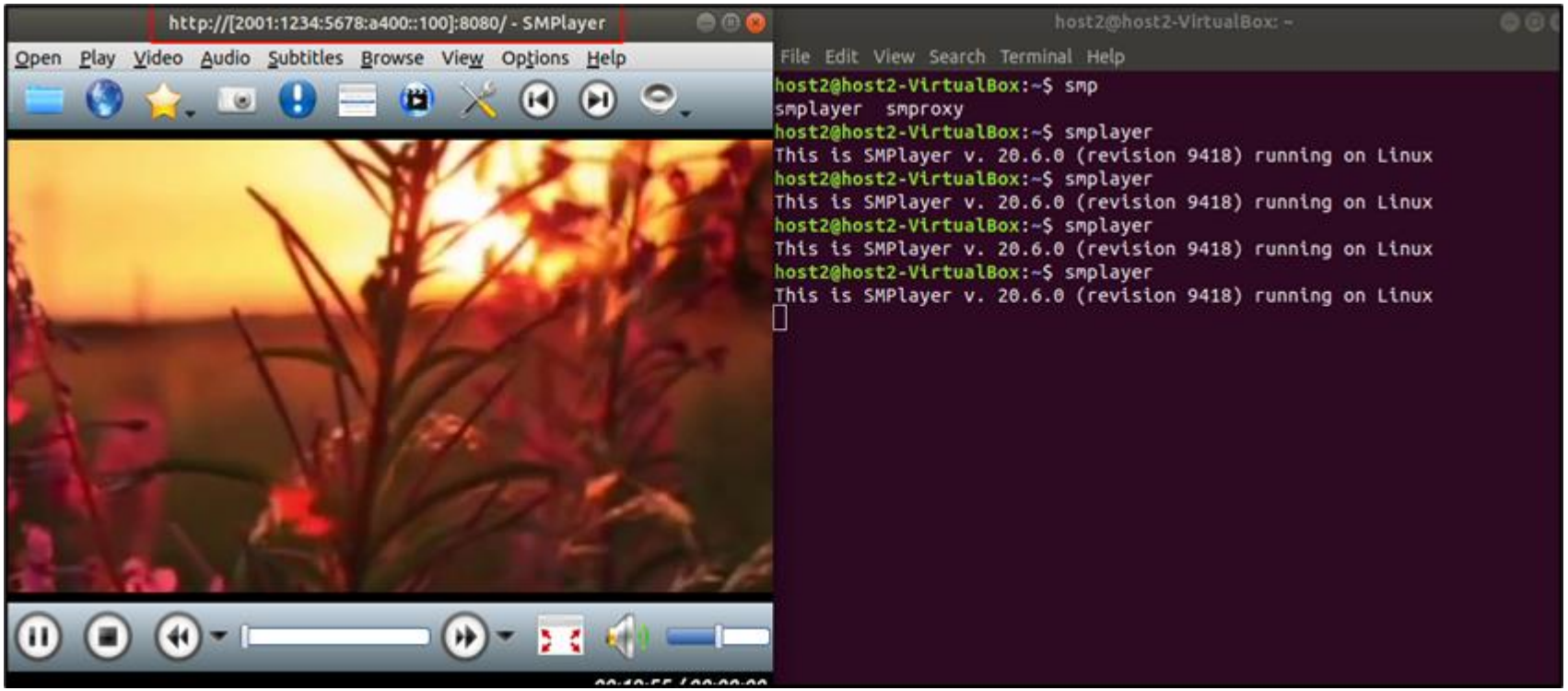
Nota. La Figura muestra los paquetes perdidos ICMPv6 de la prueba Ping en la interface del host1 en HDN.

También se ha realizado la segunda prueba el cual consiste en realizar un Stream de video, esto porque todos lo que es voz y video es más sensible a la pérdida de paquetes; donde el Host1 que actuará como servidor y el host2 que actuará como cliente, para poder realizar esta configuración del Escenario, se muestra en el **Anexo N° 3.**

En la Figura 73 se observa el momento en que el hos2 se conecta al host1 y se visualiza el video en stream.

Figura 73

Streaming en HDN

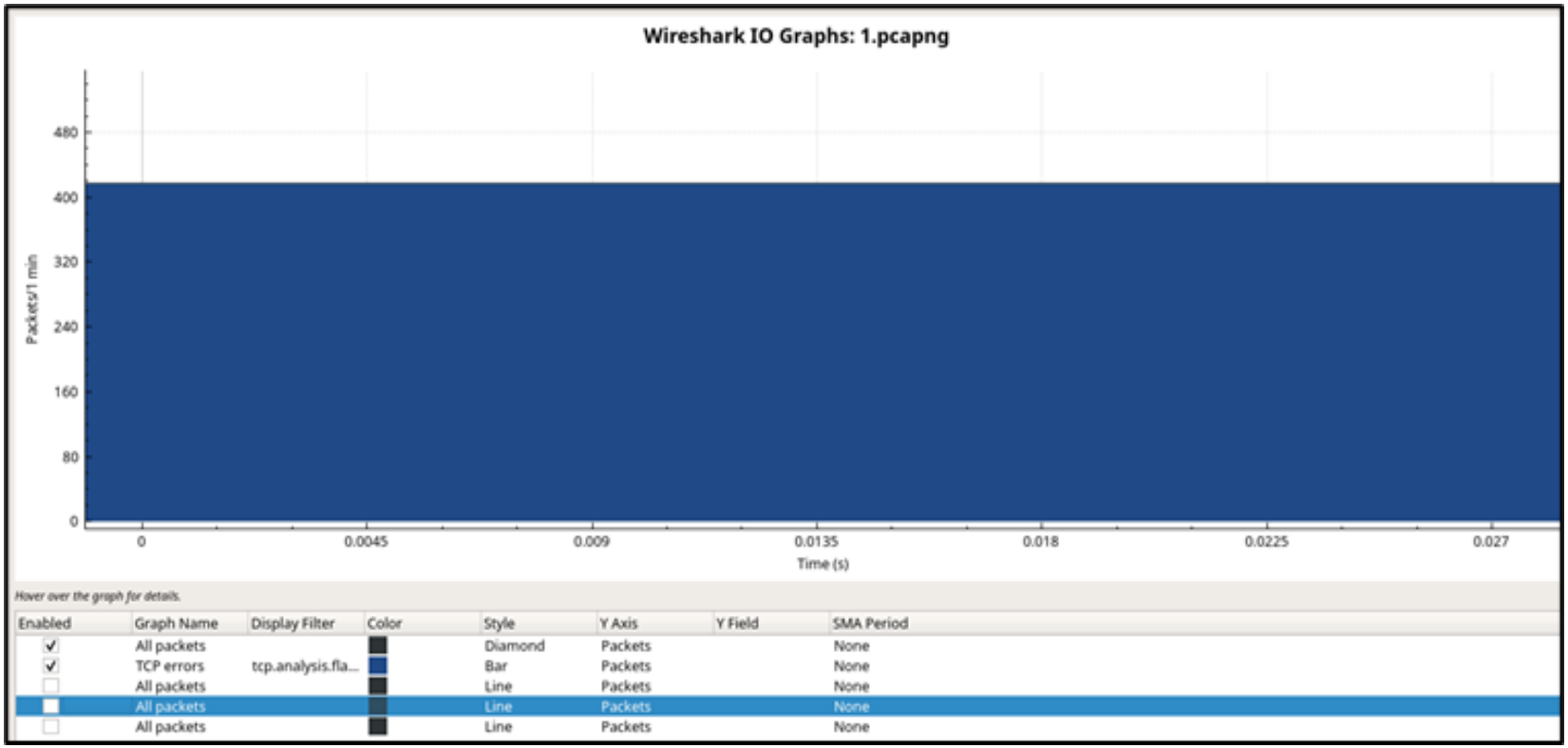


Nota. La Figura muestra como el proceso Streaming entre h1 como servidor y host2 como cliente en la implementación HDN.

La Figura 74 muestra los datos capturados del Stream, donde se aprecia la cantidad ha existido errores TCP de streaming y que se ha tenido que asegurar la transmisión del video, se observa son 417 los paquetes perdidos que contienen errores TCP, los cuales fueron reenviados para asegurar el streaming; el video tiene una duración de 60 segundos y su peso es de 14 MB.

Figura 74

Pérdida de paquetes prueba Streaming HDN



Nota. La Figura muestra la pérdida de paquetes del streming realizado en la implementación HDN.

Después de obtener los datos para de la pérdida de paquetes, la Tabla 27 nos muestra el resumen de la cantidad de paquetes perdidos por cada caso de prueba especificado que son el de Ping y el de Strem.

Seguidamente se procedió a calcular la media de la pérdida de paquetes, donde la prueba de ping nos indica que tenemos 3.4250 paquetes perdidos y en la prueba de stream es de 458.35 paquetes perdidos; asimismo si calculamos la media general entre ambas pruebas tenemos que es **de 230.8875 paquetes perdidos**.

Tabla 27

Pérdida de paquetes HDN

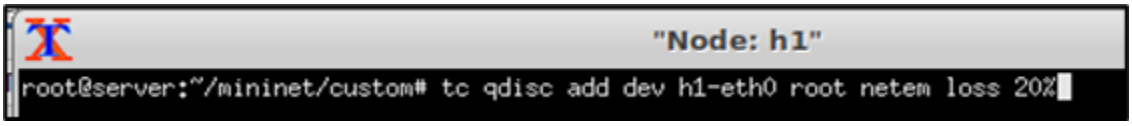
Prueba	Ping (Paquetes perdidos)	Stream (Paquetes perdidos)
1	4	417
2	5	395
3	3	477
4	3	365
5	2	342
6	3	458
7	4	343
8	2	255
9	3	462
10	3	342
11	3	483
12	3	439
13	2	372
14	7	438
15	1	420
16	5	382
17	4	363
18	3	382
19	6	414
20	2	307
21	3	465
22	8	267
23	3	549
24	3	457
25	4	450
26	4	728
27	4	584
28	2	506
29	3	595
30	2	600
31	3	485
32	4	525
33	3	421
34	4	532
35	2	561
36	3	493
37	2	633
38	4	637
39	4	408
40	4	582

Perdida de paquetes Red Avanzada SDN

Para la segunda prueba también se ha configurado la perdida de paquetes en la interfaz del host 1, esto lo podemos apreciar en la Figura 75.

Figura 75

Configuración de pérdida de paquetes en SDN

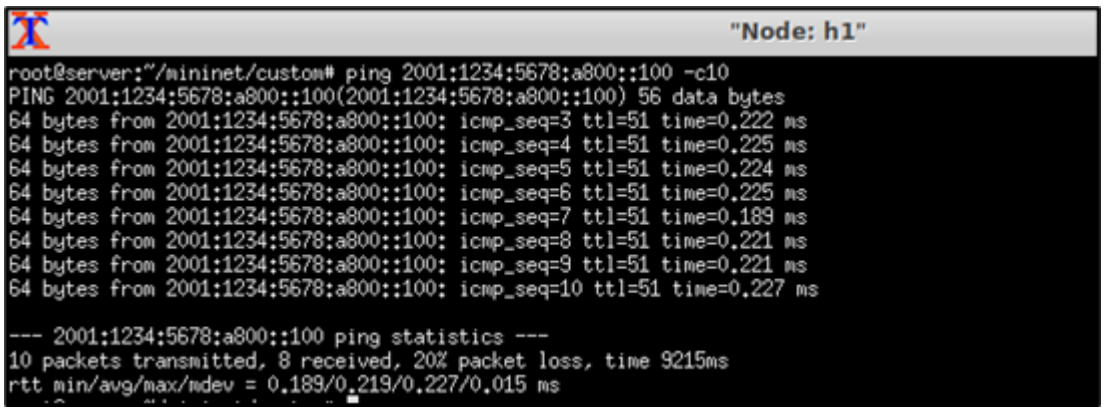


Nota. La Figura muestra la configuración forzada realizada en la interface del host1 en SDN.

Después de configurar se procedió a enviar los10 paquetes ICMPv6 mediante ping desde h1 hacia Arica, la Figura 76 muestra los resultados de esta prueba, donde que existe una pérdida de paquetes, de los cuales 8 han sido recibidos y 2 se han perdido; se observa que los paquetes con secuencia 1 y 2 fueron los que se perdieron

Figura 76

Pérdida de paquetes con ping en SDN



Nota. La Figura muestra los paquetes perdidos ICMPv6 de la prueba Ping en la interface del host1 en SDN.

De igual forma que en la Red Avanzada Tradicional se ha realizado otro tipo de prueba el cual consiste en realizar un Stream de video en el host1 que actuará como servidor y el host2 que actuará como cliente, para poder realizar esta configuración del escenario, se muestra en el Anexo N° 3.

En la Figura 77 se observa el momento en que el hos2 se conecta al host1 y se visualiza el video en stream.

Figura 77

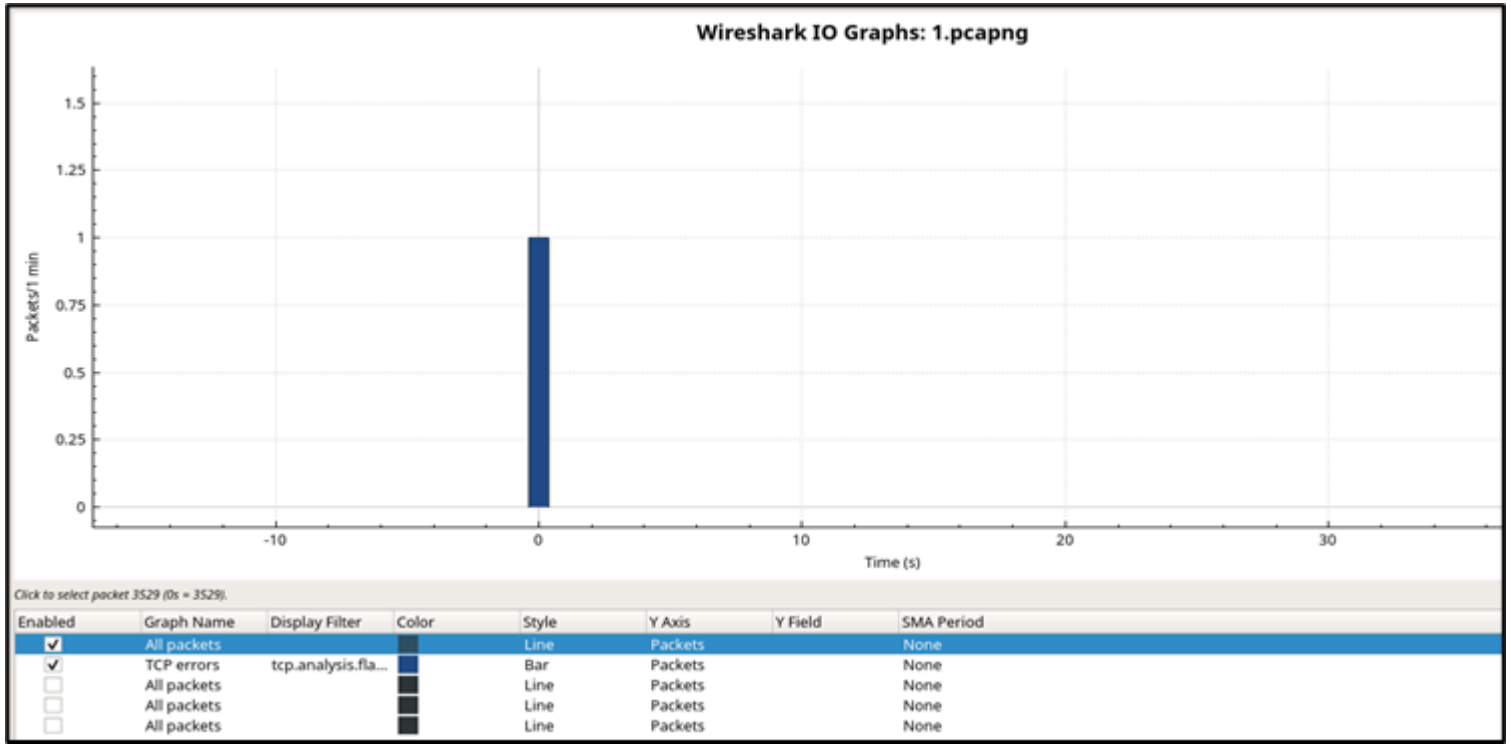
Streaming en SDN

Nota. La Figura muestra como el proceso Streaming entre host1 como servidor y host2 como cliente en implementación SDN.

La Figura 78 muestra datos capturado acerca del Stream mediante Wireshark, donde se observa que se ha tenido un solo paquete con error de tcp el cual tuvo que ser reenviado para asegurar el streaming; la Tabla muestra el resumen de la pérdida de paquetes en SDN.

Figura 78

Pérdida de paquetes Streaming en SDN



Nota. La Figura muestra la pérdida de paquetes del streming realizado en la implementación HDN

De igual manera que en la HDN, seguido de obtener los datos para de la pérdida de paquetes, la Tabla 28 nos muestra el resumen de la cantidad de paquetes perdidos por cada caso de prueba especificado que son el de Ping y el de Strem.

Asimismo, se procedió a calcular la media de la pérdida de paquetes, donde la prueba de ping nos indica que tenemos 1.9 paquetes perdidos y en la prueba de stream es de 0.05 paquetes perdidos; asimismo si calculamos la media general entre ambas pruebas tenemos que es **de 0.9750 paquetes perdidos.**

Tabla 28

Pérdida de paquetes en SDN

Prueba	Ping (Paquetes perdidos)	Stream (Paquetes perdidos)
1	2	1
2	2	0
3	1	0
4	2	0
5	3	0
6	2	0
7	1	0
8	3	0
9	1	0
10	2	0
11	2	0
12	1	0
13	3	0
14	2	0
15	2	0
16	1	0
17	2	0
18	3	0
19	1	0
20	3	0
21	2	0
22	1	0
23	2	0
24	2	0
25	1	0
26	3	0
27	1	1
28	2	0
29	1	0
30	2	0
31	3	0
32	3	0
33	2	0
34	2	0
35	1	0
36	2	0
37	1	0
38	2	0
39	2	0
40	2	0

Mediante los datos obtenidos se procedió a realizar la comparación de pérdida de paquetes en ambos escenarios de la Tabla 29, donde se observa que al enviar 10 paquetes ICMPv6 se tuvo una media de 1.9 y en la prueba de Stream 0.05 paquetes perdidos.

La Tabla 29 muestra la comparación de paquetes perdidos en ambas implementaciones.

Tabla 29

Comparación de pérdida de paquetes entre HDN y SDN

Prueba	HDN (Paquetes perdidos)		SDN (Paquetes perdidos)	
Ping	3	**231**	2	**1**
Stream	458		0	

Nota. La Tabla muestra el resumen de la cantidad de paquetes perdidos de las pruebas ejecutadas entre HDN vs SDN.

4.9.5. Nivel de Consumo de CPU

El consumo de CPU se observa cuando las topologías están en funcionamiento a través de la herramienta monitor del sistema; en ambas redes se evalúa el uso de CPU en los dos dispositivos físicos que han permitido la emulación; se ha considerado evaluar el nivel del uso de CPU en dos pruebas, los cuales se muestran en la Tabla 30, asimismo también se ha establecido realizar 40 veces cada prueba para mayor precisión.

Tabla 30

Distribución de pruebas para el consumo de CPU

Prueba	Detalle de prueba
Ping	Paquetes ICMPv6 h1-h2
Stream	Paquetes UDP h1-h2

Nota. La Tabla muestra las pruebas a ejecutar para el nivel de consumo de CPU.

Nivel de consumo de CPU Red Tradicional

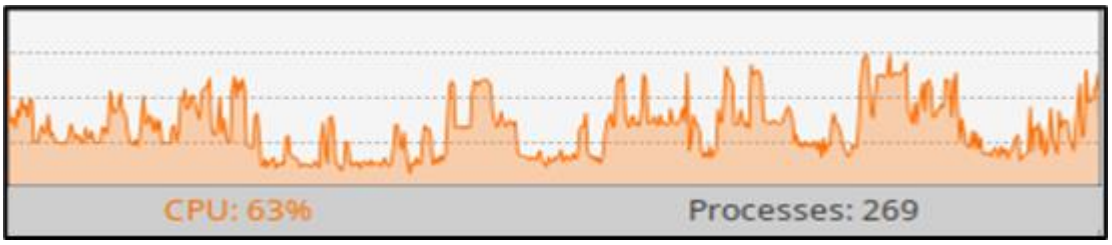
- ✓ **Prueba de Ping**

Para determinar el nivel de uso de CPU, se realiza mientras la topología se encuentra en funcionamiento, es decir se hace uso de CPU completamente desde la carga del sistema operativo, levantar el GNS3 y las VM, también todos los dispositivos de la red se encuentran encendidos y hay comunicación; aplicamos la prueba 1 realizando ping desde el host1 hacia host2; en la Figura 79 se muestra que el consumo de CPU que en el

equipo 1 es 63 % que es el efecto del funcionamiento en GNS3, este uso de CPU se dio al momento ejecutar ping, es decir la primera prueba de la Tabla 30.

Figura 79

Consumo de CPU en equipo 1 para prueba ping en HDN

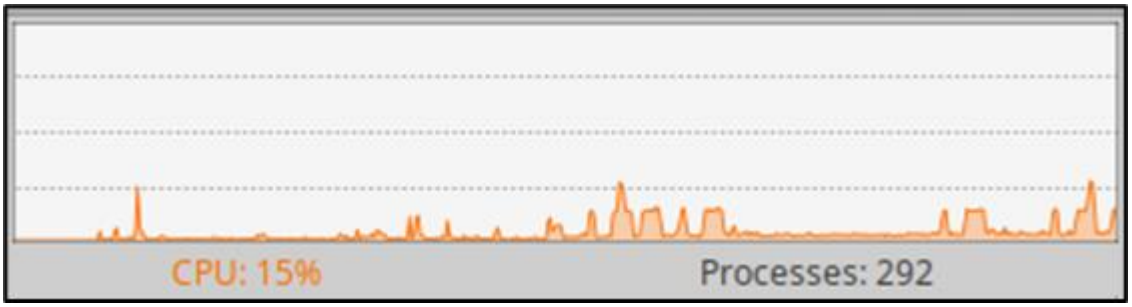


Nota. La Figura muestra el nivel consumo de CPU en % de la prueba de Ping en el equipo 1 de la implementación HDN.

En la Figura 80 se observa el uso de CPU del funcionamiento de GNS3 en el equipo 2 que es 15 %, en el mismo instante en que se está ejecutando la prueba 1 de la Tabla N° 37 desde el host1 hacia el host2.

Figura 80

Consumo de CPU en equipo 2 para prueba ping en HDN



Nota. La Figura muestra el nivel consumo de CPU en % de la prueba de Ping en el equipo 2 de la implementación HDN.

Seguidamente después de obtener estos datos, se elaboró la Tabla 31, la cual muestra el resumen del nivel de consumo de CPU que hace la prueba ping al ejecutarse en los 2 equipos; asimismo procedemos a calcular la media; en el equipo 1 esta prueba arroja que se tiene un consumo de CPU de **59.5250 %**, en el equipo 2 se tiene el consumo de CPU de **15.8750 %**, finalmente el promedio del nivel de consumo de CPU entre ambos equipos es de **37.7000 %.**

Tabla 31

Consumo de CPU prueba ping en HDN

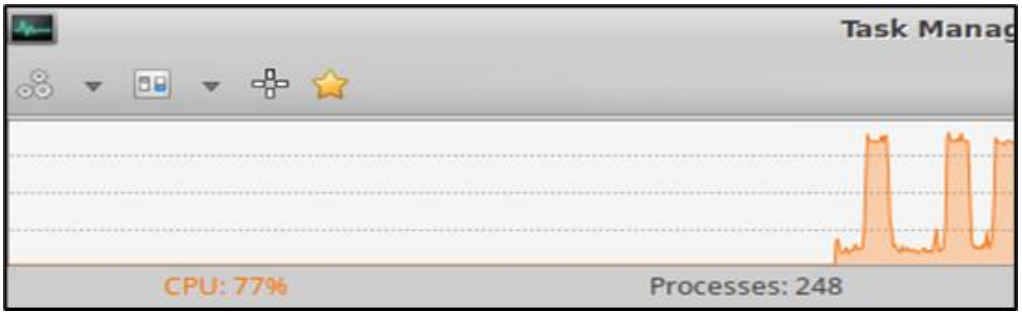
Prueba	Equipo 1 (%)	Equipo 2 (%)
1	63	15
2	56	16
3	58	14
4	58	15
5	67	15
6	59	15
7	56	15
8	61	17
9	63	17
10	62	14
11	56	17
12	57	16
13	60	16
14	63	16
15	60	17
16	57	15
17	57	17
18	59	14
19	59	16
20	62	15
21	58	15
22	59	15
23	59	16
24	57	15
25	57	14
26	57	17
27	61	17
28	59	14
29	61	16
30	62	18
31	60	18
32	57	17
33	60	14
34	62	18
35	61	15
36	63	17
37	56	16
38	57	17
39	63	17
40	59	17

- ✓ **Prueba de Stream**

De la misma manera, se ha realizado un Stream, donde host1 actúa como servidor y host2 como cliente; esta configuración se encuentra en el Anexo 5; el video tiene una duración de 60 segundos; para la toma de este dato, se ha realizado en todos los casos de prueba en el segundo 30 como estándar; la Figura 81 muestra que el consumo de CPU es 77 % en el equipo 1 y la Figura 82 nos indica que el consumo es de 19 % en el equipo 2.

Figura 81

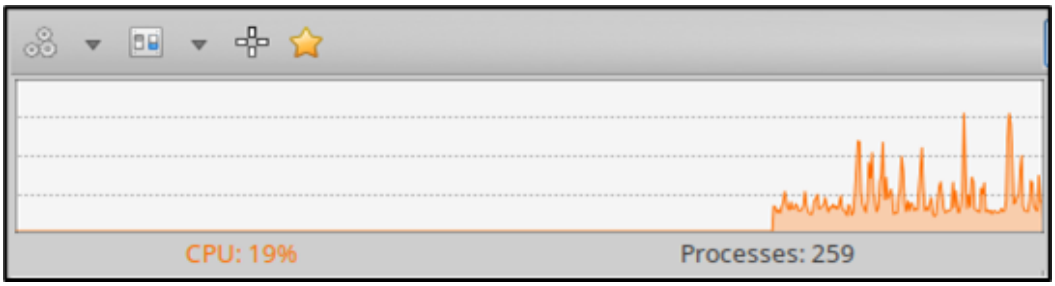
Consumo de CPU en equipo 1 para prueba Stream en HDN



Nota. La Figura muestra el nivel consumo de CPU en % de la prueba de Stream en el equipo 1 de la implementación HDN.

Figura 82

Consumo de CPU en equipo 2 para prueba Stream en HDN



Nota. La Figura muestra el nivel consumo de CPU en % de la prueba de Stream en el equipo 2 de la implementación HDN.

De igual manera que en la prueba anterior, después de obtener estos datos, se elaboró la Tabla 32, la cual muestra el resumen del nivel de consumo de CPU que hace la prueba Stream al ejecutarse en los 2 equipos; asimismo procedemos a calcular la media; en el equipo 1 esta prueba arroja que se tiene un consumo de CPU de 75.7000 %, en el equipo 2 se tiene el consumo de CPU de 19.6500 %, finalmente la media del nivel de consumo de CPU entre ambos equipos es de 47.6750 %.

