Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍA DE COMPUTADORES

Plataforma de evaluación de rendimiento de tráfico de red basada en la pila ELK

Estudante: David Ignacio Conde
Dirección: Miguel González López

A Coruña, febreiro de 2020.

A mis padres, que me apoyaron siempre para llegar hasta aquí. Gracias por no dejar que me rindiera.

Agradecimientos

- A mis padres, por su apoyo incondicional en todo momento para que acabara estos estudios.
- Al director, Miguel, por proponerme este trabajo, por los buenos consejos y por el tiempo dedicado todas las semanas a revisar el proyecto.

Resumen

Este trabajo se adentra en el campo de las herramientas libres de generación de tráfico en red, comparándolas y escogiendo algunas de ellas para la realización de una plataforma de monitorización basada en la pila ELK (Elasticsearch, Logstash, Kibana). Esta plataforma se basará en ficheros de *script* que ejecuten tests de las herramientas escogidas. Los datos que provienen de estos *tests* se procesarán mediante ELK para su estructuración, almacenamiento y posterior visualización.

Abstract

This work goes into the field of open source network traffic generation tools, comparing them and choosing some of them for the realization of a monitoring platform based on the ELK stack (Elasticsearch, Logstash, Kibana). This platform will be based on script files that run tests of the chosen tools. The data that comes from these tests will be processed by ELK for structuring, storage and subsequent visualization.

Palabras clave:

- Generador de tráfico
- Software libre
- Mgen
- Rapr
- T-rex
- Iperf
- Netperf
- Warp17
- D-ITG
- ELK
- Elasticsearch
- Logstash
- Kibana

Keywords:

- Network traffic generator
- Open source
- Mgen
- Rapr
- T-rex
- Iperf
- Netperf
- Warp17
- D-ITG
- ELK
- Elasticsearch
- Logstash
- Kibana

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este capítulo introductorio se expone el planteamiento y las líneas a seguir del trabajo.

1.1 Motivación

Un administrador de tecnologías de la información requiere de herramientas para monitorizar el tráfico de red y también para poner a prueba un sistema. Aunque existen opciones comerciales de pago y privativas, surge la necesidad de tener a mano herramientas de uso libre y gratuito.

Además, estas herramientas lanzan información muy valiosa de su ejecución. Se plantea por tanto el desarrollo de una plataforma sencilla de monitorización que recoja estos datos, los almacene de forma estructurada y los permita visualizar.

1.2 Objetivos

1. Comparar las características de las principales herramientas libres de generación de tráfico de red: funcionalidades, rendimiento, versatilidad, documentación, soporte, facilidad de uso, etc.

2. Diseñar diferentes tests de interés y comprobar su ejecución con las distintas herramientas, presentando de forma gráfica los resultados mediante la pila ELK.

1.3 Organización del proyecto

El proyecto se divide principalmente en dos partes diferenciadas. Por un lado se realiza una comparativa de siete herramientas de generación de tráfico de red. Por otro lado, de esas siete herramientas se seleccionarán algunas para desarrollar una plataforma de monitorización de tráfico de red en tiempo real basada en la pila ELK.

1.3.1 Memoria

La memoria se ha estructurado en los siguientes capítulos, siguiendo el orden establecido en la planificación:

- **Metodología y Planificación**: en primer lugar, se expone la metodología a seguir para el desarrollo del proyecto, y en segundo lugar se explica cómo se ha planificado el desarrollo del proyecto estableciendo las tareas en Microsoft Project.
- ***Software* utilizado**: descripción de cada producto de software utilizado en este trabajo.
- **Comparativa de herramientas de generación de tráfico de red**: dedicado a la comparativa de diferentes aspectos de las herramientas.
- **Plataforma de monitorización basada en la pila ELK**: capítulo dedicado a repasar el desarrollo de la plataforma basada en la pila de Elastic.
- **Conclusiones**: último capítulo dedicado a realizar reflexiones sobre el trabajo realizado, planteando trabajos futuros que se podrían hacer sobre la plataforma desarrollada.

Capítulo 2

Metodología y Planificación

2.1 Metodología

Para la realización de este trabajo se ha elegido una metodología incremental basada en hitos sobre las fases descritas en el anteproyecto. Dichas fases son las siguientes:

- Revisión del estado del arte sobre generadores de tráfico de uso libre.
- Estudio de la pila ELK.
- Desarrollo de la plataforma de ejecución de tests de rendimiento basados en generación de tráfico.
- Diseño de los filtros Logstash de ingesta en Elasticsearch de los resultados de los tests.
- Desarrollo de la interfaz gráfica de visualización en Kibana.

Se ha establecido una reunión semanal con el director del trabajo de una hora. Durante esta reunión se revisará lo realizado esa semana, el director realizará las correcciones oportunas y se pondrán objetivos para la semana siguiente. En caso de no haber sido realizados en el plazo de la semana, se pospondrán para la siguiente reunión.

Se utilizará un documento de Word, compartido con el director del trabajo, para realizar un blog con el fin de hacer un seguimiento del proyecto documentado y para la comunicación con el director por medio de los comentarios del documento.

2.2 Planificación

Para realizar la planificación del proyecto se ha utilizado el programa Microsoft Project que proveen las máquinas de referencia Windows 10 de la universidad.

A partir de la metodología anterior se ha creado un Diagrama de Gantt que recoge los hitos como tareas resumen. Bajo estas tareas resumen se han creado las tareas a realizar en el proyecto. A cada tarea se le ha presupuestado un número de horas, como se puede ver en la figura 2.1.

	Nombre de tarea	Duración	Comienzo	Fin	Pred
1	Toma de contacto	16 horas	lun 02/09/19	mar 03/09/19	
2	Preparación de entorno de pruebas	8 horas	mié 04/09/19	mié 04/09/19	1
3	**Revisión del estado del arte sobre generadores de tráfico de uso libre**	**37,88 días**	**jue 05/09/19**	**lun 28/10/19**	**2**
4	Estudio documentación MGEN	40 horas	jue 05/09/19	mié 11/09/19	2
5	Instalación y prueba MGEN	16 horas	jue 12/09/19	vie 13/09/19	4
6	Estudio documentación RAPR	28 horas	lun 16/09/19	jue 19/09/19	5
7	Instalación y prueba RAPR	16 horas	jue 19/09/19	lun 23/09/19	6
8	Estudio documentación Iperf	20 horas	lun 23/09/19	mié 25/09/19	7
9	Instalación y prueba Iperf	8 horas	jue 26/09/19	jue 26/09/19	8
10	Estudio documentación Netperf	32 horas	vie 27/09/19	mié 02/10/19	9
11	Instalación y prueba Netperf	16 horas	jue 03/10/19	vie 04/10/19	10
12	Estudio documentación TRex	56 horas	lun 07/10/19	mar 15/10/19	11
13	Instalación y prueba TRex	30 horas	mié 16/10/19	lun 21/10/19	12
14	Estudio documentación D-ITG	25 horas	lun 21/10/19	jue 24/10/19	13
15	Instalación y prueba D-ITG	16 horas	jue 24/10/19	lun 28/10/19	14
16	**Estudio de la pila ELK**	**7 días**	**lun 28/10/19**	**mié 06/11/19**	**15**
17	Instalación y configuración básica de ELK	8 horas	lun 28/10/19	mar 29/10/19	15
18	Aprendizaje de la pila ELK	48 horas	mar 29/10/19	mié 06/11/19	17
19	**Desarrollo de la plataforma de ejecución de tests de rendimiento basados en generación de tráfico.**	**7,63 días**	**mié 06/11/19**	**lun 18/11/19**	**18**
20	Diseño de pruebas	21 horas	mié 06/11/19	lun 11/11/19	18
21	Desarrollo de scripts para ejecución de pruebas	40 horas	lun 11/11/19	lun 18/11/19	20
22	**Configuración de Logstash para ingesta de datos en Elasticsearch**	**9 días**	**lun 18/11/19**	**vie 29/11/19**	**21**
23	Diseño e implementación configuración datos Iperf	16 horas	lun 18/11/19	mié 20/11/19	21
24	Diseño e implementación configuración datos Netperf	24 horas	mié 20/11/19	lun 25/11/19	23
25	Diseño e implementación configuración datos D-ITG	32 horas	lun 25/11/19	vie 29/11/19	24
26	**Diseño de la interfaz gráfica de visualización en Kibana**	**9,63 días**	**vie 29/11/19**	**vie 13/12/19**	**25**
27	Diseño e implementación de Dashboard de Iperf	16 horas	vie 29/11/19	mar 03/12/19	25
28	Diseño e implementación de Dashboard de Netperf	16 horas	mar 03/12/19	jue 05/12/19	27
29	Diseño e implementación de Dashboard de D-ITG	24 horas	jue 05/12/19	mar 10/12/19	28
30	Diseño e implementación de Canvas de D-ITG	21 horas	mar 10/12/19	vie 13/12/19	29
31	**Memoria**	**11,5 días**	**vie 13/12/19**	**mié 15/01/20**	**30**
32	Toma de contacto con Overleaf	4 horas	vie 13/12/19	vie 13/12/19	30
33	Estructuración de la memoria	8 horas	vie 13/12/19	lun 16/12/19	32
34	Redacción de la memoria	80 horas	lun 16/12/19	mié 15/01/20	33

Figura 2.1: Tareas del diagrama de Gantt y las horas presupuestadas.

En la figura 2.2 se puede ver el diagrama de Gantt resultante con respecto a las tareas.

Se ha configurado como fecha de inicio el 1 de Septiembre de 2019 y se ha modificado el calendario para añadir unos días de vacaciones de navidad. En la figura 2.3 se puede ver como se ha modificado el calendario en Microsoft Project.

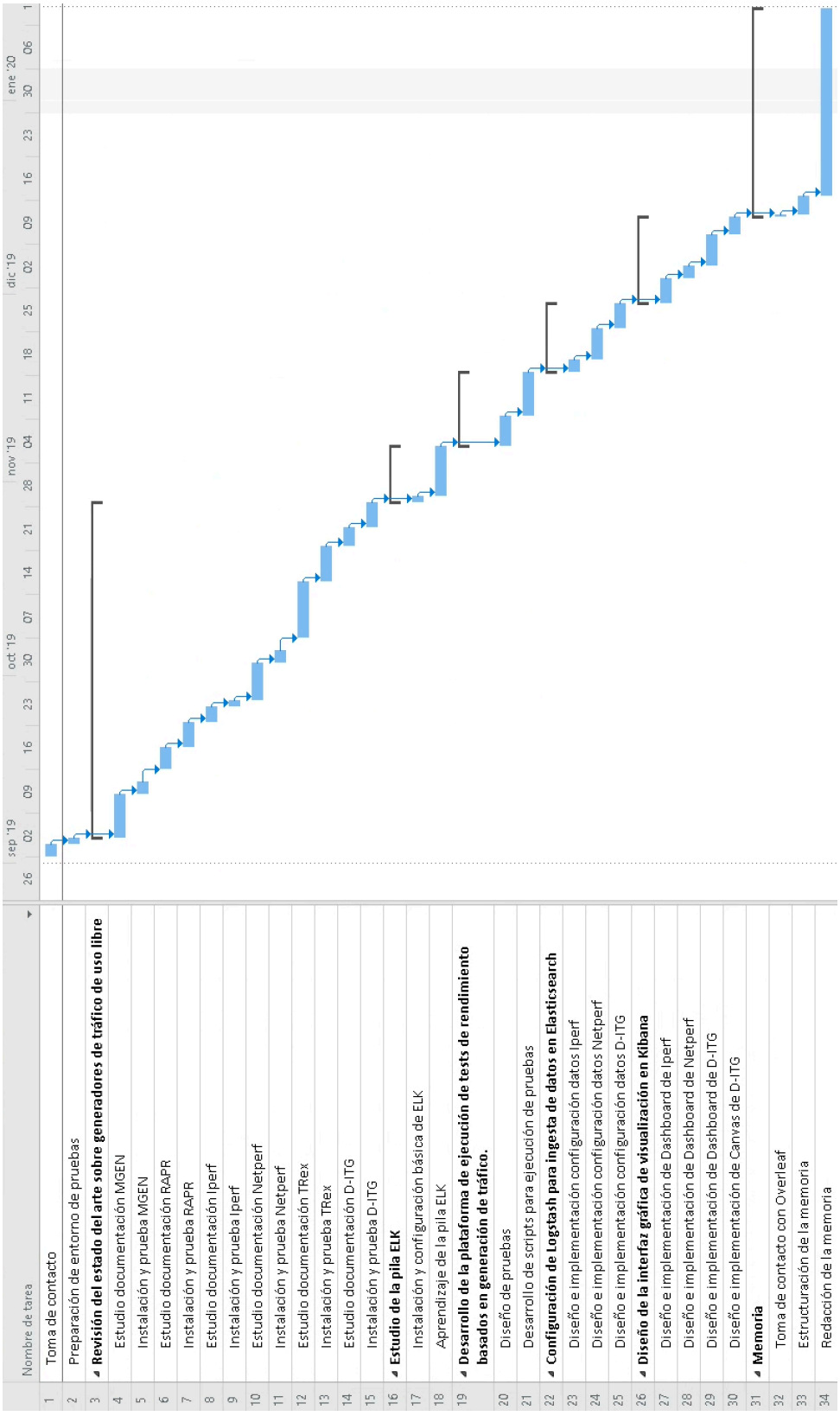

Figura 2.2: Diagrama de Gantt completo y tareas.

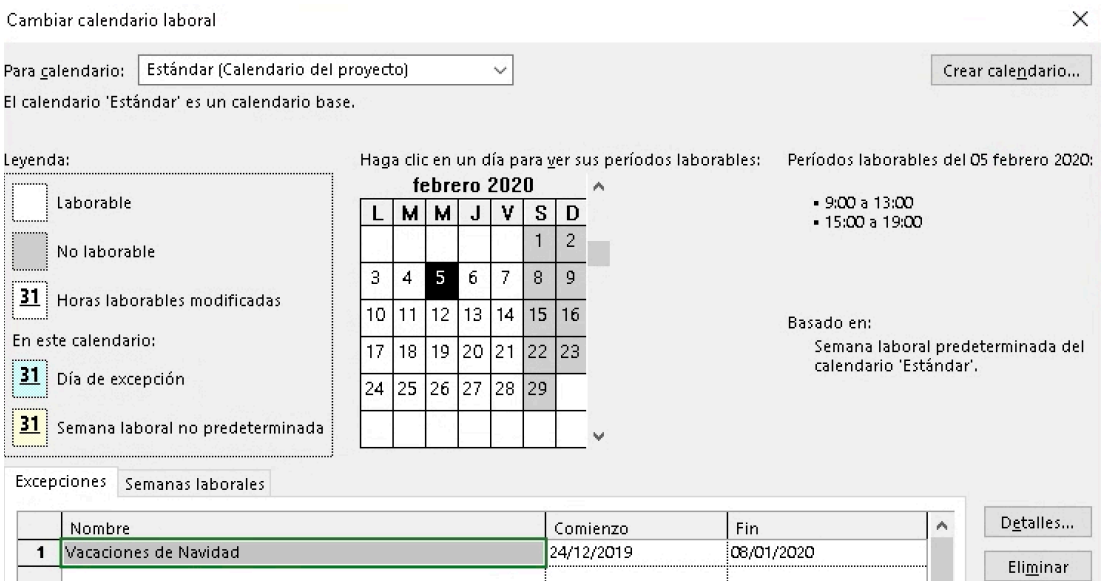

Figura 2.3: Calendario de Microsoft Project.

2.2.1 Desviaciones en la ejecución respecto a la planificación

Durante la ejecución del proyecto han habido tareas que se subestimaron en horas de trabajo y otras que se sobrestimaron, esto provocó que parte del esfuerzo se moviera sobre todo de las tareas de prueba e instalación a las de estudio de la documentación, durante el primer hito. Durante el segundo hito ocurrió de la misma manera con el diseño de las pruebas y el desarrollo de *scripts*. Mientras que el diseño sencillo agilizó la primera tarea, el esfuerzo en horas de trabajo se gastó luego en el desarrollo para programar y probar los *scripts*. La redacción de la memoria requirió volver a documentarse en reiteradas ocasiones y eso provocó un retraso bastante grande. Esto, sumado a la realización de continuas mejoras y correcciones, llevó a esta tarea a aumentar 32 horas más su duración.

2.2.2 Costes

Los recursos empleados han sido David Ignacio Conde (alumno) y Miguel González López (director). El alumno ha desempeñado 1410 horas de trabajo y el director 20 horas. Aplicando un coste por hora al alumno de 15 euros la hora y al director 30 euros la hora, el coste total de proyecto asciende a la suma de 21750 euros.

Capítulo 3

Software utilizado

En este capítulo se repasará el *software* utilizado en la realización de este trabajo. Además, se indicarán las características principales de cada uno de ellos.

3.1 CORE

De sus siglas *Common Open Research Emulator*, es una herramienta de emulación de redes en una o varias máquinas. En CORE tenemos por encima la interfaz de usuario (core-gui) que nos permite crear *canvas* y configurar nodos y sus conexiones en diferentes capas. Por debajo trabaja el core-daemon [2].

CORE nos permite ejecutar las herramientas del comparativo bajo un entorno controlado y configurable. Instalando las herramientas bajo el mismo sistema operativo, donde ejecutamos CORE, podemos lanzar estos programas desde los nodos emulados.

Características principales:

- Eficiente y escalable.
- Ejecuta aplicaciones y protocolos sin realizar modificaciones.
- Interfaz de usuario *Drag and Drop*.
- Altamente personalizable.

3.2 MGEN

MGEN (*Multi-GENerator*) es una herramienta de *software* libre desarrollada por la *U.S. Naval Research Laboratory* por el grupo de investigación PROTEAN [3].

Características principales:

- Generación de patrones de tráfico en tiempo real.
- Gestión de *logs* en diferentes formatos.
- Uso de *scripts* para manejar los patrones de tráfico generado.
- Control remoto en tiempo real.
- Configuración de ejecución mediante *scripts*.
- Emulación de patrones de tráfico *unicast* y/o *multicast* UDP y aplicaciones TCP IP.
- Posibilidad unirse y abandonar grupos *multicast*.

3.3 RAPR

Desarrollado en la *U.S. Navy Research Laboratory*, es un generador de tráfico *stateful* que replica patrones de tráfico reales usando mecanismos de transporte UDP y TCP [4].

Características principales:

- Generar y responder a tráfico de red en tiempo real.
- Utiliza MGEN para generar el tráfico de red.
- El tráfico de red se puede guardar en forma de *logs* para su análisis posterior.
- Los *logs* se pueden usar para calcular estadísticas de rendimiento sobre la tasa de transferencia, ratios de pérdida de paquetes, retardo de comunicaciones y más.

3.4 TRex

TRex es una herramienta libre, muy completa y con una gran cantidad de opciones y parámetros que personalizar para crear tráfico real. Permite generar tráfico *stateless* y *stateful* [5].

Características principales respecto a la generación *stateful*:

- Genera y analiza tráfico de capas 4 a 7. Tiene las capacidades de herramientas comerciales.
- Generador de tráfico *stateful* basado en pre-procesamiento y reproducción inteligente de plantillas de tráfico real.
- Generar y amplificar tráfico de cliente y servidor.
- Es posible añadir funcionalidad personalizada.
- Hasta 200Gb/sec de ancho de banda.
- Bajo coste.
- Fácil instalación y despliegue.
- Soporte para interfaces virtuales. Posibilidad de usar TRex en entornos completamente virtuales. (Amazon AWS, Cisco LaaS, VirtualBox/Vmware).

Características principales respecto a la generación *stateless*:

- Elaboración y generación de flujos de tráfico sin estado.
- Gran escala: admite hasta 20 millones de paquetes por segundo (mpps).
- Soporte de múltiples transmisiones.
- Posibilidad de cambiar cualquier campo dentro del paquete.
- Soporte continuo/ráfaga /ráfaga múltiple.

- Soporte interactivo: consola, GUI.
- Estadísticas por flujo, latencia y *jitter*.
- API de Python para automatización.
- Soporte multiusuario.
- Protocolos de enrutamiento.

3.5 Warp17

Warp17 es un generador de tráfico *stateful* de *software* libre desarrollado por Juniper Networks. Es una solución ligera para generar grandes volúmenes de tráfico basado en sesión con tasas de configuración muy altas [6].

Características Principales:

- Generador de Tráfico capa 1 a capa 7.
- Multiplataforma.
- Basado en DPDK.
- Generación *stateful*.
- Infraestructura TCP/UDP configurable que se puede utilizar para generar una alta configuración de conexión y velocidades de datos para el tráfico de aplicaciones.

3.6 Netperf

Netperf es una herramienta libre desarrollada desde Hewlett Packard (HP). El principal foco de Netperf es el rendimiento en transferencias de datos unidireccionales y petición/respuesta [7].

Los tests que podemos realizar en la versión 2.7 son:

- Transferencia unidireccional y petición/respuesta TCP y UDP sobre IPv4 e IPv6 usando la interfaz de *sockets* [8].
- Transferencia unidireccional y petición/respuesta TCP y UDP sobre IPv4 e IPv6 usando la interfaz XTI.
- Transferencia unidireccional y petición/respuesta a nivel de enlace usando la interfaz DLPI.
- *Sockets* de dominio Unix.
- Transferencia unidireccional y petición/respuesta SCTP sobre IPv4 e IPv6 usando la interfaz de *sockets*.

3.7 Iperf

Iperf3 es su versión más reciente, es una herramienta desarrollada por ESnet y la *Lawrence Berkeley National Laboratory* [9][10].

Características Principales:

- En TCP y SCTP:
  - Medición de ancho de banda.
  - Informar sobre el tamaño de MSS/MTU y los tamaños de lectura observados.
  - Soporte para tamaño de ventana TCP via *buffers* del *socket*.
- En UDP:
  - El cliente puede crear flujos UDP de un ancho de banda especificado.
  - Medición de pérdida de paquete.

  - Medición de retardo y *jitter* o fluctuación de retardo.

- Soporte IPv4 e IPv6.
- Multiplataforma.

3.8 D-ITG

D-ITG o *Distributed Internet Traffic Generator*, es una plataforma capaz de producir tráfico a nivel de paquete de forma precisa replicando procesos estocásticos para el IDT (*Inter Departure Time*) o tiempo de envío y PS (*Packet Size*) o tamaño de paquete. Está desarrollada en la Universidad de Nápoles por miembros del *Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione* (DIETI) [11][12].

- Soporte IPv4 e IPv6.
- Plataformas soportadas:
  - Linux (Ubuntu, Debian, Fedora, CentOS, OpenWRT).
  - Windows (XP, Vista, 7).
  - OSX (Leopard).
  - FreeBSD.
- Generación de tráfico capa 3, 4 y 7 .
- Generación de tráfico a nivel de paquete mediante modelos estocásticos para IDT (*Inter-Departure Time*) y PS (*Packet Size*).

