UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Análisis de Tráfico en Internet Utilizando Distribuciones Alfa estables en procesadores paralelos

Christian Bizumuremyi Herrero
Tutor: De Pedro Sánchez, Luis
Ponente: López de Vergara Méndez, Jorge Enrique

JUNIO 2017

Análisis de Tráfico en Internet Utilizando Distribuciones Alfa estables en procesadores paralelos

AUTOR: Christian Bizumuremyi Herrero
TUTOR: Luis De Pedro Sánchez

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2017

Resumen

[1] En el contexto actual de desarrollo de aplicaciones basadas en TCP/IP (Internet), el análisis del tráfico IP es una necesidad crucial con el objetivo de prever y solucionar posibles problemas de congestión que dificulten el uso adecuado de dichas aplicaciones. La disponibilidad de modelos matemáticos del tráfico permite anticiparse a futuros problemas en las redes de comunicaciones y además automatizar tareas de monitorización y control que forman parte de la gestión de redes actual.

En los últimos años se han propuesto modelos estadísticos basados en las distribuciones alfa estables que se han revelado prometedores, pero que demandan una capacidad de cálculo intensa que limita su aplicación en situaciones de control de red en las que las decisiones se deben tomar en tiempo real.

El abaratamiento de la tecnología de procesadores paralelos permite acelerar la ejecución de los algoritmos de estimación de parámetros de tráfico y están habilitando la utilización de las distribuciones alfa estables en el análisis y la resolución de problemas de tráfico.

El objetivo de este trabajo sería partir de la situación actual en la que se han implantado algoritmos de estimación de parámetros de tráfico basados en el modelo alfa estable y adaptarlos al análisis de tráfico real, de manera que se puedan utilizar para intentar predecir el comportamiento de una red, lo que supondría la posibilidad de detectar anomalías, cambios en la red o incluso, predecir su comportamiento para llegar a prevenir posibles problemas.

Para ello se realizará un estudio de la red, en las que se capturarán muestras de tráfico reales que posteriormente serán ajustadas para ser interpretadas usando distribuciones alfa estables.

Palabras clave

Distribución de probabilidad, Alfa estable, Servidor, sitio web, Matlab, Test Kolmogorov-Smirnov, Hipótesis nula, Ping, Matlab, Libstable, LectorTrazas, .BAT, modelo estadístico

Abstract

In the current context of developing applications based on TCP / IP (Internet), the analysis of the IP traffic is a necessity in order to anticipate and solve possible congestion problems that hinder the proper use of such applications.

The existence of mathematical models of traffic allow to anticipate future problems in the communications networks and also automate monitoring and control tasks that are part of the current network management.

In recent years statistical models have been proposed based on stable alpha distributions, which have been promising, but which demand an intense computational capacity that limits its implementation in situations of network control in which decisions must be taken in real time.

Cheaper parallel processor technologies make possible the acceleration of the execution of algorithms for estimating traffic parameters and they are enabling the use of stable alpha distributions in the analysis and resolution of traffic problems.

The objective of this work is to start from the current situation in which algorithms have been implemented to estimate traffic parameters based on the stable alpha model and to adapt them to the analysis of real traffic. So that they can be used to try to predict the behavior of a network, which would mean the possibility of detecting anomalies, changes in the network or even, predict their behavior in order to prevent potential problems.

Therefore, a study of the network will be carried out, in which real traffic samples will be captured and later adjusted to be interpreted using stable alpha distributions.

Key words

Density distribution, Alpha stable, Host, website, Matlab, Test Kolmogorov-Smirnov, Null hypothesis, Ping, Matlab, Libstable, LectorTrazas, .BAT, Statistical model.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

Glosario

API	Application Programming Interface, “Interfaz de Programación de aplicaciones”.
MATLAB	Abreviatura de MATrix LABoratory, "laboratorio de matrices".
CPU	Central Processing Unit, “Unidad central de Procesamiento”.
GPU	Graphics Processor Unit, “Unidad de Procesamiento Gráfico”.
DSP	Digital Signal Processor, “Procesador digital de Señales”.
FPGA	Field Programmable Gate Array.
IDE	Integrated Develpment Environment
KS	Kolmogórov-Smirnov
LPI	Image Processing Laboratory
GSL	GNU Scientific Library
CDF	Cumulative Distribution Fuction
PDF	Probability Distribution Function
CDF^{-1}	Inverse of Cumulative Distribution Fuction

1 Introducción

En este capítulo se detallan las necesidades que dan lugar a este Trabajo Fin de Grado (**TFG**), planteando los objetivos que habrán de cumplirse para llevarlo a cabo. Además, se expondrá la estructura que seguirá este documento.

1.1 Motivación

El análisis del tráfico **IP** es una práctica cada vez más común y necesario que lejos dista ya de ser una tarea a la que sólo un administrador de red tuviera que enfrentarse. Con una buena monitorización de la red se puede percibir una gran cantidad de fenómenos que forman parte del tráfico, desde timeouts formados por pérdidas de conexión, hasta cambios de ruta debido a redes congestionadas.

Existen infinidades de programas y aplicaciones que permiten capturar y/o interpretar el tráfico que concurre por nuestra red.

Gracias al trabajo de distintos compañeros de carrera, existen algoritmos de estimación de tráfico basados en el modelo α-estable. Este conjunto de algoritmos está recogido en la librería *Libstable*, la cual se implementará en la aplicación encargada del cálculo eficiente de distribuciones de probabilidad α-estables.

Debido a la existencia de esta herramienta nace la necesidad de adaptarla para analizar tráfico real y poder observar qué le sucede a la red desde el punto de vista que nos otorgan las distribuciones α-estables.

Para ello este trabajo consistirá en establecer un ciclo autosuficiente que capturará, adaptará, analizará y representará el trafico **IP**.

1.2 Objetivos

El principal objetivo establecido para este TFG reside establecer una serie de lo que llamaremos "fases" para conseguir interpretar tráfico red real a través de la librería *Libstable* y poder llegar interpretar/predecir el comportamiento de una red. En dichas fases se utilizarán, codificarán y/o adaptarán aplicaciones.
Lo cual ha sido posible mediante el uso de los conocimientos aprendidos durante el grado, en especial, en asignaturas como *Programación I y II, Redes Informáticas I y II, Programación y Computación de Sistemas Paralelos, Probabilidad y estadística.*

Tras la división del procedimiento en fases, el objetivo principal estará ligado al cumplimiento del objetivo de cada fase que podrían definirse como:

- Definir un modelo de datos a tratar el cual sirva para representar tráfico real.
- Adaptar el conjunto de datos a distribuciones estables y justificar su ajuste.
- Representar los parámetros de la distribución resultante obtenida con la interpretación de los datos obtenidos en previas fases.

1.3 Fases de Realización

En este apartado se da una visión generalizada de las fases descritas en el anterior apartado, son las fases que seguirá el trabajo y cuya correcta consecución nos aportará unos resultados que nos permitirán catalogar el tráfico de red a medir.
Las fases son:

- **Estudio:** Investigación sobre las distribuciones estables. Se estudiará la viabilidad de recoger tráfico red y adaptarlo a un modelo matemático estadístico como es el basado en α-estables. Para ello se recogerán datos de tráfico real cuya representación como distribución α-estable sea aceptada por el test de Kolmogorov Smirnov [2].

- **Captura:** Una vez indicado el tipo de dato a recoger, se crearán scripts que sean capaces de capturar ese tráfico en distintas plataformas, otorgándoles una estructura común que permita se traducida correctamente por la aplicación diseñada para tal tarea. En esta fase, se definirán los puntos de captura y los periodos de tiempo a analizar.

- **Adaptación:** Ya que los datos obtenidos tras la fase de captura no pueden ser leídos directamente, ha de ejecutarse una fase de adaptación de dichos datos. Esta fase almacenará tan sólo la información necesaria y útil que más adelante formará la distribución α-estable. Para ello será preciso la creación de un programa

de escritorio que sea capaz de leer los ficheros con los datos obtenidos en la fase de captura, y que devuelva los datos listos para ser representados.

- **Representación:** Fase encargada de recoger los datos ya traducidos y representarlos acorde a una distribución α-estable. Se hará uso de la librería *Libstable* la cual habrá sido adaptada para leer dichos datos y devolver la estimación de los parámetros que constituyen una distribución α-estable. Con el objetivo de acabar representado los distintos valores de dichos parámetros utilizando herramientas como Matlab que poseen funciones para visualizar los datos de las distribuciones.

- **Análisis de resultados:** Una vez representados los distintos valores que obtienen los parámetros Alfa, Beta, Gamma y Delta en nuestras muestras de tráfico real, se formularán una serie de hipótesis basadas en los datos obtenidos. Dichas hipótesis tratarán de justificar el comportamiento de la red y podrán hasta predecir su comportamiento futuro en base a las mediciones capturadas.

1.4 Estructura del Documento

Este **TFG** se encuentra dividido en unidades nombradas como capítulos, siendo este apartado el final del conjunto de apartados que conforma el primer capítulo.
Este apartado se resumirá el contenido de los capítulos posteriores del trabajo.

Nombrado como el *capítulo 2*, se detallará el estudio del estado del arte, explicando por separado todo lo necesario acerca de los conceptos teóricos y las tecnologías utilizadas.

En el *capítulo 3* se enumerarán los requisitos del proyecto necesarios para conseguir la completitud de las distintas fases que lo conforman, así como un análisis de las aplicaciones utilizadas en cada una de ellas.

Seguido tenemos el *capítulo 4* donde se explicará en detalle cada paso realizado en las distintas fases, entrado narrando los aspectos más técnicos del diseño empleado.

El *capítulo 5* servirá como ventana para exponer los resultados obtenidos y formular las distintas hipótesis y evidencias representadas en los distintos elementos resultantes obtenidos.
Por último, en el *capítulo 6*, el lector podrá encontrar donde quedarán definidas las conclusiones del autor sobre el trabajo, junto a su opinión sobre el trabajo futuro que se puedan realizar al proyecto.

2 Estado del arte

2.1 Introducción

Para conseguir una mejor implementación de cada fase se investigarán y compararán las técnicas existentes y los conceptos teóricos aplicados. También se detallarán la búsqueda y obtención de un método para la validación del ajuste de los parámetros α-estables estimados.

2.2 Distribuciones de probabilidad

La distribución de probabilidad de una variable aleatoria es una función que asigna a cada suceso definido sobre la variable aleatoria la probabilidad de que dicho suceso ocurra. La distribución de probabilidad es definida sobre el conjunto de los sucesos y cada uno de los sucesos es el rango de valores de la variable aleatoria.

Definimos variable aleatoria como aquella cuyo valor es el resultado de un evento de carácter aleatorio. Lo que supone que el valor de dicha variable cambia en sentido azaroso con cada evento. Existen dos tipos, variable aleatoria discreta y variable aleatoria continúa.

- Variable aleatoria discreta: es aquella variable cuyos valores están acotados o establecidos y forman parte de un conjunto de valores finito, frecuentemente valores enteros.

- Variable aleatoria continua: es aquella variable dada principalmente por la medición, toma valores que no rigen ninguna pauta y pueden tomar cualquier valor dentro de un intervalo.

2.3 . Distribuciones estables

De entre las diferentes distribuciones que existen, consideramos que una distribución de probabilidad es estable si dicha distribución de probabilidad cumple la propiedad que dicta:

"cualquier combinación lineal de dos o más copias independientes de una muestra aleatoria que tiene la misma distribución de probabilidad, salvo por quizá algún parámetro de localización o factor de escala". [3]

Las distribuciones estables son una rica clase de distribución de probabilidad que acepta falta de simetría y colas pesadas además de tener numerosas propiedades matemáticas interesantes.

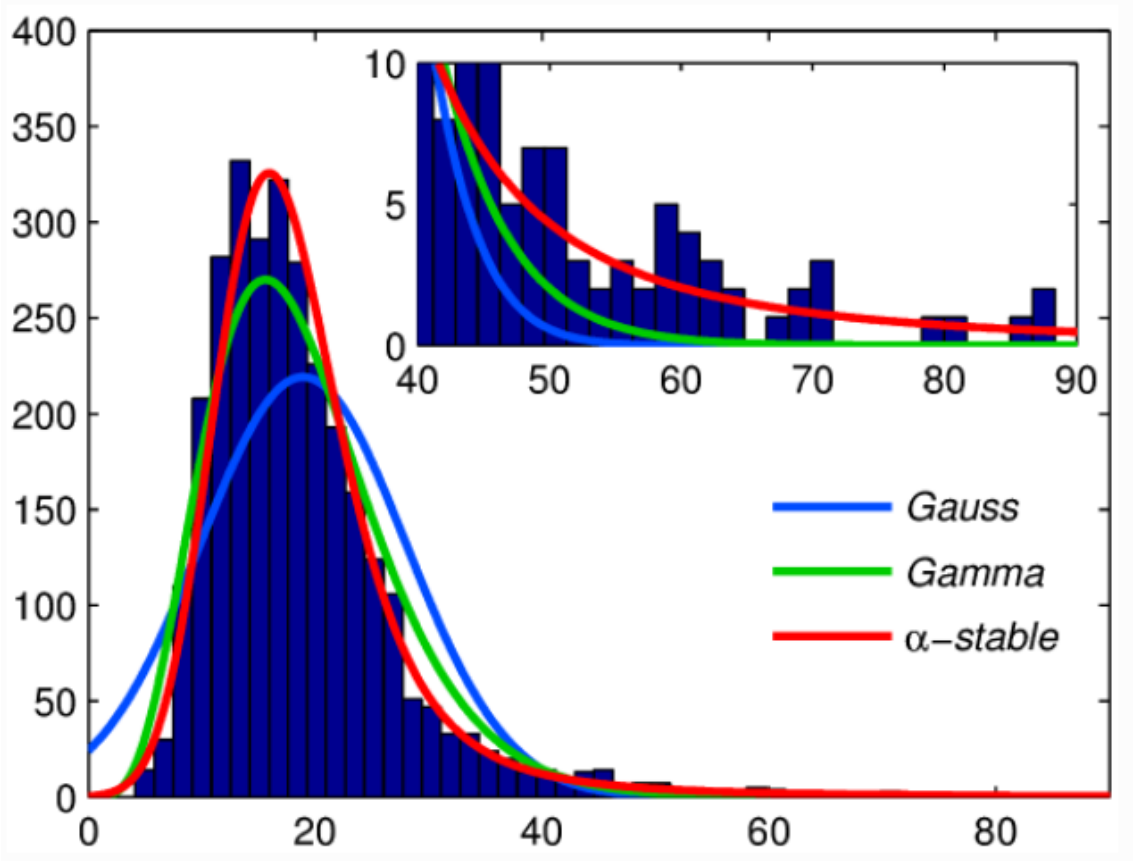


Figura 1. Comparativa de ajusta entre diversos tipos de distribuciones de probabilidad [4].[i]

Esta clase de distribuciones fue presentada por Paul Lévy en su estudio del sumatoria de variables aleatoria independientes e idénticamente distribuidas en 1920.

Una distribución de probabilidad de una variable aleatoria continua es una distribución estable si satisface la siguiente propiedad:

Sean X1 y X2 copias independientes de una variable aleatoria X. Entonces se dice que X es estable si por cada constante a > 0 y b > 0 la variable aleatoria aX1 + bX2 posee la misma distribución que cX + d para las constantes c > 0 y d. Se dice que es estrictamente estable cuando d = 0. [3]

Existen distribuciones conocidas como son la distribución normal, la distribución de Cauchy y la distribución de Lévy son poseedoras de la propiedad mencionada arriba, por lo cual se consideran casos especiales de las distribuciones α-estables.

