E.T.S. de Ingenieros Industriales y de Telecomunicación
Departamento de Automática y Computación

Programa de Doctorado en Ingeniería y Arquitectura

Tesis doctoral:

Herramientas pasivas para el análisis y evaluación de prestaciones en redes de alta velocidad

Autor: Iria Prieto Suárez
Director: Dr. Mikel Izal Azcárate
Pamplona, Febrero de 2016

upna
Universidad
Pública de Navarra
Nafarroako
Unibertsitate Publikoa

TESIS DOCTORAL: Herramientas pasivas para el análisis y evaluación de prestaciones en redes de alta velocidad

AUTOR: Iria Prieto Suárez

DIRECTOR DE TESIS: Dr. Mikel Izal Azcárate

El tribunal para la defensa de esta tesis está formado por:

PRESIDENTE:

SECRETARIO:

VOCALES:

(suplente)

REVISORES EXTERNOS:

Resumen

Hoy en día el uso de las redes de ordenadores se ha vuelto imprescindible para casi cualquier tarea. Como consecuencia de la expansión de las mismas, comprobar que una red alcanza el rendimiento y funcionamiento esperado supone una tarea crítica para los administradores. Debido a las dimensiones y a la cantidad de flujos de datos que alcanzan, no siempre es posible utilizar medidas activas. En este tipo de escenarios se puede utilizar herramientas de monitorización pasiva, aunque estos métodos no son siempre triviales. Durante la realización de esta tesis se han analizado algunos problemas para la evaluación de prestaciones a través del uso de herramientas pasivas.

El primer enfoque ha sido el estudio de un generador de tráfico, que sirviera para ser utilizado en simulaciones en las que se pueda reproducir los problemas de una red real. Con este objetivo, se ha propuesto una adaptación del algoritmo de Perlin Noise para generar tráfico al vuelo, de forma ilimitada y con las características de un modelo *Fractional Gaussian Noise* (FGN). La precisión obtenida de este modelo era comparable con la obtenida por un generador de tipo *Random Midpoint Displacement* (RMD).

Otro de los problemas analizados durante la realización de la tesis ha sido la detección de los problemas de disponibilidad percibidos por los clientes. En la actualidad, la detección de interrupciones constituye una de las tareas más críticas en una red. Con este objetivo, se ha propuesto un algoritmo para la detección de interrupciones de servicio a través del estudio del tráfico pasivo. El algoritmo

propuesto es simple, puesto que se basa en el uso de contadores sencillos y en la utilización de dos filtros, para decidir si en un intervalo de tiempo los clientes no pueden acceder a un servicio. El algoritmo ha sido probado en un entorno emulado y un escenario real con clientes accediendo a servicios populares de Internet. Para ambos escenarios se obtuvieron buenos resultados de detección.

Otro problema habitual en las redes es la estimación de la latencia de la red. Mientras que en la literatura se pueden encontrar multitud de estudios para su aproximación a través de medidas activas, se encuentra poco sobre métodos pasivos. Un estimador preciso que puede ser utilizado pasivamente es el uso del *Timestamp Transport control Protocol* (TCP). Sin embargo, se ha comprobado cómo sólo podría ser usado para un pequeño porcentaje de conexiones. Por este motivo, se ha propuesto un estimador enfocado a redes de alta velocidad, basado en la hipótesis que el servidor intentará ajustarse a la ventana anunciada por TCP en el cliente, sin mandar nunca más datos que los permitidos por este. El método ha sido comparado con los valores obtenidos por el estimador basado en el *Timestamp* TCP, y si bien no es tan preciso como este, los resultados han sido lo suficientemente buenos como para ser utilizados.

La última prestación pasiva analizada ha sido la identificación del algoritmo de control de congestión del protocolo TCP utilizado por los servidores. El algoritmo de control de congestión es un algoritmo interno de TCP, que no es anunciado en el establecimiento de la conexión y que depende del sistema operativo del servidor. En la literatura existen numerosos estudios en los que se compara el rendimiento obtenido en un escenario dependiendo del algoritmo de control de congestión utilizado. Por contra, no hay tantos para la identificación de los mismos y menos todavía de forma pasiva. Se ha realizado una primera aproximación de la identificación pasiva utilizando solamente el parámetro principal que caracteriza estos algoritmos. La estimación del parámetro ha permitido la elaboración de un algoritmo clasificador. Este algoritmo ha sido probado tanto para un escenario de entrenamiento como para uno de validación. Dichos experimentos han derivado en porcentajes de identificación elevados.

HERRAMIENTAS PASIVAS PARA EL ANÁLISIS Y EVALUACIÓN DE PRESTACIONES EN REDES DE ALTA VELOCIDAD

"O verdadero heroísmo consiste en trocar os anceios en realidades, as ideias en feitos."- Alfonso Daniel Rodríguez Castelao

Agradecimientos

Durante toda el tiempo que he estado realizando la tesis he contado con un apoyo constante e incondicional por parte de familia, amigos y grupo de investigación, espero ser capaz de mostrar toda mi gratitud en estas líneas.

En primer lugar me gustaría darle gracias a mi director de tesis, Mikel Izal Azcárate, por su paciencia y dedicación. Realmente estoy muy agradecida por buscar siempre un hueco entre la apretada agenda para atender mis dudas, revisar mis papers y por ser una fuente inagotable de nuevas ideas. También me gustaría destacar en estas primeras líneas a Eduardo Magaña y Daniel Morató por su apoyo y por compartir sus conocimientos, muchas de las ideas han surgido después de las reuniones de investigación.

A nivel personal tengo que agradecer el apoyo incondicional de la familia. A mis padres, no sólo por apoyarme durante la tesis sino también por hacerlo siempre, por su positivismo para afrontar los problemas. A mis hermanos y mi cuñada, que tan pacientemente están esperando a que acabe esta memoria para que les pinte camisetas (Álex me acuerdo que tu cuadro va el primero). A mi familia política, por ser tan cariñosos y apremiarme a terminar la tesis con sus palabras de apoyo.

Carlos, a ti te dedico un apartado único, "contigo empezó todo". Gracias por entender que mis "espera, ahora voy" cuando estaba delante del ordenador se pudieran convertir en casi horas. Sólo tú me has hecho reír y desconectar con tus locas ideas de cómo debería de ser la tesis. Gracias también por intentar hacerme

caso cuando te explicaba lo que estaba haciendo, aunque siempre me ría porque no puedas evitar aburrirte y despistarte a los pocos minutos.

Me gustaría también nombrar a mis compañeros de café (Santi, Félix, Luismi, Olga, Igor, Nicolás, Iris y Carlos) y al resto que han pasado por aquí durante estos años, por ser los responsables de lograr un ambiente de trabajo tan estupendo. Realmente aprecio nuestro rato de café diario en el que Santi nos cuenta sus alocadas experiencias con sus amigos de la infancia, los intentos de Félix por vacilar a alguien, los amagos de Olga por enseñarle algo de fútbol al resto, o simplemente por las conversaciones de juegos de mesa en las que Iris y yo nos maravillamos por todos los que conocéis.

No querría terminar sin mencionar a las cuadrillas por todo el interés que han demostrado por cómo iba avanzando. Aunque en el final me haya desesperado un poco la pregunta de cuándo iba por fin a depositar, realmente agradezco todo el interés que habéis demostrado, sois los mejores.

Por último, pero no menos importante, la realización de esta tesis ha sido posible gracias a la beca FPU de la Universidad Pública de Navarra. La tesis no sólo me ha brindado la oportunidad de comenzar una carrera de investigación, también me ha otorgado la oportunidad de dar clases y trabajar con otros grupos de investigación, experiencias muy valiosas tanto desde el punto de vista académico, laboral y a nivel personal.

Índice general

Índice de figuras

Índice de Tablas

CAPÍTULO 1

Introducción

Desde la primera red de conmutación de paquetes *ARPANET*, en 1969, las redes han sufrido una rápida evolución. Aunque la expansión y popularación no se produjo hasta la evolución a Internet y el desarrollo de *World Wide Web* (WWW), a finales de los 80. Paralelo a esta expansión y como consecuencia directa de la misma, la velocidad y prestaciones de las redes ha ido creciendo para atender la constante demanda.

En la actualidad, la comprobación que una red está funcionando correctamente es fundamental. En consecuencia, se pueden encontrar multitud de propuestas para monitorizar tanto equipos como servicios, así como para evaluar el rendimiento de las redes. Este tipo de propuestas normalmente, incluye un conjunto de herramientas y pruebas de tipo activo, es decir, requieren la cooperación de los propios servicios a monitorizar.

Mientras que la comunidad ha centrado la mayor parte del esfuerzo al desarrollo de herramientas activas para analizar el estado de una red, la evaluación de forma pasiva no se ha tenido tanto en consideración. Las medidas pasivas adquieren una gran importancia en redes ya de por sí saturadas, en las que no se desea realizar pruebas que impliquen inyectar nuevo tráfico en las mismas.

Esta tesis se enfoca hacia el análisis y evaluación de prestaciones de redes de alta velocidad, a partir del estudio pasivo del tráfico. Los diferentes capítulos abordan distintas problemáticas a las que se enfrentan hoy en día los administradores de red.

En las siguientes secciones se describe con más detalle la motivación 1.1, y los objetivos 1.2 para la realización de la tesis.

1.1. Motivación

Como se ha mencionado en la introducción, hoy en día, las redes de conmutación de paquetes se han vuelto indispensables; no sólo a nivel de usuario doméstico sino también en el ámbito empresarial. En la actualidad, es inimaginable una empresa en la que los usuarios no tengan acceso a Internet ni estén conectados internamente con otros trabajadores de la empresa. A través de las redes se accede a un gran número de servicios y recursos que se han convertido en una parte imprescindible para las empresas. Debido a esta relevancia adquirida surgen los *Centro de Procesamiento de Datos* (CPD), en los que se organizan y mantienen todo los equipos necesarios.

Los servicios pueden fallar debido a puntos críticos como fallos en el *software o hardware*, o por problemas de red. Además, se debería tener en cuenta que el objetivo, hoy en día, no es sólo garantizar un servicio sino también, que éste opere con la máxima eficacia posible.

En consecuencia, la evaluación de prestaciones en las redes se ha convertido en un factor crítico. La figura de administradores de red juega un papel vital para las empresas, monitorizando las redes para garantizar su correcto funcionamiento así como asegurando que el nivel de rendimiento de las mismas sea el esperado. Por otro lado, el incremento de la demanda de servicios ofertados y demandados ha provocado la necesidad de crear grandes redes de alta velocidad. En este tipo de entornos, las tareas de los administradores de red se pueden volver críticas por el volumen de datos que deben de tener en cuenta para monitorizar y en caso de existir, encontrar las causas de los problemas.

En la literatura se pueden encontrar diversos estudios y herramientas para la evaluación de prestaciones de forma activa. Por ejemplo, para conocer si un

servidor está caído se puede hacer uso de la conocida herramienta *Ping*, o emplear sistemas más complejos como puede ser Nagios, [Nag]. Sin embargo, en las grandes redes que soporten mucho tráfico, no siempre es posible utilizar este tipo de herramientas, ya sea por contrato o porque los sistemas están saturados. En otros casos, son posibles estos tipos de análisis pero las herramientas pueden resultar inadecuadas para el estudio de problemas concretos. Por ejemplo, en una red que sufra congestión en pequeños periodos de tiempo aparentamente aleatorios, las pruebas activas no darán información del problema. Sin embargo, un estudio pasivo del tráfico capturado durante ese día puede relevar cuales son las causas del aumento de tráfico.

Una de las dificultades de evaluar prestaciones mediante el estudio del tráfico pasivo, es que muchas de las métricas se deberán de inferir a partir de las conexiones capturadas. Es el caso, por ejemplo, de la latencia de la red. En el caso de las medidas activas se pueden realizar pruebas para conocer el retardo de una red, por ejemplo mediante el uso de la herramienta anteriormente citada, *Ping*. En el caso de las medidas pasivas, se tendrá que estimar por ejemplo, a partir de los paquetes iniciales de una conexión.

Además, para la realización de las medidas se debería de tener en cuenta multitud de factores que pueden afectar al tráfico observado: el punto de captura, posibles pérdidas de paquetes por no ser capaz de procesar todo el volumen del tráfico el driver de captura, tráfico inconexo debido al comportamiento inesperado de clientes y servidores, etc.

No obstante, la principal ventaja de este tipo de análisis frente al activo, que no es invasivo, promete ser un recurso importante para los administradores de red.

Esta tesis surge del proyecto nacional Europeo, **INSTINCT, Interdomain Strategies IN optical Core neTworks**, en el cual participaban la Universidad Pública de Navarra y la Universidad de Valladolid. El objetivo del proyecto era el estudio de la interoperabilidad de redes heterogéneas a nivel de capa óptica. Paralelamente al desarrollo de este proyecto se comenzó otra línea de investigación motivada por las preguntas que surgieron de la búsqueda de modelos de tráfico que se ajustaran lo más posible a situaciones reales, los cuáles serían usados como entradas en las redes ópticas. El análisis pasivo del tráfico de una

red convencional planteaba retos que todavía no habían sido abordados y que fueron cobrando mayor relevancia en el grupo de investigación. Motivados por la necesidad de la evaluación de prestaciones de forma pasiva en las redes de tráfico contemporáneas, el objetivo inicial de la tesis viró hacia este tipo de estudio.

Los objetivos de esta tesis, tras el cambio de dirección, son los que se describen en la siguiente sección.

1.2. Objetivos

Los objetivos de este trabajo han sido los siguientes:

- Definir las diferentes problemáticas que se puedan presentar para la evaluación de prestaciones de forma pasiva. Normalmente, los diferentes problemas a los que se enfrentan los administradores de red, han sido estudiados de forma activa. Por ello, hoy en día hay un amplio abanico de posibilidades para su estudio de forma pasiva, por lo que se deberá de decidir qué tipo de problemas se consideran intereresantes para un estudio en profundidad.

- Evaluación y desarrollo de testbeds para la reproducción de problemas de red, ya sean a través de simulación o emulación. Estos entornos permiten estudiar métodos de evaluación de problemas en las redes con el fin de analizarlos, ya que no siempre se podrá a recurrir a redes reales y provocar que se repitan problemas o situaciones anómalas. En estos casos, se deberá de trabajar con sistemas que se adapten lo más fielmente posible a este tipo de escenarios y dónde se puedan provocar las mismas causísticas que generen muestras similares.

- Determinar cómo puede decidirse a partir del tráfico, cuándo se está produciendo una interrupción a nivel de servicio. Las detección de cuándo un servicio está caído, es el tipo de problemas que usualmente se abordan mediante monitorización activa. Relacionado con este objetivo se pueden describir los siguientes:

  - Identificar y estudiar el efecto de las posibles interrupciones de servicios a nivel de tráfico de paquete.
  - Evaluación de un método de monitorización pasiva capaz de actuar como una medida activa, que detecte cuándo deja de darse un servicio.
  - Detección pasiva de servicios para un entorno real. El objetivo es analizar un caso de uso real, en la que se conozca que los usuarios hagan uso de servicios populares, como el acceso a alguna red social, para comprobar si se producen interrupciones.

- Ser capaz de medir el retardo de una red, *Round Trip Time* (RTT), a través del tráfico capturado. El RTT es una métrica importante para la evaluación de la calidad del rendimiento de una red. Su estimación pasiva no es una tarea trivial puesto que parámetros como dónde se capture pueden afectar a su cálculo. Derivado de este objetivo se pueden definir los siguientes:

  - Estudiar el estado del arte sobre la estimación del RTT a partir del estudio de trazas de tráfico.
  - Evaluación de estas medidas en un escenario real.

- Identificación de configuraciones del protocolo TCP, que puedan afectar al rendimiento directo de las conexiones TCP de la red. TCP constituye el protocolo más utilizado en las redes de tráfico. Aunque el protocolo está definido mediante RFC, existen diferentes configuraciones, dependiendo de los propios servidores o las versiones de sistema operativo utilizadas, que pueden afectar al rendimiento observado en una red. Como consecuencia de conseguir este objetivo, se deberían de obtener los siguientes:

  - Identificación de las distintas configuraciones que pueden afectar directamente al rendimiento de las conexiones.
  - Determinar si es posible identificar las configuraciones solamente con los paquetes de tráfico capturados. Algunos de los parámetros configurables de TCP son anunciados en el inicio de la conexión, pero otros

que afectan al funcionamiento interno de TCP, no aparecen entre los parámetros, como el algoritmo de control de congestión.

1.3. Estructura de la tesis

El resto de la tesis se organiza como sigue. En el capítulo 2 se realiza un estudio sobre los algoritmos de generación de tráfico utilizados en simulaciones. Tras el estudio de las ventajas e inconvenientes de este tipo de algoritmos se propone una nueva adaptación de un algoritmo para la generación de tráfico auto-similar, Perlin Noise. Este algoritmo es utilizado habitualmente en la computación gráfica. El objetivo es poder simular entornos de red cuyas características de tráfico sean lo más reales posibles. Los resultados son comparados con los obtenidos por un método descrito en la literatura así como con el modelo teórico.

El capítulo 3, presenta una aproximación para la detección de interrupciones de servicios, que pueden ser atendidos por varios servidores, de forma pasiva. En el capítulo se estudian los efectos de diversas interrupciones, causadas por diferentes problemas, a nivel de tráfico de paquete. En el mismo se propone un método para su detección y se analiza tanto en un escenario emulado, como en un escenario real en el que los clientes acceden a servicios populares a través de Internet.

El capítulo 4, se analizan diferentes formas de medir el retardo en una red, *RTT*, de manera pasiva. Nuevamente se propone un algoritmo para su estimación y se estudia para un escenario emulado y un escenario real.

En el capítulo 5, se propone un método de clasificación simple para identificar el algoritmo de control de congestión utilizado por TCP a través del estudio de conexiones agrupadas para servidores. Al comienzo del capítulo se analiza las posibilidades que se tendrían en un escenario real de observar casos que permitieran estimar los parámetros necesarios para la identificación de los algoritmos. Por su complejidad, para la presentación y análisis del algoritmo se utiliza un escenario emulado.

Finalmente en el último capítulo 6, se presentan las conclusiones y las líneas futuras del trabajo presentado.

CAPÍTULO 2

Generador de Fractional Gaussian Noise usando Perlin Noise

Aunque esta tesis está enfocada hacia el análisis pasivo de tráfico, a la hora de comprender qué situaciones se pueden dar en una red grande con muchos clientes, algunas veces la simulación constituye un punto de partida importante. Por ejemplo, si se quiere analizar modelos de saturación que puedan sufrir ya sean desde servicios hasta servidores, para no interferir con el tráfico real de la red se pueden recurrir a estudios basados en modelos teóricos de ser posible, o a simulaciones. La ventaja de las simulaciones es que permiten el modelado de situaciones anómalas que pueden ocurrir de forma esporádica.

Para que una simulación se asemeje lo más posible a una situación real, necesita de modelos de tráfico de entrada que tengan las mismas propiedades que los patrones de tráfico reales. Estas propiedades las cumplen los modelos auto-similares. De entre los modelos auto-similares uno de los más populares es el FGN. Sin embargo, el coste computacional de los algoritmos usados para generar este modelo, es muy elevado. La desventaja de estos métodos es que se debe de escoger, antes de comenzar la simulación, el tamaño de la traza que va a ser utilizada.

En este capítulo se describe la adaptación del proceso Perlin Noise, el cuál habitualmente ha sido utilizado para la generación de texturas y fondos en computación gráfica, para la generación de trazas FGN. El objetivo es describir

un método capaz de dotar a las simulaciones con tráfico auto-similar de forma constante, sin que se tenga que decidir con antelación el tamaño de la traza para ser utilizado.

De esta forma se podría utilizar para simulaciones de grandes redes en las que se requiera repetir situaciones críticas observadas en la red real, sin que ello afecte al rendimiento de la red.

El capítulo está organizado como sigue, en la sección 2.1 se describe el estado del arte de otros algoritmos para la generación de tráfico auto-similar así como la motivación para proponer una nueva aproximación. Seguidamente, sección 2.2, se presentan las definiciones y anotaciones usadas a lo largo de este capítulo. En la sección 2.3 se explica el generador de tráfico Perlin Noise y cómo obtener los parámetros deseados. La sección 2.4 muestra el método para adaptar las trazas resultantes del algoritmo a un canal con capacidad limitada y en la sección 2.5 se muestran los resultados y las comparaciones. Finalmente, en la sección 2.6, se presentan las conclusiones del capítulo.

2.1. Introducción

En numerosas ocasiones los administradores de red se enfrentan al problema de buscar a posteriori el culpable de la saturación de un router o servidor. Tratar de buscar estos problemas después de que hayan sucedido requiere, en muchos casos, haber monitorizado el tráfico de la red. Pero en los casos en los que no fue posible la monitorización, algunas veces, la simulación es un punto de partida importante para comprender qué puede estar ocurriendo. Mediante simulación se pueden reproducir redes y tráfico reales permitiendo un estudio de las mismas sin interferir en las redes reales. A diferencia de la redes reales, los experimentos podrán lanzarse tantas veces como se estime necesario sin que ello tenga repercusiones en la actividad de la red.

Con el propósito de simular el comportamiento de grandes redes, en la literatura se han descrito numerosos modelos de tráfico diferentes: procesos de Poisson, cadenas de Markov, modelos autoregresivos, etc. Sin embargo, para capturar los modelos que se observan en las redes reales de dependencia a largo plazo, *Long range dependence* (LRD), se utilizan modelos auto-similares. En la

literatura se pueden encontrar algunas propuestas de estos modelos, por ejemplo para las redes locales Ethernet [KSM94; LTW+94], o los usados para la comprensión de Vídeo de tipo, *Variable Bit Rate* (VBR), [BST+95; FM94; Gor96; HL96; JLS97; NR02].

El *Fractional Brownian Traffic*, es un modelo de tráfico auto-similar para modelar la cantidad de tráfico acumulado recibido, como una función continua con incrementos gausianos. Estos incrementos forman lo que es llamado, FGN. El modelo FGN se caracteriza por tres parámetros: media, varianza y el parámetro *Hurst*, H. La media y la varianza son los parámetros de la distribución marginal del proceso de llegadas por unidad de tiempo. El tercer parámetro, H, mide la rafagosidad del proceso. Un $H = 0{,}5$ corresponde al ruido puro gausiano con llegadas independientes, lo cual es un proceso autosimilar pero no LRD. Valores mayores, $0{,}5 < H < 1$, incrementan el proceso independiente, siendo $0{,}7 >= H <= 0{,}8$ los valores típicos utilizados para las trazas de Internet, [IAM+06].

Un método para la generación de FGN ha sido descrito por Huang et al., [HDL+95]. Aunque obtiene una buena precisión, debido al coste computacional de este proceso, $O(N^2)$, posterior a esta propuesta han sido sugeridos nuevos métodos, [Ost06; PHJ+02]. Con el objetivo de alcanzar un compromiso entre el coste computacional y la precisión, algunos otros métodos pueden encontrarse en la literatura, como por ejemplo el método desarrollado en [TDL98]. Este método es capaz de generar trazas con esfuerzo O(n) con respecto a la longitud. Otro ejemplo de método aproximado, el RMD, fue descrito en [LEW+95], aunque este último método ha sido utilizado tanto en el campo de computación gráfica como en el de generación de tráfico.

La desventaja de estos métodos es que normalmente generan tamaños fijos de bloques de ejemplos de FGN, como por ejemplo en el estudio [HDL+95], en el que se es capaz de generar trazas exactas con menos muestras que $5 \cdot 10^5$. Esto significa que el tamaño tiene que ser decidido antes de comenzar todo el proceso y además, todo el proceso para generar la traza debe de ser almacenado antes de realizarse la simulación. Este hecho supone un límite para el tiempo de simulación, especialmente en redes de alta velocidad dónde se necesita grandes trazas para simular incluso periodos de tiempos pequeños. Este comportamien-

to es producido por la generación de FGN a partir de la generación del proceso del espectro y usar transformadas rápidas de Fourier para conseguir el proceso de dominio de tiempo real, o partir de la generación primero de las muestras más lejanas y redefiniendo las muestras intermedias.

El objetivo de este capítulo es obtener un generador capaz de sintetizar al vuelo FGN, con el fin de evitar la necesidad de pre-generar grandes bloques de muestras por adelantado. Este generador puede ser utilizado para simulaciones de grandes redes o incluso para redes ópticas de alta velocidad, en las que se necesite un gran número de fuentes con modelos de tráfico LRD como entradas.

Para conseguir este generador se propone una adaptación de un proceso conocido, *Perlin Noise*. Este modelo normalmente es usado en computación gráfica para conseguir texturas y efectos naturales, pero puede ser reinterpretado como un generador FGN. La precisión conseguida con la adaptación del algoritmo para obtener trazas a partir de los parámetros que definen a un proceso FGN, media, varianza y parámetro H, son comparadas con las obtenidas con otros generadores de FGN.

Por otro lado, FGN genera tráfico con una distribución marginal gausiana, y por eso puede generar picos instantáneos mayores que la capacidad real de un canal. Este problema también es tenido en cuenta y se propone un método para limitar el tráfico FGN dependiendo de la capacidad del canal, mientras que se mantenga en media la rafagosidad deseada.

2.2. Definiciones y Notación

El proceso de Perlin Noise es definido como la suma de varios ruidos, $x_i(t)$, denominados octavas, con cada vez componentes espectrales de mayor frecuencia. Cada octava es generada a partir de un proceso aleatorio independiente. La octava base, $i = 0$, es un ruido aleatorio independiente generado a intervalos, Δt. Para cada $t = k\Delta t$ un valor aleatorio independietne es generado mediante una distribución uniforme. Los valores para los puntos intermedios, $t = k\Delta t$, se obtienen mediante interpolación de los valores generados previo y siguiente. La interpolación puede ser lineal o cualquier función suave. Las sucesivas octavas, $i = 1...n$, se construyen de la misma forma con ruido, tomando valores en un

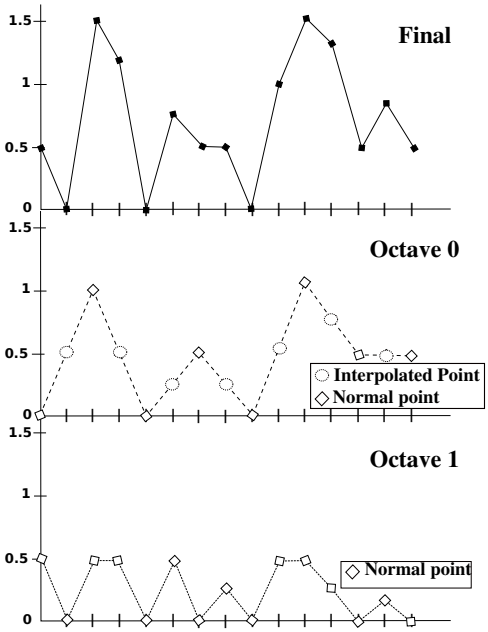


Figura 2.1: Ejemplo del algoritmo de Perlin Noise.

número creciente de puntos intermedios. En la octava, i, el ruido toma nuevos valores cada $k(\Delta t/f^i)$. Cada octava considerada tiene, habitualmente una amplitud A_i menor que la octava anterior. La amplitud es expresada normalmente a partir de un factor de persistencia, p, de forma que: $A_i = p^i$.