3.9 Elastic Stack

Elastic es una empresa que provee soluciones tanto de código abierto como comerciales para buscar, analizar y visualizar datos. Se utilizarán los siguientes productos de Elastic denominados bajo el acrónimo ELK [13]:

- **Logstash**: recibir la ingesta de datos y transformarlos.
- **Elasticsearch**: almacenar datos transformados.
- **Kibana**: visualizar los datos.

3.9.1 Logstash

Es un *pipeline* de procesamiento de datos del lado del servidor que ingesta datos de multitud de fuentes. Estos datos son transformados y enviados a Elasticsearch para su almacenamiento.

3.9.2 Elasticsearch

Es un motor de búsqueda y analíticas *RESTful* distribuido capaz de abordar un número creciente de casos de uso. Almacena de forma centralizada los datos, los cuales podremos visualizar en Kibana.

3.9.3 Kibana

Es la interfaz interactiva y la parte visual de la pila ELK. Gracias a este producto podemos entender mejor los datos realizando gráficos en tiempo real derivados de la ingesta de datos continua.

Capítulo 4

Comparativa de herramientas de generación de tráfico de red

Tras haber visto el *software* que ha sido utilizado para este trabajo, en este capítulo se realizará un análisis de diferentes aspectos de las herramientas de generación de tráfico de red, haciendo una comparación entre ellas. Se tendrán en cuenta los siguientes aspectos:

- **Instalación**: se repasarán los aspectos de descarga, instalación y puesta a punto de la herramienta de cara a su funcionamiento, destacando errores o problemas en caso de que los hubiera.
- **Documentación**: se tendrán en cuenta la extensión, claridad de la documentación y la inclusión de ejemplos a diferentes niveles.
- **Formas de uso**: se indicarán las formas de uso que tiene la herramienta, tantos las que se han probado como las que ofrece en su documentación/portal web.
- **Facilidad de uso y complejidad**: se realizarán apreciaciones subjetivas intentando aportar argumentos sobre la facilidad de uso de la herramienta. Se tendrá en cuenta la curva de dificultad para realizar *tests*/generación de tráfico sencillos, la experiencia de uso, la interacción con la información aportada en la documentación y la complejidad del programa.
- **Gestión de *logs***: se indicará cómo se gestiona la salida de datos de las herramientas.
- **Modificación de *buffers***: se hará una indicación sobre la existencia o no de esta característica. También se repasarán los tipos de *buffers* a modificar.
- **Licencia de uso**: comentario sobre la licencia de uso de la herramienta, de qué tipo es y sus consecuencias.

- **Mantenimiento**: se repasarán algunas características respecto a la actualización de la herramienta en el tiempo y los canales de comunicación con la comunidad alrededor del proyecto.
- **Dirección de tráfico generado**: algunas herramientas son capaces de generar tráfico en una dirección o en ambas, lo que da pie a realizar pruebas diferentes con ellas. Se tendrán en cuenta los mecanismos para generar tráfico en ambas direcciones en caso de que fuera posible.
- **Plataformas soportadas**: plataformas donde se puede instalar y utilizar el programa.
- **Tipos de tráfico generado**: tipos de tráfico que se genera desde estas herramientas. Se tendrán en cuenta los protocolos disponibles, la forma de generar el tráfico y también el enfoque en la generación y/o medición de este.

En la figura 4.1 se puede apreciar como las características pertenecen a diferentes ámbitos.

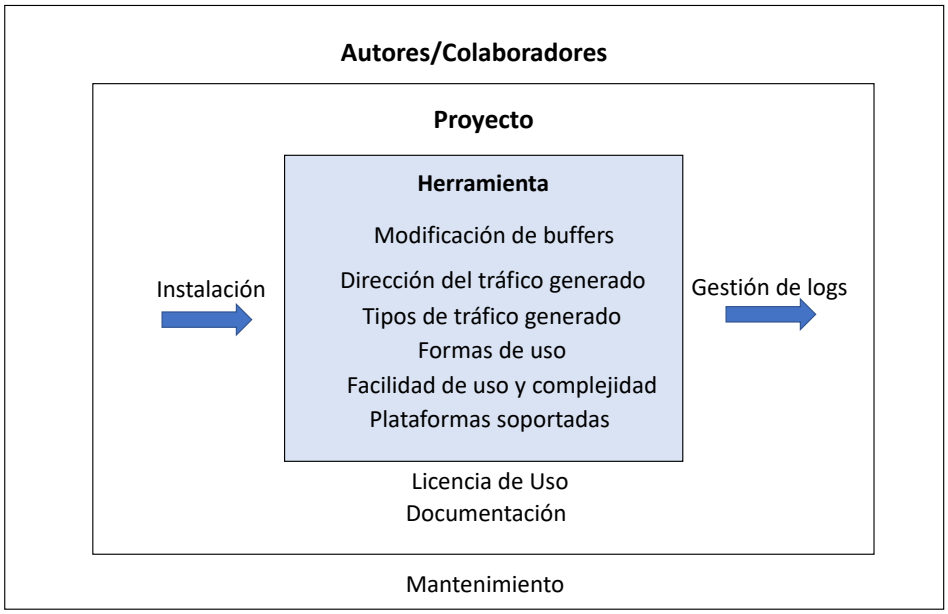

Figura 4.1: Esquema del ámbito de las características a comparar.

4.1 Introducción

Las herramientas se han instalado en una máquina virtual Ubuntu 18.04 para su prueba y análisis. Hemos decidido que como requisito para entrar en la comparativa la herramienta sea compilable y ejecutable en este sistema operativo de cara a poder probarlo en igualdad de

condiciones con el resto de herramientas. Con el objetivo de simular nodos diferentes dentro de una red para lanzar los *tests* y generar el tráfico se ha utilizado el emulador de red CORE.

Sin embargo, dado que es una tarea titánica la prueba de todos los aspectos de todas las herramientas, esta parte del trabajo se ceñirá sobre todo a la información aportada por las páginas oficiales de los proyectos y su documentación.

En primer lugar, se han descartado dos de los siete programas inicialmente propuestos para la comparación dado que no fue posible utilizarlos en el entorno de pruebas elegido (Ubuntu 18.04). La primera fue Warp17 dado que la última versión de Ubuntu donde compilaba era la 14.04, una explicación pormenorizada se encuentra en el anexo E. La segunda fue RAPR dado que el proyecto no compilaba y además estaba abandonado desde hacía muchos años, una explicación más en profundidad se encuentra en el anexo D.

4.2 Instalación

En general, casi todos los proyectos vistos requieren la compilación de sus ficheros fuente, algo que, dependiendo de si la herramienta está debidamente actualizada y mantenida, habrá más o menos problemas en esta tarea inicial. TRex es una de las herramientas que no requiere que los archivos fuente sean compilados.

Se han encontrado algunos problemas para la compilación de MGEN cuando los ficheros fuente se encontraban en un servidor FTP en vez del repositorio Github donde están actualmente. El problema residía en encontrar las versiones más recientes para compilar el programa. Una versión más amplia de lo sucedido con MGEN se expone en el anexo C.

Iperf, Netperf y D-ITG se compilaron e instalaron sin problemas.

Programa	Versión
MGEN	5.02c
Iperf	3.7
Netperf	2.7
TRex	2.73
D-ITG	2.8.1

Tabla 4.1: Versiones de los programas comparados.

4.3 Documentación

La documentación permite conocer el funcionamiento y posibilidades de un *software*. Se ha realizado una clasificación teniendo en cuenta aspectos como claridad y extensión. También se tienen en cuenta la inclusión tanto de ejemplos concretos como de ejemplos generales.

Se han denominado ejemplos concretos a los que se incluyen para ilustrar, por ejemplo, las opciones de comando o aspectos menores de la herramienta. Por otra parte, se han denominado ejemplos generales a los que se incluyen para casos de uso más complejos que permiten aportar una visión más general de las posibilidades del programa.

En la tabla 4.2 se muestra, por un lado, una clasificación ordenando de menor a mayor las características de extensión y claridad de la documentación. Por otro lado, se especifica la inclusión de ejemplos concretos y generales.

Programa	Extensión	Claridad	Ejemplos Concretos	Ejemplos Generales
MGEN	****	***	Sí	Sí
Iperf	*	****	Pocos	Sí (desactualizados v2)
Netperf	***	**	Sí	Sí
TRex	*****	*	Sí	Sí
D-ITG	**	*****	Sí	Sí

Tabla 4.2: Comparativa de la documentación.

Iperf carece de una documentación al uso como el resto de herramientas. Existen dos páginas supuestamente oficiales:

- Página web Iperf.fr[9]. Aunque está desactualizada, proporciona información sobre las versiones anteriores a la 3.1.1. Además, está disponible una relación de comandos bien explicada.
- Página web de EsNet[10]. Proporciona información a partir de la versión 3.1.1 Iperf[14].

Si atendemos a lo que se dice en el repositorio Github de Iperf [15], la página oficial es la de EsNet[10]. Sin embargo, la página web de Iperf.fr aporta información interesante como la relación entre la versión 2 y al versión 3 (hasta la versión 3.1), un manual de usuario con las opciones de comando y una serie de ejemplos de la versión 2. Esta versión 2 coexiste con la versión 3 pero no existe retrocompatibilidad entre ellas [16].

En la página de EsNet se muestra la página de manual que veríamos si ejecutamos el comando *man* en una terminal. Sin embargo, de cara a consultar la documentación de Iperf, se consultará el manual del propio programa en la terminal de la máquina de pruebas.

Netperf destaca en su documentación [17] por su redacción en un registro informal con expresiones coloquiales y argot. Parece querer suavizar el carácter técnico del documento. Sin embargo, llega a ser más un problema que una ayuda, dado que muchos de los términos son difíciles de entender a priori.

Trex posee una gran cantidad de recursos que podemos considerar documentación [18]. Los manuales principales son para el soporte *stateful* [19] y soporte *stateless* [20]. El manual

stateful presenta también las indicaciones para configurar la herramienta [21]. En este punto destaca por no tener un planteamiento inicial claro de cara a configurar las interfaces a utilizar. Hasta el punto 3.5 [22] no se llega a una conclusión sobre cómo utilizar de forma básica el programa.

D-ITG tiene la documentación [1] mejor estructurada y más clara. Está bien organizada de manera que es fácil aprender los conceptos básicos de la herramienta.

4.4 Formas de uso

Las herramientas del comparativo se pueden utilizar de diferentes formas, la tabla 4.3 recopila una relación entre las formas de uso encontradas y su relación con los programas.

Programa	Línea Comandos	API	GUI	*Script*	Fichero Config.
MGEN	X			X	
Iperf	X				
Netperf	X				
TRex	X	X	X (solo *stateless*)		X
D-ITG	X	X		X	

Tabla 4.3: Comparativa de formas de uso.

El uso por línea de comandos es el más extendido. Se realiza casi siempre ejecutando un binario seguido de opciones con sus parámetros. Una excepción es la interfaz interactiva en línea de comandos en el modo *stateless* de TRex [23].

Cada herramienta tiene una forma particular de uso, ya sea por el uso de convenciones en el paso de parámetros únicas, en el caso de Netperf, su estructuración en módulos (Fig. 4.2) con propósitos específicos, en el caso de D-ITG, la generación de tráfico mediante configuración de ficheros en TRex o el uso de ficheros de *script* en MGEN.

Entre los que se ejecutan solo con opciones de comando está Netperf, el cual tiene un sistema de convenciones único en el comparativo con una gran complejidad pero una gran cantidad de opciones. Estas opciones se dividen en dos tipos:

- **Opciones globales**: comunes a todos los test.
- **Opciones específicas** para cada test.

Además de disponer de dos tipos de test para realizar:

- **Test estándar**: tipo de test que se referencia por defecto. Ha sido el tipo de test utilizado para las pruebas.

- **Test Omni**: alternativa para realizar *tests* en Netperf. Este tipo de test sustituirá al test estándar en versiones futuras.

TRex y D-ITG tienen disponibles APIs con funciones diferentes. En el caso de TRex, se dispone de una API Python para la automatización de la generación de tráfico *stateless*. Para la generación *stateful*, sin embargo, solo se dispone del fichero de configuración de tráfico YAML que permite seleccionar diferentes parámetros para la generación del tráfico. La API disponible en D-ITG se denomina ITGapi. Es una API C++ que permite el control remoto de la generación de tráfico.

Tanto MGEN como D-ITG disponen de la opción de uso de *scripts* para la generación de tráfico. En MGEN el fichero de *script* se compone de líneas independientes que corresponden a "eventos" que suceden en periodos de tiempo establecidos al principio de cada línea. En D-ITG el enfoque se realiza en tener una línea de código por cada flujo independiente de tráfico.

Ambos tienen sus equivalencias en línea de comandos. En la versión equivalente de una línea de *script* de MGEN se pierde la referencia temporal y cambia la sintaxis. En D-ITG la equivalencia en línea de comandos es casi exacta a una línea de *script* que se introduciría por línea de comandos.

En TRex existe el concepto de simulador. El simulador tiene dos versiones, una para la generación de tráfico *stateful* (bp-sim-64, bp-sim-64-debug) [24] y otra para la generación de tráfico *stateless* (stl-sim) [25]. Con el simulador para la generación *stateful*, TRex simula clientes y servidores generando tráfico entre ellos, es decir, el simulador permite probar antes de ejecutar en el entorno de pruebas los perfiles de tráfico creados. Además, el simulador permite generar un archivo PCAP que contiene el tráfico simulado. En general, el simulador para la generación *stateless* realiza la misma tarea a nivel de paquete, permitiendo probar los perfiles de tráfico creados en Python realizando llamadas a las APIs disponibles.

4.5 Facilidad de uso y complejidad

Algunos programas son más fáciles de usar que otros, incluso siendo más versátiles y con más opciones de uso, como D-ITG. Entre las herramientas más sencillas encontramos a Iperf, que al tener un propósito específico, es más fácil de aprender a usar. Entre las más complicadas se encuentran MGEN y TRex, siendo esta última la más compleja.

Además, para tener una mayor perspectiva, hay que tener en cuenta la complejidad de cada herramienta asociada a su facilidad de uso. Una herramienta simple pero complicada de usar contrasta con una herramienta compleja y fácil de utilizar.

En la tabla 4.7 se ordenan del 1 al 5 en función de la facilidad de uso y la complejidad de la herramienta, siendo "*" la peor en cada parámetro y "*****" la mejor.

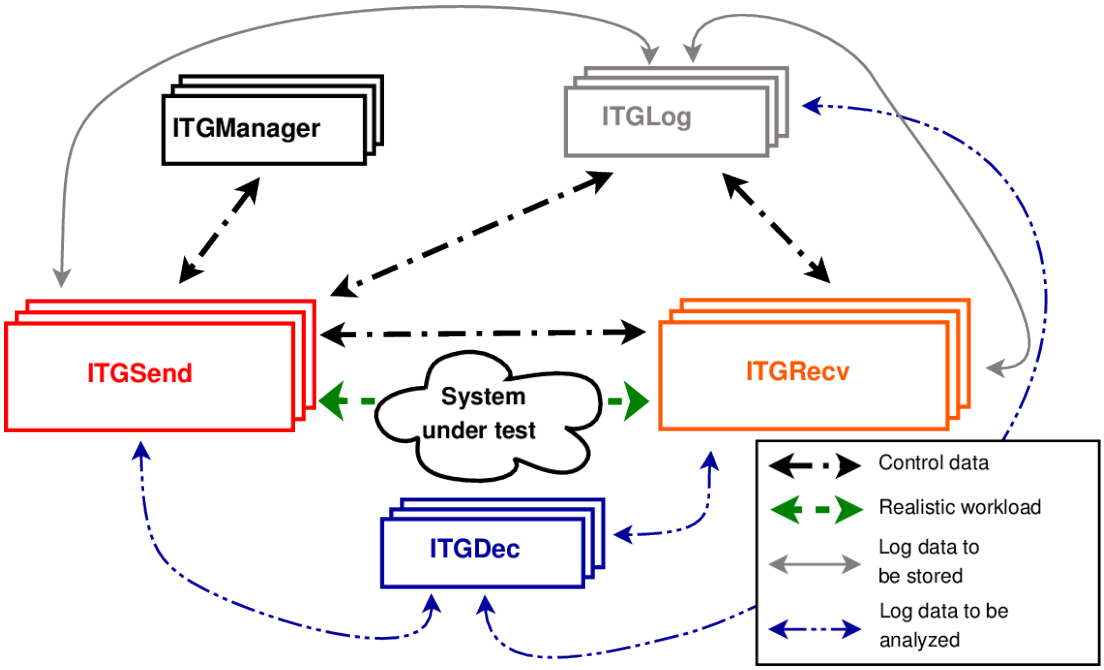

Figura 4.2: Componentes de la arquitectura de D-ITG [1].

Programa	Facilidad de uso	Complejidad de la herramienta
MGEN	**	***
Iperf	*****	****
Netperf	***	**
TRex	*	*
D-ITG	****	****

Tabla 4.4: Comparativa de facilidad de uso y complejidad.

4.6 Gestión de *logs*

Los programas vistos se podrían dividir entre los que se centran exclusivamente en realizar *tests*, produciendo una serie de resultados, y los que también se enfocan en la generación de tráfico, con el objetivo de que este sea lo más realista posible. Debido a estos diferentes enfoques en la generación de tráfico, surgen diferentes formas de gestionar la información que generan.

En D-ITG el enfoque está en una gestión eficiente, centralizada y escalable en la recolección de *logs* mediante una arquitectura que podemos ver en la figura 4.2. En otras herramientas como Netperf, simplemente devuelven un informe sobre el test realizado por su salida estándar. En las herramientas más enfocadas a la generación de tráfico como TRex y MGEN se vio que es posible generar archivos de capturas de tráfico (formato PCAP para MGEN y ERF en TRex) analizables posteriormente en programas como Wireshark. En la figura 4.3 se puede ver el ejemplo de una captura de tráfico ERF abierta en el programa de visualización de tráfico Wireshark.

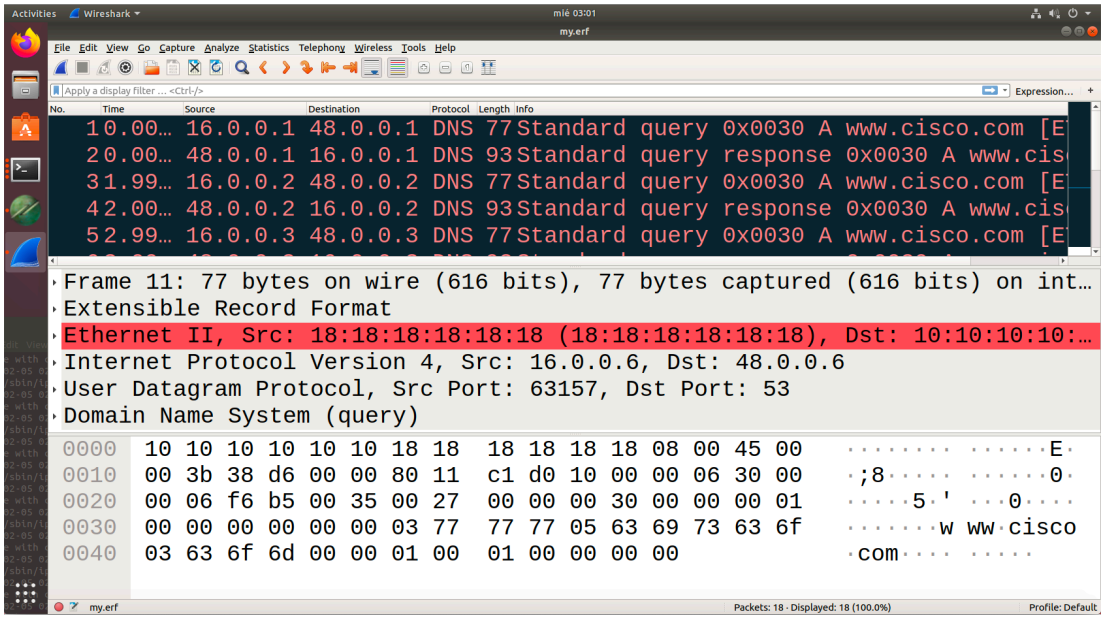

Figura 4.3: Captura de tráfico ERF de TRex visualizado en Wireshark.

La presentación de la información es importante para poder analizar los datos, ya sea visualmente o mediante herramientas de tratamiento de datos. En Iperf, tras ejecutar el cliente hacia el servidor, se genera un informe con los resultados obtenidos del test. Este informe se devuelve por defecto por la salida estándar en la terminal donde se haya ejecutado. Como alternativa se puede devolver el informe en formato de archivo estructurado JSON.

Los *logs* generados en MGEN son diferentes a los *logs* generados en otras herramientas

de este comparativo. Dado que en MGEN se tienen referencias temporales concretas, los *logs* generados se obtienen como registros de una línea con diferentes campos, a diferencia de otras herramientas donde se muestra un informe en varias líneas. En MGEN se pueden guardar los *logs* en 2 formatos:

- ***Log* de texto**: fichero de *log* de texto donde cada línea dentro de él representa un evento.
- ***Log* binario**: un fichero con registros en binario, siguiendo un formato determinado. Su ventaja reside en ocupar menos espacio de almacenamiento. Este tipo de *log* se puede convertir a un *log* de texto ejecutando el comando:

  ```
  mgen convert<ficheroLogBinario>
  ```

En Netperf ocurre lo mismo que en Iperf, la salida del programa se realiza como un informe con las mediciones de los test realizados. Por otro lado, en Netperf no existe ninguna opción alternativa a la salida estándar del programa como salida de los datos. Esta salida se puede modificar con opciones de comando, al contrario que en Iperf, donde no se podía modificar. En caso de usar un test Omni en lugar de un test estándar están disponibles los denominados *Output Selectors*, que permiten personalizar los datos que se muestran en el informe.

En TRex la ejecución del generador en modo *stateful* genera una información de salida muy parecida a la que aporta D-ITG. Esta información de salida es controlable mediante la opción de *verbosity* (opción -v) en tres niveles.

Además, en TRex se puede guardar una captura del tráfico generado en formato ERF, la cual es posible abrir e inspeccionar con programas como Wireshark (Fig. 4.3).

En D-ITG, la ejecución tanto de ITGSend como de ITGRecv permite el guardado de ficheros de *logs* mediante la opción -l. Los *logs* pueden ser enviados por la red a una instancia de ITGLog para un tratamiento posterior. Estos ficheros no son legibles por lo que necesitan ser decodificados mediante el último componente, ITGDec. La salida de ITGDec depende de las opciones especificadas en su ejecución. En general, se saca un informe de múltiples líneas con los datos recogidos en el fichero de *log* codificado proveniente de las instancias ITGSend o ITGRecv. Además con ITGDec es posible exportar los datos por separado a archivos diferentes. Estos archivos se pueden representar en forma de gráfica mediante ITGplot, un *script* de Octave.

