UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Disección y Análisis de Tráfico de la Arquitectura SNA sobre Redes IP

Juan Manuel Cornet Recchimuzzi

Tutor: Javier Aracil Rico

Mayo 2016

AGRADECIMIENTOS

A mi familia y amigos.

A Cristina.

A los que tienen ilusión por enseñar.

RESUMEN

Hoy en día todos nuestros dispositivos son capaces de conectarse a una red a través de la cual se comunican con otros terminales. Día a día, a través de las redes, se mueve una cantidad de información enorme, la cual conocemos como tráfico de una red. Para ser capaces de estudiar una red, diagnosticar problemas, prevenir ataques, corregir fallos, etc. es necesario poder investigar el tráfico que se mueve por dicha red.

Sin embargo, como decíamos antes, el volumen de datos es inmenso, y realizar el trabajo de examinar cada traza de tráfico capturado es impensable. Para ello, necesitamos contar con herramientas que puedan capturar y analizar tráfico de una red, que sean capaces de presentar los datos obtenidos de una manera legible para las personas, y que nos facilite la tarea de mantenimiento de una red.

Es ahí donde entran en juego las herramientas de análisis de tráfico, herramientas capaces de realizar capturas de tráfico, analizar dichas capturas y, en algunos casos, incluso plasmar las conclusiones en un documento a modo de informe. Este trabajo pretende navegar por el mundo del análisis del tráfico de redes, teniendo como objetivo el ser capaces de generar informes como los mencionados de manera automatizada.

Esta memoria detalla la labor llevada a cabo para poder generar automáticamente documentos basados en modelos de informe, utilizando archivos de capturas de tráfico de una red. El objetivo final es obtener un documento en formato Word que contenga la información exigida en el modelo de informe, de manera que el informe final pueda ser abierto por aplicaciones externas con las que estamos habituados a trabajar (Open Office, Microsoft Office, etc.) para su posterior estudio y, en caso de ser necesario, edición, con el objetivo de completar aspectos del documento más orientados a la estética o estilo y alejados de la funcionalidad.

El trabajo se ha desarrollado utilizando una herramienta ya existente de captura y procesado de tráfico, proporcionada por el tutor de este proyecto, y una herramienta de codificación propia en Java. La herramienta desarrollada en Java utiliza como entrada los archivos que genera la herramienta de captura de tráfico, y a partir de la información realiza un análisis con distintos algoritmos para obtener los datos requeridos por el modelo de informe utilizado. Por último, la generación automática de un informe se realiza mediante el uso de una API para el manejo de documentos Word (Apache POI), para crear y dar formato a un documento nuevo que contenga la información obtenida.

Cabe destacar que, si bien el objetivo inicial del proyecto era utilizarlo en redes SNA, el carácter que adquirió durante su desarrollo fue más genérico, dejando de lado las especificaciones de SNA. Esto no impide que la herramienta de generación de informes desarrollada, o el ya existente programa de captura de tráfico, puedan ser usados en conjuntos en una red SNA pero teniendo en cuenta que el modelo de informe usado para las pruebas (el implementado actualmente) no ofrece ningún dato específico de una red SNA. Pese a estos cambios en el objetivo del trabajo, el título del proyecto ha permanecido invariable debido a complicaciones administrativas a la hora de cambiarlo por uno más adecuado.

ABSTRACT

Nowadays every device is capable of connecting to a network to communicate with other devices. Each day, a huge amount of data flows through these networks, which we know as network traffic. To be able to study a particular network, diagnose problems, prevent attacks, fix errors, etc. it is necessary to look into the traffic that flows through that network.

However, as we said before, the volume of data managed by a network is incredibly big, and examining each trace of captured traffic is something unthinkable. For this task we must to have tools that can capture and analyze traffic from a specific network and that can display the obtained results in an easy way for people to read it, and that can ease the task of maintaining a network.

This is where traffic capturing and report generating tools come into play, tools capable of creating traffic capture files, analyzing those created files and print the results into a document acting as a report. This project dives into the network traffic analysis area, aiming to create a solution that is capable of automatizing the creation of analysis reports.

This paper details de work carried away to automatically generate reports based on report models, using network traffic capture files. The main goal is obtaining a Word document containing the information demanded by the report model, so the final report may be opened by third party applications (such as Open Office, Microsoft Office, etc.) for editing purposes, if necessary.

This project was developed using an existing tool, provided by the project professor, used to capture and process network traffic and an own-codification tool developed in Java. This tool uses, as its input, the files that the traffic capture tool generates, and beginning with this information, it performs an analysis with different algorithms to obtain the data required by the used report model. To end the process, the tool uses a Word API to create and stylize a new document containing the obtained information.

It's important to know that, even though the project was originally intended to be used in SNA networks, as the development advanced the goal of the project became more generic, leaving aside the SNA specifications. This does not prevent the developed reporting tool or the already existent traffic tool application from working together in an SNA network, but one must take into account the fact that the report model used for the tests (the one that is currently implemented) does not offer any particular data related to SNA networks. Despite this changes in the final goal of the project, the title remained the same due to administrative complications when trying to change it for one that was more fitting.

PALABRAS CLAVE

Modelo de informe, API para Word, Generador Automático, tráfico de una red, informe.

KEYWORDS

Report model, Word API, Automatic Generator, network traffic, report.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO

Término/Abreviatura	**Descripción**
API	Application Programming Interface
libpcap	Implementación Unix de la interfaz “pcap” para la captura de tráfico
.pcap	Extensión de archivos referentes a la captura de tráfico mediante libpcap
Dirección MAC	*Media Access Control*. Refiere al identificador (dirección) de un ordenador utilizado para las comunicaciones a Nivel 2 (físico)
Dirección IP	*Internet Protocol*. Refiere al identificador (dirección) de un ordenador utilizado para las comunicaciones a Nivel 3.(red)
Redes TCP/IP	Redes muy popularizadas basadas en un conjunto de protocolos, entre los que destacan TCP e IP

1. INTRODUCCIÓN

El conocimiento del funcionamiento de una red es adquirido, en gran parte, analizando el tráfico que circula a diario por dicha red. Sin embargo, nadie puede (ni debería) lanzarse a analizar gigas, o incluso magnitudes superiores (tal es el volumen de datos en el tráfico de algunas redes), de información. Es necesario realizar un filtrado previo, un *análisis y disección del tráfico*, para discernir entre información importante e información desechable.

Para realizar este tipo de tareas, se requiere de herramientas específicas que se encarguen tanto de la captura como del posterior procesado de tráfico de red. Si bien existen otras herramientas para realizar este tipo de trabajos, también existen limitaciones en cuanto a la cantidad de tráfico que pueden procesar.

El proyecto realizado en este trabajo, a diferencia de otras herramientas, no ofrece una interacción amplia y flexible con el usuario. Sin embargo, a cambio de sacrificar una elaborada interfaz, ofrece una capacidad de procesamiento superior, ya que los datos capturados en ningún momento son desplegados en pantalla, sino que se analizan de manera automática y se traslada el resultado de dichos análisis a un documento Word.

Si bien el objetivo final del proyectó cambió durante su desarrollo, la propuesta inicial estaba enfocarlo a redes que trabajen bajo la arquitectura SNA[1] (*Systems Network Architecture)*, una arquitectura de redes diseñada y patentada por IBM en los años 70, cuyo propósito es el de ofrecer mecanismos de conexión con los *mainframes* de IBM. Hasta la popularización de las redes TCP/IP, tal y como las conocemos hoy en día, muchas aplicaciones utilizaban la arquitectura SNA y, sin ir más lejos, aún en la actualidad muchos cajeros siguen utilizándola por su mayor seguridad frente al modelo TCP/IP.

El trabajo llevado a cabo en este Trabajo de Fin de Grado permite, tras haber capturado tráfico de una red, generar un documento Word a modo de informe automático. Si bien el trabajo está motivado para su uso en redes que manejan tráfico de la arquitectura SNA encapsulado sobre IP, también puede ser utilizado en cualquier tipo de red más convencional.

1.1. MOTIVACIÓN

La elección de este trabajo está motivada, principalmente, por el interés del autor por las redes de computación.

En cuanto al tema del presente trabajo, la idea de realizar un proyecto que ofrezca facilidades de captura y análisis automático de tráfico de red, surge por la necesidad real de herramientas que ofrezcan este tipo de funcionalidades. Para los administradores y profesionales de redes (ya sea en seguridad u otras áreas) es fundamental conocer qué está ocurriendo en la red en la que trabajan.

Esta necesidad puede surgir con menor o mayor urgencia, en función de si la red se encuentra en funcionamiento de manera normal, o en un estado crítico (continuando con el ejemplo de seguridad en redes, bajo un ataque o una brecha de seguridad). Por este motivo, es fundamental tener acceso a herramientas que permitan, como decíamos, conocer qué sucede en una red.

El motivo por el cual la propuesta de trabajo se planteó, originalmente, enfocado a redes SNA es el hecho de en muchos entornos de hoy en día se sigue trabajando bajo esta arquitectura. Sin embargo, debido al crecimiento y globalización del modelo TCP/IP, es necesario ofrecer soporte a estas aplicaciones, ya que es preciso que se ajusten al mencionado modelo.

Por ello, existen soluciones que encapsulan el tráfico SNA sobre IP, como Enterprise Extender, para ofrecer a estas aplicaciones soporte en las redes modernas. Al existir este tipo de soluciones, también surge la necesidad de analizar el tráfico de estas redes de manera particular, y es ahí donde encuentra su razón de ser el presente proyecto.

Si bien esta es la motivación principal del trabajo, la solución desarrollada a lo largo del mismo permite no solo analizar tráfico de redes SNA, sino también de redes más convencionales o habituales. Por ello, es necesario destacar el carácter más genérico del resultado, que vas más allá de los motivos que en un principio lo impulsaron. Es esta misma ventaja la que nos permite probar el resultado final sin trabajar con tráfico de redes SNA, pero hablaremos sobre ello más adelante en la sección de pruebas.

1.2. OBJETIVOS

El objetivo de este trabajo es el de ofrecer una solución que permita capturar, dada una red cualquiera (o como mínimo una que trabaje con la arquitectura SNA), tráfico basado en el modelo de capas, para su posterior procesamiento y análisis.

Tras haber capturado y categorizado una cantidad significativa de tráfico, se deberá generar un reporte o informe (análisis de los datos recogidos) que ofrezca información relevante sobre la red en la cual se ha realizado el estudio, como pueden ser: direcciones IP con mayor flujo de datos, conversaciones más repetidas en un periodo de tiempo, direcciones MAC con la mayor cantidad de bytes recibidos, etc.

El análisis de la información capturada se realizará de manera automática, y consistirá en un informe detallado por niveles (MAC, IP) que permita incluir gráficas, tablas, estadísticas y cualquier otro tipo de representación relevante de los datos procesados.

La solución ofrecida en este trabajo permite cumplir los objetivos descritos bajo un entorno Linux en cualquier caso, y en entornos Windows de manera más restrictiva (sacrificando algunas funcionalidades, como el uso de gráficas) dentro de una red como la que se ha descrito más arriba o incluso una red sin arquitectura SNA.

1.3. ORGANIZACIÓN DE LA MEMORIA

La memoria de este proyecto consta de seis apartados explicativos principales, los cuales se detallan a continuación:

1. **Estado del Arte**: En este apartado se detallará todo lo relacionado con el contexto bajo el cual se ha desarrollado el proyecto. Se describirá, lo más fielmente posible, tanto el entorno de trabajo como las tecnologías utilizadas.

2. **Diseño**: En esta sección del documento se explicará todo lo relativo al diseño del proyecto. Se incluyen, en este apartado, tanto el diseño del software realizado como el planteamiento con el cual se ha abordado el proyecto. Se aportarán también diagramas (diagramas de clases, esbozos del funcionamiento del software, etc.) cuando se considere útil. Estos diagramas servirán de apoyo visual para comprender a fondo el planteamiento del proyecto.

3. **Desarrollo**: En este apartado se explicará cómo se han llevado a cabo las ideas planteadas en el apartado anterior. Se explicará de qué manera se han implementado, si se ha conseguido cumplir con los objetivos propuestos o no, las posibles modificaciones, etc. Como en el caso anterior, se incluirán (en la mayoría de casos, fragmentos de código) imágenes para explicar mejor determinados conceptos o detalles del apartado.

4. **Pruebas**: En esta sección se detallarán las pruebas realizadas con el software terminado, y se expondrán los resultados más relevantes que se hayan obtenido. Se incluyen, dentro de la batería de pruebas, tanto test unitarios (pruebas de módulos independientes) como de integración (integración de los distintos módulos).

5. **Conclusiones**: En este apartado se comentarán las conclusiones finales a las que se ha llegado tras la realización del proyecto. También se incluirán comentarios sobre posibles mejoras y utilidad del trabajo realizado en un futuro. Se incluirán también, en esta sección, comentarios finales del autor acerca del trabajo.

6. **Anexos**: Los anexos servirán para incluir información relevante para el proyecto, pero cuya naturaleza exige que se haga en una sección diferenciada del resto del cuerpo.

2. ESTADO DEL ARTE

Esta sección del documento está dedicada a realizar una descripción y explicación del contexto en el cual se ha realizado el proyecto, para favorecer una mejor comprensión del mismo.

En las siguientes subsecciones o apartados, explicaremos el entorno en el que se ha realizado el proyecto, las tecnologías y medios utilizados (entornos de desarrollo, lenguajes, herramientas externas, etc.), alternativas o trabajos similares al realizado y cualquier otra información y decisión tomada que sean relevantes para comprender las circunstancias concretas bajo las cuales el proyecto fue desarrollado.

2.1. APLICACIONES SIMILARES

En el marco de alternativas globales al trabajo desarrollado, no se ha encontrado ninguna aplicación relevante que realice una tarea de *reporting* similar al generador automático desarrollado en Java y que sea de código abierto (Open Source).

Sin embargo, en el marco de aplicaciones de pago, sí que se han obtenido algunos resultados. En cualquier caso, las alternativas que se ofrecen a continuación utilizan sistemas diferentes para generar informes, los cuales no se presentan como un documento Word sino como un conjunto de estadísticas y resultados que pueden consultarse a través de la interfaz gráfica de cada aplicación.

2.1.1. PRTG NETWORK MONITOR

PRTG[2] es una herramienta de pago, desarrollada por la empresa Paessler AG, para la monitorización de la actividad en una red. Esta aplicación, al contar con interfaz gráfica, permite visualizar información sobre el tráfico capturado en su propia interfaz, sin tener que generar un documento a modo de informe.

Esta aplicación reúne sus funcionalidades en distintos grupos de “sensores” mediante los cuales monitoriza distintos aspectos de una red. En este sentido, el trabajo desarrollado se parece a esta alternativa en tanto que ambas no son solo herramientas para capturar tráfico sino que representan la información recabada de manera gráfica a modo de estadísticas.

No obstante, el generador de informes desarrollado no se basa en interfaces gráficas para plasmar la información, sino que genera directamente un informe en un documento Word (funcionalidad que no presenta la herramienta PRTG) por lo que no depende de la interacción con un usuario. Además, el proyecto se basa en librerías Open Source tanto para capturar tráfico como para generar informes, por lo que la comprensión del código es accesible a cualquiera con acceso a la documentación en línea de las librerías usadas.

El generador tampoco requiere de conocimientos previos a la hora de seleccionar la información que se va a plasmar en el informe, ya que el modelo de informe está integrado en la aplicación por lo que el usuario no “necesita saber” qué datos debe filtrar para obtener una información concreta. Sin embargo, esta carencia de interfaz gráfica e interacción con el usuario limita al generador a desplegar siempre el mismo modelo con los mismos datos, por lo que también

acarrea una desventaja en cuanto a facilidad de cambiar la información plasmada en el informe final.

En la Figura 1 podemos ver un ejemplo de uno de los tipos de salida que genera el software PRTG, en el que se muestra el volumen de tráfico que abarca cada protocolo para una dirección dada.

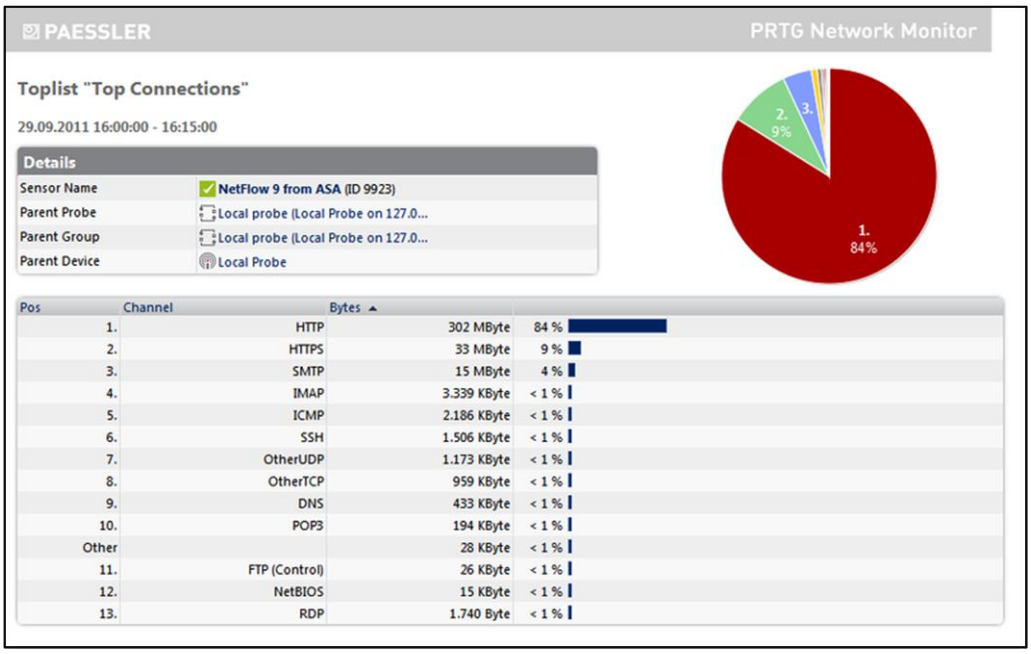


Figura 1. Salida de PRTG Network Monitor

2.1.2. SOLAR WINDS

Solar Winds[3] es una aplicación que ofrece distintos packs dedicados a la monitorización de una red, cada uno de los cuales ofrece funcionalidades distintas en función de la información final que el usuario quiera obtener. En una analogía con la aplicación comentada en el anterior apartado, cada pack de Solar Winds correspondería a un conjunto de sensores de PRTG dedicados a una tarea concreta dentro de la monitorización.