Dados n procesos aleatorios e independientes de ruido, $x_i(t)$ es interpolado cada $k\frac{\Delta t}{f^i}$. El Perlin noise es obtenido cómo:

$$n(t) = \sum_{i=0}^{n-1} p^i x_i(t) \tag{2.1}$$

En la figura 2.1 se muestra un ejemplo del algoritmo utilizando dos octavas. La notación que se va a utilizar es la que sigue:

- n = Número de octavas.
- x_i = Octava i-th. La primera octava será x_0 y la última x_{n-1}.

- p = Persistencia.
- A_i = Amplitud para cada octava i-th.
- f = Factor de interpolación.

Se debe notar que en este proceso, cada octava k genera f^k valores aleatorios para cada valor no-interpolado de la primera octava. En el dominio de frecuencia, la octava k tiene componentes espectrales que se extienden f^k veces la extensión de la primera octava. De hecho, esta generación de cada octava, $x_i(t)$, es aproximádamente un ruido blanco limitado por la frecuencia $\frac{2\pi f^i}{\Delta t}$.

Un Perlin Noise con n octavas y un factor de interpolación f, generará f^{n-1} muestras para la octava con frecuencia más alta por cada nueva muestra de la primera octava. Para la generación de tráfico, este proceso puede ser interpretado cómo la llegada de un volumen de tráfico cada ciertos intervalos fijos de tiempo, δt, correspondiente a la octava más alta. En este caso, usar n octavas significa que la primera octava genera muestras de ruido y se mueve lentamente durante f^{n-1} franjas fijas de tiempo hacia el próximo valor generado. De esta forma, cada octava contribuye a la generación de un proceso con correlación que al alcanza al menos $f^{n-1}\delta t$ cadencias de tiempo.

Desde este punto de vista, un Perlin Noise con amplitud de octava decreciente, genera procesos LRD en los que la caída de la densidad espectral con la frecuencia es controlada por p. Este proceso es análogo a un ruido gausiano fraccional, *Fractional Gaussian Noise*, cuya caída de la densidad espectral de potencia es controlado por el parámetro Hurst, H. Partiendo de esta idea, el objetivo de este capítulo es la generación de trazas FGN utilizando la aproximación del Perlin Noise en lugar del RMD.

El factor habitual de interpolación usado por el algoritmo de Perlin Noise en aplicaciones gráficas es de $f = 2$. Para lograr el objetivo planteado, la generación de tráfico, es interesante controlar el alcance de la dependencia ya que se podrían usar valores grandes de f para alcanzar correlaciones mayores.

Por último, se debe notar que para ajustar la salida del proceso de Perlin Noise, la almplitud es la suma de los procesos dados por 2.2. La media de este

proceso se puede calcular como la media de un generador de ruido uniforme, $x(t)$ el cuál es modulado por la amplitud de la octava, p^i, ecuación 2.3.

$$Amplitude_{max} = \sum_{i=0}^{n-1} p^i = \frac{p^n - 1}{p - 1} \tag{2.2}$$

$$\bar{n} = \bar{x} * \sum_{i=0}^{n-1} p^i = \bar{x}\frac{p^n - 1}{p - 1} \tag{2.3}$$

2.3. Generación del tráfico con Perlin Noise

Hasta ahora se ha presentado la idea básica de cómo funciona el algoritmo de Perlin Noise. En esta sección, se explica cómo usar el proceso adaptado para obtener trazas FGN de características similares a aquellas obtenidas mediante el método RMD. Cómo se ha citado anteriormente, el proceso FGN es caracterizado por el conjunto de parámetros que son utilizados como entradas en el algoritmo: media, varianza y el parámetro Hurst, (μ, σ^2, H). Una traza FGN puede ser transformada de forma que se cambie su media y varianza pero se mantengan inalteradas su auto-similitud y la estructura de correlación LRD definidos por el parámetro H. Por esta razón, el primer paso para generar FGN mediante el proceso de Perlin Noise descrito es conseguir un proceso que tenga una H determinada. Para ello, se debe de elegir tanto un valor de persistencia, p, cómo el número de octavas, n, y el factor de interpolación, f, ya que estos parámetros definen el dominio de la frecuencia del proceso.

En la figura 2.2 se muestra la dependencia del parámetro H con las variables p, n y f. Se debe de notar que para factores bajos de persistencia se obtienen valores altos de H. Esto puede ser explicado porque el proceso es obtenido como la suma de componentes cuya potencia está concentrada en áreas crecientes. La suma de cada octava sin modificar hace decaer la potencia del espectro asociada con LRD. Para conseguir un proceso de llegadas independiente, es decir $H = 0{,}5$, la persistencia tiene que ser incrementada con el objetivo de conseguir más peso para la octava mayor, la cuál ya es un ruido blanco.

La elección del número de octavas n y el factor de interpolación f, son importantes porque a mayor n y f, se obtiene un mayor número de muestras correladas.

Como puede ser observada en la figura, por ejemplo, si se quiere obtener un valor de $H \simeq 0{,}7$, empleando tan sólo 6 octavas, se debería de escoger una persistencia cerca de $p = 1{,}5$.

Para conseguir un proceso de Perlin Noise, $n_0(t)$, se generarán n octavas con generadores aleatorios independientes con $\bar{x_i} = 0$ y peso p^i. Este proceso tendrá la H deseada y de media 0.

Este proceso todavía no tiene la media y varianza deseadas. Como se ha mencionado anteriormente, sin embargo, el proceso se puede reajustar para que tenga determinada media y varianza sin que cambie sus propiedades de autosimilitud, este será el segundo paso. El nuevo proceso, $n(t)$ tendrá una media, μ y varianza, σ^2, concretas:

$$n(t) = \mu + \frac{\sigma}{\sigma_0} n_0(t) \tag{2.4}$$

Siendo σ_0^2 la varianza de $n_0(t)$. Se debe notar que es fácil generar el proceso $n_0(t)$ con media 0, pero no es tan sencillo obtener un proceso normalizado con varianza 1. En la siguiente sección se deriva la fórmula teórica de la varianza del proceso $n_0(t)$ construido cómo se ha descrito.

Después de reescalar el proceso con el fin de obtener unas varianzas y medias determinadas, el parámetro H permanece inalterado. De esta forma se ha obtenido una traza FGN con medias, varianzas y parámetro H según lo deseado y además cuya correlación viene dada por f^{n-1}.

Se debería de notar que el valor de σ_0 podría ser obtenido del proceso real sintetizado, $n_0(t)$, una vez que este haya sido generado. Sin embargo, el objetivo de este capítulo es generar un proceso continuo $n(t)$ que no tenga la necesidad de generar por adelantado bloques de traza de tamaño fijo. El valor teórico de σ_0 permite la generación de muestras $n(t)$ tan pronto como las muestras de $n_0(t)$ sean generadas.

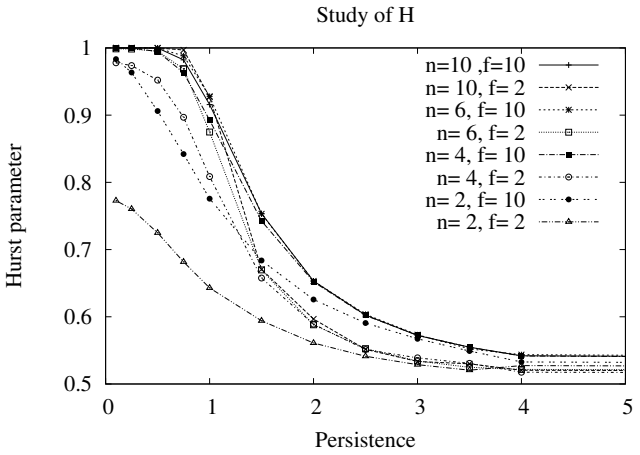


Figura 2.2: Dependencia del parámetro H con el número de octavas, la persistencia y el factor de interpolación.

2.3.1. Varianza del proceso de Perlin Noise

El objetivo de esta sección es presentar la derivación de la fórmula teórica para calcular la varianza de $n_0(t)$, de la cuál se quiere hacer uso para poder escalarla al vuelo. El proceso $n_0(t)$ es generado como la suma de n procesos independientes, $x_i(t)$, con medias $E[x_i(t)] = 0$. Esto explica que la suma de los procesos tendrá tambien una media $E[n_0(t)] = 0$, y la varianza será la suma ponderada de las varianzas de las octavas individuales, ecuación 2.5.

$$\mathrm{Var}[n_0(t)] = \sum_{i=0}^{n-1} \mathrm{Var}[p^i x_i(t)] = \sum_{i=0}^{n-1} p^{2i} \mathrm{Var}[x_i(t)] \tag{2.5}$$

La varianza de cada octava depende de la función de densidad del generador aleatorio. Cada octava es generada a partir de muestras de una variable aleatoria uniforme, x, con una función de densidad $f(x)$, tomando valores en el intervalo $(-1, 1)$ con media 0, ecuación 2.6.

$$f(x) = \begin{cases} \frac{1}{2} & when \quad -1 \leq x \leq 1 \\ 0 & Otherwise \end{cases} \tag{2.6}$$

En la octava con la frecuencia mayor, cada muestra es una ocurrencia del generador, pero las frecuencias de las octavas más bajas tienen puntos interpolados entre las muestras nuevas generadas. El número de muestras interpoladas en cada octava se puede generalizar: en la octava i se tendrán $k = f^{n-1-i}$ muestras interpoladas. Los puntos intermedios en la octava i, entre una muestra con valor a y la siguiente muestra con valor b pueden ser escritos de la forma que sigue:

$$a + (b - a)\frac{i}{k} \qquad \text{for} \quad i = 0...k \tag{2.7}$$

La varianza para la octava con mayor frecuencia, x_{n-1}, es calculada a partir de la función de densidad de la generación.

$$\mathrm{Var}[x_{n-1}(t)] = \int x^2 f(x)\,\mathrm{d}x = \int_{-1}^{1} x^2 \frac{1}{2}\,\mathrm{d}x = \frac{1}{3} \tag{2.8}$$

La varianza para otras octavas puede obtenerse por derivación, teniendo en cuenta que los puntos intermedios son fijos una vez que los valores de las muestras aleatorias se conocen. De esta forma se puede escribir como una integral condicionada para la primera muestra siendo a y la integral hasta la muestra correcta del intervalo.

$$\begin{aligned} V_a &= \int_{-1}^{1} \frac{1}{k} \sum_{i=1}^{k} \left(\frac{i\ (x-a)}{k} + a \right)^2 \frac{1}{2}\,\mathrm{d}x = \\ &= \frac{\left(6\,a^2 + 2\right)\ k^2 + \left(3 - 9\,a^2\right)\ k + 3\,a^2}{18\,k^2} \end{aligned} \tag{2.9}$$

Siendo a la muestra aleatoria de la izquierda y k el número de intervalos de interpolación en la octava i, $k = f^{n-1-i}$. La varianza es obtenida por integración de nuevo sobre la muestra izquierda de la función de densidad dependiendo sólo de k para la octava i.

$$V[x_i(t)] = \int_{-1}^{1} V_a f(a)\,\mathrm{d}a = \frac{2\,k^2 + 1}{9\,k^2} \tag{2.10}$$

Con la suma ponderada de las varianzas de cada $x_i(t)$, la varianza de $n_0(t)$ es obtenida usando la ecuación 2.5.

$$\begin{aligned} \mathrm{V}[n_0(t)] &= \sum_{i=0}^{n-1} p^{2i} V[x_i(t)] = \\ &= \frac{\sum_{i=0}^{n-1} \left(2\, f^{2\,n} + f^{2\,i+2}\right)\, p^{2\,i}}{9\, f^{2\,n}} \end{aligned} \tag{2.11}$$

De esta forma, la varianza puede ser calculada solamente mediante p, n y f, lo que permite reescalar $n_0(t)$ para un $n(t)$ con una media y varianza determinada.

Un ejemplo del resultado de este proceso es mostrado en la figura 2.3. La gráfica ha sido generada usando 6 octavas, un factor de interpolación $f = 2$, y una persistencia de $p = 1{,}4$ para lograr obtener un $H \simeq 0{,}7$. En la tabla 2.1 se muestra las características de la traza obtenida con esta configuración.

En las siguiente secciones se analiza el resultado del proceso en comparación con otros FGN obtenidos utilizando otros métodos.

Tabla 2.1: Análisis de los valores obtenidos a través de la estandarización del proceso.

$\mu_{Objetivo}$	$\mu_{Obtenida}$	$\sigma^2_{Objetivo}$	$\sigma^2_{Obtenida}$	$H_{Objetivo}$	$H_{Obtenida}$
60,000	60,000.82949	15,000	15,047.568	0.7	0.701

2.4. Generación de un canal de tráfico de capacidad limitada

En la sección previa se ha descrito un método para generar trazas FGN $n(t)$, con media y varianza concretas, a través del escaldo de procesos de Perlin Noise, $n_0(t)$. Normalmente, con el objetivo de simular el comportamiento de una red real, es interesante generar tráfico para un determinado canal que tendrá una capacidad limitada, C. El problema con el proceso de salida es que una vez que

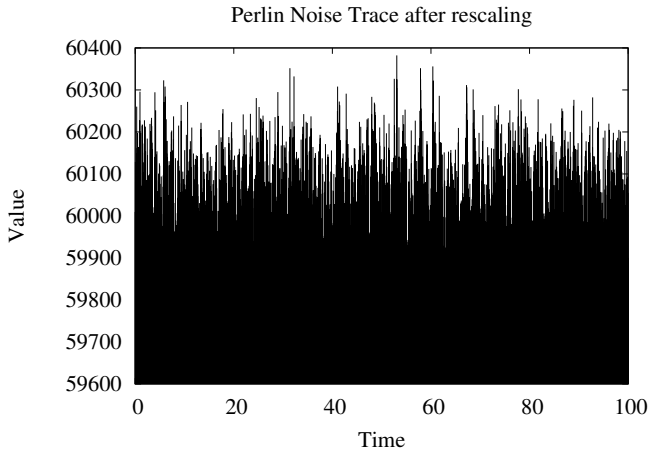


Figura 2.3: $n(t)$ Con los valores deseados $\mu = 60,000$, $\sigma^2 = 15,000$, $H = 0{,}7$ ($f = 2$, $n = 6$, $p = 1{,}4$).

la media y la varianza se han escogido puede que alguno de los valores máximos del volumen de tráfico por muestra en cada intervalo sean mayores que el límite de la capacidad del canal o incluso que sean inferiores a 0. El proceso FGN ha sido generado sin tener en cuenta estos límites.

Para lidiar con este inconveniente, se necesita un algorimo que modele la salida $n(t)$ para que se mantenga dentro de los límites de la capacidad del canal y por encima de cero. Este proceso de limitar la capacidad, $n_s(t)$, debe además, preservar en medida de lo posible, tanto la media, la varianza y el parámetro Hurst.

Con el objetivo de mantener la media en el sistema, para cada muestra de tráfico que esté por encima de la capacidad, se almacenerá el volumen que exceda de la capacidad, para ser añadido al tráfico generado en el siguiente intervalo. En los casos en que los periodos para los que el tráfico generado esté por encima del límite del canal, ocurran por un tiempo extendido, el tráfico acumulado almacenado será generado después de que la época con alta carga termine. De esta forma se mantiene la misma esperanza $\bar{n_s}(t) = \bar{n}(t)$ dada que $\bar{n}(t) < C$. Sin embargo, cuando el volumen del tráfico generado en un intervalo es negativo, se necesita de la misma forma añadir el tráfico generado al siguiente intervalo

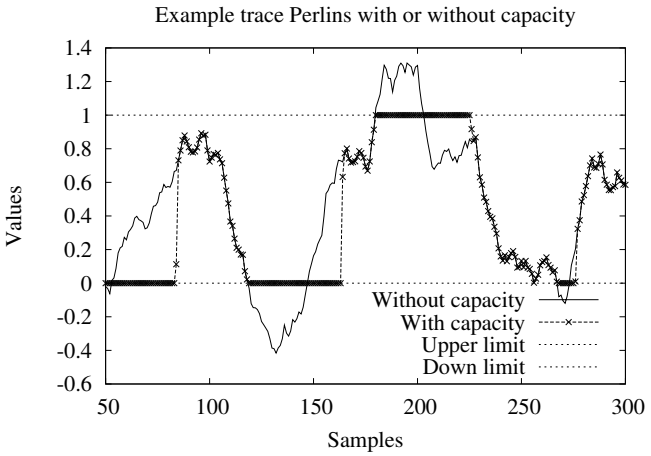


Figura 2.4: Cómo funciona realmente el agoritmo para preservar la media de un sistema real con capacidad mostrando los bytes acumuladas de una traza del Perlin Noise.

reduciendo el tráfico generado posteriormente y manteniendo la media de $n_s(t)$ igual a la de $n(t)$.

En la figura 2.4 se muestra un ejemplo. En la figura se dibuja los bytes generados en una traza de Perlin Noise con los parámetros: $H \simeq 0{,}994$, $\mu = 0{,}3$ and $\sigma^2 \simeq 0{,}2$. En la figura se dibujan el proceso inicial y la versión modelada. La modelada nunca tiene picos mayores que la capacidad dada, que en este caso se ha normalizado siendo $C = 1$, o por debajo de 0. Esta modelización de la traza original mantiene el mismo ratio promedio de crecimiento que la traza original, $\mu = 0{,}3$.

El procesado para limitar la capacidad es fácil de realizarlo siguiendo el algoritmo previo. El problema es el efecto que puede tener el modelado en la varianza de salida y en la estructura de correlado dada para el parámetro Hurst. De lo explicado anteriormente se deduce que un proceso con un rango limitado no puede tener una varianza grande arbitrariamente. En la siguiente sección se estudia el límite de la varianza y el parámetro H que pueden ser esperados utilizando el algoritmo para limitad la capacidad. Para hablar de forma general normalizamos el límite de la capacidad $C = 1$ y se escalarán todos los parámetros acorde con esto.

2.4.1. Error absoluto de la varianza

Como ya se ha citado en la anterior sección, el proceso de modelado no modifica la media del tráfico de la salida. Sin embargo, la varianza puede verse claramente afectada. El límite del canal no permitirá que la varianza puede crecer mucho, incluso si el proceso original tenía una varianza arbitraria grande. Además, se debería de tener en cuenta que la máxima varianza puede depender del valor de la media del tráfico generado. Para un proceso con media $\mu = 0{,}5$, se puede esperar que tenga $\sigma \simeq 0{,}25$. Pero el mismo valor de varianza parece muy alto para un proceso con $\mu = 0{,}01$ ya que los valores por debajo de 0 serían cortados en el proceso final.

Con el objetivo de evaluar los valores razonables del parámetro objetivo μ, σ^2, que pueden ser alcanzados de forma razonable con las trazas obtenidas mediante el algoritmo, se generan trazas con diferentes μ,σ^2 y la varianza real que se obtiene tras el modelado es comparada con la varianza objetivo. En la figura 2.5, se muestra el error absoluto de la varianza obtenida dependiendo de μ y σ, usando diferentes parámetros Hurst. Como puede ser observado, el mayor error aparece cuando $H \simeq 0{,}5$. De este proceso se pueden extrapolar límites para decidir cuáles son los valores de los parámetros que permiten obtener trazas cuyos errores de varianza sean asumibles.

La figura 2.6 muestra el plano de (μ, σ), indicando el área donde el error de la varianza es menor, por ejemplo $0{,}1$ en el caso de $H = 0{,}5$. Para el resto, $H > 0{,}5$, el error de la varianza es incluso más baja, por lo que la misma área es segura para generar todos los valores de H.

Se puede definir una zona de generación segura a través de los siguientes límites:

- $\mu \leq 0{,}15 \Rightarrow \sigma^2_{max} = 0{,}150 \quad (\sigma = 0{,}35)$
- $0{,}150 < \mu \leq 0{,}2 \Rightarrow \sigma^2_{max} = 0{,}16 \quad (\sigma = 0{,}4)$
- $0{,}2 < \mu \leq 0{,}7 \Rightarrow \sigma^2_{max} = 0{,}2025 \quad (\sigma = 0{,}45)$
- $0{,}7 < \mu \leq 0{,}9 \Rightarrow \sigma^2_{max} = 0{,}16 \quad (\sigma = 0{,}4)$
- $\mu > 0{,}9 \Rightarrow \sigma^2_{max} = 0{,}1225 \quad (\sigma = 0{,}35)$

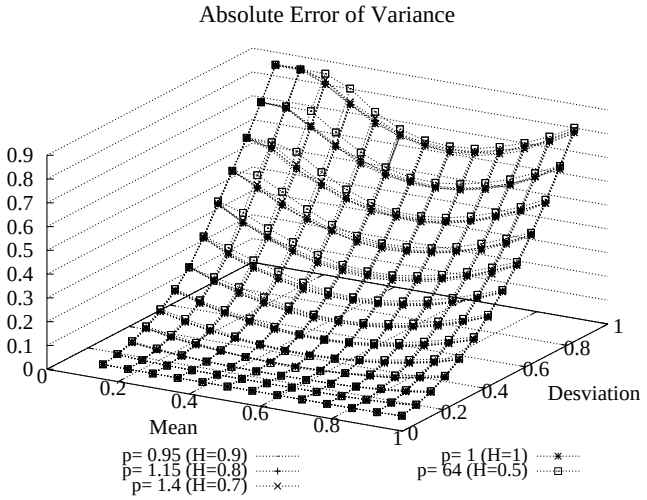


Figura 2.5: Error absoluto de la varianza para trazas con persistencias diferentes.

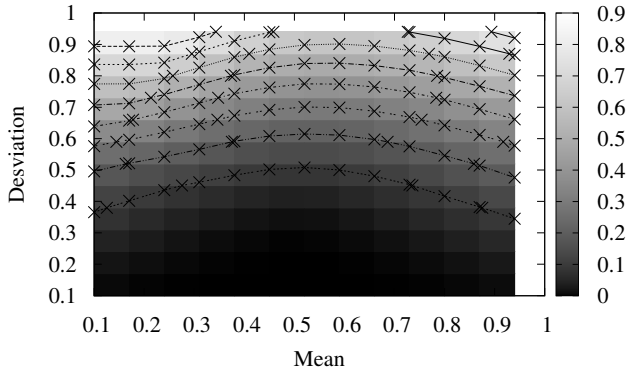


Figura 2.6: Error absoluto para trazas con $H = 0,5$.

2.5. Comparación del método RMD con Perlin Noise

En esta sección se comparan los resultados de trazas generadas con el algoritmo de Perlin Noise con otras obtenidas usando el algoritmo RMD, [LEW+95]. Este algoritmo es un método aproximado conocido por la precisión para obtener trazas FGN de una manera eficiente. Nuevamente, la capacidad del canal ha sido normalizada a $C = 1$.

En la tabla 2.2 se muestra un ejemplo de trazas comparadas en las que se buscaba unos determinados valores objetivo μ, σ^2 y H. Estos valores están dentro de la zona segura definida en la sección previa. Se puede apreciar como las trazas generadas obtienen valores cercanos a los objetivos. La varianza es ligeramente inferior a la esperada debido a que el límite del canal reduce el rango de los posibles valores que podría tener. El valor H es obtenido mediante el estimador de agregación de la varianza. En el caso de la traza de Perlin Noise, el número de octavas utilizado fue $n = 6$ con un valor de persistencia $p = 1{,}4$ para lograr un parámetro Hurst $H \simeq 0{,}7$.

Con ambos métodos se obtienen resultados suficientemente precisos a los valores objetivos, los errores relativos están por debajo del 3.5 %, tabla 2.3.

Tabla 2.2: Comparativa de los parámetros obtenidos utilizando los generadores de Perlin Noise y RMD para una capacidad limitada, $C = 1$.

	Objetivo	Perlin generado	RMD generado
μ	0.380	0.380	0.381
σ^2	0.096	0.0832	0.0927
H	0.7	0.732	0.724

Tabla 2.3: Errores absolutos de las trazas obtenidas usando Perlin Noise y FGN para una capacidad limitada.

	Perlin Trace	FGN Trace
μ	0 %	0.1 %
σ^2	1.2 %	0.33 %
H	3.2 %	2.4 %

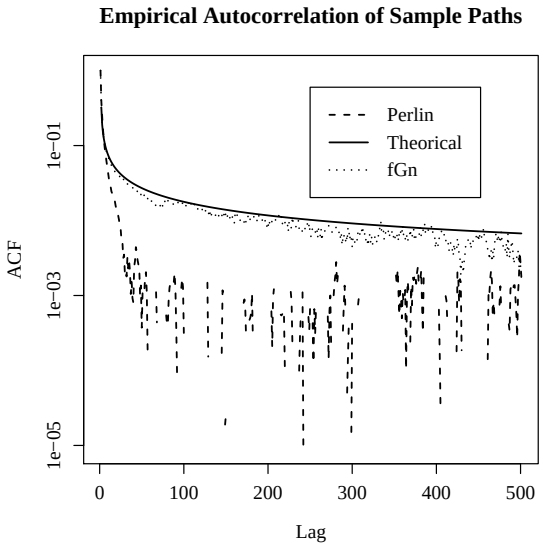


Figura 2.7: Analítica de la curva de autocorrelación con $H = 0{,}7$.

Para comprobar las propiedades de auto-similitud y LRD de la traza obtenida con el algoritmo de Perlin Noise, se comparan la función de autocorrelación y la densidad espectral con la obtenida mediante el proceso de RMD. El la figura 2.7 se muestra la función de autocorrelación junto a la teórica para un proceso FGN. La función teórica puede ser calculada mediante $\rho(k) = 1/2[(k+1)^{2H} - 2k^{2H} + (k-1)^{2H}]$, [Ber94]. La función de autocorrelación decae despacio con una cadencia k mostrando $1/2 < H < 1$. La traza obtenida mediante Perlin se desvía del RMD generado, pero su pendiente, $k \to \infty$, muestra también un comportamiento LRD.

