UNIVERSIDAD NACIONAL DE CHIMBORAZO
FACULTAD DE INGENIERÍA
CARRERA DE TELECOMUNICACIONES

RENDIMIENTO DE UNA RED DEFINIDA POR SOFTWARE (SDN) Y UNA RED TRADICIONAL DE DATOS EN APLICACIONES DE VIDEOJUEGOS EN LÍNEA.

Proyecto de investigación previo a la obtención del título de Ingeniero en Telecomunicaciones

Autor:
Jhonnatan David Castañeda Valdiviezo

Tutor:
Msc. Eduardo Daniel Haro Mendoza

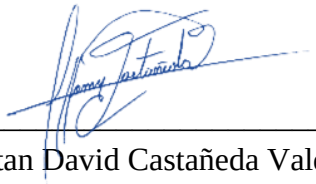
Riobamba, Ecuador. 2023

DECLARATORIA DE AUTORÍA

Yo, **Jhonnatan David Castañeda Valdiviezo**, con cédula de ciudadanía **0605428671**, autor del trabajo de investigación titulado: "**RENDIMIENTO DE UNA RED DEFINIDA POR SOFTWARE (SDN) Y UNA RED TRADICIONAL DE DATOS EN APLICACIONES DE VIDEOJUEGOS EN LÍNEA**", certifico que la producción, ideas, opiniones, criterios, contenidos y conclusiones expuestas son de mí exclusiva responsabilidad.

Asimismo, cedo a la Universidad Nacional de Chimborazo, en forma no exclusiva, los derechos para su uso, comunicación pública, distribución, divulgación y/o reproducción total o parcial, por medio físico o digital; en esta cesión se entiende que el cesionario no podrá obtener beneficios económicos. La posible reclamación de terceros respecto de los derechos de autor (a) de la obra referida, será de mi entera responsabilidad; librando a la Universidad Nacional de Chimborazo de posibles obligaciones.

En Riobamba, 20 de diciembre del 2023.



Jhonnatan David Castañeda Valdiviezo
C.I: 0605428671

DICTAMEN FAVORABLE DEL PROFESOR TUTOR

Quien suscribe, **Eduardo Daniel Haro Mendoza** catedrático adscrito a la Facultad de Ingeniería por medio del presente documento certifico haber asesorado y revisado el desarrollo del trabajo de investigación titulado "**RENDIMIENTO DE UNA RED DEFINIDA POR SOFTWARE (SDN) Y UNA RED TRADICIONAL DE DATOS EN APLICACIONES DE VIDEOJUEGOS EN LÍNEA"**, bajo la autoría de **Jhonnatan David Castañeda Valdiviezo;** por lo que se autoriza ejecutar los trámites legales para su sustentación.

Es todo cuanto informar en honor a la verdad; en Riobamba, a los 20 días del mes de noviembre del 2023.

Firmado electrónicamente por:
EDUARDO DANIEL
HARO MENDOZA

Mgs. Eduardo Daniel Haro Mendoza
C.I: 0603387531

CERTIFICADO DE LOS MIEMBROS DEL TRIBUNAL

Quienes suscribimos, catedráticos designados Miembros del Tribunal de Grado para la evaluación del trabajo de investigación **"RENDIMIENTO DE UNA RED DEFINIDA POR SOFTWARE (SDN) Y UNA RED TRADICIONAL DE DATOS EN APLICACIONES DE VIDEOJUEGOS EN LÍNEA"**, por **Jhonnatan David Castañeda Valdviezo**, con cédula de identidad número **0605428671**, bajo la tutoría de **Mgs. Eduardo Daniel Haro Mendoza**; certificamos que recomendamos la **APROBACIÓN** de este con fines de titulación. Previamente se ha evaluado el trabajo de investigación y escuchada la sustentación por parte de su autor; no teniendo más nada que observar.

De conformidad a la normativa aplicable firmamos, en Riobamba a los 20 días de diciembre de 2023.

Presidente del Tribunal de Grado
Mgs. Deysi Vilma Inca Balseca — Firma

Miembro del Tribunal de Grado
Dr. Klever Hernán Torres Rodríguez — Firma

Miembro del Tribunal de Grado
Ing. Alejandra del Pilar Pozo Jara — Firma

CERTIFICACIÓN

Que, **Castañeda Valdiviezo Jhonnatan David** con CC: **0605428671**, estudiante de la Carrera **Ingeniería en Telecomunicaciones,** Facultad de **Ingeniería**; ha trabajado bajo mi tutoría el trabajo de investigación titulado "**RENDIMIENTO DE UNA RED DEFINIDA POR SOFTWARE (SDN) Y UNA RED TRADICIONAL DE DATOS EN APLICACIONES DE VIDEOJUEGOS EN LÍNEA**", cumple con el **0%**, de acuerdo al reporte del sistema Anti plagio **URKUND**, porcentaje aceptado de acuerdo a la reglamentación institucional, por consiguiente autorizo continuar con el proceso.

Riobamba, 18 de diciembre de 2023

Firmado electrónicamente por:
EDUARDO DANIEL HARO MENDOZA

Mgs. Eduardo Daniel Haro Mendoza
TUTOR TRABAJO DE INVESTIGACIÓN

DEDICATORIA

Dedico el presente trabajo de investigación, a mis padres y hermanos, ya que siempre fueron y serán mi mejor apoyo moral y emocional.

AGRADECIMIENTO

Agradezco de todo corazón a mi virgencita de Agua Santa, por brindarme cada día la oportunidad de compartir nuevas experiencias con mi familia.

A mis amigos de videojuegos, ya que en mis momentos más solitarios son mi compañía, y gran parte de ellos contribuyeron a la realización de este trabajo de investigación.

A mi tutor de tesis Msc. Eduardo Daniel Haro Mendoza, por la paciencia, ayuda y guía brindada para la realización de este trabajo de investigación.

A mi directora de carrera Ing. Deysi Inca y docente de apoyo Ing. Verónica Aguilar, por siempre apoyarme y alentarme a completar nuevas metas académicas, y personales.

A la comunidad de Waifus Informáticas, por compartir experiencias, conocimientos y opiniones sobre varios temas de redes e informática.

A mis maestros, por las enseñanzas académicas y de vida.

ÍNDICE GENERAL

DECLARATORIA DE AUTORÍA
DICTAMEN FAVORABLE DEL PROFESOR TUTOR
CERTIFICADO DE LOS MIEMBROS DEL TRIBUNAL
CERTIFICADO ANTIPLAGIO
DEDICATORIA
AGRADECIMIENTO
ÍNDICE GENERAL
ÍNDICE DE TABLAS
ÍNDICE DE FIGURAS
RESUMEN
ABASTRACT
CAPÍTULO I 16
1.1. Introducción 16
1.2. Planteamiento del problema 18
1.3. Justificación 19
1.4. Objetivos 19
1.4.1. General 19
1.4.2. Específicos 19
CAPÍTULO II 20
2.1. Evolución de las comunicaciones digitales 20
2.2. Conceptualización y particularidades de las SDN 22
2.2.1. Arquitectura de una red definida por software 23
2.2.2. Herramientas más usadas para simular y/o desplegar una arquitectura SDN 24
2.3. Evolución de los videojuegos 27
2.3.1. Los deportes electrónicos o E-sports 30
2.3.2. Los E-Sports y su relación con las redes de datos 30
2.4. Funcionamiento de un videojuego en línea 31
2.4.1. Etapas de conexión de un videojuego en línea 32
2.5. Topologías de red y sus tipos de demora 35

CAPÍTULO III 37
3.1. Metodología 37
3.1.1. Tipo de investigación 37
3.1.2. Métodos, técnicas e instrumentos de investigación 37
3.1.3. Procedimiento 37
3.1.4. Población y muestra 38
3.1.5. Operacionalización de las Variables 39
3.2. Entorno virtual de pruebas 39
3.2.1. Topología de red 40
3.2.2. Simulación de partidas en línea de CS:GO con Python 45
CAPÍTULO IV 48
4.1. Resultados y Discusiones 48
4.1.1. Despliegue del entorno virtual de pruebas con tecnología SDN 48
4.1.2. Medida del RTT y Jitter en el entorno virtual de pruebas con tecnología de red tradicional 50
4.1.3. Medida del RTT y Jitter en el entorno virtual de pruebas con tecnología SDN .56
4.2. Análisis del RTT y Jitter, tras el cambio de tecnología de red tradicional con tecnología SDN en el entorno virtual de pruebas 61
4.2.1. Prueba de normalidad 61
4.2.2. Pruebas de hipótesis 62
CAPÍTULO V 69
5.1. Conclusiones 69
5.2. Recomendaciones 70
BIBLIOGRAFÍA 71
ANEXOS 74

ÍNDICE DE TABLAS

Tabla 1: Operacionalización de variables....39

Tabla 2: RTT promedio desde cada videojugador hacia el servidor de CS:GO con la IPv4 162.254.199.178 40

Tabla 3: Cálculo de la demora física y RTT físico de forma teórica....41

Tabla 4: Cálculo del RTT lógico de forma teórica....42

Tabla 5: RTT lógico simulado para cada número de saltos de red....42

Tabla 6: Número de saltos necesarios para aproximar el RTT lógico de la red al calculado en la Tabla 4 43

Tabla 7: RTT promedio simulado de la topología propuesta 45

Tabla 8: RTT y Jitter promedio de las 200 muestras de cada videojugador del entorno virtual de pruebas con tecnología de red tradicional 54

Tabla 9: RTT y Jitter promedio de las 200 muestras de cada videojugador del entorno virtual de pruebas con tecnología SDN 59

ÍNDICE DE FIGURAS

Figura 1: Esquematización básica de una red tradicional de datos 22
Figura 2: Esquematización básica de una red definida por software 23
Figura 4: Interacción entre un controlador SDN y un conmutador OpenFlow 25
Figura 5: Cromo Titan (Holo) | Katowice 2014 29
Figura 6: Estructura cliente – servidor 31
Figura 7: Resultado de RTT desde el videojugador a todos los servidores disponibles, generado por la consola de CS:GO 32
Figura 8: Parámetros iniciales del cliente para establecer una conexión con el servidor de partidas multijugador en línea 33
Figura 9: Registro de conexión del videojugador al servidor del juego mediante sockets, generado por la consola de CS:GO.....33
Figura 10: Métricas de red visualizadas en CS:GO mediante el comando "net_grap 1".....34
Figura 11: Captura del tráfico paquetes de datos en Wireshark, generado al entrar en una partida real de CS:GO.....34
Figura 12: Captura del tráfico de paquetes visualizado en Wireshark usando su función I/O Graph, generado al entrar en una partida real de CS:GO 34
Figura 13: Prueba de RTT desde Riobamba – Ecuador hacia el servidor de CS:GO ubicado en Atlanta – EE.UU con la dirección IP 162.254.199.178.....35
Figura 14: Mapa de la fibra óptica marítima mundial escalado a tamaño real con Google Earth 41
Figura 15: Topología de red usada para el entorno virtual de pruebas, construida a partir de cálculos y estimaciones de la topología con tecnología de red tradicional real usada para partidas en línea del videojuego CS:GO 43
Figura 16: Topología con tecnología de red tradicional desplegada con GNS3 para simular el comportamiento de la topología de red real usada en partidas en línea del videojuego CS:GO 44
Figura 17: Topología de red con un RTT aproximado de 0ms 46
Figura 18: Prueba de RTT desde H1 a H11, para la topología de red de la Figura 17.....46
Figura 19: Captura del tráfico paquetes de datos en Wireshark, generado al ejecutar el script 47
Figura 20: Captura del tráfico de paquetes visualizado en Wireshark usando su función I/O Graph, generado al ejecutar el script 47

Figura 21: Topología con tecnología SDN realizada con MiniEdit para simular partidas en línea del videojuego CS:GO 48
Figura 22: Inicialización del controlador OpenDaylight 49
Figura 23: Inicialización del script 49
Figura 24: Prueba de conexión mediante el uso del comando “pingall” 50
Figura 25: Topología de red del script 50
Figura 26: Registro del funcionamiento del script 51
Figura 29: Finalización del servidor en el entorno virtual de pruebas con tecnología de red tradicional 52
Figura 30: Archivos de texto generados por cada videojugador del entorno virtual de pruebas con tecnología de red tradicional 53
Figura 31: Contenido de los archivos de texto de la Figura 30 53
Figura 32: Paquetes enviados por segundo desde H1 durante la ejecución de los scripts 55
Figura 33: Paquetes enviados por segundo desde H11 hacia H1 durante la ejecución de los scripts 55
Figura 34: Registro del funcionamiento del script 56
Figura 35: Registro del funcionamiento del script 56
Figura 36: Registro del tráfico UDP generado a partir de la ejecución los scripts 57
Figura 37: Finalización del servidor en el entorno virtual de pruebas con tecnología SDN 57
Figura 39: Contenido de los archivos de texto de la Figura 38 58
Figura 40: Paquetes enviados por segundo desde H10 durante la ejecución de los scripts 60
Figura 41: Paquetes enviados por segundo desde H11 hacia H10 durante la ejecución de los scripts cliente_videojugadores.py y servidor_partidas_en_línea.py en el entorno virtual de pruebas con tecnología SDN 60
Figura 42: Prueba de normalidad para el RTT de la topología de red real y para el RTT de la topología de red simulada 61
Figura 43: Prueba de normalidad el RTT y Jitter de cada entorno virtual de pruebas 62
Figura 44: Prueba T de Student para muestras independientes del RTT obtenido en la topología de red real usada para partidas en línea de CS:GO y el RTT obtenido al usar la misma topología de red, pero simulada en GNS3 63
Figura 45: Gráfico con barras de error del intervalo de confianza del RTT obtenido en la topología de red real usada para partidas en línea de CS:GO y RTT obtenido al usar la misma topología de red, pero simulada en GNS3 64

Figura 46: Prueba T de Student para muestras relacionadas del RTT obtenido en el entorno virtual de pruebas con tecnología de red tradicional y RTT obtenido en el entorno virtual de pruebas con tecnología SDN 65

Figura 47: Gráfico con barras de error del intervalo de confianza del RTT obtenido en el entorno virtual de pruebas con tecnología de red tradicional y RTT obtenido en el entorno virtual de pruebas con tecnología SDN 66

Figura 48: Prueba T de Student para muestras relacionadas del RTT obtenido en el entorno virtual de pruebas con tecnología de red tradicional y RTT obtenido en el entorno virtual de pruebas con tecnología SDN 67

Figura 49: Gráfico con barras de error del intervalo de confianza del Jitter obtenido en el entorno virtual de pruebas con tecnología de red tradicional y Jitter obtenido en el entorno virtual de pruebas con tecnología SDN 68

RESUMEN

En la nueva era digital, muchos mercados y tecnologías en línea se crean, modifican y evolucionan en cortos períodos de tiempo, ya que son desarrolladas a partir de conocimiento de fácil acceso. Sin embargo, existen tecnologías como las redes tradicionales de datos, que, por privatización del mercado, son sentenciadas a no innovar con el paso del tiempo, dificultando cada vez más su integración y correcto funcionamiento.

Organizaciones sin fines de lucro, como la Open Networking Foundation y Linux Foundation fomentan la investigación sobre una tecnología de red alternativa a la tradicional, llamada tecnología de red definida por software, la cual tiene el propósito de solventar la mayoría de deficiencias de las redes tradicionales de datos, como su baja flexibilidad, baja escalabilidad, falta de innovación y dificultad en la administración.

En la actualidad, uno de los mercados en línea más exigentes en cuanto a calidad de servicio es el de los videojuegos, ya que gran parte de su éxito depende de la buena experiencia de usuario, y para lograrlo necesitan que la red de datos ofrezca latencias bajas y estabilidad en la conexión.

Con la finalidad de entender a profundidad la interacción de un videojuego con las redes de datos, se analizará de forma teórica y práctica el rendimiento de una red definida por software (SDN), y una red tradicional de datos en aplicaciones de videojuegos en línea, mediante la simulación de dos entornos virtuales que reflejarán su comportamiento en un entorno real. De ambos entornos virtuales se recolectarán datos temporales con la finalidad de evidenciar de forma estadística si el cambio de tecnología de red mejorará la experiencia de usuario en los videojuegos en línea.

Palabras clave: Evolución, Red tradicional de datos, Red definida por software, Videojuegos en línea, Counter Strike: Global Offesive, Latencia.

ABASTRACT

In the new digital era, many online markets and technologies are created, modified, and evolved quickly since they are developed from easily accessible knowledge. However, there are technologies such as traditional data networks, which, due to market privatization, are sentenced to only innovate over time, making their integration and proper functioning increasingly more challenging.

Non-profit organizations such as the Open Networking Foundation and the Linux Foundation are promoting research into an alternative network technology to the traditional one, called software-defined network technology, which wants to solve most of the shortcomings of conventional data networks, such as their low flexibility, low scalability, lack of innovation and difficulty in administration.

Currently, one of the most demanding online markets in terms of quality of service is that of video games since a large part of their success depends on a good user experience, and to achieve this, they need the data network to offer low latency and connection stability.

To understand the interaction of a video game with data networks, the performance of a software-defined network (SDN) and a traditional data network in online video game applications will be analyzed theoretically and practically by simulating two virtual environments that will reflect their behavior in a natural environment. Temporal data will be collected from both virtual environments to show statistically whether the change in network technology will improve the user experience in online video games.

Keywords: Evolution, Traditional Data Network, Software Defined Network, Online Video Games, Counter-Strike: Global Offensive, Latency

Reviewed by:
Mgs. Maria Fernanda Ponce
ENGLISH PROFESSOR
C.C. 0603818188

CAPÍTULO I

1.1.Introducción

Desde la creación del internet como tal, el mundo ha experimentado una completa transformación, ya que gracias a tecnologías emergentes el ser humano ha sido capaz de eliminar barreras físicas para estar interconectado mediante redes de datos a una "era digital". Dando inicio a nuevas tendencias que digitalizan la mayor parte de las comunicaciones, e interacciones sociales, acto que requiere el transporte de paquetes de datos por medios digitales con una buena calidad de servicio de red. La prestación de un buen servicio entre los participantes de la comunicación digital bidireccional, requiere que cada uno de los equipos conformantes de la infraestructura de red estén configurados, administrados y trabajando coordinadamente entre sí, para reducir el tiempo de respuesta y garantizar una estabilidad en la conexión; llevar a cabo todo este proceso es cada vez más complicado para los administradores de red debido a la constante expansión de infraestructura de red y que cada uno de los equipos que la componen son una caja negra, ya que los diferentes fabricantes han privatizado el mercado mediante un software propietario (Firmware) que define la lógicas funcionamiento de cada equipo de la red, haciendo que para cada equipo de distinto fabricante la forma de configuración varie. Condenando a toda la infraestructura de red a una escaza flexibilidad en cuanto a innovación, ya que pequeños cambios acarrean una gran cantidad de problemas que conllevan costos elevados y requieren largos períodos de tiempo para ser ejecutados. En este contexto, se presenta un cambio de paradigma en las redes de datos, llamado redes definidas por software o SDN (por sus siglas en inglés de Software Defined Networks) que agilizan el proceso de gestión, control, escalabilidad y además brindan tiempos de respuesta (latencias) muy bajos en la red [1].

Mientras las redes de datos crecían, otro mercado que estaba ansioso por revolucionar la forma de ver el mundo florecía, es así como entre la década de los 80's y 90's grandes compañías a cargo de consolas de videojuegos como Sega, Nintendo y Atari realizaron varios prototipos que utilicen conexión a internet para lograr que sus videojuegos sean de multijugador en línea, y además brindar servicios como el correo electrónico o transferencias bancarias. Entre los proyectos más ambiciosos se encontraba el "Master Module" lanzado en 1983 para la Atari 2600 [2] o el "Mega Modem" lanzado en 1990 para la Sega Mega Drive [3], los cuales eran módems que encajan en el puerto de cartuchos de las consolas, y mediante un servicio de suscripción mensual en la red telefónica o de TV cableada permitían el acceso a la descarga de videojuegos alojados en un servidor, sin embargo, no fueron factibles con el tiempo, debido a las limitaciones en las consolas. Con los rotundos fracasos en los servicios de suscripción, en 1996 llegó el "Sega NetLink" un módem de 28.8 kbit/s que solamente requería una línea telefónica y el mismo videojuego para que los videojugadores puedan acceder a una misma partida de forma remota (modo multijugador en línea), además ofrecía conexión a internet, un navegador web y servicio de correo electrónico [4].

Inicialmente las desarrolladoras planteaban que el futuro de los videojuegos eran las consolas, ya que a diferencia de los demás dispositivos de la época enfocaban todo su hardware en esta tarea, sin embargo, hoy en día podemos encontrar este mercado expandido en todo tipo dispositivos (laptops, ordenadores de mesa, celulares, consolas, etc) y servicios, teniendo millones y millones de usuarios en múltiples plataformas de videojuegos conectados y jugando de forma simultánea en línea.

En la última década, el mercado de los videojuegos en línea ha ido creciendo de forma casi lineal, y no es hasta el 2020 con la aparición de la pandemia de covid-19 cuando este mercado toma realmente una importancia imprescindible, ya que la cantidad de jugadores en línea crece de forma abrupta, por ejemplo, solamente en el caso de la plataforma de videojuegos de Steam desde el 23 de diciembre del 2019 al 30 de marzo del 2020 (pocos días después de que el covid-19 sea declarado oficialmente como una pandemia [5]) paso de tener 17.955.645 usuarios activos y 5.729.658 jugadores simultáneos, a tener 25.535.923 usuarios activos y 8.171.592 jugadores activos, creciendo más de 7.5 millones de usuarios en 98 días, récord que casi fue alcanzado el 2 de enero de este año 2023 con 33.078.963 usuarios activos y 10.284.568 jugadores simultáneos [6]. Debido a la inminente creciente cantidad de videojugadores en línea, cada vez es más importante evolucionar las redes tradicionales de datos a tecnologías que sean fácilmente escalables y no requieran dejar sin servicio a sus usuarios cada vez que se realice un cambio, ya que muchas empresas que sustentan su economía o funcionamiento en productos y servicios en línea dependen en su totalidad de un buen servicio de red entre sus servidores y sus usuarios. Razón por la cual, se realizará un estudio comparativo entre una red tradicional y una SDN utilizando software de simulación de código abierto (del inglés open source). Los resultados del estudio servirán de base para futuros trabajos aplicativos que requieran redes de datos con una muy baja latencia, como los videojuegos en línea, la telemedicina, operación de maquinaria de forma remota, conducción de vehículos autónomos, internet de las cosas (IoT), entre muchas otras aplicaciones existentes y futuras.

1.2. Planteamiento del problema

Desde los inicios del internet, las redes de datos que lo conforman han sido implementadas con tecnología de red tradicional, centrando su funcionamiento en el procesamiento (plano de control) y reenvío (plano de datos) de paquetes de datos, en cada uno de los dispositivos (nodos) de red[7], utilizando pequeñas fracciones de segundo (usualmente milisegundos) para ser ejecutadas correctamente. Hace 20 años, las redes de datos ofrecían al internet una buena calidad de servicio, ya que la mayoría de mercados centraban sus servicios en conexiones que no demandaban mucho ancho de banda o latencias bajas. Sin embargo, en los últimos años, con el auge de mercados como los videojuegos en línea o streaming de video, es cada vez más complicado mantener una buena calidad de servicio, por la misma lógica de funcionamiento de la tecnología de red tradicional.