Tabla 32

Consumo de CPU de prueba Stream en HDN

Prueba	Equipo 1 (%)	Equipo 2 (%)
1	77	19
2	76	21
3	72	22
4	74	18
5	79	20
6	75	19
7	79	19
8	79	16
9	76	22
10	77	22
11	76	18
12	72	18
13	76	19
14	74	20
15	72	19
16	79	18
17	78	18
18	73	22
19	73	18
20	74	18
21	78	19
22	75	22
23	79	21
24	77	19
25	76	18
26	77	19
27	79	20
28	76	21
29	72	22
30	74	18
31	72	19
32	76	19
33	74	20
34	76	21
35	75	22
36	73	19
37	78	22
38	77	19
39	75	18
40	78	22

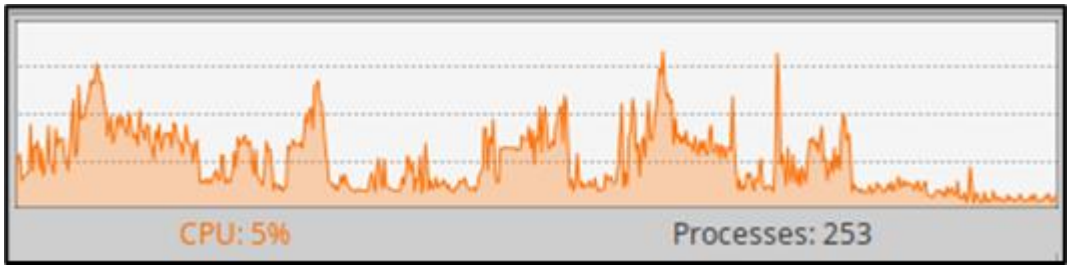
Nivel de consumo de CPU en SDN

- ✓ **Prueba de Ping**

De la misma forma, se considera el consumo de CPU desde que se levanta el sistema operativo, cargar todas las herramientas, las cuales para este caso es en el equipo 1 funcionan quagga y la ejecución del script mediante Mininet; se realizó la prueba 1 correspondiente de la Tabla 30; observamos en la Figura N° 83 que el consumo de CPU es de 5% en el equipo 1.

Figura 83

Consumo de CPU en equipo 1 para prueba Ping en SDN

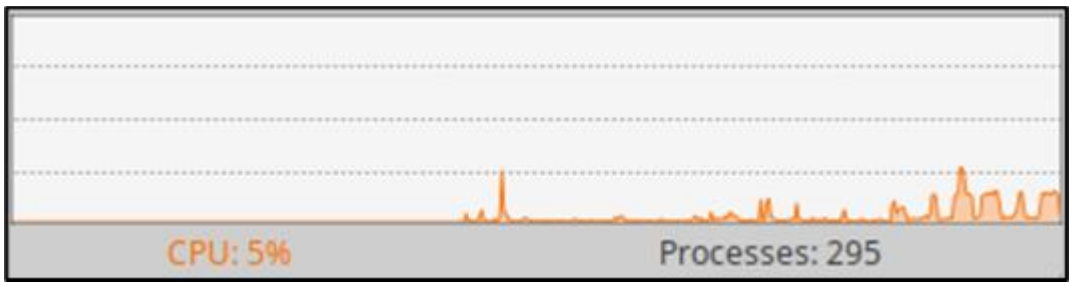


Nota. La Figura muestra el nivel consumo de CPU en % de la prueba de Ping en el equipo 1 de la implementación SDN.

En la Figura 84 observamos que el consumo de CPU en el equipo 2 también es de 5% con el funcionamiento de la topología y ejecutándose la prueba 1 de la Tabla N° 37, en este dispositivo físico reside el funcionamiento del controlador *Ryu* y es el que funciona al momento que se ejecuta la topología.

Figura 84

Consumo de CPU en equipo 2 para prueba Ping en SDN



Nota. La Figura muestra el nivel consumo de CPU en % de la prueba de Ping en el equipo 2 de la implementación SDN.

La Tabla 33 muestra el consumo de CPU para la prueba Ping realizada en ambos equipos en la SDN, la cual muestra el resumen del nivel de consumo de CPU que hace la prueba ping al ejecutarse en los 2 equipos; también procedemos a calcular la media; en el equipo 1 esta prueba arroja que se tiene un consumo de CPU de **5.3750 %**, en el equipo 2 se tiene el consumo de CPU de **4.9750 %**, finalmente la media del nivel de consumo de CPU entre ambos equipos es de **5.1750 %.**

Tabla 33

Consumo de CPU prueba Ping SDN

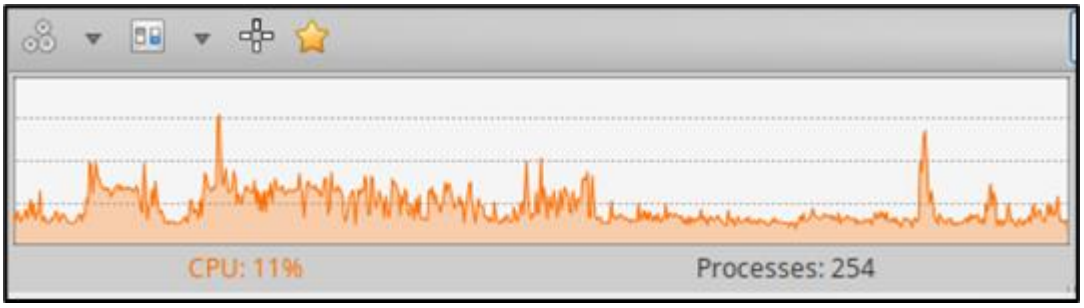
Prueba	Equipo 1 (%)	Equipo 2 (%)
1	5	5
2	5	5
3	4	4
4	5	6
5	6	5
6	5	5
7	4	6
8	4	4
9	5	5
10	6	5
11	6	6
12	5	6
13	4	4
14	5	4
15	5	4
16	6	5
17	6	4
18	5	5
19	4	5
20	7	6
21	6	6
22	5	4
23	4	5
24	6	5
25	6	5
26	7	6
27	4	4
28	6	6
29	7	6
30	7	5
31	5	6
32	4	4
33	4	4
34	5	5
35	5	4
36	7	4
37	7	5
38	6	5
39	6	5
40	6	6

- ✓ **Prueba Stream**

De la misma manera se procedió a realizar el caso de prueba del streaming del video de 60 segundos, h1 como servidor y h2 como cliente; la Figura 85 muestra que el consumo de CPU en el segundo 30 es de 11 % en el equipo 1; también se capturó el consumo de CPU en el equipo 2, el cual fue de 8 % tal como se observa en la Figura 86.

Figura 85

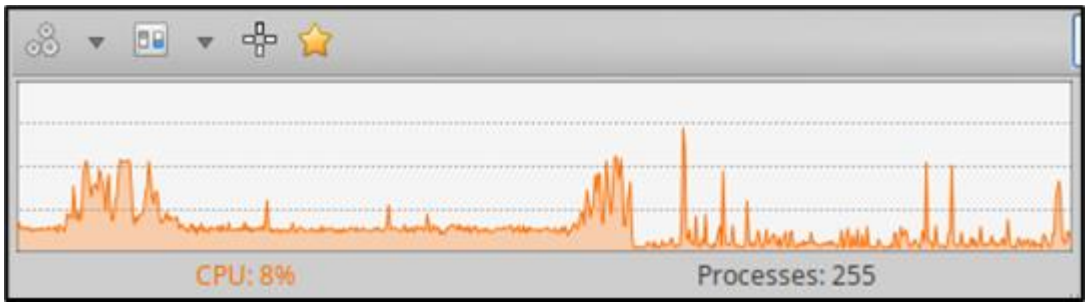
Consumo de CPU en equipo 1 para prueba Stream en SDN



Nota. La Figura muestra el nivel consumo de CPU en % de la prueba de Stream en el equipo 1 de la implementación SDN.

Figura 86

Consumo de CPU en equipo 2 para prueba Stream en SDN



Nota. La Figura muestra el nivel consumo de CPU en % de la prueba de Stream en el equipo 2 de la implementación SDN.

De forma similar que, en la prueba anterior, se elaboró la Tabla 34, la cual muestra el resumen del nivel de consumo de CPU que hace la prueba Stream al ejecutarse en los 2 equipos; procedemos a calcular la media; en el equipo 1 esta prueba arroja que se tiene un consumo de CPU de **9.5500 %**, en el equipo 2 se tiene el consumo de CPU de **8.3000 %**, finalmente la media general del nivel de consumo de CPU entre ambos equipos es de **8.9250 %.**

Tabla 34

Consumo de CPU de prueba Stream en SDN

Prueba	Equipo 1 (%)	Equipo 2 (%)
1	11	8
2	11	10
3	8	9
4	9	7
5	12	8
6	10	8
7	10	8
8	11	9
9	9	7
10	8	10
11	12	9
12	11	8
13	10	8
14	10	8
15	10	10
16	9	9
17	11	7
18	12	7
19	11	8
20	1	8
21	8	7
22	8	7
23	8	9
24	9	9
25	8	8
26	9	9
27	9	9
28	9	7
29	11	8
30	10	7
31	11	10
32	9	9
33	9	7
34	8	7
35	11	8
36	10	9
37	10	7
38	11	10
39	8	9
40	10	10

La Tabla 35 muestra el resumen del uso de CPU, de ambas implementaciones realizadas de la arquitectura de red avanzada REUNA en los 2 dispositivos físicos.

Tabla 35

Resumen de consumo CPU en HDN vs SDN

Prueba		**HDN (%)**		**SDN (%)**	
Ping	Equipo 1	59.525	**37.70**	5.375	**5.175**
	Equipo 2	15.875		4.975	
Stream	Equipó 1	75.7	**47.675**	9.55	**8.925**
	Equipo 2	19.65		8.3	

Nota. La Tabla muestra el resumen del nivel de consumo de CPU en HDN vs SDN.

4.9.6. Nivel de Consumo de Memoria

De la misma manera, para el consumo de Memoria (RAM) se evalúa estando en ejecución la topología mediante el monitor del sistema, y también se ha considerado realizar 40 veces las pruebas que se especifican en la Tabla 36 para mayor precisión.

Tabla 36

Distribución de pruebas para consumo de memoria

N°	**Prueba**
1	Ping h1-h2
4	Stream h1-h2

Nota. La Tabla muestra las pruebas a ejecutar para determinar el consumo de memoria.

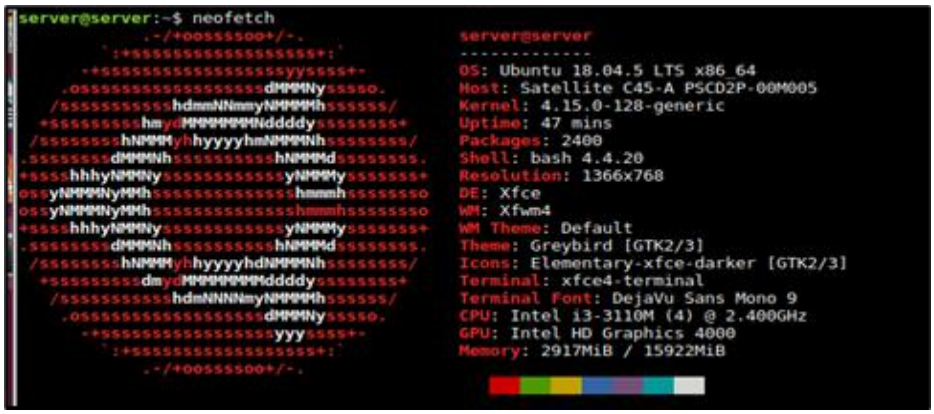
Nivel de uso de Memoria Red Tradicional

- ✓ **Prueba de ping**

El nivel de uso de memoria (RAM) al ejecutarse la topología mediante *GNS3* en el equipo 1 es de 2917 MB de memoria y a la vez ejecutándose la prueba 1 de la Tabla 36, esto lo observamos en la Figura 87; la Figura 88 evidencia que el uso de memoria en el equipo 2 que es de 4814 MB de RAM cuando se encuentra en funcionamiento la topología mediante GNS3 y ejecutándose dicha prueba.

Figura 87

Consumo de memoria en equipo 1 para prueba Ping en HDN



Nota. La Figura muestra el nivel consumo de memoria RAM en % de la prueba de Ping en el equipo 1 de la implementación HDN.

Figura 88

Consumo de memoria en equipo 2 para prueba Ping en HDN

Nota. La Figura muestra el nivel consumo de memoria RAM en % de la prueba de Ping en el equipo 2 de la implementación HDN.

En la Tabla 37 observamos el resumen del consumo de memoria de la prueba ping realizada para ambos equipos, donde después de obtener estos datos, se determina que el nivel de consumo de Memoria que hace la prueba ping al ejecutarse en el equipo 1 es de **2 917.5000 MB y** en el equipo 2 es de **4 813.7750 MB**, finalmente la media del nivel de consumo de Memoria entre ambos equipos es de **3 865.6375 MB de Memoria.**

Tabla 37

Consumo de memoria para prueba ping en HDN

Prueba	Equipo 1 (MB)	Equipo 2 (MB)
1	2 917	4 814
2	2 919	4 811
3	2 915	4 812
4	2 919	4 815
5	2 918	4 814
6	2 915	4 816
7	2 917	4 816
8	2 914	4 814
9	2 918	4 813
10	2 916	4 812
11	2 919	4 816
12	2 917	4 815
13	2 918	4 815
14	2 918	4 812
15	2 916	4 814
16	2 918	4 814
17	2 919	4 813
18	2 917	4 815
19	2 918	4 811
20	2 917	4 814
21	2 919	4 814
22	2 915	4 813
23	2 921	4 815
24	2 918	4 815
25	2 917	4 813
26	2 920	4 816
27	2 915	4 814
28	2 917	4 812
29	2 918	4 812
30	2 917	4 815
31	2 921	4 813
32	2 916	4 815
33	2 915	4 814
34	2 918	4 811
35	2 919	4 813
36	2 918	4 815
37	2 917	4 814
38	2 917	4 812
39	2 919	4 816
40	2 918	4 813

- ✓ **Prueba Stream**

La Figura 89 muestra que al realizar el Stream el equipo 1 en el segundo 30, consume 2931 MB de Memoria RAM y en la Figura 90 se observa que en el equipo 2 el consumo de Memoria es de 4841 MB de RAM.

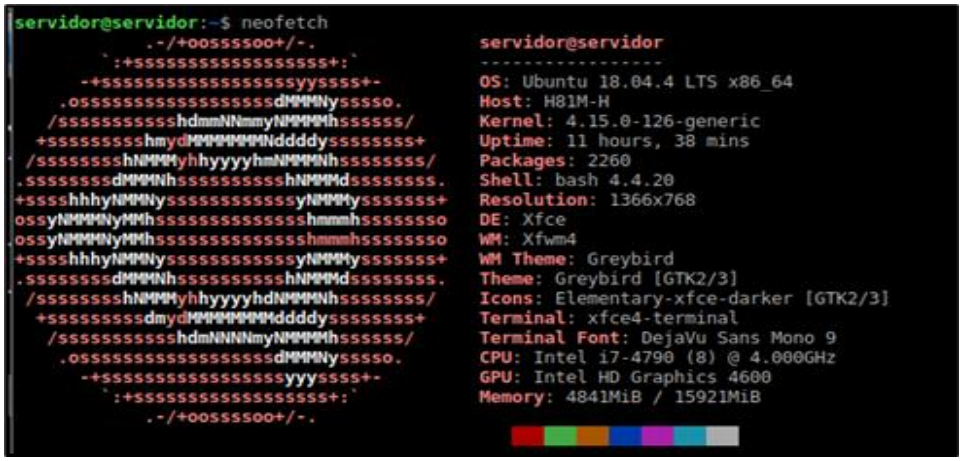
Figura 89

Consumo de memoria en equipo 1 para prueba Stream en HDN

Nota. La Figura muestra el nivel consumo de memoria RAM en % de la prueba de Stream en el equipo 2 de la implementación HDN.

Figura 90

Consumo de memoria en equipo 2 para prueba Stream en HDN



Nota. La Figura muestra el nivel consumo de memoria RAM en % de la prueba de Stream en el equipo 2 de la implementación HDN.

La Tabla 38, muestra el resumen del consumo de memoria para la prueba de Stream realizada.

Tabla 38

Consumo de memoria en prueba Stream con HDN

Prueba	Equipo 1 (MB)	Equipo 2 (MB)
1	2 917	4 814
2	2 919	4 811
3	2 915	4 812
4	2 919	4 815
5	2 918	4 814
6	2 915	4 816
7	2 917	4 816
8	2 914	4 814
9	2 918	4 813
10	2 916	4 812
11	2 919	4 816
12	2 917	4 815
13	2 918	4 815
14	2 918	4 812
15	2 916	4 814
16	2 918	4 814
17	2 919	4 813
18	2 917	4 815
19	2 918	4 811
20	2 917	4 814
21	2 919	4 814
22	2 915	4 813
23	2 921	4 815
24	2 918	4 815
25	2 917	4 813
26	2 920	4 816
27	2 915	4 814
28	2 917	4 812
29	2 918	4 812
30	2 917	4 815
31	2 921	4 813
32	2 916	4 815
33	2 915	4 814
34	2 918	4 811
35	2 919	4 813
36	2 918	4 815
37	2 917	4 814
38	2 917	4 812
39	2 919	4 816
40	2 918	4 813

De forma similar en base a la Tabla 38, procedemos a calcular el promedio; en el equipo 1 esta prueba arroja que se tiene un consumo de Memoria de **2 930.4500 MB**, en el equipo 2 se tiene el consumo de Memoria es de **4 843.4250 MB**, finalmente la media del nivel de consumo de CPU entre ambos equipos es de **3 886.9375 MB.**

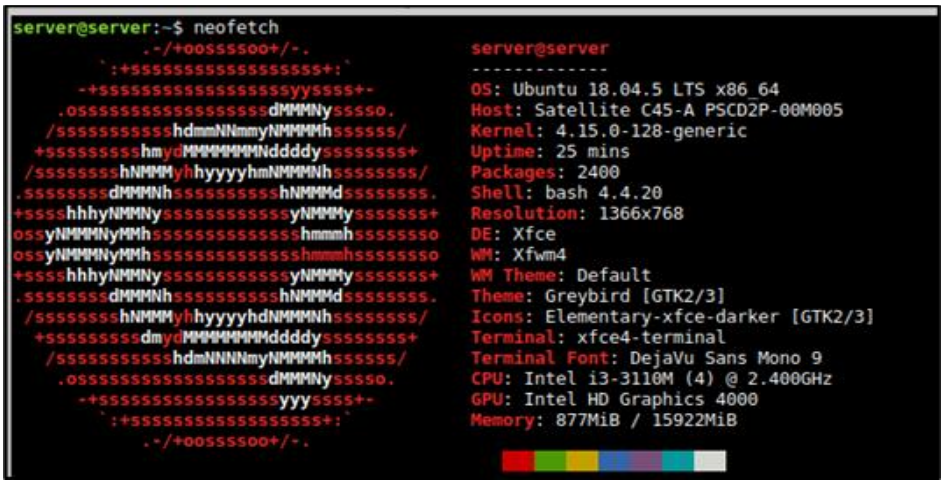
Nivel de uso de Memoria Red SDN

- ✓ **Prueba Ping**

Al ejecutarse el script y las funciones requeridas de quagga para el funcionamiento de la SDN en el equipo 1, la Figura 91 nos indica se ha usado 877 MB de memoria al ejecutar la prueba 1 de la Tabla N° 36.

Figura 91

Consumo de memoria en equipo 1 para prueba Ping en SDN

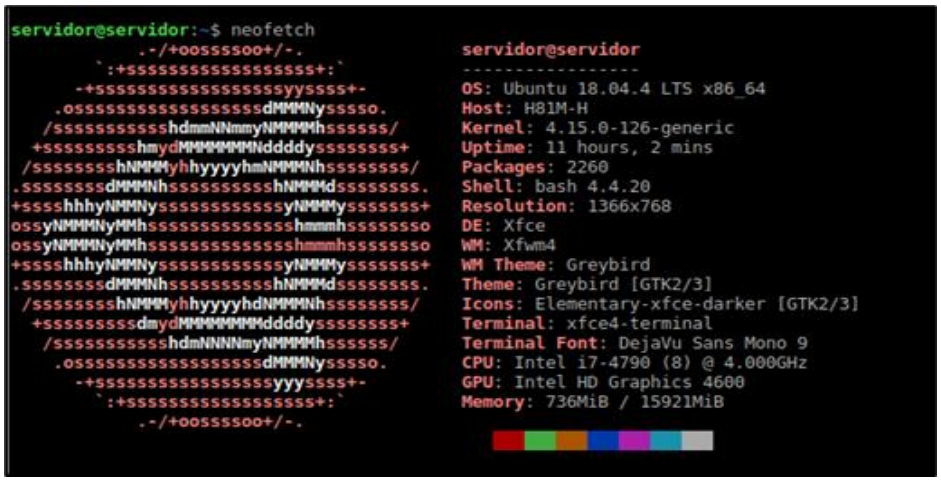


Nota. La Figura muestra el nivel consumo de memoria RAM en % de la prueba de Ping en el equipo 1 de la implementación SDN.

El uso de RAM en el dispositivo físico 2 es de **736 MB** al momento que *Ryu* se encuentra en funcionamiento y se ejecuta la prueba 1 de la Tabla 36; en la Figura 92 se puede apreciar esto.

Figura 92

Consumo de memoria en equipo 2 para prueba Ping en SDN



Nota. La Figura muestra el nivel consumo de memoria RAM en % de la prueba de Ping en el equipo 2 de la implementación SDN.

La Tabla 39 muestra el resumen del consumo de memoria para la prueba de ping en ambos equipos, en donde procedemos a calcular la media; en el equipo 1 esta prueba arroja que se tiene un consumo de Memoria de **876.4750 MB**, en el equipo 2 se tiene el consumo de Memoria es de **732.5000 MB**, finalmente la media del nivel de consumo de Memoria entre ambos equipos es de **804.4875 MB.**

Tabla 39

Consumo de memoria prueba Ping en SDN

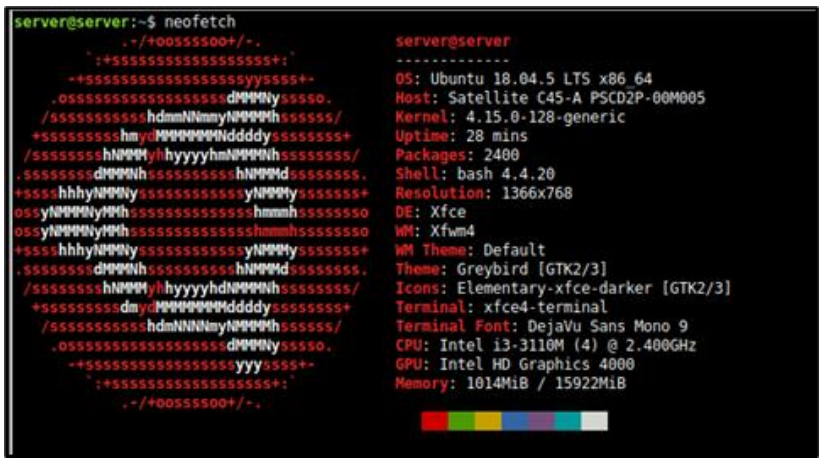
Prueba	Equipo 1 (MB)	Equipo 2 (MB)
1	877	736
2	876	732
3	878	735
4	877	726
5	875	733
6	875	727
7	876	735
8	876	734
9	877	734
10	877	728
11	878	734
12	875	729
13	876	730
14	875	736
15	878	731
16	877	730
17	875	734
18	877	735
19	877	735
20	877	736
21	876	733
22	875	731
23	879	733
24	876	735
25	874	727
26	878	732
27	877	736
28	879	735
29	879	726
30	876	729
31	877	734
32	876	735
33	874	730
34	875	736
35	875	728
36	876	736
37	877	736
38	878	731
39	874	732
40	879	735

- ✓ **Prueba Stream**

En la Figura 93 se observa el consumo de memoria que requiere la prueba en el equipo 1 que es de 1014 MB; la Figura 94 indica que el consumo de memoria RAM para ejecutarse la prueba en el equipo 2 es de 968 MB.

Figura 93

Consumo de memoria en equipo 1 para prueba Stream en SDN



Nota. La Figura muestra el nivel consumo de memoria RAM en % de la prueba de Stream en el equipo 1 de la implementación SDN.

Figura 94

Consumo de memoria en equipo 2 para prueba Stream en SDN

Nota. La Figura muestra el nivel consumo de memoria RAM en MB de la prueba de Stream en el equipo 2 de la implementación SDN.

La Tabla 40, muestra el resumen del consumo de memoria RAM que existe en ambos equipos de la prueba de stream realizada.

Tabla 40

Consumo de memoria en prueba Stream en SDN

Prueba	Equipo 1 (MB)	Equipo 2 (MB)
1	1 014	968
2	1 012	966
3	1 016	967
4	1 013	969
5	1 017	970
6	1 020	968
7	1 018	968
8	1 025	971
9	1 014	966
10	1 018	966
11	1 005	967
12	1 027	970
13	1 016	967
14	1 014	967
15	1 015	970
16	1 018	969
17	1 019	968
18	1 012	969
19	1 014	969
20	1 017	967
21	1 018	968
22	1 020	967
23	1 021	966
24	1 020	968
25	1 019	968
26	1 014	970
27	1 016	969
28	1 016	971
29	1 018	966
30	1 017	967
31	1 013	967
32	1 015	967
33	1 012	969
34	1 012	969
35	1 018	970
36	1 020	968
37	1 019	966
38	1 018	966
39	1 017	970
40	1 018	968

De forma similar en base a la Tabla 40, procedemos a calcular la media; en el equipo 1 esta prueba arroja que se tiene un consumo de Memoria de **10 16.6250 MB**, en el equipo 2 se tiene el consumo de Memoria es de **968.0500 MB**, finalmente el promedio del nivel de consumo de Memoria entre ambos equipos es de **992.3375 MB.**

En la Tabla 41 se muestra un resumen de la comparación entre una HDN y una SDN para el consumo de memoria

Tabla 41

Comparación de consumo de memoria entre HDN vs SDN

Prueba		**HDN (MB)**		**SDN (MB)**	
Ping	Equipo 1	2 917.5000	**3 865.6375**	876.4750	**804.4875**
	Equipo 2	4 813.7750		732.5000	
Stream	Equipó 1	2 930.4599	**3 886.9375**	1 016.6250	**992.3375**
	Equipo 2	4 843.4250		968.0500	

Nota. La Tabla muestra el resumen del nivel de consumo de memoria RAM de las pruebas ejecutadas en HDN vs SDN.