La función espectral para ambas trazas es mostrada en la figura 2.8 junto a la mejor aproximación para la densidad espectral de un proceso autosimilar [Ber94]. Se muestran valores H similiares a los obtenidos con el estimador de la varianza agregada. Además la densidad espectral de la traza generada con el algoritmo de Perlin Noise es la de un proceso auto-similar con el valor de H buscado.

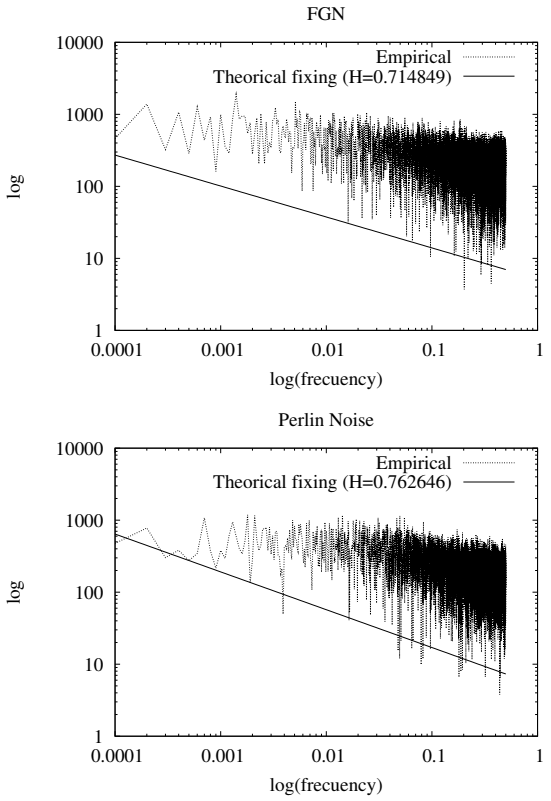


Figura 2.8: Densidad espectral de las trazas de FGN, Perlin Noise con $H = 0{,}7$ y su aproximación.

En las gráficas anteriores se aprecia como las trazas generadas mediante el algoritmo descrito obtienen parámetros comparables con los obtenidos mediante el proceso RMD, por lo que pueden ser utilizadas para generar series continuas de valores sin tener que crear de antemano bloques de tamaños fijos.

2.6. Conclusiones

Los modelos de tráfico auto-similares se usan habitualmente con el objetivo de alimentar simulaciones de redes de tráfico de paquetes y simular el comportamiento de usuarios con tráfico real. De entre estos modelos, uno quizás de los más utilizados, por tratarse de un modelo simple, es el FGN. El problema de este modelo es que su obtención es costosa computacionalmente. Algunos métodos descritos en la literatura tratan de alcanzar un compromiso entre precisión y eficiencia para la generación de trazas FGN. El problema de los métodos propuestos es que la longitud de la traza debe de ser decidida de antemano. En este capítulo se ha descrito un generador FGN que puede ser construido como una adaptación del conocido proceso de Perlin Noise, el cual ha sido utilizado habitualmente en la computación gráfica. Esta adaptación ha sido reinterpretada como un modelado de tráfico de red FGN para un canal con una determinada capacidad.

La principal ventaja de este generador basado en Perlin Noise es que el tráfico puede ser generado al vuelo sin necesidad de almancenar previamente trazas auto-similares.

La precisión obtenida para las trazas usando el generador es comparada con las obtenidas con otro generador clásico de FGN, RMD. RMD es conocido por ser rápido, simple y eficiente. Las trazas obtenidas mediante Perlin Noise han demostrado obtener resultados similares a las obtenidas con el generador RMD y los mismos parámetros.

Se ha comprobado como las propiedades de LRD y espectrales de las trazas generadas mediante el algoritmo propuesto son equivalentes a aquellas generadas mediante un proceso RMD y además se ajustan a las características teóricas de un FGN.

Se ha demostrado como el generador de tráfico basado en Perlin Noise puede ser usado como una fuente de tráfico para proveer tráfico de tipo LRD de forma continua y así eliminar la restricción de tener que escoger antes de la simulación el tamaño de traza deseada.

El algoritmo descrito fue publicado y defendido en el Globecom del 2012, [PIM+12].

CAPÍTULO 3

Detección de pérdidas de disponibilidad en servidores TCP mediante análisis pasivo del tráfico

En este capítulo se describe un algoritmo para la detección de interrupciones de servicio, en los servidores TCP, a través del análisis pasivo del tráfico. El algoritmo se basa en contadores obtenidos pasivamente de trazas reales en los que los clientes acceden a ciertos servicios de interés. El método no es invasivo ni se comunica con los servidores que ofrecen los servicios a monitorizar. Se trata de un algoritmo simple que puede ser utilizado como un sistema de monitorización a tiempo real y que no necesita de grandes requisitos a nivel *software* ni *hardware*. Además, no requiere de una instalación distribuida, basta con monitorizar en un punto de la red cerca de los clientes para los que se quieren detectar las interrupciones de servicios.

En las siguientes secciones del capítulo se explican las ventajas de un algoritmo pasivo para la monitorización de servidores, sección 3.1. En la sección 3.2 se describe en qué consiste el algoritmo y los parámetros que utiliza. A continuación, se realiza un estudio de cómo las diferentes interrupciones que sufren los servidores, pueden ser observadas por variaciones a nivel de tráfico de paquetes, sección 3.3. Durante el estudio se prueba el algoritmo en un *testbed* controlado, sección 3.3.1. Posteriormente se muestran los resultados de

las pruebas en las que los usuarios acceden a servicios populares de Internet para un escenario real, la red de la Universidad Pública de Navarra, sección 3.4. En la penúltima sección se analizan los resultados, sección 3.5. El capítulo finaliza con las conclusiones, sección 3.6.

3.1. Introducción

Como ya se ha mencionado con anterioridad, hoy en día el buen rendimiento de muchas empresas se basa en la fiabilidad en el acceso a aplicaciones de red, puesto que es a través de ellas dónde se realizan todas las operaciones. Ya no sólo a nivel de Internet, sino también internamente, los usuarios, tanto clientes como trabajadores, utilizan programas que se centralizan en granjas de servidores. Actualmente, la mayoría de los servicios de red utilizan el protocolo de comunicación *TCP*, [CMT98; YLX+11]. Sin embargo, pese a la importancia y el desarrollo que han alcanzado estos servicios, en ocasiones no se puede evitar que los clientes de la red no logren tener acceso a los mismos. Estas interrupciones pueden ser debidas a numerosas razones, desde errores humanos, a congestión en la red, actualizaciones, problemas de enrutamientos, etc.

Estos intervalos de tiempo sin servicio, aún siendo pequeños, incluso aunque se trate sólo de unos pocos minutos, pueden ser críticos. Por ejemplo, para una empresa que ofrezca sus productos a los clientes a través de servicios web, interrupciones en el servicio pueden suponer pérdidas de ventas durante dichos periodos así como futuras, debido a la desconfianza que puede suscitar en los clientes. Otro ejemplo que muestra la importancia de las interrupciones son los casos de los servidores de actualizaciones de antivirus. Para el caso de bancos u otras organizaciones en las que la seguridad es de vital importancia, una interrupción en los servidores de actualizaciones de antivirus podrían provocar problemas de infecciones.

En la actualidad existen clientes de monitorización para detectar este tipo de incidencias, por ejemplo, Nagios [Nag], Zabbix [Zab], Cacti [Cac], Munin [Mun], etc. Estos sistemas alertan a los administradores de red cuándo un determinado servicio está inoperativo. Sin embargo, pese a que son capaces de detectar este tipo de problemas, presentan una gran desventaja, se basan en comprobaciones

activas tales como *pings Internet Control Message Protocol* (ICMP) o a través de peticiones *Hypertext Transfer Protocol* (HTTP) de una página web.

Las medidas activas pueden suponer un problema en algunos entornos de red, debido por ejemplo a la sobrecarga que presentan o en casos de monitorización de terceras partes que pueden bloquear peticiones periódicas. Además, la monitorización mediante este tipo de peticiones debe de tener en cuenta los posibles cortafuegos o sistemas de detección de intrusión en las rutas intermedias o en los propios servidores, ya que quizás después de cierto tiempo, denieguen las continuas peticiones realizadas por los agentes de monitorización.

Además, se debe de tener en cuenta, que para los casos en los que se requiere detectar problemas en diferentes redes de clientes, al menos un monitor debería de ser instalado en cada subred. De otro modo algunos problemas podrían pasar inadvertidos. La efectividad de los agentes podría verse afectada dependiendo de la localización, como muestran Liu et al. [LHS+06]. Para algunos escenarios grandes, el escalado de la instalación y comprobación de los monitores puede convertirse en inviable. Prueba de ello son los estudios que se encuentran en la literatura acerca de qué estrategias seguir para optimizar la efectividad e instalación en estos casos, [LLH+08].

Otro problema que se debe hacer frente en este tipo de escenarios, es que a mayor número de agentes de monitorización mayor número de medidas y tráfico. Para evitar este crecimiento exponencial se podrían escoger algunos caminos de forma estratégica, como por ejemplo a través de algoritmos predictivos, [CKC05]. Sin embargo, este tipo de estrategias haría más tediosa la instalación y posterior mantenimiento de los agentes, además de no garantizar la total monitorización de la red. En la literatura se pueden encontrar algunos estudios acerca de cómo abordar el problema de instalación distribuida de los agentes de monitorización, [Par97; CCK98; SWW00; Son12; FDM09; TAT11; BR03].

Los agentes de monitorización activos se enfrentan a otro problema, que los clientes acceden a los servidores a través de *proxy-cachés*. En estos casos, al solicitar una petición podría ser respondida por el proxy en lugar de por el servidor, y que los servidores estuvieran realmente fallando. Una solución a nivel de administración sería cambiar el proxy para que no respondiera a los agentes, aunque en algunos casos podría ser inviable si el proxy no es controlado por la misma

entidad que realiza la monitorización. Otra posibilidad sería provocar que las peticiones realizadas por los agentes no fueran guardadas en los proxys aunque tampoco se trata de una solución trivial.

Además de los problemas citados anteriormente, se debería de tener en cuenta que los agentes de monitorización pueden experimentar problemas para alcanzar los servicios, por ejemplo debido a configuraciones de red erróneas, mientras que los clientes reales no están sufriendo incidencias. Este comportamiento generaría alertas de falsos positivos que tendrían que ser analizadas por los administradores de red, con el fin de discernir si se trata de un problema real de los servidores. Aparte de las configuraciones erróneas existen diferentes causas que pueden provocar errores en los agentes, como fallos de memoria o problemas de CPU, sobrecarga en el cliente, etc.

Un aspecto importante a considerar a la hora de instalar este tipo de sistemas es el tiempo de reacción. Es decir, los tiempos mínimos y máximos que son aceptables para detectar una interrupción de servicio. A menores tiempos, mayores niveles de peticiones activas y por lo tanto mayores interferencias con el tráfico normal de los clientes. A mayores tiempos, más se tardará en reaccionar, por lo que los clientes podrían estar más tiempo sin ser atendidos.

Relacionado con el parámetro del tiempo, se ha observado que para el caso de muchas de las interrupciones, algunos servidores, debido a sobrecargas, rechazaban las nuevas conexiones TCP contestando con paquetes de tipo *Reset* (RST) hasta que recuperaban cierta estabilidad. Aunque la mayoría de las veces estos periodos eran del orden de segundos, como ya se ha explicado, incluso esta magnitud puede ser crítica para el correcto funcionamiento de algunos negocios puesto que causarían quejas y mala reputación. En estos casos, no es fácil alcanzar un compromiso en el tiempo de monitorización sin afectar al rendimiento de la red.

La monitorización a través de medidas pasivas solucionaría algunos de los principales problemas citados. Con este propósito se pueden encontrar algunos estudios al respecto. Por ejemplo, Schatzmann et al. [SLK+11] propusieron un método basado en el análisis pasivo de tráfico capturando trazas en diferentes rutas. Aunque su método era capaz de trabajar en tiempo real, necesitaba de la captura en diferentes puntos de la red y la instalación de este tipo de medidas

en la red no es una tarea trivial, [HCT+14].

Debido a los problemas planteados, el objetivo de este capítulo es estudiar la influencia de las interrupciones en el tráfico a nivel de paquete y el análisis pasivamente. Mediante el estudio ha sido posible la elaboración de un algoritmo simple que permite detectar interrupciones de servicio para los servidores TCP. El algoritmo propuesto se basa en contadores sencillos a nivel de paquetes, como por ejemplo el número de paquetes con el flag activo RST, enviados por los servidores y el número de paquetes de datos recibidos en los clientes. Las principales ventajas del algoritmo es que no requiere de grandes requisitos de memoria ni de CPU para ser instalados en un ordenador que hará las funciones de agente de monitorización. Además, no requiere una instalación distribuida para el análisis de diferentes redes. Bastará hacerlo en puntos troncales en los que se reciba el tráfico de los clientes de las redes. El tiempo mínimo para advertir un periodo de interrupción será un argumento de entrada en el algoritmo por lo que su valor podrá ser modificado en cada ejecución.

3.2. Propuesta de algoritmo de detección de fallos de disponibilidad

Como ya se ha citado en la introducción, el método se basa en la captura y análisis pasivo de tráfico. La idea es utilizar una captura de tráfico de una determinada red cerca de los clientes con el fin de detectar cuándo algunos servicios no estén disponibles.

El objetivo del algoritmo propuesto es encontrar eventos de interrupciones para determinados servicios, es decir, que los clientes monitorizados no puedan hacer un uso satisfactorio de los servicios por la razón que sea, caídas en la red, en los servidores, etc. Se pretende detectar las no disponibilidades locales, que sólo afecten a una subred, por el administrador de red en lugar de detectar un cambio global en el servidor. Además se debe de tener en cuenta que en algunos casos, los clientes pueden ser capaces de alcanzar los servidores pero no el servicio. Por ejemplo, en el caso de un servidor Web caído, los clientes podrían intentar establecer conexiones, a nivel de red no habría ningún problema, pero el servidor les respondería con paquetes de tipo RST. Los paquetes RST, son

paquetes del protocolo TCP que tienen el flag RST activado. Estos paquetes son usados por los extremos de la conexión TCP para rechazar conexiones entrantes y para abortar conexiones establecidas en curso, con el fin de señalizar al otro extremo que debería abortar la conexión.

El algoritmo propuesto consiste en observar un flujo de tráfico, de forma pasiva, el cuál habrá sido filtrado para los servidores de interés. La traza para ser analizada se divide en intervalos fijos de tiempo, por ejemplo cada 10 segundos. En cada intervalo el tráfico es analizado evaluándose una serie de contadores simples. Los contadores usados son el número de paquetes de datos y paquetes RST, enviados entre un grupo de clientes y los servidores para los que el servicio está siendo analizado. Si se observa durante un intervalo de tiempo, aunque éste sea muy pequeño, que los servidores mandan sólo paquetes de tipo RST y no otros tipos de paquetes válidos a un grupo de clientes, eso es un claro indicador de interrupción del servicio. Aunque a veces los servidores puedan terminar las conexiones con los clientes de una forma inesperada, por ejemplo, mediante el envío de paquetes con el flag activo RST como contestación al paquete FIN enviado por los clientes, el algoritmo no mostrará falsos positivos durante dichos intervalos, ya que posiblemente durante el mismo periodo habrá otros clientes enviando y recibiendo tráfico. Para evitar falsos positivos además, si los contadores de cada intervalo muestran valores sospechosos son relacionados con los valores del anterior intervalo para distinguir si se trata de un periodo esporádico o si es algo que continúa en el tiempo.

Cuando un cliente envía paquetes al servidor y no se observa ningún paquete de vuelta enviado por éste, ese intervalo será objeto de sospecha. Si además, en los siguientes segundos el servidor no envía nada excepto quizás paquetes de tipo RST, se confirmará que dicho servicio no está funcionando correctamente. Esta identificación puede ser descrita mediante dos filtros simples aplicados a cada intervalo de tiempo. Para cada intervalo los contadores para los clientes y servidores son actualizados y analizados de forma que se relacionen con lo que ocurría en el intervalo anterior. En el lado de los clientes, el único contador que se tiene en cuenta es el número de paquetes enviados hacia los servidores, sin importar si estos paquetes eran de datos o no, es etiquetado como $packet\_cli$. Por otro lado, para el caso del lado del servidor, se tienen en cuenta

dos contadores: El número de paquetes de datos enviados, $bytes\_servers$, y el número de paquetes con el flag activo RST, $reset\_packets$.

Si durante un periodo se observa que el cliente mandaba datos pero los servidores no enviaban ningún paquete de datos, o incluso sólo se enviaban paquetes RST, el resultado de aplicar el primer filtro a dicho intervalo será 1. El resultado será también positivo, 1, en el caso que no se vean paquetes de clientes y los únicos vistos por el servidor sean paquetes de tipo RST. Este caso podría describir escenarios en los que previamente los clientes intentaron acceder pero, al no obtener respuesta dejaron de enviar paquetes y los únicos paquetes posteriores observados son los enviados por los servidores como respuesta a los enviados por los clientes y que tendrán el flag RST activado para denegar las conexiones.

El segundo filtro será 1 siempre y cuando el resultado del primer filtro del intervalo siendo analizado sea 1 y además el resultado del primer filtro del intervalo anterior sea también 1. Este proceso puede ser descrito de forma matemática mediante el empleo de dos funciones de pertenencia, como las usadas en lógica fuzzy [KY95], para cada intervalo. A continuación se describen las variables utilizadas:

- x= Número de paquetes de clientes enviados en un intervalo.
- y= Bytes enviados por el servidor en un intervalo.
- z= Número de paquetes de tipo RST enviados por los servidores en un intervalo.
- i = i^{th} Intervalo que va a ser analizado.
- $\psi_i(x, y, z)$= Primera fase del filtro compuesto que es aplicado a cada intervalo i.
- $\varphi_i(\psi_i, \psi_{i-1})$= Segunda fase del filtro compuesto aplicado para cada intervalo i teniendo en cuenta el resultado del primer filtro.

Las dos funciones de pertenencia pueden ser descritas mediante la ecuación 3.1.

$$\psi_i(x,y,z) = \begin{cases} 1 & if \quad ((x>0) \quad and \quad (y=0)) \quad or \\ & \quad ((x=0) \quad and \quad (y=0) \quad and \\ & \quad\quad (z>0)) \\ 0 & Otros \quad casos \end{cases}$$

$$\varphi_i(\psi_i,\psi_{i-1}) = \begin{cases} 1 & if \quad (\psi_i=1) \quad and \quad (\psi_{i-1}=1) \\ 0 & Otros \quad casos \end{cases} \tag{3.1}$$

Cada periodo es etiquetado con el resultado de aplicar las dos funciones de pertenencia, $(\psi_i(x,y,z), \varphi_i(\psi_i,\psi_{i-1}))$. Cuando ambos resultados son 1 se considera que el periodo tiene un problema de disponibilidad durante ambos intervalos. El periodo de interrupción se define entones desde el primer intervalo etiquetado cómo $(1,1)$ hasta el siguiente intervalo fuera de toda sospecha, que estará etiquetado como $(0,0)$. En la tabla 3.1 se muestra un ejemplo real en el que se aplicó el algoritmo. En el ejemplo mostrado se analiza una traza de clientes accediendo a servidores de Hotmail durante el día $8/11/2013$. El intervalo de tiempo para la detección de las interrupciones del servicio era de 5 segundos.

Tabla 3.1: Ejemplo del algoritmo de fase doble para algunos intervalos del día $2013/11/8$ analizando servidores de Hotmail.

Inicio	Fin	Bytes Serv (x)	RST Serv (z)	Paquetes Cli (y)	ψ	φ
9:24:55	9:25:00	7016	0	14473	0	0
9:25:00	9:25:05	0	0	1	1	0
9:25:05	9:25:10	0	8	0	1	1
9:25:10	9:25:15	1699	1	3288	0	0

En el segundo intervalo de la tabla 3.1, se observa como algún cliente había mandado un paquete sin obtener respuesta alguna por parte de los servidores, por esta razón el primer flag del algoritmo es 1. Sin embargo, como en el periodo anterior no se observaba ninguna situación fuera de lo normal, el segundo flag es 0. En el siguiente intervalo sólo se observaban paquetes de tipo RST enviados por los servidores de Hotmail. Como los únicos paquetes enviados por los

servidores son de tipo RST el primer flag para dicho intervalo será también 1. Además, como en el caso anterior el primer filtro también era 1, este intervalo se etiqueta como $(1, 1)$. En los siguientes 5 segundos, es decir para el último intervalo mostrado en la tabla, parece que el servicio había recuperado su normal funcionamiento ya que se observaban paquetes de datos por parte de los servidores, por tanto el primer flag es 0 por lo que el segundo flag será también 0.

Un análisis de la interrupción del servicio durante tan pocos segundos mostraba como los servidores estaban cerrando conexiones que habían permanecido inactivas durante más de 30 segundos. Para cerrar dichas conexiones se enviaban paquetes con el flag RST activo. Los paquetes RST no se correspondían a respuestas de paquetes de clientes previos, parece razonable que el servicio estuviera experimentando problemas y por ellos se cerraran conexiones ya iniciadas, este tipo de interrupciones son las que el algoritmo pretende discernir.

El principal parámetro del algoritmo es la duración del intervalo, que podrá ser escogido por el administrador de red dependiendo del tiempo de reacción deseado. Valores pequeños incrementarán la resolución del algoritmo siendo capaces de detectar microfallos, pero por contra incrementarán los falsos positivos. Por nuestra experiencia, se alcanza un buen compromiso para valores entre 5 y 15 segundos.

3.3. Análisis del efecto de las interrupciones en el tráfico observado

En la introducción de este capítulo se describían algunos de los problemas que podían afectar al servicio de los clientes de una red. En esta sección se analizan algunos de los problemas de conectividad más típicos, así como si tienen una repercusión directa en el tráfico a nivel de paquete.

Para la realización de los experimentos se ha utilizado un *testbed* en lugar de un escenario real ya que el escenario controlado permite verificar el motivo por el cual se produjo cada una de las interrupciones que se detecten para cada servicio.

En el *testbed* se han creado interrupciones de forma intencionada y controlada. La duración y las condiciones para dichas interrupciones pueden ser decididas previamente, para después de ese tiempo, durante el cual los clientes no logran acceder normalmente al servicio, el problema sea solucionado. Estos periodos son etiquetados y comparados con los valores obtenidos al aplicar el algoritmo sobre el tráfico capturado.

En las siguientes subsecciones se describe el escenario del *testbed*, los efectos de las interrupciones en el tráfico capturado y cómo el algoritmo detecta estos intervalos.

3.3.1. Escenario controlado

El *testbed* consiste en dos redes de clientes donde varios agentes solicitan constantemente una página web de un servidor. Por simplificar solamente se utiliza un servidor para proveer la página web, pero se podría extender fácilmente con más. En el servidor se ha instalado Apache[1], versión 2.4.7 (Ubuntu), para dar servicio a los clientes. Ambas redes de clientes están conectadas al servidor por dos routers, figura 3.1. Estos routers además juegan el papel de *sniffers* capturando todo el tráfico entre cada red de clientes y el servidor. Todo el escenario fue emulado usando un entorno virtual mediante VirtualBox[2], versión 4.3.34_Ubuntu. Ambas redes de clientes tenían una velocidad de 10MBps con una tasa de pérdida del 1 %. El propósito de la tasa de pérdidas es dar realismo al entorno de emulación. El escenario completo es descrito en la figura 3.1.

Como la cantidad de tráfico puede influir en el tiempo que los intervalos sin servicio so detectados por el algoritmo, los tiempos de llegadas de las peticiones de los clientes siguen una distribución exponencial cuya carga media era diferente para ambas redes. La red 1, tenía una media de carga mucho mayor que la red 2, concretamente de 8 y 3 Mbps respectivamente, tabla 3.2.

Pasando ya a describir los problemas que causan interrupciones de los servicios, uno de los más frecuentes que sufren las redes es debido a la caída de enlaces. En algunos casos un enlace entre la red del cliente y el servidor se cae debido a problemas *hardware* en algún interfaz o bien porque la ruta está siendo

[1] https://httpd.apache.org/
[2] https://www.virtualbox.org/

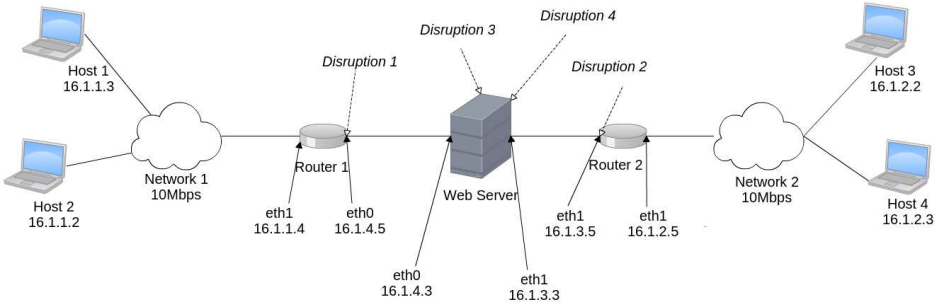


Figura 3.1: Escenario de prueba usado para el estudio del efecto de las diferentes interrupciones

cambiada. Este problema de inalcanzabilidad ha sido emulado en el *testbed* apagando los interfaces de salida en los respectivos routers, en diferentes instantes. Después de un corto periodo de tiempo los interfaces fueron levantados nuevamente. Estas interrupciones han sido etiquetadas como *Interrupción 1*, para el caso en el que la interfaz de salida del router 1 fue apagada mientras que el resto de la red funcionaba correctamente, e *Interrupción 2*, cuando la interfaz del router 2 fue apagada.

Tabla 3.2: Características de las redes.

	Red 1	Red 2
Clientes	2	2
Carga media	8 Mbps	3 Mbps
Pérdidas (%)	1 %	1 %

Otro problema usual en los servicios es debido a que las máquinas son reiniciadas. El reinicio puede ser intencionado, por ejemplo para actualizar un sistema, por problemas *hardware* o simplemente porque no está funcionando adecuadamente. Incluso si los clientes acceden a un servicio que está repartido en un conjunto de máquinas, algunos clientes particulares podrían verse afectados por una única máquina que está siendo reiniciada mientras que el tráfico no sea redirigido hacia otras. En el escenario, como únicamente se emula un único servicio corriendo en una única máquina, dicho reinicio afectaba igualmente a las dos redes. El reinicio se realizó mediante línea de comandos apuntándose

el tiempo que tardó en recuperar el funcionamiento normal. Nos referiremos a esta interrupción como *Interrupción 3*.

