

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIA APLICADAS

CARRERA DE TELECOMUNICACIONES

TRABAJO DE INTEGRACIÓN CURRICULAR

TEMA:

"IMPACTO DE ATAQUES DDOS EN LA CALIDAD DEL SERVICIO (QOS) EN LA TRANSMISIÓN DE STREAMING MULTIMEDIA EN ENTORNOS SDN"

Trabajo de titulación previo a la obtención del título de Ingeniero en Telecomunicaciones

Línea de investigación: Desarrollo, aplicación de software y cyber security (seguridad cibernética)

AUTOR:

DIEGO ISRAEL ACHINA QUILUMBAQUIN

DIRECTOR:

MSC. FABIÁN GEOVANNY CUZME RODRÍGUEZ

Ibarra, Ecuador, 2026

UNIVERSIDAD TÉCNICA DEL NORTE
DIRECCIÓN DE BIBLIOTECA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO	
APELLIDOS Y NOMBRES:	ACHINA QUILUMBAQUIN DIEGO ISRAEL

DATOS DE LA OBRA	
TÍTULO:	"IMPACTO DE ATAQUES DDOS EN LA CALIDAD DEL SERVICIO (QOS) EN LA TRANSMISIÓN DE STREAMING MULTIMEDIA EN ENTORNOS SDN"
AUTOR (ES):	ACHINA QUILUMBAQUIN DIEGO ISRAEL
FECHA: DD/MM/AAAA	16/10/2026
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	■ **PREGRADO** □ **POSGRADO**
TITULO POR EL QUE OPTA:	INGENIERO EN TELECOMUNICACIONES
DIRECTOR /ASESOR:	MSC. FABIÁN GEOVANNY CUZME RODRÍGUEZ MSC. MAURICIO HERNÁN DOMINGUEZ LIMAICO

2. CONSTANCIAS

El autor (es) manifiesta (n) que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es (son) el (los) titular (es) de los derechos patrimoniales, por lo que asume (n) la responsabilidad sobre el contenido de la misma y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 12 días del mes de octubre de 2026

EL AUTOR:

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
DIEGO ISRAEL ACHINA QUILUMBAQUIN

Diego Israel Achina Quilumbaquin

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

Ibarra, 16 de octubre de 2026

MSC. CUZME RODRÍGUEZ FABIÁN GEOVANNY

DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

CERTIFICA:

Haber revisado el presente informe final del trabajo de Integración Curricular, el mismo que se ajusta a las normas vigentes de la Universidad Técnica del Norte; en consecuencia, autorizo su presentación para los fines legales pertinentes.

(f) ..Validar únicamente en FirmaEC.
Firmado electrónicamente por:
FABIAN GEOVANNY
CUZME RODRIGUEZ
.....

MSC. CUZME RODRÍGUEZ FABIÁN GEOVANNY

DIRECTOR



UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

DEDICATORIA

Este trabajo lo dedico a mi madre y mi padre por sus sacrificios y su apoyo constante, que fueron el motor que me impulsó a culminar esta etapa de mi formación profesional.

A mi novia, por su paciencia, su compañía y sus palabras de aliento en los momentos más difíciles de este proceso.

Y a mí mismo, por el esfuerzo, la disciplina y la perseverancia dedicados a lo largo de este trabajo de integración curricular.

Achina Quilumbaquin Diego



UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

AGRADECIMIENTO

Agradezco a Dios por darme la vida, la salud y la fortaleza necesarias para culminar con éxito esta etapa académica.

A mis padres y a mi novia, por su apoyo incondicional y su motivación permanente durante todo este proceso.

A mi tutor de trabajo de integración curricular, por su guía, paciencia y conocimientos compartidos, fundamentales para el desarrollo de esta investigación.

A mis docentes y compañeros, por su apoyo y por formar parte de este camino de aprendizaje.

A la Universidad Técnica del Norte y, de manera especial, a la Carrera de Telecomunicaciones, por brindarme los conocimientos y las herramientas necesarias para mi formación profesional.

Achina Quilumbaquin Diego

RESUMEN EJECUTIVO

Las redes definidas por software (SDN) centralizan el control del tráfico y facilitan la gestión de servicios multimedia, pero esa misma arquitectura expone al servidor de streaming a ataques de denegación de servicio distribuido (DDoS) que degradan la calidad de servicio (QoS) percibida por el usuario. El presente trabajo tiene como objetivo evaluar el impacto de los ataques DDoS sobre la calidad de servicio de la transmisión de streaming multimedia en entornos SDN. Para ello se especificaron los requerimientos según el estándar ISO/IEC/IEEE 29148 y se implementó un laboratorio virtualizado en GNS3 que integra el servidor MediaMTX, clientes multi-viewer, el monitor ntopng y un generador de ataques propio (TrafficGen SDN); sobre esta topología se ejecutó el ataque tcp_exhaust contra el servicio HLS del servidor (puerto 8888) en una línea base y cuatro escenarios de intensidad creciente (de uno a cuatro workers), con tres réplicas cada uno, comparando los controladores OpenDaylight y Ryu mediante métricas de throughput, RTT, jitter y pérdida de paquetes. Los resultados muestran que el ataque degrada la calidad de servicio de forma progresiva a medida que aumenta su intensidad, afectando tanto al throughput como a la latencia, el jitter y la pérdida de paquetes del servicio de streaming; en la intensidad máxima, la pérdida alcanzó 10.75 % con OpenDaylight y 4.30 % con Ryu, el RTT medio, 92.69 ms y 64.36 ms, respectivamente, y el throughput efectivo entregado al cliente se redujo un 64.8 % y un 30.2 % respecto a la línea base de cada controlador. En las condiciones experimentales evaluadas, el controlador Ryu presentó una menor degradación de las métricas de QoS que OpenDaylight frente al ataque tcp_exhaust, conservando una mayor capacidad de servicio bajo carga.

Palabras clave: SDN, DDoS, streaming multimedia, calidad de servicio, OpenDaylight, Ryu.

ABSTRACT

Software-defined networks (SDN) centralize traffic control and simplify the management of multimedia services, yet that same architecture exposes the streaming server to distributed denial-of-service (DDoS) attacks that degrade the quality of service (QoS) perceived by the user. This work aims to evaluate the impact of DDoS attacks on the quality of service of multimedia streaming transmission in SDN environments. Requirements were specified according to the ISO/IEC/IEEE 29148 standard, and a virtualized laboratory was built in GNS3 integrating the MediaMTX server, multi-viewer clients, the ntopng monitor and a custom attack generator (TrafficGen SDN); on this topology, the tcp_exhaust attack was executed against the server's HLS service (port 8888) across a baseline and four scenarios of increasing intensity (from one to four workers), with three replicas each, comparing the OpenDaylight and Ryu controllers through throughput, RTT, jitter and packet loss metrics. The results show that the attack degrades the quality of service progressively as its intensity increases, affecting the throughput as well as the latency, jitter and packet loss of the streaming service; at the highest intensity, packet loss reached 10.75% with OpenDaylight and 4.30% with Ryu, the mean RTT 92.69 ms and 64.36 ms, respectively, and the effective throughput delivered to the client dropped by 64.8% and 30.2% relative to each controller's baseline. Under the experimental conditions evaluated, the Ryu controller exhibited less degradation of the QoS metrics than OpenDaylight against the tcp_exhaust attack, retaining a greater service capacity underload. It is concluded that the choice of SDN controller affects the resilience of the service against this type of attack and that, since no single controller prevents degradation at the highest intensities, an active mitigation mechanism is required.

Keywords: SDN, DDoS, multimedia streaming, quality of service, OpenDaylight, Ryu.

LISTA DE SIGLAS

ACK. Acknowledgment (acuse de recibo; bandera TCP)

API. Application Programming Interface (Interfaz de Programación de Aplicaciones)

C2. Command and Control (mando y control)

CPU. Central Processing Unit (Unidad Central de Procesamiento)

DDoS. Distributed Denial of Service (Denegación de Servicio Distribuida)

DLUX. OpenDaylight User Experience (interfaz web de OpenDaylight)

DMZ. Demilitarized Zone (zona desmilitarizada)

DoS. Denial of Service (Denegación de Servicio)

DPID. Datapath Identifier (identificador del conmutador OpenFlow)

FFmpeg. Fast Forward MPEG (herramienta de procesamiento multimedia)

GNS3. Graphical Network Simulator 3

HLS. HTTP Live Streaming

HTTP. Hypertext Transfer Protocol (Protocolo de Transferencia de Hipertexto)

ICMP. Internet Control Message Protocol

IEC. International Electrotechnical Commission (Comisión Electrotécnica Internacional)

IEEE. Institute of Electrical and Electronics Engineers

IETF. Internet Engineering Task Force

IP. Internet Protocol (Protocolo de Internet)

IPv4. Internet Protocol version 4

IPDV. IP Packet Delay Variation (variación del retardo de paquetes IP)

IPLR. IP Packet Loss Ratio (tasa de pérdida de paquetes IP)

IPTD. IP Packet Transfer Delay (retardo de transferencia de paquetes IP)

ISO. International Organization for Standardization

ITU/UIT. International Telecommunication Union / Unión Internacional de Telecomunicaciones

JVM. Java Virtual Machine

JSON. JavaScript Object Notation

KPI. Key Performance Indicator (indicador clave de rendimiento)

LAN. Local Area Network (Red de Área Local)

LSA. Link-State Advertisement

NIB. Network Information Base

Ntopng. Network Top Next Generation (monitoreo de tráfico de red)

ODL. OpenDaylight (controlador SDN)

OSPF. Open Shortest Path First (protocolo de enrutamiento)

OTT. Over-the-Top (servicios de contenido sobre Internet)

OVS. Open vSwitch (conmutador virtual)

OVSDB. Open vSwitch Database Management Protocol

PCAP. Packet Capture (captura de paquetes)

QoS. Quality of Service (Calidad de Servicio)

RFC. Request for Comments

RST. Reset (bandera TCP de reinicio)

RTP. Real-time Transport Protocol

RTSP. Real-Time Streaming Protocol

RTT. Round-Trip Time (tiempo de ida y vuelta)

SDN. Software Defined Networking (Redes Definidas por Software)

SYN. Synchronize (bandera TCP de sincronización)

SPAN. Switched Port Analyzer (puerto espejo)

SDP. Session Description Protocol

SyRS. System Requirements Specification (Especificación de Requisitos del Sistema)

TCP. Transmission Control Protocol

UDP. User Datagram Protocol

VLSM. Variable Length Subnet Mask (máscara de subred de longitud variable)

VM. Virtual Machine (máquina virtual)

YANG. Yet Another Next Generation (lenguaje de modelado de datos)

ÍNDICE DE CONTENIDOS

Introducción **28**

CAPÍTULO I: ANTECEDENTES **30**

1.1. Problema de investigación 30

1.2. Justificación 32

1.3. Objetivos 34

1.3.1. Objetivo general 34

1.3.2. Objetivos específicos 34

1.4. Alcance 34

2. CAPÍTULO II: MARCO TEÓRICO **38**

2.1. Redes definidas por software (SDN) 38

2.1.1. Arquitectura de SDN 38

El plano de control u 39

2.1.2. Controladores SDN 40

2.1.2.1. OpenDaylight 40

2.1.2.2. Ryu 43

2.1.2.3. NOX 45

2.1.3. Protocolos de comunicación SDN 47

2.1.3.1. Protocolo OpenFlow 48

2.1.3.2. Mensajes OpenFlow 48

2.1.3.3. Protocolo OVSDB 50

2.2. Ataques DDoS 50

2.2.1. Fases de un ataque DDoS 51

2.2.2. Tipos de ataques DDoS 51

2.2.2.1. Ataques volumétricos 51

2.2.2.2. Ataques de protocolo 52

2.2.2.3. Ataques de capa de aplicación 53

2.2.3. Ataques DDoS a redes definidas por software 55

2.3. Servicios de streaming multimedia 55

2.3.1. Componentes 56

2.3.2. Arquitectura 57

2.3.2.1. Arquitectura centralizada 57

2.3.2.2. Arquitectura de servidores independientes 58

2.3.2.3. Streaming en redes definidas por software 59

2.4. Marco normativo para la evaluación de la calidad y el rendimiento de la red . 60

2.4.1. Criterios de duración de las corridas según las RFC 2544 y RFC 6349 .60

2.4.2. Marco estadístico de rendimiento según la Recomendación ITU-T E.84061

2.4.3. Umbrales de calidad de servicio (QoS) y calidad de experiencia (QoE) para video en streaming 62

2.4.3.1. Parámetros de rendimiento de red según la Recomendación ITU-T Y.1541 62

2.4.3.2. Requisitos de calidad de experiencia (QoE) según la Recomendación ITU-T G.1010 63

2.4.1. Herramientas de medición de QoS 64

2.4.1. Métricas de QoS para evaluar el impacto del ataque DDoS 66

3. CAPÍTULO III: METODOLOGÍA 69

3.1. Análisis de los requerimientos del modelo 69

3.1.1. Partes interesadas (stakeholders) 70

3.1.2. Requerimientos del sistema 71

3.1.3. Requerimientos de la arquitectura 72

3.1.4. Evaluación de herramientas de simulación 75

3.1.5. Evaluación y selección del controlador 77

3.1.6. Justificación académica del uso de Docker 79

3.1.7. Evaluación y selección de la herramienta de generación de ataques DDoS 80

3.1.8. Evaluación y selección de la herramienta de monitoreo de tráfico 82

3.1.9. Evaluación y selección del servidor de streaming 83

3.2. Diseño del sistema 85

3.2.1. Topología de red 85

3.2.1.1. Bloque de usuarios 88

3.2.1.2. Bloque SDN 88

3.2.1.3. Bloque servidor de streaming 88

3.2.1.4. Bloque de ataques DDoS 89

3.2.1.5. Bloque de monitoreo 89

3.2.2. Topología lógica 89

3.2.2.1. Dispositivos que conforman la topología 91

3.2.2.2. Direccionamiento IPv4 de la topología 93

3.3. Diagrama de secuencia del ataque DDoS 94

3.3.1. Diseño del sistema TrafficGen SDN 97

3.3.1.1. Arquitectura de despliegue distribuido 101

3.4. Implementación de la red SDN 102

3.4.1. Integración entre el controlador y los conmutadores 103

3.4.1.1. Configuración de los Open vSwitch 103

3.4.2. Configuración del protocolo OSPF 105

3.4.3. Verificación de conectividad y enrutamiento 106

3.5. Implementación del entorno de pruebas 107

3.5.1. Virtualización con Docker en GNS3 109

3.5.2. Implementación del sistema TrafficGen SDN 111

3.5.2.1. Construcción de la imagen Docker 111

3.5.2.2. Sistema de monitoreo y captura de métricas en tiempo real 113

3.5.2.3. Comandos de ejecución y acceso al sistema distribuido 115

3.5.3. Implementación del servidor de streaming MediaMTX.....120

3.5.4. Verificación de la implementación de la red SDN 122

3.5.4.1. Asignación de IP de gestión y conectividad con el controlador 123

3.5.4.2. Creación del bridge OVS y asignación de interfaces..... 123

3.5.4.3. Verificación de la conexión entre el controlador y los Open vSwitch. 124

3.5.4.4. Instalación de flujos OpenFlow para tráfico OSPF 125

3.5.5. Verificación del despliegue del servicio de streaming.....126

3.5.5.1. Despliegue del contenedor del servidor de streaming 126

3.5.5.2. Ejecución del cliente multi-viewer y verificación del flujo a través de la SDN 127

3.5.6. Implementación del sistema de monitoreo con ntopng 128

3.5.6.1. Descarga e instalación de ntopng..... 128

3.5.6.2. Solicitud y activación de la licencia educativa 129

3.5.6.3. Configuración del port mirroring en los Open vSwitch..... 131

3.5.6.4. Verificación del flujo recibido en ntopng..... 132

3.6. Procedimiento de captura de datos 133

3.6.1. Capturas de tráfico de red (PCAP) 133

3.6.2. Almacenamiento, nomenclatura y trazabilidad de las capturas.....133

3.7. Estrategia de escalamiento del experimento 134

3.8. Normativa de los parámetros experimentales 136

3.9. Parámetros escalados frente a parámetros de referencia..... 138

3.10. Parámetros del servicio de streaming multimedia 140

3.11. Plan de experimentación 142

3.11.1. Variables del experimento.....143

3.11.2. Escenarios experimentales.....144

3.11.3. Procedimiento de ejecución por corrida 146

3.12. Procesamiento y análisis de datos..... 148

3.12.1. Extracción de métricas de calidad de servicio a partir de las capturas de tráfico 148

3.12.1.1. Extracción del throughput efectivo..... 149

3.12.1.2. Extracción de la latencia (RTT TCP)..... 150

3.12.1.3. Extracción del jitter (variación de retardo entre paquetes) 151

3.12.1.4. Extracción de la pérdida de paquetes..... 152

3.12.2. Generación de las gráficas.....153

3.12.3. Consideraciones sobre las métricas y los puntos de medición 154

4. CAPÍTULO IV: RESULTADOS Y ANÁLISIS 156

4.1. Síntesis del diseño experimental..... 156

4.2. Configuración final del experimento 157

4.2.1. Corrida piloto y verificación previa al experimento.....157

4.2.2. Orden de ejecución de las pruebas 157

4.3. Línea base sin ataque 158

4.3.1. Métricas de QoS observadas en condiciones normales 158

4.3.2. Escenario E1-ODL (controlador OpenDaylight) 158

4.3.3. Escenario E1-Ryu (controlador Ryu) 161

4.4. Resultados bajo ataque 167

4.4.1. Escenario E3: 2 workers × 20 hilos 167

4.4.1.1. Escenario E3-ODL (2 workers) 167

4.4.1.2. Escenario E3-Ryu (2 workers) 170

4.4.2. Comparación entre OpenDaylight y Ryu (2 workers) 173

4.5. Comparación entre OpenDaylight y Ryu 176

4.6. Análisis estadístico 182

4.7. Discusión de los resultados 186

4.8. Proyección hacia escenarios de mayor escala y limitaciones 189

1.1.1 Limitaciones metodológicas de la proyección 189

4.8.1. Metodología de proyección 190

4.8.2. Parámetros de escalamiento aplicados 190

4.8.3. Proyección del throughput 190

4.8.4. Proyección de la latencia (RTT) 191

4.8.5. Proyección de la pérdida de paquetes 191

4.8.6. Síntesis de la proyección 192

4.9. Trabajo futuro: propuesta preliminar de mitigación 192

4.9.1. Fundamento de la propuesta 192

4.9.2. Reglas de filtrado propuestas 193

4.9.3. Líneas de trabajo complementarias 194

Conclusiones y recomendaciones 195

Conclusiones 195

Recomendaciones 197

Referencias Bibliográficas 198

ANEXO A. DESCRIPCIÓN DE LOS REQUERIMIENTOS DEL MODELO DE SIMULACIÓN 206

ANEXO B. MANUAL DE DESPLIEGUE DE HERRAMIENTAS 219

Instalación del controlador SDN 219

Instalación de los controladores SDN en la máquina virtual 219

Controlador OpenDaylight 219

Controlador Ryu 223

ANEXO C. RESULTADOS DETALLADOS DEL ESCENARIO E2 (1 WORKER × 20 HILOS) 228

ANEXO D. RESULTADOS DETALLADOS DEL ESCENARIO E4 (3 WORKERS × 20 HILOS) 234

ANEXO E. RESULTADOS DETALLADOS DEL ESCENARIO E5 (4 WORKERS × 20 HILOS).....244

ANEXO F. SCRIPTS DE EXTRACCIÓN DE MÉTRICAS DE QOS 253

ÍNDICE DE TABLAS

Tabla 1 Controladores SDN 40

Tabla 2 Comparación entre los controladores habituales en redes SDN 45

Tabla 3 Herramientas seleccionadas para la medición de métricas de QoS 65

Tabla 4 Métricas de QoS, umbrales de referencia e instrumento de medición 68

Tabla 5 Abreviaturas de los requerimientos definidos en el estándar ISO/IEC/IEEE 29148:2011 70

Tabla 6 Partes interesadas (stakeholders) 71

Tabla 7 Requerimientos del sistema 71

Tabla 8 Requerimientos de la arquitectura de simulación 73

Tabla 9 Requerimientos para la elección de la herramienta de simulación 76

Tabla 10 Cumplimiento de requerimientos por controlador SDN 77

Tabla 11 Comparación entre Docker y máquinas virtuales para el entorno de laboratorio 79

Tabla 12 Evaluación de herramientas para la generación de ataques DDoS 81

Tabla 13 Matriz comparativa de herramientas de monitoreo de tráfico SDN frente a los requerimientos del sistema 82

Tabla 14 Evaluación de servidores de streaming frente a los requerimientos SyRA12–SyRA16 84

Tabla 15 Dispositivos que conforman la topología 91

Tabla 16 Direccionamiento IPv4 de la topología del escenario general 94

Tabla 17 Atributos y métodos principales de las clases RTSPClient y AttackSession 98

Tabla 18 Ataque implementado en AttackSession (tcp_exhaust) y su mecanismo de impacto 99

Tabla 19 Variables de entorno configurables del sistema TrafficGen SDN 99

Tabla 20 Configuración de despliegue de los nodos del sistema distribuido TrafficGen SDN 102

Tabla 21 Comandos de configuración de cada Open vSwitch..... 104

Tabla 22 Componentes del entorno de pruebas y su configuración en el entorno de emulación GNS3..... 108

Tabla 23 Capas del Dockerfile del sistema TrafficGen SDN y su propósito 112

Tabla 24 Métricas capturadas por el sistema de monitoreo del dashboard distribuido.....114

Tabla 25 Comandos de construcción, despliegue y verificación del sistema TrafficGen SDN...115

Tabla 26 Comandos de despliegue del servidor MediaMTX y publicación del flujo con FFmpeg 121

Tabla 27 Justificación normativa de los parámetros experimentales..... 137

Tabla 28 Comparativa de parámetros escalados (entorno experimental) frente a parámetros de referencia de mayor escala..... 138

Tabla 29 Parámetros del servicio de streaming en el entorno experimental frente a un escenario de referencia de mayor escala 141

Tabla 30 Escenarios experimentales definidos para la evaluación de la QoS 144

Tabla 31 Matriz comparativa de configuraciones experimentales..... 145

Tabla 32 Métricas de QoS medias por escenario y controlador..... 176

Tabla 33 Tamaño de muestra analizado por escenario y controlador (suma de las tres corridas) 183

Tabla 34 Prueba t de Welch entre OpenDaylight y Ryu por escenario y métrica 186

ÍNDICE DE FIGURAS

Figura 1 *Topología de un ataque DDoS dirigido al servidor de streaming en una red SDN* 36

Figura 2 *Capas de la arquitectura de red SDN* 39

Figura 3 *Arquitectura del controlador OpenDaylight* 42

Figura 4 *Arquitectura del controlador Ryu* 44

Figura 5 *Esquema de un ataque volumétrico* 52

Figura 6 *Ataque DDoS de protocolo sobre TCP* 53

Figura 7 *Ataque DDoS de capa de aplicación* 54

Figura 8 *Funcionamiento del streaming* 56

Figura 9 *Arquitectura de VoD centralizada* 58

Figura 10 *Arquitectura de VoD de servidores independientes* 59

Figura 11 *Topología de red SDN* 87

Figura 12 *Topología lógica de la red SDN* 90

Figura 13 *Diagrama de secuencia del ataque DDoS sobre la red SDN con streaming* 96

Figura 14 *Interfaz web del sistema TrafficGen SDN: topología lógica, métricas en tiempo real y gráficas de monitoreo con ejes Y fijos* 101

Figura 15 *Configuración de Open vSwitch en cada nodo* 105

Figura 16 *Verificación de la conexión entre el controlador y Open vSwitch* 105

Figura 17 *Convergencia entre el nodo Kali (C2) y el servidor de video* 107

Figura 18 *Descarga de los paquetes de Docker* 109

Figura 19 *Instalación de complementos previos a Docker* 110

Figura 20 *Verificación de la instalación de Docker* 110

Figura 21 *Incorporación del usuario al grupo Docker* 111

Figura 22 *Ajuste de memoria para la construcción de la imagen Docker* 113

Figura 23 *Estructura del repositorio con la carpeta wheels para la construcción de la imagen* 113

Figura 24 *Clonación del repositorio en el worker 3*.....118

Figura 25 *Extracción del paquete del worker mediante unzip* 118

Figura 26 *Ejecución del worker y conexión al nodo C2*.....119

Figura 27 *Workers en línea reportados por el controlador C2 en Kali Linux*..... 120

Figura 28 *Asignación de IP de gestión a eth0 y verificación de conectividad con el controlador SDN*..... 123

Figura 29 *Creación del bridge br0 y asignación de interfaces en los Open vSwitch*..... 124

Figura 30 *Definición del protocolo OpenFlow 1.3, identificador de cada Open vSwitch y conexión al controlador*..... 124

Figura 31 *Verificación de la conexión entre el controlador SDN y los Open vSwitch mediante ping* 125

Figura 32 *Flujos OpenFlow que permiten la circulación de OSPF a través de los Open vSwitch* 126

Figura 33 *Creación y despliegue del contenedor que ejecuta el servidor de streaming* 127

Figura 34 *Ejecución del cliente multi-viewer y verificación del consumo del stream a través de la red SDN*..... 127

Figura 35 *Verificación de la conectividad entre el servidor de streaming y el multi-viewer desde la red del cliente*..... 128

Figura 36 *Descarga e instalación de ntopng en la máquina virtual de monitoreo*..... 129

Figura 37 *Activación de la licencia educativa de ntopng mediante correo institucional*..... 130

Figura 38 *Configuración de la interfaz eth4 de ntopng como puerto de espejo (mirror)* 130

Figura 39 *Configuración del port mirroring en el Open vSwitch hacia la máquina virtual de ntopng* 131

Figura 40 *Configuración de port mirroring hacia ntopng en el Open vSwitch* 132

Figura 41 *Tráfico de la línea base de la topología visualizado en ntopng* 132

Figura 42 *Throughput efectivo del tráfico de línea base con el controlador OpenDaylight (intervalos de 1 s)* 159

Figura 43 *RTT TCP en la línea base con el controlador OpenDaylight* 159

Figura 44 *Jitter RFC 3550 en la línea base con el controlador OpenDaylight* 160

Figura 45 *Pérdida de paquetes en la línea base con el controlador OpenDaylight* 161

Figura 46 *Throughput efectivo del tráfico de línea base con el controlador Ryu (intervalos de 1 s)* 162

Figura 47 *RTT TCP en la línea base con el controlador Ryu* 162

Figura 48 *Jitter RFC 3550 en la línea base con el controlador Ryu* 163

Figura 49 *Pérdida de paquetes en la línea base con el controlador Ryu* 164

Figura 50 *Comparación del throughput efectivo entre OpenDaylight y Ryu (línea base)* 164

Figura 51 *Comparación del RTT entre OpenDaylight y Ryu (línea base)* 165

Figura 52 *Comparación del jitter entre OpenDaylight y Ryu (línea base)* 166

Figura 53 *Comparación de la pérdida de paquetes entre OpenDaylight y Ryu (línea base)* 166

Figura 54 *Throughput efectivo con OpenDaylight (2 workers × 20 hilos)* 168

Figura 55 *RTT TCP durante el ataque con OpenDaylight (2 workers × 20 hilos)* 169

Figura 56 *Jitter RFC 3550 durante el ataque con OpenDaylight (2 workers × 20 hilos)* 169

Figura 57 *Pérdida de paquetes durante el ataque con OpenDaylight (2 workers × 20 hilos)*.. 170

Figura 58 *Throughput efectivo con Ryu (2 workers × 20 hilos)* 171

Figura 59 *RTT TCP durante el ataque con Ryu (2 workers × 20 hilos)* 171

Figura 60 *Jitter RFC 3550 durante el ataque con Ryu (2 workers × 20 hilos)* 172

Figura 61 *Pérdida de paquetes durante el ataque con Ryu (2 workers × 20 hilos)* 173

Figura 62 *Comparación del throughput efectivo entre OpenDaylight y Ryu (2 workers × 20 hilos)* 173

Figura 63 *Comparación del RTT entre OpenDaylight y Ryu (2 workers × 20 hilos)* 174

Figura 64 *Comparación del jitter entre OpenDaylight y Ryu (2 workers × 20 hilos)* 175

Figura 65 *Comparación de la pérdida de paquetes entre OpenDaylight y Ryu (2 workers × 20 hilos)* 175

Figura 66 *Throughput efectivo medio por escenario y controlador* 179

Figura 67 *RTT medio por escenario y controlador* 179

Figura 68 *Jitter medio por escenario y controlador* 180

Figura 69 *Pérdida de paquetes media por escenario y controlador* 181

Figura 70 *Resumen global de métricas de QoS por escenario y controlador* 182

Figura 71 *Prerrequisitos del sistema para la instalación de ODL* 220

Figura 72 *Instalación de Java 8* 220

Figura 73 *Descarga de la distribución de ODL mediante wget* 221

Figura 74 *Descompresión del directorio de Karaf* 221

Figura 75 *Instalación y ejecución del controlador ODL* 222

Figura 76 *Instalación de las características funcionales (features)* 223

Figura 77 *Módulos de la interfaz web DLUX y YANG* 223

Figura 78 *Actualización del sistema operativo y de las aplicaciones* 224

Figura 79 *Instalación del intérprete Python 3.9 y de sus dependencias* 225

Figura 80 *Instalación de los componentes de Python* 226

Figura 81 *Descarga de pip e instalación de los paquetes de Ryu* 226

Figura 82 *Ejecución del controlador Ryu y carga de aplicaciones* 227

Figura 83 *Throughput efectivo durante el ataque tcp_exhaust con OpenDaylight (1 worker × 20 hilos)* 229

Figura 84 *RTT TCP durante el ataque con OpenDaylight (1 worker × 20 hilos)* 230

Figura 85 *Jitter RFC 3550 durante el ataque con OpenDaylight (1 worker × 20 hilos)* 230

Figura 86 *Pérdida de paquetes durante el ataque con OpenDaylight (1 worker × 20 hilos)* ... 231

Figura 87 *Throughput efectivo durante el ataque tcp_exhaust con Ryu (1 worker × 20 hilos)* 232

Figura 88 *RTT TCP durante el ataque con Ryu (1 worker × 20 hilos)* 232

Figura 89 *Jitter RFC 3550 durante el ataque con Ryu (1 worker × 20 hilos)* 233

Figura 90 *Pérdida de paquetes durante el ataque con Ryu (1 worker × 20 hilos)* 234

Figura 91 *Throughput efectivo con OpenDaylight (3 workers × 20 hilos)* 235

Figura 92 *RTT TCP durante el ataque con OpenDaylight (3 workers × 20 hilos)* 236

Figura 93 *Jitter RFC 3550 durante el ataque con OpenDaylight (3 workers × 20 hilos)* 237

Figura 94 *Pérdida de paquetes con OpenDaylight (3 workers × 20 hilos)* 237

Figura 95 *Throughput efectivo con Ryu (3 workers × 20 hilos)* 238

Figura 96 *RTT TCP durante el ataque con Ryu (3 workers × 20 hilos)* 239

Figura 97 *Jitter RFC 3550 durante el ataque con Ryu (3 workers × 20 hilos)* 239

Figura 98 *Pérdida de paquetes con Ryu (3 workers × 20 hilos)* 240

Figura 99 *Comparación del throughput efectivo entre OpenDaylight y Ryu (3 workers × 20 hilos)* 241

Figura 100 *Comparación del RTT entre OpenDaylight y Ryu (3 workers × 20 hilos)*..... 242

Figura 101 *Comparación del jitter entre OpenDaylight y Ryu (3 workers × 20 hilos)*..... 243

Figura 102 *Comparación de la pérdida de paquetes entre OpenDaylight y Ryu (3 workers × 20 hilos)* 243

Figura 103 *Throughput efectivo con OpenDaylight (4 workers × 20 hilos)* 245

Figura 104 *RTT TCP durante el ataque con OpenDaylight (4 workers × 20 hilos)* 245

Figura 105 *Jitter RFC 3550 durante el ataque con OpenDaylight (4 workers × 20 hilos)* 246

Figura 106 *Pérdida de paquetes con OpenDaylight (4 workers × 20 hilos)*..... 247

Figura 107 *Throughput efectivo con Ryu (4 workers × 20 hilos)*..... 248

Figura 108 *RTT TCP durante el ataque con Ryu (4 workers × 20 hilos)*..... 249

Figura 109 *Jitter RFC 3550 durante el ataque con Ryu (4 workers × 20 hilos)*..... 249

Figura 110 *Pérdida de paquetes con Ryu (4 workers × 20 hilos)* 250

Figura 111 *Comparación del throughput efectivo entre OpenDaylight y Ryu (4 workers × 20 hilos)* 251

Figura 112 *Comparación del RTT entre OpenDaylight y Ryu (4 workers × 20 hilos)*..... 251

Figura 113 *Comparación del jitter entre OpenDaylight y Ryu (4 workers × 20 hilos)*..... 252

Figura 114 *Comparación de la pérdida de paquetes entre OpenDaylight y Ryu (4 workers × 20 hilos)* 253

INTRODUCCIÓN

Las Redes Definidas por Software (SDN) han transformado la manera de administrar la infraestructura de red al separar el plano de control del plano de datos y concentrar la lógica de decisión en un controlador centralizado y programable. Esta arquitectura aporta flexibilidad, visibilidad y control granular sobre el tráfico, pero introduce también un punto crítico: el controlador se convierte en un objetivo de alto valor cuya afectación puede comprometer toda la red.

En paralelo, los servicios de streaming multimedia se han consolidado como una de las cargas de tráfico dominantes en Internet. Se trata de servicios especialmente sensibles a la calidad de servicio (QoS): pequeñas variaciones en la latencia, el jitter, la pérdida de paquetes o el rendimiento se traducen de forma inmediata en una degradación perceptible para el usuario. En un entorno SDN, la continuidad de estos flujos depende tanto del plano de datos como de la disponibilidad del controlador.

Los ataques de denegación de servicio distribuido (DDoS) representan una de las amenazas más relevantes en este contexto. Al saturar los recursos de la red o, específicamente, los del controlador SDN y sus tablas de flujo, un ataque DDoS puede interrumpir o degradar significativamente la transmisión multimedia. Comprender y cuantificar este impacto es indispensable para diseñar redes SDN resilientes que sostengan servicios de streaming en condiciones adversas.

El presente trabajo tiene como objetivo evaluar el impacto de los ataques DDoS sobre la calidad de servicio de la transmisión de streaming multimedia en entornos SDN. Para ello, se diseña e implementa un entorno de emulación en GNS3, virtualizado con Docker, que integra un servidor de streaming (MediaMTX), conmutadores Open vSwitch, un generador de tráfico y de ataques desarrollado para el estudio y un sistema de monitoreo de tráfico con ntopng. La

evaluación compara los controladores OpenDaylight y Ryu mediante métricas de QoS (RTT, jitter, pérdida de paquetes y throughput), tanto en condiciones normales como con distintas intensidades de ataque. A partir de los hallazgos se formula, además, una propuesta preliminar de mitigación.

El documento se organiza en cuatro capítulos: el Capítulo I presenta los antecedentes, el problema, los objetivos y el alcance; el Capítulo II desarrolla el marco teórico sobre SDN, ataques DDoS y streaming multimedia; el Capítulo III describe la metodología, el diseño y la implementación del entorno de emulación; y el Capítulo IV expone los resultados, el análisis comparativo entre controladores, la extrapolación indicativa y una propuesta preliminar de mitigación como trabajo futuro. Finalmente, se presentan las conclusiones y recomendaciones.

CAPÍTULO I: ANTECEDENTES

Este capítulo presenta las motivaciones que dieron origen al estudio, describe el problema identificado y expone las razones que justifican su desarrollo. A partir de ello, se formulan el objetivo general y los objetivos específicos que permiten alcanzarlo. Por último, se delimita el alcance del trabajo y se describe la metodología empleada para cumplir cada una de sus fases.

1.1. Problema de investigación

El uso de Redes Definidas por Software (SDN) ha transformado la forma en que se gestionan las redes, proporcionando flexibilidad y eficiencia en la administración de recursos. Sin embargo, esta tecnología también ha expuesto a los sistemas de streaming multimedia a nuevas vulnerabilidades, particularmente ante ataques Distribuidos de Denegación de Servicio (DDoS). Estos ataques pueden saturar la red, interrumpiendo la transmisión de datos y afectando la calidad del servicio (QoS) y la experiencia del usuario. A pesar de la creciente dependencia de aplicaciones de streaming, el impacto específico de los ataques DDoS en la calidad de estos servicios en entornos SDN no ha sido suficientemente explorado, lo que plantea una problemática significativa en el ámbito de la ciberseguridad y la gestión de redes [1].

Los ataques DDoS se caracterizan por inundar la red con tráfico malicioso de forma distribuida, lo que puede llevar a la degradación o interrupción total del servicio [2]. En el contexto de las redes SDN, donde el controlador centralizado gestiona el flujo de datos, un ataque DDoS puede resultar particularmente devastador. Investigaciones previas han demostrado que estos ataques pueden causar una disminución drástica en el rendimiento de la red, afectando métricas críticas como la tasa de pérdida de paquetes, el retardo y la variación del retardo, lo que se traduce en una experiencia de usuario insatisfactoria [3]; por lo cual, a falta de análisis específico sobre cómo estos ataques impactan la calidad de los servicios de streaming multimedia en redes SDN subraya la necesidad de una investigación más profunda en esta área.

Además, el crecimiento exponencial del tráfico de video en Internet, que representa más del 90% del tráfico total, resalta la urgencia de abordar esta problemática; por cuanto, a demanda de contenido multimedia ha llevado a una mayor implementación de tecnologías SDN, que, aunque ofrecen beneficios en términos de gestión de recursos, también presentan desafíos en términos de seguridad [4]; por ello, la interrupción de servicios de streaming puede tener consecuencias económicas significativas para las empresas que dependen de estos servicios para su funcionamiento, lo que hace que la investigación sobre la mitigación de ataques DDoS sea una prioridad [5].

Por último, la creciente complejidad de las redes y la diversidad de aplicaciones que dependen de la transmisión de video en tiempo real hacen que la investigación sobre el impacto de los ataques DDoS en la calidad de servicio sea aún más relevante; en vista a lo cual, las estrategias de mitigación deben ser desarrolladas y evaluadas para garantizar que las redes SDN puedan soportar la carga de tráfico sin comprometer la calidad del servicio [6]. Este estudio busca llenar el vacío existente en la literatura al analizar cómo los ataques DDoS afectan la calidad de servicio de los sistemas de streaming multimedia diferentes controladores SDN, proporcionando así un marco para futuras investigaciones y desarrollos en este campo.

1.2. Justificación

El Estado ecuatoriano a través de la implementación de la "Estrategia Nacional de Ciberseguridad" ha propuesto mejorar la ciberseguridad, que abarca un periodo de tres años (2022-2025). Esta estrategia se fundamenta en seis ejes de acción: gobernanza y coordinación nacional, resiliencia cibernética, prevención y combate a la ciberdelincuencia, ciberdefensa, desarrollo de habilidades en ciberseguridad, y cooperación internacional. Además, se busca fortalecer las capacidades institucionales y legales para abordar los riesgos cibernéticos, fomentar la confianza digital y proteger los derechos de los ciudadanos en el entorno digital. La colaboración con actores nacionales e internacionales es crucial para enfrentar las amenazas emergentes y asegurar un Ecuador más seguro en el ciberespacio [7].

En base a los Objetivos de Desarrollo Sostenible (ODS) de la ONU, se asocia al N°9: Industria, Innovación e Infraestructura. Este objetivo busca construir infraestructuras resilientes y promover la innovación, lo cual es crucial para garantizar servicios de streaming estables y de alta calidad [8]. La implementación de redes definidas por software (SDN) puede ayudar a mitigar estos ataques, mejorando así la experiencia del usuario y contribuyendo al desarrollo tecnológico sostenible.

Actualmente, la investigación sobre el impacto de los ataques DDoS en redes SDN es limitada, lo que crea un vacío significativo en la literatura existente. Aunque se han realizado estudios sobre la mitigación de ataques DDoS y la mejora de QoS en redes tradicionales, el análisis específico de cómo estos ataques afectan los servicios de streaming multimedia en entornos SDN aún no ha sido suficientemente explorado. Esta falta de investigación puede llevar a una subestimación de los riesgos y a la implementación de medidas de seguridad inadecuadas,

resultando en pérdidas económicas y de reputación para las empresas que dependen de estos servicios [9].

Proponer una arquitectura que permita analizar y medir el impacto de los ataques DDoS en la QoS en tiempo real es crucial para crear redes más resilientes. Esta investigación no solo contribuirá al desarrollo de nuevas estrategias de mitigación, sino que también proporcionará un marco para la implementación de mejores prácticas en la gestión de redes SDN. Evaluar el rendimiento de los sistemas de streaming bajo condiciones de ataque permitirá a los administradores de red tomar decisiones informadas y proactivas para proteger sus infraestructuras [10].

El análisis de los efectos de los ataques DDoS en la calidad de los servicios de streaming multimedia tiene implicaciones prácticas significativas para la industria. A medida que más empresas adoptan tecnologías SDN para optimizar la entrega de contenido, es esencial que comprendan los riesgos asociados y desarrollen mecanismos de defensa adecuados. La investigación propuesta beneficiará a los proveedores de servicios de streaming y contribuirá al avance del conocimiento en ciberseguridad y gestión de redes, proporcionando un recurso valioso para académicos y profesionales [11]

1.3. Objetivos

1.3.1. Objetivo general

Evaluar el impacto de los ataques DDoS en la calidad de servicio de la transmisión de streaming multimedia en entornos SDN.

1.3.2. Objetivos específicos

- Analizar el funcionamiento e infraestructura de una red SDN, así como la transmisión del servicio de streaming multimedia
- Establecer requerimientos para el diseño de escenarios de simulación que permitan el desarrollo del ataque DDoS en los servicios multimedia de streaming en diferentes controladores de redes SDN.
- Levantar los escenarios de simulación en base a los requerimientos establecidos para llevar a cabo la simulación para la evaluación de las métricas QoS como latencia, perdida de paquetes y jitter
- Analizar los resultados obtenidos del comportamiento de cada controlador estudiado

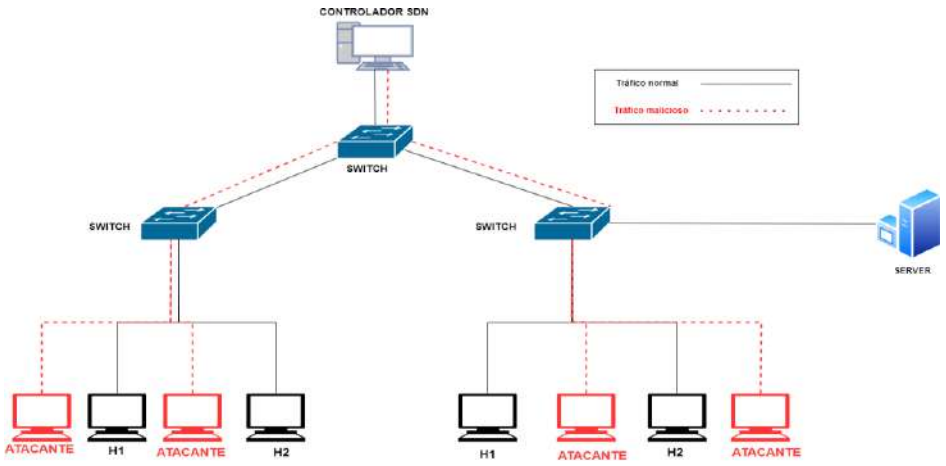
1.4. Alcance

El alcance de este estudio se centra en el análisis detallado del impacto de los ataques Distribuidos de Denegación de Servicio (DDoS) en la calidad del servicio de streaming multimedia en redes definidas por software (SDN). Se investigarán los efectos de estos ataques sobre la calidad de la transmisión multimedia. Además, se desarrollará un modelo de arquitectura de red SDN que permita simular y medir en tiempo real el impacto de los ataques DDoS. A partir de los hallazgos, se podrán formular recomendaciones prácticas opcionales para mitigar los efectos de estos ataques en los servicios de streaming, contribuyendo así al avance del conocimiento en ciberseguridad. El desarrollo del proyecto se basará en la metodología de simulación discreta, que se alinea estrechamente con los objetivos planteados. Las fases incluirán: definición del problema, modelado del sistema, implementación de la simulación, ejecución y análisis de resultados. Fase Definición Problema, se llevará a cabo una exhaustiva revisión de la literatura académica y técnica

para recopilar información que permita identificar las consecuencias de los ataques DDoS en redes definidas por software (SDN). Este análisis facilitará la identificación de los requisitos necesarios para la implementación de una red SDN eficaz. Se examinarán los diferentes tipos de controladores, así como el hardware y software esenciales para su funcionamiento. Además, se estudiará el funcionamiento de los servicios de streaming multimedia, prestando especial atención a las especificaciones de calidad del servicio (QoS). Este enfoque meticuloso garantizará que los requisitos identificados sean completos, consistentes y viables, estableciendo así una base sólida para el modelado del sistema. Fase Modelado del sistema, se toma como base los hallazgos obtenidos en la fase anterior para la elección de las mejores herramientas que se acoplen a las necesidades para el desarrollo del proyecto; a partir de lo cual, se define la arquitectura de la red SDN, detallando los componentes clave como controladores, switches; así como, las métricas de calidad en el servicio streaming multimedia a evaluar. Además, se especifican los tipos de ataques DDoS que se simularán dentro del modelo, describiendo sus características y métodos de implementación. Para la selección de los controladores en base a las investigaciones precias se hará la elección de los más implementados dentro del área de redes. La Figura 1 muestra el escenario con una topología de la idea del proyecto a realizar en la cual se considera una topología en árbol la cual consta de un controlador, servidor de streaming, equipos de distribución, hosts.

Figura 1

Topología de un ataque DDoS dirigido al servidor de streaming en una red SDN



Fase Implementación de simulación, se configura y despliega los escenarios utilizando los controladores SDN seleccionados, conforme a los modelos establecidos en un entorno de simulación. Una vez que la configuración del controlador SDN y la del servidor de streaming se haya realizado con éxito, se establecerán los parámetros necesarios para iniciar los ataques DDoS en un entorno de prueba controlado. Esto permitirá evaluar su impacto en la red y en los parámetros de calidad de servicio del streaming multimedia en la siguiente fase. Este proceso se repetirá con cada uno de los controladores elegidos para facilitar un análisis exhaustivo posterior. Fase Ejecución, se llevarán a cabo múltiples ejecuciones exhaustivas para asegurar que tanto la red SDN como las simulaciones de ataques operen correctamente en una variedad de escenarios. Se variará la intensidad de los ataques DDoS para evaluar su impacto en las métricas de calidad del servicio (QoS). Este enfoque permitirá obtener una comprensión detallada de cómo diferentes niveles de agresividad en los ataques afectan el rendimiento del sistema, garantizando así la validez y robustez de los resultados obtenidos. Fase Análisis, se realizará de manera exhaustiva en base resultados obtenidos para estimar el comportamiento en entornos reales. Se realizará un análisis comparativo entre los diferentes controladores utilizados en el entorno simulado, evaluando su

rendimiento frente a los ataques DDoS. Además, se compararán los recursos empleados en la simulación con aquellos utilizados en situaciones reales. Este enfoque permitirá recomendaciones efectivas para mitigar el impacto de los ataques DDoS en las métricas de calidad del servicio (QoS).

2. CAPÍTULO II: MARCO TEÓRICO

En este capítulo se presenta una revisión bibliográfica de trabajos previos relacionados con el tema de investigación, con el objetivo de describir los conceptos teóricos de las redes definidas por software (SDN), su funcionamiento y su configuración en entornos de emulación. Además, se identifican las vulnerabilidades asociadas a la misma y el impacto que pueden causar los ataques de denegación de servicio distribuido (DDoS) sobre su funcionamiento.

De igual manera, se describen los servicios de streaming multimedia y el marco normativo que sustenta la evaluación de la calidad de servicio en este trabajo.

2.1. Redes definidas por software (SDN)

Las redes definidas por software (SDN) son un conjunto de técnicas de redes informáticas que permiten implementar servicios de red de manera dinámica y escalable, a partir de una lógica de control centralizada en un controlador basado en software y no en el hardware de cada dispositivo. Este enfoque permite abordar problemas de seguridad comunes de las redes convencionales mediante aplicaciones de seguridad implementadas en software [12].

En esencia, las redes SDN transforman el paradigma tradicional de administración de redes, ya que ofrecen un control más inteligente, flexible y eficiente de la infraestructura de comunicaciones, lo que permite al administrador implementar optimizaciones y aprovisionar nuevos servicios con rapidez. Esta agilidad facilita la adaptación a las cambiantes necesidades del negocio y a los avances tecnológicos [13].

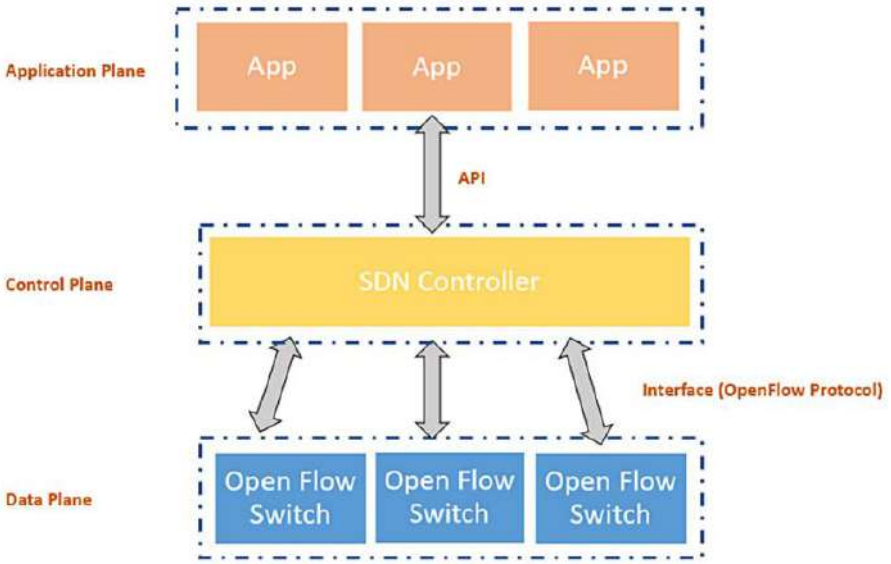
2.1.1. Arquitectura de SDN

El concepto básico de las redes definidas por software consiste en separar el plano de datos del plano de control para crear un punto de gestión centralizado, que define el funcionamiento de los dispositivos de red en función de los servicios y aplicaciones, según un modelo de tres capas.

Las capas se comunican mediante interfaces, generalmente APIs. La Figura 2 muestra la arquitectura SDN de tres capas, formada por los planos de aplicación, control y datos [14].

Figura 2

Capas de la arquitectura de red SDN



Nota. Tomado de [14].

El plano de aplicación contiene las aplicaciones SDN que determinan el comportamiento de la red: aplican mecanismos y políticas de seguridad, realizan tareas de administración y ejecutan servicios dentro de la red SDN [14].

El plano de datos está formado por los conmutadores que reenvían los paquetes de usuario según las reglas de las tablas de flujo instaladas por el controlador, sin tomar decisiones de encaminamiento propias [15].

El plano de control utiliza información de señalización para tomar decisiones de control sobre un espectro de capas más amplio que en las redes tradicionales. Integra el controlador y las aplicaciones de control que gestionan la red de manera dinámica y flexible, lo que permite una administración más adaptativa de los recursos de comunicación [15].

2.1.2. Controladores SDN

Los controladores SDN cumplen varias funciones esenciales para el funcionamiento de las redes modernas: determinan las rutas más eficientes para el tráfico, mantienen y gestionan la información de configuración de toda la infraestructura, establecen parámetros en los dispositivos de red y comunican el estado operativo del sistema [16].

En el ecosistema SDN existen diversos controladores, como NOX, POX, Floodlight, Ryu, OpenDaylight y Beacon. La Tabla 1 expone el lenguaje de programación y la versión de OpenFlow que soportan los controladores más utilizados [16].

Tabla 1

Controladores SDN

	ODL	Ryu	NOX
Soporte OpenFlow	OF v1.0	OF v1.0, v1.2, v1.3	OF v1.0
Virtualización	Mininet y Open vSwitch	Mininet y Open vSwitch	Mininet y Open vSwitch
Lenguaje	Java	Python	C++

2.1.2.1. OpenDaylight

Fundado en 2013, el proyecto OpenDaylight forma parte de LF Networking (LFN), entidad que integra la gobernanza de los proyectos participantes para mejorar la excelencia operativa, simplificar la participación de los miembros y aumentar la colaboración entre proyectos de redes de código abierto y organismos de normalización [17].

El controlador OpenDaylight se ejecuta sobre la máquina virtual de Java (JVM), por lo que funciona en cualquier sistema operativo y hardware compatibles con Java. Constituye una

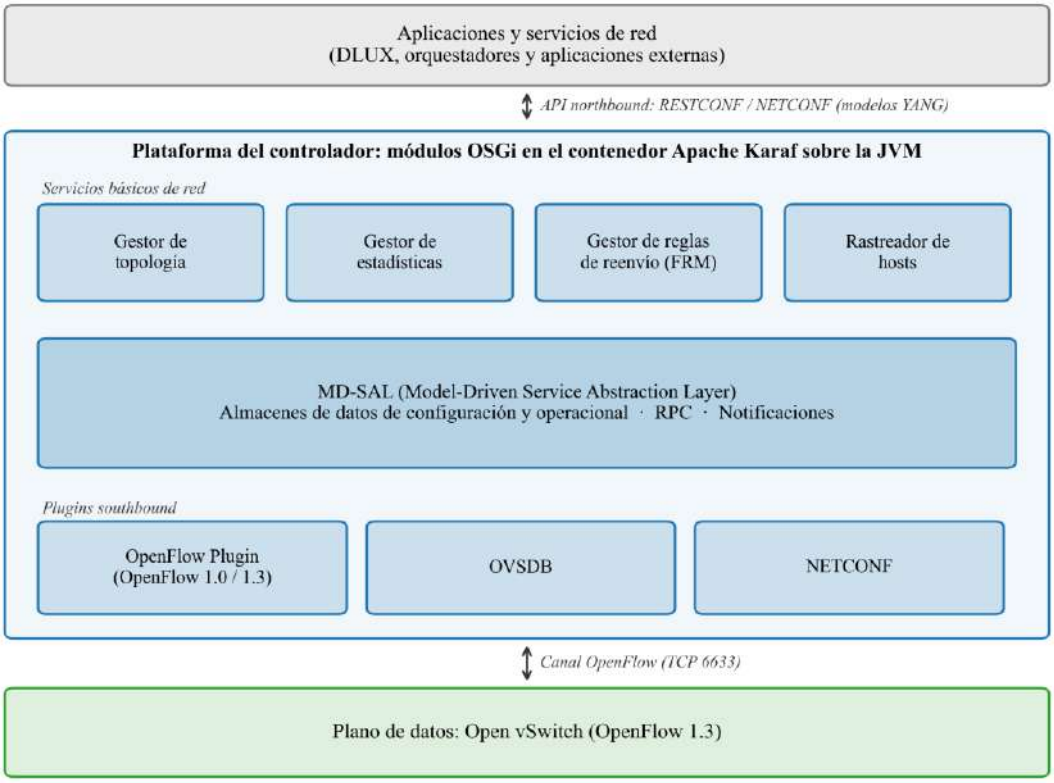
implementación del concepto de red definida por software (SDN) y utiliza las siguientes herramientas [18]:

- Maven: herramienta de automatización de la compilación. Utiliza el archivo pom.xml (modelo de objetos del proyecto) para declarar las dependencias entre paquetes y describir qué paquetes se cargan e inician.
- OSGi: marco que constituye el back-end de OpenDaylight; permite la carga dinámica de paquetes y archivos JAR y su vinculación para intercambiar información.
- Interfaces Java: permiten escuchar eventos y definir las especificaciones y los patrones de interacción entre los módulos.
- API REST: interfaces en dirección norte (northbound), como el administrador de topología, el rastreador de hosts, el programador de flujos y el enrutamiento estático.

La arquitectura de OpenDaylight se organiza en capas, como muestra la Figura 3. En la capa superior se ubican las aplicaciones y los servicios de red, que interactúan con el controlador mediante las interfaces northbound RESTCONF y NETCONF, basadas en modelos YANG. El núcleo de la plataforma es la capa de abstracción de servicios dirigida por modelos (MD-SAL, Model-Driven Service Abstraction Layer), que mantiene los almacenes de datos de configuración y operacional y canaliza las llamadas RPC y las notificaciones entre módulos; sobre ella operan servicios básicos como el gestor de topología, el gestor de estadísticas, el gestor de reglas de reenvío (FRM, Forwarding Rules Manager) y el rastreador de hosts. En la capa inferior, los plugins southbound (OpenFlow Plugin, OVSDB y NETCONF) traducen las operaciones del MD-SAL a los mensajes del protocolo de cada dispositivo. Todos estos componentes se empaquetan como módulos OSGi dentro del contenedor Apache Karaf, que se ejecuta sobre la JVM [18].

Figura 3

Arquitectura del controlador OpenDaylight



Nota. Tomado de [18].

Esta organización modular facilita la extensibilidad y la operación en redes de producción, incluido el funcionamiento en clúster, pero introduce varias capas entre la llegada de un mensaje OpenFlow y la instalación de la regla correspondiente: el mensaje se convierte en una notificación del MD-SAL, la procesa el servicio que corresponda y la regla resultante se escribe en el almacén de configuración antes de que el OpenFlow Plugin la envíe al conmutador como un mensaje FLOW_MOD. A ello se suma el consumo de memoria y de CPU de la JVM y del contenedor Karaf.

2.1.2.2. Ryu

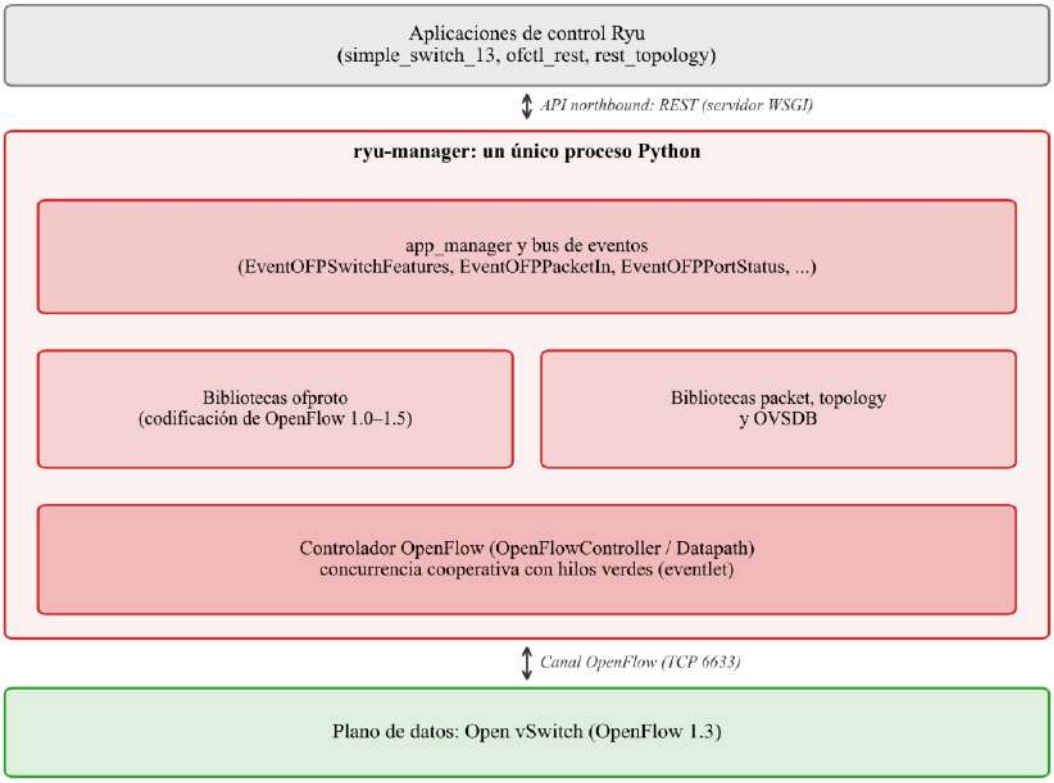
Ryu es un marco de trabajo SDN basado en componentes. Ofrece módulos con APIs bien estructuradas que permiten a los desarrolladores crear con facilidad nuevas aplicaciones de control y gestión de red. Además, soporta diversos protocolos para la administración de dispositivos de red, como OpenFlow, NETCONF y OF-Config; en particular, es compatible con las versiones 1.0, 1.2, 1.3, 1.4 y 1.5 de OpenFlow y con las extensiones de Nicira. Su código está disponible de forma libre bajo la licencia Apache 2.0 [19].

Ryu resulta adecuado para la gestión y automatización de redes, ya que permite definir comportamientos de red en tiempo real, lo que aumenta la flexibilidad y la escalabilidad. Se utiliza con frecuencia junto con Mininet para investigar SDN y controlar el flujo de paquetes en los conmutadores de red [20].

Ryu adopta una arquitectura más sencilla, basada en componentes que se ejecutan dentro de un único proceso Python (Figura 4). El programa ryu-manager carga las aplicaciones de control (en este trabajo, simple_switch_13 y las aplicaciones REST) y las comunica mediante un bus de eventos. El componente controlador OpenFlow mantiene la conexión TCP con cada conmutador, decodifica los mensajes con las bibliotecas ofproto y los entrega a las aplicaciones como eventos (por ejemplo, EventOFPPacketIn); las aplicaciones responden enviando directamente al conmutador los mensajes FLOW_MOD o PACKET_OUT. La concurrencia se gestiona con hilos verdes de la biblioteca eventlet, y la interfaz northbound se ofrece mediante un servidor WSGI que expone APIs REST [19].

Figura 4

Arquitectura del controlador Ryu



Nota. Tomado de [19].

El camino entre un evento del plano de datos y la respuesta del controlador atraviesa menos capas que en OpenDaylight, no requiere un almacén de datos intermedio y demanda menos memoria y CPU. Como contrapartida, el modelo de un único proceso con concurrencia cooperativa limita la escalabilidad y la alta disponibilidad, características para las que OpenDaylight está diseñado. Esta diferencia arquitectónica es relevante para interpretar los resultados del Capítulo IV, ya que en el laboratorio ambos controladores comparten los recursos del mismo equipo anfitrión.

2.1.2.3. NOX

El controlador NOX es una de las primeras plataformas de código abierto en el ámbito de las redes definidas por software (SDN). Funciona como un sistema operativo de red (NOS) que administra conmutadores OpenFlow y permite separar el plano de control del plano de datos. Está desarrollado principalmente en C++, lo que le otorga un alto rendimiento, y se utiliza en entornos de investigación, en emulaciones con Mininet y en aplicaciones de control de tráfico y seguridad de redes [21]. La Tabla 2 compara los controladores habituales en redes SDN según varias características relevantes para el proyecto.

Tabla 2

Comparación entre los controladores habituales en redes SDN

Criterio	Ryu	OpenDaylight (ODL)	NOX
Lenguaje	Python	Java (OSGi / Karaf)	C++ (original) / Python (POX fork)
Versión OpenFlow	OF 1.0–1.5	OF 1.0–1.5 + otros	OF 1.0
Arquitectura	Componentes / módulos Python. Ligero, orientado a aplicaciones.	Microservicios OSGi. Arquitectura modular empresarial (SAL, AD-SAL, MD-SAL).	Monolítico. Bucle de eventos de bajo nivel.

Criterio	Ryu	OpenDaylight (ODL)	NOX
Escalabilidad	Media — adecuada para laboratorios y entornos medianos.	Alta — diseñado para redes de producción a gran escala. Soporta clustering.	Baja — pensado para investigación y prototipos pequeños.
Facilidad de uso	Alta — API sencilla, documentación clara, ideal para estudiantes.	Baja — curva de aprendizaje alta, configuración compleja.	Media — bajo nivel, requiere conocimientos de C++/redes.
Interfaz gráfica	No nativa. Se puede integrar con REST + herramientas externas.	DLUX (GUI web integrada) + API REST/NETCONF/YANG.	No dispone de GUI. Solo línea de comandos.
API REST	Sí — módulo integrado (wsgi).	Sí — API REST completa + soporte NETCONF/YANG.	No nativa — requiere desarrollo personalizado.
Comunidad / soporte	Activa en entornos académicos. Mantenimiento irregular desde 2020.	Respaldada por Linux Foundation. Gran comunidad empresarial activa.	Inactiva. Proyecto académico, sin actualizaciones recientes.
Compatibilidad con GNS3 / Mininet	Excelente — uso estándar en laboratorios SDN con Mininet y GNS3.	Buena — funciona con Mininet, requiere más recursos.	Limitada — compatible con Mininet, no

Criterio	Ryu	OpenDaylight (ODL)	NOX
			recomendado para GNS3.
Uso recomendado	Investigación, laboratorios académicos, proyectos de tesis SDN.	Entornos empresariales, data centers, redes de producción.	Investigación histórica, comprensión de conceptos base SDN.
Detección DDoS / seguridad	Módulos disponibles. Fácil integración con scripts de detección en Python.	Módulos de seguridad robustos (SNBI, AAA). Soporte para análisis de tráfico.	Sin módulos de seguridad. Implementación manual requerida.
Licencia	Apache 2.0	Eclipse Public License (EPL)	GPL v3

2.1.3. Protocolos de comunicación SDN

La comunicación en redes SDN se basa en la separación del plano de control y el plano de datos, que permite la gestión centralizada mediante un controlador de software. Para ello, se emplean diversos protocolos estandarizados que permiten instruir a los dispositivos de red (routers y switches) para gestionar el tráfico de manera dinámica y flexible [22].

2.1.3.1. Protocolo OpenFlow

El protocolo OpenFlow, definido por la Open Networking Foundation (ONF), es la primera interfaz estándar diseñada específicamente para la comunicación entre la capa de control y la capa de infraestructura de la arquitectura SDN [23].

OpenFlow proporciona un protocolo abierto para programar el flujo en diferentes dispositivos de red, como conmutadores y enrutadores. Este protocolo permite acceder directamente y manipular el plano de direccionamiento de dichos dispositivos, ya sean físicos o virtuales [23].

Los elementos que conforman la arquitectura son los siguientes [23]:

1 Controlador SDN: software encargado de la conexión, configuración y administración de los conmutadores SDN de la red.

2 Tablas de flujo: se instalan en cada conmutador y determinan las acciones que se aplican al tráfico de red.

3 Conmutadores SDN: físicos o virtuales encargados del reenvío de los paquetes dentro de la red SDN.

2.1.3.2. Mensajes OpenFlow

Los mensajes OpenFlow son unidades de comunicación en las redes SDN que permiten la interacción entre el controlador y los conmutadores de la red. La especificación de OpenFlow [24] establece tres categorías de mensajes: asincrónicos, simétricos y de controlador a conmutador.

Tanto el controlador como los conmutadores pueden enviar mensajes simétricos, que el protocolo OpenFlow utiliza para intercambiar información sobre el funcionamiento de la red [24]. Los más relevantes son los siguientes:

- OFPT_HELLO: consta de un encabezado OpenFlow y de un conjunto de elementos de saludo con datos opcionales que intervienen en la negociación inicial de la conexión.
- Echo Request: consta de un encabezado OpenFlow y de un campo de datos que puede contener una marca de tiempo para medir la latencia, datos de diversa longitud para medir el ancho de banda o ningún dato, para verificar la continuidad entre el conmutador y el controlador.
- Echo Reply: consta de un encabezado OpenFlow y del campo de datos, sin modificar, del mensaje de solicitud de eco.
- OFPT_ERROR_MSG: lo utilizan el conmutador o el controlador para notificar problemas; el conmutador lo emplea principalmente para indicar el fallo de una solicitud iniciada por el controlador.

Los mensajes de controlador a conmutador los inicia el controlador y se utilizan para solicitar funciones específicas o enviar paquetes por la ruta de datos del conmutador [24]. Los mensajes más destacados son:

- OFPT_FEATURES_REQUEST y OFPT_FEATURES_REPLY: mensajes de solicitud y respuesta que intercambian información sobre la identidad y las capacidades de los conmutadores. Se envían cuando el controlador solicita estas capacidades al establecerse una nueva conexión en el canal de control.
- OFPT_FLOW_MOD: permite agregar, modificar o eliminar entradas de la tabla de flujo.
- OFPT_PORT_STATS: el controlador envía este mensaje para solicitar estadísticas de uno o varios puertos del conmutador, como los paquetes y bytes recibidos o transmitidos.