Esta es una alternativa muy similar a la anterior PRTG, ya que los datos analizados también son desplegados en una interfaz propia en lugar de en un documento Word, lo cual como vemos es una característica particular de este proyecto. Sin embargo, aunque la manera de plasmar la información final sea distinta, sí que comparte muchas similitudes, en cuanto a funcionalidades, con el generador de informes desarrollado. A continuación se listan algunos de los datos que Solar Winds es capaz de obtener y que también se incluyen en el modelo de informe de pruebas del generador:

- Top 10 direcciones/aplicaciones en generación de tráfico
- Top 10 direcciones/aplicaciones en recepción de tráfico
- Top 10 direcciones/aplicaciones en generación de paquetes
- Top 10 direcciones/aplicaciones en recepción de paquetes
- Top 10 conversaciones en la red

Además de estas funcionalidades incluye otras más avanzadas, como la programación de alertas o la restricción de uso del ancho de banda a ciertas aplicaciones, etc. Algunas de estas funcionalidades se incluyen en el pack de monitorización básico y otras en packs orientados a tareas más específicas, como el pack de optimización de una red.

En la Figura 2 presentamos, como ejemplo, la visualización gráfica que Solar Winds ofrece de algunos de los top 10 listados más arriba (a la derecha) así como otros datos relacionados con las diferentes interfaces de red monitorizadas (a la izquierda).

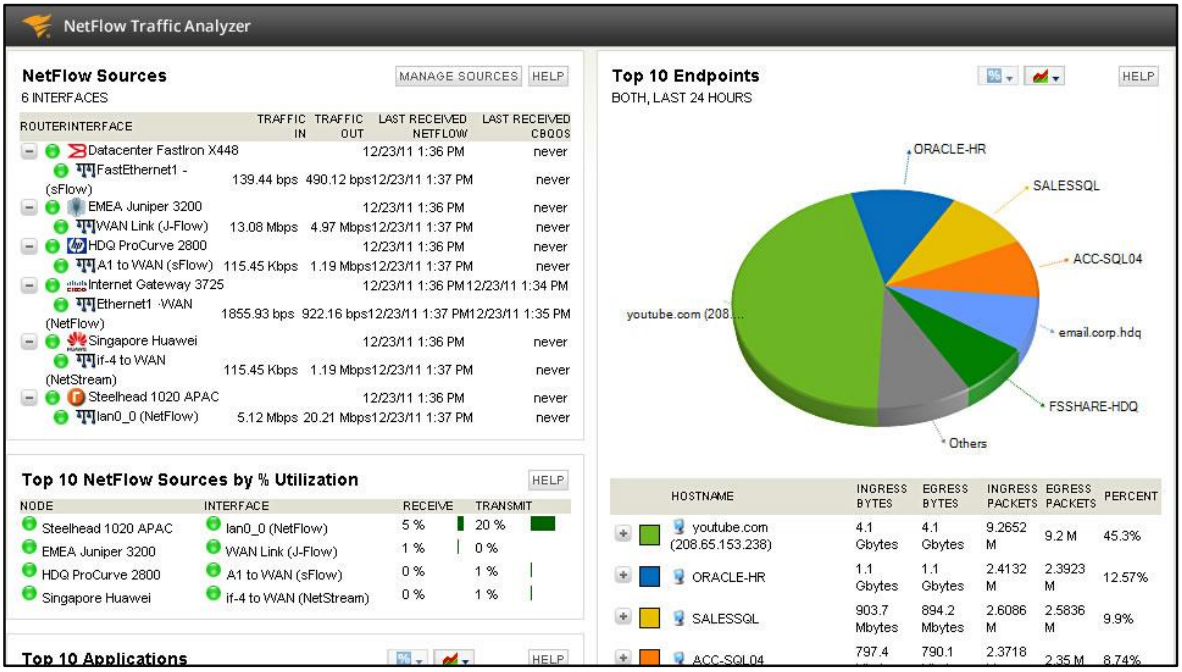


Figura 2. Salida de Solar Winds

2.2. PROCESATRAZAS

Para poder lograr el objetivo de este trabajo, necesitamos una herramienta que sea capaz de capturar tráfico de una red. En nuestro caso, esta funcionalidad la desempeñará el ProcesaTrazas, un programa proporcionado por el tutor de este trabajo y desarrollado con anterioridad por el grupo HPCN de la Universidad Autónoma de Madrid y el grupo GRSST de la Universidad Pública de Navarra. ProcesaTrazas no solo captura tráfico, sino que posteriormente ofrece un breve resumen de los datos obtenidos (si bien de manera poco conveniente).

El programa está desarrollado en C y cuenta con diversos módulos de los que se vale para realizar las tareas explicadas; justamente es el hecho de que esté desarrollado en C lo que ha condicionado la decisión de desarrollar el generador de informes principalmente orientado a entornos basados en Linux, ya que en C las librerías usadas para capturar tráfico y analizarlo corresponden librerías propias de estos sistemas.

Si bien ProcesaTrazas es un programa carente de interfaz gráfica, cuenta con un potente núcleo capaz de procesar grandes ficheros de trazas y ofrecer gran cantidad de información de salida acerca de los paquetes capturados. Esta carencia de GUI no implica complicación más allá del simple hecho de que debe ejecutarse a través de una consola de comandos, por lo que, como vemos, es indiferente a la complejidad del proyecto.

En última instancia, y siempre teniendo como contexto el uso que le damos al programa en este proyecto, la herramienta volcará una salida en un fichero de texto. El objetivo de nuestro generador de informes automáticos será el de leer ese fichero de texto y procesarlo, de manera que pueda crear un sencillo pero bien estructurado informe, fácilmente legible por una persona.

2.3. JAVA

De la gran variedad de lenguajes existentes, cada uno con sus puntos fuertes y débiles, se ha escogido Java[4] para desarrollar el proyecto.

Esta decisión viene fuertemente condicionada por el hecho de que nuestro objetivo consiste en enlazar la parte del proyecto dedicada a la captura y análisis de tráfico (ProcesaTrazas) con la generación automática de informes. Por tanto, el lenguaje elegido para esta segunda fase del proyecto debe tener las facilidades necesarias para enlazar con la primera parte, además de cumplir otros requisitos adicionales, como pueden ser el hecho de tener conocimientos y experiencia en el uso del lenguaje o que incluya funcionalidades que faciliten el desarrollo de la aplicación.

Con esto en mente, se han tomado en cuenta las siguientes consideraciones a la hora de elegir un lenguaje:

- Lenguaje de desarrollo del ProcesaTrazas (C).
- SO para el cual se ha desarrollado el ProcesaTrazas (Linux)
- Facilidad de incorporar una API para la generación de documentos

Por estos motivos, se ha elegido Java como candidato ideal, ya que su interoperabilidad entre plataformas nos permite desarrollar en cualquier Sistema Operativo (si bien las pruebas con el ProcesaTrazas deben ser en un entorno Linux, por el uso de librerías exclusivas de este SO).

En cuanto a la incorporación de una API, existen diversas maneras, pero la más sencilla consiste en incluir una librería en forma de uno o varios archivos .jar a nuestro proyecto Java y añadir las dependencias correspondientes. Este será el método que utilizaremos en el trabajo.

El último requisito a cumplir es poder enlazar con un programa hecho en C (ProcesaTrazas). En principio esto puede parecer un problema al haber elegido Java, pero sin embargo, es un obstáculo con el cual cualquier lenguaje escogido se hubiese topado, ya que, salvo que eligiésemos continuar el desarrollo en C, siempre íbamos a tener que enlazar con un lenguaje distinto. La solución a este problema será comentada más adelante en los capítulos de diseño y desarrollo.

2.4. API WORD PARA JAVA

Tras tomar la decisión de llevar a cabo el proyecto en el lenguaje Java, la siguiente incógnita a resolver es de qué manera concreta integraremos la generación de documentos en dicho lenguaje.

Para este motivo, utilizaremos una API para crear documentos pertenecientes a la categoría de Office, ya sea OpenOffice o Microsoft Office (MS Office). Entre estos documentos se incluyen:

- Documentos tipo Word (.doc, .docx)
- Documentos tipo Excel (.xls, .xlsx)

Los tipos de documento mencionados son los que serán relevantes para el proyecto, ya que serán los que la aplicación podría llegar a generar de manera automática.

En nuestra búsqueda de una API para el propósito planteado, se tomarán en cuenta solamente aquellas soluciones de código abierto (Open Source) y se admitirán, puntualmente, APIS con licencia de pago exclusivamente si pertenecen a Microsoft.

A continuación se expondrán las principales soluciones a la cuestión, así como sus ventajas e inconvenientes y se señalará la opción u opciones escogidas (es posible y quizás hasta conveniente utilizar diversas APIs para lograr obtener el mayor número posible de funcionalidades, siempre que no exista ningún inconveniente de compatibilidad entre ellas) y el por qué.

2.4.1. OPENOFFICE UNO

UNO[5], cuyas siglas significan Universal Network Object, es el modelo de interfaz para los componentes de OpenOffice, y permite manejar los mismos en forma de objetos extrapolables entre sistemas operativos, lenguajes y entornos de programación.

Como se ha dicho, UNO es una solución quizás demasiado compleja para el problema planteado, ya que podría decirse que la potencia de UNO no se limita solamente a la generación de documentos desde un programa. Las bases de OpenOffice están asentadas sobre UNO, conteniendo incluso su propio lenguaje (OpenOffice Basic), entorno de desarrollo, manual de desarrollador, kit de desarrollador, etc.

Las principales ventajas de escoger esta solución es la enorme flexibilidad que ofrece: dado que UNO puede trabajar al nivel más bajo con los componentes de OpenOffice, permite realizar cualquier tarea que pudiéramos imaginar. Sin embargo, la curva de aprendizaje de este framework es extremadamente amplia, y el tiempo requerido para poder familiarizarse con la misma es, lamentablemente, demasiado.

A pesar de ser una solución inviablemente compleja, UNO orienta la búsqueda de una API en una dirección más sencilla y menos ambiciosa, más adecuada al tipo de solución que queremos construir.

2.4.2. DOCX4J

Docx4j[6] (también docx4java) es una solución más ajustada a las necesidades de este trabajo. A diferencia de UNO, esta API está en forma de librería desarrollada principalmente en Java, y permite al usuario trabajar con algunos de los tipos de documentos propios Office y documentos PDF.

Docx4j se centra en tareas más comunes, como la creación y edición (a niveles moderados) de documentos de tipo Word, Excel, PowerPoint, etc. Para los documentos Word, sin embargo, ofrece posibilidades más amplias que el nivel básico de edición, como pueden ser: trabajar con plantillas o templates, exportaciones a nivel de PDF, personalización y estilo del documento, etc.

Para otros tipos de documentos existen extensiones específicas que amplían las funcionalidades básicas, como son pptx4j o xlsx4j, que ofrecen la posibilidad de centrar el trabajo en torno a documentos de tipo PowerPoint y Excel, respectivamente.

Esta librería está basada en trabajar con los "paquetes" XML debajo de cada documento Word (*WordprocessingML*[7]), por lo cual ofrece un grado de flexibilidad bastante bueno, a cambio de estar dispuesto a trabajar con estructuras de tipo XML.

Podríamos concluir que docx4j es una solución más específica y centrada que UNO, sacrificando flexibilidad en favor de una mayor facilidad en su uso, aunque sería una conclusión muy simplista. En realidad, UNO es mucho más que una solución para construir y editar documentos tipo Office de manera programática, por eso sería una comparación injusta poner a docx4j al nivel de UNO.

Aun a pesar de que esta librería se presenta más "amigable" de cara al desarrollador y su curva de aprendizaje no es tan pronunciada como la de UNO, no es esta la solución elegida puesto que docx4j depende mucho de JAXB[8], estándar para procesamiento de XML en Java en el cual está basada esta librería. Esta dependencia de JAXB hace que se requiera más tiempo para el aprendizaje, y si bien JAXB no es demasiado complejo de usar de manera básica, la familiarización con el mismo plantea otro obstáculo más, lo cual nos desviaría de la tarea inicial.

Dentro de la lista de contras de docx4j, cabe mencionar también que la manera en la que se tratan los documentos, gracias al procesamiento del XML que los compone por debajo, no es una forma demasiado intuitiva de tratar documentos tipo Office (si bien permite un tratamiento efectivo), además de añadir problemas de compatibilidad ya que la estructura WordprocessingML de un documento .docx varía según la versión de Word.

2.4.3. APACHE POI

Entre las opciones más populares y recomendadas para este tipo de problemas, encontramos Apache POI[9], una librería bajo Licencia Apache[10] que ofrece exactamente lo que el proyecto requiere: edición moderada de documentos de tipo Word (creación de títulos, párrafos, inserción de imágenes, tablas, etc.) y, de manera complementaria, Excel (creación de gráficas y diagramas) de manera programática, a través de una API intuitiva para el desarrollador. Adicionalmente, esta librería cuenta con otros componentes que permiten abrir y editar otros tipos de documentos Office, pero los cuales no son de interés para el presente proyecto.

Si bien esta solución también está basada en el trabajo con el XML bajo los documentos Office, esta librería ofrece una API más transparente e intuitiva de cara el desarrollador, ocultando los procesos y modificaciones en XML que realiza por debajo para realizar las tareas requeridas. Podría decirse, de hecho, que Apache POI trabaja en un nivel de abstracción mayor al del resto de APIs.

Como vemos, esta es la solución más sencilla y adecuada a nuestras necesidades, ya que cuenta con las mismas ventajas (en cuanto a funcionalidades) que docx4j pero siendo más intuitiva para el desarrollador. Por ello, esta será la API que emplearemos en el desarrollo de nuestra herramienta.

2.4.4. JAVA2WORD

Java2Word[11] es, al igual que Apache POI, una librería pura para Java que ofrece lo que se esperaría de este tipo de API: herramientas para manipular documentos de tipo Office, más concretamente, documentos tipo Word.

Esta API es un proyecto independiente mantenido por una sola persona, a diferencia de las otras alternativas, las cuales son desarrolladas por empresas o equipos bajo licencias Open Source. Sin embargo, comparte el mismo principio que Apache POI, y es el hacer la interfaz para el programador lo más transparente e intuitiva posible, siendo así una de las APIs de la lista más fáciles de aprender.

Siguiendo con las similitudes a la anterior propuesta, Java2Word también se presenta en forma de librería pura (.jar), la cual puede integrarse de manera directa en nuestro proyecto Java.

Sin embargo, existe poca documentación acerca de esta API por el simple hecho de ser mantenida como proyecto independiente. Esto no solo añade trabas a su aprendizaje, sino que disminuye el volumen de soporte online (ayudas, FAQ, guías, manuales, foros, etc.) que pueda tener esta API. Esto la convierte en una apuesta arriesgada, ya que los problemas por falta de soporte saldrían a la luz en pleno desarrollo, etapa en la cual puede ser preciso disponer de una documentación técnica detallada y otro tipo de ayudas.

Como vemos, Java2Word es una alternativa prácticamente idéntica a Apache POI, con menos soporte y con la excepción de que solo puede manejar documentos de Word, lo cual deja fuera de su alcance las gráficas y diagramas en Excel de los cuales hablábamos antes. Dado que estas representaciones gráficas de los datos son una parte fundamental de este proyecto, y por los inconvenientes ya mencionados, esta opción fue descartada en favor de la más adecuada Apache POI.

2.5. GNUPLOT

GNUPlot[12] es un software de código libre (Open Source) muy popularizado, cuya finalidad es la representación de gráficas, ya sea en dos o tres dimensiones.

Este programa cuenta con muchas funcionalidades a la hora de representar gráficas (líneas suavizadas, barras, representación de distintas funciones en una misma gráfica, etc.) y si bien en el futuro este trabajo podría llegar a hacer uso de muchas de las herramientas que ofrece GNUPlot, en la actualidad solo nos interesan algunas de las funcionalidades básicas, como: representación de una única función, nombre en los ejes X e Y, título de la función, etc.

Para integrarlo en nuestra solución, bastará con que el sistema Linux en el que se despliegue el proyecto tenga instalado GNUPlot, pues la manera de enlazar con este software será la de ejecutar, de manera programática, los comandos de consola que permitan generar las gráficas deseadas.

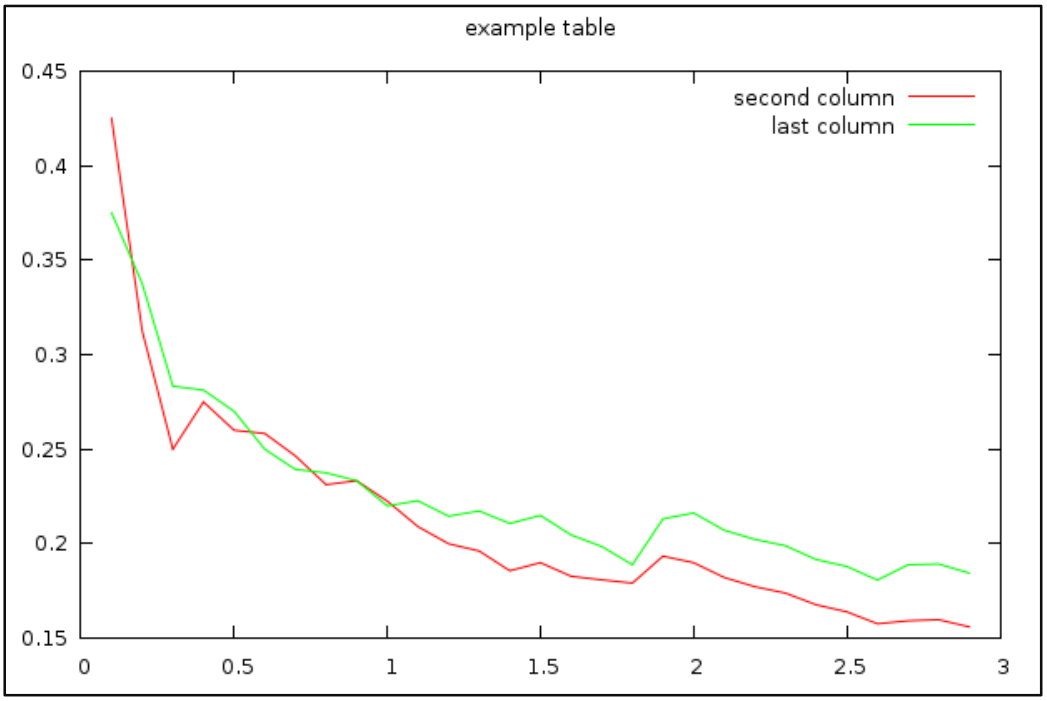


Figura 3. Ejemplo de gráfica de GNUPlot

2.6. ECLIPSE

Uno de los entornos de desarrollo (IDE) gratuitos y más populares entre los desarrolladores Java es Eclipse[13]. Si bien posee distintas versiones y plugins que le permiten acomodarse a todo tipo de desarrollos (C, C++, aplicaciones web, aplicaciones paralelas, etc.), para este proyecto no nos interesa ninguna funcionalidad añadida, por lo que nos limitaremos a su versión más simplificada y conocida como un sencillo pero potente IDE para Java que utilizaremos para desarrollar la parte de este trabajo correspondiente al análisis y generación de informes automáticos.

La elección de este software se debe principalmente a que es con el cual el autor se siente más cómodo y familiarizado, dado que es el que se utiliza oficialmente en la Escuela Politécnica Superior para impartir clases de Java.