Otro de los factores fundamentales de las redes de datos son los medios de transmisión, encargados de transportar información entre los dispositivos de red. De los medios de transmisión más utilizados en las últimas décadas por los proveedores de servicio de internet (ISP) tenemos el cable de par trenzado, cable coaxial y la radiofrecuencia (RF), siendo estos medios muy propensos a fallas mecánicas, generando inestabilidad en la conexión y latencias altas. Actualmente los ISP Ecuatorianos ofrecen latencias más bajas en comparación a años anteriores, gracias a la adopción de la fibra óptica, teniendo así tiempos de ida - vuelta (RTT) de un paquete de datos entre un cliente y un servidor, oscilante entre 40 y 100ms según varias pruebas realizadas en varias ciudades Ecuatorianas dentro de videojuegos en línea con servidores alojados en la Costa Este de los Estados Unidos, logrando así una experiencia aceptable para partidas casuales en línea, sin embargo, para entrenamientos o partidas competitivas de videojugadores profesionales se debe tener extremo cuidado con el RTT apuntando a un valor ideal de 0ms. Pese al gran avance que supone la adopción de la fibra óptica como medio de transmisión más popular, lograr un RTT tan bajo para llevar a cabo partidas competitivas entre diferentes regiones del mundo aún no es factible, debido a la lógica de funcionamiento en sí de las redes tradicionales de dato, ya que por cada nodo que un paquete de datos atraviese se añade latencia.

La parte física de una red de datos está optimizada casi en su totalidad por la fibra óptica, ya que transporta paquetes de datos casi a la velocidad de la luz, quedándonos así únicamente la parte lógica del funcionamiento de la arquitectura de red. Bajo esta necesidad de optimización a nivel lógico en el funcionamiento de una red de datos se presentan las redes definidas por software, como alternativa de funcionamiento a las redes tradicionales de datos.

El tener optimizada la parte física y lógica del funcionamiento de una red de datos desemboca en latencias aún más bajas, mejorando así la experiencia de un videojugador en línea.

1.3. Justificación

Nuestro trabajo de investigación toma una relevancia importante en el mundo actual, ya que empresas desarrolladoras de tecnologías y productos en línea evolucionan a pasos agigantados en comparación a las redes tradicionales de datos, y cada vez demandan una mejor calidad de servicio, para transportar datos cada vez más grandes, o de una forma muy rápida, ya que sus mercados y el éxito de sus negocios dependen de una buena experiencia de usuario.

Razón por la cual, esta investigación está enfocada en los cambios que implica la adopción de la lógica de funcionamiento de las redes definidas por software, frente a las redes tradicionales de datos en entornos de videojuegos en línea. La investigación se la realizará en la ciudad de Riobamba entre los meses febrero y septiembre del 2023, mediante la implementación de un entorno virtual con herramientas de código abierto para la simulación de las diferentes tecnologías de red, teniendo como beneficiario a toda la comunidad profesional y/o aficionada a las telecomunicaciones, ya que el ofrecer un buen servicio en videojuegos online es un tema de interés público que cada vez va tomando más popularidad.

1.4. Objetivos

1.4.1. General

- Evaluar el rendimiento en latencia de una red definida por software (SDN) y una red tradicional de datos, en ambientes de videojuegos en línea.
-

1.4.2. Específicos

- Analizar los conceptos bajo los cuales funciona una red definida por software y una red de datos tradicional.
- Implementar a través de herramientas de código abierto un ambiente de simulación de videojuegos en línea en una red definida por software.
- Comparar el rendimiento en aplicaciones de videojuegos en línea entre redes definidas por software y redes de datos tradicionales.

CAPÍTULO II

2.1. Evolución de las comunicaciones digitales

El primer gran salto de la humanidad hacia las comunicaciones digitales inició con el telégrafo, se logró en 1840 mediante el envío de señales eléctricas a través de cables conectados entre un solo emisor y un solo receptor; dichas señales eléctricas eran interpretadas en el extremo receptor como información utilizando el código Morse [8].

Un siglo más tarde, con los grandes avances tecnológicos en EEUU nace la necesidad de intercambiar información entre sus distintas bases de investigación utilizando ordenadores, es así como en 1958 deciden fundar la Agencia de Proyectos de Investigación Avanzados o ARPA (por sus siglas del inglés Advanced Researchs Projects Agency), y para 1972 la red de ARPA (ARPANET) que interconectaba a 40 ordenadores ubicados en las principales universidades, empresas y entidades gubernamentales fue expuesta al mundo, junto a varios documentos (RFCs) que detallaban el funcionamiento de los protocolos y procedimientos que sostenían dicha red.

En 1974 se comercializaba una guía impresa llamada ARPANET DIRECTORY [9] que detallaba todos los programas (desde editores de texto hasta videojuegos) y servicios, a los cuales un ordenador de la ARPANET (cliente) podía acceder usando la dirección de red perteneciente al ordenador que alojaba dicha información (servidor), dando así forma al modelo de red cliente- servidor; dentro de la misma red en el ordenador del SRI (empresa estadounidense dedicada a la investigación científica y tecnológica) se alojaban las mismas listas de servidores y clientes de la ARPANET en un archivo llamado hosts.txt.

Para la década siguiente de los 80, el uso y comercialización de los ordenadores personales creció de forma exponencial, a la par una infraestructura de red internacional (Internet), haciendo casi titánica la tarea de mantener actualizada la ARPANET DIRECTORY y el archivo hosts.txt, por la excesiva existencia y demanda de direcciones de red; motivos suficientes para que en 1983 mediante el RFC 791 que describe el protocolo de Internet versión 4 (IPv4) se aumente la cantidad de direcciones de red a 4.294.967.296 y también se automatice su asociación con los nombres del dominio a las que pertenece dicho ordenador (Sistema de nombres de dominio o DNS), con la descentralización de la base de datos que contenía las listas de servidores y clientes. Finalmente, los dos grandes avances que proliferaron de forma masiva el internet fueron los sitios web y los navegadores web [10].

Durante toda esta evolución, se empezaron a evidenciar problemas en la infraestructura de esta red tradicional de datos, como, su baja escalabilidad, el aumento en los tiempos de administración y configuración que se requerían conforme esta iba creciendo, sumado a la baja

flexibilidad en innovación y gran complejidad que acarreaba una actualización; debido a que cada equipo que la conformaba tenía funcionalidades de software dependientes de cada fabricante, haciéndola una caja negra para los administradores de red. En contraposición a esta infraestructura de red que cada vez era más difícil y tediosa de administrar, nace el concepto de virtualizar su administración utilizando redes definidas por software.

A inicios de los 90's se presentó un proyecto denominado Redes activas (Active Networking en inglés) conceptuaba configurar y administrar una red utilizando software, mediante la inserción de microprogramas en cada uno de los equipos que conformaban la infraestructura de red; siendo estos transportados en paquetes junto al tráfico de la información. Perdió popularidad debido a su bajo nivel de seguridad, ya que bajo la misma lógica se podría presenciar programas maliciosos de terceros que afecten el funcionamiento de la red.

Posteriormente en 1995 con el proyecto de las Redes Programables (Programable Networks en inglés) se solventó la falla de seguridad de las Redes Activas, al instalar de forma manual los programas en cada uno de los dispositivos de la infraestructura de red, para posteriormente ser administrados y/o configurados usando paquetes de información; lo que las volvió muy tediosas de desplegar.

El concepto más parecido a las redes definidas por software (SDN) empezó en el 2006 con el proyecto Ethane, impulsado por un proyecto colaborativo entre la Universidad de Berkley y la Universidad de Stanford. Proponían realizar las tareas de enrutamiento, seguridad, acceso, y demás en un solo controlador centrar para toda la red, además, a través un canal seguro se tendría un fácil acceso a toda la infraestructura de red para su posterior configuración.

La gran problemática de las redes tradicionales de datos que se aspiraba solucionar con el proyecto Ethane, radicaba en la falta de entendimiento sobre la programación que definía las políticas de envío y procesamiento de paquetes en cada uno de los equipos pertenecientes a la infraestructura de red; ya que el software encargado de operar el hardware del equipo (firmware) variaba dependiendo del fabricante, apuntando así a tener ecosistemas de red cerrados, con poca flexibilidad e innovación dependiente de cada fabricante. En este mismo contexto, para transmitir paquetes de información entre equipos se utilizan diferentes protocolos normalizados mediante RFC's que satisfacen necesidades específicas con el fin de mantener un desempeño aceptable en internet; adjunto a esta técnica cada paquete de datos se procesa (plano de control) y se reenvía (plano de datos) en cada uno de los dispositivos que conforman la infraestructura de red, tal y como se observa en la; procesos que se rehacen en cada equipo de transmisión, añadiendo retrasos en la comunicación y dificultando la prestación de una buena calidad de servicio.

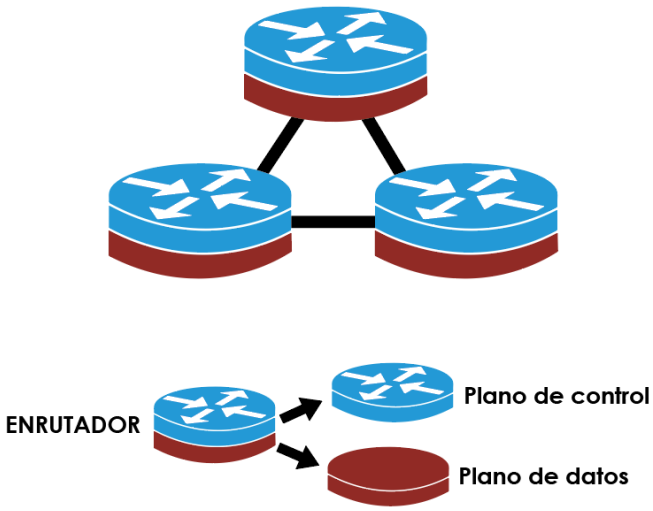


Figura 1: Esquematización básica de una red tradicional de datos
(Fuente: Autor)

Finalmente, en el año 2008 las Redes definidas por software o SDN (por sus siglas del inglés Software Defined Networking), son presentadas al mundo como nueva arquitectura de red para solventar las falencias ya mencionadas de las redes tradicionales de datos. Para evitar monopolios en las 2011 grandes empresas como Google, Microsoft, Facebook, Intel, entre muchas otras, colaboran en la creación de la Fundación de Redes Abiertas o ONF (por sus siglas del inglés Open Networking Fundation), para promover el desarrollo de esta arquitectura con software de código abierto[1].

2.2. Conceptualización y particularidades de las SDN

La tecnología de las redes definidas por software, o SDN (por sus siglas del inglés Software Defined Networking) proponen un cambio de paradigma gigantesco en la percepción y forma de elaborar, y desplegar redes de datos. Basa su lógica de funcionamiento en separar el plano de datos en conmutadores (o también llamados switches) y centralizar plano de control en un controlador, tal como se observa en la Figura 2; esta separación de los dos planos que en las redes tradicionales se encontraban en cada uno de los dispositivos de la red brindan múltiples ventajas entre las cuales podemos destacar las siguientes[1]:

- **Fácil administración:** al tener el plano de control centralizado es más fácil generar, optimizar, modificar y automatizar políticas, servicios o aplicaciones de red.
- **Escalabilidad:** Cualquier cambio que se necesite realizar en la infraestructura de red acarreará un menor tiempo de ejecución, ya que el administrador realizará los cambios

mediante API's directamente en el controlador de red y este a su vez reconfigurará todos los conmutadores.

- **Menos latencia:** al centralizar el plano de control, los conmutadores de la red únicamente se dedican a seguir instrucciones que el controlador proporciona a través de las tablas de flujo, haciendo que cada paquete de datos entrante solo se tenga que emparejar con las instrucciones de destino y puerto de salida del conmutador para seguir su recorrido por la infraestructura de red, simplificando el tener que procesarlo a medida que pasa por cada nodo de la infraestructura de red como habitualmente se hace en las redes tradicionales de datos; esta lógica de funcionamiento reduce drásticamente el tiempo que un paquete toma en viajar de un emisor a un receptor (latencia).
- **Código abierto:** al tener entidades como la ONF o la Linux Foundation como principales desarrolladoras, la investigación y uso de esta tecnología permanecerá totalmente libre, y gratuita.

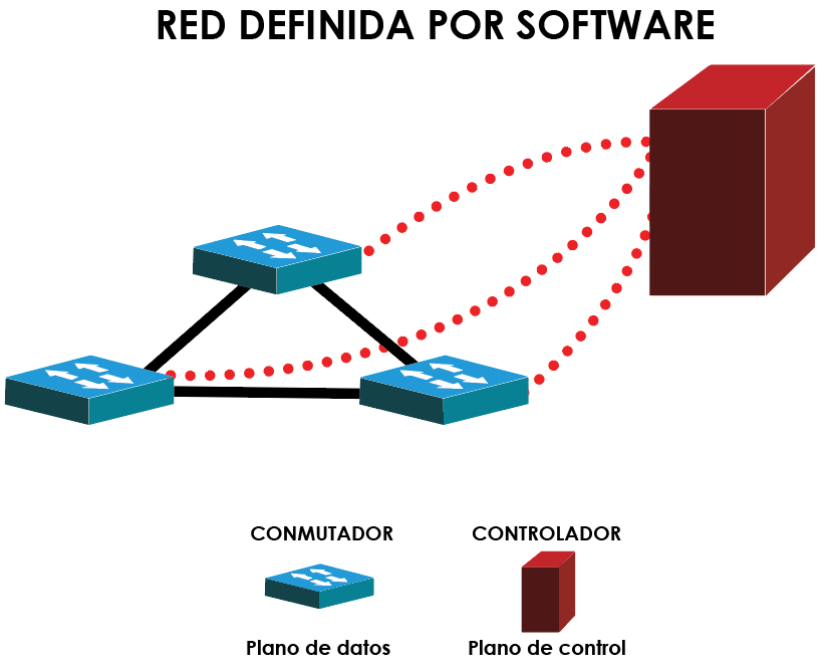


Figura 2: Esquematización básica de una red definida por software
(Fuente: Autor)

2.2.1. Arquitectura de una red definida por software

Como se puede observar en la Figura 3, la arquitectura de una red SDN consta de 3 capas, con dos API's encargadas de su interconexión [1].

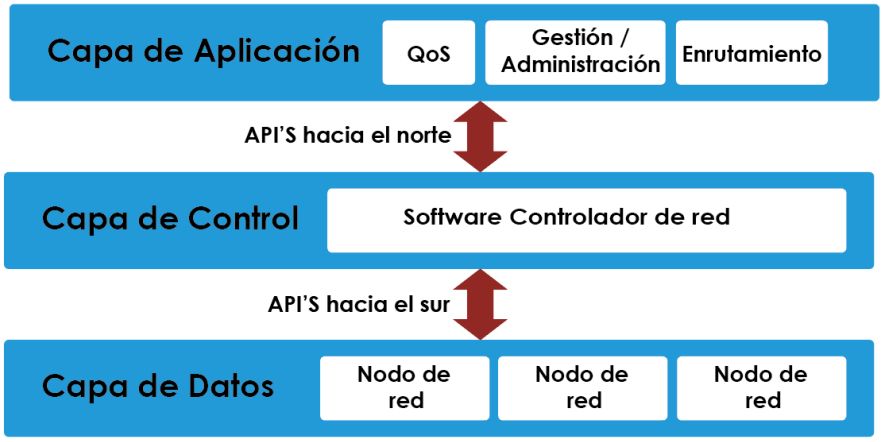


Figura 3: Esquematización básica de una red definida por software
(Fuente: Autor)

A continuación, se detalla la función de cada plano y API:

- **Plano de datos:** en este plano o capa se encuentran los conmutadores de red, que únicamente se encargan de la recepción y envío de paquetes de datos.
- **API hacia el sur:** proporciona un protocolo que intercomunica el plano de control y el plano de datos.
- **Plano de control:** consta de un controlador, encargado de gestionar y administrar la infraestructura del plano de datos.
- **API hacia el norte:** proporciona un protocolo que intercomunica el plano de aplicación y el plano de control.
- **Plano de aplicaciones:** consta de una o varias aplicaciones diseñadas en uno o más lenguajes de programación (XML, Python, Java, JSON, etc) a partir de la API proporcionada por el controlador que permiten al administrador generar, optimizar, modificar y automatizar políticas, servicios o la infraestructura de red.

2.2.2. Herramientas más usadas para simular y/o desplegar una arquitectura SDN

2.2.2.1. Mininet (Plano de datos)

Es una herramienta programada en el lenguaje de programación Python, proporciona una virtualización real de una red definida por software, brindando toda la infraestructura como los usuarios (hosts), conmutadores (switches) y controladores SDN. Se ejecuta directamente sobre el núcleo(kernel) de Linux e instancia de forma aislada cada uno de sus componentes [11].

Dispone de una herramienta llamada MiniEdit, la cual nos permite crear topologías de red, con la particularidad que una vez creadas, estas son exportadas a un script de Python para que Mininet las pueda interpretar y virtualizar[12].

Mininet utiliza varios núcleos de conmutadores, entre ellos el más destacado es Open VSwitch Kernel, el cual cuenta con soporte para OpenFlow y licencia de código abierto Apache 2.0, ampliamente utilizado en entornos virtualizados. Permite gestionar y reenviar el tráfico cursante por la red hacia la infraestructura virtual o física [13].

Para los usuarios o hosts, Mininet crea instancias de máquinas virtuales sobre el núcleo de Linux con Xterm [14].

Cuando mininet despliega toda la infraestructura de SDN y se empieza a generar tráfico, con referencia en la Figura 4 el conmutador utiliza un canal seguro y la API hacia el sur, para llenar sus tablas de flujo, y saber a donde dirigir el tráfico de acuerdo a las configuraciones almacenadas en el controlador, haciendo que para cada paquete de datos entrante exista una instrucción en la tabla de flujo que el conmutador debe ejecutar. En el caso de no existir instrucciones para un determinado paquete de datos, el conmutador vuelve a interactuar con el controlador hasta obtener instrucciones que modifiquen la tabla de flujo, o desechen el paquete de datos.

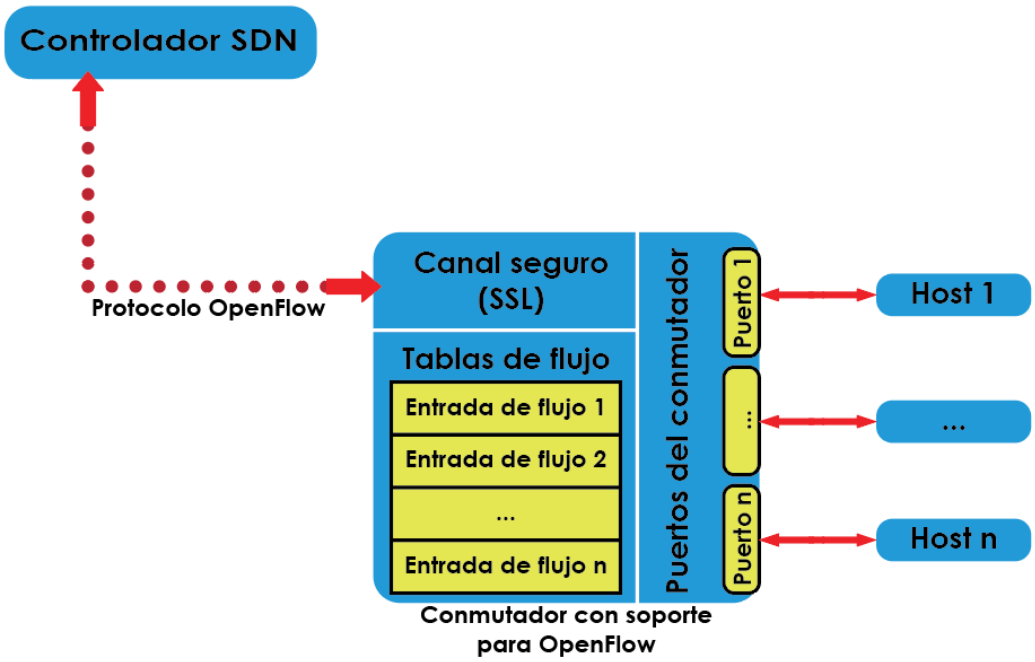


Figura 4: Interacción entre un controlador SDN y un conmutador OpenFlow
(Fuente: Autor)

2.2.2.2. OpenFlow (API hacia el sur)

Es el protocolo encargado de comunicar el plano de control y el plano de datos, mediante un canal de comunicación seguro (SSL), permitiendo al controlador incidir y modificar el comportamiento de los conmutadores ante un tráfico de paquetes de datos entrante. También es el encargado de anunciar la topología de red al controlador para llenar su árbol de datos.

2.2.2.3. OpenDaylight (Plano de control)

Es un controlador SDN de código libre, que tiene como finalidad promover el crecimiento y adopción de las redes SDN entre desarrolladores, fabricantes e investigadores para evitar la monopolización y/o privatización de esta tecnología[15].

Opendaylight está escrito en JAVA, basa su back-end OSGI y su front-end en KARAF, para la fácil instalación y administración dinámica de características modulares en el controlador sin la necesidad de reiniciarlo.

Centra su funcionamiento en un middleware (software intermediario entre dos aplicaciones) llamado MD-SAL o capa de adaptación de servicios impulsada por modelos, construido a partir del modelo de datos YANG y las características instaladas por el usuario a través de KARAF, para proporcionar servicios de comunicación y administración de su base de datos entre los diferentes planos de la arquitectura SDN. Al estar basado en el modelo de datos YANG, se generan dos tipos de árboles de datos:

- **Árbol de datos de configuración:** está conformado por un conjunto de datos que almacenan información sobre la configuración de la red SDN, dicha base de datos puede ser modificada desde el plano de aplicación para ser anunciada en el plano de datos.
- **Árbol de datos de funcionamiento:** reflejan el estado de la red SDN.

Al ser MD-SAL un middleware proporciona dos API’s para interconectar los planos restantes de la arquitectura SDN, como ejemplo tenemos a OpenFlow para el sur y REST para el norte [16].

2.2.2.4. RESTCONF API (API hacia el norte)

Proporciona un canal de comunicación HTTP entre el controlador y el plano de aplicaciones. Provee al administrador la capacidad de gestionar, modificar y visualizar las dos bases de datos almacenadas en MD-SAL mediante métodos HTTP (POST, GET, PUT y DELETE) en formato XML o JSON [16].

2.2.2.5. OpenDaylight OpenFlow Manager (Plano de aplicaciones)

Es una aplicación de administración para interactuar con la red SDN, nos permite visualizar la topología, realizar tareas de gestión y administración de la red sobre los árboles de datos presentes en MD-SAL del controlador SDN [17].

2.3. Evolución de los videojuegos

Para las tres últimas décadas del siglo XX, con la aparición de los primeros circuitos integrados y unidades centrales de procesamiento, grandes compañías encargadas de crear, fabricar y distribuir las consolas de videojuegos luchaban por monopolizar el mercado con principalmente videojuegos arcade de 4, 8, 16, 32 y 64 bits.

La consola precursora en el mundo de los videojuegos se presentó al mundo en 1972, bajo el nombre de Magnavox Odyssey, con el famoso videojuego de Ping Pong.

Tan solo 5 años más tarde, se lanzó al mercado la Atari 2600, trayendo mejoras tanto a nivel técnico como a nivel visual, destacando la una presencia de 128 colores para representar escenarios en la pantalla, y el nacimiento del legendario PAC-MAN de 1982, un juego 3D, sencillo pero adictivo de 4 bits.