4.9.7. Nivel de Consumo de Ancho de Banda

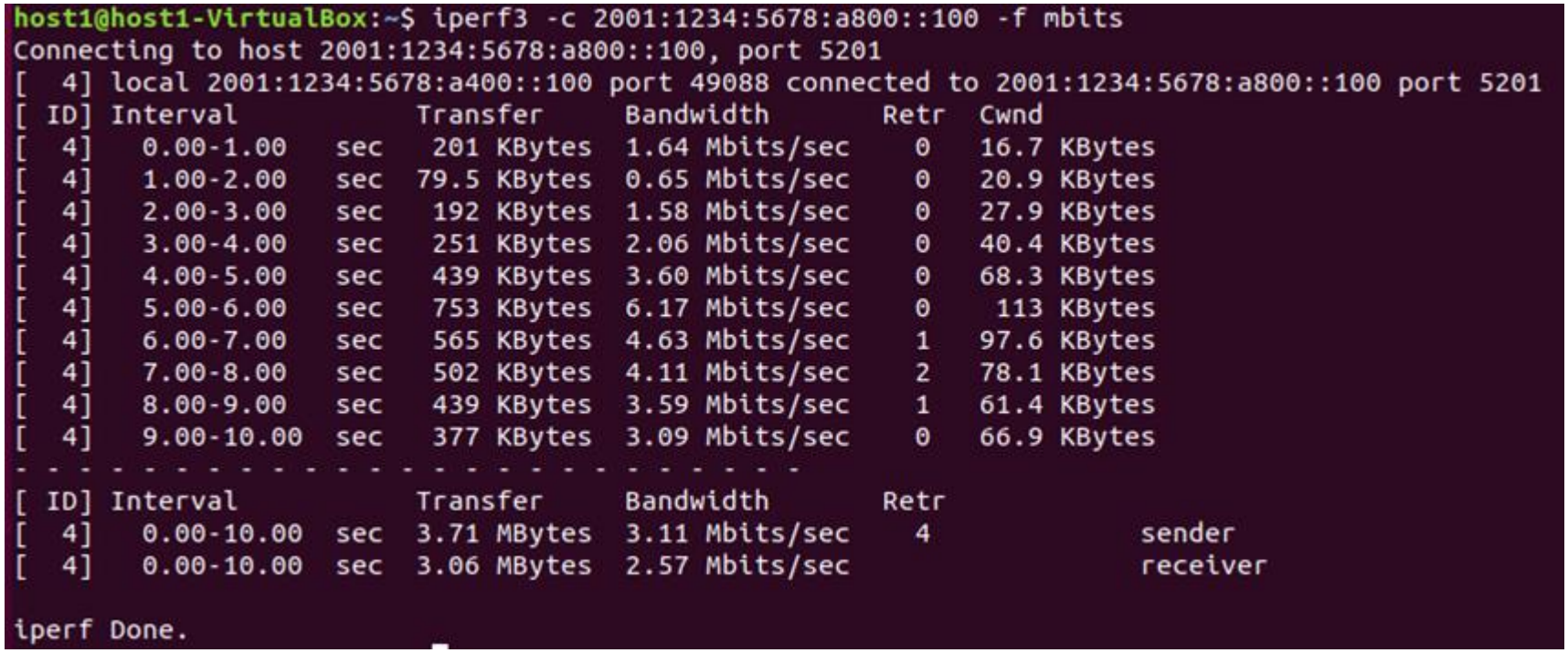
Para medir el ancho de banda en ambas emulaciones se ha utilizado la herramienta *iperf,* la cual envía paquetes UDP los cuales son los que se utilizan para video; por tanto, esta prueba es de utilidad para medir el uso de ancho de banda que se realiza *iperf* , asimismo se ha realizado 40 veces la prueba para obtener mayor precisión, donde host 1 actúa como servidor y host 2 como cliente.

Nivel de consumo de ancho de banda en red tradicional HDN

El host 1 actuará como cliente, para ello utilizamos la herramienta Iperf*;* para realizar la prueba, ingresamos el comando *iperf* seguido del parámetro -c que le indica que es cliente del servidor con IPv6 2001:1234:5678:a800::100 que es el *(host2)*, previamente a ejecutar el *client*, debemos hacer que el *host2* que actuará como *server* esté en modo escucha, para ello configuramos iperf seguido del parámetro -s que hace referencia a *server* y empezará a escuchar la comunicación; en la Figura 95 se observa la captura de la ejecución de *iperf* en el *host1*, esta herramienta nos indica que se hace el diagnostico de ancho de banda por 10 segundos que es lo que la herramienta nos proporciona por defecto, sin embargo podemos configurar este parámetro; se observa que el uso de ancho de banda es de 3.1100 Mbps.

Figura 95

Prueba 1 de Consumo de ancho de banda HDN



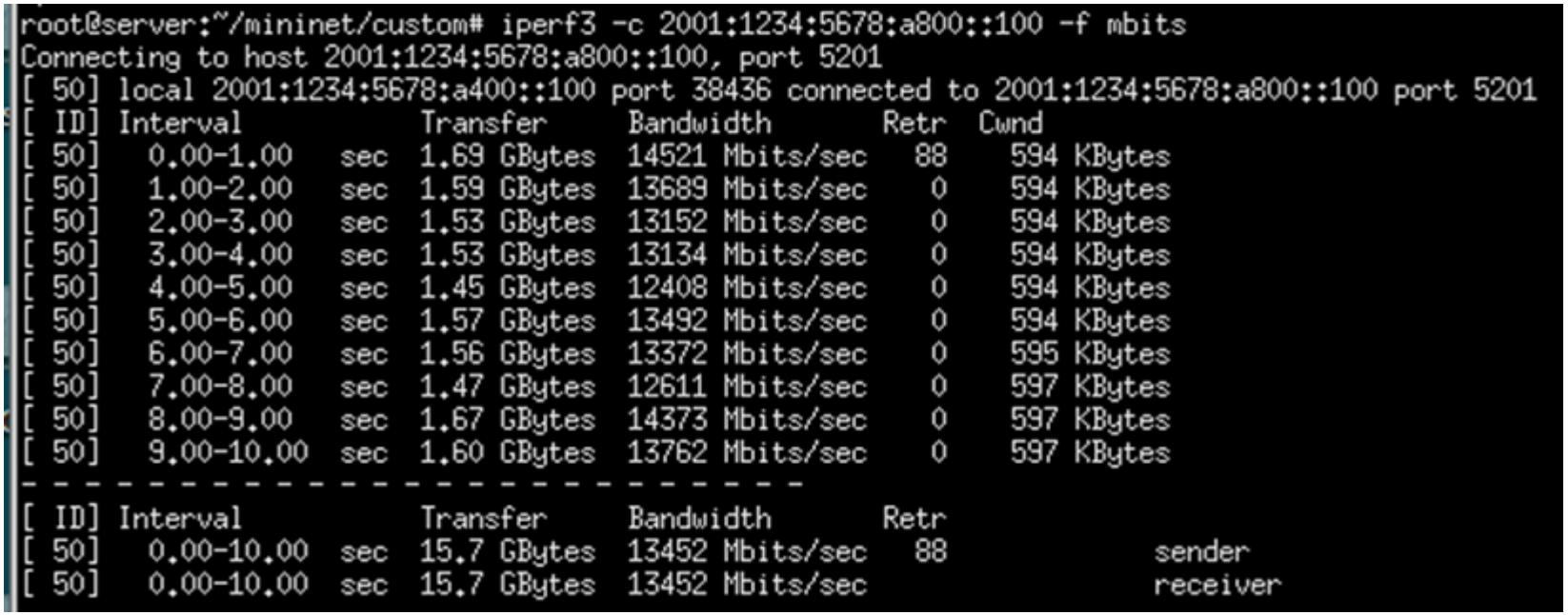
Nota. La Figura muestra el nivel consumo de Ancho de Banda en Mbps de la prueba de iperf en la implementación HDN, se observa el host1 actuando como cliente y en el otro extremo está host2 actuando como servidor.

Nivel de consumo ancho de banda red programable SDN

De la misma manera que en la red tradicional se ejecuta el *iperf* en h1 indicándole que es cliente y que será enviado hacia el h2 que actuará como *server*, en la Figura 96 se aprecia este proceso se ha ejecutado la prueba, podemos identificar que hace uso de 13 452 Mbps.

Figura 96

Consumo de ancho de banda en SDN



Nota. La Figura muestra el nivel consumo de Ancho de Banda en Mbps de la prueba de iperf en la implementación HDN, se observa el host1 actuando como cliente y en el otro extremo está host2 actuando como servidor.

Después de realizar las pruebas especificadas, en la Tabla N° 42 se observa el resumen de las pruebas realizas en ambas redes, se aprecia que la red tradicional usa **2.9568 Mbps** de ancho de banda, y la red SDN usa **13 977.9750 Mbps**, para esto debemos tener en cuenta que, para la transferir paquetes la red SDN usa mayor ancho de banda, esto significa que entregará con menores tiempos los paquetes que la red tradicional, por tanto el uso de ancho de banda que hace la red tradicional es menor que la red SDN, lo que significa que al usar menos ancho de banda afectará con mayor significancia en el rendimiento de la red de manera negativa, ya que puede requerir mucho más tiempo que la red SDN.

Tabla 42

Consumo de ancho de banda HDN vs SDN

Prueba	HDN (Mbps)	SDN (Mbps)
1	3.1100	13 452.0000
2	3.8100	13 792.0000
3	2.7000	14 308.0000
4	2.8100	14 653.0000
5	2.8600	14 113.0000
6	3.3200	14 375.0000
7	3.2600	14 137.0000
8	2.5500	14 029.0000
9	3.0100	13 804.0000
10	2.4500	14 120.0000
11	2.3400	13 919.0000
12	3.3900	14 023.0000
13	3.1600	13 732.0000
14	**2.900**	14 253.0000
15	2.4400	13 856.0000
16	3.2600	14 128.0000
17	2.7100	13 636.0000
18	3.2600	14 079.0000
19	2.7500	13 735.0000
20	2.6500	13 933.0000
21	2.9600	13 940.0000
22	3.5300	14 174.0000
23	3.6200	14 015.0000
24	2.4000	13 928.0000
25	2.3400	14 149.0000
26	3.4700	13 620.0000
27	2.4900	13 877.0000
28	3.3700	14 109.0000
29	2.4400	13 821.0000
30	2.2900	13 887.0000
31	3.0400	13 780.0000
32	3.4000	14 255.0000
33	2.3900	13 721.0000
34	2.4900	14 183.0000
35	2.8600	14 198.0000
36	3.5500	14 018.0000
37	2.8900	14 132.0000
38	3.3900	13 120.0000
39	3.4400	14 188.0000
40	3.1700	13 927.0000

V. Análisis de Resultados y Discusión

5.1. Análisis de Resultados

El estudio de la arquitectura de red avanzada REUNA en base a las implementaciones de manera tradicional (HDN) y de manera programable SDN al ser evaluadas y comparadas a través de las métricas obtenidas favorecen a una SDN en todas las dimensiones e indicadores de las variables.

La Hipótesis General y las Hipótesis Específicas responden a favor ya que los resultados permitieron cumplir con los objetivos de esta investigación que al evaluar el rendimiento de una arquitectura de red avanzada basada en el tipo de implementación, siendo el caso REUNA, tiene una relación inversamente proporcional con el rendimiento de la red, esto debido a la cantidad de instancias de control, ya que en la HDN se tiene dieciocho y en la SDN solo una; también es preciso afirmar que la implementación SDN tiene un rendimiento más eficiente que una implementación HDN; en el siguiente los siguientes cuadros se da a conocer los resultados que valida las hipótesis alineadas a los objetivos, para lo cual en el Anexo 6 se da una guía de implementación.

5.1.1. Arquitectura de Red

Para tener una mayor visualización se ha construido la Figura 97; el cual servirá para realizar este análisis del indicador cantidad de instancias de control en el siguiente cuadro.

Figura 97

Instancias de Control HDN vs SDN

Nota. La Figura muestra la cantidad de instancias de control entre HDN vs SDN.

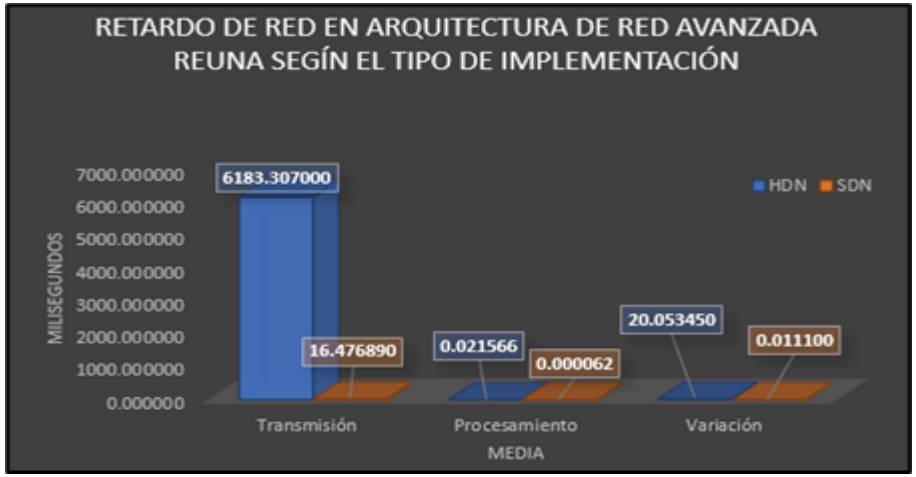
Indicador	Red Avanzada	
	HDN	**SDN**
Cantidad de Instancias de control	En base a la Figura 97, si realizamos dicho análisis, claramente afirmamos que las instancias de control en la arquitectura de Red Avanzada REUNA implementada de manera tradicional (HDN) están dadas por los 18 routers en referencia a cada ciudad, por tanto tenemos **dieciocho instancias de control**.	De la misma forma observando la Figura 97, determinamos que las instancias del plano de control en la arquitectura de Red Avanzada REUNA implementada de manera programable (SDN) está dada por un único controlador Ryu; lo que indica que solo tenemos **una instancia de control**.

5.1.2. Retardo de Red

Para tener una mayor visualización se ha construido la Figura 98; el cual servirá para realizar este análisis de los indicadores de retardo.

Figura 98

Retardo de red HDN vs SDN



Nota. LA Figura muestra los tiempos de los tres indicadores de la dimensión retardo en ambas implementaciones (HDN vs SDN).

Indicador	Red Avanzada	
	HDN	**SDN**
Retardo de Transmisión	En base a los datos obtenidos que se resumen en la Tabla 16, es preciso indicar que la prueba Ping de un paquete de 56 Bytes genera un retardo de transmisión de 231.9987 milisegundos y con un paquete de 1500 Bytes es de 237.6845 milisegundos; asimismo en la prueba traceroute el paquete de 80 Bytes ha generado un retardo de 171.6925 milisegundos y en el paquete de 1500 Bytes es de 180.9661 milisegundos; por otra parte en la prueba de transferencia de archivos al ser un .pdf d 1.4 MB tiene un retardo de 4 337.0000 milisegundos, sin embargo en el envío del archivo .mp4 de 14 MB el retardo es de 31 940.0000 milisegundos; si realizamos el análisis nos damos cuenta en cada prueba realizada enviamos 2 paquetes de distintos tamaños, estos retardos muestra lógicamente que los paquetes de mayor tamaño generan mayor retardo de transmisión. Asimismo, con los datos mencionados podemos afirmar mediante la Tabla 16 que en la prueba Ping se tiene un retardo de 234.8416 milisegundos, en la prueba de traceroute es de 176.3293 milisegundos y en la transferencia de archivos es de 18 138.7500 milisegundos. Finalmente podemos afirmar mediante la Figura 98 que la media general del retardo de transmisión es de **6 183.30700 milisegundos**, esto lo justificará finalmente el ancho de banda empleado y la forma en cómo responde esta implementación tradicional.	Evidentemente de la misma forma que en la HDN observamos la Tabla 16, para indica que la prueba Ping de un paquete de 56 Bytes genera un retardo de transmisión de 0.2122 milisegundos y con un paquete de 1500 Bytes es de 0.2756 milisegundos; asimismo en la prueba traceroute el paquete de 80 Bytes ha generado un retardo de 0.0608 milisegundos y en el paquete de 1500 Bytes es de 0.0627milisegundos; por otra parte en la prueba de transferencia de archivos al ser un .pdf d 1.4 MB tiene un retardo de 11.0000 milisegundos, sin embargo en el envío del archivo .mp4 de 14 MB el retardo es de 87.2500 milisegundos; analizando esto determinamos que en cada prueba realizada enviamos 2 paquetes de distintos tamaños, estos retardos muestra lógicamente que los paquetes de mayor tamaño generan mayor retardo de transmisión. Coincidiendo con los datos calculados se precisa que en la prueba de Ping se tiene un retardo de 0.2439 milisegundos, en la prueba de traceroute es de 0.0618 milisegundos y en la transferencia de archivos es de 49.1250 milisegundos. Es preciso indicar que la a través de la Figura 98 la media general del retardo de transmisión en esta implementación es de **16.47689 milisegundos**, lo cual será confirmado mediante el ancho de banda de la red y la manera en cómo responde esta implementación programada.
	Para este indicador apreciamos la Tabla 22, donde precisamos que la prueba Ping en el router Antofagasta tiene un retardo de procesamiento de 0.006417	De la misma forma que en la HDN para este indicador nos enfocamos en los datos de la Tabla 22, mediante la cual afirmamos que la prueba Ping en el router Antofagasta tiene

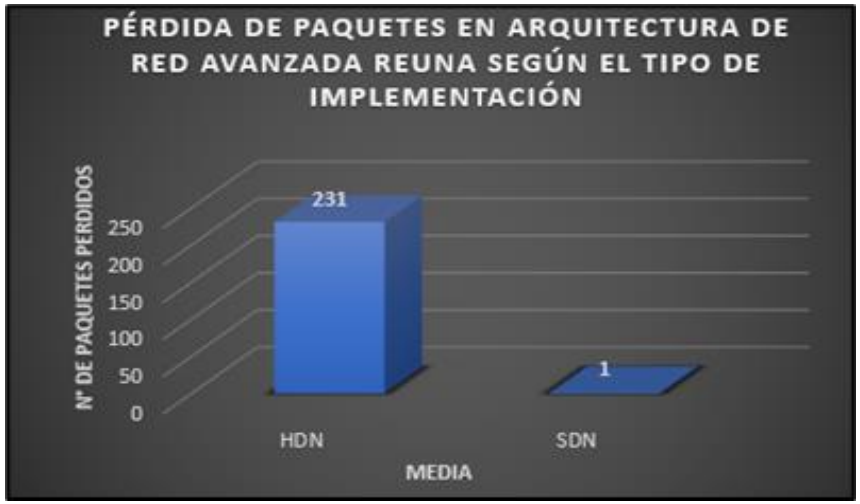
Retardo de Procesamiento	milisegundos y en el router Santiago el retardo es de 0.100845 milisegundos; de la misma manera tenemos que en la prueba traceroute en el router Antofagasta se ha calculado un retardo de 0.059366 milisegundos y en el router Santiago es de 0.101967 milisegundos. De forma precisa podemos indicar que la prueba Ping ha generado un retardo de procesamiento de 0.008251 milisegundos; asimismo en la prueba Traceroute el retardo de transmisión es de 0.034782 milisegundos. Por tanto, se ha determinado mediante la Figura 98 que la media general del retardo de procesamiento es de **0.02157 milisegundos**. Esto lo justifica la manera en que cada router demora en procesar toda la información del encabezado del paquete y enviarlo mediante su tabla de enrutamiento.	un retardo de procesamiento de 0.000012 milisegundos y en el router Santiago el retardo es de 0.000009 milisegundos; de la misma manera tenemos que en la prueba traceroute en el router Antofagasta se ha calculado un retardo de 0.000223 milisegundos y en el router Santiago es de 0.00000252 milisegundos. Con lo cual podemos indicar que la prueba Ping ha generado un retardo de procesamiento de 0.000010 milisegundos; asimismo en la prueba Traceroute el retardo de transmisión es de 0.000113 milisegundos. Preciso es indicar mediante la Figura 98 que la media general del retardo de procesamiento en esta implementación es de **0.00006 milisegundos**, que lo justifica cuanto demora en procesarse y enviarse el paquete, el cual en esta implementación lo realiza solo el controlador Ryu mediante los archivos de cada router que utiliza por medio del script y quagga.
Variación de retardo	Para realizar el análisis de este indicador debemos observar la Tabla 25, donde podemos indicar que la Variación de retardo durante un periodo de diez segundos en la prueba de enviar un paquete de 100 MB es 21.1718250 milisegundos y en la prueba de 5000 MB tenemos que las variaciones de retardo son de 18.9351 milisegundos. Precisamos que la media de la variación de retardo calculada es de **20.05345 milisegundos** mediante la Figura 98. Realizando el debido análisis este proceso es determinado en base a que el paquete debe pasar desde el equipo 1 donde está el host1 hasta el equipo 2 donde está el host 2 a través de la ruta que existe entre los routers.	Similarmente al análisis realizado en la HDN, apreciaremos los datos de la Tabla 25, mediante los cuales indicamos que la Variación de retardo durante un periodo de diez segundos en la prueba de enviar un paquete de 100 MB es 0.0201250 milisegundos y en la prueba de 5000 MB tenemos que las variaciones de retardo son de 0.0020750 milisegundos. Es válido afirmar que de acuerdo con los datos precisados que media de la variación de retardo es de **0.01110 milisegundos** a través de la Figura 98**,** esto es en base a que el paquete no ha pasado desde el equipo 1 hacia el equipo 2, sino que pasa por la ruta a través de cada router que es comandado por Ryu, pero tanto host1 como host2 se encuentran dentro del equipo1.

5.1.3. Pérdida de Paquetes

Para tener una mayor visualización se ha construido la Figura 99; el cual servirá para realizar este análisis del indicador cantidad paquetes perdidos en el siguiente cuadro.

Figura 99

Cantidad de Pérdida de paquetes HDN vs SDN



Nota. La Figura muestra la cantidad de paquetes perdidos implementaciones (HDN vs SDN).

Indicador	**Red avanzada**	
	HDN	**SDN**
Cantidad de paquetes perdidos	En relación con los datos obtenidos que observamos en la Tabla 29, indicamos que en la prueba Ping se tiene una media de 3.425 paquetes perdidos; asimismo en el Streaming realizado se tiene una media de 458.35 paquetes perdidos. Es correcto precisar a través de la Figura 99 que se tiene una media de **231 paquetes perdidos** en esta	Sincronizadamente con el análisis realizado en la HDN verificamos la Tabla 29, mediante la cual podemos indicar que en la prueba Ping se tiene una media de 1.9 paquetes perdidos, y en la prueba de Streaming se tiene una media de 0.05 paquetes perdidos. Es preciso afirmar mediante la Figura 99, que se tiene una media de **1 paquete perdido** en esta

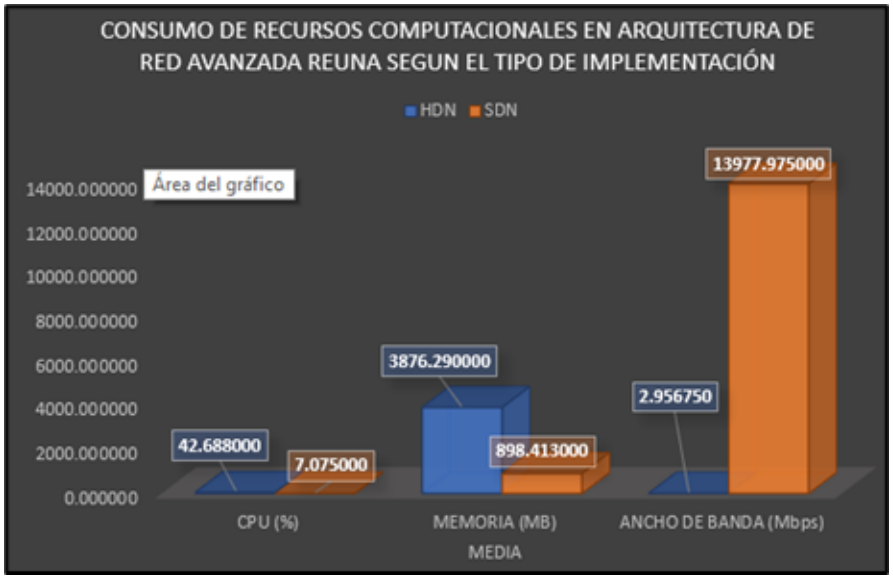
	implementación, esto se explica realizando el análisis de que todos los paquetes requieren pasar desde el host1 que se encuentra en e l equipo1 hacia el host2 que se encuentra en el host2, claramente mediante la ruta que construyen los routers que finalmente seguirán los paquetes.	implementación; realizando el análisis de esto, lo que está ocurriendo es que el paquete atraviesa toda la ruta también igual que en la HDN, pero finalmente se está quedando dentro del equipo1, lo cual justifica la minoría de paquetes perdidos dentro de la red programada, también otro factor es el consumo de ancho que se hace, ya que si en esta implementación se utiliza mayor ancho de banda, lo más probable es que los paquetes no se pierdan.

5.1.4. Consumo de Recursos computacionales

Para tener una mayor visualización se ha construido la Figura 100; el cual servirá para realizar dicho análisis.

Figura 100

Nivel consumo de recursos computacionales HDN vs SDN



Nota. La Figura muestra el consumo de recursos computacionales en ambas implementaciones (HDN vs SDN) según los tres indicadores.

Indicador	Red Avanzada	
	Tradicional	**SDN**
Nivel de consumo de CPU	El nivel de consumo de CPU se determina a través de la Tabla 35, donde al realizar este análisis según los datos obtenidos después que la topología se encuentre en funcionamiento, en la prueba Ping el equipo 1 tiene un consumo de CPU de 59.5250 % y el equipo 2, tiene un consumo de CPU de 15.8750 %; asimismo en la prueba de Streaming en el equipo 1 el consumo de 75.7000 % y en el equipo 2 es de 19.6500 %; dicho análisis implica decir que este consumo es de manera completa, incluye el sistema operativo, las herramientas instaladas y la prueba ejecutada; si bien es cierto el equipo 1 tiene mayor consumo de CPU a pesar de tener solo 6 routers, comparados con los 12 que se ejecutan en el equipo 2, pero esto lo justifica el procesador de cada equipo que son de 2.4 y 3.6 Ghz respectivamente. Con lo cual precisamos que en la prueba Ping se tiene un nivel de consumo de CPU de 37.7000 % y en la prueba de Streaming es de 47.6750 %. Asimismo, mediante la Figura 100 afirmamos que esta implementación tiene una media de consumo de CPU de **42.69 %.**	De manera similar que en el análisis de la HDN verificamos la Tabla 34, donde según los datos obtenidos después que la topología se encuentre en funcionamiento, en la prueba Ping el equipo 1 tiene un consumo de CPU de 5.3750 % y el equipo 2, tiene un consumo de CPU de 4.9750 %; asimismo en la prueba de Streaming en el equipo 1 el consumo de 9.5500 % y en el equipo 2 es de 8.3000 %; similarmente en este análisis se consideran el sistema operativo, las herramientas instaladas y las pruebas ejecutadas; el consumo de CPU en esta implementación es mínimo y eso se debe a que en el equipo 1 se utilizan los scripts y es programado, lo que finalmente no utiliza un IOS como en HDN, sino que utiliza el sistema operativo y crea interfaces virtuales, y en el equipo 2 solo corre ryu que finalmente solo es un script en Python. Mediante lo indicado afirmamos que en la prueba Ping se tiene un nivel de consumo de CPU de 5.1750 % y en la prueba de Streaming es de 8.9250 %. A través de la Figura 100 afirmamos que esta implementación tiene una media de consumo de CPU de **7.08 %**.
Nivel consumo de Memoria	Es requerido verificar la Tabla 41, mediante la cual se precisa que en la prueba Ping el equipo 1 ha utilizado total 2 917.5000 MB de memoria y en el equipo 2 el consumo de memoria es de 4 813.7750 MB; asimismo en la prueba de Streaming el consumo de memoria total en el equipo 1 es de 2 930.4500 MB y en el equipo 2 es de 4 843.4250 MB, dicho análisis implica precisar que el consumo de RAM incluye levantar el sistema operativo, funcionamiento de las	Acorde con el análisis realizado en la HDN debemos verificar la Tabla 41, mediante la cual se precisa que en la prueba Ping el equipo 1 ha utilizado 876.475 MB de memoria y en el equipo 2 el consumo de memoria es de 732.5000 MB; asimismo en la prueba de Streaming el consumo de memoria en el equipo 1 es de 1 016.6250 MB y en el equipo 2 es de 968.0500 MB, en este análisis debemos considerar que el consumo de memoria incluye funcionamiento de sistema

	herramientas y la ejecución de dichas pruebas, extendiendo este análisis precisamos que el equipo 1 consume menos memoria y esto es lógico ya que ambos equipos tienen 16 GB de RAM, sin embargo en el equipo 1 solo hay 6 IOS router corriendo y comparados en el equipo 2 que hay 12 IOS router funcionando y como es de esperar el . En definitiva, la prueba Ping realizada tiene un consumo de memoria de 3 865.6375 MB y la prueba de Streaming ha consumido 3 886.9375 MB de memoria. Mediante la Figura 100 afirmamos que esta implementación tiene una media de consumo de Memoria de **3 876.29 MegaBytes**.	operativo y las herramientas, además de la prueba ejecutada; entre equipo 1 y equipo 2 no hay demasiada diferencia de consumo, sin embargo precisamos que en el equipo 1 es mayor debido a que se ejecuta el script en Mininet y las funciones de quagga y en el equipo 2 el únicamente el script de Ryu, y como es de esperar la prueba de Streaming consume mayor memoria que la prueba Ping. Podemos afirmar que la prueba Ping realizada tiene un consumo de memoria de 804.4875 MB y la prueba de Streaming ha consumido 992.3375 MB de memoria. Asimismo, la Figura 100 confirma que esta implementación tiene una media de consumo de Memoria de **898.41 MegaBytes.**
Nivel de consumo de Ancho de Banda	Para este indicador observamos la Tabla 42, a través de la cual podemos indicar que mediante la prueba ejecutada el nivel de consumo de ancho de banda en esta implementación es de 2.9568 Mbps, la cual es la velocidad promedio que ha consumido esta implementación HDN durante las pruebas anteriores realizadas. Esta prueba que ha sido realizada entre el host 1 y host 2 mide la capacidad de consumo de ancho de banda en la red, la cual finalmente se ejecuta con las interfaces de red de los 2 equipos físicos que se conectan mediante el enlace cloud, lo que ocurre es que el paquete para pasar desde el host1 hacia el host2 debe trasladarse desde el equipo 1 donde se encuentra la VM host1 hasta el equipo 2 donde está la VM2, debidamente a través de la ruta. Mediante la Figura 100 precisamos que esta implementación tiene una media de consumo de Ancho de Banda de **2.96 Mbps**.	De la misma forma en que sea ha realizado el análisis en la HDN, para este indicador observamos la Tabla 42, mediante la cual podemos indicar que la prueba ejecutada indica que el consumo de ancho de banda en esta implementación es de 13 977.9750 Mbps, la cual es la velocidad promedio que ha consumido esta implementación HDN durante las pruebas anteriores realizadas, esto confirma los resultados de los datos de todas estas pruebas ejecutadas, también podemos indicar que esto se debe a que esta implementación al tener un solo controlador en el equipo 2 y todos los archivos de enrutamiento de 18 routers están en el equipo 1, entonces el paquete en realidad no se traslada del equipo 1 al equipo 2, sino que mediante el controlador Ryu hace que este se envié desde h1 hasta h2 mediante la ruta, pero este paquete se queda dentro del equipo 1, es decir no pasa al equipo 2, ya que los dos host se encuentran dentro del equipo 1. La Figura 100 precisa que esta implementación tiene una media de consumo de Ancho de Banda de **13 977.98 Mbps**.