Para terminar, se emuló un último problema habitual, que repentinamente se deje de dar servicio debido a un fallo en el propio servicio. Este tipo de problemas pueden ser causados por fallos de configuración o cambios en los servicios, por ejemplo cambio de un puerto de servidor, o por problemas *software*. Se diferencian de los anteriores porque en estos casos los clientes son capaces de llegar a los servidores, sin embargo los servidores no responderán a sus peticiones. En este caso los clientes podrían mandar peticiones *pings* a los servidores y éstos les contestarían, sin embargo no lograrían establecer una conexión web con el servidor. Este problema ha sido emulado mediante una parada de algunos minutos del servicio de apache en el servidor. Ha sido etiquetada como *Interrupción 4* y nuevamente afectó por igual a ambas redes de clientes.

El resumen de las interrupciones emuladas se recoge en la tabla 3.3. El tiempo estimado sin servicio recogidos en las tablas corresponde al tiempo que fue anotado manualmente cuándo comenzaba la interrupción. El tiempo de recuperación corresponde al tiempo anotado manualmente en el que el servicio fue restablecido. Estos tiempos son aproximados, a pesar de que fueron tomados cuidadosamente, ya que fueron escritos por un observador humano.

Tabla 3.3: Interrupciones programadas en el *testbed*.

Nombre	Problema	Red 1	Red 2	Inicio	Recuperación
Interrupción 1	Red	No afectada	Afectada	17:45	17:47
Interrupción 2	Red	Afectada	No afectada	17:56	17:59
Interrupción 3	Servidor	Afectada	Afectada	18:10	18:13
Interrupción 4	Servicio	Afectada	Afectada	10:03	10:05

3.3.2. Efectos en el tráfico capturado

Los diferentes problemas emulados tienen diferentes repercusiones en el tráfico observado. De hecho, para los casos de las interrupciones de la 1 a la 3, los paquetes enviados por los clientes no son capaces de alcanzar el servidor por lo que no se recibirá ningún paquete TCP por parte del servidor. En el último

caso, debido a que se trata de un fallo de servicio, los clientes alcanzan el servidor pero todavía no son capaces de establecer las conexiones TCP. De hecho, el servidor probablemente contestará con paquetes con el flag RST activado.

El tráfico ha sido analizado a nivel de paquete para cada interrupción emulada. Para ello se analizan los paquetes cercanos en el tiempo en el que fue apuntado el comienzo de la interrupción. Como el algoritmo sólo tiene en cuenta el tráfico TCP, sólo este tráfico es monitorizado, filtrándolo antes de analizarlo y aligerando así la carga de tráfico.

Analizando el tráfico se observa como para los dos primeros casos, interrupciones 1 y 2, los clientes dejan de recibir paquetes del servidor de forma inesperada. Además, se puede observar un gran número de intentos nuevos de conexión para los que no se recibe respuesta. Los intentos son localizados a través de paquetes con el flag SYN activado. Ambas redes se comportan de forma idéntica para las interrupciones de red, 1 y 2, por ello a continuación se muestra un ejemplo de algunos paquetes sólo para la red 1:

```
17:56:31 IP 16.1.1.2.37011 > 16.1.4.3.80: Flags [S],
17:56:33 IP 16.1.1.3.41460 > 16.1.4.3.80: Flags [S],
17:56:37 IP 16.1.1.2.37012 > 16.1.4.3.80: Flags [S],
17:56:41 IP 16.1.1.2.37013 > 16.1.4.3.80: Flags [S],
17:56:41 IP 16.1.1.3.41461 > 16.1.4.3.80: Flags [S],
17:56:42 IP 16.1.1.2.37014 > 16.1.4.3.80: Flags [S],
17:56:44 IP 16.1.1.3.41462 > 16.1.4.3.80: Flags [S],
```

El reinicio del servidor también tiene su efecto si se observa el tráfico a nivel de paquete. De nuevo, se ha estudiado el tráfico cerca de la marca de inicio de la interrupción 3 para ambas redes. Similarmente al caso anterior, en un momento dado los clientes no reciben ningún otro paquete del servidor y las nuevas peticiones no encuentran ninguna respuesta por parte del servidor. De nuevo durante dicho intervalo se observan un gran número de intentos de conexión:

```
18:10:14 IP 16.1.1.2.37137 > 16.1.4.3.80: Flags [S],
18:10:15 IP 16.1.1.2.37137 > 16.1.4.3.80: Flags [S],
```

```
18:10:17 IP 16.1.1.2.37137 > 16.1.4.3.80: Flags [S],
18:10:21 IP 16.1.1.2.37138 > 16.1.4.3.80: Flags [S],
18:10:21 IP 16.1.1.2.37137 > 16.1.4.3.80: Flags [S],
18:10:21 IP 16.1.1.3.41681 > 16.1.4.3.80: Flags [S],
```

El último problema analizado, interrupción 4, presenta una importante diferencia en relación al tráfico analizado a nivel de paquete con respecto a las anteriores interrupciones, de la 1 a la 3. Mientras que para las anteriores los clientes no recibían ningún paquete por parte del servidor, en el último, debido a que son capaces de llegar hasta éste, el servidor les responde con paquetes TCP con el flag RST activado denegándoles por tanto las conexiones. Este último caso puede pasar inadvertido si se emplean otros servicios de monitorización, por ejemplo mediante pings. El servidor contestaría al *ping*, puesto que ni a nivel de red ni *hardware* presenta ningún problema, sin embargo no daría servicio a los clientes. El efecto de este problema a nivel de paquete se traduce en que aunque se observan paquetes de datos de clientes, aquellos que todavía no se les ha mandado o no han recibido un *Reset*, por parte del servidor sólo se mandan paquetes RST. Pasado cierto tiempo sólo se observan intentos de conexión que son rechazados por el servidor:

```
10:04:50 IP 16.1.4.3.80 > 16.1.1.3.59757: Flags [P.],
10:04:50 IP 16.1.1.3.59757 > 16.1.4.3.80: Flags [.],
10:04:50 IP 16.1.1.3.59757 > 16.1.4.3.80: Flags [R.],
10:04:55 IP 16.1.1.2.47925 > 16.1.4.3.80: Flags [S],
10:04:55 IP 16.1.4.3.80 > 16.1.1.2.47925: Flags [R.],
10:04:55 IP 16.1.1.3.59758 > 16.1.4.3.80: Flags [S],
10:04:55 IP 16.1.4.3.80 > 16.1.1.3.59758: Flags [R.],
10:04:57 IP 16.1.1.2.47926 > 16.1.4.3.80: Flags [S],
```

Los casos emulados muestran cómo las interrupciones pueden ser descritas mediante las dos funciones de pertenencia del algoritmo. Durante los intervalos de tiempo con problemas los clientes reciben poco o ningún tráfico por parte del servidor, pudiendo terminar por no recibir nada o bien paquetes de tipo RST. Una vez que el problema es solventado, el servidor volverá a mandar paquetes

de datos al cliente, por lo que las funciones de pertenencia identificarán que el servicio ya se ha recuperado.

3.3.3. Detectando las interrupciones

En este apartado se aplica el algoritmo al tráfico capturado durante las interrupciones emuladas. Un parámetro importante que se debe de escoger para el empleo del algoritmo es el tiempo que definirá la duración de los intervalos. Como ya se ha mencionado anteriormente valores grandes pasarían por alto microfallos en los servicios, mientras que valores muy pequeños incrementarían falsos positivos si por ejemplo, son tan pequeños que sólo se ve un paquete de cliente pero la respuesta del servidor ya entra en otro intervalo. El valor escogido para el *testbed* fue de 10 segundos. El objetivo era comprobar si este tiempo puede servir para detectar los periodos de interrupciones en ambas redes de clientes, una que tiene bastante tráfico y una con poco.

El resultado de utilizar el algoritmo sobre el tráfico capturado se muestra en la figura 3.2. Se puede apreciar como los periodos detectados se corresponden bastante exactamente con el tiempo estimado para el inicio y fin de las interrupciones emuladas, las cuales se recogían en la tabla 3.3. Cabe destacar que no se observa ningún intervalo que sea un falso positivo, es decir que el algoritmo estime que se trate de una interrupción y no lo sea. Tampoco se observaron casos de falsos negativos, es decir que una interrupción no fuera detectada.

A nivel de volumen de tráfico, las interrupciones también pueden ser observadas, figura 3.3. Para los problemas en los que los clientes no son capaces de alcanzar el servidor, interrupciones 1 a la 3, se observa ausencia del tráfico del servidor, tráfico *Downstream*, mientras que para el cliente, tráfico *Upstream*, se sigue viendo algo de tráfico causado por las nuevas peticiones de establecimiento de conexión.

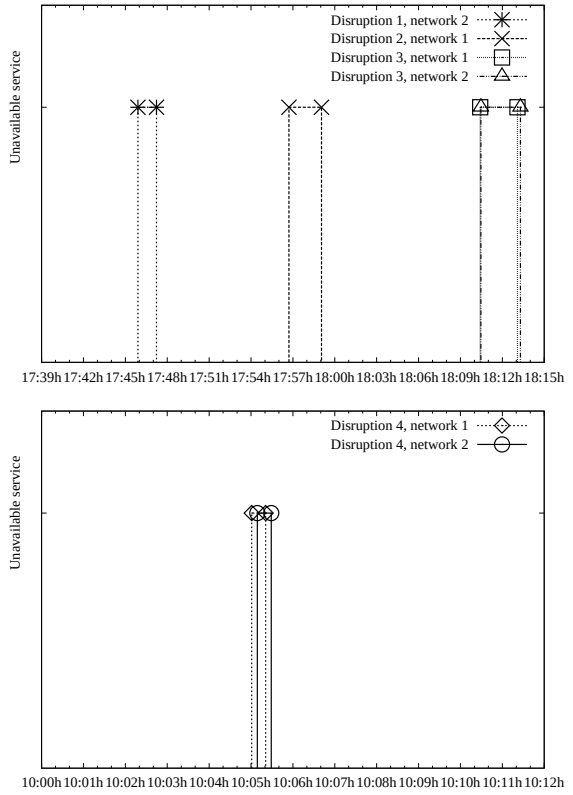


Figura 3.2: Periodos de servicios no disponibles detectados en el *tesbed*.

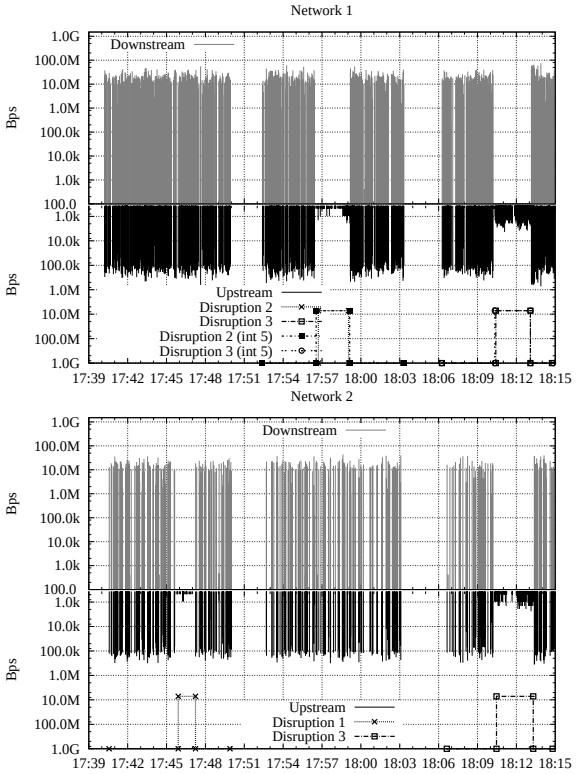


Figura 3.3: Bps para el uso del servicio Web por clientes del *tesbed* y para las interrupciones del 1 al 3.

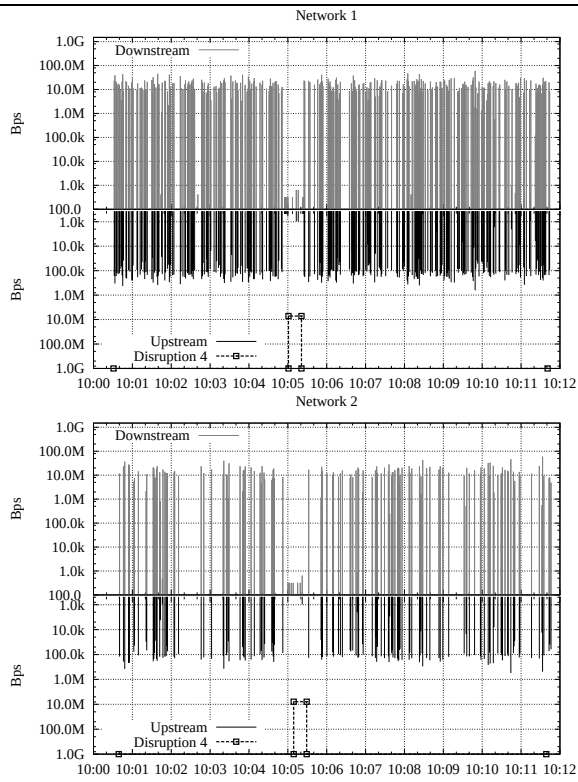


Figura 3.4: Bps para el uso del servicio Web por clientes del *tesbed* para la interrupción 4.

Sin embargo, a nivel de volumen de tráfico una identificación de interrupciones sería desaconsejable debido a que en algunos casos, como para la interrupción 4, no se observa la ausencia del tráfico enviado por parte del servidor. En estos casos el servidor contesta a los clientes mediante el envío de paquetes RST, y aunque esto tiene su efecto a nivel de volumen y se observa un decremento del tráfico, figura 3.4, sería muy difícil diferenciar estos intervalos de periodos en los que los clientes y servidores están intercambiando pocos datos o bien hay pocos clientes conectados.

El volumen del tráfico de las redes también influye a la hora de detectar los problemas. En el caso de la segunda red, para la que se tiene un tráfico menor, los intervalos se detectaban un poco más tarde de lo que estaba apuntado y para

el caso de que afectaba a las dos redes, más tarde que la primera, figura 3.4. Este efecto es debido a que en la segunda red los clientes no solicitan la página tan frecuentemente, y como el problema se detecta cuando intentan establecer la conexión con el servidor, lo cuál es más tarde que en el caso de la primera red, el intervalo de inicio es ligeramente posterior. De la misma manera, ocurre lo mismo para la recuperación de la red. Como en el caso de la segunda red no se solicita tan frecuentemente, la recuperación no se observa hasta que las primeras peticiones de los clientes son atendidas de nuevo. Sin embargo, estas diferencias de tiempo entre ambas redes son mínimas como se puede aprecair en la tabla 3.4.

Tabla 3.4: Comparación de los periodos de no disponibilidad detectados utilizando dos tiempos de intervalos, 10s y 5s.

Red	Interrupción	Periodo estimado	Intervalo 10s	Intervalo 5s
Red 1	2	17:56 - 17:59	17:56:33 - 17:59:03	17:56:23 - 17:59:03
	3	18:10 - 18:13	18:10:15 - 18:13:05	18:10:10 - 18:13:00
	4	10:03 - 10:05	10:04:51 - 10:04:51	10:04:46 - 10:05:16
Red 2	1	17:45 - 17:47	17:45:44 - 17:47:14	17:45:39 - 17:47:09
	3	18:10 - 18:13	18:10:18 - 18:13:18	18:10:13 - 18:13:13
	4	10:03 - 10:05	10:04:59 - 10:05:29	10:04:54 - 10:05:24

Otro problema que se ha tenido en cuenta en este apartado es el análisis del impacto del valor de tiempo escogido para definir los intervalos en el algoritmo. Se ha comprobado como para un tiempo de 10s los resultados eran bastante exactos. Sin embargo, para comprobar el efecto del tiempo, el algoritmo es aplicado usando un intervalo más pequeño, de 5s. Como se ha citado anteriormente, intervalos de tiempo muy pequeños pueden incrementar el número de falsos positivos, sin embargo, en el caso del *tesbed* los intervalos detectados eran los mismos que empleando un tiempo de 10s, si bien los tiempos de inicio y fin de cada intervalo variaban un poco. La principal diferencia en este escenario entre los dos tiempos es que empleando un intervalo de 5s se detectaban los problemas un poco antes, tabla 3.4.

A pesar de que el tiempo de 5s se comportaba bien para el *tesbed*, en un escenario real donde el tráfico sea más rafagoso, o incluso casi o totalmente inexistente a ciertas horas, por ejemplo durante las noches, recomendamos que

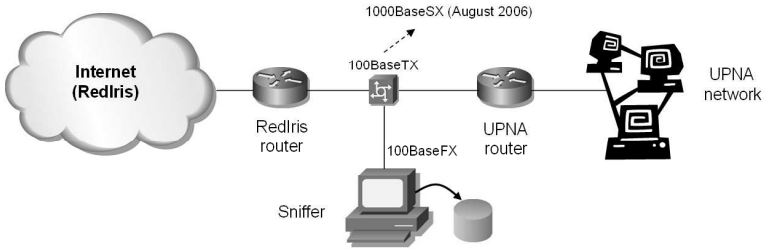


Figura 3.5: Tráfico capturado en el enlace de la Universidad Pública de Navarra

se usen valores de 10s en lugar de 5s. Este valor decrementará el posible número de falsos positivos y todavía será capaz de detectar los periodos pequeños en los que los clientes tengan problemas para utilizar los servicios.

3.4. Experimentos en un escenario real

Una vez que el algoritmo ha sido definido y probado en un escenario controlado, se procede a comprobar su funcionamiento en un escenario real con el fin de detectar los posibles problemas reales de clientes. Concretamente, la comunidad utilizada ha sido los clientes de la red de la Universidad Pública de Navarra accediendo a servicios públicos de Internet. La captura de tráfico proviene de la infraestructura del grupo de investigación, que tiene acceso a un *sniffer* de tráfico con *software* propio y el cual está colocado entre el acceso principal de la universidad y el proveedor académico de Internet (RedIris), figura 3.5. El grupo de investigación cuenta con una colección de trazas capturas a 1 Gbps desde el 2004. Para salvaguardar la confidencialidad de los usuarios sólo se capturan los 100 primeros bytes de cada paquete.

En este capítulo, los resultados se muestran para capturas de datos de la semana del 7 al 11 de Noviembre del 2013. El objetivo es comprobar la accesibilidad de los clientes a servicios populares de Internet tales como, Facebook, Yahoo, BBC y Hotmail. Con el fin de comparar los resultados con algún sistema de monitorización activa, como por ejemplo podría ser Nagios [Nag], se ha implementado un método activo simple. La razón de implementar un método simple, es evitar tanto instalaciones complicadas, como aumentos de la carga de

tráfico. Este método de monitorización activa consiste en la petición del archivo *favicon.ico* para cada servicio web que quiere ser analizado. El resultado de la petición devuelve un icono para ser mostrado en la ventana del navegador y es usado ampliamente por los servidores web. El programa realiza peticiones del favicon.ico cada 5 segundos para cada tipo de servicio considerado en el experimento. De esta forma se tiene una referencia con la que comparar los resultados de aplicar el algoritmo sobre la traza.

Las peticiones activas se realizan desde un ordenador de escritorio de la Universidad Pública de Navarra. El número de servicios contra los que es probado no es muy grande para que el ordenador no soporte altas cargas y no se tengan fallos de peticiones debido a sobrecarga. El algoritmo pasivo trabaja sobre tráfico obtenido en el extremo de la red, como se muestra en la figura 3.5. El algoritmo no ha sido evaluado al vuelo en este trabajo, pero podría ser fácilmente programado para hacerlo.

Como los servicios a ser analizados son muy populares, se pueden encontrar otros fuentes de información acerca de la disponibilidad de los mismos. Más concretamente, varias páginas web muestran intervalos de interrupciones y de quejas de usuarios de estos servicios. El problema es que normalmente la información no tiene la suficiente granularidad para ser utilizada, no se tienen en cuenta intervalos de menos de 10 minutos con problemas.

3.5. Resultados

En esta sección se muestran los resultados obtenidos tanto de la aplicación del algoritmo como de las peticiones del favicon.ico para la semana de tráfico citada anteriormente, del 7 al 11 de Noviembre del 2013. Además de los servicios populares anteriormente citados, “Facebook”, “Yahoo”, “Hotmail” y la “BBC”, se incluye en el análisis un servicio más, se trata de un periódico popular local, “Diario de Navarra”, visitado por la comunidad universitaria. Los servicios analizados, exceptuando el periódico local, son accedidos por una gran masa de usuarios alrededor del mundo por lo que disponen de grupos de direcciones *Internet Protocol* (IP) diferentes. Además probablemente están distribuidos en grandes granjas o redes de distribución de contenidos.

Incluso aunque las granjas estén diseñadas para balancear la carga y soportar picos de demanda, algunas veces, los clientes de la universidad no son capaces de alcanzar dichos servicios. Los resultados del monitor activo que solicita el archivo favicon.ico, muestra algunos intervalos para los que no consiguió el archivo, tabla 3.5. Estos resultados se muestran en la figura 3.6. En ambos se puede apreciar como el servicio con más intervalos detectados fue Hotmail.

Tabla 3.5: Intervalos de servicios no disponibles detectados por las peticiones del favicon.

Inicio	Fin	Día	Servicio
0:15:49	00:16:58	07/11/2013	Facebook
3:10:01	03:10:06	07/11/2013	Facebook
13:36:35	13:36:51	08/11/2013	Hotmail
16:08:12	16:25:40	11/11/2013	Facebook
10:34:59	10:35:21	11/11/2013	Hotmail
10:10:23	10:10:29	13/11/2013	Yahoo
23:08:21	23:08:27	13/11/2013	Yahoo
11:08:54	11:09:06	14/11/2013	Hotmail
11:39:18	11:39:36	14/11/2013	Hotmail
22:43:00	22:43:05	14/11/2013	Hotmail
8:40:31	08:40:44	15/11/2013	Facebook
20:30:03	20:30:13	15/11/2013	Hotmail
4:22:29	04:22:34	15/11/2013	Facebook

Para comprobar el algoritmo propuesto se comienza el análisis aplicándolo a un día completo de tráfico pero filtrando solamente el tráfico para el cliente del ordenador que realiza las pruebas activas, de esta forma se podrá comparar los resultados de ambos programas. Para simplificar en adelante se muestran los resultados para el día 08/11/2013 ya que los resultados son similares para otros días.

Como primer paso el tráfico es filtrado para seleccionar los paquetes del agente de monitorización y de los servidores de interés. Para poder conocer las direcciones IP de los servidores de interés estos han tenido que ser previamente identificados. Para ello, los datos de los paquetes de tráfico han sido examinados para buscar los nombres de los servidores en las peticiones HTTP.

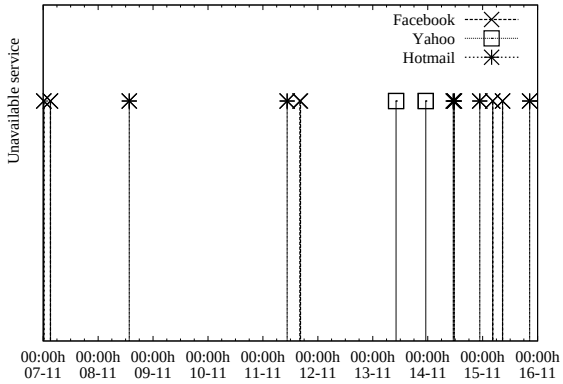


Figura 3.6: Eventos de tiempo para los que el favicon.ico no fue obtenido.

Para ambos métodos, tanto el pasivo como el activo, se encuentran problemas de disponibilidad para el servicio de Hotmail, figura 3.7. En la gráfica se dibuja el volumen de tráfico desde la máquina cliente hacia los servidores de Hotmail así como los eventos de tiempo identificados aplicando ambos métodos, pasivo y activo. Al examinar los paquetes involucrados en los eventos se observaba como esta interrupción era provocada por una única conexión que sufría un inesperado *Reset* por parte del servidor terminando así la conexión. Además en realidad, ambos algoritmos están identificando el mismo evento pero debido a que los relojes de los agentes no estaban sincronizados por *Network Time Protocol* (NTP), en la gráfica aparecen un poco desplazados uno del otro. Este hecho debería de ser tenido en cuenta para sistemas de monitorización distribuida ya que en esos casos la sincronización juega un papel crítico. En este caso, el *sniffer* sólo es un ordenador por lo que el problema de la sincronización se ha simplificado.

El análisis a nivel de paquete del evento detectado, refleja el problema sufrido durante dicha conexión, se muestra a continuación. Las *x.x.x.x* son usadas como dirección IP del cliente, mientras que *y.y.y.y* se utiliza para la dirección IP del servidor de Hotmail. Después de que la conexión hubiera sido establecida, el cliente enviaba la petición del favicon.ico utilizando para ello un paquete de 176 Bytes con el flag Push activado. Normalmente, después de que este paquete fuera enviado el servidor respondía con el fichero favicon.ico. Sin embargo, para

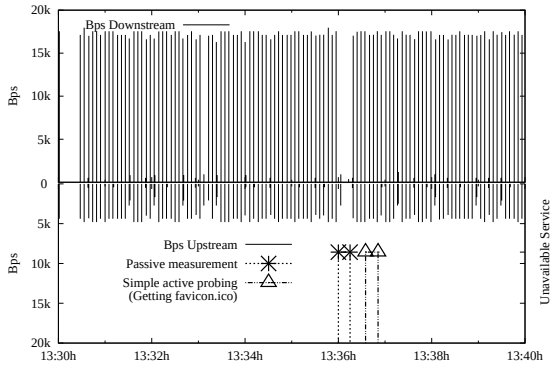


Figura 3.7: Intervalos de tiempo para los que el cliente de monitorización tuvo problemas durante el día 8 de Noviembre del 2013

este evento, el servidor enviaba un paquete de tipo *Acknowledgement* (ACK) sin datos para después de algunos segundos, alrededor de 11, abortar la conexión mediante el envío de un paquete con el flag RST activo. Este tipo de comportamiento es inesperado y durante algunos segundos, los que ha durado la conexión, el cliente se da cuenta de un mal funcionamiento del servicio.

```
13:36:02 IP x.x.x.x.59133 > y.y.y.y.http: S
13:36:02 IP y.y.y.y.http > x.x.x.x.59133: S
13:36:02 IP x.x.x.x.59133 > y.y.y.y.http: . ack 1
13:36:02 IP x.x.x.x.59133 > y.y.y.y.http: P 176
13:36:02 IP y.y.y.y.http > x.x.x.x.59133: . ack 177
13:36:13 IP y.y.y.y.http > x.x.x.x.59133: R 1 ack 177
```

Este mismo comportamiento ha sido observado en otros casos. Para otras interrupciones analizadas, se observaba como antes de que el servidor cesase su servicio dejaba de responder a los clientes para después de algún tiempo, al no reconocer las conexiones que habían sido establecidas con anterioridad, empezar a mandar paquetes con el flag RST activo.