Aunque la función de densidad de probabilidad (**PDF**) para las distribuciones estable generales no puede ser analíticamente escrita, la función característica (**CF**) si puede, considerando $\Phi(t) = \exp[\Psi(t)]$ [1]su forma es: [3]

$$\Psi(t)=\begin{cases}-|\sigma t|^{\alpha}\left[1-i\beta\tan\left(\frac{\pi\alpha}{2}\right)\operatorname{sign}(t)\right]+i\mu t, & \alpha\neq 1,\\ -|\sigma t|\left[1+i\beta\frac{2}{\pi}\operatorname{sign}(t)\ln\left(|t|\right)\right]+i\mu t, & \alpha=1,\end{cases}$$

Donde:

$$\operatorname{sign}(t) = \begin{cases} 1, & t > 0, \\ 0, & t = 0, \\ -1, & t < 0. \end{cases}$$

Los cuatro parámetros mostrados arriba son:

- Alfa (α): este parámetro define la estabilidad de la distribución. Es el parámetro más representativo, de ahí que a este tipo de distribución de probabilidad se le conozca comúnmente como distribuciones α-estables.
- Beta (β): define la asimetría de la distribución.
- Gamma (γ): define la escala de la distribución
- Delta (δ): define la posición relativa de la distribución.

Los parámetros están restringidos por los rangos: $\alpha \in (0,2]$, $\beta \in [-1,1]$, $\gamma \geq 0$ and $\delta \in R$.
Puesto que α y β determinan la forma de la distribución, son considerados parámetros de forma.
Una distribución alfa estable es denominada estándar si $\gamma = 1$ y $\delta = 0$.
Cuando $\alpha = 2$ la distribución se convierte en una distribución normal con una desviación estándar de $\gamma / \sqrt{2}$ y una media de δ (β se vuelve irrelevante).
La distribución Cauchy mencionada anteriormente es la distribución resultante de fijar el valor de $\alpha = 1$ y $\beta = 0$ junto con los parámetros de escala γ y el parámetro de localización δ. Para las distribuciones Lévy los valores a fijar son $\alpha = 0.5$ y $\beta = 1$. En estos casos especiales de distribuciones estables es posible expresar analíticamente la **PDF**. De otra forma se ha de calcular numéricamente.

A través de la fórmula de inversión de Fourier:

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \phi(t) \mathrm{e}^{-itx} \, \mathrm{d}t = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \mathrm{e}^{\psi(t) - itx} \, \mathrm{d}t. \qquad [6]$$

Sustituyendo la primera ecuación en la segunda ecuación aparece como resultado una integral que aún no puede ser resuelta analíticamente, pero sí evaluada con métodos numéricos. En consecuencia, la Transformada rápida de Fourier [5] provee un algoritmo para evaluar eficientemente la integral resultante de realizar la sustitución de las ecuaciones mostradas anteriormente.

Nolan [6], obtuvo un nuevo conjunto de ecuaciones partiendo de las originales, consiguiendo además una precisión relativa de orden 10^{-14} en el espacio de parámetros.

Sin embargo, con este nuevo set de ecuaciones, nos encontramos con dificultades a la hora de afrontar la continuidad de la distribución cuando $\alpha = 1$ que se solventa utilizando una parametrizacion ligeramente diferente basada en la parametrizacion M de Zolotarev[7]:

$$\Psi_0 = \begin{cases} -|\sigma t|^{\alpha}\left[1+i\beta\tan\left(\frac{\pi\alpha}{2}\right)\operatorname{sign}(t)\left(|\sigma t|^{1-\alpha}-1\right)\right]+i\mu_0 t, & \alpha\neq 1, \\ -|\sigma t|\left[1+i\beta\frac{2}{\pi}\operatorname{sign}(t)\ln\left(|\sigma t|\right)\right]+i\mu_0 t, & \alpha=1. \end{cases}$$

Donde los parámetros α, β and γ siguen conservando sus anteriores significados mientras que el parámetro de localización original δ y el modificado δ_0 están sujetos a:

$$\mu = \begin{cases} \mu_0 - \beta\tan\left(\frac{\alpha\pi}{2}\right)\sigma, & \alpha \neq 1, \\ \mu_0 - \beta\frac{2}{\pi}\sigma\ln(\sigma), & \alpha = 1. \end{cases} \qquad [3]$$

Con esta modificación, el resultado de la distribución es continuo en sus cuatro parámetros, que es lo conveniente cuando se estiman los parámetros de distribución o se aproximan sus **PDF** o **CDF**.

2.4 . Computación Paralela

Tradicionalmente, el software generado es escrito para computación en serie. Lo cual se puede definir por los siguientes pasos:

COMPUTACIÓN EN SERIE:

- Un problema es dividido en una serie discreta de instrucciones.
- Las instrucciones son ejecutadas de forma secuencial una después de otra.
- Estas instrucciones son ejecutadas en un mismo procesador.
- Tan solo una instrucción puede ejecutarse al mismo tiempo.

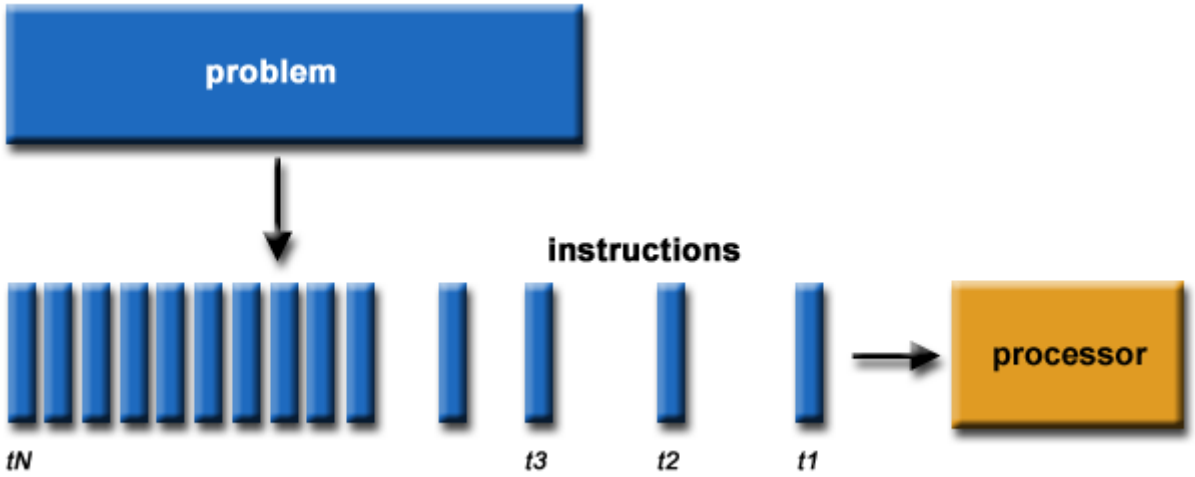


Figura 2. Representación de resolución de problema en serie. [9]

En contra la programación paralela se podría definir como:

- Un problema es divido en una cantidad discreta de partes que pueden ser resueltas individualmente.
- Cada parte está compuesta de una serie de instrucciones.
- Las instrucciones de cada parte se ejecutan simultáneamente en diferentes procesos.
- Se emplea un mecanismo de control y coordinamiento general.

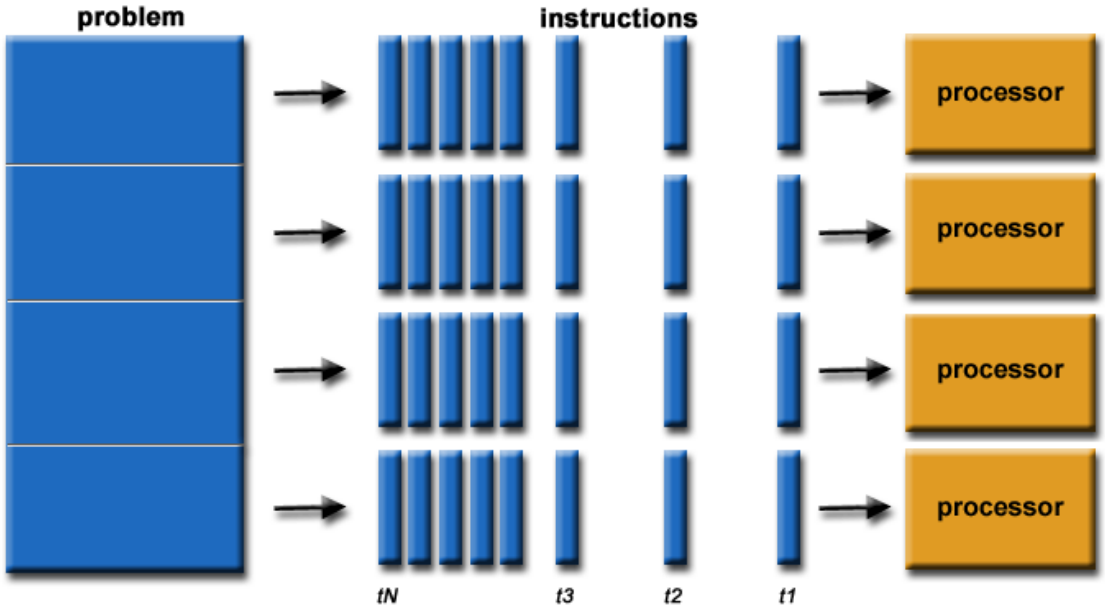


Figura 3. Representación de resolución de problema en paralelo. [9]

El cálculo numérico de las integrales expuestas en el anterior apartado requiere un coste computacional muy elevado, por ello se ha optado por la implementación de una arquitectura paralela que sea capaz de acelerar el proceso de obtención de los parámetros de la distribución a analizar.

La resolución consiste en delegar la parte de computación numérica gruesa a la **GPU**, un método denominado que podría denominarse como "co-procesamiento" donde el sistema deja de optar por un "procesamiento central" centrado solo en la **CPU**, y delega en uno repartido entre **CPU** y **GPU**. Con los que se consiste realizar un gran número de operaciones en coma flotante de forma paralela en cada núcleo de la cual se compone la tarjeta gráfica.

Existen diferentes frameworks para implementar dicha idea y tienden a depender del modelo de la gráfica del computador. Estos frameworks acaban dividiendo el trabajo en hilos de ejecución que a su vez forman bloques de N hilos que acaban siendo procesador por cada núcleo de la tarjeta.

Existen dos frameworks principales que son:

OPENCL:

> Es un framework para escribir programas que se ejecuten a través de plataformas heterogéneas compuestas de **CPUs**, **GPUs**, **DSPs**, **FPGAs** y otros posibles procesadores o aceleradores de hardware. OpenCL otorga lenguajes de programación basados en **C99** y **C++11** para programar aquellos dispositivos. OpenCL otorga una interfaz estándar para la computación paralela usando paralelismo bastado en tareas y datos. Es un estándar de código abierto válido para casi, si no todas, las tarjetas gráficas. [10]

CUDA:

> Es una plataforma y modela de interfaz de computación paralela creada por *Nvidia*. Permite a los desarrolladores de software el uso de la **GPU** para el procesamiento de cómputo general en las tarjetas Nvidia. Es una capa de software que proporciona acceso directo a un conjunto virtual de instrucciones de la **GPU** y elementos de computación paralela para la ejecución de bloque de trabajo agrupados en lo que denominan Kernels. [11]

2.5 . Conclusiones

Para modelar la información relativa al tráfico **IP** capturado y de cara a descubrir y estudiar las tendencias y comportamiento se va a usar un modelo estadístico, dicho modelo será el correspondiente al de una distribución α-estable.
Este modelo posee la ventaja de adaptarse bien a la alta variabilidad del tráfico de red, puedo identificar y definir el tráfico e incluso distinguiendo cuando el tráfico sufre un patrón anómalo.
Debido a sus cuatro parámetros característicos: alfa, beta, gamma y delta ofrecen un mayor número de grados de libertad que otros modelos, en contra, la estimación de estos parámetros supone un importante y costoso proceso computacional, lo que deriva en la idea de la implementación de técnicas de paralización. Dichas técnicas brindan acceso a las tarjetas gráficas, las cuales están compuestas de múltiples nodos en los cuales se pueden ejecutar simultáneamente diferentes segmentos obtenidos tras la desmembración del

proceso computacional. Lo que permite reducir el coste de la estimación de parámetros a una ventana de tiempo factible para su ejecución en cada uno de los ficheros generados tras la captura de tráfico.

3 Análisis

3.1 . Introducción

En este capítulo se presenta la fase de análisis realizada para este trabajo, puesto que el resultado final depende de la ejecución de distintas fases se ha realizado un análisis característico de cada una y se han recogido una serie de requisitos funcionales.

Como parte fundamental de cualquier proceso, se ha estudiado cómo se definen cada una de sus partes. Estableciendo qué y cómo se ha de ejecutar en cada paso:

3.2 Tráfico de Red

Lo primero que se tuvo que concretar fue el tipo de dato que se iba a utilizar como forma de representación del tráfico de red.
Debido a que el entorno de pruebas estaba limitado al domicilio del autor del **TFG** el rango se limitó al tráfico de red que tenía como emisor y/o receptor el host del autor.

De se bajaron dos posibilidades: o bien trabajar sobre tráfico en tiempo real, o por el contrario, almacenar trazas de tráfico de red en fichero para su posterior procesamiento.
La opción de trabajar sobre tráfico en tiempo real acabó siendo descartada, como mencionan otros compañeros que también han trabajado con la librería *Libstable* aunque en una primera instancia se trató de que funcionara sobre tráfico real nunca se consiguió. Esto es debido a que su gran costo computacional a la hora de calcular la estimación de parámetros, esto provoca un retardo, que bien es cierto que con las técnicas de paralelización se ha conseguido reducir, lo que provocaría que el resultado analizado no fuese a la par que el tráfico recogido en ese instante.

Además se concluyó que, para poder generar estimaciones de los parámetros de las distribuciones estables en base a un conjunto de datos, éste debe tener una cantidad de muestras mínima y además con una variedad de valores considerable.

Partiendo de lo visto hasta ahora, se propuso que como modelo de tráfico de red se recogiese el retardo capturado por una instrucción básica, instrucción **PING**, la cual podía generarse cada segundo y lanzarla sobre diferentes direcciones.
También, genera la información necesaria para que, bien leída y filtrada, se tenga un registro de valores a lo largo de largos periodos tiempo en orden de segundos.

Al disponer de un solo entorno de prueba desde el cual elaborar este trabajo se ha decidido recoger respuestas de múltiples páginas webs para poder contrastarlos.

Se decidió apuntar las llamadas **PING** a varias de las páginas más visitadas en España, para ello se accedió a la información brindada por **Alexa** [12].

Las páginas escogidas son las siguientes:

- www.google.es
- www.google.com
- www.Youtube.com
- www.facebook.com
- www.Amazon.es
- www.Elpais.com
- www.Aliexpress.com
- www.milAnuncios.com
- www.rolloId.net
- https://tienda.wolterskluwer.es

Se generará un fichero de salida por cada página web, el cual plasmará todo resultado devuelto por la instrucción **PING** acorde a cada dominio.

3.3 Lector de datos

Dentro del proceso de captura se presenta la necesidad de filtrar y agrupar los datos recogidos en los ficheros generados como salida de las instrucciones **PING**.
Tras la elección de analizar el tráfico de red a través de los retardos el dato más importante a recoger es el campo **RTT** en un orden de milisegundos.
Además también se ha de recoger la fecha y la hora de cada respuesta **PING** recogida.

Una llama de **PING** regular tiene devuelve una respuesta como esta:

```
C:\Users\chr4istian>ping 8.8.8.8

Haciendo ping a 8.8.8.8 con 32 bytes de datos:
Respuesta desde 8.8.8.8: bytes=32 tiempo=11ms TTL=250
Respuesta desde 8.8.8.8: bytes=32 tiempo=14ms TTL=250
Respuesta desde 8.8.8.8: bytes=32 tiempo=7ms TTL=250
Respuesta desde 8.8.8.8: bytes=32 tiempo=11ms TTL=250
```

Figura 4. Ejemplo de llamada PING en WINDOWS.

Como se puede apreciar, dentro del fichero de salida que generaremos por cada entorno se recoge información no útil para nuestro proceso.
Por ello, es necesaria la implementación de una pequeña aplicación que lea y filtre este tipo de ficheros.
Dicha aplicación tendrá el nombre de *LectorTrazas*, y es, una aplicación de escritorio desarrollada en **C#** que a través de una consola, solicita al usuario que escoja el modo en que se trocearán los datos leídos del fichero cuyo directorio es insertado por el usuario.