2.1.3.3. Protocolo OVSDB

El Open vSwitch Database Management Protocol (OVSDB), definido en la RFC 7047, es un protocolo de red utilizado para administrar y configurar instancias de Open vSwitch (OVS).Emplea JSON-RPC 1.0 para realizar operaciones de inserción, eliminación y actualización sobre una base de datos transaccional basada en esquemas, y permite a los controladores remotos monitorear y modificar la configuración de los conmutadores [25].

Los aspectos clave de OVSDB definidos en la RFC 7047 son los siguientes [26]:

- Funcionalidad: OVSDB permite a los controladores SDN administrar OVS, incluida la creación de puentes, la adición de puertos y la configuración de túneles VXLAN.
- Estructura del protocolo: el protocolo opera sobre JSON-RPC 1.0, lo que permite a los clientes enviar transacciones al ovsdb-server.
- Operaciones clave: get_schema, transact (cambios en la base de datos), monitor (seguimiento de las actualizaciones de la base de datos) y lock (control del acceso a los recursos).
- Estructura de la base de datos: un esquema define las tablas, las columnas y las restricciones, lo que garantiza la integridad de los datos.
- Componentes: la implementación incluye ovsdb-server (el demonio de la base de datos) y ovsdb-client (cliente de interacción), junto con herramientas como ovsdb-tool para gestionar las bases de datos en disco.

2.2. Ataques DDoS

Según [6], un ataque de denegación de servicio distribuido (DDoS) es un ciberataque en el que el atacante utiliza más de una máquina, normalmente comprometida, para que los servicios o

recursos de la red dejen de estar disponibles para sus usuarios legítimos. Un ataque DDoS suele desarrollarse en cuatro fases: reclutamiento, explotación, infección y utilización.

2.2.1. Fases de un ataque DDoS

Las cuatro fases de un ataque DDoS son las siguientes [14]:

- Fase de reclutamiento: el atacante escanea la red en busca de máquinas remotas con vulnerabilidades de seguridad que le permitan acceder a ellas.
- Fase de explotación: el atacante aprovecha las vulnerabilidades encontradas para acceder a las máquinas.
- Fase de infección: el atacante inyecta en dichas máquinas el código que le permite controlarlas.
- Fase de utilización: las máquinas comprometidas se utilizan para lanzar la carga útil del ataque.

2.2.2. Tipos de ataques DDoS

Los ataques DDoS representan un desafío significativo para la seguridad en línea, ya que utilizan numerosas fuentes para sobrecargar sus objetivos con tráfico excesivo. Cada variante presenta rasgos distintivos y consecuencias específicas, por lo que conocer sus diferentes modalidades resulta esencial para diseñar contramedidas efectivas. Según su naturaleza y propósito, estos ataques se clasifican en volumétricos, de protocolo y de capa de aplicación [14].

2.2.2.1. Ataques volumétricos

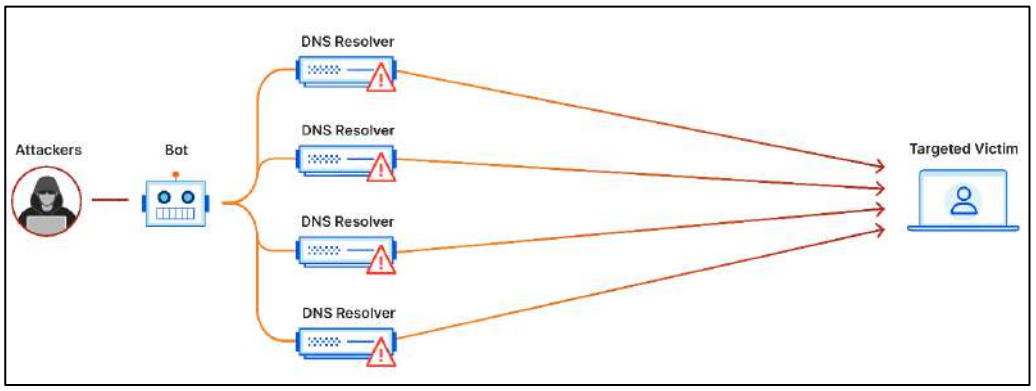
Los ataques volumétricos buscan saturar por completo los recursos de red de la víctima. El sistema objetivo recibe una avalancha de tráfico que supera su capacidad de procesamiento; uno de los métodos más comunes es la inundación ICMP, en la que múltiples peticiones de eco sobrepasan la capacidad de respuesta del sistema. Una variante es el ataque Smurf, que amplifica

el impacto al enviar peticiones ICMP a direcciones de difusión con la dirección IP de la víctima como origen [27].

Dentro de esta categoría se encuentra también la inundación UDP, en la que el atacante, mediante direcciones IP falsificadas, envía paquetes UDP al objetivo hasta agotar su ancho de banda y paralizar su capacidad de atender a los usuarios legítimos [28]. La Figura 5 muestra el funcionamiento de un ataque volumétrico que inunda un servidor DNS para impedir el acceso de la víctima a la información.

Figura 5

Esquema de un ataque volumétrico



Nota. Tomado de [29].

2.2.2.2. Ataques de protocolo

Un ataque de protocolo es un tipo de ataque DDoS dirigido a protocolos de red específicos, que busca interrumpir los servicios y agotar los recursos del sistema de destino.

- Inundación TCP SYN: el atacante envía a la víctima numerosos segmentos SYN con direcciones IP de origen falsas. Como estas direcciones son suplantadas, la víctima nunca recibe el acuse de recibo final, de modo que la cola de conexiones

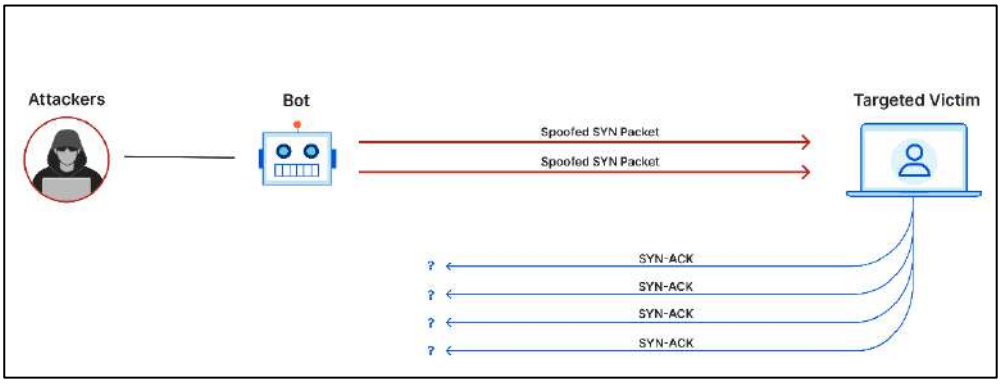
semiabiertas (backlog) del servidor se agota y el puerto atacado deja de aceptar nuevas conexiones de usuarios legítimos [27].

- Agotamiento de conexiones TCP: el atacante abre y mantiene un gran número de conexiones TCP completas contra un puerto de servicio, consumiendo la tabla de conexiones y los recursos del servidor hasta impedir que atienda a usuarios legítimos. Es el principio del ataque tcp_exhaust empleado en este trabajo [30].

La Figura 6 muestra el funcionamiento de un ataque DDoS de protocolo, en el que el envío masivo de paquetes TCP SYN impide que la víctima establezca conexiones con el servicio.

Figura 6

Ataque DDoS de protocolo sobre TCP



Nota. Tomado de [29].

2.2.2.3. Ataques de capa de aplicación

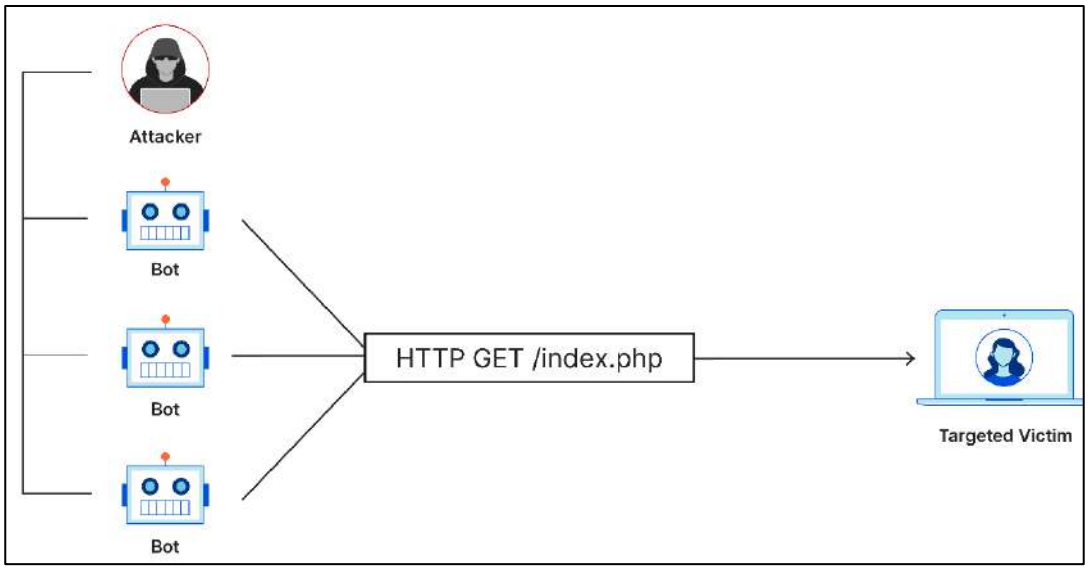
Un ataque de capa de aplicación se dirige a la capa de aplicación del modelo OSI y explota vulnerabilidades de las aplicaciones de software para interrumpir los servicios [31].

- Explotación de debilidades: estos ataques aprovechan fallas en aplicaciones web, bases de datos u otros sistemas de software, y pueden iniciar procesos que consumen recursos del servidor, como memoria y espacio en disco.

- Consumo de recursos: al enviar numerosas solicitudes de transacción, estos ataques pueden saturar el servidor e impedir que procese las solicitudes legítimas, lo que provoca la indisponibilidad del servicio. La Figura 7 muestra el esquema de un ataque DDoS dirigido a una aplicación web mediante solicitudes HTTP.
- Bajo requerimiento de ancho de banda: a diferencia de otros tipos de ataque, los de capa de aplicación pueden causar daños significativos con un ancho de banda mínimo, lo que dificulta su detección, ya que se mezclan con el tráfico legítimo [31].
- Inundación HTTP: el atacante envía un gran volumen de solicitudes HTTP aparentemente legítimas sobre conexiones TCP establecidas, agotando la capacidad de procesamiento del servidor web. Al requerir el handshake completo, no puede emplear direcciones IP suplantadas [32].

Figura 7

Ataque DDoS de capa de aplicación



Nota. Tomado de [29].

2.2.3. Ataques DDoS a redes definidas por software

Las redes definidas por software, aunque ofrecen ventajas significativas de flexibilidad y gestión centralizada, presentan vulnerabilidades inherentes que las exponen a ataques de denegación de servicio. Entre las amenazas más graves se encuentran los ataques de denegación de servicio (DoS), que afectan la disponibilidad de los servicios al agotar el ancho de banda y los recursos de red mediante la generación masiva de solicitudes desde una o múltiples fuentes [21], [33].

El estudio de [33], desarrollado en un entorno emulado con GNS3 y el controlador OpenDaylight, reporta que, antes de los ataques DoS, el consumo de ancho de banda no supera el 34 %, mientras que tras ejecutarlos se duplica y alcanza hasta el 71 % en servidores HTTP y el 82 % en servidores DHCP. De manera similar, la latencia previa a los ataques se sitúa por debajo de 1.5 ms y, después de ellos, aumenta hasta 4.779 ms, lo que indica una sobrecarga en el procesamiento del servicio.

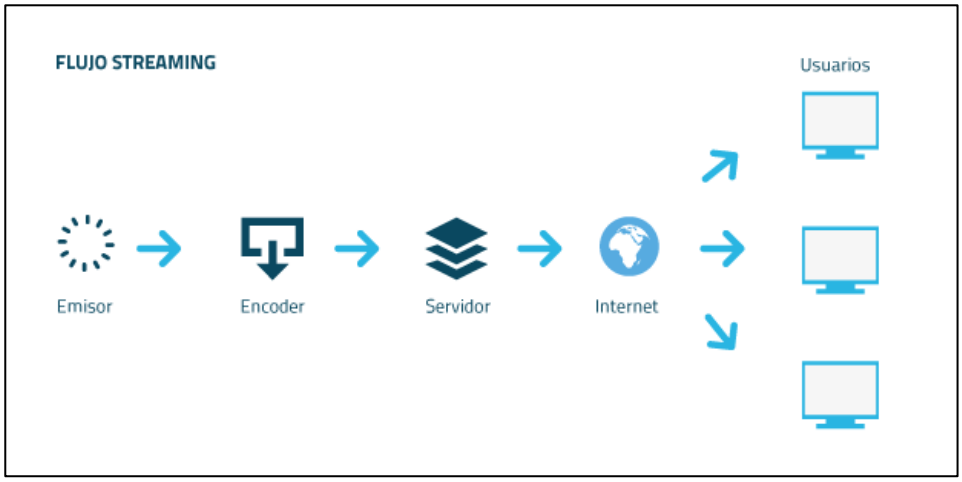
2.3. Servicios de streaming multimedia

Según [34], el streaming es una tecnología que permite distribuir contenido multimedia, principalmente audio y video, a través de una red de computadoras en tiempo real. El contenido se carga en un búfer de datos: a medida que se reproduce en la interfaz del cliente, se descarga y luego se descarta de la unidad de almacenamiento del cliente.

La Figura 8 muestra el proceso de acceso a los servicios de streaming, desde la transmisión del emisor hasta los usuarios finales.

Figura 8

Funcionamiento del streaming



Nota. Tomado de [35].

Para implementarla de forma efectiva, se requiere un ancho de banda mínimo equivalente a la tasa de transferencia del servidor. Si no se cumple este requisito, el flujo de datos presenta interrupciones que comprometen la reproducción y contravienen el propósito de esta: garantizar una transferencia continua del contenido [34].

2.3.1. Componentes

El servicio de streaming se compone de los siguientes elementos básicos [36].

- Códecs: hardware o software que comprime y descomprime el video digital para reducir el tamaño de los archivos y facilitar su almacenamiento y distribución.
- Protocolos: conjunto de normas que posibilitan la transmisión de contenido multimedia, como audio y video, de un punto a otro a través de una red de datos, y que resultan esenciales para la entrega de contenido en tiempo real [37]. Entre los más empleados se encuentran RTP sobre UDP para el transporte de medios, RTSP (Real Time Streaming

Protocol) para el control de sesiones y HLS (HTTP Live Streaming), que entrega el contenido en segmentos sobre HTTP/TCP.

- Precarga: función que administra la cantidad de contenido de video que se carga al iniciar la reproducción; contribuye a optimizar el rendimiento y a reducir el almacenamiento en búfer [34].
- Red de datos: infraestructura de transporte que conecta al servidor con los clientes; su capacidad, retardo, variación del retardo y pérdida determinan la continuidad de la reproducción [38].
- HLS (HTTP Live Streaming), definido en la RFC 8216, divide el flujo en segmentos de pocos segundos que el cliente descarga por HTTP a partir de una lista de reproducción (manifest); al operar sobre TCP, la pérdida de paquetes se manifiesta como retransmisiones y retardo, no como huecos en el video [39].
- RTSP, definido en la RFC 7826, es un protocolo de control que permite establecer, reproducir y pausar sesiones multimedia, y suele transportar los medios mediante RTP [39]. En este trabajo, RTSP se utiliza para publicar el flujo de origen en el servidor, mientras que tanto el cliente como el ataque utilizan el servicio HLS (8888/TCP).

2.3.2. Arquitectura

La arquitectura de streaming describe el proceso de envío, procesamiento y consumo de datos de audio y video en tiempo real. Puede concebirse como un sistema que permite reproducir películas o escuchar música en línea sin descargar previamente el archivo completo [34].

2.3.2.1. Arquitectura centralizada

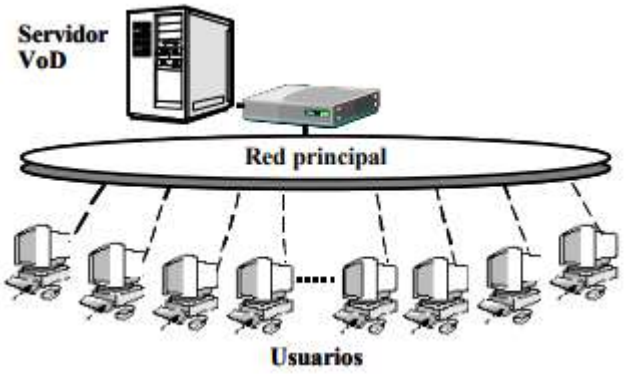
Los sistemas centralizados gestionan todas las peticiones de los usuarios a través de una red principal compartida y se clasifican en dos categorías: arquitecturas con un único servidor y

arquitecturas con múltiples nodos de servicio. En la primera, un solo nodo desde un PC estándar hasta una supercomputadora atiende todas las peticiones, lo que limita la escalabilidad, la tolerancia a fallos y la disponibilidad [40].

Para superar estas limitaciones, la segunda categoría emplea múltiples nodos de servicio, lo que permite construir un sistema escalable mediante la incorporación de nuevos nodos, más tolerante a fallos al eliminar el punto único de fallo y con mayor capacidad para atender la demanda de los usuarios [40]. La Figura 9 muestra el esquema centralizado, en el que todos los usuarios dirigen sus peticiones al único servidor de la red.

Figura 9

Arquitectura de VoD centralizada



2.3.2.2. Arquitectura de servidores independientes

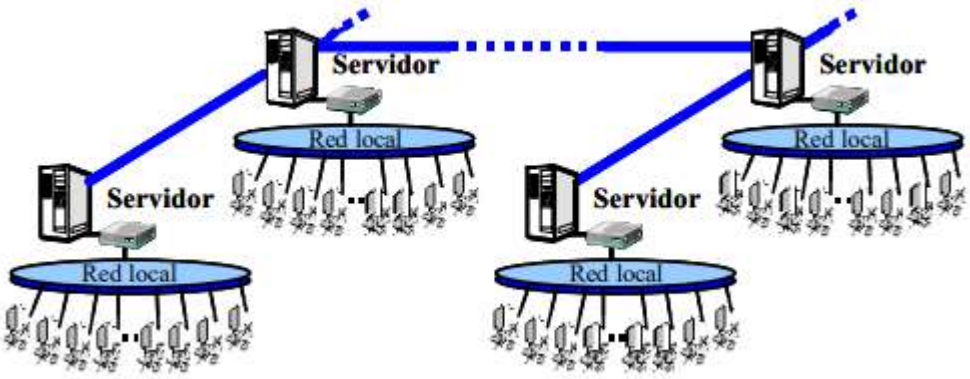
La arquitectura de servidores independientes agrupa a los usuarios en redes locales con tráfico independiente entre sí, ubica servidores de VoD cerca de cada red local y replica los contenidos en cada uno de ellos, de modo que las peticiones se atienden localmente sin acceder a un servidor centralizado. Este enfoque elimina el cuello de botella de un único punto de acceso y proporciona un sistema escalable, autónomo y con mayor ancho de banda acumulado [40], [41].

Como se ilustra en la Figura 10, la arquitectura se compone de múltiples servidores distribuidos, cada uno vinculado a su propia red local de usuarios e interconectados mediante

enlaces de comunicación, lo que permite distribuir la carga de manera eficiente y garantizar la disponibilidad del servicio de forma descentralizada [42].

Figura 10

Arquitectura de VoD de servidores independientes



2.3.2.3. Streaming en redes definidas por software

El streaming en redes SDN consiste en utilizar arquitecturas de red flexibles para optimizar la transmisión de video y audio en tiempo real. A diferencia de las redes tradicionales, en las que cada dispositivo toma decisiones de forma individual, las SDN separan el control de los datos y permiten una gestión centralizada que mejora la calidad del streaming [43].

Las principales ventajas del streaming sobre SDN son las siguientes [44]:

- Gestión centralizada: el controlador SDN ofrece una visión integral de la red, lo que facilita la administración, el monitoreo y la configuración de recursos para mejorar el rendimiento.
- Optimización del tráfico: permite implementar políticas de enrutamiento y priorización de los paquetes de video, lo que reduce el jitter (variación del retardo) y la pérdida de datos.
- Automatización y escalabilidad: mediante APIs abiertas, la red se adapta con rapidez a los cambios en la demanda y permite escalar la infraestructura de manera sencilla.

- Reducción de costos: mejora la eficiencia operativa al aprovechar mejor los recursos de red y permitir la configuración automática.

2.4. Marco normativo para la evaluación de la calidad y el rendimiento de la red

La validación del comportamiento de un servidor de video ante escenarios de estrés por denegación de servicio (DDoS) requiere un marco metodológico riguroso que combine la medición de las métricas del servicio con el análisis estadístico de los resultados. Se exponen las normas y recomendaciones que sirven de base para definir los parámetros experimentales y analizar los resultados.

2.4.1. Criterios de duración de las corridas según las RFC 2544 y RFC 6349

Para evaluar la resiliencia de la infraestructura de red ante eventos de alta demanda o saturación, es fundamental definir una duración de prueba que produzca resultados representativos. Las especificaciones del Grupo de Trabajo de Ingeniería de Internet (IETF) establecen criterios para estos intervalos, diferenciados según la capa en la que se realiza la medición [45].

La **RFC 2544** es el documento de referencia para evaluar el rendimiento de los dispositivos de interconexión de redes [45]. Respecto a la duración de las pruebas, establece lo siguiente:

- Duración **mínima por prueba:** cada prueba de transmisión de tramas destinada a evaluar la pérdida o el *throughput* debe durar al menos **60 s**.
- Justificación **de la ventana temporal:** un intervalo mínimo de un minuto permite que las colas de los búferes de los routers y conmutadores virtuales (en este caso, los nodos emulados en GNS3) alcancen un régimen estable. En pruebas más cortas (de 10 o 20 s), los resultados pueden presentar sesgos transitorios debidos a búferes aún vacíos, que ocultan la degradación real por descarte de paquetes (*tail drop*).

Mientras que la RFC 2544 evalúa las capas 2 y 3, la **RFC 6349** define una metodología específica para medir el rendimiento en la capa de transporte (TCP), aplicable a conexiones como las de un servidor de video [46]:

Tiempos de medición estables: las pruebas de rendimiento TCP deben ejecutarse durante un lapso suficiente para que el protocolo supere la fase de inicio lento (slow start) y se estabilice en la fase de evitación de congestión. En este trabajo se adopta 60 s como duración mínima de referencia.

Análisis del impacto en el tráfico: ante un ataque de inundación TCP, una duración de 60 s por corrida permite capturar múltiples ciclos de retransmisión por temporización (RTO, Retransmission Timeout) y reducciones de la ventana de congestión (CWND). Esto permite observar métricas de la RFC 6349 como TCP Efficiency (porcentaje de bytes transmitidos sin retransmisión) y Buffer Delay Percentage (incremento porcentual del RTT por el llenado de los búferes).

2.4.2. Marco estadístico de rendimiento según la Recomendación ITU-T E.840

La Recomendación **ITU-T E.840** define el marco estadístico para calificar y comparar la calidad de funcionamiento de las redes de extremo a extremo [47]. Respecto al diseño de escenarios y la captura de muestras, establece lo siguiente:

- **Consistencia y reproducibilidad:** los escenarios de prueba deben ser homogéneos y ejecutarse en condiciones controladas (por ejemplo, mediante scripts automatizados) para que las comparaciones de rendimiento sean equitativas (*like-for-like*). En entornos virtuales y de emulación, este principio respalda el uso de un número reducido de corridas, siempre que se demuestre la estabilidad de los resultados.

- **Tamaño muestral y modelado estadístico:** los indicadores clave de rendimiento (KPI) recolectados deben procesarse mediante herramientas de inferencia estadística. Para que las puntuaciones y clasificaciones (*scoring and ranking*) sean técnicamente consistentes, el tamaño de la muestra debe ser suficiente para aplicar los modelos de distribución de probabilidad correspondientes; en estadística paramétrica suele adoptarse $N \geq 30$ para acotar el margen de error [47].
- **Criterio de depuración de datos (regla del 5 %):** la norma prevé la aplicación de procedimientos de limpieza de datos (*data cleansing*). Si durante la ejecución de un escenario más del **5 % de las muestras** son erróneas o se pierden por fallas del software de medición o del entorno de prueba, y no por el rendimiento del sistema analizado, el conjunto de datos se invalida y la corrida se repite [47].

2.4.3. Umbrales de calidad de servicio (QoS) y calidad de experiencia (QoE) para video en streaming

Para determinar objetivamente el impacto de una inundación de paquetes TCP sobre un servidor de video, es necesario establecer los umbrales de tolerancia de la red y de la percepción del usuario final. Con este propósito, la arquitectura de pruebas adopta los límites de las Recomendaciones ITU-T Y.1541 (parámetros de rendimiento de red) e ITU-T G.1010 (requisitos de las aplicaciones) [48], [49].

2.4.3.1. Parámetros de rendimiento de red según la Recomendación ITU-T Y.1541

La Recomendación **ITU-T Y.1541** define las clases de calidad de servicio (QoS) aplicables a las redes IP y fija límites para el retardo, su variación y la pérdida de paquetes [48].

- **Retardo de transferencia de paquetes IP (IPTD):** para las clases 0 y 1 (servicios en tiempo real e interactivos), el retardo medio de transferencia en un solo sentido no debe superar 100 ms y 400 ms, respectivamente. Dado que en este trabajo se mide el RTT (ida y vuelta), el umbral de 100 ms se adopta como criterio conservador, equivalente a exigir del orden de 50 ms por sentido en un trayecto simétrico, antes de que la degradación del servicio impida la comunicación bidireccional o vacíe los búferes de precarga.
- **Variación del retardo de paquetes IP (*IPDV o jitter*):** no debe superar 50 ms para las clases 0 y 1, medida como la diferencia entre el cuantil superior ($1 - 10^{-3}$) del retardo y el retardo mínimo, y no como un valor medio. Un *jitter* superior a este umbral desincroniza el flujo multimedia y obliga al reproductor a pausar la reproducción para volver a llenar el búfer (*rebuffering*).
- **Tasa de pérdida de paquetes IP (IPLR):** para estas clases, la pérdida de paquetes debe ser inferior a 1×10^{-3} (0.1 %). Ante un ataque TCP, una pérdida superior a este umbral activa los mecanismos de control de congestión de TCP, que provocan retransmisiones y reducen el rendimiento global.

2.4.3.2. Requisitos de calidad de experiencia (QoE) según la Recomendación ITU-T G.1010

Mientras que la Y.1541 define la tolerancia de la infraestructura, la Recomendación ITU-T G.1010 establece los requisitos de rendimiento de las aplicaciones desde la perspectiva del usuario final (QoE) y clasifica los servicios según su sensibilidad al retardo y a las pérdidas [49]:

- **Rendimiento útil (*throughput*):** los flujos de video se clasifican según la capacidad que requiere la aplicación. En este trabajo, se considera que el canal debe ofrecer un *throughput*

continuo y estable, al *menos igual a la tasa de bits* de codificación del video, para evitar la congelación de la imagen.

- **Tolerancia en streaming de video:** la ITU-T G.1010 clasifica el video en streaming como una aplicación tolerante a errores, con una pérdida de paquetes admisible de hasta el 1 %, y exige que el retardo de inicio de la reproducción no supere los 10 s para evitar el abandono del usuario [49].

2.4.1. Herramientas de medición de QoS

La medición de las métricas definidas en la Tabla 4 requiere herramientas que operen de forma no intrusiva sobre el tráfico del servicio y que sean compatibles con la topología virtualizada en GNS3. Entre las alternativas consideradas, Wireshark constituye la opción principal por su integración nativa con GNS3, que permite iniciar capturas de tráfico desde cualquier enlace de la topología mediante un menú contextual, sin configuración adicional en los nodos [50].

Wireshark es un analizador de protocolos de red de código abierto que permite capturar y analizar en detalle las tramas, en tiempo real o de forma diferida a partir de archivos PCAP [51]. Implementa disectores para más de dos mil protocolos, entre ellos RTSP, RTP, OpenFlow 1.0 y 1.3, ICMP, TCP y UDP, lo que permite analizar en profundidad todos los flujos relevantes del escenario. En GNS3, Wireshark se invoca con un clic derecho sobre cualquier enlace virtual y comienza una captura en tiempo real sobre la interfaz correspondiente; esta característica permite observar simultáneamente el tráfico OpenFlow entre el controlador y los Open vSwitch y los flujos entre el servidor MediaMTX y los clientes. En esta investigación se aprovechan el cálculo automático del RTT mediante el análisis de pares segmento-ACK, la generación de gráficas de throughput (*Statistics → I/O Graphs*) y el filtrado por protocolo para aislar el tráfico de ataque del tráfico legítimo [51].

Como herramientas complementarias se incorporan ntopng y las métricas que expone de forma nativa el servidor MediaMTX: ntopng aporta el análisis pasivo del tráfico, la clasificación de protocolos en tiempo real y la observabilidad por flujo, y MediaMTX informa del estado de sus sesiones [52]. La Tabla 3 detalla el rol de cada herramienta en el plan de medición.

Tabla 3

Herramientas seleccionadas para la medición de métricas de QoS

Herramienta	Métrica asociada	Modo de operación	Justificación
Wireshark/ tshark	Latencia, jitter, pérdida, throughput	Captura pasiva integrada en GNS3	Integración nativa en GNS3; análisis profundo de RTSP y OpenFlow [Wireshark / GNS3].
ntopng	Throughput, jitter, pérdida y sesiones por flujo	Sonda pasiva con inspección profunda de paquetes (DPI)	Análisis pasivo de tráfico con DPI y métricas por flujo en tiempo real; ampliamente adoptado en monitoreo empresarial
Métricas MediaMTX	Sesiones RTSP, Throughput del servidor	Endpoint Prometheus	Observabilidad nativa del servidor de streaming

Las métricas de QoS reportadas en el Capítulo IV se obtienen con tshark a partir de las capturas; ntopng y MediaMTX se emplean como fuentes de verificación y visualización del estado del tráfico y del servidor.

2.4.1. Métricas de QoS para evaluar el impacto del ataque DDoS

Para medir cómo afectan los ataques DDoS al servicio de streaming, se definen cuatro métricas objetivas que relacionan la degradación de la calidad percibida por el usuario con la intensidad del ataque que recibe el servidor de streaming. Los umbrales de aceptación se toman de la Recomendación ITU-T Y.1541, que establece los objetivos de rendimiento de red para servicios IP, y de las especificaciones de la IETF aplicables al tráfico multimedia en tiempo real [53]. Para asegurar la robustez de las comparaciones, cada métrica se obtiene como el promedio de las réplicas de cada escenario lo que permite valorar si las diferencias entre OpenDaylight y Ryu son consistentes y no se deben a la variabilidad entre corridas.

Latencia (RTT): tiempo de ida y vuelta entre el cliente y el servidor de streaming. En el escenario planteado, la latencia puede aumentar si el controlador SDN no procesa oportunamente los mensajes *PACKET_IN* generados por los Open vSwitch, lo que retrasa la instalación de las reglas de flujo del tráfico del servicio [41].

La latencia se cuantifica mediante la ecuación (1):

$$\mathrm{RTT} = \mathrm{T}_{\mathrm{recepción}} - \mathrm{T}_{\mathrm{envío}} \tag{1}$$

donde $T_{envío}$ corresponde al instante en que el cliente emite la solicitud y $T_{recepción,}$ al instante en que recibe la respuesta. Esta expresión coincide con la definición de retardo de ida y vuelta de la RFC 2681 [54], y permite estimar el efecto acumulado del procesamiento del controlador SDN y de la conmutación en los Open vSwitch durante la sesión de streaming.

Jitter: variación temporal entre paquetes consecutivos de un mismo flujo. Resulta especialmente crítico en los flujos RTP, ya que un jitter elevado obliga al búfer de reproducción del cliente a aumentar su tamaño o, en su defecto, provoca interrupciones perceptibles en la reproducción del contenido multimedia [53].

De acuerdo con el modelo de jitter entre llegadas definido por la IETF en la RFC 3550 para flujos RTP [55], la variación temporal se actualiza de forma recursiva mediante la ecuación (2):

$$J(i) = J(i-1) + (|D(i-1,i)| - J(i-1))/16 \quad (2)$$

donde $D(i,j) = (R_j - R_i) - (S_j - S_i)$; R representa la marca de tiempo de llegada y S, la marca de tiempo de envío de cada paquete. El filtro de primer orden otorga mayor peso a las muestras recientes, lo que resulta apropiado para flujos multimedia en tiempo real [56].

Pérdida de paquetes: porcentaje de paquetes no entregados respecto al total transmitido. Cuando un conmutador SDN pierde la comunicación con su controlador, los Open vSwitch configurados en modo *fail-secure* descartan las tramas que no tienen un flujo coincidente, lo que genera pérdidas significativas en la sesión de streaming [24].

El porcentaje de pérdida se calcula con la ecuación (3):

$$PL(\%) = (N_{perdidos} / N_{enviados}) \times 100 \quad (3)$$

donde $N_{enviados}$ es el número total de paquetes enviados por el emisor en el intervalo de medición y $N_{perdidos}$, el número de paquetes que no llegan al receptor antes del vencimiento de un temporizador. Esta definición deriva de la métrica One-Way Packet Loss de la RFC 7680 [56], y equivale al IPLR de la Recomendación ITU-T Y.1541.

Throughput: tasa útil de transferencia entre el servidor y el cliente. Su degradación evidencia la competencia entre el tráfico legítimo y el tráfico malicioso dirigido al servidor de streaming, que consume su capacidad de conexión e impide mantener una entrega estable del flujo hacia el cliente [14]. El throughput se calcula con la ecuación (4):

$$Throughput = B_{recibidos} / \Delta t \quad (4)$$

donde $B_{recibidos}$ corresponde al número total de bits entregados correctamente al cliente y Δt, al intervalo de observación. La RFC 5136 [57],formaliza este indicador como una medida de

la capacidad efectiva de extremo a extremo, que permite diferenciar el rendimiento percibido por el usuario de la capacidad nominal del enlace en escenarios de saturación del servidor.

La Tabla 4 presenta las métricas seleccionadas, sus umbrales de referencia según la ITU-T Y.1541, la herramienta de medición asociada y el punto de captura en la topología SDN implementada en GNS3.

Tabla 4

Métricas de QoS, umbrales de referencia e instrumento de medición

Métrica	Unidad	Umbral (Y.1541)	Herramienta	Punto de captura
Latencia (RTT)	ms	≤ 100 ms (clase 0)	Wireshark	Enlace OVS – Router de Borde
Jitter	ms	≤ 50 ms (clase 0)	Wireshark, ntopng	
Pérdida de paquetes	%	≤ 1×10^{-3} (clase 0)	Wireshark, ntopng	
Throughput	Mbps	≥ tasa nominal del stream	Wireshark, ntopng	

Nota. Adaptado de la Recomendación ITU-T Y.1541 [48].

3. CAPÍTULO III: METODOLOGÍA

Este capítulo presenta la metodología empleada para reproducir, en un entorno virtualizado y reproducible, las condiciones en las que un ataque DDoS afecta a una red definida por software que transporta tráfico de streaming multimedia. Se adoptan las fases de la simulación discreta, implementadas sobre un entorno de emulación (GNS3 ejecuta el software real de controladores, conmutadores y servidores), porque permiten manipular de forma controlada las variables relevantes número de fuentes de ataque, tipo de tráfico generado, intensidad y duración, sin las restricciones de un despliegue físico. El capítulo se estructura en tres bloques. El primero analiza los requerimientos del modelo según el estándar ISO/IEC/IEEE 29148 e identifica a las partes interesadas. El segundo define las métricas de QoS y selecciona las herramientas para ejecutar el ataque y medir su impacto. El tercero detalla la implementación del entorno SDN con GNS3, Docker y los controladores OpenDaylight y Ryu, junto con el plan de experimentación y el procesamiento de datos; los resultados y su discusión se presentan en el Capítulo IV. El propósito es ofrecer una plataforma que evidencie cómo se degradan la latencia, el jitter, la pérdida de paquetes y el throughput cuando el servidor de streaming alojado en la red SDN es objeto de un ataque de agotamiento de conexiones, con el nivel de detalle suficiente para que la investigación pueda replicarse en otras instituciones. A diferencia de buena parte de la literatura sobre SDN, centrada en la detección o la mitigación de los ataques DDoS, este trabajo se concentra en cuantificar su impacto sobre la calidad del servicio de streaming, de modo que los resultados sirvan como línea base para futuras estrategias de defensa.

3.1. Análisis de los requerimientos del modelo

Con base en los fundamentos teóricos del Capítulo II, se adopta la metodología de simulación discreta para abordar cada elemento del modelo presentado en la Figura 1. Esta metodología guía la construcción de un escenario que permite analizar el impacto de los ataques

DDoS sobre el servicio de streaming multimedia en entornos SDN y, a partir de ella, se seleccionan y detallan los componentes del modelo: los controladores, el tipo de ataque y las características y versiones de las herramientas de software y hardware.

La fase de análisis identifica los elementos de software y hardware que forman parte del diseño y de la emulación. Los requisitos del proyecto se formulan con base en el estándar ISO/IEC/IEEE 29148:2011 [58]; la Tabla 5 resume la terminología y los tipos de requerimiento que se derivan de él.

Tabla 5

Abreviaturas de los requerimientos definidos en el estándar ISO/IEC/IEEE 29148:2011

Abreviatura	Descripción
StRS	Especificación de requerimientos de las partes interesadas (Stakeholders)
SyRS	Especificación de requerimientos del sistema
SRS	Especificación de requerimientos de software
SyRA	Especificación de requerimientos de arquitectura

Nota. Tomado de [58].

3.1.1. Partes interesadas (stakeholders)

Las partes interesadas (stakeholders) son las personas u organizaciones con interés directo o participación en las decisiones del proyecto. Su identificación es relevante porque condiciona la definición de los requerimientos del sistema y las prioridades del diseño. La Tabla 6 clasifica las partes interesadas de este trabajo.

Tabla 6

Partes interesadas (stakeholders)

N.º	Partes interesadas	Descripción
StRS1	Ing. Fabian Cuzme, MSC	Director del trabajo de titulación
StRS2	Ing. Mauricio Domínguez, MSC.	Asesor
StRS3	Diego Achina	Desarrollador del trabajo de titulación

3.1.2. Requerimientos del sistema

En el diseño de redes, los requerimientos del sistema constituyen un componente esencial, ya que expresan la función de cada elemento del diseño propuesto, según el estándar ISO/IEC/IEEE 29148 [58]. La Tabla 7 presenta los requerimientos del sistema del presente proyecto, con la abreviatura correspondiente a cada uno.

Tabla 7

Requerimientos del sistema

SyRS	Requerimientos del Sistema	Prioridad		
N°	Requerimiento	Alta	Media	Baja
SyRS-1	Permitir la configuración y despliegue de controladores SDN utilizando protocolos de comunicación SDN	X		
SyRS-2	Proporcionar capacidad para generar y controlar ataques DDoS simulados		X	

SyRS	Requerimientos del Sistema	Prioridad	
SyRS-3	Medir y registrar en tiempo real las métricas de QoS (latencia, pérdida de paquetes, jitter y ancho de banda)	X	
SyRS-4	Permitir la generación de tráfico de red de fondo con patrones configurables para simular condiciones reales		X
SyRS-5	Proporcionar capacidades de correlación entre patrones de ataque DDoS y degradación de métricas de QoS		X
SyRS-6	Mantener un tiempo de actividad (uptime) del 99.9 % durante la ejecución de los experimentos	X	
SyRS-7	Garantizar el aislamiento completo del entorno de pruebas respecto a redes externas		X
SyRS-8	Generar reportes detallados de análisis correlacionando ataques DDoS y métricas de QoS	X	
SyRS-9	Asegurar compatibilidad con las capacidades de hardware disponibles para el desarrollo del proyecto	X	

3.1.3. Requerimientos de la arquitectura

En esta sección se describen las propiedades generales que deben cumplir el hardware y el software para que el esquema propuesto funcione de forma efectiva. Además, se establecen los requerimientos de la arquitectura, que definen la estructura y la organización de los componentes del sistema para asegurar su escalabilidad, seguridad e interoperabilidad. Estos requisitos abarcan

la distribución del hardware, la compatibilidad del software, los protocolos de comunicación y los estándares necesarios para el rendimiento del modelo emulado.

La Tabla 8 presenta los requerimientos de la arquitectura del modelo de simulación.

Tabla 8

Requerimientos de la arquitectura de simulación

Requerimientos de la Arquitectura (SyRA)				
N°	Requerimiento	Prioridad		
		Alta	Media	Baja
	Requerimientos del equipo host			
SyRA1	PC/Portátil host: CPU Intel/AMD con soporte para virtualización (VT-x/AMD-V)	X		
SyRA2	Memoria RAM mínima: 16GB para host	X		
SyRA3	Almacenamiento: 100GB libres mínimo SSD	X		
SyRA4	Sistema Operativo: Windows 10/11 o Linux		X	
SyRA5	Tarjeta de red: 1 Gbps para conectividad externa	X		
	Requerimientos de software de simulación			
SyRA6	Soporte SDN/OpenFlow	X		

Requerimientos de la Arquitectura (SyRA)			
SyRA7	Capacidad de Virtualización	X	
SyRA8	Integración de herramientas de Monitoreo	X	
SyRA9	Optimo consumo de recursos de hardware	X	
SyRA10	Facilidad de Uso	X	
SyRA11	Documentación y soporte	X	
Requerimientos Servidor Streaming			
SyRA12	Facilidad configuración	X	
SyRA13	Control parámetros de despliegue del servicio	X	
SyRA14	Documentación		X
SyRA15	Consumo de Recursos		X
SyRA16	Uso académico		X
Requerimientos para el desarrollo de los ataques DDoS			
SyRA17	Soporte de múltiples vectores de ataque	X	
SyRA18	Control y configuración parametrizable	X	
SyRA19	Compatibilidad con diversos entornos	X	
SyRA20	Orquestación distribuida y de monitoreo integrado del servicio		X
SyRA21	Documentación disponible	X	

Requerimientos de la Arquitectura (SyRA)			
Requerimientos de controlador SDN			
SyRA22	Soporte del protocolo OpenFlow	X	
SyRA23	Licencia de código abierto	X	
SyRA24	Multiplataforma	X	
SyRA25	Exposición de API REST	X	
SyRA26	Interfaz web de administración		X
SyRA27	Documentación disponible	X	
SyRA28	Arquitectura multi-hilo	X	
Requerimientos monitoreo de tráfico			
SyRA29	Monitoreo en tiempo real con UI web		X
SyRA30	Soporte nativo sFlow / NetFlow	X	
SyRA31	Análisis L7 y detección de DDoS	X	
SyRA32	Captura y exportación pcap	X	
SyRA33	Bajo consumo de recursos en VM	X	
SyRA34	Alertas automáticas de anomalías		X
SyRA35	Licencia libre y documentación	X	

3.1.4. Evaluación de herramientas de simulación

Los requerimientos de software especifican los programas, sistemas operativos, herramientas y aplicaciones necesarios para el funcionamiento de la simulación, de modo que se garantice su compatibilidad, rendimiento y eficiencia. Entre las herramientas de simulación

evaluadas, GNS3 es la que cumple el mayor número de requerimientos del sistema y, además, permite escalar el escenario, como muestra la Tabla 9.

Tabla 9

Requerimientos para la elección de la herramienta de simulación

Requerimiento del sistema (SyRS)	GNS3	MININET	EVE-NG	CORE
SyRS1	1	1	0	0
SyRS2	1	1	1	1
SyRS3	1	0	1	0
SyRS4	1	1	1	1
SyRS5	1	0	1	0
SyRS6	1	1	1	1
SyRS7	1	1	1	1
SyRS8	1	0	1	0
SyRS9	1	1	1	1
SySA6	1	1	1	1
SySA7	1	1	1	1
SySA8	1	1	0	0
SySA9	1	1	0	1
SySA10	0	0	0	0
SySA11	1	1	0	0
Total, de evaluación	14/15	11/15	10/15	8/15

3.1.5. Evaluación y selección del controlador

La elección del controlador SDN parte de cuatro alternativas habituales en los ámbitos académico y profesional: Ryu, OpenDaylight, ONOS y NOX. Cada una se contrasta con los requerimientos del sistema de la Tabla 8 mediante una escala binaria de verificación, en la que 1 indica cumplimiento y 0, incumplimiento. El procedimiento proporciona un puntaje agregado por controlador y muestra con claridad los requerimientos que cada alternativa no cubre. La Tabla 10 presenta el resultado de esta evaluación.

Tabla 10

Cumplimiento de requerimientos por controlador SDN

ID	Requerimiento	Ryu	OpenDaylight	NOX
SySA22	Soporte de protocolo OpenFlow	1	1	1
SySA23	Codigo abierto	1	1	1
SySA24	Multiplataforma	0	1	0
SySA25	API REST	1	1	0
SySA26	Interfaz Web	0	1	0
SySA27	Documentacion	1	1	0
SySA28	Multi-thread	1	1	0
TOTAL, cumplido		5/7	7/7	2/7

Los resultados muestran que OpenDaylight es el controlador con mayor cumplimiento, con un puntaje de 7/7. OpenDaylight soporta múltiples versiones de OpenFlow y ofrece la interfaz gráfica web DLUX y una API REST completa, lo que lo posiciona como la opción más adecuada [17].

Ryu obtiene un puntaje de 5/7; es una alternativa viable para entornos académicos y de investigación por su simplicidad y facilidad de uso con Python, aunque está limitada a Linux. NOX es el menos adecuado para el presente proyecto, ya que cumple únicamente 2 de los 7 requerimientos evaluados.

Se selecciona OpenDaylight como el controlador SDN para el presente proyecto, dado que cumple la totalidad de los requerimientos definidos, y se incorpora Ryu como segundo controlador de comparación, por representar una arquitectura ligera y basada en Python frente a la plataforma modular en Java de OpenDaylight.

La elección de OpenDaylight no responde únicamente al cumplimiento numérico de los requerimientos, sino también a un factor práctico que se evidencia en las pruebas preliminares: la madurez de su ecosistema y la disponibilidad de módulos como L2Switch, OpenFlow Plugin y la interfaz DLUX facilitan su aprendizaje en escenarios de laboratorio. Ryu, aunque flexible como framework en Python, no dispone de una consola gráfica ni de una API REST tan completa como la de OpenDaylight para inspeccionar el estado del plano de control durante la ejecución. Por estas razones, el criterio final combina el cumplimiento técnico, la simplicidad operativa y la experiencia documentada en proyectos académicos similares [19].

Si bien la evaluación numérica selecciona OpenDaylight como controlador principal, en este trabajo se instalan y prueban dos controladores, OpenDaylight y Ryu, sobre el mismo escenario base. La topología, el direccionamiento, el servidor de streaming y el bloque generador de ataques DDoS no se modifican entre pruebas; solo se intercambia el controlador SDN. Este diseño experimental permite comparar de manera controlada el comportamiento de ambos controladores frente al ataque, de modo que las diferencias en las métricas de QoS puedan atribuirse principalmente al controlador y no a cambios en el entorno.

3.1.6. Justificación académica del uso de Docker

El estudio de Felter et al. [59], muestra que los contenedores Docker alcanzan un rendimiento muy cercano al nativo en CPU y red, mientras que las máquinas virtuales basadas en KVM agregan entre un 5 % y un 10 % de sobrecarga en cargas similares. En este experimento, en el que se requiere generar volúmenes significativos de tráfico para saturar el servidor de streaming con decenas de hilos concurrentes (hasta 80 en el escenario de máxima intensidad), minimizar la sobrecarga de la capa de virtualización es un requisito para que los resultados sean representativos del comportamiento del sistema [59].

La Tabla 11 compara las características esenciales de Docker y de las máquinas virtuales para este entorno de pruebas; el resultado indica que Docker ofrece mayor capacidad de adaptación con un menor consumo de recursos.

Tabla 11

Comparación entre Docker y máquinas virtuales para el entorno de laboratorio

Característica	Docker (contenedor)	Máquina Virtual (KVM/VirtualBox)
Tiempo de arranque	< 2 segundos	30 – 120 segundos
Overhead de memoria	< 10 MB (runtime)	256 – 512 MB mínimo por VM
Mecanismo de aislamiento	Namespaces + cgroups Linux	Hipervisor tipo 1 ó 2
Portabilidad	Alta — imagen reproducible mediante Dockerfile	Limitada — snapshots pesados (GB)

Característica	Docker (contenedor)	Máquina Virtual (KVM/VirtualBox)
Integración nativa GNS3	Sí — plantilla Docker	Sí — appliance QEMU/VirtualBox
Acceso a red del host	--network host directo	Adaptador de red virtual (NAT/Bridge)
Reproducibilidad científica	Alta — Dockerfile determinista	Media — dependiente del snapshot
Tamaño de imagen	~200 – 500 MB	2 – 20 GB (disco virtual)

3.1.7. Evaluación y selección de la herramienta de generación de ataques DDoS

La herramienta de generación de ataques DDoS se selecciona evaluando las alternativas frente a los requerimientos SyRA17 a SyRA21 de la Tabla 8. Las herramientas consideradas son hping3, ampliamente utilizada en investigaciones de seguridad por su capacidad de generar paquetes TCP, UDP e ICMP a bajo nivel [60]; LOIC (Low Orbit Ion Cannon), orientada a ataques de capa de aplicación [61]; Scapy, biblioteca de Python para fabricar y enviar paquetes arbitrarios [62]; y Slowloris, enfocada en el agotamiento de conexiones HTTP [63]. Ninguna de estas herramientas integra, en una sola plataforma, la orquestación distribuida de varios nodos atacantes desde un servidor de mando y control, la sincronización del ataque y el registro de métricas del servicio durante el ataque.

La Tabla 12 resume la evaluación comparativa con la misma escala binaria de la Tabla 9, en la que 1 indica cumplimiento del requerimiento y 0, incumplimiento.

Tabla 12

Evaluación de herramientas para la generación de ataques DDoS

Req.	Descripción	hping3	LOIC	Scapy	Slowloris	TrafficGen SDN
SyRA17	Varios tipos de ataque	1	0	1	0	1
SyRA18	Control y configuración	1	1	1	1	1
SyRA19	Integración Docker/GNS3	1	0	1	0	1
SyRA20	Orquestación distribuida y de monitoreo integrado del servicio	0	0	0	0	1
SyRA21	Documentación	1	1	1	1	1
Total		4/5	2/5	4/5	2/5	5/5

El análisis comparativo muestra que las herramientas estándar cubren como máximo 4 de los 5 requerimientos, principalmente por la ausencia de orquestación distribuida y de monitoreo integrado del servicio. Por esta razón, se desarrolla la herramienta propia TrafficGen SDN, basada en Python y Flask, que implementa el ataque tcp_exhaust para agotar las conexiones TCP del servicio HLS (8888/TCP) del servidor de streaming MediaMTX y producir un impacto

cuantificable y reproducible sobre la calidad del servicio. El detalle funcional de la herramienta se expone en la sección Diseño del sistema TrafficGen SDN.

3.1.8. Evaluación y selección de la herramienta de monitoreo de tráfico

La evaluación cuantitativa del impacto del ataque DDoS sobre la calidad del servicio requiere una herramienta de monitoreo capaz de capturar, agregar y visualizar métricas de tráfico en tiempo real y de mantener un histórico consultable para el análisis posterior. En esta sección se contrasta ntopng con otras alternativas habituales en el ámbito académico, según los requerimientos del sistema de monitoreo recogidos en el Anexo A.

La Tabla 13 presenta la matriz comparativa entre las herramientas evaluadas y los requerimientos del sistema de monitoreo de la Tabla 8, los requerimientos que se usan son los SyRA29 hasta el SyRA35.

Tabla 13

Matriz comparativa de herramientas de monitoreo de tráfico SDN frente a los requerimientos del sistema

Requerimiento del sistema	ntopng	Wireshark	Prometheus + Grafana	sFlow-RT
SyRA29	1	1	1	1
SyRA30	1	1	1	1
SyRA31	1	0	0	0
SyRA32	1	1	0	0
SyRA33	1	1	1	1
SyRA34	1	1	1	0

Requerimiento del sistema	ntopng	Wireshark	Prometheus + Grafana	sFlow-RT
SyRA35	1	1	1	1
Total,	7	6	5	4

A partir de la matriz comparativa, ntopng satisface todos los requerimientos definidos. Reúne en una sola plataforma la captura, la agregación y la visualización del tráfico, ofrece soporte nativo de sFlow para integrarse con los Open vSwitch del plano de datos y dispone de módulos de detección de anomalías orientados a DDoS [52]. Wireshark, aunque resulta valioso para el análisis diferido a nivel de paquete, carece de interfaz web colaborativa y de exportadores de métricas. Prometheus con Grafana ofrece históricos sólidos, pero requiere desarrollar exportadores específicos para SDN, lo que aumenta el costo de implementación. sFlow-RT ofrece analítica en tiempo real, pero su licencia comunitaria restringe el número de flujos y su documentación pública es limitada [64].

Ntopng se selecciona como herramienta de monitoreo de tráfico del experimento. Se ubica en la red de gestión (10.0.0.0/24), desde donde recibe el tráfico replicado por los Open vSwitch mediante una sesión de port mirroring. Esta separación entre el plano de monitoreo y el plano de datos evita interferencias durante la ejecución de los escenarios de ataque.

3.1.9. Evaluación y selección del servidor de streaming

El servidor de streaming es el componente que publica los flujos consumidos por los clientes legítimos y, a la vez, el blanco del ataque DDoS. Su elección se basa en la comparación de cinco alternativas de uso extendido —MediaMTX, Wowza Streaming Engine, Nginx con módulo RTMP, GStreamer (gst-rtsp-server) y Ant Media Server frente a los requerimientos

SyRA12 a SyRA16 de la Tabla 8. La Tabla 14 resume esta evaluación con la misma escala binaria de las tablas anteriores, en la que 1 indica que la herramienta satisface el requerimiento y 0, que no lo satisface.

Tabla 14

Evaluación de servidores de streaming frente a los requerimientos SyRA12–SyRA16

Req.	Descripción	MediaMTX	Wowza SE	Nginx-RTMP	GStreamer	Ant Media
SyRA12	Facilidad de configuración	1	1	0	0	1
SyRA13	Control de parámetros de despliegue	1	1	0	1	1
SyRA14	Documentación	1	1	1	1	1
SyRA15	Consumo de recursos	1	0	1	1	0
SyRA16	Uso académico / licencia abierta	1	0	1	1	1
Total		5/5	3/5	3/5	4/5	4/5

Los resultados muestran que MediaMTX es la única alternativa que satisface los cinco requerimientos. Se configura mediante un archivo YAML y variables de entorno, lo que facilita su despliegue en contenedores Docker y otorga control directo sobre puertos, protocolos y rutas de

publicación (SyRA12 y SyRA13). Su consumo de recursos es mínimo frente al de soluciones basadas en Java, como Wowza y Ant Media, más pesadas para un laboratorio virtualizado (SyRA15). A diferencia de Nginx-RTMP, orientado a RTMP y sin soporte nativo de RTSP, y de GStreamer, que exige construir manualmente las tuberías de medios, MediaMTX ofrece RTSP listo para usar y documentación actualizada (SyRA14). Su licencia de código abierto (MIT) y su adopción en trabajos académicos cumplen, además, el criterio de uso académico (SyRA16).

3.2. Diseño del sistema

El diseño del sistema establece cómo se organizan los componentes de hardware y software del escenario para que operen de forma estable, eficiente y reproducible. Comprende tres niveles complementarios: el diseño conceptual, que fija la arquitectura general y las funciones de cada bloque; el diseño lógico, que describe los procesos, el direccionamiento y los flujos de datos; y el diseño físico, que concreta la implementación sobre el entorno virtualizado. En todos ellos se atiende a la interoperabilidad entre los planos SDN, al consumo de recursos del equipo anfitrión y al rendimiento necesario para sostener el tráfico de streaming durante los ataques.

3.2.1. Topología de red

La topología descrita constituye el escenario base único de todos los experimentos. Los dispositivos, el direccionamiento IPv4, los equipos cliente, el servidor de streaming MediaMTX y el bloque generador de ataques DDoS permanecen invariables en cada prueba; el único elemento que cambia entre los casos de estudio es el controlador SDN (OpenDaylight o Ryu). De este modo, las diferencias observadas en las métricas de QoS pueden atribuirse principalmente al comportamiento del controlador frente al ataque y no a variaciones del entorno.

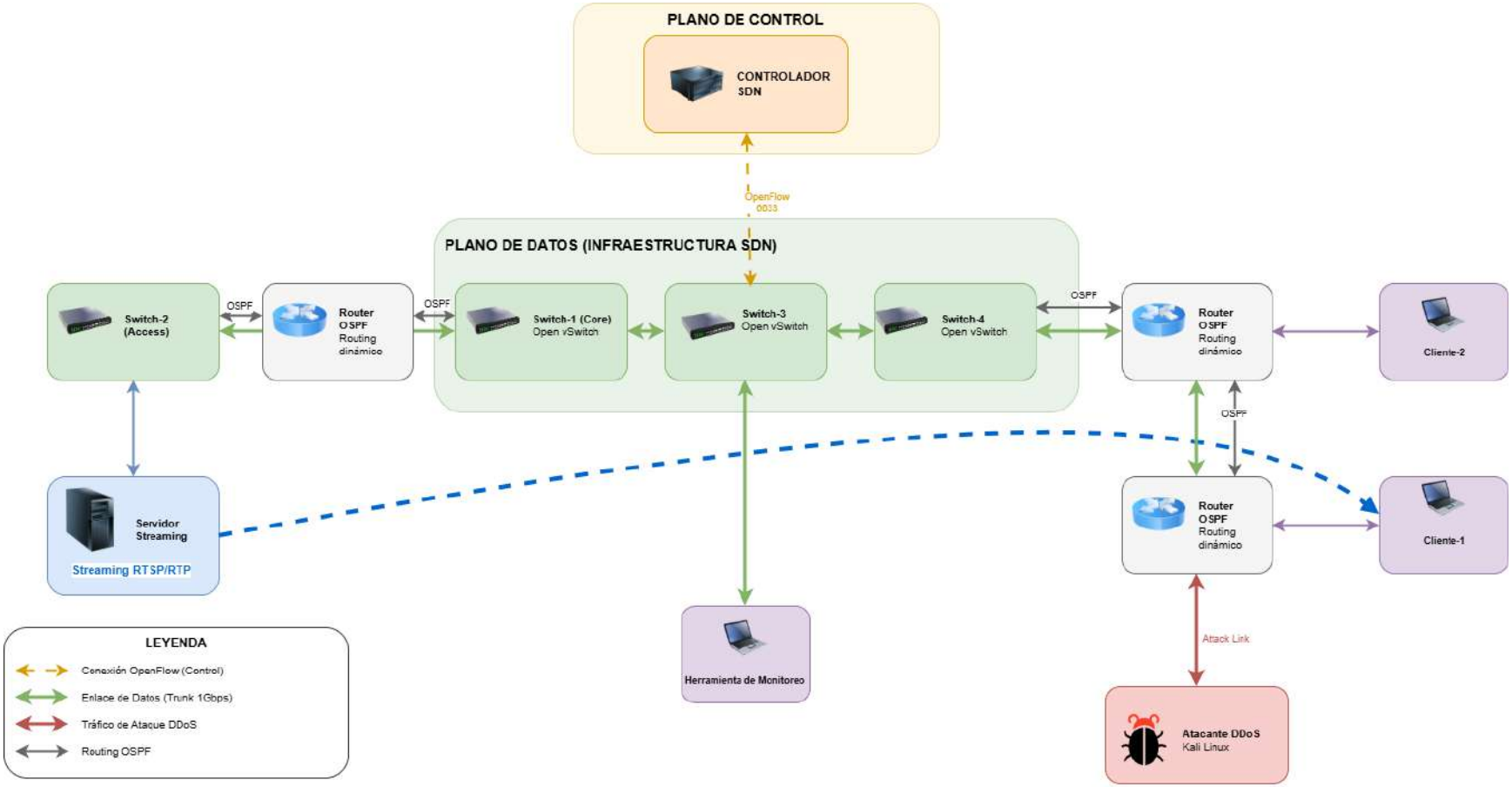
La topología de red es el escenario sobre el que se implementa y analiza el modelo. En ella confluyen todos los componentes de la red definida por software: el controlador, que gobierna el

plano de control; los conmutadores, que forman el plano de datos; el servidor de streaming, que provee el servicio bajo estudio; los clientes, que lo consumen; y los nodos atacantes, que reproducen las amenazas DDoS. Esta disposición permite evaluar el comportamiento de la red en distintas condiciones de carga y ataque de forma controlada y repetible, y medir las métricas de calidad de servicio (QoS).

La Figura 11 presenta la distribución de la red SDN, con el plano de control y el plano de datos conectados según su función; el plano de datos concentra los enlaces hacia los enrutadores, los clientes y el servidor. Para facilitar el seguimiento de las trayectorias del servicio y del ataque, el servidor de streaming se dibuja en el borde de servicio, junto al bloque de clientes y al bloque atacante, aunque su direccionamiento corresponde a la DMZ definida a continuación.

Figura 11

Topología de red SDN



La topología se organiza en cinco bloques funcionales, cada uno con un papel específico en el escenario de evaluación. A continuación, se describen el propósito y los elementos de cada bloque.

3.2.1.1. Bloque de usuarios

El bloque de usuarios representa el tráfico legítimo de la red y está conformado por los clientes que consumen el servicio de streaming multimedia desde sus equipos finales. Estos clientes solicitan el flujo al servidor de streaming por HLS y constituyen el grupo cuya calidad de servicio (QoS) se mide en los escenarios con y sin ataque DDoS. Se ubican en las LAN de la topología (subredes derivadas del bloque 192.168.10.0/24) y su tráfico atraviesa el plano de datos administrado por los Open vSwitch.

3.2.1.2. Bloque SDN

El bloque SDN agrupa el plano de control y el plano de datos de la arquitectura definida por software. En el plano de control reside el controlador (OpenDaylight o Ryu, según el escenario), encargado del cómputo de rutas, la gestión de la base de información de red (NIB, Network Information Base) y la programación de reglas de reenvío mediante el protocolo OpenFlow 1.3 [23]. En el plano de datos se ubican los Open vSwitch, que aplican las reglas instaladas por el controlador. La red de gestión 10.0.0.0/24 conecta el controlador con las interfaces administrativas (mgmt0) de los conmutadores.

3.2.1.3. Bloque servidor de streaming

El bloque del servidor está conformado por el servidor de streaming multimedia MediaMTX, que publica los flujos consumidos por el bloque de usuarios. Se ubica lógicamente en la DMZ de la topología y actúa como víctima de los ataques DDoS, lo que permite registrar la degradación de las métricas de QoS (latencia, jitter, pérdida de paquetes y throughput) a lo largo de los experimentos.

3.2.1.4. Bloque de ataques DDoS

El bloque de ataques DDoS contiene la infraestructura que genera el tráfico malicioso destinado a saturar el servidor de streaming. Está integrado por el sistema TrafficGen SDN, desplegado mediante contenedores Docker: un nodo dashboard, ubicado en la máquina virtual Kali Linux, y hasta cuatro nodos workers (bots). Un *worker es un nodo de la botnet controlado de forma remota por el atacante mediante el servidor de mando y control (C2), cuya función es generar tráfico malicioso tcp_exhaust hacia el servidor de streaming. Cada worker ejecuta 20 hilos concurrentes de conexión TCP, de modo que el volumen del ataque escala en proporción al número de nodos activos*, desplegados en la subred de clientes infectados de la LAN 1 (workers 2 a 4) y en el propio nodo Kali Linux de la LAN 2 (worker 1). En conjunto, estos nodos reproducen una botnet a escala de laboratorio capaz de generar un volumen sostenido de conexiones TCP contra el servicio HLS (8888/TCP) del servidor de streaming.

3.2.1.5. Bloque de monitoreo

El bloque de monitoreo registra de forma continua el tráfico que atraviesa el plano de datos durante los experimentos. Está compuesto por una máquina virtual dedicada que ejecuta ntopng como agente de análisis y visualización, ubicada en la red de gestión 10.0.0.0/24 con la dirección IP 10.0.0.50. La máquina virtual recibe el tráfico replicado desde el Open vSwitch central mediante una sesión de port mirroring configurada en la interfaz eth4. Esta separación entre el plano de monitoreo y el plano de datos evita interferencias en la medición y permite recopilar métricas de QoS sin afectar el rendimiento del experimento.

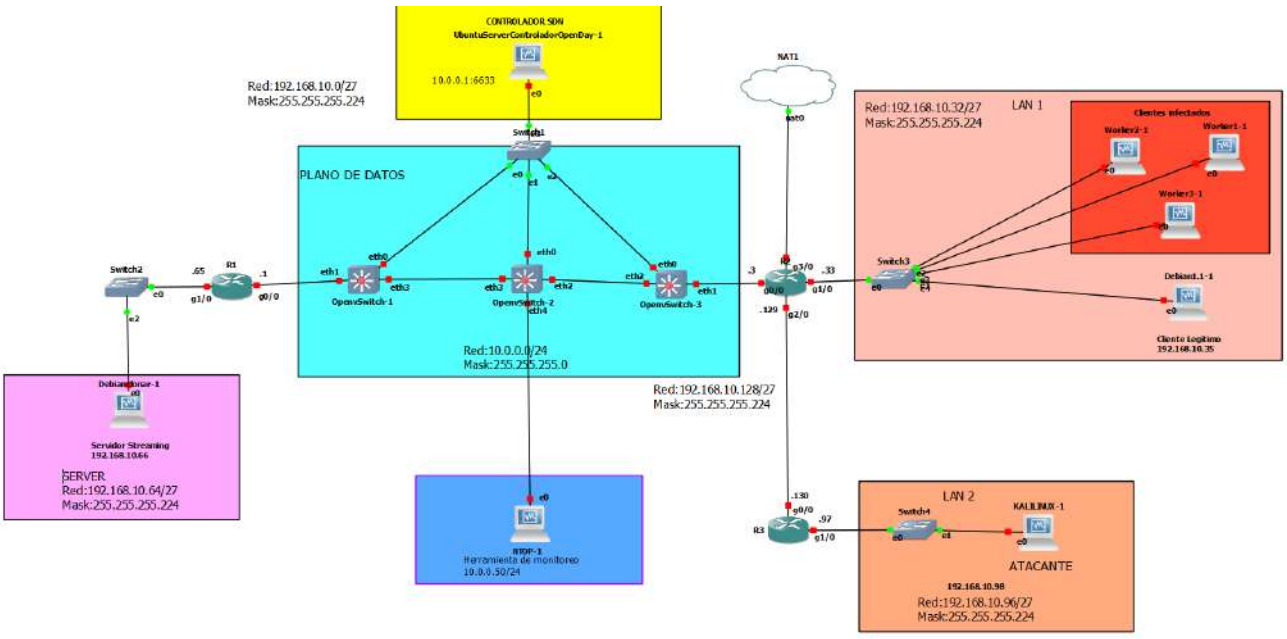
3.2.2. Topología lógica

Esta topología recoge la distribución lógica de la red, sobre la que opera la interoperabilidad SDN que enlaza a los hosts con el servidor y sitúa al atacante en el escenario. La Figura 12 presenta esta propuesta, con sus enlaces, el direccionamiento IP de cada interfaz y los equipos del escenario: el controlador SDN, el servidor de streaming, el cliente legítimo,

los cuatro nodos workers de la botnet (tres en la subred de clientes infectados de la LAN 1 y uno en el nodo Kali Linux), la máquina virtual de monitoreo con ntopng en la red de gestión y el botmaster Kali Linux en la LAN del atacante.

Figura 12

Topología lógica de la red SDN



La topología implementada en GNS3 refleja una arquitectura SDN jerárquica organizada en tres planos funcionales: el plano de control, donde reside el controlador junto con la máquina virtual de monitoreo ntopng en la red de gestión 10.0.0.0/24; el plano de datos, conformado por los tres Open vSwitch que ejecutan el reenvío; y las redes de área local que albergan los dispositivos finales (LAN 1, con el cliente legítimo de streaming y la subred de clientes infectados con los workers 2 a 4, y LAN 2, con el atacante Kali Linux, que aloja el C2 y el worker 1). Esta estructura permite una gestión centralizada del tráfico mediante el protocolo OpenFlow, mientras que el enrutamiento entre subredes lo realizan routers tradicionales con protocolos de enrutamiento dinámico.