3.5.1. Comparativa entre pruebas activas y análisis pasivo

Una vez comparados los resultados de los dos métodos de monitorización, se amplía el estudio para todo el rango de clientes que realizaban peticiones

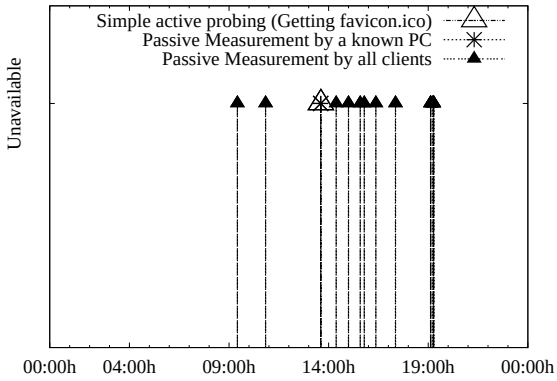


Figura 3.8: Comparativa de los eventos de no disponibilidad detectados para los clientes que accedían al servicio de Hotmail para el 8 de Noviembre del 2013.

a servicios de Hotmail durante el día 8 de Noviembre del 2013. El objetivo es distinguir si el periodo detectado por un sólo cliente es un hecho aislado o si por el contrario otros usuarios experimentaban problemas similares.

Para la realización del estudio previamente se buscan todos los clientes que accedían a Hotmail así como los servidores desde los cuáles se respondía. El algoritmo pasivo se aplica a todo el tráfico agregado de clientes y servidores. El tiempo de duración de los intervalos es fijado en 5 segundos para ser capaces de detectar microfallos como el observado para el cliente que realizaba la monitorización activa previamente explicado. Al agrupar todo el tráfico de todos los clientes fueron detectados nuevos intervalos, figura 3.8. Interesantemente, se detectan más periodos que el detectado utilizando el agente activo.

Además, mediante el empleo del tráfico agregado se observa como el evento anterior resultante de filtrar un único cliente, que además era el agente de monitorización activa, ahora no aparece. Esta desaparición es debida a que durante ese mismo intervalo otros usuarios fueron capaces de acceder a Hotmail sin problemas. Este hecho muestra como se trataba de algo aislado y no de un problema de servicio que afectara a los usuarios. Este ejemplo revela un falso positivo que refleja el peligro de utilizar un único cliente de monitorización para la detección de fallos de servicio.

Este experimento muestra otro hecho importante. Al definir el servicio a través de una agregación de las direcciones IP individuales de los servidores, somos capaces de identificar algunos intervalos de no disponibilidad de unos pocos segundos en los cuales los clientes sufrieron problemas de acceso que pasaron inadvertidos para el cliente activo de monitorización. Durante dichos intervalos, el agente no reflejaba el problema ya que el archivo que solicitaba le era servido por un proxy caché. En la tabla 3.6 se muestran los intervalos detectados utilizando la agregación de clientes y servidores por el método pasivo.

Tabla 3.6: Intervalos para el servicio de Hotmail no disponibles, t=5s.

Inicio	Fin
09:25:05	09:25:15
10:50:00	10:50:10
14:22:40	14:22:50
14:59:40	14:59:50
15:35:00	15:35:10
15:47:15	15:47:25
16:22:05	16:22:15
17:21:10	17:21:30
19:06:50	19:07:00
19:07:20	19:07:35
19:13:35	19:14:05
19:16:10	19:16:30

El análisis a nivel de paquete de los periodos encontrados revela que los problemas eran provocados por el envío de paquetes RST inesperados por parte de los servidores hacia los clientes. Antes del envío de dichos paquetes no se observaba ningún mal comportamiento por parte de los clientes que pudiera provocar estas respuestas por parte de los servidores. Durante esos segundos, repentinamente los servidores decidieron abortar las conexiones establecidas con uno o varios clientes. Como las duraciones de las paradas fue en realidad tan corta, ya que las siguientes conexiones se establecían normalmente, no se considera que se trate de interrupciones críticas. En estos casos interesaría que dichos periodos fueran ignorados para lo que bastaría con incrementar el tiempo de los intervalos en el algoritmo a 10 segundos, por ejemplo.

Tabla 3.7: Intervalos para el servicio de Hotmail no disponibles, t=10s.

Start	End
15:34:50	15:35:10
19:13:40	19:14:10

La tabla 3.7 se muestra los intervalos detectados para el mismo tráfico pero utilizando intervalos de 10s en lugar de 5s. Los casos detectados son reducidos a 2, con duraciones de 20 y 30 segundos respectivamente cada uno. Durante dichos intervalos solamente se observaban paquetes RST enviados por los servidores hacia los clientes, los cuáles previamente habían establecido conexiones normalmente. El resto de intervalos anteriormente detectados ahora pasa desapercibido porque el envío de los paquetes de tipo RST, caen dentro del mismo intervalo en el cual otros clientes están recibiendo datos por parte del servidor. Los intervalos de 10s detectados también empleando un tiempo de 5s, no coinciden de forma exacta debido a la duración de los intervalos y la sincronización de eventos. El máximo error cometido vendrá dado por el mínimo intervalo de tiempo considerado. Por ejemplo, en los ejemplos presentados el error para los intervalos de interrupción sería más o menos de 5s desde el periodo encontrado por el algoritmo. En el caso de emplear un tiempo para definir los intervalos de 10s, este sería el error por encima o por debajo del comienzo del intervalo encontrado por el algoritmo.

Al comprobar el resto de servicios para los que el agente de monitorización activa mostraba algún intervalo de interrupción, se observa como se trataba de un problema puntual de un cliente y no del servicio, ya que durante esos mismos periodos otros clientes accedían normalmente. El problema del cliente podía ser bien debido al *proxy* o bien por problemas con un servidor en concreto, pero no con el servicio.

3.5.2. Perfilado de tráfico de los servicios solicitados

Al igual que se hizo en el *tesbed*, se grafica el volumen de tráfico para los clientes y servidores de los servicios analizados, como otra medida de comprobación. La idea es observar si a nivel de tráfico se observa una disminución del volumen de tráfico por parte del servidor en los intervalos detectados. En la

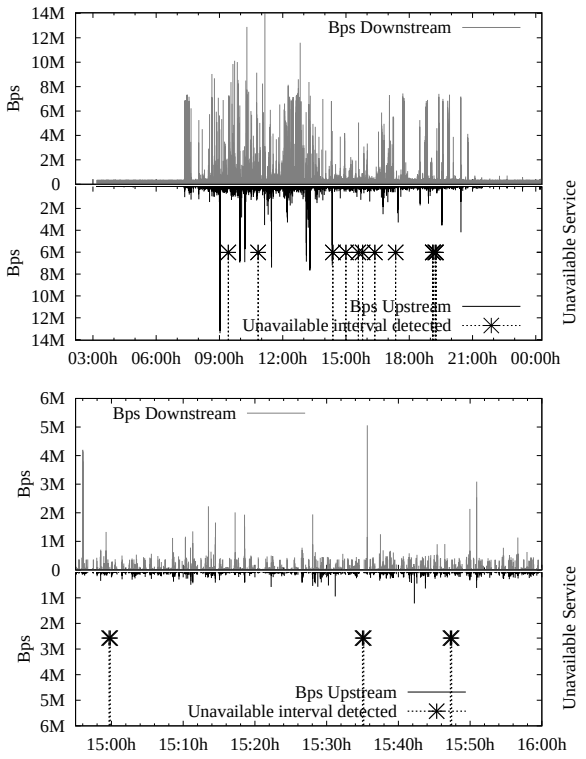


Figura 3.9: Bps del uso del servicio de Hotmail por los miembros de la comunidad universitaria.

figura 3.9 se dibuja el tráfico para el servicio de Hotmail del día 8 de Noviembre, ya que es uno de los servicios con más interrupciones detectadas. La primera imagen de la figura muestra la totalidad del tráfico para ese día, mientras que la segunda corresponde a un zoom del tiempo en el que los principales intervalos fueron detectados. En ambos casos también se grafican los intervalos de interrupciones detectados por el algoritmo utilizando un tiempo para calcular los intervalos de 5s.

En el zoom de la gráfica, la segunda de la figura, se aprecian huecos en el tráfico alrededor de los intervalos. El que más destaca es para el segundo intervalo, en el que se puede observar como el servicio parece estar funcionando hasta aproximádamente las 15:35:00, en las que se observa un lapso que se pro-

longa hasta las 15:35:10. Durante este tiempo no se observa tráfico desde el servidor, y por parte del cliente es tan pequeño que parece inexistente. Después de algunos segundos el servicio parece haberse restablecido normalmente alcanzándose incluso un pico de 5 MBps.

Como ya se mencionó anteriormente, a través del volumen del tráfico se pueden apreciar huecos o disminuciones en los que no se observe tráfico enviado por parte del servidor. Sin embargo, es difícil distinguir este tipo de periodos con momentos de tráfico de baja intensidad. Por ello el perfilado del tráfico se utiliza como una comprobación extra del algoritmo y no como un método de identificación propiamente.

3.6. Conclusiones

Hoy en día, debido a la importancia de que algunos clientes puedan conectarse con los servicios constantemente y a cualquier hora, la monitorización de los servicios se ha convertido en un factor crítico. En muchas redes, bien por problemas de permisos o bien por saturación de las mismas, es desaconsejable una monitorización activa. Por ello, en este capítulo se ha realizado un estudio de un método pasivo capaz de monitorizar servicios distinguiendo cuándo ocurren intervalos de tiempo en los que los clientes presenten problemas con los servidores.

El algoritmo propuesto es simple y se basa en contadores sencillos que son calculados cada cierto intervalo de tiempo. Por ello, puede ser instalado en agentes sin grandes requerimientos *hardware* o *software* salvo quizás buenas tarjetas capaces de monitorizar todo el tráfico de la red.

Otra ventaja de este tipo de medida es que basta con elegir el punto de la red cercano a los clientes, para el que quiera detectarse cuándo un servicio sufre una caída. Este punto suele ser habitualmente el extremo de la red de la organización que quiere ser monitorizada. En algunos sistemas de monitorización activa se requeriría de una sincronización de los relojes entre los agentes de monitorización repartidos entre las subredes a ser analizadas.

El algoritmo realizado podría ser extendido para trabajar de una forma ciega, es decir, sin pasarle un rango de direcciones IP de los servidores y clientes que corresponden a los servicios que quieren ser monitorizados. La idea sería aplicar el algoritmo bien por conexión o bien por grupo de clientes que se conectan a un mismo servidor. De esta forma el algoritmo funcionaría como un sistema de detección de anomalías que advierte al administrador de la red cuando haya posibles problemas en los servicios. Este sistema podría ser útil para organizaciones grandes en las que no se tiene una lista de los servicios accedidos por los usuarios pero que sin embargo, se necesite una pronta respuesta en caso de detectarse problemas de disponibilidad.

El algoritmo también podría ser mejorado para disminuir los casos de falsos positivos, aplicando algún método de inteligencia artificial en lugar de utilizar simplemente las dos funciones de pertenencia, por ejemplo algún método de lógica *fuzzy*.

Sin embargo, debido al interés del grupo de investigación hacia otros temas relacionados también con el análisis del tráfico pasivo, el algoritmo es usado en su versión simple y de momento no se han realizado cambios en el mismo.

El algoritmo desarrollado ha sido publicado y defendido en la conferencia Softeng 2015 obteniendo *best paper award*, [PIM+15c].

Capítulo 4

Método pasivo simple para estimar el RTT en redes con alto bandwidth-delay

En el análisis de tráfico, una de las métricas más importantes a estudiar por los adminstradores de red, con el fin de verificar que todo funciona correctamente, es el *Round Trip Time*, RTT. El RTT indica el tiempo que cuesta obtener una respuesta del otro lado de la comunicación, es un indicador de la latencia de la red. Este valor es muy útil a la hora de discernir problemas en una red, por ejemplo, un incremento instantáneo puede indicar la existencia de un problema de red como por ejemplo, un periodo corto de congestión. Además, este valor puede ser usado para analizar si el rendimiento de un servicio no es el esperado, ya sea debido a la red o bien por problemas de la aplicación o el servidor.

Debido a la importancia de la métrica, se ha considerado interesante el cálculo de la misma desde una perspectiva no intrusiva, a través del análisis del tráfico pasivo. En este capítulo se propone un algoritmo basado en la observación de patrones de RTT para aquellas conexiones TCP que no estén llenando el producto $bandwidth \times delay$. Estos resultados son comparados con otros métodos pasivos, tales como el RTT inicial a partir del *handshake*, o el obtenido por las muestras de la opción de *Timestamp*.

En las siguientes secciones se explica la motivación del estudio 4.1, la propuesta de método para la medición 4.2, los resultados obtenidos en los experimentos 4.4, un caso de estudio en un escenario real 4.5, y por último las conclusiones, 4.6.

4.1. Introducción

El RTT es una métrica importante para el análisis del rendimiento de una red. Este valor habitualmente, se define como la diferencia de tiempo entre el envío de un paquete y la recepción de la respuesta. Esto puede estimarse en el lado del emisor como el tiempo desde que envía un paquete y recibe cualquier paquete que sea consecuencia de que el receptor haya recibido el primero, figura 4.1.

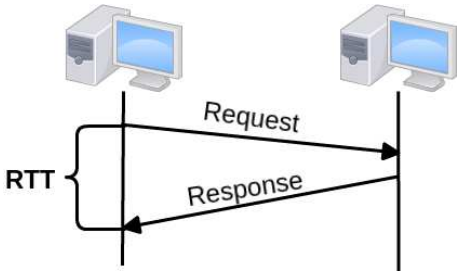


Figura 4.1: Definición del RTT.

Para estimar esto de forma pasiva, se puede medir el tiempo entre la observación de un paquete en una dirección y la observación de un paquete en dirección contraria que sea causado por el primero, preferiblemente sin que pase mucho tiempo desde que el receptor recibe el primero y envíe el paquete como consecuencia de éste. Esta estimación es dependiente de que el punto de observación esté cerca de uno de los dos extremos. Cerca del emisor se mediría algo cercano al RTT, sin embargo, cerca del receptor se mediría un valor mucho mas pequeño. Por ello en una medida pasiva es más conveniente extender el tiempo a un tercer paquete que dependa de la respuesta. De esta forma, se puede medir el RTT como la diferencia de tiempo entre la observación de un paquete en una

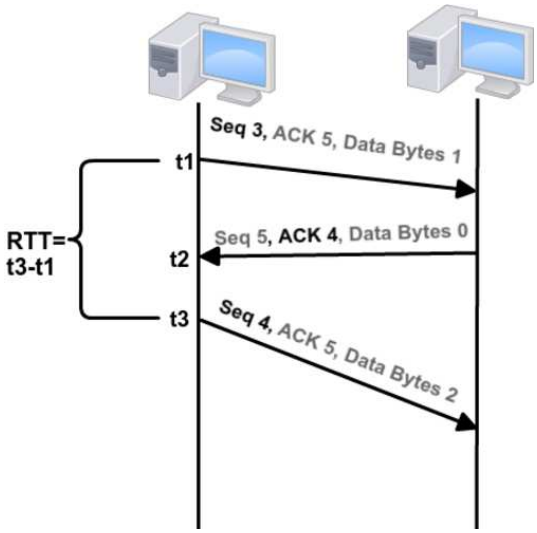


Figura 4.2: Estimación del cálculo del RTT de forma pasiva.

dirección y el siguiente paquete en esa misma dirección, el cual haya sido causado por un paquete de la dirección opuesta, que a la vez dependa del primero, figura 4.2.

Normalmente, la estimación del RTT se obtiene mediante medidas activas, como el Ping. Sin embargo, en determinados escenarios, no se quiere interferir con la actividad de la red por lo que se debe desistimar el uso de éstas. Para el cálculo basado en las observaciones, se debería tener en cuenta que diversos factores pueden afectar al tiempo medido, como por ejemplo las retransmisiones o los paquetes perdidos. Además, puede que el otro sentido tarde más de lo esperado en confirmar la recepción de un paquete por razones del propio protocolo, como el "delayed ACK" de TCP, o simplemente porque la respuesta no sea obligatoria y el flujo de datos en la otra dirección esté siendo usado como respuesta. En el último caso se puede definir como limitado por la aplicación.

Estos factores provocan que la estimación del RTT de forma pasiva no sea una tarea trivial, como se expone en algunos artículos de la literatura, [Zha86]. Para la estimación de forma pasiva se debería tener en cuenta diversas condiciones: desórdenes, retransmisiones, paquetes perdidos, dónde se realiza la captura, etc.

Para el cálculo del RTT a través de una medida pasiva se debe de etiquetar los paquetes enviados y recibidos para saber por cada paquete recibido cuál había sido el paquete enviado que provocaba esta respuesta y por lo tanto, poder calcular el RTT. Para las conexiones que utilicen el protocolo TCP una forma de hacerlo es a través de los parámetros de cabecera: número de secuencia, Seq, y de reconocimiento, ACK. El número de secuencia indica el byte del flujo de datos enviados y el ACK el siguiente número de secuencia del segmento que se espera recibir. De esta forma, estos parámetros pueden ser usados para ordenar secuencialmente los paquetes enviados y los recibidos independientemente del punto de captura.

Aunque el uso del número de secuencia y ACK permite relacionar un paquete enviado y uno recibido, todavía no asegura que el punto de captura pueda describir como fueron enviados los paquetes en el emisor. En la figura 4.3, se muestra un ejemplo de cómo la captura puede influir a la hora del cálculo del RTT. En este caso el servidor manda una ráfaga de tres paquetes al cliente. Se puede observar como los paquetes llegan retrasados al cliente. Cuando el cliente recibe el primer paquete comienza a enviar la confirmación hacia el servidor. Mientras, durante ese tiempo, en el punto capturado ya han llegado los dos primeros paquetes de la ráfaga enviados por el servidor. Los siguientes paquetes enviados por el servidor que fueron enviados a la vez que el primero, Paq2 y Paq3, tendrán un número de secuencia mayor por lo que el primer RTT se calcularía con el paquete 3 y el paquete 1. En realidad, el paquete con el que se debería de calcular sería el 4 en lugar del 3. En este caso se estaría subestimando el RTT real. El ejemplo muestra como la estimación del RTT utilizando los números de secuencia no es trivial.

Debido a que el cálculo de la medida de forma pasiva es interesante para el análisis del rendimiento de una red, algunas propuestas pueden ser encontradas en la literatura. Por ejemplo, algunos trabajos [Str13] y [VLL05], realizaban el cálculo basándose en uno de los parámetros de las cabeceras de TCP, concretamente en la opción *timestamp*. Esta medida resulta muy exacta pero la desventaja es que se trata de una opción optativa de la cabecera TCP, es decir, ambos sentidos debe de haber acordado utilizarla, [Rfc]. Durante el establecimiento de la conexión ambos extremos negocian si van a usar esta opción durante la

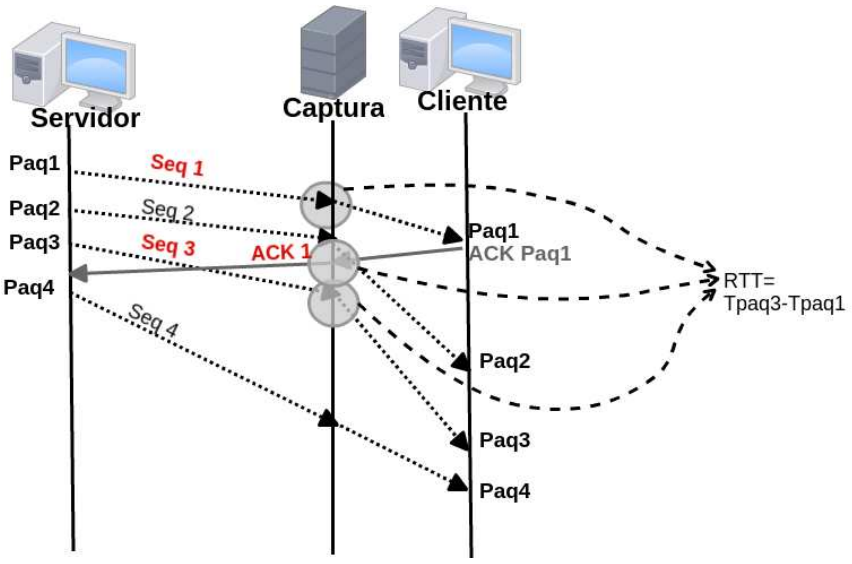


Figura 4.3: Ejemplo de problemas encontrados para el cálculo del RTT.

conexión. Si uno de los extremos no anuncia este parámetro, entonces ninguno de los extremos podrá usarlo durante toda la conexión. Mediante esta opción se solucionan algunos problemas que plantean el cálculo pasivo, como la pérdidas de paquetes o la dependencia con el punto de captura. Además, en sus artículos se demostraba que la estimación era tan buena como el uso de pruebas activas, mediante el envío de paquetes ICMP.

Mediante la opción del *timestamp*, cada respuesta recibida se puede relacionar inequívocamente con el paquete enviado previamente. Para ello cada extremo envía un valor de reloj en los paquetes que manda a través del campo "Tsval", el valor de este campo es relativo a cada extremo. Al recibir el paquete el otro extremo copiará en su paquete el valor del *timestamp* recibido y enviará el suyo propio. De esta forma, cuando se recibe un paquete en el otro extremo puede relacionarlo con el paquete que envió previamente al tener una copia del valor enviado. Para el ejemplo del cálculo del RTT visto anteriormente, 4.3, al asociar correctamente los segmentos enviados después de ver una confirmación, se evitaría subestimar el RTT real, figura 4.4.

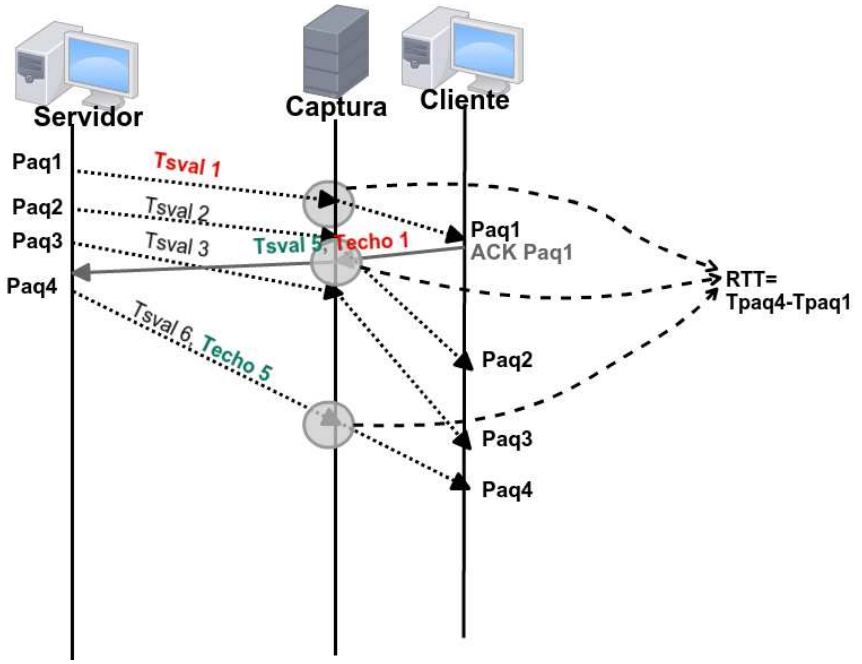


Figura 4.4: Ejemplo del cálculo del RTT mediante la opción del *timestamp*.

El principal problema del método del *timestamp*, es que todavía muy pocas conexiones TCP usan la opción del *timestamp*. Por ejemplo, en observaciones realizadas en el enlace de acceso de la Universidad Pública de Navarra, mostraba que tan sólo un 22 % de las conexiones TCP observadas negociaban exitosamente la opción del *timestamp*. El test fue realizado para 1000 conexiones capturadas durante un día laboral en Noviembre del 2013.

Otros métodos pasivos calculaban el RTT independientemente de la opción de *Timestamp*. Por ejemplo, los autores de [JD02] estimaban el valor a través del establecimiento de la conexión, *handshake*. Es decir, calculaban el RTT a partir del primer SYN y el ACK, paquetes enviados por el cliente, o bien desde el primer SYN+ACK y el siguiente paquete de datos, calculándolo en el lado del servidor. El RTT calculado de este modo no siempre es exacto porque el tiempo para el establecimiento de la conexión puede verse alterado por *middleboxes* entre el cliente y el servidor. Estos sistemas pueden responder o iniciar las conexiones por su

cuenta, por lo que el RTT calculado de esta forma daría valores más bajos de los reales. Estos sistemas también pueden producir el efecto contrario, retardar los paquetes de inicio de conexión, por lo que se observarían valores superiores a los reales.

Otros autores apuestan por técnicas más complejas. Por ejemplo, los autores en [LL03] asocian un segmento de datos con el ACK que lo había disparado. Otros intentan imitar los cambios en la ventana de congestión del emisor, [JID+04]. Estas estimaciones podrían verse afectadas por los paquetes perdidos, la opción de escalado de ventana de TCP o por implementaciones alternativas de TCP.

La motivación para el estudio de un método pasivo para el cálculo del RTT surge debido a la importancia de la métrica y los problemas suscitados por su análisis activo. Un método que calcule el RTT de forma pasiva podría ser usado para detectar problemas en redes de grandes empresas, en las que el tráfico fuera capturado en un punto de observación para su posterior estudio, sin que el análisis pudiera afectar al rendimiento de la red. Este análisis normalmente se realiza capturando el tráfico y analizándose posteriormente en caso de detectar alguna anomalía. En estos tipos de escenarios es inviable la realización de medidas activas para el cálculo del RTT. Además, aún en el caso de poder realizar alguna medida activa, se deberían de decidir previamente los puntos entre los que se desea medir el retardo, ya que si se trata de una red muy grande posiblemente no sea deseable realizar la medición para todos los puntos.

Por este motivo, este capítulo se ha centrado en el desarrollo de una herramienta capaz de estimar el RTT pasivamente para todas las conexiones observadas. El escenario de interés son las redes de alta velocidad con carga media o alta, como pueden ser los centros de datos, *datacenter*, o grandes empresas.