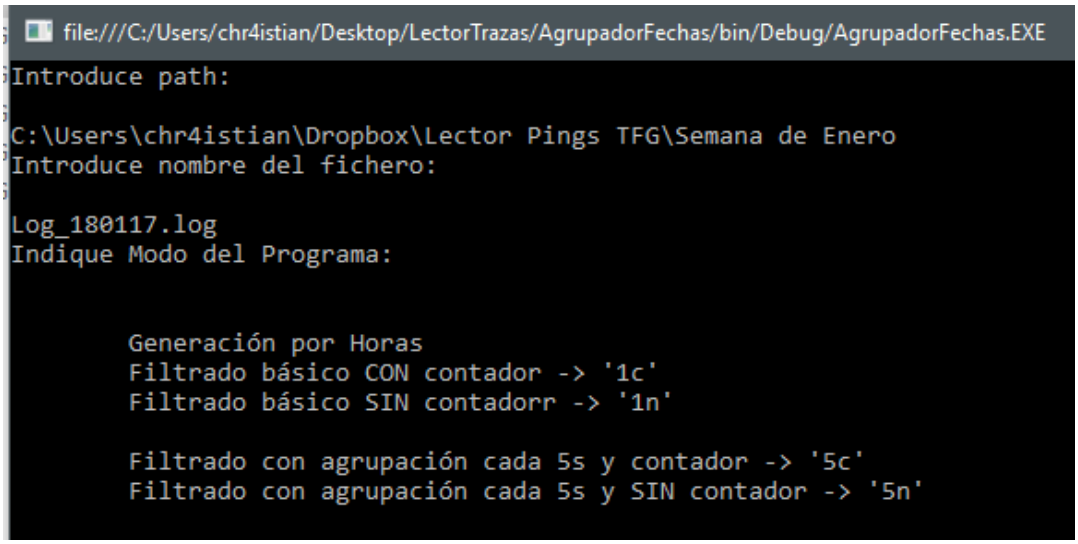


Figura 5. Ejemplo de uso de la aplicación LectorTrazas.

El programa *LectorTrazas* está diseñado para ejecutarse sobre ficheros formados por las respuestas literales obtenidas por la ejecución de la instrucción ping en una consola. Por lo cual, por defecto, filtra toda línea que no guarde información de retraso.

Existen dos formas de filtrar los retardos obtenidos:
El filtrado básico lee una por una las líneas correspondientes a la respuesta del servidor y almacena el valor del **RTT**.
El filtrado con agrupación suma los valores de los **RTT** de cada respuesta de servidor en un intervalo de 5 segundos.

Como mejora, se modificó para que generase un fichero por cada intervalo de una hora. Es decir, si el fichero a leer es el fichero generado tras la captura de la respuesta a un **PING** de 3 días a un dominio, la aplicación generará un directorio por cada día diferente que recoja al trocear el fichero, y dentro de ese repertorio almacenará un fichero por cada hora que contenga las respuestas generadas en ese intervalo. Por lo que el fichero anterior generaría 3 carpetas con 24 archivos diferentes cada una.

Los valores de timeout son considerados nulos y por ello descartados, si durante una hora no hay valores de retardos válidos (pudo haberse perdido la conexión) no habrá respuestas ligadas a esa hora y por lo tanto, no se generará el fichero de salida correspondiente a esa hora.

Los filtrados generados por las opciones con contador devuelven un fichero de dos columnas, en la que la primera columna es tan solo un contador iniciado en 1 que se va incrementando por cada fila **RTT** leído. La segunda columna es el valor del **RTT** leído expresado en milisegundos.
La versión sin contador tan solo posee la columna del valor de cada **RTT** leído.

3.4 Librería Libstable

Con el fin de trabajar con distribuciones α-estables se hará uso de la librería *Libstable.*
A diferencia de otras herramientas que son usadas para el cálculo numérico de la integral que define la FC de las α-estables, la herramienta *Libstable* devuelve resultados que alcanzan una precisión que si puede ser aceptada y sobre todo, ofrece una velocidad que la distingue de otras herramientas para su uso práctico.

Se trata de una librería C/C++ que permite una completa paralelización, rápida y precisa evaluación de las funciones de densidad, distribución y cuantil (PDF, CDF y CDF^{-1} respectivamente), la generación de variables aleatorias y la estimación de distribuciones α-estables en todo el espacio de parámetros.

Ha sido desarrollada por *Javier Royuela* del Val y *Federico Simmross Wattenberg* en el centro *Image Processing Laboratory* (**LPI**) y el código fuente se encuentra en: http://www.lpi.tel.uva.es/stable.

Pero para el desarrollo de este **TFG** se utilizará una versión modificada por *Guillermo Julián* disponible en:
https://github.com/hpcn-uam/libstable-opencl

La librería contiene todo lo necesario para trabajar con distribuciones α-estables pero en este trabajo tan solo recurre a la parte centrada en la estimación de parámetros.
Por ello, se modificará el fichero de ejemplo *fittest.c,* el cual realiza una estimación de los cuatro parámetros de un conjunto de datos generado aleatoriamente.
Esta versión modificada leerá los ficheros de salida generados por *LectorTrazas* y devolverá el ajuste de α, β, γ y δ para esos ficheros, que posteriormente será representado y analizado.

Las funciones que se utilizarán de *Libstable* son:

- *stable_fit_init()*

  Función que realiza una primera estimación de los parámetros en base al conjunto de datos y la longitud de estos. Los datos son lo que recibe como argumento, que en este caso, es la cadena con los datos leídos del fichero.
  Esta primera estimación es rápida, casi instantánea, pero muy poco precisa.

- *stable_activate_gpu()*

  Función que prepara y permite el uso de la **GPU**. Debe llamarse antes de comenzar a calcular.

- *stable_fit_grid()*

  Función que realiza la estimación de los parámetros tras una estimación previa. Esta parte es la que mas coste computacional tiene, el tiempo que tarda en ejecutarse varía según la longitud de datos que recibe como argumento. Duración promedia menos a un minuto por fichero generado por la captura tráfico de todo un día.

- *stable_free()*

  Libera la estructura de datos que almacena la información de los parámetros y libera la GPU.

3.5 Matlab

Como última herramienta involucrada en este proceso se encuentra Matlab.
Es una herramienta de software matemático muy popular. Ofrece un entorno de desarrollo integrado (**IDE**) con un lenguaje de programación propio.

Es un entorno de pago que se puede adquirir aquí:
https://es.mathworks.com/products/matlab.html

Ofrece una infinidad de usos pero en este proyecto se utilizará como herramienta de representación gráfica.
Con ella se representarán las gráficas a lo largo del tiempo y los histogramas de los datos empíricos, así como sus **CDF** y **CDF**.
También se representarán los distintos valores de las estimaciones de los parámetros en pos de obtener una vista gráfica de su evolución.

3.6 Test de bondad

Con el fin de validar las estimaciones generadas por la librería *Libstable*, y de demostrar que el trafico **IP** representado por retardos **PING** es adaptable a un modelo estadístico de distribuciones α-estables, se han investigado diversos métodos que prueben que cierto conjunto de datos encajan en una distribución dada. Generalmente estos métodos son recogidos bajo el nombre de *Goodness of Fit* [13]. Los test más comunes son:

- Chi-Cuadrado
- Kolmogorov-Smirnov
- Anderson-Darling.
- Shipiro-Wilk.

El test de bondad que se aplicará será el Kolmogorov-Smirnov (**KS**) para ello se ha extendido el fichero *fittest.c* con código que aplica dicho test a un conjunto de datos dada una estructura de datos que contenga las estimaciones de los parámetros de una distribución α-estable.

```c
double stable_kolmogorov_smirnov_gof(StableDist* dist, const double* samples, size_t nsamples) {
    double *cdf, *samples_sorted;
    double d;
    double result;
    int k = 0;

    cdf = calloc(nsamples, sizeof (double));
    samples_sorted = calloc(nsamples, sizeof (double));

    memcpy(samples_sorted, samples, nsamples * sizeof (double));
    gsl_sort(samples_sorted, 1, nsamples);

    if (dist->gpu_enabled)
        stable_cdf_gpu(dist, samples_sorted, nsamples, cdf, NULL);
    else
        stable_cdf(dist, samples_sorted, nsamples, cdf, NULL);

    result = kstest(samples_sorted, nsamples, cdf, &d);

    free(cdf);
    free(samples_sorted);

    return result;
}
```

Figura 6. Código con la implementación del KS en Libstable

```c
double probks(double alam) {
    // Source: Numerical recipes in C: The art of scientific computing.
    int j;
    double a2, fac = 2.0, sum = 0.0, term, termbf = 0.0;
    a2 = -2.0 * alam * alam;

    for (j = 1; j <= 100; j++) {
        term = fac * exp(a2 * j * j);
        sum += term;

        if (fabs(term) <= 0.001 * termbf || fabs(term) <= 1.0e-8 * sum) return sum;

        fac = -fac;
        termbf = fabs(term);
    }

    return 1.0;
}

double kstest(const double* samples, size_t nsamples, const double* cdf, double* d) {
    double max_diff = 0;

    for (size_t i = 0; i < nsamples; i++) {
        double ecdf = ((double) i) / nsamples;
        double diff = fabs(ecdf - cdf[i]);

        if (max_diff < diff)
            max_diff = diff;
    }

    if (d != NULL)
        *d = max_diff;

    double sqrt_n = sqrt(((double) nsamples));
    return probks((sqrt_n + 0.12 + 0.11 / sqrt_n) * max_diff);
}
```

Figura 7. Continuación del código con la implementación del KS en Libstable

El **K-S** Test mide la distancia absoluta entre la función de distribución Acumulada (**CDF**), generada con la estimación de los parámetros de la distribución α-estable, y la función de distribución empírica. Aceptado bajo un error indicado si los datos empíricos vienen o no de la distribución comparada.

3.7 Conclusiones

Este capítulo describe y presenta las distintas fases involucradas en el proceso a realizar en este análisis. Definiendo su utilidad, el estado y trabajo realizado sobre cada una, así como las herramientas que utiliza. También se ha presentado la necesidad de recurrir a una prueba de bondad que avale los resultados generados por la librería *Libstable*.
Al definir cómo están conectadas cada fase y las dependencias de cada una, se ha justificado el uso de cada elemento utilizado y grado de adaptación que han sufrido las distintas herramientas.

En el siguiente capítulo se entrará más en detalle en cada paso, describiendo los cambios realizados por el autor, mientras se realiza una simulación del proceso.

4 Desarrollo

4.1 Introducción

En esta sección se realizará un ciclo completo del conjunto de fases descritas para el análisis del tráfico **IP** usando un modelo estadístico de primer orden basado en distribuciones α-estables.

4.2 Captura de Tráfico

Como se ha explicado en secciones anteriores, el tráfico **IP** será representado por los retardos recogidos tras la ejecución del comando **PING** a distintos dominios web.
Para estandarizar el proceso, se han creado un pequeño programa ***.bat*** para cada uno de los dominios que ejecuta el comando de forma indefinida y almacena su salida en un fichero.

```
@echo off

set host=google.com
set logfile=Log_GoogleCom.log

echo Target Host = %host% >%logfile%
for /f "tokens=*" %%A in ('ping %host% -n 1 ') do (echo %%A>>%logfile% && GOTO Ping)
:Ping
for /f "tokens=* skip=2" %%A in ('ping %host% -n 1 ') do (
    echo %date% %time:~0,2%:%time:~3,2%:%time:~6,2% %%A>>%logfile%
    echo %date% %time:~0,2%:%time:~3,2%:%time:~6,2% %%A
    timeout 1 >NUL
    GOTO Ping)
```

Figura 8. Ejemplo de fichero .bat contra el dominio google.com

Debido a que el comando **PING** otorga una salida distinta en cada sistema operativo, y que los ficheros de extensión .bat sólo son compatibles con WINDOWS, se ha definido también un estándar de llamada **PING** para los sistemas operativos basados en LINUX, que aportará una salida que *LectorTrazas* es capaz de traducir.

```
ping google.com | xargs -n1 -i bash -c 'echo `date +\ %T`" {}"' >> google.com
```

Figura 9. Ejemplo de comando PING en sistemas operativos basados en LINUX.

El proceso de captura de tráfico deberá ejecutarse sin interrupción durante al menos una semana para obtener una cantidad suficiente de tráfico representativo. En este trabajo se ha conseguido capturar tráfico durante dos semanas enteras. Siendo la primera semana la correspondiente a la semana del 24 – 30 de Abril, y la segunda correspondiente a la semana del 22 – 28 de Mayo.

Se ha evitado capturar tráfico en periodos de tiempo no representativos como semanas con días festivos, también se han seleccionado dos semanas de carácter similar siendo ambas correspondientes a la cuarta semana del mes.
Una vez capturado el tráfico deseado, se pararán las ejecuciones, lo que generará un fichero por cada dominio web analizado. Serán esos ficheros los que utilicemos en la siguiente fase.

4.3 Filtro y Traducción

La finalidad de esta fase no es otra que preparar la información para la estimación realizada por la librearía *Libstable*.
Este proceso es totalmente necesario ya que dicha librería necesita recibir una lista de valores de distribución. Se podría modificar la captura del tráfico producido por la instrucción **PING** para que tan solo imprimiese el valor del retardo, pero de ese modo se perdería el contexto del tiempo al que pertenecen esos valores.

Por ello, una de los requisitos de esta fase es el agrupamiento del tráfico capturado considerando ventanas de una hora en un periodo temporal de 2 semanas. Lo cual nos deja un total de 336 ventanas con las que contrastar.

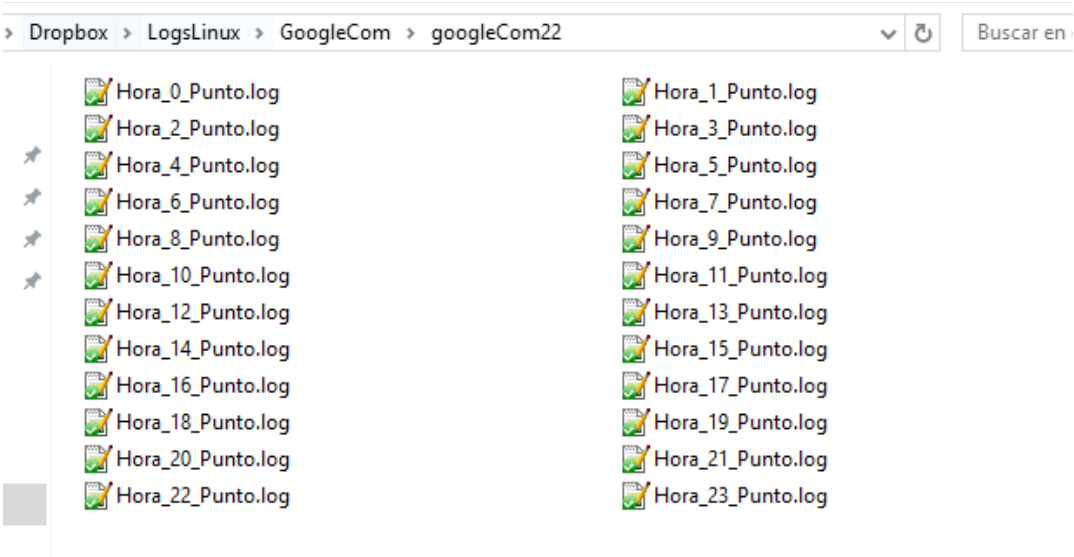


Figura 10. Ejemplo de directorio creado por Lector Trazas correspondiente al tráfico red de un día completo contra el dominio Google.com

Durante pues que no todas las instrucciones de **PING** realizadas (una cada segundo) devolvían un valor, se ha optado por descartar esos valores. Lo que provoca que no todos los ficheros correspondientes a una hora de tráfico tengan una longitud de 3600 valores.

Se lanzará el ejecutable *LectorTrazas* que solicitará al usuario un modo de uso.
En este proceso se ha seleccionado el modo *"Filtrado básico sin contador"* introduciendo el modo "*1n*". A continuación el programa solicitará al usuario que indique la ruta donde se encuentra el fichero a filtrar, para luego después preguntar por el nombre de éste.