4.7 Modificación de *buffers*

La modificación de algunos *buffers* permite adaptar la generación de tráfico a situaciones reales, como cuellos de botella. En la mayoría de los casos se permite modificar el tamaño de los *buffers* de los *sockets* de transmisión y recepción, aunque se han encontrado casos donde

Programa	*Buffers* Modificables
MGEN	Tamaño *buffers sockets* para transmisión/recepción
Iperf	Número de *Buffers* para retransmisión
Netperf	Alineamiento y *offset* de llamadas transmisión/recepción, Pre-llenado *buffers* transmisión mediante archivo
TRex	Número de *buffers* para gestión de paquetes
D-ITG	Número de *Buffers* para gestión de *logs*

Tabla 4.5: Comparativa de modificación de *buffers*.

se permiten modificar *buffers* diferentes (D-ITG y *buffers* de envío y recepción de *logs*) o de diferentes maneras (Netperf y la modificación del offset, el pre-llenado de *buffers* o el número de *buffers* para el "*buffer ring*"). En la tabla 4.5 se pueden ver las características de los *buffers* que se pueden modificar.

MGEN permite la modificación del tamaño de *buffers* tanto de *sockets* de transmisión como de recepción. En la documentación se aclara que existe un tamaño máximo que no es posible superar.

Iperf3 hereda de la versión Iperf2 la modificación del tamaño de *buffers* mediante las opciones de comando -n, -l y -k. Podemos establecer el número de *buffers* a retransmitir (-n) de un tamaño determinado (-l) durante un tiempo determinado (-t).

Netperf permite la modificación de varios tipos de *buffers*. Como se vio anteriormente, Netperf tiene dos tipos de opciones, las opciones globales y las opciones especificadas para cada test. Las opciones respecto a los *buffers* también se dividen entre estos dos tipos de opciones.

Dentro de las opciones globales se puede modificar:

- Alineamiento de *buffers* de llamadas del sistema (opciones -a y -A).
- Offset para los *buffers* de las opciones -a y -A (opciones -o y -O).
- "Pre-llenado" de *buffers* mediante archivo (opción -F).
- Número de *buffers* para el anillo de *buffers* (opción -W): Netperf respecto al resto de las herramientas del comparativo utiliza un anillo de *buffers* que van rotando en vez de recibir o enviar por el mismo.

Para los *tests* TCP, UDP y SCTP en general se pueden modificar los *buffers* de la siguiente manera:

- El tamaño de los *buffers* "pasados" a las llamadas de envíos y recepciones, solo para test de tipo "STREAM" (opciones -m y -M).

- El tamaño de los *buffers* de los *sockets* de envío y recepción (opciones -s y -S).

En TRex el tamaño de *buffers* no se controla mediante opciones de comando, al contrario que en el resto de herramientas. Se realizan los cambios en el fichero de configuración del servidor para los siguientes *buffers* [26]:

- Número de *buffers* de memoria asignados para paquetes en tránsito, por cada par de puertos.

```
1          memory    :
2             mbuf_64      : 16380
3
```

- Número de *buffers* de memoria asignados para guardar la parte del paquete que permanece sin cambios por plantilla.

```
1          memory    :
2             traffic_mbuf_64      : 16380
3
```

  Los números después de "mbuf_" y "traffic_mbuf_" son especificados por cada tamaño de paquete

En D-ITG no es posible modificar el tamaño de los *buffers* de entrada o de salida, como en otras herramientas. Sin embargo sí es posible modificar los *buffers* para la gestión de los *logs* que se envían desde ITGSend/ITGRecv hacia ITGLog, además de los *buffers* de recepción de este último.

4.8 Licencias de uso

Cada programa está bajo una licencia diferente, más o menos permisiva con el uso privativo de *software* derivado de este [27] [28][29] [30]. De todos los programas, destaca por ser el más restrictivo D-ITG, con su licencia GPLv3. Esta licencia establece que todo *software* derivado debe seguir estando bajo la misma licencia, y por tanto, ser totalmente libre [31]. El resto de programas cuentan con licencias mucho más permisivas, alejadas del concepto del *copyleft*.

En la tabla 4.6 podemos ver las licencias de cada programa del comparativo.

4.9 Mantenimiento

Respecto al mantenimiento de los proyectos vistos, se ha podido apreciar que no todas las herramientas se encontraban en repositorios públicos que permitieran realizar un seguimien-

Programa	Licencia
MGEN	NRL CODE 5520
Iperf	BSD 3 Cláusulas
Netperf	BSD 2 Cláusulas
TRex	Apache 2.0
D-ITG	GPLv3

Tabla 4.6: Comparativa de licencias de uso.

to del desarrollo del proyecto. Además, algunas de ellas llevan sin actualizaciones desde hace mucho tiempo, pero al menos funcionan en el entorno de pruebas elegido.

Como se comentó en la sección 4.2 surgieron bastantes problemas con MGEN debido a una gestión confusa de los archivos fuente en el servidor FTP donde estaban alojados antes. El punto positivo es que al final se migró el proyecto a Github, con lo que están disponibles ahora los canales de comunicación de esta plataforma de repositorios para toda comunidad alrededor de este proyecto. Desde dicha migración se puede apreciar que ha mejorado el mantenimiento.

Tras aclarar la diferencia entre las páginas oficiales de Iperf en la sección 4.2, Iperf goza de un buen mantenimiento por parte de EsNet y el resto de colaboradores, como se puede ver en el repositorio de Github [15] donde está alojado el proyecto.

Netperf lleva muchos años con su proyecto en Github [32] aunque ha dejado de actualizarse el repositorio desde hace un tiempo, con lo cual se aprecia cierto abandono.

TRex es de las herramientas con mayor soporte y comunidad del comparativo. Tiene una gran actividad en su repositorio de Github [33] y está actualizado constantemente.

D-ITG no dispone de un repositorio público para realizar un seguimiento del proyecto, además de que la herramienta lleva sin actualizarse desde el año 2013. Es la herrramienta del comparativo que lleva más tiempo sin actualizar [34].

En la tabla 4.7 se ordenan del uno al cinco siendo "*" 1a herramienta con el peor mantenimiento y "*****" la que tiene mejor mantenimiento.

Programa	Mantenimiento
MGEN	*
Iperf	****
Netperf	***
TRex	*****
D-ITG	**

Tabla 4.7: Comparativa del mantenimiento de las herramientas.

4.10 Dirección del tráfico generado

Al estudiar las diferentes herramientas se ha apreciado que no todas permiten la generación de tráfico bidireccional.

A la hora de hablar de tráfico bidireccional se tiene que distinguir entre tráfico unidireccional y bidireccional, suponiendo que tenemos dos hosts A y B [35]:

- **Flujo unidireccional**: tráfico de A $\rightarrow$ B o B $\rightarrow$ A.
- **Flujo bidireccional**: tráfico de A $\rightarrow$ B y B $\rightarrow$ A.

En Iperf en la versión 2 se disponía de una opción para realizar *tests* bidireccionales. Un test unidireccional se realiza de un punto a otro y el servidor envía de vuelta solo los resultados. En la versión actual de Iperf 3.7 se ha vuelto a incluir la generación de tráfico bidireccional. En un test de tráfico bidireccional tanto el cliente como el servidor envían y reciben tráfico.

En la tabla 4.8 se muestra la dirección del tráfico generado respecto a los diferentes programas.

Programa	Dirección del tráfico generado
MGEN	Unidireccional
Iperf	Unidireccional y bidireccional (v2 y >v3.7)
Netperf	Unidireccional y bidireccional (test simultáneos, TCP_RR)
TRex	Unidireccional y bidireccional (*stateless* con API Python)
D-ITG	Unidireccional

Tabla 4.8: Comparativa de dirección del tráfico generado.

En Netperf se carece de una opción nativa para la generación de tráfico bidireccional, sin embargo, en la documentación se nos muestran dos técnicas diferentes para emular esta característica:

- Realizando dos *tests* unidireccionales simultáneos en sentidos contrarios.
- Mediante un test TCP_RR (Request/Response).

En TRex se ha encontrado una referencia en el modo *stateless* a la generación de tráfico bidireccional. Esta referencia se encuentra en el desarrollo de programas basados en la API donde aparece esta opción [36].

Tanto MGEN como D-ITG carecen de capacidades de generación bidireccional.

4.11 Plataformas soportadas

La mayoría de los programas del comparativo son multiplataforma, a excepción de TRex que solo está soportado en Linux. En la tabla 4.9 se muestra la información aportada por las herramientas respecto a las plataformas que soportan.

Programa	Plataformas Soportadas
MGEN	Basados en UNIX (MacOS X incluído) y WIN32
Iperf	Windows, Linux, Android, MacOS X, FreeBSD, OpenBSD, NetBSD, VxWorks, Solaris,...
Netperf	UNIX, Linux, Windows,...
TRex	Linux
D-ITG	Linux, Windows (XP, Vista, 7), MacOS X (Leopard), FreeBSD

Tabla 4.9: Comparativa de plataformas soportadas.

4.12 Tipos de tráfico generado

Entre las herramientas comparadas se han visto distintos planteamientos. En la tabla 4.10 se muestran los tipos de tráfico más generales encontrados. Entre estos tipos se encuentran:

- ***Tests* predefinidos bajo distintos protocolos**: extraen mediciones sobre el tráfico generado.
- **Generación de tráfico realista basada en procesos estocásticos**: se utilizan procesos estocásticos que manejan el tamaño y el intervalo de envío de paquetes, los vamos a denominar PS (Packet Size) e IDT (Inter-Departure Time) respectivamente de sus siglas en inglés.
- **Generación de tráfico realista basada en una captura de tráfico real**: mediante un fichero PCAP se extraen los datos del tamaño y el intervalo de envío de paquetes.
- **Generación de tráfico realista basada en una captura de tráfico real**: mediante un fichero PCAP se extraen los datos del tamaño y el espacio entre paquetes (IPG, Inter-Packet Gap).
- **Generación de tráfico sin estado**: se diseña un paquete desde una plantilla y se realiza un envío de dicho paquete bajo diferentes patrones.

En MGEN, en el apartado de la documentación denominado Pattern, se muestra la información de los tipos de tráfico que genera. Estos patrones son, en su mayoría, distribuciones

Programa	*Tests*	Proc. Estocásticos	PCAP (IDT/PS)	PCAP (IPG)	*Stateless*
MGEN	X	X	X		
Iperf	X				
Netperf	X				
TRex	X			X	X
D-ITG	X	X	X		

Tabla 4.10: Comparativa de tipos de generación de tráfico.

estadísticas (Poisson, uniforme...) o simplemente aleatorias, todas ellas modificando el tamaño de los paquetes (PS) y/o el intervalo de transmisión de estos (IDT). Es posible extraer datos sobre el tamaño de los paquetes y el intervalo de transmisión mediante la carga de archivos de Tcpdump con el patrón CLONE. De esta manera en MGEN se puede emular tráfico de capturas reales.

En Iperf se realizan mediciones de tráfico sobre los protocolos UDP, TCP y SCTP. Dependiendo del protocolo utilizado se obtienen resultados sobre diferentes métricas.

En Netperf básicamente se permiten dos tipos de test: los que miden la transferencia masiva de datos, denominados *STREAM* y los que miden las transacciones por segundo, denominados *Request/Response* o RR. En todos estos *tests* existen numerosas opciones para medir el impacto de los *tests* en el consumo de CPU. Además del uso de CPU, se introduce una métrica denominada *Service Demand*. Esta métrica representa la normalización del uso de CPU y el trabajo realizado. Es una medida de eficiencia, siendo su número cuanto más bajo, mejor.

TRex en el modo *stateful* basa su generación de tráfico en el uso de perfiles de tráfico YAML que utilizan un fichero PCAP. Este perfil de tráfico permite definir diferentes aspectos que se aplicarán a la captura de tráfico PCAP. Entre estas características destaca el uso del *Inter-packet Gap* (IPG) o pausa entre paquetes. Sin entrar demasiado a estudiar esta característica, en el perfil de tráfico YAML es posible utilizar el IPG de la captura de tráfico PCAP o establecer un IPG determinado.

En el modo *stateless* de TRex se genera tráfico a más bajo nivel que en el modo *stateful*. Si la generación *stateful* estaba enfocada en la capa de aplicación, en el modo *stateless* está más enfocada en la capa de transporte. En este modo se crea un paquete con Scapy y se modifican sus campos mediante el denominado *Field Engine* [36]. Después se define un perfil de tráfico que permite enviar este paquete de forma continua, en ráfaga o multi-ráfaga. Estos modos de envío se basan en la modificación del *Inter-Stream Gap*.

D-ITG junta varios de los planteamientos anteriores. Por un lado permite realizar *tests* como los vistos en Iperf y Netperf. Por otro lado, permite generar tráfico mediante perfiles predefinidos de tráfico de capa de aplicación utilizando procesos estocásticos y adicionalmente mediante un fichero con los datos de intervalo de tiempo entre envíos y tamaño de paquete.

Capítulo 5

Plataforma de monitorización basada en la pila ELK

En este capítulo se hablará de la investigación y desarrollo de una plataforma de monitorización de red utilizando como datos de entrada los *logs* de algunas herramientas que se han comparado en el capítulo anterior.

5.1 Introducción

El planteamiento de esta plataforma surge de la necesidad de estructurar y visualizar los datos que aportan las herramientas de generación/monitorización de red vistas en el capítulo anterior.

En este trabajo se aporta una solución conceptual, basada en los conocimientos básicos de redes y programación aprendidos en el grado, junto con el aprendizaje e investigación de una tecnología y paradigma nuevos.

Se han escogido 3 herramientas analizadas anteriormente para realizar las pruebas y la posterior visualización de los datos recogidos: Iperf, Netperf y D-ITG. Las tres están enfocadas en la realización de tests con recolección de mediciones y presentación en forma de informes de datos. De los tres programas es posible extraer *logs* de texto de cara a su transformación en Logstash y visualización en Kibana.

5.2 Estudio y aprendizaje de la pila ELK

El aprendizaje de ELK se ha realizado de forma incremental, empezando por la capa más baja progresando a la capa más alta. La curva de dificultad es pronunciada y requiere de un estudio a base de prueba y error dado que los ejemplos proporcionados por la documentación de ELK no suelen aportar demasiado. Se ha visto que el foro de dudas de Elastic [37] es muy

útil de cara a buscar problemas parecidos a los planteados para realizar configuraciones en esta plataforma.

El problema fundamental que agudiza la curva de dificultad en este trabajo con ELK es el poco parecido del caso de uso que se plantea para esta plataforma con las soluciones existentes para analizar *logs* de texto multilínea. Para los casos de uso y aplicaciones más utilizadas existen los denominados *plugins* [38] que realizan las tareas de *parseo* e identificación de datos de los *logs* de forma transparente al usuario.

El reto está en conseguir encontrar diferentes soluciones a los nuevos problemas planteados por estos *logs* complejos de analizar.

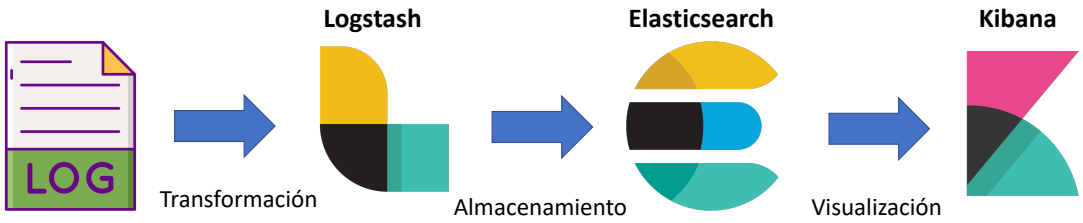

Figura 5.1: Flujo de trabajo de ELK.

5.3 Diseño de las pruebas

Las pruebas se han diseñado con el objetivo de poder visualizar métricas diferentes por cada herramienta elegida.

- **Iperf**: recoger medidas del ancho de banda de tráfico TCP.
- **Netperf**: recoger medidas de la tasa de transferencia efectiva (*throughput*), junto con la medida del impacto del test en el rendimiento del equipo donde se ejecuta mediante el uso de CPU y el *service demand*.
- **D-ITG**: mostrar la medición de rendimiento de tráfico TCP como la pérdida de paquetes, la latencia o el *jitter*.

De cara a conseguir resultados interesantes se debe disponer de un entorno real de pruebas o, en su defecto, poder emularlo mediante *software*.

5.3.1 Entorno de pruebas

Las pruebas se realizan en el programa CORE, un emulador de red en tiempo real. El programa se ejecuta en una máquina virtual Ubuntu 18.04, donde se han instalado todos los

programas del comparativo. CORE permite emular dispositivos de red con sus conexiones. Se utilizará una topología simple formada por tres redes:

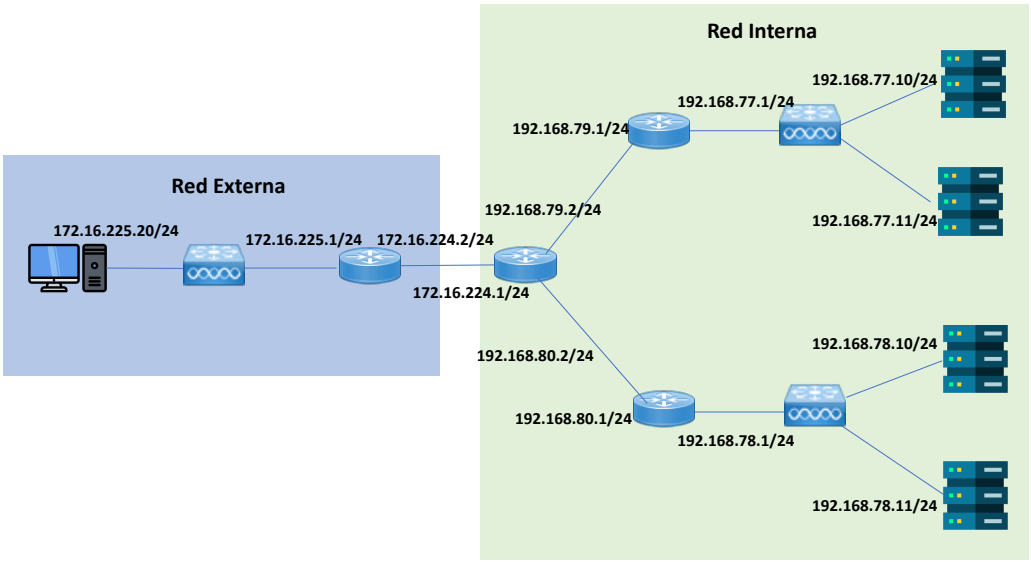

Figura 5.2: Esquema de red de CORE.

- **Redes internas**: dos redes que vamos a denominar como redes dentro de la organización.
- **Red externa**: una tercera representará una red externa al sistema bajo estudio, desde donde se lanzarán las pruebas en remoto.

Se pretende que desde una máquina en la red externa se lancen comandos remotos para que una máquina de la red interna lance un test a otra de la misma red. Los enlaces de CORE se pueden modificar para emular condiciones diferentes donde aumente la latencia, el *jitter* o la pérdida de paquetes, entre otros parámetros. De esta manera se intenta que puedan cambiar los resultados de los tests realizados y se pueda apreciar el cambio de estos valores en tiempo real en Kibana.

5.3.2 Desarrollo de tests de las herramientas de generación de gráfico de red

Los test que se han implementado en el *script* son los siguientes:

- **Test TCP de Iperf.**
- **Test TCP_RR de Netperf**.

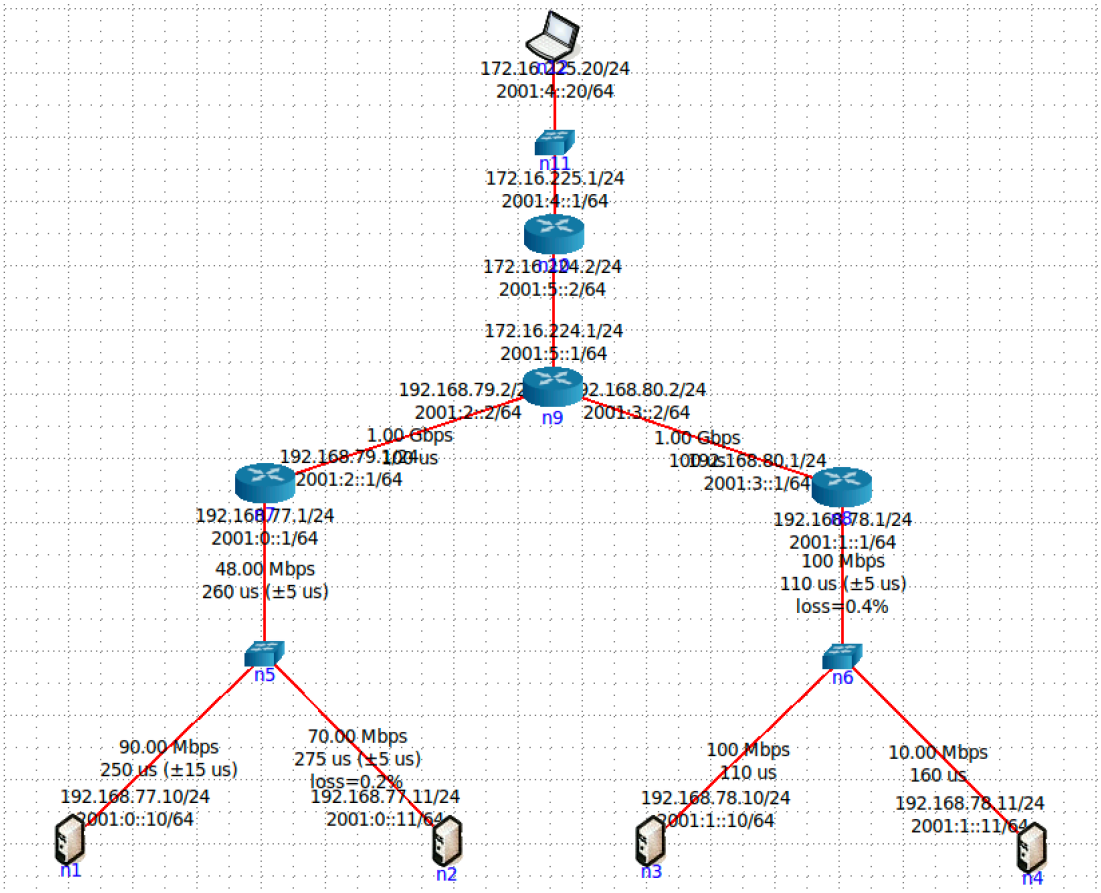

Figura 5.3: Red emulada en CORE.

- **Test TCP de D-ITG.**

El test de Iperf genera tráfico TCP entre la máquina cliente (`iperf -c`) y la máquina que ejecuta en segundo plano el servidor (`iperf -s`). Este test aporta diferentes datos en la salida estándar de la terminal donde se ejecuta, como el ancho de banda y el tamaño de la ventana de congestión TCP.