4.2. Propuesta de algoritmo pasivo para el RTT

El algoritmo propuesto provee una estimación del RTT observando el comportamiento para las conexiones TCP que no llenan su producto $bandwidth \times delay$. La idea es utilizar el requisito que todas las conexiones TCP deben cumplir en una red no limitada por el ancho de banda: el flujo de datos de una

conexión TCP está limitada por el valor de la ventana de control que es anunciada por cada lado de la conexión hacia el otro extremo.

Asumiendo que una conexión tiene siempre datos para enviar sobre TCP, si el RTT es suficientemente alto, TCP sería capaz de enviar al completo la ventana permitida por el otro lado antes de recibir la confirmación para el primer paquete de la ventana. Hasta recibir esta confirmación tendría que parar de enviar. En este caso los datos se enviarán como una fuente ON-OFF durante un periodo de tiempo definido por el RTT. Si el RTT no es tan alto, los paquetes confirmados empezarían a llegar antes de alcanzar la ventana y como resultado se vería un flujo tráfico de datos más o menos continúo. El método propuesto para analizar el RTT examina conexiones TCP infiriendo el RTT de los patrones observados ON-OFF.

Como se ha citado anteriormente, el objetivo es desarrollar un método de estimación para las conexiones utilizando solamente el tráfico capturado, es decir, de una forma pasiva. Para que el servidor pueda aprovechar el límite de la ventana anunciada por el otro extremo, las conexiones no deben de llenar su producto $bandwidht \times delay$. De otra forma los servidores se verían limitados por el ancho de banda y no la ventana anunciada. En ese caso se observaría un envío homogéneo de paquetes sin patrones ON-OFF.

La idea es comprobar si en un determinado intervalo de tiempo, que será el candidato a RTT, se han acumulado más Bytes que los permitidos durante dicho periodo, por la ventana anunciada por el otro extremo. Este hecho indica que el RTT tiene que estar por debajo de ese valor. El RTT candidato, con el que se iniciará la búsqueda, deberá de ser mayor del presumiblemente buscado, ya que en cada iteración su valor se irá decrementando, al comprobarse que se habrían enviados más datos de los permitidos. De esta forma el algoritmo fallará para el primer valor para el que se cumpla que los Bytes enviados son menores que el límite de la ventana. Este valor será un límite superior al buscado, lo que evitará posibles subestimaciones.

El algoritmo comienza a iterar con un RTT candidato presumiblemente mayor que el de la conexión. La conexión es dividida en intervalos de tiempo fijos, dependiendo del RTT candidato. En cada intervalo se contabiliza los Bytes enviados y se guarda la ventana máxima anunciada por el otro extremo. Si en

alguno de los intervalos se envían más Bytes que la ventana, entonces el candidato es descartado. El valor del candidato es entonces disminuido y se comienza una nueva iteración. De esta forma el algoritmo busca el valor más pequeño posible del candidato que pasa el test indicando que es aceptable como RTT.

Las restricciones del algoritmo se pueden relajar un poco en cuanto a definir qué ventana anunciada es la que decide que se han mandado más Bytes que los permitidos. Siendo estricto, la ventana será la ventana máxima anunciada correspondiente a dicho intervalo, algoritmo RTT1. Siendo un poco más permisivo, esta ventana podría ser la ventana máxima anunciada hasta el intervalo que está siendo analizado, versión RTT2. O incluso, la ventana máxima anunciada para toda la conexión, RTT3.

El algoritmo se define a través de funciones, a continuación se definen la notación que será utilizada:

- c: Tiempo total que dura una conexión.
- t: Candidato a probar como posible valor del RTT.
- n: Número de intervalos de duración t, de la conexión, $n = \lceil \frac{c}{t} \rceil$
- i: Intervalo siendo evaluado, $i \in 0..n$.
- B_i: Bytes totales enviados por el servidor en el intervalo, i.
- w_i: Ventana máxima anunciada en el intervalo i.
- w_i^{max}: Ventana máxima anunciada hasta el intervalo i, $w_i^{max} = max\{w_k\} \quad \forall k \in 0..i$

A continuación se describe cada versión del algoritmo a través de funciones:

1. RTT1: El candidato t es válido si en cada intervalo i, los Bytes totales enviados son menores que la ventana máxima para cada intervalo, ecuación 4.1.
2. RTT2: El candidato t es válido si en cada intervalo i, los Bytes totales enviados son menores que la ventana máxima vista para toda la conexión, descartando la observada para el establecimiento de la conexión, es decir, para los paquetes SYN, SYN+ACK y ACK, ecuación 4.2.

3. RTT3: El candidato t es válido si en cada intervalo i, los Bytes totales enviados son menores que la ventana máxima anunciada hasta el momento, ecuación 4.3.

$$RTT1 = t \quad \forall i \in 0..n \quad B_i < w_i \tag{4.1}$$

$$RTT2 = t \quad \forall i \in 0..n \quad B_i < w_n^{max} \tag{4.2}$$

$$RTT3 = t \quad \forall i \in 0..n \quad B_i < w_i^{max} \tag{4.3}$$

Como ya se ha adelantado, la búsqueda del valor válido, t, se inicia utilizando un candidato que es claramente mayor que el RTT. Cada vez que un candidato falla su valor se reduce por un valor fijo, δt. Cuando un valor, $t - \delta t$, falla el test, el valor previo t es declarado como estimación del RTT. El valor para decrementar, δt, impondrá la resolución del estimador.

Se considera que el algoritmo termina cuando el candidato falla el test. El valor elegido como resultado final será justo el candidato anterior al candidato que falla el test. De esta forma el resultado será siempre una cota superior del valor real. Si el candidato no falla el test durante la primera iteración no se tendrá información del RTT real puesto que, éste podría estar por encima del comprobado. En estos casos se podría elegir otro candidato multiplicando el valor probado por algún valor y se reiniciaría el test.

4.3. Escenario de validación

Para validar el algoritmo se ha utilizado un entorno emulado en el cual se controlaba el RTT que se pretende medir, figura 4.5. En el escenario se tienen varios clientes conectados a una red Ethernet de 10Mbps. La red virtual está conectada a una segunda red, también a 10Mbps, a través de un router, que en realidad es una máquina emulada. A la segunda red se conecta el servidor Web. A través del router se controla el RTT de las conexiones, para ello se aplica un retardo a los paquetes utilizando la herramienta Netem [Net].

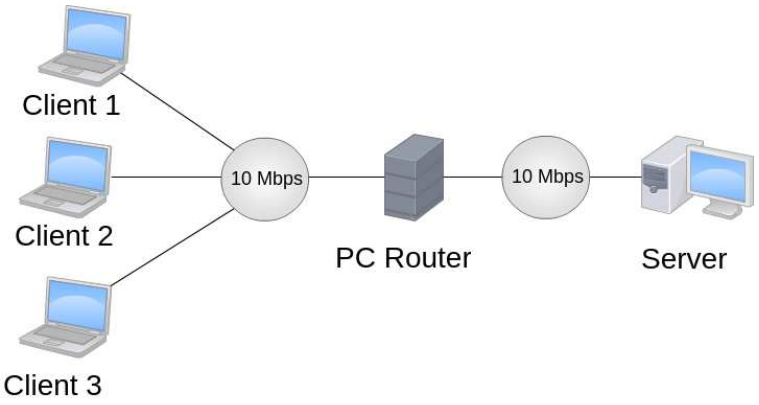


Figura 4.5: Escenario emulado de una red cuyas conexiones están limitadas por $ventana/RTT$.

Para la emulación del escenario se ha utilizado VirtualBox[1] corriendo en un PC anfitrión con Ubuntu 14.04. Tanto los clientes como el router son virtualizaciones de un sistema operativo de Ubuntu 12.04. La máquina anfitrión que corre con el software de virtualización actúa además como el servidor Web, figura 4.5.

El escenario es configurado para emular diferentes RTT, cambiándolo para ello en el router, el cuál añade la mitad del RTT que se quiere probar para cada sentido del camino. De esta forma, los escenarios son configurados para realizar experimentos con RTT de 40, 80, 120, 200 y 400ms. Con el objetivo de emular el comportamiento de los *middleboxes*, los cuales habitualmente son usados en redes de alto rendimiento, los establecimientos de las conexiones TCP, es decir los tres primeros paquetes que son enviados en las conexiones, SYN, SYN+ACK y ACK, tendrán un RTT más pequeño que el resto de paquetes, exactamente un 10 % menos.

Para tener conexiones que no estén llenando su producto $bandwidth \times delay$, en este escenario la opción del escalado de ventana de TCP ha sido desactivada.

Los experimentos consisten peticiones HTTP por parte de los clientes al servidor Web. El servidor enviará como respuesta un tamaño variable de página según una distribución uniforme de 1 a 3 MBytes. El tiempo entre las peticiones de los clientes se modela siguiendo una distribución uniforme con media de

[1] https://www.virtualbox.org/

8 segundos. Con estos patrones de peticiones y respuestas se tendrá una carga media en el canal sobre los 6Mbps.

4.4. Resultados

Como ya se ha explicado anteriormente, el RTT de un camino usado por una conexión TCP puede ser definido como el tiempo que le cuesta a un paquete viajar desde un extremo hacia el otro, más el tiempo del siguiente paquete enviado tras recibir la confirmación del extremo contrario. Esta propiedad podría ser calculada añadiendo los retardos de los enlaces en el camino. Sin embargo, una conexión TCP realmente está afectada por el retardo de cada paquete enviado, lo cuál no es siempre el RTT puro del camino debido a las variaciones que sufre esperando en colas a lo largo del camino o al tiempo de espera para la respuesta en los extremos remotos. Por eso, el RTT puede ser visto como un proceso aleatorio. La estimación de los algoritmos intenta medir esta variable aleatoria.

En el escenario presentado se han evaluado diferentes estimadores del RTT, RTT1, RTT2 y RTT3. Aplicar cualquiera de estos tres algoritmos a una conexión devolverá un único valor RTT que defina toda la conexión. Estos resultados han sido comparados con dos estimaciones obtenidas mediante métodos clásicos, el RTT calculado con el establecimiento de la conexión y el RTT obtenido mediante la opción del *Timestamp*. El RTT inicial mide el RTT a través del primer paquete SYN de la conexión y el SYN+ACK de respuesta. Este tiempo en realidad es la duración del establecimiento en tres pasos. El método que utiliza el Timestamp mide el RTT observando la opción del *Timestamp* enviada en la cabecera de cada paquete. Este método provee muestras exactas del RTT real. Con este estimador se tienen varios RTT para una conexión, ya que, durante la misma puede haber variabilidad por saturación de la red o routers. A efectos de compararlo con los estimadores propuestos, nos quedamos con el valor más pequeño observado para la conexión.

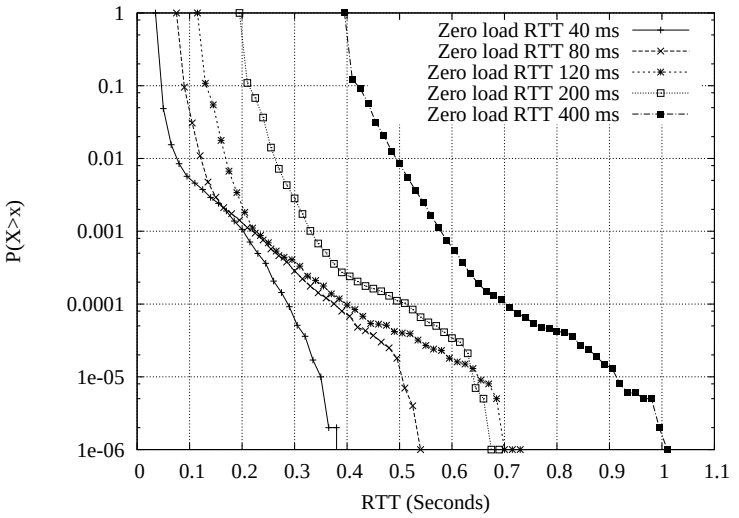


Figura 4.6: Estimación del RTT en el escenario emulado usando el método *Timestamp*.

Cada conexión TCP en el escenario de validación sigue el mismo camino, es decir, no existen balanceos de carga, por lo que las muestras obtenidas de las mediciones con el método de *Timestamp* pueden ser usadas como valores reales en el proceso aleatorio del RTT.

La figura 4.6 muestra la función de densidad de probabilidad del RTT para las diferentes configuraciones analizadas.

Para comparar los estimadores se utiliza un conjunto de 2000 conexiones para cada uno de los escenarios explicados en los que se tienen diferente RTT. Los cinco estimadores, RTT1, RTT2, RTT2, inicial y *Timestamp*, son computados para cada conexión TCP observada.

Con el fin de analizar y comparar los resultados obtenidos para los diferentes estimadores, se tienen en cuenta valores como la media, el mínimo, el máximo y la varianza. Estos resultados se muestran en la tabla 4.1. Las funciones de densidad de probabilidad son mostradas en la figura 4.7. Se puede apreciar que los resultados de todos los estimadores sobreestiman ligeramente los RTT reales excepto para la opción del *timestamp*, que es tan preciso como se esperaba. La sobreestimación es mayor cuando los RTT a estimar tienen valores pequeños.

Para la mayoría de casos, los resultados obtenidos para los métodos propuestos, estaban cerca del RTT esperado, figura 4.7. De los tres algoritmos, RTT1 es el que sobreestima menos puesto que es el más estricto. Los otros dos métodos, deciden que un valor es válido antes que el RTT1.

La configuración del escenario de validación emula el uso de *middleboxes*. Esto implica que la estimación del RTT mediante el método del RTT inicial, el cuál se obtenía a partir del establecimiento de la conexión, debería de ser ligeramente más pequeño que para el resto de la conexión, concretamente un (10 % inferior). Sin embargo, en un gran número de conexiones el retardo inicial era del mismo orden que el RTT para el resto de la conexión, ya que, el servidor experimentaba una carga moderada. Este 10 % apararece en el valor mínimo del RTT para cada experimento, tabla 4.1. Sin embargo, en la misma tabla se puede observar como los valores máximos y medios están por encima de RTT configurado.

Los resultados de algunas conexiones mostraban valores altos para todos los métodos, exceptuando el del *Timestamp*. En algún periodo de tiempo de dichas conexiones, el RTT era mayor que el configurado debido a que el servidor estaba ocupado con otras peticiones. Los métodos propuestos se adaptan al mayor RTT visto durante la conexión, por eso los valores son mayores de lo esperado. Por ejemplo, en la figura 4.8 se muestra una conexión para la que los valores RTT obtenidos con los métodos fueron mayores de lo esperado. Esta conexión pertenecía al conjunto de las conexiones de un experimento del escenario de 400ms, sin embargo los resultados de aplicar los algoritmos mostraban RTT de 700ms. Los valores obtenidos eran similares a los máximos observados con el método del *Timestamp* para estas conexiones.

En la figura 4.9 se muestran los resultados para todos los estimadores. La media es ligeramente inferior para el método “inicial” comparado con los propuestos. En cambio, la desviación es superior, lo que indica una mayor variabilidad de la medida. Además, merece la pena comentar que en algunos casos los valores obtenidos con este método constituyen una subestimación del RTT real.

Tabla 4.1: Estadísticas calculadas para el RTT

RTT Experimento	Method	Min (s)	Máx (s)	Media (s)	Varianza (s^2)
40ms	rtt1	0.054	0.396	0.087	0.003
	rtt2	0.054	0.396	0.087	0.003
	rtt3	0.054	0.396	0.087	0.003
	Initial	0.038	0.417	0.085	0.005
	Timestamp	0.036	0.041	0.040	0.000
80ms	rtt1	0.081	0.569	0.123	0.004
	rtt2	0.081	0.569	0.124	0.004
	rtt3	0.081	0.569	0.123	0.004
	Initial	0.074	0.569	0.126	0.006
	Timestamp	0.076	0.081	0.080	0.000
120ms	rtt1	0.121	0.660	0.151	0.003
	rtt2	0.121	0.660	0.151	0.003
	rtt3	0.121	0.660	0.151	0.003
	Initial	0.113	0.700	0.153	0.005
	Timestamp	0.115	0.122	0.120	0.000
200ms	rtt1	0.200	0.661	0.227	0.003
	rtt2	0.200	0.661	0.228	0.003
	rtt3	0.200	0.661	0.227	0.003
	Initial	0.190	0.712	0.224	0.005
	Timestamp	0.195	0.201	0.200	0.000
400ms	rtt1	0.398	0.945	0.422	0.002
	rtt2	0.398	0.945	0.423	0.002
	rtt3	0.398	0.945	0.422	0.002
	Initial	0.380	0.926	0.404	0.004
	Timestamp	0.396	0.402	0.400	0.000

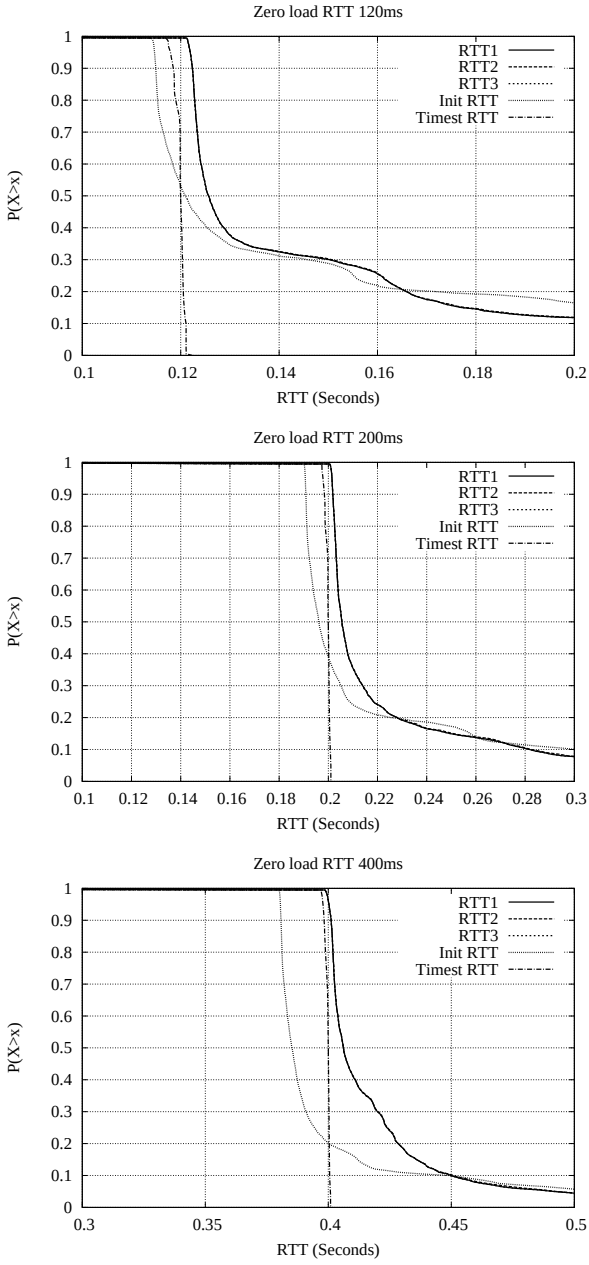


Figura 4.7: Comparación del RTT estimado obtenido con todos los métodos.

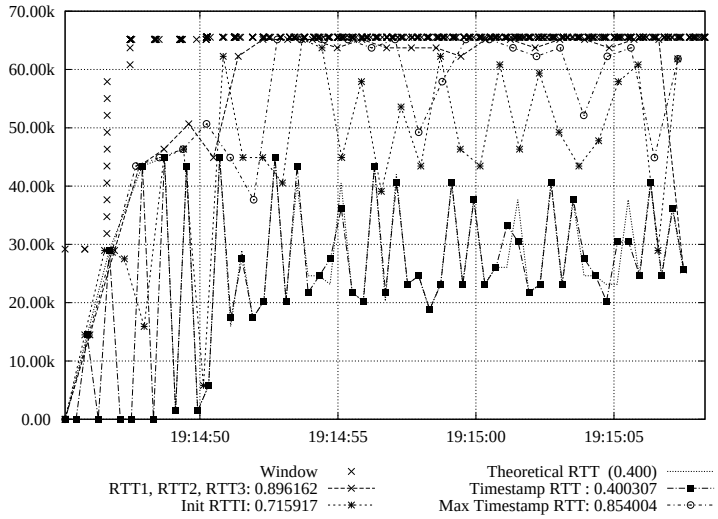


Figura 4.8: Serie temporal de Bytes observada para cada RTT candidato.

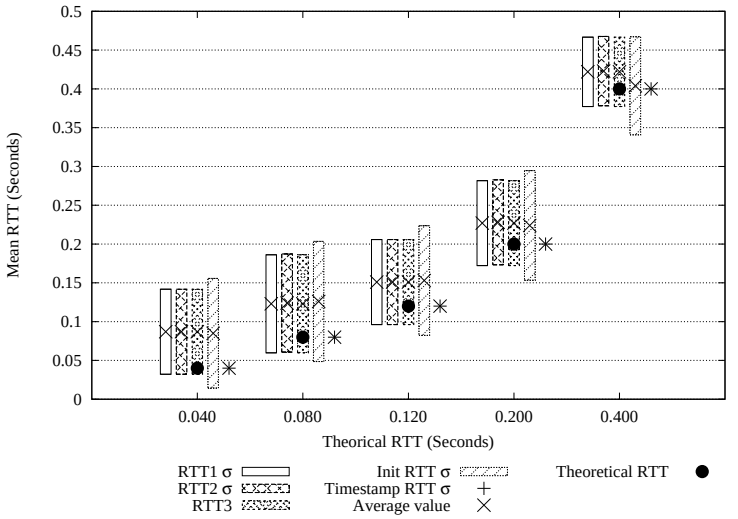


Figura 4.9: RTT medio obtenido y su desviación para todos los experimentos.

Como ya se ha citado anteriormente, los algoritmos propuestos comprueban la validez de un RTT candidato comparando los Bytes observados con la ventana anunciada. Este método requiere que las conexiones no llenen su producto $bandwidth \times delay$, para que los emisores sean capaces de enviar tantos paquetes como para llenar la ventana anunciada por el receptor. Para esos casos, el candidato inicial del RTT sería un límite superior y no daría información. Este límite puede ser calculado matemáticamente dependiendo de la velocidad del canal, figura 4.10. Los puntos $(bandwidth, RTT)$ por debajo de la línea representan las situaciones donde el algoritmo no debería de ser utilizado. Estos límites dependen de la máxima ventana anunciada permitida por TCP. Habitualmente, este valor es de 64KBytes pero puede ser ampliado mediante la opción del escalado de ventana intercambiado en el establecimiento de la conexión. En la figura 4.10 se muestra el límite para varios valores según el escalado de la ventana, de 64KB a 8MB.

En el caso de escenarios de bajo ancho de banda, los algoritmos sobreestimarán poco el RTT siempre y cuando las conexiones no utilicen el escalado de ventana, o bien usen valores bajos. En el caso de escenarios de alta velocidad, los cuáles son los escenarios principales de interés de esta tesis, se puede aplicar los algoritmos para conexiones TCP con valores altos de escalado de ventana. Por ejemplo, para una red de datos de 1Gbps cuyas conexiones utilicen un tamaño de ventana de 2MB, sería posible estimar de forma bastante exacta RTT mayores de 16ms. En un enlace con 100Mbps sería posible estimar RTT mayores de 40ms, siempre y cuando el tamaño de ventana fuera de 512Kb o menos. Para escenarios de red con menos velocidad, 10Mbps, los métodos son capaces de estimar bastante bien siempre y cuando no se use la opción de escalado de ventana y los RTT reales sean superiores a 51.2ms. Este efecto se pudo observar en el escenario de validación cuando se probó a aplicar los métodos propuestos para RTT más pequeños, 40ms. En la tabla 4.1 se mostraban los resultados de RTT1, RTT2 y RTT3 para este caso y se observaba que los resultados obtenidos no estaban nunca por debajo de 52ms.