Generará un directorio por cada día diferente leído en las instrucciones capturadas que contendrá un fichero por cada hora leída.

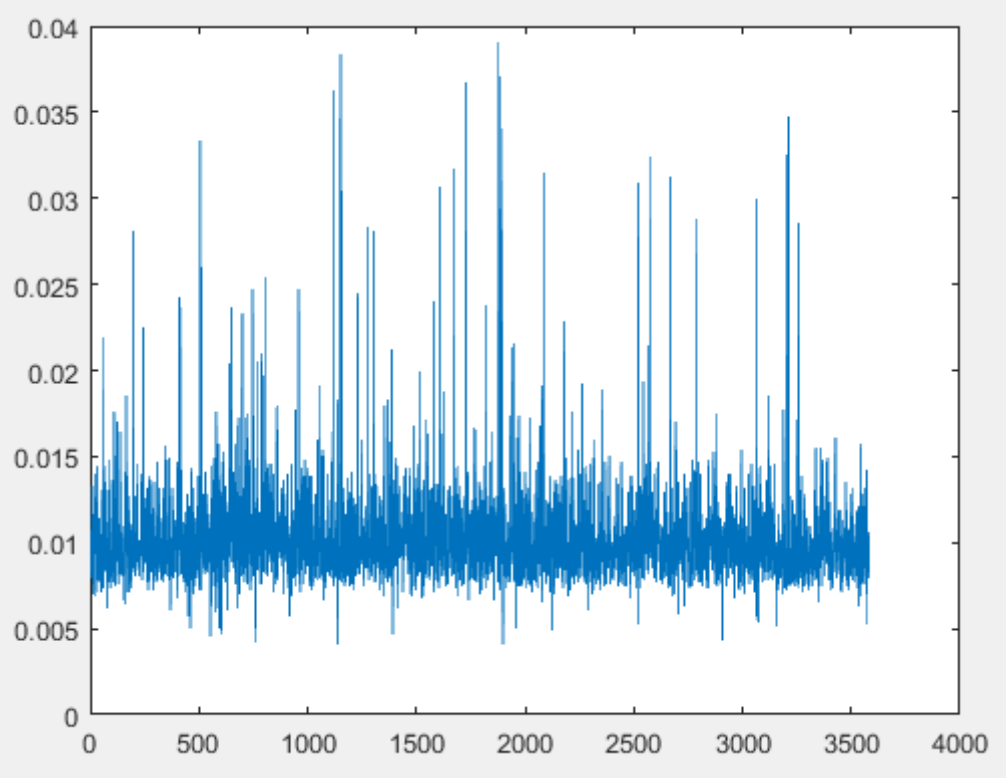


Figura 11. Representación de los valores de retardo de la instrucción PING equivalentes a una hora de tiempo.

4.4 Estimación de parámetros

Utilizando la librería Libstable, se generarán los valores de los parámetros de las distribuciones α-estables estimados correspondientes a cada conjunto de datos recogido.
Se ha modificado la función test encargada de generar estimaciones de parámetros dado un conjunto de datos de la librería original, para que sea capaz de leer los datos generados por la herramienta *LectorTrazas.*

Para poder usar la librería se ha de añadir las cabeceras y los compilados ya generados a la ruta del compilador **C/C++** y ejecutar el comando make.

Una vez compilado, se añadirán los directorios generados por LectorTrazas al proyecto, y se ejecutará el archivo fittest.c utilizando como primer parámetro la ruta del fichero que contiene los datos a estimar, y numero de datos que contiene como segundo parámetro.

El programa realizará dos estimaciones de parámetros, una primera 'rápida' dada por la ejecución de la función:
*void stable_fit_init(StableDist * dist, const double * data, const unsigned int N, double * nu_c,double * nu_z)*

Que sirve para inicializar los parámetros almacenados en la estructura *dist* con unos valores estimados aproximados. Esta estimación posee una precisión muy baja por lo que

tan solo se realiza como inicialización y paso previo a una estimación más precisa y costosa.

Para de eso se encarga la función:

*int stable_fit_grid(StableDist * dist, const double * data, const unsigned int N)*

Que requiere de la previa activación de la GPU mediante la llamada a la función:

short stable_activate_gpu(StableDist dist)*

Que según una precisión y número de iteraciones establecidos en las macros: WANTED_PRECISION y MAX_ITERATIONS respectivamente, devolverá un ajuste más preciso de los parámetros sobre la muestra de datos leída.

```
Building final target: release/fittest
christian@christian-HP:~/libstable-opencl-master$ bin/release/fitte
to.log 3582
Leyendo el fichero Hora_1_Punto.log...
Contador final: 3583
---------------------------- Primera estimacion -------------------
Alpha = 1.767551
Beta  = 1.000000
Sigma = 0.001378
Mu    = 0.011006

-------------------------------------------------------------------
---------------------------- Resultados ---------------------------
Alpha = 1.768336
Beta  = 1.000000
Sigma = 0.001374
Mu    = 0.011395

-------------------------------------------------------------------
christian@christian-HP:~/libstable-opencl-master$
```

Figura 12. Ejemplo de uso de *Libstable* para la estimación de parámetros dado un conjunto de datos

Se realizará esta estimación para cada uno de los ficheros generados por *LectorTrazas*. De esta forma se obtendrá el valor de cada parámetro estimado para cada hora, de cada día de cada uno de los dominios web analizados.

A continuación se muestra una tabla con los resultados correspondientes a la ejecución del proceso descrito hasta ahora sobre un solo dominio durante un intervalo de 1 semana:

Hora / Parámetros	Lunes 22			
	Alfa	Beta	Gamma	Delta
0	1,519992	1	0,001166	0,010608
1	1,629391	1	0,00123	0,01049
2	1,718029	1	0,001278	0,010496
3	1,821995	1	0,001324	0,010397
4	1,914588	1	0,001326	0,01024
5	1,850089	1	0,001231	0,009962
6	1,925615	1	0,00127	0,010019
7	1,887522	1	0,001282	0,01017
8	1,753421	1	0,001193	0,010181
9	1,626856	1	0,001221	0,01041
10	1,729658	1	0,001198	0,010057
11	1,39184	1	0,001098	0,010854
12	1,484798	1	0,00104	0,010384
13	1,567054	1	0,001064	0,01036
14	1,758254	1	0,001267	0,010182
15	1,831055	1	0,00121	0,010164
16	1,647323	1	0,001133	0,010197
17	1,638803	1	0,001094	0,010066
18	1,560138	1	0,001037	0,010296
19	1,578609	1	0,001068	0,010265
20	1,562908	1	0,001074	0,010361
21	1,5572	1	0,000991	0,010313
22	1,644544	1	0,001061	0,010256
23	1,626127	0,992832	0,001117	0,010352

Hora / Parámetros	Martes 23			
	Alfa	Beta	Gamma	Delta
0	1,60906	1	0,001044	0,010282
1	1,227361	0,924078	0,001098	0,012041
2	1,493975	0,887171	0,001162	0,010603
3	1,953087	1	0,001218	0,009999
4	1,790627	1	0,001065	0,009937
5	1,700894	1	0,001015	0,010023
6	1,885151	1	0,001253	0,01002
7	1,739664	1	0,001128	0,010083
8	1,808328	1	0,001168	0,009995
9	1,691324	1	0,001155	0,01019
10	1,638782	1	0,001102	0,010337
11	1,65215	1	0,001145	0,010137
12	1,633687	1	0,001162	0,010163
13	1,720725	1	0,001158	0,010071
14	1,57651	1	0,001046	0,010311
15	1,550233	1	0,001083	0,010326
16	1,548029	1	0,001001	0,010423
17	1,52896	1	0,000996	0,010483
18	1,562985	0,993357	0,001112	0,010279
19	1,594644	1	0,001138	0,010356
20	1,566477	0,992397	0,001153	0,010426
21	1,658178	1	0,001157	0,010307
22	1,619483	1	0,001139	0,010325
23	1,581794	1	0,001076	0,010497

Hora / Parámetros	Miércoles 24			
	Alfa	Beta	Gamma	Delta
0	1,568084	1	0,001077	0,010247
1	1,721861	1	0,001057	0,009891
2	1,548492	1	0,001041	0,0102
3	1,58304	1	0,000998	0,010159
4	1,579544	1	0,000964	0,010115
5	1,698781	1	0,001121	0,010091
6	1,701787	1	0,001087	0,009841
7	1,695391	1	0,001117	0,010163
8	1,820826	1	0,00124	0,010007
9	1,731412	1	0,001153	0,010139
10	1,842242	1	0,001109	0,009936
11	1,710772	1	0,001028	0,009907
12	1,61256	1	0,001127	0,010128
13	1,498101	1	0,001067	0,010318
14	1,647979	1	0,001011	0,010034
15	1,539184	1	0,000906	0,009885
16	1,730373	0,791422	0,00078	0,009159
17	1,739664	1	0,001128	0,010083
18	1,750134	0,879719	0,000711	0,009012
19	1,772463	1	0,000971	0,00974
20	1,664386	1	0,000993	0,010085
21	1,563763	1	0,000999	0,01029
22	1,499186	0,975336	0,000984	0,010313
23	1,598369	1	0,000937	0,010042

Hora / Parámetros	Jueves 25			
	Alfa	Beta	Gamma	Delta
0	1,612341	1	0,001006	0,010058
1	1,570124	1	0,000923	0,009981
2	1,667161	1	0,001044	0,010049
3	1,711377	1	0,000987	0,010033
4	1,719727	1	0,001023	0,010069
5	1,663818	1	0,00084	0,009576
6	1,698297	1	0,00082	0,009494
7	1,589745	1	0,000852	0,009771
8	1,619157	1	0,00091	0,010116
9	1,683199	1	0,000908	0,009899
10	1,661769	1	0,000884	0,009673
11	1,63686	1	0,00092	0,010046
12	1,600359	1	0,000903	0,009857
13	1,600689	1	0,000918	0,009872
14	1,603353	1	0,000932	0,009915
15	1,646555	1	0,000856	0,009815
16	1,577804	1	0,000907	0,01
17	1,562887	0,977938	0,000914	0,010056
18	1,6109	1	0,000915	0,010016
19	1,626094	1	0,000859	0,009853
20	1,556888	1	0,001024	0,010426
21	1,522853	1	0,000971	0,010425
22	1,621024	1	0,000972	0,010186
23	1,549183	1	0,000975	0,010365

Hora / Parámetros	Viernes 26			
	Alfa	Beta	Gamma	Delta
0	1,636024	1	0,000859	0,009916
1	1,581513	1	0,000886	0,00997
2	1,65059	1	0,000907	0,009896
3	1,632066	1	0,000847	0,009594
4	1,743601	1	0,000841	0,009535
5	1,893834	1	0,001076	0,009772
6	1,800292	1	0,000962	0,00979
7	1,856447	1	0,001063	0,009788
8	1,786614	1	0,00097	0,009828
9	1,63466	1	0,000867	0,009689
10	1,591761	1	0,000905	0,010088
11	1,546387	1	0,000896	0,010223
12	1,653647	1	0,000832	0,0096
13	1,682498	1	0,000937	0,0099
14	1,60716	1	0,000831	0,009817
15	1,629407	1	0,000907	0,009908
16	1,586583	1	0,000899	0,009931
17	1,4793	0,927567	0,001002	0,010386
18	1,590099	1	0,000884	0,009837
19	1,668832	1	0,000878	0,009765
20	1,659894	1	0,000943	0,009994
21	1,674839	1	0,000913	0,009871
22	1,592451	1	0,000889	0,010001
23	1,676111	1	0,000926	0,009992

Hora / Parámetros	Sábado 27			
	Alfa	Beta	Gamma	Delta
0	1,610517	1	0,000897	0,009915
1	1,64095	1	0,000909	0,009853
2	1,667246	1	0,00091	0,009902
3	1,635223	1	0,000864	0,009648
4	1,701823	1	0,000985	0,009989
5	1,788711	1	0,001051	0,009909
6	1,661913	1	0,000903	0,009677
7	1,936834	1	0,001096	0,009702
8	1,715728	1	0,00089	0,009637
9	1,60739	1	0,000903	0,009835
10	1,616714	1	0,000875	0,009793
11	1,306385	1	0,000963	0,01102
12	1,11594	-1	0,001026	-0,04637
13	1,596526	1	0,000895	0,010069
14	1,626022	1	0,000868	0,010004
15	1,625178	1	0,000906	0,010084
16	1,643432	1	0,000895	0,010029
17	1,603256	1	0,000892	0,009957
18	1,680829	1	0,000854	0,009837
19	1,597596	1	0,000889	0,010028
20	1,625859	1	0,000913	0,009886
21	1,560728	1	0,000865	0,009986
22	1,288141	0,913466	0,000931	0,01094
23	1,586094	1	0,000896	0,010127

Domingo 28				
Hora / Parámetros	**Alfa**	**Beta**	**Gamma**	**Delta**
0	1,592972	0,981946	0,000817	0,009902
1	1,635215	1	0,000859	0,009744
2	1,792954	1	0,001016	0,009887
3	1,707875	1	0,000937	0,009941
4	1,664703	1	0,000856	0,009615
5	1,70878	1	0,000941	0,009676
6	1,894362	1	0,001108	0,009735
7	1,710201	1	0,000931	0,009889
8	1,635519	1	0,000859	0,009619
9	1,560083	1	0,00085	0,009737
10	1,678943	1	0,000867	0,009701
11	1,581589	1	0,000891	0,009918
12	1,612996	1	0,000906	0,009986
13	1,612115	1	0,000887	0,009862
14	1,570411	1	0,000902	0,009939
15	1,580807	1	0,000873	0,010003
16	1,58838	1	0,000881	0,009886
17	1,672415	1	0,000931	0,010054
18	1,724019	1	0,000994	0,010171
19	1,616633	1	0,000942	0,010173
20	1,593724	1	0,00092	0,010213
21	1,521552	0,881247	0,000806	0,010096
22	1,544166	1	0,000938	0,010313
23	1,605193	1	0,001047	0,010418

Tabla 1. Parámetros estimados para Google.com durante la semana del 22 al 28 de Mayo.

4.5 Test de bondad

Para confirmar si el tráfico de red capturado se adaptaba el modelo estadístico basado en distribuciones α-estables era necesario detectar si los valores de los parámetros estimados se ajustaban de forma aproximada a los datos.
Como primer filtro se ha representado y comparado la funcione de distribución acumulada empírica (obtenida a partir de los datos reales) y la función de distribución acumulada estimada (obtenida a partir de los parámetros generados por *Libstable*).