A tan escasos años empezó la generación de los 8 bits sobresaliendo entre su gran catálogo de videojuegos el clásico Super Mario Bross lanzado en 1985 para la consola de videojuegos Nintendo Enterteiment System o también llamada NES, y es fue uno de los máximos esponentes de los videojuegos de travesía lineal de desplazamiento lateral (side scroller en inglés).

En 1992 se lanzó el Street Fighter II: The World Warrior para la Super Nintendo Enterteiment System, un videojuego de lucha de 16 bits con un máximo de dos jugadores locales. Con la aparición de la Sega Saturn y su módulo Netlink[4], videojuegos de 32 bits como el Sega Rally Championship[22] de 1995 fueron los precursores del modo multijugador en línea, ya que gracias a el módulo Netlink de 28.8 kbit/s y la red telefónica varios jugadores podían acceder a una misma partida de forma remota.

Posteriormente se lanzó la Nintendo 64, y con ello uno de los mejores videojuegos de la historia, el Super Mario 64 de 1996, un juego de 64 bits de mundo abierto con un máximo de 2 jugadores locales[18].

Mientras las grandes compañías (como Nintendo, SEGA, Sony, entre otras), arrasaban e imponían estándares de cómo debía ser un videojuego, un reducido nicho de videojugadores, en

su mayoría investigadores y estudiantes universitarios, desarrollaban, mantenían y jugaban los primeros videojuegos online, usando los pocos ordenadores y el tan primitivo acceso a la red de sus instituciones. Uno de los videojuegos que popularizó el jugar en línea, es MUD2 [19] lanzado en 1985, un juego basado en roles, en el cual los jugadores mediante una consola de texto se turnaban para avanzar por calabozos con diferentes obstáculos, y acertijos. Para inicios de los 90', un genio de la informática llamado John Carmack destruyó el monopolio de las consolas de videojuegos, desarrollando los primeros motores gráficos para computadoras, sobre los cuales funcionaban los videojuegos en red local (LAN) que concentraban a cientos de jugadores como QUAKE ll [20] y Half-Life [21]. Ultima Online [22] de 1997 fue uno de los pioneros del género de juego de rol en línea multijugador masivo (MMORPG por sus siglas en inglés), expandiendo aún más el imperio de los mundos virtual en línea.

En el 2002 Microsoft entró en la competencia de consolas con la Xbox, que junto a un enorme catálogo de videojuegos (entre los cuales sobresale HALO) y un servicio en línea llamado Xbox Live [23] que permitía a los jugadores compartir logros, chatear y acceder a partidas multijugador, proliferando aún más el estándar de jugar videojuegos en línea. En su contraparte, Blizzard en el 2004 estrenó World Of Warcraft uno de los MMORPG para computadora más influyentes de la historia, no solo por su cantidad abismal de videojugadores en línea mensuales, sino también por el valor en los estudios científicos que han sido concretados gracias a sus mecánicas complejas como el comercio, viaje a largas distancias, interacciones entre cuidades, entre otras, que simulan el mundo real; el incidente con más renombre dentro de la comunidad científica es el de la pandemia virtual de la Sangre Corrupta suscitada entre el 13 de septiembre y el 8 de octubre del año 2005, que describió a la perfección el comportamiento humano frente a una pandemia real como la sucedida 15 años más tarde con el COVID-19 [24].

En la primera década de los 2000, los videojuegos en línea representaban un mercado importante, ya que la venta de copias físicas o virtuales de estos producían ganancias millonarias. Sin embargo, el 17 de agosto del 2011 VALVE, una de las empresas más grandes de videojuegos realizó un evento llamado "The International" con una recompensa de 1 millón de dólares americanos para un equipo ganador, con la finalidad de promocionar la beta de su videojuego en línea DOTA 2, sin imaginar que dicho suceso popularizaría a escala mundial los deportes electrónicos (E-sports en inglés), ya que por la gran cifra de dinero atrajo el interés de muchos videojugadores y mercados. "The International" alcanzó un récord histórico durante la pandemia COVID-19 en el 2021, con un premio total de casi 40 millones de dólares americanos para el equipo ganador [25].

VALVE el 21 de agosto del 2012 lanzó al mercado Counter Strike: Global Offensive o simplemente CS:GO, su evento con más renombre hasta la actualidad fue la competición electrónica "EMS One Katowice 2014", ya que durante su ejecución los videojugadores podían comprar cápsulas por 25 centavos de dólar americano, en las cuales obtenían cromos aleatorios de los equipos participantes para decorar las armas dentro del videojuego; es importante

mencionar que todos los artículos obtenidos o comprados en este videojuego se pueden comerciar en un mercado mundial dentro o fuera de la plataforma oficial de Valve llamada Steam. Lo realmente impactante y asombroso es la evolución de precio que ha tenido este objeto virtual, ya que, en el transcurso de estos nueve años, cada cápsula ha alcanzado un precio de 33 mil dólares americanos por unidad, debido a que dentro de ella se pueden encontrar cromos como el "Titan (Holo) | Katowice 2014" mostrado en la Figura 5, el cual alcanza precios de casi 100 mil dólares americanos [26].

Figura 5: Cromo Titan (Holo) | Katowice 2014
(Fuente: Autor)

Al igual que el cromo mencionado anteriormente existen miles de objetos virtuales en CS:GO y otros videojuegos, que atraen a muchos inversionistas, ya que, a diferencia de otros mercados, sus precios solo se incrementan con el paso del tiempo porque solo se pueden obtener durante un evento realizado por la desarrolladora del videojuego, haciéndolos exclusivos.

Otro mercado importante dentro de los videojuegos en línea radica en la comercialización de las monedas virtuales por dinero real, usualmente se obtienen dentro del mismo videojuego al realizar actividades como matar monstros, recolectar recursos, etc. El registro más surrealista de este mercado se da en Venezuela, ya que para el octubre del 2020 el salario mínimo era de apenas 0.92 centavos de dólar americano[27]; obligando a muchos venezolanos a realizar una migración virtual a videojuegos como Runescape, en donde por cada 1 hora de juego podían recolectar y vender en mercados externos una cantidad de oro virtual equivalente a 5 dólares americanos (5.43 veces más que el salario mínimo de país)[28].

2.3.1. Los deportes electrónicos o E-sports

Los deportes electrónicos o E-sports, nacen de la participación profesional, competitiva y reglada en un videojuego en línea como DOTA 2. Difieren de los deportes tradicionales por el tipo de esfuerzo, ya que requieren un alto rendimiento a nivel mental y emocional, para ello, los videojugadores profesionales estudian y entrenan alrededor de un videojuego en línea durante largas jornadas de tiempo.
Han tomado una gran relevancia, ya que actualmente no difiere de deportes como el futbol, generando miles de puestos de trabajo e influyendo de manera positiva en la economía del lugar donde se los realice.

2.3.2. Los E-Sports y su relación con las redes de datos

Las redes de datos son el eje fundamental de los E-sports, ya que sin una buena infraestructura y parámetro óptimos de red no se pueden ejecutar.

Un jugador profesional de E-sports conoce a la perfección los tiempos de animación y ejecución de cada una de las mecánicas del videojuego, por lo cual una pequeña variación en los parámetros de la red de datos podría afectar su rendimiento y a que el encuentro competitivo no sea justo.

Los parámetros que una red de datos debe cumplir son:

- **Ping bajo:** en el contexto de un videojuego, durante una partida en línea el ping él es sustituto del término RTT, y se describe como el tiempo milisegundos que transcurre desde que el videojugador presiona alguna tecla para realizar cierta acción dentro del videojuego , hasta que la misma es visualizada por el jugador en el monitor luego de que el paquete de datos que lleva dicha acción ha viajado por la red desde el cliente (videojugador) hacia el servidor y viceversa, por lo que un ping con un valor ideal de 0ms o cercano, genera una gran diferencia entre un videojugador que tenga este valor ideal y uno que tenga valores muy superiores (80ms por ejemplo), ya que en el monitor de ambos jugadores se reproducirán las acciones con una demora equivalente a la diferencia de sus pings.

- **Estabilidad de la conexión**: se logra cuando no existe una pérdida de paquetes en el medio de transmisión y el Jitter es idealmente cero o cercano a cero. Dentro de una partida en línea de un videojuego, la pérdida de paquetes se puede apreciar gráficamente, con la repetición o congelamiento de los cuadros por segundo reflejados en el monitor de videojugador, saltando acciones que el usuario realizó, pero no se ejecutaron dentro del videojuego por la pérdida de los paquetes que llevaban dicha acción, o porque el

Jitter es demasiado alto, ya que genera una variación del ping, provocando choques entre los paquetes enviados en un tiempo t-1 y los paquetes actualizados en un tiempo t emitidos desde el servidor hacia el videojugador, o viceversa.

El tener en cuenta estos parámetros de red estrictos requeridos un entorno competitivo nos da una buena referencia para tratar de replicarlos usando diferentes tecnologías de red y compararlos[29].

2.4.Funcionamiento de un videojuego en línea

Un videojuego en línea funciona con una estructura de cliente – servidor, centrando su funcionamiento en enchufes (sockets en inglés), que son los encargados de crear un medio de conexión entre un servidor y un cliente.

El servidor asigna un puerto al socket para estar a la escucha o espera de solicitudes para establecer una conexión, como se aprecia en la Figura 6. Si uno o más clientes realizan solicitudes de conexión utilizando un puerto local, y apuntan al puerto por el cual el socket del servidor está a la escucha se establece la conexión[30].

Figura 6: Estructura cliente – servidor
(Fuente: Autor)

En un videojuego competitivo como CS:GO el servidor tiene que procesar y validad las acciones del videojugador una cantidad determinada de veces por segundo (tickrate). En particas casuales y no tan competitivas de CS:GO, los desarrolladores establecen un tickrate de 64, es decir que las acciones del videojugador se procesan y validan con el servidor cada 15.625 milisegundos. Con la finalidad de registrar muchas más acciones y tener un nivel de precisión aún mayor en partidas competitivas se crean servidores con tickrates de 128, es decir, cada 7.812 milisegundos. A nivel de red, el duplicar el tickrate de 64 a 128 implica un mayor consumo de ancho de banda, aumento en el jitter y el RTT, ya que el hardware presente en la red tradicional no está listo para soportar una duplicación en su tráfico.

Uno de los principales problemas de red al que los desarrolladores se enfrentan es la latencia, ya que aumenta de manera proporcional a la distancia y número de saltos de los nodos de red existentes entre los clientes, y el servidor. Para compensar el RTT en los videojuegos, los

desarrolladores utilizan métodos matemáticos como la extrapolación e interpolación, que junto a una compensación de latencia alteran de forma virtual el tiempo, otorgando a cada jugador un reloj que va de acuerdo a su latencia. Por ejemplo, si tenemos un jugador A con 20ms y un jugador B con 80ms de RTT, las acciones del videojugador A en la pantalla del videojugador B estarán retrasadas por 60ms; mientras que las acciones del jugador B en la pantalla del jugador A estarán extrapoladas a partir su último paquete de datos recibido, y después de 60ms se validan con la información real enviada por el jugador B [31] .

2.4.1. Etapas de conexión de un videojuego en línea

Para definir su funcionamiento, analizaremos registros de los procesos generados por la consola del videojuego Counter Strike: Global Ofensive (abreviado como CS:GO), cuando un jugador accede a una partida multijugador en línea competitiva, entre las etapas de conexión más destacadas tenemos:

2.4.1.1. Emparejamiento

El videojuego (cliente del videojugador) ejecutado desde Riobamba – Ecuador, empieza a analizar el tiempo en milisegundo de ida y vuelta (RTT) entre el cliente y sus servidores distribuidos en las diferentes regiones de cobertura (ver Figura 7), priorizando la consulta de partidas multijugador en línea disponibles en los servidores que tengan el RTT más bajo, con un límite establecido mediante el parámetro “Máximo ping aceptable de emparejamiento” ubicado en el panel de configuraciones del videojuego (ver Figura 8); para agregar al cliente a una cola de emparejamiento, que se completa cuando existen 9 clientes más.

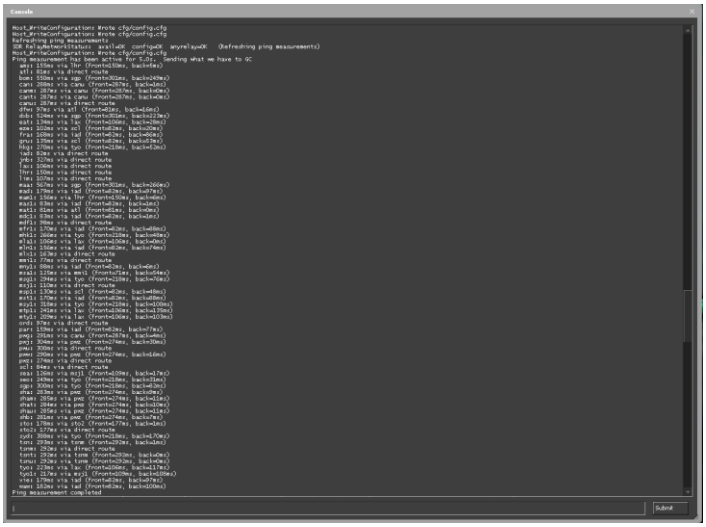


Figura 7: Resultado de RTT desde el videojugador a todos los servidores disponibles, generado por la consola de CS:GO (Fuente: Autor)

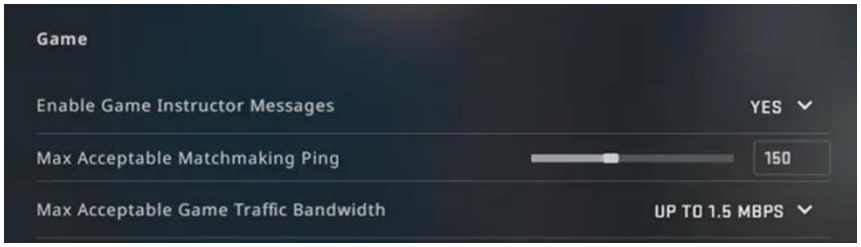


Figura 8: Parámetros iniciales del cliente para establecer una conexión con el servidor de partidas multijugador en línea (Fuente: Autor)

2.4.1.2. Conexión

Cuando se valida información del jugador, se crean sockets utilizando la dirección IP pública y un puerto del servidor para establecer una conexión (ver Figura 9), mismo proceso se repite en cada uno de los clientes que formarán parte de la partida multijugador en línea. En la Figura 9, también se puede evidenciar la conexión, ya que el estado de la interfaz gráfica (UI) del videojuego se genera un cambio de "MAINMENU" a "LOADINGSCREEN", es decir, empieza a cargar los recursos necesarios para entrar en la partida en línea.

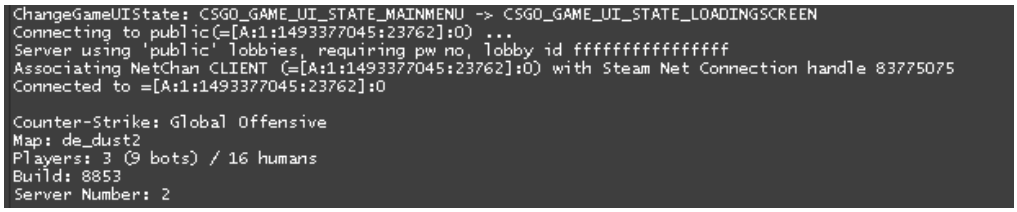


Figura 9: Registro de conexión del videojugador al servidor del juego mediante sockets, generado por la consola de CS:GO (Fuente: Autor)

2.4.1.3. Sincronización

En esta etapa de la partida el videojugador entra en un conteo regresivo que concluye cuando todos los 9 jugadores restantes completen la etapa de conexión a la partida multijugador online alojada en el servidor.

2.4.1.4. Inicio de partida

El servidor y los jugadores empiezan a intercambiar información 64 veces por segundo, haciendo uso de los protocolos UDP (para transportar datos de la partida en tiempo real). La

información sobre las métricas de red se puede observar en el videojuego mediante la ejecución en consola del comando “net_graph 1” (ver Figura 10).

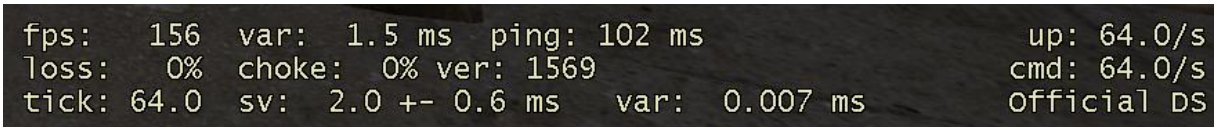


Figura 10: Métricas de red visualizadas en CS:GO mediante el comando “net_grap 1” (Fuente: Autor)

Para validar el uso del protocolo UDP en los videojuegos para intercambiar datos en línea entre los videojugadores y servidores, accedimos a una partida de CS:GO, mientras capturamos el tráfico haciendo uso de Wireshark en su versión 4.0.3. En la Figura 11, se puede observar que el emparejamiento con el RTT más bajo para nuestra partida en línea se encuentra en Atlanta - EE.UU y tiene la IP pública 162.254.199.78, y a su vez se puede validar el uso del protocolo UDP para trasportar datos de la partida en línea. Por otro lado, en la Figura 12 se puede validar que los paquetes enviados y recibidos por segundo son aproximadamente 64, lo cual equivale al tickrate evidenciado en la Figura 10 (cmd: 64.0/s).

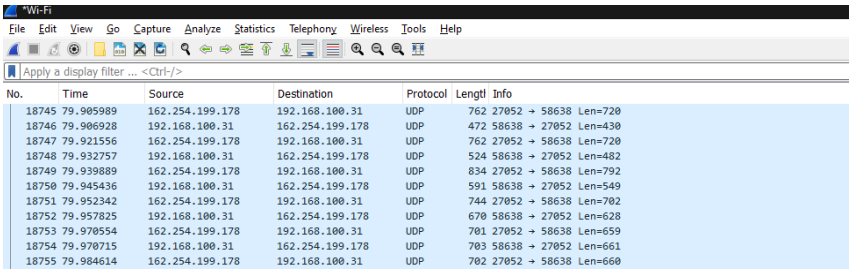


Figura 11: Captura del tráfico paquetes de datos en Wireshark, generado al entrar en una partida real de CS:GO (Fuente: Autor)

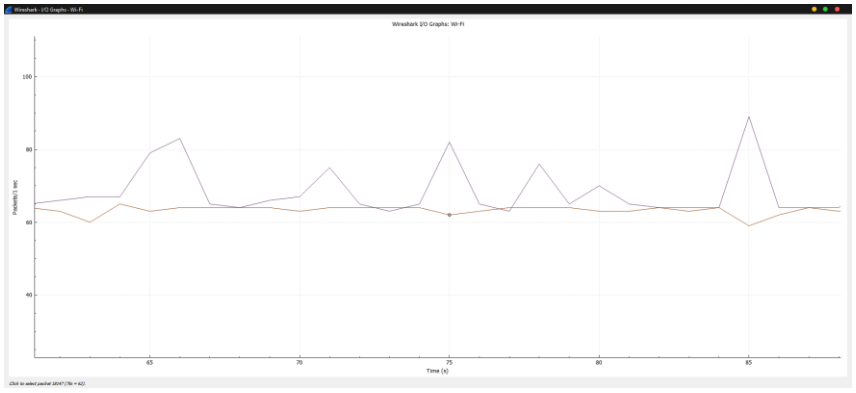


Figura 12: Captura del tráfico de paquetes visualizado en Wireshark usando su función I/O Graph, generado al entrar en una partida real de CS:GO (Fuente: Autor)

Es importante mencionar que el RTT dentro del videojuego se va a diferenciar del RTT medido con el comando “ping 162.254.199.178”, ya que como se observa en la Figura 10 se tiene un RTT promedio de 102ms, mientras que en la Figura 13 se observa un RTT promedio de 89ms. Esto debido a que cuando utilizamos el comando “ping” por la red circula una pequeña cantidad de información y no genera mucho tráfico, caso contrario a los 128 paquetes UDP por segundo que se procesan junto RTT medido dentro del videojuego, sin contar con los múltiples procesos de encriptación y seguridad que se añade en cada paquete enviado.

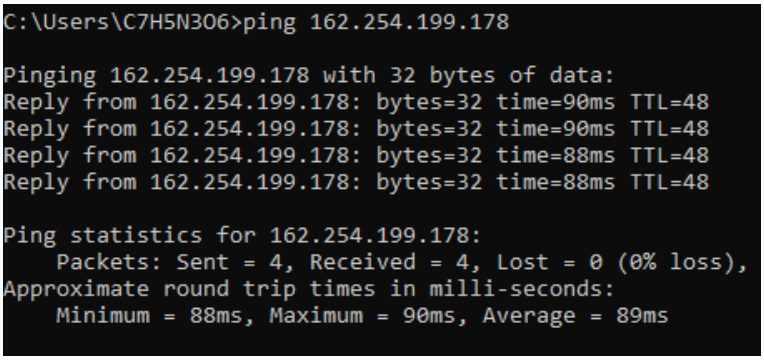


Figura 13: Prueba de RTT desde Riobamba – Ecuador hacia el servidor de CS:GO ubicado en Atlanta – EE.UU con la dirección IP 162.254.199.178 (Fuente: Autor)

2.5.Topologías de red y sus tipos de demora

En las telecomunicaciones, para que un paquete de datos llegue de un punto A, a un punto B, se requiere de un medio de transmisión (fibra óptica en el caso de las redes de datos) .En el caso de la fibra óptica, los paquetes de datos se propagan por el medio codificados en pulsos de luz a una velocidad de 4.9 microsegundos por kilómetro[32] , es decir, que si por ejemplo un paquete de datos necesita ser transportado desde un punto A a un punto B ubicado a 1000Km de distancia se requiere un tiempo de 4.9 milisegundos para que la operación sea ejecutada, a este tiempo de ejecución lo denominaremos demora física.

En su contraparte, los encargados de establecer un camino lógico o enrutamiento entre el punto A y el punto B son los nodos de red (enrutadores, conmutadores, etc), dicho proceso también requiere un tiempo de ejecución (ya que entre el punto A y el punto B pueden existir miles de puntos adicionales) al cual nombraremos demora lógica [33].

La distribución de los nodos de red y medios de transmisión forman la topología física de la red, mientras que las configuraciones y protocolos de enrutamiento definidos en los nodos de red forman una topología lógica de la red.

Dependiendo de las necesidades de los usuarios se pueden crear varios tipos de topologías, de las cuales únicamente destacaremos dos[34]:

- **Topología lineal:** se caracteriza por tener un único medio de transmisión, al cual varios nodos acceden mediante interfaces de red, de tal manera que todos comparten un mismo enlace.

- **Topología tipo árbol:** es la unión de varias topologías lineales, concentrando todos sus medios de transmisión en un mismo nodo principal, del cual se ramifican varios nodos.

La topología de red más usual en los videojuegos en línea es la lineal, ya que interconecta con un único medio de transmisión y de forma directa al videojugador con el servidor, pero, al existir más de un jugador interconectado al mismo servidor como nodo en común se transforma en una topología de árbol. Existen más topologías de red, como la de malla, pero por motivos de estudio no serán analizadas.