No obstante, este no es el único motivo del uso de Eclipse, sino que también se debe a que, al ser la IDE de Java más popularizada, cuenta con una increíble cantidad de soporte (información técnica, guías y otro tipo de ayudas en foros de pregunta y respuesta) en línea, lo cual facilita enormemente el uso de capacidades nuevas o en las cuales el autor del trabajo tiene poco conocimiento.

Por último, pero no menos importante, también existen facilidades (principalmente en forma de software complementario al entorno) ofrecidas por Eclipse (como la intuitiva y rápida inclusión de plugins o desarrollo de los mismos) que podrían resultar útiles de cara al futuro trabajo y ampliación del proyecto (por ejemplo inclusión de interfaz gráfica, análisis estructural del código, estudio de la corrección del diseño del proyecto, etc.), y que se comentarán en la sección de este documento correspondiente.

Podemos observar, en la Figura 4, el aspecto general que tiene el proyecto Java de este trabajo (paquetes de código con su clases, librerías, editor de código, interfaz del IDE, etc.) abierto en Eclipse.

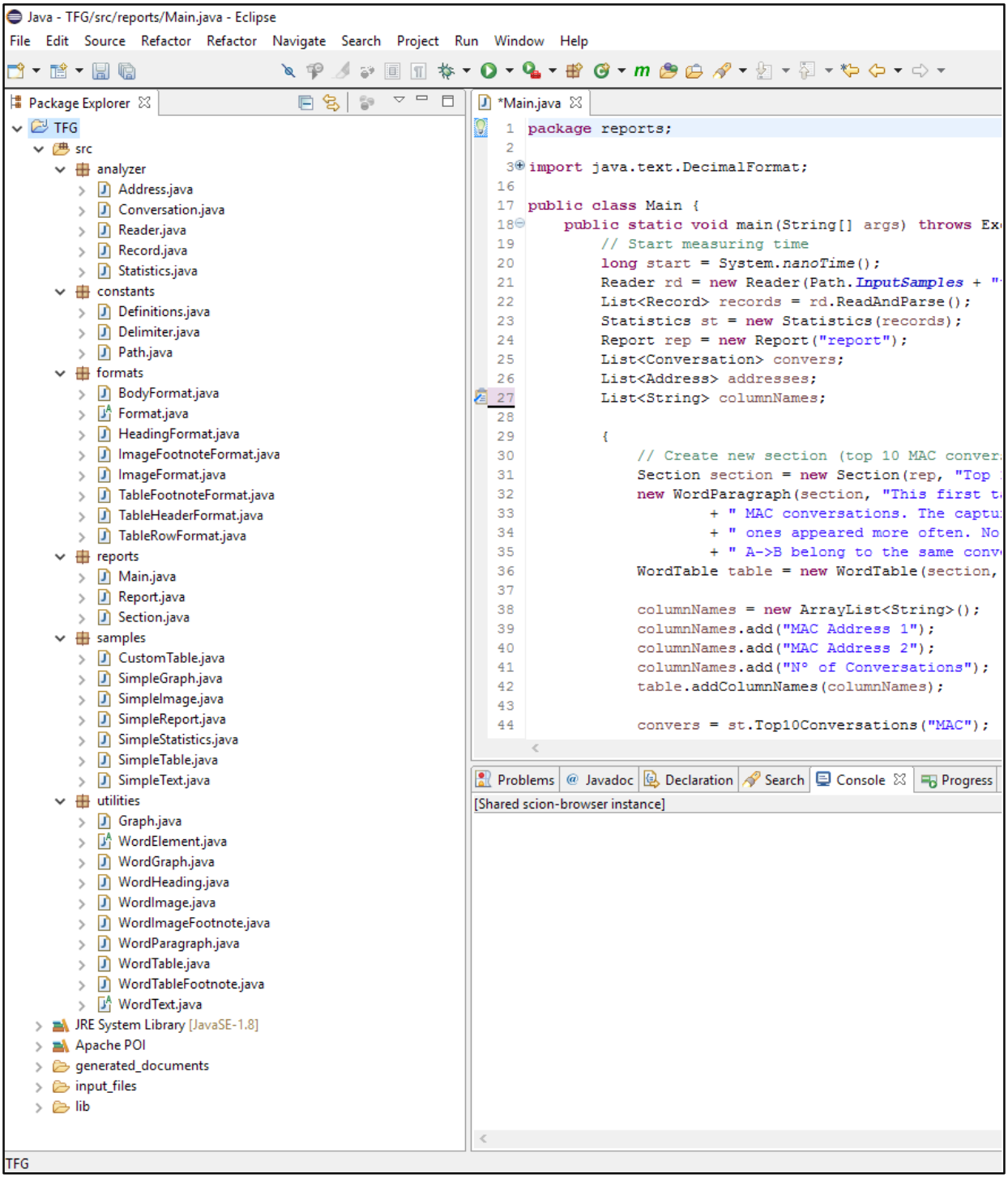


Figura 4. Proyecto JAVA en Eclipse

3. DISEÑO

Como ya se ha comentado, en esta sección vamos a exponer el planteamiento del proyecto. Se distinguirá entre el diseño del software (a nivel de diagramas de clase, relaciones entre módulos, soluciones y especificaciones técnicas, etc.) y el diseño más "general" del proyecto, significando este último, de manera simplificada, el modo en el que se va a abordar el proyecto. En este último caso nos referimos al enfoque general con el cual se ha afrontado el proyecto.

3.1. DISEÑO GENERAL

Para abordar el proyecto, tenemos que simplificar la manera en la que se entienden las distintas tareas o partes que lo componen.

Podemos decir, en líneas generales, que el proyecto se compone de dos grandes secciones:

- Captura y procesado de tráfico
- Análisis del tráfico capturado y generación de informes

En conjunto ambas partes consiguen la tarea de, partiendo de cero, generar un informe fácilmente legible del estado de una red o de un fichero que contenga tráfico (en forma de paquetes) capturado en una red.

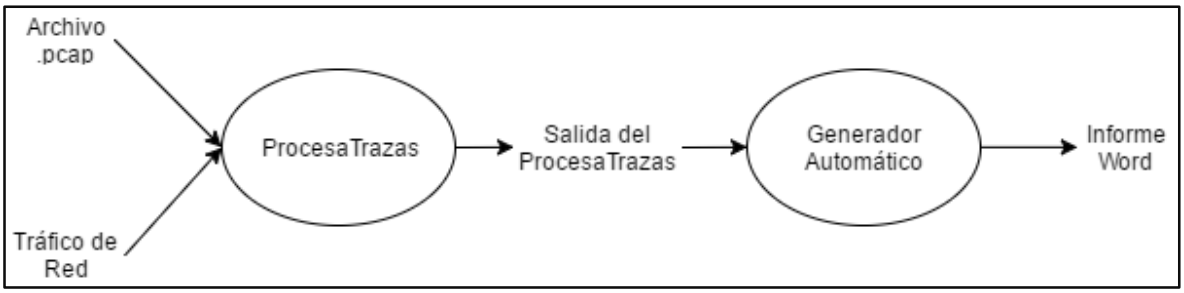


Figura 5. Diseño general del proyecto

En nuestro caso, cada una de las dos fases por las que pasan los datos está representada por una elipse en la Figura 5, en la cual podemos ver un esquema de los pasos que siguen los datos para transformarse de tráfico de red (ya se capturado o desde un archivo) hasta un documento Word a modo de informe.

3.1.1. CAPTURA Y PROCESADO DE TRÁFICO

La primera parte o paso del proyecto, secuencialmente hablando, se corresponde con la captura de tráfico. Para analizar el estado de una red, como ya se ha mencionado, ha de diseccionarse su tráfico (capturarlo y analizarlo), y esto es lo que se consigue con aplicaciones como Wireshark[14]. En este trabajo, la herramienta utilizada para este propósito ProcesaTrazas, mencionada en el apartado **2.2 ProcesaTrazas**.

Gracias a la librería *libpcap*[15], este programa es capaz de capturar tráfico sobre el cual realiza, posteriormente, una primera aproximación de "análisis" o mejor dicho, agrupación de la información: procesa los datos y genera una salida en texto plano, demasiado primitiva como para poder extraer conclusiones.

```
1  192.168.89.2 53298 192.168.0.3 5000 1445495637.722583000 1445495641.726757000 1445495637.722583000 -1 -1 -
   156147678 -1 65535 -1 65535 0 0 230491 230527 0 2 1445499447.596855000 -1 -1 156147679 -1 0 0 0 0 0 0 0 0
   0 0 (null) 0 0 (null) 0 0 0 0 0 0 0 0 32 0 0 0:(null) 0:(null) 0 0 (null) (null) 2 0 0 (null) 0 (null) 0 0 0
2  192.168.88.52 53299 192.168.1.71 5000 1445495671.435516000 1445495678.452260000 1445495671.435516000 -1 -1
   0 4046315473 -1 65535 -1 65535 0 0 230760 230873 0 2 1445499447.596855000 -1 -1 4046315474 -1 0 0 0 0 0 0
   0 0 0 0 (null) 0 0 (null) 0 0 0 0 0 0 0 32 0 0 0:(null) 0:(null) 0 0 (null) (null) 3 0 0 (null) 0 (null) 0
3  192.168.88.52 53301 192.168.0.3 5000 1445495695.421633000 1445495702.430162000 1445495695.421633000 -1 -1
   0 1835158784 -1 65535 -1 65535 0 0 231096 231175 0 2 1445499447.596855000 -1 -1 1835158785 -1 0 0 0 0 0 0
   0 0 0 0 (null) 0 0 (null) 0 0 0 0 0 0 0 32 0 0 0:(null) 0:(null) 0 0 (null) (null) 4 0 0 (null) 0 (null) 0
4  192.168.88.52 53302 192.168.0.3 5000 1445495761.165335000 1445495768.170351000 1445495761.165335000 -1 -1
   0 335005460 -1 65535 -1 65535 0 0 231766 231841 0 2 1445499447.596855000 -1 -1 335005461 -1 0 0 0 0 0 0 0
   0 0 0 (null) 0 0 (null) 0 0 0 0 0 0 0 32 0 0 0:(null) 0:(null) 0 0 (null) (null) 5 0 0 (null) 0 (null) 0 0
5  192.168.88.52 53303 192.168.1.68 5000 1445495796.634746000 1445495803.581730000 1445495796.634746000 -1 -1
   0 2544605147 -1 65535 -1 65535 0 0 232080 232163 0 2 1445499447.596855000 -1 -1 2544605148 -1 0 0 0 0 0 0
   0 0 0 0 (null) 0 0 (null) 0 0 0 0 0 0 0 32 0 0 0:(null) 0:(null) 0 0 (null) (null) 6 0 0 (null) 0 (null) 0
6  192.168.88.52 53305 192.168.0.3 5000 1445495851.227121000 1445495858.234072000 1445495851.227121000 -1 -1
   0 3170723609 -1 65535 -1 65535 0 0 232568 232648 0 2 1445500447.599931000 -1 -1 3170723610 -1 0 0 0 0 0 0
   0 0 0 0 (null) 0 0 (null) 0 0 0 0 0 0 0 32 0 0 0:(null) 0:(null) 0 0 (null) (null) 7 0 0 (null) 0 (null) 0
7  192.168.88.52 53306 192.168.1.71 5000 1445495887.550271000 1445495894.549953000 1445495887.550271000 -1 -1
   0 3470656320 -1 65535 -1 65535 0 0 232887 232951 0 2 1445500447.599931000 -1 -1 3470656321 -1 0 0 0 0 0 0
   0 0 0 0 (null) 0 0 (null) 0 0 0 0 0 0 0 32 0 0 0:(null) 0:(null) 0 0 (null) (null) 8 0 0 (null) 0 (null) 0
8  192.168.88.52 53307 192.168.1.71 5000 1445495967.080657000 1445495974.086216000 1445495967.080657000 -1 -1
```

Figura 6. Salida del ProcesaTrazas

Como vemos en la Figura 6 la salida del ProcesaTrazas contiene demasiada información en bruto como para que un usuario pudiera trabajar con ella. Es por este mismo motivo por el cual existe una segunda fase que recoge toda esta información y automatiza el análisis de los datos útiles, para generar posteriormente un informe sencillo del que extraer conclusiones.

3.1.2. ANÁLISIS Y GENERACIÓN DE INFORMES

Como comentábamos al principio de esta sección, la siguiente fase en la línea del trabajo está dedicada a tratar los datos obtenidos en la primera.

Con esto se pretende analizar toda la información en bruto obtenida por el ProcesaTrazas, y presentarla de manera que resulte fácil y cómoda de analizar, por ejemplo en un informe compuesto de distintas secciones, para representar cada conjunto de datos, como la que vemos en la Figura 7. Nótese la diferencia entre los datos en bruto, presentados anteriormente, y un análisis presentado de manera sencilla y legible.

Top 10 IP Conversations

IP Address 1	*IP Address 2*	*Nº of Conversations*
192.168.0.226	192.168.0.254	4
192.168.0.257	192.168.0.254	3
192.168.0.254	192.168.0.254	2
192.168.0.226	192.168.0.226	1

Figura 7. Sección de un informe

La siguiente pregunta qué debemos hacernos es ¿qué deseamos conocer acerca de la red? Esta es una pregunta trampa, pues existen varias respuestas en función de factores como pueden ser, entre otros:

- Objetivo que se quiere alcanzar al analizar el tráfico
- Tipo de tráfico que estamos analizando (protocolos, encriptaciones, etc.)
- Tamaño del archivo o cantidad de tráfico que se analiza
- Persona que va a estudiar el informe

Como es lógico, un experto en seguridad no buscará de esta herramienta las mismas capacidades o resultados que un técnico de sistemas que analiza fallas en la red que administra, o alguien que se dedica a hacer estudios de rendimiento de una red.

Por lo tanto, queremos responder, en última instancia, a la pregunta ¿qué datos aparecerán en el informe generado? Una vez más, la respuesta tiene trampa: en nuestro caso nos interesará, para las pruebas, un informe más enfocado en conexiones TCP basado en un modelo de prueba, ya que será más ilustrativo.

Teniendo decidido que tráfico vamos a analizar en nuestro caso, podemos plantear un modelo de informe para las pruebas, el cual se detalla a continuación.

3.1.3. MODELO DE INFORME PARA LAS PRUEBAS

El modelo de informe utilizado consistirá, de manera simplificada, en distintas secciones de un documento Word, cada una de las cuales constará de:

- Título de sección
- Breve párrafo descriptivo
- Una o más tablas de datos (si procede)
- Una o más gráficas de datos (si procede)
- Una o más imágenes (si procede)
- Cualquier otro elemento de texto

En el informe generado se distinguirán dos grandes ámbitos, en los cuales se presentará distinta información pero correspondiente a un mismo "grupo" de datos:

- **Nivel MAC:** En esta mitad del informe se presentan datos extraídos tras realizar un análisis en basado en las direcciones MAC (Nivel 2) de los dispositivos que participaban en la comunicación de información en la red.
- **Nivel IP:** En esta segunda mitad se presentan las conclusiones extraídas tras realizar un análisis en basado en las direcciones IP (Nivel 3) de los dispositivos que participaban en la comunicación de información en la red.

En la Tabla 1 encontramos agrupadas las distintas secciones que incluirá el modelo de informe de pruebas, si bien se omiten, en favor de una mayor brevedad y adecuación al contenido de esta memoria, las descripciones detalladas de cada sección (las cuales pueden encontrarse, de cualquier modo, en los informes de prueba generados).

Nivel MAC	Top 10 conversaciones MAC
	Top 20 conversaciones MAC responsables del 99% del tráfico
	Top 10 direcciones MAC de origen en producción de bytes
	Top 10 direcciones MAC de origen en producción de paquetes
	Top 20 direcciones MAC de origen responsables del 99% del tráfico
	Top 10 direcciones MAC de destino en recepción de bytes
	Top 10 direcciones MAC de destino en recepción de paquetes
	Top 20 direcciones MAC de destino responsables del 99% del tráfico
Nivel IP	Top 10 conversaciones IP
	Top 20 conversaciones IP responsables del 99% del tráfico
	Top 10 direcciones IP de origen en producción de bytes
	Top 10 direcciones IP de origen en producción de paquetes
	Top 20 direcciones IP de origen responsables del 99% del tráfico
	Top 10 direcciones IP de destino en recepción de bytes y paquetes
	Top 10 direcciones IP de destino en recepción de paquetes
	Top 20 direcciones IP de destino responsables del 99% del tráfico

Tabla 1. Modelo de informe para las pruebas

3.2. DISEÑO DEL SOFTWARE

Una vez especificadas las líneas generales del diseño del trabajo, pasamos a detallar el diseño del software en sí, es decir, de qué manera afrontaremos el desarrollo de las tareas planteadas al explicar el funcionamiento del proyecto.

Para abordar este apartado lo dividiremos, una vez más, en dos subsecciones principales, cada una de ellas dedicadas a una de las dos grandes fases que definimos con anterioridad. Por tanto, explicaremos de manera distinguida la fase de captura y procesado de tráfico (tareas competencia del ProcesaTrazas) y el análisis y generación de informes de manera automática (tareas competencia del generador de informes).

3.2.1. PROCESATRAZAS

ProcesaTrazas es el software externo proporcionado por el tutor (como comentábamos en el punto **2.2 ProcesaTrazas**), encargado de diversas tareas referentes al estudio del tráfico de una red, que se emplea en este trabajo para capturar y procesar, inicialmente, el tráfico de una red.

El software consta de distintos módulos encargados de diversas tareas, las cuales todas en conjunto ofrecen la posibilidad de analizar trazas capturadas, si bien el formato de salida de los datos concluidos (el resultado del análisis de una traza) es bastante rudimentario, constando solo de salida por pantalla sin ningún tipo de descripción+, presentación o explicación. Opcionalmente la salida por pantalla puede volcarse en un archivo de texto.

Si bien el programa cuenta con distintos módulos ejecutables, la manera principal (y por tanto la que nos interesa) en la que se emplea en este trabajo es ejecutando el módulo *procesaConexiones*. Este módulo, a su vez, cuenta con muchas opciones o parámetros diferentes, las cuales pueden especificarse al ejecutarlo de la manera estándar para un programa en Linux, indicando la opción con un signo menos delante (p.ej.: -v -s -q), y el argumento requerido (si procede).

En la Tabla 2 podemos observar las principales opciones bajo las cuales se puede ejecutar este módulo.

Opción	Argumento	Descripción
-f	input_file	Fichero de traza a analizar
-p	formato	Formato de traza (pcap, raw)
-u	NC	La traza es una lista de ficheros
-d	pkts_a_descartar	Paquetes a ignorar
-o	output_file_conexiones	Fichero donde volcar la salida
-v	NC	Salida verbose (leyenda para los datos)
-t	pcap_filter	Aplica a la traza ese filtro pcap
-s	NC	Muestra progreso por stderr
-g	NC	Saca al final estadísticas globales por stderr
-a	maxTiempoInactividad	Tiempo inactividad conexiones
-k	listaProtos	Activa el procesado de ciertos protocolos
-m	NC	Contar todos los bytes a nivel de enlace
-n	MaxTimeInterPktsCnx	Máximo numero de segundos entre paquetes de una misma conexión
-h	NC	Ayuda explicando opciones y formato de salida

Tabla 2. Opciones de ejecución del ProcesaConexiones

La manera "estándar" de ejecución del módulo, que nos interesa para este proyecto, es la que sigue:

```
./procesaConexiones –f ficheroTrazas –o ficheroSalida –s
```

Con este comando ejecutamos el módulo en cuestión, indicando que debe analizar un fichero con tráfico ya capturado (-f), volcar la salida del análisis en un fichero externo (-o) y ofrecer un informe del estado de ejecución de manera periódica (-s).