Para ello se ha integrado la librería Libstable en el entorno de Matlab y se han calculado dichas funciones utilizando los métodos de cálculo de **CDF**:

*void stable_cdf(StableDist *dist, const double x[], const int Nx, double *cdf, double *err)*

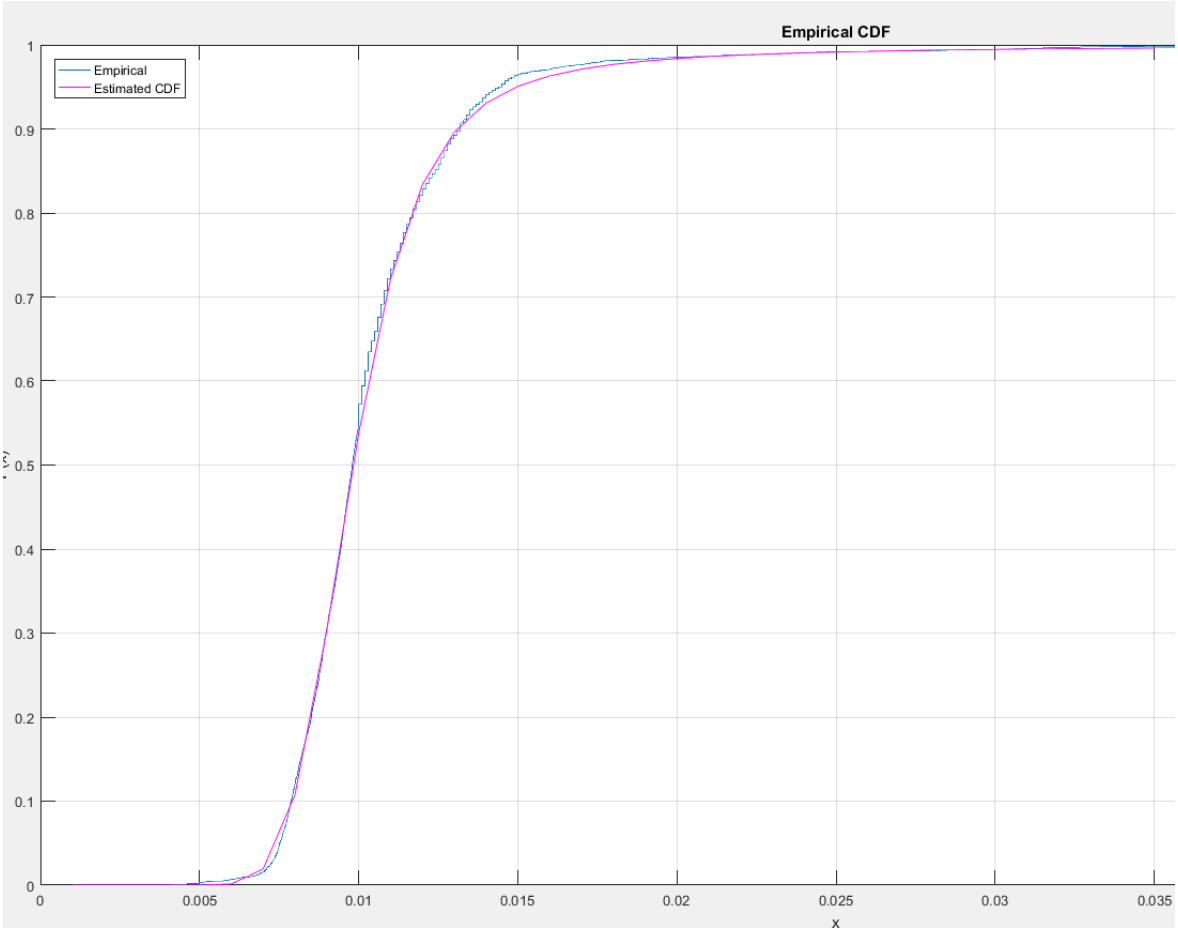


Figura 13. Comparación entra las CDF empíricas y estimadas.

Se puede apreciar, a ojo, la alta precisión de la estimación.

Otra forma de representar el ajuste es mediante el uso de la **PDF** estimada respecto al histograma de probabilidad de los datos empíricos

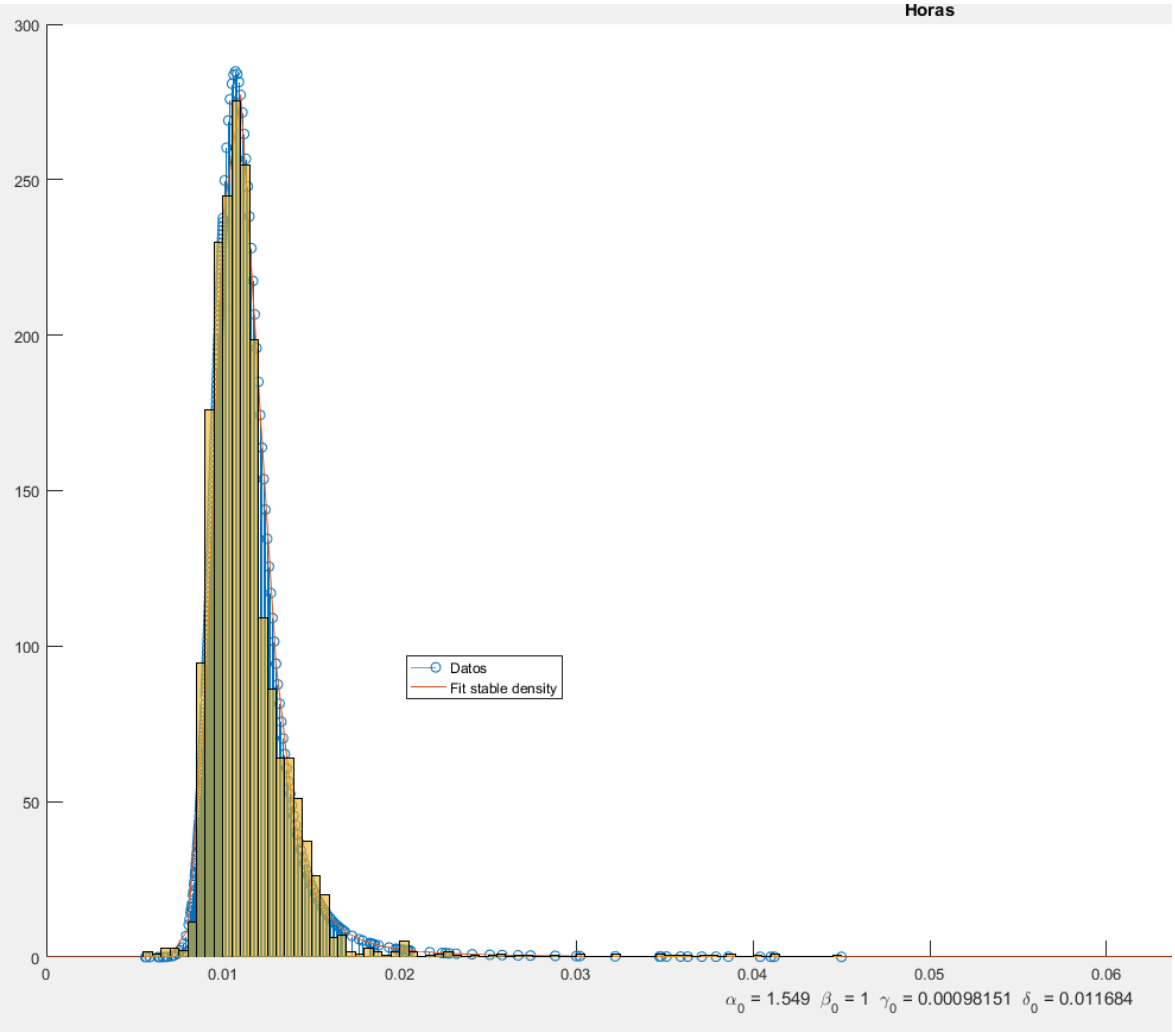


Figura 14. Representación de la PDF estimada respecto al histograma de probabilidad de los datos empíricos

En este documento se detallan solo dos ejemplos pero se ha comprobado el ajuste con la mayoría de las estimaciones generadas en este proceso y todas se ajustaban con el mismo nivel de precisión.

Pero aunque visualmente se pudiese decir que los datos pertenecen a la distribución estimada, se piensa recurrir a pruebas basadas en el test de bondad de ajuste de Kolmogorov-Smirnov, el cual se encarga de calcular la diferencia máxima absoluta entre las **CDF**.

Se realizará recurriendo la función integrada en *fittest.c:*

double stable_kolmogorov_smirnov_gof(StableDist dist, const double* samples, size_t nsamples)*

La cual, dada una aproximación de los cuatro valores y un conjunto de datos. Devuelve un p-valor entre 0 y 1 que es la probabilidad de que dada esa distancia, los datos provengan de esa distribución, cuanto más cercano a uno mejor ajuste.

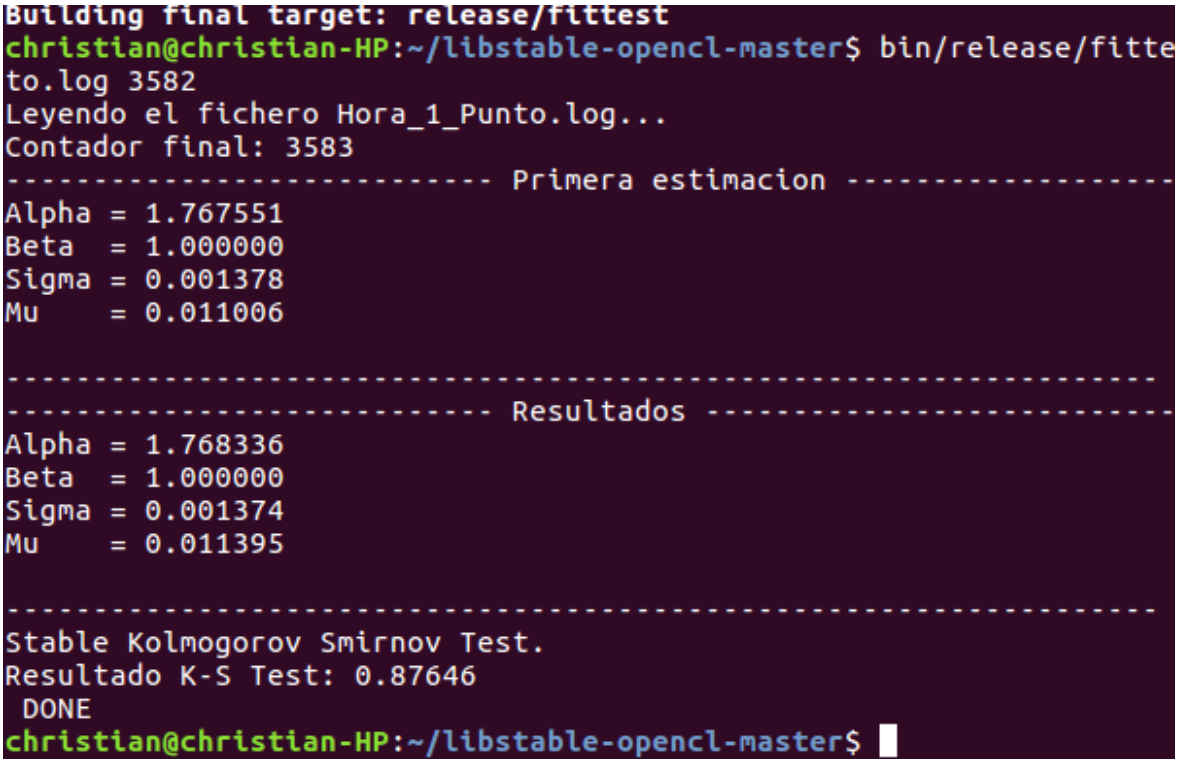


Figura 15. Resultado de ejecución de *fittest.c* modificado con la implementación del test de bondad de ajuste de Kolmogorov-Smirnov

Como se puede observar en la figura 15 el resultado del test es un valor bastante cercano a 1. Por cuestiones de tiempo, no se ha ejecutado dicho test en la totalidad de los datos que se han estimado. Pero sí se ha realizado en al menos 1 fichero correspondiente a una hora de cada día y en todos se ha obtenido un valor de ajuste > 0.7, por lo que se puede asumir la superación del test de bondad de ajuste en todo el proceso.

4.6 Representación.

Tras la estimación de los parámetros y su validación tras superar la validación del test de bondad, se precisa representar los distintos valores que obtienen los parámetros de la distribución a lo largo del intervalo de tiempo analizado.

Se ha optado por representar dichos parámetros utilizando la herramienta de Matlab.

Generando distintas representaciones de los parámetros estimados se pueden observar patrones o fenómenos que ocurren en la red.
En este punto se es consciente que, pese a que puede parecer que se han recogido una cantidad aceptable de datos, con la información capturada resultante tras dos semanas no se puede sacar ningún comportamiento cien por cien representativo de la red. Si bien se distinguen comportamientos que pueden encajar con los ya comúnmente conocidos, se precisaría de muchos más recursos para poder llegar a sacar conclusiones generales. Recursos como un mayor intervalo de estudio, de orden de meses. Captura desde distintos entornos de prueba, ya que el tráfico varía notablemente según desde donde se éste monitorizando.

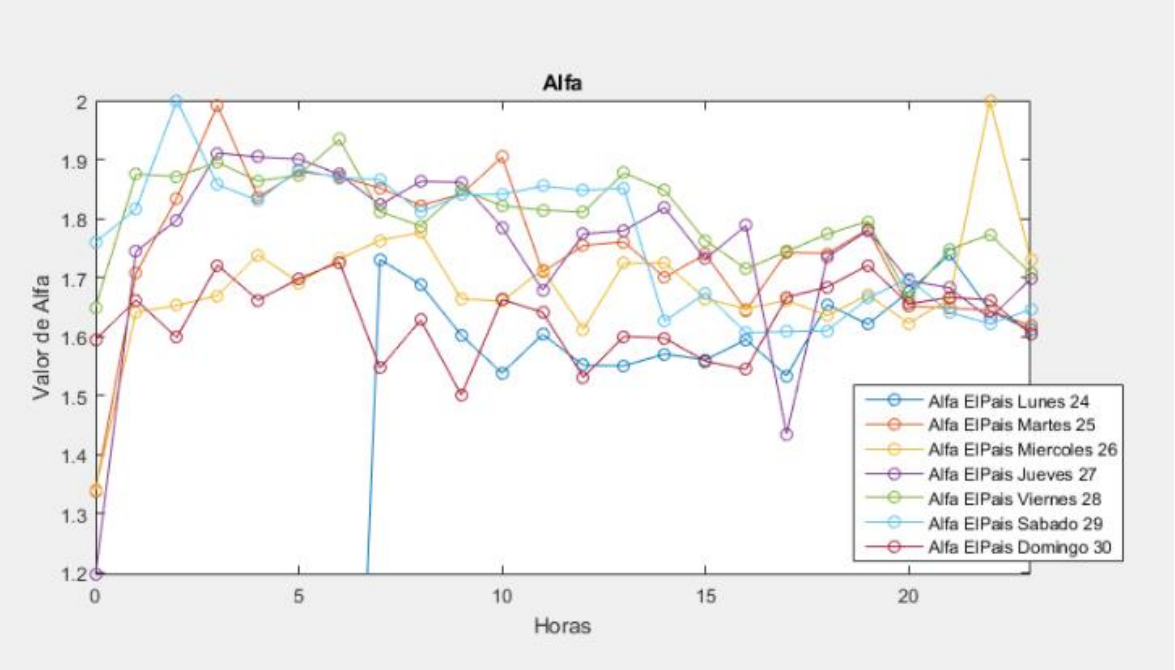


Figura 16. Representación de los distintos valores de Alfa a lo largo del día según el día para un conjunto de datos obtenidos al analizar los retardos enviados por una respuesta PING contra *elPais.com*

4.7 Conclusiones.

En este capítulo se ha detallado paso a paso cada fase que describe este proceso de análisis el cual se basa este **TFG.**
Se recorrido cada una de las fases simulando un proceso real describiendo los detallen mas técnicos y explicando el funcionamiento y el objetivo de cada parte.
También se ha intentado justificar la veracidad del ajuste de la estimación obtenida tanto visual como numéricamente utilizando el test de bondad de ajuste de Kolmogorov-Smirnov.

Por último, se ha descrito como se estudiarán los resultados, a través de gráficas representativas. En el siguiente capítulo se definirán los criterios tomados a la hora de escoger los datos para generar las gráficas y los resultados analizados a partir de éstas.
También se listarán los problemas encontrados durante el desarrollo de este análisis.

5 Resultados y Problemas.

5.1 Introducción

En el quinto capítulo se abordan las dificultades encontradas a lo largo de la ejecución de este proceso de análisis, junto a los resultados obtenidos que describen el comportamiento de la red. Empezar citando los distintos problemas que han dictaminado en cierta parte los resultados conseguidos. De entre las múltiples complicaciones encontradas merecen especial mención.