El comando lanzado desde el cliente es el siguiente:

```
iperf3 -c $REMOTE_IP -i 1 -t 5 -O 1
```

El test de Netperf petición/respuesta permite medir las transacciones por segundo que se realizan entre dos máquinas conectadas. En la documentación de Netperf se define una transacción como el "intercambio completo de una solicitud y una respuesta" [39]. De la misma manera que en Iperf, Netperf necesita de la ejecución de una instancia de servidor (`netserver`) en la máquina destino del test.

El comando lanzado desde el cliente es el siguiente:

```
netperf -T 1 -L $LOCAL_IP -H $REMOTE_IP -t TCP_RR -c -C
```

En D-ITG se ejecuta un test TCP del que se pueden extraer una gran cantidad de datos de los *logs* generados de la instancia equivalente a la que actúa como servidor en los dos test anteriores, ITGRecv. Se ha encontrado que los *logs* de la instancia ITGSend no aportaban los datos relacionados con la latencia y *jitter* entre otros. También en contraste con los otros dos tests, en D-ITG se deben procesar los *logs* con el decodificador ITGDec.

En el servidor se ejecuta el comando para generar los archivos de *log*:

```
ITGRecv -l temprecv.log
```

En el cliente se ejecuta el test:

```
ITGSend -a $REMOTE_IP -T tcp -c 1500 -C 80000 -t 6000
```

Y por último se decodifican los *logs* del servidor:

```
ITGDec temprecv.log
```

Tanto en este test como en los siguientes la salida estándar se redirige a un fichero en una carpeta compartida entre la máquina virtual y el *host*. Se ha decidido hacerlo de esta manera por razones de simplicidad siendo la prioridad el tratamiento de estos datos dentro de la pila ELK. En los dos primeros tests se redirige la salida generada por el comando cliente (iperf -c {...} y netperf {...}), en el test de D-ITG se redirige la salida del decodificador ITGDec.

Por ejemplo, en Iperf la salida se redirige de esta manera:

```
{...}  > $PATH_SALIDA/iperf/iperf$(date +%s).log
```

Se genera un fichero con un *timestamp epoch* en el nombre teniendo en cuenta hasta los segundos [40]. De esta manera nos aseguramos de que no se repitan los nombres de ficheros.

5.3.3 Desarrollo de *scripts* para la realización de pruebas

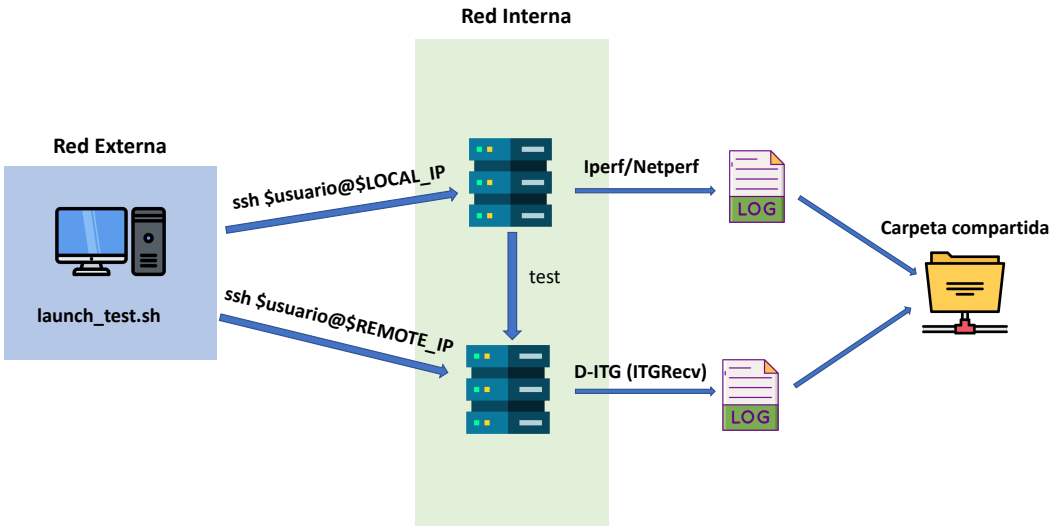

Figura 5.4: Esquema de ejecución de los test implementados.

Se aprovechará el uso generalizado por línea de comandos de estas herramientas para implementar dos *scripts* en Bash que ejecuten estas herramientas pasando por parámetros campos comunes a los tres test. Se ha preferido Bash frente a Python por simplicidad, dado que no se necesitaba de la potencia de Python en este caso.

- **launch_test.sh**: se conecta en remoto a una máquina y ejecuta desde esta el test especificado hacia una máquina destino. Existen tres parámetros que se deben especificar obligatoriamente:
  - **Tipo de test (-t)**: test que se va a realizar, con una nomenclatura que permita identificar el programa a ejecutar junto con el tipo de test que ejecuta (e.g. netperf_rrtcp).
  - **IP local (-l)**: dirección IP de la máquina desde donde se lanza el test.
  - **IP remota (-r)**: dirección IP de la máquina hacia donde se lanza el test.
- **loop.sh**: realiza un bucle del *script* anterior con un número de ejecuciones determinadas recibiendo los mismos parámetros.

Script launch_test.sh

Este *script* lanza de forma remota el comando para que desde la máquina con la "IP local" se lance un test "Tipo de test" a la máquina con "IP remota". Las tres herramientas elegidas funcionan de la misma manera, lanzando un test hacia una IP destino.

En este caso se ha utilizado una simplificación de la conexión remota SSH por la dificultad de configurar las claves públicas en la simulación de CORE. Tanto esto como la contraseña trivial utilizada en la máquina virtual, se han realizado de forma consciente de que en todo momento se está realizando un trabajo bajo un ámbito de investigación, y no como un producto de *software*.

```
SERVER_PID=$(sshpass -p 1234 ssh $usuario@$REMOTE_IP "sh -c
'nohup netserver > /dev/null 2>&1 & echo \$!'")
```

Se ha pensado este *script* para que se puedan añadir más tests al estar bajo una cláusula CASE. En el anexo B.1 se encuentra este *script* completo.

Script loop.sh

El segundo *script* automatiza la repetición de tests en bucle un número de veces determinado. Es de gran ayuda para poder crear una gran cantidad de datos ejecutando los tests múltiples veces. De esta manera se puede tener una muestra de datos que aporte información suficiente y en tiempo real.

```
# BUCLE
for ((i = 1 ; i <= $ITERATIONS ; i++));do
  echo "Counter: $i"
  #Tiempo entre iteraciones
  sleep $SLEEP_TIME
  # Ejecución del scrpt
  $PATH_SCRIPT -t $TEST_TYPE -l $LOCAL_IP -r $REMOTE_IP
```

En el anexo B.2 se encuentra este *script* completo.

5.4 Ingesta de Datos con Logstash

Como se ha visto anteriormente en el apartado 5.3.2, se han recogido una serie de archivos que contienen los datos de los tests realizados. Se ha elegido que cada archivo recoja los datos de un test individual para evitar conflictos en la ingesta de datos.

Logstash permite recoger estos archivos y transformarlos en datos estructurados que pueden ser almacenados en Elasticsearch, siguiendo la pila ELK de forma ascendente.

Cada test necesita una configuración de Logstash diferente para *ingestar* los datos, dado que no es viable, ni tampoco legible, una configuración que sea apta para más de un caso de uso. Por tanto, se han desarrollado tres configuraciones:

- **iperf-tcp.conf**.
- **netperf-tcp.conf**.
- **ditg-receiver.conf**.

Aunque en principio se trabajó en una configuración para los *logs* de ITGSend, se cambió posteriormente por *logs* de ITGRecv ya que, como se comentó en el apartado 5.3.2, son los que aportan todos los datos del test referentes a la latencia, *jitter* etc.

Dado que no es viable parar y lanzar Logstash cada vez que queramos cambiar de configuración, se estudió la posibilidad de usar todas las configuraciones de forma simultánea. Esto es posible gracias al propio diseño de Logstash, que nos permite lanzar diferentes configuraciones a la vez para un mismo servidor de Logstash mediante lo que se denominan *pipelines*.

La configuración de las *pipelines* [41] se realiza en el archivo pipelines.yml de la carpeta /config de Logstash. En este archivo se determina el *id* de la *pipeline* (un nombre identificativo), el *path* hacia donde se encuentra la configuración de Logstash que se quiere asignar a esa *pipeline* y el tipo de cola que va a usar para encadenar las peticiones. El tipo de cola define cómo de persistentes son los datos frente a un error. Lo recomendado es usar una cola persistente en vez de una cola en memoria (por defecto) [42].

Contenido del archivo pipelines.yml:

```
- pipeline.id: iperf
  path.config: "{...}/iperf-tcp.conf"
  queue.type: persisted
- pipeline.id: netperf
  path.config: "{...}/netperf-tcp.conf"
  queue.type: persisted
- pipeline.id: ditg-receiver
  path.config: "{...}/ditg-receiver.conf"
  queue.type: persisted
```

Los ficheros de configuración de Logstash siguen un formato determinado con tres partes diferenciadas:

- ***Input***: entrada de datos, definición de como entran.
- ***Filter***: filtros que modifican, añaden o eliminan información.
- ***Output***: salida de los datos.

A continuación se explicarán cada una de las partes mencionadas.

5.4.1 Configuración de entrada de datos en Logstash

La entrada de datos se define en el apartado *Input* [43] de la configuración de Logstash. Se ha utilizado una entrada de datos mediante ficheros [44] dentro un *path* determinado. En la definición del *path*, para que se busquen archivos que sigan un patrón determinado se ha utilizado el comodín "*", de forma que los archivos que comiencen por la palabra antes del asterisco son los tenidos en cuenta por Logstash.

Anteriormente (5.4) se ha hablado de la decisión de dedicar un archivo independiente para los datos de cada test, aludiendo a un posible conflicto. Este posible conflicto puede ser provocado por el archivo *sincedb*, que lleva un seguimiento de la posición de cada archivo[45]. De esta forma, Logstash puede seguir la búsqueda dentro de los ficheros, si estos cambian desde la última vez que se "visitaron". Sin embargo, en la práctica se ha visto que esta característica no funciona siempre bien con *logs* multilínea. Por este motivo, en lugar de utilizar un solo archivo y sobreescribirlo se ha decidido dedicar un archivo por cada test.

Contenido del fichero de configuración de la parte de entrada de datos:

```
1   input {
2     #Input mediante file.
3     file{
4       #Se buscan todos los archivos que comienzan por ditg-log
5       path => ".../ditgrcv*.log"
6       start_position => "beginning"
7       codec => multiline {
8         #El patrón pretende descartar los datos totales
9         pattern => "^\*+  TOTAL RESULTS   \*+$"
10        what => "previous"
11        negate => true
12      }
13    }
14  }
```

Logstash trata cada línea de estos ficheros de *log* como un mensaje. Este mensaje sea modificado en la parte *Filter* o no, pasa a ser en la salida un "evento". Un evento tiene forma de documento JSON con campos estructurados entre llaves. Contiene el mensaje y campos de metadatos, entre ellos el *timestamp* de cuando se ha creado.

5.4.2 Estudio y transformación de datos de entrada mediante filtros

Se ha visto como Logstash explora el sistema de ficheros siguiendo un criterio establecido por el usuario en el fichero configuración. En la parte *Filter* [46] los mensajes se pueden transformar mediante filtros que modifican, añaden o eliminan información.

Para convertir los mensajes anteriores a eventos se han transformado los datos adecua-

damente. El objetivo, para este caso de uso, es conseguir extraer la información en campos parseando el texto del mensaje.

Sin embargo, en primer lugar se requiere de un estudio del formato de los ficheros de *log* para tener clara la estrategia a seguir en el apartado de filtros.

Estudio de los ficheros de *log*

Estos ficheros, *logs* de texto con múltiples líneas, constituyen un caso de uso complicado para Logstash. Los casos de uso más comunes consisten en *logs* de una sola línea que empiezan con una información temporal como un *timestamp*. Esto hace que sea mucho más sencillo manejarse con ellos, ya que incluso en muchos casos, ya se incluyen soluciones ad hoc que no requieren ni de un planteamiento ni de un estudio para implementarlas [46] [47].

En este caso, dentro de estas líneas se mezcla información relevante con información poco útil. Son datos no estructurados y hay que hacer que lo sean, extrayendo además solo la información importante.

Para ello se han estudiado dos aproximaciones para analizar todos estos datos:

- La primera consiste en tratar cada línea como un mensaje separado que tiene que analizar el servidor.
- La segunda consiste en juntar las líneas del fichero en un único mensaje, aportando un patrón determinado para seleccionar que líneas formarán del conjunto.

La primera aproximación se ha utilizado donde los *logs* aportaban datos en una serie temporal, como los de Iperf:

```
Connecting to host 192.168.77.11, port 5201
[  4] local 192.168.77.10 port 48816 connected to 192.168.77.11
    ↪ port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec  8.43 MBytes  70.7 Mbits/sec   16   43.8
    ↪ KBytes       (omitted)
[  4]   0.00-1.00   sec  8.02 MBytes  67.2 Mbits/sec   12   38.2
    ↪ KBytes
[  4]   1.00-2.00   sec  7.95 MBytes  66.7 Mbits/sec   14   18.4
    ↪ KBytes
[  4]   2.00-3.00   sec  7.95 MBytes  66.7 Mbits/sec   15   46.7
    ↪ KBytes
[  4]   3.00-4.00   sec  8.02 MBytes  67.2 Mbits/sec   10   21.2
    ↪ KBytes
[  4]   4.00-5.00   sec  7.95 MBytes  66.7 Mbits/sec    9   35.4
    ↪ KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
```

```
[  4]   0.00-5.00   sec  39.9 MBytes  66.9 Mbits/sec   60
    ↪  sender
[  4]   0.00-5.00   sec  40.2 MBytes  67.5 Mbits/sec
    ↪  receiver

iperf Done.
```

También se ha utilizado en los *logs* de Netperf por constar de dos líneas de datos bien diferenciadas.

```
MIGRATED TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0
    ↪ AF_INET to 192.168.77.11 () port 0 AF_INET : demo : first
    ↪ burst 0 : cpu bind
Local /Remote
Socket Size   Request Resp.  Elapsed Trans.   CPU    CPU    S.dem
    ↪ S.dem
Send   Recv   Size    Size   Time    Rate     local  remote local
    ↪ remote
bytes  bytes  bytes   bytes  secs.   per sec  % S    % S    us/Tr
    ↪ us/Tr

16384  131072 1       1      10.03   66.52   0.65   0.65   391.941
    ↪ 391.896
16384  131072
```

La segunda aproximación se ha utilizado en los *logs* de D-ITG, donde teníamos muchos datos simples en diferentes líneas e interesaba juntarlos todos.

```
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
------------------------------------------------------------
Flow number: 1
From 192.168.77.10:47712
To   192.168.77.11:8999
------------------------------------------------------------
Total time                =      6.041060 s
Total packets             =         33743
Minimum delay             =      0.000318 s
Maximum delay             =      0.077858 s
Average delay             =      0.058020 s
Average jitter            =      0.000267 s
Delay standard deviation  =      0.010020 s
Bytes received            =      50614500
Average bitrate           =  67027.309777 Kbit/s
Average packet rate       =   5585.609148 pkt/s
Packets dropped           =             0 (0.00 %)
Average loss-burst size   =      0.000000 pkt
```

```
20 ----------------------------------------------------------
```

Desarrollo e implementación de filtros

El filtro más útil para *logs* de texto plano es el filtro Grok [48]. Este filtro compara el mensaje con un patrón compuesto de expresiones regulares pudiendo guardar, en caso de coincidir el patrón con el mensaje, los campos especificados como campos del evento resultante. Para un manejo más sencillo, Logstash tiene una serie de abreviaturas de las expresiones regulares más comunes [49].

Sin embargo, al no existir un patrón predeterminado para un número de tipo *float* (e.g. 7.4), se ha creado un patrón personalizado, el cual tiene que ser depositado en una carpeta determinada para acceder a él dentro de la configuración.

Se ha creado un archivo denominado *custom* con el patrón personalizado:

```
1 FLOAT \s*-?\d+(\.\d+)?
```

En el filtro Grok se establece el *path* del archivo de patrones personalizados:

```
1 grok{
2       # Patrones creados por el desarrollador
3        patterns_dir => ["/etc/patterns/custom"]
4          {...}
5     }
```

De cara a poder desarrollar el patrón para el mensaje en el filtro Grok y para desarrollar la abreviatura personalizada FLOAT se han utilizado dos herramientas:

- **Grok debugger de Kibana**: herramienta interna de Kibana para poder probar patrones para el filtro Grok. Al principio, al desconocer esta herramienta, se empezó a utilizar una similar *online* [50], sin embargo, fallaba bastante al no poder reconocer las abreviaturas que, en teoría, tendrían que funcionar.
- **RegExr**: herramienta online para aprender, realizar y probar expresiones regulares, se ha utilizado sobre todo para crear el patrón personalizado FLOAT [51].

Ejemplo de patrón Grok:

```
1 grok{
2         {...}
3       # Match si el mensaje sigue este patrón
4        match => { "message" =>
5        "%{GREEDYDATA}%{NUMBER:id}%{GREEDYDATA}%{IP:local_ip:ip}
6         %{GREEDYDATA}%{NUMBER:local_port:int}
7         %{GREEDYDATA}%{IP:remote_ip:ip}%{GREEDYDATA}
```

```
8           %{NUMBER:remote_port}"}
9           {...}
10      }
```

Respecto a la primera solución aplicada a Iperf y Netperf, los datos de la IP local e IP remota del test se encuentran separados del resto de datos, en líneas diferentes.

Para este problema se estudió una manera de "pegar" los mensajes que coincidían con los dos patrones utilizados. Por un lado, se clasifica el mensaje que corresponde con los datos de las direcciones origen y destino del test. Por otro, se clasifican los datos correspondientes a los resultados del test.

En la figura 5.5 se puede apreciar un resumen del proceso que se sigue desde el fichero de *log* de texto hasta un evento listo para indexar en Elasticsearch. En la imagen se puede apreciar como los eventos no deseados son eliminados para quedarnos con los datos importantes.

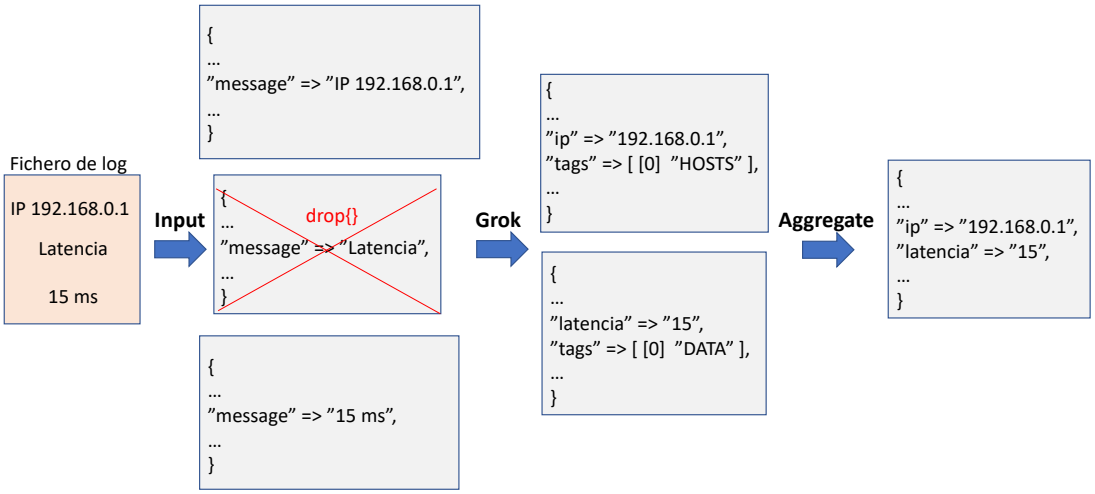

Figura 5.5: Esquema de la transformación y procesamiento de datos mediante filtros.

La clasificación se realiza añadiendo un *tag* [52] a los eventos con sus filtros correspondientes. Por un lado, se parsea el mensaje con el filtro Grok en el apartado de *match* [53], y por otro, se añade la etiqueta al evento y más tarde, dependiendo del *tag* que contienen, se realiza una acción u otra. En este caso se han usado dos etiquetas diferentes:

- `[HOSTS]` para los eventos con información de direcciones IP y/o puertos origen y destino.

```
1  if "local" in [message] {
2    grok{
```

```
    # Patrones creados por el desarrollador
     patterns_dir => ["/etc/patterns/custom"]
    # Match si el mensaje sigue este patrón
     match => { "message" => {...} "}
    # Etiqueta identificativa de cara su clasificacion
     add_tag => ["HOSTS"]
  }
```

- [DATA] para los eventos con información sobre mediciones del test.

```
} else if "(omitted)" not in [message] {
  grok{
    {...}
    # Etiqueta identificativa de cara su clasificacion
    add_tag => ["DATA"]
    #Añadidos los campos de información que vamos a copiar como
  campos vacíos
    add_field => {
      "local_ip" => ""
      "local_port" => ""
      "remote_ip" => ""
      "remote_port" => ""
    }
  }
} else {
  drop{}
}

```

En el caso de Iperf se tiene que copiar el evento que tiene la información de HOSTS en los eventos con la información de DATA. En Netperf, dado que se produce solo un evento clasificado como DATA, consistiría en juntar estos dos eventos solamente.

Para esto, se ha utilizado el filtro *Aggregate*. Que, según dice la información de la documentación de Logstash [54]: "el objetivo de este filtro es agregar la información disponible entre varios eventos (típicamente líneas de registro) que pertenecen a una misma tarea y finalmente insertar la información agregada en el evento final de la tarea".

En la figura 5.6 se puede ver el proceso del filtro *Aggregate* y el uso del mapeado para juntar la información de líneas separadas.

Llegar a la solución planteada en este trabajo no es totalmente evidente, ya que el filtro *Aggregate* en sus ejemplos se muestra como si fuera solo apto para juntar información de líneas parecidas. En este caso se quieren juntar líneas totalmente diferentes y separadas, por lo que se necesita de la característica del mapeado de este filtro implementado de una forma

Figura 5.6: Esquema de uso del mapeado en el filtro *Aggregate*.

diferente. Tras una investigación de soluciones planteadas en los foros de dudas de Elastic, [55][56] se llegó a esta solución final utilizando el mapa y la API Event.

La API *Event* [57] permite manejar la información de los campos de un evento. Esta API se utilizará para copiar y modificar la información de los eventos en el mapa.