Como vemos en el comando anterior, en el trabajo no se emplea, para las pruebas, la funcionalidad de captura de tráfico del ProcesaTrazas, sino que se utilizan, en su lugar, capturas de tráfico públicas obtenidas en Internet. Utilizar el programa de esta manera tiene principalmente dos ventajas: reducimos el tiempo de ejecución y a la vez que podemos corroborar los resultados de manera más fiable al utilizar siempre el mismo fichero de trazas.

En la Figura 8 podemos observar la salida estándar del módulo *procesaConexiones* al ejecutarlo con la opción -s, con sus correspondientes informes del estado de la ejecución.

```
1877438 pkts diffPkts: 100371 pkts/s: 864 pktTime: 1445513273 (13:27:53) diffPktTime: 12221 cnxInMem: 267083
(volcadas: 505209 diffVolcadas: 465) (cleaning: 0.01 segs, 1 pasadas) presentTime: 1451840625.057128 difRealT
ime: 116.12
1879490 pkts diffPkts: 2052 pkts/s: 683 pktTime: 1445513290 (13:28:10) diffPktTime: 12238 cnxInMem: 268118 (v
olcadas: 505209 diffVolcadas: 0) (cleaning: 0.00 segs, 0 pasadas) presentTime: 1451840628.057222 difRealTime:
 3.00
1881477 pkts diffPkts: 1987 pkts/s: 661 pktTime: 1445513307 (13:28:27) diffPktTime: 12256 cnxInMem: 269119 (v
olcadas: 505209 diffVolcadas: 0) (cleaning: 0.00 segs, 0 pasadas) presentTime: 1451840631.059654 difRealTime:
 3.00
1883499 pkts diffPkts: 2022 pkts/s: 673 pktTime: 1445513324 (13:28:44) diffPktTime: 12272 cnxInMem: 270134 (v
olcadas: 505209 diffVolcadas: 0) (cleaning: 0.00 segs, 0 pasadas) presentTime: 1451840634.061032 difRealTime:
 3.00
1885514 pkts diffPkts: 2015 pkts/s: 671 pktTime: 1445513341 (13:29:1) diffPktTime: 12289 cnxInMem: 271138 (vc
lcadas: 505209 diffVolcadas: 0) (cleaning: 0.00 segs, 0 pasadas) presentTime: 1451840637.061391 difRealTime:
3.00
1887467 pkts diffPkts: 1953 pkts/s: 650 pktTime: 1445513358 (13:29:18) diffPktTime: 12306 cnxInMem: 272117 (v
olcadas: 505209 diffVolcadas: 0) (cleaning: 0.00 segs, 0 pasadas) presentTime: 1451840640.062350 difRealTime:
 3.00
```

Figura 8. Módulo ProcesaConexiones con opción -s

Una variante con la que se puede ejecutar el módulo y que puede resultar particularmente interesante para nuestro trabajo es como sigue:

```
./procesaConexiones –f ficheroTrazas –o ficheroSalida –s -v
```

La opción -v añade al documento de salida una pequeña "leyenda"; esto es, antes de cada dato, ofrece una palabra clave que lo describe. Esta es una funcionalidad que nos interesa especialmente, puesto que aunque su objetivo no es otro que hacer el "informe" del ProcesaTrazas más legible, nosotros utilizaremos esta "leyenda" para detectar los elementos que nos interesan y obtener sus valores respectivos.

Se ofrece una muestra de la salida generada por el módulo, ejecutado con la opción -v, en la Figura 9. Es interesante comparar esta salida con la de la Figura 6.

```
1  192.168.89.2 53298 192.168.0.3 5000 5firstPacketTime: 1445495637.722583000 7lastPacketTime: 1445495641.726757000 9f
   13firstACKSrcToDstTime: -1 15firstACKDstToSrcTime: -1 17firstDataSrcToDstTime: -1 19firstDataDstToSrcTime: -1 21las
   1445495641.726757000 27lastDstToSrcTime: -1 29numberPacketsSrcToDst: 5 31numberPacketsDstToSrc: 0 33numberSYNSrcToD
   41numberRSTSrcToDst: 0 43numberRSTDstToSrc: 0 45numberPacketsDataSrcToDst: 0 47numberPacketsDataDstToSrc: 0 49numbe
   55numberHolesDstToSrc: 0 57numberDisorderSrcToDst: 0 59numberDisorderDstToSrc: 0 61bytesIPSrcToDst: 220 63bytesIPDs
   71bytesTCPDataDstToSrc: 0 73bytesIPDisorderSrcToDst: 0 75bytesIPDisorderDstToSrc: 0 77bytesTCPDisorderSrcToDst: 0 7
   85minWindowSrcToDst: 65535 87minWindowDstToSrc: -1 89maxWindowSrcToDst: 65535 91maxWindowDstToSrc: 0 93firstPacketF
   101reasonDump: 2 103whenDecissionDump: 1445499447.596855000 105lastAckNumberSrcToDst: -1 107lastAckNumberDstToSrc:
   113numberReTxSrcToDst: 0 115bytesTCPReTxSrcToDst: 0 117bytesPhyReTxSrcToDst: 0 119huboHuecosSrcToDst: 0 121numberRe
   127huboHuecosDstToSrc: 0 129bytesPhyDisorderSrcToDst: 0 131bytesPhyDisorderDstToSrc: 0 133timeAckSrcToDstForLastDat
   139timeLastAckSrcToDst: -1 141timeFirstFINDstToSrc: -1 143timeLastAckDstToSrc: -1 145maxBlockedTimeSrcToDst: 0 147t
   151timeMaxBlockedTimeDstToSrc: -1 153srcMACSrcToDst: 70:71:bc:3a:0d:e8 155dstMACSrcToDst: 00:0a:dc:64:85:c2 157srcM
   163numPktsFragmentos: 0 165numPktsWin0SrcToDst: 0 167numPktsWin0DstToSrc: 0 169maxTimeSinceWin0SrcToDst: 0 171maxTi
   175numValuesTimeSinceWin0SrcToDst: 0 177valuesTimeSinceWin0SrcToDst: (null) 179numCierresWin0DstToSrc: 0 181numValu
   185numberDataKeepAliveSrcToDst: 0 187numberDataKeepAliveDstToSrc: 0 189numberTimesWindowDownSrcToDst: 0 191numberTi
   197numPktsCabeceraIncompleta: 0 199windowScaleSrcToDst: 32 201windowScaleDstToSrc: 0 203numPktsCabeceraTCPIncomplet
   209numPktsFueraVentanaSrcToDst: 0 211numPktsFueraVentanaDstToSrc: 0 213numDupsSrcToDst: (null) 215numDupsDstToSrc:
   223VLANidSrcToDst: (null) 225enMasVLANsSrcToDst: 0 227VLANidDstToSrc: (null) 229enMasVLANsDstToSrc: 0 231minBytesEn
   237minBytesEntrePSHDstToSrc: 0 239maxBytesEntrePSHDstToSrc: 0 241mediaBytesEntrePSHDstToSrc: -nan 243GRESrcToDst: 0
   249timeFirstRSTDstToSrc: -1 251appLevelProto: (null)
2  192.168.88.52 53299 192.168.1.71 5000 5firstPacketTime: 1445495671.435516000 7lastPacketTime: 1445495678.452260000
   13firstACKSrcToDstTime: -1 15firstACKDstToSrcTime: -1 17firstDataSrcToDstTime: -1 19firstDataDstToSrcTime: -1 21las
   1445495678.452260000 27lastDstToSrcTime: -1 29numberPacketsSrcToDst: 7 31numberPacketsDstToSrc: 0 33numberSYNSrcToD
   41numberRSTSrcToDst: 0 43numberRSTDstToSrc: 0 45numberPacketsDataSrcToDst: 0 47numberPacketsDataDstToSrc: 0 49numbe
   55numberHolesDstToSrc: 0 57numberDisorderSrcToDst: 0 59numberDisorderDstToSrc: 0 61bytesIPSrcToDst: 308 63bytesIPDs
   71bytesTCPDataDstToSrc: 0 73bytesIPDisorderSrcToDst: 0 75bytesIPDisorderDstToSrc: 0 77bytesTCPDisorderSrcToDst: 0 7
   85minWindowSrcToDst: 65535 87minWindowDstToSrc: -1 89maxWindowSrcToDst: 65535 91maxWindowDstToSrc: 0 93firstPacketF
   101reasonDump: 2 103whenDecissionDump: 1445499447.596855000 105lastAckNumberSrcToDst: -1 107lastAckNumberDstToSrc:
```

Figura 9. Salida de ProcesaConexiones con opción –v

Es especialmente importante hacer hincapié en el formato de salida del ProcesaTrazas. Como hemos visto, resulta imposible extraer conclusiones de unos datos presentados en forma de ficheros de texto plano y que no contienen de manera explícita la información final que buscamos, por lo que es vital tratarlos con un programa externo que permita dar forma a la información recopilada.

Sin embargo, es justamente esta desventaja la que aprovecharemos para enlazar con la segunda fase del proyecto, ya que el formato de fichero de texto plano nos permitirá abrir estos datos en un programa externo (el generador de informes) para posteriormente procesarlos, realizar el análisis correspondiente al modelo de informe y, por último, presentarlos de una manera más adecuada.

3.2.2. GENERADOR DE INFORMES

Una vez tenemos el tráfico capturado y el ProcesaTrazas ha generado su salida, el siguiente paso (la tarea principal de este proyecto) consiste en el procesado, análisis y presentación de los datos extraídos anteriormente mediante el uso de una herramienta de generación de informes de manera automatizada.

Como se ha comentado en otras secciones, este generador de informes (Generador Automático de ahora en adelante) está desarrollado en Java y enlaza con la salida del ProcesaTrazas mediante la lectura de ficheros de texto plano.

La dificultad de esta tarea no radica solo en conseguir generar documentos mediante una API que permita crear archivos Word; es decir, no se trata únicamente del trabajo de integrar una API y ser capaces de utilizarla, sino que también implica una tarea de recogida de los datos generados en la primera fase y un procesado selectivo: el propio análisis de la información.

Para poder generar un informe como el que describíamos en la sección **3.1.3 Modelo de Informe para las Pruebas** es necesario realizar diversas operaciones (matemáticas, de ordenación, filtrado, etc.) que permitan obtener los números y resultados que queremos plasmar, en última instancia, en el informe. En resumen, existe una tarea de preparación de la información obtenida del ProcesaTrazas, ya que los datos y conclusiones que queremos reflejar en el informe no aparecen de manera explícita en la salida de dicho programa, sino que deben extraerse tras una serie de pasos.

A nivel de proyecto software en Java, el Generador Automático se ha planteado como un proyecto dividido en distintos módulos. A continuación, se listan y describen cada uno de esos módulos:

- **Pruebas de la API:** La tarea de este módulo es la de realizar pruebas de funcionalidad sobre la API que se ha integrado en el proyecto para el tratamiento de documentos Word. Su complejidad debería mantenerse reducida, y debería realizarse en primer lugar, puesto que permite al desarrollador familiarizarse con la API.
- **Abstracción de la API:** Este módulo tiene como tarea la de realizar una encapsulación de las funciones que incluye la API para Word. Es decir, el objetivo que se pretende conseguir es simplificar al máximo la interacción con la API para que crear, por ejemplo, una imagen o una tabla en Word no implique más que una o dos funciones. De esta manera se consigue una mayor transparencia de cara al desarrollo final y a la inclusión de elementos en el informe generado.
- **Constantes de Formatos:** Para poder mantener una uniformidad y coherencia en todo el informe generado (coherencia de estilo, tamaños, fuente, etc.), se define este módulo el cual englobará todo el código necesario para asegurar que el documento permanece fiel a un estilo en toda su extensión, y que ofrece una apariencia coherente de principio a fin. Para ello, este módulo debe abarcar todas las tareas de formateado de párrafos, tablas, imágenes y otros elementos que se incluyan en el informe final, y ofrecer una capa transparente a la hora de dar formato a los elementos introducidos en el informe.

- **Análisis de Datos de Entrada:** Este módulo agrupa todas las tareas que deben realizarse sobre los datos de entrada (procedentes de la salida del ProcesaTrazas) para generar la información que realmente se representará en el informe. Dicho de otro modo, este módulo contiene el código necesario para tratar los datos de entrada y obtener todos los valores que se piden representar en el modelo de informe de, e insertarlos en forma de elementos de un documento Word.
- **Constantes y Herramientas:** En este módulo del proyecto Java estarán todos los fragmentos de código auxiliar que no encajen de manera lógica en ningún otro módulo, o que puedan describirse como herramientas de ayuda, funciones estáticas o de uso único, constantes globales, rutas de acceso, etc.
- **Reportes:** Todas las elementos encargados de estructurar el modelo de un informe o reporte, como definir de qué elementos está compuesto un informe (secciones, títulos, descripciones, etc.) o su contenido, así como el punto de ejecución principal de la herramienta, estarán agrupados en este módulo.

En base a lo descrito en la anterior lista, podemos confeccionar un pequeño diagrama que representa, a grandes trazos, la composición del proyecto Java. En la Figura 10, podemos diferenciar módulos funcionales (en rectángulos), módulos "de apoyo" (en nubes) y módulos de prueba (rectángulo de borde doble).

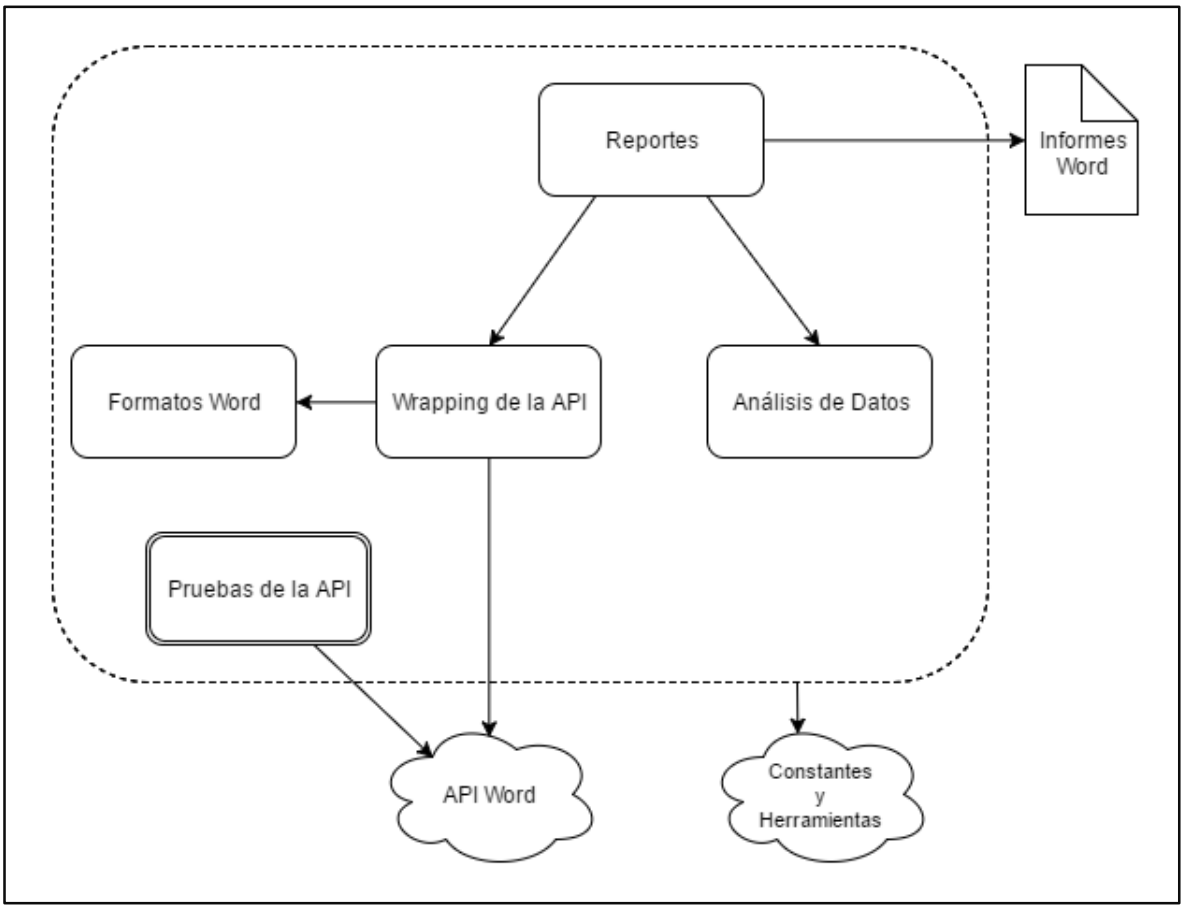


Figura 10. Diagrama general del proyecto Java

Tras haber definido de manera general las bases sobre la que asentaremos el proyecto Java, podemos pasar a definir un diagrama de clases (con posibilidad de ampliación o modificaciones), lo cual nos permite concretar a un nivel mayor las ideas definidas vagamente en los párrafos anteriores. Este diagrama, reflejado en la Figura 11, se encuentra simplificado debido a las limitaciones (en cuanto a dimensiones) que implica plasmarlo en papel, por lo que no se incluyen ni atributos ni métodos de las clases, además de excluir las clases de pruebas.