En base a los resultados obtenidos se valida la hipótesis “La relación que existe entre el rendimiento y el tipo de implementación de una arquitectura de red avanzada es inversamente proporcional”, ya que al realizar las implementaciones; estas generan distinto rendimiento de la red, esto evidencia que el rendimiento obtenido se debe al tipo de implementación y es inversamente proporcional, ya que una programable es más eficiente que una tradicional; todo esto limitado por ser una emulación; además claramente se nota que es debido a que la HDN carga IOS de los router la cual causa mayor efecto en el rendimiento de la red al ser 18 instancias de control, sin embargo en la SDN es mucho más liviano y el rendimiento es más eficiente, esto se debe a que solo existe un controlador y la forma como actúa es mediante un script; por tanto se acepta la Hipótesis planteada, debido a que el rendimiento se ve relacionado inversamente proporcional con el tipo de implementación, es decir a más instancias de control menor rendimiento en la red; con esto nos referimos que tendrá más tiempo en el retardo, existirá más pérdida de paquetes y más consumo de recursos computacionales.

5.2. Discusión

Antecedente: “Análisis de rendimiento en redes IPv6”

Discusión: Este antecedentes de investigación implementa un laboratorio IPv6 con OSPFv3 para dar un análisis del rendimiento de la red considerando las métricas como jitter, cantidad de paquetes y pérdidas, los tiempos de transferencia que finalmente determina la calidad de servicio en el escenario; pero el enfoque de este antecedente es netamente para redes tradicionales; lo que en comparación con este trabajo de investigación sería la implementación de una arquitectura de red avanzada de manera tradicional (HDN), sin embargo la diferencia con el presente trabajo de investigación es que además de que también se implementa dicha arquitectura de red avanzada de manera tradicional con el fin de determinar su rendimiento, se ha realizado otra implementación que es de manera programable (SDN) con el fin también de evaluar su rendimiento; pero finalmente en este trabajo no solo nos enfocamos en determinar el rendimiento sino que damos una solución a la implementación adecuada basada en el rendimiento más eficiente; el cual en base a los resultados y análisis es una de manera programable (SDN); el antecedente también menciona que el retardo aumenta considerablemente; en una prueba que realizó de paquetes de 50000 bytes, el retardo en momentos de congestión fue superior a 3800 milisegundos, en nuestro trabajo se realizó más pruebas que finalmente dan una media de 6183.70 milisegundos.

Antecedente: "Plataforma de pruebas para evaluar el desempeño de las redes definidas por software basadas en el protocolo OpenFlow"

Discusión: Este antecedente se enfoca en determinar el funcionamiento de las Redes Definidas por Software, pero solo a nivel del protocolo OpenFlow, el cual es el protocolo que encontramos en la capa de infraestructura, el antecedente manifiesta el comportamiento de la Red Definida por Software a nivel de IPv4 según los controladores que va poniendo a prueba como POX, RYU, ONOS, FLOODLIGHT y OPENDAYLIGHT; en nuestra investigación se abarca el protocolo de enrutamientos dinámico OSPFv3 que damos utilidad en dos escenarios de una arquitectura de red avanzada los cuales una se implementa de manera tradicional (HDN) y la otra de manera programable (SDN), inicialmente se ha podido configurar también los controladores OPENDAYLIGHT ONOS Y RYU, que finalmente solo es el controlador RYU el que se ha establecido utilizar debido a la compatibilidad, hemos coincidido con 3 controladores los cuales según el autor funcionan mejor bajo Ubuntu 14 y usa el protocolo OpenFlow 1.0, coincide esta investigación ya que inicialmente se tuvo problemas usando Ubuntu 18 y 16, lo que hizo que optáramos por Ubuntu 14, sin embargo si se puede hacer funcionar en Ubuntu 18, además para IPv6 en nuestra investigación usamos OpenFlow 1.3, pero el enfoque de nuestra investigación no se basa en ver el comportamiento de los controladores, sino evaluar el rendimiento de la red considerando nuestras dimensiones que son el retardo, pérdida de paquetes y consumo de recursos computacionales.

Antecedente: "Diseño de una Red Industrial"

Discusión: En este antecedente de investigación se tiene como objetivo diseñar una red basada en requerimiento de producción, en el cual se plantea dos opciones que son la Red tradicional y la Red Definida por Software, si bien es cierto el antecedente va enfocado a un entorno más real que es de producción proponer un diseño, pero su limitación es que son basadas en IPv4; en la presente investigación se considera implementar la arquitectura de red avanzada REUNA mediante una emulación para ambas tipos de implementación de red, es decir de una manera tradicional (HDN) y la otra de manera programable (SDN), pero dentro de nuestro objetivo consideramos evaluar el rendimiento de la red para elaborar una guía eficiente de implementación de una arquitectura de red avanzada, claramente teniendo en cuenta el rendimiento de red.

Antecedente: "Evaluación de desempeño y configuraciones de las SDN mediante la simulación.

Discusión: Este antecedente de investigación se enfoca netamente a las redes Definidas por Software, donde se evalúa métricas que se utilizan en las redes tradicionales como perdidas de paquetes, latencia, variaciones y tiempos, también es exclusivo ya que solo trabaja bajo el protocolo *OPenFlow* a nivel de *Switches* y sugerir el desempeño del controlador *OpenDayLight*, en la presente investigación tenemos un enfoque más extenso donde se trabaja con dos escenarios diferentes para una arquitectura de red avanzada, donde independientemente del controlador, se evalúa el rendimiento de la red coincidiendo con algunos de los indicadores del antecedente, que son la pérdida de paquetes, tiempos y variaciones para lo cual se ha considerado dentro de retardos y pérdida de paquetes en ambas implementaciones, las diferencias engloban en que este antecedente trabaja bajo IPv4 y a nivel de conmutación o capa 2, sin embargo en nuestra investigación ya trabajamos a nivel de enrutamientos que sería capa 3.

Antecedente: IPv6 Connectivity and Management Emulation for REUNA, the Chilean Advanced Network".

Discusión: Este antecedente de investigación realiza una emulación de la Red Avanzada de Chile (REUNA), donde utiliza el protocolo OSPFv3 para lograr conectividad, y dentro de sus consideraciones de gestión evalúa el consumo de recurso computacionales; si bien es cierto este antecedente realiza la emulación de REUNA, en la presente investigación a diferencia del antecedente además de los recursos computacionales se ha considerado comparar mediante el tipo de red, retardo de red y pérdida de paquetes; además el enfoque de esta investigación es orientado a evaluar el rendimiento para dar una solución de implementación eficiente de cualquier arquitectura de red avanzada; el antecedente menciona que 45.0 %, en nuestro caso tenemos una media de 42.688 % que se aproximan con similitud, sin embargo en la implementación SDN vemos que el consumo de CPU es de 7.075 %, lo cual es algo mucho más eficiente.

Conclusiones

1. El indicador (cantidad de instancias de control) de la variable independiente (tipo de red) se relaciona de manera inversa con el rendimiento de la red, la cual ha permitido desarrollar satisfactoriamente la implementación de la arquitectura de red avanzada REUNA, una de manera tradicional HDN mediante la herramienta GNS3 y otra de manera programable (SDN), primeramente se ha realizado una investigación amplia y diversas pruebas, la herramienta Mininet es la que mayormente es utilizada y es la que permite programar la red y ejecutar los scripts, adicional a ello se ha utilizado la herramienta Quagga que permite ejecutar el demonio zebra para poder utilizar los protocolos de enrutamiento, que para este escenario es OSPFv3 y el controlador Ryu que es el que ejercerá el plano de control al momento de ejecutar el script. La arquitectura de red está dada en base a las instancias de control; de las cuales en la HDN son 18 que están dadas por el plano de control de cada router en referencia a cada Ciudad de la topología; en la SDN es una sola instancia de control que es representado por el controlador Ryu; los resultados arrojan que la implementación SDN tiene mayor rendimiento; por tanto, se determinó la relación que existe entre el rendimiento de la red es inversamente proporcional con el tipo de implementación de una arquitectura de red avanzada, ya que a menos cantidad de instancias de control se tenga el rendimiento es más eficiente; esto debido a que los controladores ejecutan un proceso para hacer posible que los paquetes lleguen a su destino dentro de la red, lo cual es lógico pensar que si hay más controladores los valores del rendimiento incrementa.

2. Se ha determinado a través de las pruebas que la relación que existe entre el retardo y el tipo de implementación de una arquitectura de red avanzada es inversamente proporcional; es preciso indicar que los datos en la implementación HDN es mayor que en la implementación SDN, ya que se tiene una media de tiempo de retardo de transmisión de **6 183.30700 milisegundos** en la HDN en comparación de los **16.47689 milisegundos** en la SDN, el retardo de procesamiento generado en la HDN es de **0.02157 milisegundos** comparado con los **0.00006 milisegundos** que se tiene en la SDN, y la variación de retardo generado por la HDN es de **20.05345 milisegundos**, mientras tanto en la SDN solo tenemos **0.01110 milisegundos,** claramente se afirma que mientras más controladores haya según el tipo de implementación tendremos menor rendimiento, ya que tendremos mucho más tiempo requerido para enviar los paquetes.

3. Se ha determinado mediante las pruebas que la relación que existe entre la pérdida de paquetes y el tipo de implementación de arquitectura de red avanzada es inversamente proporcional y que es mucho mayor en la implementación HDN que en la implementación SDN; esto debido a que en la HDN existe una media de **3.43 paquetes perdidos** ICMPv6, comparado con los **1.90 paquetes perdidos** ICMPv6 que ha generado la SDN; en los paquetes UDP de un s*treaming* video realizado, en la HDN se obtuvo **458.35 paquetes perdidos**, sin embargo en la SDN solo se tiene **0.05 paquetes perdidos** UDP en el *streaming* realizado; en la HDN se tiene una media de **231 paquetes perdidos**, y para el caso de la SDN hay un promedio de **1 paquete perdido**; por tanto mientras más instancias de control existan según el tipo de implementación tendremos menor rendimiento de red, ya que habrán más perdida de paquetes y se requerirá más reenvíos

4. Se ha determinado mediante las pruebas que la relación que existe entre el consumo de recursos computacionales y el tipo de implementación de arquitectura de red avanzada es inversamente proporcional y que es mucho mayor en la implementación HDN que en la implementación SDN, ya que el nivel de consumo de CPU en la HDN es de **42.69 %**, frente al **7.08 %** que consume la SDN; el nivel de consumo de Memoria, en este caso RAM en la HDN es de **3 876.29 MB**, frente al **898.41 MB** que consume la SDN; el nivel de consumo de ancho de banda en la HDN muestra que es de **2.96 Mbps**, comparados con los **13 977.98 Mbps** que consume la SDN, esto indica que una implementación SDN de una arquitectura de red avanzada tiene una mayor capacidad de uso del ancho de banda y eso se refleja cuando se aumenta el tamaño de los paquetes, si se usa más ancho de banda se enviará más rápido y serán menos paquetes, también cabe precisar que esto se debe a que en esta emulación los enlaces de la red tradicional son enlaces GigabitEthernet de los dispositivos router y se conectan a la máquina virtual a través de la interfaz de los equipos que se conectan con enlace cloud entre ellos, sin embargo para la red avanzada SDN los enlaces son creados de manera virtual en los mismos equipos, reafirmamos que mientras menos sean las instancias de control, tendremos mejor nivel de consumo de nuestros recursos computacionales y que finalmente el rendimiento será mejor.

5. Las implementación SDN de una arquitectura de red avanzada genera un rendimiento más eficiente comparado con una implementación HDN, ya que las SDN tienen un mejor tiempo de retardo, mejor nivel de pérdida de paquetes y tienen un mejor nivel de consumo de recursos computacionales que una implementación tradicional HDN, y por tanto

esto en gran parte se debe a que la implementación SDN solo se ejecuta una sola instancia de control a diferencia de la implementación HDN que debe ejecutar tantas instancias de control por cada dispositivo de capa de red (router) que finalmente repercutirán en el rendimiento de la red, lo cual es un factor primordial tratándose de arquitectura de redes avanzadas enfocadas a la investigación y sector académico.

Recomendaciones

1. Que para casos de producción en la implementación SDN considerar la opción de no ejecutar un solo controlador, sino que previa evaluación determinar añadir controladores, si bien es cierto en esta investigación el rendimiento de la arquitectura de red avanzada REUNA implementada de manera programable SDN es la más eficiente, en un entorno de producción debemos considerar que es un solo punto donde se ejerce el plano de control, que dado cualquier complicación en un punto de la red puede llegar a afectar una mayor área de la esperada, finalmente esta recomendación es orientada enfocada al rendimiento de la red.

2. Se recomienda a las organizaciones y principalmente a las universidades seguir impulsando a desarrollar investigaciones en redes avanzadas que finalmente son un enlace para la conectividad IPv6 que actualmente está tomando mayor énfasis en entornos de producción.

3. Debido a que las redes definidas por software, nos cambia el paradigma de como pensar en el diseño e implementación para el funcionamiento de una red, se recomienda que las universidades que tengan especialidades relacionadas a las tecnologías de redes, impulsen a las escuelas considerar este paradigma SDN necesario dentro de su plan de estudios, ya que es una oportunidad para seguir desarrollando investigaciones, realizar implementaciones basadas en la programación de la red, que permitirán dar un paso de grandes opciones y que finalmente sea un enlace para no seguir pensado en el funcionamiento de las redes de una forma tradicional, sino que en realidad tenemos una opción eficiente de implementar redes que finalmente son las que operan para el funcionamiento de muchas de las aplicaciones y plataformas que existen dentro de las organizaciones; por tanto es necesario que los profesionales orientados a las redes tengan las habilidades de programar una red, ya que es un especialista en redes el que se encarga de que a nivel de infraestructura y configuraciones se tenga una red eficiente y que finalmente cuando ocurran cambios en la red, no requiera hacer muchos cambios en distintos dispositivos sino que pueda ser uno o unos cuantos puntos donde se realiza estas modificaciones, esto ayudará a tener un mejor control de red.

4. Este trabajo espera ser un impulso para que finalmente las universidades, organizaciones y demás entorno empresarial puedan migrar hacia una red definida por

software, con múltiples opciones de programar su propia red; también se espera que este trabajo sirva como antecedente para futuras tesis e investigaciones que permita realizar otras evaluaciones en arquitecturas de redes avanzadas con implementaciones de manera tradicional HDN y/o de manera programable SDN, en base a la experiencia realizada se sugiere otro tipo de enfoque que se requiere investigar; como seguridad, gestión y entre otros factores a identificar que ayuden a optimizar el funcionamiento de las redes avanzadas.

Referencias Bibliográficas

Abdullah, A., Kemal, A., & Selcuk, U. (2015). SDN-based Resilience for Smart Grid Communications. *IEEE*, 33.

Albert Greenberg, G. H. (2005). A Clean Slate 4D Approach to Network Control and. *AT&T*, 499-514.

Amoedo, D. (12 de 04 de 2020). *Ubunlog*. Obtenido de Iftop, vigila el consumo de ancho de banda de tu red en tiempo real: https://ubunlog.com/iftop-vigila-ancho-banda/

Ariganello, E. (2014). *REDES CISCO Guía de estudio para la certificación CCNA Routing y Switching.* Madrid-España: Ra-Ma S.A Editorial y Publicaciones.

Barona, L. L. (2013). *Una nueva alternativa a Mininet: emulación de una red definida por software.*

Bolatti, D. C. (2018). *Evaluación de performance en Redes Definidas por Software.* Chaco, Argentina: RedUNCI - UNNE - ISBN 978-987-3619-27-4.

Cabezas Bullemore, A., & Bravo Marchant, M. S. (2010). *Redes avanzadas en América Latina: Infraestructura para el desarrollo regional en ciencia, tecnología e innvovación.* -: -.

Cai, Z. (2011). Maestro: Achieving Scalability and Coordination in Centralized Network Control Plane. *Rice University*, 102-105.

Cantli, J. R. (1997). *CONECTIVIDAD DE REDES DE COMPUTADORAS.*

Castillo Velazquez , J. I., Ramírez Díaz, E. Y., & Marchand Niño, W. R. (2019). Use of GNS3 Cloud Enviroment for Network Management Emulation when Comparing SNMP vs Syslog Applied Over an Advanced Network. *IEEE*.

Castillo Velázquez, J. I. (2019). *REDES DE DATOS Contexto y evolución.* México: Samsara.

Castillo Velázquez, J., Cobos Panduro, V., & Marchand Niño, W. (2018). IPV6 Connectivity and Management Emulation for REUNA, the Chilean Advanced Network. *IEEE*.

Centeno, C. M. (2014). Controladores SDN, elementos para su selección y evaluación. *Telem@tica*.

Cisco. (2010). *REDES CISCO.* aRGENTINA: Redes Cisco / coordinado por Daniel Benchimol. - 1a ed. - Banfield - Lomas de Zamora : Gradi.

De Cusatis, C. (2013). *HandBook of Fiber Optic Data Conmunication: A Practical Guide to Optical Networking.* Academic Press.

Diego Kreutz, F. M. (2015). Software-Defined Networking:. *IEEE*, 63.

EcuRed. (25 de Marzo de 2015). *EcuRed (Conocimiento con todos y para todos)*. Obtenido de https://www.ecured.cu/Protocolos_de_red

Enríquez, A. E. (2015). *Análisis de rendimiento en redes IPv6.*

Feamster, N., Rexford, J., & Zegura, E. (2014). The Road to SDN: An intellectual History of Programmable Networks. *ACM SIGCOOM*.

Ferguson, J. (09 de 12 de 2019). *Ubuntu Wiki*. Obtenido de Quagga: https://wiki.ubuntu.com/JonathanFerguson/Quagga

Fernández, P. Y. (2015). *Programación de redes SDN mediante el controlador POX*.

García, F. J. (2016). *Estudio de las tecnologías SDN y NFV*.

GÉANT. (19 de Febrero de 2020). *GÉANT PROJECT*. Obtenido de Global Connectivity: World Regions: https://geant3plus.archive.geant.net/Network/Global-Connectivity/Pages/World-Regions.aspx

Gnome. (2014). *Gnome*. Obtenido de Monitor del Sistema: https://help.gnome.org/users/gnome-system-monitor/stable/index.html.es

GNS3. (2020). *GNS3*. Obtenido de https://gns3.com/software

Gonzales, M. V. (1999). *Análisis de los factores de retardo de red que afectan el transporte de la voz sobre redes públicas frame relay.* San Nicolás de los Garza, N.L.

Hidalgo, D. A. (2014). *Diseño e implementación de una aplicación de red bajo la arquitectura SDN.*

IBM. (09 de Fecbrero de 2020). *Rendimiento de red*. Obtenido de Rendimiento de red: https://www.ibm.com/support/knowledgecenter/es/ssw_ibm_i_73/rzahx/rzahxebusnet.htm

Icaza, J. A. (2016). *DISEÑO DE ESCENARIOS VIRTUALES DE DISTRIBUCION DE CONTENIDO MULTIMEDIA CON SOPORTE DE REDES DEFINIDAS POR SOFTWARE.*

Jianqing, W., Yan , H., Jiaming, K., Qin, T., & Xin, H. (2015). *A Study on the Dependability of Software Defined Networks.* China.

Kayssi, O. S. (2018). QoS Guarantee over Hybrid SDN/non-SDN Networks. *IEEE*.

Maistre, R. L. (23 de 10 de 2012). *LIGHT READING*. Obtenido de https://www.lightreading.com/carrier-sdn/sdn-architectures/google-sdn-works-for-us/d/d-id/699197

Mamamta, T. T. (2017). CORAL-SDN: A Software-Defined Networking Solution for the Internet of Things. *IEEE*.

Manzanares López, P., Muñoz Gea, J., Malgosa Sanahuja, J., & Flores de la Cruz, A. (2019). A virtualized infrastructure to offer network mapping functionality in SDN networks. *WILEY*, 1-10.

Mazza, N. h. (2014). *Gestión Estratégica de Recursos Informáticos.*

Mejía Fajardo, Á. M. (2004). Redes Convergentes. *CIBERTEC*.

Minh, P., & Doan B, H. (2016). SDN applications - the intent-based Northbound interface realisation for extended applications. *IEEE*, 372-377.

Mininet, T. (2018). *Mininet Overview*. Obtenido de http://mininet.org/overview/

Moreno, J. C. (2015). *Estudio de las redes definidas por software y escenarios virtuales de red orientados al aprendizaje.*

Muro, Y. A. (2016). *PLATAFORMA DE PRUEBAS PARA EVALUAR EL DESEMPEÑO DE LAS REDES DEFINIDAS POR SOFTWARE BASADAS EN EL PROTOCOLO OPENFLOW.*

Musante, M. (2017). *Diseño de una red industrial.*

Open Networking Foundation. (2019). *Redes Definidas por Software*. Obtenido de https://www.opennetworking.org/sdn-definition/?nab=0

Pinilla, R. Á. (2015). *Estudio de las redes definidas por software mediante el desarrollo de escenarios virtuales basados en el controlador OpendayLight.*

Ramírez González, T. (21 de 05 de 2016). *ComputerHoy*. Obtenido de ¿Qué es el comando Ping y cómo funciona?: https://computerhoy.com/noticias/internet/que-es-comando-ping-como-funciona-42607

Ramos de Santiago, F. J. (2010). *Análisis e implementación de un sistema real de medida de ancho de banda.* Madrid.

REDCLARA. (28 de Diciembre de 2016). *Qué son las Redes Avanzadas*. Obtenido de Qué son las Redes Avanzadas: https://www.redclara.net/index.php/red-y-conectividad/que-son-las-redes-avanzadas

REDCLARA. (Agosto de 2020). *¿Qué Diferencia a RedCLARA de Otras Redes?* Obtenido de ¿Qué Diferencia a RedCLARA de Otras Redes?: https://www.redclara.net/index.php/es/red/redclara/que-diferencia-a-redclara-de-otras-redes

REUNA. (19 de Febrero de 2019). *REUNA*. Obtenido de https://www.reuna.cl/index.php/nuestra-red/#red-nacional

REUNA. (09 de Febrero de 2020). *REUNA*. Obtenido de https://www.reuna.cl/index.php/reuna/

RFC 793. (Septiembre de 1981). *PROTOCOLO DE CONTROL DE TRANSMISIÓN*. Obtenido de RFC: 793: https://www.rfc-es.org/rfc/rfc0793-es.txt

RFC-1247. (Julio de 1991). *OSPF Version 2*. Obtenido de https://tools.ietf.org/html/rfc1247

RFC-1884. (Diciembre de 1995). *IP Version 6 Addressing Architecture*. Obtenido de https://tools.ietf.org/html/rfc1884

RFC-2460. (Diciembre de 1998). *Internet Protocol, Version 6 (IPv6)*. Obtenido de https://tools.ietf.org/html/rfc2460

RFC-5340. (Julio de 2008). *OSPF for IPv6*. Obtenido de https://tools.ietf.org/html/rfc5340

Rouse, M. (Dicimebre de 2013). *searchdatacenter*. Obtenido de Emulación de hardware: https://searchdatacenter.techtarget.com/es/definicion/Emulacion-de-hardware

Sampieri, R. H. (2014). *Metodología de la Investigación.* México.

Serrano, C. D. (2015). *REDES DEFINIDAS POR SFOTWARE (SDN): OPENFLOW.*

Stalings, W. (2004). *COMUNICACIONES Y REDES DE COMPUTADORAS Séptima edición.* Madrid: PEARSON EDUCACIÓN S.A.

Subharthi, J. R. (2017). Networking for Cloud Computing: A Survey. *IEEE*.

Tanenbaum, A. S. (2003). *Redes de computadoras.* Pearson Educación México.

Tatang, D., Quinkert, F., Frank, J., Ropke, C., & Holz, T. (2017). SDN-GUARD: Protecting SDN Controllers Against SDN Rootkits. *IEEE*, 297-302.