También es importante mencionar que la demora lógica y la demora física siempre están intrínsecas en una topología de red. En el contexto de una comunicación bidireccional estas demoras se suelen medir tanto de vuelta, como de vuelta (RTT por su acrónimo del inglés round-trip time) con la función "ping" presente en las terminales de comando de varios sistemas operativos como Windows, GNU/Linux, Android, etc. Por motivos prácticos también definiremos la siguiente fórmula para obtener el RTT total:

$$RTT = RTT_{físico} + RTT_{lógico} \quad (1)$$

Donde:

- $RTT_{físico}$ es la demora física de ida y vuelta desde punto A a un punto B existente en la topología física de la red.

- $RTT_{lógico}$ es la demora lógica de ida y vuelta desde punto A a un punto B existente en la topología lógica de la red.

CAPÍTULO III

3.1. Metodología

3.1.1. Tipo de investigación

La presente investigación, se emplea una metodología experimental para el análisis y comparación de la tecnología de red tradicional, y tecnología de red definida por software, cuando son utilizadas en un entorno de videojuegos en línea, sobre una misma topología de red.

3.1.1.1. Investigación cuantitativa

Se empleará un método de recolección de datos cuantitativos, ya que la variable principal en estudio es el tiempo.

3.1.2. Métodos, técnicas e instrumentos de investigación

3.1.2.1. Método Experimental

En la investigación se utilizará el método experimental, utilizando pruebas de rendimiento sobre un entorno virtual que consta de una misma topología, replicada con tecnología de red tradicional y SDN, la cual permitirá el análisis de los parámetros necesarios de una red de datos en entornos de videojuegos en línea.

3.1.2.2. Técnica de recolección de datos

3.1.2.2.1. Observación.

A través de la observación se realizará un registro de los datos de acuerdo al entorno virtual de pruebas, mismos que serán almacenados con scripts de Python de forma ordenada.

3.1.3. Procedimiento

Inicialmente se realizará un estudio a detalle del funcionamiento de la tecnología de red tradicional, la tecnología de red definida por software y la influencia del RTT en la topología de red, así como sus principales herramientas de software disponibles para desplegar entornos virtuales de pruebas.

Usando el registro de consola proporcionado por el videojuego CS:GO se analizará el funcionamiento de sus servidores para partidas en línea, su interacción con los clientes (videojugadores) e influencia sobre una topología de red.

Con la finalidad de obtener un entorno virtual de pruebas fiel a la realidad, se utilizará el acceso remoto a 10 computadoras de videojugadores de diferentes países de Latinoamérica, para obtener datos del RTT desde su ubicación hacia un servidor de partidas en línea de CS:GO mediante el comando "ping" y así replicarlo de forma teórica con una topología con tecnología de red tradicional, misma que será simulada en GNS3 para obtener datos del RTT simulado.

Con los datos obtenidos tanto del RTT real como del RTT simulado realizaremos una prueba estadística para verificar si la topología con tecnología de red tradicional simulada replica de manera fiel los parámetros de la topología de red real. Una vez, la topología con tecnología de red simulada sea validada, la replicamos, pero con tecnología SDN.

Para recrear el comportamiento del videojuego CS:GO en partidas en línea, crearemos scripts de Python que serán validados con Wireshark, tomando en cuenta parámetros como el tickrate y el protocolo usado para la transferencia de paquetes entre los videojugadores, y el servidor. También, se automatizará la recolección de datos del RTT y Jitter de la red, mientras la simulación de partida en línea es ejecutada.

Una vez validados los scripts, se los usará para crear dos entornos virtuales de pruebas. El primero ejecutará estos scripts, junto a la topología con tecnología de red simulada en GNS3. El segundo entorno virtuales de pruebas utilizará los mismos scripts, pero con el cambio a tecnología SDN de la topología planteada en GNS3. De tal forma, que ambos reflejen comportamiento de las diferentes tecnologías de red sobre una topología cuando varios videojugadores acceden a una partida en línea de CS:GO.

3.1.4. Población y muestra

3.1.4.1. Población

La población está compuesta por los datos temporales obtenidos en una red definida por software y en una red tradicional de datos, al ser utilizadas para entornos de videojuegos en línea.

3.1.4.2. Muestra

La muestra se tomará como un método de muestreo aleatorio de las poblaciones definidas anteriormente.

Al azar se seleccionarán 10 videojugadores de diferentes partes de Latinoamérica y un servidor de un videojuego en línea, para replicar mediante un entorno virtual de pruebas el escenario de una partida de un videojuego en línea tanto en tecnología de red tradicional, como SDN; con la finalidad de obtener muestras de datos temporales fieles a la realidad, para posteriormente generalizar hipótesis sobre la población antes mencionadas.

3.1.5. Operacionalización de las Variables

Variable	**Definición conceptual**	**Indicadores**	**Técnicas e Instrumentos**
INDEPENDIENTE			
Topología de red	Se define como infraestructura y tecnología de red empleada para que un servidor de CS:GO pueda establecer una comunicación bidireccional con los videojugadores	Número de saltos de red Distancia física entre un videojugador y un servidor	Pruebas de RTT desde cada videojugador hacia un servidor de CS:GO, junto a un análisis teórico del RTT físico y RTT lógico
DEPENDIENTE			
Rendimiento de red	Se define como el tiempo y variación del mismo necesario para que un paquete de datos pueda ser enviado transmitido desde un servidor de CS:GO a los videojugadores, y viceversa	RTT Jitter	Scripts de Python

Tabla 1: Operacionalización de variables

3.2. Entorno virtual de pruebas

Los componentes más importantes de este estudio radican en la topología de red, tecnología de red e interpretación del funcionamiento de un videojuego en línea.

Un buen entorno virtual de pruebas necesita ser contrastado con datos reales, es por ello, que partimos con la creación de una topología de red, sabiendo que actualmente en la mayoría del mundo se sigue utilizando tecnología de red tradicional. También es importante tener en cuenta que una partida competitiva en línea real de CS: GO está compuesta por 10 videojugadores y un servidor que se encarga de recibir, procesar y retransmitir paquetes de datos de los mismos.

3.2.1. Topología de red

Con la finalidad de replicar la topología de red real se utilizó la IP de un servidor de CS:GO con el RTT más bajo, localizándose este en Atlanta - Estados Unidos, con la dirección de red IPv4 162.254.199.178, y desde 6 países de Latinoamérica con acceso a 10 computadoras de videojugadores se realizaron pruebas del tiempo de ida y vuelta (RTT) mediante el comando "ping" en una consola de comandos hacia la IP del servidor, en la Tabla 2 se puede observar el resumen de esta recolección de datos.

La parte física de la topología de red real está conformada por varios kilómetros de fibra óptica que interconectan a cada uno de los 10 jugadores de los 6 países de Latinoamérica con el servidor ubicado en Atlanta- EE.UU. Para estimar las distancias reales de fibra óptica desde cada videojugador hacia el servidor, se utilizó un mapa de la fibra óptica submarina escalado a tamaño real con Google Earth 7.3.6.934, tal y como se puede apreciar en la Figura 14. Con las distancias obtenidas, en la Tabla 3 se calculó el RTT de la fibra óptica desde cada videojugador, hacia el servidor, tomando en cuenta que por cada kilómetro de fibra óptica se añade al enlace una demora física de 4.9 microsegundos [32].

País	**Ciudad**	**Videojugador**	**RTT promedio real (ms)**
México	Veracruz	H1	58.000
		H2	62.000
Colombia	Santa Marta	H3	72.000
El Salvador	Jiquilisco	H4	74.000
		H5	76.000
Perú	Cusco	H6	113.000
		H7	109.000
Ecuador	Riobamba	H8	92.000
		H9	86.000
Chile	Santiago de Chile	H10	152.000

Tabla 2: RTT promedio desde cada videojugador hacia el servidor de CS:GO con la IPv4 162.254.199.178

Figura 14: Mapa de la fibra óptica marítima mundial escalado a tamaño real con Google Earth (Fuente: Autor)

La parte física de la topología de red real está conformada por varios kilómetros de fibra óptica que interconectan a cada uno de los 10 jugadores de los 6 países de Latinoamérica con el servidor ubicado en Atlanta- EE.UU. Para estimar las distancias reales de fibra óptica desde cada videojugador hacia el servidor, se utilizó un mapa de la fibra óptica submarina escalado a tamaño real con Google Earth 7.3.6.934, tal y como se puede apreciar en la Figura 14. Con las distancias obtenidas, en la Tabla 3 se calculó el RTT de la fibra óptica desde cada videojugador, hacia el servidor, tomando en cuenta que por cada kilómetro de fibra óptica se añade al enlace una demora física de 4.9 microsegundos [32].

Videojugador	**País**	**Ciudad**	**Distancia de fibra óptica (Km)**	**Demora física teórica (ms)**	**RTT físico teórico (ms)**
H1 - H2	México	Veracruz	2592	11.9232	23.8464
H3	Colombia	Santa Marta	3553	16.3438	32.6876
H4 - H5	El Salvador	Jiquilisco	2626	12.0796	24.1592
H6 - H7	Perú	Cusco	6657	30.6222	61.2444
H8 - H9	Ecuador	Riobamba	4734	21.7764	43.5528
H10	Chile	Santiago de Chile	8683	39.9418	79.8836
			Promedio:	22.1145	44.229

Tabla 3: Cálculo de la demora física y RTT físico de forma teórica

En la Tabla 4, usando la fórmula (1) nos podemos aproximarnos al RTT real, usando el RTT físico teórico para calcular el RTT lógico teórico, de tal forma, que si los sumamos obtenemos el RTT real, pero calculado de forma teórica.

País	Ciudad	RTT promedio real (ms)	RTT físico teórico (ms)	RTT lógico teórico (ms)
México	Veracruz	60	23.8464	36.1536
Colombia	Santa Marta	72	32.6876	39.3124
El Salvador	Jiquilisco	75	24.1592	50.8408
Perú	Cusco	111	61.2444	49.7556
Ecuador	Riobamba	89	43.5528	45.4472
Chile	Santiago de Chile	152	79.8836	72.1164

Tabla 4: Cálculo del RTT lógico de forma teórica

Uno de los simuladores de redes tradicionales más usados por sus resultados fieles a la realidad es GNS3, razón por la cual lo usamos en su versión 2.2.42, junto al enrutador Cisco C7200, para obtener el RTT lógico agregado a la topología de red por cada salto que realiza el paquete de datos desde el videojugador hacia el servidor, los resultados de la simulación se pueden observar en la Tabla 5.

Número de saltos / Enrutadores	RTT simulado (ms)
1	12
2	21
3	34
4	44
5	52
6	72
7	82

Tabla 5: RTT lógico simulado para cada número de saltos de red

En la Tabla 6, utilizamos el RTT simulado de la Tabla 5 para estimar el número de saltos de red necesarios con la finalidad de aproximar el RTT lógico de la red al calculado en la Tabla 4.

País	Ciudad	RTT promedio real (ms)	RTT lógico teórico (ms)	Número de saltos necesarios
México	Veracruz	60	36.1536	3
Colombia	Santa Marta	72	39.3124	3
El Salvador	Jiquilisco	75	50.8408	5
Perú	Cusco	111	49.7556	5
Ecuador	Riobamba	89	45.4472	4
Chile	Santiago de Chile	152	72.1164	6

Tabla 6: Número de saltos necesarios para aproximar el RTT lógico de la red al calculado en la Tabla 4

Teniendo el número de saltos de red necesarios de la Tabla 2, junto a la demora física necesaria para replicar la topología real y en base a la teoría del apartado 2.5 del capítulo 2, podemos definir la topología de la Figura 15, como teóricamente correcta.

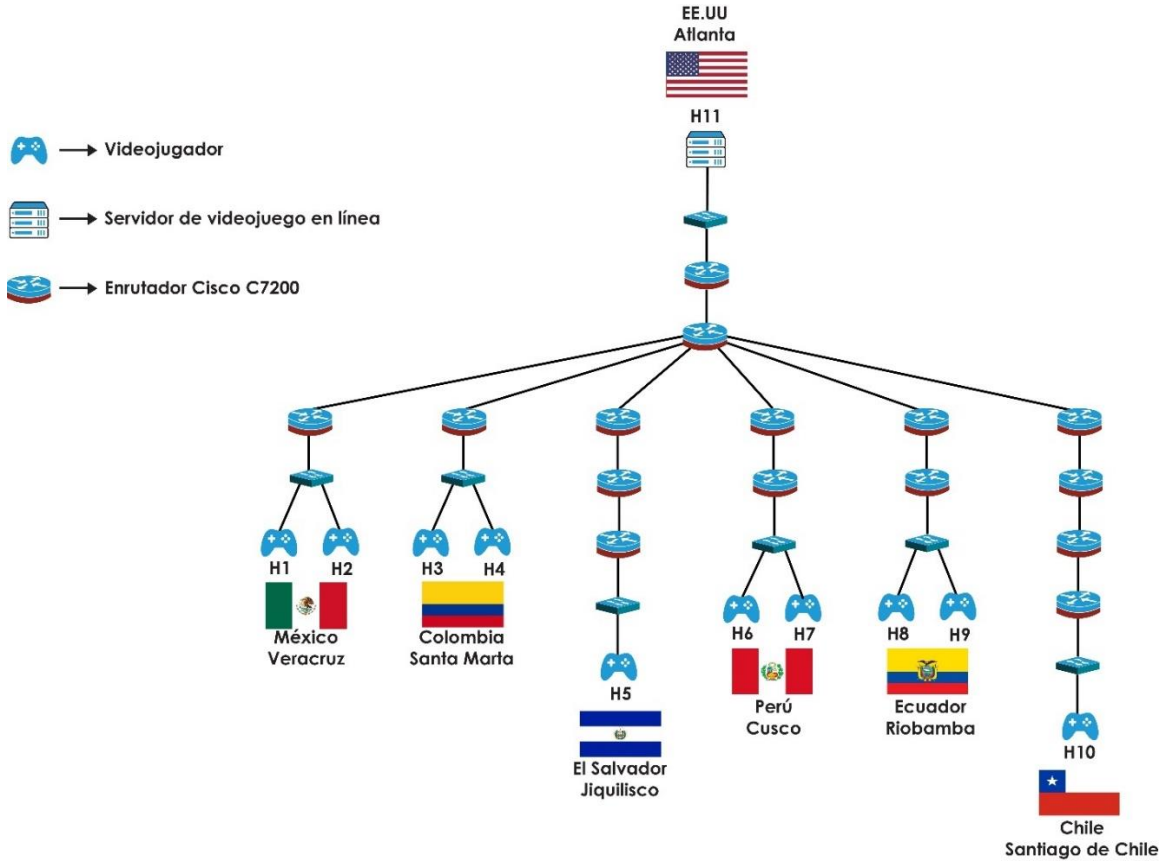


Figura 15: Topología de red usada para el entorno virtual de pruebas, construida a partir de cálculos y estimaciones de la topología con tecnología de red tradicional real usada para partidas en línea del videojuego CS:GO (Fuente: Autor)

3.2.1.1. Validación de la topología de red

Para validar de forma práctica que la topología de la Figura 15 que refleja el comportamiento real de la red usada por el videojuego CS:GO para partidas en línea, procederemos a realizar su en simulación en el software GNS3, agregando la demora física teórica obtenida en la Tabla 3 sobre los enlaces de cada videojugador con el servidor, tal y como se puede observar en la Figura 16. También se usó el enrutador Cisco C7200 y una imagen de Ubuntu 23.04 sobre Docker 23.0.4 como terminales del servidor, y los 10 videojugadores.

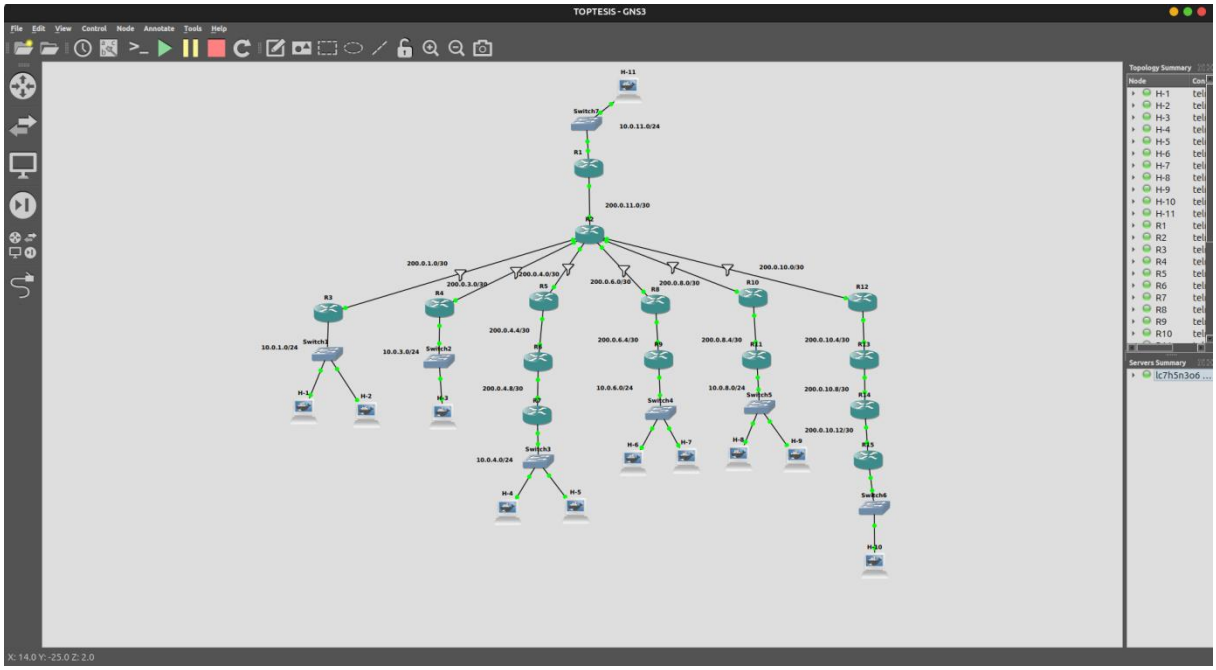


Figura 16: Topología con tecnología de red tradicional desplegada con GNS3 para simular el comportamiento de la topología de red real usada en partidas en línea del videojuego CS:GO (Fuente: Autor)

Con el comando “ping -c100” se realizaron 100 pruebas de RTT desde cada uno de los videojugadores hacia el servidor, para obtener una medida promedio de RTT de la simulación de la Figura 16, redactado en la Tabla 7.

Como se puede observar en la Tabla 7, el RTT promedio simulado varía del RTT promedio real por un máximo de -1.466 ms, por lo tanto, podemos decir que la topología es totalmente válida para nuestro entorno de pruebas, si se desea fundamentación estadística de esta afirmación, se la puede encontrar en el apartado 4.2.2.1 del capítulo 4.

Videojugador	RTT promedio real (ms)	RTT promedio simulado (ms)
H1	58.000	56.237
H2	62.000	61.782
H3	72.000	75.136
H4	74.000	80.782
H5	76.000	78.353
H6	113.000	115.825
H7	109.000	112.484
H8	92.000	90.621
H9	86.000	89.29
H10	152.000	148.146
Promedio:	89.400	90.866

Tabla 7: RTT promedio simulado de la topología propuesta

3.2.2. Simulación de partidas en línea de CS:GO con Python

Después de analizar el comportamiento real de red en CS:GO en el apartado 2.4 del capítulo 2 , replicamos su comportamiento cliente - servidor mediante programación en Python.

Partiendo de que la comunicación se realiza mediante el protocolo de tiempo real no orientado a la conexión UDP, se utilizó la librería “socket”, para crear los enlaces de conexión entre el servidor y los 10 videojugadores, y la librería “threading” para aislar las funciones que se encargarán de enviar, procesar y retransmitir los paquetes de datos.

El script del servidor servidor_partidas_en_línea.py, cuenta con dos hilos, el primero recibe y almacena los datos de cada uno de 10 videojugadores en una cola de datos, para posteriormente con un segundo hilo procesarlos y retransmitirlos en broadcast a los clientes. Otra función importante del segundo hilo, que al igual que el videojuego CS:GO, es la espera de conexión exitosa de los 10 videojugadores para iniciar la partida mediante la activación de una variable bandera, permitiendo una sincronización de los paquetes de datos.

El script que simula a cada videojugador cliente_videojugadores.py, cuenta con tres hilos, el primero recibe los paquetes de datos desde el servidor y es el encargado de recibir una bandera que indica la conexión exitosa de 10 videojugadores permitiendo el inicio del segundo hilo, encargado de enviar datos del videojugador relacionados a la partida en línea 64 veces por segundo. El último hilo se encarga de tomar 200 muestras de RTT cada 200 milisegundos y almacenarlas en un archivo de texto .txt para su posterior análisis. Cuando las 200 muestras son

tomadas al final del documento de texto se imprimen variables estadísticas como el promedio y la desviación estándar de los datos del RTT (Jitter).

3.2.2.1. Validación del modelo

Para validar los scripts de Python que simulan una partida en línea real de CS:GO los ejecutamos sobre una topología de red que interconecta de forma directa al videojugador con el servidor (ver Figura 17), con la finalidad de obtener un RTT casi igual a cero (ver Figura 18); a la par de realizar una captura del tráfico de red usando Wireshark.

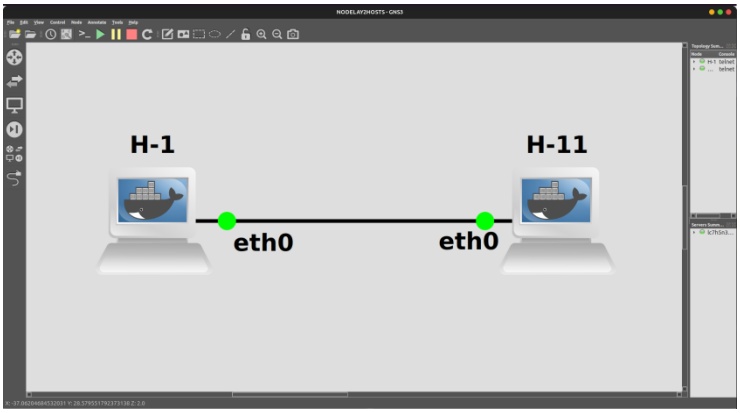


Figura 17: Topología de red con un RTT aproximado de 0ms
(Fuente: Autor)

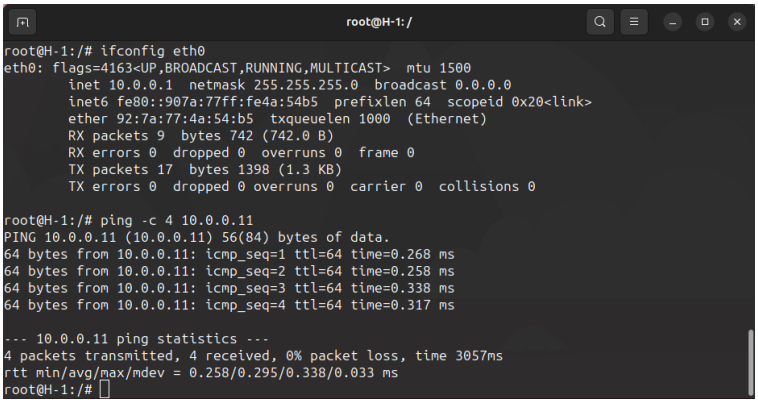


Figura 18: Prueba de RTT desde H1 a H11, para la topología de red de la Figura 17
(Fuente: Autor)

Como se puede observar, en la Figura 19 los paquetes de datos intercambiados entre el videojugador y el servidor utilizan el protocolo UDP como lo mencionado en el apartado 2.4.1.4 del capítulo 2. Además, en la Figura 20 se puede observar una media de 127 paquetes por segundo, de los cuales aproximadamente 64 son los paquetes de datos enviados por el

videojugador y 64 son la respuesta a esos paquetes por parte del servidor, verificando de esta manera que la comunicación entre el servidor y el videojugador tiene un tickrate de 64.