3.2.2.1. Dispositivos que conforman la topología

Definido el escenario y ubicados los dispositivos según la topología del apartado anterior, esta sección detalla cada host, equipo de red y servidor que la integran, con énfasis en el consumo de recursos en el entorno de emulación. La Tabla 15 recoge, para cada dispositivo, la memoria RAM y el número de CPU asignados.

Tabla 15

Dispositivos que conforman la topología

Dispositivo	RAM	CPU	Sistema Operativo	Rol en la red
Controlador	3 GB	2	Linux-Ubuntu 22.04-1	Controlador SDN
Switches SDN	256 MB	1	Open vSwitch 2.17.8	Switches SDN (3 unidades)
Routers	512 MB	1	Cisco IOS 15.2(4)S6	Enrutadores (R1, R2, R3)
Servidor Streaming	3 GB	2	Linux	Servidor de streaming multimedia
Cliente legítimo	1 GB	1	Linux	Cliente de streaming (LAN 1)
Atacante (Kali Linux)	1 GB	1	Linux	Botmaster del sistema TrafficGen SDN (LAN 2)
Worker 1	1 GB	1	Linux Debian 12	Bot atacante (TrafficGen SDN, LAN 1)
Worker 2	1 GB	1	Linux Debian 12	Bot atacante (TrafficGen SDN, LAN 1)
Worker 3	1 GB	1	Linux Debian 12	Bot atacante (TrafficGen SDN, LAN 1)
Worker 4	1 GB	1	Linux Debian 12	Bot atacante (TrafficGen SDN, LAN 1)
VM Monitoreo (ntopng)	2 GB	2	Linux Debian 12	Agente de captura y análisis de tráfico (Red de gestión 10.0.0.50)
Total	16.3 GB	18		

Como se observa en la Tabla 15, la topología requiere en total aproximadamente 16.3 GB de memoria RAM y 18 CPU virtuales con todos los dispositivos activos simultáneamente. Estos valores se asignan con base en la documentación de cada componente, como se expone a continuación:

- El controlador constituye el plano de control centralizado de la arquitectura SDN y es responsable del cómputo de rutas, la gestión de la base de información de red (NIB) y la comunicación OpenFlow con todos los conmutadores. De acuerdo con la documentación oficial de Red Hat, los requerimientos mínimos para un entorno básico de ejecución del controlador son 2 núcleos de CPU, 2 GB de memoria RAM y 10 GB de almacenamiento [65]; a partir de estos valores, se asignan recursos iguales o superiores según las necesidades del sistema.
- Los conmutadores implementan el plano de datos mediante Open vSwitch (OVS) 2.17.8, implementación de referencia de OpenFlow en entornos virtualizados. Según la documentación de GNS3, 256 MB de RAM por conmutador OVS son suficientes para sostener el demonio *ovs-vswitchd*, la base de datos *ovsdb-server* y tablas de flujo de tamaño moderado, del orden de cientos de entradas [50].
- Los enrutadores de GNS3 utilizan la imagen oficial de Cisco IOS 15.2(4)S6 para la plataforma c7200; según la documentación de GNS3, la imagen *c7200-adventerprisek9-mz.152-4.M.bin* requiere un mínimo de 512 MB de RAM para arrancar y operar de forma estable [66].
- Para los entornos Linux se consideran los requisitos recomendados por el proyecto Debian: 2 GB de RAM y 10 GB de almacenamiento [67]. A los nodos workers de la botnet se asigna 1 GB de RAM y 1 vCPU por instancia, valor

suficiente para sostener el contenedor Docker del sistema TrafficGen SDN con 20 hilos concurrentes de ataque. La máquina virtual del agente de monitoreo recibe 2 GB de RAM y 2 vCPU, ya que ntopng mantiene en memoria las tablas de flujos activos, ejecuta el análisis de capa 7 sobre el tráfico replicado y sirve la interfaz web de visualización en tiempo real.

Estos requisitos son compatibles con el equipo anfitrión HP OMEN, que dispone de 32 GB de RAM y 12 hilos de CPU (6 núcleos físicos con Hyper-Threading). La asignación total de 16.3 GB queda dentro del margen disponible, y la demanda de 18 vCPU se sostiene mediante la sobreasignación (overcommit) habitual en entornos de virtualización, dado que no todas las máquinas virtuales operan simultáneamente al 100 % de carga. El margen restante es suficiente para el sistema operativo anfitrión y el propio GNS3.

3.2.2.2. Direccionamiento IPv4 de la topología

Establecida la distribución lógica y los dispositivos del escenario, la siguiente fase corresponde a la asignación del direccionamiento IP de cada enlace, configurado en los enrutadores, los conmutadores SDN, los servidores y los hosts de cada LAN. La Tabla 16 detalla, para cada dispositivo, su interfaz, dirección IP, Gateway y máscara de subred.

Como bloque de referencia se utiliza 192.168.10.0/24, sobre el que se aplica VLSM para obtener varias subredes: /27 para las LAN de clientes y la DMZ del servidor de streaming, y /30 para los enlaces punto a punto entre los routers de borde. Esta segmentación separa el tráfico de gestión del tráfico de datos, condición necesaria para estudiar de forma aislada un ataque sobre el servidor de streaming.

La red de gestión 10.0.0.0/24 agrupa al controlador y a las interfaces administrativas (mgmt0) de los tres Open vSwitch: el controlador utiliza la dirección 10.0.0.1 y los conmutadores OpenVsw1, OpenVsw2 y OpenVsw3, las direcciones 10.0.0.2, 10.0.0.3 y 10.0.0.4, respectivamente. Al residir en la misma subred, los Open vSwitch establecen la sesión

OpenFlow 1.3 con el controlador en el puerto TCP 6633 sin gateway intermedio, lo que reduce la latencia del plano de control y elimina puntos de fallo asociados al enrutamiento.

Tabla 16

Direccionamiento IPv4 de la topología del escenario general

Dispositivo	Interfaz	Dirección IP	Gateway	Máscara de subred
R1	g0/0	192.168.10.1	N/D	255.255.255.224
	g1/0	192.168.10.65	N/D	
R2	G0/0	192.168.10.3	N/D	255.255.255.224
	G1/0	192.168.10.33	N/D	
	g2/0	192.168.10.129	N/D	
	G3/0	DHCP	N/D	
R3	g1/0	192.168.10.97	N/D	255.255.255.224
	g2/0	ND	N/D	
	g0/0	192.168.10.130	N/D	255.255.255.224
OpenVsw1	mgmt0	10.0.0.2	N/D	255.255.255.0
OpenVsw2	mgmt0	10.0.0.3	N/D	255.255.255.0
OpenVsw3	mgmt0	10.0.0.4	N/D	255.255.255.0
Controlador	mgmt0	10.0.0.1	N/D	255.255.255.0
Servidor	e0	192.168.10.66	192.168.10.65	255.255.255.224
Cliente legítimo	e1	192.168.10.35	192.168.10.33	255.255.255.224
Atacante (Kali Linux)	e0	192.168.10.98	192.168.10.97	255.255.255.224
Worker 1	e0	192.168.10.98	192.168.10.97	255.255.255.224
Worker 2	e0	192.168.10.36	192.168.10.33	255.255.255.224
Worker 3	e0	192.168.10.37	192.168.10.33	255.255.255.224
Worker 4	e0	192.168.10.38	192.168.10.33	255.255.255.224
VM Monitoreo (ntopng)	mgmt0	10.0.0.50	N/D	255.255.255.0

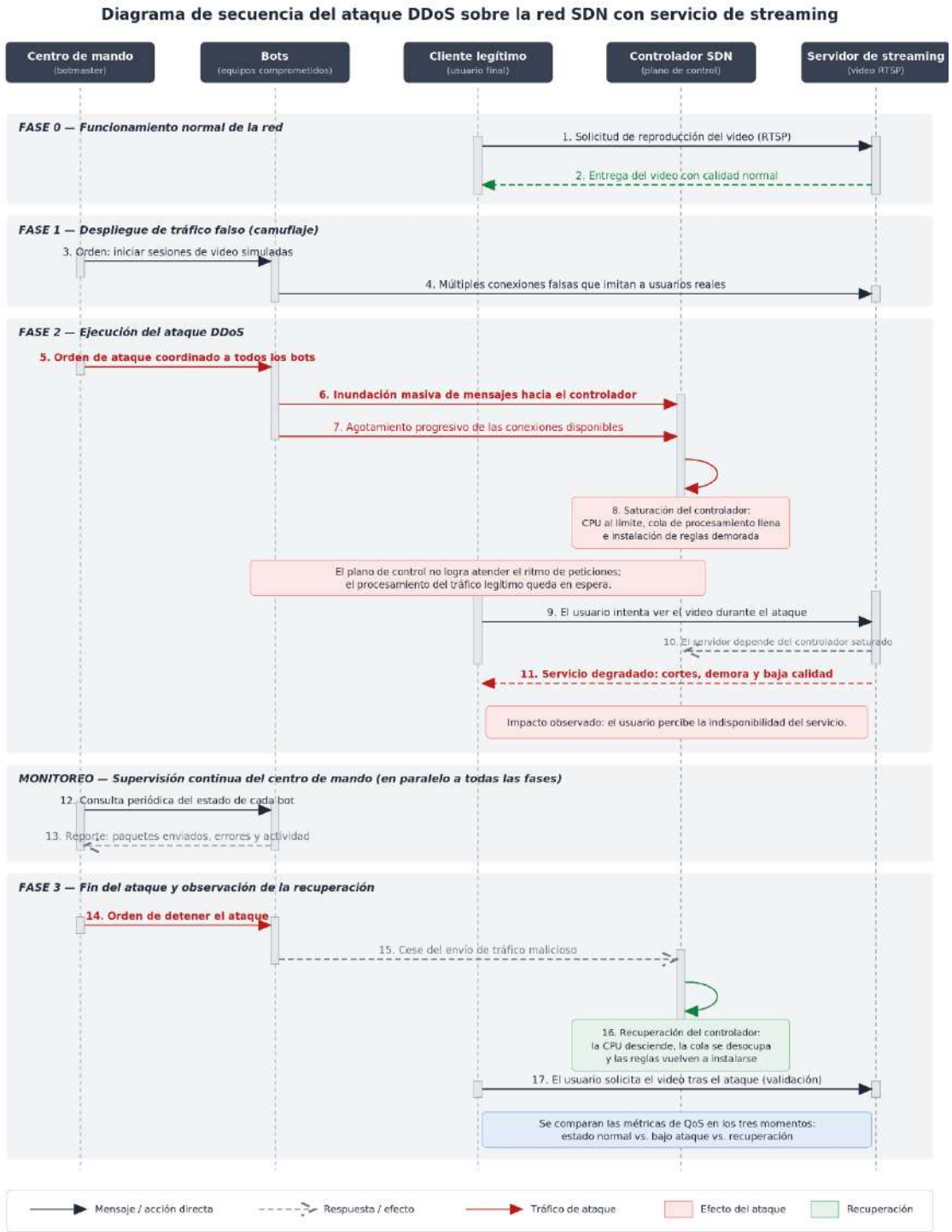
3.3. Diagrama de secuencia del ataque DDoS

Para describir el comportamiento dinámico del ataque sobre la topología propuesta, se elabora el diagrama de secuencia de la Figura 13, que modela los mensajes intercambiados entre los actores durante el ataque DDoS. El diagrama muestra en orden temporal cómo el tráfico generado por la botnet interfiere con el tráfico legítimo de streaming y cómo la saturación del servidor afecta al servicio.

- Actor 1, atacante (botmaster): operador que controla el ataque desde el nodo dashboard del sistema TrafficGen SDN. Envía comandos de mando y control (C2) a la botnet para iniciar, configurar y finalizar el ataque. No interactúa directamente con el servidor de streaming, sino a través de los workers desplegados como bots en la topología; cumple, por tanto, el rol de orquestador del ataque distribuido.
- Actor 2, cliente legítimo del servicio: usuario final que consume el servicio de streaming multimedia mediante peticiones HLS al servidor MediaMTX. Su tráfico atraviesa el plano de datos formado por los Open vSwitch y depende del controlador SDN para la programación de las reglas de reenvío. Es el actor cuya calidad de servicio (QoS) se degrada por efecto del ataque, lo que se evidencia en el aumento del RTT y del jitter, en la pérdida de paquetes y en la caída del throughput del flujo de video.

Figura 13

Diagrama de secuencia del ataque DDoS sobre la red SDN con streaming



La secuencia inicia con el envío del comando C2 desde el Actor 1 hacia la botnet (paso 1). En paralelo, el Actor 2 (cliente legítimo) mantiene una sesión HLS legítima con el servidor

de streaming (pasos 2 y 2'). Cuando los workers reciben la orden de iniciar el ataque, dirigen una avalancha de sesiones TCP contra el servicio HLS (8888/TCP) del servidor de streaming MediaMTX, el mismo que consume el cliente, agotando sus conexiones concurrentes y su capacidad de servicio. El agotamiento de las conexiones concurrentes incrementa la latencia con la que el servidor atiende las solicitudes (paso 4) y repercute directamente sobre el cliente legítimo (paso 5), traduciéndose en respuestas degradadas con incremento de RTT y jitter, pérdida de paquetes y caída del throughput (paso 6). Finalmente, el Actor 1 emite el comando de fin del ataque (paso 7).

3.3.1. Diseño del sistema TrafficGen SDN

Como pieza central de la investigación se desarrolla TrafficGen SDN, una herramienta propia escrita en Python que cumple dos funciones: monitorea varias sesiones RTSP simultáneas contra el servidor de streaming y, al mismo tiempo, dirige el ataque DDoS contra ese mismo servidor MediaMTX, al que satura con una avalancha de conexiones TCP contra su servicio HLS (8888/TCP). Ambas funciones operan de forma concurrente en un único contenedor Docker, lo que permite observar en tiempo real el efecto del ataque sobre la calidad del servicio de streaming.

La arquitectura del sistema se basa en el framework web *Flask* (versión 3.x), que actúa como servidor HTTP para la interfaz gráfica y la API REST [68]. La lógica de generación de tráfico se ejecuta en hilos daemon independientes (*threading.Thread*), de modo que el servidor web permanece disponible durante la generación de ataques de alta intensidad. El acceso concurrente a los almacenes de datos compartidos se protege mediante objetos *threading.Lock()*, lo que garantiza la consistencia de las métricas presentadas en la interfaz gráfica.

La Tabla 17 presenta las dos clases principales en torno a las cuales se estructura el sistema.

Tabla 17

Atributos y métodos principales de las clases RTSPClient y AttackSession

Clase	Atributo / Método principal	Descripción
RTSPClient	server_ip, server_port	Dirección del servidor RTSP objetivo (192.168.10.66:8554)
RTSPClient	stream_path	Ruta del stream solicitado (ej. live/stream1)
RTSPClient	status	Estado actual: connecting → streaming → error → stopped
RTSPClient	latency_ms / rtt_history	RTT de la última operación OPTIONS; historial de 60 muestras
RTSPClient	_session()	Establece sesión RTSP: DESCRIBE → (200 OK) → OPTIONS keep-alive cada 4 s
AttackSession	target_ip, target_port	Servidor de streaming objetivo, servicio HLS (192.168.10.66:8888)
AttackSession	mode	Modo de ataque: tcp_exhaust
AttackSession	n_threads	Número de hilos concurrentes (20 por worker en los escenarios E2 a E5)
AttackSession	get_info()	Retorna métricas en tiempo real: pps, Mbps, conexiones, errores

Cada instancia de RTSPClient ejecuta un ciclo de vida autónomo: establece la conexión TCP con el puerto RTSP del servidor, envía un mensaje DESCRIBE para solicitar la descripción SDP (Session Description Protocol) del stream y, si recibe una respuesta 200 OK, inicia un bucle de keep-alive con mensajes OPTIONS cada cuatro segundos. Este mecanismo

permite mantener la sesión de control sin consumir ancho de banda de video, lo que reproduce el comportamiento de un cliente IPTV o de videovigilancia que ha establecido la sesión de señalización, pero no decodifica el contenido multimedia. En condiciones de ataque, el monitor registra como errores y reconexiones las sesiones RTSP que no obtienen respuesta del servidor. La Tabla 18 describe el ataque implementado en la clase AttackSession y su mecanismo de impacto, y la Tabla 19, las variables de entorno configurables del sistema.

Tabla 18

Ataque implementado en AttackSession (tcp_exhaust) y su mecanismo de impacto

Modo de ataque	Protocolo	Mecanismo de impacto
tcp_exhaust	TCP	Vector principal: agota las conexiones TCP del servicio HLS (8888/TCP) del servidor de streaming MediaMTX abriendo y manteniendo conexiones concurrentes (20 hilos por worker), impidiendo el servicio a los clientes legítimos.

Tabla 19

Variables de entorno configurables del sistema TrafficGen SDN

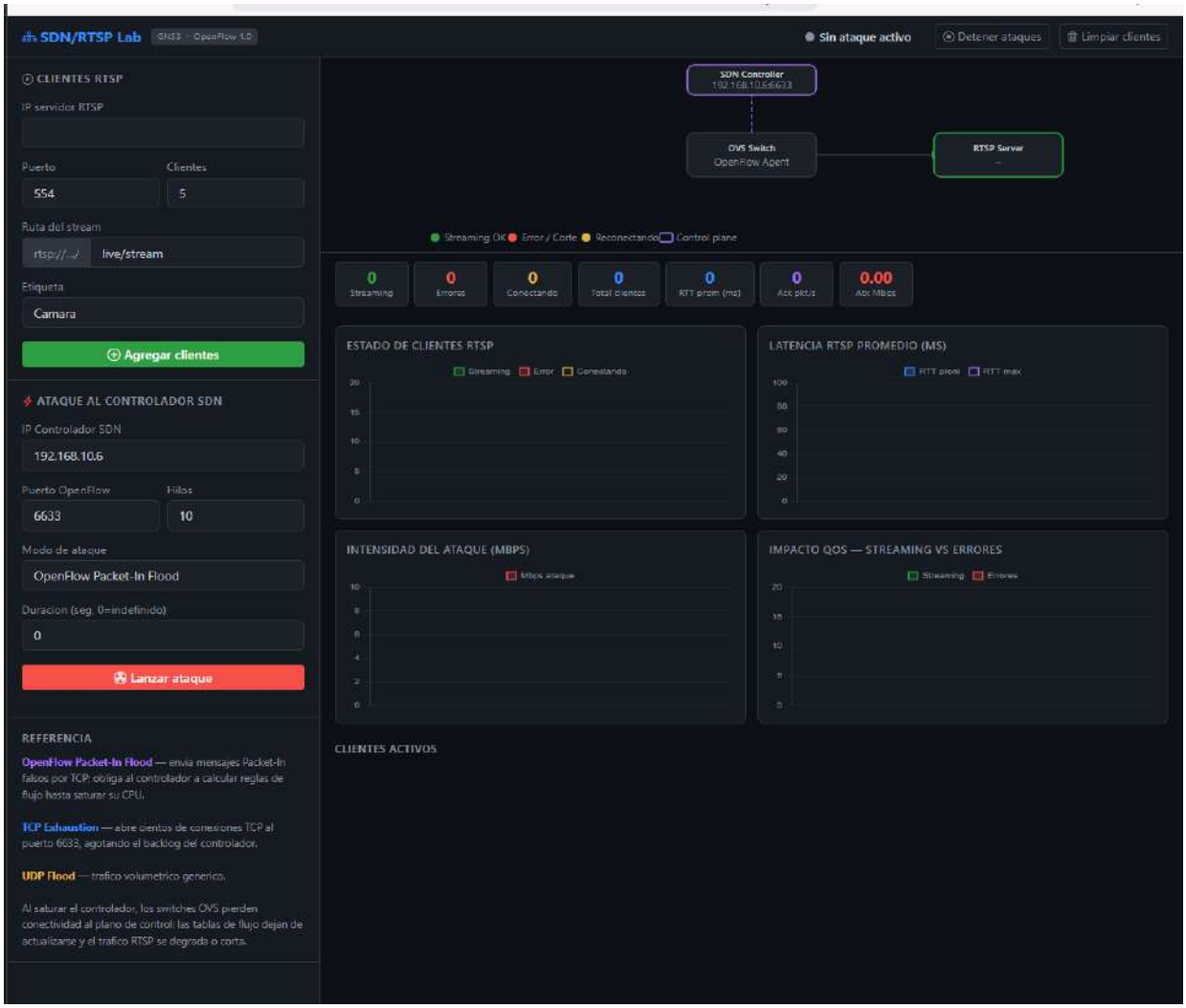
Parámetro	Variable de entorno	Valor en los experimentos	Rango válido
Modo de operación	MODE	worker / dashboard	standalone | worker | dashboard
Puerto HTTP	PORT	80 (worker) / 8090 (dashboard)	1 – 65535

Parámetro	Variable de entorno	Valor en los experimentos	Rango válido
Nombre del nodo	NODE_NAME	hostname	Cadena de texto (≤ 128 car.)
Workers remotos	WORKERS		URLs separadas por coma
IP del servidor objetivo	TARGET_IP	192.168.10.66	IPv4 válida
Puerto del servicio objetivo (HLS)	TARGET_PORT	8888	1 – 65535
Máximo de hilos	MAX_THREADS	500	1 – 500

La interfaz gráfica web, accesible desde cualquier navegador en el puerto configurado, presenta en tiempo real las métricas del sistema, actualizadas cada segundo mediante llamadas asíncronas a la API REST local (Figura 14). La visualización integra una representación SVG de la topología lógica de la red, una barra de estadísticas globales y cuatro gráficas independientes implementadas con la biblioteca Chart.js, cuyo eje Y fijo (que parte de cero con rangos predefinidos) garantiza una lectura consistente de los datos incluso con baja actividad.

Figura 14

Interfaz web del sistema TrafficGen SDN: topología lógica, métricas en tiempo real y gráficas de monitoreo con ejes Y fijos



En esta subsección se describen los elementos de diseño del sistema TrafficGen SDN como botnet de laboratorio: la arquitectura de despliegue distribuido entre el nodo de mando y control (dashboard) y los workers, y el modo de ataque sostenido contra el servicio HLS del servidor de streaming. Estos elementos constituyen la base conceptual de la implementación y la ejecución posteriores.

3.3.1.1. Arquitectura de despliegue distribuido

La imagen Docker admite tres modos de operación, seleccionables en tiempo de ejecución mediante la variable de entorno MODE: standalone, worker y dashboard. Este diseño

permite desplegar la misma imagen en roles diferentes sin necesidad de contenedores separados. En el modo worker, el nodo expone únicamente la API REST y acepta comandos remotos para iniciar o detener sesiones RTSP y ataques; en el modo dashboard, el nodo sirve la interfaz gráfica completa y agrega en tiempo real las métricas de todos los workers configurados en la variable WORKERS.

En el laboratorio SDN de esta investigación, el despliegue consiste en un nodo dashboard en la máquina virtual Kali Linux (192.168.10.98, puerto 8090), que actúa como punto de control centralizado (C2), y los nodos worker en (192.168.10.98, puerto 80) y en la red LAN 1 por el puerto 80, como generadores de tráfico distribuido (Tabla 20). La imagen se transfiere al nodo Debian mediante docker save / docker load, dado que ese segmento de red no tiene conectividad con Internet para descargarla de un registro público.

Tabla 20

Configuración de despliegue de los nodos del sistema distribuido TrafficGen SDN

Nodo	IP - Puerto	MODO	NODOS	Rol
Kali Linux	192.168.10.98:8090	dashboard	c2-kali	Control y visualización
Kali Linux	192.168.10.98:80	worker	Worker 1	Generador de tráfico
Debian	192.168.10.36:80 hasta 192.168.10.38: 80	worker	Worker 2 al 4	Generadores de tráfico

3.4. Implementación de la red SDN

La implementación comprende la configuración del plano de datos, mediante el protocolo de enrutamiento dinámico OSPF entre los routers de borde, y del plano de control, a través de la integración de los tres nodos Open vSwitch con el controlador SDN. Ambos planos

operan de forma independiente: el primero sobre la subred de datos (192.168.10.0/24) y el segundo sobre la red de gestión dedicada (10.0.0.0/24).

3.4.1. Integración entre el controlador y los conmutadores

El protocolo OpenFlow define la interfaz estándar entre el plano de control y el plano de datos de una arquitectura SDN [23]. El controlador (OpenDaylight o Ryu, según el escenario) se comunica con cada Open vSwitch mediante mensajes OpenFlow 1.3 (OF13) transmitidos por la red de gestión dedicada (10.0.0.0/24). Separar la red de gestión del plano de datos evita bucles de retroalimentación y garantiza que los mensajes de control no compitan con el tráfico del servicio. En el caso de Ryu, el controlador se ejecuta con las aplicaciones simple_switch_13 (conmutación reactiva L2) y ofctl_rest (API REST para consultar los flujos) y escucha en el puerto TCP 6633.

3.4.1.1. Configuración de los Open vSwitch

Open vSwitch (OVS) es un conmutador virtual multicapa de código abierto que implementa el protocolo OpenFlow y opera como elemento de reenvío en el plano de datos SDN [23]. Cada nodo OVS de la topología requiere la creación de un bridge virtual (br0), la incorporación al bridge de las interfaces de datos excepto la interfaz de gestión eth0, la conexión con el controlador y la habilitación de OpenFlow 1.3. Además, se configura el modo de fallo seguro (fail-mode secure), en el que el conmutador descarta todo el tráfico sin un flujo coincidente instalado, de modo que el controlador ejerce el control exclusivo del reenvío.

La Tabla 21 muestra los comandos básicos para establecer la conexión con el controlador alojado en la máquina virtual. Estos comandos se aplican en cada OVS, con eth0 como puerto de gestión, e incluyen la dirección IP, el puerto y la versión del protocolo utilizados para la conexión.

Tabla 21

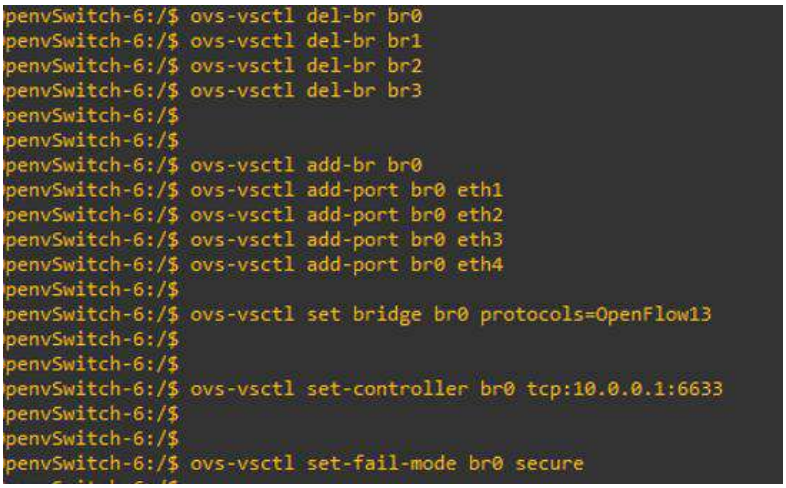
Comandos de configuración de cada Open vSwitch

Nro.	Comando	Descripción / Parámetro variable
1	`ovs-vsctl add-br br0`	Crear el bridge virtual. Igual en todos los OVS.
2	`ovs-vsctl add-port br0 eth1`	Agregar interfaz de datos eth1 al bridge.
3	`ovs-vsctl add-port br0 eth2`	Agregar interfaz de datos eth2 al bridge.
4	`ovs-vsctl add-port br0 eth3`	Agregar interfaz de datos eth3 al bridge.
5	`ovs-vsctl add-port br0 eth4`	Solo OVS-4: interfaz hacia SW-Ataque.
6	`ovs-vsctl set-controller br0 tcp:10.0.0.1:6633`	Registrar el controlador SDN. IP fija en todos los OVS.
7	`ovs-vsctl set bridge br0 protocols=OpenFlow13`	Habilitar OpenFlow versión 1.3.
8	`ovs-vsctl set-fail-mode br0 secure`	Modo fallo seguro: descarta tráfico sin flujo activo.
9	`ip addr add [IP_GESTIÓN]/24 dev eth0`	Variable: ver Tabla 16 para cada switch.
10	`ip link set eth0 up`	Habilitar la interfaz de gestión.

La Figura 15 muestra la configuración aplicada en GNS3, que debe replicarse en cada equipo con las direcciones asignadas. La interfaz eth0 de los OVS no debe agregarse al bridge, ya que, si se agrega, el nodo pierde la comunicación de gestión con el controlador; solo las interfaces eth1, eth2, eth3 y eth4 se incorporan al bridge.

Figura 15

Configuración de Open vSwitch en cada nodo



La conexión con el controlador se verifica mediante el comando `ovs-vsctl show`, que muestra la configuración ingresada y el mensaje *is_connected: true, con el* que se valida el enlace con el controlador SDN (Figura 16).

Figura 16

Verificación de la conexión entre el controlador y Open vSwitch



3.4.2. Configuración del protocolo OSPF

OSPF (Open Shortest Path First, RFC 2328) es un protocolo de enrutamiento de estado de enlace que permite a los routers intercambiar información de topología y calcular las rutas óptimas mediante el algoritmo de Dijkstra. En la topología de la Figura 12, R1 actúa como gateway del servidor de streaming (192.168.10.64/27) y R2, como gateway de las LAN de

clientes (192.168.10.32/27, 192.168.10.96/27 y 192.168.10.128/27). Ambos routers operan en el área 0 de OSPF y establecen la adyacencia a través de la malla SDN, que actúa como medio de tránsito transparente en capa 2. La convergencia de OSPF requiere que los Open vSwitch tengan flujos activos que permitan el paso de los mensajes multicast de control [69].

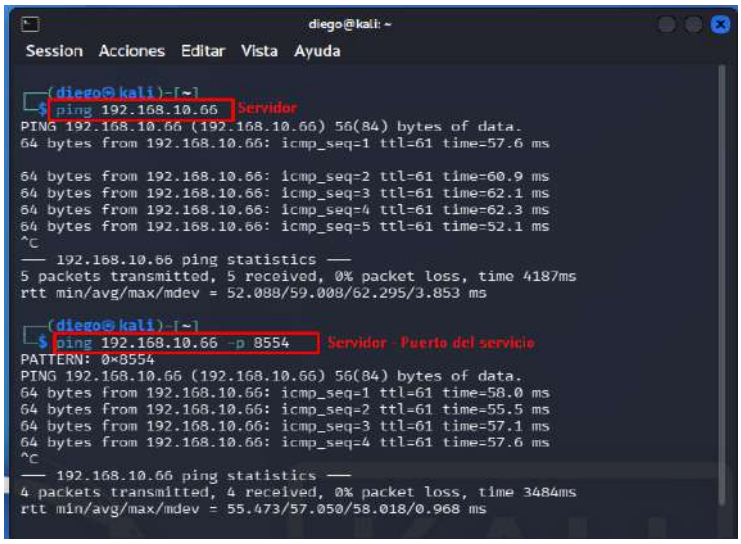
En esta topología híbrida, el papel de OSPF se reduce al borde de la topología: los Open vSwitch no participan en el cálculo de rutas, sino que se limitan a propagar las tramas que el controlador autoriza mediante los flujos instalados. Por ello, es necesario verificar manualmente que los flujos OpenFlow permitan el paso del tráfico multicast 224.0.0.5 y 224.0.0.6, que los routers OSPF emplean para descubrir vecinos y enviar LSA; de lo contrario, la adyacencia entre R1 y R2 no se establece y la red opera como dos islas inconexas. Durante la implementación se constata que esta verificación es uno de los puntos más sensibles del entorno, ya que una saturación del controlador durante un ataque DDoS puede afectar la convergencia de OSPF y, con ella, la accesibilidad del servidor MediaMTX desde las LAN de clientes.

3.4.3. Verificación de conectividad y enrutamiento

Para validar que el plano de datos converge antes de ejecutar las pruebas, se verifica el alcance desde el nodo Kali (donde reside el C2) hasta el servidor de streaming. La Figura 17 muestra el resultado de las pruebas de conectividad y la convergencia del enrutamiento entre ambos extremos.

Figura 17

Convergencia entre el nodo Kali (C2) y el servidor de video



3.5. Implementación del entorno de pruebas

El diseño y la puesta en marcha del entorno de pruebas son clave para asegurar la validez y la reproducibilidad de los experimentos. El entorno recrea, mediante virtualización, las condiciones operativas de una red SDN sometida a distintos niveles de carga: por un lado, el tráfico legítimo de los clientes de streaming y, por otro, los ataques de saturación dirigidos al servidor de streaming.

A partir del análisis comparativo de la Tabla 9, se selecciona GNS3 como entorno de emulación de red para implementar el escenario experimental. GNS3 tiene un uso extendido en los ámbitos académico e industrial y permite emular topologías de red complejas; entre sus capacidades destaca la integración nativa de imágenes Docker, máquinas virtuales QEMU y equipos físicos mediante conexiones seriales o Ethernet, lo que permite reunir todos los componentes del experimento en un mismo entorno [70].

La Tabla 22 lista los componentes del entorno de pruebas y su configuración en GNS3: un controlador SDN compatible con OpenFlow 1.3, en la dirección 10.0.0.1:6633; tres conmutadores virtuales Open vSwitch (OVS) configurados como agentes OpenFlow; un

servidor de streaming multimedia basado en MediaMTX, accesible en 192.168.10.66:8888; y los nodos cliente desde los que se despliegan los contenedores Docker del sistema desarrollado.

La elección de los rangos de direccionamiento responde a la necesidad de aislar el tráfico de control del tráfico de datos sin recurrir a VLAN adicionales, lo que reduce la complejidad del monitoreo y el diagnóstico durante las pruebas. Cada subred se dimensiona según el número de equipos que se despliegan y los espacios libres requeridos para los enlaces punto a punto entre R1, R2 y los Open vSwitch. El servidor MediaMTX y el controlador se ubican en redes separadas para no exponer el plano de control a los clientes de streaming, lo que reproduce con razonable fidelidad la práctica de un operador de servicios. Este detalle topológico es relevante para el diseño metodológico, dado que condiciona los puntos de captura definidos en la Tabla 22 y, por extensión, la validez de las mediciones de QoS que se discuten en el capítulo siguiente.

Tabla 22

Componentes del entorno de pruebas y su configuración en el entorno de emulación GNS3

Componente	Tecnología	Dirección IP:Puerto	Rol en la topología
Controlador SDN	OpenFlow 1.3	10.0.0.1:6633	Plano de control de la red SDN
Switches virtuales	Open vSwitch (OVS)	10.0.0.0/24	Conmutación con soporte OpenFlow
Servidor streaming	MediaMTX + FFmpeg	192.168.10.66:8888	Distribución de streams RTSP/HLS
Nodo cliente Debian	Debian 11 (GNS3 VM)	192.168.10.35	Despliegue del contenedor Monitor

Componente	Tecnología	Dirección IP:Puerto	Rol en la topología
Nodo cliente Kali	Kali Linux (GNS3 VM)	192.168.10.98	Despliegue del contenedor Atacante

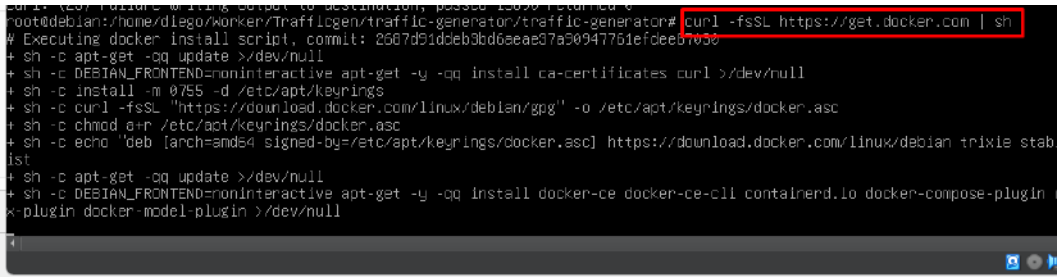
3.5.1. Virtualización con Docker en GNS3

Docker es una plataforma de código abierto orientada al empaquetado de aplicaciones en contenedores. Su funcionamiento se basa en los mecanismos de aislamiento del kernel de Linux, en particular en los namespaces y los cgroups. A diferencia de la virtualización tradicional basada en hipervisor, los contenedores Docker comparten el núcleo del sistema operativo anfitrión, lo que reduce el consumo de memoria y acorta los tiempos de arranque. Este comportamiento resulta útil en escenarios de laboratorio en los que varios nodos virtualizados comparten los recursos del anfitrión [70].

La preparación de los nodos del laboratorio comienza con la instalación de Docker en cada máquina virtual. La Figura 18 muestra la descarga de los paquetes oficiales de Docker desde el repositorio recomendado.

Figura 18

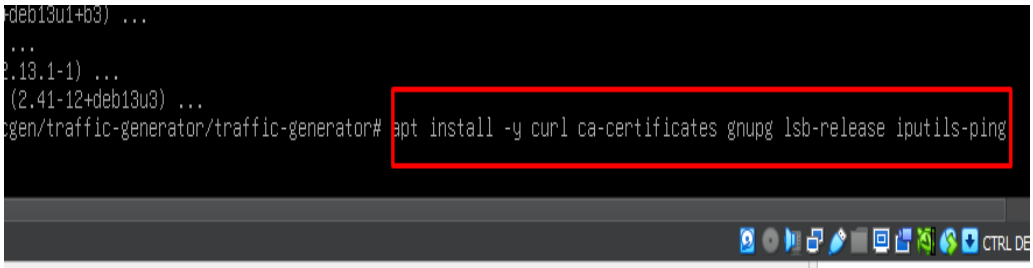
Descarga de los paquetes de Docker



Antes de instalar Docker, se incorporan los complementos del sistema (curl, ca-certificates, gnupg y dependencias asociadas) necesarios para gestionar el repositorio oficial y verificar las firmas de los paquetes, como se ilustra en la Figura 19.

Figura 19

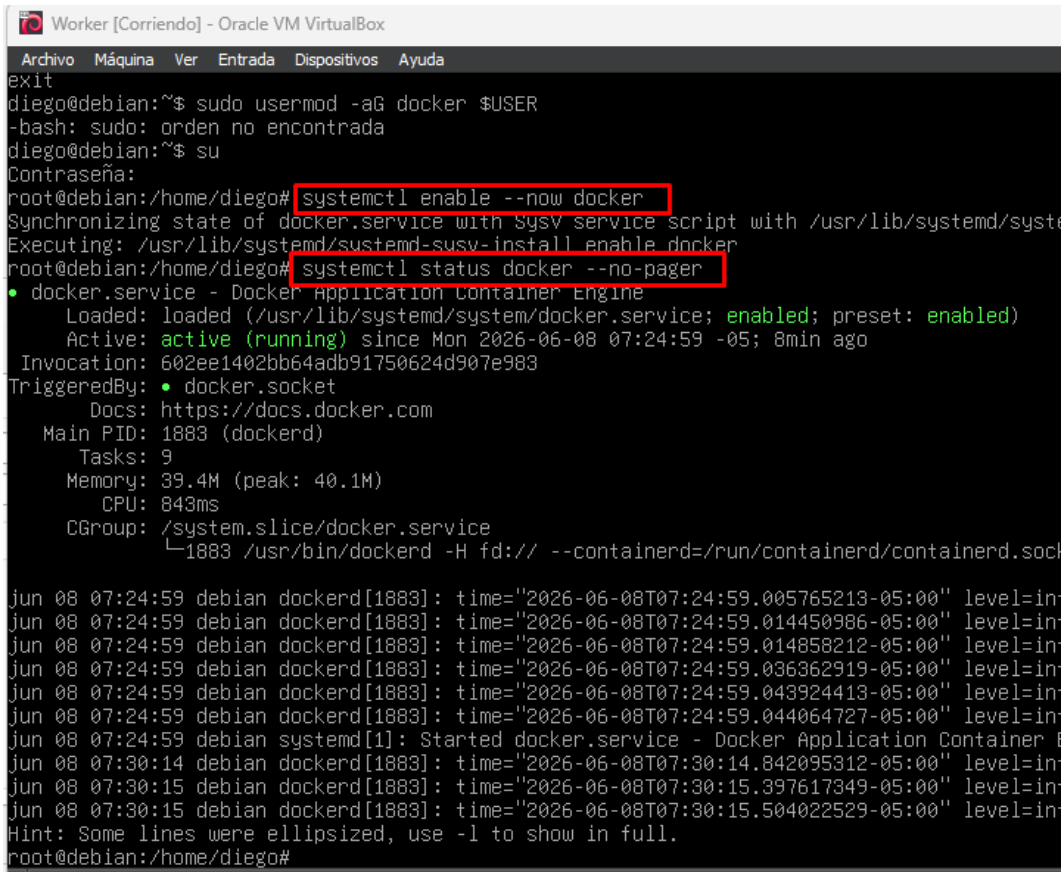
Instalación de complementos previos a Docker



Una vez completada la instalación, se ejecuta el contenedor hello-world para confirmar que el demonio de Docker está activo y que la cadena de descarga, ejecución y limpieza opera correctamente. La Figura 20 evidencia la verificación.

Figura 20

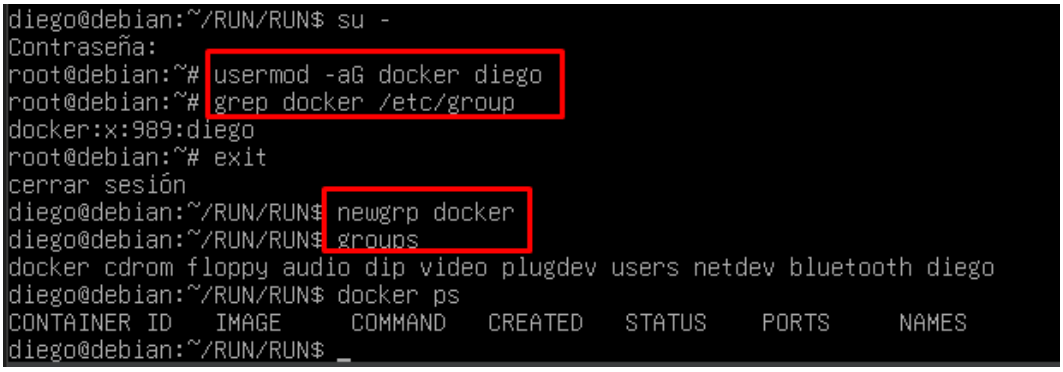
Verificación de la instalación de Docker



Para evitar el uso de privilegios elevados en cada operación, el usuario del laboratorio se incorpora al grupo docker, lo que habilita la ejecución de comandos sin sudo. La Figura 21 muestra el comando aplicado y el efecto al iniciar sesión nuevamente.

Figura 21

Incorporación del usuario al grupo Docker



3.5.2. Implementación del sistema TrafficGen SDN

Definida la arquitectura, esta subsección recoge los aspectos prácticos del despliegue: la construcción de la imagen Docker que empaqueta el sistema TrafficGen SDN, el subsistema de monitoreo que captura las métricas durante los experimentos y los comandos utilizados para construir, desplegar y verificar el sistema distribuido.

3.5.2.1. Construcción de la imagen Docker

Con el objetivo de mantener la reproducibilidad del entorno en las máquinas virtuales de GNS3, el sistema TrafficGen SDN se empaqueta en una imagen Docker construida a partir de python:3.11-slim. La receta de construcción reside en el archivo Dockerfile, en el directorio traffic-generator/ del repositorio. La imagen agrupa las dependencias del sistema (curl, net-tools, iproute2) y las bibliotecas de Python Flask y Gunicorn, que se instalan con pip sin caché para reducir el tamaño final del artefacto [71].

Se emplea Gunicorn como servidor web en lugar del servidor de desarrollo Werkzeug, ya que este último no ofrece la concurrencia requerida para ejecutar decenas de hilos de ataque en paralelo. La configuración seleccionada utiliza workers de tipo gthread, con 2 procesos y 8

hilos por proceso, un tiempo de espera (timeout) de 120 s y un nivel de registro warning. Con estos parámetros, la API REST mantiene su disponibilidad incluso con cargas intensivas [72]. La Tabla 23 describe las capas del Dockerfile y su propósito.

Tabla 23

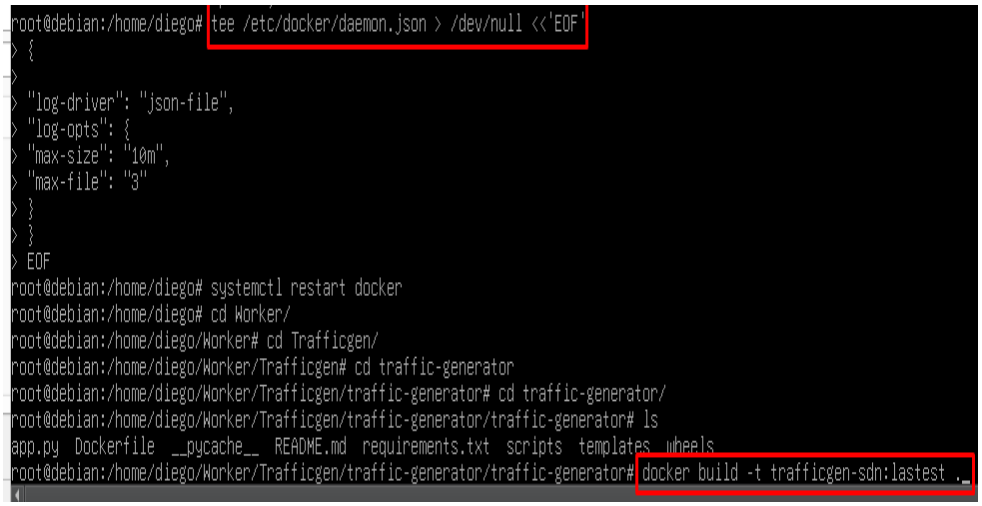
Capas del Dockerfile del sistema TrafficGen SDN y su propósito

Instrucción Dockerfile	Propósito
`FROM python:3.11-slim`	Imagen base oficial de Python 3.11, versión slim sin paquetes innecesarios
`RUN apt-get install ...`	Herramientas de diagnóstico de red: ping, netstat, ip (iproute2), curl
`RUN pip install flask gunicorn`	Dependencias de la aplicación: servidor web Flask y WSGI Gunicorn
`COPY app.py templates/`	Lógica de negocio (app.py) e interfaz web (templates/)
`ENV MODE PORT NODE_NAME WORKERS`	Variables de entorno configurables en tiempo de ejecución sin reconstruir la imagen
`CMD gunicorn --workers 2 --threads 8`	Servidor WSGI de producción con soporte multi-hilo y timeout de 120 s

Durante la construcción de la imagen se observa que el proceso puede agotar la memoria de los nodos con menor capacidad. La Figura 22 muestra el ajuste aplicado al archivo de configuración para evitar errores por consumo excesivo de memoria durante la construcción.

Figura 22

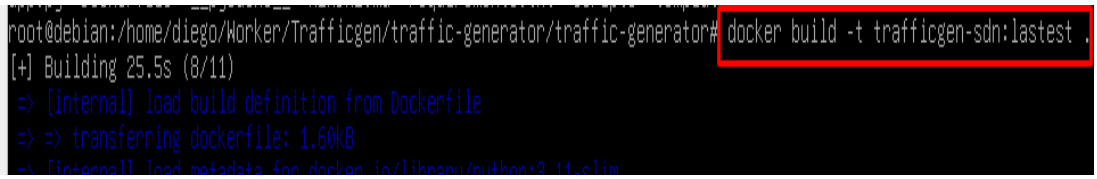
Ajuste de memoria para la construcción de la imagen Docker



La imagen se construye a partir de un directorio que incorpora la carpeta wheels con las dependencias precompiladas de Python. Esto reduce el tiempo de build y elimina la necesidad de descargar paquetes durante la ejecución, como se observa en la Figura 23.

Figura 23

Estructura del repositorio con la carpeta wheels para la construcción de la imagen



3.5.2.2. Sistema de monitoreo y captura de métricas en tiempo real

El dashboard agrega en tiempo real las métricas de todos los workers mediante el sondeo periódico del endpoint /api/worker/status de cada nodo, con un ciclo de refresco de 2 s y un tiempo de espera máximo de 6 s por worker. La interfaz presenta siete indicadores clave de rendimiento (KPI) en la barra de estado superior: workers en línea, sesiones RTSP activas, errores acumulados, RTT promedio en ms, paquetes de ataque por segundo, tráfico de ataque en Mbps y número de ataques activos. Además, muestra cuatro gráficas de series temporales con una ventana deslizante de 60 puntos (2 min de historial visible).

Cada gráfica capta un aspecto distinto del impacto del ataque. Las gráficas de sesiones RTSP y de latencia evidencian el impacto sobre el servicio multimedia; la gráfica de tráfico de ataque refleja la intensidad del vector DDoS; y la gráfica de disponibilidad de workers monitorea el estado del sistema distribuido de generación de tráfico. Los ejes Y tienen rangos fijos (suggestedMax en Chart.js) para evitar que la escala automática distorsione la percepción de la magnitud del impacto al comparar escenarios. La Tabla 24 resume las métricas que captura este sistema.

Tabla 24

Métricas capturadas por el sistema de monitoreo del dashboard distribuido

Gráfica	Métrica agregada	Unidad	Aspecto que evidencia
Sesiones RTSP agregadas	Streaming activos / Errores / Conectando de todos los workers	Sesiones	Degradación del servicio multimedia bajo ataque DDoS
Latencia RTSP promedio	RTT promedio y máximo del keep-alive OPTIONS	ms	Aumento de latencia en el plano de datos durante el ataque
Tráfico de ataque agregado	Mbps totales generados por todos los workers	Mbps	Intensidad del vector DDoS sobre el servidor de streaming
Workers online / offline	Nodos del sistema distribuido disponibles vs. caídos	Unidades	Disponibilidad y estabilidad del sistema de generación de tráfico

3.5.2.3. Comandos de ejecución y acceso al sistema distribuido

El despliegue del sistema requiere que la imagen Docker trafficgen-sdn:latest esté disponible en cada nodo de la topología GNS3. La construcción se realiza una sola vez en el nodo principal (Kali Linux) y la imagen se transfiere a los demás nodos mediante docker save / docker load, dado que las máquinas virtuales de GNS3 no tienen acceso directo a registros públicos de contenedores. La Tabla 25 detalla el flujo completo de construcción, despliegue y verificación del sistema.

Tabla 25

Comandos de construcción, despliegue y verificación del sistema TrafficGen SDN

Nº	Comando	Descripción
1	`docker build -t trafficgen-sdn:latest .`	Construye la imagen Docker desde el Dockerfile ubicado en traffic-generator/. Debe ejecutarse en el directorio del proyecto.
2	`docker save trafficgen-sdn:latest | gzip > /tmp/trafficgen.tar.gz`	Exporta la imagen a un archivo comprimido para transferencia sin registro público de contenedores.
3	`scp /tmp/trafficgen.tar.gz user@192.168.10.35:/tmp/`	Transfiere la imagen empaquetada al nodo worker en Debian mediante SCP.
4	`docker load < /tmp/trafficgen.tar.gz`	Importa la imagen en el nodo Debian. Ejecutar en 192.168.10.35 tras la transferencia.

Nº	Comando	Descripción
5	`docker run -d --name trafficgen-worker --network host --cap-add=NET_RAW --ulimit nofile=65535:65535 -e MODE=worker -e PORT=80 -e NODE_NAME=worker-3 trafficgen-sdn:latest`	Inicia el worker en Debian (192.168.10.35). --network host: comparte la pila de red del host. --cap-add=NET_RAW: permite raw sockets para la generación de tráfico. --ulimit: amplía el límite de descriptores de archivo a 65535.
6	`docker run -d --name trafficgen-worker-kali --network host --cap-add=NET_RAW --ulimit nofile=65535:65535 -e MODE=worker -e PORT=80 -e NODE_NAME=worker-35 trafficgen-sdn:latest`	Inicia el segundo worker en Kali Linux (192.168.10.98, puerto 80). Coexiste con el dashboard en el mismo host dado que usan puertos distintos.
7	`docker run -d --name trafficgen-dashboard --network host --ulimit nofile=65535:65535 -e MODE=dashboard -e PORT=8090 -e NODE_NAME=c2-kali -e WORKERS="http://192.168.10.35:80,http://192.`	Inicia el nodo C2 (dashboard) en Kali Linux puerto 8090. La variable WORKERS enumera las URLs de los workers a sondear. No requiere --cap-add=NET_RAW porque no genera tráfico directo.

N°	Comando	Descripción
	168.10.98:80" trafficgen-sdn:latest	
8	curl -s http://192.168.10.35:80/api/worker/info	Verifica que el worker en Debian está operativo. Debe devolver JSON con modo: “worker” y uptime_s mayor a 0.
9	curl -s http://192.168.10.98:80/api/worker/info	Verifica que el segundo worker en Kali está operativo.
10	curl -s http://192.168.10.98:8090/api/dashboard/workers	Lista el estado de todos los workers desde el C2. Devuelve JSON con online: true cuando ambos workers responden.
11	http://192.168.10.98:8090 (navegador web)	Acceso a la interfaz gráfica del dashboard. Disponible desde cualquier equipo con conectividad a 192.168.10.98 en la red de gestión del laboratorio GNS3.

Una vez que el dashboard detecta todos los workers en línea (indicador verde en la sección *Workers Conectados*), el operador puede iniciar sesiones en todos los workers simultáneamente mediante el Broadcast*: Clientes RTSP* y lanzar ataques coordinados desde el Broadcast de ataque. Las cuatro gráficas de monitoreo muestran datos en tiempo real desde el primer ciclo de sondeo (2 s).

El despliegue de cada worker de la botnet comienza con la clonación del repositorio del proyecto en el nodo correspondiente. La Figura 24 muestra el procedimiento ejecutado en el worker 3.

Figura 24

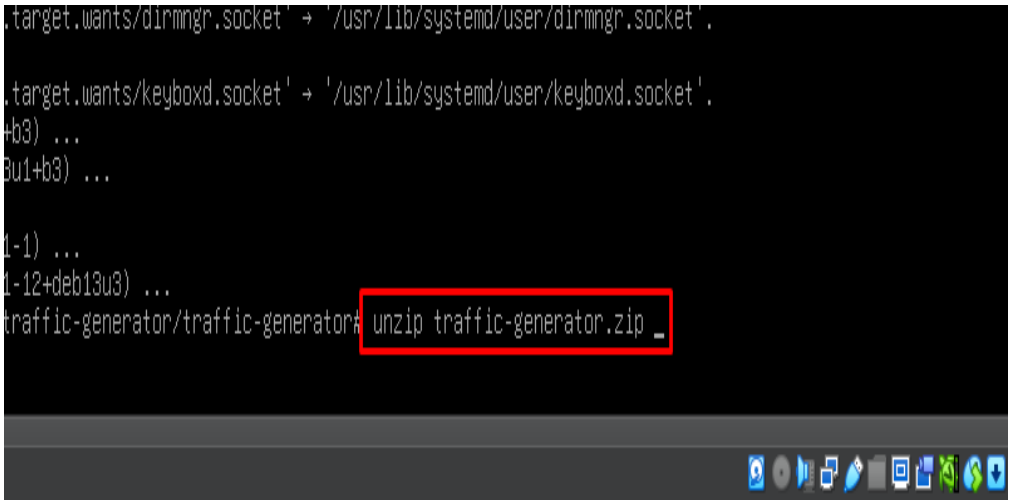
Clonación del repositorio en el worker 3



Cuando el código del worker se distribuye como archivo comprimido, se utiliza unzip para extraer su contenido en el directorio de trabajo, tal como se muestra en la Figura 25.

Figura 25

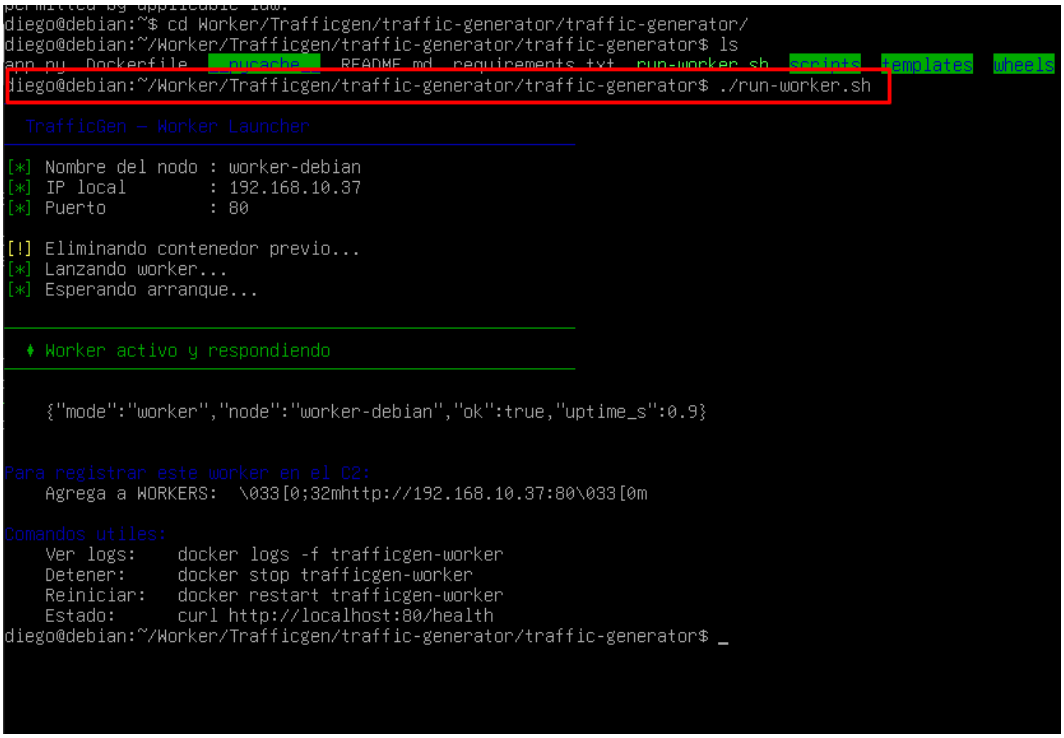
Extracción del paquete del worker mediante unzip



Tras compilar la imagen y preparar el contenedor, se lanza el proceso del worker. La Figura 26 documenta su ejecución y la conexión inicial hacia el nodo C2.

Figura 26

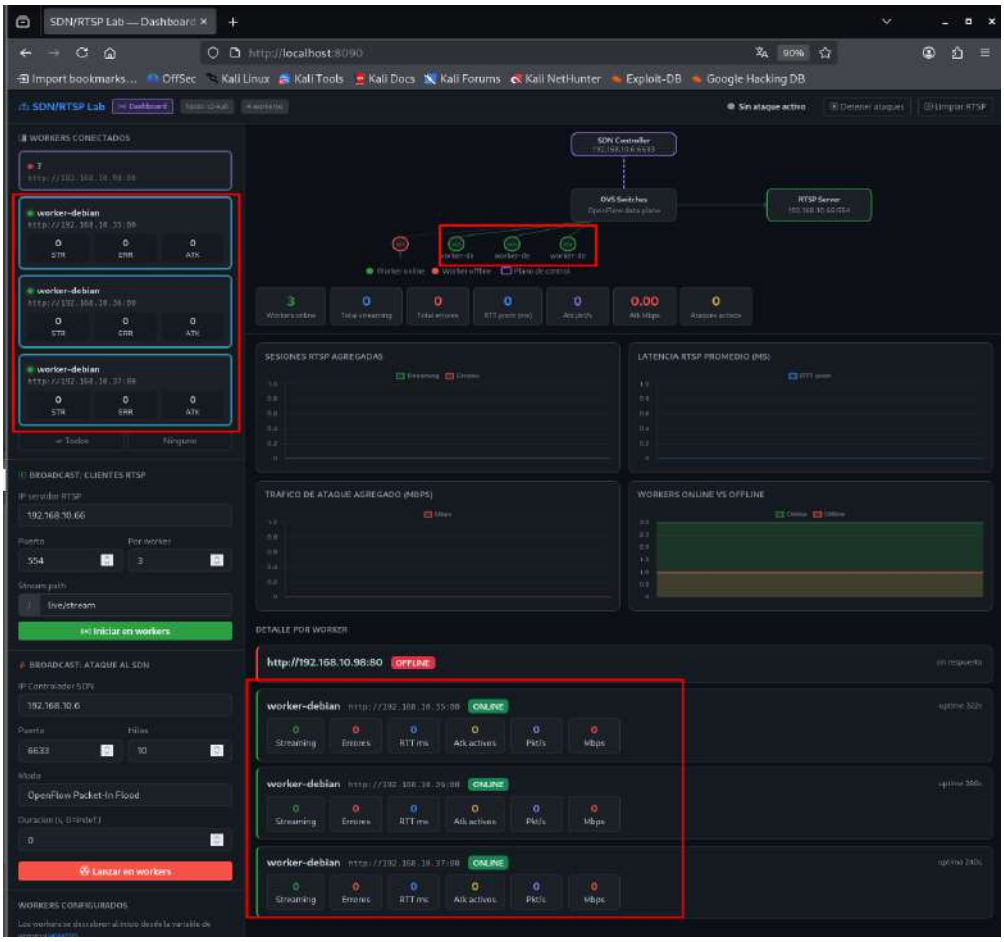
Ejecución del worker y conexión al nodo C2



Una vez inicializados todos los workers, el controlador C2 alojado en Kali Linux reporta los nodos disponibles en su tablero. La Figura 27 muestra el estado de la botnet con los workers en línea.

Figura 27

Workers en línea reportados por el controlador C2 en Kali Linux



3.5.3. Implementación del servidor de streaming MediaMTX

El servidor de streaming seleccionado (MediaMTX) publica y distribuye los flujos multimedia que consumen los clientes legítimos y que, en el presente trabajo, constituye el objetivo principal del ataque DDoS. El servicio HLS se expone en el puerto 8888: es el que consume el cliente legítimo y, a la vez, el blanco de las conexiones maliciosas de la botnet del sistema TrafficGen SDN. El servicio RTSP, en el puerto 8554, se emplea para publicar el flujo de origen.

El servidor se ejecuta en un contenedor Docker en el host destinado al servicio y publica sus puertos hacia la topología GNS3. Su comportamiento se controla mediante el archivo de

configuración mediamtx.yml, en el que se habilitan los protocolos RTSP y HLS y se declaran las rutas (paths) de los flujos publicados.

Dado que MediaMTX no genera contenido por sí mismo, la fuente de video se inyecta mediante FFmpeg, que actúa como publicador (publisher) RTSP: lee un archivo de video en bucle o, en su defecto, una fuente de prueba sintética y lo envía a una ruta del servidor en el puerto 8554, desde donde queda disponible para los clientes. Este esquema reproduce un servicio de video bajo demanda o de videovigilancia IP sin depender de hardware de captura.

La Tabla 26 resume los comandos empleados para levantar el servidor y publicar el flujo de prueba en el puerto 8554. Una vez publicada la ruta, MediaMTX la reexpone por HLS en http://<IP_servidor>:8888/stream/, dirección por la que la consume el cliente legítimo y contra cuyo puerto actúa el ataque.

Tabla 26

Comandos de despliegue del servidor MediaMTX y publicación del flujo con FFmpeg

N°	Comando	Descripción
1	`docker run --rm -d --name mediamtx -p 8888:8888 -p 8554:8554 -p 8554:8554/udp -p 8000:8000/udp -p 8001:8001/udp -v $(pwd)/mediamtx.yml:/mediamtx.yml bluenviron/mediamtx:latest`	Despliega el servidor MediaMTX en un contenedor Docker, publicando los puertos RTSP 8554/TCP y HLS 8888/TCP y montando el archivo de configuración mediamtx.yml.
2	`ffmpeg -re -stream_loop -1 -i video.mp4 -c copy -f rtsp -`	Publica en bucle un archivo de video hacia la ruta

N°	Comando	Descripción
	`rtsp_transport tcp rtsp://127.0.0.1:8554/stream`	/stream del servidor mediante RTSP sobre el puerto 8554, sin recodificar (modo copy).
3	`ffmpeg -re -f lavfi -i testsrc=size=1280x720:rate=25 -f lavfi -i sine=frequency=1000 -c:v libx264 -preset veryfast -tune zerolatency -c:a aac -f rtsp rtsp://127.0.0.1:8554/stream`	Genera una fuente de video y audio sintética (patrón de prueba 720p a 25 fps) y la publica como flujo RTSP en el puerto 8554, útil cuando no se dispone de un archivo de video.
4	`ffprobe -rtsp_transport tcp rtsp://192.168.10.x:8554/stream`	Verifica desde un cliente que el flujo RTSP se encuentra publicado y es accesible en el puerto 8554.
5	`curl http://127.0.0.1:9997/v3/paths/list`	Consulta la API nativa de MediaMTX para comprobar las rutas activas y el número de lectores conectados al servidor.

3.5.4. Verificación de la implementación de la red SDN

Una vez configurados el plano de control y el plano de datos, se comprueba el funcionamiento de la topología mediante la asignación de direcciones IP de gestión a los Open

vSwitch, la creación del bridge OVS y la asignación de interfaces, la definición del protocolo OpenFlow 1.3 y del identificador de cada conmutador, la conexión con el controlador SDN y la configuración de los flujos OpenFlow que permiten la convergencia de OSPF.

3.5.4.1. Asignación de IP de gestión y conectividad con el controlador

Cada Open vSwitch dispone de una interfaz eth0 dedicada exclusivamente a la administración del nodo. Se asigna la dirección IPv4 correspondiente al rango de gestión 10.0.0.0/24 y se verifica la conectividad de capa 3 hacia el controlador SDN (10.0.0.1). La Figura 28 evidencia la asignación de la IP de gestión y la respuesta exitosa del comando `ping` hacia el controlador, condición necesaria para el establecimiento posterior de la sesión OpenFlow.

Figura 28

Asignación de IP de gestión a eth0 y verificación de conectividad con el controlador SDN

```
OpenvSwitch-1:/$ ip addr add 10.0.0.10/24 dev eth0
OpenvSwitch-1:/$ ping -c 3 10.0.0.1
PING 10.0.0.1 (10.0.0.1): 56 data bytes
64 bytes from 10.0.0.1: seq=0 ttl=64 time=0.034 ms
64 bytes from 10.0.0.1: seq=1 ttl=64 time=0.041 ms
64 bytes from 10.0.0.1: seq=2 ttl=64 time=0.040 ms

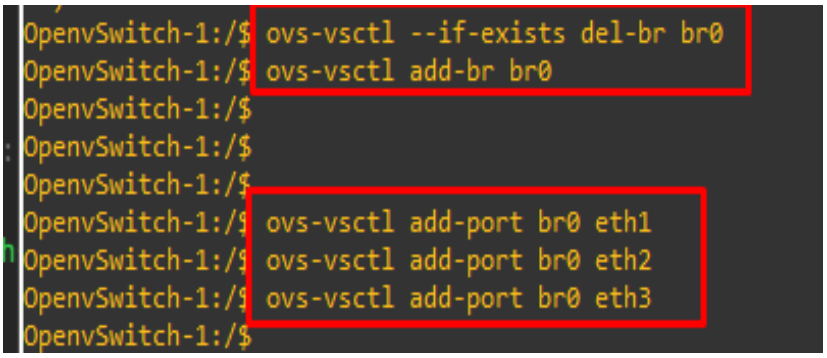
--- 10.0.0.1 ping statistics ---
```

3.5.4.2. Creación del bridge OVS y asignación de interfaces

Sobre cada Open vSwitch se crea el bridge lógico br0, que constituye el datapath controlado por OpenFlow. A continuación, se incorporan las interfaces físicas correspondientes al plano de datos. La Figura 29 muestra la ejecución de los comandos `ovs-vsctl add-br br0` y `ovs-vsctl add-port br0 ethX`, así como la salida de `ovs-vsctl show`, que confirma la conformación del bridge con sus puertos asociados.