Vergel, C. M. (2016). Evaluación de desempeño y configuraciones de las SDN mediante la simulación. *Revista Técnica de la Empresa de Telecomunicaciones de Cuba, S.A.*

Wireshark. (2020). *Wireshark*. Obtenido de Sobre Wireshark: https://www.wireshark.org/

Anexos

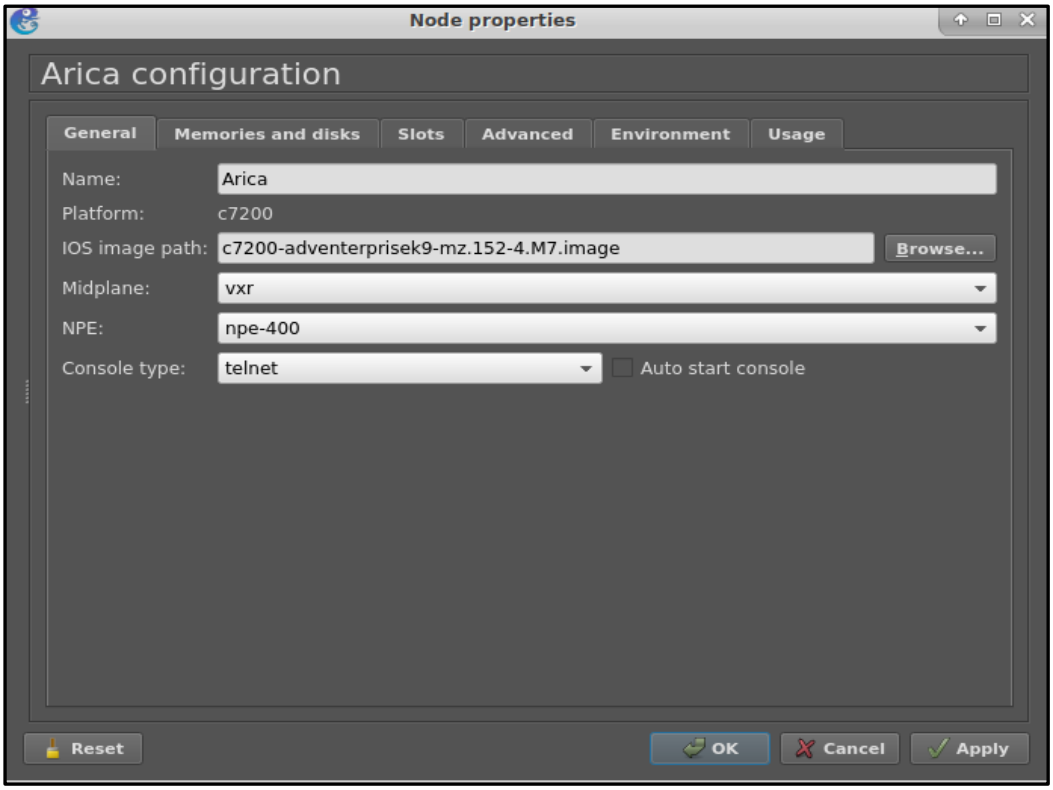
Anexo 1: Matriz de Consistencia

FORMULACIÓN DEL PROBLEMA	OBJETIVOS	HIPÓTESIS	VARIABLE	DIMENSIONES	INDICADORES	MÉTODOS Y TÉCNICAS
GENERAL ¿Qué relación existe entre el rendimiento y el tipo de implementación de una arquitectura de red avanzada? **ESPECÍFICOS** 1. ¿Qué relación existe entre el retardo y cada una de las implementaciones de arquitectura de red avanzada? 2. ¿Qué relación existe entre la perdida de paquetes y el tipo de implementación de una arquitectura de red avanzada?	**GENERAL** Evaluar la relación que existe entre el rendimiento y el tipo de implementación de una arquitectura de red avanzada para elaborar una guía de implementación. **ESPECÍFICOS** 1. Evaluar la relación que existe entre el retardo y el tipo de implementación de una arquitectura de red avanzada. 2. Evaluar la relación que existe entre la pérdida de paquetes y cada una de las implementaciones	**GENERAL** La relación que existe entre el rendimiento y el tipo de implementación de una arquitectura de red avanzada es inversamente proporcional. **ESPECÍFICOS** 1. La relación que existe entre el retardo y el tipo de implementación de una arquitectura de red avanzada es inversamente proporcional. 2. La relación que existe entre la pérdida de paquetes y el tipo implementación de arquitectura de red avanzada es	Tipo de red Rendimiento de red	Arquitectura de red Retardo de red Pérdida de paquetes	• Cantidad de instancias de plano de control • Retardo de transmisión (Milisegundos) • Retardo de procesamiento (milisegundos) • Variación de retardo (Milisegundos) • Pérdida de paquetes (número de paquetes enviados/recibidos/ perdidos)	**Tipo de Investigación:** Aplicada **Enfoque de Investigación:** Cuantitativo **Nivel de Investigación:** Explicativo **Diseño de Investigación:** Cuasi-experimental **Población:** 47 redes avanzadas **Muestra:** 1, red avanzada de Chile (REUNA)

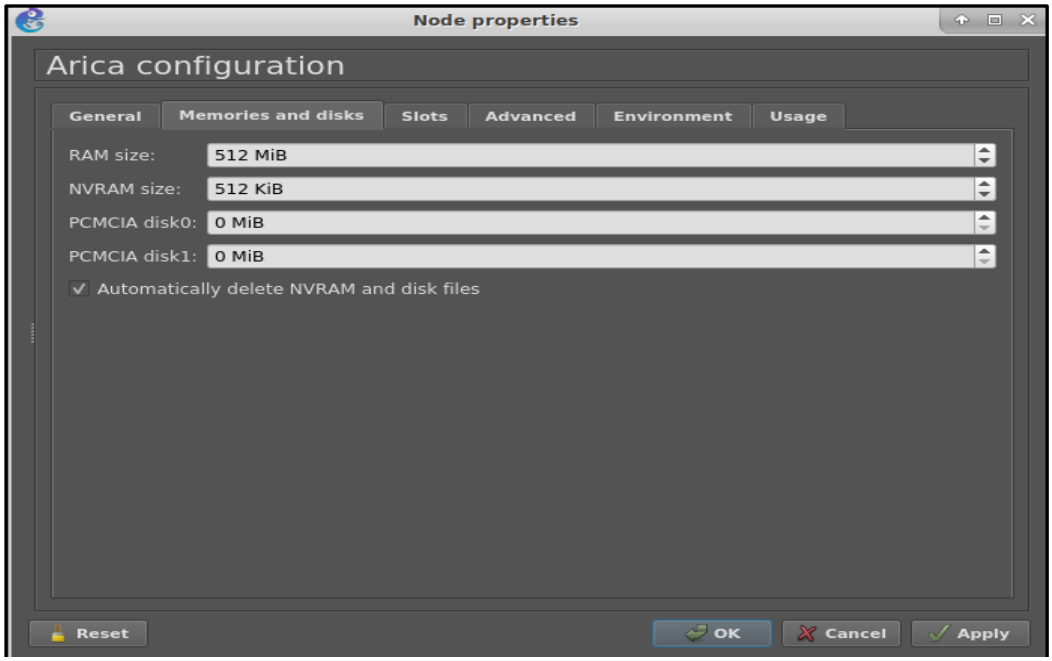
	de arquitectura de red avanzada	inversamente proporcional.			
3. ¿Qué relación existe entre el nivel de consumo de recursos computacionales y el tipo de implementación de una arquitectura de red avanzada?	3. Evaluar la relación que existe entre el nivel de consumo de recursos computacionales y el tipo de implementación de una arquitectura de red avanzada	3. La relación que existe entre el nivel de consumo de recursos computacionales y el tipo de implementación de una arquitectura de red avanzada es inversamente proporcional.	Consumo de recursos computacionales	• Nivel consumo de CPU (%) • Nivel consumo de memoria (MB) • Nivel de consumo de ancho de banda (Mbps)	**Herramientas de recolección de datos:** • Gestión de estado • Captura de paquetes y logs • Wireshark **Software de Emulación de red avanzada tradicional:** Mininet. **Software de Emulación de red avanzada SDN:** GNS3.

Anexo 2: Configuración de GNS3 para HDN

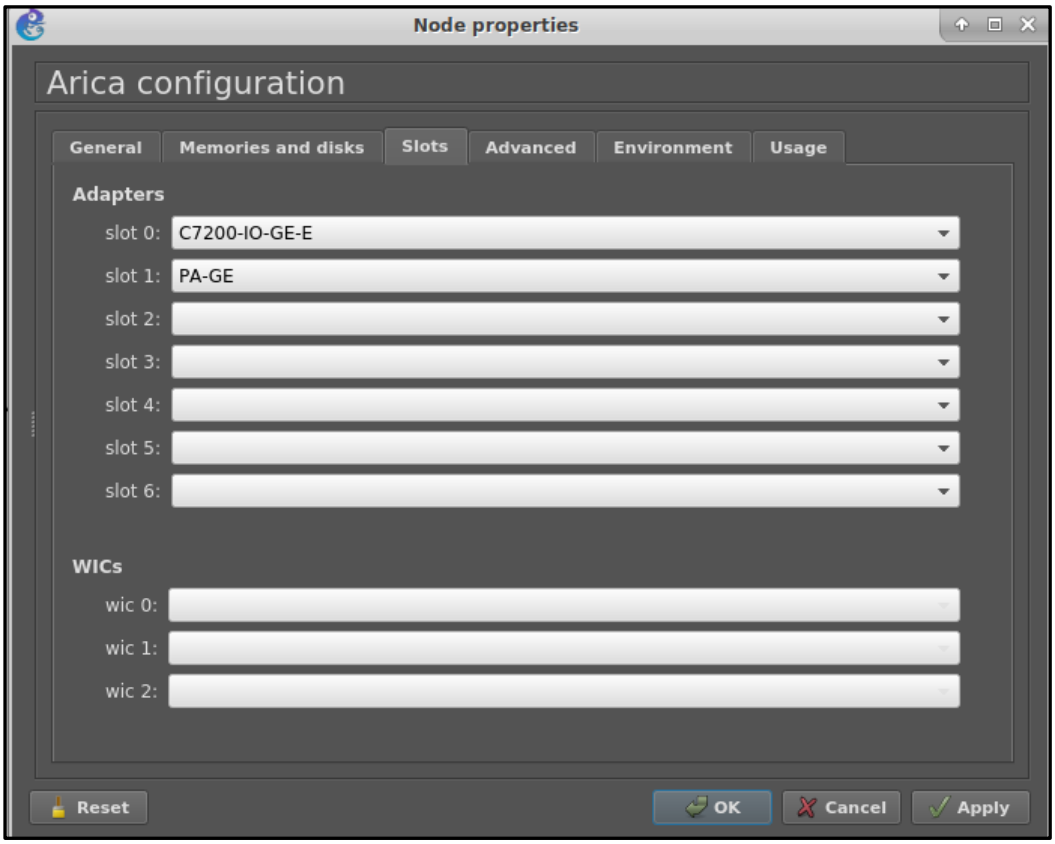
Añadimos el IOS router C7200 Para el router Arica, con la imagen IOS previamente descargada.



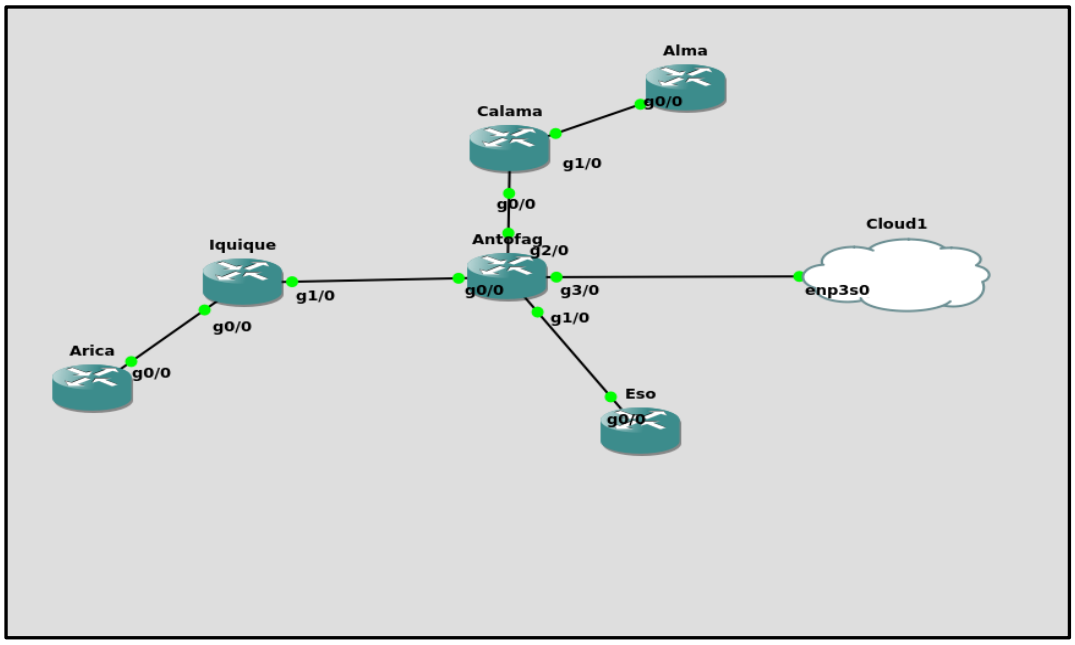
Dentro de las opciones del router nos dirigimos a la sección de Memoria y disco, donde le daremos 512 MB que serán de Memoria, que serán debidamente ejecutados ya que previamente emo instalados dynamips.



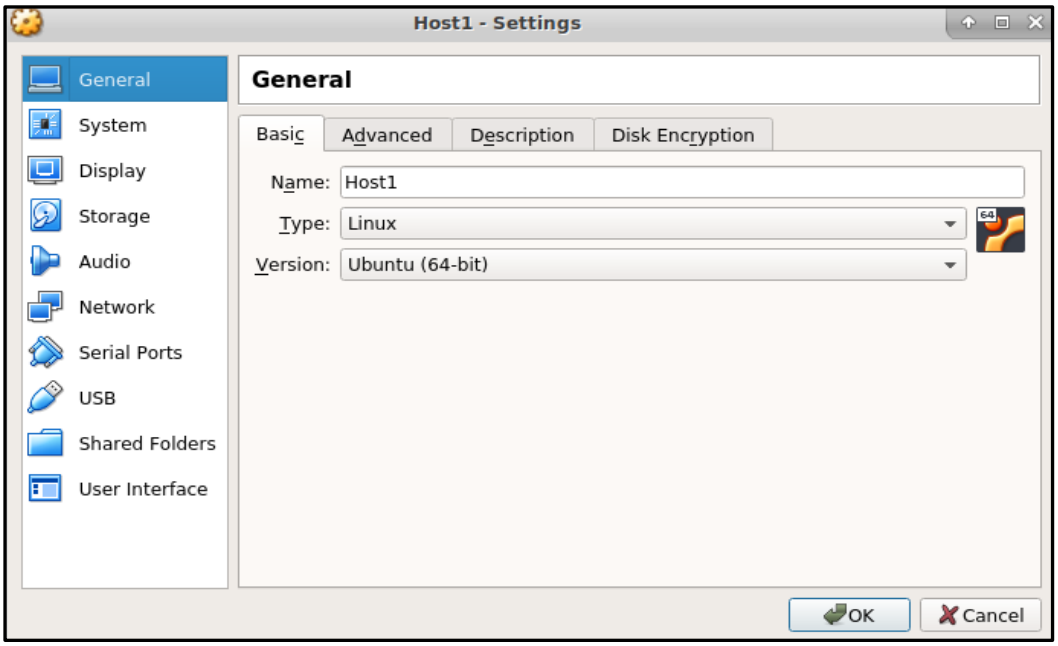
También agregamos los slots que se requieran, estas son las interfaces para los enlaces entre cada router, por defecto carga el slot 0 del router con una interface GigabitEthernet, sin embargo, añadiremos otra en el slot 1 como PA GE, GE hace referencia a GigabitEthernet, que servirá para conectar otro router.



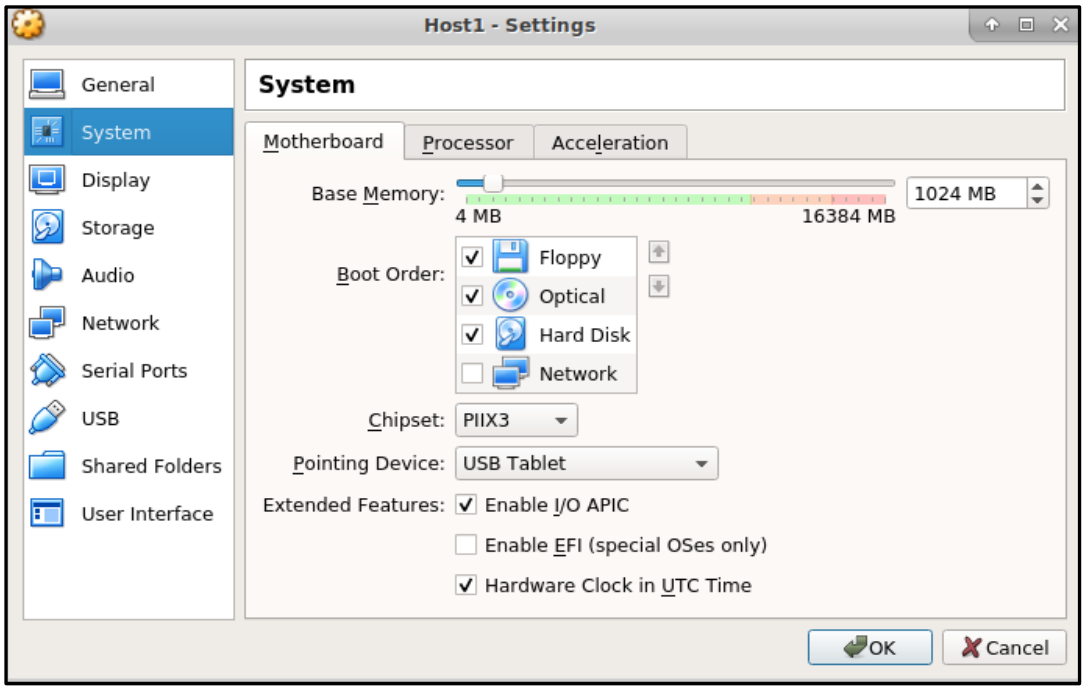
Este proceso solo se realiza una vez; y se añade el router a la topología, si es que se requiere cambiar algo en específico como añadir más slots a los nuevos routers; se entra a la configuración y se edita las preferencias. Quedará nuestra topología completa.



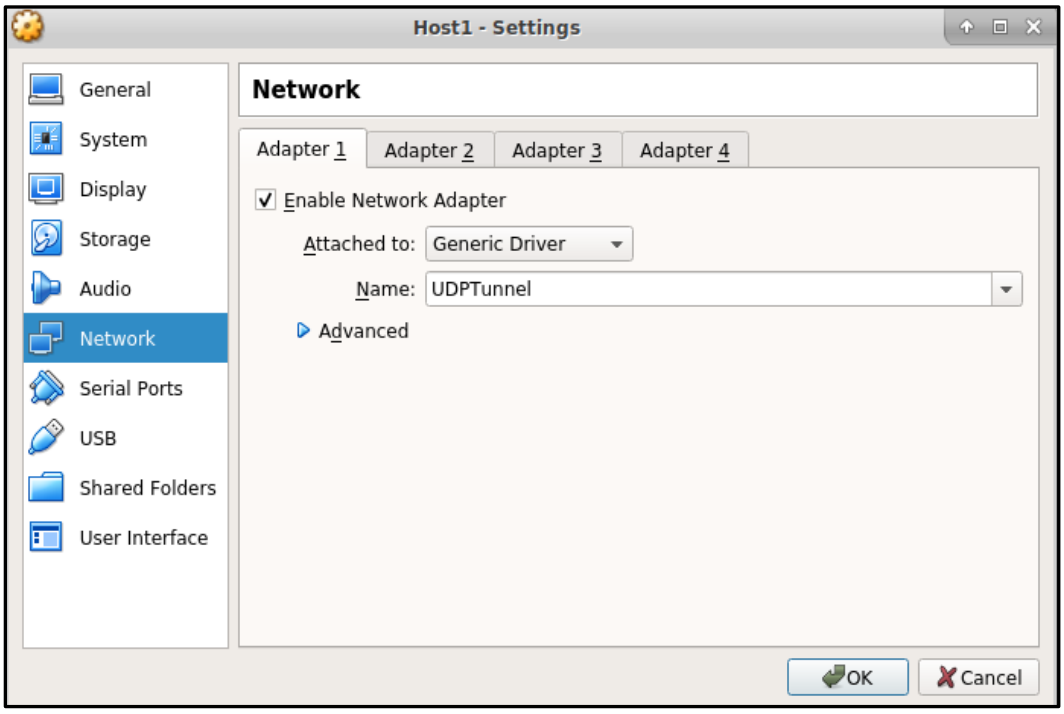
Sin embargo, requerimos añadir host, para ello debemos crear las máquinas virtuales, como se ha mencionado dentro del informe GNS3 tiene la capacidad de reconocer las máquinas virtuales creadas en VirtualBox, por ello es que a través de VirtualBox creamos los hosts.



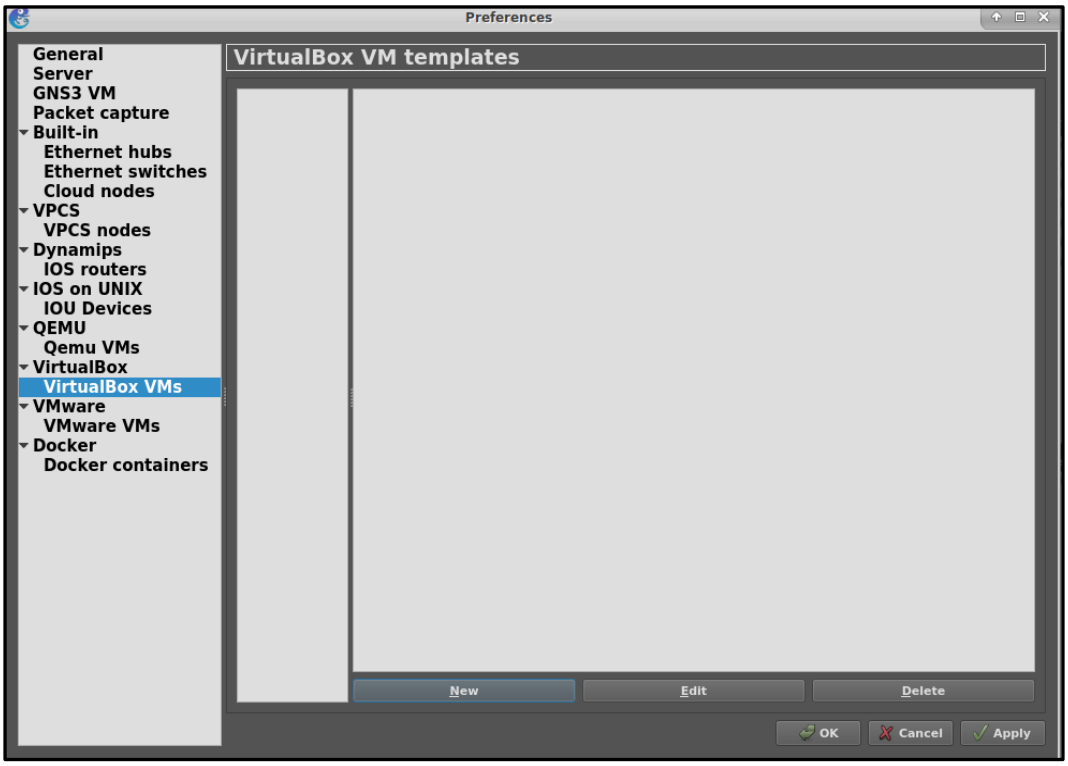
Para el caso de la máquina virtual le estamos indicando que trabajará con 1024 MB de RAM



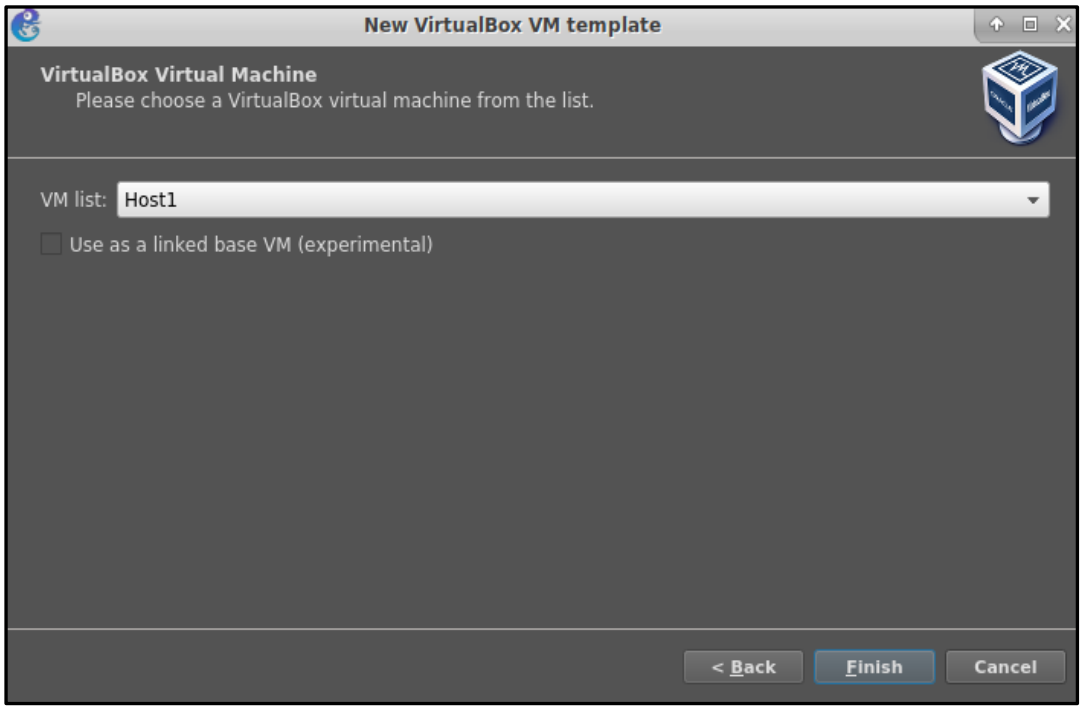
La interfaz de red de conexión para esta VM tiene ser como Generic Driver, de lo contrario no podrá establecer conexión.



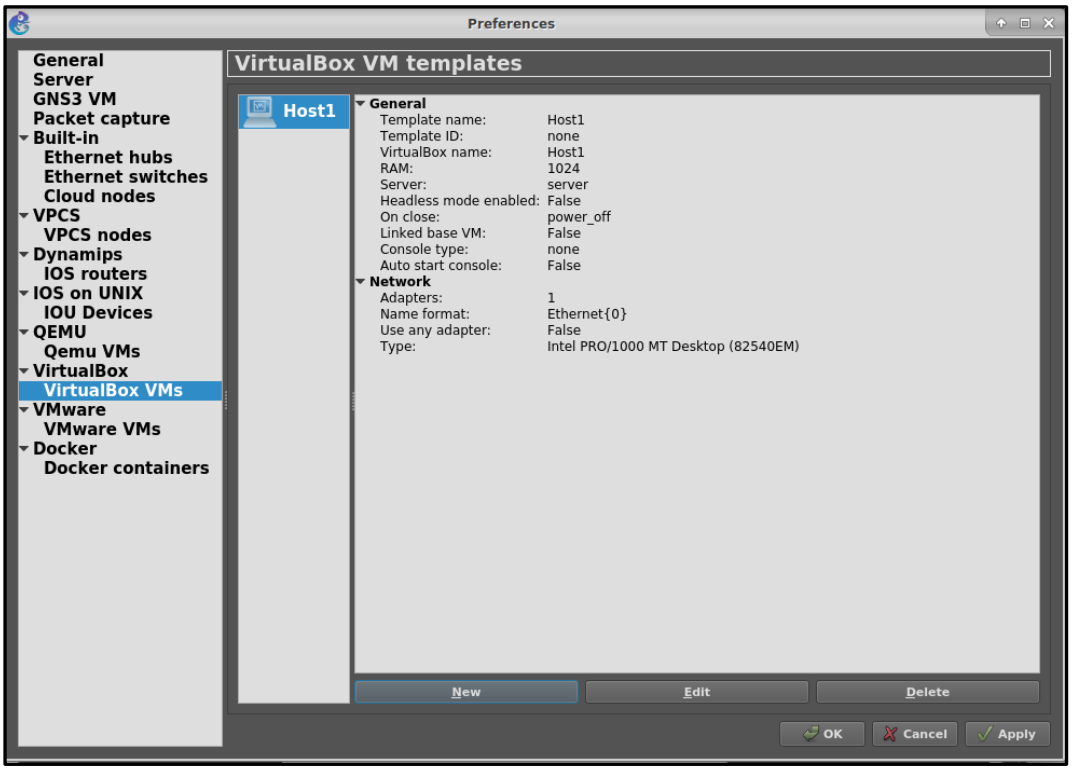
En GNS3 nos dirigiremos a las opciones en preferences hasta la sección de VirtualBox y para encontrar la VM creada seleccionamos la opción VirtualBox VMs, que se encontrará vacío, ya que no hemos añadido ninguna aún.



