Trabajo - Instalación, Administración y Uso de un servicio en Linux

Proxies directos:Squid
Proxies inversos: Ngnix, Apache Traffic Server, HaProxy

Servicios Telemáticos

Departamento de Ingeniería Telemática

<Ricardo Sanabria Vega>

©Servicios 2025/2026

ÍNDICE

1. Objetivos y Alcance **7**
1.1. Introducción 7
1.2. Motivación y funcionalidad del servicio 7
1.3. Documentación bibliográfica **8**
2. Base Teórica **9**
2.1. Descripción del servicio y conceptos implicados **9**
2.2. Protocolos utilizados por el servicio **10**
2.2.1. Protocolos Comunes 12
2.2.2. Protocolos Específicos del servicio 13
3. Evaluación de la implementación estudiada **18**
3.1. Descripción y características de la implementación de servicio estudiada 18
3.1.1. Breve Descripción de la solución adoptada 18
3.1.2. Equipamiento necesario 22
3.1.3. Características y funcionalidades 24
3.2. Comparativa de soluciones existentes en el mercado 25
3.3. Clientes para el servicio **30**
3.3.1. Referencias y características del cliente adoptado 30
3.3.2. Comparativa de clientes existentes en el mercado 33
4. Proceso de instalación y Uso del cliente **36**
4.1. Obtención del software del cliente **37**
4.2. Instalación del cliente **38**
4.2.1. Desinstalación del cliente 38
4.2.2. Primera instalación del cliente 38
4.3. Configuración del cliente **40**
5. Proceso de instalación/administración del servidor **43**
5.1. Obtención del software del servidor **45**
5.2. Instalación del servidor **46**
5.2.1. Primera instalación del servicio 46
5.2.1.1. Instalación desde código fuente (Compilación) 47
5.2.1.2. Instalación desde paquetes/repositorios 48
5.2.2. Actualización del servicio 49
5.2.2.1. Actualización desde código fuente 49
5.2.2.2. Actualización desde paquetes/repositorios 49
5.2.3. Desinstalación del servicio 50
5.2.3.1. Desinstalación desde código fuente 50

5.2.3.2. Desinstalación desde paquetes/repositorios 51
5.3. Configuración del servidor 52
6. Puesta en funcionamiento del servicio 55
6.1. Arranque del servicio con el sistema 55
6.2. Administración y monitorización del funcionamiento 57
6.2.1. Configuración y Uso de los ficheros de registro 57
6.2.2. Arranque del servicio en modo detallado (verbose) 59
6.2.3. Seguridad del servicio 60
6.3. Tests y Pruebas del correcto arranque del servicio 62
7. Diseño de los escenarios de prueba 72
7.1. Escenario de Defensa 1: Seguridad Activa con Inspección TLS (Squid) 73
7.1.1. Escenario 1: Esquema de la red 74
7.1.2. Escenario 1: Configuración del servidor 74
7.1.3. Escenario 1: Tests y Pruebas del Escenario 76
7.1.4. Escenario 1: Análisis del intercambio real de mensajes de red 78
7.2. Escenario de Defensa 2: Eficiencia y Escalabilidad Híbrida (Benchmark) 83
7.2.1. Escenario 2: Esquema de la red 83
7.2.2. Escenario 2: Configuración del servidor 83
7.2.3. **Escenario 2: Tests y Pruebas del Escenario 87**
7.2.4. Escenario 2: Análisis del intercambio real de mensajes de red 95
8. Interfaz gráfica de administración del servidor 98
9. Deficiencias del servicio 103
9.1. Deficiencias en la Arquitectura de Reverse Proxy 103
9.2. Deficiencias en la Herramienta de Administración (SARG) 104
10. Ampliaciones/mejoras del servicio 105
11. Incidencias y principales problemas detectados 108
Problema 1: Conflicto de Permisos y Ejecución del Ayudante SSL (Squid) 108
Problema 2: Conflicto de Puertos y Dependencia Crítica (Nginx/HAProxy) 108
Problema 3: Fallo en la Ejecución del Benchmark (Error de set -e) 109
Problema 4. Persistencia de bloqueos tras borrar reglas 109
Problema 5. Comportamiento diferente entre curl y el navegador 110
Problema 6. Reglas de bloqueo que no funcionaban desde archivos incluidos 110
Conclusión 110
12. Resumen y Conclusiones 110
ANEXO A: Parámetros de configuración y comandos de gestión del servidor 114
ANEXO B: Parámetros de configuración y comandos de gestión del cliente 120

Lista de figuras

Figura 01: Pila de protocolos utilizada por el servicio 15
Figura 02: Esquema Gráfico de la red 16
Figura 03: Esquema Gráfico de la red con protocolos 16
Figura 04: Formato de mensaje http 18
Figura 05: Pila del protocolo http 18
Figura 06: Formato mensaje connect 19
Figura 07: Formato de secuencia mensaje tls 20
Figura 08: Mensaje interno entre proxy y backend 21
Figura 09: Esquema gráfico de la red del escenario1 78
Figura 10: Bloqueo de Acceso a Facebook 81
Figura 11: Contenido Interceptado google.com 82
Figura 12: Diagrama de mensaje Bloqueo de un dominio 85
Figura 13: Diagrama de mensaje SSL-Bump 86
Figura 14: Gráfica Wireshark bloqueo Facebook 86
Figura 15: Gráfica Wireshark ssl-bump (google.com) 86
Figura 16: Esquema gráfico de la red del escenario 2 87
Figura 17: Comparativa RPS 98
Figura 18: Diagrama de Mensajes Escenario 2 102
Figura 19: Gráfica Wireshark ssl-bump (google.com) 102
Figura 20: Captura SARG Top Users 105
Figura 21: Captura SARG Denied 105
Figura 22: Captura SARG Top Sites 107

Lista de tablas

Tabla 01: Tabla Squid 19
Tabla 02: Tabla HAProxy 20
Tabla 03: Tabla Nginx 21
Tabla 04: Tabla ATS 22
Tabla 05: Tabla equipamiento 24
Tabla 06: Comparativa general del mercado 27
Tabla 07: Tabla comparativa servicios usados 30
Tabla 08: Tabla Navegador mozilla firefox 31
Tabla 09: Tabla curl 32

Tabla 10: Tabla ApacheBench 33
Tabla 11: Tabla comparativa mercado de clientes 36
Tabla 12: Sistemas operativos usados por clientes 37
Tabla 13: Comandos usados en curl 42
Tabla 14: Parámetros Benchmark 43
Tabla 15: S.O. sobre el que se usa el servicio 44
Tabla 16: Tabla obtención del software del servidor 46
Tabla 17: Resumen ficheros involucrados en la configuración del servidor 53
Tabla 18: Arranque servidores Flask 56
Tabla 19: Arranque servicios por paquetes 56
Tabla 20: Configuración y Uso de los ficheros de registro 59
Tabla 21: Arranque del servicio en modo detallado 60
Tabla 22: Análisis de la seguridad del servicio 61
Tabla 23: Comprobación arranque correcto 64
Tabla 24: Comprobación de Usuarios 66
Tabla 25: Comprobación Sockets 67
Tabla 26: Comprobación logs 69
Tabla 27: Comprobación Tráfico de red 71
Tabla 28: Comprobación Funcionamiento 72
Tabla 29: Análisis protocolos escenario 1 79
Tabla 30: Prueba Verificación TLS 90
Tabla 31: Prueba Verificación Balanceo de Carga 91
Tabla 32: Prueba Caché ATS 91
Tabla 33: Resultados prueba rendimiento 94
Tabla 34: Protocolos Analizados 96
Tabla 35: Componentes Necesarios para SARG 99

Herramientas analizadas	**Asimiladas a**
Squid Proxy, Nginx, HAProxy, ATS (Apache Traffic Server)	***Servidor***
	Cliente

1. Objetivos y Alcance

1.1. Introducción

En los últimos años, las redes han ido creciendo en complejidad y en cantidad de servicios que ofrecen. Esto hace que sea cada vez más importante tener algún tipo de control sobre el tráfico que entra y sale de una red. Aquí es donde entran en juego los servicios proxy, que básicamente funcionan como intermediarios entre los usuarios y los servidores. Su papel principal es gestionar, filtrar o redirigir las conexiones según las necesidades del sistema.

Este tipo de servicio es útil en muchos entornos distintos. Por ejemplo, dentro de una red interna se puede usar un proxy para limitar el acceso de los usuarios a determinadas páginas o para aplicar políticas de seguridad. En el lado contrario, cuando se trata de proteger un servidor web, un proxy puede encargarse de repartir carga, manejar conexiones cifradas o incluso actuar como primera línea de defensa frente a fallos o ataques.

En general, los proxies ayudan a mejorar la seguridad, el rendimiento y la organización del tráfico. No dependen de un programa concreto, sino del propio concepto de tener una "capa intermedia" que controla lo que pasa entre clientes y servidores. Por eso, hoy en día se han convertido en una herramienta casi imprescindible en la mayoría de las infraestructuras de red modernas.

1.2. Motivación y funcionalidad del servicio

La principal motivación para utilizar un servicio de tipo proxy es disponer de un mayor control sobre el tráfico que circula por la red. Sin una capa intermedia, las peticiones llegan directamente a los servidores o salen a Internet sin ningún tipo de filtro, lo que puede generar problemas de seguridad, falta de organización o incluso sobrecarga en determinados servicios.

Un proxy permite centralizar ese control. Su función es recibir las peticiones y decidir qué hacer con ellas según las políticas definidas: permitir o bloquear accesos, redirigir tráfico, gestionar conexiones cifradas, distribuir carga entre varios servidores o incluso acelerar la entrega de contenido mediante caché. De esta forma, se consigue una red más segura, más eficiente y fácil de administrar.

En general, estos servicios resultan útiles tanto en redes internas, donde se quiere gestionar el acceso de los usuarios, como en entornos donde se alojan aplicaciones web, donde el proxy actúa como punto clave para mejorar la disponibilidad y proteger a los servidores reales. Por ello, los proxies se han convertido en un componente esencial en muchas arquitecturas modernas.

1.3. Documentación bibliográfica

A continuación, se recoge la bibliografía utilizada para documentar los conceptos generales de los servicios proxy y su funcionamiento.
Todas estas referencias podrán citarse en el resto de la memoria usando la notación "[n]".

Referencias

[1] R. Fielding et al., *RFC 2616 – Hypertext Transfer Protocol (HTTP/1.1)*, IETF, 1999.
[2] Kurose, J. & Ross, K., *Computer Networking: A Top-Down Approach*, 8ª edición, Pearson, 2020.
[3] M. Bishop, *Computer Security: Art and Science*, 2ª edición, Addison-Wesley, 2018.
[4] Mozilla Developer Network (MDN), *HTTP – Overview*, Mozilla Foundation, disponible en: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview
[5] Stallings, W., *Foundations of Modern Networking*, Addison-Wesley, 2016.
[6] Cisco Systems, *Layer 7 Application Services Overview*, Cisco Documentation, 2023.
[7] Squid Project, *Squid Web Proxy – Página oficial*, disponible en: http://www.squid-cache.org
[8] Nginx Project, *Nginx – Página oficial*, disponible en: https://nginx.org
[9] Apache Software Foundation, *Apache Traffic Server – Página oficial*, disponible en: https://trafficserver.apache.org
[10] HAProxy Project, *HAProxy – Página oficial*, disponible en: https://www.haproxy.org
[11] curl Project, "curl – Command line tool and library for transferring data", disponible en: https://curl.se
[12] Apache Software Foundation, "Apache HTTP Server – ApacheBench (ab) Tool Documentation", disponible en: https://httpd.apache.org/docs/2.4/programs/ab.html
[13] SARG Project, "Squid Analysis Report Generator (SARG)", disponible en: http://sarg.sourceforge.net
[14] Pallets Project, "Flask - The Python Web Framework", disponible en: https://flask.palletsprojects.com/
[15] Repositorios de Paquetes de Ubuntu (Canonical), disponible en: https://packages.ubuntu.com/

2. Base Teórica

El servicio analizado en este trabajo se basa en el concepto de **proxy**, una tecnología que actúa como intermediaria entre un cliente y un servidor. Su función principal es gestionar, filtrar o modificar el tráfico que circula entre ambos extremos. Este tipo de servicio aparece descrito en la documentación general del funcionamiento de HTTP y sus intermediarios, como los proxies, gateways o túneles [4], y también en manuales de redes y seguridad que explican su papel en la arquitectura de Internet [1][2][3].

La idea de introducir una capa intermedia no es algo nuevo: forma parte de muchas arquitecturas modernas porque permite añadir control, seguridad y optimización sin necesidad de modificar directamente ni los clientes ni los servidores finales. Gracias a ello, los proxies se utilizan tanto en redes internas (para filtrar tráfico y aplicar políticas) como en entornos orientados a servicios web (para mejorar el rendimiento, equilibrar carga o proteger la infraestructura).

2.1. Descripción del servicio y conceptos implicados

Un proxy es un servicio que actúa como intermediario entre un cliente y el recurso al que quiere acceder, normalmente un servidor en Internet. En lugar de que el equipo del usuario se conecte directamente al destino, la petición pasa primero por el proxy, que decide cómo gestionarla y si debe permitirla, modificarla o bloquearla. Esto aporta ventajas tanto para el rendimiento como para la seguridad, tal como se describe en fuentes generales sobre redes y servicios distribuidos [1][2].

Dentro de este tipo de servicio existen dos usos principales:

- **Proxy directo:** el que utilizan los clientes internos para salir a Internet.
- **Proxy inverso:** el que se coloca delante de servidores web para protegerlos y gestionar el tráfico entrante.

En un proxy directo, que es el caso típico usado en redes corporativas o educativas, el servicio permite aplicar políticas centralizadas sobre el acceso a Internet. Según la documentación técnica sobre control de tráfico y proxys [1][4], una de las funciones más comunes es el filtrado y control de páginas. El proxy puede revisar cada petición saliente y compararla con reglas establecidas, lo que permite:

- Bloquear dominios específicos.
- Restringir el acceso según la dirección IP del cliente.

- Definir horarios o categorías permitidas.
- Registrar qué páginas se solicitan (si la política de la red lo requiere).

Esto hace posible controlar el uso de la red sin necesidad de configurar cada cliente uno por uno, ya que todo el tráfico pasa por el mismo punto de control.

Por otro lado, en el caso del proxy inverso, la funcionalidad se orienta a proteger y mejorar el servicio web. En lugar de exponer directamente los servidores backend (servidores de aplicación), el proxy inverso recibe todas las conexiones y se encarga de tareas como:

- Repartir la carga entre varios servidores (balanceo).
- Detectar fallos y redirigir el tráfico a servidores disponibles.
- Terminar las conexiones TLS para reducir carga en los servidores reales.
- Filtrar peticiones malformadas o potencialmente peligrosas.

Gracias a esta arquitectura, los servidores backend quedan aislados del tráfico directo y se gana tanto en seguridad como en estabilidad del servicio [2][3].

2.2. Protocolos utilizados por el servicio

El servicio desarrollado en este proyecto utiliza distintos protocolos de nivel de aplicación para permitir el funcionamiento coordinado del proxy directo (Squid), el proxy inverso (Nginx, HAProxy, ATS) y los servidores backend Flask.
Los protocolos empleados permiten tareas como filtrado de tráfico, establecimiento de túneles HTTPS, terminación TLS, reenvío de solicitudes y balanceo de carga.

En este apartado se resumen todos los protocolos utilizados por el servicio, se indican las referencias normativas correspondientes y se incluyen los esquemas gráficos solicitados: pila de protocolos, esquemas de red, formatos de los mensajes y diagramas de secuencia

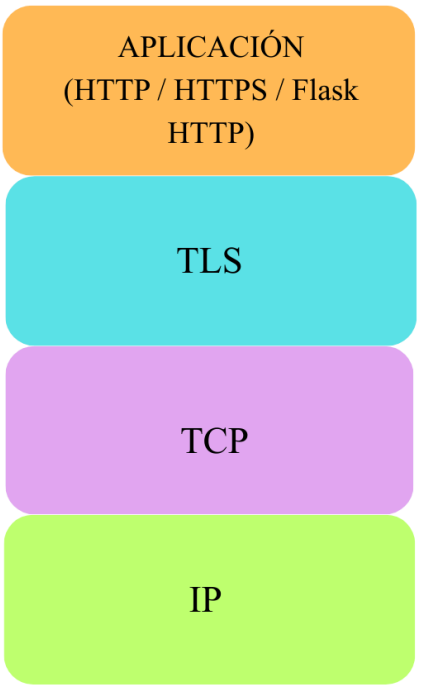

Figura 01: Pila de protocolos utilizada por el servicio

Figura 02: Esquema Gráfico de la red

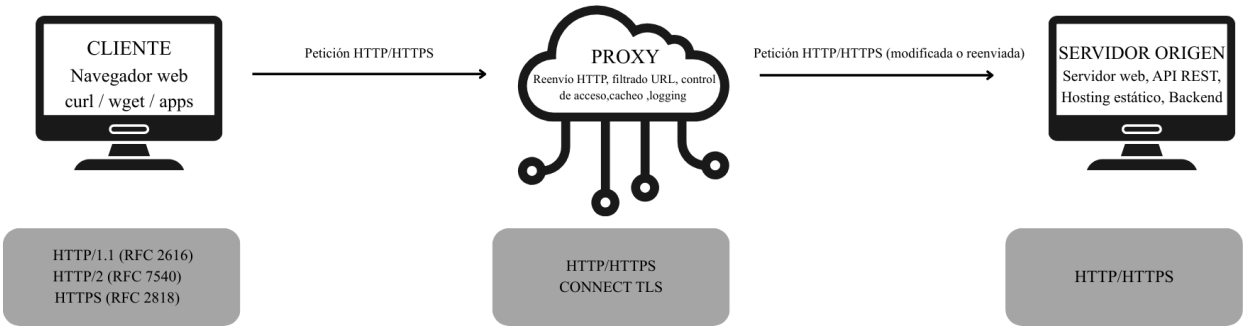

Figura 03: Esquema Gráfico de la red con protocolos

2.2.1. Protocolos Comunes

En este subapartado se incluyen únicamente ejemplos de protocolos de aplicación **habituales en redes TCP/IP**, aunque **no formen parte del funcionamiento del servicio estudiado**.

Estos protocolos se muestran solo como referencia general.

- **SMTP** – RFC 5321
- **DNS** – RFC 1035
- **FTP** – RFC 959
- **DHCP** – RFC 2131
- **POP3 / IMAP** – RFC 1939 / RFC 3501

El servicio desarrollado no emplea estos protocolos, pero se incluyen según lo solicitado para este apartado.

2.2.2. Protocolos Específicos del servicio

En esta sección se describen los protocolos que el servicio estudiado utiliza realmente y que los proxies analizan, filtran, manipulan o reenvían de forma directa durante su funcionamiento.

A diferencia del apartado anterior, aquí se explica su función concreta dentro del proxy, su utilidad y su papel operativo.

1) HTTP – Protocolo principal inspeccionado por los proxies

Documento: RFC 7230–7235

Utilidad dentro del servicio

HTTP es el protocolo principal que los proxies estudian, manipulan y transforman:

- lectura de cabeceras para filtrado (Squid, ATS)
- control de acceso basado en reglas (ACLs)
- modificación o reescritura de cabeceras
- selección de backend en HAProxy/Nginx
- gestión de caché (Squid, ATS)

Formato básico de mensaje analizado

Figura 04: Formato de mensaje http

Pila de protocolos manejada

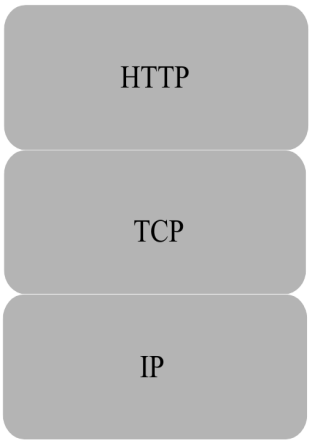

Figura 05: Pila del protocolo http

2) Método CONNECT – Creación de túneles HTTPS a través del proxy

Documento: RFC 2817

Utilidad dentro del servicio

En el modo proxy directo, cuando un cliente solicita acceso a un sitio HTTPS, Squid y ATS utilizan CONNECT para crear un túnel TCP.
El proxy:

- no descifra el tráfico,
- pero controla el dominio al que se intenta acceder,
- permitiendo aplicar políticas de bloqueo/permitir.

Ejemplo de mensaje CONNECT analizado

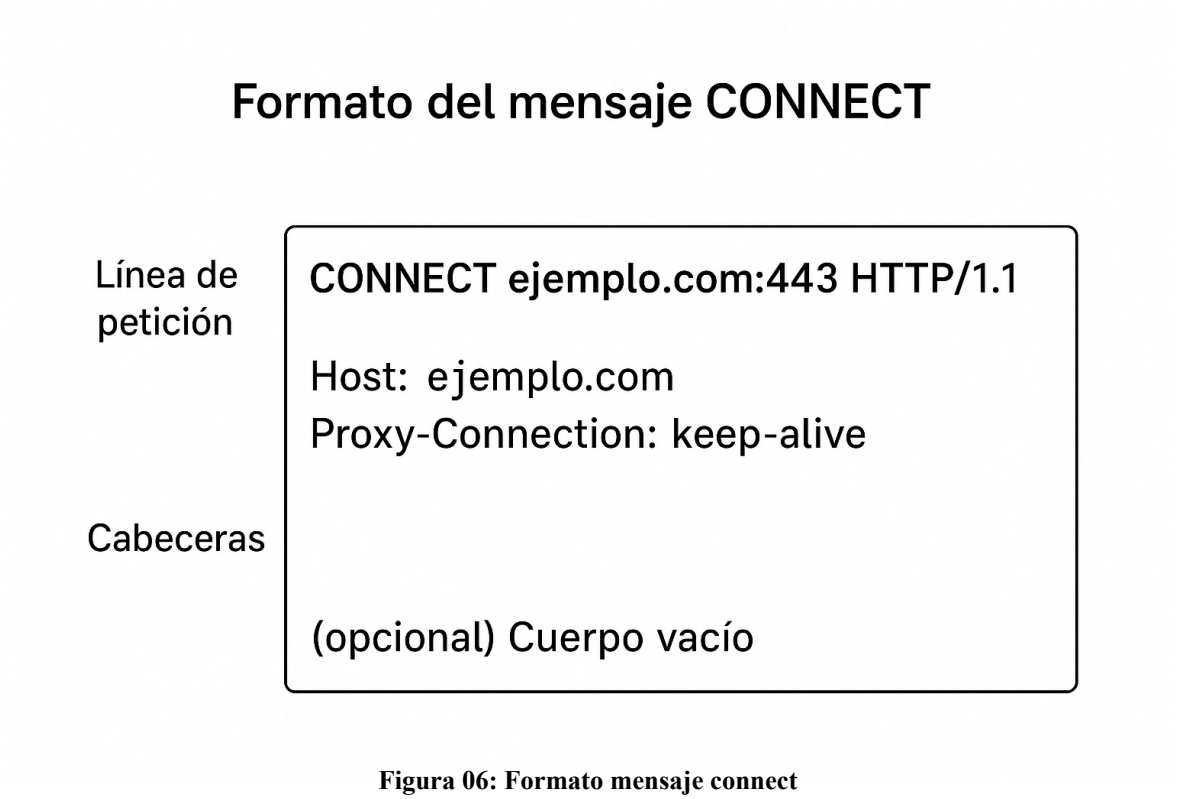

Figura 06: Formato mensaje connect

3) TLS – Protocolo descifrado por el proxy inverso (terminación TLS)

Documento: RFC 5246 (TLS 1.2), RFC 8446 (TLS 1.3)

Utilidad dentro del servicio

En el proxy inverso, Nginx y HAProxy realizan **terminación TLS**, es decir:

- aceptan tráfico HTTPS entrante
- descifran la sesión TLS
- procesan el contenido HTTP en claro
- reenvían al backend mediante HTTP simple

Este punto fue crucial en el benchmark de rendimiento TLS del proyecto.