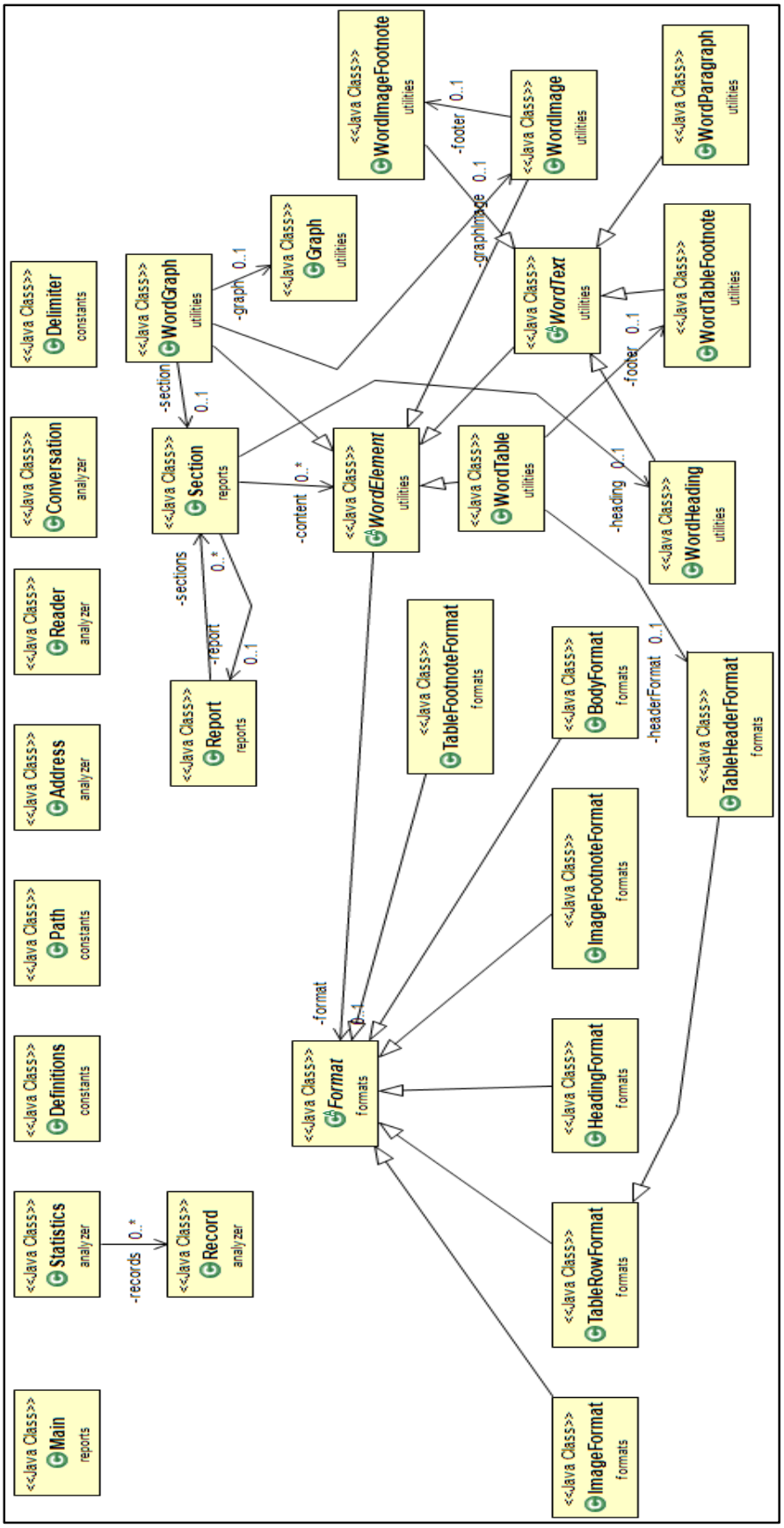


Figura 11. Diagrama de clases simplificado

4. DESARROLLO

Llegados a este punto de la memoria, comentaremos en los próximos apartados como se han llevado a cabo las ideas planteadas en el apartado de diseño. Esta sección del documento se dividirá en varias subsecciones, las cuales pretenden reflejar el orden de pasos seguidos durante el desarrollo del proyecto de manera cronológica.

Además de explicar la manera en la que se ha desarrollado el trabajo, también podremos ver con qué nivel de exactitud se han seguido los planes propuestos en el apartado anterior.

4.1. GENERACIÓN DE FICHEROS CON PROCESATRAZAS

El primer paso seguido al desarrollar el trabajo fue el de la familiarización con la ya tan mencionada herramienta de procesado de archivos de captura de tráfico (ficheros .pcap).

Los primeros pasos consisten en ejecutar distintas veces el programa para familiarizarnos con la salida, así como entender para qué sirve cada una de las opciones de ejecución de las que dispone. Como se ha comentado, la opción de ejecución más importante para nuestro cometido es la de -v, ya que nos abre una posibilidad que se explica en el siguiente punto.

4.1.1. FICHEROS DE CONVERSACIONES CLAVE-VALOR

El módulo del ProcesaTrazas que en concreto utilizamos en el proyecto es el *procesaConexiones*. Como su propio nombre ya nos indica, este módulo se encarga de analizar un fichero de captura de tráfico (o directamente desde la red) y procesa las distintas conexiones (o conversaciones) que encuentra en él.

Como sabemos, dos ordenadores pueden establecer una conexión TCP a través de la red, y durante esa conexión se produce el envío de mensajes en forma de paquetes entre ambos extremos de la conexión, uno de los cuales es llamado *cliente* y el otro *servidor*.

El módulo encargado de procesar estas conexiones realiza un volcado de datos en un fichero de salida en base a las distintas conversaciones que deduce en un fichero de entrada. En este fichero de salida podremos encontrar una fila por cada una de las conversaciones que ha encontrado en el fichero .pcap de entrada, y en cada una de estas líneas, encontraremos datos (separados por espacios) que incluyen, entre otros:

- **IP cliente:** Dirección IP del extremo de la conexión que actúa como cliente.
- **IP servidor:** Dirección IP del extremo de la conexión que actúa como servidor.
- **MAC cliente:** Dirección MAC del extremo de la conexión que actúa como cliente.
- **MAC servidor:** Dirección MAC del extremo de la conexión que actúa como servidor.
- **Bytes cliente-servidor:** Indica la cantidad de bytes que se han observado en la dirección cliente -> servidor.
- **Bytes servidor-cliente:** Indica la cantidad de bytes que se han observado en la dirección servidor -> cliente.

Si bien estos datos se han presentado en la memoria de manera sencilla, no todos están grabados de manera explícita en el fichero de salida, sino que es necesario analizar varios valores por cada conversación para poder obtenerlos. Por ejemplo, la cantidad de bytes que envía uno de los dos extremos se encuentra divida entre muchas categorías (bytes a nivel IP, bytes a nivel TCP, etc.) y en muchos casos alguno de los datos necesarios puede sencillamente no figurar (aparecer como una referencia nula) si el procesador de conexiones no ha sido capaz de extraer esa información.

Ahora bien, sabiendo que existe una línea por cada conversación observada, necesitaremos procesar cada uno de los datos que aparecen en estas pero filtrando solamente aquella información que nos interese, ya que por cada línea pueden aparecer hasta más de 250 datos referentes a esa conversación: ahí es donde entra en juego la mencionada opción -v.

Gracias a esta opción, cada uno de los datos de una conversación viene precedido por una pequeña "leyenda" o mejor dicho, una etiqueta o *clave*. Por tanto ahora cada fila del fichero no corresponde sólo a datos separados por espacios, sino también a etiquetas, las cuales podemos encontrar de antemano en la documentación de ayuda del ProcesaTrazas. Con esto conseguimos saber qué etiquetas esperar y por tanto con qué nombre hemos de buscarlas al procesar el fichero.

El formato de cada línea ha pasado de ser

valor1 valor2 valor3

a

clave1 *valor1* **clave2** *valor2* **clave3** *valor3*

Algo importante de estas claves, es que cada dato sobre la conversación es único, es decir, no vendrá informado dos veces. Por esto, cada una de las claves también será única, lo cual nos permitirá encontrar cualquier dato que deseemos buscándolo en la línea de la conversación mediante su clave única.

Quizá el lector ya haya imaginado que una de las ventajas de utilizar este sistema de pares clave-valor es que podremos utilizar, en el desarrollo del Generador Automático, estructuras basadas en claves hash para optimizar las búsquedas de los datos que nos interesan de las conversaciones.

Sabiendo la estructura de los datos que vamos a procesar con el Generador Automático, y teniendo una noción de cómo podemos hacer este procesado de manera eficiente (mapas hash), el siguiente paso es pasar a la codificación.

4.2. GENERACIÓN AUTOMÁTICA DE INFORMES

Con una idea clara de la entrada del Generador Automático, y teniendo un modelo de informe (definido en el apartado **3.1.3 Modelo de Informe para las Pruebas**) al que aspirar como salida, se comienza con el desarrollo del generador.

Esta implementación estará dividida en módulos, como se comentó con anterioridad, y por tanto dedicaremos una sección distinta a la mayoría de ellos, o mejor dicho, a los que implican mayor tiempo, complejidad y, sobre todo, funcionalidad.

4.2.1. WRAPPING DE LA API PARA WORD

La primera tarea a llevar a cabo es la de asentar las bases para generar un documento Word. Como se comentó con anterioridad, se eligió Apache POI por ser una de las APIs con mayor facilidad de uso, una flexibilidad bastante alta y por tener una curva de aprendizaje moderada. Además, existe una cantidad considerable de documentación y ayudas on-line, las cuales han sido de gran ayuda a la hora de realizar la codificación, como se había supuesto.

Si bien Apache POI es una librería bastante potente y, dentro de lo que cabe, relativamente amigable para su uso, ha exigido una gran cantidad de tiempo poder familiarizarse con ella, ya que la documentación no está integrada en la librería, si no que se encuentra diseminada por internet.

Para dar los primeros pasos con esta API, se desarrollaron tests específicos dentro del módulo de pruebas que se encargaban de probar distintas funcionalidades (crear texto, crear una tabla, insertar una imagen, etc.).En la Figura 12 y Figura 13 podemos ver fragmentos de algunos de dichos tests.

```
public class SimpleTable {
    public static void main(String[] args)throws Exception
       {
       // Blank Document
       XWPFDocument document= new XWPFDocument();

       // Write the Document in file system
       FileOutputStream out = new FileOutputStream(new File(Path.OutputSamples+"simpletable.docx"));

       // Create table
       XWPFTable t1 = document.createTable();

       XWPFTableRow tr1 = t1.getRow(0);
       XWPFParagraph p1 = null;

       // IMPORTANT code sample to get the default paragraph in a table cell
       // getParagraphArray(0) DOES NOT WORK
       for(XWPFParagraph p : tr1.getCell(0).getParagraphs()){
           p1 = p;
       }

       tr1.getCell(0).setVerticalAlignment(XWPFVertAlign.CENTER);
       p1.setAlignment(ParagraphAlignment.CENTER);

       XWPFRun r1 = p1.createRun();
       r1.setText("Hello.");
       tr1.addNewTableCell().setText("What's up?");
       tr1.addNewTableCell().setText("Goodbye!");

       // Write in the document and close it, as well as the output stream
       document.write(out);
       out.close();
       document.close();

       System.out.println("simpletable.docx written succesfully");
       }
}
```

Figura 12. Creación de una tabla de una celda

```
public class SimpleText
{
   public static void main(String[] args)throws Exception
   {
   // Blank Document
   XWPFDocument document= new XWPFDocument();

   // Write the Document in file system
   FileOutputStream out = new FileOutputStream(new File(Path.OutputSamples+"simpletext.docx"));

   // Add text in the 1st line of 1st paragraph
   XWPFParagraph p1 = document.createParagraph();
   XWPFRun r1 = p1.createRun();
   r1.setBold(true);
   r1.setFontSize(21);
   r1.setText("Hello, I'm a paragraph.");
   r1.addBreak();

   // Add text in the 2nd line of 1st paragraph
   XWPFRun r12 = p1.createRun();
   r12.setFontSize(21);
   r12.setBold(true);
   r12.setText("I'm still the same paragraph!");

   // Add text in 1st line of 2nd paragraph
   XWPFParagraph p2 = document.createParagraph();
   XWPFRun r2 = p2.createRun();
   r2.setBold(true);
   r2.setFontSize(21);
   r2.setText("Good bye");

   // Write in the document and close it, as well as the output stream
   document.write(out);
   out.close();
   document.close();

   System.out.println("simpletext.docx written succesfully");
   }
}
```

Figura 13. Creación de dos párrafos sencillos

Como se puede apreciar, el uso de la API no es algo tan transparente como cabría esperar (algunos nombres pueden resultar confusos, se necesitan demasiadas operaciones, etc.), por ello el primer paso a dar fue el de realizar un "wrapping" o encapsular funcionalidades de la librería en un módulo de codificación propia. Con esto conseguimos dos objetivos fundamentales:

- Abstraer un nivel más la creación y edición de documentos Word mediante una API, reduciendo la complejidad y número de operaciones necesarias para llevar a cabo diversas tareas.
- Encapsular solamente las funcionalidades requeridas por el proyecto, descartando aquellas operaciones que, si bien son admitidas por Apache POI, no son relevantes al proyecto.

Como vemos, el objetivo prácticamente es el de crear una "librería" propia, aunque esta palabra realmente es un concepto muy ambicioso para el encapsulado que se ha hecho, ya que como decíamos, no todas (ni mucho menos) de las funcionalidades que Apache POI ofrece se han incluido en el wrapping.

En este encapsulamiento, se han incluido principalmente las siguientes funciones:

- **Gráficas:** Se han incluido las gráficas como un elemento de documentos Word, si bien se ha hecho con componentes externos además de los proporcionados por la API. Tal como se había comentado con anterioridad, se utiliza GNUPlot de manera externa para la realización de gráficas, las cuales son posteriormente tratadas como imágenes dentro de la API. Esto se debe principalmente a que utilizar un componente externo resulta más sencillo que el componente de gráficas para Word o Excel integrado en la API (recordemos que Apache POI no solo permite interactuar con Word, sino también con otros programas integrantes de Office).
- **Imágenes:** El uso e inclusión de imágenes es una de las funcionalidades principales que queremos añadir a nuestro wrapping, puesto que no solo las gráficas están basadas en este componente, sino que la inclusión de datos en un informe mediante una imagen es algo realmente útil y deseable.
- **Texto:** Por su puesto cualquier documento necesita, como medida básica, la inclusión de texto plano en modo de párrafos. Esta es una de las características fundamentales del encapsulado ya que permite abrir también otras funcionalidades que se basan en ella.
- **Títulos:** Se soporta el uso de títulos para encabezar apartados o secciones de los documentos. Este elemento está basado en el componente de texto, realizando algunas modificaciones.
- **Anotaciones:** Al igual que con los títulos, las anotaciones a pie de imágenes o tablas, se soporta modificando el componente de texto.
- **Tablas:** El componente por excelencia para la representación de datos, el cual además es el componente principal del modelo de informe de prueba mencionado con anterioridad. Las tablas no solo son uno de los elementos más útiles, si no también uno de los más complejos a la hora de utilizarlos mediante Apache POI, por el gran número de factores que hay que tener en cuenta: bordes, espacios superiores e inferiores, tamaño de celdas, alineación de texto y de tabla, formato de texto en las celdas, etc.

En las siguientes figuras se presentan pequeños extractos del código de las clases de Java que encapsulan las funcionalidades de gráficas, imágenes y tablas citadas anteriormente.

```
public class Graph {
    private String title;
    private String xName;
    private String yName;
    private List<Double> xData;
    private List<Double> yData;

    public Graph(String t, String xN, String yN, List<Double> xD, List<Double> yD){
        this.title = t;
        this.xName = xN;
        this.yName = yN;
        this.xData = xD;
        this.yData = yD;
    }

    public void draw() throws Exception{
        Runtime rt = Runtime.getRuntime();
        String folder = Path.OutputGraphs+this.title;

        rt.exec("mkdir "+folder);

        folder = folder.concat("/");

        FileWriter outData = new FileWriter(folder+this.title+".gdata");
        FileWriter outCommands = new FileWriter(folder+this.title+".gcom");

        for(int i = 0; i < this.xData.size(); i++){
            outData.write(this.xData.get(i)+" "+this.yData.get(i));
            outData.write("\n");
        }

        this.writeCommands(outCommands);

        rt.exec("gnuplot "+folder+this.title+".gcom");

        outData.close();
        outCommands.close();
    }
```

Figura 14. Extracto de código del manejo de gráficas

```
public WordImage(Section s, String n, int t) throws Exception {
    super(new ImageFormat());
    this.paragraph = s.getReport().getDocument().createParagraph();
    this.container = this.paragraph.createRun();
    this.name = n;
    this.type = t;

    // Assign default width and height. The user might change this values in the future
    this.width = Definitions.DefaultImageWidth;
    this.height = Definitions.DefaultImageHeight;

    s.addContent(this);
}

public WordImage(Section s, String n, int t, String f) throws Exception {
    super(new ImageFormat());
    this.paragraph = s.getReport().getDocument().createParagraph();
    this.container = this.paragraph.createRun();
    this.name = n;
    this.type = t;

    // Assign default width and height. The user might change this values in the future
    this.width = Definitions.DefaultImageWidth;
    this.height = Definitions.DefaultImageHeight;

    this.footer = new WordImageFootnote(s, f);

    s.addContent(this);
}

public void insert() throws Exception {
    FileInputStream in = new FileInputStream (Path.InputSamples+this.name);
    ImageFormat format = (ImageFormat)this.getFormat();

    this.paragraph.setAlignment(format.alignment);

    this.container.addPicture(in, this.type, this.name, Units.toEMU(this.width), Units.toEMU(this.height));
}
```

Figura 15. Extracto de código del manejo de imágenes

```
public void addColumnNames(List<String> names){
    XWPFTableRow firstRow = this.table.getRow(0);
    XWPFParagraph par = getCellParagraph(firstRow.getCell(0), 0);
    XWPFRun run = par.createRun();

    // Create the columns
    for(String name : names){
        run.setText(name);

        // If the are more names, add a new cell, get it's paragraph and create the new run
        if(names.indexOf(name) < (names.size()-1)){

            par = getCellParagraph(firstRow.addNewTableCell(), 0);
            run = par.createRun();
        }
    }

    // Style the column names
    for(XWPFTableCell cell : firstRow.getTableCells()){
        cell.setVerticalAlignment(this.headerFormat.vAlign);

        // Get the cell paragraph and style it
        par = getCellParagraph(cell, 0);
        par.setAlignment(this.headerFormat.alignment);
        par.setIndentationLeft(this.headerFormat.spaceLeft);
        par.setIndentationRight(this.headerFormat.spaceRight);
        par.setSpacingBefore(this.headerFormat.spaceBefore);
        par.setSpacingAfter(this.headerFormat.spaceAfter);

        // Get the paragraph run and style it
        run = par.getRuns().get(0);
        run.setFontFamily(this.headerFormat.fontFamily);
        run.setFontSize(this.headerFormat.fontSize);
        run.setBold(this.headerFormat.isBold);
        run.setItalic(this.headerFormat.isItalic);
        run.setUnderline(this.headerFormat.underline);
    }
}

public void addNewRow(List<String> values){
    XWPFTableRow newRow = this.table.createRow();
```

Figura 16. Extracto de código del manejo de tablas

Cada uno de estos elementos cuenta con su correspondiente clase de testeo en el módulo de pruebas. Este módulo se comentará con más detalle en la sección de resultados y pruebas de esta memoria, donde podremos ver de manera más amplia los resultados de las pruebas de cada funcionalidad, así como del resultado total del proyecto.

4.2.2. FORMATOS DE ELEMENTOS DE WORD

El módulo de formatos, si bien es un módulo más simple y abarca una tarea más pequeña que los demás módulos de funcionalidades (excluimos el de pruebas y constantes), comprende una tarea muy importante sin la cual el documento final no sería más que texto plano.