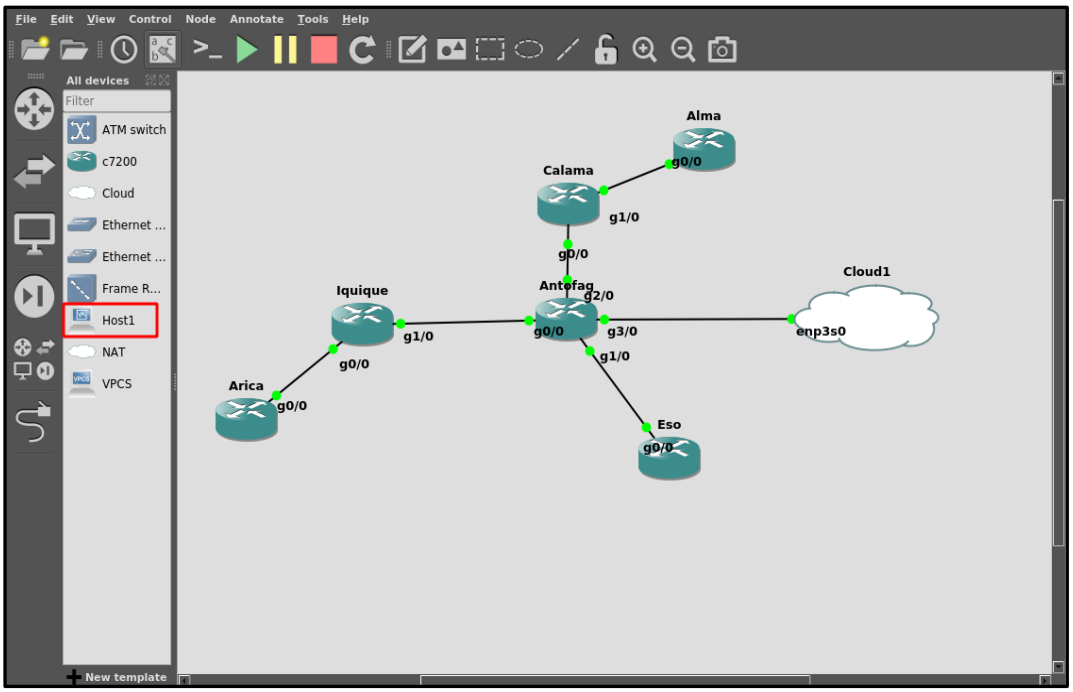
Para añadir una VM a nuestra topología nos dirigimos a la opción new, y nos aparecerá un ComboBox de la lista de máquinas virtuales que tenemos creadas, seleccionamos la que hemos creado Host1 y finalizamos



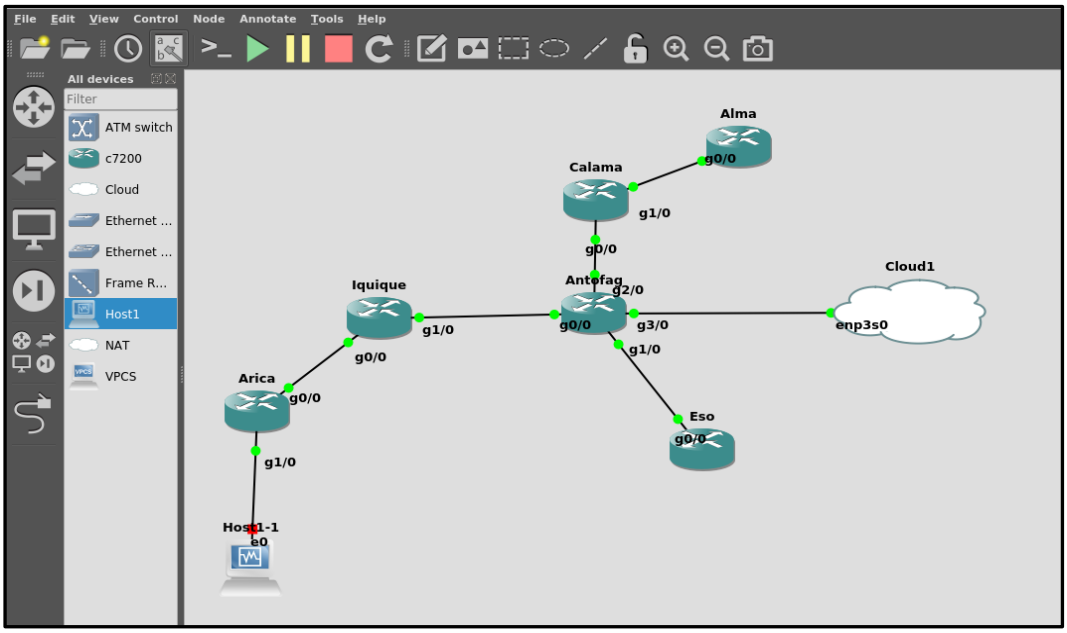
Ahora vemos que ya tenemos una VM Host 1 añadida a GNS3, con sus especificaciones después que fue creada.



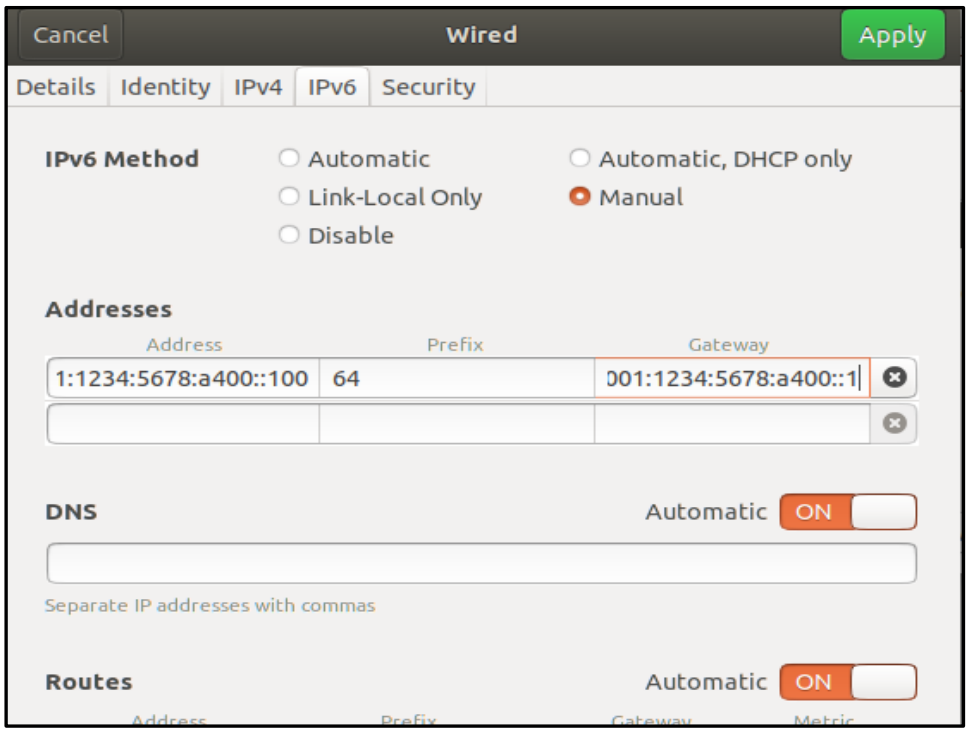
Finalmente, en nuestra área de trabajo de nuestra topología ya podemos visualizar la VM Host1 disponible para añadirla según se requiera



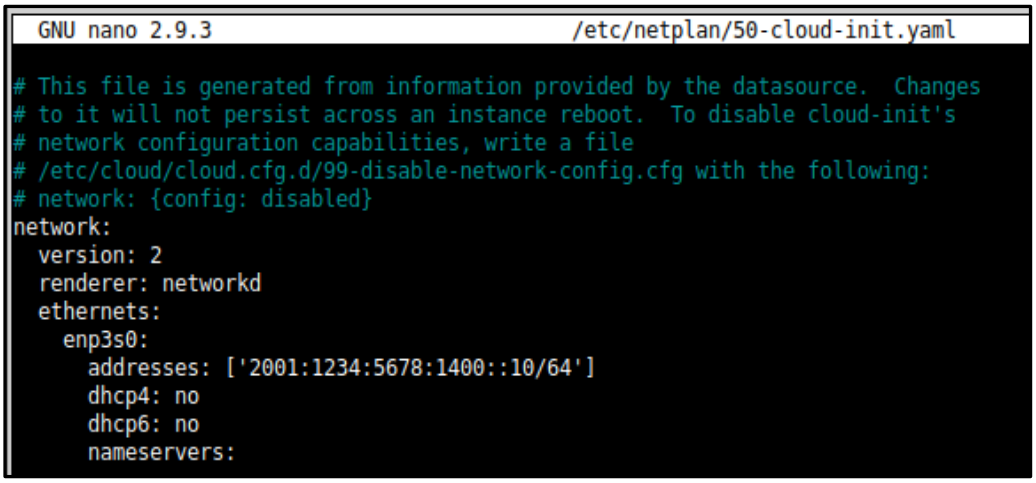
Ahora solo queda arrastrar la VM hacia el área de trabajo y conectarla mediante su enlace hacia el router, y ya podemos trabajar.



Cabe indicar que, si iniciamos la máquina virtual independientemente por VirtualBox, tendremos conflictos, para encender lo hacemos desde la misma área de trabajo en las opciones de la VM con clic derecho; ahora ya podemos configurar la IPv6 de este Host1.

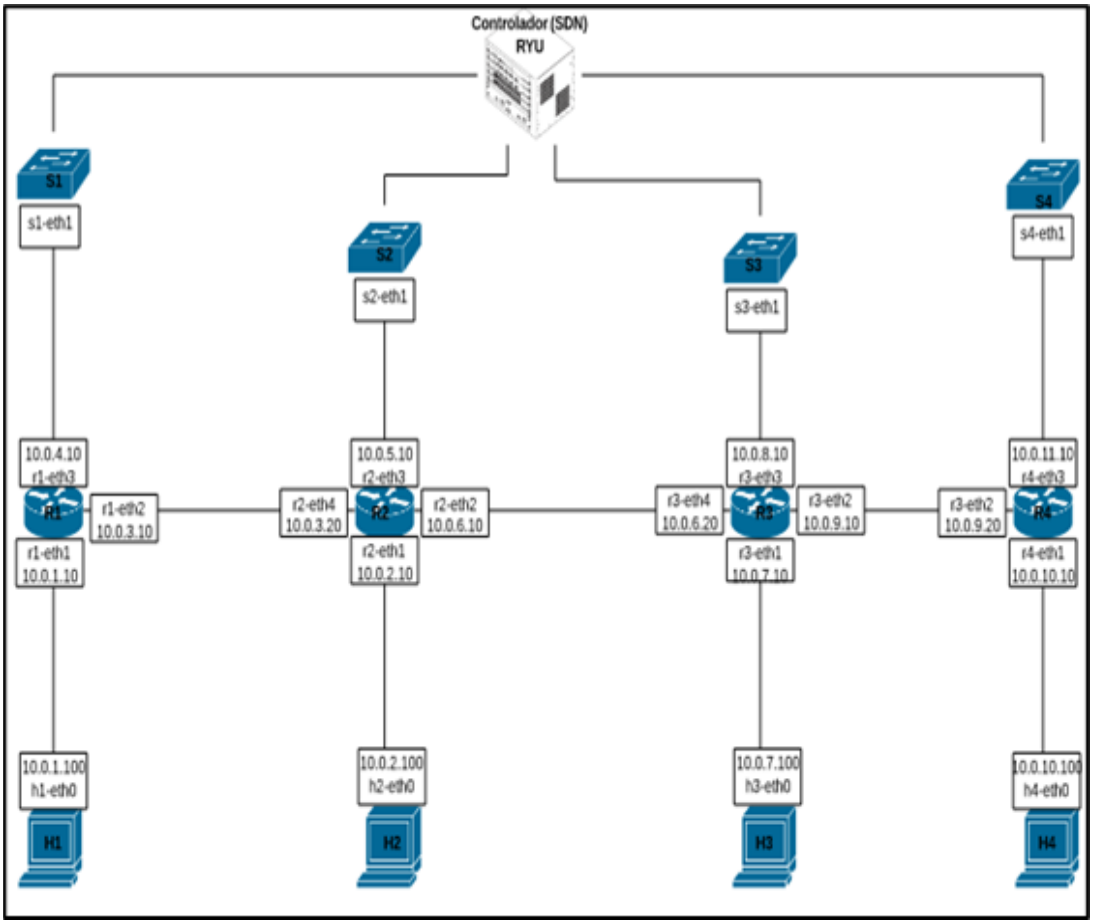


Las direcciones IPv6 deben configurarse según la asignación realizada para cada uno de los dispositivos y Host, nuestro dispositivo físico tiene la siguiente dirección IPv6, que se encuentra entre el router Antofagasta y Serena para que el enlace cloud funcione correctamente.



Anexo 3: Programación de Topología de Prueba SDN con IPv4 y OSPFv2

Topología que consta de cuatro *switch* que se conectan a cuatro routers respectivamente en redes diferentes y cuatro hosts conectados a los cuatro routers respectivamente en otras redes diferentes, se ha realizado la programación de la red y la configuración para OSPFv2.



Se crea el Archivo prueba.py, donde en parte del segmento de código implementado donde se realiza la importación de los objetos que proporciona Mininet y permitirá crear nuestra topología, donde el objeto *Topo* contiene información de la estructura de la red que estamos programando, el objeto *Mininet* es donde se ejecutarán las acciones de la red que estemos programando, el objeto *Node* es el que permite definir un nodo en la implementación de nuestra red, otro objeto que se ha importado es *RemoteController* que nos permite que toda la topología creada sea ejecutada a través de un controlador ya sea local o remoto, en esta prueba inicial hemos considerado experimentar los *switch*, para ello también se ha utilizado el objeto *OVSSwitch* el cual permite que los switch reenvíen tráfico, otros objetos muy importante que se ha utilizado son *setLogLevel* que nos permite

crear logs de la ejecución de nuestro script, el objeto *info* nos permite visualizar en consola lo que el emulador está realizando durante el inicio y finalización de la emulación, el objeto *CLI* es el que permite la comunicación con los nodos y además nos permite utilizar la interfaz *CLI*, la funcionalidad *time* es quien nos permite dar tiempo de ejecución al *api* y la funcionalidad *os* nos proporciona utilidad de matar procesos, también utilizamos una clase llamada *LinuxRouter* que permitirá hacer el reenvío de IP, para ello se debe utilizar los métodos *config* donde podemos expresar el renvío de IP como estado 1 habilitado y en el método *terminate* le indicamos 0 como deshabilitado.

```
#!/usr/bin/python
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Node, RemoteController, OVSSwitch
from mininet.log import setLogLevel, info
from mininet.cli import CLI
import time
import os

class LinuxRouter( Node ):
    "Nodo habilitado para reenvio de IP."
    #Reenvio de IP
    def config( self, **params ):
        super( LinuxRouter, self).config( **params )
        self.cmd( 'sysctl net.ipv4.ip_forward=1' )

    def terminate( self ):
        self.cmd( 'sysctl net.ipv4.ip_forward=0' )
        super( LinuxRouter, self ).terminate()
```

El segmento 2 del código, donde se ha creado la clase con nombre SDNv4, y a través de método build nos permitirá construir la emulación de nuestra topología hasta que se termine de ejecutar, se ha asignado una dirección IPv4 por defecto a los router, creando los enlaces y definiendo el Gateway que permitirá conectividad entre redes, los switch son definidos en Mininet por ello para agregarlos se realiza a través de la propiedad self.addSwitch, y se ingresa al menos un parámetro, que sería el nombre del Switch, aunque también se puede agregar un identificador pid, también para un host se debe agregar a través de la propiedad selfaddHost con parámetros igual al Switch; sin embargo, no existe formas de crear router, por ello la necesidad de integrar quagga, que a través de los protocolos le dará la funcionalidad de router a un nodo, por ello se agrega como self.addNode que es una propiedad de Mininet que tiene como funcionalidad crear nodos, ya estos nodos los podemos dar la funcionalidad que se requiere, incluso como host o switch, pero para nuestro caso se requiere darle la funcionalidad de router, esta propiedad

también recibe como parámetro el nombre y a la vez se le asigna la IP por defecto que tendrá, para construir los enlaces como es lógico deben ser entre 2 dispositivos, es decir, entre host-router ó entre router-switch y entre switch-host, lo lógico entonces es pasar dos parámetros asociados a los 2 dispositivos, donde el valor del primero se designa como intfName1 y el segundo como intfName2, la propiedad intfName nos permite darle el nombre al enlace y también añadir IPv4 al dispositivo, esto a través de los parámetros params1 y params2 que para algunos casos de la topología lo requiere y los otros vienen dados por el defaultIP que se ha designado, otra función que podemos apreciar es la de defaultRoute que permite indicar cual es la dirección Gateway para un segmento de red.

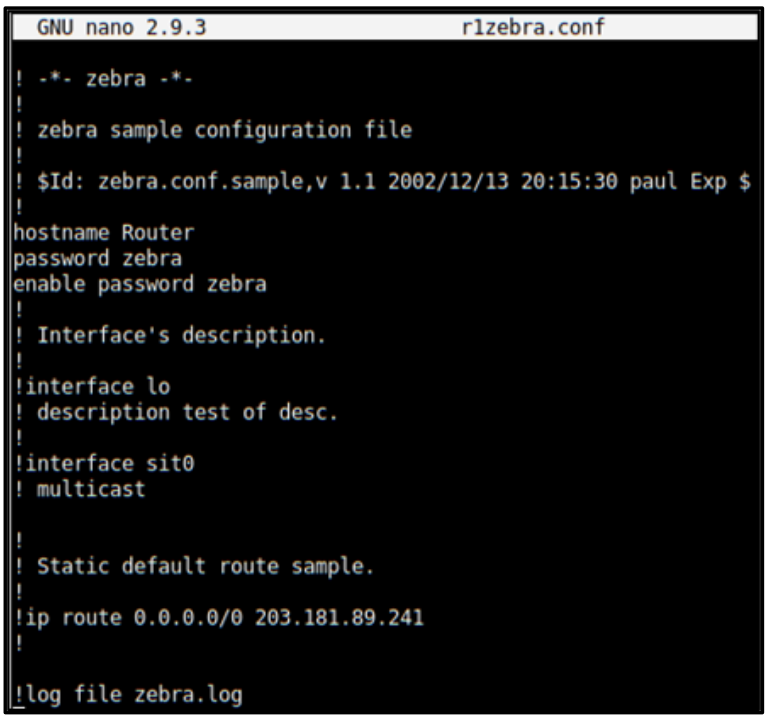
```
class SDNv4( Topo ):
    "A LinuxRouter connecting three IP subnets"

    def build( self, **_opts ):

        defaultIP1 = '10.0.3.10/24'  # Agregamos IP por default para los router
        defaultIP2 = '10.0.3.20/24'
        defaultIP3 = '10.0.6.20/24'
        defaultIP4 = '10.0.9.20/24'
        router1 = self.addNode( 'r1', cls=LinuxRouter, ip=defaultIP1 )
        router2 = self.addNode( 'r2', cls=LinuxRouter, ip=defaultIP2 )
        router3 = self.addNode( 'r3', cls=LinuxRouter, ip=defaultIP3 )
        router4 = self.addNode( 'r4', cls=LinuxRouter, ip=defaultIP4 )
        switch1 = self.addSwitch('s1',dpid='1000000000000001')
        switch2 = self.addSwitch('s2',dpid='1000000000000002')
        switch3 = self.addSwitch('s3',dpid='1000000000000003')
        switch4 = self.addSwitch('s4',dpid='1000000000000004')

        h1 = self.addHost( 'h1', ip='10.0.1.100/24', defaultRoute='via 10.0.1.10',dpid='0000000000000001') #Gateway
        h2 = self.addHost( 'h2', ip='10.0.2.100/24', defaultRoute='via 10.0.2.10',dpid='0000000000000002')
        h3 = self.addHost( 'h3', ip='10.0.7.100/24', defaultRoute='via 10.0.7.10',dpid='0000000000000003')
        h4 = self.addHost( 'h4', ip='10.0.10.100/24', defaultRoute='via 10.0.10.10',dpid='0000000000000004')

        self.addLink(router1,router2,intfName1='r1-eth2',intfName2='r2-eth4') # Enlaces
        self.addLink(router2,router3,intfName1='r2-eth2',params1={ 'ip' : '10.0.6.10/24' },intfName2='r3-eth4')
        self.addLink(router3,router4,intfName1='r3-eth2',params1={ 'ip' : '10.0.9.10/24' },intfName2='r4-eth4')
        self.addLink(h1,router1,intfName2='r1-eth1',params2={ 'ip' : '10.0.1.10/24' })
        self.addLink(h2,router2,intfName2='r2-eth1',params2={ 'ip' : '10.0.2.10/24' })
        self.addLink(h3,router3,intfName2='r3-eth1',params2={ 'ip' : '10.0.7.10/24' })
        self.addLink(h4,router4,intfName2='r4-eth1',params2={ 'ip' : '10.0.10.10/24' })

        self.addLink(switch1,router1,intfName1='s1-eth1',intfName2='r1-eth3',params2={'ip':'10.0.4.10/24'})
        self.addLink(switch2,router2,intfName1='s2-eth1',intfName2='r2-eth3',params2={'ip':'10.0.5.10/24'})
        self.addLink(switch3,router3,intfName1='s3-eth1',intfName2='r3-eth3',params2={'ip':'10.0.8.10/24'})
        self.addLink(switch4,router4,intfName1='s4-eth1',intfName2='r4-eth3',params2={'ip':'10.0.11.10/24'})
```

En la última parte del código que se muestra en la Figura siguiente, es donde se programa la ejecución de la topología a través de la clase run, es aquí donde se hace funcionar OSPF, primeramente hacemos el llamado a nuestra topología creada que tiene nombre SDNv4 a través de la propiedad topo, también es aquí donde se envía toda la funcionalidad hacia el controlador a través de la propiedad RemoteController, nuestro controlador ryu se encuentra en la IP 192.168.8.150 y es enviado a través del puerto 6633 además se le indica que inicie con la propiedad start; debemos tener en cuenta que para los routers necesitamos llamarlos en esta sección con la propiedad getNodeByName haciéndole pasar como parámetro el nombre que se le ha declarado al nodo en la segunda parte del código que serían los routers r1, r2 r3 y r4 para que puedan utilizar la configuración OSPF, seguidamente se hace el llamado a zebra que es el demonio principal que proporciona enrutamiento para que pueda crear los sockets, que son las interfaces de programación de aplicación para los protocolos TCP/IP, esto se debe hacer para cada router con la propiedad cmd que es la que lee y crea archivos, una vez creado el socket entonces zebra puede utilizar la configuración de los archivos de cada router, es decir se utiliza los "rxzebra.conf", donde x define el número del router, una vez que el script reconoce zebra, se crea un socket que consiste en una extensión .api y otra .interface que

```
def run():
    "Test linux router"
    topo = SDNv4()
    net = Mininet(controller=RemoteController,topo=topo)
    c1 = net.addController('c1', ip='192.168.8.150', port=6633)
    net.start()
    info( '*** Routing Table on Router:\n' )

    r1=net.getNodeByName('r1')
    r2=net.getNodeByName('r2')
    r3=net.getNodeByName('r3')
    r4=net.getNodeByName('r4')
    info('Iniciando servicios zebra y ospfd:\n')

    r1.cmd('zebra -f /usr/local/etc/r1zebra.conf -d -z ~/Desktop/Sockets/r1zebra.api -i ~/Desktop/Sockets/r1zebra.interface')
    time.sleep(1)#Aqui zebra crea los api socket
    r2.cmd('zebra -f /usr/local/etc/r2zebra.conf -d -z ~/Desktop/Sockets/r2zebra.api -i ~/Desktop/Sockets/r2zebra.interface')
    r3.cmd('zebra -f /usr/local/etc/r3zebra.conf -d -z ~/Desktop/Sockets/r3zebra.api -i ~/Desktop/Sockets/r3zebra.interface')
    r4.cmd('zebra -f /usr/local/etc/r4zebra.conf -d -z ~/Desktop/Sockets/r4zebra.api -i ~/Desktop/Sockets/r4zebra.interface')

    r1.cmd('ospfd -f /usr/local/etc/r1ospfd.conf -d -z ~/Desktop/Sockets/r1zebra.api -i ~/Desktop/Sockets/r1ospfd.interface')
    r2.cmd('ospfd -f /usr/local/etc/r2ospfd.conf -d -z ~/Desktop/Sockets/r2zebra.api -i ~/Desktop/Sockets/r2ospfd.interface')
    r3.cmd('ospfd -f /usr/local/etc/r3ospfd.conf -d -z ~/Desktop/Sockets/r3zebra.api -i ~/Desktop/Sockets/r3ospfd.interface')
    r4.cmd('ospfd -f /usr/local/etc/r4ospfd.conf -d -z ~/Desktop/Sockets/r4zebra.api -i ~/Desktop/Sockets/r4ospfd.interface')

    CLI( net )
    net.stop()
    os.system("killall -9 ospfd zebra")
    os.system("rm -f *api*")
    os.system("rm -f *interface*")

if __name__ == '__main__':
    setLogLevel( 'info' )
    run()
```

son los que finalmente harán que ospfd lea la configuración de los archivos rxospfd.conf, donde x representa el número del router.

En nuestro script hacemos el llamado a los archivos *zebra*, para esto se ha configurado los archivos *rxzebra* para cada router, donde x representa el número del router respectivo, todos los archivos *zebra* son una copia del archivo *zebra.conf.sample* original que nos proporciona *quagga*, en la Figura N° 22 se aprecia la configuración del archivo *r1zebra.conf*.



Para la configuración OSPFv2 para cada router, también creamos un archivo por cada router que serán llamados en el script, en la Figura N° 23, se observa el archivo de configuración *r2ospfd.conf* para el *router2*, debemos tener en cuenta que el puerto 2604 referente al servicio *ospfd* debe estar ejecutándose al igual que el 2601 relaciona al servicio *zebra*, esta configuración se ha realizado en base al archivo *ospfd*.conf.sample que nos proporciona *quagga*, donde se le agrega un nombre identificador y agregamos las redes adyacentes al router, en la parte final se ha colocado un *log*, el cual es algo práctico para identificar los errores.