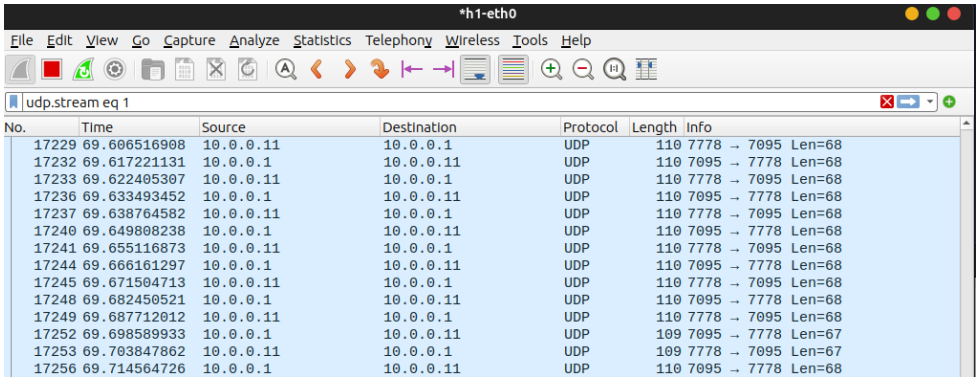


Figura 19: Captura del tráfico paquetes de datos en Wireshark, generado al ejecutar el script cliente_videojugadores.py en el videojugador y servidor_partidas_en_línea.py en el servidor (Fuente: Autor)

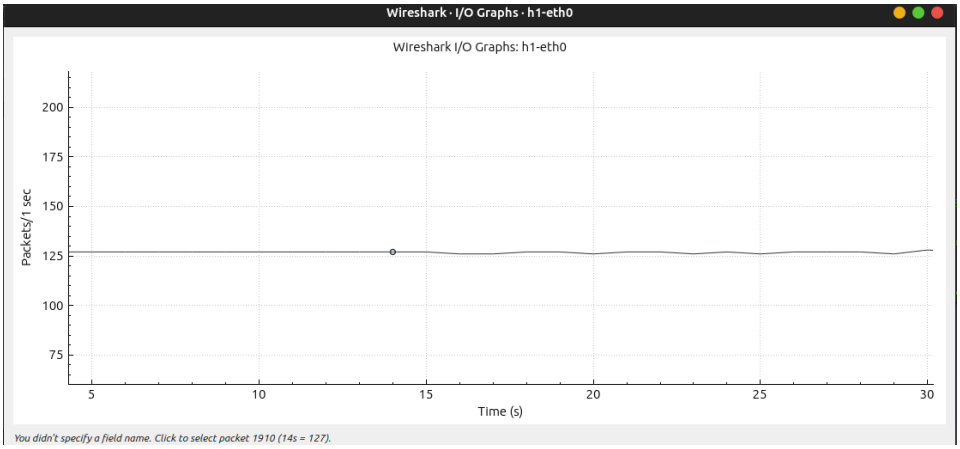


Figura 20: Captura del tráfico de paquetes visualizado en Wireshark usando su función I/O Graph, generado al ejecutar el script cliente_videojugadores.py en el videojugador y servidor_partidas_en_línea.py en el servidor (Fuente: Autor)

CAPÍTULO IV

4.1. Resultados y Discusiones

4.1.1. Despliegue del entorno virtual de pruebas con tecnología SDN

A diferencia del entorno virtual de pruebas con tecnología de red tradicional desplegado en el apartado 3.2.1.1 del capítulo 2, en la tecnología SDN los 15 enrutadores Cisco C7200 de la topología presentada en la Figura 15 son reemplazados por 15 conmutadores Open VSwitch, además, se adicionó el controlador SDN OpenDaylight versión 0.8.4, especificando en MiniEdit que c0 sea un controlador remoto ubicado en 127.0.0.1:6633 tal y como se puede observar en la Figura 21. El RTT físico de la Tabla 3 se mantuvo para cada uno de los enlaces que interconectan a los videojugadores con el servidor, ya que solo se provocará una variación del RTT lógico debido al cambio de tecnología de red.

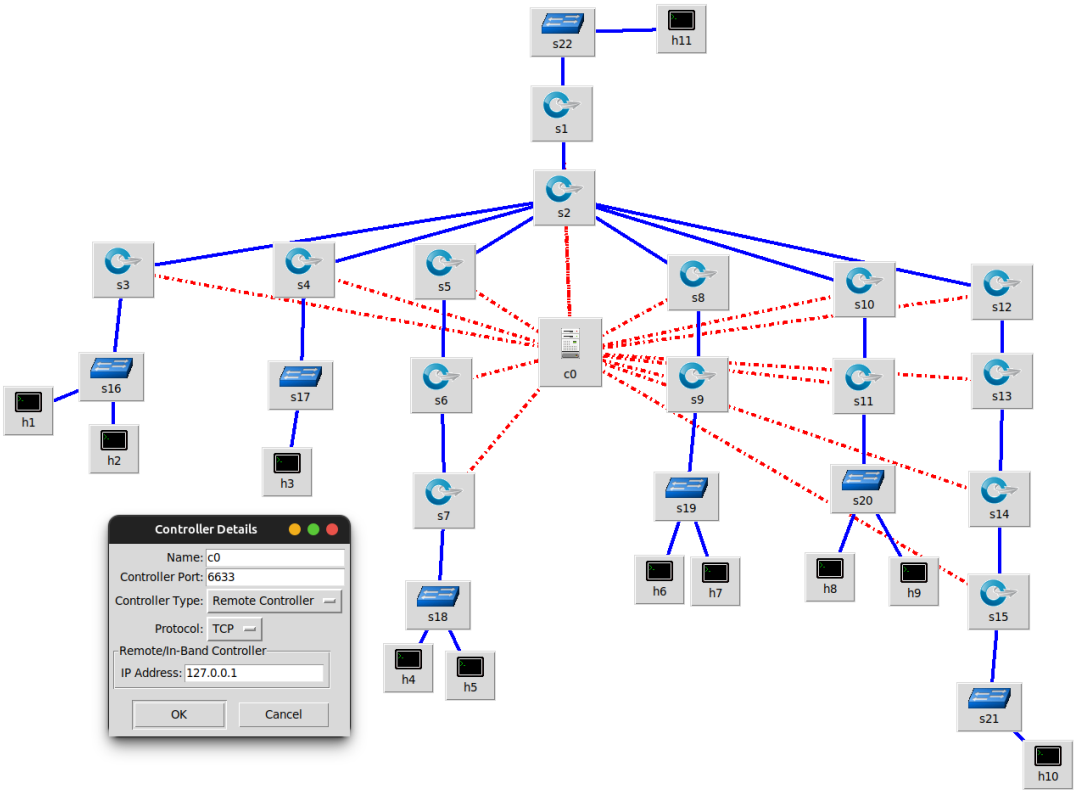


Figura 21: Topología con tecnología SDN realizada con MiniEdit para simular partidas en línea del videojuego CS:GO (Fuente: Autor)

Para levantar la topología de la Figura 21, primero se inicializó el controlador OpenDaylight como se puede observar en la Figura 22 y luego el script Topologia_SDN.py que detalla la topología de red creada con MiniEdit.

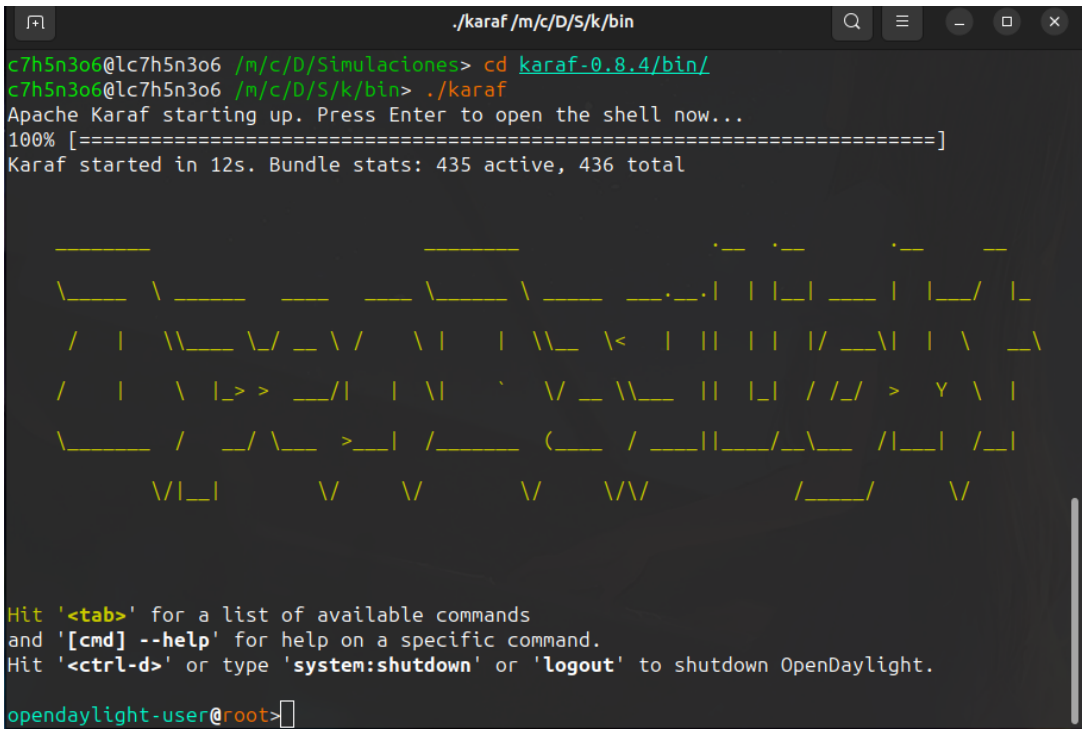


Figura 22: Inicialización del controlador OpenDaylight
(Fuente: Autor)

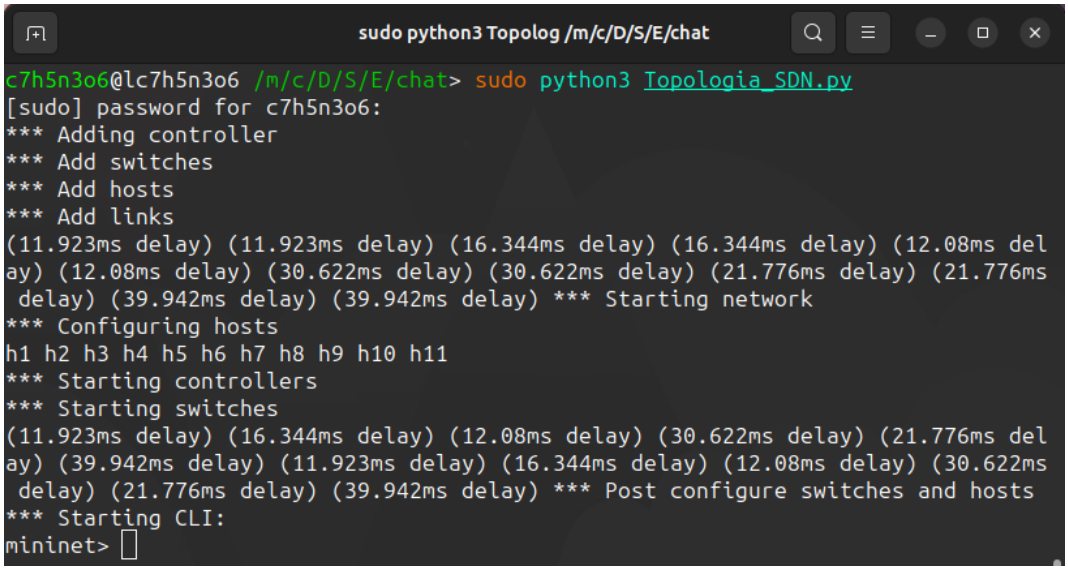


Figura 23: Inicialización del script Topologia_SDN.py
(Fuente: Autor)

Para comprobar que la topología se inicializó con éxito usamos el comando “pingall” en la terminal de Mininet como se indica en la Figura 24 o de forma gráfica visualizamos la topología de red utilizando el plano de aplicación y OpenDaylight OpenFlow Manager (ver Figura 25).

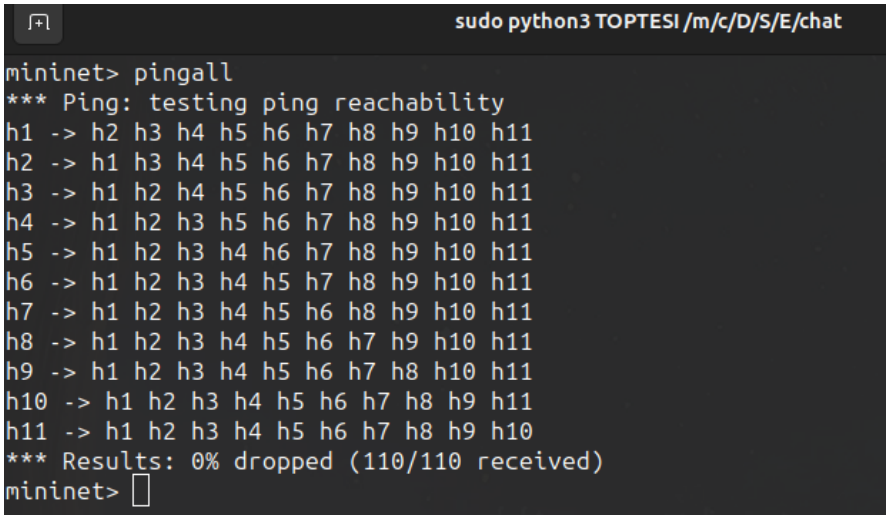


Figura 24: Prueba de conexión mediante el uso del comando “pingall” (Fuente: Autor)

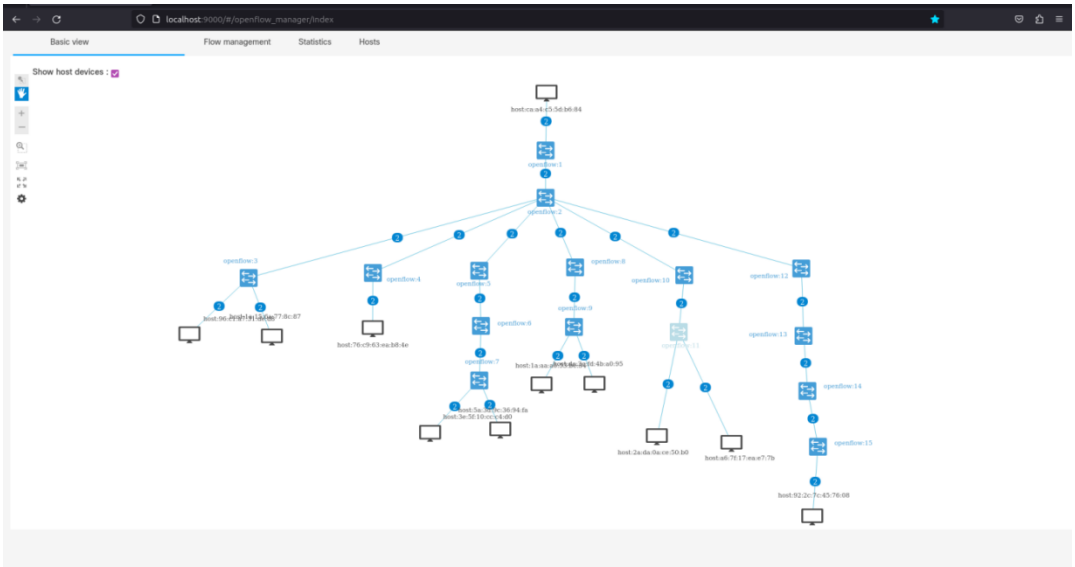


Figura 25: Topología de red del script Topología_SDN.py visualizada con OpenDaylight OpenFlow Manager (Fuente: Autor)

4.1.2. Medida del RTT y Jitter en el entorno virtual de pruebas con tecnología de red tradicional

Para realizar la medición RTT en el entorno virtual de pruebas con tecnología de red tradicional usamos la topología de la Figura 16, e inicializamos el script servidor_partidas_en_línea.py en el servidor (H11) (ver
Figura 26) y el script cliente_videojugadores.py en los videojugadores (H1, H2, H3, H4, H5, H6, H7, H8, H9, H10) (ver Figura 27), para empezar a generar tráfico de paquetes UDP en la

red, tal y como se exhibe en la Figura 28. También es importante inicializar la captura del tráfico de red con Wireshark sobre el videojugador H1, para su posterior análisis.

Figura 26: Registro del funcionamiento del script servidor_partidas_en_línea.py en el entorno virtual de pruebas con tecnología de red tradicional (Fuente: Autor)

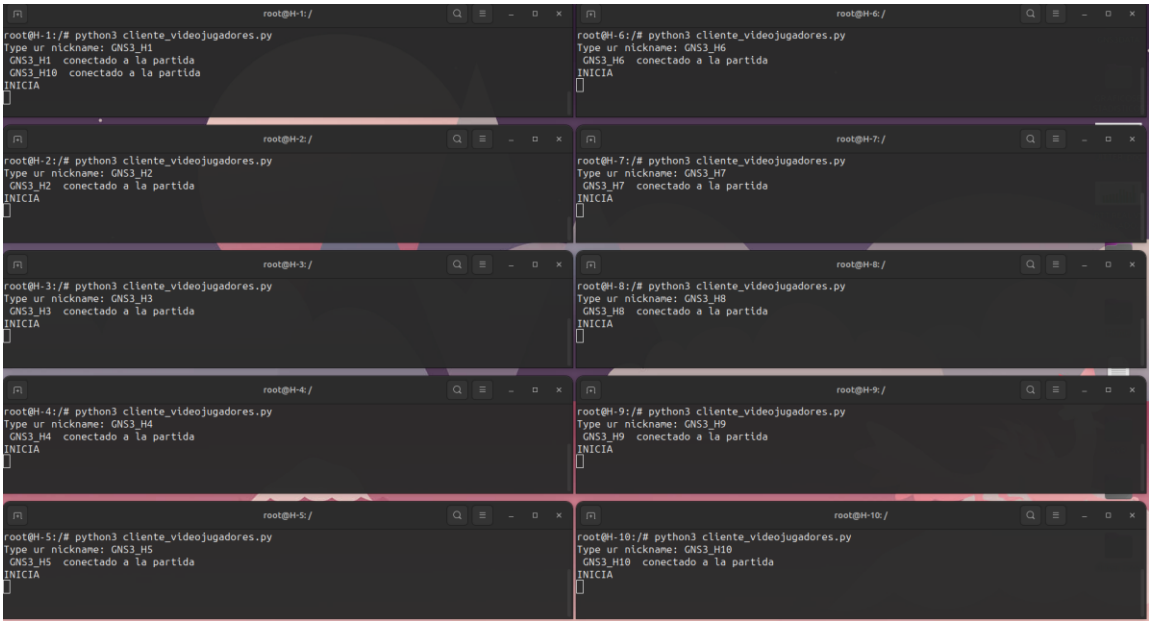


Figura 27: Registro del funcionamiento del script cliente_videojugadores.py en el entorno virtual de pruebas con tecnología de red tradicional (Fuente: Autor)

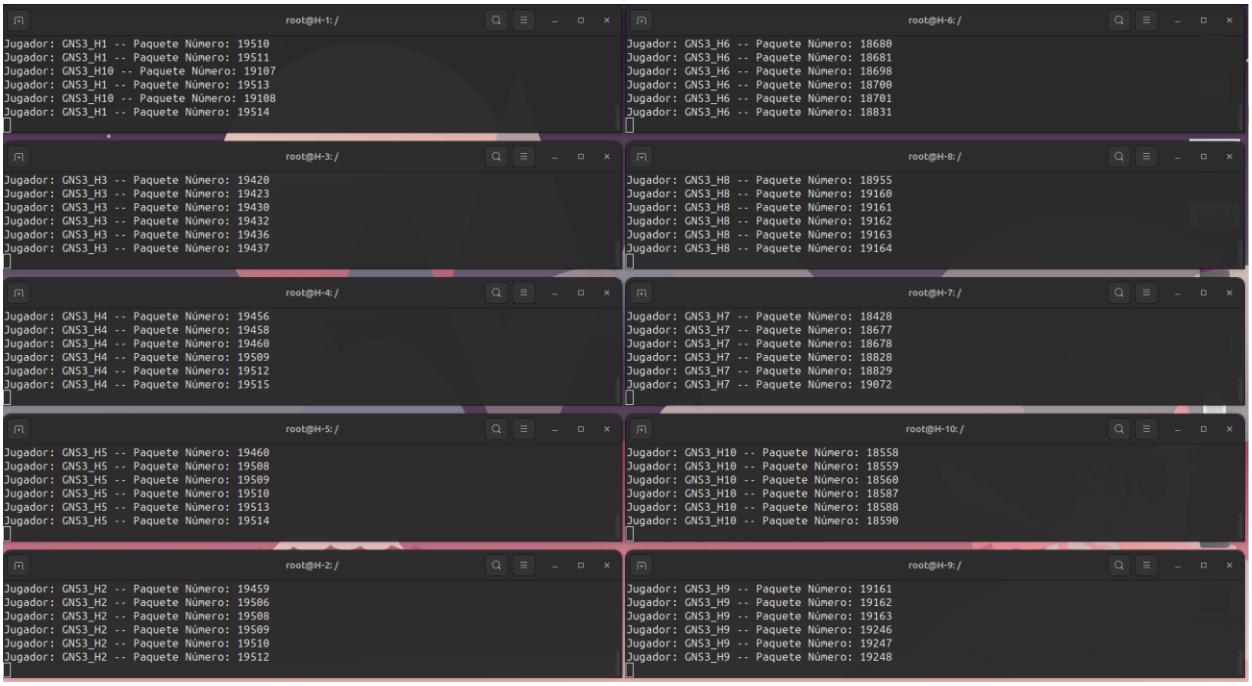


Figura 28: Registro del tráfico UDP generado a partir de la ejecución los scripts cliente_videojugadores.py y servidor_partidas_en_línea.py sobre el entorno virtual de pruebas con tecnología de red tradicional (Fuente: Autor)

Tras la recolección automática de las 200 muestras del RTT, finalizamos la conexión del servidor (ver Figura 29) para obtener acceso a los archivos de texto .txt generado por cada videojugador (ver Figura 30).