En este módulo se ha llevado a cabo la tarea de relacionar cada uno de los elementos de Word encapsulado con un objeto de formato. El objetivo de un objeto de formato es el de dictar todas las reglas en cuanto a la colocación, estilo del texto (fuente, tamaño, negrita, etc.), espaciado, márgenes, bordes y demás variables, para permitir que su introducción en un documento sea todo lo que tengamos que realizar.

Con este planteamiento queremos conseguir que los formatos de, por ejemplo los encabezados (o títulos) y las tablas sea el mismo para el informe final, y así, una vez insertada una tabla en el informe, esta quedaría formateada de la manera que deseemos.

Definiendo formatos constantes para cada elemento de Word que maneja la aplicación, evitamos el tener que especificar de manera explícita, cuando creamos un objeto Word, todas las variables que definen su formato dentro del documento. Esto no solo evita repetir código y simplificar el uso de nuestro módulo, sino que en un futuro permite cambiar la apariencia final de un documento modificando solo los ficheros de formato. Una manera de ver esta tarea, y que es una de las principales inspiraciones para haberla llevado a cabo, es pensar en una web, donde se separa el contenido (HTML, PHP, JavaScript, etc.) del estilo y apariencia final (CSS).

En la siguiente lista se exponen los distintos formatos que se han incluido en este módulo:

- **Cuerpo:** Refiere al formato que se le aplica al texto plano del documento de salida, que se encontrará en forma de párrafos.
- **Título:** El formato construido para los encabezados de las secciones de un documento. Entre otras cosas, se necesita que sea más grande, en negrita y subrayado.
- **Imagen:** Define parámetros como la anchura por defecto, altura por defecto, bordes del marco, posición en el documento y otros para permitir la colocación de cualquier imagen en el documento.
- **Cabecera de Tabla:** Define el formato para la primera fila de una tabla o la fila de cabecera, que contiene los títulos de las columnas y que no es parte de las filas de datos.
- **Fila de Tabla:** El formato aplicable a cualquier fila de datos de una tabla. Contiene parámetros como distancia a los bordes de la celda (vertical y horizontal), formato de los bordes y formato del texto interno.
- **Anotación de Imagen:** Refiere al formato de una anotación al pie de una imagen, el cual suele utilizar texto más pequeño y en negrita.
- **Anotación de Tabla:** Este formato es un concepto exactamente igual de anotación de imagen, pero con pequeñas variaciones para hacerlo apto para una tabla. Como decíamos, las tablas son uno de los componentes más complejos de Apache POI, tanto que requieren de una logística y formato especiales para poder incorporar una anotación.

En la Figura 16 podemos ver los valores del formato para una anotación al pie de una imagen.

```
public class ImageFootnoteFormat extends Format{
    public ImageFootnoteFormat(){
        this.fontFamily = "CALIBRI";
        this.fontSize = 10;
        this.isBold = true;
        this.isItalic = false;
        this.underline = UnderlinePatterns.NONE;
        this.spaceBefore = 0;
        this.spaceAfter = 50;
        this.spaceLeft = this.spaceRight = 0;
        this.alignment = ParagraphAlignment.CENTER;
    }
}
```

Figura 17. Valores de formato para una anotación de imagen

4.2.3. ANÁLISIS Y PROCESADO DE LOS DATOS - LECTURA

Una vez que tenemos asentadas unas bases sólidas para poder enlazar con un documento Word, y tenemos ya todas las funcionalidades que necesitamos para generar nuestro modelo de informe funcionando, el siguiente paso consiste en manejar la información de entrada.

El módulo de análisis del proyecto está dedicado a esta tarea en exclusiva, al procesado de la información en bruto que habíamos comentado anteriormente en la sección del ProcesaTrazas. Como decíamos entonces, la piedra angular de este análisis es poder almacenar los datos en un mapa hash, donde guardaremos los pares clave-valor mencionados.

La primera tarea consiste en leer los datos de entrada, para lo cual se ha creado una clase Reader, que, además de realizar el trabajo de lectura, "parsea" los datos y los mete en estructuras propias (clase Record, dedicada a almacenar una línea o registro completo del fichero de entrada). Como vemos, esta clase realiza una tarea fundamental, ya que es el punto donde enlazamos las dos mitades del trabajo que comentábamos con anterioridad: la captura de tráfico y volcado de esos datos en un fichero, por un lado, y la lectura y el procesado de este fichero para generar un informe, por otro.

Como decíamos antes, la carga de los datos es una tarea bastante optimizada (de hecho es algo muy importante que lo sea, pues estamos hablando de ficheros que pueden tener millones de líneas), ya que basta con leer, de cada línea, cada elemento separado por espacios y formar nuevos pares clave-valor en nuestro mapa hash. Posteriormente, extraer los datos que nos interesan de este mapa, para crear nuestra propia estructura Record, es algo que se realiza de manera inmediata gracias a la unicidad de cada clave.

En la Figura 18 se puede ver una porción de la clase Reader, encargada de la tarea de leer y formatear los datos. Podemos apreciar en la imagen la creación de los registros clave-valor del mapa, así como la búsqueda en dicho mapa de las direcciones MAC.

```
public List<Record> ReadAndParse() throws IOException {
    List<Record> records = new ArrayList<Record>();
    FileReader fileReader = new FileReader(this.inputFile);
    BufferedReader bufferedReader = new BufferedReader(fileReader);
    String line;

    // Read and parse each line into a record
    while((line = bufferedReader.readLine()) != null){
        records.add(this.ParseRecord(line));
    }

    // Close the reader
    bufferedReader.close();

    return records;
}

private Record ParseRecord(String rawData) {
    HashMap<String, String> mappedData = new HashMap<String, String>();
    List<String> data;
    long bytesSent = 0;
    long bytesReceived = 0;
    long packetsSent = 0;
    long packetsReceived = 0;

    // Split every data with blank space as delimiter
    data = new ArrayList<String>(Arrays.asList(rawData.split(" ")));

    // Get the source and destination IP addresses (always in position 0 and 2)
    String srcIP = data.get(Definitions.SourceIPPosition);
    String dstIP = data.get(Definitions.DestIPPosition);

    // Map each pair of data as key-value pairs
    for (int i = 0; i < data.size(); i += 2) {
        mappedData.put(data.get(i), data.get(i + 1));
    }

    // Get the source and destination MAC addresses
    String sourceMAC = mappedData.get(Definitions.SourceMACTag);
    String destMAC = mappedData.get(Definitions.DestMACTag);
```

Figura 18. Extracto de la clase Reader

Una vez que tenemos todas las líneas (recordemos que cada línea del fichero de entrada representaba información sobre una conexión o conversación) procesadas y almacenadas en nuestro conjunto de estructuras, es hora de procesar estos datos para sacar las conclusiones que se plasmarán en el informe.

4.2.4. ANÁLISIS Y PROCESADO DE LOS DATOS - PROCESAMIENTO

Con nuestro conjunto de datos almacenando la información por conversación proveniente del ProcesaTrazas, ya podemos centrarnos en realizar cualquier tipo de operación y manejo de estos datos para obtener la información concreta que se plasmará en nuestro informe. Recordemos que el modelo de informe que se utiliza ya se ha detallado con anterioridad, por lo tanto nuestro objetivo final será obtener los datos que se piden en cada uno de los puntos de ese modelo.

A tal fin, codificaremos todos los algoritmos necesarios dentro de la clase que se encarga de esta tarea, la clase Statistics. Esta clase está dedicada en exclusiva a contener el conjunto de algoritmos cuyos valores de retorno satisfacen las exigencias del modelo actual de informe. En todo caso, siempre mencionamos modelo "actual" porque el modelo puede ampliarse en cualquier momento que se desee. Esta ampliación solamente conllevaría añadir un nuevo algoritmo en la clase Statistics, ya que el proyecto ha sido diseñado de tal manera que sea fácilmente escalable, para que las bases estén bien sentadas para poder ampliarlo de manera sencilla siguiendo el procedimiento que se explicará en el apartado **6.2 Trabajo Futuro**.

Actualmente, la clase Statistics soporta 8 métodos, cada uno de los cuales obtiene la información que se pide en cada uno de los 8 apartados del modelo actual. Como habrá notado el lector a su paso por el apartado que detalla el modelo de informe, existe un paralelismo claro entre los datos que se estudian a Nivel 2 (nivel MAC) y a Nivel 3 (nivel IP). Por tanto, no es necesario repetir los algoritmos, sino que basta con indicarles a estos métodos, mediante un parámetro, si deben utilizar los datos MAC o IP para calcular el resultado final.

Para cada uno de los cálculos, existe un esquema similar que se sigue a la hora de procesar los datos, y que consiste en analizar el conjunto entero de registros, y para cada uno de ellos extraer los datos que se piden (paquetes enviados, bytes recibidos, etc.), realizar los cálculos pertinentes y guardar el resultado en un mapa hash.

Como vemos, vuelve a ser particularmente interesante el hecho de utilizar una estructura de datos basada en claves hash ya que el volumen de registros que analizamos puede llegar a ser considerablemente grande, y cada uno de estos registros puede distinguirse por un dato único que variará según el algoritmo en el que nos encontremos, pudiendo ser éste la dirección IP de origen, o la dirección MAC de destino, etc.

Existe un caso particular en el que este dato único distintivo que mencionamos tiene que mirarse de manera especial, y es el caso en el que analizamos datos o estadísticas a nivel de conversaciones. Puesto que una conversación se refleja en el fichero de salida de ProcesaTrazas como una conexión cliente-servidor, tendremos que tener en cuenta que una conversación entre los extremos A y B, actuando A como cliente y B como servidor, debe ser computada de la misma manera que una conversación en la que B actúa como cliente y A como servidor. Es decir, no nos importa quién actúe como cliente y quién como servidor (distinción que sí se refleja en el fichero de salida del ProcesaTrazas), lo que nos interesa es que existe una comunicación entre A y B, sea en el sentido que sea.

Con este fin, se ha creado una clase propia Conversation, en donde se pueden almacenar los datos, relativos a una conversación, que exige el modelo de informe y que además cumple la mencionada relación de equivalencia (implementando los métodos de comparación y de devolución de código hash). Para el resto de información del informe, podemos utilizar la clase más sencilla Address, que solo necesita guardar una dirección y los datos solicitados por el modelo de informe.

El último punto a tener en cuenta al codificar los algoritmos que obtienen los datos finales, es el de detectar casos particulares en los que el cálculo de la información debe realizarse de una manera distinta por diversos motivos. La siguiente lista presenta dichos motivos y la solución que se ha tomado para cada caso.

- **Direcciones nulas:** Existe el caso de que algunas de las direcciones registradas por el ProcesaTrazas sea, por algún motivo, nula (ya sea porque la información original estaba corrupta en cierta medida o porque el programa no ha sido capaz de extraer información útil de algunos registros). Si bien en el caso de cálculo de información para direcciones no nos afecta (p.ej.: en ningún caso se tomarían en cuenta los paquetes enviados por una dirección nula), en el caso de las conversaciones hay que asegurarse de no tomar en cuenta información de una conversación en el que uno de los extremos no ha quedado registrado debidamente. Por tanto, cualquier tipo de conversación con uno de los extremos no identificados, es desechada.
- **Conversaciones con uno mismo:** Cuando en el fichero analizado existen conversaciones de un ordenador consigo mismo, hay que tener en cuenta que cierta información aparecerá duplicada. Por ejemplo, si un ordenador A se enviase a sí mismo 20 bytes, en el caso en el que estuviéramos registrando el volumen de tráfico generado por esa conversación, deberíamos tener en cuenta que los 20 bytes que figuran en la dirección cliente->servidor son los mismos 20 que figurarán en la dirección servidor->cliente. Es decir, necesitamos registrar la mitad de información en algunos datos cuando el cliente y el servidor son el mismo en una conversación.

En la Figura 19 y Figura 20 se presentan fragmentos de código de algunos de los algoritmos implementados para el modelo de informe de pruebas.

```
    } else {
        // Find top 10 repeated IP conversations
        for (Record r : this.records) {
            Conversation conver = new Conversation(r.getSourceIP(), r.getDestIP());
            if (repetitions.containsKey(conver)) {
                // Add 1 to the number of repetitions of this conversation
                long count = repetitions.get(conver);
                repetitions.put(conver, count + 1);
            } else {
                // Create a new key entry for this particular conversation
                repetitions.put(conver, new Long(1));
            }
        }
    }

    // Add every conversation to the list
    conversations.addAll(repetitions.keySet());

    // Sort the list by its value in the repetitions map
    Collections.sort(conversations, new Comparator<Conversation>() {
        @Override
        public int compare(Conversation conver1, Conversation conver2) {
            Long count1 = repetitions.get(conver1);
            Long count2 = repetitions.get(conver2);

            // Compare in reverse order to get descending list results
            return count2.compareTo(count1);
        }
    });

    // Get the top 10 sublist if there were more than 10 different conversations
    if (conversations.size() > 10) {
        conversations = conversations.subList(0, 10);
    }

    // Save the number of appearances of each conversation
    for (Conversation conver : conversations) {
        conver.setCount(repetitions.get(conver));
    }

    return conversations;
}
```

Figura 19. Extracto del cálculo del Top 10 Conversaciones IP

```
// Add every address to the list
addresses.addAll(sentBytes.keySet());

// Sort the list by its value in the sent bytes map
Collections.sort(addresses, new Comparator<Address>() {
    @Override
    public int compare(Address address1, Address address2) {
        Long count1 = sentBytes.get(address1);
        Long count2 = sentBytes.get(address2);

        // Compare in reverse order to get descending list results
        return count2.compareTo(count1);
    }
});

// Save the number of sent bytes of each address
for (Address address : addresses) {
    address.setSentBytes(sentBytes.get(address));
}

// Get results until 99% of traffic is reached (or 20 addresses are retrieved)
long traffic = 0;
int top = 0;
for(Address address : addresses){
    // Set traffic % of this address
    double trafficVolume = (double)address.getSentBytes() / (double)totalBytesOfCapture;
    address.setTrafficVolume(trafficVolume * 100);

    topAddresses.add(address);
    traffic += address.getSentBytes();

    // If the 99% of traffic has been reached, return the addresses that are already in the top
    if(((double) traffic/totalBytesOfCapture) >= 0.99){
        return topAddresses;
    }

    top++;

    // If 20 addresses are already retrieved, return the top 20
    if(top >= 20){
        return topAddresses;
    }
```

Figura 20. Extracto del cálculo del Top MAC Origen en Tráfico

Teniendo implementado los algoritmos y con una base sólida para plasmar los resultados en un documento Word, queda desarrollar un módulo que enlace estos componentes: extraer información utilizando el módulo de análisis y volcar estos resultados en un documento utilizando las herramientas encapsuladas en el módulo de Word.

4.2.5. INFORMES Y VOLCADO DE DATOS

Para representar el modelo de informe en el código, se ha implementado un módulo por separado que indica qué es un informe y el contenido que tiene que haber en él.

Para esta tarea, definimos una clase Section que definirá cada una de las partes de las que se compone un informe. Todas las secciones contendrán un título, un párrafo descriptivo y a continuación un conjunto libre de elementos de Word. Como vemos, lo que se busca con esto es normalizar el contenido de los informes para que cada uno de los datos finales a plasmar siga un patrón común. El título y la descripción son, justamente, elementos descriptivos del contenido de cada sección, mientras que el conjunto de elementos de Word permite introducir en una sección tantas tablas, gráficas, texto o imágenes como queramos, por lo que ofrece flexibilidad a la hora de ampliar el modelo de informe. Si en un futuro cada sección de nuestro modelo contase con una gráfica extra, la sección del informe correspondiente podría admitirlo libremente, así como más texto descriptivo para dicha gráfica si fuera necesario.

Al haber definido las piezas que componen nuestro informe, definir la clase Report es tan fácil como permitir que exista un conjunto de secciones. De este modo, definimos los informes como el conjunto ordenado de distintas secciones, cada una de las cuales presenta sus propios datos.

De manera adicional, se pueden añadir funcionalidades extra, como es el añadir saltos de página de manera sencilla y mediante un solo método en la clase Report. Esto permitiría que cada sección estuviera ubicada en su propia página. En nuestro caso nos interesa particularmente, ya que es deseable evitar, en la mayor medida posible, que las tablas se queden cortadas y divididas entre dos páginas distintas.

En las figuras de la página siguiente pueden observarse extractos de código de las clases Section y Report.

```
public class Section {
    private Report report;
    private WordHeading heading;
    private List<WordElement> content;

    public Section(Report rep, String t){
        this.report = rep;
        this.content = new ArrayList<WordElement>();
        this.heading = new WordHeading(this, t);
    }

    public void addContent(WordElement elem){
        this.content.add(elem);
    }
```

Figura 21. Extracto de código de la clase Section

```
public Report(String n, String location) throws Exception{
    this.document = new XWPFDocument();

    this.name = n;
    this.locationPath = location+this.name+".docx";

    this.file = new FileOutputStream(new File(this.locationPath));
}

public void addSection(Section s){
    this.sections.add(s);
}

public void save() throws Exception{
    this.document.write(this.file);
    this.file.close();
    this.document.close();

    System.out.println(this.name+".docx written succesfully");
}

public void addPageBreak(){
    XWPFParagraph para = this.document.getLastParagraph();
    para.createRun().addBreak(BreakType.PAGE);
}

public String getName() {
    return this.name;
}
```

Figura 22. Extracto de código de la clase Report

Una vez que tenemos definida la estructura de un informe y tenemos las herramientas necesarias para leer y analizar la información de entrada, solo queda definir un camino de ejecución que se

encargue sistemáticamente de leer, analizar y agregar una nueva sección a nuestro informe por cada uno de los datos que se nos exige en el modelo.

En la Figura 23 podemos ver un fragmento de código de la clase Main, donde se puede observar la lectura y procesado de datos, y la posterior creación de una sección nueva utilizando uno de los algoritmos definidos previamente para analizar la información. Como vemos, el resultado del análisis de la información se plasma a modo de tabla en una sección que es agregada al conjunto de secciones de nuestro informe. También se aprecia el uso de la funcionalidad de salto de página que comentábamos anteriormente.

```
public class Main {
    public static void main(String[] args) throws Exception {
        // Start measuring time
        long start = System.nanoTime();
        Reader rd = new Reader(Path.InputSamples + "inputfile");
        List<Record> records = rd.ReadAndParse();
        Statistics st = new Statistics(records);
        Report rep = new Report("report");
        List<Conversation> convers;
        List<Address> addresses;
        List<String> columnNames;

        {
            // Create new section (top 10 MAC conversations)
            Section section = new Section(rep, "Top 10 MAC Conversations");
            new WordParagraph(section, "This first table shows the top MAC c
                    + " MAC conversations. The capture file (.pcap) analyzec
                    + " ones appeared more often. No distinction between sou
                    + " A->B belong to the same conversation as messages in
            WordTable table = new WordTable(section, "Table 1: Shows MAC adc

            columnNames = new ArrayList<String>();
            columnNames.add("MAC Address 1");
            columnNames.add("MAC Address 2");
            columnNames.add("N° of Conversations");
            table.addColumnNames(columnNames);

            convers = st.Top10Conversations("MAC");

            // Read each conversation and add it to the table
            for(Conversation conver : convers) {
                List<String> row = new ArrayList<String>();
                row.add(conver.getAddress1());
                row.add(conver.getAddress2());
                row.add(String.valueOf(conver.getCount()));
                table.addNewRow(row);
            }
        }

        // Put each section in its own page
        rep.addPageBreak();
```

Figura 23. Lectura de datos y creación de una sección

En esta línea, todas las secciones son creadas de manera similar, siguiendo el procedimiento presentado a continuación:

1. Llamada al algoritmo correspondiente sobre los datos de entrada
2. Utilizar la salida del algoritmo (resultado del análisis) para crear un elemento Word con el que representar esos datos (tablas, en nuestro caso)
3. Añadir el elemento Word a una nueva sección, que a su vez es insertada en el documento
4. Añadir un salto de página

Como paso final, solo queda grabar el documento con el método correspondiente. Como vemos, el proceso de creación del documento sigue un algoritmo bastante definido una vez que tenemos el diseño adecuado.