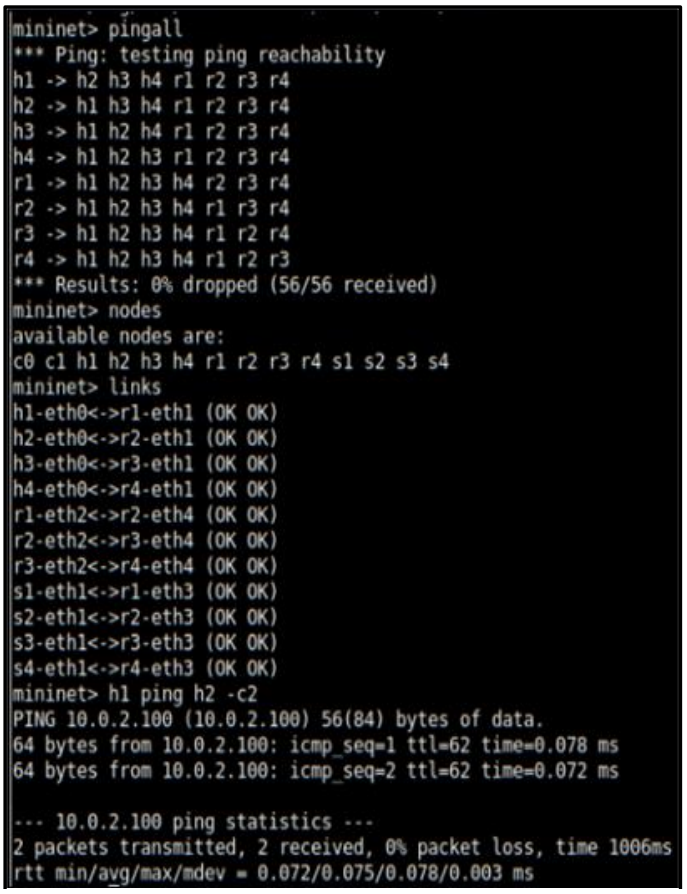
```
 GNU nano 2.9.3                    r2ospfd.conf

hostname r2_ospfd
password 123
enable password 123

router ospf
  ospf router-id 10.0.3.20
  network 10.0.2.0/24 area 0
  network 10.0.3.0/24 area 0
  network 10.0.5.0/24 area 0
  network 10.0.6.0/24 area 0
debug ospf event
log file /usr/local/etc/r2ospfd.log
```

Teniendo programada nuestra red y los archivos de configuración, ejecutamos dentro de Mininet el archivo SDNv4.py, para esta investigación se ejecuta dentro mininet/custom, también se ha instalado la herramienta OpenSource flowmanager en nuestro controlador, que nos proporciona tener una mejor visualización, en la Figura N°24 se aprecia el inicio del controlador

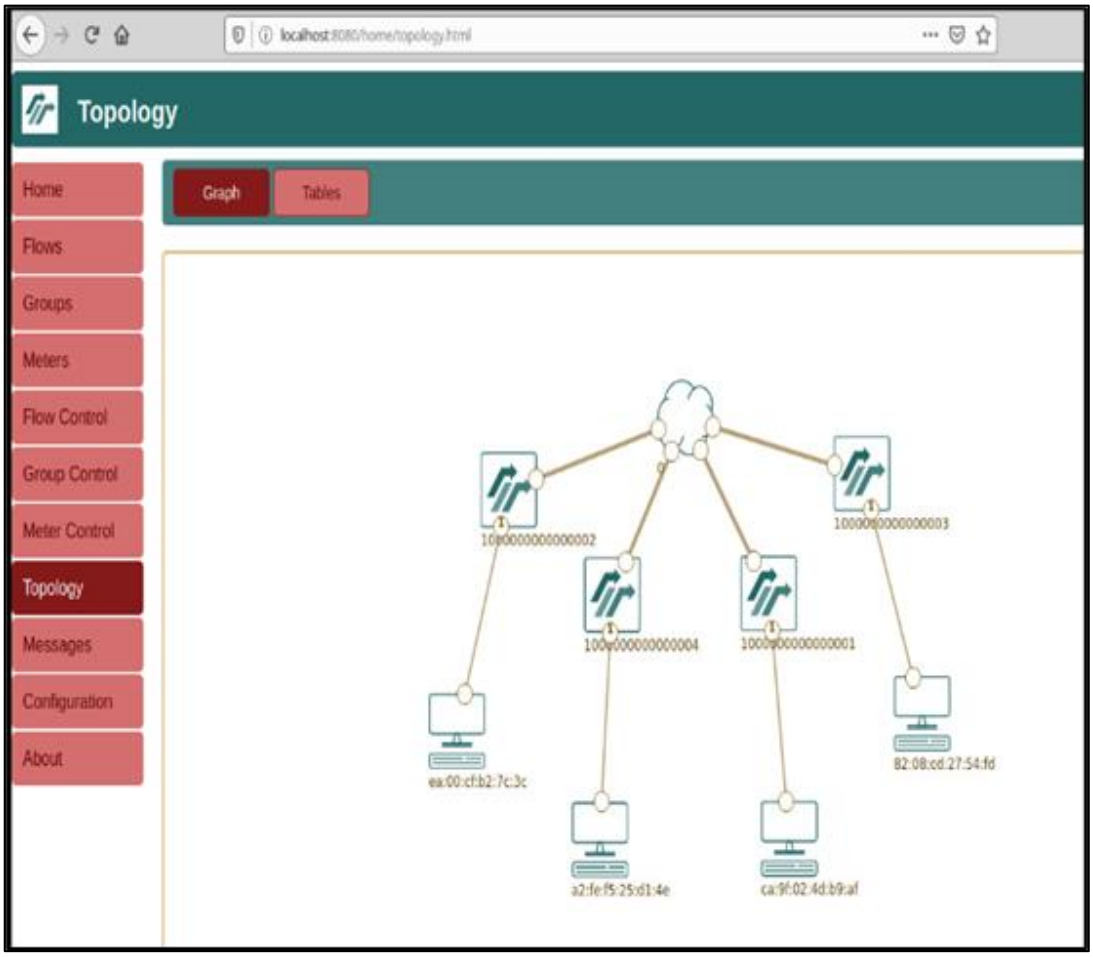
```
servidor@servidor:~$ ryu-manager --observe-links ~/flowmanager/flowmanager.py
ryu.app.rest_router
loading app /home/servidor/flowmanager/flowmanager.py
You are using Python v2.7.17.final.0
loading app ryu.app.rest_router
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu.topology.switches of Switches
instantiating app ryu.app.rest_router of RestRouterAPI
instantiating app /home/servidor/flowmanager/flowmanager.py of FlowManager
instantiating app ryu.controller.ofp_handler of OFPHandler
(3624) wsgi starting up on http://0.0.0.0:8080
```

Algo muy importante es que debemos verificar que los socket se deben crear, es decir los rxzebra.api, los rxzebra.interface y los rxospfd.interface deben crearse en la ruta que se ha designado Desktops/Sockets después de ejecutar el script, ya que estos permiten el funcionamiento de los protocolos de enrutamientos, en nuestro caso OSPFv2 mediante el demonio zebra, en la Figura N° 25 podemos observar que se crea el socket teniendo los archivos .api de zebra y los archivos .interface de zebra y ospfd para cada uno de los routers de nuestra red programada.

En la siguiente Figura se aprecia que después de ejecutar el *script* y habiéndose creado los *sockets* adecuadamente, se procedió a verificar que todos los nodos se hayan creado a través del comando *nodes*, asimismo se verificó el estado de todos enlaces de red que se ha programado a través del comando *links*, también se ha ejecutado el comando *pingall*, que demuestra que existe conectividad entre todos los nodos de la topología programada, finalmente se ha enviado dos paquetes *icmp* exitosos desde el *host1* hacia el *host2* a través de la funcionalidad de *ping*.



Debido a que en esta topología de prueba, existen dispositivos switch, es que podemos visualizar la topología a través del controlador *Ryu* integrado con *flowmanager*, en la Figura N° 27 se puede visualizar esto, el propósito de este escenario es enrutamiento, pero también se ha encontrado que en casos como esta topología, debemos acceder a cada switch y habilitar el reenvío a través de *openvswitch* para *OpenFlow*, primero por medio de la propiedad *ovs-vsctl set Bridge Sx protocols=OpenFlow13*, seguidamente de *ovs-ofctl -O openflow13 dump-flows Sx*, donde x refiere al número de *switch* tal como se le ha declarado el nombre.


localhost:8080/home/topology.html
Topology
Home
Flows
Groups
Meters
Flow Control
Group Control
Meter Control
Topology
Messages
Configuration
About
Graph
Tables
ea:00:cf:b2:7c:3c
a2:fe:f5:25:d1:4e
ca:9f:02:4d:b9:af
82:08:cd:27:54:fd

Anexo 4: Configuración de Servicio FTP

- ✓ **HDN**

En los hosts se instala de manera directa con el comando apt-get install vsftpd y nos conectamos mediante el protocolo ftp.

```
host2@host2-VirtualBox:~$ sudo apt-get install vsftpd
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  vsftpd
0 upgraded, 1 newly installed, 0 to remove and 658 not upgraded.
Need to get 115 kB of archives.
After this operation, 334 kB of additional disk space will be used.
Get:1 http://pe.archive.ubuntu.com/ubuntu bionic/main amd64 vsftpd amd64 3.0.3-9build1 [115 kB]
Fetched 115 kB in 1s (108 kB/s)
Preconfiguring packages ...
Selecting previously unselected package vsftpd.
(Reading database ... 128890 files and directories currently installed.)
Preparing to unpack .../vsftpd_3.0.3-9build1_amd64.deb ...
Unpacking vsftpd (3.0.3-9build1) ...
Processing triggers for ureadahead (0.100.0-20) ...
Setting up vsftpd (3.0.3-9build1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/vsftpd.service → /lib/systemd/system/vsftpd.service.
Processing triggers for systemd (237-3ubuntu10.3) ...
Processing triggers for man-db (2.8.3-2) ...
Processing triggers for ureadahead (0.100.0-20) ...
```

Verificamos que el estado del servicio FTP este activo y corriendo dentro del host2 que actuará como servidor.

```
host2@host2-VirtualBox:~$ systemctl status vsftpd.service
● vsftpd.service - vsftpd FTP server
   Loaded: loaded (/lib/systemd/system/vsftpd.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2020-11-23 08:05:36 -05; 2min 3s ago
 Main PID: 2455 (vsftpd)
    Tasks: 1 (limit: 2322)
   CGroup: /system.slice/vsftpd.service
           └─2455 /usr/sbin/vsftpd /etc/vsftpd.conf

Shin 23 08:05:36 host2-VirtualBox systemd[1]: Starting vsftpd FTP server...
Shin 23 08:05:36 host2-VirtualBox systemd[1]: Started vsftpd FTP server.
```

Verificamos la versión del servicio, y le indicamos que permita todas las conexiones por los puertos 20 y 21 del protocolo ftp; además desactivamos el firewall para evitar inconvenientes al momento de transferir archivos.

```
host2@host2-VirtualBox:~$ vsftpd -version
vsftpd: version 3.0.3
host2@host2-VirtualBox:~$ ufw allow 21
ERROR: You need to be root to run this script
host2@host2-VirtualBox:~$ sudo ufw allow 21
Rules updated
Rules updated (v6)
host2@host2-VirtualBox:~$ sudo ufw allow 20
Rules updated
Rules updated (v6)
host2@host2-VirtualBox:~$ sudo ufw status
Status: inactive
```

Agregamos un usuario test con su respectivo directorio que será utilizado para la utilidad de este servicio.

```
host2@host2-VirtualBox:~$ sudo adduser test
Adding user `test' ...
Adding new group `test' (1001) ...
Adding new user `test' (1001) with group `test' ...
Creating home directory `/home/test' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for test
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
```

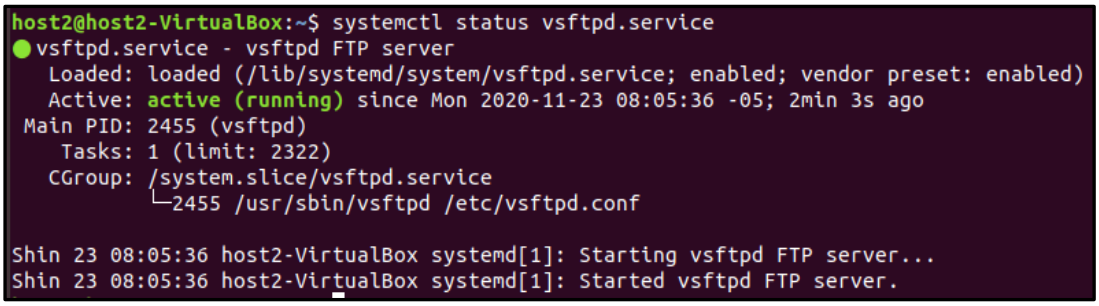
Le indicamos que este directorio le pertenecerá al usuario root para evitar eliminaciones del directorio, creamos una carpeta donde se hará la transferencia de archivos y este si la asignamos al usuario creado.

```
host2@host2-VirtualBox:~$ sudo chown root:root /home/test/
host2@host2-VirtualBox:~$ sudo mkdir /home/test/archivos
host2@host2-VirtualBox:~$ sudo chown test:test /home/test/archivos/
```

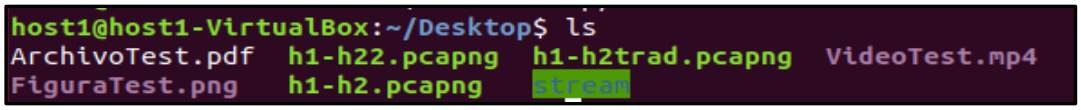
Finalmente revisamos el archivo de configuración de ftp con fines de bunas prácticas, deshabilitamos el usuario anónimo y habilitamos el local_enable y habilitamos permisos para poder copiar y escribir y nos aseguramos de que chroot_local_user esté habilitado.

```
host2@host2-VirtualBox:~$ sudo nano /etc/vsftpd.conf
```

Guardamos los cambios y reiniciamos el servicio con restart y finalmente volvemos a verificar el estado del servicio.



Verificamos en host1 los archivos que vamos a transferir ArchivoTest.pdf y VideoTest.mp4.

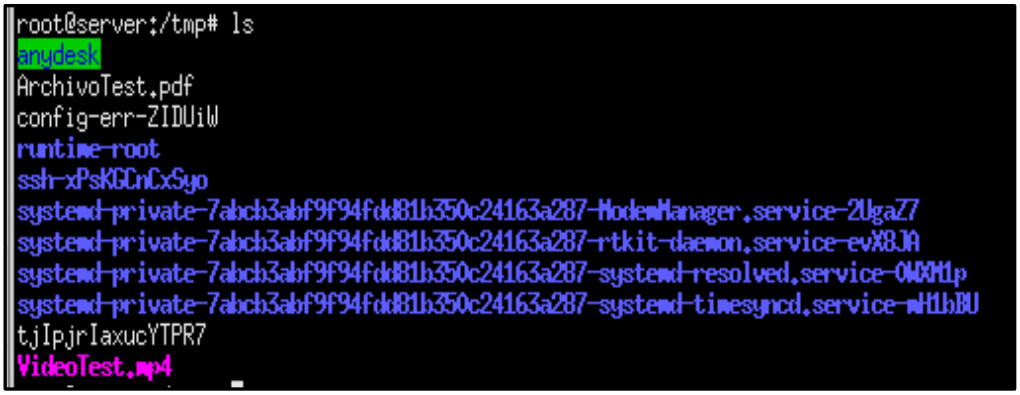


Finalmente nos conectamos desde el cliente host 1 hacia el servidor host2 y subimos los archivos con el comando put.

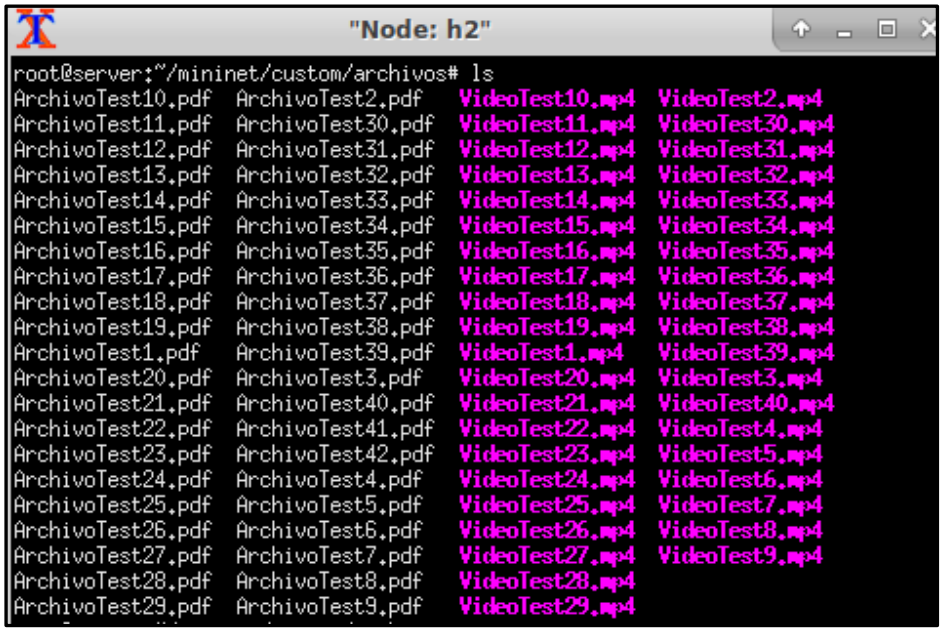
- ✓ **SDN**

Para el caso del servidor FTP en SDN, también instalamos de la misma manera el servidor FTP; pero debido a que los hosts se encuentran dentro del mismo equipo físico tal como nos permite Mininet hacer uso. Debemos habilitar primero IPv6 dentro de los archivos de Mininet con el host 2 que será quien actúe como servidor.

Como se ha mencionado, debido a que se encuentra en el mismo equipo Mininet nos permite utilizar los protocolos del equipo físico principal pero ejecutándolo como un temporal, para ello nos dirigimos al directorio /tmp en el host 1 que actuará como cliente y cargamos los archivos que vamos a utilizar para transferir del host 1 hacia host2 ArchivoTest.pdf y VideoTest.mp4.



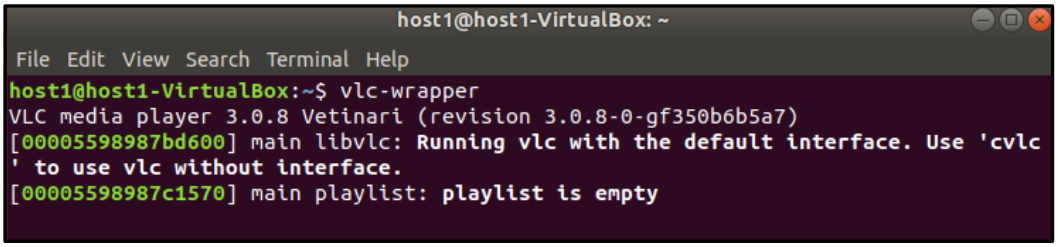
Finalmente transferimos los archivos entre host 1 y host2 mediante el comando put, que es una herramienta nativa de las distribuciones GNU/Linux.



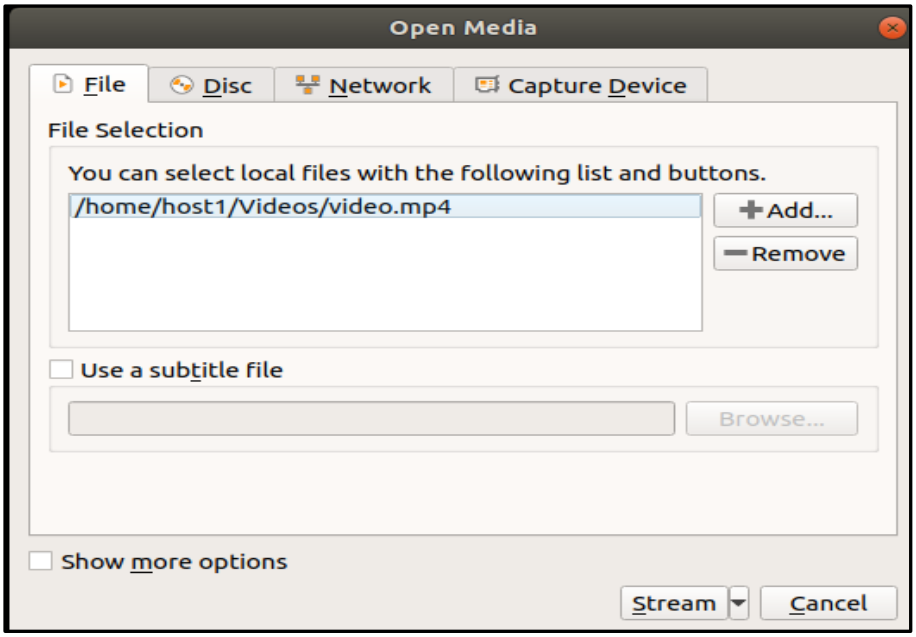
Anexo 5: Configuración de Streaming

- ✓ **HDN**

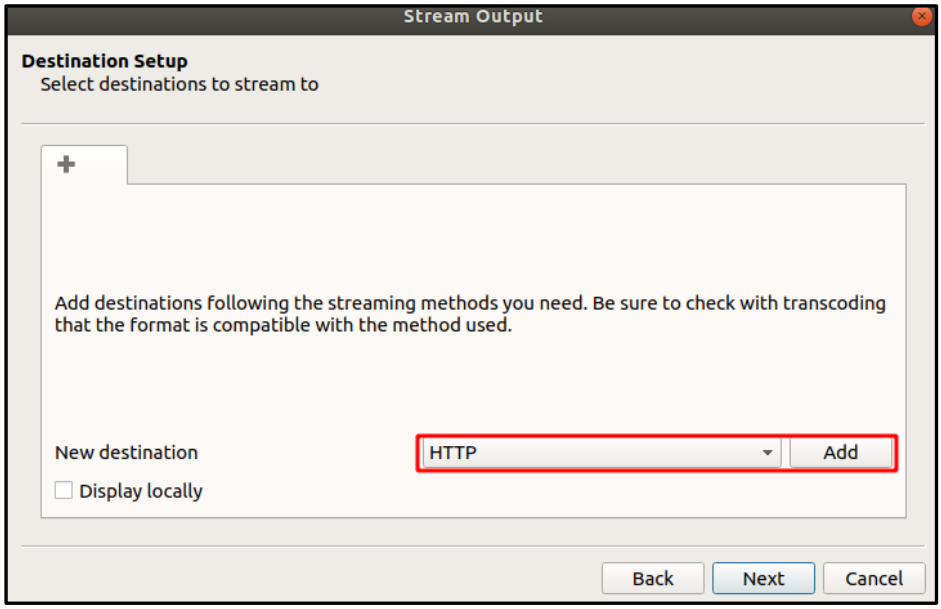
Primeramente, dentro del host1 inicializamos la herramienta vlc-wrapper que nos permitirá hacer la transmisión.



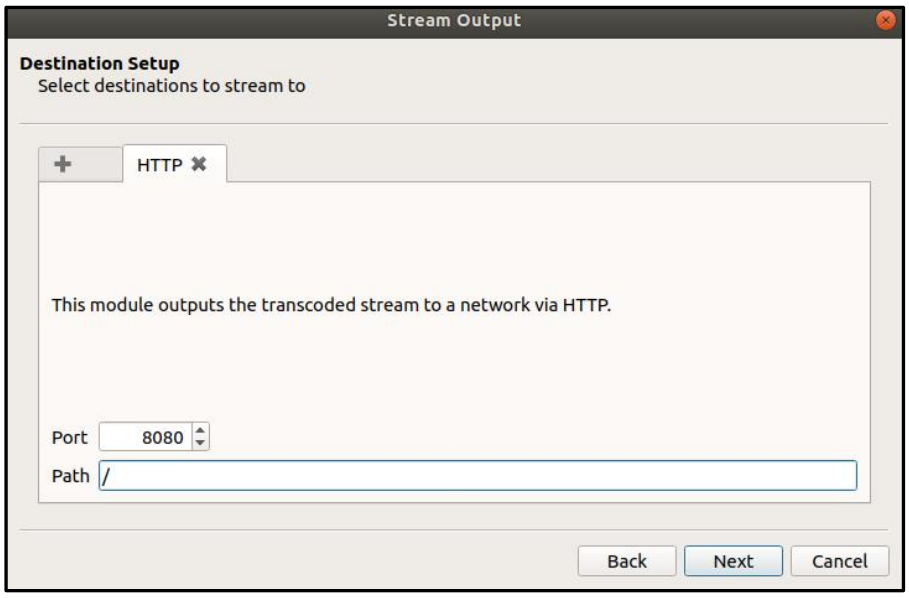
Seguidamente nos dijimos a opciones y seleccionamos Stream, añadimos el video que será transmitido VideoTest.mp4.



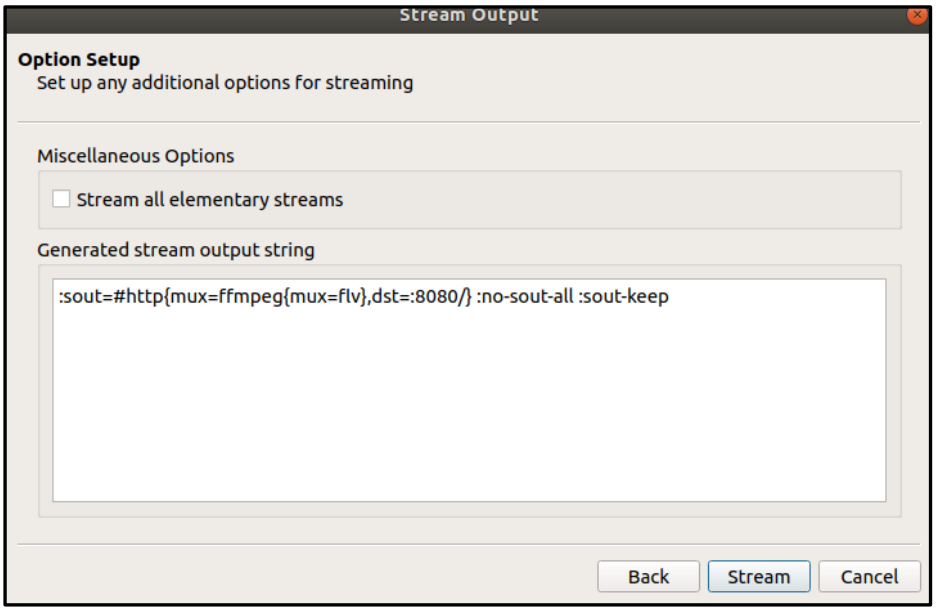
Se podría utilizar rtp, pero mediante Mininet no nos permite utilizar este protocolo; por ello añadimos http.



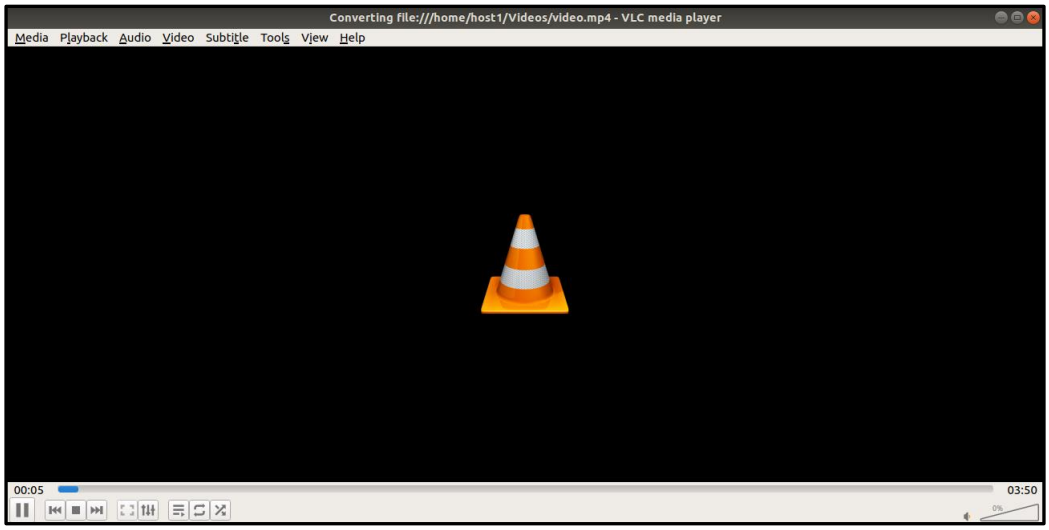
Y le indicamos que sea en el puerto 8080, y que la transmisión será desde el directorio raíz donde se encuentre el video.



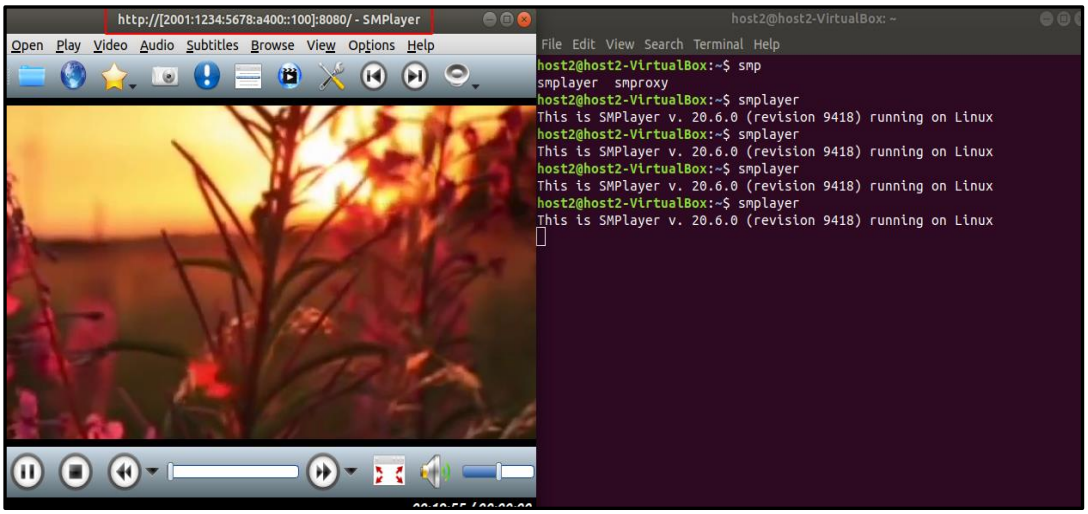
Finalmente se generará un segmento de parámetros que permite el Stream de acuerdo con la configuración realizada y le indicamos que empiece a transmitir.