El mapa se crea [58] en los eventos de tipo `[HOSTS]` copiando la información del evento en el mapa.

```
if "HOSTS" in [tags] {
  aggregate{
    task_id => "%{id}"
    #Se copian en el mapa todos los campos del evento
    code => "
         map['local_ip'] = event.get('local_ip')
         map['local_port'] = event.get('local_port')
         map['remote_ip'] = event.get('remote_ip')
         map['remote_port'] = event.get('remote_port')
    "
    #Crear si no existe y actualizar en otro caso
    map_action => "create_or_update"
  }
```

Luego, en los eventos de tipo `[DATA]`, habiendo añadido previamente los mismos campos vacíos, se sobreescriben los campos del evento vacíos con la información del mapa.

```
} else if "DATA" in [tags] {
```

```
    aggregate{
      task_id => "%{id}"
      #Copiar el mapa en los campos añadidos anteriormente
      code => "
           event.set('local_ip', map['local_ip'])
           event.set('local_port', map['local_port'])
           event.set('remote_ip', map['remote_ip'])
           event.set('remote_port', map['remote_port'])
      "
      #Actualizar
      map_action => "update"
      #Podemos lanzar el mapa como un nuevo evento.
      #push_previous_map_as_event => false
      #Tiempo que dura el mapa, este tiempo que se ha establecido
    es igual al por defecto
      #Se ha dejado a modo de aclaración
      timeout => 1800
```

Si los campos del mapa coinciden con los del evento que pasa por el filtro aggregate, se pueden sobreescribir los campos del mapa con los del evento, en caso contrario se informará de un error como un tag en el evento que ha pasado por el filtro.

El filtro agreggate tiene una desventaja, ya que está considerado un filtro que reduce bastante el rendimiento de Logstash. Incluso se recomienda utilizar solo un *worker* al utilizar una configuración que tenga este filtro [59].

5.4.3 Salida de datos de Logstash hacia Elasticsearch

Logstash se puede configurar de diferentes maneras de cara a la salida de los eventos. Esta parte de *Output* [60] se ha configurado de forma que, por un lado se obtenga por la salida estándar la información, y por otro, se indexen los eventos en la base de datos de Elasticsearch de cara a poder visualizarlos en Kibana.

```
output{
  stdout {
  # Salida estandar
    codec => rubydebug
  }
  # Salida de elasticsearch
  elasticsearch{
    hosts => "localhost"
    #Indice bajo el que se guardan los eventos. En este caso el
    índice contiene el nombre del archivo
    index => "ditg-receiver-%{index_name}"
  }
}
```

5.5 Almacenamiento en Elasticsearch y Visualización en Kibana

Conceptualmente podemos ver Elasticsearch y Kibana como uno solo, ya que Kibana funciona también como una interfaz de usuario de la base de datos de Elasticsearch, permitiendo su gestión [61] y exploración de datos [62].

Para establecer una barrera conceptual con lo expuesto anteriormente, ahora mismo dentro de Elasticsearch están ”guardados” documentos en un formato estructurado con una serie de campos. En este trabajo no se va a entrar en profundidad en la manera en la que se indexan y buscan los datos en esta base de datos. Tampoco se va a profundizar en cómo es guardan los datos ni como Elasticsearch distribuye la información en *shards* y *replicas*.

Todos los datos almacenados están ordenados mediante un timestamp que coincide con el momento en el que se crea el evento en Logstash. Este timestamp se podría modificar perfectamente en el proceso de transformación de datos de Logstash, en caso de que la información de los *logs* aportara una referencia temporal concreta [63].

En Kibana principalmente se muestran los datos de tres maneras:

- **Dashboard**: panel de control interactivo basado en un conjunto de visualizaciones [64].
- **Canvas**: por su nombre, podemos verlo como un lienzo en blanco, donde se pueden utilizar diferentes elementos visuales junto con la representación de los datos de elasticsearch [65].
- **Discovery**: apartado de Kibana destinado a explorar los datos de la base de datos de Elasticsearch [62].

Se han realizado tres *Dashboard* y un *Canvas* con el objetivo de mostrar los datos de los tests diseñados e implementados anteriormente, los cuales comentaremos a continuación.

5.6 Visualización de datos en Kibana mediante *Dashboard*

Un *Dashboard* es un panel de control interactivo que consta de una serie de visualizaciones. Las visualizaciones permiten mostrar los datos provenientes de los índices de Elasticsearch [66]. La gestión de las visualizaciones es sencilla aunque su configuración requiere de un conocimiento de Elasticsearch básico para el uso de agregaciones [67].

La configuración del *Dashboard* se realiza añadiendo visualizaciones dentro del espacio predeterminado con la posibilidad de modificar, entre otros aspectos, su tamaño dentro de este [68].

Para acceder a los datos de Elasticsearch se deben establecer patrones de índices. En la figura 5.8 se pueden ver los patrones de índices creados. Si se quiere crear un nuevo patrón, en la figura 5.7 se muestra el menú de Kibana dedicado para ello. El asterisco funciona de comodín y permite definir una parte inicial que coincida con los índices almacenados [69].

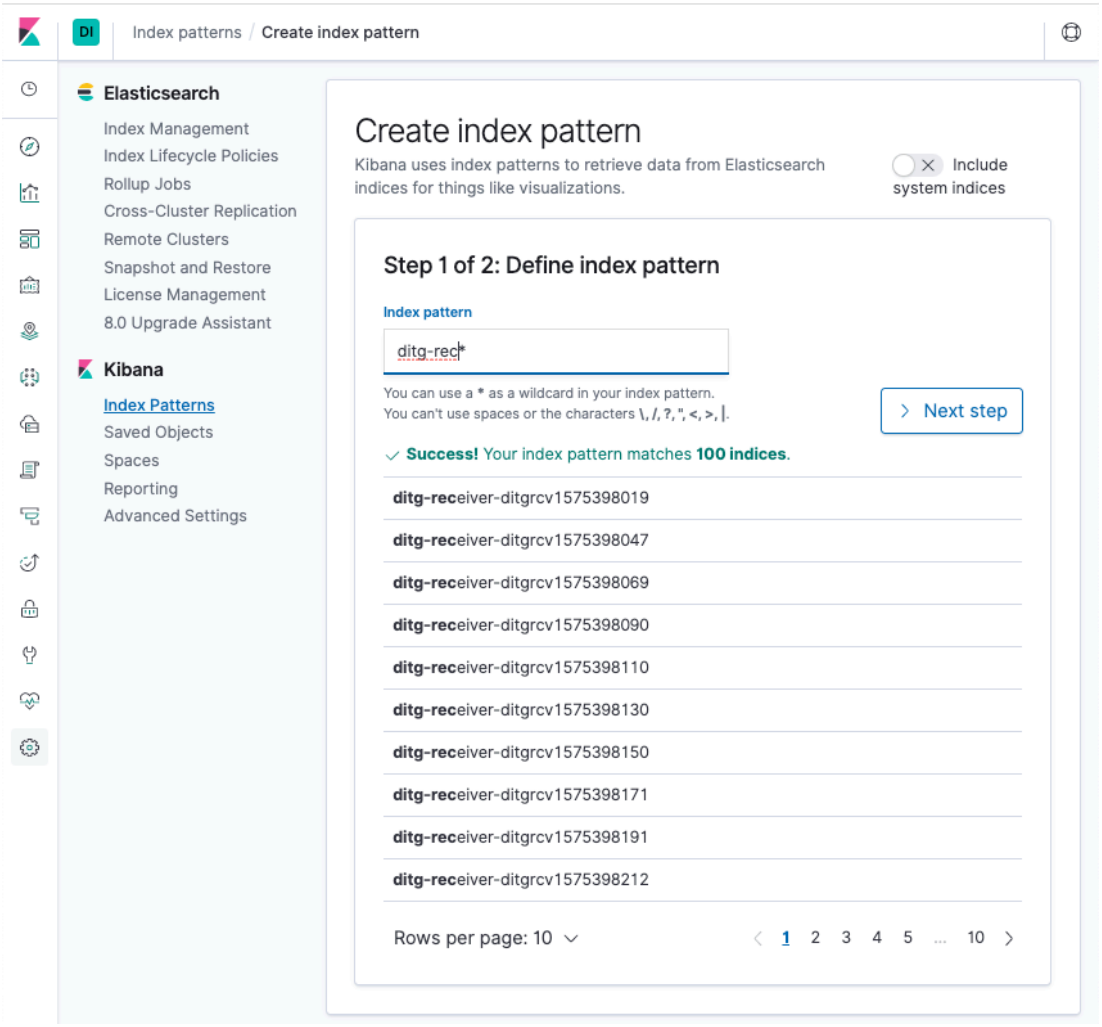

Figura 5.7: Creación de un nuevo patrón de índices.

El libro *Learning Elastic Stack 7.0* [70] proporcionó la base de conceptos de cara a entender las opciones del menú de las visualizaciones. Se comentarán simplemente las características de las visualizaciones que se han utilizado en este trabajo en concreto.

El enfoque del *Dashboard* es el de tener una forma de visualizar los datos pudiendo realizar

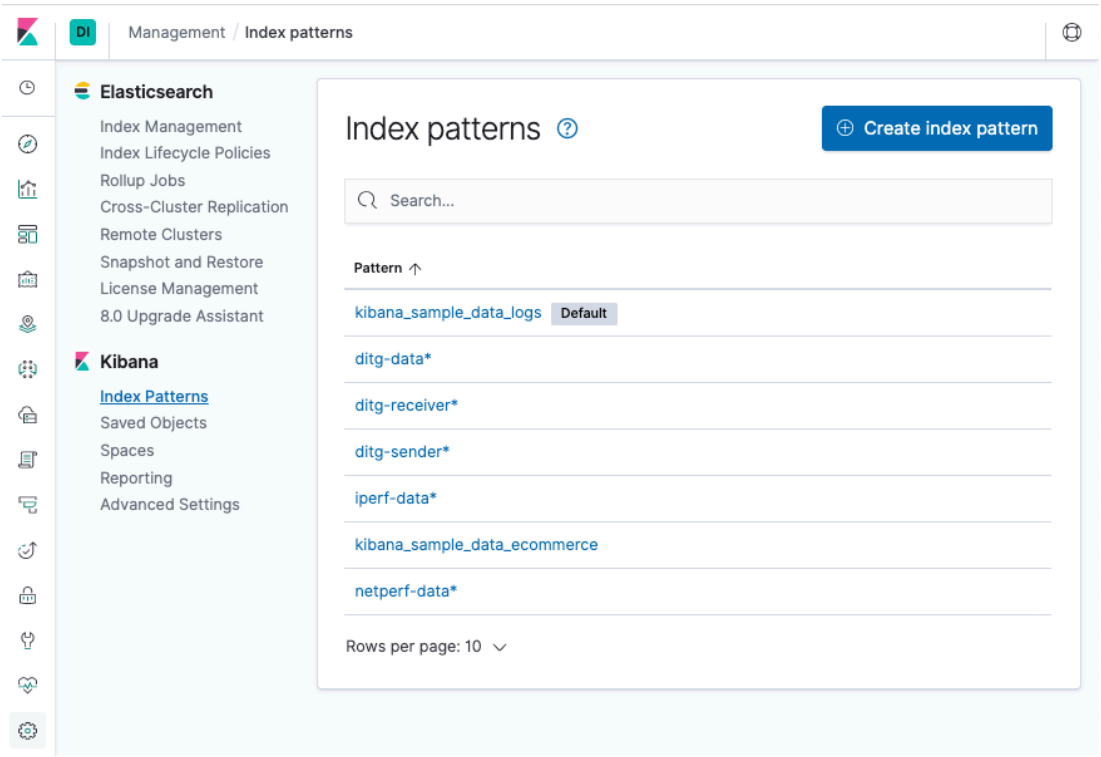

Figura 5.8: Patrones de índices creados.

una serie de filtros de manera sencilla, que permitan ver lo más rápido posible un problema entre unos datos que, de otra manera, llevaría tiempo interpretar. Además, la mayoría de los elementos son interactivos y permiten sobre todo explorar intervalos concretos en una gráfica de tiempo con gestos intuitivos.

Se han creado tres *Dashboard*, uno para cada test, donde se mostrarán todos los datos posibles que estos puedan proporcionar. A continuación se repasarán tanto sus elementos comunes como los elementos particulares determinados por cada test. Las visualizaciones utilizadas en este trabajo son las proporcionadas por Kibana. En caso de querer una solución personalizada, es posible crear visualizaciones propias desde cero.

En las figuras 5.9 y 5.10 se pueden ver tanto el menú de visualizaciones con todas las visualizaciones creadas como el menú de creación de una nueva visualización con las opciones que proporciona Kibana.

5.6.1 Selector de IP

En primer lugar, en todos estos paneles de control se ha utilizado una visualización llamada *Controls*. Este tipo de visualización permite seleccionar un campo que se aplicará como filtro en el *Dashboard*. En el selector aparecerán todos los valores diferentes que tiene este campo

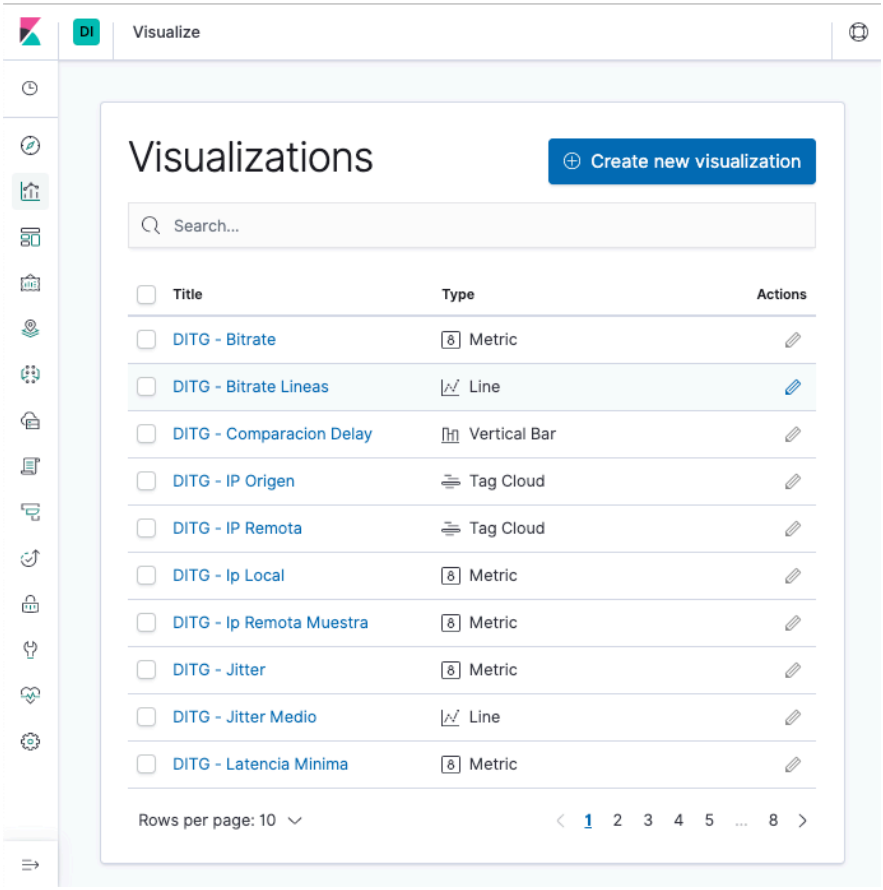

Figura 5.9: Menú de visualizaciones.

en todos los registros en el intervalo de tiempo seleccionado.

En este caso, en los tres paneles se ha elegido poder seleccionar la IP destino e IP origen, con el objetivo de mostrar test concretos que vayan de una máquina a otra determinada. Si no se incluyera esto y se tuvieran tests a diferentes máquinas (diferentes IP destino), los datos estarían mezclados y habría que establecer filtros manualmente. En la figura 5.11 podemos ver el selector de IP.

5.6.2 Indicadores Informativos

Otro elemento común es la muestra de la dirección IP origen y la dirección IP destino de los tests analizados. Además, en todos los *Dashboard* podemos ver el número de test analizados en el intervalo de tiempo seleccionado.

En la figura 5.12 se muestran dos visualizaciones juntas que muestran la IP origen y la IP destino del test respectivamente. Ambas son visualizaciones de tipo *Metric*.

En la figura 5.13 se puede ver el menú de una visualización de este tipo, junto con la

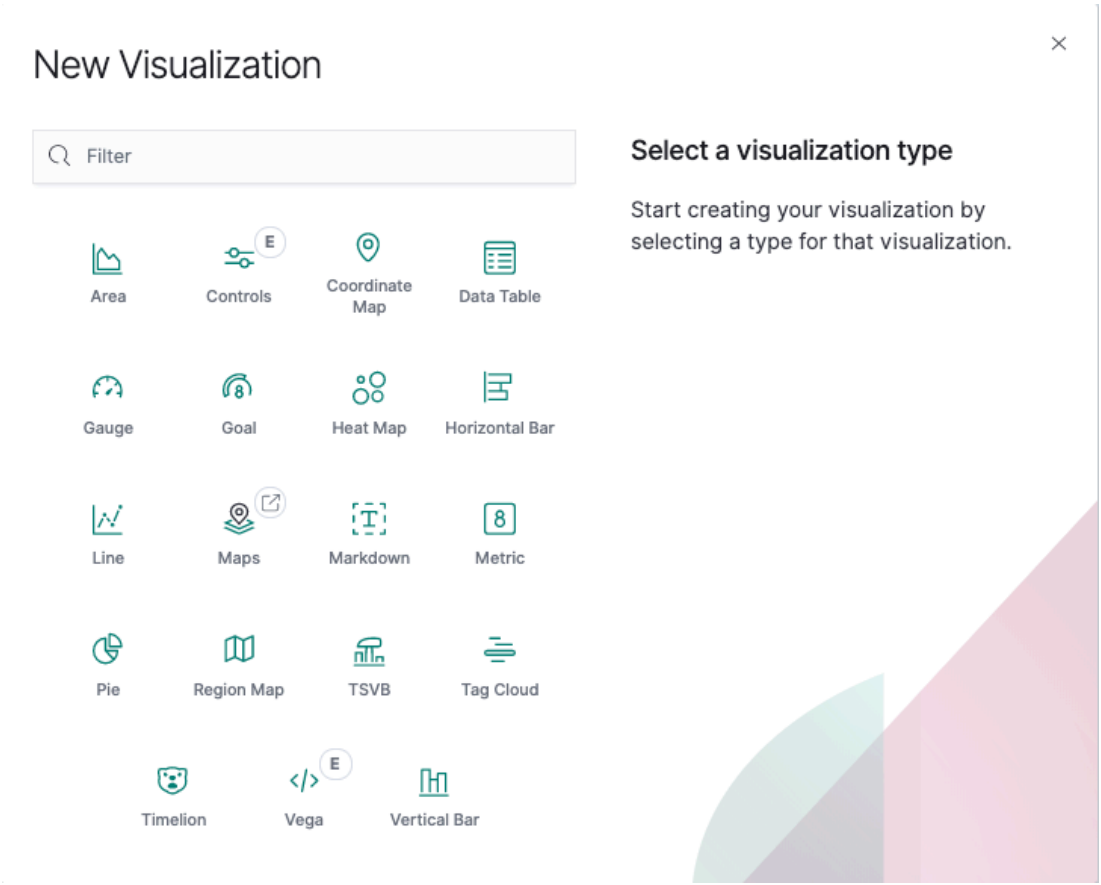

Figura 5.10: Menú de creación de una nueva visualización.

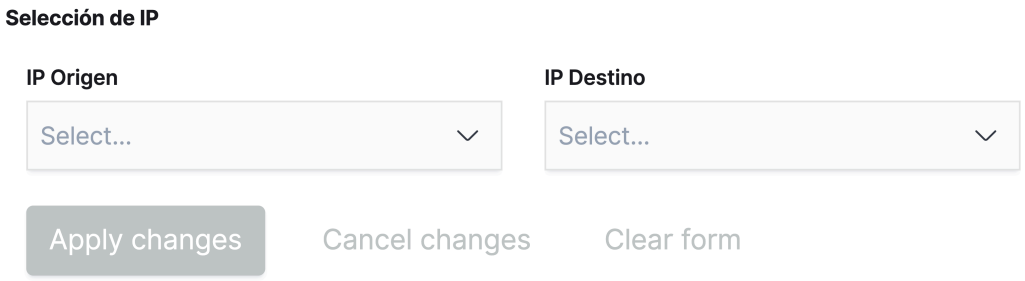

Figura 5.11: Visualización de tipo *Controls* para seleccionar IP origen/destino.

configuración necesaria en la parte izquierda.

5.6.3 Intervalos de tiempo

El control del intervalo de tiempo se puede modificar en el menú superior del *Dashboard*. Adicionalmente, se puede acotar el intervalo de forma intuitiva seleccionando porciones de

Figura 5.12: Visualización que muestra la IP origen y la IP destino.

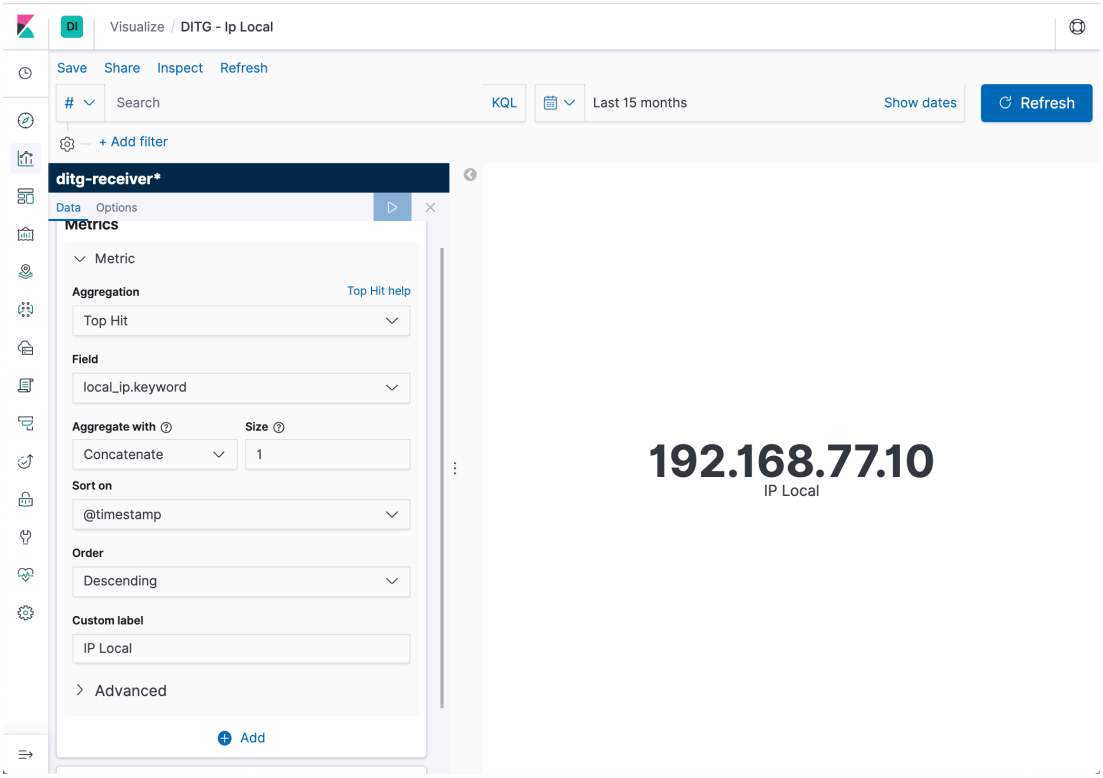

Figura 5.13: Menú de edición de una visualización de tipo *Metric*.

tiempo en cualquier visualización consistente en una gráfica en función del tiempo.