Figura 29

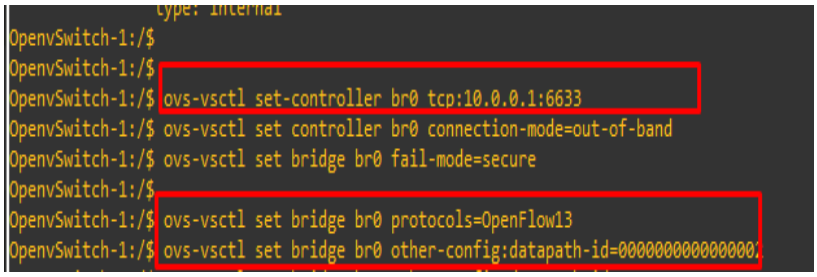
Creación del bridge br0 y asignación de interfaces en los Open vSwitch



Para garantizar la compatibilidad con el controlador y aprovechar las capacidades extendidas del protocolo, cada bridge se configura para operar con OpenFlow 1.3. Además, se asigna un identificador de datapath (DPID) único por nodo y se establece la conexión TCP persistente con el controlador en el puerto 6633. La Figura 30 ilustra la definición del protocolo OpenFlow 1.3, la asignación del DPID de cada conmutador y la confirmación del enlace con el controlador mediante el indicador is_connected: true.

Figura 30

Definición del protocolo OpenFlow 1.3, identificador de cada Open vSwitch y conexión al controlador



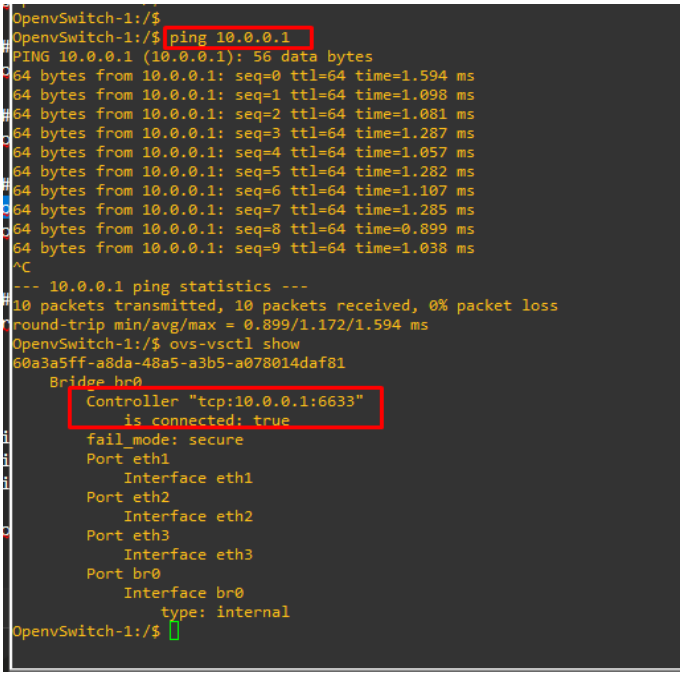
3.5.4.3. Verificación de la conexión entre el controlador y los Open vSwitch

Para corroborar la operatividad del enlace de control, se ejecuta una prueba de conectividad ICMP desde el controlador SDN hacia cada Open vSwitch. Esta verificación asegura que el plano de control puede intercambiar mensajes OpenFlow con los conmutadores

del plano de datos sin pérdida de paquetes. La Figura 31 documenta el resultado satisfactorio de la prueba.

Figura 31

Verificación de la conexión entre el controlador SDN y los Open vSwitch mediante ping

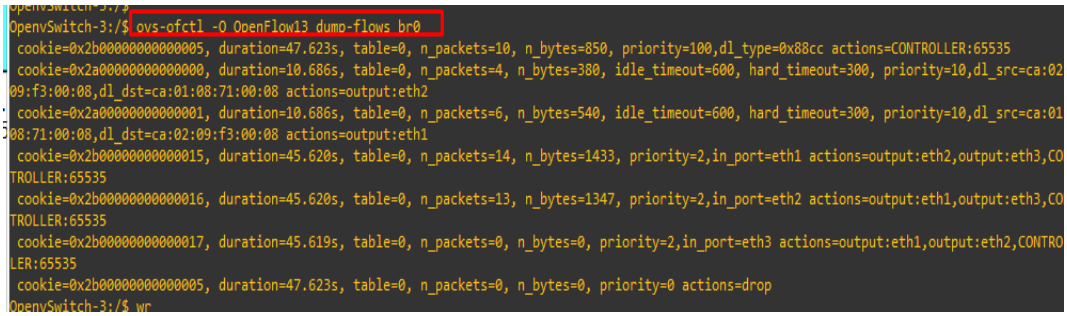


3.5.4.4. Instalación de flujos OpenFlow para tráfico OSPF

Dado que OSPF emplea los grupos multicast 224.0.0.5 y 224.0.0.6, es necesario instalar en cada Open vSwitch flujos específicos que permitan la propagación de los mensajes Hello y Link-State Advertisement (LSA) entre los routers de borde. La Figura 32 muestra los flujos OpenFlow agregados con prioridad elevada para que el tráfico OSPF atraviese el plano de datos sin ser descartado por la regla por defecto.

Figura 32

Flujos OpenFlow que permiten la circulación de OSPF a través de los Open vSwitch



3.5.5. Verificación del despliegue del servicio de streaming

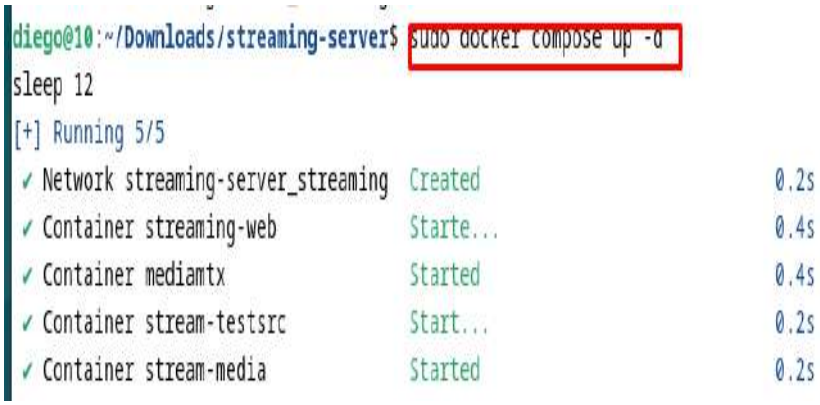
El servicio de streaming multimedia se despliega como un contenedor Docker dentro de la red SDN, mientras que el cliente legítimo se ubica en otro nodo de la topología para forzar el tránsito del flujo por el plano de datos. Las siguientes figuras documentan la creación del contenedor del servidor y la validación de la conectividad de extremo a extremo entre el cliente y el servidor.

3.5.5.1. Despliegue del contenedor del servidor de streaming

El contenedor del servidor MediaMTX se construye a partir de la imagen oficial bluenviron/mediamtx, complementada con un servicio FFmpeg que publica de forma continua los archivos multimedia almacenados en el volumen ./media. La Figura 33 muestra el resultado del comando `docker compose up -d` en el nodo servidor, en el que se observan activos los servicios mediamtx, stream-media, stream-testsrc y streaming-web.

Figura 33

Creación y despliegue del contenedor que ejecuta el servidor de streaming

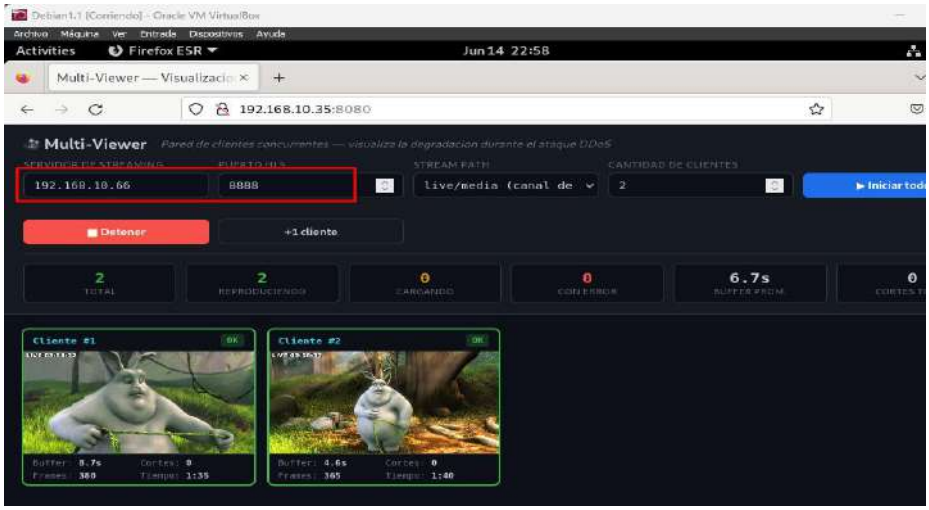


3.5.5.2. Ejecución del cliente multi-viewer y verificación del flujo a través de la SDN

El cliente multi-viewer se despliega en una máquina virtual independiente, conectada a una LAN distinta de la del servidor. El acceso al stream se realiza mediante HLS por el puerto 8888, lo que obliga a los paquetes a atravesar los routers de borde y los Open vSwitch del plano de datos. La Figura 34 muestra la interfaz del multi-viewer reproduciendo simultáneamente varias sesiones del stream live/media.

Figura 34

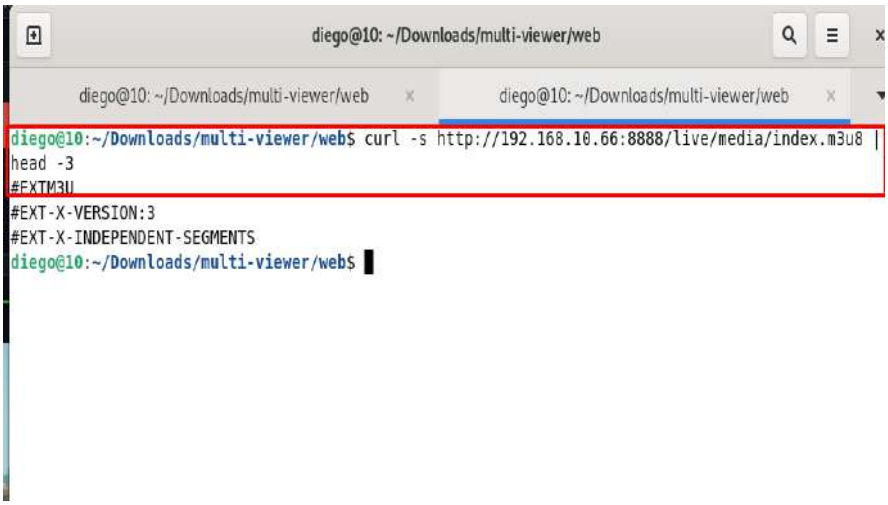
Ejecución del cliente multi-viewer y verificación del consumo del stream a través de la red SDN



Además, se verifica desde el cliente la resolución de la URL del manifest HLS y la recepción de los segmentos publicados por el servidor. La Figura 35 evidencia la respuesta del servidor MediaMTX a las solicitudes generadas desde la red del cliente, lo que confirma la conectividad de extremo a extremo del servicio.

Figura 35

Verificación de la conectividad entre el servidor de streaming y el multi-viewer desde la red del cliente



3.5.6. Implementación del sistema de monitoreo con ntopng

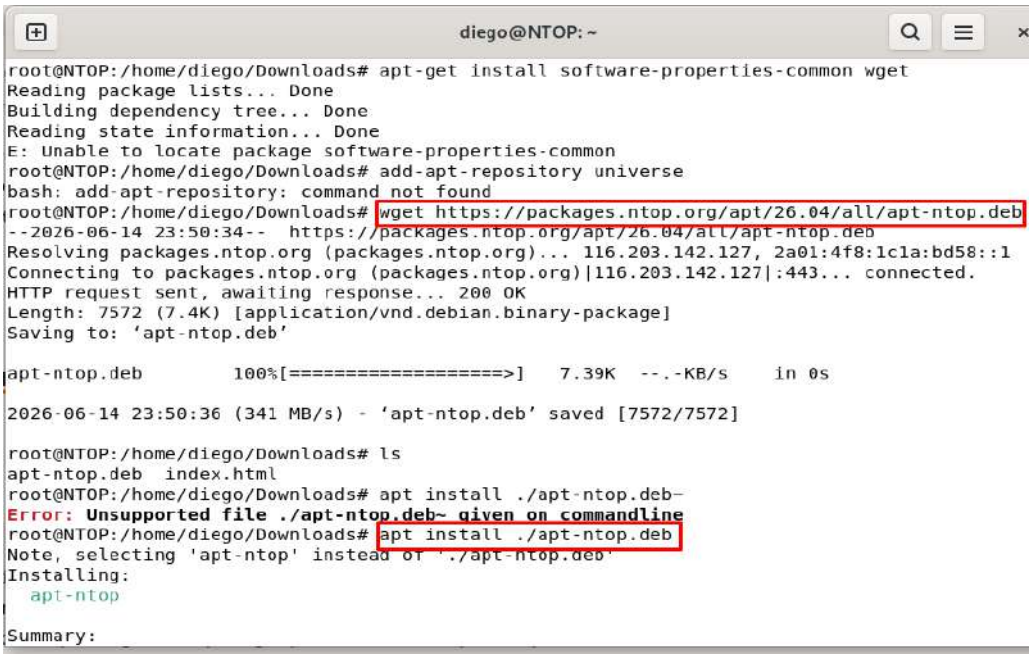
La integración de ntopng en el entorno de pruebas comprende cuatro fases: la descarga e instalación del software, la solicitud de la licencia educativa, la configuración de la interfaz de monitoreo y la habilitación del port mirroring desde los Open vSwitch hacia la máquina virtual que aloja la herramienta.

3.5.6.1. Descarga e instalación de ntopng

La instalación se realiza sobre una máquina virtual Debian 12 dedicada, conectada a la red de gestión 10.0.0.0/24. Se utilizan los paquetes oficiales del repositorio de ntop, lo que permite recibir actualizaciones de seguridad y disponer de la versión Community con las funcionalidades requeridas [52]. La Figura 36 documenta el proceso de descarga e instalación del software.

Figura 36

Descarga e instalación de ntopng en la máquina virtual de monitoreo

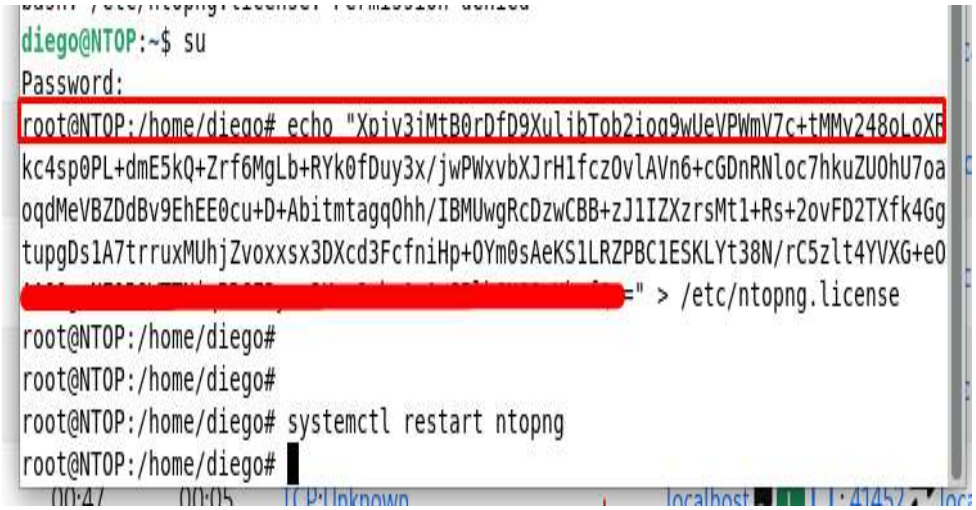


3.5.6.2. Solicitud y activación de la licencia educativa

Ntopng ofrece una licencia educativa gratuita orientada a la investigación y la docencia, que se solicita mediante el correo institucional. Esta licencia habilita módulos de detección de anomalías, históricos extendidos y soporte completo de sFlow sin restricciones en el número de flujos. La Figura 37 muestra la activación de la licencia educativa para este trabajo de titulación.

Figura 37

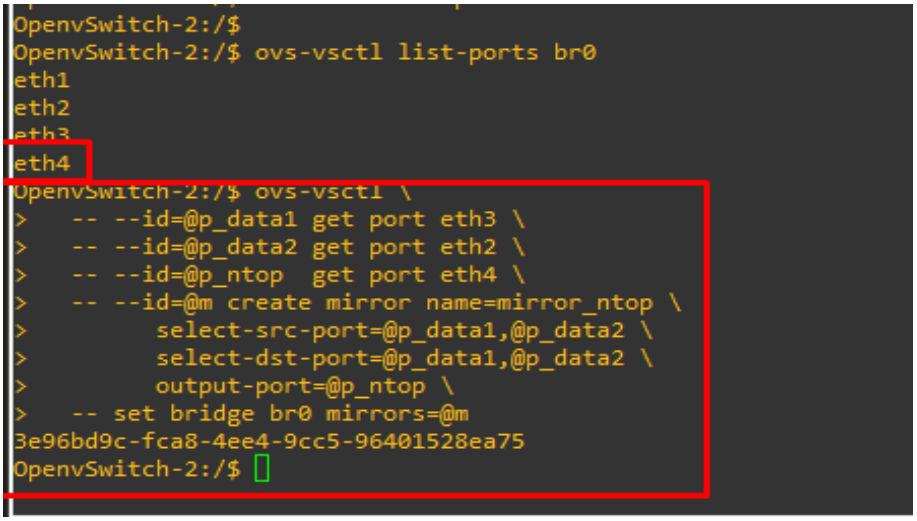
Activación de la licencia educativa de ntopng mediante correo institucional



A la máquina virtual de ntopng se añade una segunda interfaz de red (eth4), destinada exclusivamente a recibir el tráfico replicado por los Open vSwitch. Esta separación permite que la gestión del nodo y el flujo de datos analizado viajen por interfaces distintas, lo que evita interferencias en la medición. La Figura 38 ilustra la configuración aplicada a la interfaz eth4 para habilitarla como entrada de port mirroring.

Figura 38

Configuración de la interfaz eth4 de ntopng como puerto de espejo (mirror)

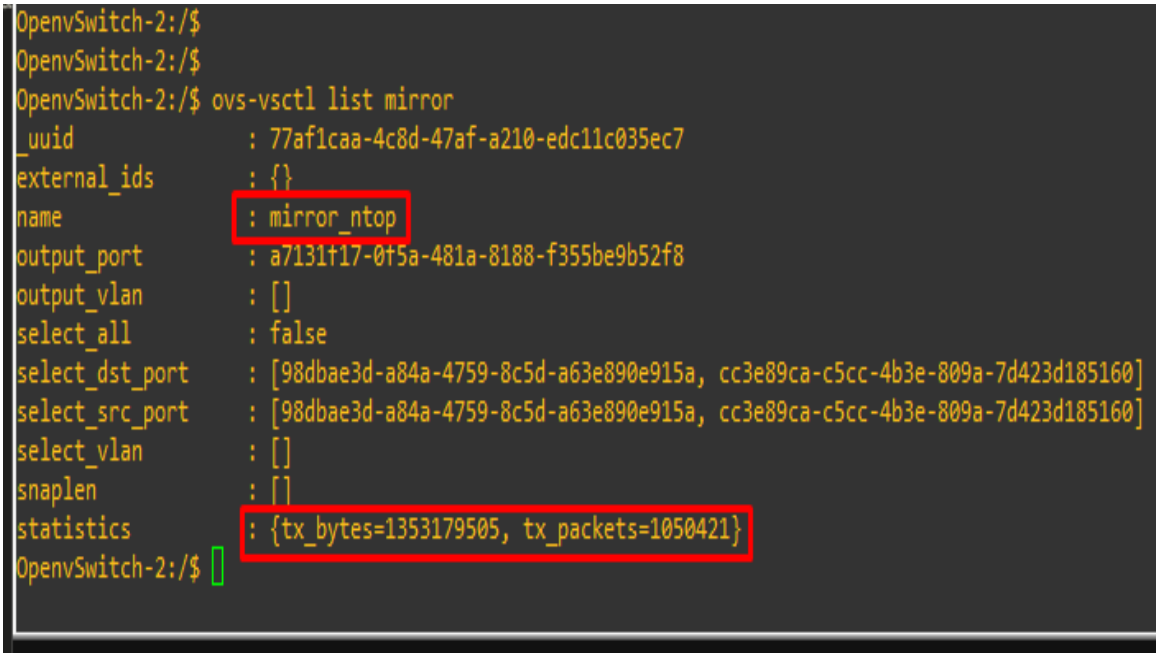


3.5.6.3. Configuración del port mirroring en los Open vSwitch

Para que ntopng disponga de visibilidad completa del tráfico que circula por el plano de datos, se configura una sesión de port mirroring en cada Open vSwitch del backbone SDN. La sesión replica el tráfico de entrada y salida de los puertos del datapath hacia la interfaz eth4 de la máquina virtual de ntopng. La Figura 39 muestra los comandos `ovs-vsctl` ejecutados para crear la entidad mirror y asociarla al bridge br0.

Figura 39

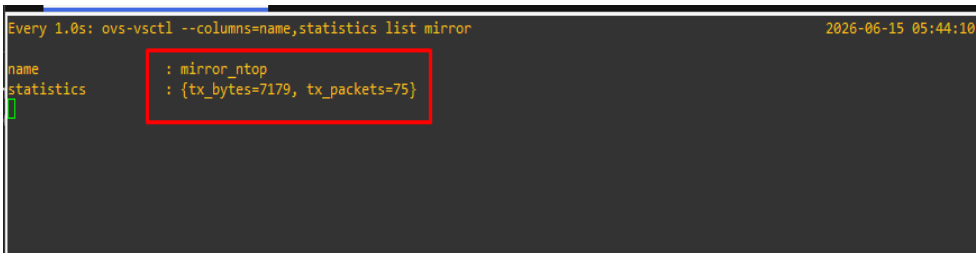
Configuración del port mirroring en el Open vSwitch hacia la máquina virtual de ntopng



A modo de verificación, se inspecciona la configuración resultante mediante *`ovs-vsctl list mirror`*, que muestra la entidad mirror con sus puertos de origen y destino asociados. La Figura 40 evidencia esta configuración en el Open vSwitch.

Figura 40

Configuración de port mirroring hacia ntopng en el Open vSwitch

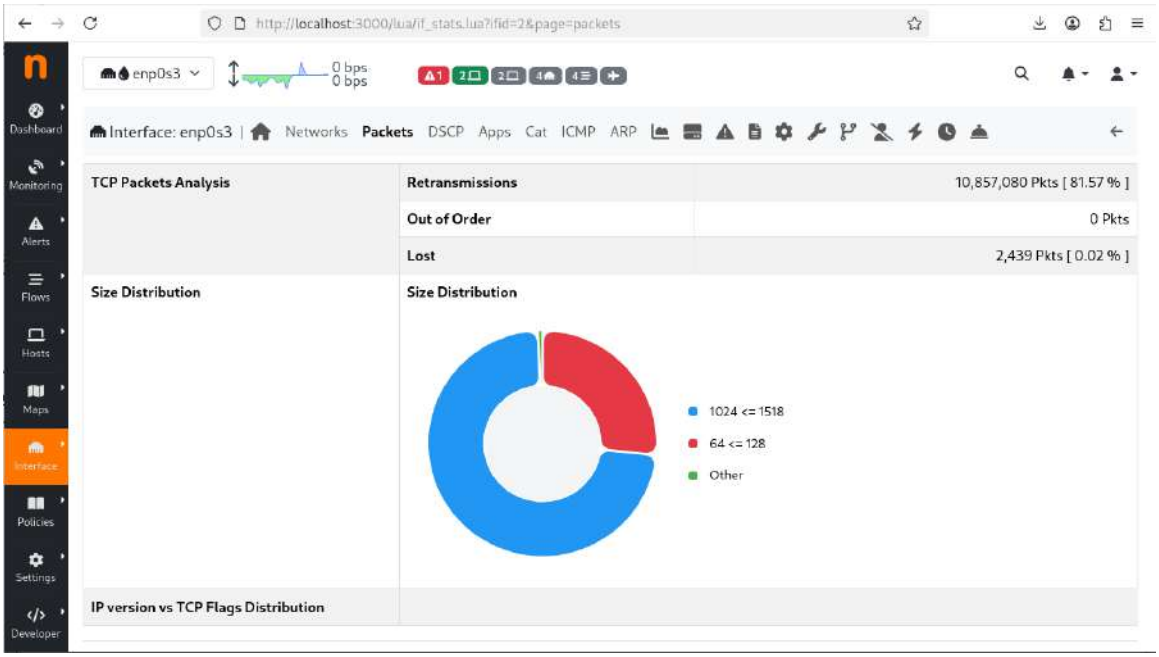


3.5.6.4. Verificación del flujo recibido en ntopng

Una vez activa la sesión de mirroring y configurada la interfaz de recepción, se comprueba que ntopng identifica correctamente el tráfico de la topología. La Figura 41 corresponde al panel principal de la herramienta durante el escenario de línea base, en el que se distinguen los flujos HLS entre el servidor MediaMTX y el cliente multi-viewer y los mensajes de control OSPF entre los routers.

Figura 41

Tráfico de la línea base de la topología visualizado en ntopng



3.6. Procedimiento de captura de datos

Esta sección describe el procedimiento sistemático para registrar las evidencias experimentales de cada corrida. Las capturas se organizan por punto de observación (enlace del plano de datos y servidor de streaming) y se conservan junto con un identificador único de corrida que permite vincular cada archivo con el escenario y el instante en que se genera.

3.6.1. Capturas de tráfico de red (PCAP)

La captura de tráfico constituye la fuente primaria de evidencia para el cálculo de las métricas de QoS. En cada corrida se inicia una instancia de tshark sobre la interfaz del Open vSwitch 1 (eth1), punto por el que transita todo el tráfico entre el cliente legítimo y el servidor de streaming. La captura se almacena en formato pcapng con marca de tiempo de microsegundos y sin límite de tamaño de paquete (snaplen 0), lo que garantiza la integridad de las tramas para el análisis posterior.

Adicionalmente, se ejecuta una segunda instancia de tshark en el servidor MediaMTX para registrar el tráfico entrante al puerto HLS (8888), en el que confluyen las peticiones legítimas y las conexiones maliciosas generadas por el atacante. Los archivos resultantes se nombran siguiendo el patrón captura_<punto>_<escenario>_<corrida>.pcapng y se almacenan en el directorio de la corrida correspondiente. Al finalizar cada captura, se verifica que el archivo tenga un tamaño no nulo y que contenga paquetes dentro de la ventana temporal esperada mediante el comando *capinfos*.

3.6.2. Almacenamiento, nomenclatura y trazabilidad de las capturas

La totalidad de los artefactos generados durante el experimento se organiza en una estructura de directorios jerárquica que facilita la trazabilidad entre cada archivo y el escenario que lo produjo. El directorio raíz del experimento contiene una carpeta por controlador (ODL/ y RYU/) y, dentro de cada una, una carpeta por réplica (run_001/, run_002/ y run_003/). Cada carpeta de réplica contiene las capturas de los cinco escenarios, nombradas según el patrón

E<n>_<descripción>-<controlador>.pcapng (por ejemplo, E1_baseline-ODL.pcapng o E3_tcp_exhaust_2w-RYU.pcapng).

Esta convención de nomenclatura permite localizar cualquier captura sin ambigüedad y facilita la automatización del procesamiento, ya que el script del Anexo F descubre las réplicas a partir de esta estructura.

3.7. Estrategia de escalamiento del experimento

El entorno experimental virtualizado basado en GNS3 impone restricciones de hardware que obligan a reducir la escala de los experimentos respecto a un escenario de referencia de mayor escala. Por ello, el diseño metodológico se concibe en dos fases: una fase a pequeña escala, ejecutada en el laboratorio virtual de GNS3 sobre el equipo anfitrión, y una fase posterior de extrapolación de los resultados hacia escenarios de referencia de mayor escala, que sirve de base para investigaciones futuras. Esta segunda fase corresponde a un escalamiento o extrapolación de los resultados experimentales, y no a una validación experimental en una infraestructura SDN física, que no forma parte del alcance de esta investigación. La metodología de benchmarking de la RFC 2544 [45], que normaliza las pruebas de desempeño de los dispositivos de red throughput, latencia, tasa de pérdida de tramas y ráfagas back-to-back, se emplea para fundamentar la medición y comparación de las métricas, y no para justificar la extrapolación de un entorno virtual hacia una infraestructura productiva. La reducción de escala es una decisión propia del diseño experimental, motivada por las restricciones del equipo anfitrión, y su representatividad se discute como una limitación del estudio.

La estrategia de escalamiento define tres conjuntos de parámetros: el primero determina el número de actores activos en cada corrida (clientes y workers); el segundo, la duración y el número de iteraciones; y el tercero, la calidad del servicio multimedia. Los valores escalados se eligen para no superar el 80 % de utilización sostenida de CPU y memoria RAM del equipo

anfitrión, estimada a partir de los recursos asignados a cada máquina virtual (Tabla 15). Este umbral es un criterio operativo propio, adoptado para evitar que la saturación del anfitrión y no distorsione las mediciones; se aplica como criterio de dimensionamiento al asignar los recursos del entorno. Los valores de referencia de mayor escala, en cambio, se derivan de las recomendaciones de la ITU-T, el IEEE y otras instituciones de normalización citadas en la sección siguiente, por lo que constituyen parámetros de referencia y no mediciones en una infraestructura real. La proyección hacia escenarios de mayor escala se plantea como una extrapolación que aplica factores de escala a cada parámetro de carga (número de clientes, número de workers atacantes, hilos concurrentes, ancho de banda, CPU y RAM), bajo los supuestos explícitos de proporcionalidad, de conservación de la tasa de bits por cliente y de la relación entre clientes y atacantes observada en el laboratorio. Las métricas de QoS proyectadas (throughput, jitter, latencia y pérdida) se estiman aplicando a la línea base los factores de degradación registrados experimentalmente, sin suponer linealidad estricta fuera del rango medido. Formalmente, para cada métrica de QoS (M) y cada nivel de intensidad de ataque (k), la extrapolación se apoya en el factor de degradación relativo medido en el laboratorio, definido como $\delta(M,k) = M_{lab,k(ataque)} \div M_{lab,k(línea\ base)}$; dicho factor se aplica a un valor base de referencia de mayor escala para estimar la métrica correspondiente, según $M_{ref,k(ataque)} \approx M_{ref(línea\ base)} \times \delta(M,k)$. Con esta formulación, del laboratorio al escenario de referencia no se transfiere el valor absoluto de la métrica, sino el factor de degradación relativo, bajo el supuesto de que dicho factor se conserva de forma aproximada al variar la escala. Este modelo no afirma que el porcentaje de degradación sea idéntico ni garantizado a mayor escala: constituye una estimación de primer orden, válida dentro del rango de carga efectivamente medido (escenarios E1 a E5), cuya exactitud disminuye a medida que los parámetros del escenario de referencia se alejan de dicho rango. Los factores de escala de la Tabla 28 (por ejemplo, ×25 en el número de clientes o de ×10 a ×100 en el ancho de banda) describen los

parámetros de carga del escenario de referencia, pero no constituyen por sí mismos el modelo de extrapolación de las métricas, que se sustenta en los factores de degradación δ(M,k).

En términos operativos, la estrategia de escalamiento se resume en cuatro aspectos. Qué se escala: el número de clientes legítimos, el número de workers atacantes, los hilos concurrentes, el ancho de banda del enlace, los recursos de cómputo (CPU y memoria RAM), el tamaño de los búferes, la tasa de muestreo del monitoreo y la resolución y la tasa de bits del stream. Cómo se escala: aplicando a cada parámetro los factores de la Tabla 28 y proyectando las métricas de QoS mediante el factor de degradación relativo, sin ejecutar nuevas mediciones. Qué se conserva: la proporción entre recursos y carga, la tasa de bits por cliente, la relación entre clientes legítimos y nodos atacantes, los patrones característicos del tráfico HLS (distribución de tamaños de paquete y entrega en ráfagas), el códec, la topología lógica y el tipo de ataque (tcp_exhaust). Qué no se conserva ni se garantiza: los valores absolutos de las métricas, la linealidad fuera del rango medido, el comportamiento de la CPU y la memoria a gran escala y los efectos de red ausentes en el laboratorio (congestión WAN, latencia geográfica); tampoco existe validación empírica en una infraestructura de mayor escala. Esta distinción delimita el alcance de las proyecciones y sus limitaciones metodológicas, que se detallan en el capítulo de resultados.

3.8. Normativa de los parámetros experimentales

La duración de las corridas, el número de iteraciones y los umbrales de QoS se seleccionan con base en estándares y recomendaciones internacionales, que proporcionan criterios de referencia para configurar los parámetros experimentales. La Tabla 27 recoge la trazabilidad entre cada parámetro experimental y la norma o recomendación de referencia.

Tabla 27

Justificación normativa de los parámetros experimentales

Parámetro	Estándar / Recomendación	Justificación
Duración por corrida	RFC 2544 / RFC 6349	La RFC 2544 establece una duración mínima de 60 s por prueba de throughput y pérdida, y la RFC 6349 exige que las pruebas TCP duren lo suficiente para superar el inicio lento y estabilizarse. Con base en estos criterios, se adopta una ventana de medición de 300 s (criterio propio), que supera el mínimo de 60 s y permite observar la degradación sostenida del streaming.
Número de iteraciones (réplicas)	Criterio propio (marco estadístico de la ITU-T E.840)	Se ejecutan 3 réplicas independientes por escenario como criterio experimental propio para estimar la variabilidad entre corridas y reportar la media y la desviación estándar. La ITU-T E.840 proporciona el marco estadístico para comparar los resultados.
Umbrales de RTT, jitter, pérdida y throughput	ITU-T Y.1541 / ITU-T G.1010	La ITU-T Y.1541 define las clases de QoS para redes IP (clases 0 y 1: IPTD ≤ 100 ms y ≤ 400 ms en un solo sentido; IPDV ≤ 50 ms; IPLR ≤ 1×10^{-3}); en este trabajo, el

Parámetro	Estándar / Recomendación	Justificación
		umbral de 100 ms se aplica como criterio conservador al RTT. La ITU-T G.1010 clasifica los servicios multimedia según su tolerancia al retardo y a las pérdidas desde la perspectiva del usuario.

3.9. Parámetros escalados frente a parámetros de referencia

A partir de los estándares descritos, la Tabla 28 presenta la equivalencia entre los valores adoptados en la fase experimental y los valores de referencia de un escenario de mayor escala. Estos valores de referencia se derivan de los informes anuales de tráfico de Cisco y de la documentación de operadores comerciales de streaming [66]. Esta correspondencia permite extrapolar los hallazgos del laboratorio hacia escenarios de referencia de mayor tamaño, entendida como una estimación y no como una medición en dichos escenarios.

Tabla 28

Comparativa de parámetros escalados (entorno experimental) frente a parámetros de referencia de mayor escala

Parámetro	Valor experimental (escalado)	Valor de referencia (mayor escala)	Factor de escala	Justificación
Clientes legítimos	4 viewers	100 viewers	× 25	100 viewers × 5 Mbps (HD 1080p) ≈ 500–520 Mbps, que cabe en un

Parámetro	Valor experimental (escalado)	Valor de referencia (mayor escala)	Factor de escala	Justificación
				enlace de 1 Gbps. Netflix recomienda “5 Mbps o más” para HD
Workers atacantes	1 a 4 workers	50 a 200 nodos	× 50	Las botnets reales son mucho mayores: Mirai llegó a unas 600 000 infecciones. [73]
Duración de cada corrida	300 s (5 min)	300 s (5 min)	× 1	RFC 2544, sección 24: mínimo de 60 s por prueba [45]. Los 300 s ya figuran como criterio propio.
Ancho de banda del enlace	100 Mbps virtual	1–10 Gbps	× 10 hasta × 100	La guía de Red Hat pide como mínimo 2×1Gbps por nodo controlador [74].
RAM controlador	3 GB	32 GB	× 10	Red Hat OSP [74], mínimo de 20 GB, con 1.5 GB por núcleo como mínimo y 3 GB por núcleo recomendados. Con 8–16 núcleos salen

Parámetro	Valor experimental (escalado)	Valor de referencia (mayor escala)	Factor de escala	Justificación
				12–48 GB, así que 32 GB queda dentro del rango.

La adopción de estos factores de escala busca que el equipo anfitrión, que dispone de 32 GB de RAM y 12 hilos de CPU, opere por debajo del 80 % de utilización durante las corridas, para evitar saturaciones del propio entorno de emulación que puedan contaminar las mediciones. Los resultados obtenidos a escala reducida se interpretan como indicadores del comportamiento del sistema bajo ataque; su extrapolación a mayor escala se sustenta en el modelo de factores de degradación descrito, y no en una hipótesis de linealidad atribuible a la RFC 2544, que regula la metodología de medición y no el escalamiento de infraestructuras.

3.10. Parámetros del servicio de streaming multimedia

El servicio MediaMTX se configura con parámetros específicos de códec, resolución y tasa de bits, ajustados al esquema de escalamiento general. La Tabla 29 detalla la configuración del laboratorio frente a la configuración equivalente de un escenario de referencia de mayor escala de streaming OTT (Over-the-Top), con valores tomados de las recomendaciones de Netflix Open Connect [75]. En el flujo experimental, el origen de video se publica una sola vez en el servidor MediaMTX mediante RTSP (8554/TCP), y MediaMTX lo reexpone por HLS sobre HTTP en el puerto 8888/TCP, por el que el cliente multi-viewer consume el video. El ataque tcp_exhaust se dirige a ese mismo puerto: abre y mantiene numerosas conexiones TCP contra el servicio HLS (8888/TCP) y agota la capacidad del servidor para atender conexiones, de modo que compite directamente con las solicitudes del cliente legítimo. Como el tráfico

legítimo y el malicioso comparten puerto y protocolo, el ataque no puede bloquearse por puerto sin interrumpir el servicio. Tanto el tráfico legítimo como el de ataque se cursan íntegramente sobre TCP, sin intervención de RTP/UDP en las mediciones. Asimismo, la Recomendación ITU-T P.1203 [76] se utiliza como modelo de evaluación de la calidad de experiencia para streaming adaptativo, y no como una norma que fije una tasa de 5 Mbps: dicho valor corresponde a una configuración experimental y no a una prescripción normativa.

Tabla 29

Parámetros del servicio de streaming en el entorno experimental frente a un escenario de referencia de mayor escala

Parámetro	Experimental (escalado)	Referencia (mayor escala)	Norma de referencia
Resolución de video	480p (854×480)	1080p (1920×1080)	ITU-R BT.709
Códec de video	H.264 Baseline	H.264 High / H.265	ITU-T H.264 / H.265
Tasa de bits de video	800 kbps	5000 kbps	Configuración experimental; ITU-T P.1203 como modelo de evaluación
Frame rate	25 fps	30 / 60 fps	ITU-R BT.709 Sec.4
Códec de audio	AAC-LC	AAC-LC / E-AC-3	ISO/IEC 14496-3
Tasa de bits de audio	128 kbps	192–256 kbps	ETSI TS 102 366

Parámetro	Experimental (escalado)	Referencia (mayor escala)	Norma de referencia
Frecuencia de muestreo	48 kHz	48 kHz	AES3
Protocolo de transporte	HLS sobre HTTP/1.1	HLS / DASH sobre HTTP/2	RFC 8216 / ISO/IEC 23009-1
Duración del segmento HLS	4 s	2–6 s	RFC 8216 Sec-4.3
Cantidad de clientes simultáneos	4 viewers	100 viewers	Criterio propio (Tabla 28)

La configuración escalada se selecciona porque conserva los patrones de tráfico HLS característicos de escenarios de mayor escala: una distribución bimodal de tamaños de paquete (paquetes pequeños de control TCP y paquetes de tamaño MTU para los segmentos), una entrega en ráfagas con periodicidad de 2 a 6 s por segmento y una proporción de audio respecto al video del orden del 16 % [77]. Estas características se conservan al reducir la tasa absoluta de bits, lo que permite extrapolar las observaciones de QoS del entorno experimental hacia escenarios de referencia de mayor escala, con las limitaciones propias de una estimación no validada empíricamente.

3.11. Plan de experimentación

Una vez establecida la infraestructura del laboratorio, el plan de experimentación define cómo se induce de forma controlada la degradación del servicio de streaming alojado en la red SDN y cómo se cuantifica su efecto sobre la calidad del servicio. El diseño experimental adopta un enfoque de manipulación controlada de dos variables independientes: el controlador SDN (OpenDaylight o Ryu) y la intensidad del ataque, expresada en número de workers activos. Las

variables dependientes son las cuatro métricas de QoS descritas en la sección “Métricas de QoS para evaluar el impacto del ataque DDoS” (latencia, jitter, pérdida de paquetes y throughput), mientras que la topología, el direccionamiento, el códec del stream, los recursos asignados a cada máquina virtual y el modo de ataque (tcp_exhaust) se mantienen como variables controladas.

3.11.1. Variables del experimento

El diseño experimental considera tres tipos de variables, cuya clasificación se mantiene en todas las corridas para asegurar la trazabilidad de los resultados:

Variables independientes (manipuladas por el investigador): el controlador SDN (OpenDaylight o Ryu) y la intensidad del ataque tcp_exhaust, expresada en número de workers activos (0, 1, 2, 3 o 4, con 20 hilos concurrentes por worker). El diseño resultante es factorial 2 × 5 (controlador × intensidad). Estas variables definen las condiciones experimentales y son las únicas que cambian entre escenarios.

Variables dependientes (observadas y medidas): latencia (RTT) en milisegundos, jitter en milisegundos, porcentaje de pérdida de paquetes y throughput efectivo entregado al cliente en megabits por segundo. Cada métrica se captura según los procedimientos descritos en la sección “Métricas de QoS para evaluar el impacto del ataque DDoS”, con el alcance precisado en la subsección “Consideraciones sobre las métricas y los puntos de medición”, y se contrasta con los umbrales de referencia de la Recomendación ITU-T Y.1541.

Variables controladas (mantenidas constantes): la topología física y lógica de la red, el direccionamiento IPv4, el códec utilizado en el stream, la versión del protocolo OpenFlow (1.3), los recursos asignados a cada máquina virtual (CPU, RAM, almacenamiento) y la duración nominal de cada corrida. La invariabilidad de estas variables busca minimizar el efecto de factores de confusión, de modo que las diferencias observadas entre escenarios puedan atribuirse principalmente a las variables independientes manipuladas. Debe

reconocerse, no obstante, que en un entorno virtualizado persisten factores no controlados —la carga del sistema operativo anfitrión, el planificador de CPU, la propia capa de virtualización, el jitter temporal y el procesamiento interno de Open vSwitch y del controlador—, cuyo efecto residual se acota mediante la repetición de corridas y el reporte de su variabilidad.

3.11.2. Escenarios experimentales

Se definen cinco escenarios de prueba que abarcan desde la operación normal de la red hasta la saturación máxima del servidor de streaming. El escenario E1 (línea base) corresponde a la red sin ningún tipo de ataque, con el cliente legítimo activo, y proporciona los valores de referencia para cuantificar la magnitud de la degradación en los demás escenarios. Los escenarios E2, E3, E4 y E5 inyectan el ataque tcp_exhaust con intensidad creciente (1, 2, 3 y 4 workers, respectivamente), lo que permite evaluar la respuesta del servicio de streaming con ambos controladores ante cargas progresivas. Todos los escenarios con ataque son distribuidos (DDoS), porque cada worker es un nodo atacante independiente coordinado desde el servidor C2; lo único que los diferencia es el número de nodos activos. La Tabla 30 resume los escenarios definidos.

Tabla 30

Escenarios experimentales definidos para la evaluación de la QoS

ID	Escenario	Modo de ataque	Configuración del ataque	Duración por réplica
E1	Línea base sin ataque	—	0	300 s
E2	Ataque tcp_exhaust con 1 worker	tcp_exhaust (HLS, 8888/TCP)	1 worker (20 hilos)	300 s

ID	Escenario	Modo de ataque	Configuración del ataque	Duración por réplica
E3	Ataque tcp_exhaust con 2 workers	tcp_exhaust (HLS, 8888/TCP)	2 workers (20 hilos c/u)	300 s
E4	Ataque tcp_exhaust con 3 workers	tcp_exhaust (HLS, 8888/TCP)	3 workers (20 hilos c/u)	300 s
E5	Ataque tcp_exhaust de máxima intensidad	tcp_exhaust (HLS, 8888/TCP)	4 workers (20 hilos c/u)	300 s

Nota. La definición de worker se presenta en la sección "Bloque de ataques DDoS" del diseño de la topología de red.

La elección de 1 a 4 workers no es arbitraria: 1 worker representa un atacante único de bajo perfil; 2 workers, la colaboración de dos fuentes simultáneas; 3 workers, un escenario de múltiples atacantes coordinados; y 4 workers, el límite máximo configurado en el sistema TrafficGen SDN, en el que se espera una degradación severa del servicio de streaming. Cada escenario se ejecuta con tres réplicas independientes para reducir el efecto de mediciones atípicas y calcular promedios y desviaciones estándar; este número de réplicas es un criterio experimental propio para estimar la variabilidad entre corridas. La Tabla 31 presenta la matriz de configuraciones experimentales.

Tabla 31

Matriz comparativa de configuraciones experimentales

ID	Escenario	Controlador	Modo de ataque	Intensidad	Réplicas
E1-ODL	Línea base	OpenDaylight	—	—	3
E1-Ryu	Línea base	Ryu	—	—	3

ID	Escenario	Controlador	Modo de ataque	Intensidad	Réplicas
E2-ODL	Ataque bajo	OpenDaylight	tcp_exhaust (HLS, 8888/TCP)	1 worker · 20 hilos	3
E2-Ryu	Ataque bajo	Ryu	tcp_exhaust (HLS, 8888/TCP)	1 worker · 20 hilos	3
E3-ODL	Ataque medio	OpenDaylight	tcp_exhaust (HLS, 8888/TCP)	2 workers · 20 hilos	3
E3-Ryu	Ataque medio	Ryu	tcp_exhaust (HLS, 8888/TCP)	2 workers · 20 hilos	3
E4-ODL	Ataque alto	OpenDaylight	tcp_exhaust (HLS, 8888/TCP)	3 workers · 20 hilos	3
E4-Ryu	Ataque alto	Ryu	tcp_exhaust (HLS, 8888/TCP)	3 workers · 20 hilos	3
E5-ODL	Ataque máximo	OpenDaylight	tcp_exhaust (HLS, 8888/TCP)	4 workers · 20 hilos	3
E5-Ryu	Ataque máximo	Ryu	tcp_exhaust (HLS, 8888/TCP)	4 workers · 20 hilos	3

Nota. La matriz se compone de 10 configuraciones distintas (5 escenarios × 2 controladores) y cada una se ejecuta con 3 réplicas independientes, lo que da un total de 30 corridas experimentales. Todas comparten la misma topología, el direccionamiento, los clientes finales y el servidor MediaMTX descritos en la sección “Diseño del sistema”.

3.11.3. Procedimiento de ejecución por corrida

Cada corrida experimental sigue un esquema de siete pasos que favorece la repetibilidad de las mediciones y permite trazar la evolución del sistema a lo largo del experimento. El procedimiento es idéntico para ambos controladores, de modo que las

diferencias en los resultados puedan atribuirse al cambio de controlador y no al protocolo de ejecución.

- Paso 1: preparación del entorno. Se inicia GNS3 y se levanta la topología completa: el controlador SDN seleccionado, los Open vSwitch, los routers, los clientes finales, el servidor MediaMTX y los nodos de la botnet TrafficGen SDN. Se verifica la conectividad de extremo a extremo mediante ping bidireccional entre el cliente y el servidor.
- Paso 2: calibración de la línea base. Se ejecuta una prueba de 60 s sin ataque y se verifica que las cuatro métricas de QoS sean estables y comparables con las de la línea base de referencia; en caso contrario, se reinicia el entorno y se repite la calibración.
- Paso 3: inicio de la captura. Se inician simultáneamente las capturas de tshark en los puntos definidos en la sección “Capturas de tráfico de red (PCAP)”, ntopng y el dashboard de TrafficGen SDN, y se registra la marca de tiempo de inicio.
- Paso 4: ejecución del ataque. Se emite el comando C2 desde el dashboard según el escenario activo (intensidad del ataque) y se mantiene en paralelo el flujo HLS legítimo hacia el cliente.
- Paso 5: ventana de medición. El ataque se mantiene activo durante una ventana de 300 s (5 min), que se considera suficiente para alcanzar un régimen estacionario en las métricas y capturar los efectos transitorios del inicio del ataque.
- Paso 6: finalización del ataque y de la captura. Se detiene el envío de tráfico malicioso desde el dashboard, se espera un intervalo de 30 s para el vaciado de los búferes y se cierran las capturas de tshark, ntopng y el dashboard.

- Paso 7: almacenamiento y limpieza. Se exportan los archivos pcap, los CSV del dashboard y las métricas nativas de MediaMTX. Antes de la siguiente corrida, se reinician las tablas de flujo de los Open vSwitch y el controlador SDN para asegurar un estado limpio.

Para evitar ambigüedades, se distinguen tres intervalos temporales: (i) la calibración de la línea base (paso 2), de 60 s, que solo verifica la estabilidad del entorno y no se incluye en el análisis; (ii) la ventana de medición (paso 5), de 300 s, que corresponde a la captura analizada en cada escenario, sin ataque en E1 y con el ataque activo en E2 a E5; y (iii) el intervalo de vaciado de 30 s posterior al ataque (paso 6), que tampoco se analiza. La duración nominal de cada corrida analizada es de 300 s, valor que coincide con las Tablas 28 y 30; las capturas registran duraciones efectivas de entre 302 y 326 s por las tolerancias de inicio y cierre de las herramientas de captura.

3.12. Procesamiento y análisis de datos

Esta sección describe el procesamiento y el análisis de los datos obtenidos en las corridas experimentales. En primer lugar, se presenta la forma en que se almacenan las capturas de tráfico, organizadas por controlador y número de corrida según la nomenclatura definida. A continuación, se explica cómo se extrae cada métrica de QoS a partir de esos archivos.

3.12.1. Extracción de métricas de calidad de servicio a partir de las capturas de tráfico

Esta sección describe el procedimiento para obtener las cuatro métricas de calidad de servicio (throughput, latencia, jitter y pérdida de paquetes) a partir de los archivos de captura en formato pcapng. A modo de ejemplo, se ilustra el proceso con la captura de la línea base del controlador OpenDaylight (ODL/run_001/E1_baseline-ODL.pcapng). El procedimiento se replica de manera idéntica en los cinco escenarios experimentales y con ambos controladores (OpenDaylight y Ryu); solo cambia el archivo de captura de entrada. Cada escenario se ejecuta

en tres corridas (carpetas run_001 a run_003); el script procesa cada corrida por separado y promedia los resultados: las series por segundo se promedian segundo a segundo sobre la duración común y los valores globales se reportan como media ± desviación estándar entre corridas. El Anexo F presenta el script completo que automatiza la extracción de las cuatro métricas y la generación de las gráficas.

3.12.1.1. Extracción del throughput efectivo

El throughput efectivo se define como la cantidad de bits por segundo que el servidor de streaming entrega al cliente legítimo; corresponde al caudal útil del servicio y excluye el tráfico de ataque y el resto del tráfico que atraviesa el enlace observado. Para calcularlo se utiliza la herramienta de línea de comandos tshark (motor de Wireshark) [51], que genera estadísticas de entrada/salida agrupadas en intervalos definidos por el usuario.

El comando empleado para obtener el throughput en intervalos de un segundo es el siguiente:

```
tshark -r E1_baseline-ODL.pcapng -q -z \
 "io,stat,1,frame,ip.src==192.168.10.66 && ip.dst==192.168.10.35 && (tcp.srcport==8554 ||
tcp.srcport==8888)"
```

Donde:

- -r E1_baseline-ODL.pcapng: especifica el archivo de captura de entrada correspondiente al escenario de línea base.
- -q: suprime la salida de paquetes individuales y muestra únicamente las estadísticas agregadas.
- -z "io,stat,1,frame,<filtro>": activa el módulo de estadísticas de entrada/salida con intervalos de 1 segundo y obtiene dos columnas: todas las tramas del punto de captura (frame) y las tramas que el servidor (192.168.10.66) envía al cliente legítimo (192.168.10.35) desde los puertos del servicio de streaming (8554 y 8888).

La salida del comando es una tabla con el intervalo temporal y, para cada columna, la cantidad de tramas y el volumen en bytes; el throughput efectivo se obtiene de la columna filtrada. El throughput en megabits por segundo se calcula con la ecuación (5):

$$\text{Throughput (Mbps)} = (\text{Bytes} \times 8) \,/\, 1\,000\,000 \qquad (5)$$

Este cálculo se aplica a cada intervalo de un segundo y genera una serie temporal que muestra la evolución del caudal a lo largo de la corrida. Los valores resultantes se exportan a un archivo CSV, a partir del cual se generan las gráficas con la biblioteca matplotlib de Python.

La primera columna proporciona, además, el throughput agregado del punto de captura, que incluye el tráfico de ataque. Este valor se conserva como referencia de la carga total del enlace, pero no se utiliza como métrica de QoS, ya que no representa el caudal útil recibido por el cliente.

3.12.1.2. Extracción de la latencia (RTT TCP)

La latencia se cuantifica mediante el tiempo de ida y vuelta (Round-Trip Time, RTT) de los segmentos TCP, conforme a la definición de la RFC 2681 [54]. Wireshark calcula el RTT TCP como la diferencia temporal entre el envío de un segmento de datos y la recepción del ACK correspondiente, y registra el valor en el campo tcp.analysis.ack_rtt.

El comando empleado para extraer las muestras de RTT es:

```
tshark -r E1_baseline-ODL.pcapng -T fields \
-e frame.time_relative -e tcp.analysis.ack_rtt \
-Y "tcp.analysis.ack_rtt and ip.addr==192.168.10.35"
```

Donde:

- -T fields: selecciona el modo de salida por campos tabulados.
- -e frame.time_relative: exporta el tiempo relativo del paquete respecto al inicio de la captura, en segundos.

- -e tcp.analysis.ack_rtt: exporta el RTT calculado por el disector TCP de Wireshark, en segundos.
- -Y "tcp.analysis.ack_rtt and ip.addr==192.168.10.35": filtra los paquetes del cliente legítimo que contienen una estimación válida de RTT (aquellos en los que Wireshark empareja un segmento con su ACK).

El RTT se expresa originalmente en segundos y se convierte a milisegundos (multiplicando por 1000) para su representación en las gráficas. A partir de la serie de muestras individuales, depurada con el criterio mediana ± 3 MAD (desviación absoluta mediana), se calculan la media aritmética, la mediana, el percentil 95 y el valor máximo, que caracterizan tanto el comportamiento típico como los picos extremos de latencia.

La fórmula del RTT aplicada por Wireshark corresponde a la definición de la RFC 2681, expresada en la ecuación (6):

$$\mathrm{RTT} = T_{\text{recepción, ACK}} - T_{\text{envío, segmento}} \tag{6}$$

3.12.1.3. Extracción del jitter (variación de retardo entre paquetes)

El jitter se calcula conforme al modelo de variación del retardo entre llegadas definido en la RFC 3550 para flujos RTP [55]. Aunque el tráfico de streaming emplea HLS sobre TCP (y no RTP), la métrica de jitter de la RFC 3550 se aplica como indicador de la estabilidad temporal del plano de datos, dado que cuantifica la regularidad con que los paquetes llegan al receptor.

Para la extracción se utiliza tshark con la exportación de los campos de tiempo entre paquetes consecutivos:

```
    tshark -r E1_baseline-ODL.pcapng -T fields \
  -e frame.time_relative -e frame.time_delta_displayed \
  -Y "tcp and ip.addr==192.168.10.35 and (tcp.port == 8554 or
tcp.port == 8888)"
```

Donde:

- -e frame.time_delta_displayed: exporta el intervalo temporal entre el paquete actual y el anterior en la vista filtrada, en segundos.
- -Y "tcp and ip.addr==192.168.10.35 and (tcp.port == 8554 or tcp.port == 8888)": filtra los paquetes TCP del cliente de streaming.

A partir de los intervalos entre paquetes exportados, el jitter se actualiza de forma recursiva con la ecuación (7), que corresponde a la fórmula de la RFC 3550 presentada en la ecuación (2):

$$J(i) = J(i-1) + (|D(i-1, i)| - J(i-1)) / 16 \quad (7)$$

donde D(i−1, i) representa la diferencia entre intervalos consecutivos entre paquetes. Este cálculo se implementa en un script de Python que procesa la salida de tshark, agrupa las muestras por segundo y exporta la media y el valor máximo de cada intervalo a un archivo CSV.

El script de procesamiento (véase el Anexo F) implementa esta lógica de forma recursiva; además, filtra exclusivamente el tráfico del cliente de streaming (puertos TCP 8554 y 8888) y aplica un umbral de pausa de 50 ms para descartar los intervalos atípicos causados por la segmentación HLS. Los valores de jitter se agregan en promedios por segundo para generar la serie temporal graficada.

La implementación completa del cálculo iterativo del jitter, con el filtrado por puertos del cliente y la agregación por segundo, se encuentra en la función extraer_jitter() del script del Anexo F.

3.12.1.4. Extracción de la pérdida de paquetes

La pérdida de paquetes se estima mediante dos métodos, según el tipo de escenario. En la línea base (E1) se utiliza el campo tcp.analysis.lost_segment de Wireshark, que detecta huecos reales en la secuencia TCP. En los escenarios con ataque (E2 a E5) se emplea

tcp.analysis.retransmission, excluidas las retransmisiones espurias (tcp.analysis.spurious_retransmission), con un filtrado por percentil 90 (P90) que descarta el 10 % de los segundos con valores más extremos para reducir el impacto de ráfagas puntuales de retransmisiones.

Los comandos empleados son:

```
# Línea base (E1): segmentos perdidos por segundo
tshark -r E1_baseline-ODL.pcapng -q -z io,stat,1,tcp,tcp.analysis.lost_segment
# Escenarios E2 a E5: retransmisiones no espurias por segundo
tshark -r E2_tcp_exhaust_1w-ODL.pcapng -q \
  -z "io,stat,1,tcp,tcp.analysis.retransmission && !tcp.analysis.spurious_retransmission"
```

La pérdida de paquetes se calcula con la ecuación (8):

$$\text{Pérdida (\%)} = (\text{Paquetes retransmitidos} / \text{Paquetes totales}) \times 100 \quad (8)$$

Esta métrica se calcula de forma global (sobre toda la corrida) y por intervalos de un segundo, lo que permite identificar los periodos de mayor pérdida durante el ataque. Los resultados se contrastan con el umbral de la Recomendación ITU-T Y.1541, que establece una pérdida máxima de 0.1 % (1×10^{-3}) para los servicios de clase 0.

3.12.2. Generación de las gráficas

Las series temporales resultantes de los cuatro procesos de extracción se exportan a archivos CSV etiquetados con el identificador del escenario y del controlador. Mediante la biblioteca numpy de Python se promedian las tres corridas de cada escenario y se obtienen los picos y las desviaciones estándar por escenario y controlador. Finalmente, con la biblioteca matplotlib se generan las gráficas comparativas entre la línea base y cada nivel de ataque, aplicando un formato uniforme que incluye:

- Títulos descriptivos con nombres completos (sin abreviaciones) que identifican la métrica, el controlador y el escenario.
- Escalas idénticas en los ejes Y para gráficas del mismo tipo, lo que permite la comparación visual directa entre escenarios.
- Líneas de referencia horizontales que marcan los umbrales de la recomendación ITU-T Y.1541.
- Leyendas con nomenclatura uniforme y colores consistentes para cada controlador.

El procedimiento descrito se ejecuta de forma automatizada mediante el script extraer_metricas_qos.py, cuyo código fuente completo se incluye en el Anexo F. Este script unifica la extracción de las cuatro métricas, aplica los filtros descritos (mediana ± 3 MAD para el RTT, umbral de pausa para el jitter, P90 para la pérdida en los escenarios de ataque y detección de segmentos perdidos en la línea base) y genera las gráficas estandarizadas. Para cada escenario (E1 a E5) y cada controlador (OpenDaylight y Ryu), el script procesa los archivos pcapng de sus tres corridas, promedia los resultados y los exporta en archivos CSV, lo que permite reproducir los resultados presentados en el capítulo siguiente.

3.12.3. Consideraciones sobre las métricas y los puntos de medición

Las cuatro métricas se obtienen de la captura del punto de observación principal (interfaz eth1 del Open vSwitch 1), por el que transita el tráfico entre el cliente legítimo y el servidor de streaming. Su interpretación requiere precisar el alcance de cada una:

- Throughput: corresponde al throughput efectivo, es decir, a los bits por segundo que el servidor de streaming entrega al cliente legítimo desde los puertos 8554 y 8888. Excluye el tráfico de ataque y el resto del tráfico del punto de captura, por lo que representa el caudal útil del servicio.
- RTT: se obtiene del campo tcp.analysis.ack_rtt, filtrado por la dirección IP del cliente legítimo. Incluye, además del retardo de la red, el tiempo de respuesta del servidor y el

efecto de los ACK retardados, por lo que refleja la capacidad de respuesta del servicio en su conjunto. Se aplica un filtrado por mediana ± 3 MAD para descartar valores atípicos, como los ACK retrasados durante las ráfagas de descarga HLS, que en algunas corridas superan el 25 % de las muestras y que el filtrado IQR no elimina.

- Jitter: se aplica la fórmula recursiva de la RFC 3550 a los intervalos entre llegadas de los paquetes TCP del cliente (puertos 8554 y 8888), descartando los intervalos superiores a 50 ms asociados a la segmentación HLS. Dado que la RFC 3550 se concibió para RTP/UDP y que el filtro acota los valores, el resultado es un indicador de la variabilidad temporal y no un jitter RTP en sentido estricto; su comparación con el umbral de 50 ms de la Y.1541 es, por tanto, indicativa.
- Pérdida de paquetes: sobre un transporte fiable como TCP, la pérdida en la red se manifiesta como retransmisiones. En la línea base se contabilizan los segmentos perdidos (tcp.analysis.lost_segment) y en los escenarios de ataque, las retransmisiones no espurias, con un filtrado P90 que atenúa las ráfagas puntuales. Como las retransmisiones también pueden deberse a temporizaciones causadas por la sobrecarga del servidor, esta métrica aproxima el IPLR de la Y.1541, pero no equivale estrictamente a él.

La comparación entre OpenDaylight y Ryu es válida en términos relativos, ya que ambos controladores se evalúan con el mismo procedimiento, mientras que la comparación de los valores absolutos con los umbrales de la ITU-T Y.1541 debe interpretarse como indicativa. Con todo, la pérdida estimada en la línea base (0.02 % a 0.06 %) se mantiene por debajo del umbral de 0.1 %.

4. CAPÍTULO IV: RESULTADOS Y ANÁLISIS

Este capítulo presenta los resultados experimentales y su análisis. La metodología completa selección de herramientas, diseño de la topología, instalación de los controladores, implementación del entorno de pruebas y procedimiento de captura se detalla en el Capítulo III; este capítulo recoge únicamente una síntesis del diseño experimental, necesaria para interpretar los resultados. El capítulo se organiza de la siguiente manera: (i) configuración final del experimento; (ii) resultados de la línea base; (iii) resultados bajo ataque; (iv) comparación entre OpenDaylight y Ryu; (v) análisis estadístico; (vi) discusión de los resultados; (vii) proyección hacia escenarios de referencia de mayor escala y limitaciones; y (viii) propuesta preliminar de mitigación como trabajo futuro.

4.1. Síntesis del diseño experimental

El experimento evalúa el efecto de un ataque de denegación de servicio tcp_exhaust sobre la calidad de servicio (QoS) de un servicio de streaming HLS y compara dos controladores SDN, OpenDaylight y Ryu. Las variables independientes son la intensidad del ataque (número de workers, con 20 hilos concurrentes cada uno) y el controlador; el modo de ataque permanece fijo (tcp_exhaust), y la topología, el direccionamiento, el códec y los recursos asignados a cada máquina virtual se mantienen como variables controladas. Las variables dependientes son cuatro métricas de QoS: throughput, RTT, jitter y pérdida de paquetes. Se definen cinco escenarios: E1 (línea base, sin ataque) y E2 a E5 (intensidad creciente, de 1 a 4 workers), ejecutados con tres réplicas por escenario y controlador. Los valores de referencia de mayor escala empleados en la proyección son parámetros representativos tomados de la literatura, no mediciones propias. El detalle de estos elementos y su justificación normativa se presentan en el Capítulo III.

4.2. Configuración final del experimento

Antes de ejecutar las corridas oficiales, el entorno requiere una serie de preparativos para que las capturas sean comparables entre escenarios y reproducibles. Esta sección describe las corridas piloto de verificación y el orden de ejecución de las pruebas.

4.2.1. Corrida piloto y verificación previa al experimento

Antes de iniciar las corridas oficiales, se ejecutan dos o tres corridas piloto cuyo propósito no es generar resultados, sino validar que toda la cadena de captura funciona según lo previsto. En ellas se verifica que los archivos pcap se generan con tamaño no nulo, que ntopng recibe el tráfico replicado por el puerto SPAN, que las API del controlador y del servidor de streaming responden bajo carga, que el script de orquestación arranca y detiene cada subproceso en el orden correcto y que el tiempo total de cada corrida coincide con la planificación. Cualquier ajuste necesario —en la ventana de medición, en los tiempos de espera de las API o en los recursos asignados a las máquinas virtuales se realiza en esta fase, y solo cuando las corridas piloto concluyen sin errores se procede a las corridas oficiales.

4.2.2. Orden de ejecución de las pruebas

Las corridas oficiales siguen una secuencia de menor a mayor intensidad de ataque. Primero se ejecutan las corridas de línea base, sin ataque, repetidas al menos tres veces para obtener un valor de referencia estable y su desviación estándar. A continuación, se ejecutan los escenarios de ataque tcp_exhaust con el controlador OpenDaylight en sus distintos niveles de intensidad. Completada la batería de pruebas con OpenDaylight, se reinicia el laboratorio y se repite la misma secuencia con el controlador Ryu, sin modificar la topología, el direccionamiento ni los recursos asignados. Este orden busca que las únicas variables que cambian entre conjuntos comparables sean el controlador y la intensidad del ataque, condición necesaria para atribuir las diferencias de QoS a esas variables y no a alteraciones del entorno.

4.3. Línea base sin ataque

Esta sección presenta las métricas de QoS registradas con la red en condiciones normales, sin ataque, que constituyen la referencia para cuantificar la degradación en los demás escenarios. Las métricas se obtienen con el procedimiento descrito en el Capítulo III (sección "Procesamiento y análisis de datos") y su alcance se precisa en la subsección "Consideraciones sobre las métricas y los puntos de medición"; en resumen, el throughput es el efectivo entregado al cliente, el RTT corresponde a las conexiones TCP del cliente, el jitter es un indicador de variabilidad temporal y la pérdida se estima a partir de las retransmisiones TCP. Para las métricas de retardo se reporta la media junto con el percentil 95. Cada métrica se obtiene con tres réplicas por escenario, y los valores reportados corresponden a su promedio.

4.3.1. Métricas de QoS observadas en condiciones normales

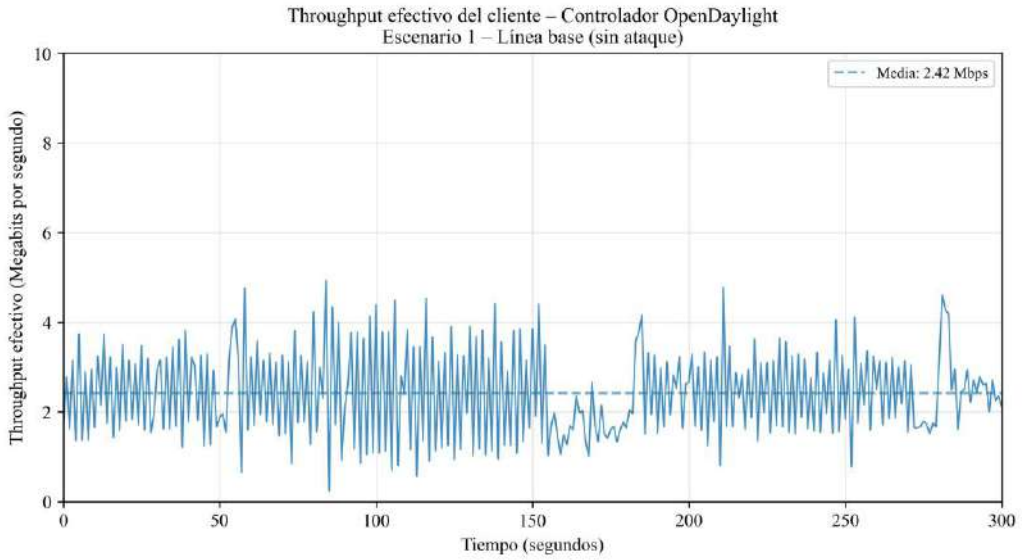
Para complementar el análisis de la línea base y disponer de evidencia visual del comportamiento de la red en ausencia de ataque, se procesan las capturas pcap de las tres corridas de cinco minutos, promediadas segundo a segundo, y se generan las gráficas que se presentan a continuación, que constituyen la referencia cuantitativa para comparar el desempeño bajo los escenarios de ataque. Todos los escenarios se procesan con el script descrito en la sección "Procesamiento y análisis de datos" del Capítulo III (código en el Anexo F), por lo que el proceso de extracción es el mismo en todos los casos y las gráficas comparten formato y escala.