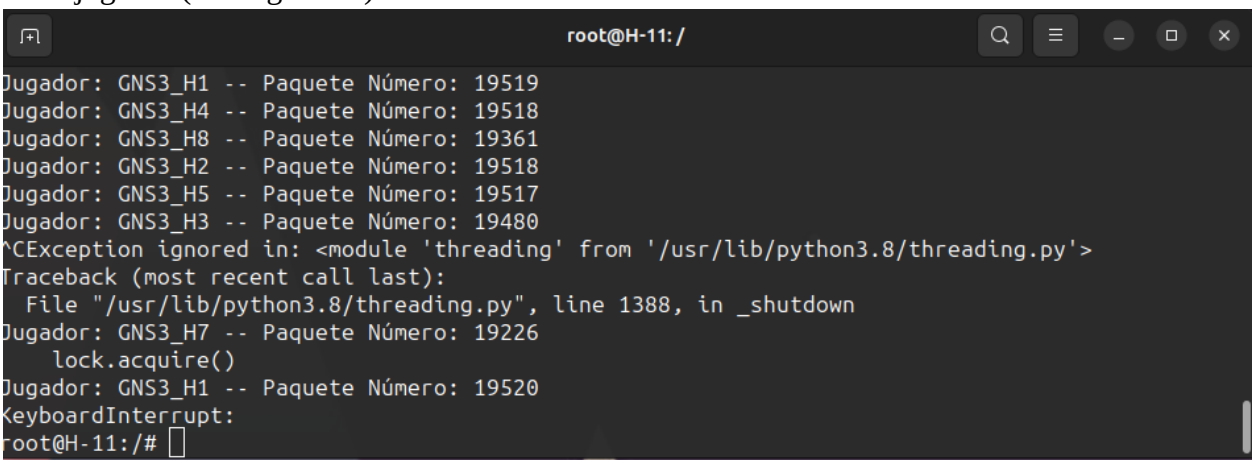


Figura 29: Finalización del servidor en el entorno virtual de pruebas con tecnología de red tradicional (Fuente: Autor)

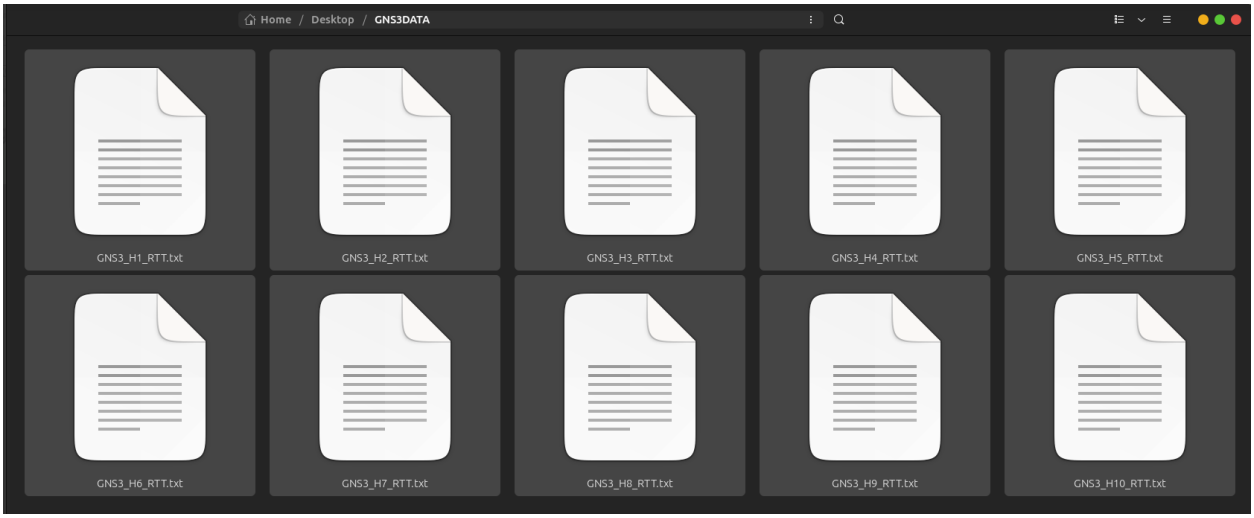


Figura 30: Archivos de texto generados por cada videojugador del entorno virtual de pruebas con tecnología de red tradicional (Fuente: Autor)

Como se puede observar en la Figura 31, cada archivo de texto contiene información de 200 pruebas RTT, así como el promedio y Jitter promedio.

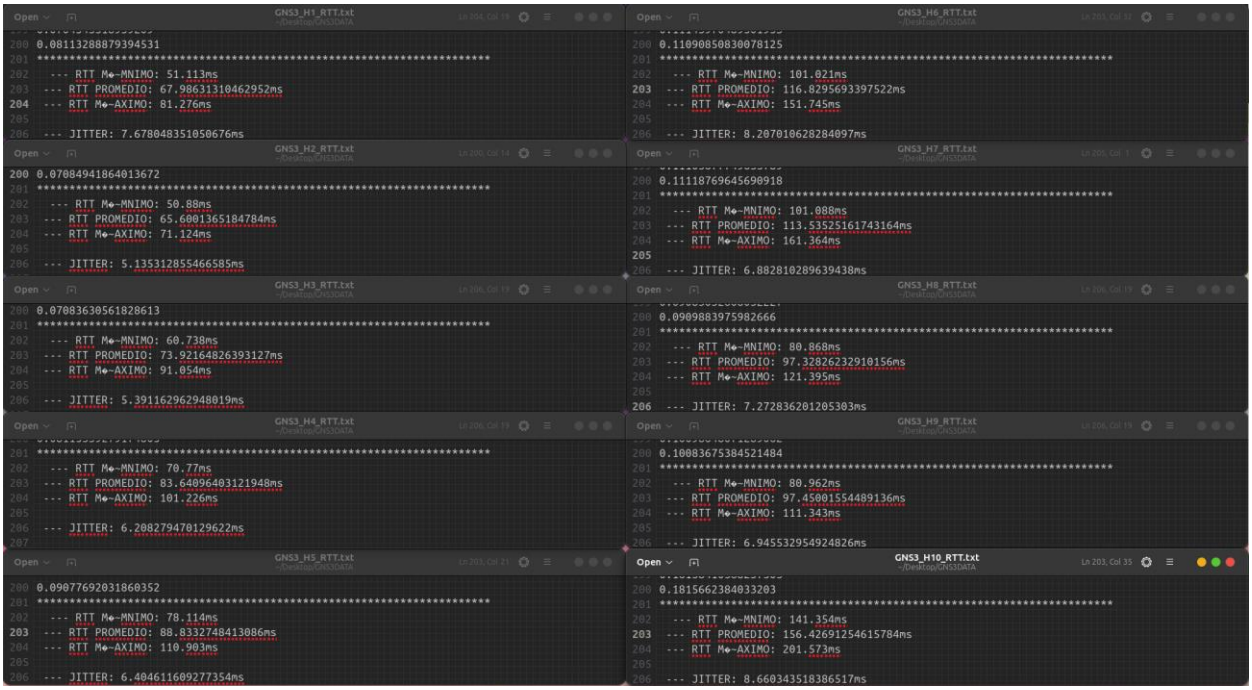


Figura 31: Contenido de los archivos de texto de la Figura 30 (Fuente: Autor)

En la Tabla 8, se resume los resultados de las 200 muestras de RTT realizadas en entorno virtual de pruebas con tecnología de red tradicional.

Videojugador	RTT promedio entorno virtual de pruebas con tecnología de red tradicional (ms)	Jitter promedio entorno virtual de pruebas con tecnología de red tradicional (ms)
H1	67.986	7.678
H2	65.600	5.135
H3	73.921	5.391
H4	83.641	6.208
H5	88.833	6.405
H6	116.829	8.207
H7	113.535	6.883
H8	97.328	7.272
H9	97.450	6.945
H10	156.427	8.660
Promedio:	93.858	6.878

Tabla 8: RTT y Jitter promedio de las 200 muestras de cada videojugador del entorno virtual de pruebas con tecnología de red tradicional

Al usar el filtro “ip.src == IPv4_H1 && ip.src == IPv4_H11 && udp” en Wireshark, obtenemos la Figura 32, en la que puede observar que la cantidad de paquetes enviados por el jugador H1 es aproximadamente 64 por segundo. Por otro lado, si aplicamos “ip.src == IPv4_H11 && ip.src == IPv4_H1 && udp” sobre la misma captura de paquetes obtenemos la Figura 33 en la que evidenciamos envió de aproximadamente 638 paquetes por segundo por parte del servidor H11 hacia el videojugador H1, debido a que los 10 videojugadores envían 64 paquetes cada segundo y el servidor reenvía los mismos en broadcast.

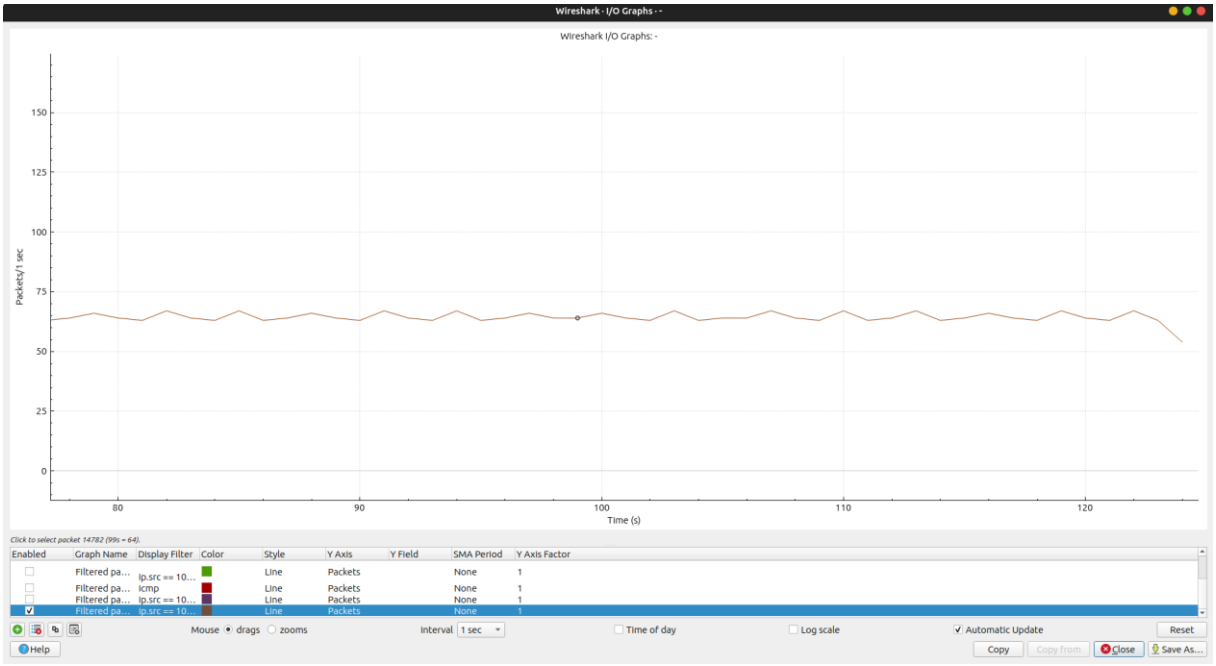


Figura 32: Paquetes enviados por segundo desde H1 durante la ejecución de los scripts cliente_videojugadores.py y servidor_partidas_en_línea.py en el entorno virtual de pruebas con tecnología de red tradicional (Fuente: Autor)

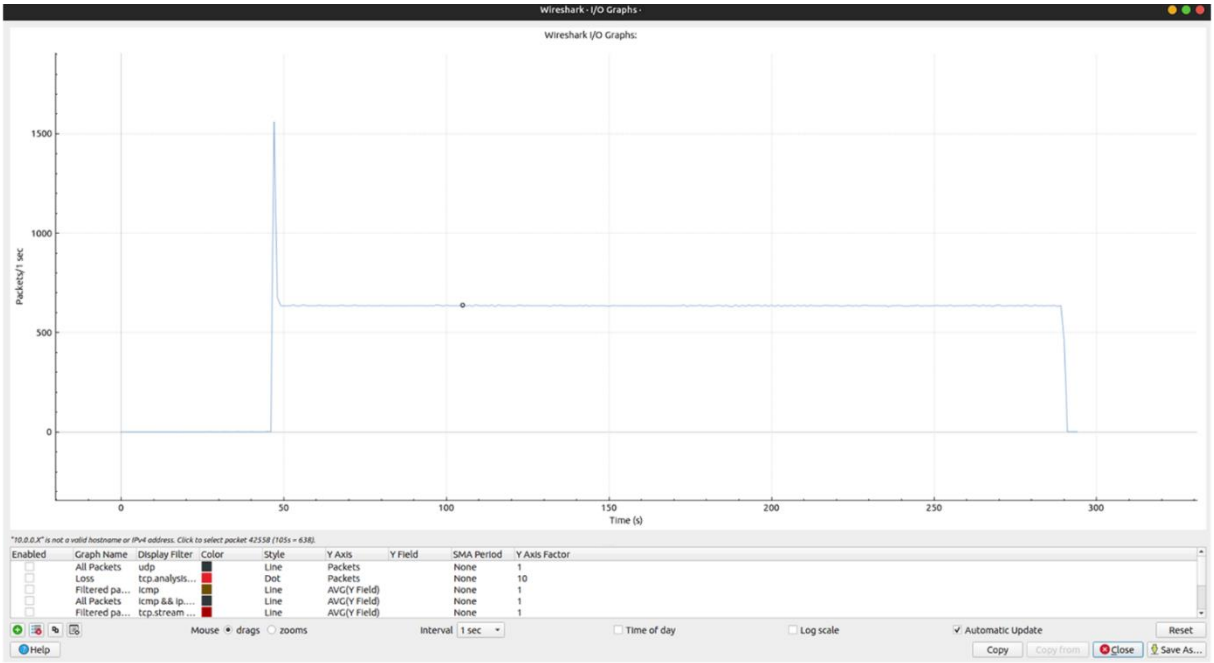


Figura 33: Paquetes enviados por segundo desde H11 hacia H1 durante la ejecución de los scripts cliente_videojugadores.py y servidor_partidas_en_línea.py en el entorno virtual de pruebas con tecnología de red tradicional (Fuente: Autor)

4.1.3. Medida del RTT y Jitter en el entorno virtual de pruebas con tecnología SDN

Al igual que en el apartado anterior, realizamos la medición del RTT en el entorno virtual de pruebas, pero usando la topología la Figura 21 y tecnología de SDN. Inicializamos el script cliente_videojugadores.py en los videojugadores (H1, H2, H3, H4, H5, H6, H7, H8, H9, H10) (ver Figura 34) y el script servidor_partidas_en_línea.py en el servidor (H11) (ver Figura 35) con ayuda de Xterm, para empezar a generar tráfico de paquetes UDP en la red (ver Figura 36) y desde el videojugador H10 capturáramos dicho tráfico usando Wireshark para su posterior análisis.

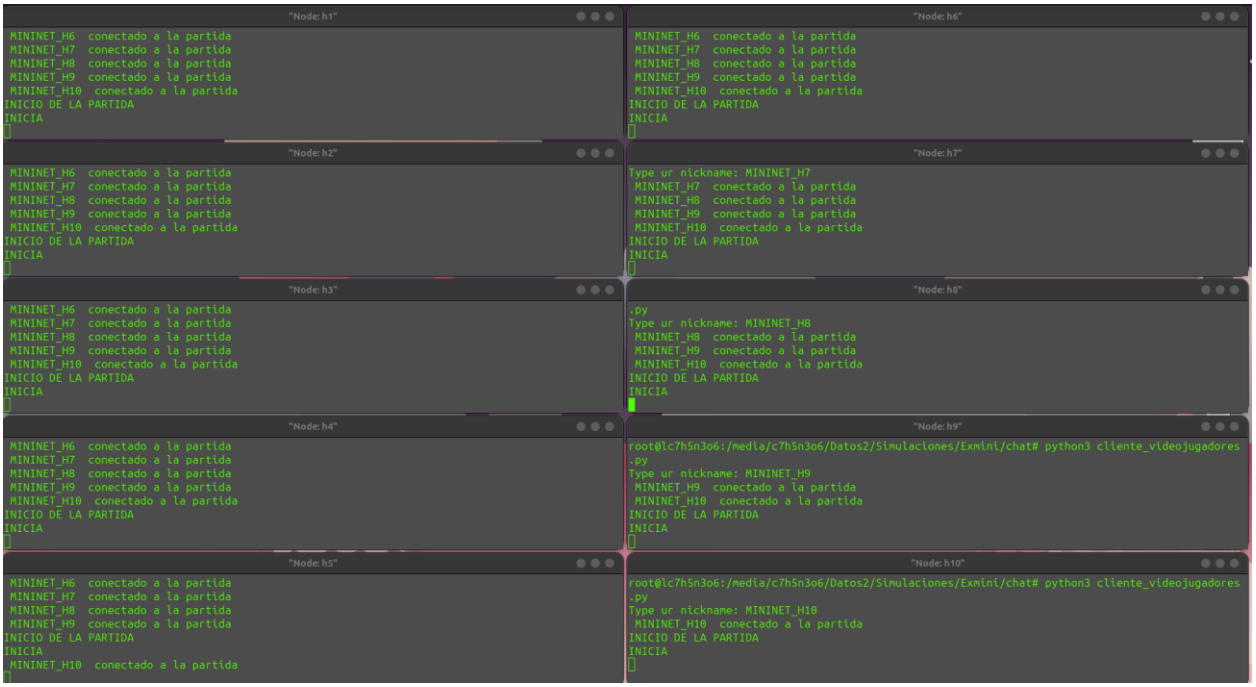


Figura 34: Registro del funcionamiento del script cliente_videojugadores.py en el entorno virtual de pruebas con tecnología SDN (Fuente: Autor)

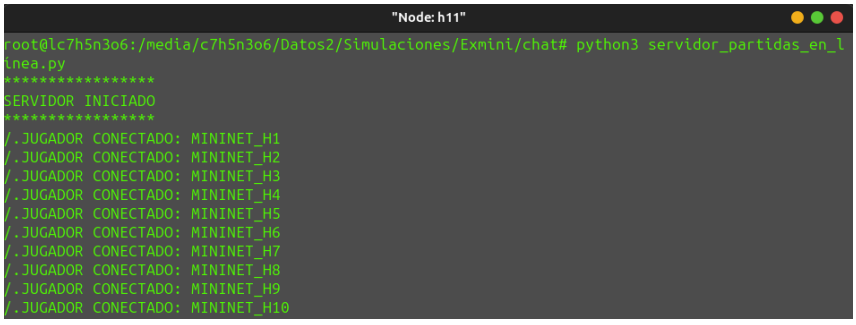


Figura 35: Registro del funcionamiento del script servidor_partidas_en_línea.py en el entorno virtual de pruebas con tecnología SDN (Fuente: Autor)

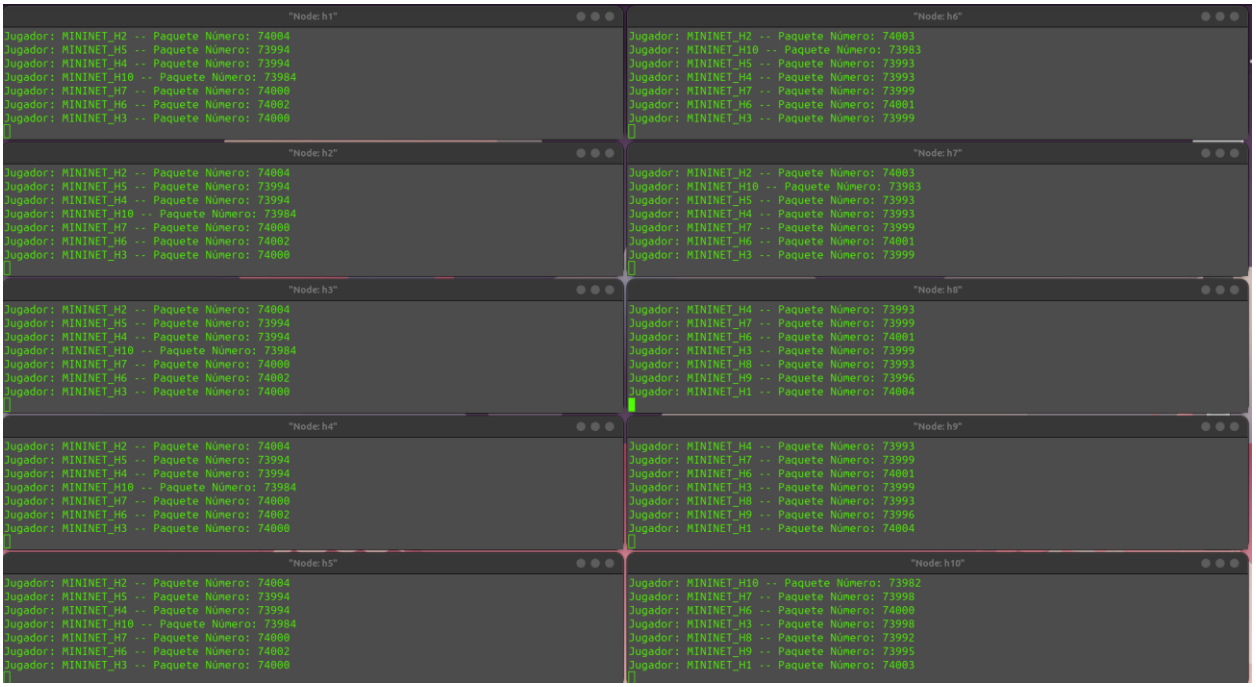


Figura 36: Registro del tráfico UDP generado a partir de la ejecución los scripts cliente_videojugadores.py y servidor_partidas_en_línea.py sobre el entorno virtual de pruebas con tecnología SDN (Fuente: Autor)

Una vez la recolección automática de las 200 muestras del RTT es finalizada, terminamos la conexión del servidor (ver
Figura 37) para obtener acceso de lectura a los archivos de texto .txt generados por los 10 videojugador (ver Figura 38).

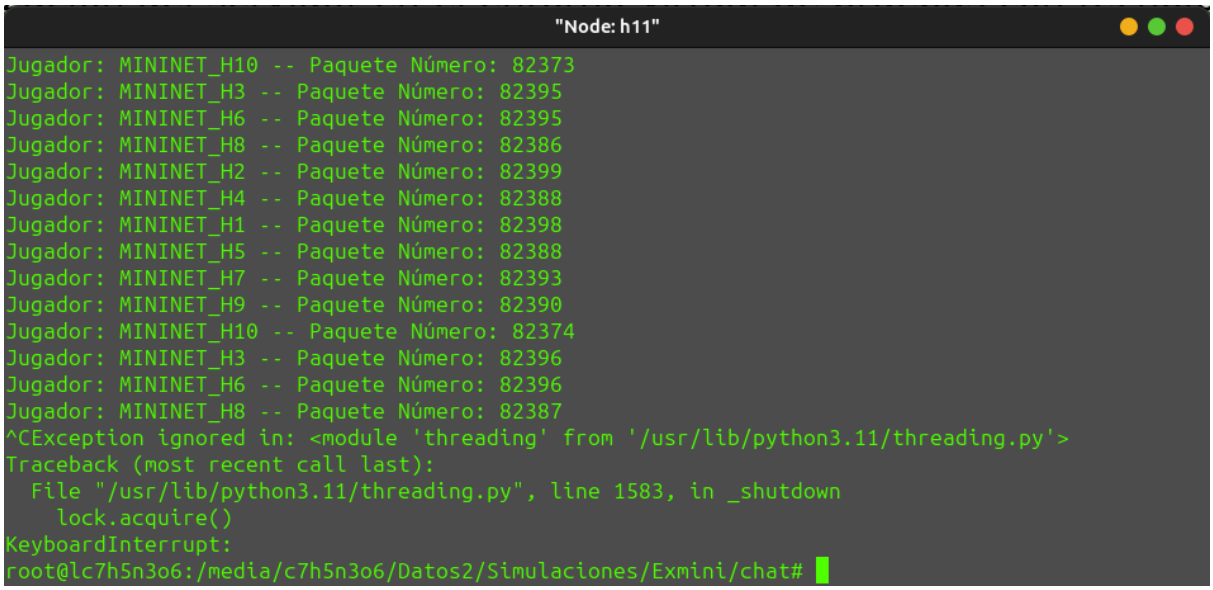


Figura 37: Finalización del servidor en el entorno virtual de pruebas con tecnología SDN (Fuente: Autor)

Figura 38: Archivos de texto generados por cada videojugador del entorno virtual de pruebas con tecnología SDN (Fuente: Autor)

Como se puede observar en la Figura 39, cada archivo de texto contiene información de 200 pruebas RTT, así como el promedio y Jitter promedio.