En la figura 5.14 se puede apreciar cómo se está seleccionando un intervalo en la gráfica, pulsando con el ratón en en la parte de la gráfica donde se quiere empezar a acotar. Se arrastra el ratón hasta el punto deseado y se levanta la pulsación. El resultado del acotamiento se muestra en la figura 5.15.

Dada la utilidad que tiene esta característica en el *Dashboard*, se introducen visualizaciones de gráficas de líneas y barras cuando es posible. Se busca una visión de cómo varían estos valores y poder detectar picos y anomalías de forma sencilla.

Además de poder seleccionar un intervalo de tiempo en el *Dashboard*, en la parte superior derecha se encuentra un menú (Fig 5.16) donde se puede seleccionar de formas diferentes el intervalo de tiempo. Además, se incluye la opción de refrescar los datos mostrados de forma

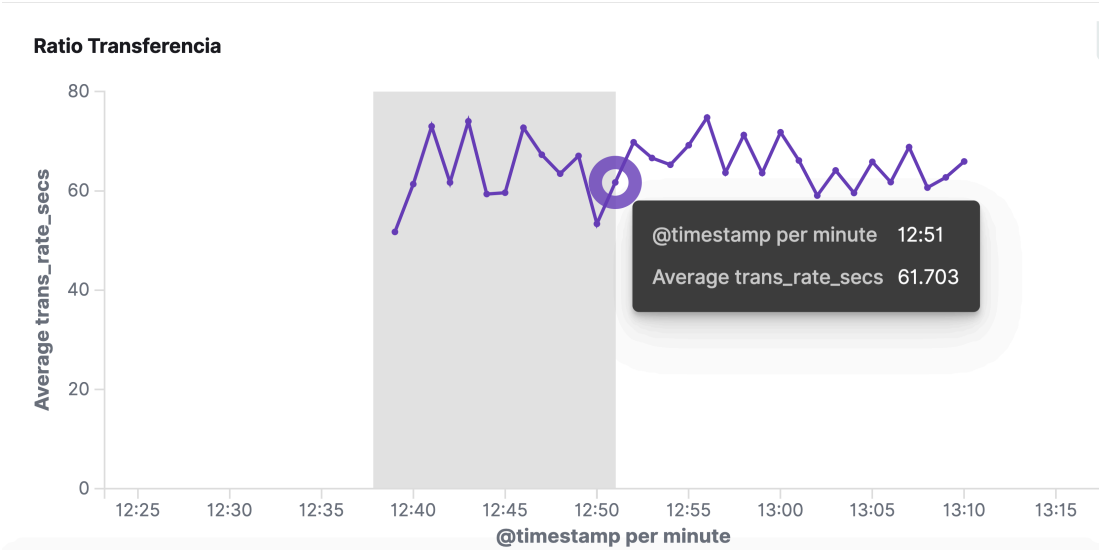

Figura 5.14: Selección del intervalo de tiempo en una visualización en el *Dashboard*.

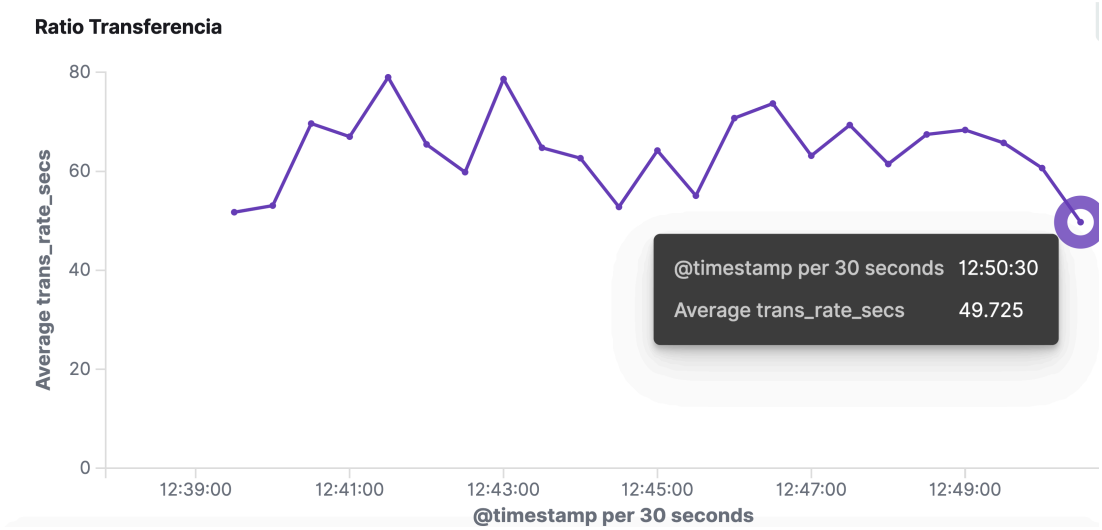

Figura 5.15: Tiempo acotado tras la selección del intervalo en el *Dashboard*.

periódica, esto permite utilizar Kibana para mostrar los datos en tiempo real refrescándose automáticamente la visualización o el *Dashboard*.

5.6.4 Elementos del *Dashboard* de Iperf

Se ha realizado una distribución por filas, situando a la izquierda los valores numéricos mediante una visualización de tipo *Metric*. A la derecha, se sitúa una gráfica de cada métrica en función del tiempo. Las métricas situadas a la izquierda consisten en un valor medio entre

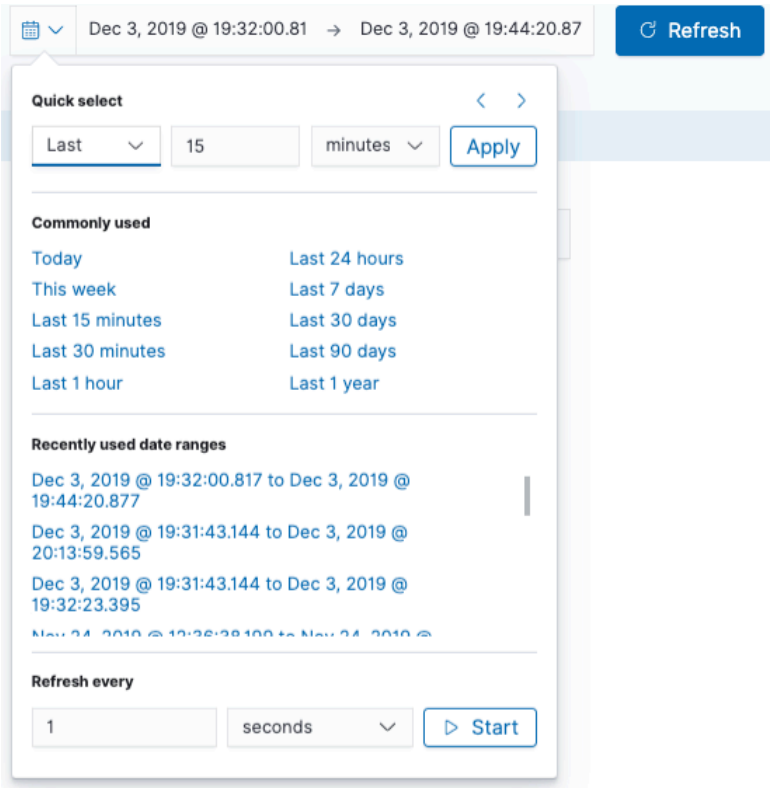

Figura 5.16: Menú para seleccionar el intervalo de tiempo de forma manual.

los datos de todos los test en el intervalo de tiempo seleccionado.

En la figura 5.17 se puede ver el Dashboard de Iperf en la interfaz de Kibana, la figura está ajustada a una resolución de 16:9, con lo que es difícil apreciar los elementos de forma horizontal. Por ello, tanto esta como el resto de figuras relacionadas con los Dashboard, estarán de forma apaisada.

Entre las métricas a mostrar encontramos:

- **Ancho de Banda**: mediante una visualización de tipo Metric se representa el ancho de banda entre las instancias cliente y servidor de Iperf.

- **Reintentos de conexión**: se representa con una visualización tipo *Metric* el número de veces que se vuelve a transmitir un paquete perdido.

- **Tamaño de ventana de congestión**: mediante una visualización de tipo *Metric* se muestra el número de bytes de la ventana de congestión.

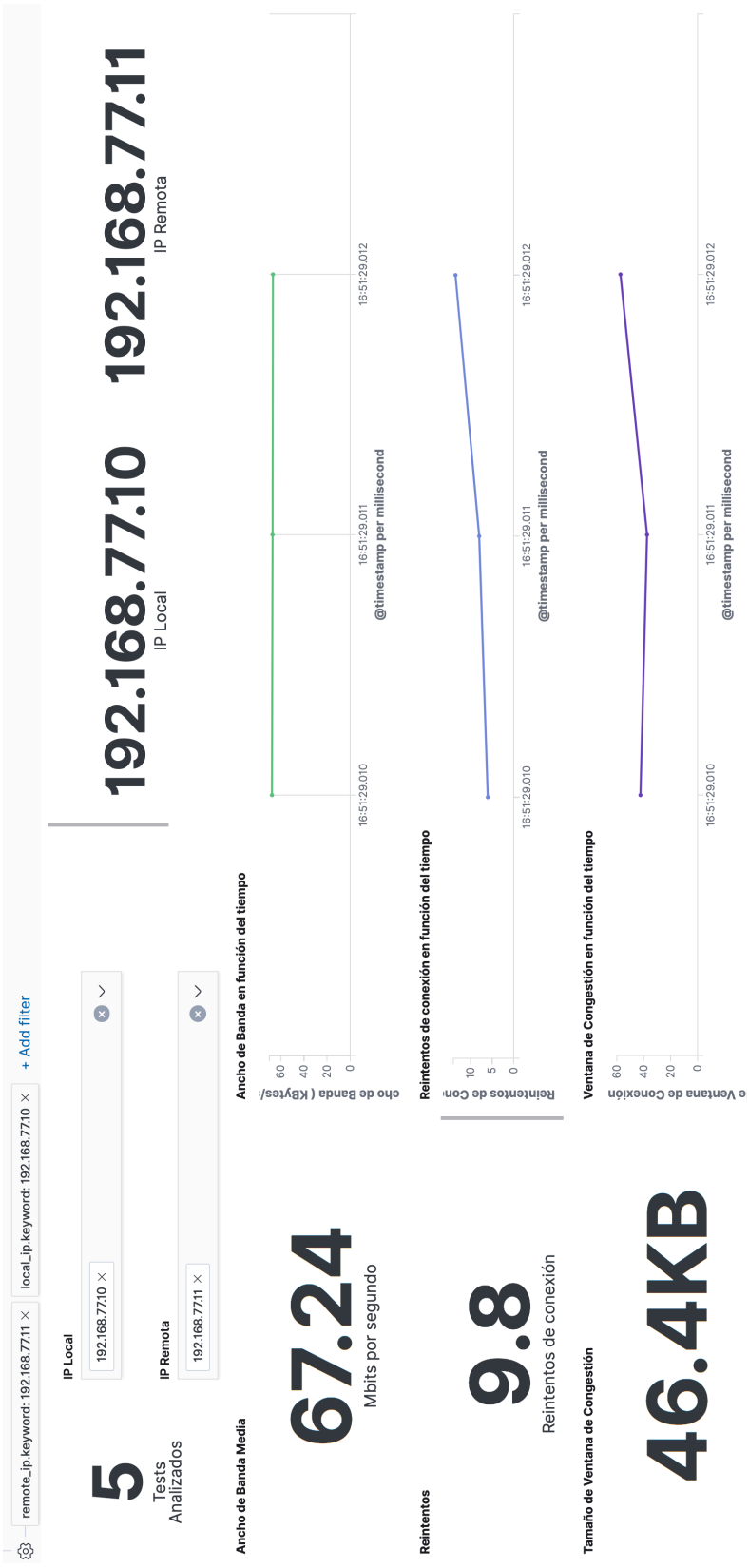

Figura 5.17: Dashboard que muestra datos de tests TCP de Iperf.

5.6.5 Elementos del *Dashboard* de Netperf

Las visualizaciones se distribuyen manera que, después de la primera línea con elementos informativos, se sitúen a la izquierda las medidas correspondientes al uso de CPU y el *Service Demand* ambas referidas al impacto del test en el rendimiento del dispositivo. A la derecha se incluyen datos respecto al volumen de transacciones por segundo de este test *Request/Response* TCP.

En la figura 5.18 se puede ver el Dashboard de Netperf donde se muestran las siguientes métricas:

- **Tasa de transferencia**: una visualización de tipo *Metric* que muestra las transacciones por segundo obtenidas en el test TCP_RR de Netperf.
- ***Service Demand***: una visualización de tipo *Metric* que representa los microsegundos que se tarda en procesar una transacción.
- **Uso de CPU**: se muestra mediante una visualización de tipo *Gauge* el porcentaje de uso de CPU que consume el test en la máquina que actúa como cliente y la que actúa como servidor por separado.

5.6.6 Elementos del *Dashboard* de D-ITG

En la figura 5.19 se puede ver el Dashboard de D-ITG, donde los datos son representados nuevamente mediante visualizaciones distribuidas en filas. Siguiendo en orden descendente respecto a la posición en el *Dashboard* tenemos:

- **Paquetes perdidos**: se muestra el porcentaje de paquetes perdidos a la izquierda mediante una visualización de tipo *Metric*, que cambia de color de verde a rojo si supera el umbral establecido para la pérdida de paquetes en el apartado 5.7.1. A la derecha se sitúa una visualización de tipo *Metric*, con los paquetes totales enviados y totales perdidos. Por último, una visualización de tipo *Line* con una gráfica de paquetes perdidos en función del tiempo.
- **Ancho de banda**: se muestra la media en Mbits por segundo con una visualización de tipo *Metric*, y a su derecha una visualización de tipo *Line* con una gráfica de líneas en función del tiempo.
- **Latencia media, máxima y mínima**: a la izquierda se sitúa una visualización de tipo *Metric* con estos tres datos. A la derecha se sitúa una visualización de tipo *Line*, en esta visualización se han configurado tres gráficas superpuestas: dos gráficas de barras para la latencia máxima y la mínima, y una gráfica de líneas para la latencia media.

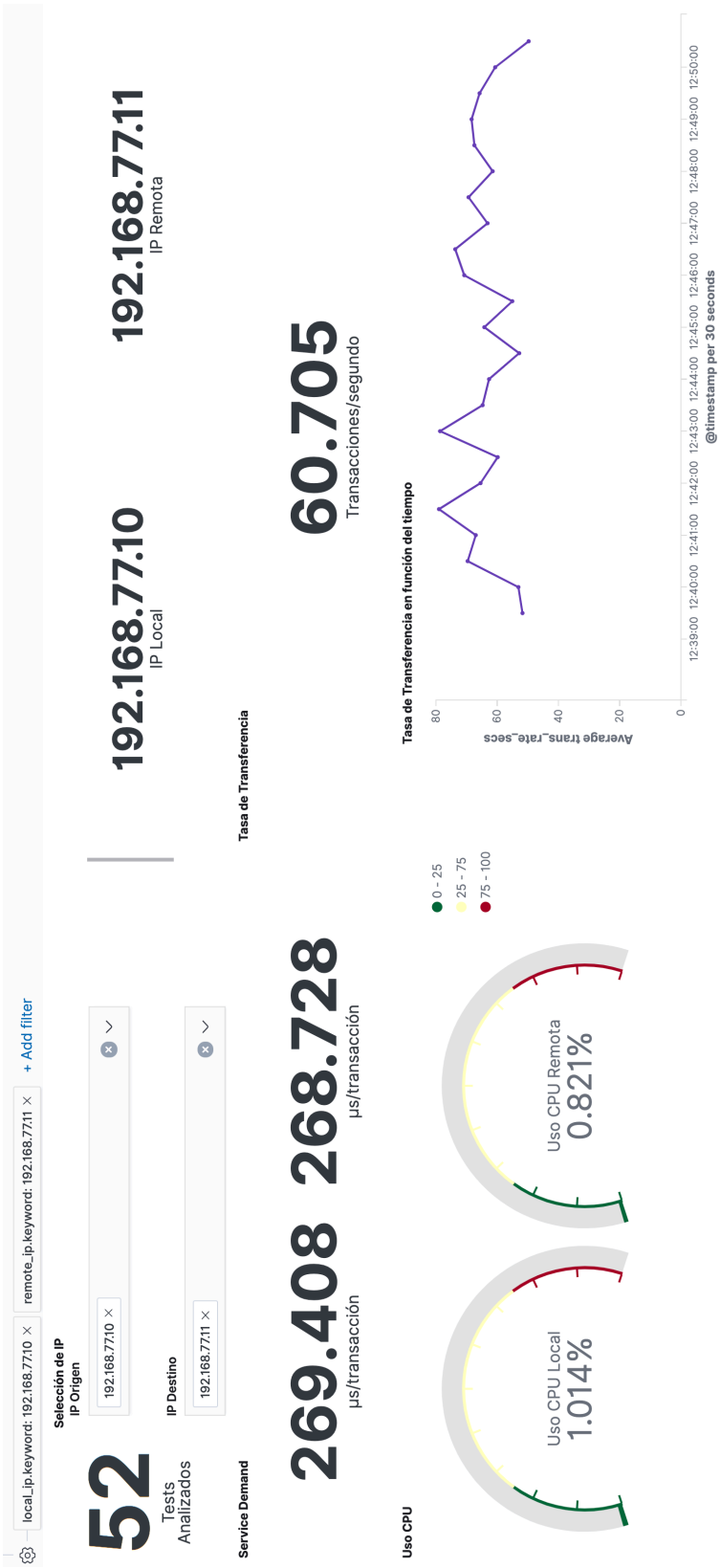

Figura 5.18: Dashboard que muestra datos de tests TCP Request/Response de Netperf.

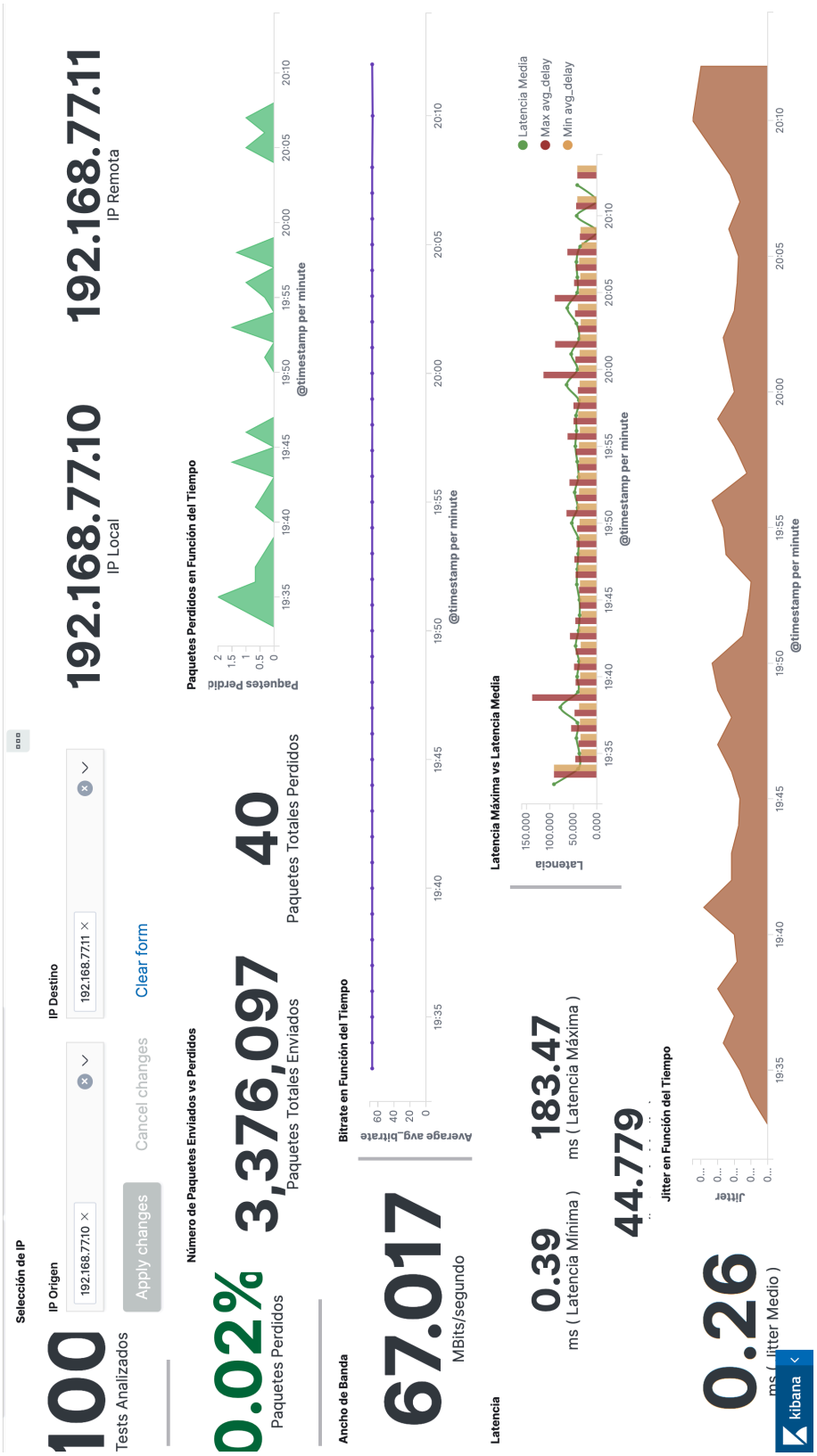

Figura 5.19: Dashboard que muestra datos de tests TCP de D-ITG.

- ***Jitter***: a la izquierda se sitúa una visualización de tipo *Metric*, y a su derecha una visualización de tipo *Line* con una gráfica de área en función del tiempo.

5.7 Visualización de datos en Kibana mediante *Canvas*

Canvas permite representar los datos de una forma más visual, aunque perdiendo interacción con los datos respecto al *Dashboard*. Se pueden realizar representaciones visuales que permitan interpretar los datos de forma más intuitiva. El concepto es parecido a una presentación de Microsoft PowerPoint o Google Slides dado que se pueden crear presentaciones al estilo de estos programas.

Para este trabajo se ha realizado el Canvas que se puede ver en la figura 5.20. Este *Canvas* permite interpretar de un de un vistazo los resultados que arrojan los tests TCP de D-ITG. Se ha elegido D-ITG ya que es el test que proporciona más datos de cara a hacer una valoración.

Figura 5.20: *Canvas* basado en un test TCP de D-ITG.

5.7.1 Establecimiento de umbrales para la evaluación de tests

Para poder realizar una valoración sobre los datos que arrojan los tests, se deben establecer una serie de umbrales que permitan representar el carácter positivo o negativo de dichos datos.