Finalmente, empezar a transmitir en el host 1 no se verá el video, sino que es el cliente quien verá lo que se está transmitiendo.

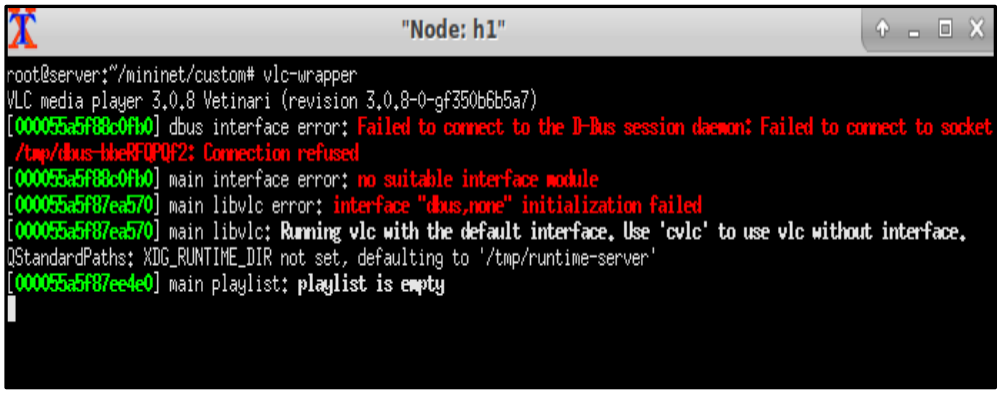


El cliente host2 debe conectarse al servidor host 1; para ello utilizaremos la herramienta smplayer la cual la iniciamos y nos conectamos mediante la opción browser e ingresamos el protocolo la ip y el puerto de conexión; que finalmente empezaremos a ver lo que se está transmitiendo en vivo.

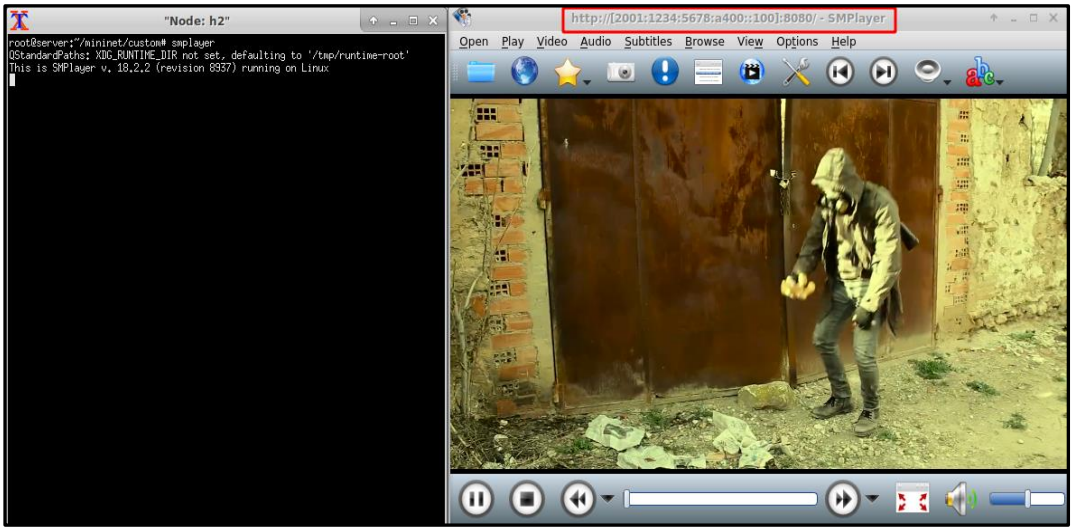


- ✓ **SDN**

En SDN prácticamente es lo mismo que en HDN; se emite desde host1 como servidor inicializando vlc-wrapper y realizando la misma configuración.



Finalmente, host2 se conecta host1 mediante la herramienta smplayer con el protocolo http la IPv6 y el puerto, podrá visualizar el video que se está transmitiendo en vivo desde el host1


"Node: h2"
root@server:~/mininet/custom# smplayer
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
This is SMPlayer v. 18.2.2 (revision 6937) running on Linux
http://[2001:1234:5678:a400::100]:8080/ - SMPlayer
Open Play Video Audio Subtitles Browse View Options Help

Anexo 6: Guía de Implementación de una arquitectura de red Avanzada.

Para implementar una red avanzada, debemos realizar un proceso de análisis de la topología; para ello si se optase de la manera tradicional se empezaría por ver los recursos y realizar la implementación en GNS3 u otra herramienta. Sin embargo, coincidiendo con base a los resultados obtenidos de la parte desarrollo en esta investigación, la implementación de una arquitectura de red avanzada que tiene un mayor rendimiento es de forma programable, es decir una red definida por software (SDN).

Por tanto, esta guía de implementación de una arquitectura de red avanzada está enfocada de manera programable (SDN), considerando que este escenario es realizado en un entorno de emulación. Por lo cual debemos seguir una secuencia de procesos.

1. Recursos computacionales

Para realizar la emulación debemos considerar los equipos que tengamos a disposición, podemos usar cualquier computador; en base a esto si solo se cuenta con un equipo, lo cual es muy limitante, se tendría que hacer correr todas las herramientas dentro del mismo; sin embargo, se recomienda al menos tener 2 equipos, tal como se ha desarrollado en esta investigación; uno para hacer funcionar la topología y otro para ejercer la función de control.

2. Definiendo herramientas:

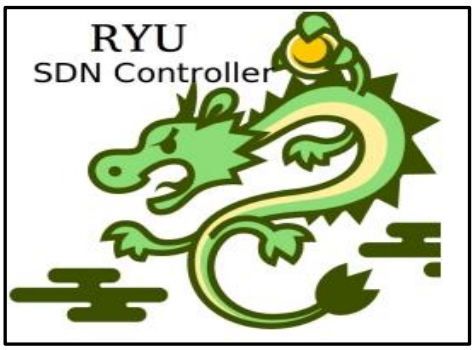
Si bien es cierto al principio, para definir las herramientas e ir aplicando pruebas a nivel de una arquitectura de red avanzada puede tomar tiempo ya que primeramente se hará una investigación múltiple, sin embargo, esta guía permitirá ya tener definido a modo genérico las herramientas que se detallan a continuación.

Herramienta de emulación:

No es complicado, elegir una herramienta que nos permita insertar programación de la red, ya que después de investigar, totalidad de los trabajos relacionados a SDN, hablan de Mininet como una herramienta de gran utilidad para emular redes definidas por software, ya que nos permite ejecutar scripts y nos provee clases y métodos que nos ayudan a programar la red.

Controlador:

Definir el controlador a utilizar puede tomar pruebas iniciales de hacer funcionar dicho controlador, podemos encontrar controladores SDN comerciales como el de Cisco, Juniper y otros, sin embargo en esta investigación se ha utilizado un controlador OpenSource por mencionar algunos que destacan OpenDayLight, FloodLight, Ryu y ONOS; se puede elegir trabajar con cualquiera; independientemente de cualquier complicación en la instalación de dicho controlador; todos cumplen la misma función de ejecutar la instancia de control para la red.



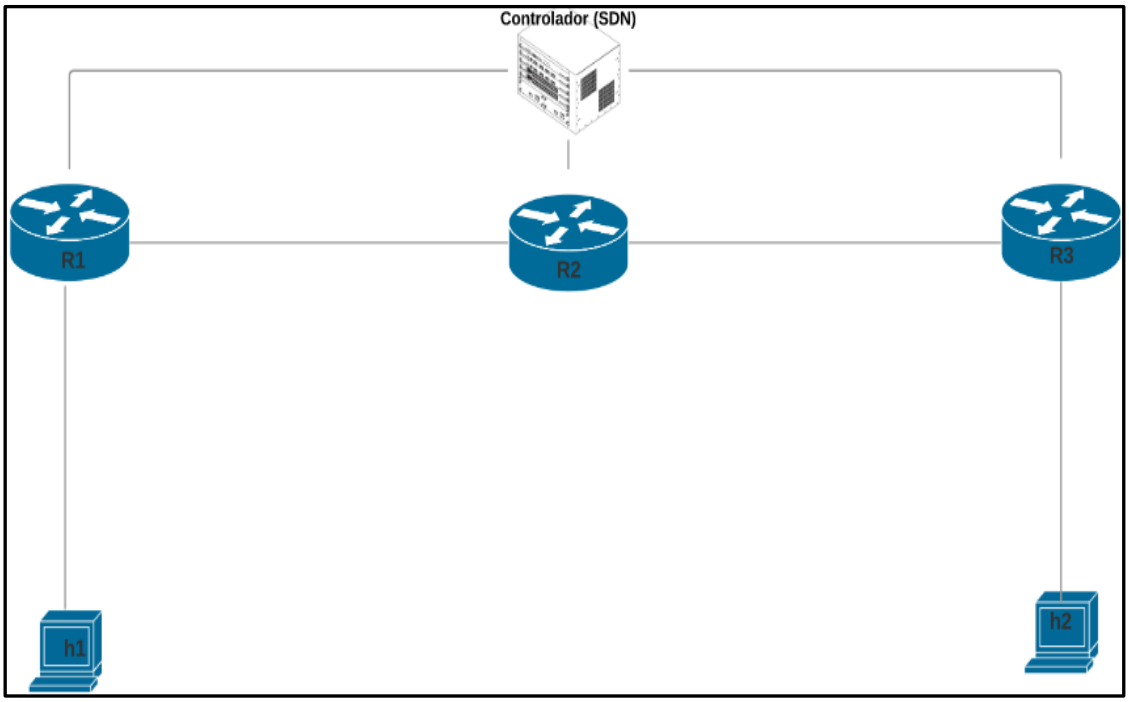
Herramienta para enrutamiento:

Cuando realizamos enrutamientos; considerando las emulaciones; en una red tradicional como HDN, sabemos que por medio del IOS cargado podemos utilizar los protocolos de enrutamiento ya sea dinámico o estático; para el caso de las redes programables como SDN; considerando que Mininet es OpenSource y están mejores adaptadas para funcionar bajo distribuciones GNU/Linux; además de haber definido utilizar Mininet; precisamos que se puede hacer enrutamientos; sin embargo debemos buscar soluciones que permitan hacer más efectivo los ruteos; por ello después de investigar, se ha dado por definido utilizar el proyecto Quagga, el cual nos permite utilizar protocolos de enrutamiento; debido a que en las redes avanzadas se define OSPF como el protocolo a utilizar por sus capacidad de trabajo de construcción de tablas de enrutamiento en base a

costos y por su algoritmo; considerando la v3 para IPv6; es precisamente Quagga el que nos permitirá utilizar OSPFv3, mediante el demonio zebra, el cual se encarga de manejar las tablas de enrutamiento.

3. Especificaciones de la Topología

Debemos tener el diseño de nuestra topología de la arquitectura de red avanzada que estemos implementando; sin embargo, para cumplir con el objetivo de esta guía, tomaremos como base la siguiente topología que ya ha sido probada para OSPFv3 inicialmente; la implementación de una red avanzada ya depende de la topología que estemos trabajando, solo debemos adecuarla.



Ya teniendo nuestra topología diseñada como tal, debemos considerar parámetros que serán utilizados dentro en la red; que permitirán cumplir con el objetivo de dicha implementación; estos parámetros son muy importantes y se describen a continuación.

Direcciones IPV6

Al ser una arquitectura de red avanzada debemos tener en claro los rangos de direcciones de direcciones IPv6 en cada enlace de los dispositivos; para ello se estima elaborar un cuadro de asignaciones de todas las direcciones IPv6, que se requiera en la topología de red que se está trabajando.

Router ID

También debemos considerar de la misma manera elaborar un cuadro de asignaciones de los router id, para asignar a cada uno de los router; también debemos decidir si vamos a trabajar con multitareas o con un solo para la configuración OSPFv3; aunque no hay necesidad de usar multitareas, por lo que se estima apropiado simplemente usar un área.

Enlaces

Es necesario tener una tabla o especificaciones donde se haga referencia a los nombres de cada enlace que existe en cada dispositivo; esto con el objetivo de que al momento de programar la red, sepamos ya que nombres asignar a cada enlace, ya que dichos nombres se llega a utilizar más de una vez y debemos precisar exactitud o existirán conflictos al ejecutar el script.

Teniendo estos parámetros ya establecidos, esto nos ayudará al momento de programar la red a tener una visualización y capacidad de asignación de los parámetros para no tener posibles confusiones.

4. Instalación de herramientas y configuración

Ya teniendo completa nuestra topología con los parámetros a asignar, procederemos a instalar las herramientas adecuadamente; sin embargo, el proceso de instalación puede diferir ya que podíamos haberlo realizado anteriormente; pero debemos considerar primeramente que entre la máquina donde opera Mininet junto a Quagga debe tener comunicación con el otro equipo donde se ejecuta el controlador; esto a través de una dirección IPv6. La instalación de dichas herramientas se encuentra dentro del contenido de

este documento; para instalar Mininet hay que seguir la secuencia de la página 52 de este documento y realizar el test de funcionamiento básico; asimismo en la página 56 de este documento encontramos la instalación del controlador Ryu, el cual como se ha manifestado puede ser otro controlador, pero debemos hacer el test del funcionamiento de dicho controlador; finalmente la instalación de Quagga la encontraremos dentro de la página 54 de este documento, donde debemos seguir la secuencia de pasos y finalmente ver que zebra y ospfd referente a ospfv3 estén corriendo en sus puertos respectivos como servicio, una herramienta que ayuda a verificar esto es Nmap.

5. Programando el Script

Para empezar a programar nuestra red, debemos tener en cuenta que todo script que se programe deberá ser ejecutados dentro de la carpeta de instalación de Mininet, el cual debe ser en la ruta Mininet/Custom; además se debe asignar un nombre a este archivo con la extensión .py, es decir tomando en cuenta que el nombre del archivo sea RedAvanzada; se tendría como RedAvanzada.py.

El script debe empezar por su naturaleza con un #!/usr/bin/python, esto simplemente para indicarle al sistema operativo que mediante el intérprete Shell sepa que es un archivo Python y lo pueda ejecutar. Seguidamente debemos importar las bibliotecas y objetos que nos proporciona Mininet el cual permitirá crear nuestra topología de red; importar dichas bibliotecas u objetos dependerá de las funcionalidades a realizar; en las páginas 57 y 58 de este documento se menciona los detalles y explicación de cada uno de ellos.

```
#!/usr/bin/python
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Node, Controller, RemoteController
from mininet.log import setLogLevel, info
from mininet.cli import CLI
import time
import os
```

Después de importar las librerías, es necesario llamar la clase LinuxRouter que permitirá hacer el reenvío de IP; en la página 58 de este documento se especifica la razón de utilizar esta clase; dentro de esta clase se debe utilizar los métodos config donde podemos

expresar el renvío de IP como estado 1 habilitado y en el método terminate le indicamos 0 como deshabilitado.

```
class LinuxRouter(Node):
    def config(self, **params):
        super(LinuxRouter, self).config(**params)
        self.cmd('sysctl -w net.ipv4.ip_forward=1')
        self.cmd('sysctl -w net.ipv6.conf.all.forwarding=1')
    def terminate(self):
        self.cmd('sysctl -w net.ipv4.ip_forward=0')
        self.cmd('sysctl -w net.ipv6.conf.all.forwarding=0')
        super(LinuxRouter, self).terminate()
```

Seguidamente debemos definir la clase ya propia de nuestro código, en donde incluiremos nuestro código para hacer funcionar una red avanzada como SDN; en el siguiente código observamos el nombre de nuestra clase GuiaRedAvSDN, la cual debemos hacer pasar el parámetro Topo, el cual es relacionado para poder construir la topología; asimismo es aquí donde definimos el método build con los parámetros self de enlace y sus opciones; definimos una variable como la queramos llamar, por mencionar VardefaultIP, la cual tomará como valor una dirección IP y su máscara de red, para ello debemos revisar el cuadro de asignaciones especificadas con las direcciones IPv6 para todos los puntos de nuestra arquitectura de red en implementación, consideremos que esta variable toma una dirección IP para una interface en cada router, y a partir de ahí crear las demás, por tanto creamos 3 de estas variables; también hay que tener en cuenta, que Mininet por si solo no acepta IPv6, por tanto para crear los enlaces los hacemos con IPv4, que finalmente solo será para dar paso a IPv6; esto se encuentra explicado en la página 59 de este documento.

```
class GuiaRedAvSDN(Topo):
def build(self, **_opts):
    VarDefaultIP1 = 'IP-1/Mascara'
    VarDefaultIP2 = 'IP-2/Mascara'
    VarDefaultIP2 = 'IP-3/Mascara'
```

Asimismo, es momento de añadir los routers, para ello definimos variables referentes a los routers; y mediante la propiedad self.addNode pasamos el nombre de cada router, la clase LinuxRouter y la IP que hemos designado como VardefaultIP, claramente esto en base a la topología que estemos implementando.

```
VarR1 = self.addNode('r1', cls=LinuxRouter, ip=VarDefaultIP1)
VarR2 = self.addNode('r2', cls=LinuxRouter, ip=VarDefaultIP2)
VarR3 = self.addNode('r3', cls=LinuxRouter, ip=VarDefaultIP3)
```

En caso de tener hosts extremos unidos a los routers extremos de la red; podemos de la misma manera añadirlos, con la propiedad self.addHost, donde también le pasamos su nombre, su dirección IPv4 que serían IP-4 e IP-5 y su máscara, además de incluir el Gateway que sería IP-6 y IP-7 para h1 y h2 respectivamente.

```
VarH1 = self.addHost('h1', ip='IP-4/Mascara', defaultRoute='via IP-6')
VarH2 = self.addHost('h2', ip='IP-5/Mascara', defaultRoute='via IP-7')
```

Seguidamente vamos a construir los enlaces que existen, para ello debemos revisar la tabla construida de especificaciones de estos parámetros, con la propiedad self.addLink, consideremos que los enlaces son entre dos nodos, para ello pasamos los parámetros que requiere como las variables designadas a cada router, el nombre del enlace que se está construyendo con la propiedad intfName; asimismo cuando le asignamos una dirección IP por defecto, es decir la IP de VarDefaultIP, esta IP es para una salida del router, si tenemos más enlaces en cada router corresponde asignar esas IP, para esta guía sería la IP-8 que corresponde a r2-r3 que no está como un defaultIP, ya que el default solo es uno, asimismo debemos también construir los enlaces con los host.

```
self.addLink(VarR1, VarR2, intfName1='r1-r2', intfName2='r2-r1')
self.addLink(VarR2, VarR3, intfName1='r2-r3',params1={ 'ip' : 'IP-8' ,intfName2='r3-r2')
self.addLink(h1, VarR1, intfName2='r1-h1', params2={'ip': 'IP-6/Mascara'})
self.addLink(h2, VarR3, intfName2='r3-h2', params2={'ip': 'IP-7/Mascara'})
```

A continuación, debemos definir un método que hace referencia a cuando ya se está ejecutando, donde podemos ir i incluyendo líneas que evidencien el proceso de ejecución, mediante la propiedad topo debemos incluir la clase que hemos creado GuiaRedAvSDN, que es básicamente la topología, y mediante la propiedad net le indicamos que existe un controlador remoto a través de la dirección IPv6, y su puerto 6633; es necesario incluir la propiedad net.start.

```
def run():
    topo = GuiaRedAvSDN()
    net = Mininet(controller=RemoteController, topo=topo)
    c1 = net.addController('c1', ipv6='DirIPv6Controlador', port=6633)
    net.start()
```

Es momento de incluir los nodos de la topología creada en la clase GuiaRedAvSDN en una nueva variable mediante la propiedad net.getNodeByName, esto es necesario porque posteriormente indicaremos que se utilice los protocolos de enrutamiento mediante la utilidad de quagga.

```
r1 = net.getNodeByName('r1')
r2 = net.getNodeByName('r2')
r3 = net.getNodeByName('r3')
r4 = net.getNodeByName('h1')
r5 = net.getNodeByName('h2')
```

Una vez definido estas variables, es por fin el momento de configurar IPv6 en los enlaces mediante la propiedad cmd que incluye parámetros de config como la Dirección IPv6 y su máscara.

```
r1.cmd('ifconfig r1-r2 inet6 add IPv6-1/Mascara')
r2.cmd('ifconfig r2-r1 inet6 add IPv6-2/Mascara')
r2.cmd('ifconfig r2-r3 inet6 add IPv6-3/Mascara')
r3.cmd('ifconfig r3-r2 inet6 add IPv6-4/Mascara')
```

También debemos hacer lo mismo con los enlaces de los hosts, pero adicionalmente después de añadir las IPv6 debemos indicarle su Gateway de salida que serían las direcciones IPv6-5 y IPv6-7 respectivamente de h1 y h2.

```
r1.cmd('ifconfig r1-h1 inet6 add IPv6-5/Mascara')
h1.cmd('ifconfig h1-r1 inet6 add DirIPv6-6/Mascara')
h1.cmd('route -6 add default gw IPv6-5 dev h1-r1')
r3.cmd('ifconfig r3-h2 inet6 add IPv6-7/Mascara')
h2.cmd('ifconfig h2-r3 inet6 add IPv6-8/Mascara')
h2.cmd('route -6 add default gw IPv6-7 dev h2-r3')
```

A continuación, debemos crear los sockets, para ello llamaremos a los archivos de zebra que crearemos posteriormente mediante la ruta donde hemos instalado, y la creación del socket se dará con un archivo .api y un .interface del archivo zebra, debemos indicarle la ruta donde debe crearse.

```
r1.cmd('zebra -f ../r1zebra.conf -d -z ~../r1zebra.api -i ~../r1zebra.interface')
time.sleep(1)
r2.cmd('zebra -f ../r2zebra.conf -d -z ~../r2zebra.api -i ~../r2zebra.interface')
r3.cmd('zebra -f ../r3zebra.conf -d -z ~../r3zebra.api -i ~../r3zebra.interface')
```

Ahora es tiempo de utilizar los archivos OSPFv3 de cada router que crearemos posteriormente, esto mediante la capacidad del api creado por zebra; debemos crear un .interface de cada uno de estos archivos ospfv3 ya que estos contendrán los parámetros que requiere ospfv3

```
    r1.cmd('ospf6d -f ../r1ospf6d.conf -d -z ~../r1zebra.api -i ~../r1ospf6d.interface')
    r2.cmd('ospf6d -f ../r2ospf6d.conf -d -z ~../r2zebra.api -i ~../r2ospf6d.interface')
    r3.cmd('ospf6d -f ../r3ospf6d.conf -d -z ~../r3zebra.api -i ~../r3ospf6d.interface')
```

En la parte final del script debemos incluir el siguiente segmento de código, para que pueda funcionar adecuadamente el script.

```
    CLI(net)
    net.stop()
    os.system("killall -9 ospfd ospf6d zebra")
    os.system("rm -f *api*"")
    os.system("rm -f *interface*"")

if __name__ == '__main__':
    setLogLevel('info')
    run()
```

Finalmente hemos terminado de programar la red; sin embargo, aún falta realizar otras configuraciones.

6. Configuración de Archivos Zebra y OSPF

Para esto, debemos revisar nuestra tabla de asignaciones de los enlaces de la red; en este caso definiremos de la siguiente forma genérica para poder programar la red. En h1 tenemos el enlace con el router r1, el cual es una interface, a este lo definimos como h1-r1; y de esta manera siguiendo este patrón de definición de nombre tenemos que, en h2 sería h2-r3; en r1 tendremos r1-h1 y r1-r2; en r2 los enlaces son r2-r1 y r2-r3; en r3 tenemos r3-r2 y r3-h2.

Entonces como ya tenemos Quagga instalado correctamente, debemos crear los archivos zebra y OSPFv3 que han sido llamados en el script al momento de programar la red; el archivo zebra debe incluir el siguiente segmento de código, que nos trae por defecto, en realidad solo se utiliza los parámetros de la conexión de zebra, para que pueda utilizar los demás protocolos de enrutamiento.

```
! zebra sample configuration file
! $Id: zebra.conf.sample,v 1.1 2002/12/13 20:15:30 paul Exp $
hostname Router
password zebra
enable password zebra
! Interface's description.
!interface lo
! description test of desc.
!interface sit0
! multicast
! Static default route sample.
!ip route 0.0.0.0/0 203.181.89.241
!log file zebra.log
```

Es necesario crear los archivos ospf6d en referencia al protocolo ospfv3 para cada router de nuestra topología en implementación, en nuestra guía claramente consta de 3 router, en tal sentido en esta guía mostraremos el archivo ospfv3 para r1; los demás archivos son similares, con la diferencia de cambiar las direcciones IPv6 su router id y los nombres de las interfaces que hay en los en los enlaces entre los distintos nodos de la topología, estos parámetros también son explicados dentro de la página 67 de este documento.

```
hostname r2
password zebra
log stdout
service advanced-vty
debug ospf6 neighbor state

interface r2-r1
 ipv6 ospf6 cost 1
 ipv6 ospf6 hello-interval 10
 ipv6 ospf6 dead-interval 40
 ipv6 ospf6 retransmit-interval 5
 ipv6 ospf6 priority 1
 ipv6 ospf6 transmit-delay 1
 ipv6 ospf6 instance-id 0

interface r2-r3
 ipv6 ospf6 cost 1
 ipv6 ospf6 hello-interval 10
 ipv6 ospf6 dead-interval 40
 ipv6 ospf6 retransmit-interval 5
 ipv6 ospf6 priority 1
 ipv6 ospf6 transmit-delay 1
 ipv6 ospf6 instance-id 0

interface lo0
 ipv6 ospf6 cost 1
 ipv6 ospf6 hello-interval 10
 ipv6 ospf6 dead-interval 40
 ipv6 ospf6 retransmit-interval 5
 ipv6 ospf6 priority 1
 ipv6 ospf6 transmit-delay 1
 ipv6 ospf6 instance-id 0

router ospf6
 router-id 2.2.2.2
! redistribute static route-map static-ospf6
 interface r2-r1 area 0.0.0.0
 interface r2-r2 area 0.0.0.0

access-list access4 permit 127.0.0.1/32
ipv6 access-list access6 permit 3ffe:501::/32
ipv6 access-list access6 permit 2001:1::/48
ipv6 access-list access6 permit ::1/128
ipv6 prefix-list test-prefix seq 1000 deny any

route-map static-ospf6 permit 10
 match ipv6 address prefix-list test-prefix
 set metric-type type-2
 set metric 2000

line vty
 access-class access4
 ipv6 access-class access6
 exec-timeout 0 0

log file /usr/local/etc/antofagospf6d.log
```

7. Verificando Conectividad

Después de ya haber realizado todos los procesos mencionados, en este punto ya tenemos programado el script y configurado los archivos zebra y opsf6d para ospfv3, por lo que solo nos queda ejecutar el script y verificar el función amiento adecuado; debemos tener ejecutándose el demonio zebra y osp6d referente a ospfv3; es primordial verificar que se creen los sockets dentro de la ruta que se ha asignado; en caso de que estos no se creen debemos verificar la ruta de los archivos zebra, y de los sockets que sean adecuadamente, hay que considerar que Python es sensible a cualquier diferencia de caracteres o espacios; también podemos verificar los logs que se colocaron en los archivos ospf6d y zebra, así como también dentro del script podemos poner logs que nos ayuden a ver el funcionamiento paso a paso.