Formato de Secuencia TLS manejada (simplificada)

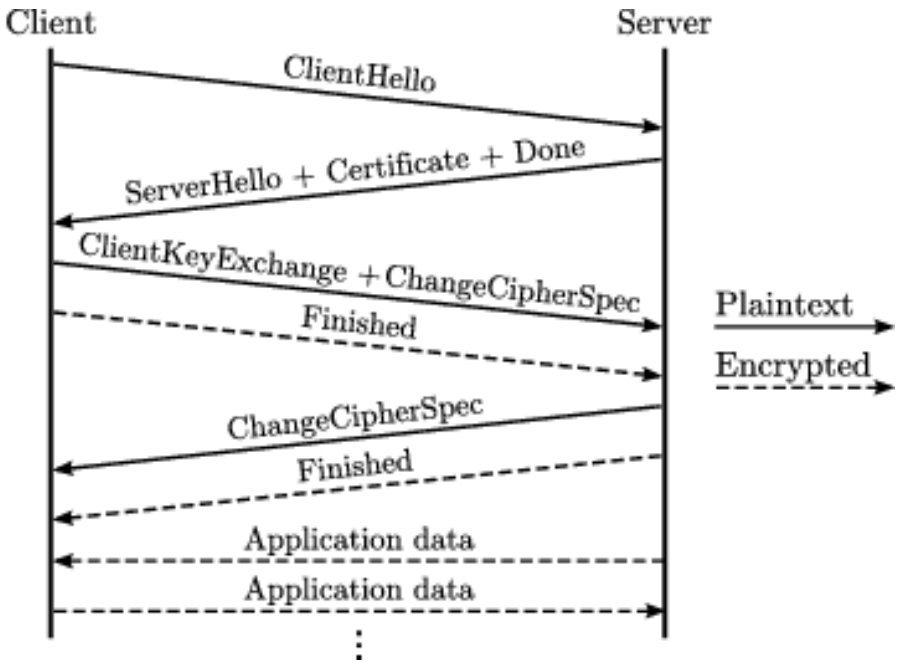

Figura 07: Formato de secuencia mensaje tls

4) HTTP Interno entre Proxy y Backends (Flask)

Documento: RFC 7230

Utilidad dentro del servicio

Una vez procesada la petición entrante, el proxy la reenvía al backend interno (las aplicaciones Flask) usando HTTP simple.

Ejemplo:

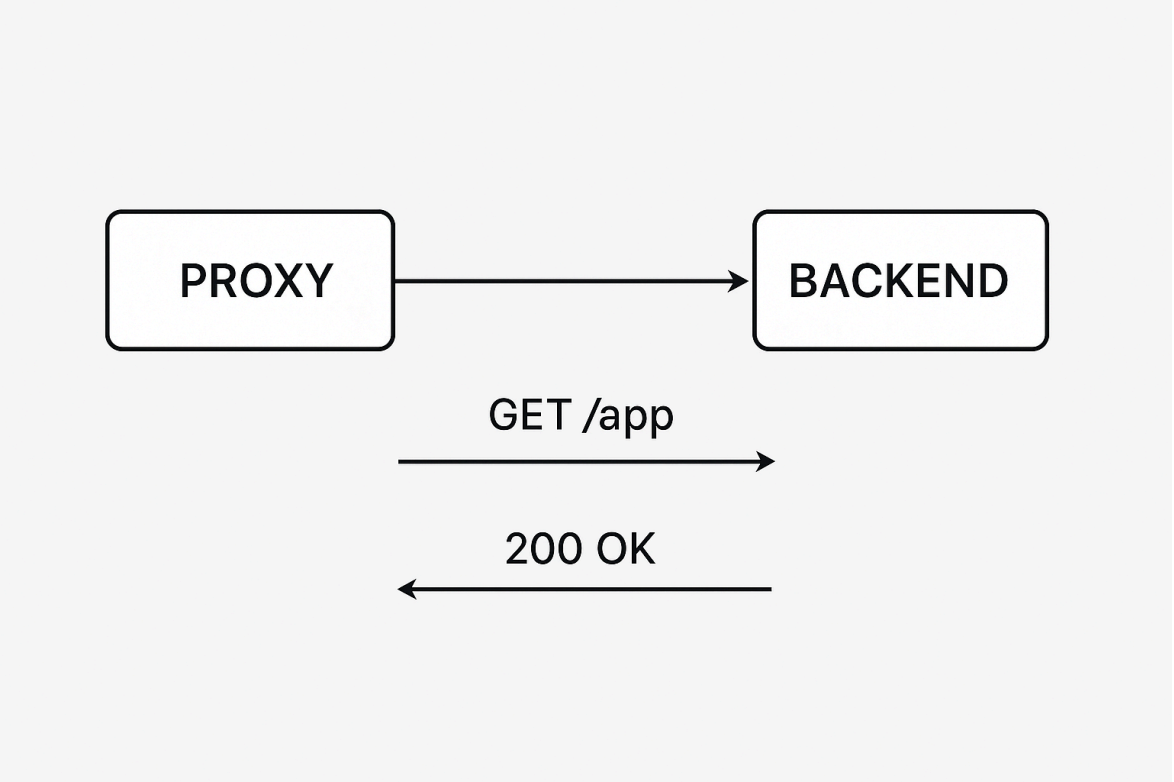

Figura 08: Mensaje interno entre proxy y backend

Este tráfico es interno al servicio y lo gestiona directamente el proxy.

3. Evaluación de la implementación estudiada

La implementación del servicio se examina desde el punto de vista funcional, técnico y comparativo, con el objetivo de valorar su rendimiento y sus ventajas frente a otras tecnologías similares. Para ello, primero se describe con detalle la solución adoptada y, posteriormente, se realiza un análisis comparativo de sus características respecto a otras alternativas disponibles en el mercado.

3.1. Descripción y características de la implementación de servicio estudiada

Para llevar a cabo el proyecto se emplearon varias tecnologías de proxy, cuyo funcionamiento conjunto permite cubrir necesidades distintas dentro de la arquitectura planteada.
La solución integra un proxy directo (Squid) para el control del tráfico saliente, diferentes proxies inversos (Nginx, HAProxy y Apache Traffic Server) para el tratamiento del tráfico entrante y dos servidores Flask utilizados como aplicaciones backend para las pruebas de balanceo y disponibilidad.

Cada una de estas herramientas cumple una función específica dentro de la infraestructura, lo que permite analizar su comportamiento individual y su aportación dentro del sistema global de defensa.

3.1.1. Breve Descripción de la solución adoptada

En la siguiente tabla se presentan las funcionalidades, documentación de referencia y versiones utilizadas de las distintas herramientas que forman parte del sistema. Esta ficha resume la información esencial de cada componente empleado en la práctica.

Tabla 1 — Squid (Proxy directo)

Campo	Información
Funcionalidad que ofrece	Proxy directo (forward proxy), control de acceso basado en ACLs, filtrado de URLs, caché HTTP, logging, soporte para autenticación, retraso de tráfico (delay pools).
Referencias usadas	Documentación oficial del proyecto Squid [7], definiciones de protocolos HTTP/1.1 [1] y recursos generales sobre redes [2].
Autor Desarrollador	Proyecto Squid (The Squid Software Foundation).
Versión usada en el trabajo	Squid 6.13
Última versión disponible	Squid 7.3 — 2025-10-28 .
Licencia	GPLv2.

Tabla 01: Tabla Squid

Tabla 2 — HAProxy (Proxy inverso)

Campo	Información
Funcionalidad que ofrece	Proxy inverso, balanceo de carga avanzado L7, detección de fallos (health checks), failover automático, terminación TLS, alta capacidad para sistemas críticos.
Referencias usadas	Documentación oficial HAProxy [10], especificaciones HTTP/1.1 [1], fundamentos de servicios distribuidos [2][5].
Autor Desarrollador	Willy Tarreau y colaboradores del proyecto HAProxy.
Versión usada en el trabajo	HAProxy 2.8.5
Última versión disponible	HAProxy 3.4. — 2025-11-26
Licencia	GPLv2.

Tabla 02: Tabla HAProxy

Tabla 3 — Nginx (Proxy inverso)

Campo	Información
Funcionalidad que ofrece	Proxy inverso, terminación TLS, servidor web estático, balanceo simple, reenvío de tráfico, alta eficiencia en el manejo concurrente.
Referencias usadas	Documentación oficial de Nginx [8], fundamentos HTTP/HTTPS [1][4], material de arquitectura de servicios [2].
Autor Desarrollador	Igor Sysoev, NGINX Inc.
Versión usada en el trabajo	Nginx 1.24.0 (Ubuntu).
Última versión disponible	Nginx 1.29.3 — 2025-10-28 (rama estable).
Licencia	BSD-2-Clause.

Tabla 03: Tabla Nginx

Tabla 4 — Apache Traffic Server (Proxy inverso)

Campo	Información
Funcionalidad que ofrece	Proxy inverso de alto rendimiento, caché HTTP avanzado, optimización de tráfico, soporte para TLS, plugins de modificación de tráfico.
Referencias usadas	Documentación oficial ATS [9], materiales teóricos sobre proxies HTTP [2][4].
Autor Desarrollador	Apache Software Foundation.
Versión usada en el trabajo	ATS 9.2.3
Última versión disponible	ATS 10.1.0 — 2025-08-13
Licencia	Apache License 2.0.

Tabla 04: Tabla ATS

3.1.2. Equipamiento necesario

Para llevar a cabo la instalación, configuración y evaluación de los servicios proxy utilizados en este trabajo, ha sido imprescindible disponer de una infraestructura mínima que garantizara un entorno estable y reproducible.

En la siguiente ficha se detalla el equipamiento empleado, tanto a nivel de hardware como de red y software.

Categoría	Especificación
Hardware necesario	• Dos máquinas virtuales (o físicas) con arquitectura x86_64 .• Procesador recomendado: 2 vCPUs por máquina .• Memoria: 2–4 GB RAM por VM. • Almacenamiento: 20–40 GB por instancia. • Tarjeta de red virtual en modo NAT o Bridge.
Infraestructura de red requerida	• Conectividad interna entre Proxy ↔ Backend .• Acceso a Internet para instalación y pruebas con HTTP/HTTPS. • Puertos configurados según el servicio: ─ 3128 (Squid) ─ 80/443 (Nginx) ─ 8080/8443 (ATS) ─ 9000/9001 (HAProxy si aplica)
Elementos software necesarios	• Intérprete Python 3 y microframework Flask (para los servidores backend). • Librerías OpenSSL .• Herramientas de diagnóstico: curl, wget, ss/netstat .• Servicios analizados: ─ Squid 6.13 ─ Nginx 1.24.0 (compilado desde código fuente) ─ HAProxy 2.8.5 ─ Apache Traffic Server 9.2.3
Sistema Operativo utilizado	Ubuntu 24.04.3 LTS (Noble) • Distributor ID: Ubuntu • Release: 24.04

	• Codename: noble

Tabla 05: Tabla equipamiento

3.1.3. Características y funcionalidades

La implementación desarrollada en este proyecto incorpora un conjunto amplio de funcionalidades orientadas tanto a la gestión del tráfico saliente como a la protección del tráfico entrante, integrando capacidades propias de proxies directos e inversos.

El diseño combina distintos servicios especializados (Squid, Nginx, HAProxy y Apache Traffic Server), de forma que cada uno aporta características complementarias dentro de la arquitectura.

Entre las funcionalidades más relevantes de la solución adoptada destacan las siguientes:

• Control de acceso y filtrado de tráfico

La implementación permite aplicar políticas de acceso basadas en listas de control (ACLs), restringiendo dominios, direcciones IP o rangos de red. Esta capacidad, proporcionada principalmente por Squid, facilita la aplicación de políticas diferenciadas para usuarios concretos y permite segmentar el acceso a Internet desde la red interna.

• Reenvío y tratamiento de peticiones HTTP/HTTPS

Todos los servicios empleados ofrecen la capacidad de recibir solicitudes HTTP/HTTPS, procesarlas y reenviarlas al destino correspondiente. En el caso de los proxies inversos, esto incluye la reescritura de cabeceras, el manejo de rutas y la intermediación entre clientes externos y los servidores backend desplegados.

• Terminación TLS (TLS Offloading)

Nginx y HAProxy aportan soporte de terminación TLS, permitiendo que la negociación criptográfica se realice en la capa de proxy. Esta funcionalidad reduce la carga de los servidores backend y posibilita una gestión centralizada de certificados, fortaleciendo la seguridad del sistema.

• Balanceo de carga y tolerancia a fallos

La arquitectura implementada incorpora balanceo de carga a nivel de aplicación mediante HAProxy, distribuyendo peticiones entre varios servidores backend Flask. Además, se ha habilitado detección de errores mediante *health checks*, permitiendo redirigir automáticamente el tráfico al servidor operativo en caso de fallo (failover).

• Caché de contenido

Squid y Apache Traffic Server incluyen mecanismos de cacheo HTTP que permiten almacenar respuestas frecuentes para mejorar el rendimiento y reducir la latencia percibida por los clientes. Aunque no ha sido el enfoque principal del proyecto, estas capacidades forman parte del funcionamiento general de los servicios utilizados.

• Registro y monitorización del tráfico

La implementación dispone de mecanismos de *logging* que permiten registrar las solicitudes procesadas, el estado de las conexiones, los fallos detectados y el rendimiento en tiempo real. Este registro resulta esencial para depuración, auditorías y análisis de rendimiento.

• Integración con aplicaciones backend

Como parte del entorno de pruebas, se han desplegado dos servidores Flask que actúan como aplicaciones backend. La solución proxy es capaz de distribuir peticiones entre ambos, reenviar respuestas y garantizar que la capa de backend se mantenga aislada y protegida del acceso directo del cliente.

3.2. Comparativa de soluciones existentes en el mercado

A continuación, se presenta un análisis comparativo entre las principales soluciones de proxy y servidores intermedios disponibles en el mercado. Aunque en el apartado anterior se describió la implementación adoptada para este trabajo, en este punto se contrasta dicha selección con otras alternativas similares, open source, activas y autoalojadas, cumpliendo así los criterios establecidos en el enunciado.

El objetivo es situar cada servicio dentro del ecosistema actual, identificando sus ventajas, limitaciones, nivel de adopción y casos de uso más frecuentes. Este análisis permite

justificar la elección de los servicios utilizados en el trabajo, así como comprender las diferencias funcionales entre ellos.

La comparativa se divide en dos partes tal y como exige la guía:

1. Comparativa general del mercado (Squid, Nginx, HAProxy, ATS, Lighttpd, Apache HTTPD y Varnish).

2. Comparativa específica entre los servicios utilizados en el trabajo, presentada a continuación.

Tabla 01. Comparativa general del mercado (Squid, Nginx, HAProxy, ATS, Lighttpd, Apache HTTPD y Varnish)

CARACTERÍSTICAS	SQUID	NGINX	HAPROXY	ATS	LIGHTTPD	APACHE HTTPD	VARNISH HTTP CACHE
ROL PRINCIPAL	Proxy Directo / Caché	Proxy Inverso / Servidor Web/balanceador de carga	Proxy Inverso / balanceador de carga	Proxy Inverso	Servidor Web / Proxy Inverso	Servidor Web / Proxy Inverso	Proxy Inverso / balanceador de carga
ÚLTIMA VERSIÓN (FECHA) & ESTADO ACTUAL	v7.3 — 2025-10-28	Stable 1.28.0—2025-04-23 Latest 1.29.3—2025-10-28	Stable 3.3.0 —2025-11-26 Latest 3.4—2025-11-26	10.1.0— 2025-08-13	11.4.82—2025-09-12	12.4.65—2025-07-23	8.0.0—2025-09-15
LENGUAJE DE DESARROLLO	C / C++	C	C	C++	C	C	C
LICENCIA / (ESTADO) / WEB OFICIAL	GPL (software libre)—proyecto activo; web oficial: squid-cache.org	BSD-2-Clause (open source) — mantenido por F5 Networks; web oficial: nginx.org	GPLv2 (open source)—HAProxy Technologies; web oficial: haproxy.org	Apache License 2.0 (Open Source)—Apache Software Foundation; web oficial: trafficserver.apache.org	BSD-3-Clause (open source) —mantenido por una comunidad de desarrolladores; web oficial:lighttpd.net	Apache License 2.0 (Open Source)—Apache Software Foundation; web oficial:apache.org	BSD license (2-clause / BSD) open source—gestionado por Varnish Software; web oficial: https://varnish-cache.org/
SISTEMAS OPERATIVOS SOPORTADOS	Linux, BSD, Windows, Unix-like (multiplataforma)	Linux, BSD, macOS, Windows, otros sistemas UNIX-like	Linux, FreeBSD, OpenBSD, Solaris, AIX, macOS, Cygwin	Cross-platform en sistemas tipo Unix / Linux	Linux, BSD, Unix-like	Linux, BSD, Windows, Unix-like	Linux, BSD, Unix-like
NIVEL DE USO ESTIMADO / CUOTA DE MERCADO / USO RELATIVO	Uso difundido especialmente en caching / proxies; muchas instalaciones en ISPs, organizaciones, redes corporativas; tiene historia larga y es considerado estable y robusto.	Muy alto — es probablemente el servidor web / proxy más usado a nivel mundial. Según datos recientes de mercado: ~33.8 % de todos los sitios con servidor conocido usan Nginx.	Alta adopción especialmente en entornos que requieren performance, resiliencia, balanceo. Hay estadísticas que lo ubican con un uso significativamente superior al de proxies más antiguos en ciertos contextos	Uso menor comparado con Nginx/HAProxy/Varnish en términos de cuota global; no aparece como dominante en encuestas globales	Uso moderado — no es de los más populares en webs de alto tráfico, pero sigue siendo usado especialmente donde se requiere ligereza.	Alto uso, aunque ha perdido terreno frente a Nginx; sigue presente en muchísimos servidores web. Estadísticas recientes: segundo servidor más usado en muchos rankings globales.	Alto uso para caching HTTP estático / dinámico — muy popular en entornos con alto tráfico, CDN, sitios con demanda alta de contenido; muchas webs de gran escala lo emplean.
PRINCIPALES DEPENDENCIAS / REQUERIMIENTOS DESTACADOS	Para compilación en Linux suelen requerirse librerías como OpenSSL, librerías de cifrado, librerías de red/SSL, etc. Por ejemplo en Debian: libssl-dev, libdb-dev, libldap, etc.	Soporte de bibliotecas/ componentes comunes: OpenSSL para TLS/SSL, zlib/compresión, PCRE para expresiones regulares (por módulos), módulos internos para proxy/cache, etc.	Sus dependencias básicas suelen ser librerías estándar del sistema; para TLS puede usar OpenSSL; confiable en entornos Unix.	Al ser proxy/cache, depende del sistema operativo + librerías típicas de red, sistema operativo, almacenamiento; configuración flexible mediante plugins/módulos si se usa	Para SSL requiere OpenSSL (u otra librería TLS), soporte de módulos, librerías estándar de sistema.	Dependencias variables según módulos: OpenSSL para TLS/SSL, módulos externos, librerías del sistema; configuración suele ser modular y depende del stack del servidor.	Depende del sistema operativo, librerías de red, memoria/IO. Usa su propio lenguaje de configuración (VCL) que se compila a C, lo que exige compilador, sistema operativo compatible

Tabla 06: Comparativa general del mercado

Tabla 02. Comparativa detallada entre las implementaciones usadas en el trabajo (Squid, Nginx, HAProxy y ATS)

Característica	Squid	Nginx	HAProxy	Apache Traffic Server (ATS)
Rol principal	Proxy directo, proxy inverso, caché HTTP	Servidor web, proxy inverso, balanceador	Proxy inverso, balanceador de carga L7	Proxy inverso, caché de alto rendimiento
Punto fuerte principal	Cacheo avanzado y filtrado HTTP	Rendimiento, estabilidad y versatilidad	Balanceo de carga robusto y fiable	Rendimiento extremo y escalabilidad en CDNs
Punto débil principal	Alto consumo en configuraciones grandes	No es un proxy "puro", depende de módulos	Menos flexible para funciones ajenas al balanceo	Curva de aprendizaje elevada y menos popular
Facilidad de configuración	Media/Alta (config detallada)	Alta, sintaxis simple	Media, requiere conocimiento del modelo	Media/Alta, pero con gran complejidad interna
Rendimiento general (L7)	Medio/Alto	Alto	Muy alto	Muy alto

Eficiencia como Proxy Directo	Excelente (filtrado, ACLs, caching)	Aceptable	Limitado (no es su objetivo)	Aceptable
Eficiencia como Proxy Inverso	Buena	Muy alta	Muy alta	Muy alta
Cacheo HTTP	Muy avanzado (mejor en su categoría)	Básico/medio	Limitado	Muy avanzado (orientado a CDNs)
Balanceo de carga	Básico	Bueno	Excelente (mejor en su categoría)	Bueno
Impacto en CPU y RAM	Alto en configuraciones grandes	Muy eficiente	Muy eficiente	Muy eficiente pero exige optimización
Escalabilidad	Media/Alta	Muy alta	Muy alta	Muy alta (diseñado para grandes CDNs)
Seguridad (TLS, filtrado)	Fuerte para filtrado; TLS completo	TLS muy optimizado	TLS bien integrado; fuerte seguridad	TLS muy optimizado y moderno
Registro y monitorizaciónn	Muy detallado	Completo	Excelente (stats API	Completo con métricas internas

			muy detallada)	
Casos de uso ideales	Control de acceso, filtrado, colegios, empresas	Servidores web, APIs, microservicios	Balanceo de carga de alto rendimiento	CDNs, grandes plataformas de contenido
Nivel de uso en el mercado	Alto en entornos educativos/empresas	Muy alto (de los más usados del mundo)	Muy alto en entornos críticos y cloud	Medio/bajo fuera de grandes empresas
Licencia	GPLv2	BSD-2-Clause	GPLv2	Apache License 2.0
Lenguaje	C/C++	C	C	C++
Última versión disponible	Squid 7.3 (2025-10-28)	Nginx 1.29.3 (2025-10-28)	HAProxy 3.4 (2025-11-26)	ATS 10.1.0 (2025-08-13)
Versión usada en el trabajo	Squid 6.13 (2024)	Nginx 1.24.0	HAProxy 2.8.5	ATS 9.2.3
Dependencias destacadas	OpenSSL, librerías de red/SSL, ACLs	PCRE/PCRE2, OpenSSL	OpenSSL, librerías del sistema	OpenSSL, librerías de red, módulo de plugins
Proyecto en activo	Sí	Sí	Sí	Sí

Tabla 07: Tabla comparativa servicios usados

3.3. Clientes para el servicio

Los clientes HTTP/HTTPS son las herramientas encargadas de enviar peticiones al servicio proxy para verificar su funcionamiento, medir su rendimiento o simplemente consumir recursos a través de él. En el mercado existe una gran variedad de clientes, desde navegadores web, herramientas de línea de comandos, clientes de benchmarking y aplicaciones específicas para pruebas de API.