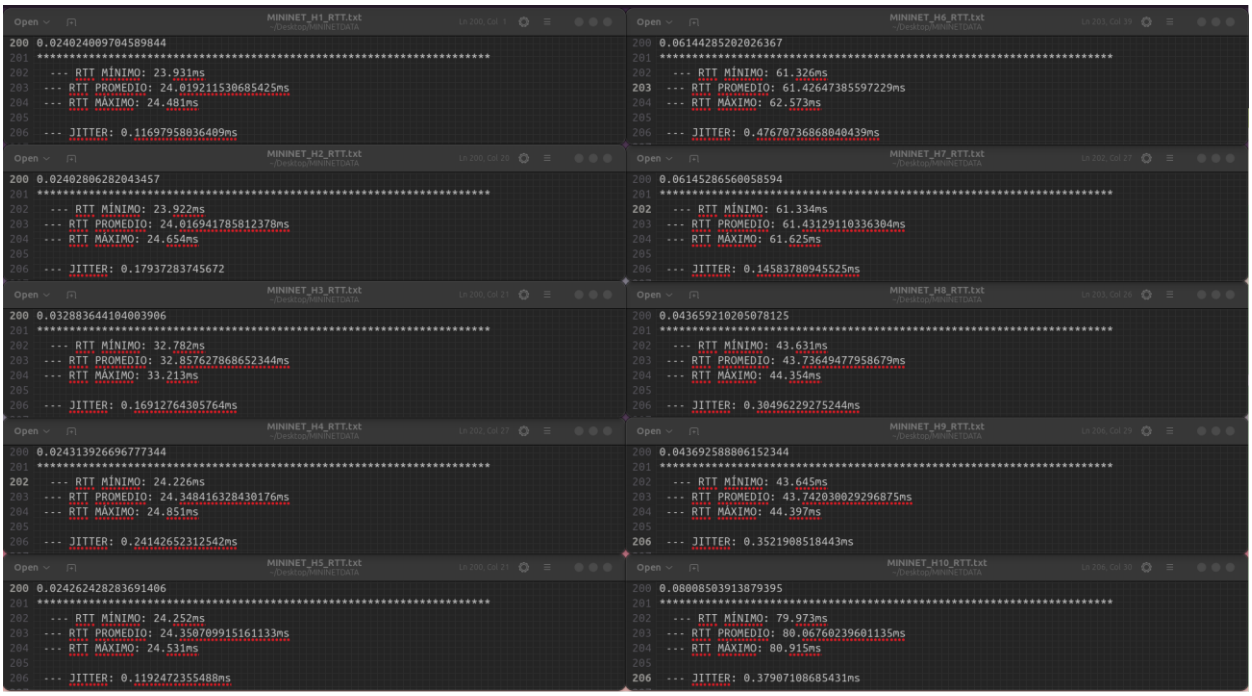


Figura 39: Contenido de los archivos de texto de la Figura 38 (Fuente: Autor)

En la Tabla 9, se resume los resultados de las 200 muestras de RTT realizadas en entorno virtual de pruebas con tecnología SDN.

Videojugador	RTT promedio entorno virtual de pruebas con tecnología SDN (ms)	RTT promedio entorno virtual de pruebas con tecnología SDN (ms)
H1	24.019	0.117
H2	24.017	0.179
H3	32.858	0.169
H4	24.348	0.241
H5	24.351	0.119
H6	61.426	0.477
H7	61.431	0.146
H8	43.736	0.305
H9	43.742	0.352
H10	80.068	0.475
Promedio:	41.999	0.258

Tabla 9: RTT y Jitter promedio de las 200 muestras de cada videojugador del entorno virtual de pruebas con tecnología SDN

Al usar el filtro “ip.src == IPv4_H10 && ip.src == IPv4_H11 && udp” en Wireshark, obtenemos la Figura 40, en la que puede observar que la cantidad de paquetes enviados por el jugador H10 hacia el servidor H11 es aproximadamente 64 por segundo. Si ejecutamos el filtro “ip.src == IPv4_H11 && ip.src == IPv4_H10 && udp” en la misma captura de paquetes de datos UDP obtenemos la Figura 41, en la vizualizamos un envió de aproximadamente 634 paquetes por segundo desde el servidor H11 hacia el videojugador H10, debido al broadcast generado por el servidor (Fuente: Autor)

Figura 40: Paquetes enviados por segundo desde H10 durante la ejecución de los scripts cliente_videojugadores.py y servidor_partidas_en_línea.py en el entorno virtual de pruebas con tecnología SDN (Fuente: Autor)

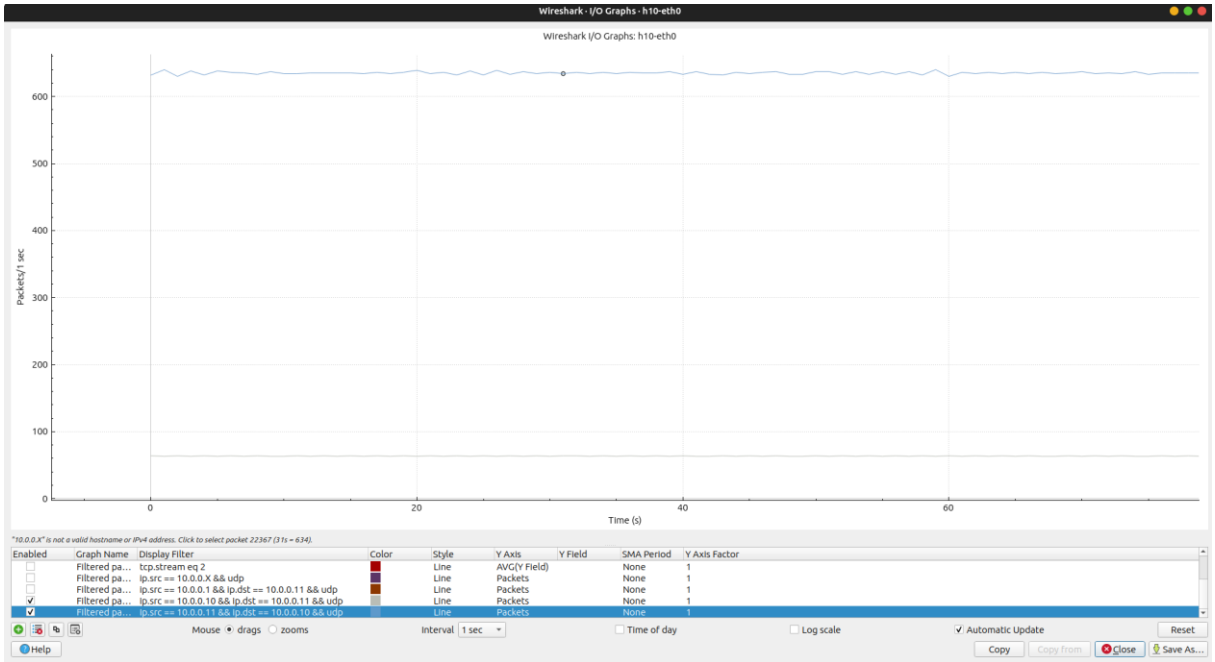


Figura 41: Paquetes enviados por segundo desde H11 hacia H10 durante la ejecución de los scripts cliente_videojugadores.py y servidor_partidas_en_línea.py en el entorno virtual de pruebas con tecnología SDN (Fuente: Autor)

4.2. Análisis del RTT y Jitter, tras el cambio de tecnología de red tradicional con tecnología SDN en el entorno virtual de pruebas

En el siguiente apartado se evidenciará de forma estadística la existencia o ausencia de una diferencia significativa entre los resultados de las pruebas de RTT y Jitter promedio obtenidos del entorno virtual de pruebas con tecnología de red tradicional, y del entorno virtual de pruebas con tecnología SDN. Para el efecto, se utilizará el software IBM SPSS Statistics en su versión 29.0.1.0.

4.2.1. Prueba de normalidad

Hipótesis:

H_0: Los datos siguen una distribución normal
H_1: Los datos no siguen una distribución normal

Tipo de prueba estadística:

Usamos la prueba estadística de Shapiro-Wilk, ya que nuestro número de muestras es 10 (correspondiente a los 10 videojugadores), cumpliendo con su condición de que el número de muestras sean menor o igual a 30.

Condiciones de aceptación y rechazo de la hipótesis nula:

Si p-valor ≥ 0.05, se acepta H_0 y se rechaza H_1
Si p-valor < 0.05, se acepta H_1 y se rechaza H_0

Tests of Normality

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
RTT_TOPOLOGIA_REAL	.181	10	.200*	.900	10	.219
RTT_TOPOLOGIA_SIMULADA	.204	10	.200*	.933	10	.474

Figura 42: Prueba de normalidad para el RTT de la topología de red real y para el RTT de la topología de red simulada (Fuente: Autor)

Tests of Normality						
	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
RTT_TRADICIONAL	.180	10	.200*	.909	10	.274
JITTER_TRADICIONAL	.104	10	.200*	.975	10	.935
RTT_SDN	.212	10	.200*	.856	10	.068
JITTER_SDN	.216	10	.200*	.869	10	.098

Figura 43: Prueba de normalidad el RTT y Jitter de cada entorno virtual de pruebas (Fuente: Autor)

En la Figura 42 y Figura 43 se presentan los resultados para las pruebas de normalidad para cada una de las variables. En la misma, se puede evidenciar que el p-valor para la prueba de Shapiro-Wilk en cada una de las variables es mayor de 0.05, por lo tanto, aceptamos la hipótesis nula H_0, que refleja una distribución normal de los datos de cada muestra, con nivel de confianza del 95%.

4.2.2. Pruebas de hipótesis

4.2.2.1. Prueba de hipótesis para la variable RTT de la topología de red real usada para partidas en línea de CS:GO, y su simulación en GNS3

Hipótesis:

$$H_0: \mu_1 = \mu_2$$
$$H_1: \mu_1 \neq \mu_2$$

- H_0: La media poblacional del RTT obtenido en la topología de red real usada para partidas en línea de CS:GO es igual a la media poblacional del RTT obtenido al usar la misma topología de red, pero simulada en GNS3.

- H_1: La media poblacional del RTT obtenido en la topología de red real usada para partidas en línea de CS:GO no es igual a la media poblacional del RTT obtenido al usar la misma topología de red, pero simulada en GNS3.

Tipo de prueba:

Se empleará una prueba T de Student para muestras independientes con dos colas. Su estadístico se representa con la variable t.

Condiciones de aceptación y rechazo de la hipótesis nula:

$$\text{Si } t < t_{\left(1-\frac{\alpha}{2}\right),(n_1+n_2-2)}\text{, se acepta } H_0 \text{ y se rechaza } H_1$$
$$\text{Si } t > t_{\left(1-\frac{\alpha}{2}\right),(n_1+n_2-2)}\text{, se acepta } H_1 \text{ y se rechaza } H_0$$

Sabiendo que $n_1 = n_2 = 10$ y $\alpha = 0.05$, obtenemos un valor crítico para $t_{\left(1-\frac{\alpha}{2}\right),(n_1+n_2-2)}$ de 2.201.

Independent Samples Test

		Levene's Test for Equality of Variances		t-test for Equality of Means							
						Significance				95% Confidence Interval of the Difference	
		F	Sig.	t	df	One-Sided p	Two-Sided p	Mean Difference	Std. Error Difference	Lower	Upper
RTT	Equal variances assumed	.014	.907	-.116	18	.454	.909	-1.465600	12.587019	-27.909945	24.978745
	Equal variances not assumed			-.116	17.985	.454	.909	-1.465600	12.587019	-27.911511	24.980311

Figura 44: Prueba T de Student para muestras independientes del RTT obtenido en la topología de red real usada para partidas en línea de CS:GO y el RTT obtenido al usar la misma topología de red, pero simulada en GNS3 (Fuente: Autor)

En la Figura 44, obtenemos un valor para el estadístico t igual a -0.116, por lo tanto, con un nivel de confianza del 95% aceptamos la hipótesis nula (H_0), y concluimos que la media poblacional del RTT obtenido en la topología de red real usada para partidas en línea de CS:GO es igual a la media poblacional del RTT obtenido al usar la misma topología de red, pero simulada en GNS3.

De manera visual en la podemos volver a reiterar la aceptación de la hipótesis nula (H_0) con el uso de un gráfico con barras de error. En la Figura 45, podemos observar que los la mayoría de valores medios del RTT obtenido en la topología de red real usada para partidas en línea de CS:GO, está dentro del intervalo de confianza de la media del RTT obtenido al usar la misma topología de red, pero simulada en GNS3, reiterando la aceptación de la hipótesis nula (H_0).

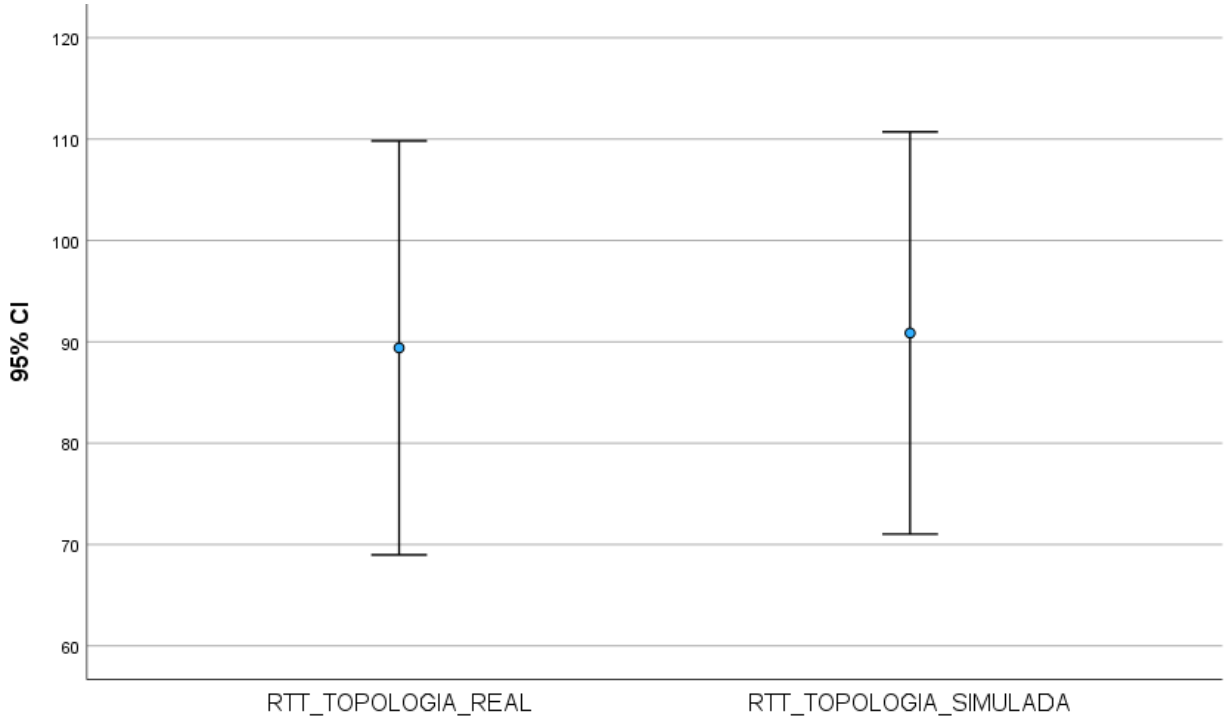


Figura 45: Gráfico con barras de error del intervalo de confianza del RTT obtenido en la topología de red real usada para partidas en línea de CS:GO y RTT obtenido al usar la misma topología de red, pero simulada en GNS3 (Fuente: Autor)

4.2.2.2. Prueba de hipótesis para la variable RTT del entorno virtual de pruebas con tecnología de red tradicional y del entorno virtual de pruebas con tecnología SDN

Hipótesis:

$$H_0: \mu_1 \leq \mu_2$$
$$H_1: \mu_1 > \mu_2$$

- H_0: La media poblacional del RTT obtenido en el entorno virtual de pruebas con tecnología de red tradicional es menor o igual a la media poblacional del RTT obtenido en el entorno virtual de pruebas con tecnología SDN.

- H_1: La media poblacional del RTT obtenido en el entorno virtual de pruebas con tecnología de red tradicional es mayor a la media poblacional del RTT obtenido en el entorno virtual de pruebas con tecnología SDN.

Tipo de prueba:

Se empleará una prueba T de Student para muestras relacionadas con una cola a la derecha. Su estadístico se representa con la variable t.

Condiciones de aceptación y rechazo de la hipótesis nula:

$$\text{Si } t < t_{(1-\alpha),(n-1)}\text{, se acepta } H_0 \text{ y se rechaza } H_1$$
$$\text{Si } t > t_{(1-\alpha),(n-1)}\text{, se acepta } H_1 \text{ y se rechaza } H_0$$

Sabiendo que $n = 10$ y $\alpha = 0.05$, obtenemos un valor crítico para $t_{(1-\alpha),(n-1)}$ de 1.833.

Paired Samples Test

		Paired Differences					t	df	Significance	
					95% Confidence Interval of the Difference					
		Mean	Std. Deviation	Std. Error Mean	Lower	Upper			One-Sided p	Two-Sided p
Pair 1	RTT_TRADICIONAL - RTT_SDN	54.045000	10.872686	3.438245	46.267149	61.822851	15.719	9	<.001	<.001

Figura 46: Prueba T de Student para muestras relacionadas del RTT obtenido en el entorno virtual de pruebas con tecnología de red tradicional y RTT obtenido en el entorno virtual de pruebas con tecnología SDN (Fuente: Autor)

En la Figura 46 obtenemos un valor para el estadístico t igual a 15.719, por lo tanto, con un nivel de confianza del 95% aceptamos la hipótesis alterna (H_1), y concluimos que la media poblacional del RTT obtenido en el entorno virtual de pruebas con tecnología de red tradicional es mayor a la media poblacional del RTT obtenido en el entorno virtual de pruebas con tecnología SDN.

De manera visual en la podemos volver a reiterar la aceptación de la hipótesis alterna (H_1) con el uso de un gráfico con barras de error. Al no existir ningún valor medio del RTT obtenido en el entorno virtual de pruebas con tecnología de red tradicional, dentro del intervalo de confianza de la media del RTT obtenido en el entorno virtual de pruebas con tecnología SDN se rechaza la hipótesis nula (H_0) y se acepta la hipótesis alterna (H_1).

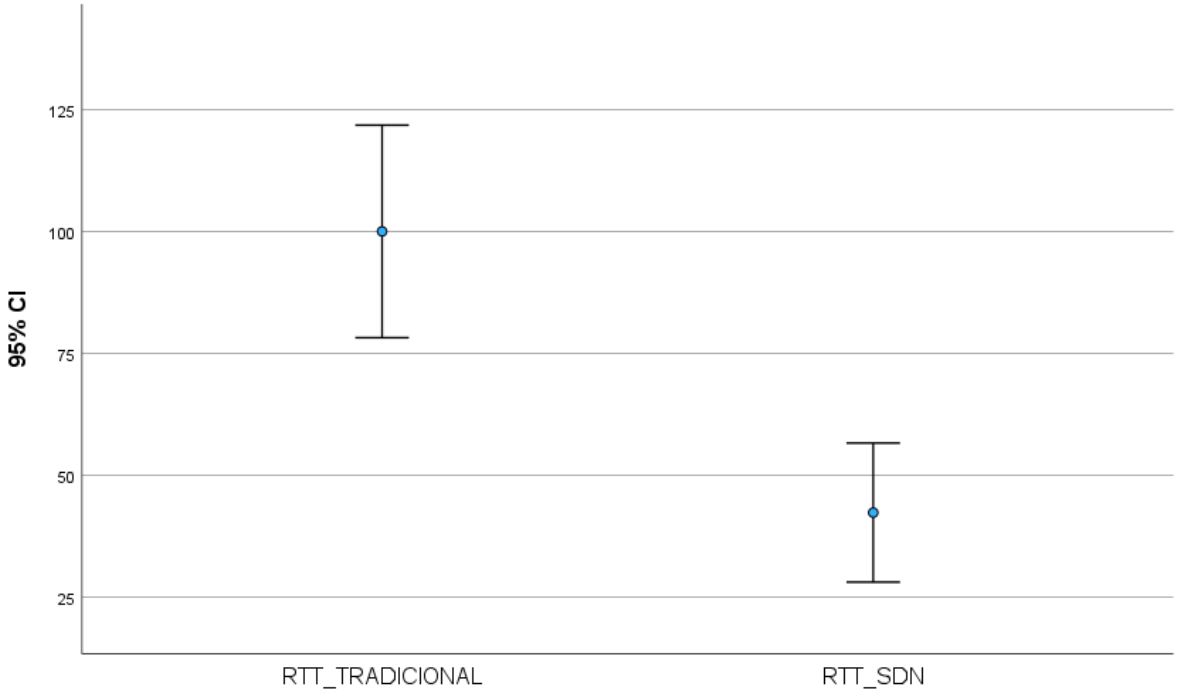


Figura 47: Gráfico con barras de error del intervalo de confianza del RTT obtenido en el entorno virtual de pruebas con tecnología de red tradicional y RTT obtenido en el entorno virtual de pruebas con tecnología SDN (Fuente: Autor)

4.2.2.3. Prueba de hipótesis para la variable Jitter del entorno virtual de pruebas con tecnología de red tradicional y del entorno virtual de pruebas con tecnología SDN

Hipótesis:

$$H_0: \mu_1 \leq \mu_2$$
$$H_1: \mu_1 > \mu_2$$

- H_0: La media poblacional del Jitter obtenido en el entorno virtual de pruebas con tecnología de red tradicional es menor o igual a la media poblacional del Jitter obtenido en el entorno virtual de pruebas con tecnología SDN.

- H_1: La media poblacional del Jitter obtenido en el entorno virtual de pruebas con tecnología de red tradicional es mayor a la media poblacional del Jitter obtenido en el entorno virtual de pruebas con tecnología SDN.

Tipo de prueba:

Se empleará una prueba T de Student para muestras relacionadas con una cola a la derecha. Su estadístico se representa con la variable t.

Condiciones de aceptación y rechazo de la hipótesis nula:

$$\text{Si } t < t_{(1-\alpha),(n-1)}\text{, se acepta } H_0 \text{ y se rechaza } H_1$$
$$\text{Si } t > t_{(1-\alpha),(n-1)}\text{, se acepta } H_1 \text{ y se rechaza } H_0$$

Sabiendo que $n = 10$ y $\alpha = 0.05$, obtenemos un valor crítico para $t_{(1-\alpha),(n-1)}$ de 1.833.