Aunque los extremos pueden acordar usar un determinado valor de escalado de ventana en los escenarios reales, esto no implica que los servidores anuncien, en algún momento de la conexión, el valor máximo permitido. Un estudio

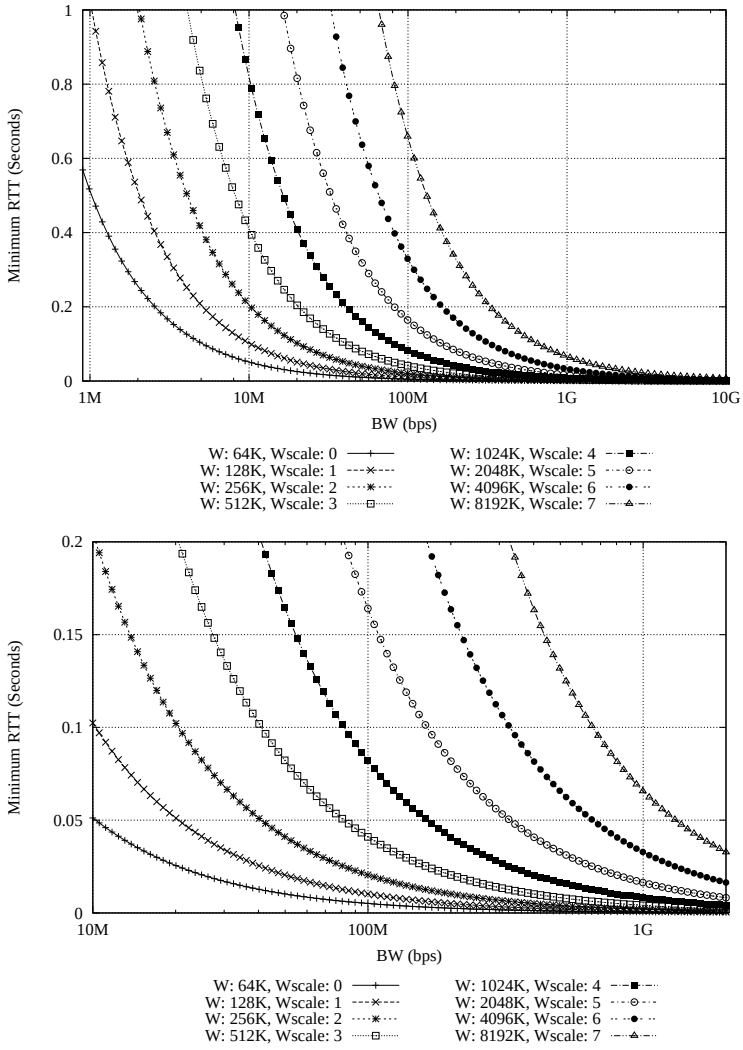


Figura 4.10: Medias y errores de desviación obtenidos para los RTT calculados con cada experimento y para cada escenario.

en el enlace de acceso de la Universidad Pública de Navarra mostraba que las conexiones TCP normalmente negociaban el escalado de ventana hasta 8MB. Sin embargo, durante la conexión no anunciaban ventanas de este tamaño. La

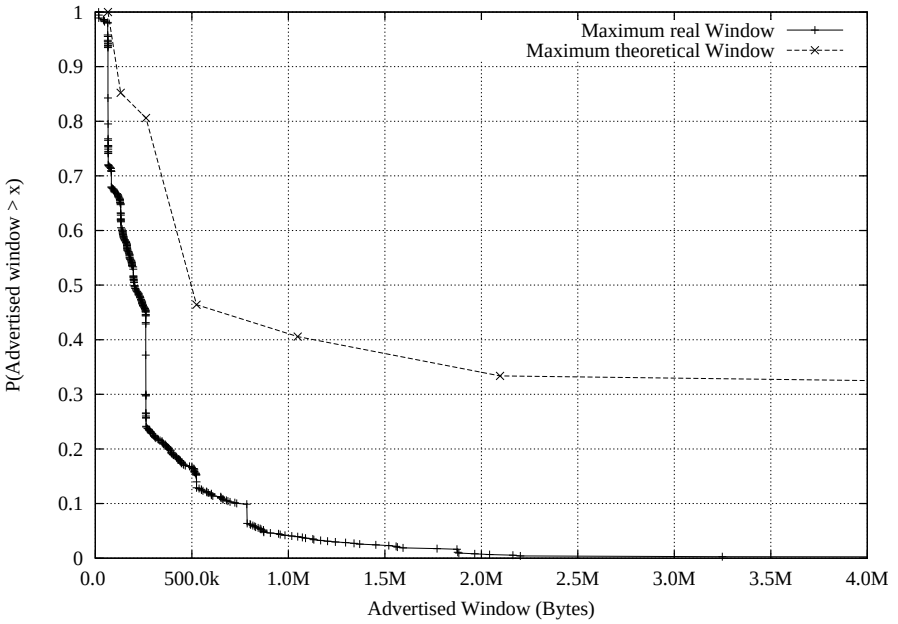


Figura 4.11: Función de supervivencia de la ventana máxima anunciada y permitida en las conexiones estudiadas del enlace de la Universidad Pública de Navarra.

figura 4.11 muestra la función de supervivencia de la ventana anunciada usada en las conexiones comparada con la función de supervivencia de la ventana negociada durante el establecimiento de la conexión. Alrededor de un 30 % de las conexiones negociaban ventanas de 512KB, pero sólo aproximádamente un 15 % anuncian esa ventana en algún momento de la conexión. Este estudio revela que alrededor del 85 % de las conexiones TCP observadas en el enlace se les podría aplicar los métodos de cálculo de los RTT y obtener unos resultados bastante fiables.

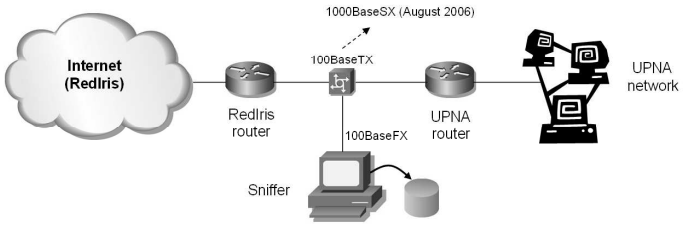


Figura 4.12: Tráfico capturado en el enlace de la Universidad Pública de Navarra.

4.5. Caso de estudio: Escenario real, Universidad Pública de Navarra

En esta sección se analizan las prestaciones de los algoritmos, para el cálculo del RTT, en un escenario real, concretamente, en la red de la Universidad Pública de Navarra. En este escenario casi todo el porcentaje del tráfico TCP es causado por conexiones Web, ya sean entrantes hacia servidores de la universidad o bien salientes, es decir, conexiones hacia servidores Web externos.

Para analizar el tráfico se sigue el mismo procedimiento explicado en el capítulo 3.4. Mediante una sonda se capturan las cabeceras de los paquetes de las conexiones tanto entrantes como salientes, figura 4.12.

Los resultados se comparan con el valor mínimo obtenido para el método del *Timestamp*, tal y como se hizo para la validación de los métodos. Por tanto, aunque se han analizado las conexiones de una hora, sólo se presentan los resultados para aquellas conexiones que tuvieran activadas la opción del *Timestamp*.

Además de los métodos propuestos, los resultados se comparan con un nuevo método que utiliza estimaciones realizadas en cada extremo. El método se basa en el cálculo del RTT a partir de las diferencias de tiempo entre un paquete enviado en un sentido y su confirmación por el protocolo TCP del otro, observadas para cada sentido. La estimación de este método consiste en la suma de los valores mínimos obtenidos para cada extremo. Si se calcula solamente con un paquete enviado y su confirmación se estimaría el RTT que se ve desde el punto de observación hasta uno de los extremos. Al calcularse para ambos sentidos la suma sería un estimador del RTT de la conexión. En la figura 4.13 se muestra un ejemplo de cómo se calcula este valor de RTT.

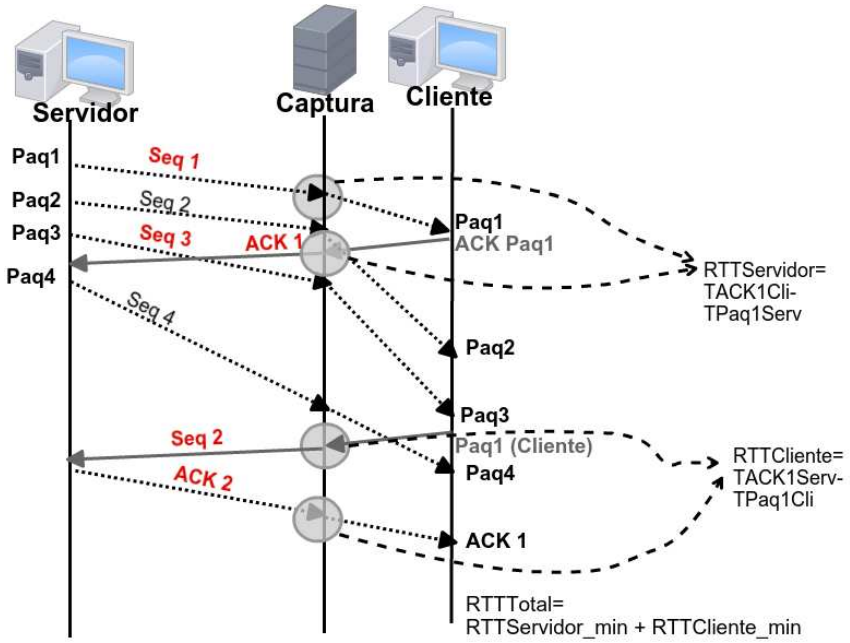


Figura 4.13: Ejemplo del cálculo del RTT mediante las sumas paraciales de los RTT de cada extremo.

Este método todavía presenta el problema que dependiendo de las condiciones de captura puede realizar subestimaciones. Además, debido a los *Delayed ACK* puede obtener valores bastante por encima de los esperados, por eso se eligen para la suma, con el fin de minimizar este efecto, los valores mínimos vistos durante la conexión. Pese a los problemas que puede presentar resulta interesante el estudio de estos valores para las conexiones de la universidad, puesto que podrían aportar información para comparar mediante el valor del RTT calculado con el método del Timestamp y poder así estudiar cómo ve la sonda las conexiones y cómo son en realidad.

Otra variación con respecto a la validación, es que el algoritmo del *Timestamp* se aplica también en los dos sentidos, tanto para el cliente como para el servidor, quedándonos con el valor mínimo observado en la conexión. En algunos casos, debido a los pocos datos que se envían, si se calcula solamente el RTT en un sentido se tendrán pocas muestras. Este algoritmo no está influido por el punto

de captura, no depende de si se está cerca del emisor o receptor, por lo que se puede calcular en ambos sentidos. De esta forma, se obtendrá un mínimo más real acorde con lo observado para algunos casos.

Para el caso de estudio, se analizan por intervalos de horas las conexiones para el día 11 de Noviembre del 2013. Aunque se analizan varias horas durante ese día, en adelante, se presentan los resultados para una de las horas más cargadas, de 10 a 11 de la mañana, ya que para el resto se obtienen resultados similares.

Con el objetivo de analizar el impacto de aplicar el algoritmo aún a las conexiones que no cumplan el criterio de llenar el producto $bandwidth \times delay$, en un primer estudio los algoritmos son aplicados a todas las conexiones, 4.5.1. En un segundo estudio, se aplicará solamente a aquellas que anuncien ventanas que cumplan el criterio $bandwidth \times delay$, 4.5.2.

4.5.1. Estudio para todas las conexiones

En el estudio se diferencian las conexiones entrantes, es decir, clientes de fuera de la universidad que acceden a los servidores de la misma, y conexiones salientes, conexiones desde dentro de la universidad hacia servidores externos. Como se ha citado anteriormente, los resultados se muestran solamente para una hora de datos, de 10 a 11 de la mañana del 11 de Noviembre del 2013. En la tabla 4.2 se muestran las estadísticas de las conexiones.

Tabla 4.2: Datos generales de las conexiones capturadas en la UPNA de 10-11h del 11-11-2013.

	Conexiones salientes		Conexiones entrantes	
Métrica	Número	(%)	Número	(%)
Conexiones Totales UPNA	240012	–	500937	–
Con datos para RTT	231193	96.33	456819	91.2
Conexiones Timestamp	16435	6.84	68098	13.6

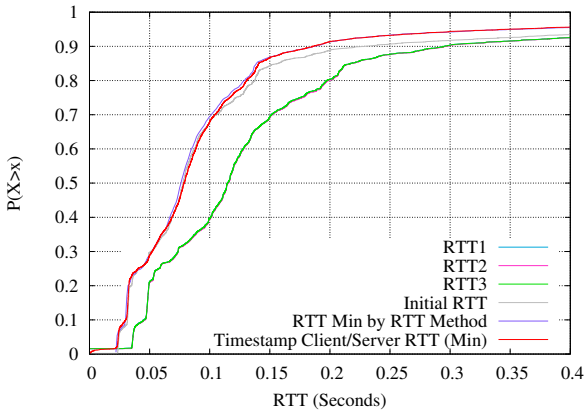
Los algoritmos propuestos parten de un *Timestamp* inicial que debería ser mayor que el RTT buscado. Se puede partir de valores imposibles y esperar que éste se vaya decrementando en cada iteración hasta que se encuentre el óptimo.

Para que el algoritmo sea más rápido, se parte de la estimación del RTT calculado con el *handshake* de la conexión, multiplicado por un escalar, con el fin de asegurarnos que sea un valor por encima del buscado. Sin embargo, para todas las conexiones no se puede obtener un valor inicial porque el *handshake* no es observado debido a la captura. Estas conexiones en realidad son unas pocas, tan sólo para un 8.8 %, el resto son etiquetadas como "Con datos para el RTT", tabla 4.2.

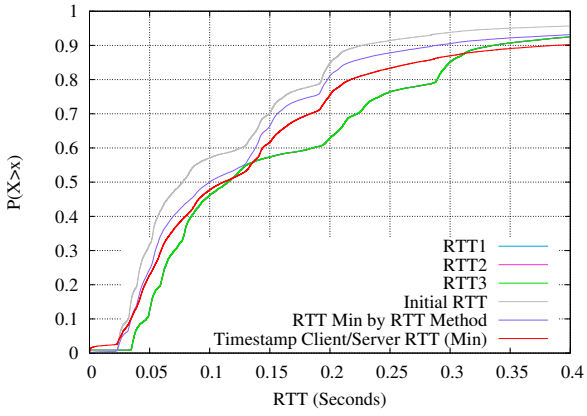
De la tabla de estadísticas, 4.2, se destaca el porcentaje tan bajo de conexiones en las que está presente la opción del *Timestamp*, tan sólo aproximádamente un 7 % para las conexiones entrantes y aproximádamente un 14 % de las salientes. El RTT calculado con el método del *Timestamp*, se va a utilizar como valor fidedigno para comparar el resto, por lo que, sólo se muestran los resultados para estas conexiones.

En la figura 4.14, se muestran las distribuciones de los valores obtenidos.

Analizando solamente las distribuciones de los RTT de los *Timestamp* se observa como las conexiones salientes parecen tener RTT superiores a los entrantes. Mientras que un 90 % de las conexiones entrantes hacia la Universidad tienen valores inferiores a 200ms, de las conexiones salientes sólo el 75 % tienen valores inferiores. La diferencia es todavía mayor si se baja a los 100ms, mientras que para las entrantes se tiene aproximádamente un 70 %, para las salientes tan sólo un 47 %. La diferencia de tiempos entre las conexiones entrantes y salientes se puede visualizar mejor en las gráficas de histogramas por porcentajes, 4.15. Estos porcentajes son lógicos si se piensa en términos de los perfiles de clientes que acceden. Mientras que la mayoría de las conexiones entrantes serán realizadas por clientes de la propia universidad accediendo desde sus domicilios hacia las páginas oficiales, y por tanto a pocos saltos de la red RedIris, las salientes serán realizadas por la comunidad universitaria hacia cualquier contenido de Internet en cualquier parte del mundo.

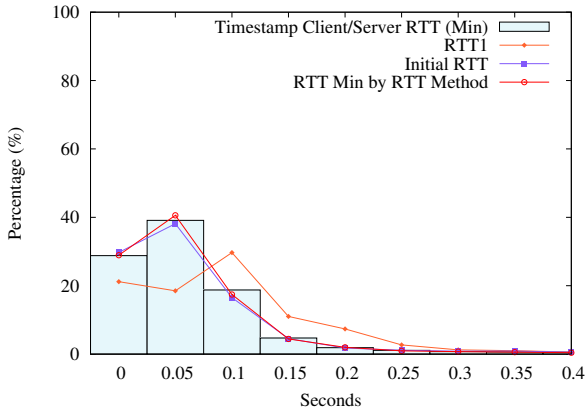


1) Conexiones entrantes

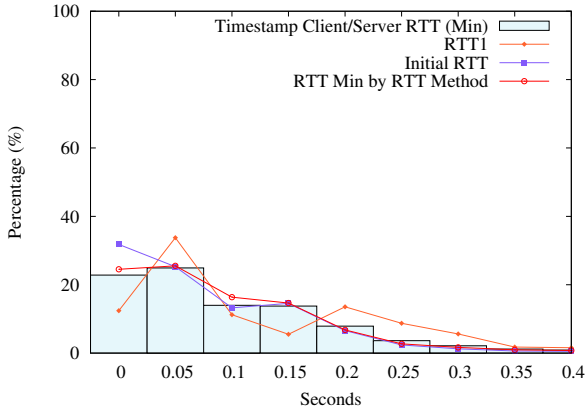


2) Conexiones salientes

Figura 4.14: Distribuciones acumuladas de los RTT calculados para las conexiones entrantes y salientes de la UPNA.



1) Conexiones entrantes



2) Conexiones salientes

Figura 4.15: Histogramas en porcentajes de los RTT calculados para las conexiones entrantes y salientes de la UPNA.

Comparando los resultados obtenidos para los diferentes RTT, para ambos tipos de conexiones, se observa que la distribución de los RTT obtenidos a partir del *handshake*, es decir el *RTT inicial*, es menor que la obtenida con el *Timestamp*. Este efecto es más visible para las conexiones salientes. Este comportamiento es provocado por *Proxys* que permiten que parezca que una conexión se establezca más rápido de lo que luego realmente va. Esta era una de las causas que motivaban el estudio de un algoritmo para el cálculo del RTT independientemente del obtenido con los primeros paquetes de la conexión.

Para ambos casos se observa que las distribuciones de los métodos, *RTT1, RTT2, RTT3*, obtienen unos resultados similares para el conjunto estudiado. De hecho, en las gráficas se visualiza sólo el RTT3 puesto que las líneas de los otros dos están por debajo de ésta. Los resultados obtenidos son sobreestimaciones de los RTT comparándose con los obtenidos por el método del *Timestamp*, es decir, a partir del RTT inicial los algoritmos convergen por encima de los valores reales. Esto es debido principalmente a dos causas. La primera es que no se está teniendo en cuenta si se cumple el criterio $bandwidth \times delay$. La segunda es causada por la naturaleza del tráfico, al ser Web los servidores no mandan todo lo posible dentro de la ventana por lo que parece que los algoritmos convergen rápidamente por cumplir que en cada RTT no se llena la ventana.

En el caso de las conexiones salientes la distribución del método *RTT Req-ACK* tiene una diferencia notable con la del *Timestamp*. En algunos casos, el servidor parece tardar en mandar el siguiente paquete de datos, por lo que los valores obtenidos con el *Timestamp* son altos. Al ser las conexiones pequeñas, no se obtienen muchas medidas para el RTT. Estos dos hechos provocan que los valores para el *Timestamp* estén por encima de los reales. A continuación se muestra un ejemplo real de conexión en la que se observa este efecto, las direcciones IP han sido anonimizadas para mantener la confidencialidad. En azul se muestran los paquetes que intervienen en el cálculo del RTT por *Timestamp*, y en verde los que intervienen para el RTT por *Req-Ack*.

```
1384161073.686449 IP y.y.y.y.42742 > x.x.x.x.http: S 194961804:194961804(0) win 5840 <mss 1460,sackOK,
timestamp 137550045 0,nop,wscale 7>
1384161073.727997 IP x.x.x.x.http > y.y.y.y.42742: S 3717127432:3717127432(0) ack 194961805 win 62392
<mss 1430,sackOK,timestamp 707209592 137550045,nop,wscale 6>
1384161073.728656 IP y.y.y.y.42742 > x.x.x.x.http: . ack 1 win 46 <nop,nop,timestamp137550087 707209592>
1384161073.728946 IP y.y.y.y.42742 > x.x.x.x.http: P 1:680(679) ack 1 win 46 <nop,nop,timestamp
```

```
137550087 707209592>
1384161073.770549 IP x.x.x.x.http > y.y.y.y.42742: . ack 680 win 997 <nop,nop,timestamp
707209634 137550087>
1384161073.800667 IP x.x.x.x.http > y.y.y.y.42742: P 1:250(249) ack 680 win 997 <nop,nop,timestamp
707209664 137550087>
1384161073.801504 IP y.y.y.y.42742 > x.x.x.x.http: . ack 250 win 54 <nop,nop,timestamp
137550160 707209664>
1384161092.142161 IP y.y.y.y.42742 > x.x.x.x.http: P 680:1360(680) ack 250 win 54 <nop,nop,timestamp
137568500 707209664>
1384161092.212532 IP x.x.x.x.http > y.y.y.y.42742: P 250:499(249) ack 1360 win 1002 <nop,nop,timestamp
707228076 137568500>
1384161092.213447 IP y.y.y.y.42742 > x.x.x.x.http: . ack 499 win 63 <nop,nop,timestamp
137568572 707228076>
1384161111.570823 IP y.y.y.y.42742 > x.x.x.x.http: F 1360:1360(0) ack 499 win 63 <nop,nop,timestamp
137587929 707228076>
1384161111.612386 IP x.x.x.x.http > y.y.y.y.42742: F 499:499(0) ack 1361 win 1002 <nop,nop,timestamp
707247476 137587929>
1384161111.613268 IP y.y.y.y.42742 > x.x.x.x.http: . ack 500 win 63 <nop,nop,timestamp
137587972 707247476>
```

4.5.2. Estudio para las conexiones que cumplen el criterio $bandwidth \times delay$

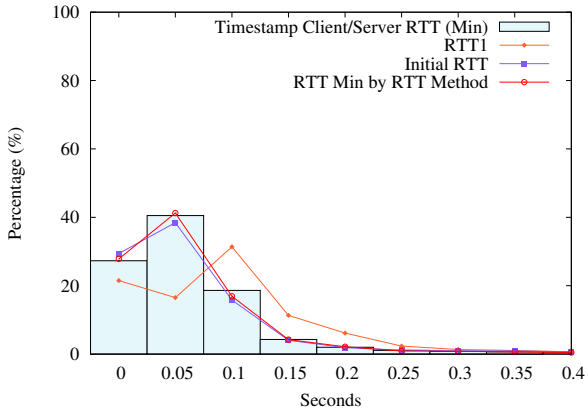
En esta subsección se presenta los resultados solamente para aquellas conexiones que cumplen el criterio $bandwidth \times delay$ en las que se minimizaría la sobreestimación en el caso de los algoritmos de cálculo del RTT propuestos. La idea es comprobar cómo estaba afectada la sobreestimación por las conexiones que no eran capaces de llenar su ventana. Para ello se observa la ventana anunciada por el servidor y se toma como ancho de banda el de la Universidad 100 Mbps, aunque en realidad, podría ser menor que éste. En la tabla 4.3 se muestra el número de conexiones que cumplen este criterio. En el caso de las salientes, el conjunto de conexiones se ve reducido tan sólo al 20 % debido a que las ventanas anunciadas son mayores que las anunciadas por los servidores de la UPNA.

Tabla 4.3: Conexiones capturadas en la UPNA en una hora, 10-11h, que no se llenan por $bandwidth \times delay$.

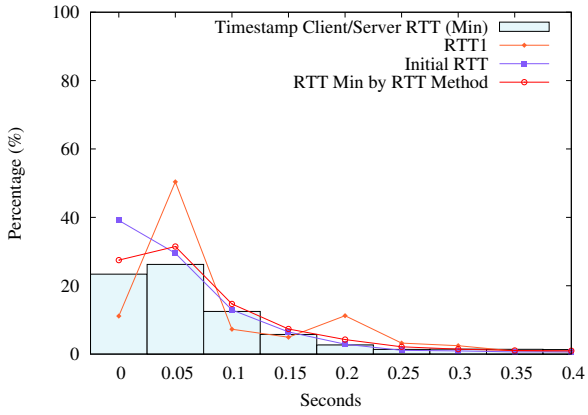
Métrica	Total	Cumplen criterio	Porcentaje (%)
Conexiones Entrantes	16435	13204	80.3
Conexiones Salientes	68098	13866	20.4

Para las conexiones entrantes, debido a que el 80 % de las estudiadas anteriormente cumplían el requisito, no se observan diferencias en el histograma de porcentajes, figura 4.16. Para el caso de las salientes, se observa diferencia para los porcentajes de las distribuciones obtenidas con el método *RTT1*. Se tienen más casos en los que se sobreestima los valores. Estos casos son provocados por algunas conexiones que tienen RTT iniciales muy bajos comparados con los obtenidos con el *Timestamp*, incluso de diferencia de segundos. Al ser tan bajo, los algoritmos convergen en la primera iteración sin dar resultados buenos.

Si se analizan solamente en términos de errores, cogiendo como valores buenos los del *Timestamp*, se observa que se tienen mejores resultados para casos de RTT pequeños, figura 4.17. En la figura, la parte inferior de las cajas representa el percentil 10, la superior el percentil 90, la cruz el valor medio y las líneas los valores mínimos y máximos observados dentro de un intervalo definido por el método del *Timestamp*.

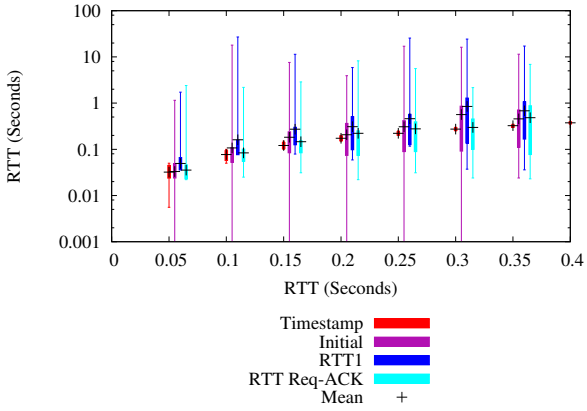


1) Conexiones entrantes

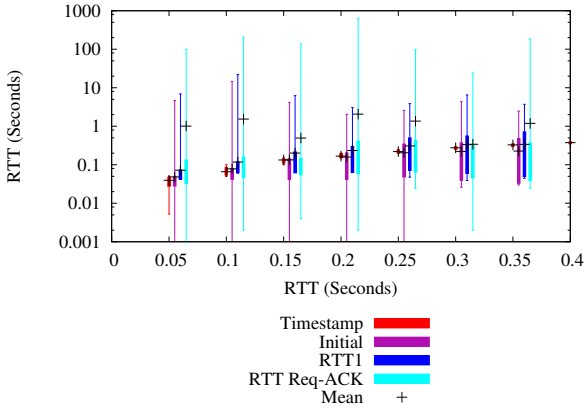


2) Conexiones salientes

Figura 4.16: Histogramas en porcentajes de los RTT calculados para las conexiones entrantes y salientes de la UPNA que cumplen $bandwidth \times delay$.



1) Conexiones entrantes



2) Conexiones salientes

Figura 4.17: Estadísticas de los RTT calculados para las conexiones entrantes y salientes de la UPNA.

4.6. Conclusiones

La métrica del RTT es una herramienta útil desde el punto de vista de la evaluación del rendimiento de una conexión TCP. Este valor influye en la calidad de servicio percibida por el usuario especialmente a nivel de aplicación, en protocolos con múltiples peticiones como por ejemplo HTTP.

Pese a la importancia de este valor, su cálculo pasivo no es una tarea trivial en redes grandes, ya que se debe observar cada paquete de cada conexión para ambos sentidos del tráfico. Este estudio requiere tener en cuenta muchos parámetros tales como desórdenes, retransmisiones, pérdidas durante la captura, etc. Debido a estas dificultades, habitualmente, esta medida se obtiene mediante pruebas activas como por ejemplo, a través de peticiones ICMP (Ping). La gran desventaja de este tipo de medidas es que no siempre es posible realizarlas, bien por restricciones de *Firewalls*, o bien porque a veces no será deseable inyectar más tráfico a un sistema. En este tipo de escenarios sería preferible obtener el valor mediante técnicas pasivas, a través del análisis del tráfico capturado.

Con este objetivo, en este capítulo se ha presentado un algoritmo para estimar el RTT de las conexiones capturadas de forma pasiva. Para estimar su precisión se ha comparado con otros dos métodos pasivos clásicos: a partir de los paquetes de establecimiento de la conexión y a partir de la opción de *Timestamp*, el cuál es un parámetro opcional de las conexiones.

El algoritmo propuesto se basa en el hecho que el emisor no puede mandar más datos que los anunciados por la ventana del receptor. Con esta base se han descrito tres algoritmos que difieren un poco en lo referente a cómo se es de estricto para definir cuál es la ventana máxima anunciada por el receptor que limita el envío de paquetes.