5.2 Problemas

5.2.1 Integración librería *Libstable*

Dicha librería es una herramienta para manejar distribuciones α-estables que funciona tanto es sistemas WINDOWS como LINUX. Pero que depende en gran medida de cierta configuración del sistema como la previa instalación de la librería **GSL**, la librería Pthreads y disponer de la librería y cabeceras OpenCL.
Fueron estas últimas las que más retrasaron la consecución del proyecto ya que, aun instaladas por el controlador de la tarjeta gráfica existía un "bug**"** que citando textualmente:

"Some AMD drivers fail when including header files in OpenCL code. An easy workaround is to copy the included headers to the _/tmp_ folder, where AMD does the temporary copies when compiling code. Running _cp -r includes /tmp_ should suffice to work around this driver bug." [13]

5.2.2 Valores de retardos recogidos en las respuestas PING

Otro problema que se sorteo es la obtención de valores demasiado pequeños en las respuestas obtenidas en las llamadas **PING** realizadas desde ciertos puntos. El retardo no variaba de 1 milisegundo, que es el valor mínimo que recoge el **PING** de WINDOWS, el 99% de las ejecuciones del comando **PING**. Lo cual dejaba un conjunto de datos discreto que no era posible adaptar a una distribución α-estables que es una distribución continua. Por ello se descartó la captura de retardos desde un segundo entorno de pruebas que disponía el autor de este **TFG**.

5.2.3 Granularidad de los datos capturados

Incluso cuando los valores estaban por encima de 1 milisegundo, existían dominios cuya respuesta oscilaba siempre entre valores < 5 milisegundos lo cual provocaba que el conjunto de datos resultante también fuese un conjunto de datos discreto.

5.2.4 Mixturas

Quizás es el problema más recurrente e importante.
Ciertos valores recogidos presentaban lo que pasaremos a denominar una Mixtura. Aparece cuando el dato recogido presenta la forma de dos o más funciones de distribución estable.

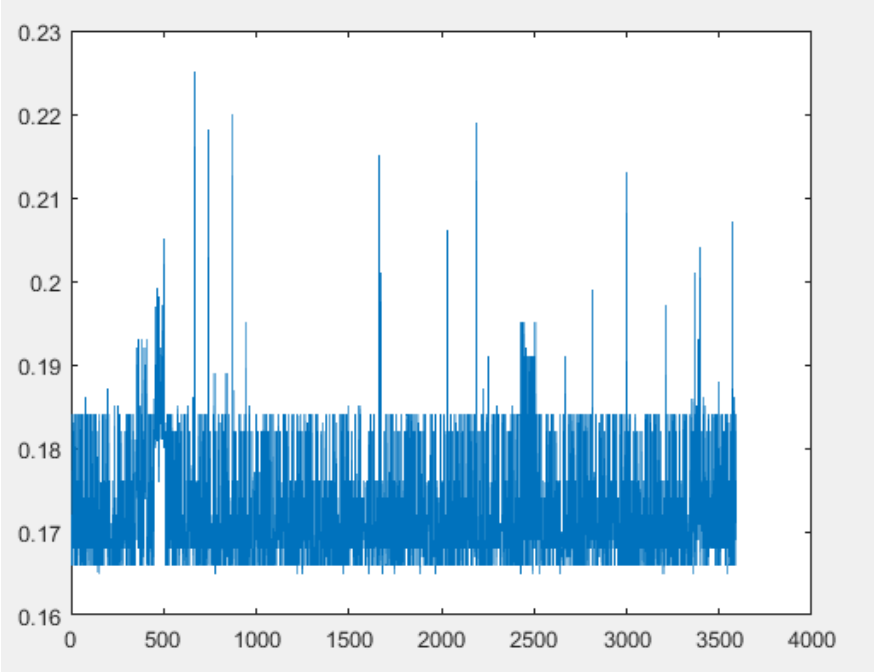


Figura 17. Ejemplo de tráfico capturado con mixtura

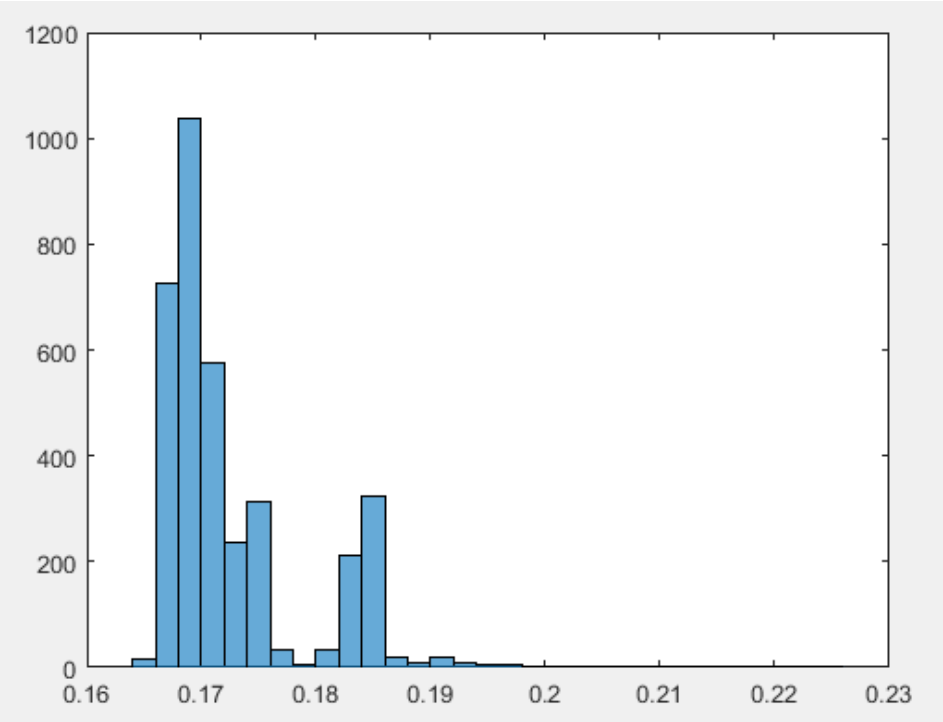


Figura 18. Ejemplo de tráfico capturado con mixtura

Como se puede observar, las figuras 17 y 18 representan un ejemplo claro de Mixtura. Siendo el valor de *X = 18* el punto de separación entre ambas. Esto describe que durante esa hora, hubo dos tipos de respuesta que manejaban tiempos distintos. Una primera distribución que oscila entre los valores *0.16 - 0.18* y una segunda que oscila entre *0.18 – 0.2*.
Existen infinidad de agentes influyentes en tráfico de red pero podemos formular algunas hipótesis de por qué ocurre este fenómeno si el comando que se ha lanzado no ha variado durante esa hora.

Una posibilidad pudiese ser un aumento repentino del tráfico de red que provocase un aumento en el valor de retardo del **PING**, por lo que llamadas posteriores a ese incremento del uso de red devolviesen valores de retardo más elevados.
Pero observando la figura 17 se comprueba que los distintos valores se van alternando continuamente. Excepto en el instante *X = 450 -> X = 510*, donde vemos que sí se forma una especia de ola la cual aumenta el valor de retardo tanto de los valores pertenecientes a la distribución 1 como a los de la distribución 2.

La explicación más lógica es dada por el comportamiento del tráfico red. Generalmente existe más de un ruta para llegar a un destino, la ruta escogida por defecto para la invocación y respuesta del comando **PING** suele ser la más rápida, pero en caso de congestión, pueden escogerse otras rutas, comportamiento que explicaría perfectamente lo sucedido.

El problema viene cuando no es algo puntual, el 100% de los datos capturados presentan síntomas de mixturas. Esto se debe a que el entorno de pruebas que está recogiendo el tráfico dispone de más de una salida a internet, algo muy común en universidades o empresas.

Esto es algo con lo que el autor de este **TFG** se ha encontrado, se ha realizado este proceso de análisis desde dos entornos de prueba simultáneamente, recogiendo tráfico de red desde dos puntos diferentes apuntado a los mismo dominios en las llamadas **PING**.
El primer entorno es el domicilio del autor, no presentaba mixturas y es el que se ha presentado en el capítulo 4 de este **TFG**. El segundo entorno se correspondía al lugar de trabajo del autor, una empresa. Este segundo entorno capturaba datos con mixturas el 100% de las veces, por lo que los datos no pudieron ser tratados.

La librería *Libstable* no está configurada para tratar mixturas por lo que trata los datos como si pertenecieran a la misma distribución.

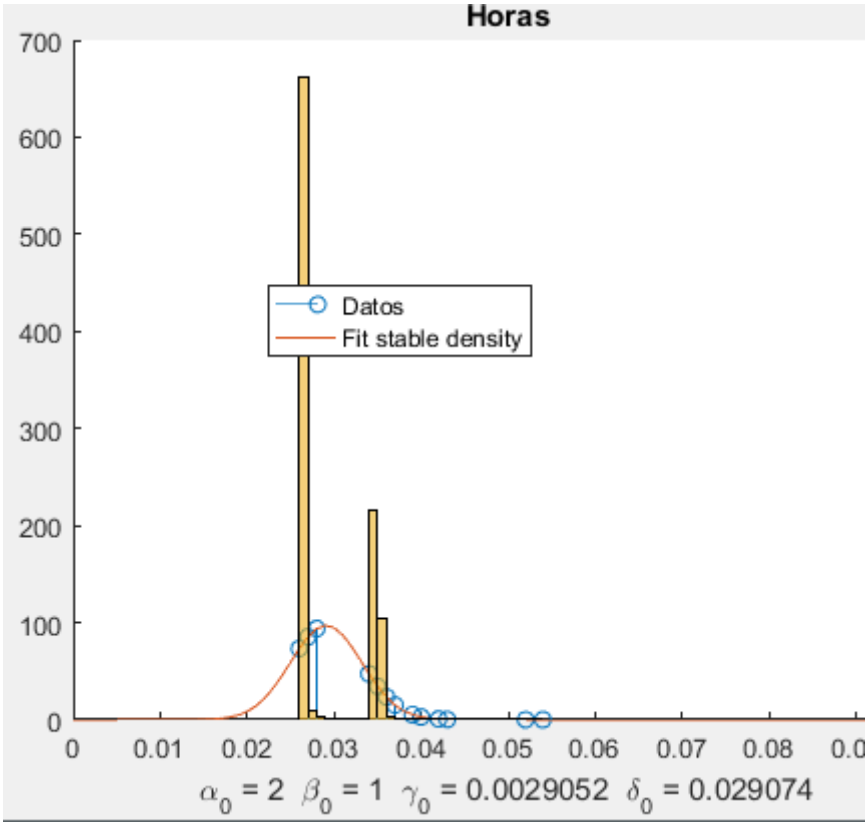


Figura 19. Demostración de fallo de estimación frente a conjunto de datos con mixturas

5.3 Resultados

En el Anexo B se adjuntarán algunas de las gráficas generadas con la representación de los parámetros estimados.

Como se ha mencionado antes, los valores de α y β son denominados parámetros de forma, ya que son estos los que definen que forma va a tener la distribución.

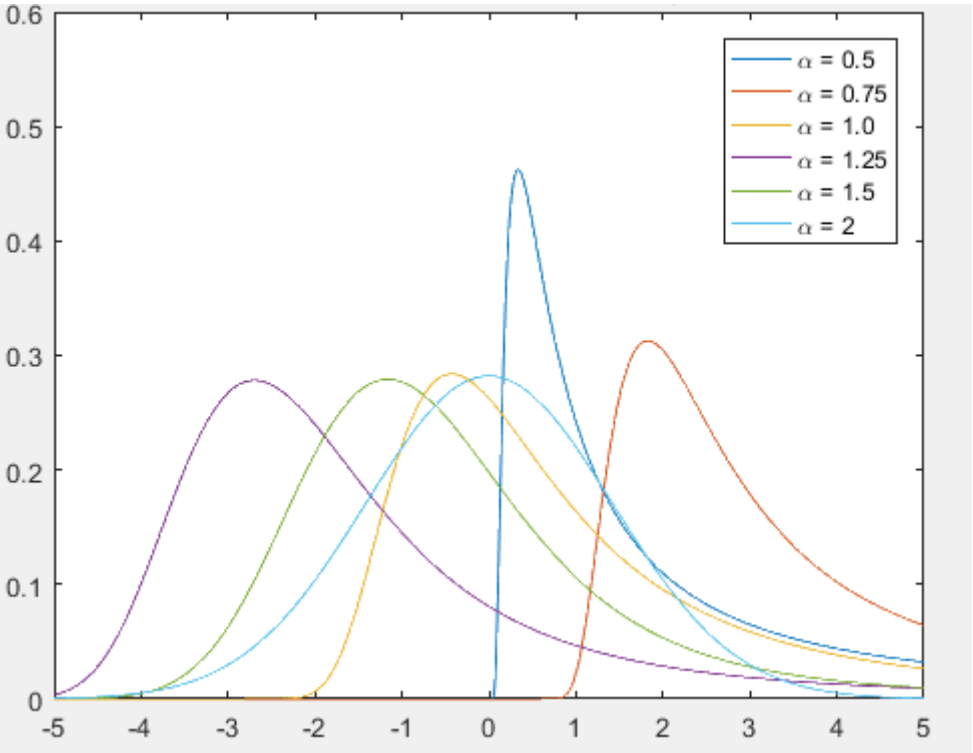


Figura 20. Representación de varianza PDF en función del parámetro α para un β = 1

La figura 20 representa como va cambiando la forma que obtiene la **PDF** según un valor fijo para β = 1 (que es el valor que obtiene un 99% del tráfico capturado) y un valor variable para α.
Con esto se puede deducir que cuanto menor sea el valor de α, mayor será concentración de los datos analizados en un intervalo pequeño. Conforme α vaya incrementándose se puede apreciar que los datos se van distribuyendo hasta el punto de adaptar un forma simétrica en α = 2, donde se considera una distribución Gaussiana.

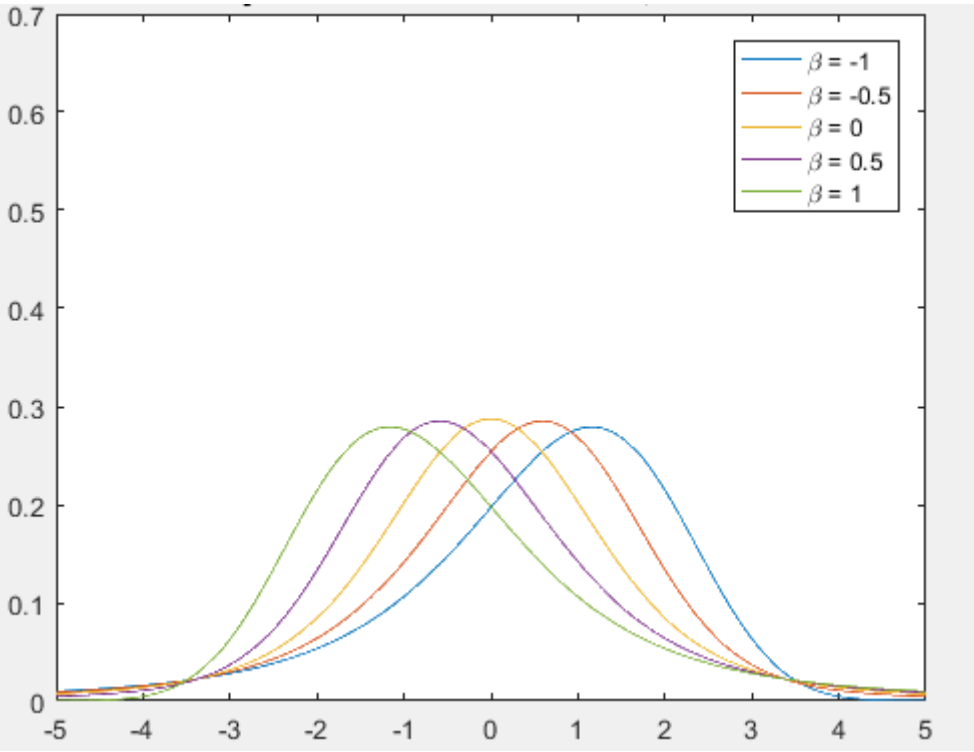


Figura 21. Representación de la variación de la PDF en función de β para un valor constante α = 1.5

En la figura 21 se puede observar el impacto de β en la PDF para un valor constante de α = 1.5 (que es un valor común en las estimaciones del tráfico que se ha capturado). Se aprecia que valores de β por debajo de 0 inclinan la gráfica hacia la derecha y valores de β mayores que 0 definen una gráfica inclinada a la izquierda. El valor de β = 0 hace que la gráfica sea totalmente simétrica.