4.3.2. Escenario E1-ODL (controlador OpenDaylight)

El throughput efectivo que recibe el cliente con OpenDaylight, medido en intervalos de un segundo (Figura 42), permanece estable en ausencia de ataque, con una media de 2.42 Mbps y fluctuaciones reducidas, propias de la transmisión HLS sin perturbaciones. Este valor constituye la referencia para cuantificar la degradación posterior.

Figura 42

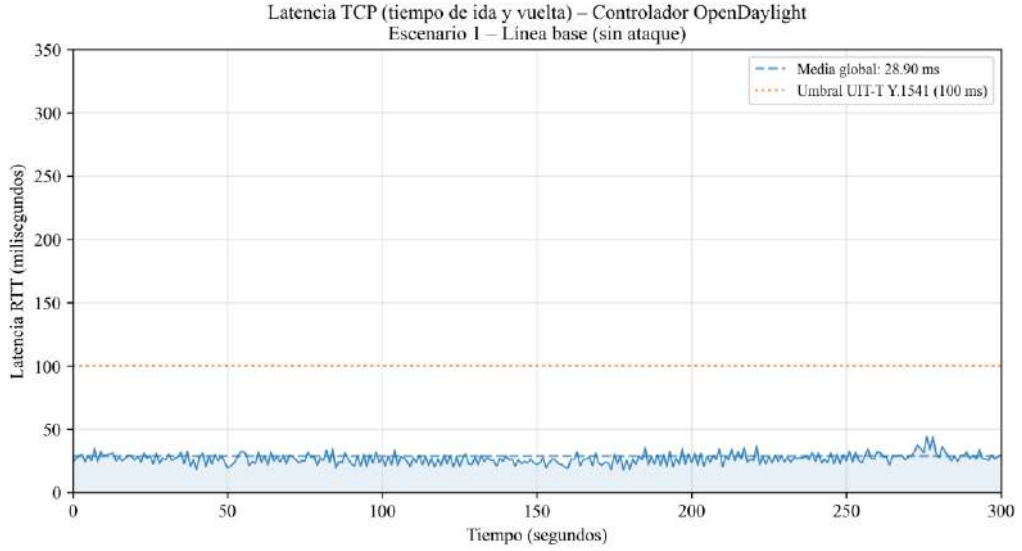
Throughput efectivo del tráfico de línea base con el controlador OpenDaylight (intervalos de 1 s)



El RTT TCP de la línea base con OpenDaylight promedia 28.90 ms, con una mediana de 27.78 ms (Figura 43). Este valor equivale al 29 % del umbral de 100 ms adoptado a partir de la Recomendación ITU-T Y.1541, por lo que la red dispone de un margen amplio antes del ataque.

Figura 43

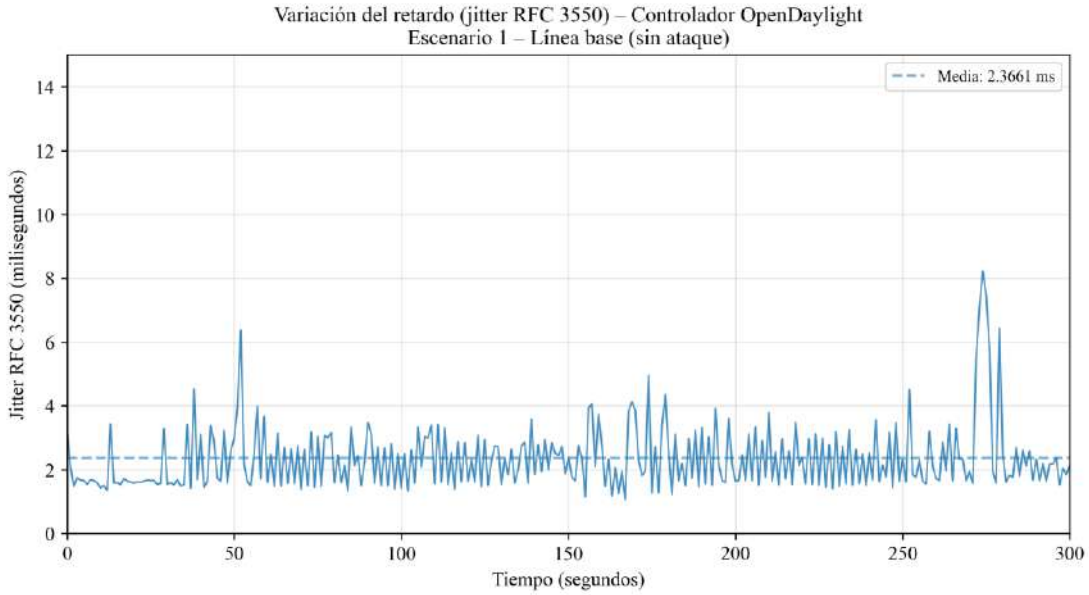
RTT TCP en la línea base con el controlador OpenDaylight



El jitter de la línea base con OpenDaylight, calculado según la RFC 3550, presenta una media de 2.37 ms (Figura 44). Este valor se sitúa un orden de magnitud por debajo del umbral de 50 ms de la ITU-T Y.1541, lo que indica que, en ausencia de ataque, el plano de datos entrega los paquetes con una regularidad temporal muy superior a la exigida por el servicio.

Figura 44

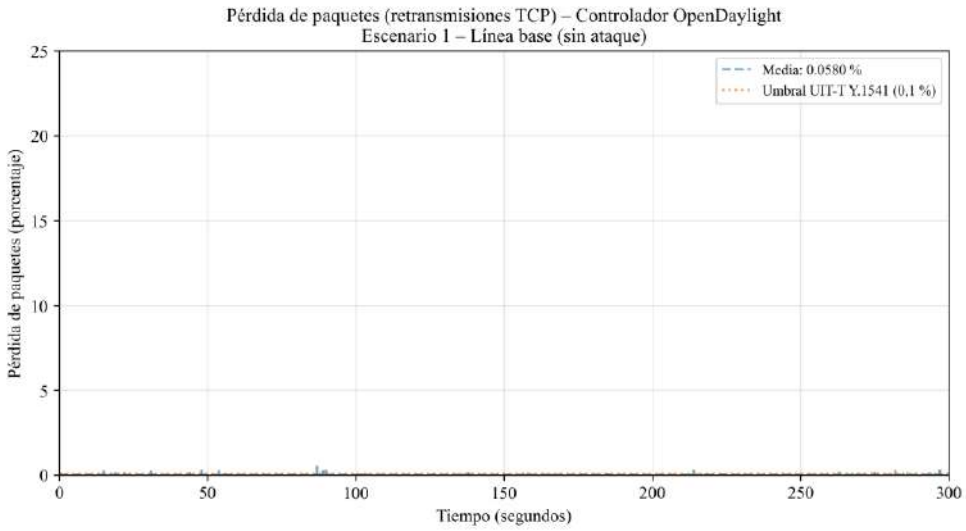
Jitter RFC 3550 en la línea base con el controlador OpenDaylight



La Figura 45 presenta la tasa de pérdida de paquetes de la línea base con OpenDaylight, medida como segmentos TCP perdidos (huecos en la secuencia). La media es de 0.0580 %, valor que no excede el umbral de 0.1 % de la Recomendación ITU-T Y.1541; por tanto, la línea base cumple este umbral. Esta pérdida residual no es atribuible al ataque, sino al nivel mínimo de pérdida propio de la transmisión TCP en la topología emulada en GNS3 y al método de estimación, que no equivale estrictamente al IPLR (véase la subsección Consideraciones sobre las métricas y los puntos de medición). Como resultado, la degradación se encuentra debajo del umbral.

Figura 45

Pérdida de paquetes en la línea base con el controlador OpenDaylight



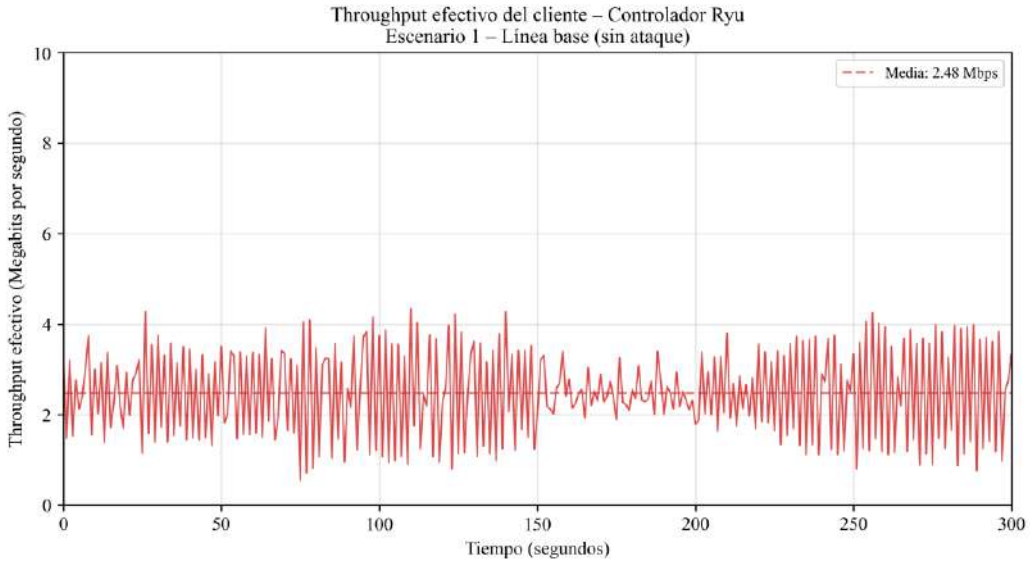
4.3.3. Escenario E1-Ryu (controlador Ryu)

A continuación, se presentan las gráficas de la línea base obtenidas con el controlador Ryu en las mismas condiciones experimentales, a partir del promedio de tres corridas de cinco minutos (300 s), con el cliente multi-viewer HLS activo y sin ataque.

El throughput efectivo de la línea base con Ryu registra una media de 2.48 Mbps con un comportamiento estable (Figura 46), prácticamente equivalente al de OpenDaylight (2.42 Mbps). Esta coincidencia sugiere que el entorno no introduce sesgos apreciables entre controladores, de modo que las diferencias observadas posteriormente pueden atribuirse al controlador y no a la plataforma de prueba.

Figura 46

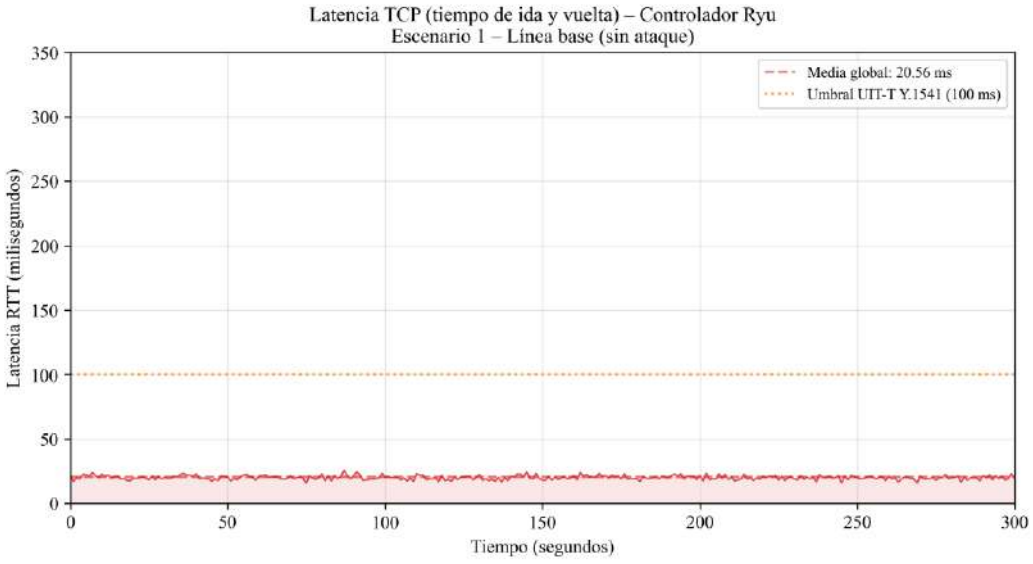
Throughput efectivo del tráfico de línea base con el controlador Ryu (intervalos de 1 s)



El RTT TCP de la línea base con Ryu promedia 20.56 ms (Figura 47), un 28.8 % inferior al de OpenDaylight (28.90 ms). Dado que en reposo los flujos ya están instalados y el controlador apenas interviene en el reenvío, esta diferencia podría deberse también a la distinta carga que cada controlador impone sobre el equipo anfitrión; su origen se analiza en la discusión.

Figura 47

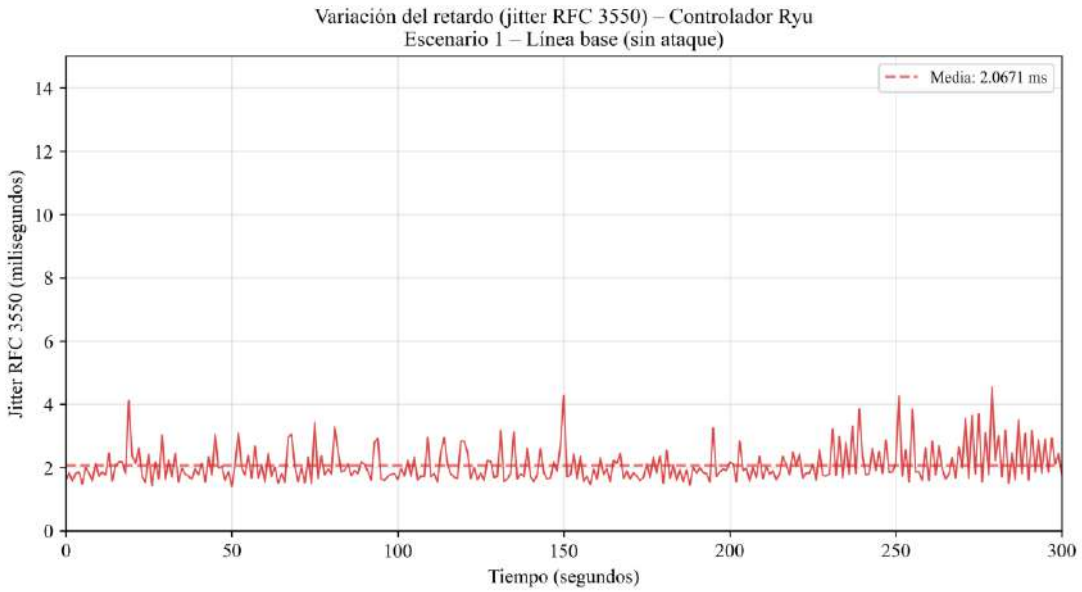
RTT TCP en la línea base con el controlador Ryu



El jitter de la línea base con Ryu alcanza una media de 2.07 ms (Figura 48), comparable al de OpenDaylight (2.37 ms). En reposo, ambos controladores ofrecen, por tanto, una estabilidad temporal equivalente y holgada frente al umbral de 50 ms.

Figura 48

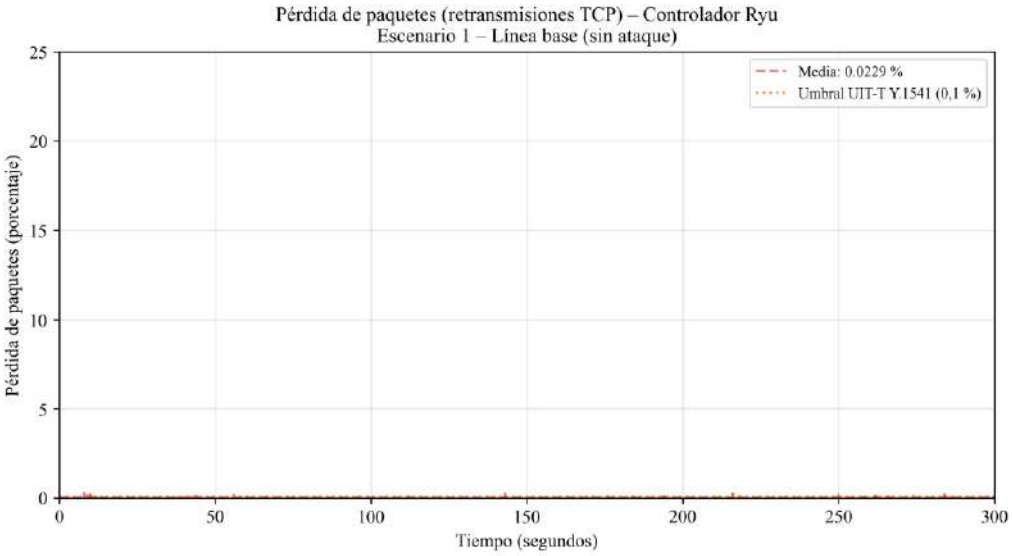
Jitter RFC 3550 en la línea base con el controlador Ryu



La pérdida de paquetes de la línea base con Ryu es de 0.0229 % (Figura 49), inferior a la de OpenDaylight (0.0580 %); ambas se mantienen por debajo del umbral de 0.1 %, de modo que el nivel mínimo de pérdida del entorno emulado es bajo con los dos controladores.

Figura 49

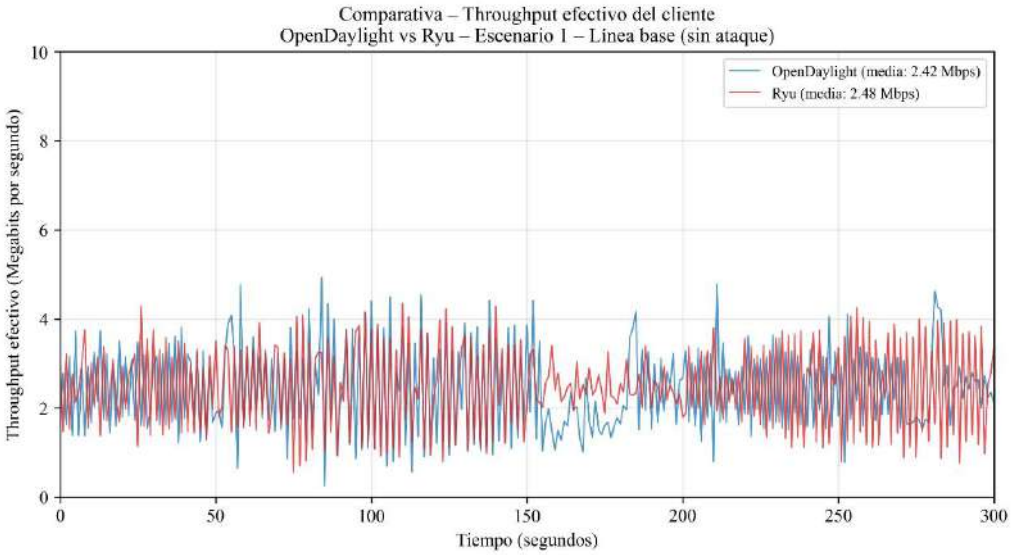
Pérdida de paquetes en la línea base con el controlador Ryu



La superposición de las curvas de throughput de ambos controladores en la línea base (Figura 50) no revela diferencias prácticas: las dos trazas se solapan durante toda la ventana, lo que confirma la equivalencia del punto de partida antes del ataque.

Figura 50

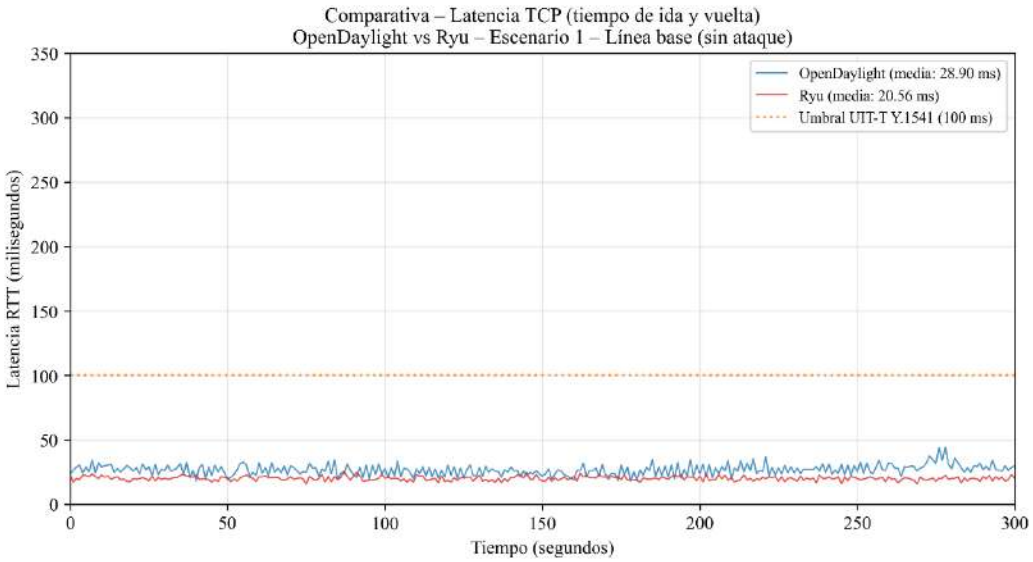
Comparación del throughput efectivo entre OpenDaylight y Ryu (línea base)



La comparación del RTT de la línea base (Figura 51) muestra a Ryu sistemáticamente por debajo de OpenDaylight. Aunque ambos operan holgadamente dentro del umbral de 100 ms, esta brecha, modesta en reposo, se acentúa bajo ataque.

Figura 51

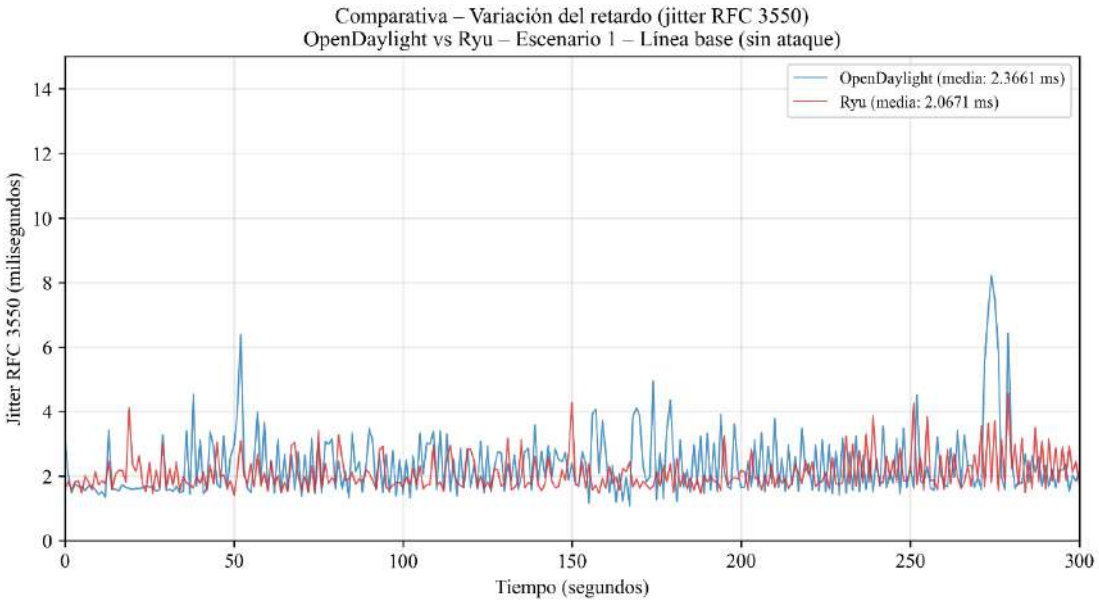
Comparación del RTT entre OpenDaylight y Ryu (línea base)



La Figura 52 compara el jitter de la línea base: ambos controladores presentan valores semejantes y muy inferiores al umbral de la ITU-T Y.1541, por lo que ninguno parte con una ventaja significativa de estabilidad temporal.

Figura 52

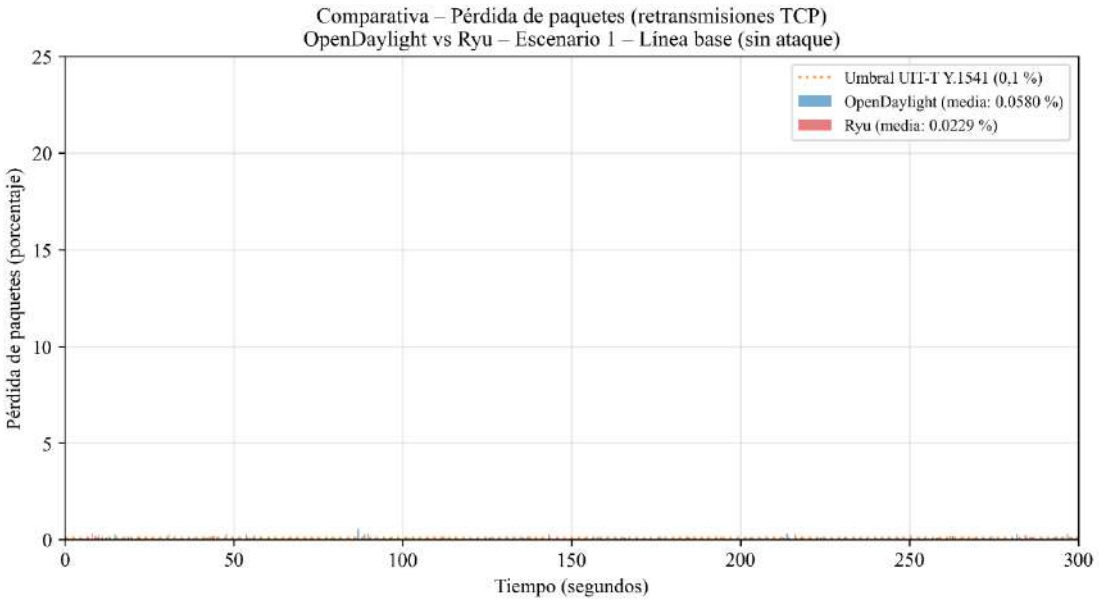
Comparación del jitter entre OpenDaylight y Ryu (línea base)



Las tasas de pérdida de paquetes en reposo son mínimas y prácticamente idénticas en ambos controladores (Figura 53), lo que completa la caracterización de una línea base homogénea a partir de la cual se atribuyen las degradaciones observadas en los escenarios de ataque.

Figura 53

Comparación de la pérdida de paquetes entre OpenDaylight y Ryu (línea base)



4.4. Resultados bajo ataque

Los resultados de esta sección corresponden a los escenarios de ataque DDoS con el módulo tcp_exhaust ejecutados contra el servidor de streaming MediaMTX (192.168.10.66), cuyo servicio HLS (puerto 8888), el mismo que consume el cliente legítimo, es el objetivo del agotamiento de conexiones, con una intensidad creciente de 1 a 4 workers concurrentes de 20 hilos cada uno. Cada escenario se ejecuta con el plano de control gestionado por OpenDaylight y por Ryu, para comparar el comportamiento de ambos controladores bajo la misma carga de ataque. Para facilitar la lectura, esta sección desarrolla en detalle el escenario representativo E3 (2 workers × 20 hilos); los escenarios E2 (1 worker), E4 (3 workers) y E5 (4 workers) se documentan íntegramente en los Anexos C, D y E, y su síntesis cuantitativa se recoge en la Tabla 32 de la sección Comparación entre OpenDaylight y Ryu.

4.4.1. Escenario E3: 2 workers × 20 hilos

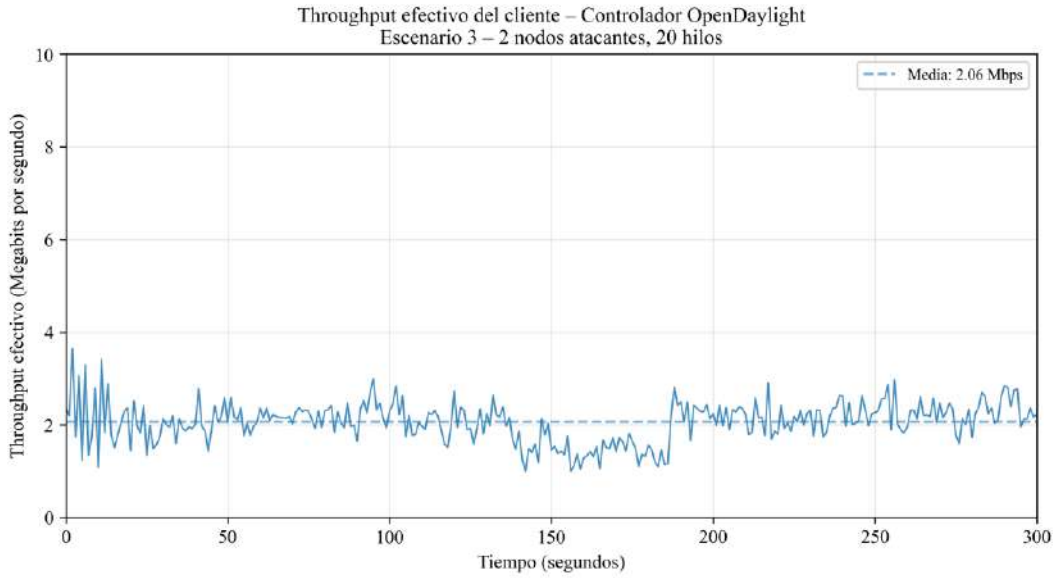
En el escenario E3, 2 workers concurrentes generan tráfico TCP simultáneo contra el servidor de streaming. En promedio por corrida, con OpenDaylight se registran 389 151 paquetes en 312.3 s (107.3 MB), mientras que con Ryu se registran 713 418 paquetes en 320.0 s (149.8 MB). Las capturas difieren en volumen entre controladores; el origen de esta diferencia se analiza en la discusión.

4.4.1.1. Escenario E3-ODL (2 workers)

Con 2 workers y OpenDaylight, el throughput efectivo (Figura 54) registra una media de 2.06 Mbps con picos de 3.65 Mbps, un 14.6 % menos que los 2.42 Mbps de la línea base. En el mismo intervalo, el throughput agregado del enlace alcanza 2.75 Mbps, de modo que la mayor parte de la diferencia corresponde al tráfico del atacante y no al servicio entregado al cliente.

Figura 54

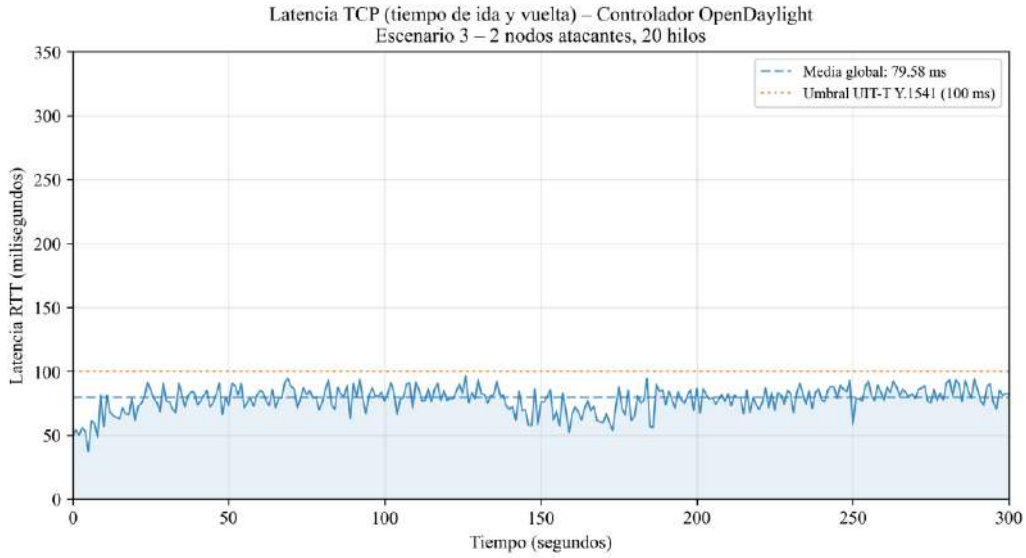
Throughput efectivo con OpenDaylight (2 workers × 20 hilos)



La Figura 55 muestra el RTT TCP estimado a partir de los ACK. El RTT medio es de 79.58 ms, con una mediana de 76.61 ms y un percentil 95 de 143.32 ms. La media se mantiene por debajo del umbral de 100 ms, pero representa 2.8 veces la línea base (28.90 ms), y el percentil 95 supera el umbral, lo que indica que una parte de las respuestas ya se entrega con retardos incompatibles con la clase 0. Estas métricas se obtienen de las capturas mediante tshark, con el script documentado en el Anexo F.

Figura 55

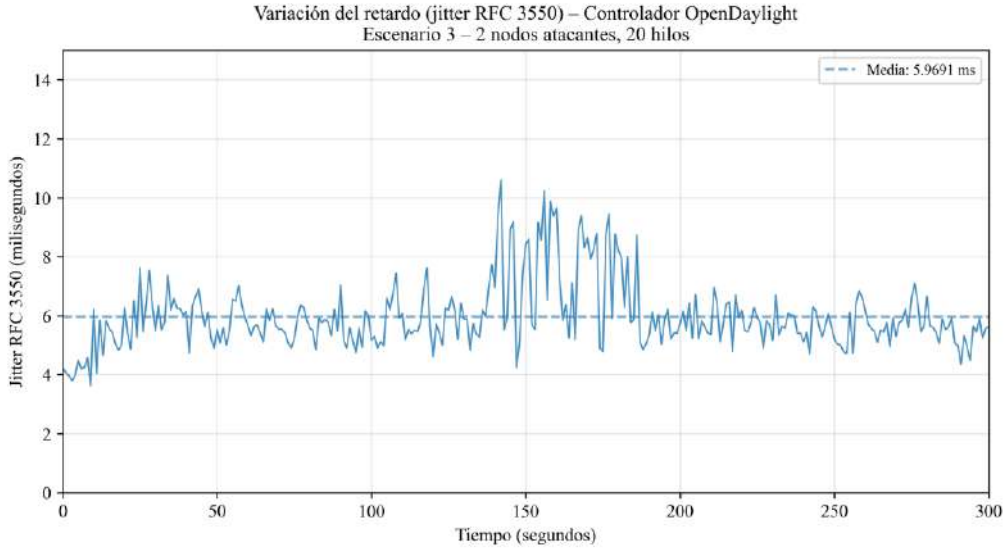
RTT TCP durante el ataque con OpenDaylight (2 workers × 20 hilos)



El jitter (Figura 56) se calcula según la definición de variación del retardo entre llegadas de la RFC 3550 (sección 6.4.1), a partir de las marcas de tiempo de las capturas y con el script del Anexo F; como se indicó, su aplicación a un transporte TCP/HLS es una aproximación. La media es de 5.97 ms, con un percentil 95 de 8.72 ms y un máximo de 10.61 ms en la serie por segundo: 2.5 veces la línea base (2.37 ms), aunque por debajo del umbral de 50 ms de la Y.1541.

Figura 56

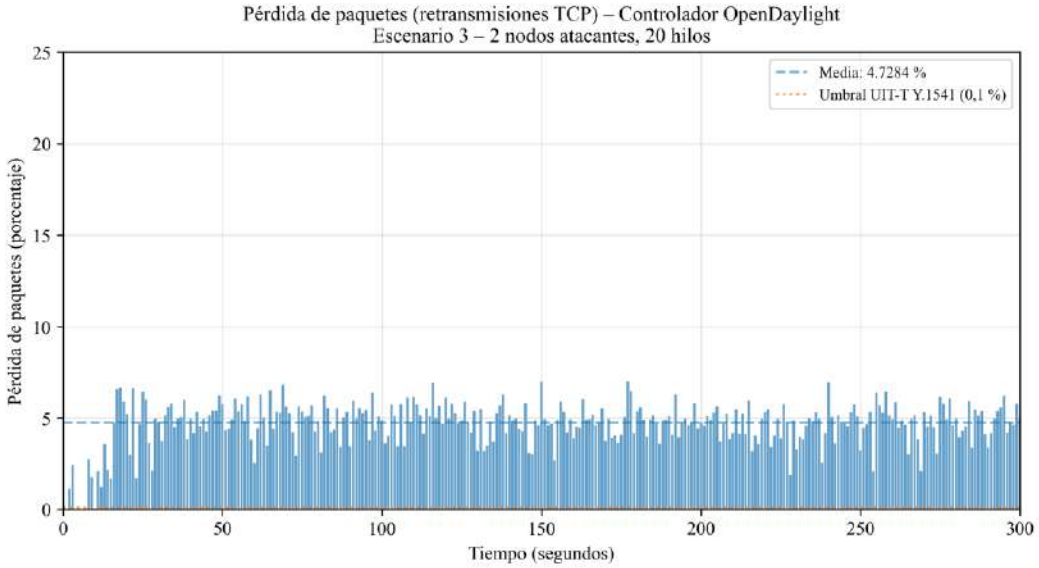
Jitter RFC 3550 durante el ataque con OpenDaylight (2 workers × 20 hilos)



Durante el ataque con 2 workers y OpenDaylight, la tasa de pérdida de paquetes asciende a una media de 4.73 % (Figura 57), muy por encima de la línea base (0.058 %), del 0.1 % admitido por la ITU-T Y.1541 y del 1 % que la ITU-T G.1010 tolera para video en streaming, lo que confirma el impacto del ataque tcp_exhaust sobre la calidad del servicio.

Figura 57

Pérdida de paquetes durante el ataque con OpenDaylight (2 workers × 20 hilos)

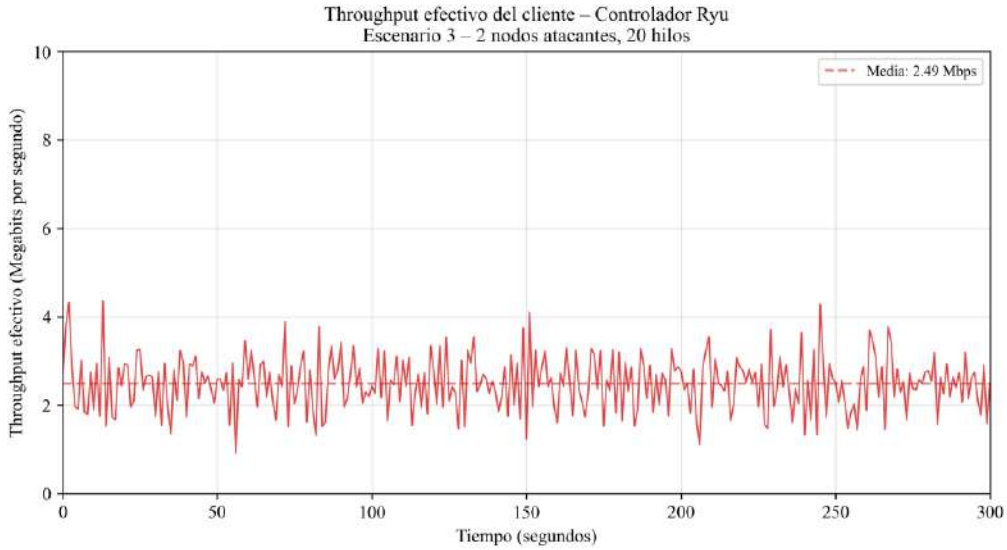


4.4.1.2. Escenario E3-Ryu (2 workers)

Con Ryu y 2 workers, el throughput efectivo (Figura 58) registra una media de 2.49 Mbps (pico de 4.36 Mbps), 1.21 veces el de OpenDaylight en idénticas condiciones y un 0.2 % más que su línea base (2.48 Mbps). Con Ryu, el cliente conserva por tanto una mayor parte del caudal útil del servicio bajo la misma carga de ataque.

Figura 58

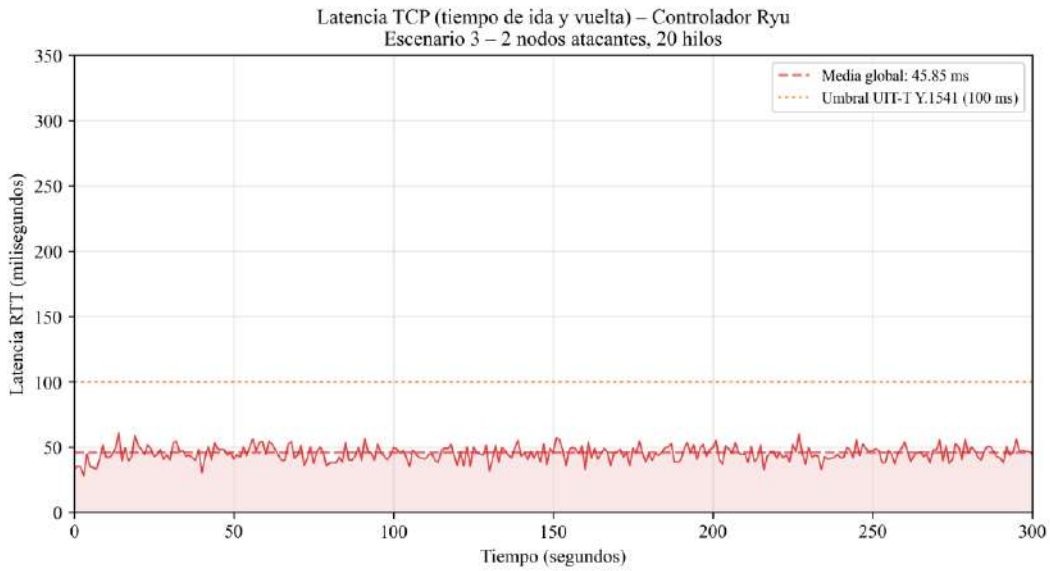
Throughput efectivo con Ryu (2 workers × 20 hilos)



El RTT de Ryu con 2 workers promedia 45.85 ms, con un percentil 95 de 82.79 ms (Figura 59). Aunque supone 2.2 veces su línea base (20.56 ms), la media y el percentil 95 se mantienen por debajo del umbral de 100 ms; frente a los 79.58 ms de OpenDaylight, la latencia media es 1.7 veces menor, una diferencia que separa claramente a ambos controladores ante el ataque tcp_exhaust.

Figura 59

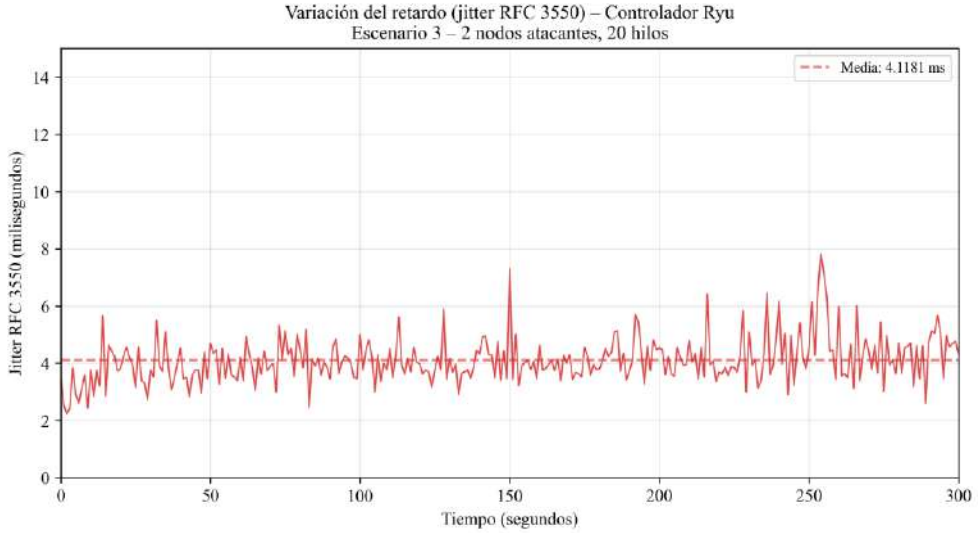
RTT TCP durante el ataque con Ryu (2 workers × 20 hilos)



El jitter de Ryu con 2 workers (Figura 60) se sitúa en 4.12 ms de media (percentil 95 de 5.64 ms), 1.4 veces por debajo del registrado con OpenDaylight (5.97 ms). La estabilidad temporal se ve menos comprometida que con OpenDaylight, aunque este resultado debe interpretarse con cautela por el filtro de pausas aplicado en el cálculo (véase el Anexo F).

Figura 60

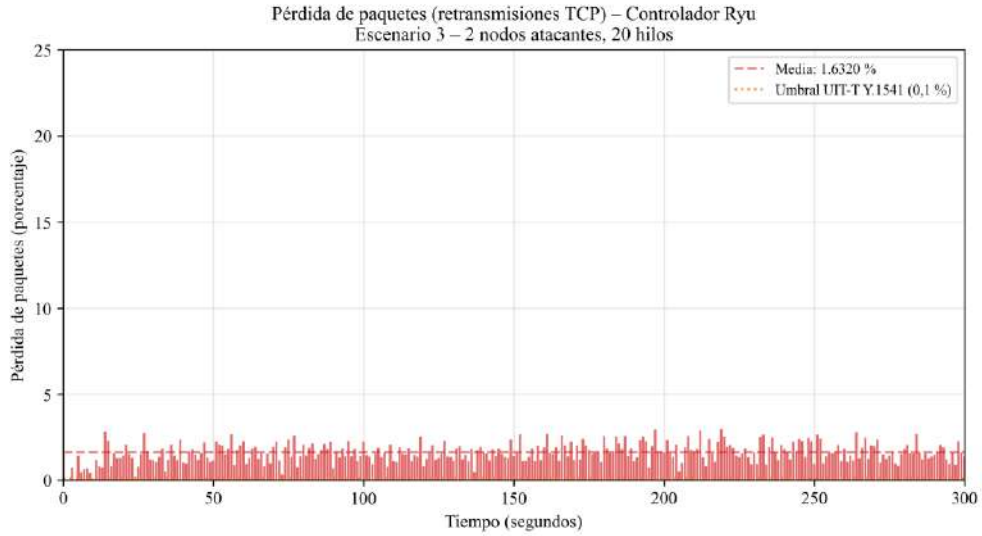
Jitter RFC 3550 durante el ataque con Ryu (2 workers × 20 hilos)



La pérdida de paquetes de Ryu con 2 workers alcanza una media de 1.63 % (Figura 61): un deterioro respecto a su línea base (0.023 %), pero aproximadamente un tercio del 4.73 % de OpenDaylight. El efecto del ataque tcp_exhaust sobre la integridad del flujo es, por tanto, menor con Ryu.

Figura 61

Pérdida de paquetes durante el ataque con Ryu (2 workers × 20 hilos)

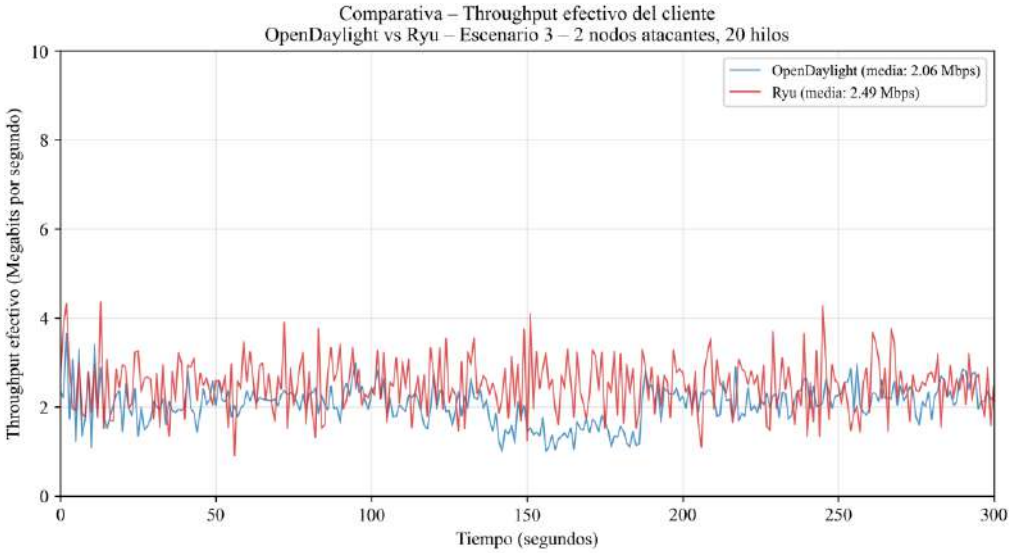


4.4.2. Comparación entre OpenDaylight y Ryu (2 workers)

La superposición de ambas curvas durante el ataque con 2 workers (Figura 62) muestra a Ryu por encima de OpenDaylight en el 73 % de los segundos de la ventana de medición. La diferencia entre medias (2.49 frente a 2.06 Mbps) equivale a un 20.6 % más de caudal útil entregado al cliente con Ryu.

Figura 62

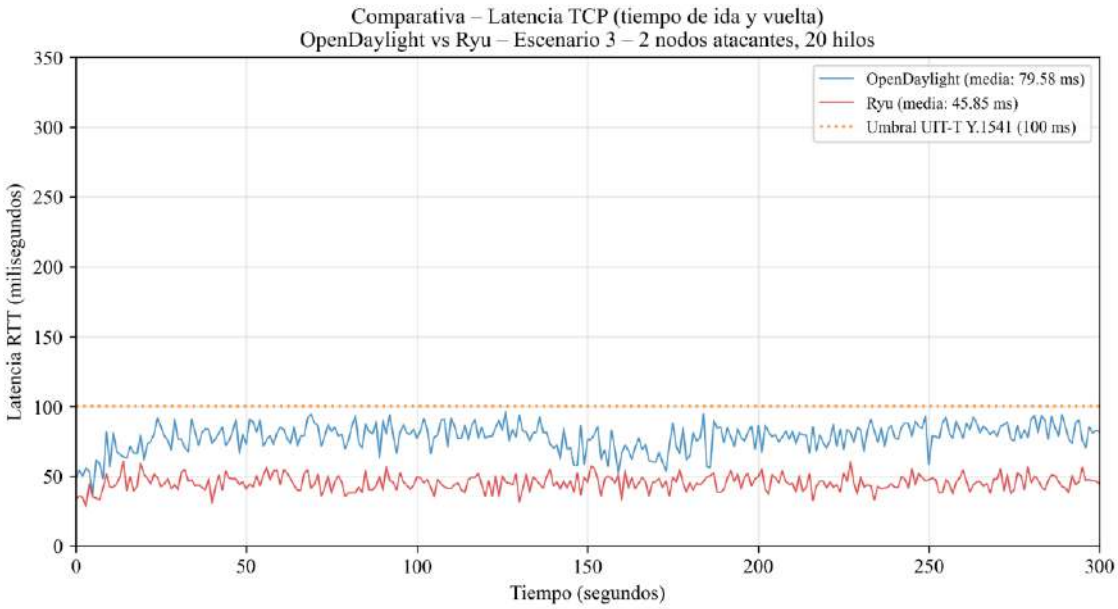
Comparación del throughput efectivo entre OpenDaylight y Ryu (2 workers × 20 hilos)



La Figura 63 compara el RTT de ambos controladores. Ante el ataque tcp_exhaust, Ryu presenta un throughput efectivo 1.2 veces superior, un RTT medio 1.7 veces inferior y un jitter medio 1.4 veces menor, además de cursar una tasa de paquetes 1.8 veces mayor. La tasa de paquetes es una variable de carga y no de desempeño: que Ryu conserve mayor throughput y menor RTT mientras procesa una carga de tráfico atacante más elevada indica que mantiene una mayor capacidad de servicio bajo dicha carga, y no una "mejor tasa de paquetes" en sí misma. Las posibles causas de esta diferencia se examinan en la sección "Discusión de los resultados".

Figura 63

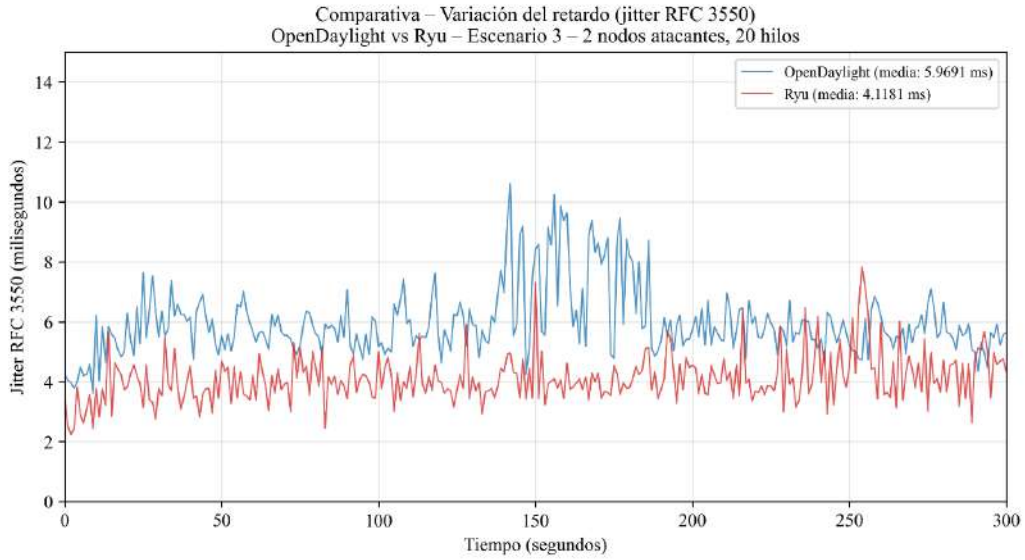
Comparación del RTT entre OpenDaylight y Ryu (2 workers × 20 hilos)



El jitter de ambos controladores con 2 workers permanece por debajo del umbral de 50 ms de la ITU-T Y.1541 (Figura 64); no obstante, Ryu conserva un margen sensiblemente mayor, de modo que su ventaja no se limita al caudal y la latencia, sino que se extiende a la estabilidad temporal.

Figura 64

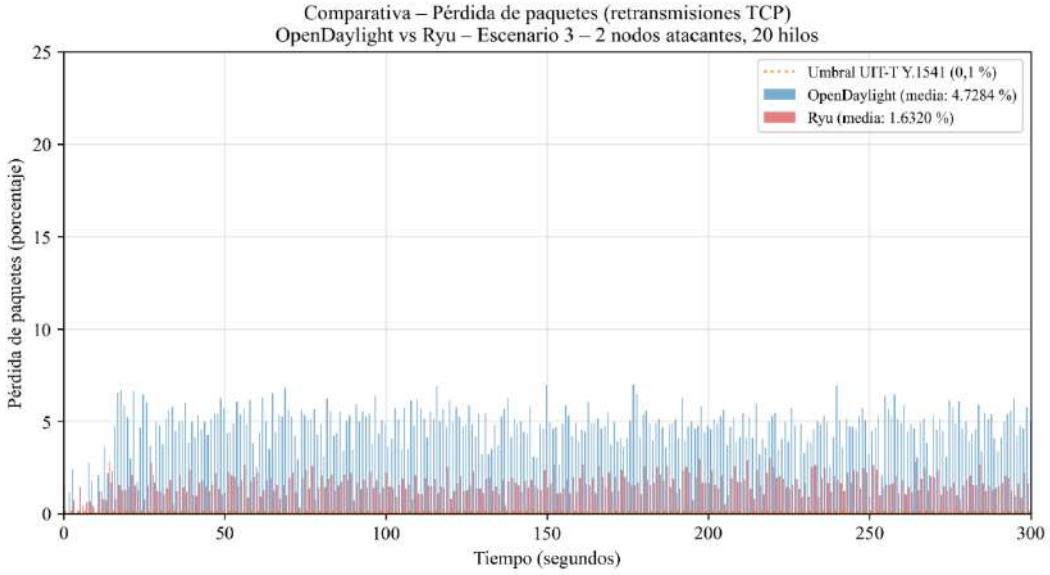
Comparación del jitter entre OpenDaylight y Ryu (2 workers × 20 hilos)



La pérdida de paquetes contrasta de forma marcada entre controladores: 4.73 % con OpenDaylight frente a 1.63 % con Ryu (Figura 65). Esta brecha, cercana a un factor de 3, resume en la métrica más sensible para el usuario el efecto más severo del ataque con OpenDaylight.

Figura 65

Comparación de la pérdida de paquetes entre OpenDaylight y Ryu (2 workers × 20 hilos)



El comportamiento en los demás niveles de intensidad se resume en la Tabla 32 y se documenta gráficamente, con el mismo conjunto de métricas por controlador, en los Anexos C (E2, 1 worker), D (E4, 3 workers) y E (E5, 4 workers).

4.5. Comparación entre OpenDaylight y Ryu

El análisis conjunto de los cuatro escenarios con ataque revela un patrón consistente a partir de la intensidad media: en E3, E4 y E5 el controlador Ryu presenta mejores valores que OpenDaylight en las cuatro métricas de QoS, mientras que en E2 Ryu ya presenta menor RTT, jitter y pérdida, aunque el throughput efectivo es similar (2.52 Mbps con OpenDaylight y 2.35 Mbps con Ryu). La Tabla 32 resume los valores medios obtenidos con ambos controladores para cada configuración de workers.

Tabla 32

Métricas de QoS medias por escenario y controlador

Escenario	Controlador	Throughput efectivo (Mbps)	Pérdida (%)	RTT medio (ms)	Jitter medio (ms)
E1 (base)	OpenDaylight	2.42 ± 0.12	0.06 ± 0.03	28.90 ± 3.14	2.37 ± 0.36
E1 (base)	Ryu	2.48 ± 0.03	0.02 ± 0.01	20.56 ± 1.44	2.07 ± 0.04
E2 (1w)	OpenDaylight	2.52 ± 0.04	2.19 ± 0.06	56.45 ± 1.68	4.22 ± 0.14
E2 (1w)	Ryu	2.35 ± 0.15	1.58 ± 1.23	28.83 ± 2.11	2.69 ± 0.17
E3 (2w)	OpenDaylight	2.06 ± 0.17	4.73 ± 0.19	79.58 ± 0.91	5.97 ± 0.42
E3 (2w)	Ryu	2.49 ± 0.01	1.63 ± 0.14	45.85 ± 0.70	4.12 ± 0.26
E4 (3w)	OpenDaylight	1.41 ± 0.21	7.16 ± 0.75	84.32 ± 7.20	7.79 ± 0.43
E4 (3w)	Ryu	2.17 ± 0.23	5.19 ± 3.35	56.07 ± 2.80	5.80 ± 0.38
E5 (4w)	OpenDaylight	0.85 ± 0.05	10.75 ± 0.36	92.69 ± 1.87	10.35 ± 0.10

Escenario	Controlador	Throughput efectivo (Mbps)	Pérdida (%)	RTT medio (ms)	Jitter medio (ms)
E5 (4w)	Ryu	1.73 ± 0.13	4.30 ± 0.21	64.36 ± 1.27	6.76 ± 0.18

Nota. Media ± desviación estándar de las tres corridas de cada escenario y controlador.

De la Tabla 32 se desprende que la ventaja de Ryu no es uniforme. Sin ataque (E1), ambos controladores presentan un throughput efectivo equivalente (2.42 frente a 2.48 Mbps), mientras que Ryu registra un RTT un 28.8 % menor (20.56 frente a 28.90 ms). Con el ataque, la diferencia de latencia aparece desde E2 (RTT 2.0 veces menor con Ryu) y, a partir de E3, se extiende a las cuatro métricas: Ryu entrega al cliente entre 1.2 y 2.0 veces más throughput efectivo que OpenDaylight y mantiene un RTT medio entre 1.4 y 1.7 veces menor. La ventaja en pérdida es máxima en E3 (2.9 veces menor con Ryu) y en E5 (2.5 veces), y se reduce en E4 (1.4 veces) por la mayor variabilidad entre corridas de Ryu. En ningún escenario el RTT medio supera el umbral de 100 ms; sin embargo, el percentil 95 lo rebasa con OpenDaylight desde E2 y con Ryu solo en E4 y E5, lo que indica que la cola de la distribución de retardos se degrada antes con OpenDaylight.

Con el ataque (E2–E5), el jitter medio crece con la intensidad en ambos controladores, pero con Ryu se mantiene entre 1.3 y 1.6 veces por debajo del observado con OpenDaylight (6.76 frente a 10.35 ms en E5); en E1, ambos presentan valores similares (2.07 y 2.37 ms). Este comportamiento sugiere una mayor estabilidad temporal del reenvío gestionado por Ryu bajo carga de ataque, aunque debe interpretarse con la cautela indicada para el cálculo del jitter.

Como hipótesis explicativa de la ventaja de Ryu, que no puede verificarse con los datos de este trabajo porque no se registraron métricas del controlador ni del anfitrión, se plantean tres factores, derivados de la arquitectura de cada controlador descrita en el Capítulo II (Figura 3 y Figura 4): (i) la arquitectura reactiva de Ryu, implementada en Python sobre un bucle de

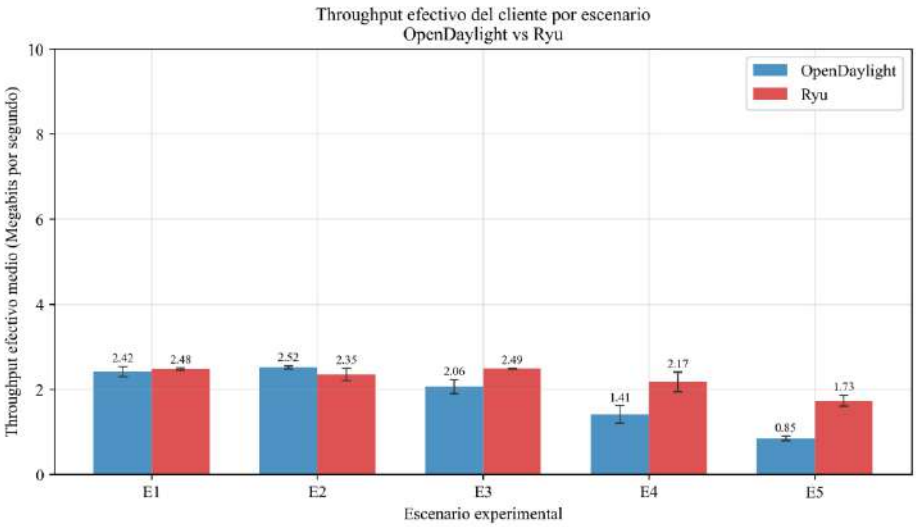
eventos asíncrono, procesaría los mensajes PACKET_IN con menor sobrecarga que el pipeline modular OSGi de OpenDaylight; (ii) el modo de instalación de flujos por defecto de Ryu produciría menor congestión del canal OpenFlow; y (iii) el menor consumo de memoria y CPU de Ryu le permitiría escalar mejor con la tasa de eventos generada por el ataque.

Debe considerarse además una explicación alternativa: el ataque tcp_exhaust se dirige al servidor y no al controlador, y con los flujos ya instalados las nuevas conexiones entre los mismos hosts no generan necesariamente mensajes PACKET_IN. Como OpenDaylight (JVM) consume más CPU y memoria del mismo equipo anfitrión que ejecuta el servidor y los workers, parte de la diferencia podría deberse a la contención de recursos del host y no al procesamiento del plano de control. Asimismo, los estudios de rendimiento de controladores con cbench suelen reportar mayor capacidad de establecimiento de flujos en OpenDaylight que en Ryu [78], lo que refuerza la necesidad de sustentar esta interpretación con evidencia medida.

La Figura 66 resume el throughput efectivo medio de cada controlador en los cinco escenarios. Con OpenDaylight, el caudal útil entregado al cliente pasa de 2.42 Mbps en la línea base a 0.85 Mbps en E5 (un 64.8 % menos que su línea base), mientras que con Ryu pasa de 2.48 a 1.73 Mbps (un 30.2 % menos que su línea base). A partir de E3, la pendiente de degradación es claramente más acusada con OpenDaylight.

Figura 66

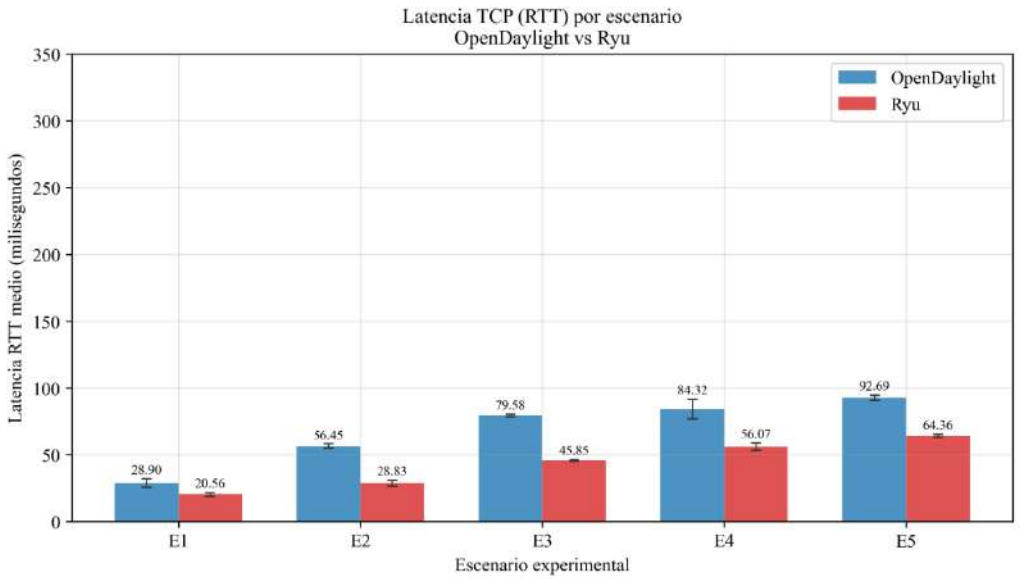
Throughput efectivo medio por escenario y controlador



El RTT medio por escenario (Figura 67) crece de forma monótona con ambos controladores, pero ninguno supera en promedio el umbral de 100 ms: con OpenDaylight la media se aproxima al umbral en E5 (92.69 ms), mientras que con Ryu se mantiene en 64.36 ms. Considerando el percentil 95, OpenDaylight rebasa el umbral desde E2 (1 worker) y Ryu a partir de E4 (3 workers): dos niveles de ataque adicionales que Ryu tolera antes de que la cola de retardos supere el umbral adoptado.

Figura 67

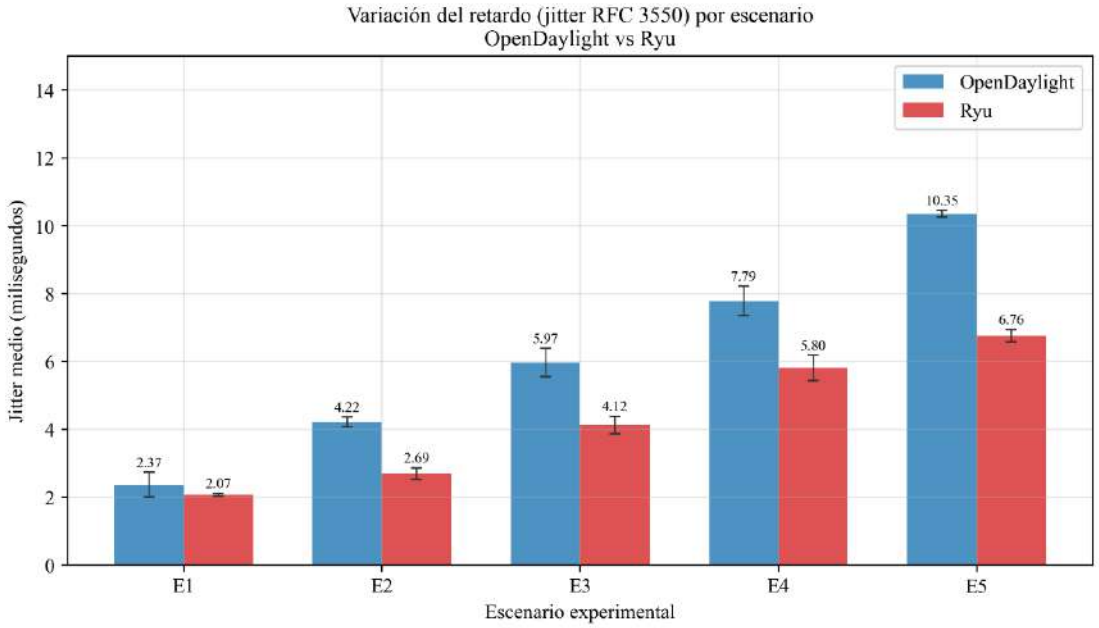
RTT medio por escenario y controlador



El jitter medio por escenario (Figura 68) permanece por debajo del umbral de 50 ms en todos los casos para ambos controladores, y crece de forma monótona con la intensidad del ataque, con valores sistemáticamente menores con Ryu. Estos valores deben interpretarse con cautela, porque el filtro de pausas de 50 ms aplicado en el cálculo limita por construcción los valores de jitter (véase el Anexo F).