En el caso de querer añadir nuevas secciones a nuestro modelo inicial de informe, bastaría sencillamente con añadir el algoritmo correspondiente en la clase de Statistics (o realizar una versión modificada de alguno de los existentes) y luego realizar un procedimiento como el descrito en la lista anterior para añadir una nueva sección, con la salida de dicho nuevo algoritmo, a nuestro informe automático.

5. PRUEBAS Y RESULTADOS

En esta sección presentaremos los resultados de las diversas pruebas llevadas a cabo para cada una de las mitades del proyecto, tanto con la herramienta ProcesaTrazas como con el Generador Automático.

5.1. PROCESATRAZAS

Este apartado de la sección está destinado a explicar las pruebas que se realizaron con la herramienta de captura y procesado de tráfico empleada en el trabajo.

Como recordamos de secciones anteriores, el módulo del ProcesaTrazas utilizado es el correspondiente al procesado de conexiones. Las pruebas sobre este módulo son mayoritariamente en cuanto a tiempo, tamaño de los archivos de entrada y extensión y tamaño de los ficheros de texto de salida.

Nos interesa ser capaces de procesar documentos grandes, por lo que queremos obtener una aproximación de cuánto tiempo necesita la herramienta para analizar un archivo de entrada en función del tamaño del archivo en cuestión.

Para esta tarea, recordemos que contamos con diversas opciones de ejecución para el ProcesaTrazas, una de las cuales presenta información útil para nuestro fin al acabar la ejecución del programa.

Por último, recordar que si bien la motivación inicial del trabajo era analizar tráfico de redes con arquitectura SNA, por facilidad y por el carácter más genérico que adquirió el trabajo durante su desarrollo, se utilizan archivos de captura ya generados específicamente para pruebas (obtenidos de internet) que contienen tráfico más "normal" basado en TCP/IP.

5.1.1. TAMAÑO DE LA ENTRADA Y TIEMPO DE EJECUCIÓN

A continuación, pasaremos a exponer algunos de los casos que bajo los cuales se ha probado la herramienta, detallando el tamaño de la entrada para cada caso, así como el tiempo de ejecución consumido y el tamaño de la salida. Al ejecutar el módulo ProcesaConexiones se han utilizado las opciones -s y -g para ser informados al final de la ejecución del tiempo consumido, entre otros datos, así como la opción -v. Propondremos también una manera de ejecución alternativa utilizando alguno de los parámetros del ProcesaTrazas.

- Para el primer caso de pruebas se ha utilizado un fichero de entrada de 7KB, el cual ha tardado un total de 89ms en ser procesado. El tamaño total del fichero de salida es de 8KB.

- La misma prueba se ha realizado para un archivo de entrada de tamaño 359MB. El tiempo de ejecución del ProcesaTrazas en este caso ha sido de 435ms y el tamaño del fichero de salida es de 83MB.

Podemos concluir, a partir de las pruebas realizadas, que en líneas generales el tiempo de ejecución no aumenta de manera exponencial con respecto al tamaño del archivo de entrada y que el fichero de salida también tiene un tamaño razonable (incluso con la opción -v).

El mayor factor a tener en cuenta a la hora de procesar archivos de entrada .pcap en esta primera parte del trabajo no es tanto el tiempo de ejecución al procesar un archivo, sino el tamaño de la salida, ya que un archivo de salida excesivamente grande podría complicar el correcto funcionamiento del Generador Automático. Por este motivo, nos interesa saltar al siguiente apartado de la sección de pruebas.

5.1.2. TAMAÑO DE LA SALIDA

Debemos tener en cuenta que el uso de la opción -v, la cual añadía al fichero de salida la "leyenda" o claves que comentábamos en su momento, no solo acarrea un mayor tiempo de ejecución ya que implica escribir mucha más información en el archivo de salida, sino que aún más importante es el hecho de que al escribir una etiqueta por cada dato, duplica el número de "palabras" que escribe por línea, lo cual tiene un impacto enorme en el tamaño del fichero de salida. Por este motivo, en esta sección analizamos los dos mismos casos que en la sección previa pero sin el uso de la opción -v.

- Para el primer caso, se ha ejecutado el mismo primer fichero que en las pruebas anteriores pero sin la opción -v. El tamaño del archivo de entrada, que era de 7KB, había generado un fichero de salida de 8KB, en comparación con los 4KB de este caso.
- El segundo caso, el del fichero de entrada de 359MB, ha disminuido considerablemente el tamaño de la salida al ser ejecutado sin la opción -v: en concreto ha pasado de 83MB a 47MB.

Es una opción a tener en cuenta en el futuro el buscar una alternativa viable para dejar de utilizar esta opción, consiguiendo así evitar "desperdiciar" tamaño en los ficheros de salida. Podría incluso darse el caso en el que no utilizar esta opción permitiese procesar ficheros de entrada mayores, ya que si por algún motivo existiese un límite de tamaño para el fichero de salida generado, usar o no esta opción podría marcar la diferencia entre rebasar ese límite o no.

No extenderemos esta sección con más resultados, puesto que el objetivo final del trabajo (si bien es necesario tener en mente el rendimiento del ProcesaTrazas) se centra en el Generador Automático.

5.2. GENERADOR AUTOMÁTICO

Para la sección de pruebas del Generador Automático, si bien nos interesa principalmente probar su correcta funcionalidad, también es necesario comprobar que la velocidad a la hora de generar informes sea razonable. Aun siendo este el caso, también es necesario conocer los tiempos de ejecución de la herramienta, por lo tanto se han añadido marcas temporales al comienzo y final de la ejecución en la clase Main del módulo de informes.

Los ficheros de entrada que se han utilizado para las pruebas corresponden a los ficheros de salida generados por el ProcesaTrazas en las pruebas anteriores. Concretamente, a los generados con la opción -v, ya que como comentábamos el Generador Automático aún requiere de las claves en los ficheros para encontrar los valores.

Dado que los resultados de las pruebas de esta herramienta producen documentos Word de una extensión considerable, presentaremos a continuación los tiempos de ejecución para ambos ficheros de prueba, pero sólo adjuntaremos los resultados para el segundo caso, el del fichero de mayor tamaño, ya que debido a su mayor contenido genera un informe con unos datos más reales e interesantes.

- Fichero 1 (7KB): informe generado en 112ms.
- Fichero 2 (359MB): informe generado en 1213ms.

Los resultados de las pruebas son tal y como se esperaban, pues los informes tienen el formato deseado (una sección por página, con tablas centradas, títulos destacados, etc.) y no presentan ninguna irregularidad. En cuanto a la corrección de los datos, se ha comprobado con ficheros más pequeños que los algoritmos funcionan como era de esperar: en concreto se ha probado el correcto funcionamiento de los algoritmos con pequeños ficheros de 2 o 3 líneas, para que sea posible comparar los datos de salida con los resultados calculados a mano.

Aprovechando la naturaleza de los documentos de salida del Generador Automático, adjuntaremos el contenido íntegro del informe generado en las pruebas en lugar de presentar contenido del mismo en forma de imágenes. El documento Word generado al procesar el fichero número 2 puede encontrarse en el **Anexo A**. Como podemos ver en el anexo, se ha decidido generar el informe de prueba en inglés, siguiendo las mismas pautas que para la codificación del trabajo y para ofrecer a la herramienta una mayor flexibilidad de cara al lector final del informe.

En la Tabla 3 podemos encontrar un resumen de las pruebas que se han realizado al “ciclo completo” de los datos (desde la red o un fichero .pcap hasta ser plasmados en un informe), incluyendo algunos casos que no figuran de manera escrita y con opciones de ejecución -s, -g y -v para el Procesa Trazas. Recordemos que el tamaño de la salida del ProcesaTrazas (PT) es el tamaño de la entrada del Generador Automático, por lo que se obvia esta repetición.

Tamaño Entrada (PT)	Tamaño Salida (PT)	Tiempo de Ejecución (PT)	Tiempo de Ejecución (GA)
7KB	8KB	89ms	112ms
42MB	11MB	121ms	289ms
359MB	83MB	435ms	1213ms
863MB	313MB	7567ms	1745ms
3’5GB	874MB	46s	3049ms

Tabla 3. Resumen de las pruebas realizadas

En la tabla se desprecian los tamaños de salida del Generador Automático, puesto que es prácticamente invariable. En cuanto a los tiempos de ejecución, podemos concluir que son bastante buenos teniendo en cuenta la cantidad de datos que se está procesando en cada ejecución.

5.3. FUNCIONALIDADES WORD

Más allá de las "pruebas reales" que se han comentado en las secciones anteriores (en las que se parte de un fichero .pcap y se acaba con el resultado final), también es interesante presentar en esta memoria el resultado de las pruebas individuales que se ha hecho a cada una de las funcionalidades que se ha incluido en el wrapping de la API.

Por este motivo, al igual que hacíamos antes con el informe generado, incluiremos la salida generada a modo de distintos anexos, aprovechando su naturaleza de documentos Word. En estos anexos se pretende enseñar el resultado de las pruebas para:

- Creación de una tabla
- Creación de una imagen
- Creación de una función

Si bien se han realizado otras pruebas de apoyo (pruebas para crear un documento vacío, un documento con una única sección, un documento con secciones diferentes, etc.), resultan menos relevantes, ya que prueban funcionalidades que se utilizan posteriormente en otros tests. Por este motivo, basta con exponer las pruebas que abarquen la mayor diversidad de elementos utilizados posible.

Los mencionados anexos pueden encontrarse tras el Anexo A, en el que se presenta un ejemplo de informe generado.

6. CONCLUSIONES Y TRABAJO FUTURO

Esta sección está dedicada a explicar los pensamientos del autor tras la realización del trabajo, así como a proponer ideas para una posible continuación del trabajo desarrollado. Se expondrán en un apartado el estado del trabajo al concluir su desarrollo y reflexiones sobre los objetivos conseguidos, y en otro apartado las ideas sobre la posible línea de desarrollo a seguir.

6.1. CONCLUSIONES

Si bien el trabajo comenzó como un proyecto destinado a redes de la arquitectura SNA, a lo largo de su desarrollo quedó claro que su objetivo final iba a ser algo más amplio y genérico que el enfoque inicial. Aunque en principio esto es algo desconcertante, resultó ser algo completamente positivo, pues el trabajo final no solo es capaz de cumplir con los objetivos iniciales sino que se ha convertido en una herramienta que va más allá de la idea que lo impulsó.

El análisis de las redes es una tarea de gran importancia hoy en día, y haber podido automatizar con éxito tareas "de campo" como el procesado y estudio del tráfico que circula por la red, permite agilizar las tareas realmente importantes como el diagnóstico de problemas y la prevención de amenazas o posibles fallos. Si bien el trabajo desarrollado no está, ni mucho menos, enfocado a su lanzamiento en un entorno real o el mundo comercial, el tema que toca es realmente interesante y cualquier aporte constructivo que pueda realizarse a esta área (ya sea en forma de ideas o con herramientas que sirvan de precedentes para aplicaciones comerciales desarrolladas en un futuro) es sin duda bienvenido.

Para mencionar un aspecto negativo, señalaremos que el modelo de informe utilizado para las pruebas podría haber sido más exhaustivo o diverso. Sin embargo, el objetivo final del trabajo no es tanto ofrecer un modelo de informe lo más extenso posible como sí lo es el hecho de convertir en posible la idea de generar de manera automática *cualquier* informe. Resulta más interesante el hecho de poder ampliar o modificar el modelo de informe en el futuro de manera rápida y sencilla, algo que de hecho es posible, y que puede realizarse tal y como se explicó en la sección de pruebas, gracias a que el trabajo se desarrolló de manera que sea fácilmente escalable.

El resultado final coincide con lo esperado, por lo que puede concluirse que el trabajo es satisfactorio en cuanto a los objetivos planteados. A pesar de esto, como cualquier otro proyecto, aún tiene un margen muy amplio de mejora, motivo por el cual existe la siguiente sección de esta memoria, donde podemos esbozar algunas de las ideas dentro de ese margen.

6.2. /TRABAJO FUTURO

A continuación vamos a proponer una liste de ideas que pueden implementarse en un futuro para seguir completando y mejorando el trabajo desarrollado.

- Como comentábamos en la sección de pruebas, el uso de la opción -v al ejecutar el ProcesaTrazas penaliza el tamaño de fichero que este genera. Una manera de evitar su uso es, al leer los datos que genera el ProcesaTrazas, crear un mapa hash donde las claves, en lugar de ser palabras presentes en el fichero, sean un número que corresponde a la posición de cada dato dentro de su fila en el archivo. Esta posición podría conocerse de antemano gracias a la documentación del ProcesaTrazas (al igual que pasara con las claves).
- Tal y como está implementada la creación de secciones, todo se realiza de manera conjunta y repetitiva. Una mejora deseable sería implementar una función que crease una sección en base a un conjunto de datos y un fichero de entrada, que podría contener tanto el título de la sección como la descripción de la misma y metadatos sobre los elementos Word (tablas, gráficas, etc.) que debe contener.
- Sería deseable añadir una capa a la implementación que abstraiga el concepto de modelo de informe. Esto permitiría definir lo que es un *modelo* de informe y los contenidos que ha de tener. De esta manera, podríamos tener distintos modelos según el tipo de red que se está analizando o según el propósito de la ejecución (pruebas o uso real de la aplicación).
- Una buena idea a tener en cuenta en el futuro sería eliminar la dependencia de GNUPlot para la funcionalidad de gráficas en los documentos, y en cambio utilizar el propio módulo de la API para Word. Esto permitiría que el programa fuera plenamente funcional independientemente del sistema operativo en el que se ejecute, y además permitiría insertar gráficas en los documentos que podrían modificarse manualmente, ya que utilizando GNUPlot no existe esta posibilidad una vez que las gráficas están generadas e insertadas en el documento (recordemos que las gráficas son tratadas como imágenes actualmente).
- Como último detalle, se sugiere el añadir una portada e índice sencillos al informe generado, que si bien son tareas que pueden realizarse a mano ya que se alejan del objetivo del informe, ofrecen un aspecto más cuidado o estilizado al informe final.

7. REFERENCIAS

[1] IBM. Networking on z/OS, 2006, pages 99—102
https://www.ibm.com/support/knowledgecenter/api/content/nl/es/zosbasics/com.ibm.zos.znetwork/znetwork_book.pdf

[2] Paessler AG. PRTG Network Monitor, 2013.
https://www.es.paessler.com/prtg/product-information

[3] Solar Winds. Network Bandwith Analyzer, 2012.
http://www.solarwinds.com/es/lp/network-bandwidth-analyzer-pack.aspx

[4] Oracle. Oracle Technology Network: Java, 2008.
http://www.oracle.com/technetwork/java/

[5] Sun Microsystems. OpenOffice.org 3.1 Developer's Guide, April 2009, pages 47—52.
https://wiki.openoffice.org/w/images/d/d9/DevelopersGuide_OOo3.1.0.pdf

[6] Plutext. Docx4j – Getting Started, 2008, pages 3—9.
https://github.com/plutext/docx4j/blob/master/docs/Docx4j_GettingStarted.pdf?raw=true

[7] Aspose. About WordprocessingML, 2015.
http://www.aspose.com/docs/display/wordsnet/WordprocessingML+(DOCX,+XML)

[8] Oracle. Java Projects: Project JAXB, 2013.
https://jaxb.java.net/2.2.11/docs/ch01.html#documentation

[9] The Apache Software Foundation. Apache POI – The Java API for Microsoft Documents, September 2015. https://poi.apache.org/

[10] The Apache Software Foundation. Apache License Version 2.0, January 2004.
http://www.apache.org/licenses/LICENSE-2.0

[11] Correa, Leonardo. Java2Word, February 2012.
https://github.com/leonardoanalista/java2word

[12] Williams, Thomas and Kelley, Colin. GNUPlot – An Interactive Plotting Program, 2014, pages 16 – 17. http://www.gnuplot.info/docs_4.6/gnuplot.pdf

[13] The Eclipse Foundation. Eclipse, June 2015.
https://eclipse.org/

[14] The Wireshark Foundation. Wireshark Developer's Guide, April 2016, pages 1—3.
https://www.wireshark.org/download/docs/developer-guide-us.pdf

[15] Jacobson, Van, Leres, Craig and McCanne, Steven. Pcap – Packet Capture Library, December 2015. http://www.tcpdump.org/manpages/pcap.3pcap.html

8. ANEXOS

A. INFORME AUTOMÁTICO CON EL MODELO DE PRUEBAS

Top 10 MAC Conversations

This first table shows the top MAC conversations (limited to 10). This means it shows the top most repeated unique MAC conversations. The capture file (.pcap) analyzed showed that the conversations displayed in the following table were the ones appeared more often. No distinction between source and destination addresses were made, meaning messages in the direction A->B belong to the same conversation as messages in the direction B->A.

MAC Address 1	***MAC Address 2***	***Nº of Conversations***
00:21:70:63:3f:ff	00:90:7f:3e:02:d0	1027
00:21:70:67:6f:50	00:90:7f:3e:02:d0	839
00:90:7f:3e:02:d0	00:21:70:64:74:34	734
00:21:70:67:15:9f	00:90:7f:3e:02:d0	651
00:21:70:63:3a:e9	00:90:7f:3e:02:d0	630
d4:be:d9:4b:24:79	00:90:7f:3e:02:d0	590
00:21:70:64:71:60	00:90:7f:3e:02:d0	558
00:21:70:67:6d:de	00:90:7f:3e:02:d0	522
00:90:7f:3e:02:d0	00:50:43:01:4d:d4	512
00:21:70:67:32:60	00:90:7f:3e:02:d0	512

Table 1: Shows MAC addresses and repetitions

Top 10 IP Conversations

The idea of this section is pretty much the same as the previous one, but with IP addresses instead of MAC ones. We're jumping onto Level 3 data analysis of the capture file. We'll be repeating this kind of approach in the rest of the report, so IP sections' descriptions will be shorter, since they will be pretty much the same as MAC ones.