Se ha decidido utilizar los requisitos de calidad de servicio para vídeo en *streaming* de Cisco [71]. Estos requisitos incluyen muchos de los datos que se analizan en los tests de D-ITG.

De cara a su representación en el *Canvas* se decidió introducir un indicador que cambiaba de color de verde a rojo cuando se superaban los umbrales. En la tabla 5.1 se muestra la relación entre el valor y el color del indicador.

Medida	Valor	Color
Latencia	$\geq$ 150 ms	Rojo
Latencia	$<$ 150 ms	Verde
Pérdida de Paquetes	$\geq$ 1 %	Rojo
Pérdida de Paquetes	$<$ 1 %	Verde
Jitter	$>$ 30 ms	Rojo
Jitter	$\leq$ 30 ms	Verde

Tabla 5.1: Umbrales basados en QoS de Cisco y su representación en *Canvas*.

Tres indicadores al estilo de un semáforo permiten interpretar mediante una evaluación los resultados de los tests. Los resultados se dividen en tres:

- **Bien**: si no se supera ningún umbral.
- **Regular**: si se supera algún umbral.
- **Mal**: si no se superan dos o más umbrales.

El número que aparece en el Canvas es el resultado de una sentencia SQL, como la siguiente para determinar el número de resultados clasificados como "MAL":

```
SELECT COUNT(*) as errores
    FROM "ditg-receiver*" WHERE (avg_jitter_usecs >= 30000 AND
    percent_pack_dropped
 >= 1) OR ( avg_jitter_usecs >= 30000 AND avg_delay_usecs
>= 150000) OR ( percent_pack_dropped
 >= 1 AND avg_delay_usecs
>= 150000)
```

En la tabla 5.2 se resume la relación entre los fallos, el color y el resultado de una evaluación de un test.

5.7.2 Presentación de datos en Canvas

En *Canvas* a diferencia del *Dashboard* los datos se pueden conseguir de diferentes fuentes (Fig.5.21). Se ha decidido utilizar las sentencias SQL hacia la base de datos de Elasticsearch.

Resultado	Color	Fallos
MAL	Naranja	2 o 3
REGULAR	Amarillo	1
BIEN	Verde	0

Tabla 5.2: Relación entre umbrales superados y color.

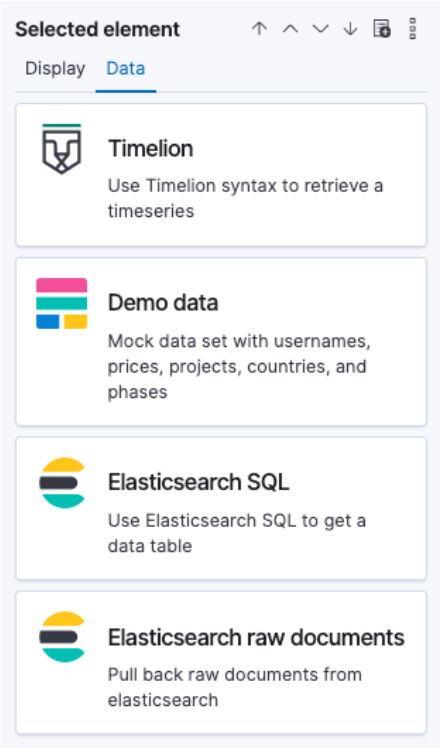

Figura 5.21: Fuentes de datos para *Canvas*.

Se han introducido elementos visuales que, una vez entendido su significado, permiten tener una percepción de los datos a un golpe de vista. Se muestran tres datos que representan la calidad de la conexión TCP, latencia, pérdida de paquetes y *jitter*.

- **Latencia**.
- **Pérdida de Paquetes**.
- ***Jitter***.

Para estos tres datos se incluye una representación numérica de la media entre todos los tests del intervalo de tiempo seleccionado. Para seleccionar el intervalo de tiempo se ha incluido un elemento denominado *Time Filter* en la parte inferior, siendo el único elemento

interactivo del *Canvas*. Las métricas numéricas se obtienen con una sentencia SQL como la siguiente, para la latencia:

```
SELECT (avg(avg_delay_usecs))/1000 as latencia
FROM "ditg-receiver*"
```

Encima de cada métrica se encuentra una imagen de una llama que se va rellenando. Aquí se utiliza el elemento *Image Reveal* de *Canvas*. Este elemento se compone de dos imágenes superpuestas: una en escala de grises en segundo plano y la misma imagen en color en primer plano que va aumentando de tamaño según un porcentaje.

El criterio a seguir es el porcentaje en tanto por uno del valor medio entre todos los tests respecto al umbral. Este porcentaje se extrae mediante los datos que aportan las sentencias SQL a la base de datos de Elasticsearch, por ejemplo para la latencia:

```
SELECT ((avg_delay_usecs)/150000) as porcentaje
    FROM "ditg-receiver*"
```

Luego se selecciona cómo se muestran estos valores extraídos como tabla, de la manera que se muestra en la figura 5.22.

```
"clamp(mean(porcentaje),0,1)")
```

Se realiza una media de los valores que se han pasado del tanto por uno anterior y restringe los datos entre 0 y 1 mediante la operación clamp() [72]. De esta manera podemos ver de forma general como de cerca están de la media los datos respecto al umbral.

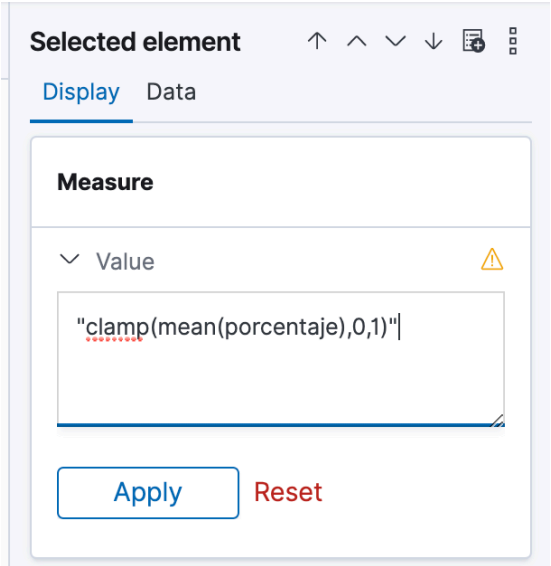

Figura 5.22: Selección de datos para el elemento *Image Reveal*.

Capítulo 6

Conclusiones

En este último capítulo de la memoria se presentaran las lecciones aprendidas y las líneas futuras de trabajo.

6.1 Lecciones aprendidas

La realización de este trabajo ha supuesto un aprendizaje continuo tanto en la comparativa de herramientas como en el desarrollo de la plataforma basada en la pila ELK.

6.1.1 Lecciones aprendidas en la comparativa de herramientas

En relación con lo que se ha aprendido en el grado, realizar una comparativa de carácter técnico es un nuevo reto. El análisis de la documentación y la extracción de características importantes para realizar una comparación requiere de más tiempo del que se pueda estimar a priori. Realmente esta tarea nunca termina hasta finalizar el proyecto, dado que constantemente se acaba revisando una y otra vez la documentación y se puntualizan y remarcan ciertos datos.

En este trabajo se ha aprendido la identificación de un software obsoleto o sin un soporte continuado, algo que solo se había visto en este momento. El software utilizado en la carrera funciona y está mantenido, además de que muchas dudas las acaba resolviendo el profesor. Sin embargo, en esta ocasión se ha tenido que contactar con responsables de los proyectos para pedir ayuda con la instalación o pedir soporte.

Se ha requerido de un gran esfuerzo para analizar la documentación y probar las herramientas, dado que en algunas ocasiones, como en la configuración de las interfaces de TRex, se perdió mucho tiempo en configurarlas porque se desconocía cómo hacerlo dentro del ámbito del emulador CORE.

Cada programa utilizado ha supuesto enfrentarse a un problema nuevo, con la consecuente sensación constante de frustración y de no haber avanzado. Gestionar esta frustración ha

sido fundamental para poder sacar el proyecto adelante.

6.1.2 Lecciones aprendidas en el desarrollo de la plataforma

El aprendizaje de ELK ha sido duro, dado que empezar a usarlo desde cero es difícil y no existen muchos recursos de calidad , a parte de su guía de referencia y el foro de dudas. Vuelve a entrar en juego la gestión de la frustración y la búsqueda de recursos alternativos basados en versiones obsoletas de ELK. Estos recursos obsoletos requieren de una traslación a la versión actual, lo que requiere tiempo y estudio.

Entre las lecciones aprendidas y las competencias ganadas mediante el desarrollo de la plataforma basada en la pila ELK se podrían destacar las siguientes:

- Aprendizaje de uso de base de datos basada en Lucene. Una base de datos totalmente diferente a las bases de datos relaciones SQL vistas en el grado.
- Análisis de logs y aprendizaje de uso de filtros Logstash para logs de texto multilínea, uno de los casos de uso más complejos de ELK.

6.2 Trabajos futuros

Han surgido diferentes ideas para trabajos futuros:

- Utilización de los scripts de forma periódica mediante la *Crontab* de Linux.
- Comparativa de más características de las herramientas.
- Estudio y comparativa de rendimiento de tests utilizando ELK.
- Profundizar en el uso de la herramienta TRex para ingestar datos en ELK.
- Profundizar en el uso de la herramienta MGEN para ingestar datos en ELK .

Apéndices

Apéndice A

Ficheros de Configuración de Logstash

A.1 Configuración de Logstash para Iperf

```
input {
  #Input mediante file.
  file{
    #Se buscan todos los archivos iperf*.log
    path =>
    "/Users/davidignacio/Documents/data_test/iperf/iperf*.log"
    start_position => "beginning"
  }
}
filter{
  # Se ha dividido la ingesta en dos partes, en esta parte de los
    filtros simplemente obtenemos
  # los eventos y los separamso por tipos poniéndoles un tag

  grok {
    #Guardamos el nombre del archivo para su posterior indexación
    en elasticsearch
    match => ["path", "davidignacio/Documents/data_test/iperf/
    %{GREEDYDATA:index_name}.log" ]
  }
  if "local" in [message] {
    grok{
      # Patrones creados por el desarrollador
       patterns_dir => ["/etc/patterns/custom"]
      # Match si el mensaje sigue este patrón
       match => { "message" =>
    "%{GREEDYDATA}%{NUMBER:id}%{GREEDYDATA}
```

```
24
    %{IP:local_ip:ip}%{GREEDYDATA}%{NUMBER:local_port:int}%{GREEDYDATA}
25      %{IP:remote_ip:ip}%{GREEDYDATA}%{NUMBER:remote_port}"}
26      # Etiqueta identificativa de cara su clasificacion
27      add_tag => ["HOSTS"]
28    }
29  } else if "(omitted)" not in [message] {
30    grok{
31      # Patrones creados por el desarrollador
32      patterns_dir => ["/etc/patterns/custom"]
33      # Match si el mensaje sigue este patrón
34      match => { "message" =>
    "%{GREEDYDATA}%{NUMBER:id:int}%{DATA}%{SPACE}%{FLOAT:ini:float}
35 (?:-)%{FLOAT:end:float}%{SPACE}%{WORD}%{SPACE}
36 %{FLOAT:transfered_bytes:float}%{SPACE}%{WORD:unit_trans}
37 %{SPACE}%{FLOAT:bandwidth:float}%{SPACE}%{DATA:unit_bandwidth}
38 %{SPACE}%{NUMBER:retry:int}%{SPACE}%{NUMBER:cwnd:float}
39 %{SPACE}%{WORD:unit_cwnd}"}
40      # Etiqueta identificativa de cara su clasificacion
41      add_tag => ["DATA"]
42      #Añadidos los campos de información que vamos a copiar como
    campos vacíos
43      add_field => {
44        "local_ip" => ""
45        "local_port" => ""
46        "remote_ip" => ""
47        "remote_port" => ""
48      }
49    }
50  } else {
51    drop{}
52  }
53
54  # Segunda parte donde según la clasificación que hemos dado a los
    tags
55  if "HOSTS" in [tags] {
56    aggregate{
57      task_id => "%{id}"
58      #Se copian en el mapa todos los campos del evento
59      code => "
60           map['local_ip'] = event.get('local_ip')
61           map['local_port'] = event.get('local_port')
62           map['remote_ip'] = event.get('remote_ip')
63           map['remote_port'] = event.get('remote_port')
64      "
65      #Crear si no existe y actualizar en otro caso
```

```
66        map_action => "create_or_update"
67      }
68      #Eliminar el evento de Host, no se va a utilizar, se elimina.
69      drop{}
70    } else if "DATA" in [tags] {
71      aggregate{
72        task_id => "%{id}"
73        #Copiar el mapa en los campos añadidos anteriormente
74        code => "
75             event.set('local_ip', map['local_ip'])
76             event.set('local_port', map['local_port'])
77             event.set('remote_ip', map['remote_ip'])
78             event.set('remote_port', map['remote_port'])
79        "
80        #Actualizar
81        map_action => "update"
82        #DEBUG : Como recordatorio de que podemos lanzar el mapa como
      un nuevo evento.
83        #push_previous_map_as_event => false
84        #Tiempo que dura el mapa, este tiempo que se ha establecido
      es igual a por defecto
85        #se ha dejado a modo de aclaración
86        timeout => 1800
87      }
88    }
89    if "DATA" in [tags] {
90        ##Convertir las unidades a bytes y bytes por segundo
91        #Selecciona el multiplicador para pasar a bytes
92        if [unit_trans] == "MBytes" {
93          mutate { add_field => { "unit_trans_multiplier" => 1048576
      } }
94        } else if [unit_trans] == "Kbytes" {
95          mutate { add_field => { "unit_trans_multiplier" => 1024 } }
96        }
97        if [unit_bandwidth] == "Mbits/sec" {
98          mutate { add_field => { "unit_bandwidth_multiplier" =>
      1048576 } }
99        } else if [unit_bandwidth] == "Kbits/sec" {
100         mutate { add_field => { "unit_bandwidth_multiplier" => 1024
      } }
101       }
102       if [unit_cwnd] == "MBytes" {
103         mutate { add_field => { "unit_cwnd_multiplier" => 1048576 }
      }
104       } else if [unit_cwnd] == "KBytes" {
105         mutate { add_field => { "unit_cwnd_multiplier" => 1024 } }
```

```
106       }
107
108       #Se añade los nuevos campos para los bytes
109       mutate {
110         add_field => {
111           "trans_bytes" => "0.0"
112           "bandwidth_bytes" => "0.0"
113           "cwnd_bytes" => "0.0"
114         }
115         #Se convierten todos los campos a float para multiplicarlos
    por los valores recogidos de los logs.
116         convert => {
117           "trans_bytes" => "float"
118           "unit_trans_multiplier" => "float"
119           "bandwidth_bytes" => "float"
120           "unit_bandwidth_multiplier" => "float"
121           "cwnd_bytes" => "float"
122           "unit_cwnd_multiplier" => "float"
123         }
124       }
125       #Conversión de las unidades a bytes y almacenadas en los
    nuevos campos.
126       ruby {
127         code => "
128         event.set('trans_bytes',event.get('transfered_bytes')*
    event.get('unit_trans_multiplier'))
129         event.set('bandwidth_bytes',event.get('bandwidth')*
    event.get('unit_bandwidth_multiplier'))
130         event.set('cwnd_bytes',event.get('cwnd')*
    event.get('unit_cwnd_multiplier'))
131         "
132       }
133       #Conversión a integer para el posterior formateo en Kibana
    como "Bytes"
134       mutate{
135         convert => {
136           "trans_bytes" => "integer"
137           "bandwidth_bytes" => "integer"
138           "cwnd_bytes" => "integer"
139         }
140       }
141   }
142   # Se descartan los eventos que corresponden a errores de parseo
143   if "_grokparsefailure" in [tags]  {
144   drop { }
145   }
```

```
}
output {
    stdout {
      codec => rubydebug
    }
    # Salida de elasticsearch
    elasticsearch{
      hosts => "localhost"
      #Indice bajo el que se guardan los eventos. En este caso el
    índice contiene el nombre del archivo
      index => "iperf-data-%{index_name}"
    }
}
```

A.2 Configuración de Logstash para Netperf

```
input {
  #Input mediante file.
  file{
    #Se buscan todos los archivos que compienzan por netperf-log
    path =>
    "/Users/davidignacio/Documents/data_test/netperf/netperf*.log"
    start_position => "beginning"
  }
}
filter{
  grok {
    #Guardamos el nombre del archivo para su posterior indexación
    en elasticsearch
    match => ["path", "davidignacio/Documents/data_test/netperf/
    %{GREEDYDATA:index_name}.log" ]
  }
  if "MIGRATED" in [message]{
    grok{
        # Match si el mensaje sigue este patrón
        match => { "message" =>
    "%{GREEDYDATA}%{IP:local_ip:ip}%{GREEDYDATA}%{IP:remote_ip:ip}
        %{GREEDYDATA}"}
        # Etiqueta identificativa de cara su clasificacion
        add_tag => ["HOSTS"]
        add_field => {
          "id" => "1"
        }
    }
```

```
26  } else {
27    grok{
28      # Patrones creados por el desarrollador
29      patterns_dir => ["/etc/patterns/custom"]
30      # Match si el mensaje sigue este patrón
31      match => { "message" =>
      "%{NUMBER:send_socket_bytes:int}%{SPACE}
32
      %{NUMBER:size_recv_bytes:int}%{SPACE}%{NUMBER:req_size_bytes:int}
33      %{SPACE}%{NUMBER:res_size_bytes:int}%{SPACE}
34      %{FLOAT:elaps_time_secs:float}%{SPACE}
35      %{FLOAT:trans_rate_secs:float}%{SPACE}
36      %{FLOAT:local_cpu_usage:float}%{SPACE}
37      %{FLOAT:remote_cpu_usage:float}%{SPACE}
38      %{FLOAT:local_service_demand:float}%{SPACE}
39      %{FLOAT:remote_service_demand:float}"}
40      # Etiqueta identificativa de cara su clasificacion
41      add_tag => ["DATA"]
42      #Añadidos los campos de información que vamos a copiar como
      campos vacíos
43      add_field => {
44        "id" => "1"
45        "local_ip" => ""
46        "remote_ip" => ""
47      }
48    }
49  }
50  # Segunda parte donde según la clasificación que hemos dado a los
      tags
51  if "HOSTS" in [tags] {
52    aggregate{
53      task_id => "%{id}"
54      #Se copian en el mapa todos los campos del evento
55      code => "
56           map['local_ip'] = event.get('local_ip')
57           map['remote_ip'] = event.get('remote_ip')
58      "
59      #Crear si no existe y actualizar en otro caso
60      map_action => "create_or_update"
61    }
62    #Eliminar el evento de Host, no se va a utilizar, se elimina.
63    drop{}
64  } else if "DATA" in [tags] {
65    aggregate{
66      task_id => "%{id}"
67      #Copiar el mapa en los campos añadidos anteriormente
```

```
68        code => "
69             event.set('local_ip', map['local_ip'])
70             event.set('remote_ip', map['remote_ip'])
71        "
72        #Actualizar
73        map_action => "update"
74        #DEBUG : Como recordatorio de que podemos lanzar el mapa como
      un nuevo evento.
75        #push_previous_map_as_event => false
76        #Tiempo que dura el mapa, este tiempo que se ha establecido
      es igual a por defecto
77        #se ha dejado a modo de aclaración
78        timeout => 1800
79      }
80    }
81    # Se descartan los eventos que corresponden a errores de parseo
82    if "_grokparsefailure" in [tags] {
83    drop { }
84    }
85  }
86
87  output{
88    stdout {
89      codec => rubydebug
90    }
91    # Salida de elasticsearch
92    elasticsearch{
93      hosts => "localhost"
94      #Indice bajo el que se guardan los eventos. En este caso el
      índice contiene el nombre del archivo
95      index => "netperf-data-%{index_name}"
96    }
97
98  }
```

A.3 Configuración de Logstash para D-ITG

```
1  input {
2    #Input mediante file.
3    file{
4      #Se buscan todos los archivos que compienzan por ditg-log
5      path => "/Users/davidignacio/Documents/data_test/ditg/receiver/
       ditgrcv*.log"
6      start_position => "beginning"
7
```

```
8       codec => multiline {
9         #El patrón pretende descartar los datos totales
10        pattern => "^\*+  TOTAL RESULTS   \*+$"
11        what => "previous"
12        negate => true
13      }
14    }
15  }
16  filter{
17    grok {
18      #Guardamos el nombre del archivo para su posterior indexación
        en elasticsearch
19      match => ["path",
        "davidignacio/Documents/data_test/ditg/receiver/
20      %{GREEDYDATA:index_name}.log" ]
21    }
22    grok{
23      # Patrones creados por el desarrollador
24      patterns_dir => ["/etc/patterns/custom"]
25      #Patron sobre el mensaje de una línea creado por el codec
        multilínea
26      match => { "message" =>
        "%{IP:local_ip:ip}%{DATA}%{NUMBER:local_port}%{DATA}
27      %{IP:remote_ip:ip}%{DATA}%{NUMBER:remote_port}%{DATA}
28      %{SPACE}%{FLOAT:total_time_secs:float}%{DATA}%{SPACE}
29      %{NUMBER:total_packets:int}%{DATA}%{SPACE}
30      %{FLOAT:min_delay:float}%{DATA}%{SPACE}
31      %{FLOAT:max_delay:float}%{DATA}%{SPACE}
32      %{FLOAT:avg_delay:float}%{DATA}%{SPACE}
33      %{FLOAT:avg_jitter:float}%{DATA}%{SPACE}
34      %{FLOAT:delay_stddev:float}%{DATA}%{SPACE}
35      %{FLOAT:bytes_recv:float}%{DATA}%{SPACE}
36      %{FLOAT:avg_bitrate_kbits:float}%{DATA}%{SPACE}
37      %{FLOAT:avg_packetrate:float}%{DATA}%{SPACE}
38      %{NUMBER:npackets_dropped:int}%{DATA}
39      %{FLOAT:percent_pack_dropped:float}
40      %{DATA}%{SPACE}%{FLOAT:avg_loss-burst_size:float}"}
41    }
42    ##Conversion de las unidades en segundos a microsegundos
43    mutate{
44      add_field => {
45        "multiplier" => 1000000
46        "total_time_usecs" => 0
47        "min_delay_usecs" => 0
48        "max_delay_usecs" => 0
49        "avg_delay_usecs" => 0
```

```
50        "stddev_delay_usecs" => 0
51        "avg_jitter_usecs" => 0
52      }
53      convert => {
54        "multiplier" => float
55        "total_time_usecs" => float
56        "min_delay_usecs" => float
57        "max_delay_usecs" => float
58        "avg_delay_usecs" => float
59        "stddev_delay_usecs" => float
60        "avg_jitter_usecs" => float
61      }
62    }
63    ruby{
64      code => "
65        event.set('total_time_usecs', event.get('total_time_secs')*
      event.get('multiplier'))
66        event.set('min_delay_usecs', event.get('min_delay')*
      event.get('multiplier'))
67        event.set('max_delay_usecs', event.get('max_delay')*
      event.get('multiplier'))
68        event.set('avg_delay_usecs', event.get('avg_delay')*
      event.get('multiplier'))
69        event.set('stddev_delay_usecs', event.get('delay_stddev')*
      event.get('multiplier'))
70        event.set('avg_jitter_usecs', event.get('avg_jitter')*
      event.get('multiplier'))
71      "
72    }
73    # Se descartan los eventos que corresponden a errores de parseo
74    if "_grokparsefailure" in [tags] {
75    drop { }
76    }
77  }
78  output{
79    stdout {
80      codec => rubydebug
81    }
82    # Salida de elasticsearch
83    elasticsearch{
84      hosts => "localhost"
85      #Indice bajo el que se guardan los eventos. En este caso el
      índice contiene el nombre del archivo
86      index => "ditg-receiver-%{index_name}"
87    }
88  }
```