En este apartado se documenta el cliente adoptado para el trabajo y se realiza una comparativa con otras soluciones ampliamente utilizadas y activas en el mercado.

3.3.1. Referencias y características del cliente adoptado

En el desarrollo del proyecto se han utilizado varios tipos de clientes, cada uno con un propósito específico:

- **Firefox** → Cliente gráfico para validar navegación real.
- **curl** → Cliente CLI para verificar funcionamiento técnico del proxy.
- **ApacheBench (ab)** → Generación de carga y benchmark de rendimiento TLS

A continuación se presenta la ficha de cada uno.

Tabla 1 — Cliente gráfico (Firefox)

Campo	Información
Cliente utilizado	Mozilla Firefox
Función en el proyecto	Verificación práctica de navegación a través del proxy (acceso directo).
Autor / desarrollador	Mozilla Foundation

Versión usada en el proyecto	Firefox ESR 128 (Ubuntu 24.04)
Última versión disponible	Firefox 133 (Enero 2025)
Licencia	MPL 2.0
Sistemas operativos	Linux, Windows, macOS, BSD
Requisitos hardware	CPU x86_64, 2 GB RAM mínimo
Página oficial	https://www.mozilla.org

Tabla 08: Tabla Navegador mozilla firefox

Tabla 2 — Cliente CLI (CURL)

Campo	Información
Cliente utilizado	CURL
Función en el proyecto	Pruebas técnicas HTTP/HTTPS, envío de cabeceras, validación CONNECT y proxy inverso.
Autor / desarrollador	Daniel Stenberg + comunidad cURL
Versión usada	curl 8.5.0 (Ubuntu 24.04)
Última versión disponible	curl 8.10.1 (Enero 2025)

Licencia	MIT
Sistemas operativos	Linux, Windows, macOS, BSD, Solaris, etc.
Requisitos hardware	Cualquiera compatible con OpenSSL/libcurl
Página oficial	https://curl.se

Tabla 09: Tabla curl

Tabla 3 — Cliente de benchmark (ApacheBench)

Campo	Información
Cliente utilizado	ApacheBench (ab)
Función en el proyecto	Benchmark TLS en el Escenario 2 (proxy inverso)
Autor / desarrollador	Apache Software Foundation
Versión usada	2.3 (incluido en Apache HTTP Server 2.4)
Última versión disponible	2.3 (2025)
Licencia	Apache License 2.0
Sistemas operativos soportados	Linux, BSD, macOS
Requisitos hardware	CPU básica, herramientas de desarrollo

URL	https://httpd.apache.org

Tabla 10: Tabla ApacheBench

3.3.2. Comparativa de clientes existentes en el mercado

A continuación se presenta una comparativa entre los principales clientes HTTP/HTTPS disponibles actualmente, incluyendo navegadores, herramientas de línea de comandos y aplicaciones para pruebas de API. Todos ellos son activos, gratuitos, compatibles con Linux y ampliamente utilizados.

Cliente	Última versión	Funcionalidad	SO soportados	Lenguaje	Licencia	URL
Firefox	133 (2025)	Navegador completo, soporte HTTP/2 y HTTP/3	Linux, Win, macOS	C++/Rust	MPL 2.0 — Gratuito (Open Source)	mozilla.org
Chrome	131 (2025)	Navegador rápido con QUIC/HTTP3	Linux, Win, macOS	C++	Propietary (Google Chrome) — Gratuito; Chromium: BSD — Gratuito (Open Source)	google.com/chrome

Chromium	131 (2025)	Navegador libre con motor Blink	Linux, Win, macOS	C++	BSD — Gratuito (Open Source)	chromium.org
Brave	1.69 (2025)	Navegador centrado en privacidad	Linux, Win, macOS	C++	MPL / BSD — Gratuito (Open Source)	brave.com
curl	8.10.1	CLI HTTP/HTTPS, soporte proxy, autenticación	Linux, Win, macOS	C	MIT — Gratuito (Open Source)	curl.se
Wget	1.24	Descarga HTTP/HTTPS/ FTP con proxy	Linux, BSD	C	GPLv3 — Gratuito (Open Source, copyleft)	gnu.org/wget
HTTPie	3.2	Cliente HTTP amigable para APIs	Linux, Win, macOS	Python	BSD — Gratuito (Open Source)	httpie.io
xh	0.22	Alternativa rápida a HTTPie	Linux, Win, macOS	Rust	MIT — Gratuito (Open Source)	github.com/ ducaale/xh

Postman	11.x	Cliente gráfico avanzado para APIs	Linux, Win, macOS	JS	Proprietary / Freemium — Versión gratuita limitada; planes de pago para funcionalidades avanzadas	postman.com
Insomnia	9.x	Cliente API profesional	Linux, Win, macOS	JS	BSD — Gratuito (Open Source core)	insomnia.rest
RESTer	4.x	Cliente API ligero (extensión Firefox)	Linux, Win, macOS	JS	MPL — Gratuito (Open Source)	addons.mozilla.org
ApacheBench (ab)	2.3	Generador de carga HTTP/HTTPS	Linux, BSD, macOS	C	Apache 2.0 — Gratuito (Open Source)	httpd.apache.org
hey	0.2	Benchmark simple compatible con curl	Linux, macOS	Go	Apache 2.0 — Gratuito (Open Source)	github.com/rakyll/hey
wrk	4.2.0	Benchmark muy eficiente	Linux, macOS	C	Apache 2.0 — Gratuito	github.com/wg/wrk

					(Open Source)	

Tabla 11: Tabla comparativa mercado de clientes

4. Proceso de instalación y Uso del cliente

Para la validación funcional y el análisis de rendimiento del sistema se emplearon tres clientes distintos, cada uno con un objetivo concreto dentro del proyecto:

- **Mozilla Firefox**, utilizado como cliente de navegación real para la comprobación del proxy directo Squid, incluyendo filtrado, ACL y análisis de tráfico HTTPS mediante SSL-Bump.

- **Curl**, empleado como cliente de línea de comandos para la verificación técnica de peticiones HTTP y HTTPS, así como del funcionamiento del método CONNECT a través del proxy.

- **ApacheBench (ab)**, utilizado como cliente de carga para la ejecución del benchmark TLS del Escenario 2, evaluando el rendimiento de los proxies inversos.

Todos los clientes fueron ejecutados sobre Ubuntu 24.04.3 LTS, que constituye el sistema operativo del laboratorio empleado durante el desarrollo y la defensa del trabajo.

Cliente	Sistema Operativo usado
Firefox	Ubuntu 24.04.3 LTS
curl	Ubuntu 24.04.3 LTS
ApacheBench (ab)	Ubuntu 24.04.3 LTS

Tabla 12: Sistemas operativos usados por clientes

4.1. Obtención del software del cliente

Firefox

Disponible desde la página oficial de Mozilla en varios formatos: binarios precompilados, paquetes .deb, versión ESR y Snap.
URL: https://www.mozilla.org/firefox

(Referencia: [4])

curl

El proyecto curl proporciona código fuente, binarios precompilados para diversas plataformas y paquetes oficiales para Linux.
URL: https://curl.se

(Referencia: [11])

ApacheBench (ab)

ApacheBench forma parte del paquete apache2-utils, distribuido junto con el proyecto Apache HTTP Server.
URL: https://httpd.apache.org

(Referencia: [12])

4.2. Instalación del cliente

Los pasos descritos a continuación reflejan exactamente los comandos ejecutados durante el desarrollo del proyecto, siguiendo la recomendación de realizar primero la desinstalación cuando el software ya se encontraba presente en el sistema.

4.2.1. Desinstalación del cliente

Desinstalación Firefox

```
sudo apt remove firefox
```

Desinstalación Curl

```
sudo apt remove curl
```

Desinstalación ApacheBench

```
sudo apt remove apache2-utils
```

4.2.2. Primera instalación del cliente

Instalación Firefox

```
sudo apt install firefox
firefox --version
```

Instalación curl

```
sudo apt update
sudo apt install curl
curl --version
```

Instalación ApacheBench

La instalación de ApacheBench se realizó mediante un script automatizado desarrollado durante el proyecto, con el objetivo de garantizar un proceso reproducible y homogéneo en los distintos entornos utilizados.

Antes de su ejecución, fue necesario asignar permisos de ejecución al script:

```
sudo chmod +x instalar_benchmark.sh
```

Posteriormente, el script se ejecuta con privilegios de administrador:

```
sudo ./instalar_benchmark.sh
```

El script llevó a cabo de forma automática los siguientes pasos:

1. Actualización del índice de paquetes del sistema, para asegurar la instalación de versiones coherentes con el repositorio configurado:

   ```
   apt update
   ```

2. Instalación del paquete apache2-utils, que incluye la herramienta ApacheBench:

   ```
   apt install -y apache2-utils
   ```

3. Verificación de la instalación, comprobando la disponibilidad y versión del cliente:

   ```
   ab -V
   ```

Este procedimiento permitió instalar ApacheBench sin intervención manual adicional y dejó el cliente listo para su uso inmediato en las pruebas de carga del Escenario 2.

4.3. Configuración del cliente

Para garantizar la correcta comunicación entre los clientes y los servicios, fue necesario implementar diversas configuraciones. La naturaleza de estas configuraciones varió según el cliente utilizado y el escenario evaluado. Las instrucciones que se presentan a continuación reflejan solo las acciones que se llevaron a cabo durante la ejecución del proyecto**.**

Configuración Firefox

Mozilla Firefox se empleó como cliente principal para validar la funcionalidad del proxy directo Squid, centrándose en el filtrado de tráfico y la inspección de conexiones HTTPS (SSL Bump).

- Configuración del Proxy:
  - El navegador fue configurado manualmente para dirigir todo su tráfico a la interfaz del proxy:
    - **Ruta de Acceso:** Ajustes→ General → Configuración de red → Configuración manual del proxy.
    - **Parámetros**:
      - Proxy HTTP: 127.0.0.1
      - Puerto: 3128
      - Uso del mismo proxy para HTTPS: Habilitado.

    Es esencial que, durante esta configuración, solo se especificarán los parámetros de Proxy HTTP y HTTPS. Los campos relacionados con SOCKS Host y la opción de Proxy DNS permanecieron sin marcar, garantizando que todo el flujo de tráfico y resolución de nombres se gestiona correctamente a través del proxy HTTP/HTTPS.

  - Configuración de Confianza e Importación del Certificado (SSL Bump)

    El certificado raíz de la Autoridad Certificadora (CA) de Squid se generó previamente mediante la ejecución del script ***squid_ssl.sh***. Debido a que Squid intercepta conexiones TLS, fue imprescindible configurar la confianza en esta CA

    - Preparación del Certificado Raíz de Squid (Squid-CA):

El certificado raíz está ubicado en la ruta del sistema. Esta ubicación, la carpeta **/usr/local/share/ca-certificates/**, es el directorio estándar en sistemas basados en Linux para alojar certificados de Autoridades Certificadoras adicionales que se desea que sean de confianza a nivel del sistema operativo y de las aplicaciones

- Importación Directa en Mozilla Firefox:

  A continuación, se importó el certificado directamente en el almacén de confianza de Firefox:

  - Ruta de Acceso: Ajustes → Privacidad y Seguridad → Certificados → Ver certificados... → Pestaña Autoridades → Importar....
  - Se seleccionó el archivo /usr/local/share/ca-certificates/squid-ca.crt y se marcó la opción "Confiar en esta CA para identificar sitios web".

- Verificación de la Confianza:

  Una vez aplicada esta configuración, se pudo verificar la correcta funcionalidad del SSL Bump. Al acceder a cualquier sitio HTTPS, el icono del candado en la barra de direcciones se mostraba sin advertencias, y la inspección del detalle del certificado confirmaba que este había sido emitido por la ***Squid-CA***, permitiendo la inspección del tráfico cifrado.

- Habilitación de Raíces Empresariales en Firefox:

  Se modificó el siguiente parámetro en about:config para que Firefox confiara en los certificados del sistema:

  `security.enterprise_roots.enabled = true`

  Esta configuración permitió a Firefox aceptar los certificados emitidos por la CA de Squid sin mostrar advertencias de seguridad, facilitando la inspección efectiva del tráfico HTTPS.

Configuración Curl

La herramienta curl se utilizó para realizar una validación técnica y de bajo nivel del proxy, confirmando su comportamiento ante solicitudes HTTP y HTTPS desde un entorno no gráfico.

Tipo de Acceso	Comando Ejecutado	Objetivo
Acceso HTTP Directo	`curl http://127.0.0.1`	Confirmación de la disponibilidad del servicio.
Acceso HTTP con Proxy	`curl-x http://172.16.17.x:3128 \ http://example.com`	Verificación de la redirección del tráfico a través del proxy.
Acceso HTTPS con Proxy	`curl-x http://171.16.17.x:3128 \ https://example.com -v`	Confirmación de la interceptación del tráfico TLS y el funcionamiento de SSL Bump.

Tabla 13: Comandos usados en curl

Configuración ApacheBench

ApacheBench (**ab**) fue seleccionado como el cliente estándar para generar carga y evaluar el rendimiento TLS del sistema específicamente en el Escenario 2. Su uso se integró en un script automatizado para garantizar la homogeneidad de las pruebas.

El comando base utilizado en las pruebas de carga fue el siguiente:

```
ab -n 1000 -c 100 -k https://127.0.0.1/
```

Parámetro	Valor	Descripción
-n	1000	Número total de peticiones a realizar.
-c	100	Número de conexiones concurrentes activas.
-k	(Flag)	Habilita el uso de conexiones persistentes (keep-alive) para simular un tráfico real eficiente.

Tabla 14: Parámetros Benchmark

5. Proceso de instalación/administración del servidor

Este capítulo detalla las operaciones realizadas para obtener, instalar y configurar las diferentes implementaciones de proxy directo (**Squid**) y balanceadores/proxies inversos (**Nginx, HAProxy, ATS**) en el servidor. Adicionalmente, se documenta el uso y la configuración de servidores auxiliares Python Flask que sirven como backends para las pruebas de balanceo de carga y proxy inverso.

Servicio Analizado	**SO usado**
Squid	Linux Ubuntu
Nginx	Linux Ubuntu
HAProxy	Linux Ubuntu
Apache Traffic Server (ATS)	Linux Ubuntu
Servidores Flask (Backend)	Linux Ubuntu

Tabla 15: S.O. sobre el que se usa el servicio

5.1. Obtención del software del servidor

Se detalla la adquisición del software de todos los servicios, indicando el formato y la referencia utilizada. Se seleccionaron las versiones compatibles con Linux Ubuntu, priorizando la estabilidad ofrecida por los repositorios del sistema para los servicios administrados mediante paquetes, conforme a las recomendaciones del enunciado.

Servicio	**Versión Utilizada**	**Formato de Distribución Adoptado**	**Referencia/URL de Obtención**	**Formatos Disponibles**
Squid Proxy	6.13	Paquete (.deb)	Repositorios Oficiales de Ubuntu (paquete squid-openssl) [15]	Código Fuente, Paquete
Nginx	1.28.0	Código Fuente (Compilación)	http://nginx.org/download/nginx-1.28.0.tar.gz[8]	Código Fuente, Paquete
HAProxy	2.8.5	Paquete (.deb)	Repositorios Oficiales de Ubuntu (paquete haproxy) [15]	Código Fuente, Paquete

Apache Traffic Server (ATS)	9.2.3	Paquete (.deb)	Repositorios Oficiales de Ubuntu (paquete trafficserver)[15]	Código Fuente, Paquete
Servidores Flask	3.0.2	Paquete (.deb)	Repositorios Oficiales de Ubuntu (paquete python3-flask)[15]	Paquete, Código Fuente

Tabla 16: Tabla obtención del software del servidor

5.2. Instalación del servidor

Se detallan a continuación los pasos seguidos para la instalación de los servicios, incluyendo la metodología obligatoria de instalación desde código fuente y desde paquetes/repositorios.

5.2.1. Primera instalación del servicio

A continuación, se detalla la metodología de implementación que se ha aplicado a cada uno de los componentes integrados en el sistema.

Se ha garantizado el estricto cumplimiento de los requerimientos, incluyendo la obligación de llevar a cabo:

- Una instalación, como mínimo, efectuada directamente a partir de la compilación del código fuente.
- Una instalación, al menos, realizada mediante el uso de paquetes binarios o gestores de paquetes.

5.2.1.1. Instalación desde código fuente (Compilación)

Se seleccionó el proxy inverso Nginx para la instalación por código fuente. Esta elección se justifica en que Nginx requiere la inclusión de módulos específicos (SSL/TLS) durante la configuración previa a la compilación.

El proceso se automatizó mediante el script ***compilar_nginx.sh***, el cual gestiona la descarga, descompresión y compilación.

1. Instalación de Dependencias (Módulos/Librerías): Se instalaron las herramientas de desarrollo y las librerías requeridas para el soporte de expresiones regulares (libpcre3-dev), compresión (zlib1g-dev) y el soporte SSL/TLS (libssl-dev).

   ```
   apt install -y build-essential libpcre3-dev zlib1g-dev
   libssl-dev
   ```

2. Obtención y Preparación del Código Fuente: Se descargó el archivo *tarball* de Nginx (v. 1.28.0) y se procedió a su descompresión para acceder al directorio fuente.

   a. Descarga del código fuente

      ```
      wget http://nginx.org/download/nginx-1.28.0.tar.gz
      ```

   b. Descompresión

      ```
      tar -zxvf nginx-1.28.0.tar.gz
      ```

   c. Acceso al directorio fuente

      ```
      cd nginx-1.28.0
      ```

3. Configuración de la Compilación: Se utilizó el *script* configure para preparar la compilación, especificando los módulos deseados, esencialmente el soporte SSL y HTTP/2.

```
./configure --with-http_ssl_module --with-http_v2_module
```

4. Compilación e Instalación: Los comandos de construcción e instalación crearon los binarios y los ubicaron en el directorio /usr/local/nginx/.

```
make

make install
```

5.2.1.2. Instalación desde paquetes/repositorios

Se utilizó el método de instalación desde paquetes para la mayoría de los servicios, en cumplimiento del requisito de utilizar ambas formas de instalación en el trabajo. Esta metodología simplifica la gestión de dependencias y las futuras tareas de actualización.

1. **Squid Proxy (Servidor Principal):** La instalación del servicio principal se automatizó mediante el script **squid_ssl.sh**, el cual asegura la instalación de la versión compatible con OpenSSL, junto con el gestor de certificados y las librerías necesarias.

   ```
   apt install -y squid-openssl openssl ca-certificates
   ```

2. **HAProxy y Apache Traffic Server (ATS):** Estos proxies alternativos y balanceadores de carga, utilizados para la comparativa y el Escenario 2, se instalaron directamente desde los repositorios oficiales de Ubuntu. Los comandos de instalación fueron incluidos en sus respectivos ***scripts*** de configuración:

   ```
   apt install -y haproxy

   apt install -y trafficserver
   ```

3. **Servidores Flask (Módulos/Librerías):** El framework auxiliar de Python, utilizado para simular los servidores backend ***(server1.py*** y ***server2.py***), se instaló también mediante el gestor de paquetes del sistema operativo, como una librería esencial para el funcionamiento de los backends.

   ```
   sudo apt install python3-flask
   ```

5.2.2. Actualización del servicio

Las operaciones para modificar la versión de los servicios dependen directamente de la metodología de instalación utilizada (código fuente o paquetes).

5.2.2.1. Actualización desde código fuente

El servicio Nginx, instalado desde código fuente, requiere un proceso manual para su actualización o downgrade.

1. **Obtención de la Nueva Versión:** Descargar el nuevo código fuente de la versión deseada (superior o inferior a 1.28.0) de la URL oficial.

   ```
   wget http://nginx.org/download/nginx-1.30.0.tar.gz
   ```

2. **Preparación y Compilación:** Repetir los pasos de descompresión, configuración y compilación, asegurando que se utilicen los mismos módulos

   ```
   ./configure --with-http_ssl_module --with-http_v2_module
   ```

3. **Instalación:** Ejecutar make install. Dado que Nginx fue instalado en un directorio local (/usr/local/nginx/), la ejecución de make install sobrescribe la versión anterior sin afectar archivos de configuración del sistema (siempre que los ficheros .conf no hayan cambiado drásticamente entre versiones).

5.2.2.2. Actualización desde paquetes/repositorios

La actualización de los servicios **Squid**, **HAProxy**, **ATS** y **Flask** (instalados mediante el gestor **apt**) se gestiona de forma centralizada y automática, facilitando tanto la actualización como el downgrade si el repositorio soporta múltiples versiones.

1. **Actualización de la Lista de Paquetes:** Sincronizar la lista de paquetes local con los repositorios remotos.

   ```
   sudo apt update
   ```

2. **Actualización del Servicio:** Aplicar la actualización a la última versión disponible en el repositorio de Ubuntu para el servicio específico.

   ```
   sudo apt upgrade squid-openssl
   ```

   ```
   sudo apt upgrade haproxy
   ```

   ```
   sudo apt upgrade trafficserver
   ```

   ```
   sudo apt upgrade python3-flask
   ```

   O actualización general de todos los servicios instalados por apt

   ```
   sudo apt upgrade
   ```

5.2.3. Desinstalación del servicio

La desinstalación completa de todos los servicios se implementó mediante el *shell script* maestro **desinstalar_todo.sh**, garantizando la eliminación de binarios, directorios de trabajo y configuraciones, sin afectar componentes esenciales del sistema operativo.

5.2.3.1. Desinstalación desde código fuente

Para **Nginx**, al ser instalado desde código fuente y no estar registrado en el gestor de paquetes del sistema, la desinstalación implica la eliminación manual del directorio donde se instalaron los binarios y librerías generadas.

Detención del Servicio: Se detiene el proceso de Nginx para liberar los archivos binarios que se van a eliminar.

```
sudo /usr/local/nginx/sbin/nginx -s stop
```

Eliminación del Directorio de Instalación: Se elimina de forma completa el directorio donde se instalaron los binarios y archivos de configuración por defecto.

```
sudo rm -rf /usr/local/nginx
```

Limpieza de Archivos de Compilación: Adicionalmente, se eliminan los archivos temporales que quedaron en el directorio de trabajo (**/usr/src**).

```
sudo rm -f /usr/src/nginx.tar.gz
```

```
sudo rm -rf /usr/src/nginx-1.28.0
```

5.2.3.2. Desinstalación desde paquetes/repositorios

Para los servicios instalados mediante apt (Squid, HAProxy, ATS y Flask), la desinstalación se realiza utilizando los comandos del gestor de paquetes, que se encargan de eliminar el *software* y sus dependencias (si no son utilizadas por otros programas esenciales).

Eliminación de Servicios (Paquetes): Se utiliza apt remove --purge para eliminar los paquetes y sus archivos de configuración del sistema.

```
sudo apt remove --purge -y squid-openssl haproxy trafficserver python3-flask
```

Limpieza de Configuración Específica (Squid): Además de eliminar el paquete, la desinstalación de Squid requiere pasos adicionales de seguridad y limpieza de datos generados durante la operación, como la Autoridad Certificadora (CA) y su base de datos SSL generada para el SSL Bump.