IP Address 1	*IP Address 2*	*Nº of Conversations*
172.16.133.116	172.16.139.250	898
172.16.133.42	172.16.139.250	361
172.16.133.67	172.16.139.250	325
172.16.133.12	172.16.139.250	311
172.16.133.25	172.16.139.250	308
172.16.133.28	172.16.139.250	308
172.16.133.30	172.16.139.250	308
172.16.133.97	172.16.139.250	307
172.16.133.20	172.16.139.250	306
172.16.133.21	172.16.139.250	306

Table 2: Shows IP addresses and repetitions

Top Traffic MAC Conversations

This section tries to clarify what are the MAC conversations that represent the vast majority of the traffic analyzed. In the following table we'll find the conversations that make up 99% of the traffic, with traffic meaning here the amount of bytes. If the 99% quota wasn't reached in less than 20 different conversations, then at least we'll see a top 20 of the conversations that represent most of the traffic.

MAC Address 1	*MAC Address 2*	*Share of Traffic (%)*
00:21:70:67:6e:2a	00:90:7f:3e:02:d0	6,11
14:10:9f:d3:ec:9d	00:90:7f:3e:02:d0	5,56
00:90:7f:3e:02:d0	00:21:70:67:5f:47	5,53
00:21:70:63:40:c5	00:90:7f:3e:02:d0	4,77
00:21:70:64:71:ce	00:90:7f:3e:02:d0	4,41
00:21:70:63:3a:e9	00:90:7f:3e:02:d0	4,17
00:90:7f:3e:02:d0	d4:be:d9:28:21:33	3,85
00:21:70:63:3f:ff	00:90:7f:3e:02:d0	3,77
00:21:70:63:32:bf	00:90:7f:3e:02:d0	3,19
00:90:7f:3e:02:d0	00:50:43:01:4d:d4	3,13
00:90:7f:3e:02:d0	00:21:70:63:3b:d6	2,92
00:21:70:67:61:4b	00:90:7f:3e:02:d0	2,86
00:21:70:67:15:9f	00:90:7f:3e:02:d0	2,32
00:21:70:67:6f:50	00:90:7f:3e:02:d0	2,16
00:90:7f:3e:02:d0	00:21:70:67:70:23	2,15
14:10:9f:cf:e1:71	00:90:7f:3e:02:d0	2,07
00:90:7f:3e:02:d0	00:19:b9:da:15:a0	1,86
9c:b7:0d:3f:fb:dd	00:90:7f:3e:02:d0	1,85
d4:be:d9:4b:24:79	00:90:7f:3e:02:d0	1,84
00:90:fb:3f:de:42	00:90:7f:3e:02:d0	1,77

Table 3: Shows MAC addresses and traffic volume

Top Traffic IP Conversations

This section of the report emulates the analysis performed in the previous one, but with a Level 3 approach, using IP addresses.

IP Address 1	*IP Address 2*	*Share of Traffic (%)*
172.16.133.95	157.56.240.102	5,71
172.16.133.36	67.217.64.99	5,23
67.217.64.99	172.16.133.26	4,84
172.16.133.25	74.125.170.42	3,60
172.16.133.73	74.125.170.143	2,75
172.16.133.116	96.43.146.48	2,60
172.16.133.39	174.129.24.9	2,17
172.16.128.201	172.16.133.6	1,86
172.16.133.39	132.245.1.150	1,51
172.16.133.114	157.56.242.198	1,51
172.16.133.55	157.56.232.214	1,44
172.16.133.87	74.125.226.70	1,43
172.16.133.18	157.56.238.6	1,29
169.233.51.66	172.16.133.78	1,09
172.16.133.67	23.33.47.100	1,07
172.16.133.163	15.193.0.234	1,04
208.92.54.5	172.16.133.56	1,03
172.16.133.37	23.13.211.100	1,02
172.16.133.184	205.216.16.228	1,00
172.16.133.93	74.63.52.167	0,97

Table 4: Shows IP addresses and traffic volume

Top 10 Production MAC Addresses (bytes)

In this part of the document, we'll start focusing our analysis in addresses, rather than conversations. Our starting section shows a top 10 of the MAC addresses in bytes production. The concept explains itself: the MAC addresses that produced (sent) the biggest amount of bytes.

MAC Address (source)	***Bytes Sent***
00:90:7f:3e:02:d0	597149875
00:21:70:67:6e:2a	52511373
14:10:9f:d3:ec:9d	49440853
00:21:70:63:3f:ff	16286356
00:21:70:63:3a:ab	14318890
00:21:70:63:32:bf	8852640
00:21:70:63:3b:d6	8485090
c0:18:85:5a:fa:04	8430083
00:21:70:67:6d:03	8393454
00:21:70:63:32:60	8054570

Table 5: Shows source MAC addresses and amount of bytes sent

Top 10 Production IP Addresses (bytes)

As previously done on the report, we jump onto Level 3 analysis of the previous idea: find the top 10 IP addresses that generated the biggest amount of bytes.

IP Address (source)	***Bytes Sent***
172.16.133.95	52511373
172.16.133.36	49440853
67.217.64.99	44834347
96.43.146.48	33245128
74.125.170.42	32875668
74.125.170.143	25135668
174.129.24.9	19699730
96.43.146.176	17335944
157.56.242.198	17154820
172.16.128.201	16368541

Table 6: Shows source IP addresses and amount of bytes sent

Top 10 Reception MAC Addresses (bytes)

Two sections before, we analyzed the bytes sent or produced by MAC addresses. In this section we take the exact same idea, but apply it the other way around: we want to know the top 10 MAC addresses that received the biggest amount of bytes.

MAC Address (destination)	***Bytes Received***
00:90:7f:3e:02:d0	323751427
00:21:70:67:5f:47	48168921
00:21:70:63:40:c5	40275510
00:21:70:64:71:ce	37747491
00:21:70:63:3a:e9	34694219
d4:be:d9:28:21:33	34234436
00:50:43:01:4d:d4	26097470
00:21:70:67:61:4b	23300624
00:21:70:63:32:bf	20535696
00:21:70:67:15:9f	18913071

Table 7: Shows destination MAC addresses and amount of bytes received

Top 10 Reception IP Addresses (bytes)

Just like we did before, we continue with the Level 2 (MAC) -> Level 3 (IP) dynamic. In the following table we can find the top 10 IP addresses in bytes received (with the biggest amount of bytes received).

IP Address (destination)	***Bytes Received***
172.16.139.250	100513077
157.56.240.102	52327526
172.16.133.26	48168921
67.217.64.99	47879324
172.16.133.25	40275510
172.16.133.73	37747491
172.16.133.78	34694219
172.16.133.39	34234436
172.16.133.132	26097470
172.16.133.87	23300624

Table 8: Shows destination IP addresses and amount of bytes received

Top 10 Production MAC Addresses (packets)

If we analyzed before the MAC addresses that held the title for biggest amount of bytes sent, we're now jumping onto biggest amount of packages sent. The following table may display a totally different list from the one in the mentioned section, as a big number of bytes does not necessarily always imply a big number of packages (and vice versa).

MAC Address (source)	***Packets Sent***
00:90:7f:3e:02:d0	425163
00:21:70:67:6e:2a	30575
14:10:9f:d3:ec:9d	26994
00:21:70:63:3f:ff	22331
00:21:70:63:3a:e9	16540
c0:18:85:5a:fa:04	15764
00:21:70:63:32:bf	13928
00:21:70:63:3b:d6	13548
00:21:70:63:40:c5	12583
00:21:70:67:6d:03	12365

Table 9: Shows source MAC addresses and amount of packets sent

Top 10 Production IP Addresses (packets)

As the reader will notice, we're following the same path one more time: use the same approach but on Level 3. We can see in the following table the top 10 IP addresses in packets sent or produced.

IP Address (source)	***Packets Sent***
172.16.133.95	30575
67.217.64.99	29937
172.16.133.36	26994
96.43.146.48	23231
172.16.133.116	22331
172.16.133.78	16540
172.16.133.42	15764
74.125.170.42	14776
157.56.240.102	13989
172.16.133.67	13928

Table 10: Shows source IP addresses and amount of packets sent

Top 10 Reception MAC Addresses (packets)

As we did with the amount of bytes received, we now display the MAC addresses with the biggest amount of packets received. As always, the list is limited to the top 10 addresses.

MAC Address (destination)	***Packets Received***
00:90:7f:3e:02:d0	529495
00:21:70:67:5f:47	27610
00:21:70:63:3a:e9	24430
00:21:70:63:40:c5	22147
00:21:70:64:71:ce	18827
d4:be:d9:28:21:33	17632
00:50:43:01:4d:d4	15449
00:21:70:67:61:4b	14822
00:21:70:63:3f:ff	13645
00:21:70:67:6e:2a	12425

Table 11: Shows destination MAC addresses and amount of packets received

Top 10 Reception IP Addresses (packets)

One more time, we switch the previous section analysis to IP Level. The top 10 IP addresses with the biggest amounts of packets are displayed in the table of this section.

IP Address (destination)	*Packets Received*
172.16.139.250	233259
67.217.64.99	28917
157.56.240.102	28503
172.16.133.26	27610
172.16.133.78	24430
172.16.133.25	22147
172.16.133.73	18827
172.16.133.39	17632
96.43.146.48	16708
172.16.133.132	15449

Table 12: Shows destination IP addresses and amount of packets received

Top Traffic Source MAC Addresses

As in one of the early sections, where we analyzed the conversations that made up the biggest portion of traffic, we're now going to find out which are the source MAC addresses that represent the 99% of the whole traffic. This means knowing the volume of data produced (sent) that each MAC address produced, and with that information, find the addresses that represent the 99% or, as we did previously, display a top 20 of source addresses even if they don't reach the 99% quota.

MAC Address (source)	*Share of Traffic (%)*
00:90:7f:3e:02:d0	64,84
00:21:70:67:6e:2a	5,70
14:10:9f:d3:ec:9d	5,37
00:21:70:63:3f:ff	1,77
00:21:70:63:3a:ab	1,55
00:21:70:63:32:bf	0,96
00:21:70:63:3b:d6	0,92
c0:18:85:5a:fa:04	0,92
00:21:70:67:6d:03	0,91
00:21:70:63:32:60	0,87
68:94:23:00:2a:28	0,84
00:21:70:63:22:40	0,82
9c:8e:99:f3:8c:1e	0,71
00:21:70:67:6f:50	0,65
14:10:9f:d4:90:db	0,57
00:21:70:67:6b:55	0,52
d4:be:d9:4b:24:79	0,47
00:21:70:64:71:60	0,40
00:21:70:63:40:c5	0,40
00:21:70:63:3a:e9	0,40

Table 13: Shows source MAC addresses and traffic volume

Top Traffic Source IP Addresses

Once again, we apply the same idea that has been guiding the whole report: analyze the same concept but taking into account IP addresses (Level 3 information) instead of MAC addresses (Level 2 information). Following these lines we'll find the table that represents the source IP addresses that hold the 99% of data traffic (or the top 20 with the biggest shares of traffic).

IP Address (source)	*Share of Traffic (%)*
172.16.133.95	5,70
172.16.133.36	5,37
67.217.64.99	4,87
96.43.146.48	3,61
74.125.170.42	3,57
74.125.170.143	2,73
174.129.24.9	2,14
96.43.146.176	1,88
157.56.242.198	1,86
172.16.128.201	1,78
172.16.133.116	1,77
172.16.133.55	1,55
132.245.1.150	1,54
74.125.226.70	1,39
169.233.51.66	1,08
23.33.47.100	1,05
208.92.54.5	1,02
157.56.238.6	1,01
23.13.211.100	0,99
15.193.0.234	0,98

Table 14: Shows source IP addresses and traffic volume

Top Traffic Destination MAC Addresses

We jump one more time to the other end of a conversation. We're now looking for the MAC addresses in the receiving end (destination addresses) that represent the 99% of the traffic captured in the processed file. As we previously did, we limit these addresses to an amount of 20.

MAC Address (destination)	*Share of Traffic (%)*
00:90:7f:3e:02:d0	35,16
00:21:70:67:5f:47	5,23
00:21:70:63:40:c5	4,37
00:21:70:64:71:ce	4,10
00:21:70:63:3a:e9	3,77
d4:be:d9:28:21:33	3,72
00:50:43:01:4d:d4	2,83
00:21:70:67:61:4b	2,53
00:21:70:63:32:bf	2,23
00:21:70:67:15:9f	2,05
00:21:70:63:3b:d6	2,00
00:21:70:63:3f:ff	2,00
00:21:70:67:70:23	1,97
00:19:b9:da:15:a0	1,78
14:10:9f:cf:e1:71	1,71
00:90:fb:3f:de:42	1,64
9c:b7:0d:3f:fb:dd	1,56
00:21:70:67:6f:50	1,52
5c:26:0a:03:13:b7	1,45
d4:be:d9:4b:24:79	1,37

Table 15: Shows destination MAC addresses and traffic volume

Top Traffic Destination IP Addresses

For the las time in this document, we switch to Level 3 analysis using the approach showed in the previous section. Note: This is the las section in the current report model. More tables, images and/or graphs can be added using the Automatic Report Generation tool. Adding more sections is as simple as adding a new calculation to the analyzed data, then create an element (graph, table, etc.) with the information and add to the report a new section with this element, a title and a description. Thanks for reading.

IP Address (destination)	*Share of Traffic (%)*
172.16.139.250	10,91
157.56.240.102	5,68
172.16.133.26	5,23
67.217.64.99	5,20
172.16.133.25	4,37
172.16.133.73	4,10
172.16.133.78	3,77
172.16.133.39	3,72
172.16.133.132	2,83
172.16.133.87	2,53
172.16.133.67	2,23
172.16.133.93	2,05
96.43.146.48	2,01
172.16.133.37	2,00
172.16.133.116	2,00
172.16.133.163	1,97
172.16.133.6	1,78
172.16.133.18	1,71
172.16.133.184	1,64
172.16.133.114	1,56

Table 16: Shows destination IP addresses and traffic volume

Final Notes

Note that the data displayed in this report is analyzed from test capture files found on the internet, so full accuracy (or sense) is not to be expected. Also, some of the registers in those test files might not be in the correct format (a format that the processing tool does not recognize) or might use protocols no supported by the processing tool, so depending on the files, more or less records are lost (in the current report, just a few records are lost since it's mostly normal traffic). This is the end of the current report model. More tables, images and/or graphs can be added using the Automatic Report Generation tool. Adding more sections is as simple as adding a new calculation to the analyzed data (from the capture files), then creating an element (graph, table, etc.) with the information and adding a new section with that element, a title and a description to the report. Thanks for reading.

B. TABLAS PARA WORD

Some Custom Table

IP.src	*IP.dst*	*Bytes*
192.168.1.9	192.168.1.35	10254
192.168.1.9	192.168.1.35	10254
192.168.1.9	192.168.1.35	10254
192.168.1.9	192.168.1.35	10254
192.168.1.9	192.168.1.35	10254
192.168.1.9	192.168.1.35	10254
192.168.1.9	192.168.1.35	10254

Table 1: Custom Table Test

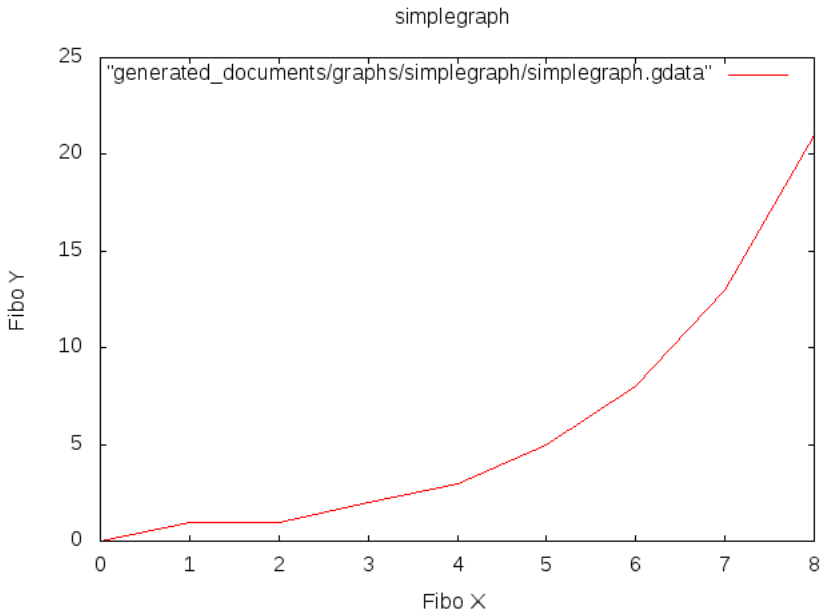
C. IMÁGENES PARA WORD

Test Image

Picture 1: My first image

D. GRÁFICAS PARA WORD

Test Graph



Graph 1: My first graph with Fibonacci data

E. MANUAL DE INSTALACIÓN

El programa desarrollado en este trabajo cuenta con dos versiones para ser ejecutado y obtener como resultado los informes presentados en las pruebas:

1. Desde el IDE (Debug)
2. Archivo ejecutable (Release)

Para las pruebas se utiliza la versión de Debug, por lo que se ejecuta el proyecto directamente desde Eclipse, el IDE utilizado para su desarrollo. Las distintas pruebas se realizan aplicando pequeñas modificaciones al código previas a cada ejecución.

Le versión de Release, un archivo empaquetado para entregar, consta de los siguientes elementos dentro de la carpeta del proyecto:

- Archivo ejecutable principal (.jar)
- Conjunto de elementos de entrada (carpeta y subcarpetas propios)
- Conjunto de elementos generados (carpeta y subcarpetas propios)
- Conjunto de librerías de dependencia (carpeta y subcarpetas propios)

El ejecutable principal utiliza las librerías de dependencia y los ficheros de entrada (generados por el ProcesaTrazas) para generar los documentos y archivos de salida (informes, imágenes, gráficas, etc.). Tanto las rutas a los archivos de entrada como las rutas de los elementos de salida están predefinidas, por lo cual no pueden ser modificadas.

Para ejecutar el archivo .jar necesitamos un entorno que cuente con Java instalado, teniendo en cuenta que dependiendo del Sistema Operativo elegido contaremos con más o menos funcionalidades. A continuación se explica el proceso de ejecución en Linux y Windows, con las ventajas y/o desventajas que ofrece cada uno

- Windows
  - No cuenta con la funcionalidad de gráficas.
  - Utiliza un archivo de entrada por defecto que debe llamarse “inputfile”.
  - Genera un documento de salida con el nombre predefinido de “report”.
  - Puede ejecutarse directamente lanzando el archivo como cualquier otro ejecutable en Windows.
- Linux
  - Cuenta con la funcionalidad de gráficas.
  - Puede especificarse el archivo de entrada como primer argumento al ejecutar el programa (debe estar en la carpeta de input_files)
  - Puede especificarse el nombre del archivo de salida como segundo argumento al ejecutar el programa (se genera en la carpeta generated_documents/reports)
  - Debe ejecutarse el archivo .jar utilizando el comando “java -jar ejecutable” donde “ejecutable” corresponde al nombre del archivo .jar principal.