Figura 68

Jitter medio por escenario y controlador



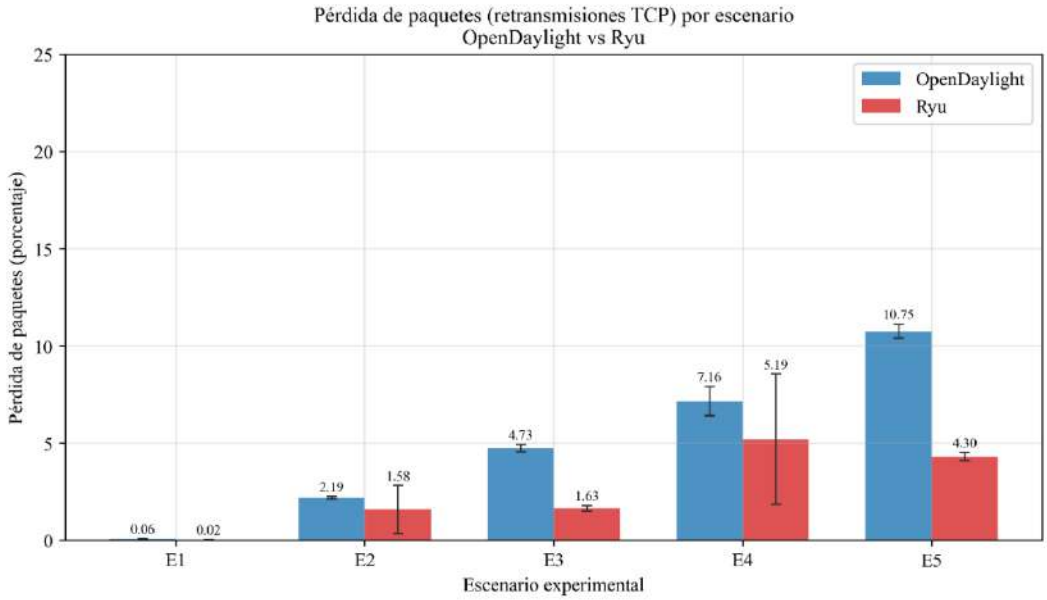
La pérdida de paquetes media por escenario (Figura 69) muestra que OpenDaylight multiplica entre 1.4 y 2.9 veces la tasa de Ryu en los escenarios con ataque; la separación es máxima en E3 y E5, mientras que en E2 y E4 se reduce por la mayor variabilidad entre corridas de Ryu.

La dispersión de la pérdida de Ryu en E4 (5.19 ± 3.35 %) y en E2 (1.58 ± 1.23 %) no refleja una variación uniforme entre corridas, sino una réplica atípica en cada escenario: en E4, las tres corridas registraron 3.61 %, 2.93 % y 9.04 %, y en E2, 3.00 %, 0.97 % y 0.77 %. El análisis por segundo de las dos corridas atípicas muestra caídas breves y repetidas de todo el tráfico del enlace: en 53 segundos de la corrida atípica de E4 y en 31 de la de E2, la tasa de paquetes cae por debajo del 45 % de su mediana y el throughput efectivo baja de 0.3 Mbps,

con picos de pérdida de hasta 53 % en esos segundos. En el resto de las corridas con ataque, de ambos controladores, este comportamiento aparece en ocho segundos o menos. La corrida atípica de E4 registró además menos tráfico total que las otras dos (1 891 frente a unos 2 300 paquetes por segundo), por lo que su mayor pérdida no se explica por un ataque más intenso, sino por interrupciones intermitentes del plano de datos cuyo origen no puede determinarse con las capturas; como todos los componentes comparten el mismo equipo anfitrión, la contención de recursos es una causa plausible. Estas corridas se mantienen en el análisis, porque no se definió un criterio de exclusión previo y, con tres réplicas, descartar una sesgaría la estimación. A modo de análisis de sensibilidad, sin la réplica atípica la pérdida de Ryu sería de 3.27 ± 0.48 % en E4 y de 0.87 ± 0.14 % en E2, y la diferencia con OpenDaylight resultaría significativa en ambos escenarios (prueba t de Welch, p = 0.006 y p = 0.031). La mediana de las tres corridas (3.61 % en E4 y 0.97 % en E2), menos sensible a valores atípicos, confirma que la pérdida típica de Ryu es inferior a la de OpenDaylight (7.40 % y 2.17 %, respectivamente).

Figura 69

Pérdida de paquetes media por escenario y controlador

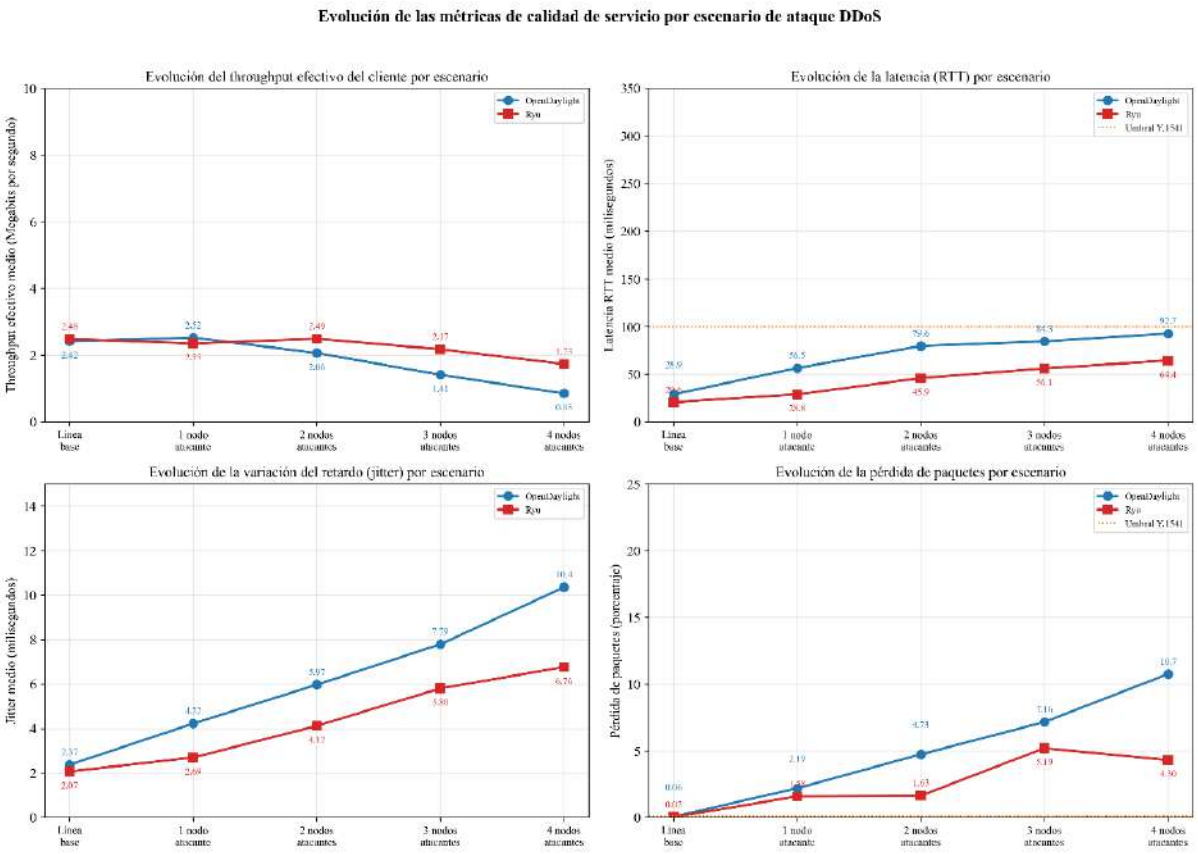


El panel que consolida las cuatro métricas de QoS (Figura 70) permite una lectura conjunta: el throughput, el RTT y la pérdida se degradan de forma pronunciada con

OpenDaylight a medida que aumentan los nodos atacantes, mientras que Ryu mantiene valores sensiblemente mejores, lo que refuerza, dentro de las condiciones evaluadas, una mayor tolerancia al ataque.

Figura 70

Resumen global de métricas de QoS por escenario y controlador



4.6. Análisis estadístico

Además de las medias, el análisis estadístico considera la dispersión de las métricas. Los valores medios corresponden al promedio de las tres corridas ejecutadas por escenario y controlador. Los percentiles 95 del RTT muestran que OpenDaylight registra colas de retardo más altas en términos absolutos: en E3, 143.32 ms frente a 82.79 ms con Ryu, y en E5, 181.52 ms frente a 111.62 ms. En términos relativos, el percentil 95 equivale a 1.8 y 1.8 veces la media en E3, y a 2.0 y 1.7 veces en E5, respectivamente, por lo que la mayor latencia de OpenDaylight se debe principalmente a un desplazamiento de toda la distribución y no solo a episodios aislados.

La Tabla 33 resume el tamaño de muestra que sustenta cada métrica, sumado sobre las tres corridas de cada escenario y controlador, después de aplicar los filtros descritos en el Capítulo III: segundos analizados, paquetes del servicio entregados al cliente (base del throughput efectivo), muestras de RTT que superan el filtro mediana ± 3 MAD frente al total observado, intervalos utilizados en el cálculo del jitter y paquetes TCP considerados en la estimación de la pérdida. El filtro del RTT conserva entre el 81 % y el 96 % de las muestras, y ningún grupo cuenta con menos de 23 626 muestras válidas. Aunque la prueba estadística se aplica sobre las medias de las tres corridas (n = 3 por grupo), cada una de esas medias resume miles de observaciones, lo que reduce la incertidumbre de cada valor medio.

Tabla 33

Tamaño de muestra analizado por escenario y controlador (suma de las tres corridas)

Escenario	Controlador	Tiempo analizado (s)	Paquetes servidor-cliente (throughput)	ACK con RTT medido (válidos / totales)	Intervalos de jitter calculados	Paquetes TCP analizados (base de la pérdida)
E1	OpenDaylight	936	189 432	39 807 / 41 889	232 435	233 410
E1	Ryu	937	194 833	34 459 / 36 335	230 660	232 458
E2	OpenDaylight	951	201 545	57 018 / 60 781	265 263	910 528
E2	Ryu	929	182 751	43 440 / 45 258	229 270	1 091 034
E3	OpenDaylight	937	163 504	50 700 / 57 059	225 133	1 047 066
E3	Ryu	960	200 055	52 953 / 55 403	257 306	1 930 795
E4	OpenDaylight	961	115 776	38 755 / 45 264	163 690	1 169 412

Escenario	Controlador	Tiempo analizado (s)	Paquetes servidor-cliente (throughput)	ACK con RTT medido (válidos / totales)	Intervalos de jitter calculados	Paquetes TCP analizados (base de la pérdida)
E4	Ryu	927	169 459	47 936 / 53 273	226 300	1 854 111
E5	OpenDaylight	968	71 890	23 626 / 29 311	98 548	1 144 862
E5	Ryu	924	134 944	42 893 / 48 797	189 417	1 976 606

Nota. Paquetes y muestras no son equivalentes. Los paquetes servidor - *cliente son las tramas que el servidor (192.168.10.66) envía al cliente legítimo (192.168.10.35) desde los puertos 8554 y 8888; sobre ellas se calcula el throughput efectivo. Los ACK con RTT medido son los acuses de recibo del cliente para los que tshark obtiene el campo tcp.analysis.ack_rtt; como un ACK puede confirmar varios segmentos, son menos que los paquetes de datos, y los válidos son los que superan el filtro mediana ± 3 MAD. Los intervalos de jitter son los intervalos entre paquetes consecutivos del tráfico del cliente, en ambos sentidos, que entran en el cálculo de la RFC 3550 tras descartar las pausas mayores a 50 ms. Los paquetes TCP analizados incluyen todo el tráfico TCP del punto de captura, también el de ataque, y constituyen el denominador de la pérdida; en los escenarios con ataque se excluyen los segundos por encima del percentil 90. Un mismo paquete puede intervenir en varias métricas, por lo que las columnas no son aditivas.*

Para determinar si estas diferencias son estadísticamente significativas, se aplica la prueba t de Welch sobre los valores de las tres corridas de cada escenario y controlador, complementada con el tamaño del efecto (g de Hedges). La Tabla 34 resume los resultados. El RTT es significativamente menor con Ryu en los cinco escenarios ($p < 0.03$) y el jitter, en los

cuatro escenarios con ataque ($p < 0.01$); el throughput efectivo difiere de forma significativa en E3, E4 y E5 ($p < 0.05$) y la pérdida, en E3 y E5 ($p < 0.001$), mientras que en E2 y E4 la variabilidad entre corridas de Ryu impide concluir una diferencia en esta métrica; como se explicó antes, esa variabilidad procede de una réplica atípica en cada escenario y, sin ella, la diferencia resultaría significativa. Dado que cada grupo tiene solo tres corridas, la potencia de la prueba es limitada: un resultado no significativo no demuestra igualdad y debe leerse junto con el tamaño del efecto.

La prueba t de Welch compara las medias de dos grupos sin suponer que sus varianzas son iguales. Para cada métrica y escenario, la estadística se calcula de acuerdo con la ecuación (9):

$$t = (\bar{x}_{ODL} - \bar{x}_{Ryu}) \,/\, \sqrt{\,(s^2_{ODL} \,/\, n_{ODL} + s^2_{Ryu} \,/\, n_{Ryu})} \tag{9}$$

y sus grados de libertad se aproximan con la ecuación de Welch–Satterthwaite (10):

$$\nu = (s^2_{ODL}/n_{ODL} + s^2_{Ryu}/n_{Ryu})^2 \,/\, [s^2_{ODL}/n_{ODL})^2/\,(n_{ODL} - 1) + (s^2_{Ryu}/n_{Ryu})^2/\,(n_{Ryu} - 1)] \tag{10}$$

donde $\bar{x}$ es la media de la métrica en las corridas de cada controlador, s^2 es su varianza muestral y $n = 3$ es el número de corridas por grupo. El valor p se obtiene de la distribución t de Student con ν grados de libertad (prueba bilateral) y la diferencia se considera significativa cuando $p < 0.05$. El tamaño del efecto se cuantifica con la g de Hedges, según las ecuaciones (11) a (13):

$$g = J \cdot (\bar{x}_{ODL} - \bar{x}_{Ryu}) \,/\, s_p \tag{11}$$

$$s_p = \sqrt{\{\,[\,(n_{ODL} - 1) \cdot s^2_{ODL} + (n_{Ryu} - 1) \cdot s^2_{Ryu}\,]\,/\,(n_{ODL} + n_{Ryu} - 2)\,\}} \tag{12}$$

$$J = 1 - 3 \,/\, [4\,(n_{ODL} + n_{Ryu}) - 9] \tag{13}$$

donde s_p es la desviación estándar combinada de ambos grupos y J es el factor de corrección del sesgo para muestras pequeñas (con $n = 3$ corridas por grupo, $J = 0.80$). Un valor

positivo de g indica que la métrica es mayor con OpenDaylight y, según la convención de Cohen, $|g| \geq 0.8$ corresponde a un efecto grande.

Tabla 34

Prueba t de Welch entre OpenDaylight y Ryu por escenario y métrica

Escenario	Throughput efectivo	RTT	Jitter	Pérdida
E1	p = 0.466	p = 0.028	p = 0.285	p = 0.136
	g = -0.57	g = 2.73	g = 0.94	g = 1.45
E2	p = 0.177	p < 0.001	p < 0.001	p = 0.482
	g = 1.27	g = 11.57	g = 7.95	g = 0.56
E3	p = 0.048	p < 0.001	p = 0.005	p < 0.001
	g = -2.87	g = 33.36	g = 4.25	g = 14.64
E4	p = 0.013	p = 0.012	p = 0.004	p = 0.418
	g = -2.79	g = 4.14	g = 3.93	g = 0.65
E5	p = 0.003	p < 0.001	p < 0.001	p < 0.001
	g = -7.11	g = 14.20	g = 19.58	g = 17.49

Nota. Prueba t de Welch con n = 3 corridas por controlador; g de Hedges positivo indica un valor mayor con OpenDaylight.

4.7. Discusión de los resultados

Los resultados permiten dimensionar el impacto del ataque tcp_exhaust sobre un servicio de streaming multi-viewer implementado en una red SDN y contrastar el comportamiento de dos controladores ampliamente utilizados en la comunidad SDN: OpenDaylight y Ryu.

En primer lugar, el ataque tcp_exhaust degrada la calidad del servicio desde la intensidad más baja: con un solo worker, el RTT medio pasa de 28.90 a 56.45 ms con OpenDaylight y de 20.56 a 28.83 ms con Ryu, y la pérdida pasa de valores inferiores al 0.1 % a 2.19 % y 1.58 %, respectivamente. La degradación se intensifica con el número de workers: en E5, la pérdida alcanza 10.75 % con OpenDaylight y 4.30 % con Ryu, el throughput efectivo cae a 0.85 Mbps con OpenDaylight y 1.73 Mbps con Ryu, y los paquetes intercambiados con el cliente disminuyen un 54.7 % y un 17.6 % respecto a la línea base de cada controlador.

En segundo lugar, el RTT es el indicador que mejor separa a los controladores: mientras que la línea base registra 28.90 ms (OpenDaylight) y 20.56 ms (Ryu), con OpenDaylight se alcanzan 79.58, 84.32 y 92.69 ms en E3, E4 y E5, respectivamente, y con Ryu, 45.85, 56.07 y 64.36 ms, con diferencias significativas en todos los escenarios. El RTT medio no supera el umbral de 100 ms adoptado a partir de la Recomendación ITU-T Y.1541 en ningún escenario, pero el percentil 95 lo rebasa con OpenDaylight desde E2 y con Ryu desde E4.

En tercer lugar, el jitter se mantiene por debajo del umbral de 50 ms de la Y.1541 en todos los escenarios y con ambos controladores, aunque en los escenarios E3 a E5 los valores medios con OpenDaylight (6.0–10.4 ms) superan entre 1.3 y 1.5 veces los observados con Ryu (4.1–6.8 ms). Esto sugiere que la variabilidad temporal entre paquetes no es el factor limitante principal en este tipo de ataque y que la degradación se concentra en la latencia y el throughput del servicio.

En cuarto lugar, la comparación entre controladores muestra que, en las condiciones experimentales evaluadas, Ryu presenta una menor degradación que OpenDaylight frente al ataque tcp_exhaust (véase la sección “Análisis estadístico”): en los escenarios E3 a E5, Ryu mantiene un RTT entre 1.4 y 1.7 veces menor y entrega al cliente entre 1.2 y 2.0 veces más throughput efectivo que OpenDaylight con la misma carga de ataque (Tabla 32). Este hallazgo tiene implicaciones prácticas para el diseño de redes SDN destinadas a servicios de streaming:

la selección del controlador influye en la resiliencia del servicio frente a ataques de agotamiento de conexiones, y la arquitectura de Ryu resulta ventajosa cuando no existen mecanismos específicos de mitigación. No obstante, estos resultados provienen de un entorno experimental virtualizado en GNS3 sobre un único anfitrión, por lo que su validez externa debe interpretarse a la luz de la equivalencia entre los parámetros escalados del laboratorio y los de un escenario de referencia de mayor escala (Tablas 27 a 29).

Estos resultados son coherentes en su tendencia con los de [33], que, en un entorno emulado en GNS3 con OpenDaylight, observa que los ataques de denegación de servicio duplican el consumo de ancho de banda y elevan la latencia de los servicios afectados. La magnitud absoluta no es directamente comparable, porque dicho estudio evalúa servidores HTTP y DHCP bajo ataques DoS internos, mientras que en este trabajo se mide un servicio HLS bajo un ataque distribuido de agotamiento de conexiones y el RTT se estima a partir de los ACK de TCP. A diferencia de [33], que evalúa un único controlador, este trabajo muestra que, bajo el mismo ataque, la elección del controlador modifica de forma significativa la latencia, el jitter y la pérdida percibidos por el cliente.

Estos resultados deben interpretarse considerando las amenazas a la validez del estudio: (i) todos los componentes se ejecutan en un único equipo anfitrión con sobreasignación de CPU (18 vCPU sobre 12 hilos), por lo que la mayor demanda de recursos de la JVM de OpenDaylight puede contribuir a las diferencias observadas; (ii) el throughput efectivo se mide en el punto de captura del Open vSwitch y no en el reproductor del cliente, por lo que no refleja directamente el estado de su búfer de reproducción; (iii) el RTT, el jitter y la pérdida se estiman a partir de capturas TCP y no equivalen estrictamente a los parámetros IPTD, IPDV e IPLR de la Y.1541; y (iv) cada escenario cuenta con tres corridas, lo que limita la potencia de las pruebas estadísticas.

Finalmente, los resultados justifican la necesidad de incorporar mecanismos activos de mitigación de DDoS, ya que incluso con el controlador de mejor desempeño (Ryu) la pérdida supera el umbral de 0.1 % en todos los escenarios con ataque y el percentil 95 del RTT rebasa los 100 ms en E4 y E5. La sección “Trabajo futuro: propuesta preliminar de mitigación” plantea reglas de filtrado basadas en las características del tráfico de ataque observadas en este capítulo.

4.8. Proyección hacia escenarios de mayor escala y limitaciones

Los resultados del laboratorio virtualizado se rigen por los factores de escala definidos en el Capítulo III. Esta sección aplica dichos factores para proyectar las métricas de calidad de servicio hacia un escenario de referencia de mayor escala, con el objetivo de estimar el comportamiento de cada controlador ante un ataque DDoS de magnitud proporcional.

4.8.1. Limitaciones metodológicas de la proyección

La proyección de esta sección debe interpretarse con cautela, pues no se sustenta en un modelo matemático de escalamiento formalmente validado, sino en una extrapolación de primer orden basada en el supuesto de invariancia de la degradación relativa: se asume que el porcentaje de degradación de cada métrica, por ser adimensional, se conserva de forma aproximada cuando se preserva la proporción entre recursos y carga. Este supuesto no se ha verificado empíricamente en una infraestructura de mayor escala, por lo que los valores proyectados constituyen estimaciones indicativas y no mediciones. En particular, la proyección presenta las siguientes limitaciones: (i) los resultados provienen de un único equipo anfitrión, por lo que no capturan efectos propios de infraestructuras distribuidas, como la congestión en enlaces WAN, la latencia entre nodos geográficamente separados o la contención de recursos entre servicios; (ii) no se ajusta un modelo de regresión ni se estiman intervalos de confianza sobre los factores de degradación, de modo que la incertidumbre de las proyecciones no está cuantificada; (iii) se supone proporcionalidad de la degradación fuera del rango efectivamente medido (E1 a E5), supuesto cuya validez disminuye a medida que aumentan los factores de

escala; y (iv) el comportamiento de la CPU y de la memoria a gran escala no se modela de forma explícita. Por estas razones, las proyecciones deben entenderse como una aproximación orientada a dimensionar tendencias, y no como una predicción cuantitativa validada del comportamiento del sistema a mayor escala.

4.8.2. Metodología de proyección

La proyección se fundamenta en el principio de similitud de carga: si la relación entre la capacidad del sistema (ancho de banda, CPU y memoria) y la intensidad del ataque (workers, hilos y conexiones concurrentes) se mantiene proporcional, las métricas relativas de degradación (porcentaje de caída del throughput y factor de incremento del RTT) pueden extrapolarse al escenario de referencia. Formalmente, para cada métrica M se define la degradación relativa con la ecuación (14):

$$\text{Degradación relativa } (\%) = ((M_{\text{línea base}} - M_{\text{ataque}}) \,/\, M_{\text{línea base}}) \times 100 \qquad (14)$$

Se supone que esta degradación relativa, por ser adimensional, se conserva entre el entorno escalado y el escenario de referencia de mayor escala siempre que la relación entre recursos y carga sea proporcional. A partir de ella, los valores absolutos en el escenario de referencia se estiman con la ecuación (15):

$$M_{\text{referencia}} = M_{\text{línea base, referencia}} \times (1 - \text{Degradación relativa} \,/\, 100) \qquad (15)$$

4.8.3. Parámetros de escalamiento aplicados

Los factores de escala aplicados entre el laboratorio y el escenario de referencia de mayor escala son los recogidos en la Tabla 28 del Capítulo III.

4.8.4. Proyección del throughput

En el laboratorio, la línea base registra un throughput efectivo medio de 2.42 Mbps (OpenDaylight) y 2.48 Mbps (Ryu), correspondiente a un stream H.264 a 480p con una tasa de video de 800 kbps y 4 espectadores. En el escenario de referencia de mayor escala, con 100 espectadores que consumen un stream H.264 High a 1080p de 5000 kbps, el throughput de

línea base equivalente es de aproximadamente 520 Mbps, conforme a los parámetros de referencia del Capítulo III.

Al aplicar las variaciones relativas del throughput efectivo observadas en cada escenario, el throughput proyectado con OpenDaylight es de 542, 444, 303 y 183 Mbps en E2, E3, E4 y E5, respectivamente, y con Ryu, de 492, 521, 456 y 363 Mbps.

Los resultados proyectados indican que, en un escenario de referencia con 100 espectadores, el caudal útil entregado con OpenDaylight quedaría por debajo de la demanda agregada de 520 Mbps en E3, E4, E5, y con Ryu, en E2, E4, E5, con una degradación sensiblemente menor en este último.

4.8.5. Proyección de la latencia (RTT)

La latencia de la línea base del laboratorio es de 28.90 ms con OpenDaylight y de 20.56 ms con Ryu. En un escenario de referencia de mayor escala, la latencia base depende de la distancia geográfica entre el cliente y el servidor; para un despliegue en red local o en una CDN se estima entre 10 y 20 ms [79]. Al aplicar los factores de incremento observados se obtiene:

Con una latencia base de 10 a 20 ms, el RTT proyectado con OpenDaylight es de 20–39, 28–55, 29–58 y 32–64 ms en E2, E3, E4 y E5, respectivamente, y con Ryu, de 14–28, 22–45, 27–55 y 31–63 ms.

La proyección indica que, con una latencia base de 10 a 20 ms, ninguno de los controladores superaría en promedio el umbral de 100 ms adoptado a partir de la ITU-T Y.1541 en los escenarios evaluados; la diferencia entre controladores se mantiene, con valores proyectados hasta 1.4 veces mayores con OpenDaylight.

4.8.6. Proyección de la pérdida de paquetes

La pérdida de paquetes, estimada a partir de las retransmisiones TCP, aumenta con la intensidad del ataque. En el laboratorio, la línea base registra tasas de pérdida de 0.02 % a 0.06 %, nivel mínimo propio de la transmisión TCP en la topología emulada, mientras que en los

escenarios con ataque la tasa alcanza valores de entre 1.58 % (Ryu, E2) y 10.75 % (OpenDaylight, E5), muy por encima del umbral de 0.1 % de la Y.1541. En el escenario de referencia de mayor escala, estos porcentajes de pérdida se traducirían en interrupciones visibles del stream (congelamiento de imagen y artefactos de decodificación) que afectarían directamente a la experiencia del usuario.

4.8.7. Síntesis de la proyección

La proyección de los resultados del laboratorio hacia un escenario de referencia de mayor escala sugiere que las tendencias de degradación observadas se mantendrían. El controlador Ryu, que presenta menor degradación en el laboratorio, podría sostener el servicio de streaming ante ataques de baja intensidad (equivalentes a E2 y E3); sin embargo, a partir de intensidades medias (equivalentes a E4), ambos controladores requerirían mecanismos activos de mitigación para preservar la calidad del servicio dentro de los umbrales adoptados. Esta conclusión refuerza la necesidad de reglas de filtrado y protección como las que se proponen en la sección siguiente.

4.9. Trabajo futuro: propuesta preliminar de mitigación

A partir de la caracterización del impacto del ataque tcp_exhaust, se propone un mecanismo de mitigación basado en la instalación directa de reglas de filtrado (DROP) en los Open vSwitch del plano de datos. A diferencia de soluciones más complejas que requieren integrar múltiples componentes de software, esta propuesta se centra en reglas OpenFlow que pueden desplegarse de forma inmediata.

4.9.1. Fundamento de la propuesta

El análisis de los escenarios de ataque muestra que el tráfico malicioso del módulo tcp_exhaust presenta tres características identificables que permiten su filtrado:

- Origen conocido: las direcciones IP de los nodos atacantes (workers de la botnet) son identificables en las estadísticas de tráfico como emisores con volúmenes anómalos.

- Puerto de destino específico: el ataque se dirige al puerto HLS 8888 del servidor de streaming, el mismo que utiliza el cliente legítimo, por lo que no puede bloquearse por puerto sin interrumpir el servicio.
- Patrón de paquetes: predominancia de paquetes pequeños (40-79 bytes) correspondientes a mensajes SYN, ACK y RST que saturan la tabla de conexiones del servidor.

4.9.2. Reglas de filtrado propuestas

Se definen dos tipos de reglas OpenFlow que se instalan directamente en los Open vSwitch del plano de datos mediante el comando ovs-ofctl:

Regla 1: bloqueo por dirección IP de origen (nodo atacante identificado)

```
ovs-ofctl add-flow br0 \   "table=0, priority=100, \
ip, nw_src=192.168.10.98, \    actions=drop" \   -O OpenFlow13
```

Esta regla descarta todo el tráfico proveniente de la dirección IP del nodo atacante identificado (192.168.10.98). Se instala con prioridad 100 para que prevalezca sobre las reglas de reenvío genéricas instaladas por el controlador.

Regla 2: bloqueo de nuevas conexiones de un nodo atacante hacia el servicio HLS

```
ovs-ofctl add-flow br0 \
  "table=0, priority=90, \
   tcp, nw_src=192.168.10.36, nw_dst=192.168.10.66,
tp_dst=8888, tcp_flags=+syn, \
   actions=drop" \
  -O OpenFlow13
```

Esta regla descarta los segmentos SYN con los que un nodo atacante abre nuevas conexiones hacia el puerto HLS 8888, de modo que el agotamiento de conexiones se detiene sin bloquear el puerto que utiliza el cliente legítimo (192.168.10.35). Es una alternativa más

selectiva que la regla 1 y debe instalarse para cada nodo atacante identificado (192.168.10.98, 192.168.10.36, 192.168.10.37 y 192.168.10.38).

4.9.3. Líneas de trabajo complementarias

La mitigación de ataques DDoS en redes SDN constituye un campo de investigación activo en el que se han desarrollado múltiples propuestas dentro del ámbito académico. En particular, se identifican las siguientes líneas de trabajo complementarias:

- Desarrollo de sistemas de detección de intrusiones (IDS) adaptados al plano de control SDN, capaces de identificar patrones de ataque en tiempo real y generar reglas OpenFlow de forma automática.
- Implementación de mecanismos de rate limiting (limitación de tasa) a nivel de OpenFlow que permitan contener el volumen de tráfico malicioso sin bloquear por completo las conexiones legítimas.
- Integración de herramientas de monitoreo como ntopng o Snort con el controlador SDN para establecer un ciclo cerrado de detección-respuesta que no requiera intervención manual.
- Evaluación de controladores SDN de nueva generación (como ONOS) que incorporan módulos nativos de seguridad y gestión de políticas de acceso.

La combinación de las reglas de filtrado estáticas propuestas en este trabajo con las líneas de investigación señaladas permitiría construir una defensa en profundidad que fortalezca la resiliencia de las redes SDN frente a ataques DDoS, tanto en entornos académicos como en despliegues de producción.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

En las condiciones experimentales evaluadas, la degradación de la QoS del servicio HLS se manifestó como un incremento de las retransmisiones y del retardo de las conexiones TCP del cliente, y no como una ausencia de datos en la reproducción. Esto se debe a que el servicio se transporta sobre TCP y a que, en la arquitectura SDN analizada, el reenvío del tráfico depende de las reglas que instala el controlador. Por ello, las métricas de QoS se estimaron a partir de las conexiones TCP del cliente, y su comparación con los umbrales de las Recomendaciones ITU-T Y.1541 y G.1010 tiene carácter indicativo.

La especificación de requerimientos conforme a la norma ISO/IEC/IEEE 29148 permitió seleccionar de forma justificada los componentes del entorno experimental y diseñar escenarios de evaluación controlados y comparables. A partir de esos requerimientos se implementó un laboratorio en GNS3 con los controladores OpenDaylight y Ryu, el servidor de streaming MediaMTX y el monitor de tráfico ntopng, en el que la topología, el direccionamiento, los recursos asignados y los parámetros del flujo de video se mantuvieron constantes en todas las corridas. La integración de la herramienta de ataque TrafficGen SDN completó el entorno y permitió variar de forma controlada la intensidad del ataque, de modo que el controlador y el número de nodos atacantes fueron las únicas variables independientes. Con ello se ejecutaron las 30 corridas experimentales previstas y se cumplieron los objetivos planteados.

Los resultados muestran que el ataque tcp_exhaust degrada la calidad de servicio del streaming desde su intensidad más baja y que la degradación aumenta con el número de nodos atacantes, con ambos controladores. La pérdida de paquetes superó los umbrales de las Recomendaciones ITU-T Y.1541 y G.1010 desde un solo worker y, en la intensidad máxima, alcanzó 10.75 % con OpenDaylight y 4.30 % con Ryu. En ese mismo escenario, el RTT medio

llegó a 92.69 ms y 64.36 ms, y el throughput efectivo cayó un 64.8 % y un 30.2 %, respectivamente. El jitter se mantuvo por debajo de 50 ms, por lo que la pérdida, la latencia y el throughput efectivo fueron los indicadores más sensibles al ataque.

En el entorno evaluado, Ryu presentó menor degradación de la QoS que OpenDaylight frente al ataque tcp_exhaust. Según la prueba t de Welch ($p < 0.05$; $n = 3$ réplicas por grupo), el RTT fue menor con Ryu en los cinco escenarios, el throughput efectivo fue mayor en E3, E4 y E5 y la pérdida fue menor en E3 y E5; en E2 y E4, una réplica atípica de Ryu impidió establecer diferencias en la pérdida. Estos resultados no permiten identificar el mecanismo de la diferencia ni afirmar que un controlador sea intrínsecamente más resistente a los ataques DDoS. Una explicación plausible es la arquitectura más sencilla y ligera de Ryu, hipótesis que debe comprobarse en un entorno de producción real con recursos dedicados para cada controlador.

Recomendaciones

Considerar controladores ligeros, como Ryu, en redes SDN que alojen servicios de streaming y no dispongan de mecanismos de mitigación dedicados, ya que en este estudio Ryu presentó menor degradación de la QoS frente al ataque tcp_exhaust. Esta recomendación se limita al escenario y al tipo de ataque evaluados, por lo que debe confirmarse con estudios adicionales antes de generalizarla.

Implementar y validar experimentalmente el mecanismo de mitigación propuesto, basado en reglas de filtrado OpenFlow, y complementarlo con la detección automática de tráfico anómalo. Ningún controlador evitó por sí solo la degradación: con ambos, la pérdida de paquetes superó los umbrales de referencia desde el escenario con un solo worker.

Repetir la comparación entre OpenDaylight y Ryu en un entorno de producción, con infraestructura física y recursos dedicados para cada controlador, para comprobar si la ventaja asociada a la arquitectura ligera de Ryu se mantiene fuera del laboratorio y contrastar los parámetros escalados con mediciones reales.

Ampliar el estudio a otros controladores SDN, como ONOS, y a otras familias de ataques de denegación de servicio (volumétricos, de protocolo y dirigidos al plano de control), para generalizar las conclusiones sobre la resiliencia de las redes SDN que transportan servicios de streaming.

REFERENCIAS BIBLIOGRÁFICAS

[1] D. A. López and C. Piñeros, “SISTEMA DE SEGURIDAD CONTRA DDOS APLICADO EN UNA RED SDN,” Bogotá, Dec. 2021.

[2] A. Habib, M. M. Hefeeda, and B. K. Bhargava, “Detecting Service Violations and DoS Attacks Detecting Service Violations and DoS Attacks,” 2002.

[3] A. A. Bahashwan, M. Anbar, S. Manickam, T. A. Al-Amiedy, M. A. Aladaileh, and I. H. Hasbullah, “A Systematic Literature Review on Machine Learning and Deep Learning Approaches for Detecting DDoS Attacks in Software-Defined Networking,” May 01, 2023, *MDPI*. doi: 10.3390/s23094441.

[4] K. Prayag Pradipkumar, “Detection of Distributed Denial of Service (DDoS) Attacks in Software Defined Networks (SDN) Using Machine Learning Algorithms.,” 2023.

[5] B. Han, X. Yang, Z. Sun, J. Huang, and J. Su, “OverWatch: A Cross-Plane DDoS Attack Defense Framework with Collaborative Intelligence in SDN,” *Security and Communication Networks*, vol. 2018, 2018, doi: 10.1155/2018/9649643.

[6] M. Conti, A. Gangwal, and M. S. Gaur, “A comprehensive and effective mechanism for DDoS detection in SDN,” in *International Conference on Wireless and Mobile Computing, Networking and Communications*, IEEE Computer Society, Nov. 2017. doi: 10.1109/WiMOB.2017.8115796.

[7] MINTEL, “ESTRATEGIA NACIONAL DE CIBERSEGURIDAD DEL ECUADOR,” 2022.

[8] ONU, “Objetivos de Desarrollo Sostenible (ODS)-Infraestructura .” Accessed: Oct. 18, 2024. [Online]. Available: https://www.un.org/sustainabledevelopment/es/infrastructure/

[9] S. Badotra *et al.*, “A DDoS Vulnerability Analysis System against Distributed SDN Controllers in a Cloud Computing Environment,” *Electronics (Switzerland)*, vol. 11, no. 19, Oct. 2022, doi: 10.3390/electronics11193120.

[10] M. Singh, V. Rathore, V. P. Patel, K. Mishra, R. N. Singh, and R. Bhardwaj, “Software Defined Networking Based Detection of DDoS Attacks,” *International Journal of Emerging Technology and Advanced Engineering*, vol. 13, no. 4, pp. 96–104, Apr. 2023, doi: 10.46338/ijetae0423_11.

[11] M. J. Ormachea Mejía, C. A. Almidón Ortiz, W. E. Vicente Ramos, and L. E. Pacheco Moscoso, “Gestión del tráfico de red en la calidad de servicio ‘QoS’ WAN en Tambopata-Perú 2021,” *Revista de Ciencias Sociales (Ve), vol. XXVIII, núm. 2, 2022*, 2021, Accessed: May 19, 2024. [Online]. Available: https://www.redalyc.org/articulo.oa?id=28070565020

[12] J. Cáceres and C. Casilimas, “Arquitectura y funcionamiento de redes definidas por software (SDN). Universidad Distrital Francisco Jose de Caldas. Especialización Teleinformática,” *Universidad Distrital Francisco Jose de Caldas.*, pp. 1–15, 2021.

[13] G. Pereira and E. Gamess, “Lineamientos para el Despliegue de Redes SDN/OpenFlow,” *Revista Venezolana de Computación*, vol. 4, no. 2, pp. 21–33, 2017, [Online]. Available: https://svc.net.ve/ReVeCom/Vol04-No02/ReVeCom-vol04-no02-p021-033.pdf

[14] L. F. Eliyan and R. Di Pietro, “DoS and DDoS attacks in Software Defined Networks: A survey of existing solutions and research challenges,” *Future Generation Computer Systems*, vol. 122, pp. 149–171, Sep. 2021, doi: 10.1016/j.future.2021.03.011.

[15] D. Morillo, “IMPLEMENTACIÓN DE UN PROTOTIPO DE UNA RED DEFINIDA POR SOFTWARE (SDN) EMPLEANDO UNA SOLUCIÓN BASADA EN SOFTWARE,” Quito- Ecuador, Escuela Politécnica Nacional, 2014.

[16] C. Manuel, R. Vergel, and C. A. Calderón, “Controladores SDN, elementos para su selección y evaluación,” 2014. [Online]. Available: https://www.researchgate.net/publication/320711755

[17] OpenDaylight, “OpenDaylight.” Accessed: Mar. 17, 2026. [Online]. Available: https://www.opendaylight.org/about

[18] OpenDaylight, “Descripción general del controlador OpenDaylight — Documentación de OpenDaylight Documentación de Chromium.” Accessed: Mar. 17, 2026. [Online]. Available: https://docs.opendaylight.org/en/latest/user-guide/opendaylight-controller-overview.html

[19] RYU-SDN, “Marco SDN de Ryu.” Accessed: Mar. 17, 2026. [Online]. Available: https://ryu-sdn.org/

[20] A. J. Pachés, J. Óscar, and R. Martínez, “Estudio del controlador SDN Ryu sobre una Raspberry-Pi Model 4,” Oct. 19, 2020. Accessed: Mar. 17, 2026. [Online]. Available: https://riunet.upv.es/handle/10251/152347

[21] Rao Sridhar, “SDN Series Part Three: NOX, the Original OpenFlow Controller - The New Stack.” Accessed: Mar. 17, 2026. [Online]. Available: https://thenewstack.io/sdn-series-part-iii-nox-the-original-openflow-controller/

[22] RedHat, “Las redes definidas por software (SDN): qué son, ventajas, arquitectura, seguridad y casos prácticos.” Accessed: Feb. 17, 2026. [Online]. Available: https://www.redhat.com/es/topics/hyperconverged-infrastructure/what-is-software-defined-networking

[23] Open Networking Foundation, “OpenFlow Switch Specification, Version 1.3.5 (Protocol version 0x04),” Mar. 2015. [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf

[24] Open Networking Foundation, “OpenFlow Switch Specification Version 1.5.1 (Protocol version 0x06) for information on specification licensing through membership agreements,” 2015. [Online]. Available: http://www.opennetworking.org

[25] IETF, “RFC 7047 - The Open vSwitch Database Management Protocol.” Accessed: Feb. 17, 2026. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7047

[26] Linux Fundation, “ovsdb — Open vSwitch 3.5.90 documentation.” Accessed: Feb. 16, 2025. [Online]. Available: https://docs.openvswitch.org/en/latest/ref/ovsdb.7/

[27] O. E. Tayfour and M. N. Marsono, “Collaborative Detection and Mitigation of Distributed Denial-of-Service Attacks on Software-Defined Network,” *Mobile Networks and Applications*, vol. 25, no. 4, pp. 1338–1347, Aug. 2020, doi: 10.1007/s11036-020-01552-0.

[28] U. Red Sdn, D. Andrés López Naranjo, C. Piñeros Caro, and L. Carlos Trujillo Arboleda, “SISTEMA DE SEGURIDAD CONTRA DDOS APLICADO EN,” 2021.

[29] CloudFlare, “What is a distributed denial-of-service (DDoS) attack? | Cloudflare.” Accessed: Aug. 02, 2025. [Online]. Available: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/

[30] A. Sangodoyin, T. Sigwele, P. Pillai, Y. F. Hu, I. Awan, and J. Disso, “DoS Attack Impact Assessment on Software Defined Networks,” Mar. 2018.

[31] A. Singh and B. B. Gupta, “Distributed Denial-of-Service (DDoS) Attacks and Defense Mechanisms in Various Web- Enabled Computing Platforms: Issues, Challenges, and Future

Research Directions," *Int. J. Semant. Web Inf. Syst.*, vol. 18, no. 1, 2022, doi: 10.4018/IJSWIS.297143.

[32] B. Dhruba and K. Jugal, "Evolution, Detection, Prevention, Reaction, and Tolerance DDoS Attacks," New York, Apr. 2016. doi: https://doi.org/10.1201/b20614.

[33] M. V. Ramos Valencia, M. B. Paredes Regalado, N. P. Layedra Larrea, and S. A. Salazar Cazco, "Evaluación de ataques de denegación de servicio - insider - en redes definidas por software," *Tesla Revista Científica*, vol. 5, no. 1, p. e486, Jun. 2025, doi: 10.55204/trc.v5i1.e486.

[34] M. Barria and E. Hales, "Tecnologia Streaming," Aug. 2013.

[35] W. Mendoza, "¿Cómo funciona el Streaming? - Centro de Ayuda de WEBPERU." Accessed: Aug. 02, 2025. [Online]. Available: https://www.wperu.com/ayuda/como-funciona-el-streaming/

[36] P. Henken, "Video codec: here's what you need to know | Kaltura." Accessed: Jan. 11, 2025. [Online]. Available: https://corp.kaltura.com/blog/video-codec/

[37] T. Ruether, "Protocolos de transmisión: Todo lo que Necesitas Saber | Wowza." Accessed: Jan. 11, 2025. [Online]. Available: https://www.wowza.com/blog/streaming-protocols

[38] A. Poor, "¿Qué es el streaming por Internet y cómo funciona? | Avast." Accessed: May 21, 2023. [Online]. Available: https://www.avast.com/es-es/c-what-is-streaming

[39] IETF, "RFC 7826 - Real-Time Streaming Protocol Version 2.0." Accessed: Sep. 23, 2026. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7826

[40] F. Cores, "Arquitecturas Distribuidas para Sistemas de Video-bajo-Demanda a gran escala," 2003.

[41] J. Kurose and K. Ross, *Redes de computadores y arquitecturas de comunicaciones : supuestos prácticos*. Pearson Prentice Hall, 2005.

[42] E. Intriago, "ANÁLISIS DE TECNOLOGÍAS DE STREAMING: EVALUACIÓN DE PROTOCOLOS Y DISEÑO DE UN CASO DE ESTUDIO," 2016.

[43] D. Jácome, "REDES DEFINIDAS POR SOFTWARE Y SU IMPACTO EN EL USO DE APLICACIONES DE VIDEO," Jun. 2025, [Online]. Available: https://orcid.org/0009-0007-6658-6971

[44] J. LOZADA, " Implementación de una red híbrida SDN/MPLS para la gestión de servicios diferenciados mediante simulación de tiempo real GNS3," Universidad Técnica del Norte, Ibarra, 2022.

[45] "RFC 2544: Benchmarking Methodology for Network Interconnect Devices | RFC Editor." Accessed: Sep. 23, 2026. [Online]. Available: https://www.rfc-editor.org/info/rfc2544/

[46] "RFC 6349: Framework for TCP Throughput Testing | RFC Editor." Accessed: Sep. 23, 2026. [Online]. Available: https://www.rfc-editor.org/info/rfc6349/

[47] ITU-T, "Statistical framework for end-to-end network performance benchmark scoring and ranking," 2018. [Online]. Available: http://handle.itu.int/11.1002/1000/11

[48] Itu-t, "ITU-T Rec. Y.1541 Amendment 1 (12/2013) Network performance objectives for IP-based services Amendment 1: New Appendix XII - Considerations for low speed access networks," 2011. [Online]. Available: http://handle.itu.int/11.1002/1000/11830-en.

[49] ITU-T, "ITU-T End-user multimedia QoS categories," 2001.

[50] GNS3 Staff, "Comenzando con GNS3 | GNS3 Documentación." Accessed: Jan. 19, 2025. [Online]. Available: https://docs.gns3.com/docs/

[51] “Wireshark User’s Guide.” Accessed: Apr. 21, 2026. [Online]. Available: https://www.wireshark.org/docs/wsug_html_chunked/

[52] “ntopng Documentation — ntopng 6.7 documentation.” Accessed: Sep. 23, 2026. [Online]. Available: https://www.ntop.org/guides/ntopng/

[53] S. Engineering Standards Committee of the IEEE Computer Society, “ISO/IEC/IEEE 29148:2011(E), Systems and software engineering — Life cycle processes — Requirements engineering,” 2011. [Online]. Available: www.iso.org

[54] ITU, “ITU-T,” Dec. 2011.

[55] “RFC 2681: A Round-trip Delay Metric for IPPM | RFC Editor.” Accessed: Sep. 23, 2026. [Online]. Available: https://www.rfc-editor.org/info/rfc2681/

[56] H. Schulzrinne *et al.*, “RTP: A Transport Protocol for Real-Time Applications- RFC 3550,” Jul. 2003, doi: 10.17487/RFC3550.

[57] “RFC-7680 A One-Way Loss Metric for IP Performance Metrics (IPPM).” Accessed: Sep. 23, 2026. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7680.html

[58] P. Chimento, J. Ishac, P. Chimento, and J. Ishac, “RFC-5136 Defining Network Capacity,” Feb. 2008, doi: 10.17487/RFC5136.

[59] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An Updated Performance Comparison of Virtual Machines and Linux Containers,” in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Philadelphia, PA, USA: IEEE, Mar. 2015, pp. 171–172. doi: 10.1109/ISPASS.2015.7095802.

[60] “iperf3 — iperf3 3.21 documentation.” Accessed: Apr. 21, 2026. [Online]. Available: https://software.es.net/iperf/

[61] S. Mansfield-Devine, “Anonymous: serious threat or mere annoyance?,” 2011, doi: 10.1016/S1353-4858(11)70004-6.

[62] “Scapy.” Accessed: Apr. 21, 2026. [Online]. Available: https://scapy.net/

[63] “gkbrk/slowloris: Low bandwidth DoS tool. Slowloris rewrite in Python.” Accessed: Apr. 21, 2026. [Online]. Available: https://github.com/gkbrk/slowloris

[64] RedHat, “Chapter 2. What do you need to run OpenDaylight? | Red Hat OpenDaylight Installation and Configuration Guide | Red Hat OpenStack Platform | 13 | Red Hat Documentation.” Accessed: Apr. 16, 2026. [Online]. Available: https://docs.redhat.com/en/documentation/red_hat_openstack_platform/13/html/red_hat_opendaylight_installation_and_configuration_guide/what_do_you_need_to_run_opendaylight

[65] MININET, “Mininet Overview - Mininet.” Accessed: Apr. 16, 2026. [Online]. Available: https://mininet.org/overview/

[66] GNS3, “Cisco IOS images for Dynamips | GNS3 Documentation.” Accessed: Apr. 16, 2026. [Online]. Available: https://docs.gns3.com/docs/emulators/cisco-ios-images-for-dynamips/

[67] Debian Org, “3.4. Cumplimiento de los requisitos mínimos de hardware.” Accessed: Apr. 16, 2026. [Online]. Available: https://www.debian.org/releases/stable/amd64/ch03s04.en.html

[68] J. Moy, “OSPF Version 2,” Apr. 1998. doi: 10.17487/RFC2328.

[69] Docker Inc., “Docker Documentation – Get Started,” 2024.

[70] Docker Inc., “Python – Official Image | Docker Hub,” 2024.

[71] B. Chesneau, “Gunicorn: Python WSGI HTTP Server for UNIX,” 2024.

ANEXO A. DESCRIPCIÓN DE LOS REQUERIMIENTOS DEL MODELO DE SIMULACIÓN

Este anexo describe en detalle los requerimientos del sistema y de la arquitectura de simulación formulados en el Capítulo III (Tablas 7 y 8), según la estructura del estándar ISO/IEC/IEEE 29148:2011; las Tablas A1 y A2 presentan su descripción detallada. Cada requerimiento incluye su nomenclatura, una denominación breve, una descripción que justifica su inclusión en el modelo y un nivel de prioridad (alta, media o baja), determinado según su impacto sobre la operatividad del entorno de simulación, la fidelidad de las mediciones de QoS y la reproducibilidad del experimento.

A.1 Nomenclatura empleada

Descripción	Abreviatura
Requerimientos de Stakeholders	StRS
Requerimientos del Sistema	SyRS
Requerimientos de Arquitectura	SyRA

Tabla A1

Descripción de los requerimientos del sistema (SyRS)

Requerimientos del Sistema (SyRS)			
Nomenclatura	Requerimiento	Descripción	Prioridad
SyRS1	Configuración y despliegue de controladores SDN	El sistema debe permitir la instalación, configuración y despliegue de controladores SDN (como OpenDaylight, ONOS o Ryu) utilizando protocolos estándar de comunicación SDN, principalmente OpenFlow. Esto garantiza que el plano de control pueda gestionar dinámicamente los flujos de la red y que el entorno de simulación reproduzca de forma fiel el comportamiento de una infraestructura SDN real.	Alta
SyRS2	Generación y control de ataques DDoS simulados	El entorno debe proporcionar la capacidad de generar ataques DDoS controlados, permitiendo seleccionar el tipo de ataque (volumétrico, de protocolo o de	Media

Requerimientos del Sistema (SyRS)			
		aplicación) y modular su intensidad y duración. Esta funcionalidad es indispensable para evaluar el impacto del ataque sobre el servidor y el servicio de streaming de forma reproducible.	
SyRS3	Medición en tiempo real de métricas de QoS	El sistema debe medir y registrar en tiempo real las métricas de QoS asociadas al servicio de streaming: latencia (RTT), pérdida de paquetes, jitter y ancho de banda efectivo. La obtención continua de estas métricas permite correlacionar el comportamiento del ataque con la degradación percibida por el usuario, según los umbrales de la recomendación ITU-T Y.1541.	Alta
SyRS4	Generación de tráfico de red de fondo	El sistema debe permitir la inyección de tráfico de red de fondo con patrones configurables (constantes, ráfagas, distribuciones probabilísticas) para simular condiciones realistas de uso de la red. Esto evita que la evaluación se	Media

Requerimientos del Sistema (SyRS)			
		realice sobre un entorno ideal y aporta validez externa al experimento.	
SyRS5	Correlación entre ataques DDoS y métricas de QoS	El sistema debe proporcionar mecanismos de correlación temporal entre los patrones del ataque DDoS lanzado y la degradación observada en las métricas de QoS. Esta capacidad es clave para identificar relaciones causales y construir las gráficas de impacto que dan soporte al análisis cuantitativo del trabajo.	Media
SyRS6	Disponibilidad del entorno experimental	El sistema debe mantener un tiempo de actividad (uptime) cercano al 99.9 % durante la ejecución de los experimentos. La estabilidad del entorno virtualizado es necesaria para que los resultados sean comparables entre escenarios y para que la pérdida de paquetes observada provenga del ataque y no de fallos de la plataforma.	Alta

Requerimientos del Sistema (SyRS)			
SyRS7	Aislamiento del entorno de pruebas	El entorno de simulación debe estar completamente aislado de redes externas. Esto es indispensable por razones éticas y de seguridad, ya que se generan ataques DDoS que, fuera del laboratorio, constituirían una conducta ilícita y podrían afectar a terceros.	Media
SyRS8	Generación de reportes de análisis	El sistema debe generar reportes detallados que correlacionen los ataques DDoS ejecutados con las métricas de QoS registradas. Estos reportes constituyen la evidencia técnica del trabajo y permiten replicar la investigación en otras instituciones académicas.	Alta
SyRS9	Compatibilidad con el hardware disponible	El sistema debe ser compatible con las capacidades de hardware disponibles para el desarrollo del proyecto, en particular en términos de CPU con soporte de virtualización, memoria RAM y almacenamiento. Esta	Alta

Requerimientos del Sistema (SyRS)			
		restricción asegura que el modelo sea ejecutable sin requerir infraestructura especializada adicional.	

Tabla A2

Descripción de los requerimientos de la arquitectura de simulación (SyRA)

Requerimientos de la Arquitectura de Simulación (SyRA)			
Nomenclatura	Requerimiento	Descripción	Prioridad
		Requerimientos del equipo host	
SyRA1	CPU con soporte de virtualización	El equipo host debe contar con un procesador Intel o AMD que disponga de extensiones de virtualización por hardware (VT-x en Intel o AMD-V). Estas tecnologías son indispensables para ejecutar de forma eficiente las máquinas virtuales y contenedores empleados por GNS3, Docker y el controlador SDN.	Alta
SyRA2	Memoria RAM mínima de 16 GB	El host debe disponer de al menos 16 GB de memoria RAM para soportar simultáneamente el sistema operativo anfitrión, el	Alta

Requerimientos de la Arquitectura de Simulación (SyRA)

		controlador SDN, los nodos virtuales del topología en GNS3, el servidor de streaming y la herramienta de ataque DDoS sin saturación.	
SyRA3	Almacenamiento SSD mínimo de 100 GB libres	El equipo debe disponer de al menos 100 GB libres en un disco de estado sólido (SSD). El uso de SSD reduce los tiempos de arranque de las máquinas virtuales y mejora el rendimiento de las operaciones de escritura asociadas a la captura de tráfico y a los registros de QoS.	Alta
SyRA4	Sistema Operativo Windows 10/11 o Linux	El sistema operativo anfitrión debe ser Windows 10/11 (64 bits) o una distribución Linux moderna compatible con GNS3, Docker y las herramientas de medición. Esta flexibilidad facilita la reproducción del entorno en distintos laboratorios académicos.	Media
SyRA5	Tarjeta de red de 1 Gbps	El host debe disponer de una interfaz de red de 1 Gbps para la conectividad externa requerida durante la descarga de imágenes, actualizaciones y, en su caso, conexión con servicios	Alta

Requerimientos de la Arquitectura de Simulación (SyRA)			
		complementarios. Esto evita que el ancho de banda del host se convierta en un cuello de botella.	
Requerimientos del software de simulación			
SyRA6	Soporte SDN/OpenFlow	La herramienta de simulación debe ofrecer soporte nativo para entornos SDN y para el protocolo OpenFlow, de modo que se pueda integrar un controlador externo y verificar el comportamiento del plano de control durante el ataque.	Alta
SyRA7	Capacidad de virtualización	El software debe permitir la virtualización de equipos de red (switches y routers) y la integración con contenedores o máquinas virtuales para los nodos finales (cliente, servidor de streaming y atacante), permitiendo construir topologías realistas.	Alta
SyRA8	Integración con herramientas de monitoreo	El simulador debe permitir la integración con herramientas externas de monitoreo y captura de tráfico (Wireshark, sFlow, etc.), elemento clave para registrar las métricas de QoS necesarias en el análisis.	Alta

Requerimientos de la Arquitectura de Simulación (SyRA)			
SyRA9	Óptimo consumo de recursos	El software de simulación debe mostrar un consumo razonable de CPU y memoria para que el host pueda ejecutar simultáneamente todos los componentes del modelo sin sobrecarga.	Alta
SyRA10	Facilidad de uso	La herramienta debe disponer de una interfaz intuitiva que facilite la construcción y modificación de topologías, así como la repetición de experimentos en distintas configuraciones.	Alta
SyRA11	Documentación y soporte	La herramienta seleccionada debe contar con documentación oficial y una comunidad activa, lo cual reduce el tiempo de resolución de problemas y asegura la trazabilidad de las decisiones técnicas tomadas en el proyecto.	Alta
Requerimientos del servidor de streaming			
SyRA12	Facilidad de configuración	El servidor de streaming debe ser sencillo de configurar y desplegar dentro del entorno virtualizado, permitiendo modificar parámetros como bitrate, códecs y resolución sin necesidad de un proceso de instalación complejo.	Alta

Requerimientos de la Arquitectura de Simulación (SyRA)			
SyRA13	Control de los parámetros de despliegue	El servicio debe ofrecer control sobre los parámetros de despliegue (puertos, protocolos, tipo de transporte), de modo que el experimento pueda reproducirse de manera controlada en cada escenario.	Alta
SyRA14	Documentación	El servidor de streaming debe disponer de documentación oficial actualizada que permita configurar el servicio y diagnosticar problemas durante los experimentos.	Media
SyRA15	Consumo de recursos	El servidor debe presentar un consumo de recursos compatible con la ejecución simultánea del resto de componentes del modelo en el host.	Media
SyRA16	Uso académico	La solución empleada debe permitir su uso con fines académicos sin restricciones de licenciamiento, lo cual habilita su replicación en otros trabajos de titulación e investigaciones de la institución.	Media
Requerimientos de la herramienta de ataques DDoS			

Requerimientos de la Arquitectura de Simulación (SyRA)			
SyRA17	Lanzamiento de varios tipos de ataque	La herramienta debe permitir lanzar distintos tipos de ataque DDoS (volumétricos, de protocolo y de aplicación) para evaluar cómo cada uno afecta de forma diferenciada al servidor y al servicio de streaming.	Alta
SyRA18	Control y configuración de la herramienta	Debe ser posible configurar parámetros del ataque tales como intensidad, número de fuentes, duración y patrón temporal, de modo que el experimento pueda variar de forma sistemática las condiciones de prueba.	Alta
SyRA19	Integración con variedad de entornos	La herramienta debe integrarse con el entorno virtualizado GNS3/Docker, ejecutándose desde un nodo atacante dentro de la topología sin requerir configuraciones específicas del host.	Alta
SyRA20	Orquestación distribuida y de monitoreo integrado del servicio	Permite tener un C2 que controla a los bots, además de permitir visualizar el estado de las conexiones.	Alta
SyRA21	Documentación	La herramienta debe contar con documentación que respalde su uso académico y facilite la reproducibilidad del experimento.	Alta

Requerimientos de la Arquitectura de Simulación (SyRA)			
Requerimientos del controlador SDN			
SyRA22	Soporte de protocolo OpenFlow	El controlador SDN seleccionado debe ofrecer soporte completo del protocolo OpenFlow para interactuar con los switches virtuales de la topología y reproducir el entorno SDN sobre el que opera el servicio de streaming evaluado.	Alta
SyRA23	Código abierto	El controlador debe ser de código abierto, lo que asegura la transparencia del comportamiento del plano de control y permite ajustar parámetros internos durante los experimentos.	Alta
SyRA24	Multiplataforma	El controlador debe poder ejecutarse en distintos sistemas operativos (Windows y Linux) o, en su defecto, dentro de un contenedor, garantizando la portabilidad del entorno de simulación.	Alta
SyRA25	API REST	El controlador debe exponer una API REST que permita consultar el estado de la red y los flujos en tiempo de experimento, facilitando la integración con scripts de medición y análisis.	Alta

Requerimientos de la Arquitectura de Simulación (SyRA)			
SyRA26	Interfaz Web	Disponer de una interfaz web es deseable para inspeccionar la topología y verificar el estado del controlador durante los experimentos, aunque no es estrictamente indispensable.	Media
SyRA27	Documentación	El controlador debe contar con documentación oficial actualizada que permita diagnosticar el comportamiento del plano de control bajo carga durante el ataque.	Alta
SyRA28	Multi-thread	El controlador debe soportar procesamiento multi-hilo para gestionar de forma realista las solicitudes Packet-In recibidas durante un ataque DDoS	Alta

ANEXO B. MANUAL DE DESPLIEGUE DE HERRAMIENTAS

Instalación del controlador SDN

El procedimiento de instalación descrito en esta sección se aplica de forma independiente a los dos controladores del estudio, OpenDaylight y Ryu. Ambos se despliegan sobre la misma máquina virtual y se integran en la topología definida en la sección “Diseño del sistema”. El resto del escenario se mantiene idéntico entre ambos casos de estudio, tal como se describe en la sección “Topología de red”.

Instalación de los controladores SDN en la máquina virtual

Los controladores se instalan sobre el sistema operativo Ubuntu 22.04, por su compatibilidad con ambos entornos. A continuación, se describe el procedimiento seguido para cada uno.

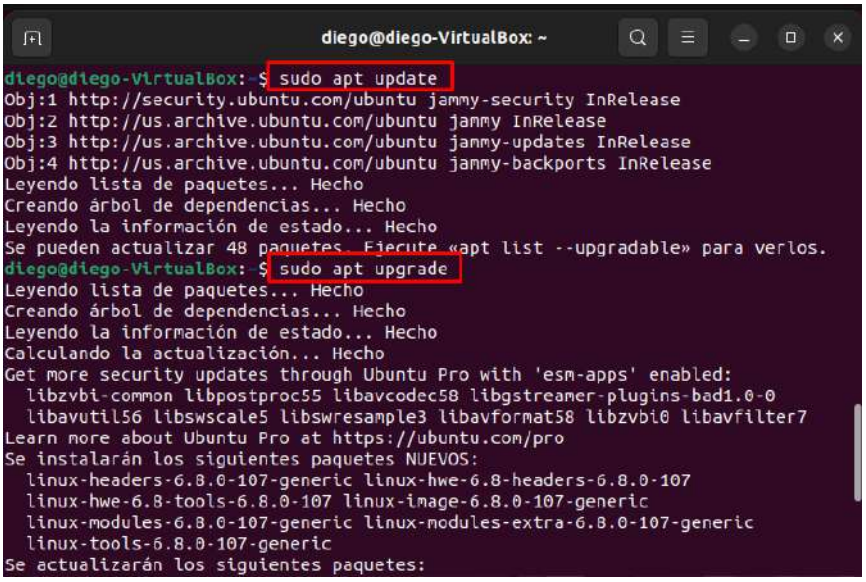
Controlador OpenDaylight

El controlador OpenDaylight (ODL) es una plataforma de código abierto basada en el framework Apache Karaf, diseñada para la gestión centralizada de redes definidas por software (SDN). Su instalación se realiza a partir de la distribución oficial disponible en la documentación de ODL (https://docs.opendaylight.org). A continuación, se describe el procedimiento ejecutado sobre Ubuntu, con cada comando utilizado y su propósito [17].

1. Antes de instalar el controlador, se prepara el sistema operativo: se actualizan los repositorios y paquetes del sistema mediante los comandos `sudo apt update` y `sudo apt upgrade`, como se evidencia en la Figura 71.

Figura 71

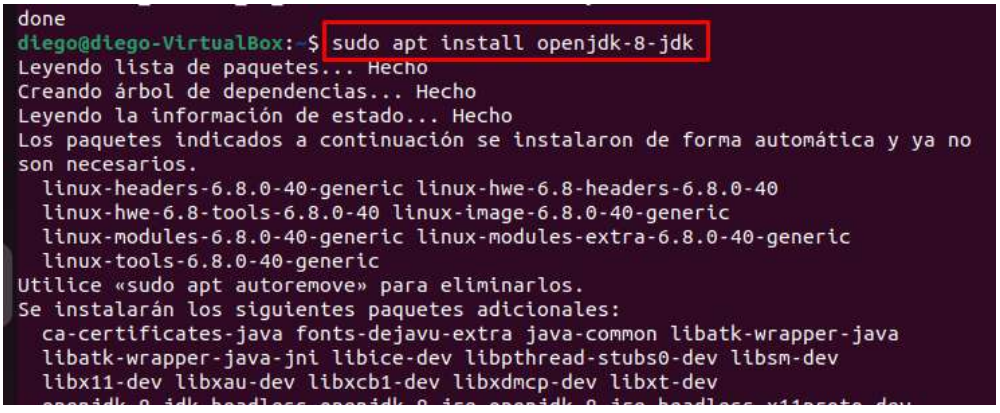
Prerrequisitos del sistema para la instalación de ODL



2. Dado que OpenDaylight requiere Java 8 para ejecutar Karaf, se instala OpenJDK 8 mediante el comando `sudo apt install openjdk-8-jdk`, cuya ejecución se ilustra en la Figura 72. Posteriormente, se ejecuta `java -version` para comprobar que la versión instalada es la 1.8.0.

Figura 72

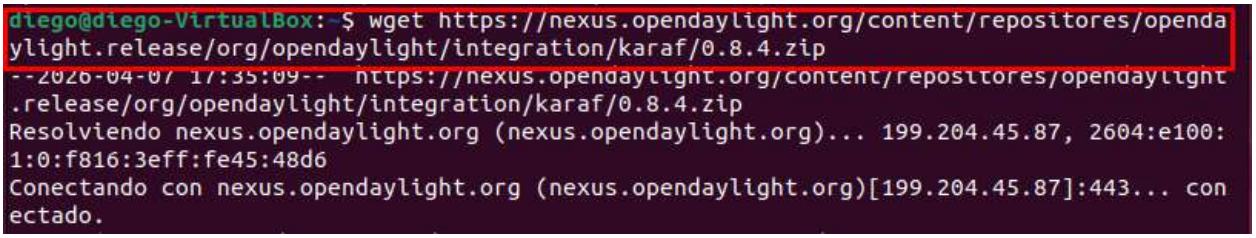
Instalación de Java 8



3. A continuación, se obtiene la distribución del controlador OpenDaylight con la herramienta de línea de comandos wget, que descarga el archivo comprimido directamente desde el repositorio Nexus de OpenDaylight. En concreto, se descarga el archivo karaf-0.8.4.zip, que contiene la distribución completa del contenedor Apache Karaf con los artefactos del controlador. La Figura 73 muestra la ejecución de dicho comando y el establecimiento de la conexión para la descarga.