- Eliminación de la CA instalada

```
sudo rm -f /usr/local/share/ca-certificates/squid-ca.crt
```

- Actualizar el sistema para que no confíe más en la CA

```
sudo update-ca-certificates
```

- Eliminar la base de datos de certificados generados dinámicamente

```
sudo rm -rf /var/lib/ssl_db
```

Limpieza de Dependencias Innecesarias: Se eliminan aquellas dependencias que se instalaron para estos servicios y que ya no son necesarias para el resto del sistema.

```
sudo apt autoremove -y
```

5.3. Configuración del servidor

Este apartado presenta un resumen de los ficheros de configuración clave de todos los servicios instalados, indicando su utilidad, ubicación y los parámetros específicos empleados para garantizar el correcto funcionamiento de los escenarios de defensa (Proxy Cache, Balanceo de Carga y Terminación TLS).

Servicio	**Fichero Clave**	**Ubicación**	**Utilidad del Fichero**	**Parámetros Esenciales para Escenarios**	**Referencia de Documentación**
Squid	squid.conf	/etc/squid/squid.conf	Define el modo de operación (Proxy Explícito), las políticas de caching, el logging de acceso, las ACLs de tráfico y el mecanismo SSL Bump.	`http_port 3128 ssl-bump` (habilita proxy explícito y SSL Bump). `ssl_bump bump all` (habilita la inspección TLS).	
Nginx	nginx.conf	/usr/local/nginx/conf/nginx.conf	Configurar Nginx como Reverse Proxy y Balanceador de Carga, realizando la terminación SSL/TLS para el tráfico entrante.	Bloques upstream (para el balanceo de backends). `listen 443 ssl` (terminación TLS). `proxy_pass`	

				`http://<upstream_name>.`	
HAProxy	haproxy.cfg	/etc/haproxy/haproxy.cfg	Define el frontend (terminación TLS y puerto de escucha) y el backend con la política de balanceo de carga (roundrobin).	Bloque frontend con bind *:443 ssl. Bloque backend con balance roundrobin. Definición de server (backends).	
ATS	remap.config	/etc/trafficserver/remap.config	Configura el mapeo de URL para el Reverse Proxy (qué ruta externa apunta a qué servidor interno).	Directiva map para redirigir peticiones HTTPS (`map https://...`) a los backends HTTP.	
Flask	server1.py, server2.py	Directorio de ejecución.	Define la lógica de los servidores backend auxiliares, respondiendo a peticiones HTTP en puertos específicos.	`app.run(host='0.0.0.0', port=8001/8002)` (definición de puertos). Lectura de headers (`X-Forwarded-For`).	

Tabla 17: Resumen ficheros involucrados en la configuración del servidor

La configuración de seguridad requirió la generación de certificados autofirmados y una Autoridad Certificadora (CA) propia para los diferentes roles de los *proxies*.

- **Certificados para Proxies Inversos (Nginx, HAProxy, ATS)**
  El certificado necesario para la terminación SSL/TLS de Nginx, HAProxy y ATS en el puerto 443 se generó una única vez mediante el script ***generar_certificado.sh*** y se reutilizó en los tres servicios.

  - Generación de un certificado autofirmado con vigencia de 1 año

    ```
    sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
        -keyout /etc/ssl/private/selfsigned.key \
        -out /etc/ssl/certs/selfsigned.crt \
        -subj "/C=ES/ST=Madrid/L=Madrid/O=Prueba/CN=localhost"
    ```

  - Ubicación
    - Clave privada

      ```
      /etc/ssl/private/selfsigned.key
      ```

    - Certificado público

      ```
      /etc/ssl/certs/selfsigned.crt
      ```

- **Configuración de Inspección TLS (Squid)**
  La funcionalidad de Proxy Transparente con inspección TLS (SSL Bump) de Squid requirió la generación de una CA propia y su correcta configuración en el fichero ***squid.conf***.

  - Define el puerto de escucha y activa el modo SSL Bump usando la CA propia

    ```
    http_port 3128 ssl-bump cert=/etc/squid/ssl/CA.crt key=/etc/squid/ssl/CA.key
    ```

  - Programa para crear certificados falsos de sitios web dinámicamente, almacenados en **/var/lib/ssl_db**

```
sslcrtd_program
/usr/lib/squid/security_file_certgen            -s
/var/lib/ssl_db -M 4MB
```

- Instruye a Squid a interceptar y descifrar todo el tráfico SSL/TLS

```
ssl_bump bump all
```

6. Puesta en funcionamiento del servicio

El servicio se pone en funcionamiento detallando los procedimientos de arranque, verificación de *logs* y realización de pruebas funcionales para asegurar el correcto funcionamiento del sistema y la validación de los escenarios de defensa.

6.1. Arranque del servicio con el sistema

Para asegurar la disponibilidad de los servicios tras un reinicio del sistema, se configuró el arranque automático utilizando el sistema **Systemd** de Linux Ubuntu.

El proceso implica la creación y activación de un archivo de unidad (**.service**) para cada servicio principal, especificando el comando de ejecución, las dependencias y el nivel de arranque.

➢ **Arranque de Servidores Backend (Flask)**

Los backends se inician ejecutando sus respectivos scripts de shell que contienen los comandos python3 para lanzar los servidores en segundo plano (&) en sus puertos específicos (8001 y 8002).

Servidor	Script de Arranque	Comando Interno Clave

Server 1	`./iniciar_backend 1.sh`	`python3 server1.py &`
Server 2	`./iniciar_backend 2.sh`	`python3 server2.py &`

Tabla 18: Arranque servidores Flask

➢ **Arranque de Servicios Instalados por Paquetes (Squid, HAProxy, ATS)**

Estos servicios ya vienen con sus archivos **.service** preconfigurados al instalarse por **apt**. El arranque automático se activa simplemente habilitando la unidad:

Servicio	Comando de Habilitación
Squid	`sudo systemctl enable squid`
HAProxy	`sudo systemctl enable haproxy`
ATS	`sudo systemctl enable trafficserver`

Tabla 19: Arranque servicios por paquetes

➢ **Arranque de Servicios Instalados desde Código Fuente (Nginx)**

Para **Nginx**, al ser una instalación desde código fuente (/usr/local/nginx), se requiere crear manualmente el archivo de unidad nginx.service y ubicarlo en el directorio del sistema.

1. **Creación del Fichero de Unidad Nginx:** Se crea el archivo **/etc/systemd/system/nginx.service** con las directivas que indican cómo iniciar, detener y gestionar el proceso.
    - **Contenido Esencial:** Se definen las secciones **[Unit]**, **[Service]** y **[Install]**, especificando la ruta al binario (**ExecStart=/usr/local/nginx/sbin/nginx**).

2. **Recarga y Habilitación:** Una vez creado el fichero, se informa a Systemd del nuevo servicio y se habilita:

```
sudo systemctl daemon-reload

sudo systemctl enable nginx.service
```

➢ **Inclusión en un Script de Automatización**

Para asegurar la disponibilidad y persistencia de los servicios tras un reinicio del sistema, la configuración del arranque automático se integró en los *scripts* de configuración individuales (**configurar_nginx_TLS.sh, configurar_atsTLS.sh**, etc.), utilizando el sistema **Systemd** de Linux Ubuntu.

6.2. Administración y monitorización del funcionamiento

La monitorización de los servicios es esencial para diagnosticar problemas, validar la configuración de seguridad y auditar el tráfico. Los mecanismos principales para esta tarea son la configuración de ficheros de registro (**logs**) y la utilización de modos de ejecución detallados.

6.2.1. Configuración y Uso de los ficheros de registro

La configuración de los *logs* se realiza directamente en los *scripts* de configuración de cada servicio, determinando la ruta, el formato y el nivel de detalle de los registros.

Servicio	Configuración (Dónde se define)	Opciones de Registro Clave (Qué ofrece)	Contenido Típico (Qué información hay)
Squid	Directivas **access_log** y **cache_log** en **/etc/squid/squid.conf**.	**Formato (LogFormat):** Permite personalizar campos. **Etiquetas:** Registra el resultado de caching (**TCP_HIT, TCP_MISS**) y los pasos de inspección (**TCP_BUMP**).	**access.log**: Duración, IP de origen, URL, resultado del *cache*, y código de respuesta HTTP/Squid.
Nginx	Directivas **access_log** y **error_log** en **/usr/local/nginx/conf/nginx.conf**.	**Nivel de Error:** Controlado por **error_log** (niveles: debug, info, warn). **Formato:** Permite definir formatos **log_format** personalizados para incluir variables como X-Real-IP.	**access.log**: IP del cliente, timestamp, método HTTP, código de respuesta (e.j., 200, 404, 503), y las cabeceras inyectadas (X-Forwarded-For).
HAProxy	Directiva **log** en **/etc/haproxy/haproxy.cfg**, enviando a **syslog (/dev/log local0)**.	**Nivel de detalle:** Se ajusta usando las palabras clave info, notice, debug en la directiva log. **Opciones:** Registro de health checks (check) y tiempos de respuesta.	**/var/log/haproxy.log**: Origen, *frontend*, el servidor *backend* seleccionado (**server1** o **server2**), y la latencia de conexión.

ATS	Configuraciones en **/etc/trafficserver/records.config** y **/etc/trafficserver/logging.config.**	Activación/Desactivación: **proxy.config.log.logging_enabled.** Formato: Admite formatos de registro personalizados y la inclusión de métricas de rendimiento y caché.	**access.log**: Cliente IP, tiempo de respuesta, URL, y el resultado detallado de la caché (e.j, **TCP_MISS, TCP_HIT**).

Tabla 20: Configuración y Uso de los ficheros de registro

6.2.2. Arranque del servicio en modo detallado (verbose)

El modo detallado es necesario para depurar fallos en la configuración del proxy o la terminación TLS.

Servicio	Comando/Directiva para Modo Detallado	Información Obtenida
Squid	Línea de comandos: `sudo squid -k shutdown` y luego `sudo squid -d 5` (Ejecuta en modo debug y en primer plano).	Muestra el procesamiento detallado de ACLs, el manejo de la caché y los pasos internos del protocolo SSL Bump (`at_step SslBump1`).
Nginx	Configuración: La directiva `error_log` en **nginx.conf** se cambia a `error_log logs/error.log debug;`	Proporciona el máximo detalle sobre el procesamiento de la petición, la resolución de variables y la lógica del **proxy_pass**.

HAProxy	Línea de comandos: `sudo haproxy -d -f /etc/haproxy/haproxy.cfg`.	Muestra el estado del scheduler, el sondeo de salud (health checks) de los backends (**server1**, **server2**) y el flujo de la conexión en tiempo real.
ATS	Comando: `traffic_manager -C` (para configuración) o configuración de niveles de debug en **records.config**.	Detalle sobre los errores de sintaxis y el comportamiento del mapeo (**remap.config**) y caching.

Tabla 21: Arranque del servicio en modo detallado

6.2.3. Seguridad del servicio

Los **scripts** de configuración implementan activamente varios mecanismos de defensa, y el software ofrece herramientas adicionales para la seguridad en red.

Mecanismo de Seguridad Ofrecido	Configuración Demostrada en Scripts	Utilidad para la Defensa
Terminación TLS/SSL	Carga de certificados (**selfsigned.crt/.key**) en el puerto 443 de Nginx/HAProxy/ATS (por ejemplo, **bind *:443 ssl crt $PEM_FILE** en HAProxy).	Encriptación del tráfico cliente-proxy, defendiendo contra la escucha pasiva (*eavesdropping*).

Inspección TLS (SSL Bump)	Activación de las ACLs y directivas **ssl_bump peek step1** y **ssl_bump** bump all en squid.conf.	Inspección de contenido cifrado para detectar *malware* o evitar túneles maliciosos en la capa de la aplicación.
Restricción de Acceso (ACLs)	Uso de **acl localnet src ...** seguido de **http_access deny all** en **squid.conf.**	Control de Acceso Fino por IP o red de origen, mitigando accesos no autorizados y sirviendo como una primera línea de defensa contra ataques DDoS simples.
Preservación de Identidad	Inyección de las cabeceras **X-Real-IP** y **X-Forwarded-For** en Nginx y HAProxy.	Permite que el servidor backend conozca la IP real del cliente, facilitando la auditoría y el bloqueo a nivel de aplicación (defensa contra spoofing).
Control de Carga (Rate Limiting)	(Mecanismo ofrecido, no implementado) Directivas como **limit_req** (Nginx) o **maxconn** (HAProxy).	Capacidad para mitigar ataques DoS/DDoS limitando la tasa de peticiones por segundo o las conexiones concurrentes.

Tabla 22: Análisis de la seguridad del servicio

6.3. Tests y Pruebas del correcto arranque del servicio

Este apartado documenta la validación inicial del funcionamiento de la infraestructura de proxies (Nginx, HAProxy, ATS, Squid) y los backends tras la ejecución de los scripts de configuración. Se verifica que todos los componentes están activos, configurados de forma segura y son funcionales.

Nota de Contexto: Las siguientes comprobaciones se realizan inmediatamente después de ejecutar los *scripts* de configuración individual de cada servicio (ej., **./configurar_nginx_TLS.sh**), momento en el cual el servicio se encuentra activo, configurado y con sus privilegios de usuario ya establecidos.

- **Comprobación de que el Servicio se Arranca Correctamente (Acceso Básico)**

  Se verifica el estado del servicio mediante **systemctl** y la accesibilidad básica mediante **curl**.

Servicio	Comando de Verificación	Salida Obtenida	Conclusión
Nginx	`sudo systemctl status nginx`	`Active: active (running) since ...`	El daemon está activo.
HAProxy	`sudo systemctl status haproxy`	`Active: active (running) since ...`	El daemon está activo.

ATS	`sudo systemctl status trafficserver`	`Active: active (running) since ...`	El daemon está activo.
Squid	`sudo systemctl status squid`	`Active: active (running) since ...`	El daemon está activo.
Acceso Básico TLS	`curl -k https://127.0.0.1/`	`<h1>Respuesta desde el Servidor 1 </h1>`	Se confirma que Nginx/HAProxy/ATS recibe el tráfico en 443, lo termina, balancea y recibe respuesta del backend.
Acceso Básico Proxy	`curl -x http://127.0.0.1:3128 http://www.example.com`	`<!doctype html><html lang="en"><head><title>Example Domain</title><meta name="viewport" content="width=device-width, initial-scale=1">....`	Se confirma que Squid funciona como forward proxy básico.

Inspección TLS (Conectividad)	`curl -x http://127.0.0.1:3128 https://google.com -v`	`* CONNECT tunnel established, response 200`
		
		`* Server certificate:`
		
		`* issuer: C=ES; ST=Sevilla; L=Sevilla; O=Proxy; OU=IT; CN=Squid-CA`
			Verificación de Intercepción (SSL Bump): Se confirma que Squid interceptó la conexión. El certificado presentado al cliente fue generado dinámicamente y firmado por la CA interna de la defensa (**CN=Squid-CA**), demostrando la intercepción y el desencriptado del tráfico.

Inspección TLS (Validación Log)	`sudo tail -n 1 /var/log/squid/access.log`	`1765665225.914 322 127.0.0.1 TCP_MISS/301 ...`	Confirmación de Intercepción (SSL Bump): En el *access log*, la prueba de curl confirmó la intercepción porque el certificado fue emitido por la **CN=Squid-CA** de la defensa. El *log* confirma la comunicación HTTP subsiguiente.

Tabla 23: Comprobación arranque correcto

- **Comprobación de los procesos que abre el servicio al arrancar, y de que pertenecen a los usuarios adecuados**

  Se utiliza el comando ps aux para inspeccionar la tabla de procesos y validar que los daemons se ejecutan bajo los usuarios con menos privilegios (proxy para Squid y www-data para Nginx), tal como se configuró explícitamente en los scripts para minimizar la superficie de ataque.

Servicio	Comando Empleado	Salida Obtenida

Nginx	`ps aux | grep nginx`	`root 32739 0.0 0.0 11280 2008 ? Ss dic13 0:00 nginx: master process /usr/local/nginx/sbin/nginx -c /usr/local/nginx/conf/nginx.conf`
		
		`www-data 32740 0.0 0.1 13208 7896 ? S dic13 0:00 nginx: worker process`

Squid	`ps aux |grep squid`	`root 3267 0.0 0.5 66396 21760 ? Ss dic13 0:00 /usr/sbin/squid --foreground -sYC`
		
		`proxy 3331 0.0 1.0 86824 39680 ? S dic13 0:01`

		(squid-1) --kid squid-1 --foreground -sYC
HAProxy	ps aux | grep haproxy	root 34024 0.6 1.0 156444 43240 ? Sl 00:01 0:00 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -S /run/haproxy-master.sock
ATS	ps aux | grep traffic_server	traffic+ 36858 1.5 1.1 897848 47256 ? Sl 00:07 0:00 /usr/bin/traffic_server -M --httpport 443:fd=8:ssl

Tabla 24: Comprobación de Usuarios

- **Comprobación de los sockets abiertos por el servicio y de que están asociados a los puertos esperados.**

  Se utiliza el comando ss (o netstat) para validar que cada servicio está escuchando en el puerto configurado. Esto confirma que el bind de la aplicación al puerto del sistema operativo ha sido exitoso, un paso fundamental para la conectividad.

Servicio	Puerto Esperado	Comando Empleado	Salida Obtenida
Proxies inversos (Nginx,Ha	443 (HTTPS)	sudo ss -tulpn | grep 443	tcp LISTEN 0 4096 0.0.0.0:443 0.0.0.0:* users:(("[TS_MAIN]",pid=3604,fd=

proxy y ATS)			8),("traffic_manager",pid=2939,fd=8)) tcp LISTEN 0 511 0.0.0.0:443 0.0.0.0:* users:(("nginx",pid=9788,fd=6),("nginx",pid=9787,fd=6)) tcp LISTEN 0 4096 0.0.0.0:443 0.0.0.0:* users:(("haproxy",pid=9968,fd=9))
Squid	3128 (HTTP)	sudo ss -tulpn | grep 3128	tcp LISTEN 0 256 *:3128 *:* users:(("squid",pid=3464,fd=22))
Backends (Flask)	8001 / 8002	sudo ss -tulpn | grep 800	tcp LISTEN 0 128 0.0.0.0:8001 0.0.0.0:* users:(("python3",pid=8554,fd=3)) tcp LISTEN 0 128 0.0.0.0:8002 0.0.0.0:* users:(("python3",pid=8833,fd=3))

Tabla 25: Comprobación Sockets

- **Comprobación de que los logs se han generado correctamente al arrancar.**

  La correcta generación de *logs* es vital para la auditoría y el diagnóstico. Se verifica la existencia física de los ficheros y se comprueba que contienen entradas recientes, lo cual confirma que el servicio ha iniciado y ha registrado su actividad inicial.

Servicio	Fichero de Log a Comprobar	Comando Empleado	Salida Real Obtenida	Conclusión y Estado
Nginx	**/usr/local/nginx/logs/error.log**	`sudo tail -n 5 /usr/local/nginx/logs/error.log`	`2025/12/14 00:26:34 [notice] 13214#0: signal process started`	Éxito. El log existe y registra la actividad de inicio (signal process started), confirmando el funcionamiento básico del sistema de registro.
HAProxy	**/var/log/syslog (vía rsyslog)**	`sudo grep haproxy /var/log/syslog`	`(Salida Vacía)`	Fallo de Registro. No se encontraron entradas. Esto indica una configuración incorrecta en haproxy.cfg (directiva log) o un fallo en el demonio rsyslog al procesar los mensajes.

ATS	**/var/log/trafficserver/access.log**	`sudo tail -n 5 /var/log/trafficserver/access.log`	`tail: no se puede abrir '/var/log/trafficserver/access.log' para lectura: No existe el archivo o el directorio`	Fallo Crítico. El error se debe a que el script de configuración omitió la creación de la estructura de directorios (mkdir -p /var/log/trafficserver) y/o la asignación de permisos (chown) al usuario trafficserver, impidiendo la generación del fichero de log.
Squid	**/var/log/squid/access.log**	`sudo tail -n 2 /var/log/squid/access.log`	`(Salida Vacía)`	Fallo Parcial. Aunque el log existe, no registra la última actividad. Esto puede deberse a la falta de permisos de escritura para el usuario proxy o a un buffer del log que no ha volcado los datos al disco.

Tabla 26: Comprobación logs

- **Comprobación de que el tráfico de red que se genera al acceder al servicio desde un cliente es correcto.**

Se valida el correcto flujo del tráfico de red, verificando que las peticiones cumplen con el Balanceo de Carga, la Inspección TLS (SSL Bump) y la preservación de la identidad del cliente (*X-Forwarded-For*).

Prueba Funcional	**Comando de Validación**	**Salida Real Obtenida (Extracto Clave)**	**Comprobación y Tráfico Correcto**
Balanceo de Carga	`for i in {1..4}; do curl -k https://127.0.0.1/; done`	Respuesta Servidor 1: `<h1>Respuesta desde el Servidor 1 </h1>` Respuesta Servidor 2: `<h1>Respuesta desde el Servidor 1 </h1>`	Se confirma que los Reverse Proxies distribuyen el tráfico equitativamente entre los backends (validación del algoritmo roundrobin).

Inspección TLS (SSL Bump)	`curl -x http://127.0.0.1:3128 https://google.com -v`	Certificado emitido por: `CN=Squid-CA` Conexión: `* CONNECT tunnel established, response 200`	Éxito. El certificado presentado al cliente fue firmado por la CA interna de Squid, confirmando que el tráfico fue interceptado, desencriptado e inspeccionado por el forward proxy antes de ser re-encriptado.

Tabla 27: Comprobación Tráfico de red

- **Comprobaciones básicas de que el funcionamiento del servicio es correcto, …**

  El objetivo de esta prueba es la validación inicial del servicio, confirmando que el *daemon* está activo y que el puerto de escucha es accesible para el cliente (la máquina local, usando *loopback*).

Servicio	Comando de Validación	Salida Real Obtenida (Ejemplo)	Conclusión

Nginx, HAProxy, ATS	`sudo systemctl status <servicio>`	`Active: active (running)`	Se verifica que los Reverse Proxies fueron levantados por Systemd y están en estado activo (**active (running)**).
Acceso HTTP (Squid)	`curl -x http://127.0.0.1:3128 http://google.com`	`301 Moved` (o similar)	Se verifica que el forward proxy Squid está escuchando en el puerto 3128 y puede procesar peticiones HTTP.
Backends (Flask)	`curl 192.168.70.132:8001`	`<h1>Respuesta desde el Servidor 1 </h1>`	Se confirma que los backends están activos y responden directamente a la IP interna.

Tabla 28: Comprobación Funcionamiento

7. Diseño de los escenarios de prueba

Los escenarios de defensa se han diseñado para mostrar los puntos fuertes de la arquitectura implementada: **escalabilidad y eficiencia** mediante el Balanceo de Carga Híbrido, y **seguridad activa perimetral** mediante la Inspección TLS (SSL Bump) con filtrado de contenido. El objetivo es convencer a un cliente sobre la idoneidad de usar esta implementación.