Desde el punto de tráfico de red, la inclinación supone el grado de retardo sufrido. Las muestras capturadas marcan, en su gran mayoría, un valor de β = 1. Lo que permite decir que los valores más pequeños, correspondientes a los RTT más rápidos, son los valores promedio y que existen valores definidos por RTT más lentos que forman la cola de la distribución.

Con respecto a los valores de α estimados, se puede apreciar que todos los valores se encuentran, en su mayoría, en un intervalo de [1,5 – 2].
Los valores comprendidos entre esos valores se corresponden a un comportamiento que definiremos como “normal”.

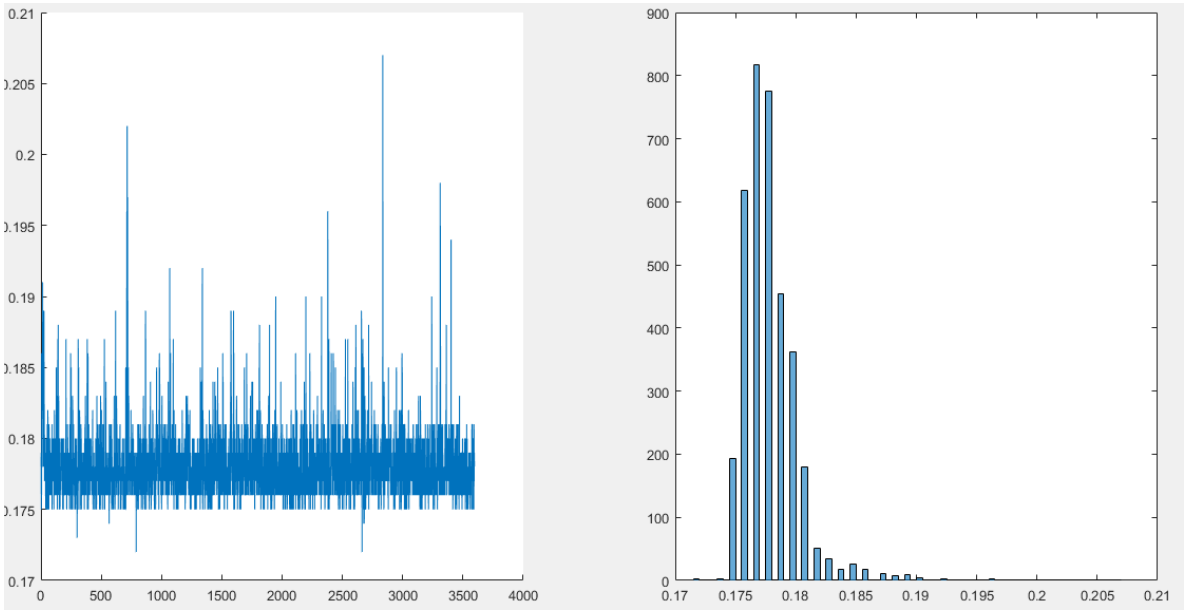


Figura 22. Ejemplo de comportamiento "normal" de la red.

En cambio podemos observar un comportamiento anómalo en valores $\alpha < 1.5$.
Que normalmente van asociados a mixturas o asociado a retardos muy largos.

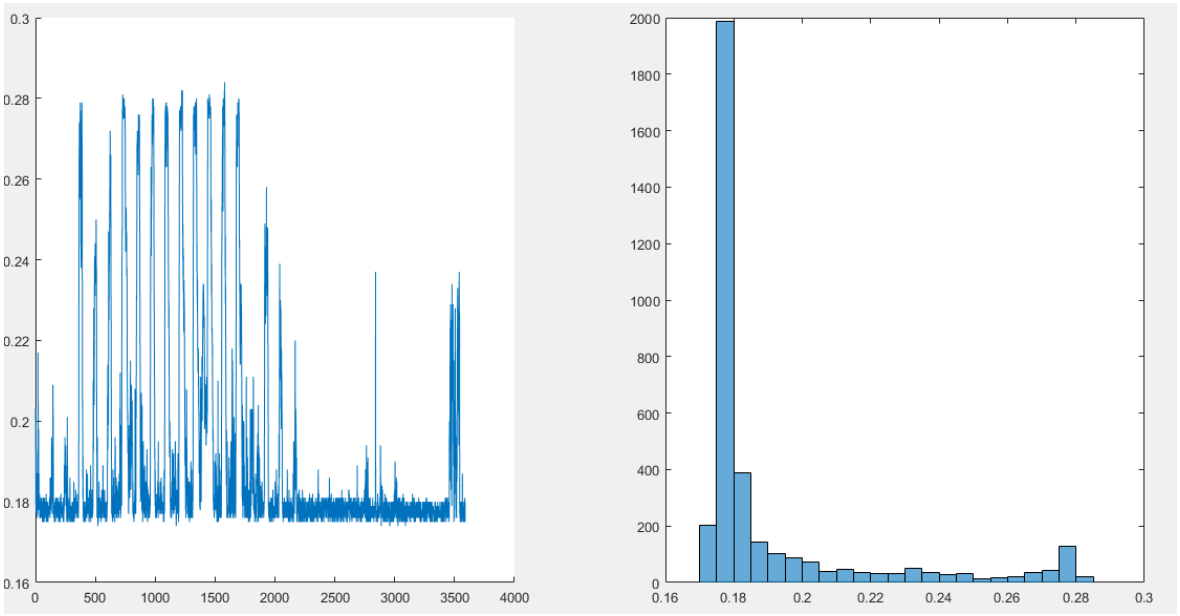


Figura 23. Ejemplo de captura de tráfico congestionado.

La figura 23 es un ejemplo de tráfico con congestión que produce un valore de $\alpha = 0{,}750974$**.**
Con esta forma de ver el tráfico de red podemos fácilmente detectar flujos anómalos, al menos los correspondientes a aumento de tráfico y cambios en la ruta realizada por los paquetes **IP**.

Se ha obtenido el valor de la mediana de todos los valores de alfa de cada uno de los websites (Anexo B) en pos a sacar una conclusión del comportamiento general de los retardos por horas a lo largo de una semana.

Partiendo que para poder describir el comportamiento de la red se precisa de más registros, al menos los equivalentes a tres meses, pero podemos percibir que el valor de alfa tiende a elevarse hasta alcanzar su punto máximo sobre las 3:00 – 5:00 de la mañana, alcanzar el punto mínimo sobre las 12:00 para luego incrementar levemente hasta el final del día.

Un incremento del valor de α describe un tráfico que tiende a estabilizarse, devolviendo un valor de RTT concentrado, eso se puede traducir como un tráfico poco congestionado. Algo lógico considerando que son horas de baja actividad.

Por ello, que el valor mínimo de α sea en sobre las 12:00 puede ser justificado como intervalo de tiempo en el que hay un mayor tráfico de red debido a ser una hora laboral que no coincide con comidas o descansos y común a varios sectores laborales.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

En este apartado se exponen las conclusiones del proyecto desde el punto de vista de análisis del trabajo realizado.
Se ha conseguido definir un proceso capaz de capturar tráfico red para después adaptarlo a un modelo estadístico de primer orden basado en distribuciones α-estables.
Dicho proceso permite obtener una visión del tráfico de red desde el punto de vista de las α-estables y demostrar que el tráfico de red no siempre encaja como una distribución Gaussiana, sino que las distribuciones α-estables son más precisas a la hora de modelar dicho tráfico.

Para conseguirlo, se han descrito una serie de fases o pasos cuya consecución nutre al proceso de unos resultados que consumirá el paso siguiente, creado un ciclo que parte desde la captura del tráfico a modelar hasta la representación de la estimación realizada.

El trabajo no hubiese sido posible sin el conocimiento adquirido durante la carrera.
En particular el conocimiento adquirido en las asignaturas *Redes I y Redes II* he sido necesario para entender la arquitectura del tráfico IP y ser capaz de entender los resultados obtenidos.
También destacar la asignatura *Arquitectura de Sistema Paralelos* la cual nos ha permitido entender y manipular la librería *Libstable* que implementa un código basado en OpenCL.

Conocimientos generales sobre programación han sido vitales para la automatización y desarrollo de aplicaciones como *LectorTrazas* basadas en un lenguaje orientado a objetos. Dichos conocimientos han sido adquiridos especialmente en las asignaturas de *Programación I y II.*

Una tarea que al principio, bien parecía abstracta para el autor, ha ido cogiendo forma conforme avanzada la propia investigación y aprendizaje. Que deja un sabor agridulce al ser consciente de cuán poco se ha representado y analizado ahora que se ha adquirido un conocimiento considerable de las distribuciones α-estables y de su capacidad de adaptación al tráfico red.
Que contrasta con la tremenda satisfacción personal obtenida tras haber sido capaz de comprender y manejar un concepto desconocido como eran las distribuciones α-estables, junto al aprendizaje de herramientas como Matlab y *Libstable* cuyo conocimiento previo era nulo.

6.2 Trabajo futuro

En los capítulos anteriores se ha conseguido llegar a analizar el tráfico de red utilizando distribuciones α-estables, pero como cualquier trabajo de investigación existen múltiples mejoras posibles, o reenfoques para utilizarlo como base para otros proyectos.

El autor de este trabajo partía desde lo que se podía considerar el desconocimiento de las distribuciones estables, por lo que una gran parte de este proyecto se ha enfocado en la presentación de estas, su manipulación y adaptación a un tráfico real. Desde este punto, se pueden plantear varias mejoras o nuevas implantaciones.

6.2.1 Mejoras

- Algoritmos de tratamiento de mixturas.
  De cara a la resolución del problema más significativo, las mixturas, se propone implementar o acoplar funcionalidad que permita tratar los datos que presenten este tipo de anomalía. Una propuesta sería un programa capaz de localizar los mínimos y máximos locales de un conjunto de datos y ser capaz de generar conjuntos de datos diferentes a partir de uno que presente una mixtura.

- Adaptación de la librería *Libstable* a **C#**.
  Actualmente, el lenguaje de programación **C#** es un lenguaje cuyo uso está en auge. También posee mayor versatilidad y potencia que el lenguaje **C/C++** que utiliza la librería *Libstable*. Además, ya que el programa generado en este proyecto, *LectorTrazas*, está implementado en **C#** sería posible aunar todo el proceso de captura y estimación en una misma librería.

6.2.2 Trabajos Futuros.

- Utilizar distintos tipos de representación de tráfico red.
  Estimar parámetros α-estables utilizando los retardos enviados por las respuestas de instrucciones **PING** es un buen paso como punto de partida. Pero la red ofrece infinidad de información que puede ser capturada y adaptada para poder ser analizada posteriormente. Lo que otorgaría otro enfoque del comportamiento de la red, que ayudará a establecer patrones y detectar anomalías.

- Aumentar los recursos.
  En este trabajo se ha contado con recursos limitados, pero se han creado las herramientas y dictado las fases necesarias para nuevas implementaciones sean cómodas y sencillas. Tan solo habría que realizar la fase de captura desde múltiples entornos de prueba, algo tan sencillo como ejecutar los ficheros .**BAT** creados.
  También aumentar la duración de los intervalos de tiempo a analizar, aumentado el rango de dos semanas a meses.

- Implementación de un detector de anomalías.
  Visto que el comportamiento anómalo de la red influye en gran medida en los valores de los parámetros de las distribuciones α-estables, es posible crear un sistema de detección de anomalías que se base en estos valores para captar y describir un suceso: aumento de tráfico, cambio de ruta, caídas…

Referencias

[1] Descripción del Proyecto de fin de Grado propuesto por: **Luis de Pedro Sánchez** Análisis de Tráfico en Internet Utilizando Distribuciones Alfa estables en procesadores paralelos

[2] En estadística, la prueba de Kolmogórov-Smirnov es una prueba no paramétrica que determina la bondad de ajuste de dos distribuciones de probabilidad entre sí. https://es.wikipedia.org/wiki/Prueba_de_Kolmog%C3%B3rov-Smirnov

[3] J. Royuela-del-Val, F. Simmross-Wattenberg, C. Alberola-López, "Libstable: Fast, parallel and high-precision computation of α-stable distributions in R, C/C++ and MATLAB" Journal of Statistical Software (in press).

[4] Laboratorio de procesado de Imagen.
https://www.lpi.tel.uva.es/node/610

[5] Transformada de Fourier.
http://www4.ujaen.es/~jmalmira/transformada_fourier_almira.pdf

[6] John P. Nolan (1997), "Numerical calculation of stable densities and distribution functions". Communications in Statistics. Stochastic Models.

[7] Nolan JP (1999). "An Algorithm for Evaluating Stable Densities in Zolotarev's (M) Parameterization." Mathematical and Computer Modelling, 29(10–12), 229–233. doi: 10.1016/s0895-7177(99)00105-3.

[9] Blaise Barney, Lawrence Livermore National Laboratoy. "Introduction to Parallel Computing"
https://computing.llnl.gov/tutorials/parallel_comp

[10] OpenCL official website,
https://www.khronos.org/opencl/

[11] CUDA Zone,
https://developer.nvidia.com/cuda-zone

[12] Alexa Internet, Inc. Keyword Research, Competitive Analysis, & Website Ranking
http://www.alexa.com/

[13] PennState Eberly College of Science. STAT 504 |Analysis of Discrete Data. 2.4 - Godness of Fit.
https://onlinecourses.science.psu.edu/stat504/node/60

Anexos

Anexo A:

Fittest.c Modificado

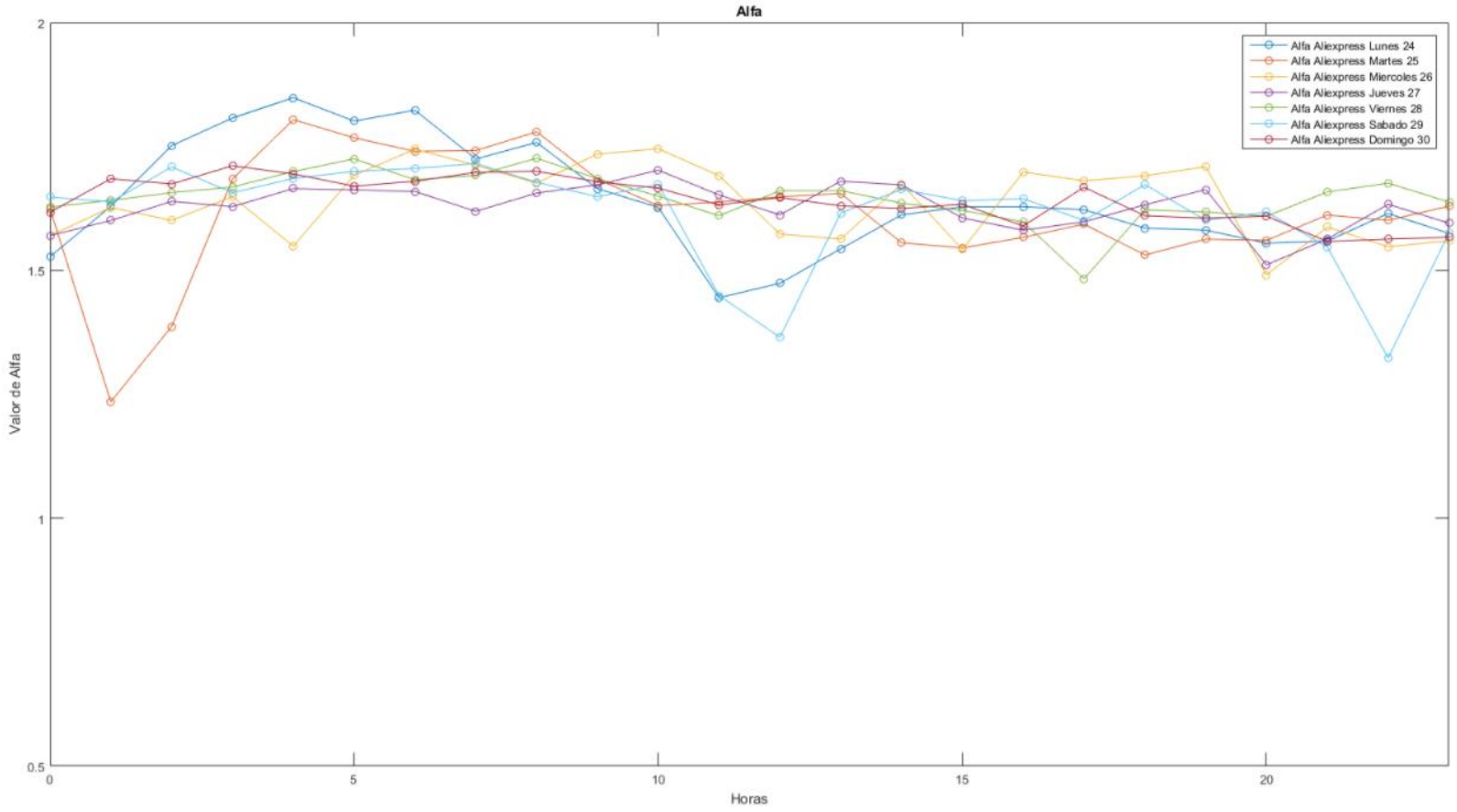
```
int main(int argc, char* argv[]) {

    char *aux;
    double alfa, beta, sigma, mu;
    double ma = 0, mb = 0, ms = 0, mm = 0;
    double *data;
    double singleData, testResult;
    int i = 1, k = 0, N;
    FILE *ficheroDatos;
    char buffer[100]; // Buffer de 2 Kbytes

    StableDist *dist = NULL;
    alfa = 0.1;
    beta = -0.9;
    sigma = 1.0;
    mu = 0.0;

       //Recibir por argumento la ruta del fichero y su longitud.
    if (argc > 1) {
        ficheroDatos = fopen(argv[1], "r");
        N = atoi(argv[2]);
        data = (double*) malloc(N * sizeof (double));

    } else {
        ficheroDatos = fopen("Hora_0_Punto.log", "r");
        N = 3576;
        data = (double*) malloc(N * sizeof (double));
    }

    data = (double*) malloc(N * sizeof (double));


    if ((dist = stable_create(alfa, beta, sigma, mu, 0)) == NULL) {
        printf("Error when creating the distribution");
        exit(1);
    }

    stable_set_THREADS(1);
    stable_set_absTOL(1e-16);
    stable_set_relTOL(1e-8);
    stable_set_FLOG("errlog.txt");

    if (ficheroDatos == NULL) {
        printf("El fichero %s no existe o no puede ser abierto.\n", argv[1]);
        return 0;
    }
    printf("Leyendo el fichero %s...\n", argv[1]);

    while (k <= N) {
        fgets(buffer, 100, ficheroDatos);
        singleData = atof(buffer);
        data[k] = singleData;
        k++;
    }
    printf("Contador final: %d\n", k);
    fclose(ficheroDatos);
```

```
    stable_fit_init(dist, data, N, NULL, NULL);

    ma = dist->alfa;
    mb = dist->beta;
    ms = dist->sigma;
    mm = dist->mu_0;

    printf("---------------------------- Primera Estimación ----------------------------
---\n");
    printf("Alpha = %f\n", ma);
    printf("Beta  = %f\n", mb);
    printf("Sigma = %f\n", ms);
    printf("Mu    = %f\n", mm);

    stable_activate_gpu(dist);

     stable_fit_grid(dist, data, N);

    ma = dist->alfa;
    mb = dist->beta;
    ms = dist->sigma;
    mm = dist->mu_0;

    printf("---------------------------- Resultados ------------------------------\n");
    printf("Alpha = %f\n", ma);
    printf("Beta  = %f\n", mb);
    printf("Sigma = %f\n", ms);
    printf("Mu    = %f\n", mm);
    printf("\n----------------------------------------------------------------------\n");

       printf("Prueba de Bondad. Stable Kolmogorov Smirnov Test.\n");
    //testResult = stable_kolmogorov_smirnov_gof(dist, data, (size_t) N);

    printf("Resultado K-S Test: %f", testResult);

    printf("DONE\n");

    free(data);
    stable_free(dist);

    fclose(stable_get_FLOG());

    return 0;
}
```

Anexo B:

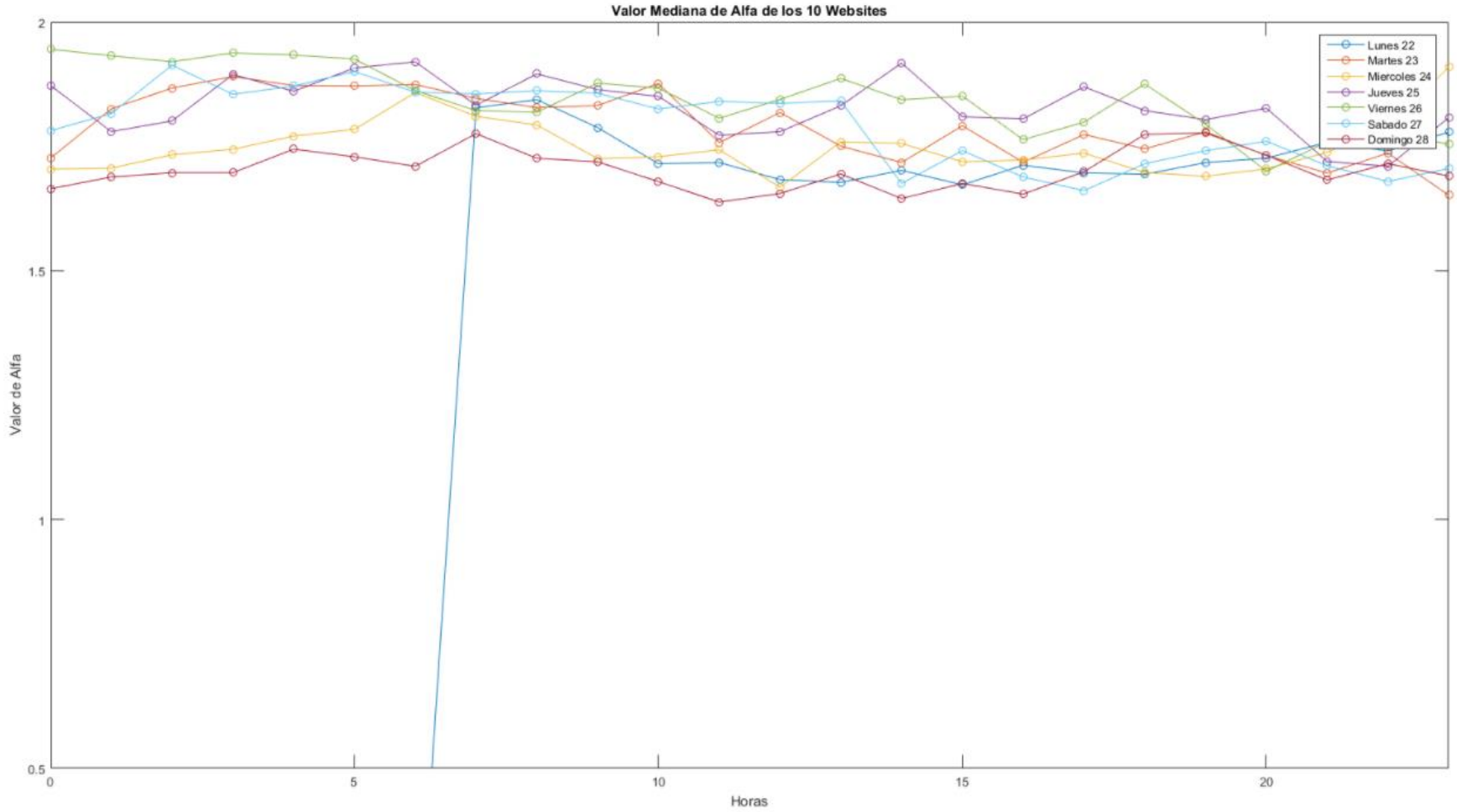
Gráficas Parámetros estables

Valor de la mediana resultante de la agrupación de las estimaciones de Alfa de los 10 websites. Mayo 24 – 30

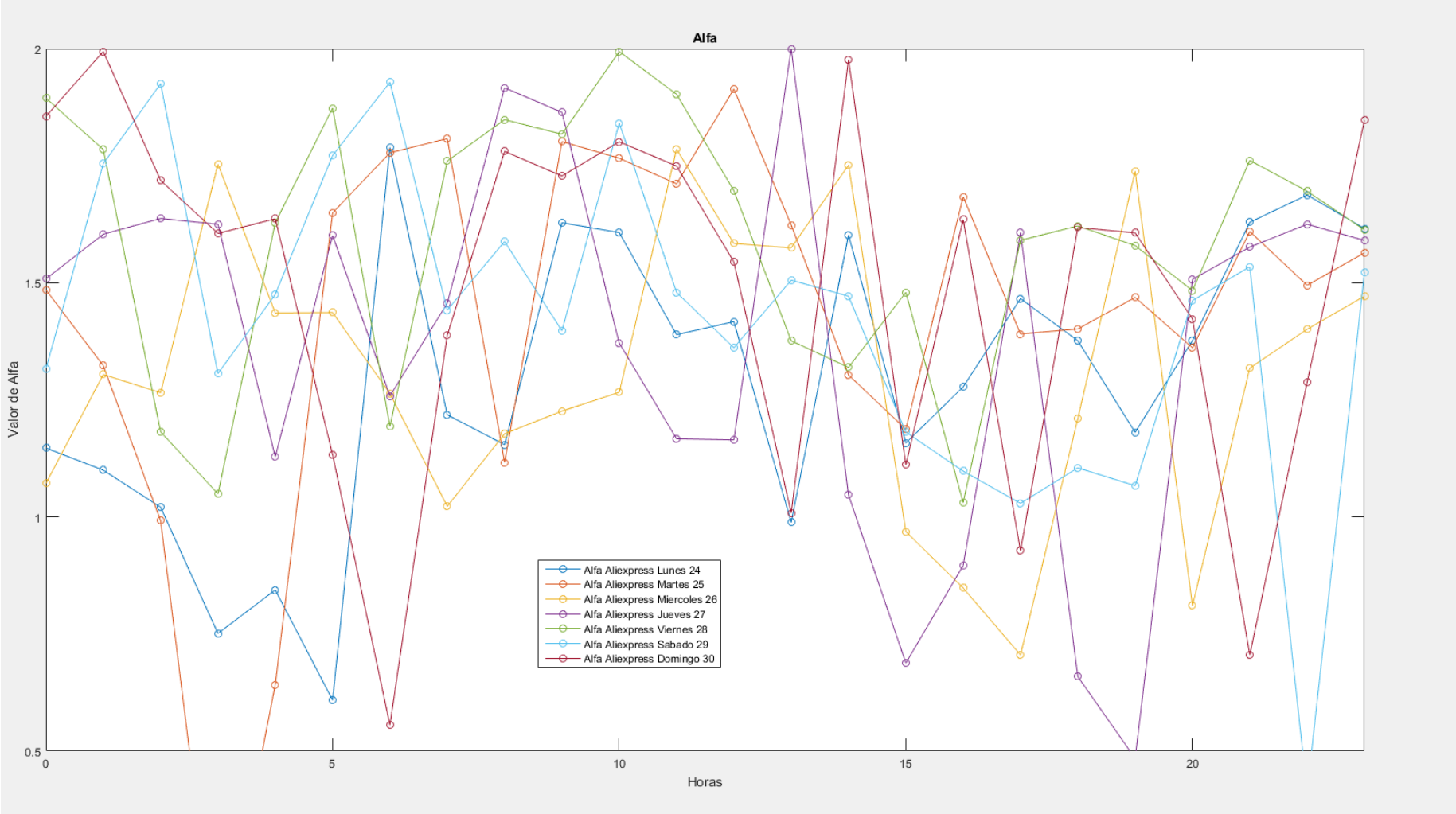


Valor de la mediana resultante de la agrupación de las estimaciones de Alfa de los 10 websites. Abril del 22 – 28.

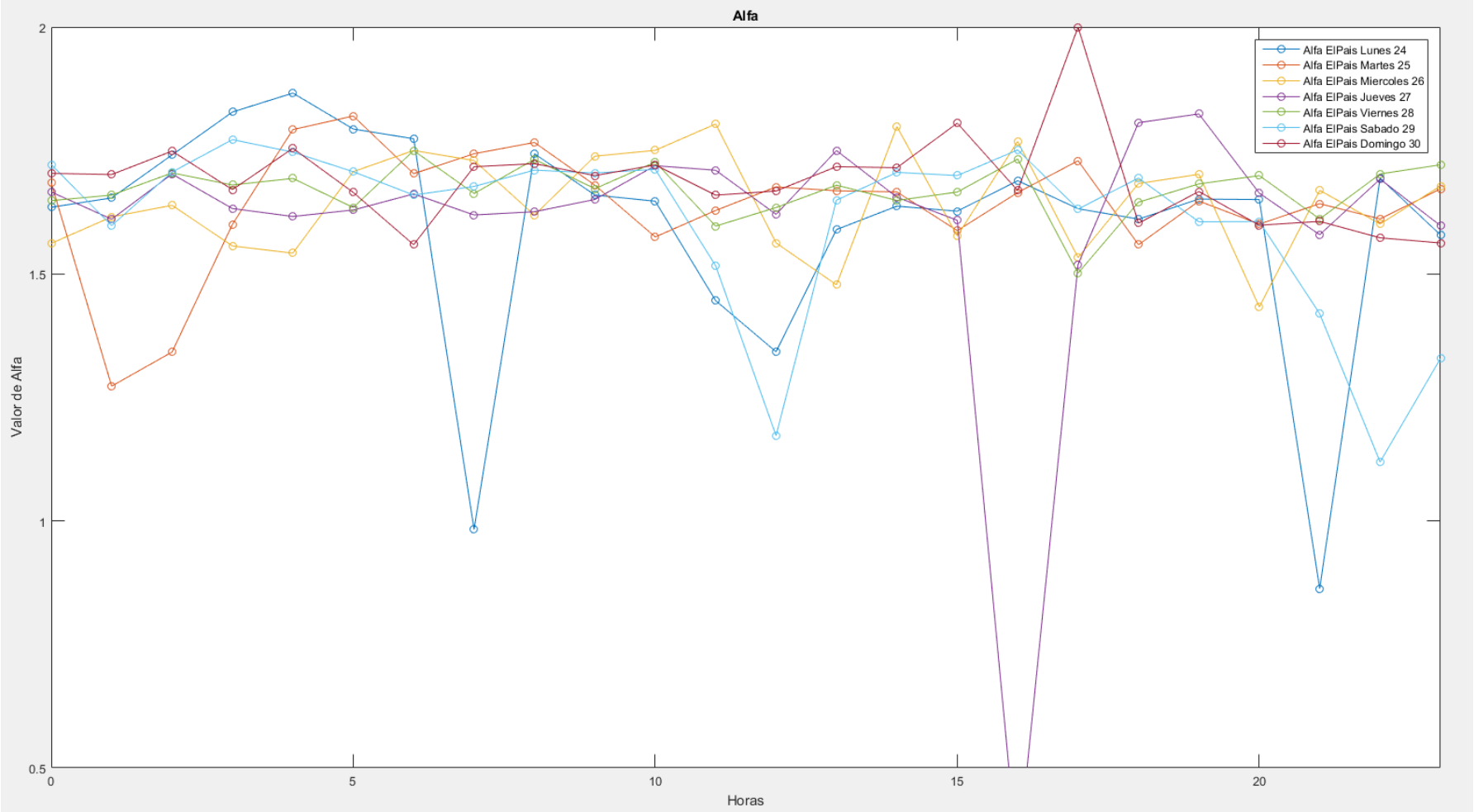
(*7 primeras horas del Lunes 22 corruptas*)



Representación de los valores Alfa sobre el website Aliexpress durante la semana de 24 – 30 de Mayo



Representación de los valores Alfa sobre el website elPais durante la semana de 24 – 30 de Mayo



Representación de los valores Alfa sobre el website Facebook durante la semana de 24 – 30 de Mayo