Figura 73

Descarga de la distribución de ODL mediante wget



4. Una vez finalizada la descarga, el archivo se descomprime mediante el comando `unzip karaf-0.8.4.zip`, que extrae la estructura de directorios del controlador, incluidos los binarios de ejecución, los archivos de configuración y las bibliotecas necesarias. La Figura 74 presenta la salida de este comando.

Figura 74

Descompresión del directorio de Karaf

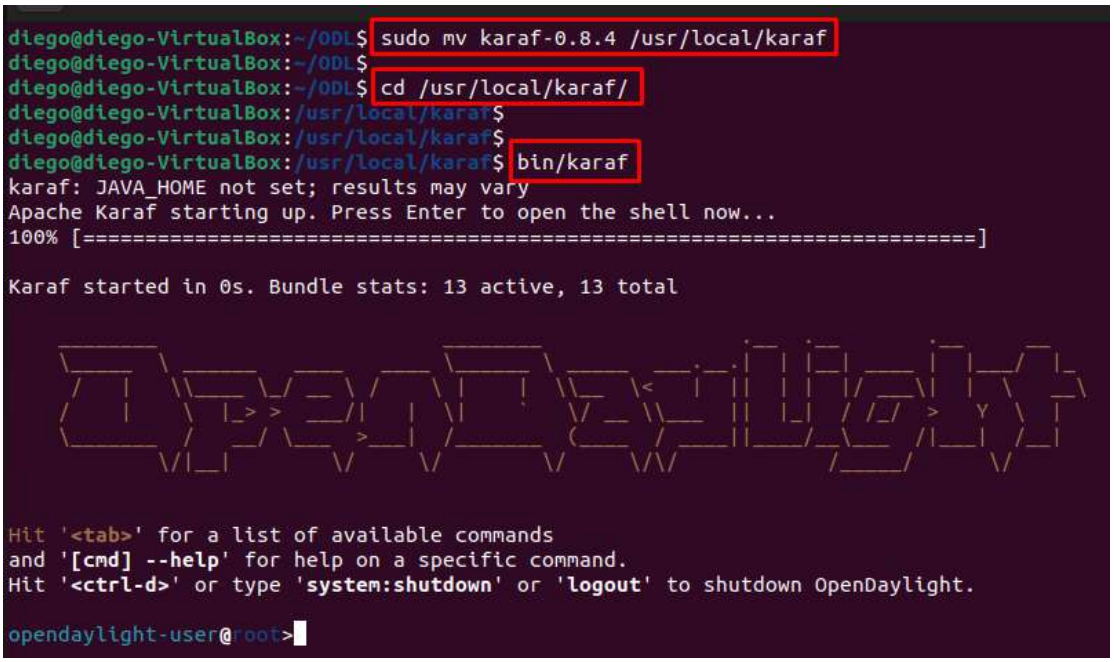


5. Con el objetivo de establecer una ubicación estándar para la instalación, el directorio resultante se traslada a la ruta /usr/local/karaf mediante el comando `sudo`

`mv karaf-0.8.4 /usr/local/karaf`, según las convenciones habituales de los sistemas Linux. Posteriormente, se accede al directorio de instalación con `cd /usr/local/karaf/` y se ejecuta el binario del contenedor con el comando `bin/karaf`. Como resultado, el sistema despliega la consola de Apache Karaf (Figura 75), lo que indica que el controlador OpenDaylight se inicia correctamente.

Figura 75

Instalación y ejecución del controlador ODL

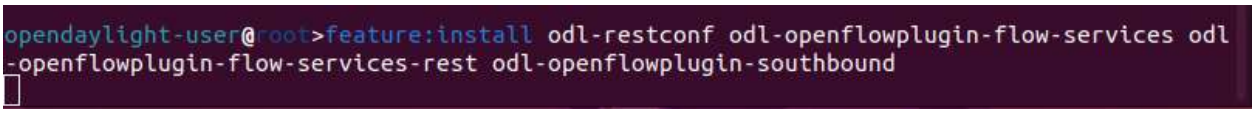


6. Con el controlador en ejecución, se instalan las características (features) que habilitan las funcionalidades requeridas para la gestión de la red SDN. En primer lugar, se instala el conjunto de features relacionadas con el protocolo OpenFlow y la interfaz northbound, ejecutando desde la consola de Karaf el comando `feature:install odl-restconf odl-openflowplugin-flow-services odl-openflowplugin-flow-services-rest odl-openflowplugin-southbound`. Este comando, cuya ejecución se registra en la Figura 76, habilita la interfaz RESTCONF, que provee una API basada en HTTP para la comunicación con aplicaciones externas (northbound); los servicios de

gestión de flujos del plugin OpenFlow, que permiten programar las tablas de flujo; y la interfaz southbound, que establece la comunicación directa entre el controlador y los dispositivos de red compatibles con OpenFlow.

Figura 76

Instalación de las características funcionales (features)



7. Para disponer de una interfaz gráfica que facilite la administración y el monitoreo visual de la red, se instala el módulo DLUX (OpenDaylight User Experience) mediante el comando `feature:install odl-dlux-core odl-dluxapps-topology odl-dluxapps-yangui odl-dluxapps-yangvisualizer`. Como se aprecia en la Figura 77, este comando despliega el núcleo de la interfaz web DLUX junto con tres módulos complementarios: el visualizador de topología, que muestra gráficamente la estructura de la red y los dispositivos conectados; la interfaz YANG UI, que permite interactuar con el modelo de datos del controlador mediante formularios web; y el visualizador YANG, que facilita la exploración jerárquica de los módulos YANG del controlador.

Figura 77

Módulos de la interfaz web DLUX y YANG



Controlador Ryu

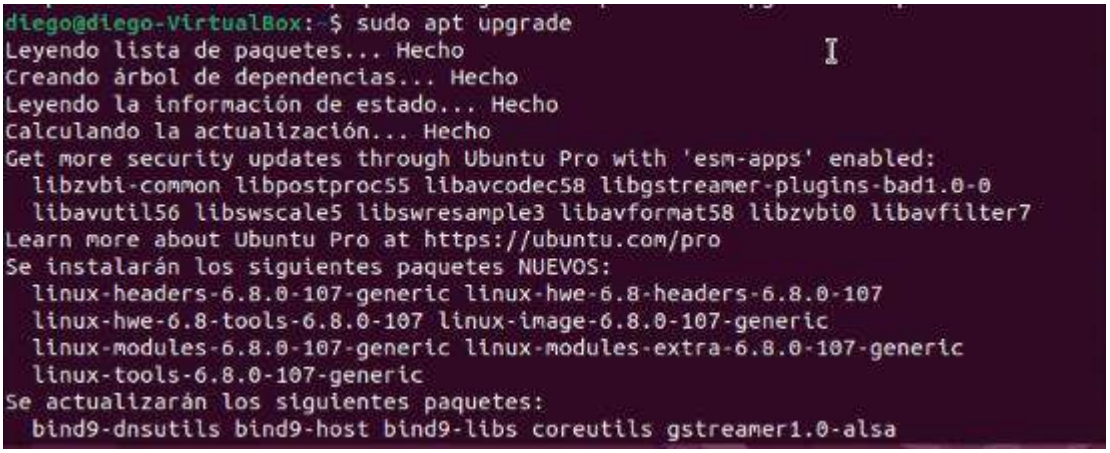
En este apartado se detalla el procedimiento de instalación y puesta en funcionamiento del controlador Ryu sobre Ubuntu, virtualizado mediante Oracle VirtualBox. El proceso comprende la

preparación del entorno, la instalación de las dependencias necesarias y la ejecución inicial del controlador con sus aplicaciones básicas; cada etapa se acompaña de las capturas de pantalla que evidencian la ejecución de los comandos.

1. De manera análoga al procedimiento de OpenDaylight, el primer paso consiste en actualizar el sistema operativo para asegurar la compatibilidad de los paquetes y disponer de las versiones más recientes de las dependencias; para ello, se ejecuta el comando `sudo apt update` y, a continuación, el comando `sudo apt upgrade` para instalar las actualizaciones disponibles. La Figura 78 refleja este proceso, con la resolución de dependencias y la descarga de los paquetes actualizados del kernel y de las bibliotecas del sistema.

Figura 78

Actualización del sistema operativo y de las aplicaciones

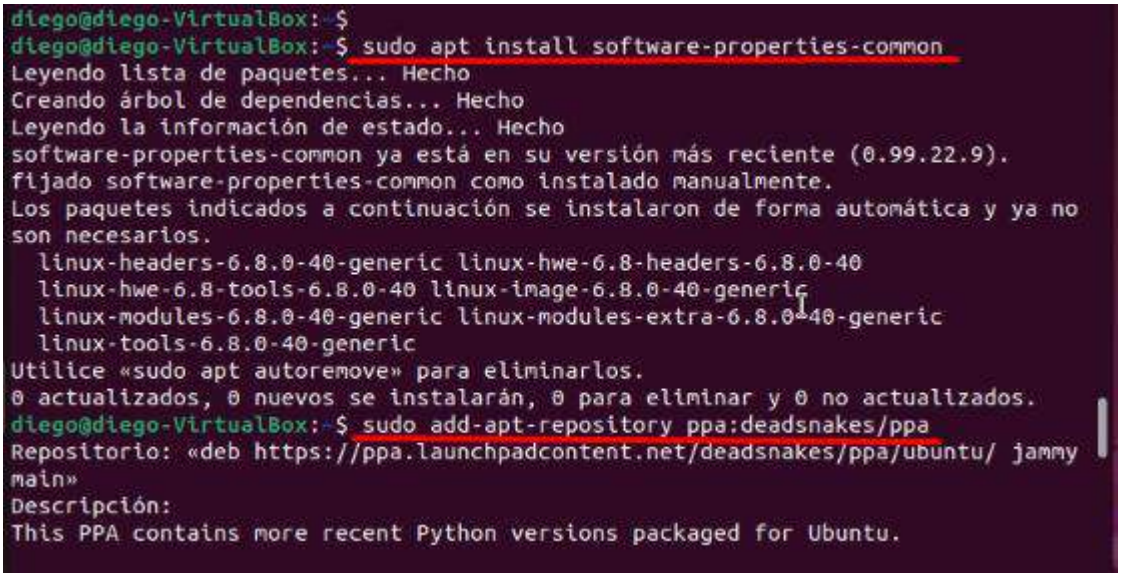


2. Dado que el framework Ryu está desarrollado en Python, se requiere una versión compatible del intérprete; en particular, Python 3.9 ofrece la compatibilidad necesaria con las bibliotecas del controlador. Se instala el paquete software-properties-common mediante el comando `sudo apt install software-properties-common`, que proporciona las herramientas para gestionar

repositorios PPA (Personal Package Archive), y se añade el repositorio deadsnakes mediante `sudo add-apt-repository ppa:deadsnakes/ppa`, ampliamente utilizado para obtener versiones adicionales del intérprete. La Figura 79 evidencia ambos procedimientos.

Figura 79

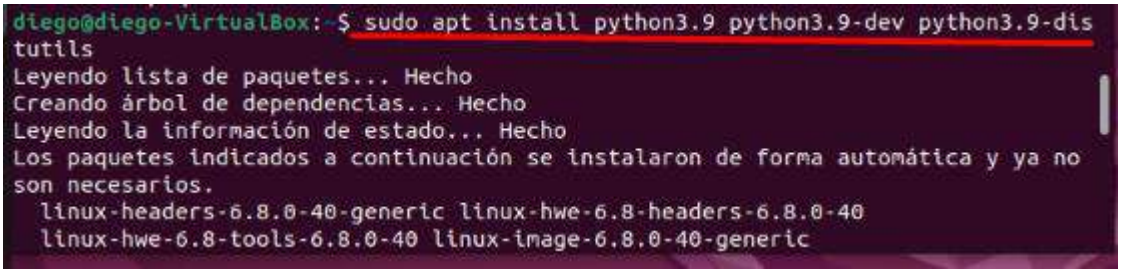
Instalación del intérprete Python 3.9 y de sus dependencias



3. El comando `sudo apt install python3.9 python3.9-dev python3.9-distutils` instala de forma simultánea el intérprete, las cabeceras de desarrollo (necesarias para la compilación de extensiones nativas en C) y las utilidades de distribución de paquetes. La Figura 80 registra su ejecución, con la resolución de las dependencias y el inicio de la descarga.

Figura 80

Instalación de los componentes de Python



4. Con el intérprete de Python instalado, el siguiente paso consiste en disponer del gestor de paquetes pip, necesario para obtener e instalar bibliotecas del ecosistema Python. Para ello, se descarga el script oficial de instalación de la Python Packaging Authority (PyPA) mediante el comando `curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py`. Posteriormente, se instalan el framework Ryu y la biblioteca eventlet con `sudo pip3 install ryu eventlet`. La biblioteca eventlet es esencial para el controlador, ya que implementa el modelo de concurrencia basado en hilos verdes (green threads) que Ryu emplea para gestionar de forma eficiente las conexiones simultáneas con múltiples dispositivos de red. La Figura 81 muestra la descarga del script de pip y la instrucción de instalación de los paquetes requeridos.

Figura 81

Descarga de pip e instalación de los paquetes de Ryu



5. Completada la instalación de todas las dependencias, se inicia el controlador Ryu: desde el directorio de aplicaciones del framework, se ejecuta el comando `ryu-manager` con la carga simultánea de tres aplicaciones: ryu.app.simple_switch_13, que implementa la lógica básica de un conmutador L2 mediante OpenFlow 1.3; ryu.app.rest_conf_switch, que expone una interfaz REST para configurar los parámetros de los conmutadores gestionados; y ryu.app.rest_topology, que habilita el descubrimiento y la consulta de la topología de red mediante peticiones HTTP. La Figura 82 muestra la salida del controlador durante su inicialización, en la que se verifica la carga de cada aplicación y la instanciación de sus contextos.

Figura 82

Ejecución del controlador Ryu y carga de aplicaciones



ANEXO C. RESULTADOS DETALLADOS DEL ESCENARIO E2 (1 WORKER × 20 HILOS)

Este anexo reúne el análisis detallado del escenario E2 (1 worker × 20 hilos, baja intensidad), con el mismo conjunto de métricas de QoS para ambos controladores. La síntesis comparativa se encuentra en la Tabla 32 del Capítulo IV.

Escenario E2: 1 worker × 20 hilos (baja intensidad)

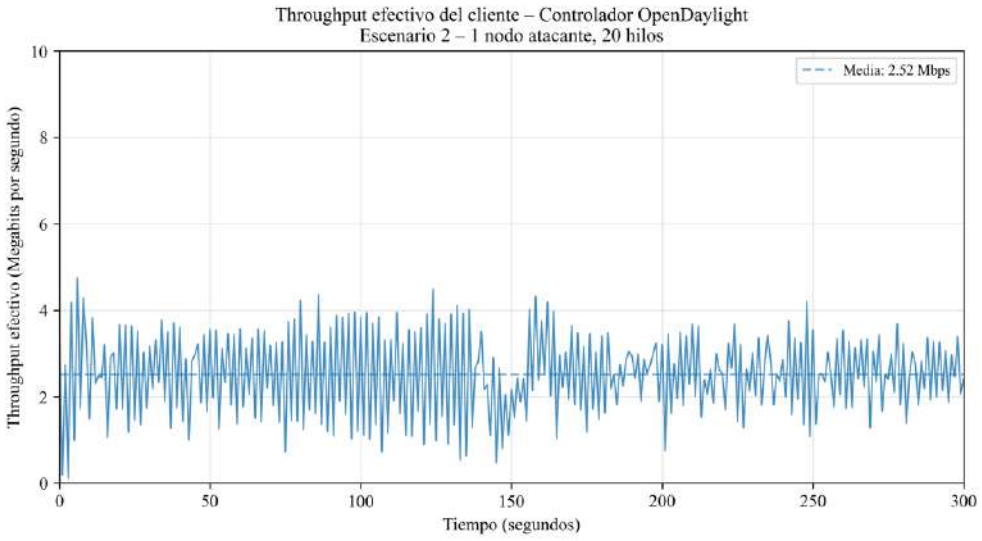
El primer escenario con ataque corresponde a una carga DDoS de baja intensidad sobre el servidor de streaming: un único worker atacante con 20 hilos concurrentes ejecuta el módulo tcp_exhaust contra el servicio HLS del servidor MediaMTX (puerto 8888), el mismo que consume el cliente legítimo. En promedio, cada corrida dura 317.0 s con OpenDaylight y 309.7 s con Ryu, y los valores reportados corresponden al promedio de las tres corridas.

Escenario E2-ODL (1 worker × 20 hilos)

Con 1 worker y OpenDaylight, el throughput efectivo (Figura 83) registra una media de 2.52 Mbps (pico de 4.75 Mbps), un 4.3 % más que la línea base de 2.42 Mbps, mientras que el throughput agregado del enlace, que incluye el tráfico del atacante, alcanza 3.06 Mbps. Los paquetes intercambiados con el cliente se sitúan en 89 235 por corrida (14.5 % respecto a la línea base).

Figura 83

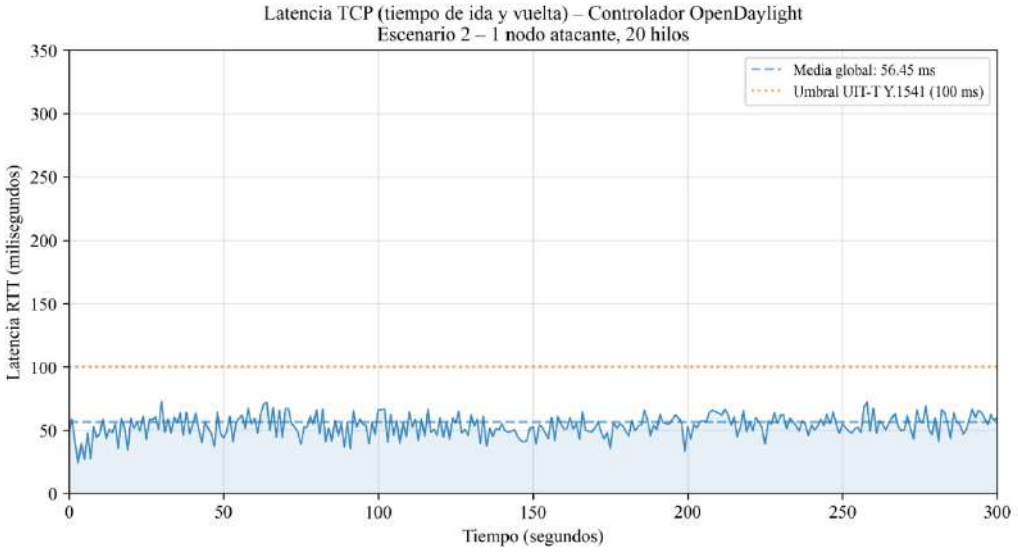
Throughput efectivo durante el ataque tcp_exhaust con OpenDaylight (1 worker × 20 hilos)



Con OpenDaylight, el RTT del ataque de baja intensidad (Figura 84) sube a 56.45 ms de media (percentil 95 de 107.85 ms), 2.0 veces el de la línea base (28.90 ms). La media no rebasa el umbral de 100 ms, aunque el percentil 95 ya lo supera, lo que sitúa a este escenario al inicio de la curva de degradación.

Figura 84

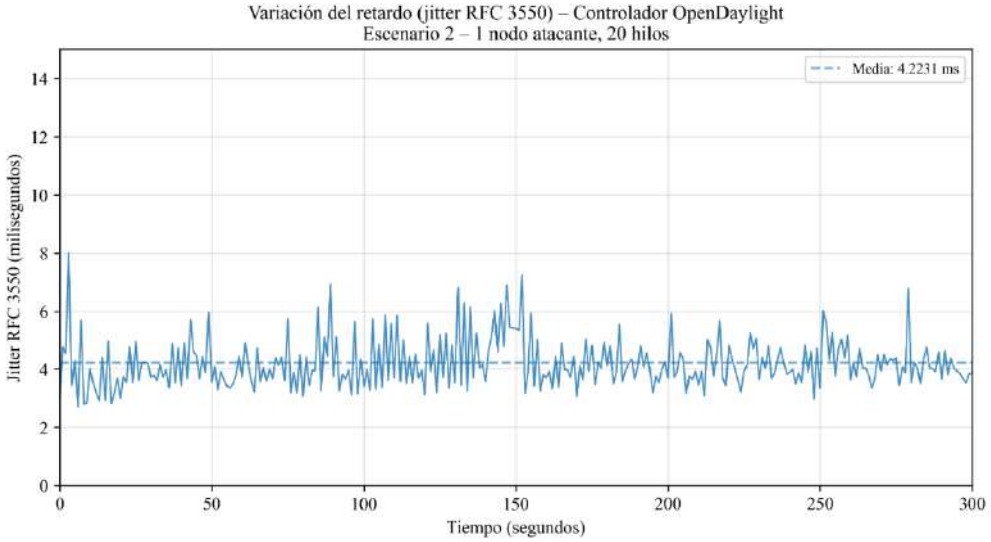
RTT TCP durante el ataque con OpenDaylight (1 worker × 20 hilos)



El jitter con 1 worker y OpenDaylight promedia 4.22 ms (Figura 85), 1.8 veces el valor de la línea base (2.37 ms), aunque dentro del umbral de 50 ms de la ITU-T Y.1541.

Figura 85

Jitter RFC 3550 durante el ataque con OpenDaylight (1 worker × 20 hilos)

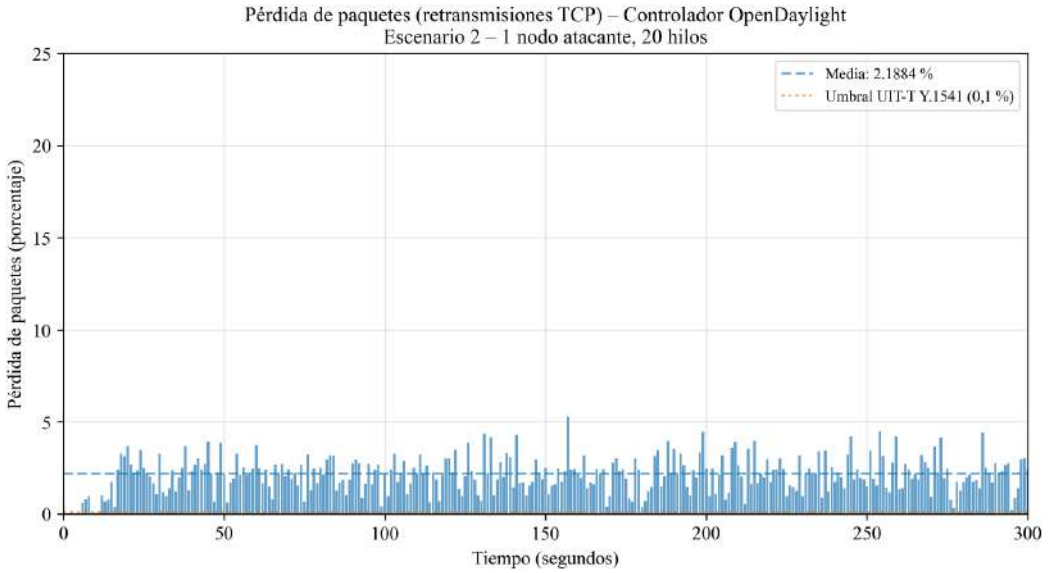


La pérdida de paquetes con 1 worker y OpenDaylight alcanza 2.19 % (Figura 86), frente al 0.058 % de la línea base, y supera tanto el umbral de 0.1 % de la ITU-T Y.1541 como el 1 % que

la ITU-T G.1010 admite para video en streaming. Este incremento marca el inicio efectivo de la degradación del servicio desde el escenario de menor intensidad.

Figura 86

Pérdida de paquetes durante el ataque con OpenDaylight (1 worker × 20 hilos)



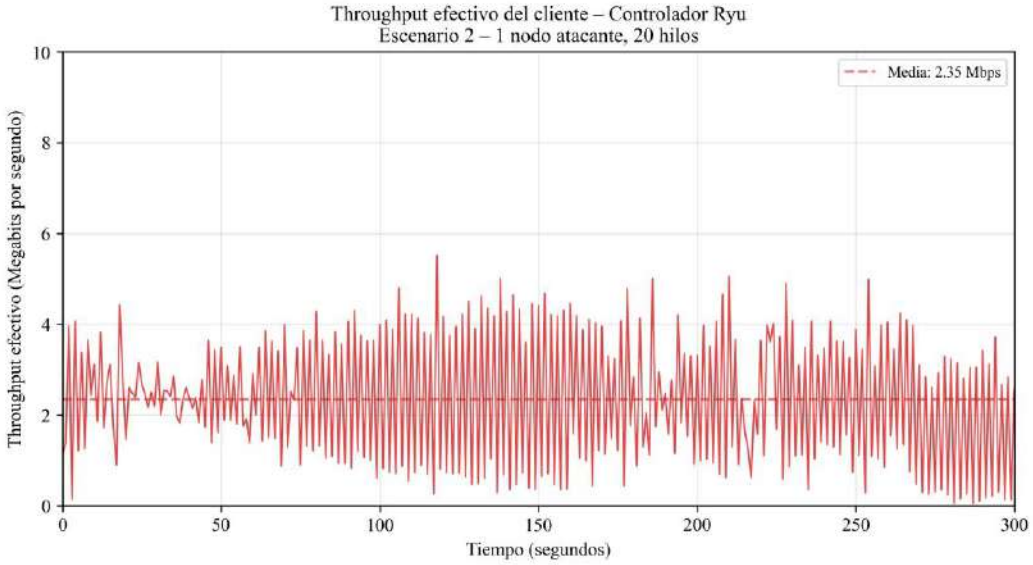
Escenario E2-Ryu (1 worker × 20 hilos)

A continuación, se presentan las cuatro gráficas de QoS (throughput, RTT, jitter y pérdida de paquetes) obtenidas con el controlador Ryu en las mismas condiciones que las de OpenDaylight. En promedio, cada corrida dura 309.7 s y contiene 388 115 paquetes (116.7 MB), de los cuales el 20 % corresponde al tráfico con el cliente legítimo. Cada métrica se contrasta con su valor de línea base con el mismo controlador.

Con Ryu y 1 worker, el throughput efectivo (Figura 87) registra una media de 2.35 Mbps (pico de 5.51 Mbps), un 5.4 % menos que su línea base (2.48 Mbps) y 0.93 veces el de OpenDaylight en idénticas condiciones. Los paquetes intercambiados con el cliente se sitúan en 76 881 por corrida (-0.8 % respecto a su línea base). El procedimiento de extracción se documenta en el Anexo F.

Figura 87

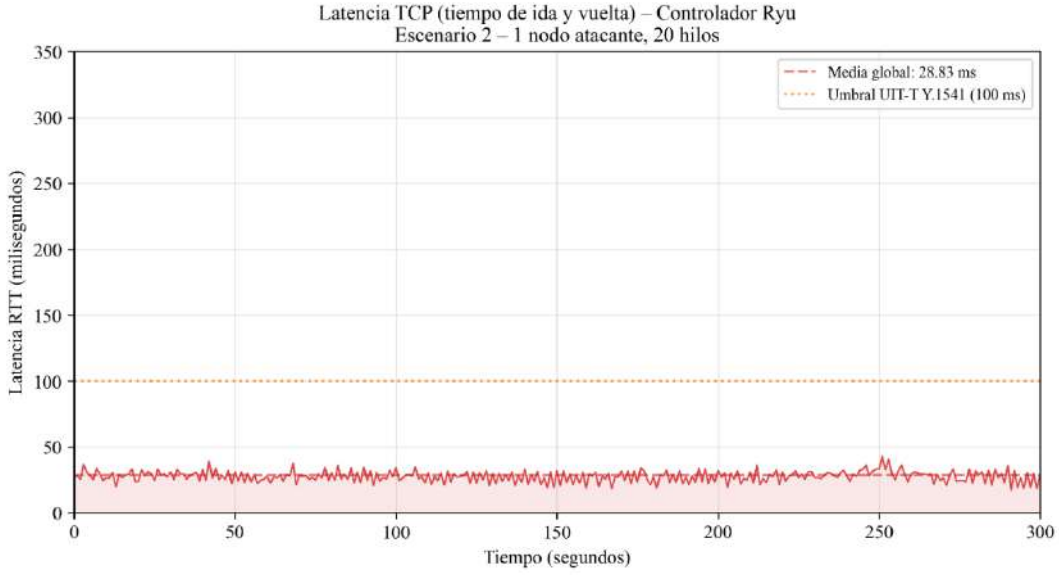
Throughput efectivo durante el ataque tcp_exhaust con Ryu (1 worker × 20 hilos)



El RTT de Ryu con 1 worker promedia 28.83 ms (Figura 88), un 48.9 % por debajo de los 56.45 ms de OpenDaylight en idéntico escenario. Incluso en la intensidad más baja, Ryu conserva la ventaja de latencia observada en la línea base.

Figura 88

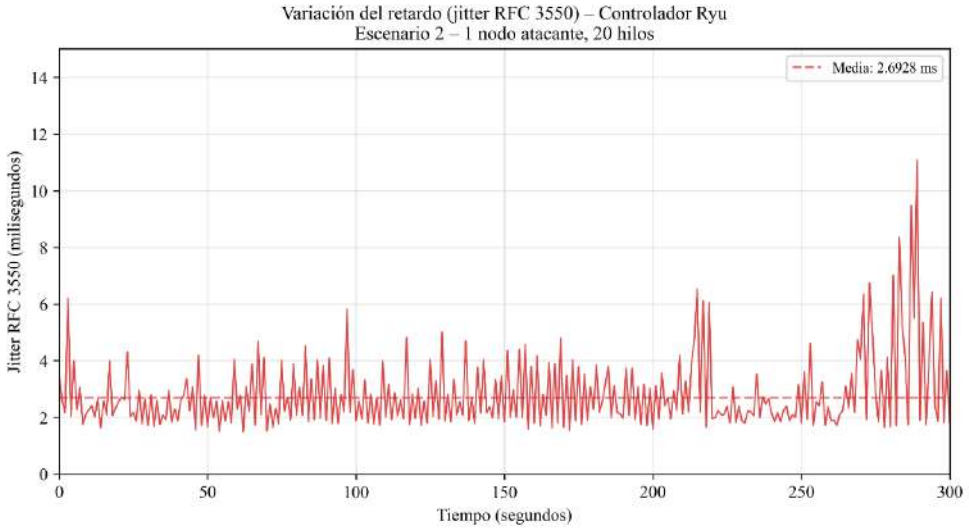
RTT TCP durante el ataque con Ryu (1 worker × 20 hilos)



El jitter de Ryu con 1 worker (Figura 89) se sitúa en 2.69 ms, 1.3 veces su línea base (2.07 ms) y 1.6 veces por debajo del de OpenDaylight (4.22 ms), por lo que el ataque de baja intensidad apenas altera la regularidad temporal del plano de datos gestionado por Ryu.

Figura 89

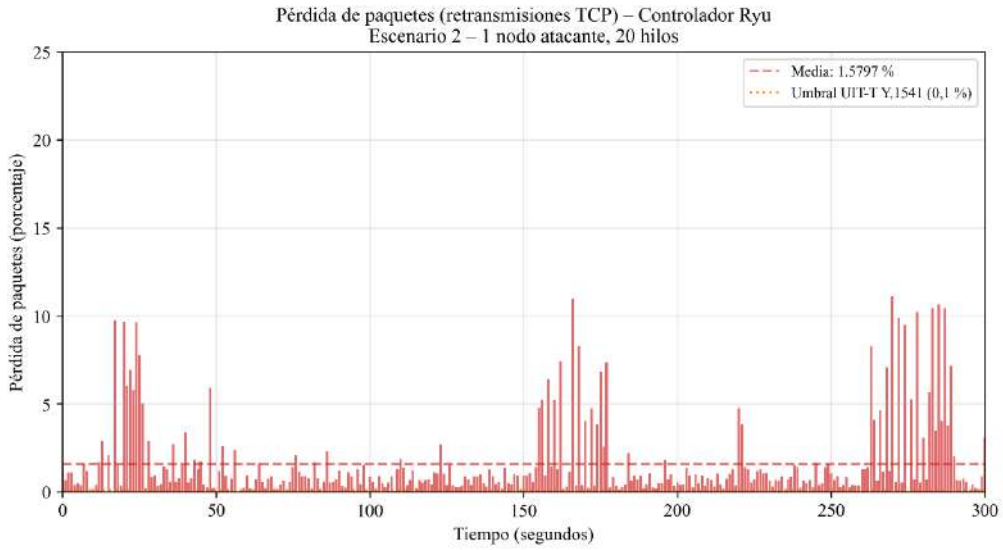
Jitter RFC 3550 durante el ataque con Ryu (1 worker × 20 hilos)



La pérdida de paquetes de Ryu con 1 worker asciende a 1.58 % (Figura 90), frente al 0.023 % de su línea base, y es menor que el 2.19 % de OpenDaylight, aunque con mayor variabilidad entre corridas (desviación estándar de 1.23 puntos), que procede de una sola corrida (3.00 % frente a 0.97 % y 0.77 % en las otras dos) afectada por interrupciones intermitentes del plano de datos, analizadas en el Capítulo IV.

Figura 90

Pérdida de paquetes durante el ataque con Ryu (1 worker × 20 hilos)



ANEXO D. RESULTADOS DETALLADOS DEL ESCENARIO E4 (3 WORKERS × 20 HILOS)

Este anexo reúne el análisis detallado del escenario E4 (3 workers × 20 hilos, alta intensidad), con las mismas métricas de QoS para OpenDaylight y Ryu. La síntesis comparativa se encuentra en la Tabla 32 del Capítulo IV.

Escenario E4: 3 workers × 20 hilos

En el escenario E4, 3 workers concurrentes generan tráfico TCP simultáneo contra el servidor de streaming. En promedio por corrida, con OpenDaylight se registran 434 065 paquetes en 320.3 s (88.4 MB), mientras que con Ryu se registran 667 462 paquetes en 309.0 s (132.6 MB). Las capturas difieren en volumen entre controladores; el origen de esta diferencia se analiza en la discusión.

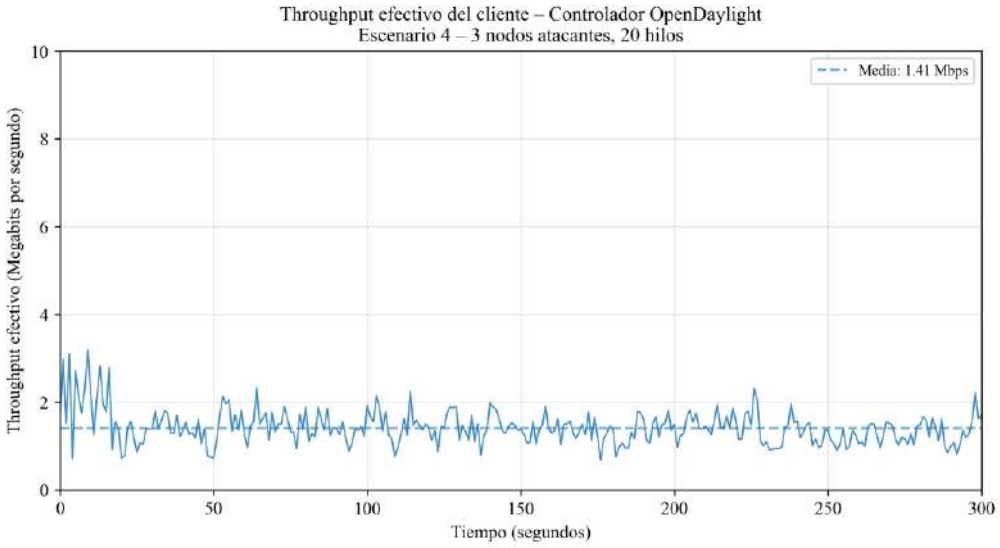
Escenario E4-ODL (3 workers)

Con 3 workers y OpenDaylight, el throughput efectivo (Figura 91) registra una media de 1.41 Mbps (pico de 3.19 Mbps), un 41.7 % menos que la línea base de 2.42 Mbps, mientras que el throughput agregado del enlace, que incluye el tráfico del atacante, alcanza 2.21 Mbps. Los

paquetes intercambiados con el cliente se sitúan en 55 998 por corrida (-28.2 % respecto a la línea base).

Figura 91

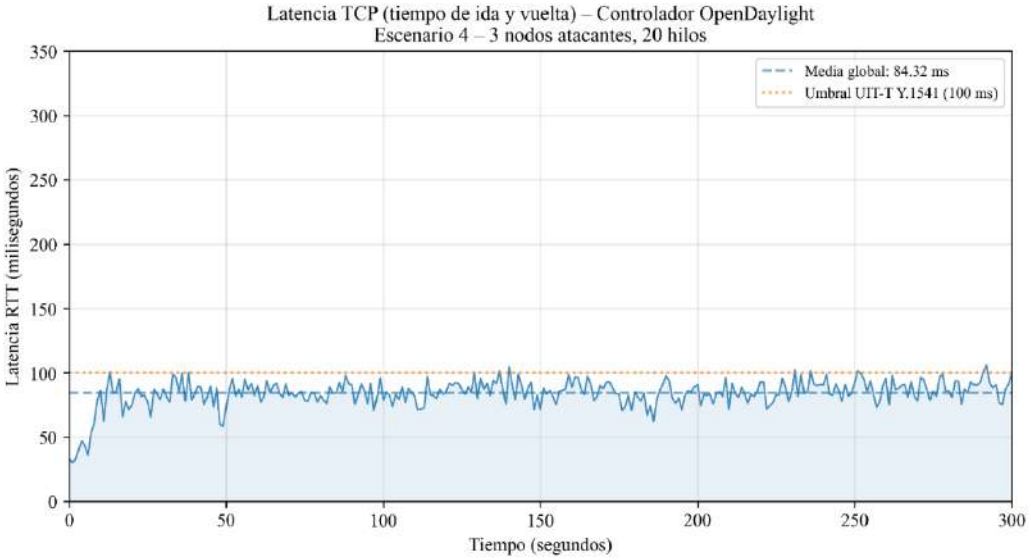
Throughput efectivo con OpenDaylight (3 workers × 20 hilos)



Con 3 workers y OpenDaylight, el RTT promedia 84.32 ms, con un percentil 95 de 157.06 ms (Figura 92), 2.9 veces la línea base (28.90 ms). La media se mantiene por debajo del umbral de 100 ms, pero el percentil 95 lo supera en un 57 %.

Figura 92

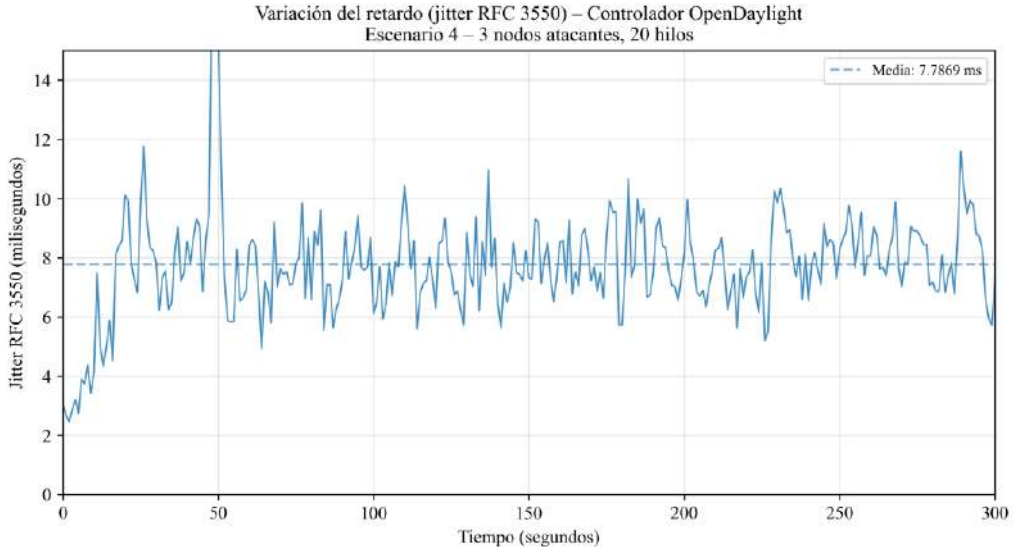
RTT TCP durante el ataque con OpenDaylight (3 workers × 20 hilos)



El jitter con 3 workers y OpenDaylight (Figura 93) promedia 7.79 ms (percentil 95 de 9.94 ms; máximo de 17.19 ms en la serie por segundo). La media sigue por debajo del umbral de 50 ms, pero el crecimiento de los picos respecto a escenarios anteriores revela la presión creciente sobre el plano de datos conforme se intensifica el ataque.

Figura 93

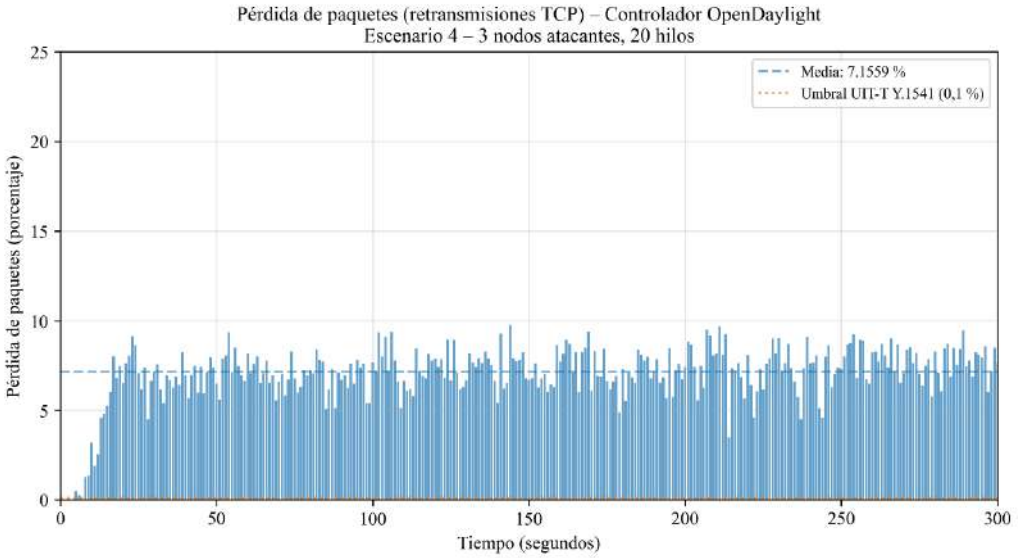
Jitter RFC 3550 durante el ataque con OpenDaylight (3 workers × 20 hilos)



La pérdida de paquetes con 3 workers y OpenDaylight (Figura 94) escala hasta 7.16 %, más de cien veces la línea base (0.058 %) y muy por encima del umbral Y.1541, lo que confirma el agravamiento sostenido del daño con cada worker añadido.

Figura 94

Pérdida de paquetes con OpenDaylight (3 workers × 20 hilos)

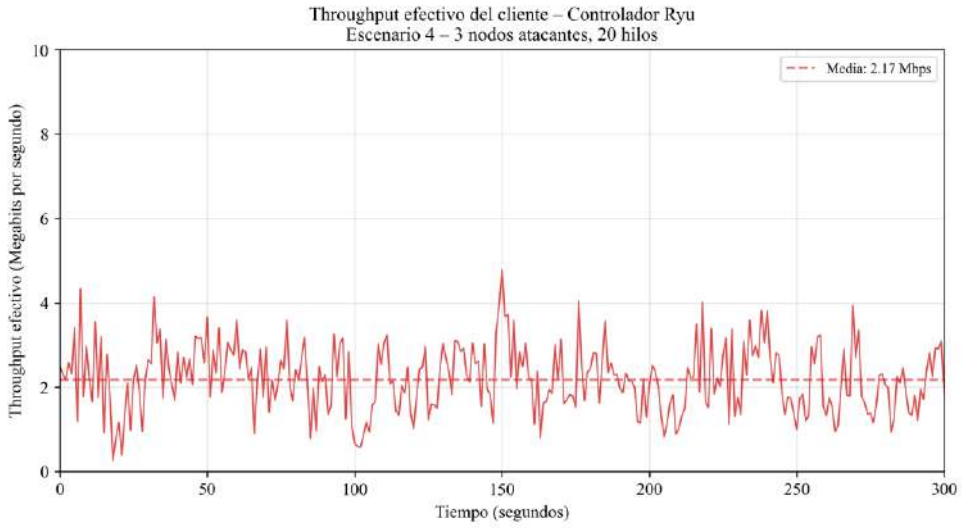


Escenario E4-Ryu (3 workers)

Con Ryu y 3 workers, el throughput efectivo (Figura 95) registra una media de 2.17 Mbps (pico de 4.79 Mbps), un 12.4 % menos que su línea base (2.48 Mbps) y 1.54 veces el de OpenDaylight en idénticas condiciones. Los paquetes intercambiados con el cliente se sitúan en 76 327 por corrida (-1.6 % respecto a su línea base).

Figura 95

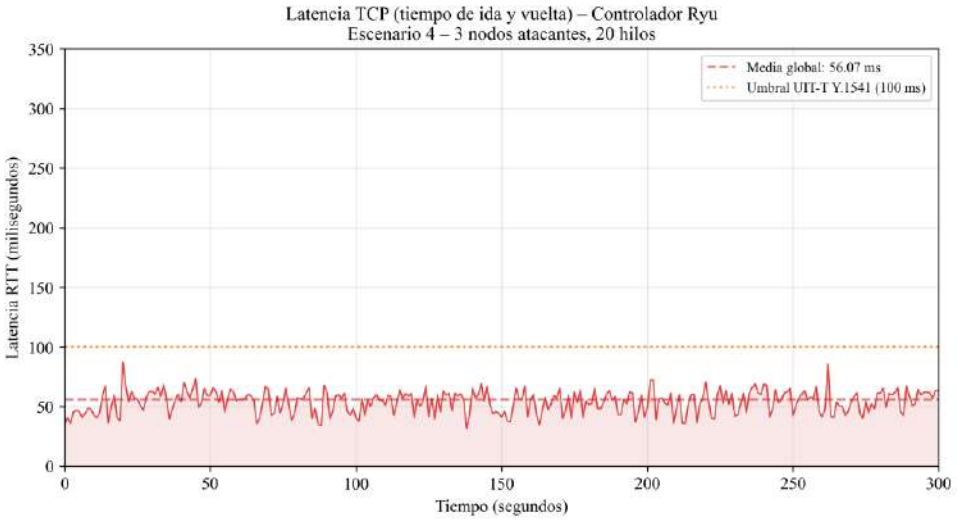
Throughput efectivo con Ryu (3 workers × 20 hilos)



El RTT de Ryu con 3 workers (Figura 96) promedia 56.07 ms (percentil 95 de 101.19 ms). La media se mantiene por debajo del umbral de 100 ms y el percentil 95 apenas lo alcanza, de modo que el ataque también afecta a Ryu a esta intensidad; aun así, la latencia media es 1.5 veces menor que la de OpenDaylight (84.32 ms).

Figura 96

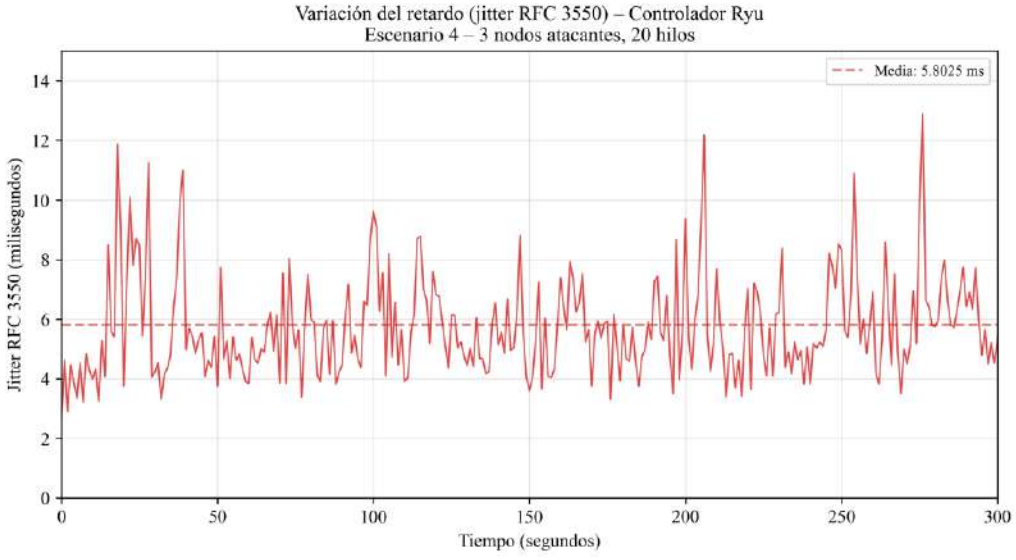
RTT TCP durante el ataque con Ryu (3 workers × 20 hilos)



El jitter de Ryu con 3 workers (Figura 97) se sitúa en 5.80 ms (percentil 95 de 8.78 ms), unas 1.3 veces por debajo del de OpenDaylight (7.79 ms). La regularidad temporal se conserva pese al aumento de intensidad, en línea con el mejor comportamiento de Ryu en el resto de las métricas.

Figura 97

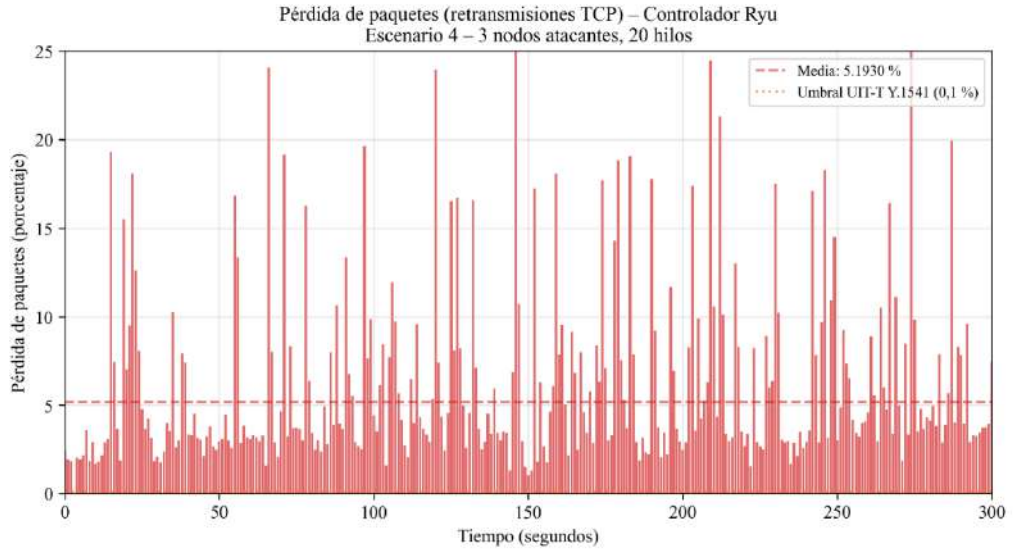
Jitter RFC 3550 durante el ataque con Ryu (3 workers × 20 hilos)



La pérdida de paquetes de Ryu con 3 workers se sitúa en 5.19 % (Figura 98), frente al 7.16 % de OpenDaylight. Aunque el servicio también se ve afectado, Ryu mantiene la tasa en un 73 % de la de OpenDaylight, con una variabilidad elevada entre corridas (desviación estándar de 3.35 puntos). Esa variabilidad procede de una sola corrida (9.04 % frente a 3.61 % y 2.93 % en las otras dos), afectada por interrupciones intermitentes del plano de datos, como se analiza en el Capítulo IV; la mediana de las tres corridas es de 3.61 %.

Figura 98

Pérdida de paquetes con Ryu (3 workers × 20 hilos)

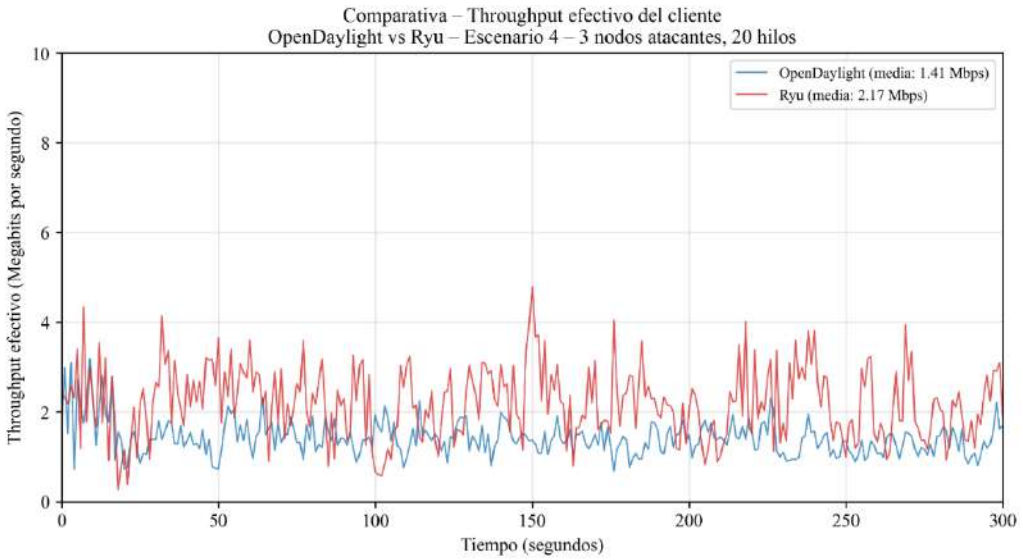


Comparación entre OpenDaylight y Ryu (3 workers)

La superposición de curvas con 3 workers (Figura 99) muestra a Ryu por encima de OpenDaylight en el 79 % de los segundos. La diferencia de medias (2.17 frente a 1.41 Mbps) supone un 54.2 % más de caudal útil entregado al cliente con Ryu.

Figura 99

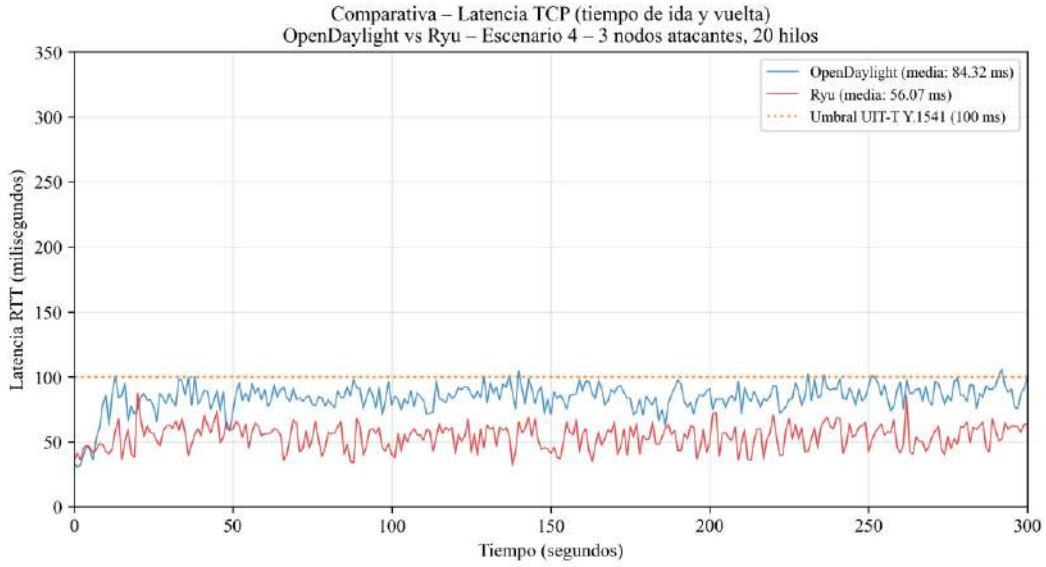
Comparación del throughput efectivo entre OpenDaylight y Ryu (3 workers × 20 hilos)



La Figura 100 compara el RTT con 3 workers; junto con la Tabla 32, muestra que, en las condiciones evaluadas, Ryu presenta un throughput efectivo 1.5 veces superior, un RTT medio 1.5 veces inferior y un jitter medio 1.3 veces menor, además de cursar una tasa de paquetes 1.6 veces mayor.

Figura 100

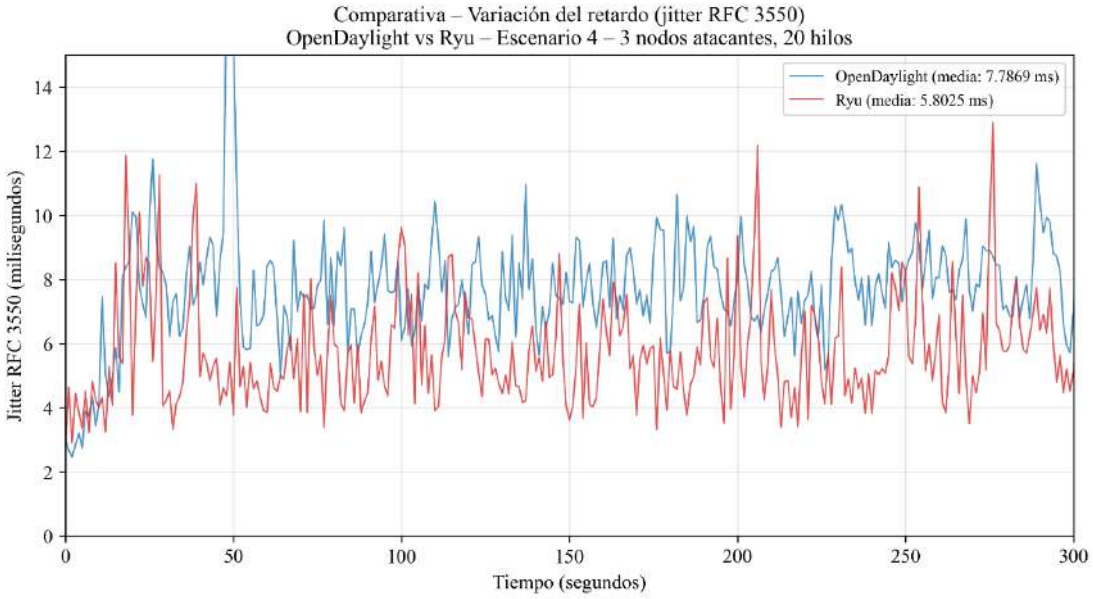
Comparación del RTT entre OpenDaylight y Ryu (3 workers × 20 hilos)



El jitter de ambos controladores con 3 workers (Figura 101) permanece por debajo del umbral de 50 ms, si bien Ryu conserva un margen mayor, de modo que la estabilidad temporal se mantiene como una ventaja sostenida a su favor también a esta intensidad.

Figura 101

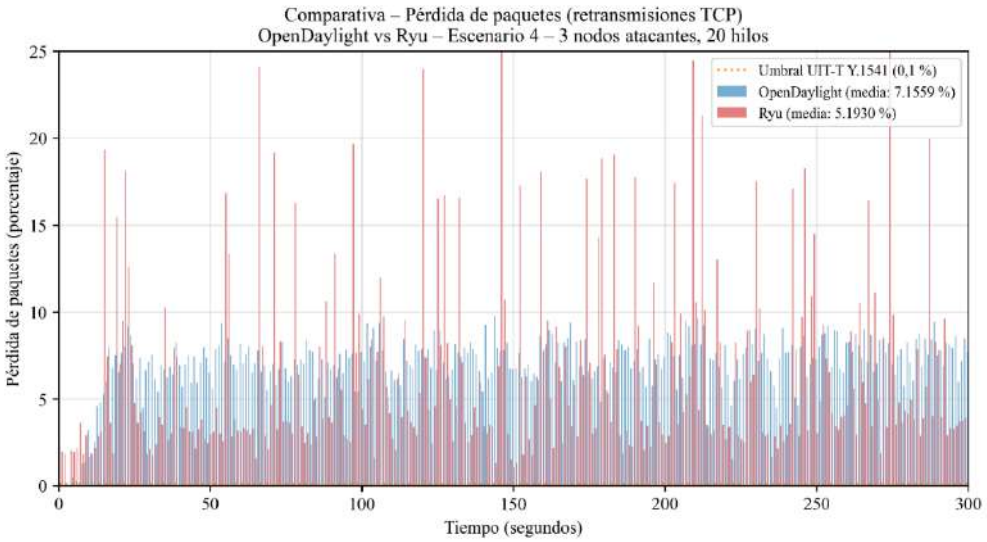
Comparación del jitter entre OpenDaylight y Ryu (3 workers × 20 hilos)



La pérdida de paquetes (Figura 102) es 1.96 puntos porcentuales mayor con OpenDaylight que con Ryu con 3 workers, una diferencia menor que la de E3 (3.10 puntos) debido a la mayor variabilidad de Ryu en este escenario.

Figura 102

Comparación de la pérdida de paquetes entre OpenDaylight y Ryu (3 workers × 20 hilos)



ANEXO E. RESULTADOS DETALLADOS DEL ESCENARIO E5 (4 WORKERS × 20 HILOS)

Este anexo reúne el análisis detallado del escenario E5 (4 workers × 20 hilos, máxima intensidad), con las mismas métricas de QoS para ambos controladores. La síntesis comparativa se encuentra en la Tabla 32 del Capítulo IV.

Escenario E5: 4 workers × 20 hilos

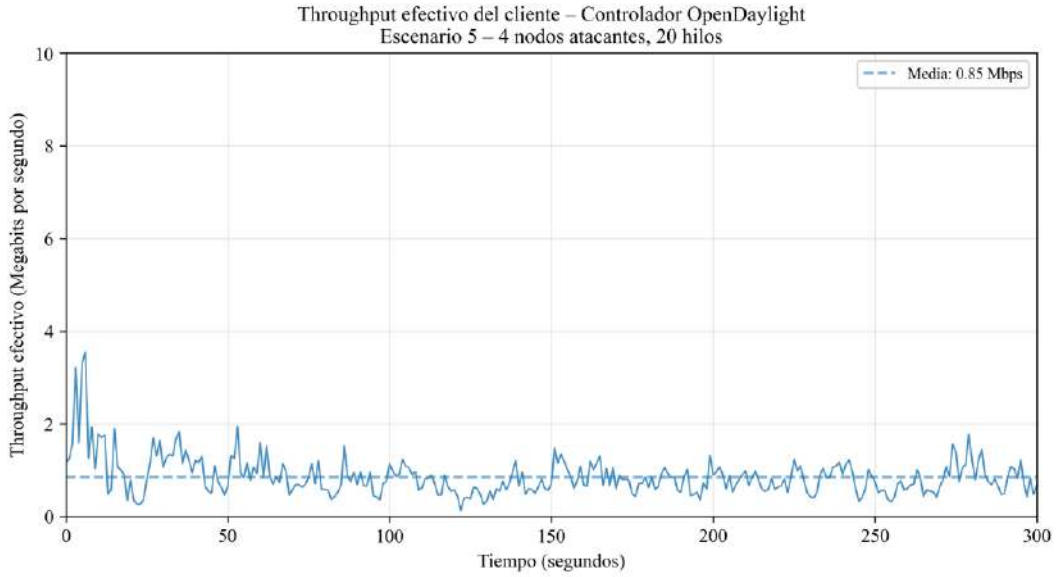
En el escenario E5, 4 workers concurrentes generan tráfico TCP simultáneo contra el servidor de streaming. En promedio por corrida, con OpenDaylight se registran 424 983 paquetes en 322.7 s (67.1 MB), mientras que con Ryu se registran 729 036 paquetes en 308.0 s (120.8 MB). Las capturas difieren en volumen entre controladores; el origen de esta diferencia se analiza en la discusión.

Escenario E5-ODL (4 workers)

Con 4 workers y OpenDaylight, el throughput efectivo (Figura 103) registra una media de 0.85 Mbps (pico de 3.55 Mbps), un 64.8 % menos que la línea base de 2.42 Mbps, mientras que el throughput agregado del enlace, que incluye el tráfico del atacante, alcanza 1.66 Mbps. Los paquetes intercambiados con el cliente se sitúan en 35 325 por corrida (-54.7 % respecto a la línea base).

Figura 103

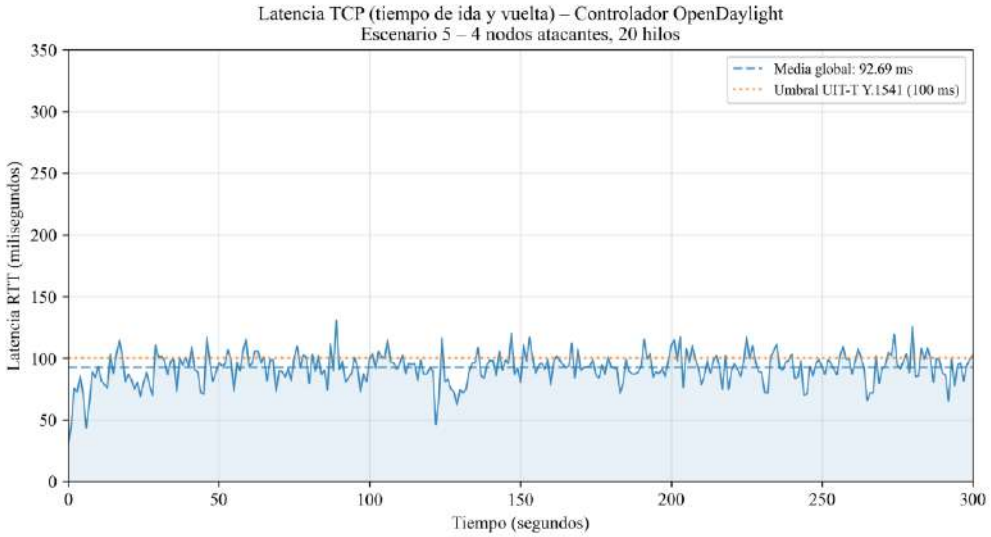
Throughput efectivo con OpenDaylight (4 workers × 20 hilos)



El RTT con 4 workers y OpenDaylight promedia 92.69 ms, con un percentil 95 de 181.52 ms (Figura 104), 3.2 veces la línea base (28.90 ms). La media se aproxima al umbral de 100 ms y el percentil 95 lo supera en un 82 %, lo que confirma una congestión severa.

Figura 104

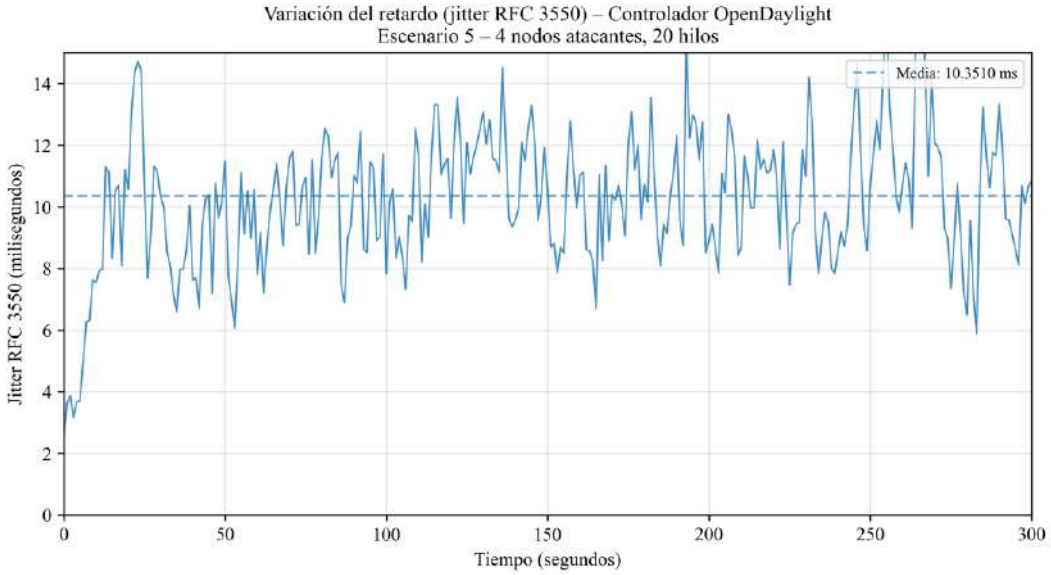
RTT TCP durante el ataque con OpenDaylight (4 workers × 20 hilos)



El jitter con 4 workers y OpenDaylight (Figura 105) promedia 10.35 ms (percentil 95 de 13.58 ms; máximo de 22.39 ms en la serie por segundo), 4.4 veces la línea base (2.37 ms). La media y los picos permanecen por debajo del umbral de 50 ms, aunque la tendencia creciente confirma la presión sobre el plano de datos en la intensidad máxima.