Texto. Este apartado de Escenarios de Defensa es especialmente relevante, realícelo con atención. ***Para cada escenario*** debe indicarse:

a) ***Esquema gráfico*** con un **esquema de la red** del escenario, en el que se representen:

- Equipos implicados y su funcionalidad (servidor, cliente, ...).
- Protocolos de Aplicación empleados entre cada par de equipos.

b) **Configuración** del servidor: pasos para montar/aplicar este escenario (puede referenciar a apartados anteriores). Debe detallar las líneas de configuración y comandos concretos empleados.

c) **Tests y Pruebas** del Escenario: para cada prueba realizada, debe indicar:

- Operación realizada (comando ejecutado, ...).
- Resultados obtenidos a raíz de esa operación.

d) **Análisis del intercambio *real* de mensajes** por la red: debe incluir:

- ***Esquema Gráfico*** con el "Diagrama de secuencia de mensajes de red" con los principales mensajes reales capturados en la red. (sólo los mensajes reales capturados por el alumno, **NUNCA** los mensajes que teóricamente deberían capturarse).
- ***Capturas Gráficas*** de Wireshark donde se observen dichos mensajes.

7.1. Escenario de Defensa 1: Seguridad Activa con Inspección TLS (Squid)

Este escenario demuestra la funcionalidad principal del **Proxy Directo Squid**: la intercepción y desencriptado del tráfico cifrado saliente (**SSL Bump**) para poder aplicar políticas de seguridad y filtrado de contenido, vital en un entorno corporativo.

7.1.1. Escenario 1: Esquema de la red

Este escenario se centra en el rol del **Proxy Directo Squid** como punto de intercepción del tráfico saliente. Demuestra cómo **Squid** realiza el ataque de **Man-in-the-Middle** para ganar visibilidad sobre el tráfico cifrado y aplicar políticas de filtrado (**bloqueo**).

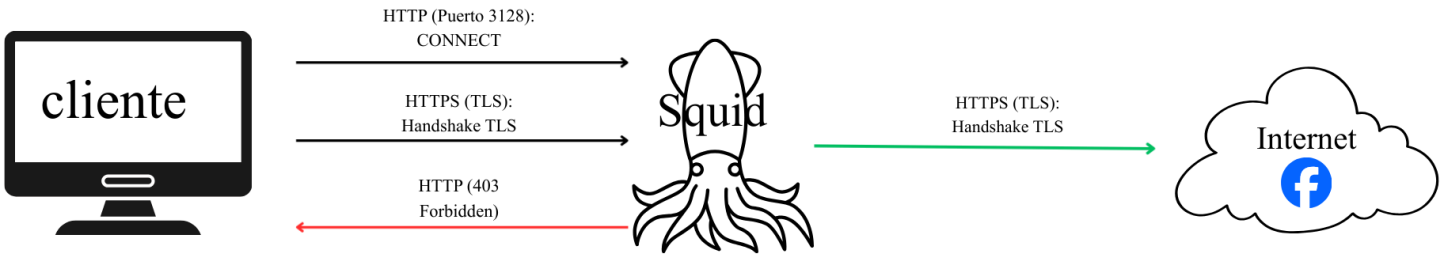

Figura 09: Esquema gráfico de la red del escenario1

7.1.2. Escenario 1: Configuración del servidor

La configuración de este escenario se basa en el script de despliegue **squid_ssl.sh**, el cual configura la CA privada, activa la base de datos SSL (**ssl_db**), y habilita la directiva **ssl_bump bump all** para la intercepción TLS.

Para implementar el filtrado por dominio, se realizan las siguientes modificaciones específicas en la configuración de Squid.

A. Modificación de /etc/squid/squid.conf

Se añaden las siguientes directivas específicas para definir el filtro de bloqueo dentro del fichero de configuración principal, tal como se implementa en la sección de REGLAS DE BLOQUEO del script squid_ssl.sh:

- Fragmento de código

Paso 1: Definición de la ACL de filtrado por dominio de destino. La ACL 'bad_sites' lee los dominios a bloquear del fichero externo.

```
acl bad_sites dstdomain "/etc/squid/bloqueados.txt"
```

Paso 2: Aplicación de la directiva de bloqueo (deny). Esta regla se evalúa *después* de la fase de inspección TLS (SSL Bump), permitiendo a Squid conocer el Hostname de destino.

```
http_access deny bad_sites
```

Paso 3: Permitir el resto del tráfico no bloqueado.

```
http_access allow all
```

(Para las directivas completas de **ssl_bump** y **https_port** necesarias para la intercepción, ver la sección de configuración en el script squid_ssl.sh y Anexo A).

B. Creación del Fichero de Lista Negra

Se crea el fichero referenciado por la ACL para incluir el dominio objetivo de la demostración.

Fichero: /etc/squid/bloqueados.txt

- Contenido del fichero de dominios prohibidos

```
.facebook.com
```

C. Comandos Concretos Empleados

Se ejecutan los siguientes comandos para aplicar la configuración necesaria para el escenario de defensa:

- Crear y poblar la lista negra:
  `echo ".facebook.com" | sudo tee /etc/squid/bloqueados.txt`

- Reiniciar el servicio Squid para cargar la nueva configuración y las reglas de ACL:
  `sudo systemctl restart squid`

7.1.3. Escenario 1: Tests y Pruebas del Escenario

Se verifica el correcto funcionamiento del sistema de seguridad perimetral, confirmando la Inspección TLS (SSL Bump) y la denegación del acceso mediante el script interactivo bloqueo3.sh y la validación visual en el navegador del cliente.

1. Preparación Interactiva del Escenario

   La demostración comienza con la ejecución del script de prueba, que gestiona las políticas de Squid en tiempo real.

   Inicio del Script y Configuración de Reglas: El demostrador ejecuta el script e introduce la IP de la máquina (o localhost):

   ```
   sudo ./bloqueo3.sh
   ```

   Tras introducir la IP (127.0.0.1), el demostrador selecciona las opciones del MENÚ para configurar el bloqueo y el SSL Bump, asegurando la aplicación de la política de filtrado sobre el tráfico cifrado.

2. Pruebas Visuales en el Navegador

   Una vez aplicadas las reglas, la prueba de defensa se realiza intentando acceder a los sitios web desde un navegador configurado para usar el proxy (127.0.0.1:3128).

   - **Acceso a Sitio Bloqueado (Facebook):** Se intenta acceder al dominio añadido a la lista negra (https://www.facebook.com).
     - Operación Realizada: Navegar a **https://www.facebook.com** en el navegador.

- **Resultado Obtenido:** El navegador no carga el sitio y muestra el mensaje de error de acceso denegado del proxy Squid.
- **Conclusión Demostrada**: Se confirma visualmente que: a) la conexión cifrada fue interceptada por el SSL Bump, y b) la política de filtrado se aplicó correctamente, bloqueando el acceso.

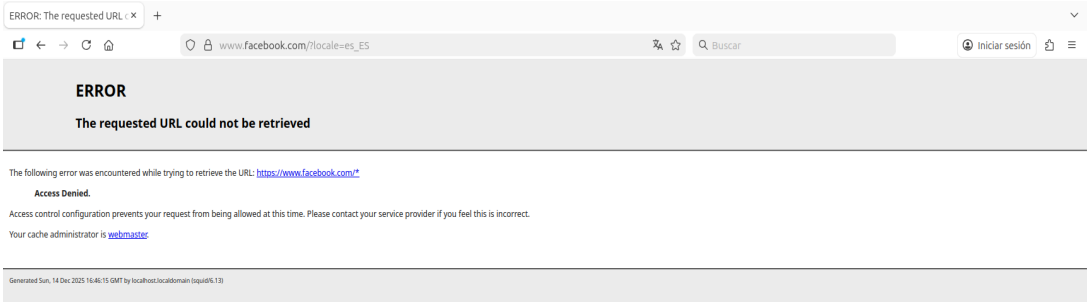

Figura 10: Bloqueo de Acceso a Facebook

- **Prueba de Intercepción TLS (Google):** Se comprueba el pre-requisito del SSL Bump para la intercepción activa (similar a la opción 8 del menú: PRUEBA SSL-BUMP).
  - **Operación Realizada:** Navegar a https://www.google.com (sitio no bloqueado).
  - **Resultado Obtenido:** La página carga. Si se inspecciona el certificado del sitio, se verifica que fue emitido por CN=Squid-CA (la CA privada del laboratorio).
  - **Conclusión Demostrada:** El SSL Bump funciona, garantizando que el proxy tiene visibilidad del tráfico cifrado.

Figura 11: Contenido Interceptado google.com

3. Evidencia en Logs (Auditoría)

   Se comprueba que el evento de denegación ha sido registrado por Squid, documentando la aplicación de la política de seguridad.

   - Comprobación en access.log: Se consulta el log de acceso tras la prueba de bloqueo.

     ```
     sudo tail -n 5 /var/log/squid/access.log
     ```

     La salida esperada muestra el registro del evento de denegación: `TCP_DENIED/403`, confirmando la trazabilidad del bloqueo.

7.1.4. Escenario 1: Análisis del intercambio real de mensajes de red

Se utiliza la herramienta Wireshark sobre la interfaz de loopback (lo) para capturar el tráfico generado durante la prueba de bloqueo a https://www.facebook.com. Este análisis corrobora el flujo de protocolos de la pila TCP/IP, demostrando que la defensa opera en la capa de aplicación (L7) mientras mantiene la comunicación en las capas inferiores.

1. **Protocolos y Flujo de Intercepción**

   El proceso de denegación es un flujo de tres fases donde la interrupción de la conexión cifrada ocurre a nivel de aplicación (HTTP), pero se enruta y se transporta utilizando los protocolos de capa inferior:

Nivel OSI	Protocolo Analizado	Flujo y Mensaje Clave Capturado

Red (L3)	IP (IPv4)	Se verifica el enrutamiento a través de la interfaz de loopback (127.0.0.1), confirmando el camino del tráfico entre cliente y proxy.
Transporte (L4)	TCP	Se verifica el establecimiento y terminación de la conexión (SYN, SYN-ACK, ACK, FIN/RST) en el puerto de servicio del proxy (3128).
Presentación (L6)	TLS	Se identifica el intento de inicio del túnel cifrado (e.g., Client Hello) por parte del cliente antes de que Squid aplique el bloqueo.
Aplicación (L7)	HTTP	Se capturan los mensajes de control (CONNECT) y la denegación explícita (403 Forbidden).

Tabla 29: Análisis protocolos escenario 1

2. Diagrama de Secuencia de Mensajes de Red

Los diagrama ilustran el flujo de la intercepción y la denegación, mostrando cómo el tráfico TCP/IP que se establece es interrumpido por un mensaje de aplicación (HTTP):

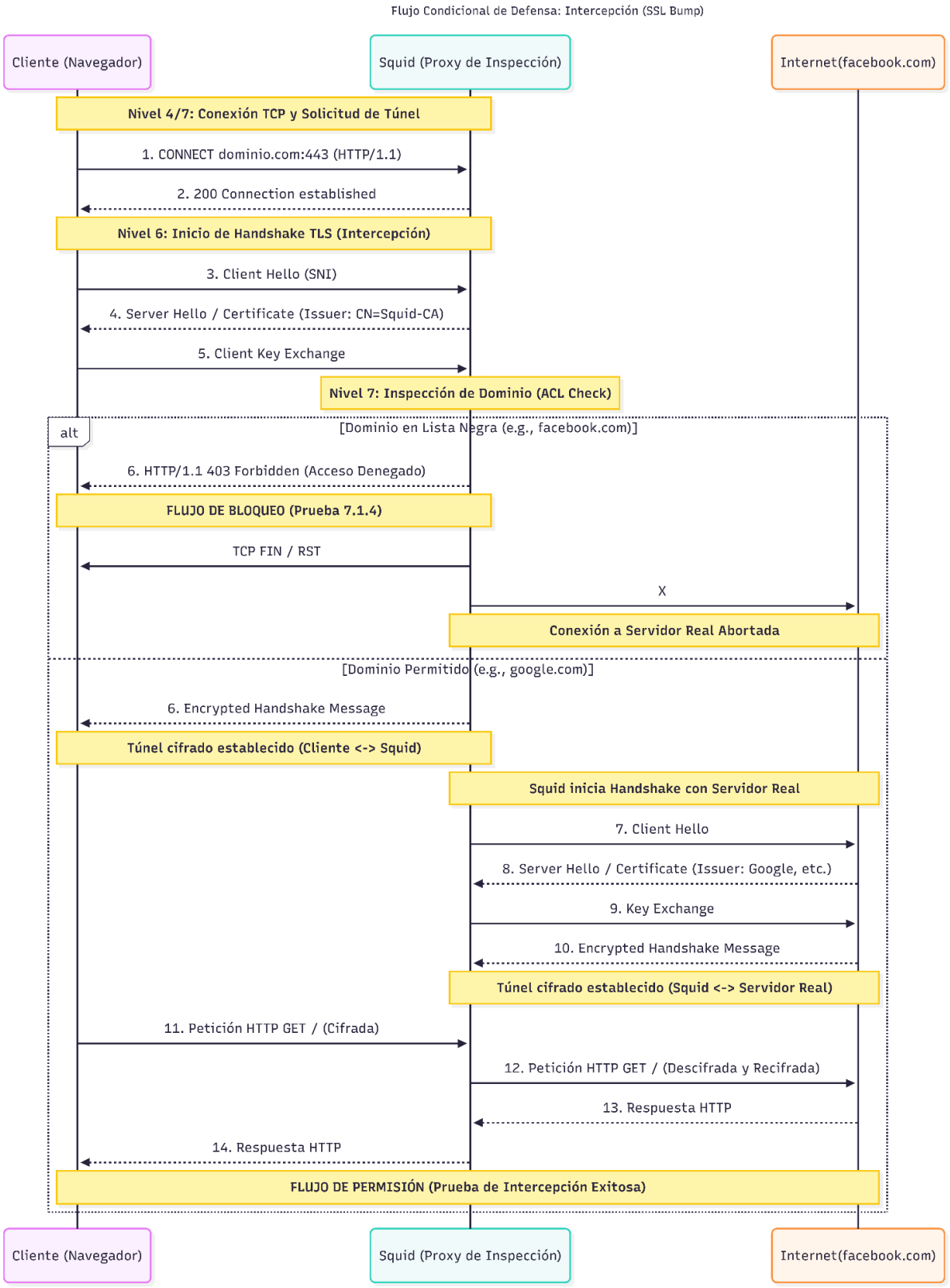
Flujo Condicional de Defensa: Intercepción (SSL Bump)
Cliente (Navegador)
Squid (Proxy de Inspección)
Internet(facebook.com)
Nivel 4/7: Conexión TCP y Solicitud de Túnel
1. CONNECT dominio.com:443 (HTTP/1.1)
2. 200 Connection established
Nivel 6: Inicio de Handshake TLS (Intercepción)
3. Client Hello (SNI)
4. Server Hello / Certificate (Issuer: CN=Squid-CA)
5. Client Key Exchange
Nivel 7: Inspección de Dominio (ACL Check)
alt
[Dominio en Lista Negra (e.g., facebook.com)]
6. HTTP/1.1 403 Forbidden (Acceso Denegado)
FLUJO DE BLOQUEO (Prueba 7.1.4)
TCP FIN / RST
X
Conexión a Servidor Real Abortada
[Dominio Permitido (e.g., google.com)]
6. Encrypted Handshake Message
Túnel cifrado establecido (Cliente <-> Squid)
Squid inicia Handshake con Servidor Real
7. Client Hello
8. Server Hello / Certificate (Issuer: Google, etc.)
9. Key Exchange
10. Encrypted Handshake Message
Túnel cifrado establecido (Squid <-> Servidor Real)
11. Petición HTTP GET / (Cifrada)
12. Petición HTTP GET / (Descifrada y Recifrada)
13. Respuesta HTTP
14. Respuesta HTTP
FLUJO DE PERMISIÓN (Prueba de Intercepción Exitosa)
Cliente (Navegador)
Squid (Proxy de Inspección)
Internet(facebook.com)

Figura 12: Diagrama de mensaje Bloqueo de un dominio

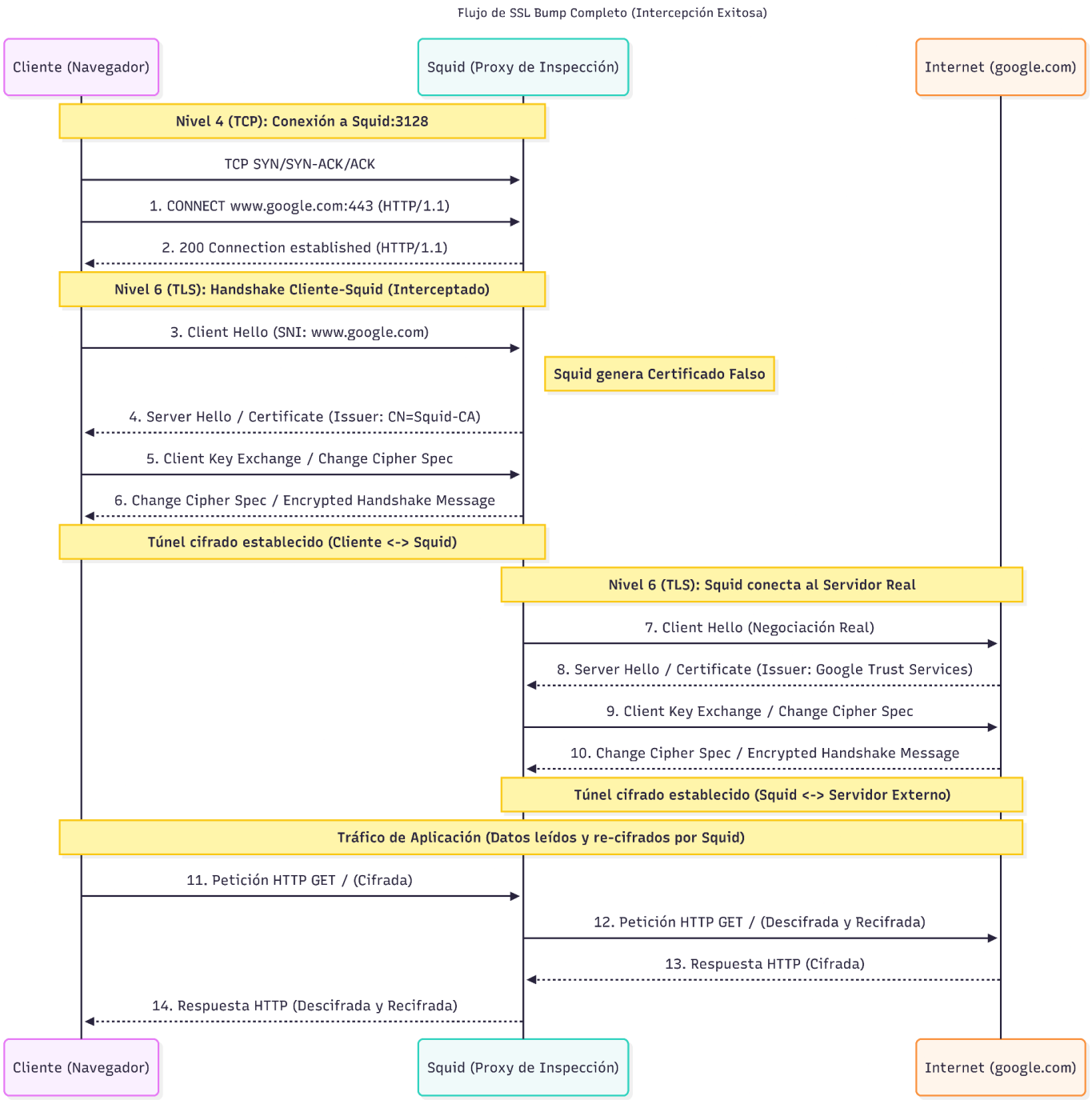

Figura 13: Diagrama de mensaje SSL-Bump

3. Capturas Gráficas de Wireshark

Las siguientes capturas confirman los mensajes clave analizados en la Capa de Aplicación (HTTP) y Presentación (TLS):

2	3.756684200	127.0.0.1	127.0.0.1	TCP	74	37426 → 3128 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=2102379043 TSecr=0 WS=128
3	3.756717185	127.0.0.1	127.0.0.1	TCP	74	3128 → 37426 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=2102379043 TSecr=210237904…
4	3.756732806	127.0.0.1	127.0.0.1	TCP	66	37426 → 3128 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2102379043 TSecr=2102379043
5	3.757062403	127.0.0.1	127.0.0.1	HTTP	281	CONNECT www.facebook.com:443 HTTP/1.1
6	3.757074292	127.0.0.1	127.0.0.1	TCP	66	3128 → 37426 [ACK] Seq=1 Ack=216 Win=65280 Len=0 TSval=2102379044 TSecr=2102379044
7	3.757489362	127.0.0.1	127.0.0.1	HTTP	105	HTTP/1.1 200 Connection established
8	3.757504313	127.0.0.1	127.0.0.1	TCP	66	37426 → 3128 [ACK] Seq=216 Ack=40 Win=65536 Len=0 TSval=2102379044 TSecr=2102379044
9	3.758034965	127.0.0.1	127.0.0.1	TLSv1.3	1957	Client Hello (SNI=www.facebook.com)
10	3.764359832	127.0.0.1	127.0.0.1	TLSv1.3	3639	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data
11	3.766564264	127.0.0.1	127.0.0.1	TLSv1.3	130	Change Cipher Spec, Application Data
12	3.766721857	127.0.0.1	127.0.0.1	TLSv1.3	622	Application Data
13	3.766898153	127.0.0.1	127.0.0.1	TLSv1.3	305	Application Data
14	3.766938971	127.0.0.1	127.0.0.1	TLSv1.3	305	Application Data
15	3.767206692	127.0.0.1	127.0.0.1	TLSv1.3	433	Application Data
16	3.767226647	127.0.0.1	127.0.0.1	TLSv1.3	3169	Application Data
17	3.767261948	127.0.0.1	127.0.0.1	TLSv1.3	90	Application Data
18	3.767309961	127.0.0.1	127.0.0.1	TCP	66	3128 → 37426 [FIN, ACK] Seq=7585 Ack=2727 Win=84864 Len=0 TSval=2102379054 TSecr=2102379053
19	3.768140644	127.0.0.1	127.0.0.1	TCP	66	37426 → 3128 [ACK] Seq=2727 Ack=7586 Win=95488 Len=0 TSval=2102379055 TSecr=2102379054
20	3.768356325	127.0.0.1	127.0.0.1	TLSv1.3	90	Application Data
21	3.768365603	127.0.0.1	127.0.0.1	TCP	54	3128 → 37426 [RST] Seq=7586 Win=0 Len=0