Paired Samples Test

		Paired Differences					t	df	Significance	
					95% Confidence Interval of the Difference					
		Mean	Std. Deviation	Std. Error Mean	Lower	Upper			One-Sided p	Two-Sided p
Pair 1	JITTER_TRADICIONAL - JITTER_SDN	6.620400	1.052135	.332714	5.867748	7.373052	19.898	9	<.001	<.001

Figura 48: Prueba T de Student para muestras relacionadas del RTT obtenido en el entorno virtual de pruebas con tecnología de red tradicional y RTT obtenido en el entorno virtual de pruebas con tecnología SDN (Fuente: Autor)

En la Figura 48 obtenemos un valor para el estadístico t igual a 19.898, por lo tanto, con un nivel de confianza del 95% aceptamos la hipótesis alterna (H_1), y concluimos que la media poblacional del Jitter obtenido en el entorno virtual de pruebas con tecnología de red tradicional es mayor a la media poblacional del Jitter obtenido en el entorno virtual de pruebas con tecnología SDN.

De manera visual en la podemos volver a verificar la aceptación de la hipótesis alterna (H_1) con el uso de un gráfico con barras de error. Como se puede observar en la Figura 49, al no existir ningún valor medio del Jitter obtenido en el entorno virtual de pruebas con tecnología de red tradicional, dentro del intervalo de confianza de la media del Jitter obtenido en el entorno virtual de pruebas con tecnología SDN se rechaza la hipótesis nula (H_0) y se acepta la hipótesis alterna (H_1).

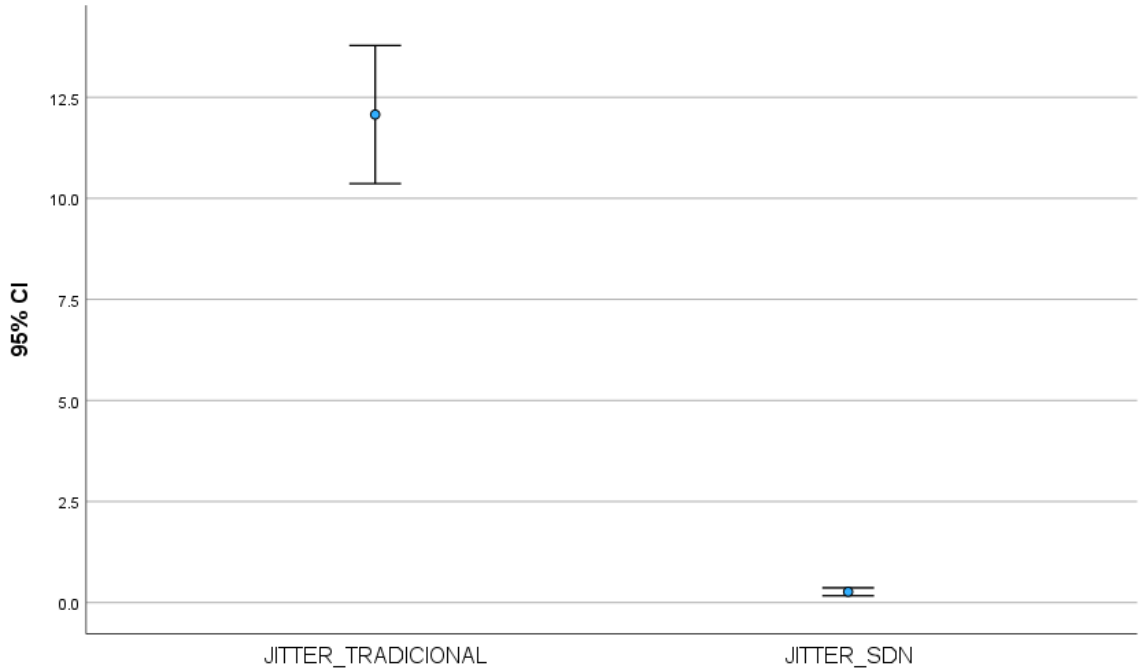


Figura 49: Gráfico con barras de error del intervalo de confianza del Jitter obtenido en el entorno virtual de pruebas con tecnología de red tradicional y Jitter obtenido en el entorno virtual de pruebas con tecnología SDN (Fuente: Autor)

CAPÍTULO V

5.1. Conclusiones

El análisis teórico de las tecnologías (tradicional y SDN) y topologías de red fue fundamental en la realización de esta investigación, ya que nos permitió replicar exitosamente mediante software el comportamiento de la topología con tecnología de red tradicional real usada para partidas en línea del videojuego CS:GO desde varios países de Latinoamérica, usando el concepto del RTT lógico y el RTT físico.

Implementamos un ambiente de simulación de videojuegos creando dos entornos virtuales. A nivel de red, en el primer entorno virtual se utilizó la topología con tecnología de red tradicional replicada en GNS3 a partir de la red real y scripts hechos en Python para simular el tráfico de red generado durante la ejecución de una partida en línea del videojuego CS:GO. En el segundo entorno virtual se utilizó la misma topología de red y scripts de Python del primer entorno virtual, pero con reemplazo de la tecnología de red tradicional por tecnología SDN usando Mininet y el controlador OpenDaylight.

Cuando realizamos métricas estadísticas sobre los datos recolectados en ambos entornos virtuales obtuvimos que la media aritmética del RTT y Jitter tomados del primer entorno es de 93.858 ms, y 6.878 ms respectivamente, mientras, que en el segundo entorno virtual obtuvimos una media aritmética para el RTT de 41.999 ms y para el Jitter 0.258 ms. Siendo el RTT del primer entorno 2.235 veces más grande que el RTT del segundo entorno, mientras que el Jitter del primer entorno es 26.658 veces más grande que el Jitter del segundo entorno. Si volvemos al entorno real, en el cual están inspiradas estas simulaciones podemos decir, que un cambio en la tecnología de red mejoraría de forma drástica el RTT y estabilidad en la conexión de los videojuegos en línea, ya si lo comparamos con la media aritmética del RTT físico teórico (44.229ms) el margen agregado a la red por el RTT lógico se vuelve prácticamente despreciable, es decir, que la única limitante para tener un RTT bajo estaría dado por las distancias físicas entre cada videojugador y el servidor del videojuego.

Para ambos entornos virtuales las pruebas estadísticas de cada una de las variables nos llevan a la conclusión de que el rendimiento de una red definida por software, es muy superior al rendimiento de una red tradicional de datos, cuando son usadas para entornos de videojuegos en línea.

5.2. Recomendaciones

Para futuros trabajos, se recomienda utilizar la tecnología SDN, para investigar sobre la virtualización de funciones de red (NFV), virtualización de infraestructura con OpenStack y Docker, ya que gracias a todas estas tecnologías cada vez es más viable, económico y rentable desplegar servidores en más regiones del mundo. Con la finalidad de puede reducir aún más el RTT presente en las redes de datos.

Si futuras investigaciones se desarrollarán sobre el sistema GNU/Linux, es muy recomendable usar las versiones de destrucción LTS, ya que sus versiones más recientes presentan muchos bugs en el sistema y la documentación para solventarlos es muy escaza.

BIBLIOGRAFÍA

[1] C. Quimbayo Rodriguez, “PROPUESTA METODOLÓGICA PARA LA SELECCIÓN DE CONTROLADORES DE REDES SDN A NIVEL EMPRESARIAL,” UNIVERSIDAD SANTO TOMAS, BOGOTÁ D.C., 2020.

[2] “Atari 2600 VCS GameLine Master Module : scans, dump, download, screenshots, ads, videos, catalog, instructions, roms.” Accessed: Jan. 22, 2023. [Online]. Available: http://www.atarimania.com/game-atari-2600-vcs-gameline-master-module_20315.html

[3] “Sega Mega Modem - Sega Retro.” Accessed: Jan. 22, 2023. [Online]. Available: https://segaretro.org/Sega_Mega_Modem

[4] “NetLink Internet Modem - Sega Retro.” Accessed: Jan. 22, 2023. [Online]. Available: https://segaretro.org/NetLink_Internet_Modem

[5] “La OMS caracteriza a COVID-19 como una pandemia - OPS/OMS | Organización Panamericana de la Salud.” Accessed: Jan. 22, 2023. [Online]. Available: https://www.paho.org/es/noticias/11-3-2020-oms-caracteriza-covid-19-como-pandemia

[6] “Steam (App 753) · Steam Charts · SteamDB.” Accessed: Jan. 22, 2023. [Online]. Available: https://steamdb.info/app/753/charts/

[7] J. Días, “ANÁLISIS COMPARATIVO DE INTERCAMBIO DE PAQUETES ENTRE REDES DEFINIDAS POR SOFTWARE Y REDES TRADICIONALES, APOYADO EN LA HERRAMIENTA DE SIMULACIÓN MININET,” Universidad Santo Tomás Colombia, 2020.

[8] N. Fernández and P. Carmo, “Historia de internet,” Facultad de Informática de Barcelona. Accessed: Feb. 02, 2023. [Online]. Available: https://www.fib.upc.edu/retro-informatica/historia/internet.html#:~:text=En%201972%20ARPANET%20se%20present%C3%B3,y%20se%20crearon%20otras%20redes.

[9] Standford Research Institute, *ARPANET DIRECTORY*. NETWORK INFORMATION CENTER, 1974. Accessed: Feb. 02, 2023. [Online]. Available: https://archive.computerhistory.org/resources/access/text/2021/11/102805038-05-01-acc.pdf

[10] V. Rajaraman, “A Concise History of the Internet—I,” *Resonance*, vol. 27, no. 11, 2022, doi: 10.1007/s12045-022-1483-2.

[11] “Mininet: An Instant Virtual Network on Your Laptop (or Other PC) - Mininet.” Accessed: Sep. 02, 2023. [Online]. Available: http://mininet.org/

[12] L. Miguel *et al.*, “SDN Redes definidas por Software usando MiniNet,” *Revista Científica y Tecnológica UPSE*, vol. 9, no. 1, pp. 48–56, Jun. 2022, doi: 10.26423/RCTU.V9I1.489.

[13] “Open vSwitch.” Accessed: Sep. 02, 2023. [Online]. Available: https://www.openvswitch.org/

[14] “Ubuntu Manpage: xterm - terminal emulator for X.” Accessed: Sep. 02, 2023. [Online]. Available: https://manpages.ubuntu.com/manpages/xenial/man1/xterm.1.html

[15] D. Haro Mendoza, L. Tello Oquendo, and L. Marrone, “A comparative evaluation of the performance of open-source SDN controllers,” *LAJC*, vol. 7, no. 2, pp. 64–77, Dec. 2020.

[16] “OpenDaylight - Documentation for devs.” Accessed: Sep. 02, 2023. [Online]. Available: https://docs.opendaylight.org/projects/controller/en/latest/dev-guide.html

[17] “GitHub - CiscoDevNet/OpenDaylight-Openflow-App.” Accessed: Sep. 03, 2023. [Online]. Available: https://github.com/CiscoDevNet/OpenDaylight-Openflow-App
[18] Infoconsolas, “Generaciones de consolas de videojuegos | 100% RETRO | INFOCONSOLAS.” Accessed: Oct. 25, 2023. [Online]. Available: https://infoconsolas.com/generaciones-de-consolas-de-videojuegos/
[19] Roy Trubshaw and Richard Bartle, “A brief history of MUD2 .” Accessed: Feb. 14, 2023. [Online]. Available: https://www.mud2.com/CMS/index.php/about/history
[20] “Quake II on Steam.” Accessed: Apr. 04, 2023. [Online]. Available: https://store.steampowered.com/app/2320/Quake_II/
[21] “Half-Life on Steam.” Accessed: Apr. 04, 2023. [Online]. Available: https://store.steampowered.com/app/70/HalfLife/
[22] “Ultima Online: New Legacy – Ultima Online.” Accessed: Mar. 31, 2023. [Online]. Available: https://uo.com/ultima-online-new-legacy/
[23] “Microsoft XBOX – Retro Maquinitas.” Accessed: Apr. 05, 2023. [Online]. Available: https://retromaquinitas.com/consolas/consolas-microsoft/xbox/
[24] W. C. Kriz, “Gaming in the Time of COVID-19,” *Simul Gaming*, vol. 51, no. 4, pp. 403–410, Aug. 2020, doi: 10.1177/1046878120931602.
[25] “The International - Liquipedia Dota 2 Wiki.” Accessed: Apr. 05, 2023. [Online]. Available: https://liquipedia.net/dota2/The_International
[26] “EMS Katowice 2014 Challengers - CSGOSKINS.GG.” Accessed: Oct. 21, 2023. [Online]. Available: https://csgoskins.gg/items/ems-katowice-2014-challengers
[27] “Salario mínimo en Venezuela: 0,92 dólares – DW – 01/10/2020.” Accessed: Oct. 21, 2023. [Online]. Available: https://www.dw.com/es/salario-m%C3%ADnimo-en-venezuela-092-d%C3%B3lares/a-55127544
[28] S. F. Saar, “De jogo a trabalho: uma etnografia de trabalhadores no jogo Runescape, no contexto da crise da Venezuela,” UNIVERSIDADE FEDERAL DE SANTA CATARINA, Santa Catarina, 2021.
[29] M. Watson and C. McNulty, “Technology in esports,” *The Science of Esports*, pp. 119–128, Sep. 2023, doi: 10.4324/9781003322382-10.
[30] Definición .de, “Definición de Cliente Servidor,” Definición .de.
[31] “Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization - Valve Developer Community.” Accessed: Oct. 21, 2023. [Online]. Available: https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization
[32] “Latencia en Fibra Óptica – FOM.” Accessed: Oct. 21, 2023. [Online]. Available: https://fibrasopticasdemexico.com/latencia-en-fibra-optica/
[33] S. Sengupta, H. Kim, and J. Rexford, “Continuous in-network round-Trip time monitoring,” in *SIGCOMM 2022 - Proceedings of the ACM SIGCOMM 2022 Conference*, 2022. doi: 10.1145/3544216.3544222.

[34] Ms. M. N. Sankpal, Ms. P. V. Chinchwade, Ms. R. R. Kognole, and Ms. K. N. Rode, “Computer Network Topologies,” *Int J Res Appl Sci Eng Technol*, vol. 10, no. 4, 2022, doi: 10.22214/ijraset.2022.41553.

ANEXOS

Script usado para generar la topología SDN:

Topologia_SDN.py

```
#!/usr/bin/env python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                      controller=RemoteController,
                      ip='127.0.0.1',
                      protocol='tcp',
                      port=6633)

    info( '*** Add switches\n')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
    s4 = net.addSwitch('s4', cls=OVSKernelSwitch)
    s5 = net.addSwitch('s5', cls=OVSKernelSwitch)
    s6 = net.addSwitch('s6', cls=OVSKernelSwitch)
    s7 = net.addSwitch('s7', cls=OVSKernelSwitch)
    s8 = net.addSwitch('s8', cls=OVSKernelSwitch)
    s9 = net.addSwitch('s9', cls=OVSKernelSwitch)
    s10 = net.addSwitch('s10', cls=OVSKernelSwitch)
    s11 = net.addSwitch('s11', cls=OVSKernelSwitch)
    s12 = net.addSwitch('s12', cls=OVSKernelSwitch)
    s13 = net.addSwitch('s13', cls=OVSKernelSwitch)
    s14 = net.addSwitch('s14', cls=OVSKernelSwitch)
    s15 = net.addSwitch('s15', cls=OVSKernelSwitch)
```

```
    s16 = net.addSwitch('s16', cls=OVSKernelSwitch, failMode='standalone')
    s17 = net.addSwitch('s17', cls=OVSKernelSwitch, failMode='standalone')
    s18 = net.addSwitch('s18', cls=OVSKernelSwitch, failMode='standalone')
    s19 = net.addSwitch('s19', cls=OVSKernelSwitch, failMode='standalone')
    s20 = net.addSwitch('s20', cls=OVSKernelSwitch, failMode='standalone')
    s21 = net.addSwitch('s21', cls=OVSKernelSwitch, failMode='standalone')
    s22 = net.addSwitch('s22', cls=OVSKernelSwitch, failMode='standalone')

    info( '*** Add hosts\n')
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
    h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)
    h4 = net.addHost('h4', cls=Host, ip='10.0.0.4', defaultRoute=None)
    h5 = net.addHost('h5', cls=Host, ip='10.0.0.5', defaultRoute=None)
    h6 = net.addHost('h6', cls=Host, ip='10.0.0.6', defaultRoute=None)
    h7 = net.addHost('h7', cls=Host, ip='10.0.0.7', defaultRoute=None)
    h8 = net.addHost('h8', cls=Host, ip='10.0.0.8', defaultRoute=None)
    h9 = net.addHost('h9', cls=Host, ip='10.0.0.9', defaultRoute=None)
    h10 = net.addHost('h10', cls=Host, ip='10.0.0.10', defaultRoute=None)
    h11 = net.addHost('h11', cls=Host, ip='10.0.0.11', defaultRoute=None)

    info( '*** Add links\n')
    s3s2 = {'delay':'11.923ms'}
    net.addLink(s3, s2, cls=TCLink , **s3s2)
    s4s2 = {'delay':'16.344ms'}
    net.addLink(s4, s2, cls=TCLink , **s4s2)
    s5s2 = {'delay':'12.08ms'}
    net.addLink(s5, s2, cls=TCLink , **s5s2)
    s2s8 = {'delay':'30.622ms'}
    net.addLink(s2, s8, cls=TCLink , **s2s8)
    s2s10 = {'delay':'21.776ms'}
    net.addLink(s2, s10, cls=TCLink , **s2s10)
    s2s12 = {'delay':'39.942ms'}
    net.addLink(s2, s12, cls=TCLink , **s2s12)
    net.addLink(s2, s1)
    net.addLink(s1, s22)
    net.addLink(s3, s16)
    net.addLink(s4, s17)
    net.addLink(s5, s6)
    net.addLink(s6, s7)
    net.addLink(s7, s18)
    net.addLink(s8, s9)
    net.addLink(s9, s19)
    net.addLink(s10, s11)
    net.addLink(s11, s20)
    net.addLink(s15, s14)
    net.addLink(s14, s13)
    net.addLink(s13, s12)
    net.addLink(s15, s21)
```

```
    net.addLink(s22, h11)
    net.addLink(s16, h1)
    net.addLink(s16, h2)
    net.addLink(s17, h3)
    net.addLink(s18, h4)
    net.addLink(s18, h5)
    net.addLink(s19, h6)
    net.addLink(s19, h7)
    net.addLink(s20, h8)
    net.addLink(s20, h9)
    net.addLink(s21, h10)

    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s1').start([c0])
    net.get('s2').start([c0])
    net.get('s3').start([c0])
    net.get('s4').start([c0])
    net.get('s5').start([c0])
    net.get('s6').start([c0])
    net.get('s7').start([c0])
    net.get('s8').start([c0])
    net.get('s9').start([c0])
    net.get('s10').start([c0])
    net.get('s11').start([c0])
    net.get('s12').start([c0])
    net.get('s13').start([c0])
    net.get('s14').start([c0])
    net.get('s15').start([c0])
    net.get('s16').start([])
    net.get('s17').start([])
    net.get('s18').start([])
    net.get('s19').start([])
    net.get('s20').start([])
    net.get('s21').start([])
    net.get('s22').start([])

    info( '*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()

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

```
    myNetwork()
```

Script usado para similar un cliente/videojugador de CS:GO para partidas en línea:

cliente_videojugadores.py

```
import socket
import threading
import random
from ping3 import ping
import numpy as np
import time
import os

#Obtener hostname e IP de la máquina
#hostname=socket.gethostname()
#IPAddr=socket.gethostbyname(hostname)
stream = os.popen('hostname -I')
output = stream.read()
x = output.replace(' \n', '')
client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client.bind((x, random.randint(7000, 8000)))
#client.bind((str(IPAddr), 9999))
name = input("Type ur nickname: ")
#name = "host1"
tickrate = 1/64
ti = 0
client.sendto(f"/.JUGADOR CONECTADO: {name} ".encode(),
("10.0.0.11",7778))

def recibir():

    x = 0
    while True:

        try:

            message, _ = client.recvfrom(1024)
            repeated = message.decode()

            if name in repeated:
                if "conectado" in repeated:
                    print (repeated)
                else:
```

```
                    print (str(repeated))
                    if x == 0:
                        h3.start()
                        x = 1
                    else:
                        pass









            else:
                print (str(repeated))
                #pass
            if "INICIO DE LA" in repeated:
                print("INICIA")
                h2.start()

            else:
                pass
        except:
            pass
def enviar():
    xx = 1

    #global ti
    while True:

        message = ("Paquete Número: " + str(xx)  )
        client.sendto(f"Jugador: {name} -- {message}".encode(),
("10.0.0.11",7778))



        xx += 1

        #Tickrate = 64/s = 1 paquete cada 0.015625ms
        time.sleep(0.015625)


        #os.system('clear')

def rtt_test():

    n_test = 200
    x = 1
    rttarray = []
```

```
    pruebatxt = open(name+'RTT.txt', 'w')

    while x <= n_test:

        rtt = ping('10.0.0.11')
        rttarray.append(rtt)




        x+=1
        pruebatxt.write(str(rtt)+"\n")
        time.sleep(0.1)



    # Multiplicado x1000 para pasar de s a ms
    rttpromedio = np.mean(rttarray)*1000
    jitter = np.std(rttarray)*1000




pruebatxt.write("************************************************************
************\n ")
    pruebatxt.write (" --- RTT MÍNIMO: " + str(round(min(rttarray)*1000, 3
))+"ms\n")
    pruebatxt.write (" --- RTT PROMEDIO: " + str(rttpromedio)+"ms\n")
    pruebatxt.write (" --- RTT MÁXIMO: " + str(round(max(rttarray)*1000, 3
))+"ms\n")

    pruebatxt.write ("\n --- JITTER: " +  str(jitter)+"ms\n")

    pruebatxt.write
("\n************************************************************************
")

    pruebatxt.close()


h1 = threading.Thread(target = recibir)
h2 = threading.Thread(target = enviar)
h3 = threading.Thread(target = rtt_test)
h1.start()
```

Script usado para simular un servidor de CS:GO para partidas en línea:

servidor_partidas_en_línea.py

```
import socket
import threading
import queue
import time

messages = queue.Queue()
clients = []

#Definir el puerto y dirección IP, a la par del protocolo UDP
server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server.bind(("10.0.0.11", 7778))
print("*****************")
print("SERVIDOR INICIADO")
print("*****************")
def recibir():
    while True:
        try:
            #recvfrom : The number of bytes to be read from the UDP socket
            message, addr = server.recvfrom(1024)
            messages.put((message, addr))
        except:
            pass
# En message se almacena temporalmente el mensaje, y luego se
# pasa al array de messages / En addr se almacena
# la IP y el puerto del HOST
def conexion():
    while True:
        while not messages.empty():
            message, addr = messages.get()
            print(message.decode())
            # .append agrega un elemento al final del array
            # En este caso agrega addr a clients
            if addr not in clients:
                clients.append(addr)
                flag1 = len(clients)
            #client posición por posición de clients
            #print(len(clients))

            for client in clients:
                try:
                    if message.decode().startswith("/.JUGADOR "):
                        name =
message.decode()[message.decode().index(":")+1:]
                        server.sendto(f"{name} conectado a la
partida".encode(), client)
```

```
                    else:
                        server.sendto(message, client)


                except:
                    clients.remove(client)
                #time.sleep(0.5)
            if  flag1 == 10:
                for client in clients:
                    server.sendto(f"INICIO DE LA PARTIDA".encode(),
client)
                flag1 = 99
                time.sleep(2)


            else:
                pass
            #BROADCAST



h1 = threading.Thread(target = recibir)
h2 = threading.Thread(target = conexion)
h1.start()
h2.start()
```