Figura 105

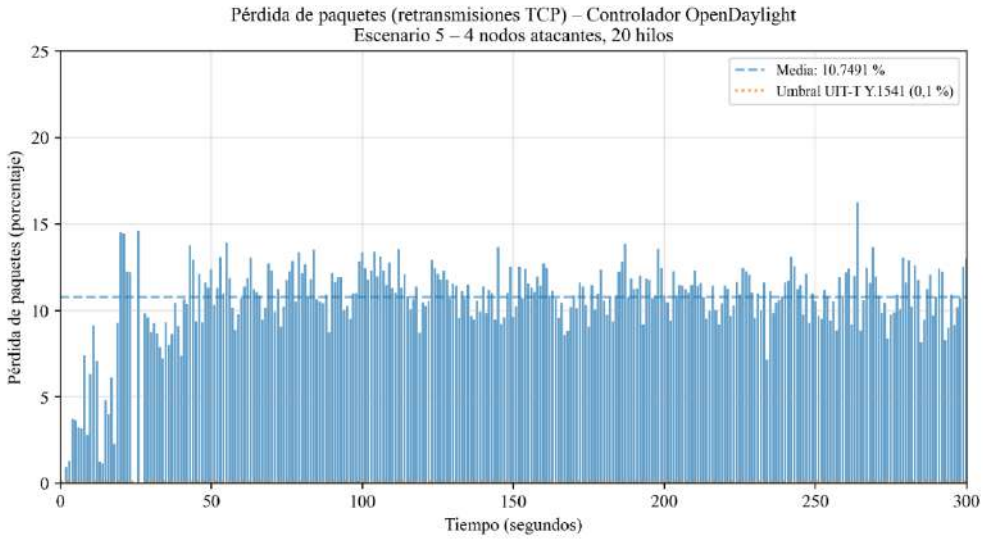
Jitter RFC 3550 durante el ataque con OpenDaylight (4 workers × 20 hilos)



La pérdida de paquetes con 4 workers y OpenDaylight alcanza 10.75 % (Figura 106), el valor máximo del experimento y unas 185 veces la línea base (0.058 %), lo que fija el peor caso del estudio con OpenDaylight.

Figura 106

Pérdida de paquetes con OpenDaylight (4 workers × 20 hilos)

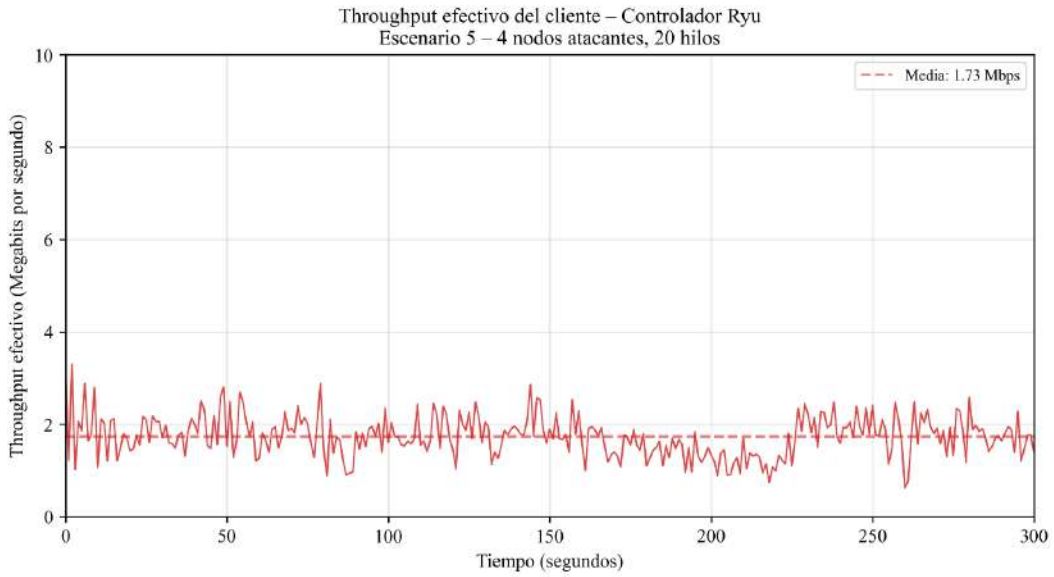


Escenario E5-Ryu (4 workers)

Con Ryu y 4 workers, el throughput efectivo (Figura 107) registra una media de 1.73 Mbps (pico de 3.31 Mbps), un 30.2 % menos que su línea base (2.48 Mbps) y 2.04 veces el de OpenDaylight en idénticas condiciones. Los paquetes intercambiados con el cliente se sitúan en 63 885 por corrida (-17.6 % respecto a su línea base).

Figura 107

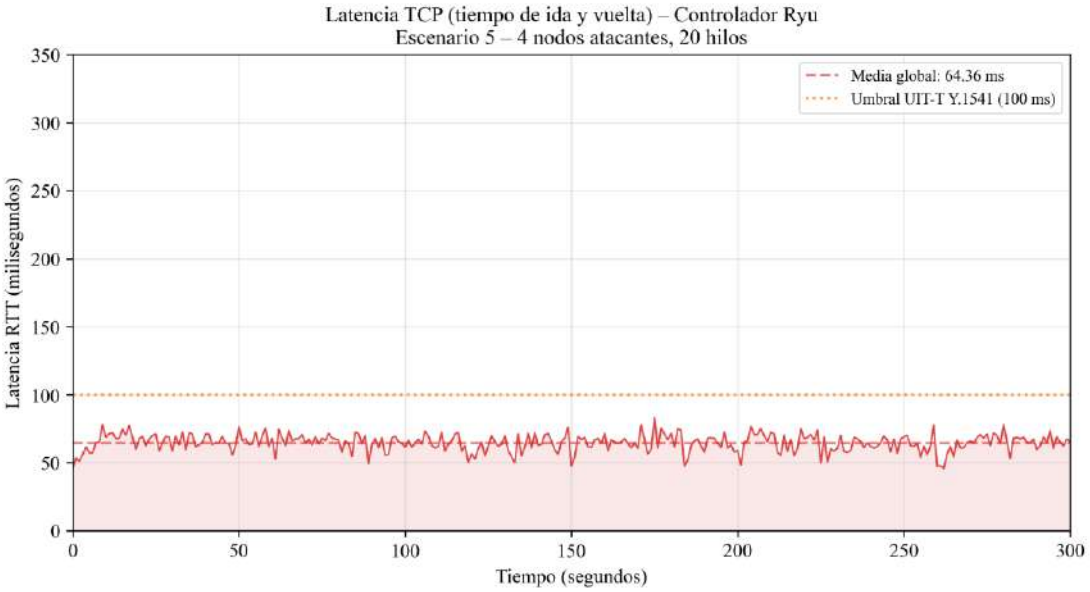
Throughput efectivo con Ryu (4 workers × 20 hilos)



El RTT de Ryu con 4 workers (Figura 108) promedia 64.36 ms (percentil 95 de 111.62 ms). La media se mantiene por debajo del umbral de 100 ms, aunque el percentil 95 lo supera ligeramente; la latencia media es 1.4 veces menor que la de OpenDaylight (92.69 ms), de modo que la diferencia entre controladores persiste incluso en el peor escenario.

Figura 108

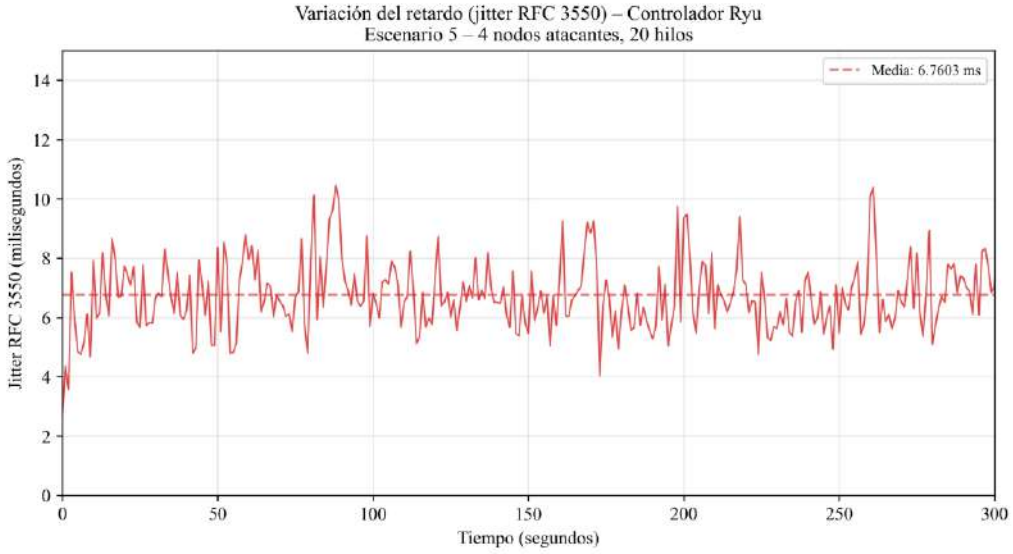
RTT TCP durante el ataque con Ryu (4 workers × 20 hilos)



El jitter de Ryu con 4 workers se sitúa en 6.76 ms, con un percentil 95 de 8.88 ms (Figura 109), unas 1.5 veces por debajo del de OpenDaylight (10.35 ms). La estabilidad temporal de Ryu se mantiene en la intensidad máxima, con picos menores que los de OpenDaylight.

Figura 109

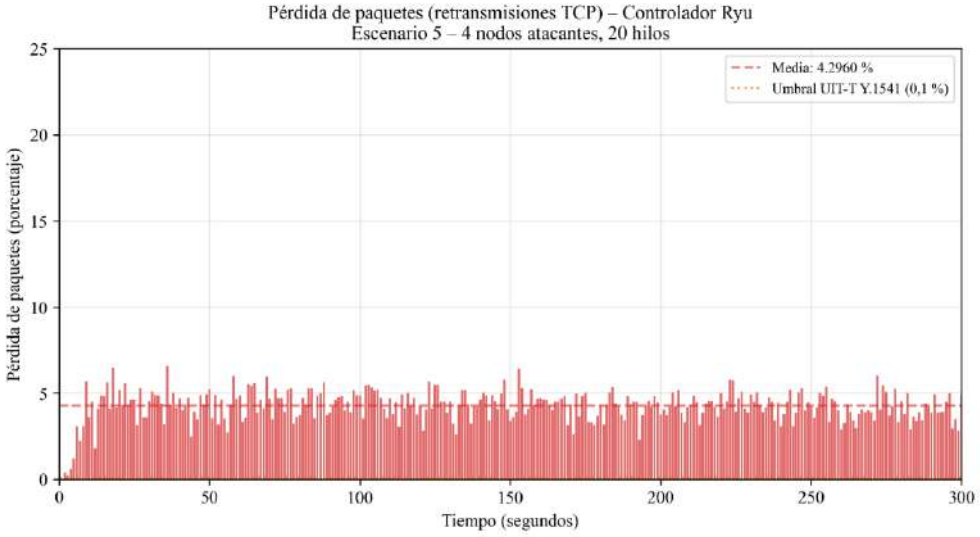
Jitter RFC 3550 durante el ataque con Ryu (4 workers × 20 hilos)



La pérdida de paquetes de Ryu con 4 workers se sitúa en 4.30 % de media (Figura 110), por debajo incluso de la registrada en E4 (5.19 %), y equivale a menos de la mitad del 10.75 % de OpenDaylight en el mismo escenario, lo que mantiene una diferencia de calidad clara a favor de Ryu.

Figura 110

Pérdida de paquetes con Ryu (4 workers × 20 hilos)

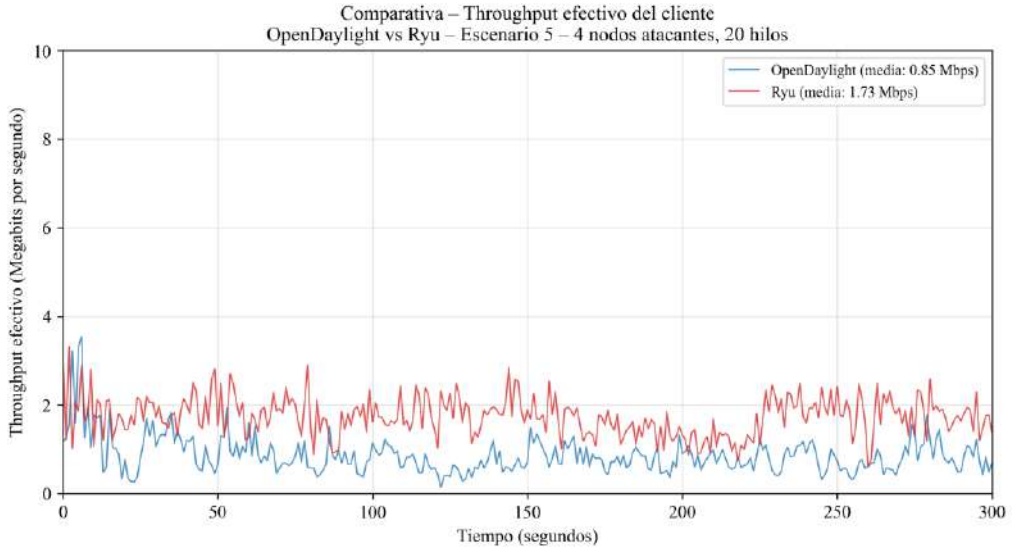


Comparación entre OpenDaylight y Ryu (4 workers)

La superposición de curvas con 4 workers (Figura 111) muestra a Ryu por encima de OpenDaylight en el 94 % de los segundos. La diferencia de medias (1.73 frente a 0.85 Mbps) supone un 103.5 % más de caudal útil entregado al cliente con Ryu.

Figura 111

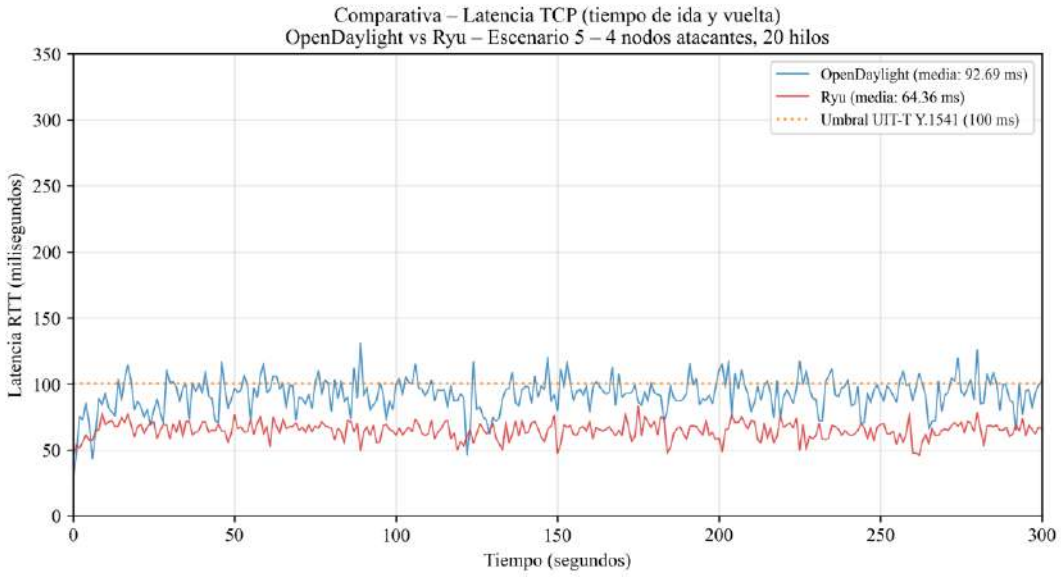
Comparación del throughput efectivo entre OpenDaylight y Ryu (4 workers × 20 hilos)



La Figura 112 compara el RTT con 4 workers; junto con la Tabla 32, muestra que Ryu presenta un throughput efectivo 2.0 veces superior, un RTT medio 1.4 veces inferior y un jitter medio 1.5 veces menor, además de cursar una tasa de paquetes 1.8 veces mayor.

Figura 112

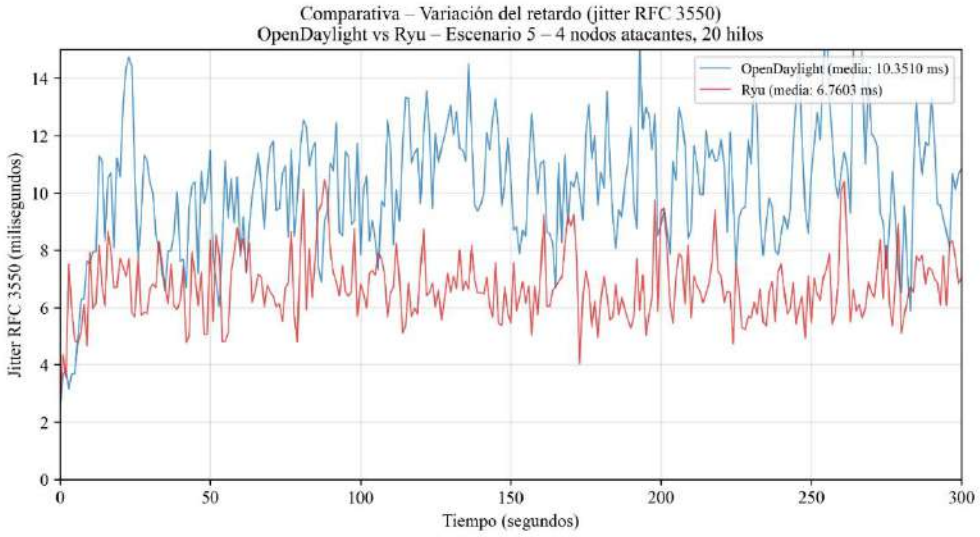
Comparación del RTT entre OpenDaylight y Ryu (4 workers × 20 hilos)



El jitter de ambos controladores con 4 workers (Figura 113) permanece por debajo del umbral de 50 ms, aunque Ryu conserva la media más baja, con la misma tendencia observada en el throughput, el RTT y la pérdida.

Figura 113

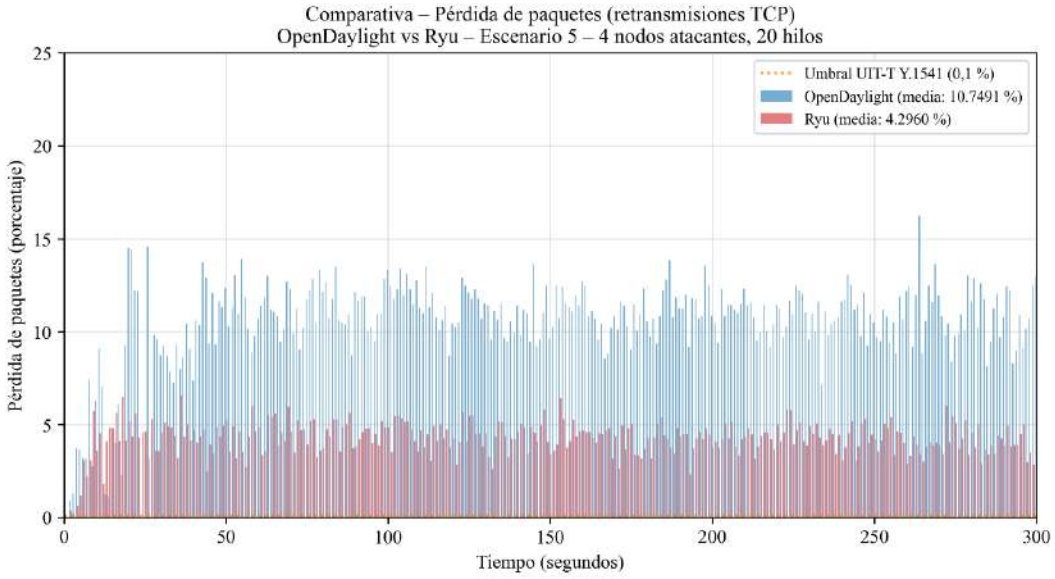
Comparación del jitter entre OpenDaylight y Ryu (4 workers × 20 hilos)



La pérdida de paquetes en la máxima intensidad (Figura 114) confirma una degradación de OpenDaylight muy superior a la de Ryu (10.75 % frente a 4.30 %). Esta comparación cierra el anexo evidenciando que, en los escenarios de intensidad media y alta y frente al ataque tcp_exhaust, el efecto sobre la calidad del servicio es sistemáticamente más severo con OpenDaylight que con Ryu.

Figura 114

Comparación de la pérdida de paquetes entre OpenDaylight y Ryu (4 workers × 20 hilos)



ANEXO F. SCRIPTS DE EXTRACCIÓN DE MÉTRICAS DE QOS

Este anexo presenta el script unificado desarrollado en Python para la extracción automatizada de métricas de calidad de servicio (QoS) a partir de archivos de captura pcapng y para la generación de las gráficas estandarizadas del Capítulo IV. El script integra las siguientes funciones:

• Extracción del throughput efectivo del cliente (bytes que el servidor entrega al cliente legítimo por los puertos 8554 y 8888) y del throughput agregado del punto de captura, mediante io,stat de tshark.

• Extracción de latencia TCP (RTT) con filtrado por mediana ± 3 MAD (desviación absoluta mediana) para eliminar valores atípicos.

• Cálculo de jitter según RFC 3550: J(i) = J(i-1) + (|D(i-1,i)| - J(i-1))/16, filtrado a tráfico del cliente de streaming (puertos 8554/8888).

• Pérdida de paquetes: tcp.analysis.lost_segment para línea base, tcp.analysis.retransmission con filtrado P90 para escenarios de ataque.

• Promedio de las tres corridas de cada escenario (series por segundo sobre la duración común y valores globales como media ± desviación estándar) y registro del tamaño de muestra de cada métrica.

• Generación de las gráficas del Capítulo IV y de los Anexos C, D y E con escalas comunes por métrica: throughput efectivo 0–10 Mbps, RTT 0–350 ms, jitter 0–15 ms y pérdida 0–25 %, con resolución de 300 DPI.

Dependencias: Python 3.11+, numpy y matplotlib. Herramienta externa: tshark 4.x (Wireshark CLI). Uso: python extraer_metricas_qos.py [--listar | --solo-extraccion | --solo-graficas | --escenario <ID> | --forzar | --jobs <N>]

Script: extraer_metricas_qos.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
  EXTRACCIÓN DE MÉTRICAS QoS Y GENERACIÓN DE GRÁFICAS (CON RÉPLICAS)
  Tesis: Impacto de ataques DDoS sobre servicio de streaming en redes SDN
  Autor: Diego Achina
  Herramientas: tshark (Wireshark CLI), Python 3.11, numpy, matplotlib
================================================================================

Estructura esperada de las capturas (una carpeta run_XXX por réplica):
  <CAPTURAS_BASE>/ODL/run_001/E1_baseline-ODL.pcapng ... E5_tcp_exhaust_4w-ODL.pcapng
  <CAPTURAS_BASE>/ODL/run_002/...
  <CAPTURAS_BASE>/RYU/run_001/E1_baseline-RYU.pcapng ...

Flujo del script:
  1. Las réplicas se descubren automáticamente en las carpetas run_*. Si dos
     réplicas de un escenario son el mismo archivo (copias idénticas), se
     cuentan una sola vez y se emite un aviso.
  2. De cada réplica se extraen throughput efectivo del cliente (y agregado), PPS, RTT,
jitter (RFC 3550) y
     pérdida de paquetes. El resultado se guarda en metricas_csv/replicas/
     y se reutiliza en ejecuciones posteriores (use --forzar para recalcular).
  3. Las réplicas de un escenario se combinan:
       - Series por segundo: media y desviación estándar entre réplicas,
         segundo a segundo, sobre la duración común.
       - Valores globales: media ± desviación estándar de los valores
         globales de cada réplica.
  4. Si ninguna réplica de un escenario contiene tráfico suficiente del
     cliente de streaming (caso ODL E3), el jitter y la pérdida se
     interpolan linealmente entre los escenarios adyacentes.
  5. Se generan las mismas figuras de la tesis (Fig. 42–112) a partir de los
     promedios por segundo de las réplicas (barras con desviación estándar).

Uso:
  python extraer_metricas_qos.py                    # Extracción + gráficas
```

```
          python extraer_metricas_qos.py --listar           # Estado de las réplicas configuradas
          python extraer_metricas_qos.py --solo-extraccion  # Solo extracción y promedios
          python extraer_metricas_qos.py --solo-graficas    # Solo gráficas (requiere extracción
previa)
          python extraer_metricas_qos.py --escenario ODL_E2 # Solo los escenarios que contengan
ese texto
          python extraer_metricas_qos.py --forzar           # Ignora extracciones previas en caché
          python extraer_metricas_qos.py --jobs 3           # Procesa 3 capturas en paralelo

        Requisitos:
          - tshark 4.x (Wireshark CLI) en la ruta configurada en TSHARK
          - Python 3.11+ con numpy y matplotlib
        """

        import argparse
        import csv
        import hashlib
        import json
        import math
        import os
        import re
        import subprocess
        import sys
        from collections import defaultdict
        from concurrent.futures import ThreadPoolExecutor

        try:
            import numpy as np
        except ImportError:
            print("ERROR: numpy no instalado. Ejecute: pip install numpy")
            sys.exit(1)

        if hasattr(sys.stdout, "reconfigure"):
            sys.stdout.reconfigure(errors="replace")

        # ============================================================================
        # CONFIGURACIÓN — Adaptar estas rutas a su entorno
        # ============================================================================
        TSHARK = r"C:\Program Files\Wireshark\tshark.exe"
        CLIENT_IP = "192.168.10.35"
        SERVER_IP = "192.168.10.66"
        PUERTOS_STREAMING = (8554, 8888)     # RTSP y HLS del servidor MediaMTX
        UMBRAL_PAUSA_S = 0.050               # Intervalos > 50 ms se consideran pausas HLS
        MIN_PAQUETES_CLIENTE = 1000          # Mínimo de paquetes del cliente para una réplica
válida
        UMBRAL_MAD = 3.0                     # Filtro de atípicos del RTT: mediana ± 3 MAD (Leys
et al., 2013)
        REPLICAS_ESPERADAS = 3
        VERSION_EXTRACCION = 5               # Cambiar invalida la caché de réplicas

        # Throughput efectivo: tráfico del servicio entregado por el servidor al cliente legítimo
        FILTRO_EFECTIVO = (f"ip.src=={SERVER_IP} && ip.dst=={CLIENT_IP} && "
                           f"(tcp.srcport=={PUERTOS_STREAMING[0]} ||
tcp.srcport=={PUERTOS_STREAMING[1]})")

        CAPTURAS_BASE = r"D:\Capturas tesis\Capturas ryu workeratack 1"
        CONTROLADORES = ("ODL", "RYU")
        ESCENARIOS = ["E1_linea_base", "E2_1worker", "E3_2workers", "E4_3workers", "E5_4workers"]

        # Nombre de archivo: E<n>_<descripción>-<CONTROLADOR>.pcapng
        PATRON_CAPTURA = re.compile(r"^(E[1-5])_.*-(ODL|RYU)\.pcapng?$", re.IGNORECASE)



        def descubrir_capturas():
            """Retorna {escenario: [rutas]} a partir de <CAPTURAS_BASE>/<CTRL>/run_*/."""
            por_codigo = {e.split("_")[0]: e for e in ESCENARIOS}
```

```
            capturas = {f"{c}_{e}": [] for c in CONTROLADORES for e in ESCENARIOS}
            for ctrl in CONTROLADORES:
                carpeta = os.path.join(CAPTURAS_BASE, ctrl)
                if not os.path.isdir(carpeta):
                    continue
                for run in sorted(os.listdir(carpeta)):
                    dir_run = os.path.join(carpeta, run)
                    if not (run.lower().startswith("run_") and os.path.isdir(dir_run)):
                        continue
                    for nombre in sorted(os.listdir(dir_run)):
                        m = PATRON_CAPTURA.match(nombre)
                        if m and m.group(2).upper() == ctrl:
                            label = f"{ctrl}_{por_codigo[m.group(1).upper()]}"
                            capturas[label].append(os.path.join(dir_run, nombre))
            return capturas



        def huella(ruta, bloque=8 * 1024 * 1024):
            """Identificador del contenido (tamaño + MD5 del inicio y final) para detectar
copias."""
            tam = os.path.getsize(ruta)
            h = hashlib.md5()
            with open(ruta, "rb") as f:
                h.update(f.read(bloque))
                if tam > bloque:
                    f.seek(max(bloque, tam - bloque))
                    h.update(f.read(bloque))
            return f"{tam}-{h.hexdigest()}"



        def id_replica(ruta):
            return os.path.basename(os.path.dirname(ruta))



        def replicas_unicas(label, rutas, avisar=True):
            """Descarta réplicas cuyo contenido es idéntico a otra ya incluida."""
            vistas, unicas = {}, []
            for ruta in rutas:
                k = huella(ruta)
                if k in vistas:
                    if avisar:
                        print(f"  AVISO: {label}: {id_replica(ruta)} es una copia idéntica de "
                              f"{id_replica(vistas[k])}; se cuenta una sola vez")
                    continue
                vistas[k] = ruta
                unicas.append(ruta)
            return unicas



        CAPTURAS = descubrir_capturas()

        BASE_DIR = os.path.dirname(os.path.abspath(__file__))
        METRICS_DIR = os.path.join(BASE_DIR, "metricas_csv")
        REPLICAS_DIR = os.path.join(METRICS_DIR, "replicas")
        GRAPHS_DIR = os.path.join(BASE_DIR, "graficas_qos")
        RESUMEN_PATH = os.path.join(METRICS_DIR, "resumen_global.json")

        METRICAS_SERIE = ("throughput", "throughput_agregado", "pps", "rtt", "jitter", "loss")
        METRICAS_RESUMEN = ("throughput_medio_mbps", "throughput_agregado_mbps", "pps_medio",
"rtt_medio_ms",
                            "rtt_mediana_ms", "rtt_p95_ms", "jitter_medio_ms",
"perdida_paquetes_pct")
        # Tamaños de muestra que se suman entre réplicas
        CONTEOS = ("segundos", "paquetes_efectivos", "muestras_rtt_total", "muestras_rtt_validas",
                   "muestras_jitter", "paquetes_tcp_perdida")
        # Métricas que dependen del tráfico del cliente: solo se promedian réplicas válidas
```

```
        METRICAS_CLIENTE = {"jitter", "loss", "jitter_medio_ms", "perdida_paquetes_pct"}



        # ======================================================================
        # SECCIÓN 1: EXTRACCIÓN DE MÉTRICAS DE UNA CAPTURA (tshark)
        # ======================================================================

        def run_tshark(args, timeout=1800):
            """Ejecuta tshark y retorna su salida estándar."""
            r = subprocess.run([TSHARK] + args, capture_output=True, text=True,
                               encoding="utf-8", errors="replace", timeout=timeout)
            if r.returncode != 0 and not r.stdout.strip():
                raise RuntimeError(f"tshark falló: {r.stderr.strip()[:300]}")
            return r.stdout



        _FILA_IO_STAT = re.compile(r"^\|\s*(\d+(?:\.\d+)?)\s*<>\s*(\d+(?:\.\d+)?)\s*\|(.*)$")



        def _filas_io_stat(stdout):
            """
            Interpreta la tabla de 'tshark -z io,stat'. Retorna [(segundo, [columnas])].
            El último intervalo parcial ('<> Dur') se descarta.
            """
            filas = []
            for line in stdout.splitlines():
                m = _FILA_IO_STAT.match(line.strip())
                if not m:
                    continue
                cols = []
                for c in m.group(3).split("|"):
                    try:
                        cols.append(float(c.strip()))
                    except ValueError:
                        continue
                filas.append((int(float(m.group(1))), cols))
            return filas



        def extraer_throughput(pcap):
            """
            Throughput por segundo en intervalos de 1 s:
              - efectivo: bytes que el servidor de streaming entrega al cliente legítimo
                (FILTRO_EFECTIVO); es el throughput que se reporta como métrica de QoS.
              - agregado: todos los bytes del punto de captura (incluye el tráfico de ataque).
            Comando: tshark -r <pcap> -q -z io,stat,1,frame,"<FILTRO_EFECTIVO>"
            Fórmula: Throughput (Mbps) = (bytes × 8) / 1 000 000
            Retorna [(segundo, paquetes_totales, mbps_agregado, mbps_efectivo,
paquetes_efectivos)].
            """
            serie = []
            for seg, cols in _filas_io_stat(run_tshark(["-r", pcap, "-q", "-z",
f"io,stat,1,frame,{FILTRO_EFECTIVO}"])):
                if len(cols) >= 4:
                    serie.append((seg, cols[0], cols[1] * 8 / 1_000_000, cols[3] * 8 / 1_000_000,
cols[2]))
            return serie



        def extraer_rtt(pcap):
            """
            RTT TCP (ms) del tráfico del cliente, con filtrado robusto de valores atípicos
            basado en la mediana y la desviación absoluta mediana (MAD):
                se descartan las muestras con |RTT - mediana| > UMBRAL_MAD × 1,4826 × MAD
            Se usa MAD en lugar de IQR porque las ráfagas de descarga HLS generan ACK
```

```
            retrasados varios segundos que pueden superar el 25 % de las muestras, límite
            a partir del cual el IQR deja de descartar valores atípicos (el MAD tolera hasta 50
%).
            Comando: tshark -r <pcap> -T fields -e frame.time_relative -e tcp.analysis.ack_rtt
                     -Y "tcp.analysis.ack_rtt and ip.addr==<CLIENT_IP>"
            Retorna ([(tiempo_s, rtt_ms)] tras el filtro, número de muestras antes del filtro).
            """
            out = run_tshark(["-r", pcap, "-T", "fields",
                              "-e", "frame.time_relative", "-e", "tcp.analysis.ack_rtt",
                              "-Y", f"tcp.analysis.ack_rtt and ip.addr=={CLIENT_IP}"])
            muestras = []
            for line in out.splitlines():
                p = line.split("\t")
                if len(p) < 2:
                    continue
                try:
                    muestras.append((float(p[0]), float(p[1]) * 1000))
                except ValueError:
                    continue
            if not muestras:
                return [], 0
            vals = np.array([r for _, r in muestras])
            mediana = np.median(vals)
            mad = 1.4826 * np.median(np.abs(vals - mediana))
            if mad == 0:
                return muestras, len(muestras)
            return [m for m in muestras if abs(m[1] - mediana) <= UMBRAL_MAD * mad], len(muestras)



        def extraer_jitter(pcap):
            """
            Jitter (ms) según RFC 3550 sobre el tráfico del cliente de streaming:
                J(i) = J(i-1) + (|D(i-1,i)| - J(i-1)) / 16
            Filtro: tcp and ip.addr==<CLIENT_IP> and (tcp.port == 8554 or tcp.port == 8888)
            Los intervalos mayores a UMBRAL_PAUSA_S (pausas entre segmentos HLS) reinician
            la referencia. Retorna ([(segundo, jitter_medio_ms)], paquetes_del_cliente,
muestras_de_jitter).
            """
            puertos = " or ".join(f"tcp.port == {p}" for p in PUERTOS_STREAMING)
            filtro = f"tcp and ip.addr=={CLIENT_IP} and ({puertos})"
            out = run_tshark(["-r", pcap, "-T", "fields",
                              "-e", "frame.time_relative", "-e", "frame.time_delta_displayed",
                              "-Y", filtro])
            jitter, prev, n_paquetes = 0.0, None, 0
            por_segundo = defaultdict(list)
            for line in out.splitlines():
                p = line.split("\t")
                if len(p) < 2:
                    continue
                try:
                    t, delta = float(p[0]), float(p[1])
                except ValueError:
                    continue
                n_paquetes += 1
                if delta > UMBRAL_PAUSA_S:
                    prev = None
                    continue
                if prev is not None:
                    jitter += (abs(delta - prev) - jitter) / 16.0
                    por_segundo[int(t)].append(jitter * 1000)
                prev = delta
            serie = [(s, float(np.mean(v))) for s, v in sorted(por_segundo.items())]
            return serie, n_paquetes, sum(len(v) for v in por_segundo.values())



        def extraer_perdida(pcap, linea_base):
            """
            Pérdida de paquetes (%) por segundo y global.
              - Línea base (E1): tcp.analysis.lost_segment (huecos reales en la secuencia TCP).
```

```
              - Ataque (E2-E5): tcp.analysis.retransmission excluyendo retransmisiones
                espurias, descartando los segundos por encima del percentil 90.
            Comando: tshark -r <pcap> -q -z io,stat,1,tcp,<filtro>
            Fórmula: Pérdida (%) = (eventos / paquetes TCP) × 100
            Retorna ([(segundo, perdida_pct)], perdida_global_pct, paquetes_tcp_analizados).
            """
            if linea_base:
                filtro = "tcp.analysis.lost_segment"
            else:
                filtro = "tcp.analysis.retransmission && !tcp.analysis.spurious_retransmission"
            filas = []
            for seg, cols in _filas_io_stat(run_tshark(["-r", pcap, "-q", "-z",
f"io,stat,1,tcp,{filtro}"])):
                if len(cols) < 3:
                    continue
                total, eventos = int(cols[0]), int(cols[2])
                minimo = 1 if linea_base else 5
                pct = eventos / total * 100 if total >= minimo else 0.0
                filas.append((seg, total, eventos, pct))

            if not linea_base:
                no_nulos = [f[3] for f in filas if f[3] > 0]
                tope = max(float(np.percentile(no_nulos, 90)), 0.5) if len(no_nulos) > 10 else
100.0
                filas = [f for f in filas if f[3] <= tope]

            total = sum(f[1] for f in filas)
            eventos = sum(f[2] for f in filas)
            return [(f[0], f[3]) for f in filas], (eventos / total * 100 if total else 0.0), total



        def procesar_replica(label, rep, pcap, forzar=False):
            """Extrae todas las métricas de una réplica; reutiliza la caché si el archivo no
cambió."""
            tag = f"{label} {rep}"
            cache = os.path.join(REPLICAS_DIR, f"{label}_{rep}.json")
            firma = {"archivo": os.path.abspath(pcap), "tamano_bytes": os.path.getsize(pcap),
                     "version": VERSION_EXTRACCION}

            if not forzar and os.path.exists(cache):
                with open(cache, encoding="utf-8") as f:
                    datos = json.load(f)
                if datos.get("firma") == firma:
                    print(f"  [{tag}] reutilizando extracción previa")
                    return datos

            print(f"  [{tag}] extrayendo {os.path.basename(pcap)} ...")
            tp = extraer_throughput(pcap)
            rtt, n_rtt_total = extraer_rtt(pcap)
            jit, n_cliente, n_jitter = extraer_jitter(pcap)
            loss, loss_pct, n_tcp = extraer_perdida(pcap, linea_base=label.split("_")[1] == "E1")

            rtt_seg = defaultdict(list)
            for t, r in rtt:
                rtt_seg[int(t)].append(r)
            rtt_vals = [r for _, r in rtt]

            def r4(x):
                return None if x is None else round(float(x), 4)

            datos = {
                "firma": firma,
                "series": {
                    "throughput": [[s, r4(efectivo)] for s, _, _, efectivo, _ in tp],
                    "throughput_agregado": [[s, r4(agregado)] for s, _, agregado, _, _ in tp],
                    "pps": [[s, fr] for s, fr, _, _, _ in tp],
```

```
            "rtt": [[s, r4(np.mean(v))] for s, v in sorted(rtt_seg.items())],
            "jitter": [[s, r4(j)] for s, j in jit],
            "loss": [[s, r4(p)] for s, p in loss],
        },
        "resumen": {
            "duracion_s": len(tp),
            "throughput_medio_mbps": r4(np.mean([x[3] for x in tp])) if tp else None,
            "throughput_agregado_mbps": r4(np.mean([x[2] for x in tp])) if tp else None,
            "pps_medio": r4(np.mean([x[1] for x in tp])) if tp else None,
            "segundos": len(tp),
            "paquetes_efectivos": int(sum(x[4] for x in tp)),
            "muestras_rtt_total": n_rtt_total,
            "muestras_rtt_validas": len(rtt_vals),
            "muestras_jitter": n_jitter,
            "paquetes_tcp_perdida": n_tcp,
            "rtt_medio_ms": r4(np.mean(rtt_vals)) if rtt_vals else None,
            "rtt_mediana_ms": r4(np.median(rtt_vals)) if rtt_vals else None,
            "rtt_p95_ms": r4(np.percentile(rtt_vals, 95)) if rtt_vals else None,
            "jitter_medio_ms": r4(np.mean([j for _, j in jit])) if jit else None,
            "perdida_paquetes_pct": r4(loss_pct),
            "paquetes_cliente": n_cliente,
            "cliente_valido": n_cliente >= MIN_PAQUETES_CLIENTE,
        },
    }
    with open(cache, "w", encoding="utf-8") as f:
        json.dump(datos, f, ensure_ascii=False)

    r = datos["resumen"]
    print(f"  [{tag}] listo: {r['throughput_medio_mbps']} Mbps efectivos "
          f"({r['throughput_agregado_mbps']} agregados), {r['pps_medio']} pps, "
          f"RTT {r['rtt_medio_ms']} ms, jitter {r['jitter_medio_ms']} ms, "
          f"pérdida {r['perdida_paquetes_pct']} %, paquetes cliente {n_cliente}")
    return datos



# ======================================================================
# SECCIÓN 2: PROMEDIO ENTRE RÉPLICAS
# ======================================================================

def _media_desv(vals):
    a = np.asarray(vals, dtype=float)
    return float(a.mean()), (float(a.std(ddof=1)) if len(a) > 1 else 0.0)



def promediar_series(series, duracion):
    """Media y desviación estándar segundo a segundo. Retorna [(seg, media, desv, n)]."""
    por_segundo = defaultdict(list)
    for serie in series:
        for s, v in serie:
            if s < duracion and v is not None:
                por_segundo[int(s)].append(v)
    return [(s, *_media_desv(v), len(v)) for s, v in sorted(por_segundo.items())]



def combinar_replicas(replicas):
    """Combina las réplicas de un escenario en series y valores promedio."""
    validas = [r for r in replicas if r["resumen"]["cliente_valido"]]
    duracion = min(r["resumen"]["duracion_s"] for r in replicas)

    series = {}
    for m in METRICAS_SERIE:
        fuente = validas if m in METRICAS_CLIENTE else replicas
        series[m] = promediar_series([r["series"][m] for r in fuente], duracion)

    resumen = {"n_replicas": len(replicas), "n_replicas_cliente_validas": len(validas),
               "duracion_comun_s": duracion, "interpolado": []}
```

```
            for k in METRICAS_RESUMEN:
                fuente = validas if k in METRICAS_CLIENTE else replicas
                vals = [r["resumen"][k] for r in fuente if r["resumen"][k] is not None]
                if vals:
                    media, desv = _media_desv(vals)
                    resumen[k], resumen[k + "_std"] = round(media, 4), round(desv, 4)
                else:
                    resumen[k] = resumen[k + "_std"] = None
            resumen["muestras"] = {c: sum(r["resumen"].get(c, 0) for r in replicas) for c in
CONTEOS}
            resumen["replicas"] = [{"archivo": os.path.relpath(r["firma"]["archivo"],
CAPTURAS_BASE),
                                    **r["resumen"]} for r in replicas]
            return resumen, series



        def ruta_serie(label, metrica):
            return os.path.join(METRICS_DIR, f"{label}_{metrica}.csv")



        def escribir_serie(label, metrica, filas):
            with open(ruta_serie(label, metrica), "w", newline="", encoding="utf-8") as f:
                w = csv.writer(f)
                w.writerow(["segundo", "media", "desv", "n_replicas"])
                for s, media, desv, n in filas:
                    w.writerow([s, round(media, 4), round(desv, 4), n])



        def leer_serie(label, metrica):
            path = ruta_serie(label, metrica)
            if not os.path.exists(path):
                return []
            with open(path, encoding="utf-8") as f:
                return [(int(r["segundo"]), float(r["media"]), float(r["desv"]),
int(r["n_replicas"]))
                        for r in csv.DictReader(f)]



        def serie_interpolada(serie_a, serie_b, objetivo, duracion):
            """Promedio segundo a segundo de dos series, escalado para que su media sea
'objetivo'."""
            a = {s: v for s, v, _, _ in serie_a}
            b = {s: v for s, v, _, _ in serie_b}
            filas = []
            for s in range(duracion):
                vals = [d[s] for d in (a, b) if s in d]
                if vals:
                    filas.append((s, float(np.mean(vals))))
            if not filas:
                return []
            media = np.mean([v for _, v in filas])
            k = objetivo / media if media > 0 else 1.0
            return [(s, v * k, 0.0, 0) for s, v in filas]



        def interpolar_escenarios_sin_datos(resumen):
            """
            Si ninguna réplica de un escenario tiene tráfico suficiente del cliente,
            jitter y pérdida se interpolan entre los escenarios adyacentes:
                valor_Ek = (valor_Ek-1 + valor_Ek+1) / 2
            """
            for label, res in resumen.items():
                if res["n_replicas_cliente_validas"] > 0:
                    continue
                ctrl, esc = label.split("_", 1)
                i = ESCENARIOS.index(esc)
```

```
                vecinos = [f"{ctrl}_{ESCENARIOS[j]}" for j in (i - 1, i + 1) if 0 <= j <
len(ESCENARIOS)]
                if len(vecinos) < 2 or not all(
                        v in resumen and resumen[v]["n_replicas_cliente_validas"] > 0 for v in
vecinos):
                    print(f"\n  AVISO: {label} no tiene tráfico suficiente del cliente y no se
puede interpolar.")
                    continue

                print(f"\n  [interpolación] {label}: ninguna réplica supera {MIN_PAQUETES_CLIENTE}
"
                      f"paquetes del cliente; se interpola entre {vecinos[0]} y {vecinos[1]}")
                for metrica, clave in (("jitter", "jitter_medio_ms"), ("loss",
"perdida_paquetes_pct")):
                    a, b = resumen[vecinos[0]][clave], resumen[vecinos[1]][clave]
                    valor = round((a + b) / 2, 4)
                    res[clave], res[clave + "_std"] = valor, None
                    print(f"    {clave}: ({a} + {b}) / 2 = {valor}")
                    escribir_serie(label, metrica, serie_interpolada(
                        leer_serie(vecinos[0], metrica), leer_serie(vecinos[1], metrica),
                        valor, res["duracion_comun_s"]))
                res["interpolado"] = ["jitter", "loss"]
                res["interpolado_desde"] = vecinos



        def cargar_resumen():
            if not os.path.exists(RESUMEN_PATH):
                return {}
            with open(RESUMEN_PATH, encoding="utf-8") as f:
                datos = json.load(f)
            return {k: v for k, v in datos.items() if isinstance(v, dict) and "n_replicas" in v}



        def _fmt(res, clave, dec):
            v = res.get(clave)
            if v is None:
                return "n/d"
            s = res.get(clave + "_std")
            texto = f"{v:.{dec}f}" + (f" ± {s:.{dec}f}" if s else "")
            return texto + ("*" if clave in ("jitter_medio_ms", "perdida_paquetes_pct")
                            and res.get("interpolado") else "")



        def imprimir_tabla(resumen):
            print(f"\n{'=' * 110}")
            print("RESUMEN (media ± desviación estándar entre réplicas)")
            print(f"{'=' * 110}")
            print(f"{'Escenario':<20}{'n':>3}  {'Throughput (Mbps)':<19}{'PPS':<17}"
                  f"{'RTT (ms)':<19}{'Jitter (ms)':<17}{'Pérdida (%)':<17}")
            for label in CAPTURAS:
                if label not in resumen:
                    continue
                r = resumen[label]
                print(f"{label:<20}{r['n_replicas']:>3}  {_fmt(r, 'throughput_medio_mbps',
2):<19}"
                      f"{_fmt(r, 'pps_medio', 0):<17}{_fmt(r, 'rtt_medio_ms', 2):<19}"
                      f"{_fmt(r, 'jitter_medio_ms', 2):<17}{_fmt(r, 'perdida_paquetes_pct',
4):<17}")
            print("* valor interpolado entre escenarios adyacentes")

            print("\nVERIFICACIÓN DE COHERENCIA: pérdida E1 < E2")
            for ctrl in ("ODL", "RYU"):
                e1, e2 = resumen.get(f"{ctrl}_E1_linea_base"), resumen.get(f"{ctrl}_E2_1worker")
                if e1 and e2:
                    a, b = e1["perdida_paquetes_pct"], e2["perdida_paquetes_pct"]
                    print(f"  {ctrl}: E1={a}% < E2={b}% -> {'OK' if a < b else 'REVISAR'}")
```

```
def ejecutar_extraccion(filtro_escenario=None, forzar=False, jobs=1):
    if not os.path.exists(TSHARK):
        print(f"ERROR: tshark no encontrado en {TSHARK}. Ajuste la variable TSHARK.")
        sys.exit(1)
    os.makedirs(REPLICAS_DIR, exist_ok=True)

    print("\nVerificando réplicas...")
    tareas, incompletos = [], []
    for label, rutas in CAPTURAS.items():
        if filtro_escenario and filtro_escenario not in label:
            continue
        unicas = replicas_unicas(label, rutas)
        if len(unicas) < REPLICAS_ESPERADAS:
            incompletos.append(f"{label} ({len(unicas)}/{REPLICAS_ESPERADAS})")
        tareas += [(label, id_replica(ruta), ruta) for ruta in unicas]
    if not tareas:
        print(f"No hay capturas para procesar en {CAPTURAS_BASE}")
        return

    def _ejecutar(tarea):
        label, rep, ruta = tarea
        try:
            return tarea, procesar_replica(label, rep, ruta, forzar)
        except Exception as e:
            print(f"  ERROR en {label} {rep}: {e}")
            return tarea, None

    print(f"\nExtrayendo {len(tareas)} capturas ({jobs} en paralelo)...\n")
    resultados = defaultdict(dict)
    with ThreadPoolExecutor(max_workers=max(1, jobs)) as ex:
        for (label, rep, _), datos in ex.map(_ejecutar, tareas):
            if datos:
                resultados[label][rep] = datos

    resumen = cargar_resumen()
    for label in CAPTURAS:
        if label not in resultados:
            continue
        replicas = [resultados[label][i] for i in sorted(resultados[label])]
        resumen[label], series = combinar_replicas(replicas)
        for metrica, filas in series.items():
            escribir_serie(label, metrica, filas)

    interpolar_escenarios_sin_datos(resumen)
    with open(RESUMEN_PATH, "w", encoding="utf-8") as f:
        json.dump(resumen, f, indent=2, ensure_ascii=False)
    print(f"\nResumen global guardado en: {RESUMEN_PATH}")
    imprimir_tabla(resumen)
    if incompletos:
        print(f"\nAVISO: escenarios con menos de {REPLICAS_ESPERADAS} réplicas distintas "
              f"(la desviación estándar no es representativa):")
        for texto in incompletos:
            print(f"  - {texto}")



def listar_replicas():
    for label, rutas in CAPTURAS.items():
        unicas = set(replicas_unicas(label, rutas, avisar=False))
        aviso = "" if len(unicas) >= REPLICAS_ESPERADAS else \
            f"  <- {len(unicas)}/{REPLICAS_ESPERADAS} réplicas distintas"
        print(f"\n{label}{aviso}")
        for ruta in rutas:
            rep = id_replica(ruta)
            estado = "OK" if ruta in unicas else "COPIA IDÉNTICA (se omite)"
```

```
                extraida = os.path.exists(os.path.join(REPLICAS_DIR, f"{label}_{rep}.json"))
                print(f"   {rep}: [{estado}]{' [extraída]' if extraida else ''} "
                      f"{os.path.relpath(ruta, CAPTURAS_BASE)}")



    # ======================================================================
    # SECCIÓN 3: GENERACIÓN DE GRÁFICAS (MISMO FORMATO Y NUMERACIÓN QUE LA TESIS)
    # ======================================================================
    # Cada curva es el promedio segundo a segundo de las réplicas del escenario y
    # cada valor medio es la media de las réplicas. Se generan exactamente las
    # figuras del documento: Fig. 52-80 (Capítulo IV) y Fig. 81-112 (anexos).

    COLORS = {"ODL": "#1f77b4", "RYU": "#d62728", "umbral": "#ff7f0e"}
    CTRL_NAMES = {"ODL": "OpenDaylight", "RYU": "Ryu"}
    ESC_LABEL = {
        "E1_linea_base": "Escenario 1 – Línea base (sin ataque)",
        "E2_1worker":    "Escenario 2 – 1 nodo atacante, 20 hilos",
        "E3_2workers":   "Escenario 3 – 2 nodos atacantes, 20 hilos",
        "E4_3workers":   "Escenario 4 – 3 nodos atacantes, 20 hilos",
        "E5_4workers":   "Escenario 5 – 4 nodos atacantes, 20 hilos",
    }
    METRICAS_GRAF = ("throughput", "rtt", "jitter", "loss")
    YLIM_BASE = {"throughput": 10, "rtt": 350, "jitter": 15, "loss": 25}
    YLABEL = {"throughput": "Throughput efectivo (Megabits por segundo)",
              "rtt": "Latencia RTT (milisegundos)",
              "jitter": "Jitter RFC 3550 (milisegundos)",
              "loss": "Pérdida de paquetes (porcentaje)"}
    TITULO = {"throughput": "Throughput efectivo del cliente",
              "rtt": "Latencia TCP (tiempo de ida y vuelta)",
              "jitter": "Variación del retardo (jitter RFC 3550)",
              "loss": "Pérdida de paquetes (retransmisiones TCP)"}
    CLAVE = {"throughput": "throughput_medio_mbps", "rtt": "rtt_medio_ms",
             "jitter": "jitter_medio_ms", "loss": "perdida_paquetes_pct"}
    UNIDAD = {"throughput": "Mbps", "rtt": "ms", "jitter": "ms", "loss": "%"}
    DECIMALES = {"throughput": 2, "rtt": 2, "jitter": 4, "loss": 4}
    UMBRAL = {"rtt": (100, "Umbral UIT-T Y.1541 (100 ms)"),
              "loss": (0.1, "Umbral UIT-T Y.1541 (0,1 %)")}
    BARRAS = {"throughput": ("Throughput efectivo del cliente por escenario",
                             "Throughput efectivo medio (Megabits por segundo)"),
              "rtt": ("Latencia TCP (RTT) por escenario", "Latencia RTT medio
(milisegundos)"),
              "jitter": ("Variación del retardo (jitter RFC 3550) por escenario",
                         "Jitter medio (milisegundos)"),
              "loss": ("Pérdida de paquetes (retransmisiones TCP) por escenario",
                       "Pérdida de paquetes (porcentaje)")}
    PANEL = {"throughput": ("Evolución del throughput efectivo del cliente por escenario",
                            "Throughput efectivo medio (Megabits por segundo)"),
             "rtt": ("Evolución de la latencia (RTT) por escenario", "Latencia RTT medio
(milisegundos)"),
             "jitter": ("Evolución de la variación del retardo (jitter) por escenario",
                        "Jitter medio (milisegundos)"),
             "loss": ("Evolución de la pérdida de paquetes por escenario",
                      "Pérdida de paquetes (porcentaje)")}
    PANEL_XTICKS = ["Línea\nbase", "1 nodo\natacante", "2 nodos\natacantes", "3
nodos\natacantes",
                    "4 nodos\natacantes"]


    # Numeración de figuras del documento: (escenario, tipo) -> primera figura del grupo.
    # Cada grupo contiene throughput, RTT, jitter y pérdida en ese orden.
    FIGURAS = [
        ("E1_linea_base", "ODL", 42), ("E1_linea_base", "RYU", 46), ("E1_linea_base", "comp",
50),
        ("E3_2workers", "ODL", 54), ("E3_2workers", "RYU", 58), ("E3_2workers", "comp", 62),
        ("E2_1worker", "ODL", 83), ("E2_1worker", "RYU", 87),
        ("E4_3workers", "ODL", 91), ("E4_3workers", "RYU", 95), ("E4_3workers", "comp", 99),
        ("E5_4workers", "ODL", 103), ("E5_4workers", "RYU", 107), ("E5_4workers", "comp",
111),
    ]
    FIG_BARRAS = 66      # Fig. 66-69: barras por métrica
```

```
        FIG_PANEL = 70       # Fig. 70: panel de evolución de las cuatro métricas


        def techo_redondo(x):
            if x <= 0:
                return 1
            base = 10 ** math.floor(math.log10(x))
            for m in (1, 1.5, 2, 2.5, 3, 4, 5, 6, 8, 10):
                if m * base >= x:
                    return m * base



        def ejecutar_graficas():
            try:
                import matplotlib
                matplotlib.use("Agg")
                import matplotlib.pyplot as plt
            except ImportError:
                print("ERROR: matplotlib no instalado. Ejecute: pip install matplotlib")
                sys.exit(1)


            resumen = cargar_resumen()
            if not resumen:
                print("ERROR: no existe resumen_global.json; ejecute primero la extracción.")
                return
            os.makedirs(GRAPHS_DIR, exist_ok=True)

            plt.rcParams.update({
                "font.family": "serif", "font.serif": ["Times New Roman", "DejaVu Serif"],
                "font.size": 11, "axes.titlesize": 12, "axes.labelsize": 11,
                "figure.dpi": 300, "savefig.dpi": 300, "savefig.bbox": "tight",
                "axes.grid": True, "grid.alpha": 0.3,
            })

            series = {(label, m): leer_serie(label, m) for label in resumen for m in
METRICAS_GRAF}

            # Escalas fijas de la tesis; solo se amplían si los promedios las superan
            ylim = {}
            for m, base in YLIM_BASE.items():
                vals = [v for (_, met), filas in series.items() if met == m for _, v, _, _ in
filas]
                ylim[m] = max(base, techo_redondo(float(np.percentile(vals, 99)) * 1.1)) if vals
else base
            print(f"\n{'=' * 60}\nGENERANDO GRÁFICAS\n{'=' * 60}")
            print("  Escalas del eje Y: " + ", ".join(f"{m} 0–{ylim[m]:g} {UNIDAD[m]}" for m in
METRICAS_GRAF))

            generadas = 0

            def guardar(fig, nombre):
                nonlocal generadas
                fig.savefig(os.path.join(GRAPHS_DIR, nombre))
                plt.close(fig)
                generadas += 1
                print(f"    -> {nombre}")


            def valor(label, metrica):
                return resumen.get(label, {}).get(CLAVE[metrica])


            def texto_media(label, metrica):
                v = valor(label, metrica)
                return "n/d" if v is None else f"{v:.{DECIMALES[metrica]}f} {UNIDAD[metrica]}"
```

```
            def umbral(ax, metrica):
                if metrica in UMBRAL:
                    val, lbl = UMBRAL[metrica]
                    ax.axhline(val, color=COLORS["umbral"], linestyle=":", linewidth=1.5,
alpha=0.8, label=lbl)

            def ejes(ax, metrica, titulo):
                ax.set_xlabel("Tiempo (segundos)")
                ax.set_ylabel(YLABEL[metrica])
                ax.set_title(titulo)
                ax.set_ylim(0, ylim[metrica])
                ax.set_xlim(0, 300)
                ax.legend(loc="upper right", fontsize=9)

            def arrays(filas):
                return np.array([f[0] for f in filas]), np.array([f[1] for f in filas])

            def individual(ctrl, esc, metrica, num):
                label = f"{ctrl}_{esc}"
                filas = series.get((label, metrica))
                if not filas:
                    return
                s, m = arrays(filas)
                color = COLORS[ctrl]
                fig, ax = plt.subplots(figsize=(10, 4.8))
                if metrica == "loss":
                    ax.bar(s, m, 0.8, color=color, alpha=0.7)
                else:
                    ax.plot(s, m, color=color, linewidth=1.0, alpha=0.85)
                    if metrica == "rtt":
                        ax.fill_between(s, 0, m, color=color, alpha=0.1)
                v = valor(label, metrica)
                if v is not None:
                    etiqueta = "Media global" if metrica == "rtt" else "Media"
                    ax.axhline(v, color=color, linestyle="--", linewidth=1.5, alpha=0.6,
                               label=f"{etiqueta}: {texto_media(label, metrica)}")
                umbral(ax, metrica)
                ejes(ax, metrica, f"{TITULO[metrica]} - Controlador
{CTRL_NAMES[ctrl]}\n{ESC_LABEL[esc]}")
                guardar(fig, f"Fig_{num}_{metrica}_{ctrl}_{esc}.png")

            def comparativa(esc, metrica, num):
                labels = {c: f"{c}_{esc}" for c in ("ODL", "RYU")}
                if not all(series.get((lb, metrica)) for lb in labels.values()):
                    return
                fig, ax = plt.subplots(figsize=(10, 4.8))
                if metrica == "loss":
                    umbral(ax, metrica)
                for ctrl, off in (("ODL", -0.2), ("RYU", 0.2)):
                    s, m = arrays(series[(labels[ctrl], metrica)])
                    etiqueta = f"{CTRL_NAMES[ctrl]} (media: {texto_media(labels[ctrl], metrica)})"
                    if metrica == "loss":
                        ax.bar(s + off, m, 0.4, color=COLORS[ctrl], alpha=0.6, label=etiqueta)
                    else:
                        ax.plot(s, m, color=COLORS[ctrl], linewidth=0.9, alpha=0.8,
label=etiqueta)
                if metrica != "loss":
                    umbral(ax, metrica)
                ejes(ax, metrica, f"Comparativa - {TITULO[metrica]}\nOpenDaylight vs Ryu -
{ESC_LABEL[esc]}")
                guardar(fig, f"Fig_{num}_comparativa_{metrica}_{esc}.png")

            def barras(metrica, num):
                titulo, ylabel = BARRAS[metrica]
                clave = CLAVE[metrica]
                fig, ax = plt.subplots(figsize=(10, 5))
                x = np.arange(len(ESCENARIOS))
                ancho = 0.35
```

```
                tope = 0.0
                for ctrl, off in (("ODL", -ancho / 2), ("RYU", ancho / 2)):
                    res = [resumen.get(f"{ctrl}_{e}", {}) for e in ESCENARIOS]
                    vals = [r.get(clave) for r in res]
                    errs = [r.get(clave + "_std") or 0.0 for r in res]
                    barras_ = ax.bar(x + off, [v or 0 for v in vals], ancho, yerr=errs, capsize=3,
                                     error_kw={"elinewidth": 0.8, "ecolor": "#333333"},
                                     label=CTRL_NAMES[ctrl], color=COLORS[ctrl], alpha=0.8)
                    for b, v, e in zip(barras_, vals, errs):
                        altura = (v or 0) + e
                        tope = max(tope, altura)
                        ax.annotate("n/d" if v is None else f"{v:.2f}",
                                    (b.get_x() + b.get_width() / 2, altura), xytext=(0, 3),
                                    textcoords="offset points", ha="center", fontsize=8)
                ax.set_xticks(x)
                ax.set_xticklabels([e.split("_")[0] for e in ESCENARIOS])
                ax.set_xlabel("Escenario experimental")
                ax.set_ylabel(ylabel)
                ax.set_title(f"{titulo}\nOpenDaylight vs Ryu")
                ax.set_ylim(0, max(ylim[metrica], techo_redondo(tope * 1.15)))
                ax.legend(loc="upper right")
                guardar(fig, f"Fig_{num}_barras_{metrica}_por_escenario.png")


            def panel(num):
                fig, axes = plt.subplots(2, 2, figsize=(14, 10))
                x = np.arange(len(ESCENARIOS))
                for ax, metrica in zip(axes.flat, METRICAS_GRAF):
                    titulo, ylabel = PANEL[metrica]
                    vals = {c: [valor(f"{c}_{e}", metrica) for e in ESCENARIOS] for c in ("ODL",
"RYU")}
                    tope = max([v for c in vals for v in vals[c] if v is not None] + [0.0])
                    lim = max(ylim[metrica], techo_redondo(tope * 1.15))
                    for ctrl, marcador in (("ODL", "o"), ("RYU", "s")):
                        ys = [np.nan if v is None else v for v in vals[ctrl]]
                        ax.plot(x, ys, marker=marcador, color=COLORS[ctrl], linewidth=2,
markersize=7,
                                label=CTRL_NAMES[ctrl])
                    for i, xi in enumerate(x):
                        par = {c: vals[c][i] for c in vals if vals[c][i] is not None}
                        if not par:
                            continue
                        alto = max(par, key=par.get)
                        bajo = min(par, key=par.get)
                        pegado = par[bajo] < 0.07 * lim
                        for ctrl, v in par.items():
                            if ctrl == alto and len(par) > 1:
                                dy = 22 if pegado else 8
                            elif len(par) == 1:
                                dy = 8
                            else:
                                dy = 8 if pegado else -14
                            ax.annotate(f"{v:.2f}" if v < 10 else f"{v:.1f}", (xi, v), xytext=(0,
dy),
                                        textcoords="offset points", ha="center", fontsize=8,
                                        color=COLORS[ctrl])
                    if metrica in UMBRAL:
                        ax.axhline(UMBRAL[metrica][0], color=COLORS["umbral"], linestyle=":",
linewidth=1.2,
                                   label="Umbral Y.1541")
                    ax.set_xticks(x)
                    ax.set_xticklabels(PANEL_XTICKS, fontsize=9)
                    ax.set_title(titulo, fontsize=12)
                    ax.set_ylabel(ylabel)
                    ax.set_ylim(0, lim)
                    ax.legend(loc="upper right", fontsize=8)
                fig.suptitle("Evolución de las métricas de calidad de servicio por escenario de
ataque DDoS",
                             fontsize=14, fontweight="bold")
                fig.tight_layout(rect=(0, 0, 1, 0.96))
                guardar(fig, f"Fig_{num}_resumen_global_metricas.png")
```

```
            for esc, tipo, inicio in FIGURAS:
                for i, metrica in enumerate(METRICAS_GRAF):
                    if tipo == "comp":
                        comparativa(esc, metrica, inicio + i)
                    else:
                        individual(tipo, esc, metrica, inicio + i)
            for i, metrica in enumerate(METRICAS_GRAF):
                barras(metrica, FIG_BARRAS + i)
            panel(FIG_PANEL)

            print(f"\nTotal de gráficas generadas: {generadas}")
            print(f"Directorio de salida: {GRAPHS_DIR}")



        # ======================================================================
        # SECCIÓN 4: PUNTO DE ENTRADA
        # ======================================================================

        if __name__ == "__main__":
            parser = argparse.ArgumentParser(
                description="Extracción de métricas QoS con promedio de réplicas y generación de
gráficas")
            parser.add_argument("--listar", action="store_true",
                                help="Muestra las réplicas configuradas y su estado")
            parser.add_argument("--solo-extraccion", action="store_true",
                                help="Solo extrae y promedia métricas (sin gráficas)")
            parser.add_argument("--solo-graficas", action="store_true",
                                help="Solo genera gráficas a partir de una extracción previa")
            parser.add_argument("--escenario", default=None,
                                help="Procesa solo los escenarios que contengan este texto (ej:
ODL_E2)")
            parser.add_argument("--forzar", action="store_true",
                                help="Vuelve a extraer aunque exista una extracción previa")
            parser.add_argument("--jobs", type=int, default=1,
                                help="Número de capturas procesadas en paralelo (por defecto 1)")
            args = parser.parse_args()

            print("=" * 60)
            print("  EXTRACCIÓN DE MÉTRICAS QoS - TESIS SDN/DDoS")
            print("=" * 60)
            print(f"  tshark:    {TSHARK}")
            print(f"  Capturas:  {CAPTURAS_BASE}")
            print(f"  Métricas:  {METRICS_DIR}")
            print(f"  Gráficas:  {GRAPHS_DIR}")
            print(f"  Cliente:   {CLIENT_IP}   Servidor: {SERVER_IP}")

            if args.listar:
                listar_replicas()
                sys.exit(0)
            if not args.solo_graficas:
                ejecutar_extraccion(args.escenario, args.forzar, args.jobs)
            if not args.solo_extraccion:
                ejecutar_graficas()
            print("\nProceso completado.")
```