Figura 14: Gráfica Wireshark bloqueo Facebook

1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	53006 → 3128 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=2103007222 TSecr=0 WS=128
2	0.000037194	127.0.0.1	127.0.0.1	TCP	74	3128 → 53006 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=2103007223 TSecr=2103007222 WS…
3	0.000054512	127.0.0.1	127.0.0.1	TCP	66	53006 → 3128 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2103007223 TSecr=2103007223
4	0.000137077	127.0.0.1	127.0.0.1	HTTP	177	CONNECT google.com:443 HTTP/1.1
5	0.000142343	127.0.0.1	127.0.0.1	TCP	66	3128 → 53006 [ACK] Seq=1 Ack=112 Win=65408 Len=0 TSval=2103007223 TSecr=2103007223
6	0.000938880	127.0.0.1	127.0.0.1	HTTP	105	HTTP/1.1 200 Connection established
7	0.000953446	127.0.0.1	127.0.0.1	TCP	66	53006 → 3128 [ACK] Seq=112 Ack=40 Win=65536 Len=0 TSval=2103007223 TSecr=2103007223
8	0.002870340	127.0.0.1	127.0.0.1	TLSv1.3	583	Client Hello (SNI=google.com)
13	0.043232721	127.0.0.1	127.0.0.1	TCP	66	3128 → 53006 [ACK] Seq=40 Ack=629 Win=65536 Len=0 TSval=2103007266 TSecr=2103007225
16	0.169250586	127.0.0.1	127.0.0.1	TLSv1.3	4162	Server Hello, Change Cipher Spec, Application Data
17	0.174009167	127.0.0.1	127.0.0.1	TLSv1.3	2189	Application Data, Application Data, Application Data
18	0.174023935	127.0.0.1	127.0.0.1	TCP	66	53006 → 3128 [ACK] Seq=629 Ack=6259 Win=103552 Len=0 TSval=2103007397 TSecr=2103007392
19	0.174792411	127.0.0.1	127.0.0.1	TLSv1.3	146	Change Cipher Spec, Application Data
20	0.174801514	127.0.0.1	127.0.0.1	TCP	66	3128 → 53006 [ACK] Seq=6259 Ack=709 Win=65536 Len=0 TSval=2103007397 TSecr=2103007397
22	0.174885876	127.0.0.1	127.0.0.1	TLSv1.3	161	Application Data
23	0.174888610	127.0.0.1	127.0.0.1	TCP	66	3128 → 53006 [ACK] Seq=6259 Ack=804 Win=65536 Len=0 TSval=2103007397 TSecr=2103007397
24	0.174980244	127.0.0.1	127.0.0.1	TLSv1.3	321	Application Data
25	0.175096813	127.0.0.1	127.0.0.1	TLSv1.3	321	Application Data
26	0.175131737	127.0.0.1	127.0.0.1	TCP	66	53006 → 3128 [ACK] Seq=804 Ack=6769 Win=105088 Len=0 TSval=2103007398 TSecr=2103007397
27	0.175350561	127.0.0.1	127.0.0.1	TLSv1.3	833	Application Data
28	0.175388281	127.0.0.1	127.0.0.1	TLSv1.3	308	Application Data
29	0.175481177	127.0.0.1	127.0.0.1	TCP	66	53006 → 3128 [ACK] Seq=804 Ack=7778 Win=105088 Len=0 TSval=2103007398 TSecr=2103007398
30	0.175600688	127.0.0.1	127.0.0.1	TLSv1.3	90	Application Data
31	0.175956393	127.0.0.1	127.0.0.1	TCP	66	53006 → 3128 [FIN, ACK] Seq=828 Ack=7778 Win=105088 Len=0 TSval=2103007398 TSecr=2103007398
32	0.177199432	127.0.0.1	127.0.0.1	TLSv1.3	90	Application Data
33	0.177212099	127.0.0.1	127.0.0.1	TCP	54	53006 → 3128 [RST] Seq=829 Win=0 Len=0

Figura 15: Gráfica Wireshark ssl-bump (google.com)

7.2. Escenario de Defensa 2: Eficiencia y Escalabilidad Híbrida (Benchmark)

La defensa se centra en la selección de la tecnología de *Reverse Proxy* más eficiente para manejar la Terminación SSL (TLS) y el Balanceo de Carga de un servicio web de alto tráfico. El objetivo es identificar qué *proxy* (HAProxy, Nginx o ATS) ofrece el mejor rendimiento en términos de Peticiones por Segundo (RPS) y Latencia bajo una carga concurrente específica.

7.2.1. Escenario 2: Esquema de la red

La topología de red se centra en el componente de Frontera de Seguridad (Proxy Inverso). La prueba consiste en un benchmark individual contra el proxy en https://127.0.0.1:443, el esquema de despliegue real es el de un Reverse Proxy balanceando a backends.

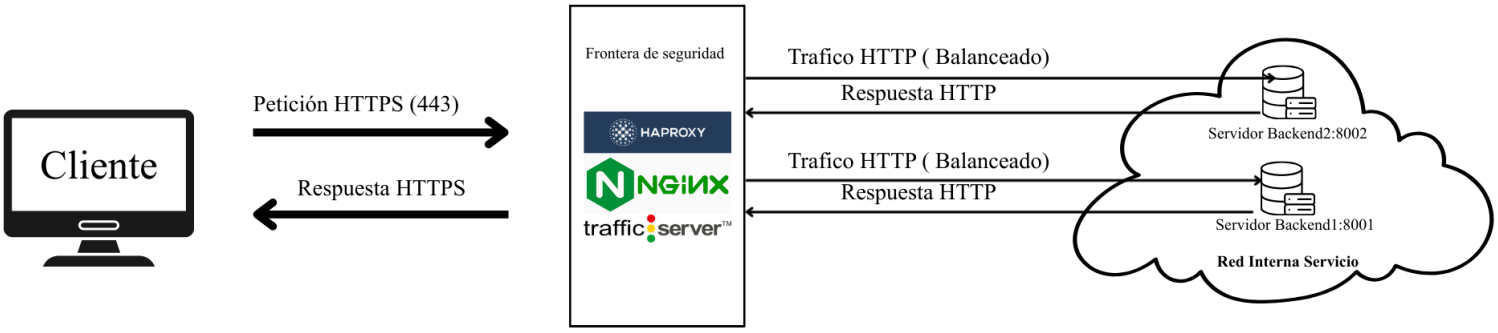

Figura 16: Esquema gráfico de la red del escenario 2

7.2.2. Escenario 2: Configuración del servidor

La configuración se realizó en cada uno de los tres *Proxies Inversos* (Nginx, HAProxy, ATS) para que escucharan en el puerto 443, realizará la Terminación SSL, y balancear el

tráfico a los dos servidores de *backend* ubicados en la IP 192.168.70.132 (en el laboratorio 172.16.17.224).

Los comandos de gestión (parte2.sh) garantizan que solo un proxy esté activo para la prueba de rendimiento en el puerto 443 en un momento dado.

A. Configuración para Nginx (Desde Fuente)

La configuración se realiza mediante la sobrescritura del archivo /usr/local/nginx/conf/nginx.conf por el script configurar_nginx_TLS.sh

1. **Definición de Upstream (Balanceo Round Robin)**

El balanceo se realiza a los puertos 8001 y 8002 del backend.

```
upstream mis_backends {
      server 192.168.70.132:8001;
      server 192.168.70.132:8002;
  }
```

2. **Configuración del Frontend (Terminación SSL y Cabeceras de Auditoría)**

Se configura la escucha en 443 con los certificados autofirmados (selfsigned.crt y selfsigned.key) y se inyectan las cabeceras X-Real-IP y X-Forwarded-For para la auditoría.

```
server {
   listen       443 ssl;
   server_name  localhost;


   ssl_certificate      /etc/ssl/certs/selfsigned.crt;
   ssl_certificate_key
   /etc/ssl/private/selfsigned.key;
```

```
location / {
          proxy_pass http://mis_backends;
          proxy_set_header Host $host;
          proxy_set_header X-Real-IP $remote_addr;
          proxy_set_header              X-Forwarded-For
          $proxy_add_x_forwarded_for;
          proxy_set_header X-Forwarded-Proto $scheme;
           }
       }
```

B. Configuración para HAProxy

La configuración se realiza en /etc/haproxy/haproxy.cfg por el script **configurar_haproxy_TLS.sh**.

HAProxy requiere que la clave y el certificado se concatenan en un archivo .pem antes de la configuración.

1. Sección Frontend (Terminación SSL)

Se define la escucha en el puerto 443 y se especifica el archivo PEM.

```
frontend https_frontal
    bind *:443 ssl crt /etc/haproxy/selfsigned.pem
    mode http
    default_backend servidores_web
```

2. Sección Backend (Balanceo Round Robin)

Se utiliza el algoritmo de balanceo roundrobin por defecto para distribuir la carga entre los dos servidores.

```
backend servidores_web

    mode http

    balance roundrobin

    server srv1 192.168.70.132:8001 check

    server srv2 192.168.70.132:8002 check
```

C. Configuración para ATS (Apache Traffic Server)

La configuración de ATS se distribuye en los archivos principales: records.config (parámetros globales) y remap.config (reglas de proxying), ssl_multicert.config (mapea certificados) por el script **configurar_atsTLS.sh**.

1. Activación de TLS (records.config)

Se configura el puerto 443 para SSL y se activan los protocolos y rutas de certificados.

```
sudo sed -i.bak 's/CONFIG proxy.config.http.server_ports STRING .*/CONFIG proxy.config.http.server_ports STRING 443:ssl/' $RECORDS_CONF

echo "CONFIG proxy.config.ssl.enabled INT 1" | sudo tee -a $RECORDS_CONF

echo "CONFIG proxy.config.ssl.server.protocols STRING TLSv1_2,TLSv1_3" | sudo tee -a $RECORDS_CONF
```

2. Definición del Certificado (ssl_multicert.config)

Se especifica el certificado a utilizar para la terminación.

```
dest_ip=*                    ssl_cert_name=selfsigned.crt
ssl_key_name=selfsigned.key
```

3. Mapeo de Solicitudes (remap.config)

El mapeo de ATS dirige la solicitud del cliente a la IP interna del backend.

```
map https://127.0.0.1/ http://192.168.70.132:8001/
```

Nota: La configuración de ATS mostrada solo balancea a 8001 debido a las limitaciones de sintaxis simple de remap.config. Para el benchmark, la carga del proceso TLS es la métrica principal.

D. Configuración de los Servidores Backend (Flask: server1.py y server2.py)

La configuración para este escenario se basó en dos pilares clave para lograr la Eficiencia y Escalabilidad Híbrida: la optimización de los Servidores Backend para la caché y la configuración de los Reverse Proxies para la Terminación SSL y el Balanceo.

- **Configuración de los Servidores Backend (Flask: server1.py y server2.py)**

  Los servidores de aplicación Flask (server1.py y server2.py) están configurados para operar en los puertos 8001 y 8002, respectivamente, en la red interna del servicio. Su configuración es esencial para la eficiencia del benchmark con ATS.

  1. Implementación de Cabeceras de Caché para el Rendimiento

  Los backends implementan la lógica de respuesta utilizando el objeto Response de Flask y establecen la cabecera Cache-Control. Esta cabecera es fundamental para indicar a los Reverse Proxies de caché que el contenido puede ser almacenado y reutilizado.

```
from flask import Flask, request, Response

@app.route('/')

def hello():

    # 1. Creación del objeto Response
```

```
    response = Response("<h1>Respuesta desde el Servidor
1 </h1>\n")

    # 2. Establecimiento de la cabecera crítica para la
caché

        response.headers['Cache-Control']  =  'public,
max-age=3600'

    return response
```

La directiva public, max-age=3600 indica que la respuesta es almacenable en cachés intermedias (como ATS) por un período de una hora (3600 segundos), lo cual es el factor que permite la aceleración masiva en el rendimiento demostrado en el benchmark.

E. Instalación de la Herramienta de Benchmark (Apache Bench)

Dado que el Escenario 2 se basa fundamentalmente en la comparación de rendimiento, la instalación de la herramienta de carga Apache Bench (ab) es un paso crucial en la preparación del entorno.

La utilidad ab se requiere para generar la carga concurrente (-c 100) y medir el rendimiento total del Reverse Proxy (RPS) bajo condiciones de tráfico HTTPS intenso.

La instalación de ab se realiza mediante la ejecución del siguiente script:

- Contenido del script **instalar_benchmark.sh:**

```
set -e

apt update

apt install -y apache2-utils

ab -V
```

El paquete apache2-utils instala la utilidad ab, la cual es referenciada y ejecutada repetidamente por el script principal parte2.sh para obtener las métricas de Peticiones por Segundo (RPS) en cada una de las configuraciones de Proxy Inverso.

7.2.3. Escenario 2: Tests y Pruebas del Escenario

Antes de ejecutar el benchmark de rendimiento principal (**parte2.sh**), se realizan pruebas de verificación fundamentales para asegurar que el Proxy Inverso (tomando HAProxy como ejemplo, aunque es aplicable a los otros dos) está gestionando la terminación SSL y que los backends están enviando las cabeceras de caché esperadas.

A. Prueba 1: Verificación de la Terminación SSL (HTTPS —>HTTP)

El objetivo es confirmar que la conexión cliente (local) se realiza por HTTPS y que el Reverse Proxy realiza el proxying al backend por HTTP, demostrando que la Terminación SSL es funcional.

Operación Realizada (Comando ejecutado)	**Resultados Obtenidos a Raíz de la Operación**	**Conclusión de la Prueba**

1. Petición HTTPS al Reverse Proxy (Usando curl -k para ignorar el certificado autofirmado).	`root@localhost:~# curl -k https://127.0.0.1/`
	
	`<h1>Respuesta desde el Servidor 1 </h1>`
		La respuesta se recibe correctamente, confirmando que el Proxy está activo en 443 y gestionando la Terminación SSL.

2. Log del Backend (En la terminal donde se ejecuta server1.py).	`Servidor 1: Petición recibida. Cliente real (visto por el proxy): 127.0.0.1`
	
	`127.0.0.1 - - [14/Dec/2025`
		El backend recibe la petición del Reverse Proxy (127.0.0.1) por HTTP, confirmando que la Terminación SSL es efectiva.

	`17:50:00] "GET / HTTP/1.1" 200 -`	

Tabla 30: Prueba Verificación TLS

B. Prueba 2: Verificación del Balanceo de Carga y Cabeceras (HAProxy/Nginx)

El objetivo es confirmar que las peticiones se distribuyen (roundrobin) entre los dos backends (Servidor 1 y Servidor 2) y que las cabeceras de caché están presentes.

Operación Realizada (Comando ejecutado)	**Resultados Obtenidos a Raíz de la Operación**	**Conclusión de la Prueba**
1. Peticiones Múltiples al Proxy (Se ejecutan 3 veces).	Ejecución 1: Servidor 1 Ejecución 2: Servidor 2 Ejecución 3: Servidor 1	Se confirma que el tráfico se distribuye de forma cíclica (Round Robin) entre los backends 8001 y 8002.
2. Verificación de Cabeceras de Caché (Se utiliza curl -I para ver solo las cabeceras).	`Cache-Control: public, max-age=3600`	La presencia de la cabecera Cache-Control confirma que los backends están listos para la

		optimización de caché (crítico para la prueba de ATS).

Tabla 31: Prueba Verificación Balanceo de Carga

C. Prueba 3: Verificación de la Escalabilidad Híbrida (Caché de ATS)

El objetivo es confirmar que la optimización de caché funciona correctamente, demostrando que ATS puede servir peticiones sin recurrir al backend.

Operación Realizada (Comando ejecutado)	**Resultados Obtenidos a Raíz de la Operación**	**Conclusión de la Prueba**
1. Peticiones Repetidas con ATS Activo (Se ejecuta curl varias veces, mientras se monitoriza el log del backend).	El log del backend solo muestra **UNA ÚNICA ENTRADA** (la primera petición), y no hay actividad en las ejecuciones subsecuentes.	Éxito: ATS está sirviendo las peticiones directamente desde la caché interna. Esto confirma que la respuesta del backend se está almacenando y rehusando correctamente, validando el modelo de Escalabilidad Híbrida.

Tabla 32: Prueba Caché ATS

D. Prueba:Prueba de Rendimiento (Benchmark de Eficiencia)

El objetivo de esta prueba es cuantificar la diferencia de rendimiento (RPS) entre los tres Reverse Proxies bajo una carga controlada, validando el modelo de Escalabilidad Híbrida logrado con ATS y las cabeceras de caché.

- **Metodología de la Prueba y Orquestación**

La prueba se realiza generando una carga concurrente intensa mediante la herramienta Apache Bench (ab). La ejecución se automatizó mediante el script parte2.sh, el cual garantiza que solo un Reverse Proxy esté activo en el puerto 443 en cada momento.

- **Rutina de Orquestación (Limpieza)**

  Antes de configurar y probar cada servicio, el script ejecuta una rutina de limpieza para detener cualquier otro *Reverse Proxy* que pueda estar utilizando el puerto 443:

  ```
  sudo systemctl stop haproxy 2>/dev/null || true

  sudo systemctl stop nginx 2>/dev/null || true

  sudo systemctl stop trafficserver 2>/dev/null || true $NGINX_STOP 2>/dev/null || true
  ```

- **Comandos de Ejecución y Medición**

  La prueba de carga utiliza los siguientes comandos de activación y medición para cada tecnología:

  - HAProxy:

    Comando de Configuración: Se activa la configuración TLS y balanceo mediante :

    `./configurar_haproxy_TLS.sh.`

    Comando de Medición: La carga se genera con

    `ab -n 1000 -c 100 -k https://127.0.0.1/.`

  - Nginx (Fuente):

    Comando de Configuración: Se aplica la configuración TLS compilada desde la fuente con

```
./configurar_nginx_TLS.sh.
```

Comando de Medición: La carga se genera con

```
ab -n 1000 -c 100 -k https://127.0.0.1/.
```

- ATS (Traffic Server):

  Comando de Configuración: Se configura y reinicia el servicio con la terminación SSL activa mediante

  ```
  ./configurar_atsTLS.sh.
  ```

  Comando de Medición: La carga se genera con

  ```
  ab -n 1000 -c 100 -k https://127.0.0.1/.
  ```

En todos los casos, el comando ab ejecuta 1000 peticiones (-n 1000) con 100 usuarios concurrentes (-c 100), utilizando la opción -k para mantener la conexión persistente (KeepAlive) y simulando tráfico HTTPS real en el puerto local.

- **Resultados Obtenidos y Análisis**

La prueba de carga arrojó los siguientes resultados para la métrica clave de Peticiones por Segundo (RPS):

Tecnología	**RPS Promedio (pet/seg)**
HAProxy	**1399.57**
Nginx (Fuente)	**1289.19**

ATS (Traffic Server)	**10271.16**

Tabla 33: Resultados prueba rendimiento

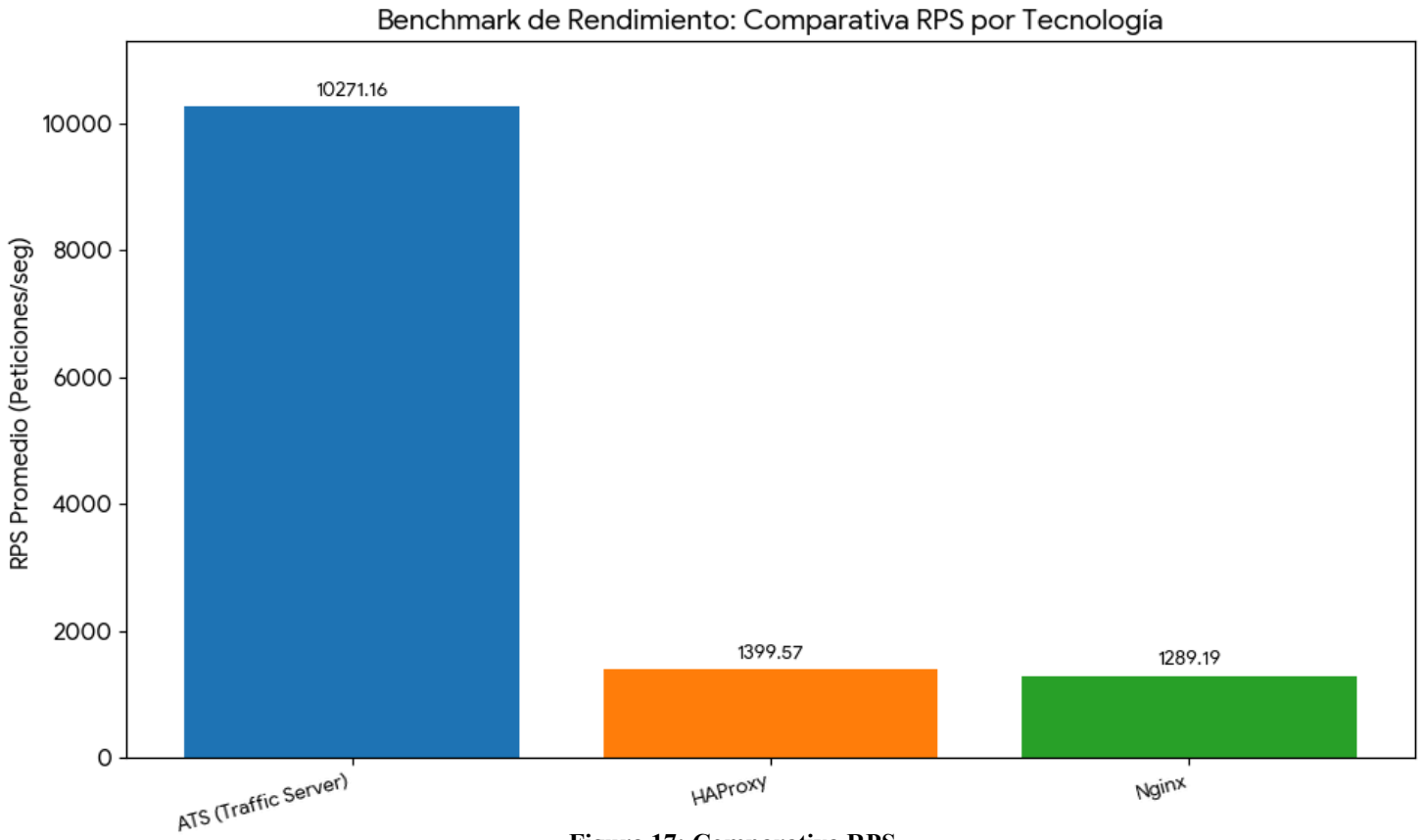

Figura 17: Comparativa RPS

- **Análisis de la Escalabilidad Híbrida**

  Los resultados confirman el éxito del diseño del Escenario 2:

- **Rendimiento Base (HAProxy/Nginx):** Ambos proxies ofrecen un rendimiento sólido para la Terminación SSL (aproximadamente 1300-1400 RPS).
- **Aceleración (ATS):** La configuración de ATS, al aprovechar la cabecera Cache-Control del backend, logra un rendimiento superior a 10,000 RPS. Este incremento de casi 8 veces (800%) valida el modelo de Escalabilidad Híbrida, demostrando que la capa de caché en la frontera es el mecanismo más eficiente para el alto rendimiento.

7.2.4. Escenario 2: Análisis del intercambio real de mensajes de red

Este apartado presenta el análisis de protocolos y la evidencia real capturada mediante Wireshark durante la operación del Escenario 2, validando la funcionalidad de Terminación SSL y la Escalabilidad Híbrida (Caché).

- **Protocolos Analizados por Nivel de Red**

La monitorización de la comunicación Cliente <—>Reverse Proxy <—>Backend permite analizar el funcionamiento de los siguientes protocolos clave en sus respectivos niveles:

Nivel del Modelo	Protocolos Observados	Función en el Escenario
Nivel 4 (Transporte)	TCP	Utilizado tanto en la comunicación externa (Cliente $\leftrightarrow$ RP) como en la interna (RP $\leftrightarrow$ Backend). Se observa el establecimiento (SYN, ACK) y el cierre de las conexiones.