Apéndice B

Scripts

B.1 Launch.sh

```
#!/bin/bash

#### FLAGS
# Flags opciones obligatorias
tflag=false
lflag=false
rflag=false
#Flag debug
debflag=true

#### CONSTANTES
usuario=tfgdiconde #Usuario al que conectarse
PATH_SALIDA=/mnt/hgfs/data_test #Carpeta de salida de los datos de
    iperf

# Tests Implementados ( Array que contiene los tests disponibles )
tests=(iperf_tcp netperf_rrtcp ditg_tcp)

#### FUNCIONES
#Función de ayuda donde se especifica el uso del script
uso() {
  echo "Uso: $0 [-t <tipo_test>] [-l <ip_local>] [-r <ip_remota>]";
  exit 1;
}
#Impresión de las opciones disponibles para -t, los tests que se
    pueden realizar
list_tests () {
    echo "Tests Disponibles:"
  for i in "${tests[@]}"
    do
```

```
29        echo "  --> $i"
30      done
31  }
32
33  debug(){
34    # Datos introducidos
35    echo "TEST_TYPE = $TEST_TYPE"
36    echo "LOCAL_IP = $LOCAL_IP"
37    echo "REMOTE_IP = $REMOTE_IP"
38  }
39
40  #### MAIN
41  # PARAMETROS
42  #Lista de opciones
43  options=':t:l:r:h'
44  while getopts $options op; do
45    case ${op} in
46      t) #Tipo de test junto la herramienta
47        TEST_TYPE=${OPTARG}; tflag=true
48        ;;
49      l) #Ip origen
50        LOCAL_IP=${OPTARG}; lflag=true
51        ;;
52      r) #Ip destino
53        REMOTE_IP=${OPTARG}; rflag=true
54        ;;
55      h) #Mostrar ayuda
56        list_tests; uso; exit 1
57        ;;
58      \?)
59        echo "Opción desconocida: -$OPTARG" >&2; exit 1
60        ;;
61    esac
62  done
63
64  if ((OPTIND == 1))
65  then
66      echo "No se ha seleccionado ninguna opcion"
67  fi
68  #Borra todas las opciones que se han parseado por getops de la
      lista de parámetros.
69  # Después de este punto $1 será referido al primer argumento que no
      sea una opción pasado al script.
70  shift $((OPTIND-1))
71
```

```
72 # Comprobación opciones obligatorias, si no están todas se imprime
       la ayuda (función uso).
73 if ! $tflag || ! $lflag || ! $rflag
74 then
75   echo "Las opciones -t -l -r tienen que estar presentes
       obligatoriamente" >&2
76   uso
77   list_tests
78   exit 1
79 fi
80
81 ## DEBUG
82 if $debflag
83 then
84   debug
85 fi
86 ##
87
88 #### TESTS
89 case $TEST_TYPE in
90   #Test de iperf sobre tcp
91   iperf_tcp)
92     #Lanzar servidor mediante comando sobre ssh
93     # Ejecutar el servidor en remoto (iperf -s) y guardar el pid
       del proceso creado remotamente
94     SERVER_PID=$(sshpass -p 1234 ssh $usuario@$REMOTE_IP "sh -c
       'nohup iperf3 -s > /dev/null 2>&1 & echo \$!'")
95     #Espera de 2 segundos para dejar tiempo a que arranque el
       servidor
96     sleep 2s
97     #Lanzar cliente en la máquina local
98     sshpass -p 1234 ssh $usuario@$LOCAL_IP "sh -c 'iperf3 -c
       $REMOTE_IP -i 1 -t 5 -O 1'" > $PATH_SALIDA/iperf/iperf$(date
       +%s).log
99     #Matamos el proceso del servidor
100    sshpass -p 1234 ssh $usuario@$REMOTE_IP kill $SERVER_PID
101    ;;
102  #Test Netperf Request Response sobre TCP
103  netperf_rrtcp)
104    #Lanzar servidor mediante comando sobre ssh
105    # Ejecutar el servidor en remoto (netserver) y guardar el pid
       del proceso creado remotamente
106    SERVER_PID=$(sshpass -p 1234 ssh $usuario@$REMOTE_IP "sh -c
       'nohup netserver > /dev/null 2>&1 & echo \$!'")
107    SERVER_PID=$((SERVER_PID+1))
```

```
108     #Espera de 2 segundos para dejar tiempo a que arranque el
        servidor
109     sleep 2s
110     #Lanzar cliente en la máquina local
111     sshpass -p 1234 ssh $usuario@$LOCAL_IP "sh -c 'netperf -T 1 -L
        $LOCAL_IP -H $REMOTE_IP -t TCP_RR -c -C'" >
        $PATH_SALIDA/netperf/netperf$(date +%s).log
112     #Matamos el proceso del servidor
113     sshpass -p 1234 ssh $usuario@$REMOTE_IP kill $SERVER_PID
114   ;;
115   ditg_tcp)
116     #Lanzar servidor mediante comando sobre ssh
117     # Ejecutar el servidor en remoto (ITGRecv) y guardar el pid del
        proceso creado remotamente
118     SERVER_PID=$(sshpass -p 1234 ssh $usuario@$REMOTE_IP "sh -c
        'nohup ITGRecv -l temprecv.log > /dev/null 2>&1 & echo \$!'")
119     #Espera de 2 segundos para dejar tiempo a que arranque el
        servidor
120     sleep 2s
121     #Lanzar cliente en la máquina local. Genera log tamporal
122     sshpass -p 1234 ssh $usuario@$LOCAL_IP "sh -c 'ITGSend -a
        $REMOTE_IP -T tcp -c 1500 -C 80000 -t 6000 -l tempsend.log'"
123     #Decodificamos los logs con IGDec
124     sshpass -p 1234 ssh $usuario@$REMOTE_IP "sh -c 'ITGDec
        temprecv.log'" > $PATH_SALIDA/ditg/receiver/ditgrcv$(date
        +%s).log
125     sshpass -p 1234 ssh $usuario@$LOCAL_IP "sh -c 'ITGDec
        tempsend.log'" > $PATH_SALIDA/ditg/sender/ditgsend$(date +%s).log
126     #Borramos el log temporal
127     sshpass -p 1234 ssh $usuario@$LOCAL_IP "sh -c 'rm tempsend.log'"
128     sshpass -p 1234 ssh $usuario@$REMOTE "sh -c 'rm temprecv.log'"
129     #Matamos el proceso del servidor
130     sshpass -p 1234 ssh $usuario@$REMOTE_IP kill $SERVER_PID
131   ;;
132   *)
133     echo "------------------------------------------------"
134     echo "Error con opción -t: El test especificado no existe"
135     list_tests
136     echo "------------------------------------------------"
137     ;;
138 esac
```

B.2 Loop.sh

```
#!/bin/bash
#####
# Este script ejecuta en bucle temporal el test seleccionado de
    launch_test.sh
####
##### CONSTANTES
#Path del script a ejecutar
PATH_SCRIPT=/mnt/hgfs/script/launch_test.sh
#Tiempo entre ejecuciones de Tests
SLEEP_TIME=5s
#Veces que se va a ejecutar el Bucle
ITERATIONS=100
#### MAIN
# PARAMETROS
while getopts ":t:l:r:" op; do
  case ${op} in
    t) #Tipo de test junto la herramienta
      TEST_TYPE=${OPTARG}
      ;;
    l) #Ip origen
      LOCAL_IP=${OPTARG}
      ;;
    r) #Ip destino
      REMOTE_IP=${OPTARG}
      ;;
  esac
done
# BUCLE
for ((i = 1 ; i <= $ITERATIONS ; i++));do
  echo "Counter: $i"
  #Tiempo entre iteraciones
  sleep $SLEEP_TIME
  # Ejecución del scrpt
  $PATH_SCRIPT -t $TEST_TYPE -l $LOCAL_IP -r $REMOTE_IP
done
####
```

Apéndice C

MGEN

Al enfrentarnos a la prueba de la herramienta se encontraron bastantes trabas iniciales para encontrar una versión que compilara sin errores en la máquina de prueba. Gran parte de estos problemas fue la falta de experiencia con proyectos de software libre que no tienen lanzamientos de versiones actualizadas.

Brian Adamson, miembro de la NRL encargado del proyecto de mgen, respondió a mis dudas rápidamente y me indicó que debía usar las nightly-snapshots tanto de Mgen como de la librería de la que depende Protolib para que fuera posible compilarlo en Ubuntu 18.04 (Fig.C.1).

De: **Brian Adamson** | brian.adamson@nrl.navy.mil jueves, 27 de jun de 2019 20:56

Para: **David Ignacio Conde** | d.iconde@udc.es

Have you tried the newest Protolib and MGEN code from our SVN “nightly snapshots”? These issue have probably been addressed in our SVN but not yet pushed into a tarball or whatever source of the MGEN code you are using.

The URLs for the MGEN and Protolib nightly snapshots, respectively, are;

downloads.pf.itd.nrl.navy.mil/mgen/nightly_snapshots/

and

downloads.pf.itd.nrl.navy.mil/protolib/nightly_snapshots/

Note you need to put the “protolib” source tree (or a symbolic link to it) in the top level of the “mgen” source tree for the makefiles to work.

- Brian

Figura C.1: Correo de Brian Adamson dando soporte al problema de compilación.

Apéndice D

RAPR

En un principio se intentó instalar la herramienta, pero sin mucho éxito. Se producían errores de compilación que no se habían solucionado ya que la herramienta lleva sin soporte desde 2008. Este dato fue confirmado por uno de los responsables de mantener el proyecto, como se puede ver en la figura C.1:

Re: Final Degree Project

De: **Brian Adamson** | brian.adamson@nrl.navy.mil — lunes, 11 de feb de 2019 14:21

Para: **David Ignacio Conde** | d.iconde@udc.es

Cc: **rapr_info@nrl.navy.mil** | rapr_info@nrl.navy.mil

HI David,

RAPR is not something we are actively maintaining at the moment. More recently, we have been using the Python mgen.Controller interface (in the mgen source code) to programmatically control MGEN message generation and do similar things we have done with RAPR.

- Brian

Figura D.1: Correo de Brian Adamson sobre el abandono de RAPR.

"RAPR es algo que no estamos manteniendo activamente en este momento. Recientemente hemos estado usando la interfaz Python de mgen.Controller (en el código fuente de MGEN) para controlar programáticamente la generación de mensajes de Mgen y hacer cosas similares a las que hemos hecho con RAPR."

Con lo cual se decidió abandonar la prueba y comparación de esta herramienta.

Apéndice E

Warp17

Aunque Warp17 parecía a priori una herramienta sencilla de cara al comparativo, surgieron problemas a la hora de instalarla en una máquina moderna. Esto provocó que fuera eliminada del comparativo, dado que no se podía usar junto al resto de herramientas.

La instalación se convirtió en un problema dado que las instrucciones para su compilado están mal en el último *commit* del repositorio *git*. Donde se especificaba que una de sus dependencias más importantes, DPDK, debía usar una versión posterior a la que realmente soportaba. A esta conclusión se llegó tras buscar el error mostrado en la compilación [73].

Además en la documentación especifica que se requiere de una máquina Ubuntu 14.04, mientras que la que hemos elegido para el resto de herramientas es 18.04.

Además los intentos de otros usuarios de instalarla en 18.04 fueron un fracaso, como se refleja en el foro de problemas de su respositorio Github [74].

Lista de acrónimos

ELK *Elasticsearch Logstash Kibana.*

NRL *Naval Research Laboratory.*

CORE *Common Open Research Emulator.*

MGEN *Multi-Generator.*

GUI *Graphical User Interface.*

API *Application Programming Interface.*

D-ITG *Distributed Internet Traffic Generator.*

IDT *Inter-Departure Time.*

PS *Packet Size.*

DIETI *Dipartimento di Ingegneria Elettricae delle Tecnologie dell'Informazione .*

FTP *File Transfer Protocol.*

JSON *JavaScript Object Notation.*

ERF *Endace Record Format.*

PCAP *Packet Capture.*

TCP *Transmission Control Protocol.*

UDP *User Datagram Protocol.*

SCTP *Stream Control Transmission Protocol.*

GPL *General Public License.*

SQL *Structured Query Language.*

IPG *Inter-Packet Gap.*

Bibliografía

[1] "D-ITG 2.8.1 Manual - Traffic." [Online]. Available: http://www.grid.unina.it/software/ITG/manual/D-ITG-2.8.1-manual.pdf

[2] "CORE - US Naval Research Laboratory - Navy.mil." [En línea]. Disponible en: https://www.nrl.navy.mil/itd/ncs/products/core

[3] "MGEN - US Naval Research Laboratory - Navy.mil." [En línea]. Disponible en: https://www.nrl.navy.mil/itd/ncs/products/mgen

[4] "RAPR - US Naval Research Laboratory - Navy.mil." [En línea]. Disponible en: https://www.nrl.navy.mil/itd/ncs/products/rapr

[5] "Cisco's TRex." [En línea]. Disponible en: https://trex-tgn.cisco.com/

[6] "WARP17." [En línea]. Disponible en: http://warp17.net/

[7] "The Netperf Homepage - GitHub Pages." [En línea]. Disponible en: https://hewlettpackard.github.io/netperf/

[8] "Socket Interface - an overview | ScienceDirect Topics." [En línea]. Disponible en: https://www.sciencedirect.com/topics/computer-science/socket-interface

[9] "What is iperf / iperf3 ?" [En línea]. Disponible en: https://iperf.fr/

[10] "iperf3 — iperf3 3.7 documentation - ESnet Software." [En línea]. Disponible en: https://software.es.net/iperf/

[11] "D-ITG, Distributed Internet Traffic Generator." [En línea]. Disponible en: http://www.grid.unina.it/software/ITG/

[12] A. Botta, A. Dainotti, and A. Pescapè, "A tool for the generation of realistic network workload for emerging networking scenarios," *Computer Networks*, vol. 56, no. 15, pp. 3531–3547, 2012.

[13] "Elastic Stack: Elasticsearch, Kibana, Beats y Logstash | Elastic." [En línea]. Disponible en: https://www.elastic.co/es/elastic-stack

[14] "iperf3 Project News - ESnet Software." [En línea]. Disponible en: http://software.es.net/iperf/news.html#older-news

[15] "esnet/iperf: iperf3: A TCP, UDP, and SCTP network ... - GitHub." [En línea]. Disponible en: https://github.com/esnet/iperf

[16] "iperf2 / iperf3." [En línea]. Disponible en: https://fasterdata.es.net/performance-testing/network-troubleshooting-tools/iperf/

[17] "Care and Feeding of Netperf 2.7.X." [En línea]. Disponible en: https://hewlettpackard.github.io/netperf/doc/netperf.html

[18] "Documentation - Cisco's TRex ." [En línea]. Disponible en: https://trex-tgn.cisco.com/trex/doc/

[19] "Manual-html - Cisco's TRex." [En línea]. Disponible en: https://trex-tgn.cisco.com/trex/doc/trex_manual.html

[20] "TRex Stateless support - Cisco's TRex." [En línea]. Disponible en: https://trex-tgn.cisco.com/trex/doc/trex_stateless.html

[21] "First time Running." [En línea]. Disponible en: https://trex-tgn.cisco.com/trex/doc/trex_manual.html#_first_time_running

[22] "Linux interfaces." [En línea]. Disponible en: https://trex-tgn.cisco.com/trex/doc/trex_manual.html#_linux_interfaces

[23] "RPC Architecture." [En línea]. Disponible en: https://trex-tgn.cisco.com/trex/doc/trex_stateless.html#_rpc_architecture

[24] "Simulator." [En línea]. Disponible en: https://trex-tgn.cisco.com/trex/doc/trex_appendix_simulator.html

[25] "Tutorial: Simple IPv4/UDP packet simulator." [En línea]. Disponible en: https://trex-tgn.cisco.com/trex/doc/trex_stateless.html#_tutorial_simple_ipv4_udp_packet_simulator

[26] "Memory section configuration." [En línea]. Disponible en: https://trex-tgn.cisco.com/trex/doc/trex_manual.html#_memory_section_configuration

[27] "The 2-Clause BSD License | Open Source Initiative." [En línea]. Disponible en: https://opensource.org/licenses/BSD-2-Clause

[28] "The 3-Clause BSD License | Open Source Initiative." [En línea]. Disponible en: https://opensource.org/licenses/BSD-3-Clause

[29] "NRL Code 5520 Software License - US Naval Research Laboratory." [En línea]. Disponible en: https://www.nrl.navy.mil/itd/ncs/products/license

[30] "Apache License, Version 2.0 - Apache Software." [En línea]. Disponible en: https://www.apache.org/licenses/LICENSE-2.0

[31] "The GNU General Public License v3.0 - GNU ... - GNU.org." [En línea]. Disponible en: https://www.gnu.org/licenses/gpl-3.0.html

[32] "HewlettPackard/netperf: Netperf is a benchmark that ... - GitHub." [En línea]. Disponible en: https://github.com/HewlettPackard/netperf

[33] "cisco-system-traffic-generator/trex-core: trex-core site - GitHub." [En línea]. Disponible en: https://github.com/cisco-system-traffic-generator/trex-core

[34] "Download - D-ITG, Distributed Internet Traffic Generator." [En línea]. Disponible en: http://www.grid.unina.it/software/ITG/download.php

[35] "WHAT IS UNIDIRECTIONAL AND BIDIRECTIONAL NETFLOW?" 2019. [En línea]. Disponible en: https://knowledgebase.paloaltonetworks.com/KCSArticleDetail?id=kA10g000000ClYYCA0

[36] "traffic_config." [En línea]. Disponible en: https://trex-tgn.cisco.com/trex/doc/trex_stateless.html#_traffic_config

[37] "Discuss the Elastic Stack." [En línea]. Disponible en: https://discuss.elastic.co/)

[38] "Working with plugins | Logstash Reference [7.5]." [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/working-with-plugins.html

[39] "Using Netperf to Measure Request/Response." [En línea]. Disponible en: https://hewlettpackard.github.io/netperf/doc/netperf.html#Using-Netperf-to-Measure-Request_002fResponse

[40] "Shell command: date." [En línea]. Disponible en: https://renenyffenegger.ch/notes/Linux/shell/commands/date

[41] "Multiple Pipelines | Logstash Reference [7.5] | Elastic." [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/multiple-pipelines.html

[42] "Persistent Queues | Logstash Reference [7.5] | Elastic." [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/persistent-queues.html

[43] "Input plugins | Logstash Reference [7.5] | Elastic." [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/input-plugins.html

[44] "File input plugin | Logstash Reference [7.5] | Elastic." [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/plugins-inputs-file.html

[45] "File input plugin | Logstash Reference [7.5] | Elastic." [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/working-with-plugins.html

[46] "Logstash Plugins · GitHub." [En línea]. Disponible en: https://github.com/logstash-plugins?page=1

[47] "Elastic Support Matrix." [En línea]. Disponible en: https://www.elastic.co/es/support/matrix#matrix_logstash_plugins

[48] "Grok filter plugin | Logstash Reference [7.5] | Elastic." [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html

[49] "Patrones proporcionados por Logstash." [En línea]. Disponible en: https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns

[50] "Grok Debugger." [En línea]. Disponible en: https://grokdebug.herokuapp.com/

[51] "RegExr: Learn, Build, & Test RegEx." [En línea]. Disponible en: https://regexr.com/

[52] "add_tag." [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html#plugins-filters-grok-add_tag

[53] "match." [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html#plugins-filters-grok-match

[54] "Aggregate filter plugin | Logstash Reference [7.5] | Elastic." [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/plugins-filters-aggregate.html

[55] "Merge information from two different lines into single event." [En línea]. Disponible en: https://discuss.elastic.co/t/merge-information-from-two-different-lines-into-single-event/77791/2

[56] "Combine 2 events into one." [En línea]. Disponible en: https://discuss.elastic.co/t/combine-2-events-into-one/70825/3

[57] "Event API | Logstash Reference [7.5] | Elastic." [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/event-api.html

[58] "map_action." [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/plugins-filters-aggregate.html#plugins-filters-aggregate-map_action

[59] "Aggregate filter plugin | Description." [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/plugins-filters-aggregate.html#plugins-filters-aggregate-description

[60] "Output plugins | Logstash Reference [7.5] | Elastic." [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/output-plugins.html

[61] "Index management | Kibana Guide [7.5] | Elastic." [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/managing-indices.html

[62] "Discover | Kibana Guide [7.5] | Elastic." [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/discover.html

[63] "Date Filter Plugin." [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/plugins-filters-date.html

[64] "Dashboard | Kibana Guide [7.5] | Elastic." [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/dashboard.html

[65] https://www.elastic.co/guide/en/kibana/current/canvas.html, "Canvas | Kibana Guide [7.5] | Elastic."

[66] "Visualize | Kibana Guide [7.5] | Elastic." [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/visualize.html

[67] "Aggregations | Elasticsearch Reference [7.5] | Elastic." [En línea]. Disponible en: https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations.html

[68] "Create a dashboard | Kibana Guide [7.5] | Elastic." [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/dashboard-create-new-dashboard.html#customizing-your-dashboard

[69] "Creating an index pattern | Kibana Guide [7.5] | Elastic." [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/index-patterns.html

[70] P. Shukla and S. K. M. N, *Learning Elastic Stack 7.0*, 2nd ed. Packt, 2019.

[71] C. Lewis and S. Pickavance, "Implementing Quality of Service Over Cisco MPLS VPNs," 2006. [En línea]. Disponible en: http://www.ciscopress.com/articles/article.asp?p=471096&seqNum=6

[72] "TinyMath functions | Kibana Guide [master] | Elastic." [En línea]. Disponible en: https://hewlettpackard.github.io/netperf/doc/netperf.html

[73] "Compile errors in 1.8 · Issue 89 · Juniper/warp17 · GitHub." [En línea]. Disponible en: https://github.com/Juniper/warp17/issues/89

[74] "Ubuntu 18.04 LTS support · Issue 92 · Juniper/warp17 · GitHub." [En línea]. Disponible en: https://github.com/Juniper/warp17/issues/92