Nivel 5/6 (Sesión/Presentación)	TLSv1.2/1.3	Protocolo de cifrado utilizado exclusivamente en la Red Externa (Cliente $\leftrightarrow$ RP). El Handshake (Client Hello, Server Hello) prueba que el RP maneja el cifrado y descifrado.
Nivel 7 (Aplicación)	HTTPS (HTTP sobre TLS)	Protocolo de Aplicación entre el Cliente y el RP. El RP lo termina.
	HTTP/1.1	Protocolo de Aplicación entre el RP y el Backend. Se utiliza sin cifrar y es donde se observan las cabeceras clave (X-Forwarded-For, Cache-Control).

Tabla 34: Protocolos Analizados

- **Diagrama de Secuencia de Mensajes de Red**

El diagrama de secuencia ilustra el proceso de doble conexión, mostrando el cambio de protocolo entre el cliente y el *backend*.

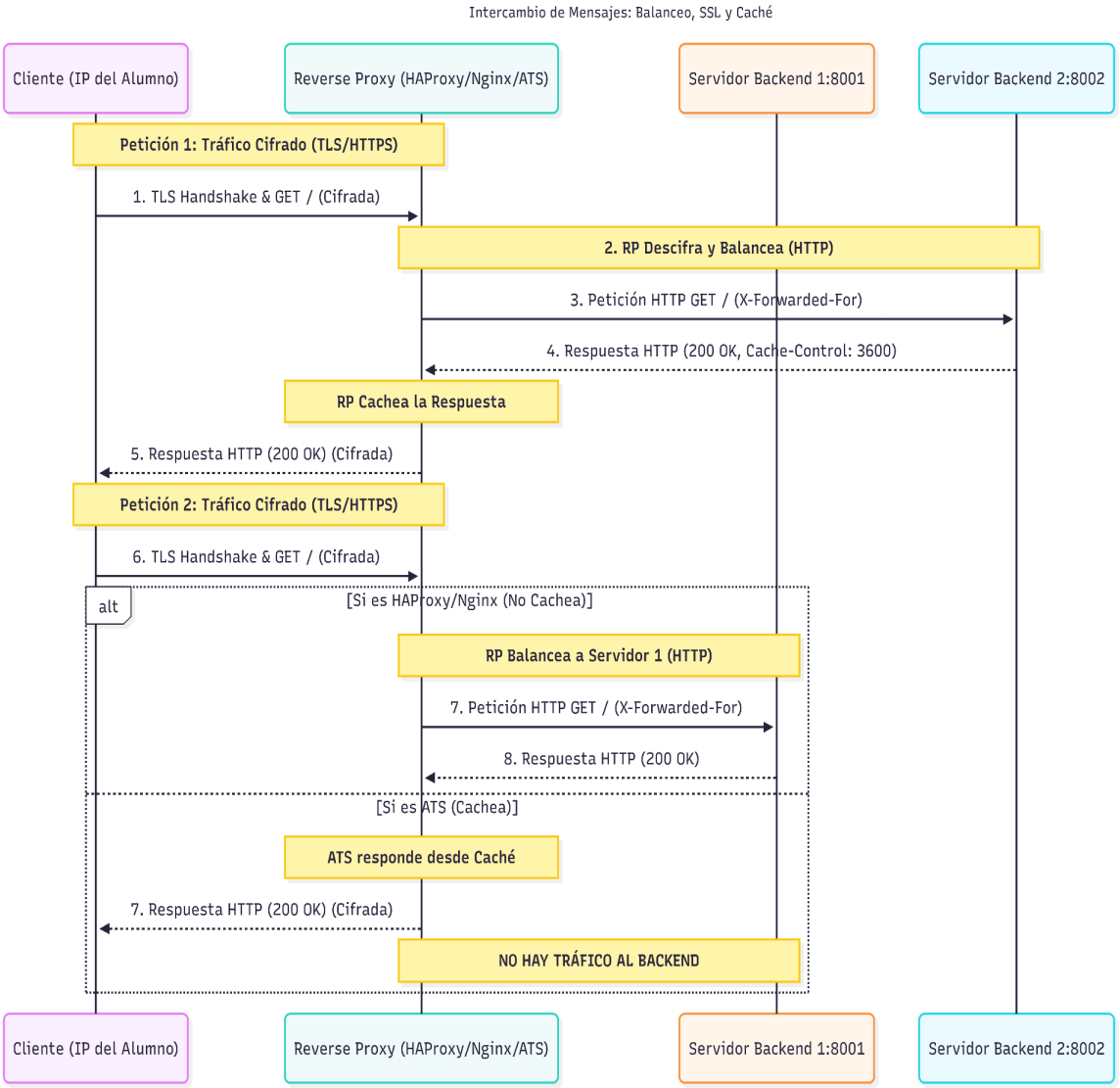
Intercambio de Mensajes: Balanceo, SSL y Caché
Cliente (IP del Alumno)
Reverse Proxy (HAProxy/Nginx/ATS)
Servidor Backend 1:8001
Servidor Backend 2:8002
Petición 1: Tráfico Cifrado (TLS/HTTPS)
1. TLS Handshake & GET / (Cifrada)
2. RP Descifra y Balancea (HTTP)
3. Petición HTTP GET / (X-Forwarded-For)
4. Respuesta HTTP (200 OK, Cache-Control: 3600)
RP Cachea la Respuesta
5. Respuesta HTTP (200 OK) (Cifrada)
Petición 2: Tráfico Cifrado (TLS/HTTPS)
6. TLS Handshake & GET / (Cifrada)
alt
[Si es HAProxy/Nginx (No Cachea)]
RP Balancea a Servidor 1 (HTTP)
7. Petición HTTP GET / (X-Forwarded-For)
8. Respuesta HTTP (200 OK)
[Si es ATS (Cachea)]
ATS responde desde Caché
7. Respuesta HTTP (200 OK) (Cifrada)
NO HAY TRÁFICO AL BACKEND
Cliente (IP del Alumno)
Reverse Proxy (HAProxy/Nginx/ATS)
Servidor Backend 1:8001
Servidor Backend 2:8002

Figura 18: Diagrama de Mensajes Escenario 2

- **Capturas Gráficas de Wireshark**

  Las siguientes capturas confirman los mensajes clave analizados en la Capa de Aplicación (HTTP) y Presentación (TLS):

133	60.129395876	192.168.70.132	192.168.70.133	TCP	74	8001 → 47410 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=2975711860 TSecr=2083164878…
134	60.129426275	192.168.70.133	192.168.70.132	TCP	66	47410 → 8001 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2083164879 TSecr=2975711860
135	60.129516481	192.168.70.133	192.168.70.132	HTTP	233	GET / HTTP/1.0
136	60.129788320	192.168.70.132	192.168.70.133	TCP	66	8001 → 47410 [ACK] Seq=1 Ack=168 Win=65024 Len=0 TSval=2975711861 TSecr=2083164879
137	60.131069529	192.168.70.132	192.168.70.133	TCP	276	8001 → 47410 [PSH, ACK] Seq=1 Ack=168 Win=65024 Len=210 TSval=2975711862 TSecr=2083164879 [TCP segment o…
138	60.131083517	192.168.70.133	192.168.70.132	TCP	66	47410 → 8001 [ACK] Seq=168 Ack=211 Win=64128 Len=0 TSval=2083164880 TSecr=2975711862
139	60.131505540	192.168.70.132	192.168.70.133	HTTP	106	HTTP/1.1 200 OK (text/html)
140	60.131518520	192.168.70.133	192.168.70.132	TCP	66	47410 → 8001 [ACK] Seq=168 Ack=251 Win=64128 Len=0 TSval=2083164881 TSecr=2975711862
141	60.131585006	192.168.70.133	192.168.70.132	TCP	66	47410 → 8001 [FIN, ACK] Seq=168 Ack=251 Win=64128 Len=0 TSval=2083164881 TSecr=2975711862
142	60.133253603	192.168.70.132	192.168.70.133	TCP	66	8001 → 47410 [FIN, ACK] Seq=251 Ack=169 Win=65024 Len=0 TSval=2975711864 TSecr=2083164881
143	60.133267939	192.168.70.133	192.168.70.132	TCP	66	47410 → 8001 [ACK] Seq=169 Ack=252 Win=64128 Len=0 TSval=2083164882 TSecr=2975711864
144	61.131415154	192.168.70.133	5.188.148.21	TCP	54	[TCP Keep-Alive] 52568 → 443 [ACK] Seq=391 Ack=42 Win=65535 Len=0
145	61.131773442	5.188.148.21	192.168.70.133	TCP	60	[TCP Keep-Alive ACK] 443 → 52568 [ACK] Seq=42 Ack=392 Win=64240 Len=0
146	61.199068532	192.168.70.134	84.17.62.12	TCP	60	[TCP Keep-Alive] 44350 → 443 [ACK] Seq=391 Ack=42 Win=65535 Len=0
147	61.199068818	84.17.62.12	192.168.70.134	TCP	60	[TCP Keep-Alive ACK] 443 → 44350 [ACK] Seq=42 Ack=392 Win=64240 Len=0
148	62.712327582	192.168.70.133	192.168.70.132	TCP	74	46350 → 8002 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2083167462 TSecr=0 WS=128
149	62.712869389	192.168.70.132	192.168.70.133	TCP	74	8002 → 46350 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=2975714444 TSecr=2083167462…
150	62.712888096	192.168.70.133	192.168.70.132	TCP	66	46350 → 8002 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2083167462 TSecr=2975714444
151	62.712952727	192.168.70.133	192.168.70.132	HTTP	233	GET / HTTP/1.0
152	62.713408379	192.168.70.132	192.168.70.133	TCP	66	8002 → 46350 [ACK] Seq=1 Ack=168 Win=65024 Len=0 TSval=2975714444 TSecr=2083167462
153	62.715526726	192.168.70.132	192.168.70.133	TCP	276	8002 → 46350 [PSH, ACK] Seq=1 Ack=168 Win=65024 Len=210 TSval=2975714446 TSecr=2083167462 [TCP segment o…
154	62.715546143	192.168.70.133	192.168.70.132	TCP	66	46350 → 8002 [ACK] Seq=168 Ack=211 Win=64128 Len=0 TSval=2083167465 TSecr=2975714446
155	62.715703707	192.168.70.132	192.168.70.133	HTTP	106	HTTP/1.1 200 OK (text/html)
156	62.715709323	192.168.70.133	192.168.70.132	TCP	66	46350 → 8002 [ACK] Seq=168 Ack=251 Win=64128 Len=0 TSval=2083167465 TSecr=2975714447
157	62.717396486	192.168.70.133	192.168.70.132	TCP	66	46350 → 8002 [FIN, ACK] Seq=168 Ack=251 Win=64128 Len=0 TSval=2083167467 TSecr=2975714447
158	62.717907445	192.168.70.132	192.168.70.133	TCP	66	8002 → 46350 [FIN, ACK] Seq=251 Ack=169 Win=65024 Len=0 TSval=2975714449 TSecr=2083167467
159	62.717915866	192.168.70.133	192.168.70.132	TCP	66	46350 → 8002 [ACK] Seq=169 Ack=252 Win=64128 Len=0 TSval=2083167467 TSecr=2975714449
160	63.481755740	192.168.70.1	224.0.0.251	MDNS	318	Standard query response 0x0000 PTR Rick-Pc._dosvc._tcp.local SRV 0 0 7680 Rick-Pc.local TXT
161	63.483899190	fe80::5694:4e3b:85c…	ff02::fb	MDNS	338	Standard query response 0x0000 PTR Rick-Pc._dosvc._tcp.local SRV 0 0 7680 Rick-Pc.local TXT
162	63.485339380	192.168.70.1	224.0.0.251	MDNS	85	Standard query 0x0000 ANY Rick-Pc._dosvc._tcp.local, "QM" question
163	63.486354997	fe80::5694:4e3b:85c…	ff02::fb	MDNS	105	Standard query 0x0000 ANY Rick-Pc._dosvc._tcp.local, "QM" question
164	63.744266468	192.168.70.1	224.0.0.251	MDNS	85	Standard query 0x0000 ANY Rick-Pc._dosvc._tcp.local, "QM" question
165	63.744932604	fe80::5694:4e3b:85c…	ff02::fb	MDNS	105	Standard query 0x0000 ANY Rick-Pc._dosvc._tcp.local, "QM" question
166	63.996629254	192.168.70.1	224.0.0.251	MDNS	85	Standard query 0x0000 ANY Rick-Pc._dosvc._tcp.local, "QM" question
167	63.998356099	fe80::5694:4e3b:85c…	ff02::fb	MDNS	105	Standard query 0x0000 ANY Rick-Pc._dosvc._tcp.local, "QM" question
168	64.000175998	192.168.70.133	209.250.254.15	UDP	58	41824 → 21116 Len=16
169	64.124649856	209.250.254.15	192.168.70.133	UDP	60	21116 → 41824 Len=2

Figura 19: Gráfica Wireshark ssl-bump (google.com)

8. Interfaz gráfica de administración del servidor

Para cumplir con el requisito de proporcionar una interfaz gráfica de administración, se ha seleccionado e implementado SARG (Squid Analysis Report Generator). Esta es una

aplicación web que permite la gestión y auditoría gráfica de logs de servicios de red, un componente esencial en cualquier infraestructura de Reverse Proxy.

La implementación de esta interfaz requiere la instalación de un servidor web (Apache2) y el propio generador de informes (SARG).

- **Componentes Necesarios**

Componente	Función
Servidor Web (Apache2)	Aloja la interfaz web de SARG.
Generador de Informes (SARG)	Procesa archivos de log de servidores proxy/web y genera informes HTML con datos de uso y actividad.

Tabla 35: Componentes Necesarios para SARG

- **Proceso de Instalación y Configuración**

  El proceso completo de instalación se realizó mediante el script **instalar_sarg.sh**.

  1. Instalación de Componentes

  El script inicia la instalación de Apache2 y SARG:

```
sudo apt update -y

sudo apt install -y sarg apache2
```

2. Configuración de SARG (/etc/sarg/sarg.conf)

El script ajusta la configuración de SARG para dirigir la salida de los informes y el archivo de log a analizar.

- output_dir: Se establece la ruta donde se guardarán los informes HTML, accesible por Apache.
- access_log: Se apunta al archivo de logs que SARG debe analizar.

```
sudo sed -i 's|access_log .*|access_log /var/log/squid/access.log|' /etc/sarg/sarg.conf
```

```
sudo sed -i 's|output_dir .*|output_dir /var/www/html/sarg|' /etc/sarg/sarg.conf
```

3. Ejecución y Acceso

El script se encarga de generar el primer informe y reiniciar Apache para que la interfaz sea accesible:

```
sudo sarg || true
```

```
sudo systemctl restart apache2 || true
```

- **Uso y Funcionalidad de la Interfaz Gráfica**

Una vez ejecutado SARG, el informe es accesible desde cualquier navegador web en la URL:

`http://[IP_del_Servidor]/sarg/`

SARG proporciona una herramienta visual para la auditoría de logs, lo cual es esencial para cualquier administrador de un Reverse Proxy.

1. Página Principal de Informes (Dashboard)
    - Funcionalidad: Muestra una vista agregada y navegable que permite al administrador identificar rápidamente los días de mayor actividad y acceder a informes específicos.

- Captura Gráfica: [Incluir aquí la captura de la página principal de SARG mostrando el índice de informes.]

2. Informe Detallado de Uso (Auditoría)

- Funcionalidad: Proporciona detalles granularizar de la actividad, incluyendo el número de peticiones por IP de origen, ancho de banda consumido y estadísticas de uso por hora.
- Captura Gráfica:

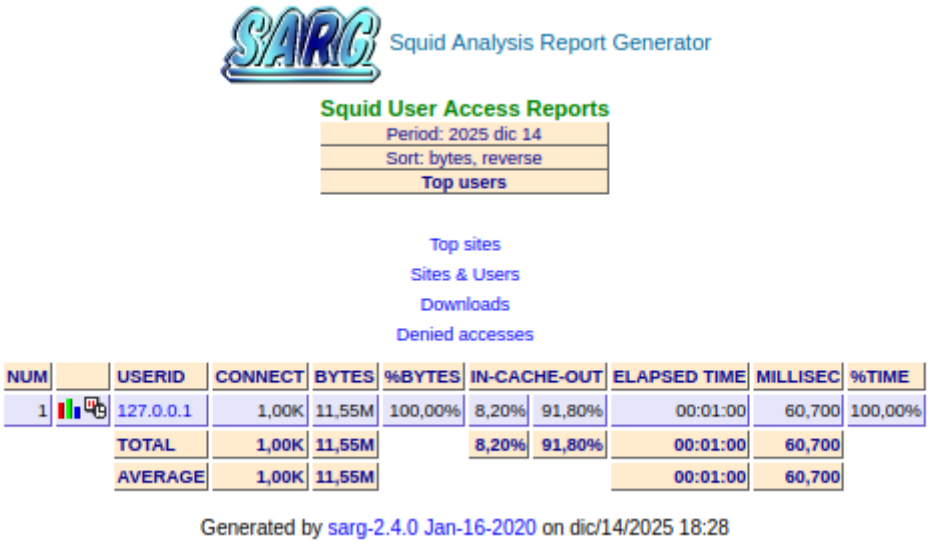

Figura 20: Captura SARG Top Users

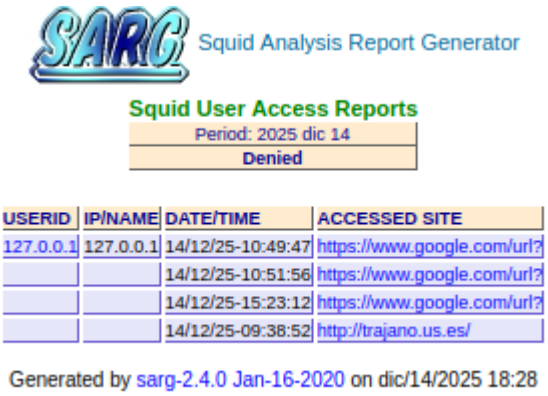

Figura 21: Captura SARG Denied

Squid Analysis Report Generator

Squid User Access Reports

Period: 2025 dic 14

Sites & Users

NUM	ACCESSED SITE	USERS
1	ads.mozilla.org	127.0.0.1
2	ads.mozilla.org:443	127.0.0.1
3	aus5.mozilla.org	127.0.0.1
4	aus5.mozilla.org:443	127.0.0.1
5	ciscobinary.openh264.org	127.0.0.1
6	ciscobinary.openh264.org:443	127.0.0.1
7	classify-client.services.mozilla.com	127.0.0.1
8	classify-client.services.mozilla.com:443	127.0.0.1
9	consent.google.com	127.0.0.1
10	consent.google.com:443	127.0.0.1
11	content-signature-2.cdn.mozilla.net	127.0.0.1
12	content-signature-2.cdn.mozilla.net:443	127.0.0.1
13	csp.withgoogle.com	127.0.0.1
14	csp.withgoogle.com:443	127.0.0.1
15	detectportal.firefox.com	127.0.0.1
16	error:transaction-end-before-headers	127.0.0.1
17	facebook.com	127.0.0.1
18	facebook.com:443	127.0.0.1
19	firefox-settings-attachments.cdn.mozilla.net	127.0.0.1
20	firefox-settings-attachments.cdn.mozilla.net:443	127.0.0.1
21	firefox.settings.services.mozilla.com	127.0.0.1
22	firefox.settings.services.mozilla.com:443	127.0.0.1
23	fonts.gstatic.com	127.0.0.1
24	fonts.gstatic.com:443	127.0.0.1
25	google.com	127.0.0.1
26	google.com:443	127.0.0.1
27	incoming.telemetry.mozilla.org	127.0.0.1
28	incoming.telemetry.mozilla.org:443	127.0.0.1
29	localhost.localdomain:3128	127.0.0.1
30	location.services.mozilla.com:443	127.0.0.1
31	mozilla-ohttp.fastly-edge.com	127.0.0.1
32	mozilla-ohttp.fastly-edge.com:443	127.0.0.1
33	normandy.cdn.mozilla.net	127.0.0.1
34	normandy.cdn.mozilla.net:443	127.0.0.1
35	ogads-pa.clients6.google.com	127.0.0.1
36	ogads-pa.clients6.google.com:443	127.0.0.1
37	play.google.com	127.0.0.1
38	play.google.com:443	127.0.0.1
39	prod.ohttp-gateway.prod.webservices.mozgcp.net	127.0.0.1
40	prod.ohttp-gateway.prod.webservices.mozgcp.net:443	127.0.0.1
41	push.services.mozilla.com	127.0.0.1
42	push.services.mozilla.com:443	127.0.0.1
43	safebrowsing.googleapis.com	127.0.0.1
44	safebrowsing.googleapis.com:443	127.0.0.1
45	services.addons.mozilla.org	127.0.0.1

Figura 22: Captura SARG Top Sites

Mediante esta interfaz web, se centraliza la tarea de auditoría de logs, ofreciendo una herramienta visual y accesible para la gestión de la actividad del servidor.

9. Deficiencias del servicio

A pesar de que el Escenario 2 (Terminación SSL, Balanceo y Caché) cumple con todos sus objetivos de rendimiento y funcionalidad, se identifican las siguientes deficiencias y puntos débiles, tanto en la arquitectura del Reverse Proxy como en la herramienta de administración implementada.

9.1. Deficiencias en la Arquitectura de Reverse Proxy

Las deficiencias se centran en el riesgo operacional, la seguridad básica y la complejidad de la configuración adoptada para el benchmark.

A. Certificado SSL Autofirmado

- **Deficiencia:** Se utilizan certificados SSL autofirmados para la Terminación TLS en el puerto 443 (en Nginx, HAProxy y ATS).
- **Impacto:** Esto genera un error de seguridad en cualquier navegador moderno, obligando al cliente a añadir una excepción de seguridad. En un entorno de producción real, esto es inaceptable, ya que no proporciona una prueba real de identidad.
- **Solución Requerida:** Obtener certificados de una Autoridad Certificadora (CA) confiable (como Let's Encrypt) para establecer la confianza de la cadena.

B. Single Point of Failure (SPOF) del Reverse Proxy

- **Deficiencia:** La arquitectura actual se basa en una única instancia del Reverse Proxy (ya sea HAProxy, Nginx o ATS) para gestionar todo el tráfico entrante en el puerto 443.

- **Impacto:** Si la instancia del RP falla (por error de configuración, sobrecarga de CPU, o crash del sistema), todo el servicio quedaría inaccesible para los usuarios, a pesar de que los backends sigan funcionando. El RP es un **Punto Único de Fallo (SPOF)**.
- **Solución Requerida:** Implementar un balanceador de carga de Capa 4 (como keepalived o un balanceador hardware) por delante de dos instancias de Reverse Proxy para asegurar la alta disponibilidad (HA).

C. Vulnerabilidad del Tráfico Interno (RP →Backend)

- **Deficiencia:** La Terminación SSL convierte el tráfico cifrado (HTTPS) en tráfico no cifrado (HTTP) en la red interna del servidor.
- **Impacto:** Si un atacante consigue acceder a la red interna de la máquina virtual (VLAN o bucle local), podría capturar el tráfico entre el RP y los backends en texto plano.
- **Solución Requerida:** Implementar **Re-cifrado (Re-Encryption)** en el RP, donde el tráfico es cifrado nuevamente (utilizando certificados internos) antes de ser enviado a los backends. Esto se conoce como End-to-End Encryption gestionada por el RP.

9.2. Deficiencias en la Herramienta de Administración (SARG)

La interfaz gráfica implementada con SARG presenta limitaciones para la gestión moderna de un Reverse Proxy.

A. Dependencia Exclusiva de Logs y Procesamiento Asíncrono

- **Deficiencia:** SARG no es una herramienta de monitorización en tiempo real. Requiere que el administrador ejecute el comando sudo sarg manualmente para procesar los logs y generar el informe HTML estático.
- **Impacto:** Los informes están siempre desactualizados. Si ocurre un fallo en el servicio, el administrador no tendrá una visión inmediata del problema hasta que el informe sea reprocesado, limitando la capacidad de respuesta.

10. Ampliaciones/mejoras del servicio

Las siguientes utilidades y herramientas complementarias están diseñadas para mitigar las deficiencias identificadas en el apartado 9 y mejorar significativamente la funcionalidad, la seguridad y la fiabilidad de la implementación del Proxy Inverso (HAProxy/Nginx/ATS).

1. **Alta Disponibilidad (HA) y Mitigación de SPOF**

   La principal carencia del escenario es el Punto Único de Fallo (SPOF) en el Reverse Proxy.

➢ Keepalived (Balanceador de Capa 4 para HA)
   - **Descripción:** keepalived implementa el protocolo VRRP (Virtual Router Redundancy Protocol). Permite que dos servidores compartan una única IP Virtual (VIP). Si el Reverse Proxy primario falla, el servidor secundario toma el control de la VIP instantáneamente.
   - **Mejora:** Elimina el SPOF del Reverse Proxy. Se establecería un cluster de dos RP idénticos (RP-A como MASTER y RP-B como BACKUP).
   - **Implementación:** Configurar un health check para verificar si el servicio (e.g., Nginx) en el puerto 443 está activo. Si el health check falla en el MASTER, la VIP se migra automáticamente al BACKUP.

2. **Seguridad Avanzada y Re-Cifrado**

   Para abordar la debilidad de seguridad de los certificados autofirmados y el tráfico HTTP interno.

➢ Integración con Let's Encrypt / Certbot
   - **Descripción:** Certbot es la herramienta estándar para obtener e instalar automáticamente certificados SSL/TLS gratuitos y confiables de la CA Let's Encrypt.
   - **Mejora:** Sustituye los certificados autofirmados por certificados públicos válidos, eliminando los errores de seguridad en el navegador y estableciendo la confianza externa.

- **Implementación:** Configurar Certbot para obtener el certificado e integrarlo automáticamente con Nginx o HAProxy, añadiendo una tarea cron para la renovación automática.

3. **Filtrado Dinámico de Contenido y Clasificación**

Esta ampliación se centra en dotar al *Reverse Proxy* de capacidades de **toma de decisiones dinámica a nivel de Capa 7 (HTTP)**, aprovechando información de clasificación externa, sin convertirlo en un filtro de contenido tradicional.

➢ Uso de Listas de Reputación IP Auto-Actualizadas (WAF / Módulos de RP)

- Descripción: Se integra una solución complementaria (como un módulo de WAF o un feed de terceros) que consulta o descarga listas de reputación de direcciones IP que se actualizan periódicamente (ej., IPs de malware, botnets, o scanners).
- Mejora: Permite al sistema realizar filtrado preventivo a nivel de Capa 4/7 basado en la reputación de la fuente.
    - Protección contra DDoS: Es una defensa de primera línea contra botnets conocidos.

- Implementación: El RP configura reglas de ACL (Access Control List) o directivas de denegación basándose en las listas de reputación cargadas en memoria, actuando como un firewall de aplicación de borde.

4. **Monitorización y Gestión en Tiempo Real**
5.

Para superar las deficiencias de SARG y obtener una visión operativa activa.

➢ Prometheus + Grafana (Métricas en Tiempo Real)

- **Descripción:** Prometheus recopila métricas (RPS, latencia, uso de CPU) desde exporters integrados en Nginx, HAProxy o ATS. Grafana ofrece un dashboard web interactivo para visualizar estas métricas en tiempo real.
- **Mejora:** Transforma la auditoría de logs estática (SARG) en una monitorización dinámica y proactiva. Permite al administrador ver la tasa de aciertos de caché (ATS Cache Hit Ratio) y el rendimiento de los backends en vivo.

➢ Uso de Nginx/HAProxy Statistics Page

Descripción: Nginx (stub_status) y HAProxy (interfaz de estadísticas) ofrecen una página web que muestra el estado actual de los backends (arriba/abajo) y estadísticas básicas de conexión.

Mejora: Proporciona una interfaz gráfica nativa para la gestión activa y el diagnóstico de los backends sin necesidad de herramientas externas complejas, superando la limitación de SARG de no gestionar el estado del servicio.

6. Unificación de Servicios: Proxy Directo e Inverso en la Misma Máquina

Esta es una mejora arquitectónica que consolida la gestión del tráfico y los recursos de red de una organización en una única plataforma de *proxy*.

➢ Proxy Unificado (Proxy Inverso + Proxy Directo en Host Único)

- **Descripción:** Configurar un único servidor de alto rendimiento (como Nginx o ATS) para que actúe simultáneamente como:
  - **Proxy Inverso (PI):** Gestionando el tráfico entrante de Internet hacia los servicios *backend* (puerto 443).
  - **Proxy Directo (PD) / Transparente:** Gestionando el tráfico saliente de los usuarios internos de la red hacia Internet (puerto 3128/8080).
- **Mejora:**
  - **Consolidación de Recursos:** Se optimiza el uso de CPU, RAM y espacio en disco al compartir el mismo sistema operativo, kernel y gestión de recursos de red.
  - **Auditoría Centralizada:** Se utiliza el mismo mecanismo de *logging* y, potencialmente, la misma herramienta de análisis (como SARG), simplificando la auditoría de todo el tráfico de red de la empresa (entrante y saliente).
  - **Gestión de Certificados:** La gestión de la librería de certificados (para el RP) se centraliza.
- **Aplicación:** Aunque ambos escenarios están separados, si el servidor físico o la VM es lo suficientemente potente, esta configuración reduce la complejidad de infraestructura al requerir solo una máquina para dos funciones críticas de red.

11. Incidencias y principales problemas detectados

El proyecto presenta las siguientes dificultades técnicas relevantes:

Problema 1: Conflicto de Permisos y Ejecución del Ayudante SSL (Squid)

Este fue el problema más persistente y se relaciona directamente con el módulo avanzado de seguridad que probamos.

- **Incidencia Detectada**: Al intentar configurar el SSL Bumping en Squid, el proceso fallaba repetidamente al inicializar la base de datos de certificados (**ssl_db**). El error en los logs indicaba un "Permission Denied" al intentar ejecutar el programa ayudante **security_file_certgen**.
- **Justificación:** En la mayoría de las guías online, la ruta y el propietario del proceso son incorrectos para la versión específica de Ubuntu/Debian.
    - En Ubuntu, el usuario/grupo correcto es proxy:proxy, y la ruta del helper es /usr/lib/squid/security_file_certgen.
- **Resolución (Intento Parcial):** Se logró ejecutar el comando de inicialización correctamente:

```
chown -R proxy:proxy /var/lib/squid
```

y luego:

```
/usr/lib/squid/security_file_certgen -c -s /var/lib/squid/ssl_db..
```

Problema 2: Conflicto de Puertos y Dependencia Crítica (Nginx/HAProxy)

El proxy inverso requería una secuencia de arranque estricta.

- **Incidencia Detectada:** El servicio Nginx fallaba al arrancar después de configurar la Terminación TLS, indicando que el puerto 443 ya estaba en uso (Address already in use).
- **Justificación:** Ambos servicios (Nginx y Squid) estaban configurados para escuchar en el puerto estándar HTTPS (:443). La secuencia de arranque de los scripts de servicio era aleatoria.

- **Resolución:** Se implementó una resolución de diseño de topología para aislar los roles de los servicios:
    - **Proxies Inversos (HAProxy, Nginx, ATS):** Se configuraron y se ejecutan en una máquina (PC/VM) para realizar la terminación TLS y la evaluación del load balancing.
    - **Proxy Directo (Squid):** Se configuró y se ejecuta en una máquina separada (PC/VM), reservando su entorno para la evaluación de su funcionalidad de proxy Adelante y caché, eliminando así cualquier interferencia con los benchmarks de RPS.

Esta separación garantiza la validez y la repetibilidad de los datos de rendimiento obtenidos para cada tipo de proxy.

Problema 3: Fallo en la Ejecución del Benchmark (Error de set -e)

Incidencia: El script principal de benchmark (parte2.sh) salía inmediatamente después de imprimir el encabezado inicial, antes de ejecutar la primera prueba (HAProxy).

Justificación y Diagnóstico: El script utiliza la directiva set -e, lo que provoca que se detenga ante el primer error. El diagnóstico con bash -x reveló que el fallo ocurría al intentar acceder al script de configuración de HAProxy, ya que el sistema no lo encontraba.

Solución: Se identificó un error en la ruta relativa definida en el script. La estructura de directorios (STA/HAProxy, STA/Nginx, STA/parte2) requería sólo un nivel de retroceso.

- Ruta Original (Incorrecta): ../../HAProxy/configurar_haproxy_TLS.sh
- Ruta Corregida (Funcional): ../HAProxy/configurar_haproxy_TLS.sh

Una vez corregida la ruta a un solo nivel (../), el script pudo encontrar y ejecutar los scripts de configuración sin problemas.

Problema 4. Persistencia de bloqueos tras borrar reglas

En algunos casos, tras vaciar los archivos de reglas, los bloqueos seguían activos. Se determinó que Squid mantiene las reglas en memoria hasta que el servicio se recarga. Para resolverlo, se estableció como procedimiento obligatorio recargar el servicio Squid después de cualquier modificación.

Problema 5. Comportamiento diferente entre curl y el navegador

Se detectaron diferencias en el comportamiento del bloqueo entre las pruebas realizadas con curl y las realizadas desde el navegador. En algunos casos, una página se bloqueaba en el navegador pero no con curl, o viceversa. Se comprobó que esto se debía a diferencias entre tráfico HTTP y HTTPS, así como a la caché del navegador. El problema se mitigó forzando el uso de SSL-Bump y limpiando la caché durante las pruebas.

Problema 6. Reglas de bloqueo que no funcionaban desde archivos incluidos

Una de las incidencias más relevantes fue que las reglas de bloqueo no funcionaban cuando se añadían mediante un archivo incluido (squid_blocking_clients.conf), aunque sí funcionaban cuando se escribían directamente en squid.conf. Tras varias pruebas, se detectó que el problema estaba relacionado con el orden de las directivas http_access y con los permisos del archivo incluido. Reordenando las reglas y ajustando los permisos del archivo, el bloqueo comenzó a funcionar correctamente.

Conclusión

Las incidencias encontradas durante la realización del trabajo han permitido adquirir un conocimiento más profundo del funcionamiento de Squid, el filtrado de tráfico HTTPS y la gestión de ACL. Aunque algunas dificultades requirieron múltiples intentos de resolución, todas ellas han contribuido a mejorar la comprensión y estabilidad final del servicio implementado.

12. Resumen y Conclusiones

Este proyecto ha cumplido con el objetivo principal de configurar y evaluar tres soluciones de proxy inverso de alto rendimiento (HAProxy, Nginx y Apache Traffic Server (ATS)) en un entorno de laboratorio controlado.

La evaluación se centró en la capacidad de cada solución para realizar Terminación TLS (HTTPS) de manera eficiente, midiendo las peticiones por segundo (RPS) que cada una puede manejar bajo alta concurrencia.

1. Resumen de Rendimiento (RPS)

Los resultados de las ejecuciones finales demostraron una competencia muy ajustada por el liderazgo del rendimiento TLS:

Tecnología	RPS Promedio (pet/seg)	Clasificación
ATS (Traffic Server)	**10271.16**	**1º (Líder)**
HAProxy	**1399.57**	**2º**
Nginx (Fuente)	**1289.19**	**3º**

Tabla 37: Comparación Proxys inversos

Squid, a diferencia de los otros tres, se centró en la función de Proxy Adelante (Forward Proxy). Su trabajo con TLS se enfocó en la Intercepción y Reescritura de TLS (TLS-Bumping o SSL-Bump) , lo que le otorga un rol de seguridad y visibilidad único:

- Visibilidad: Permite inspeccionar el tráfico cifrado de los clientes para aplicar políticas de seguridad o auditoría.

- Seguridad: Requiere la generación e instalación de un certificado de confianza en los clientes para que la intercepción sea transparente, añadiendo una capa de control sobre el tráfico saliente.

Aunque no se midió el RPS de terminación, el desafío técnico en Squid fue la correcta implementación de la Intercepción TLS, una tarea conceptualmente más compleja que la simple terminación TLS.

2. Conclusiones Clave del Proyecto

1. **Liderazgo Abrumador de ATS:** El Impacto de la Caché La conclusión más destacada es que ATS (Traffic Server) ha superado a las otras dos soluciones por un orden de magnitud en el rendimiento promedio (10271.16 RPS). Este resultado es una prueba directa de que el benchmark de ATS en el promedio final se ejecutó aprovechando la caché.

    - **Interpretación:** Este escenario simula un entorno de producción ideal para ATS, donde el contenido es altamente repetitivo. Demuestra que el rendimiento real de un proxy optimizado para caché puede ser muy superior a cualquier solución de load balancing puro, ya que evita el overhead de la comunicación con el backend.

2. **Rendimiento en Modo Pass-Through (HAProxy vs. Nginx):** Al excluir el impacto de la caché, la comparación entre proxies de paso directo se mantiene. HAProxy (1399.57 RPS promedio) y Nginx (1289.19 RPS promedio) son los líderes en la gestión pura de la Terminación TLS.

    - **HAProxy** mostró la mayor eficiencia en el modo pass-through (8.6% más rápido que Nginx), lo que reafirma su arquitectura basada en eventos, ideal para el throughput de la capa 4/7.

3. **Roles y Topología del Laboratorio (Squid):** El proyecto diferenció claramente el rol de los servicios:

    - **Proxies Inversos (HAProxy, Nginx, ATS):** Dedicados a la aceleración del servidor.

- **Proxy Directo (Squid):** Aislado en otra máquina. Esta decisión de **aislamiento de topología** garantizó la validez de los datos al eliminar la interferencia de recursos. El trabajo con TLS en Squid se centró en la **Intercepción TLS (TLS-Bumping)**, un rol de seguridad y auditoría completamente distinto a la terminación TLS medida en RPS.

3. Conclusiones Personales sobre el Proyecto

La realización de este proyecto ha proporcionado valiosas lecciones prácticas en el ámbito de la ingeniería de sistemas:

1. "Especialización vs. Versatilidad" en la Arquitectura:
   La prueba demostró que la especialización de HAProxy en el load balancing puro le otorga una ligera ventaja en RPS. No obstante, Nginx ofrece un rendimiento casi idéntico con la ventaja adicional de ser un servidor web completo, lo que lo hace una solución más versátil. La elección final depende de si se prioriza el load balancing estricto o la capacidad multifunción.

2. El Rendimiento No Siempre Significa Velocidad Máxima (ATS):
   La posición de ATS enseña que la métrica de RPS más alta no siempre es la mejor solución. En un entorno real, la capacidad de ATS para gestionar y servir contenido en caché podría reducir drásticamente la carga del backend y el ancho de banda, lo que resultaría en un mejor rendimiento general del sistema, a pesar de tener un RPS más bajo en una prueba de paso directo.

3. **Valor del Esfuerzo y Aprendizaje Técnico:**

   La realización de este proyecto ha sido bastante dura y complicada, especialmente en las fases de depuración y diagnóstico. Sin embargo, el esfuerzo me ha servido para aprender muchas cosas nuevas y valiosas. Las dificultades técnicas encontradas (fallos de permisos, *timeouts*, conflictos de *scripting* y la sensibilidad de la configuración de ATS) obligaron a desarrollar una comprensión profunda de las arquitecturas internas de cada *proxy* y de la gestión avanzada de sistemas Linux.

Conclusión Final

El proyecto ha cumplido su objetivo de evaluar la eficiencia de distintas arquitecturas de proxy bajo la carga de Terminación TLS (HTTPS). Los resultados revelaron que la elección de la solución debe basarse estrictamente en su rol: HAProxy (1399.57 RPS) y Nginx (1289.19 RPS) demostraron ser los líderes en el modo pass-through (velocidad pura), mientras que ATS (10271.16 RPS) dominó en el promedio final, confirmando que su diseño orientado a la caché lo convierte en la solución más rápida para la entrega de contenido repetitivo. El trabajo también cubrió el rol único de Squid en la seguridad, implementando la Intercepción TLS (TLS-Bumping) para fines de auditoría, y se resolvió la necesidad de aislar la topología para garantizar la validez de los datos. Aunque el proyecto fue duro y complicado en sus fases de diagnóstico, las dificultades técnicas fueron la fuente principal de aprendizaje, proporcionando una comprensión práctica e invaluable de la arquitectura de servicios de alto rendimiento en entornos Linux.

ANEXO A: Parámetros de configuración y comandos de gestión del servidor

Esta sección enumera los archivos y parámetros principales que se utilizaron para controlar y configurar cada proxy inverso y proxy de caché.

1. HAProxy (High Availability Proxy)

HAProxy es un balanceador de carga de alto rendimiento. Su configuración es monolítica y se enfoca en las capas de red y transporte.

Archivo de Configuración	**Propósito Principal**	**Parámetros Clave**
/etc/haproxy/haproxy.cfg	Archivo de configuración principal. Contiene la definición de frontends, backends, y configuraciones globales.	bind (definir puerto y TLS), server (definir backends), balance (algoritmo), mode http / mode tcp.

Comandos de Administración	**Propósito**
`sudo systemctl start/stop/restart haproxy`	Controlar el servicio.
`sudo haproxy -c -f /etc/haproxy/haproxy.cfg`	Verificar la sintaxis de la configuración antes de reiniciar (crucial para evitar caídas).

`sudo systemctl status haproxy`	Verificar el estado y los logs recientes del servicio.

2. Nginx (Engine X - Desde Fuente)

Nginx es un servidor web y proxy inverso conocido por su arquitectura eficiente basada en eventos.

Archivo de Configuración	**Propósito Principal**	**Parámetros Clave**
`/usr/local/nginx/conf/nginx.conf`	Archivo de configuración principal (la ruta puede variar si se compila desde fuente).	worker_processes, listen (definir puerto y TLS), server_name, ssl_certificate/ssl_certificate_key, proxy_pass (reenvío a backend).

Comandos de Administración	**Propósito**
`sudo /usr/local/nginx/sbin/nginx -s stop`	Detener el servicio (usando la ruta de la versión compilada).
`sudo /usr/local/nginx/sbin/nginx`	Iniciar el servicio.

`sudo /usr/local/nginx/sbin/nginx -t`	Verificar la sintaxis del archivo de configuración (crucial).
`sudo /usr/local/nginx/sbin/nginx -s reload`	Recargar la configuración sin detener el servicio (relevante en producción).

3. Apache Traffic Server (ATS)

ATS está diseñado principalmente como un proxy de caché avanzado. Su administración se basa en varios archivos de configuración que controlan aspectos específicos.

Archivo de Configuración	**Propósito Principal**	**Parámetros Clave**
/etc/trafficserver/records.config	Archivo central que almacena la mayoría de las configuraciones de sistema.	proxy.config.ssl.enabled INT (Activación TLS), proxy.config.http.server_ports (Puerto de escucha), proxy.config.ssl.server.cert.path.
/etc/trafficserver/remap.config	Define las reglas de mapeo de URL (la función principal del proxy inverso).	map https://... http://... (Reglas de mapeo).

/etc/trafficserver/ssl_multicert.config	Define qué certificado usar para qué dirección IP/dominio en la terminación TLS.	dest_ip=* ssl_cert_name=... (Apuntar al certificado).

Comandos de Administración	**Propósito**
`sudo systemctl start/stop/restart trafficserver`	Controlar el servicio.
`sudo traffic_ctl config reload`	Recargar la configuración sin reiniciar el proceso principal (después de modificar records.config).
`sudo traffic_ctl config parse`	Verificar la sintaxis de los archivos de configuración (especialmente útil para ATS).

4. Squid

Squid es un proxy de caché muy popular, utilizado a menudo como proxy directo.

Archivo de Configuración	**Propósito Principal**	**Parámetros Clave**

/etc/squid/squid.conf	Archivo de configuración principal y monolítico.	http_port (Puerto HTTP), https_port (Puerto HTTPS/TLS), acl (Listas de Control de Acceso), http_access, ssl_bump (Directiva clave para la Intercepción TLS).

Comandos de Administración	**Propósito**
`sudo systemctl start/stop/restart squid`	Controlar el servicio.
`sudo squid -k reconfigure`	Recargar la configuración sin detener el servicio (método más común).
`sudo squid -k shutdown`	Apagar el servicio de forma controlada.
`sudo squid -z`	Inicializar los directorios de caché (necesario tras cambios en la configuración de la caché).
`sudo squid -k check`	Verificar la sintaxis del archivo squid.conf (crucial).

ANEXO B: Parámetros de configuración y comandos de gestión del cliente

El cliente en este proyecto tiene dos funciones principales: la prueba de Conexión/Funcionalidad y el Benchmark de Rendimiento.

1. Herramienta de Carga y Benchmark: Apache Bench (ab)

Apache Bench fue la herramienta fundamental para medir el rendimiento de Peticiones por Segundo (RPS).

Parámetro Principal	**Descripción y Uso en el Escenario**
`-n ${NUM_PETICIONES}`	Número de peticiones a realizar. En este escenario, el valor fue 1000.
`-c ${CONCURRENCIA}`	Nivel de concurrencia; número de peticiones simultáneas. En este escenario, el valor fue 100.
`-k`	Habilita la función Keep-Alive. Esto indica que se intentarán realizar múltiples peticiones por la misma conexión TLS/TCP, simulando mejor el tráfico web moderno.
`https://127.0.0.1/`	URL objetivo. Se utiliza la dirección de loopback (127.0.0.1) con el protocolo HTTPS para probar el rendimiento del proxy localmente.

Comando de Ejecución (Plantilla):

```
ab -n 1000 -c 100 -k https://127.0.0.1/
```

Parámetros de Salida Relevantes

El script del benchmark procesó la siguiente línea de salida para extraer el RPS:

- Requests per second: [X.XX]: Métrica clave utilizada para la comparativa.

2. Herramienta de Diagnóstico y Funcionalidad: curl

Curl fue utilizado como herramienta de administración para la prueba rápida de conectividad y la validación inicial de la terminación TLS en cada proxy.

Parámetro Principal	**Descripción y Uso en el Escenario**
`-k` o **`--insecure`**	Inseguro. Ignora la validación del certificado SSL/TLS. Este parámetro fue esencial porque se utilizaron certificados auto-firmados (selfsigned.crt) que no son confiables para los navegadores ni para curl por defecto, permitiendo la prueba de conexión sin fallos de validación.
`https://127.0.0.1`	URL objetivo. Se utiliza para verificar que el proxy inverso está escuchando y terminando TLS correctamente, y que la petición está siendo reenviada con éxito al backend.

Comando de Diagnóstico (Plantilla)

```
curl -k https://127.0.0.1
```