Los algoritmos propuestos, debido a cómo se han definido, dan una sobreestimación del valor real de RTT. Este hecho a menudo es deseado puesto que el valor del RTT es usado para dedicir una escala de tiempo por encima de éste. El estimador del RTT a través de la opción del *Timestamp* cumple también ser un sobreestimador, nunca devuelve un valor por debajo del valor real.

Por otra parte, el estimador a partir del establecimiento de la conexión, puede devolver valores más pequeños que los RTT reales debido a la presencia de

middleboxes. Éstos pueden responder o predecir los establecimientos de las conexiones en nombre de los extremos.

Los algoritmos se han diseñado para escenarios en los que las conexiones TCP no llenen su producto $bandwidth \times delay$, este valor depende del valor del RTT, del ancho de banda del camino y del escalado de ventana que puede ser usado si se anuncia en el establecimiento de la conexión TCP. Este requerimiento se cumple especialmente en escenarios con un ancho de banda alto como podrían ser proveedores de servicios *Over-The-Top* (OTT), o incluso algunas redes de centros de datos. Los estimadores pueden ser usados conjuntamente con el método del *Timestamp* para obtener más muestras de RTT en aquellas trazas que en las que no se pueda realizar esta estimación.

En este capítulo se ha probado el rendimiento de los métodos propuestos para un escenario emulado, comprobándose que los resultados no eran tan exactos como los obtenidos a través del método del Timestamp pero proporcionaban valores suficientemente precisos para ser usados. La medida del *Timestamp* se basa en la la observación activa de los paquetes, es un método muy exacto difícilmente mejorable. La desventaja de éste es que requiere que las conexiones TCP usen esta opción, lo que hoy en día todavía no parece muy habitual.

También se ha comprobado su efectividad en un escenario real, como el tráfico Web de la Universidad Pública de Navarra. Tomando el RTT obtenido por el método del *Timestamp* como el valor adecuado de las conexiones, se observa que para conexiones con RTT normales se obtienen unos errores asumibles. Cuánto más grande es el RTT obtenido por el *Timestamp*, mayor variabilidad se obtienen para el resto de métodos. Para estas conexiones el servidor no manda datos durante cierto periodos lo que provoca que no haya suficiente tráfico para que los estimadores funcionen correctamente.

Durante el estudio del RTT se ha propuesto otro método basado en el cálculo de RTT parciales para cada sentido del tráfico. Este método puede ser una opción interesante para aquellos escenarios en los que ambos extremos envíen datos.

El algoritmo desarrollado ha sido publicado y defendido en la conferencia Internet 2015, [PIM+15b].

CAPÍTULO 5

Identificación pasiva del algoritmo de control de congestión en conexiones TCP

En los capítulos anteriores se han analizado de forma pasiva diferentes medidas del tráfico que indican el rendimiento directo de los servicios, bien sea detectando periodos de no disponibilidad, provocados por interrupciones en los servidores, o bien midiendo el retardo que sufren las conexiones, RTT. Otro factor que influye en el rendimiento de los servicios de red es el algoritmo de control de congestión empleado por TCP en los servidores. El objetivo de este capítulo es la identificación del algoritmo de forma pasiva.

El control de congestión es el algoritmo mediante el cuál se decide qué forma es la óptima para enviar al emisor el mayor número de paquetes en cada RTT sin causarle congestión. El algoritmo básico de control de congestión propuesto en las primeras versiones de TCP se denomina *Additive Increase/Multiplicative Decrease* (AIMD). Este algoritmo se caracteriza porque mientras no detecta pérdidas va aumentando linealmente el número de paquetes enviados en cada turno, es decir, en cada RTT. Cuando se detecta una pérdida se produce un decremento para evitar que se siga congestionando.

Desde su propuesta, se han realizado diversos estudios en los que se pretende optimizar el rendimiento de las conexiones en función del escenario y su-

cesivas implementaciones de TCP han ido incorporando estos algoritmos. Las diferentes variantes de los algoritmos propuestos difieren en la función de crecimiento y decrecimiento, incluso en algunas variantes estas fases son definidas en función de nuevos parámetros que antes no eran considerados, como el RTT o el número de paquetes perdidos. Algunos de estos algoritmos más conocidos son BIC, Cubic, Vegas, Westwood, etc.

Dependiendo de los Sistemas Operativos de los servidores, el algoritmo de control de congestión empleado por defecto será diferente. Por ejemplo, en sistemas de Linux con Ubuntu 14, por defecto el algoritmo empleado es Cubic, mientras que en las distribuciones típicas de Windows 10 sigue siendo *AIMD*.

El objetivo de este capítulo es la identificación pasiva de algunos de los algoritmos de congestión más conocidos.

El resto del capítulo se estructura de la siguiente forma. En la sección 5.1 se introduce el problema y se explican las fases del algoritmo de control de congestión. En la siguiente sección 5.2, se explican las etapas del algoritmo propuesto para la identificación. A continuación se realiza un estudio del número de eventos que se tienen en una red real 5.3, para posteriormente perfeccionar el algoritmo utilizando un escenario emulado para el cuál se conozcan realmente los algoritmos utilizados,5.4. Finalmente se termina el capítulo con las conclusiones, 5.5.

5.1. Introducción

El rendimiento de una conexión puede verse disminuida debido a varios factores: que la red esté congestionada, que el RTT sea muy alto, mal funcionamiento de un servidor o cliente, funcionamiento inesperado de un servicio o debido también a cómo se comporta el sistema ante las pérdidas. Hasta el momento se había analizado principalmente dos de ellos, mal funcionamiento del servicio y el retardo de las conexiones. Continuando con el estudio pasivo del protocolo TCP, este capítulo se centra en un aspecto más concreto del protocolo, la identificación del algoritmo de control de congestión que se esté empleando en el servidor.

El algoritmo de control de congestión es el encargado de limitar cuantos paquetes son enviados al receptor en cada RTT tratando de evitar la congestión en la conexión, ya sea causada bien por saturación en la red o bien en el servidor. Una conexión que utilice el protocolo TCP se encontrará siempre en una de estas fases del algoritmo de control de congestión:

1. *Slow Start*: Es la fase en la que se inicia la conexión o a la que se vuelve tras una perdida de paquetes que se considere grave. Durante esta fase, en cada RTT y cada vez que hayan llegado las confirmaciones por parte del cliente, es decir los ACK, se envían el doble de paquetes que en la fase anterior. El número de Bytes enviados en cada RTT sigue un crecimiento exponencial.

2. Evitación de congestión o *Congestion Avoidance*: Esta fase es la definida como crecimiento en el algoritmo de control de congestión. Se utiliza cuando se espera que aumentar la velocidad pueda provocar pérdidas, por lo que trata de aumentar la velocidad cuidadosamente. La fase del Slow Start es abandonada al llegar a cierto umbral que marca la zona en la que anteriormente se habían observado pérdidas. Este umbral puede depender del estado de conexiones anteriores, o bien, si no se tiene esa información, es definido por el valor por defecto del sistema operativo. Una vez alcanzado dicho umbral se inicia la fase de evitación de congestión cuyo objetivo es el envío óptimo de paquetes que no saturen el camino de la red. El crecimiento en este estado dependerá del algoritmo empleado, inicialmente con el algoritmo AIMD el crecimiento era lineal, pero en las nuevas versiones puede ser exponencial o contener diferentes fases dependiendo con funciones de crecimiento diferentes.

   Cuando se detecta una pérdida la fase de evitación de congestión se abandona y la ventana de paquetes enviados en cada RTT es decrementada. Las pérdidas son detectadas bien por ACK duplicados o bien porque no se ha recibido confirmación para algún paquete pasado un tiempo, *Timeout*. Dependiendo de como sean las pérdidas de esta fase se vuelve al Slow Start, en el caso de que se haya agotado el tiempo sin haber recibido la

confirmación, o se inicia la Recuperación rápida, en el caso recibir ACK duplicados.

En el caso de que la pérdida sea detectada por el disparo de un *Timeout*, se guarda la ventana máxima actual y se inicia, como ya se ha citado, la fase de *Slow Start*. Este tiempo en muchos sistemas operativos es por defecto 200ms aunque varía en función del RTT detectado para cada conexión. El umbral para abandonar ahora la fase de *Slow Start*, dependerá del valor de la ventana máxima observada en la fase anterior y al factor por el que se multiplica la ventana antes de las pérdidas para obtener este umbral, β. En el algoritmo AIMD original $\beta = 0{,}5$, es decir, se vuelve a la fase de *Congestion Avoidance* al alcanzar la mitad del valor de la ventana que había justo antes de las pérdidas. El parámetro β dependerá del algoritmo de control de congestión que esté siendo utilizado.

3. Recuperación rápida, *Fast Recovery*: En caso de que esta pérdida sea detectada por haber recibido 3 ó 2 ACK duplicados por parte del cliente, se inicia la fase de *Fast Recovery* retransmitiéndose por *Fast Retransmit*. En estos casos la ventana no sufre un gran decremento y la fase de evitación de congestión continúa cuando se reciben confirmaciones correspondientes a datos nuevos.

En la figura 5.1 pueden ser observadas todas las fases descritas del protocolo TCP.

El algoritmo básico de control de congestión es el AIMD [Aim], el cuál es implementando en algoritmos más conocidos cómo Reno [V.J90], y NewReno [New]. Desde su nacimiento, se han propuesto otros algoritmos de congestión para ser utilizados en diferentes redes y obtener mejores rendimientos. Algunos de los algoritmos implementados por los sistemas operativos son: BIC [XHR04], Cubic [HRX08], Vegas [BOP94], HighSpeed [Hig], HTCP [Htc], Illinois [LBS06], Westwood [MCG+01], CTCP [TSZ+05].

Como se ha citado anteriormente, los algoritmos propuestos intentan optimizar el rendimiento de los conexiones en escenarios concretos. Para ello varían el parámetro β y en algunos casos otros parámetros. En algunos casos el valor de β difiere del de AIMD para un valor fijo mayor, como en el caso de Cubic donde

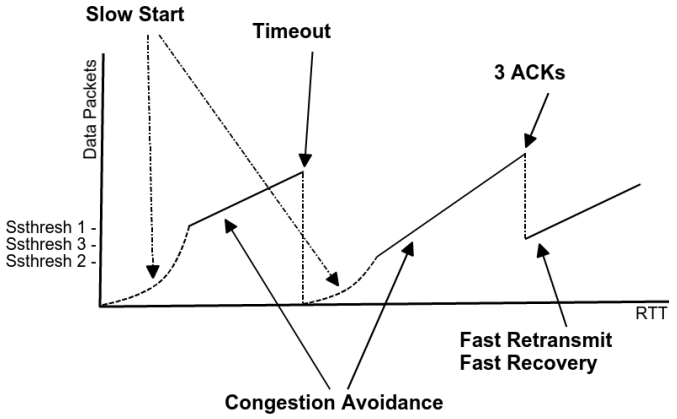


Figura 5.1: Fases del protocolo TCP

$\beta = 0{,}7$, o bien haciendo depender este parámetro del RTT y los paquetes perdidos en las ventanas anteriores, como por ejemplo son los casos de HighSpeed o Illinois por ejemplo. Además algunos algoritmos también cambian la función de crecimiento de la fase de evitación de congestión, por ejemplo, el algoritmo Cubic que puede crecier mediante una función cúbica.

En la tabla 5.1 se muestran de algunos de los diferentes algoritmos de control de congestión, para qué tipo de escenarios fueron diseñados y los sistemas operativos que los soportan.

Debido a que el *trhoughput* de las conexiones pueden verse afectadas en gran medida por el algoritmo de control de congestión que se utilice, en la literatura se pueden encontrar trabajos relacionados con el estudio del rendimiento de las conexiones dependiendo del algoritmo, [GM04; LY01; BM04].

Otros estudios se han centrado en la búsqueda de algoritmos que se adapten mejor a un escenario concreto, [DC15; SW00; JFW+09; DC15; ST14].

El objetivo de este capítulo es la identificación del algoritmo de control de congestión empleado por los servidores en las conexiones, desde un enfoque pasivo. La idea es seguir con la resolución de problemas a través de la observación de las conexiones, sin que se tengan que realizar nuevas peticiones a los servidores de una red.

El algoritmo de control de congestión es un método interno del protocolo TCP que no puede ser elegido a nivel de aplicación. TCP usa control de congestión no asistido por la red, por lo que el emisor decide cuánto enviar según su percepción de las pérdidas y la congestión de la red. No hay negociación entre emisor y receptor para decir qué algoritmo de control de congestión se va a utilizar. El algoritmo es independiente a cada extremo, por lo que cada uno puede utilizar uno diferente. Por tanto, no se puede saber qué algoritmo se está utilizando simplemente observando en el tráfico los parámetros de los paquetes. En Linux, estos parámetros son accesibles mediante las directrices de *sysctl*, y en Windows, es posible conocerlo desde línea de comandos con las directrices *netsh*. Aunque se pueda conocer cuál se está utilizando, los administradores de red no siempre tendrán los permisos o accesos necesarios para averiguar qué tipo de algoritmo se está empleando.

Aunque típicamente los algoritmos de control de congestión estén ligados a sistemas operativos concretos, no se obtendrán resultados fiables al aplicar métodos de *fingerprinting* para identificar los sistemas operativos y de ahí deducir el algoritmo de control de congestión, ya que podrían no haberse utilizado los algoritmos por defecto para esa distribución. Por ejemplo, hoy en día en los sistemas operativos con distribuciones de Ubuntu recientes el algoritmo de control de congestión empleado por defecto es Cubic. No obstante, se pueden escoger entre una amplia familia: Reno, Cubic, HTCP, HighSpeed, Illinois, etc. En Windows, el algoritmo por defecto es NewReno, sin embargo en las últimas distribuciones se puede cambiar para utilizar CTCP.

En la literatura el principal estudio para la identificación del algoritmo es el de [YSL+14]. Los autores proponen un método de identificación por medio de medidas activas y además dejan disponible una herramienta para ser utilizada, CAAI. La idea principal es provocar retransmisiones por *Timeout* en los servidores para calcular el valor en el que se decrementa la ventana cuándo termine la fase posterior de *Slow-Start*, así como aproximar mediante funciones como es el crecimiento posterior. Con este propósito realizan peticiones a los servidores en los que se manipula la conexión del cliente para provocar que el servidor crea que haya pérdidas y lo obligue a abandonar la fase de evitación de congestión.

Tabla 5.1: Escenarios para los que fueron desarrollados los diferentes algoritmos de control de congestión.

Algoritmo	Escenario	Sistema operativo
AIMD	Algoritmo propuesto inicialmente	Linux, Windows
BIC	Redes de alta velocidad y alta latencia	Linux (Kernel 2.6.8-2.6.19)
Cubic	Redes de alta velocidad y alta latencia, derivado de BIC menos agresivo	Linux (Kernel 2.6.19 - 3.1)
Vegas	Enfatiza el retardo de paquete en lugar de la pérdida de paquetes	Linux (Kernel 2.2 - 2.3) y FreeBSD
HTCP	Aumenta la agresividad para redes de alta velocidad y alta latencia. Algoritmo basado en las pérdidas	Linux (Kernel 2.6) y FreeBSD
Highspeed	Diseñado para redes con alto *bandwidth x delay*	Linux (Kernel 2.4.16)
Illinois	Redes de alta velocidad de larga distancia. Se basa en las pérdidas de paquetes para decidir cuánto enviar	Linux (Kernel 2.6.X)
CTPC	Redes con alta latencia	Windows

La identificación posterior se realiza mediante la menor distancia de los coeficientes obtenidos para la función de crecimiento tras el *Slow Start* que sigue al *Timeout* y el valor de β calculado. La distancia se obtiene al comparar los valores nuevos con los que habían obtenido ellos previamente en el escenario de entrenamiento.

La principal desventaja del método descrito es que la identificación se realiza mediante medidas activas y como ya se ha mencionado en otros capítulos, este tipo de medidas son intrusivas por inyectar tráfico extra. Además, dependiendo de la red no siempre será posible forzar nuevas peticiones debido a restricciones de *Firewalls*, por ejemplo.

El objetivo de este capítulo comienza por el estudio del parámetro β. Este parámetro por si solo parece distinguir entre algunos de los principales algo-

ritmos, como NewReno y CUBIC, los algoritmos utilizados por defecto en las últimas distribuciones de Windows y Linux. Sin embargo, cómo se va a explicar en detalle en la siguiente sección, el cálculo de este valor en las conexiones no es una tarea trivial.

5.2. Algoritmo pasivo de medida del parámetro β

El algoritmo de control de congestión se puede describir a groso modo, por la función de crecimiento utilizada en la fase de evitación de congestión y por el parámetro de decrecimento, β, que se aplica cuándo se realiza una retransmisión por Timeout. El parámetro β es el factor utilizado para que la fase de crecimiento rápido, *Slow Start* termine presumiblemente antes de alcanzar tasas, que puedan provocar pérdidas graves y se comience la fase de crecimiento limitado, evitación de congestión.

Como se ha mencionado, cuando los algoritmos de evitación de congestión observan una retransmisión por *Timeout* reconocen ese estado como un punto de pérdidas claras en el canal, por lo que, deciden reiniciar el funcionamiento del algoritmo volviendo a la fase de *Slow Start*. Esta fase, como se ha explicado anteriormente, se abandona dependiendo del máximo que había sido enviado en la anterior etapa antes de las pérdidas graves, es decir, la ventana máxima alcanzada anterior, y el parámetro β. El objetivo de esta fase es recuperar las pérdidas rápidamente pero, terminando antes de llegar a las tasas anteriores en las que se tuvieron pérdidas graves. Por ejemplo, para el algoritmo propuesto inicialmente, AIMD, β tiene un valor de $0{,}5$. Al producirse un *Timeout* saldrá de la siguiente fase, *Slow Start*, al alcanzar la mitad del máximo alcanzado de Bytes enviados en un RTT de la fase anterior.

La identificación del tipo de crecimiento puede depender de varios factores dependiendo del algoritmo. Por ejemplo, el algoritmo BIC aplica una búsqueda binaria dependiendo de los Bytes máximos observados. Debido a que puede depender en gran medida de los estados anteriores y de factores actuales, la identificación del tipo de función de crecimiento que se está utilizando no es una tarea trivial.

Por otro lado, el valor de β podría ser lo suficientemente descriptivo para distinguir el algoritmo de control de congestión. Por ejemplo, si fuera siempre 0.5, independientemente del valor máximo observado de bytes enviados en un RTT de las pérdidas y del RTT, se podría decir con bastante seguridad que se trataría del algoritmo AIMD. Lo mismo pasaría si fuera siempre 0.7 en cuyo caso sería por ejemplo Cubic, o por ejemplo, en el caso de estar entre 0.5 y 0.8

dependiendo de los máximos observados, sería BIC. Por esta razón, este capítulo se centra cómo primera aproximación, en el estudio del parámetro β para tratar de distinguir el algoritmo empleado.

Para llevar a cabo este estudio se tendrán que tener en cuenta parámetros cuyo valor puede que no se conozca con total exactitud, por ejemplo el RTT. Aunque para algunos parámetros se tengan que hacer aproximaciones, si se tienen suficientes conexiones de diferentes clientes con un mismo servidor, en principio, se deberían de tener eventos en los que se pudiera calcular β. Aunque, en algunos eventos la estimación de β fueran aproximaciones poco certeras, al contar con suficientes enventos para realizar una distribución, éstos deberían de ser casos aislados en la distribución y por tanto descartados. Por ejemplo, si un servidor emplea el algoritmo de BIC, al observar todos los eventos posibles de *Timeout* y estimar las β, en la distribución se deberían de destacar los valores en torno a 0.5 y 0.8. Por esta razón, se estudian para un mismo servidor todas las conexiones en las que se observe alguna retransmisión por *Timeout*.

Otro dato a tener en cuenta para la agrupación de eventos de *Timeout* incluso de conexiones de diferentes clientes con un mismo servidor, es que los algoritmos de control de congestión tratan de evitar las retransmisiones por *Timeout*. Pese a ello, debido a problemas puntuales de las conexiones de la red, se siguen observando este tipo de eventos cada cierto tiempo.

Para poder realizar el cálculo de β para cada conexión, se estima un RTT y se contabilizan los paquetes enviados por cada intervalo definido por la estimación. Para la estimación del RTT se podría utilizar el algoritmo propuesto en el capítulo anterior. A partir de la estimación del RTT se aplican las siguientes fases para cada conexión hacia un mismo servidor:

1. **Identificación de retransmisión por Timeout.** El parámetro β se puede calcular después de que se sale de una fase de *Slow Start* producida por una retransmisión por Timeout. Por ello lo primero a identificar son este tipo de eventos. Para la identificación de este tipo de eventos influye el punto de captura. Si se está razonablemente cerca del servidor se tendrá cierta seguridad en que no se están perdiendo los paquetes enviados por el servidor, por lo que se puede emular lo que está recibiendo el servidor. Otro punto a tener en cuenta es que cuando se tienen varios paquetes retrans-

mitidos por *Timeout*, nos interesa el primer evento de éstos. Una vez que se produce uno, se retransmiten todos los paquetes pendientes ya en la fase de *Slow Start*. Para tener eventos puros de este tipo se va a seguir una serie de reglas estrictas con el fin de identificar las siguientes fases correctamente:

a) Se han visto menos de 2 ACKs duplicados. En algunos sistemas operativos basta con recibir 2 confirmaciones duplicadas por parte del cliente para reenviar un paquete. Para evitar calcular a partir de eventos incorrectos, puesto que no eran retransmisiones por *Timeout*, se evita tener en cuenta las retransmisiones producidas tras verse 2 o más ACKs duplicados.

b) El tiempo entre el envío del primer paquete enviado y su retransmisión es de al menos 200ms. Este valor es el mínimo definido inicialmente en los sistemas operativos, al menos en Linux.

c) En el RTT anterior no se había visto ninguna retransmisión. Esta última regla es para evitar señalar eventos que siguen a una primera retransmisión. En esta primera fase se busca el accionador de la siguiente fase, es decir, el evento del primer *Timeout* que provoca el reenvío de nuevos paquetes y el abandono de la fase actual, evitación de congestión.

2. **Seguimiento del Slow Start** El valor de β se define por cuándo termina esta fase y comienza la evitación de congestión. Para distinguir cuándo se acaba esta fase nos basamos en la propia definición; cada vez que se ve en un RTT las confirmaciones de los paquetes enviados, en el siguiente intervalo se envían el doble de paquetes. Por ello, si se divide el número de paquetes de un intervalo con el anterior debería de dar un cociente de 2. Como el RTT puede que no sea del todo exacto se acepta cierto margen de error en este resultado, concretamente de $0{,}5$. Además, si en un intervalo no se tienen paquetes, para el cálculo se tiene en cuenta el anterior con paquetes. De esta forma se comienza a calcular el cociente para los intervalos posteriores de observar un evento de *Timeout*. Para poder realizar una estimación aceptablememente buena, se van a tener en cuenta aque-

llos casos en los que al menos se vean 3 RTT consecutivos que cumplan el Slow Start.

3. **Cálculo de** β. Para el cálculo de β se necesita el valor al salir de la fase de *Slow Start* y el máximo número de paquetes enviados en los últimos turnos anteriores en el que no se había producido ningún *Timeout*.

En la figura 5.2 se muestra el análisis de una traza emulada en un entorno controlado. La traza consiste en una petición HTTP a un servidor Web que ha sido configurado para utilizar el algoritmo de control de congestión New-Reno. Para conseguir una pérdida por Timeout se utiliza la herramienta CAAI, [YSL+14]. Esta herramienta permite emular la retransmisión por Timeout puesto que intercepta los paquetes enviados por el emisor y omite mandar las confirmaciones, obligando a que se produzca la retransmisión por *Timeout*. Además, con esta herramienta se emula un RTT muy alto, de $1s$. Como se puede apreciar en la figura, aún conociendo el RTT y cuándo se produce el Timeout, los paquetes enviados en las siguientes fases difieren ligeramente del valor teórico, debido al propio servicio y al estado de la red.

La traza mostrada anteriormente, figura 5.2, refleja la dificultad de deducir el algoritmo de congestión. Aún en un escenario controlado, en el que el *Timeout* y el RTT son forzados con el fin de lograr una mejor estimación, no se logra un entorno perfecto, aunque es lo suficientemente bueno para diferenciar las fases del algoritmo.

5.2.1. Identificación del algoritmo de congestión de TCP a partir del parámetro β

Para la identificación del algoritmo utilizando el parámetro de decrecimento de ventana, β, se analiza los diferentes valores de este parámetro dependiendo del algoritmo utilizado para algunos de los algoritmos más populares, 5.2.

Se puede apreciar cómo el algoritmo propuesto por Windows, CTCP, tiene la misma β que el algoritmo AIMD. En realidad para esta algoritmo varía la función de crecimiento de la fase de evitación de congestión. Para esta primera identificación el objetivo será clasificarlo junto con el AIMD.

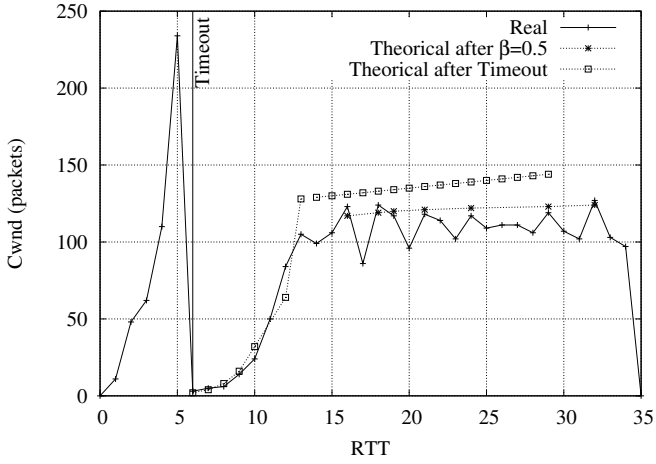


Figura 5.2: Ejemplo de traza en la que se controla cuando se produce una retransmisión por *Timeout* y que usa el algoritmo NewReno para el control de congestión.

Tabla 5.2: Valores de β dependiendo del algoritmo de control de congestión.

Algoritmo	Valores posibles de β
AIMD	0.5
CUBIC	0.7
HTCP	$0{,}5 <= \beta <= 0{,}8$ (Depende del RTT)
HighSpeed	$0{,}5 <= \beta < 0{,}9$ (Depende de las pérdidas)
Illinois	$0{,}5 <= \beta < 0{,}875$ (Depende del RTT)
CTCP	0.5

Cabe destacar que, a la hora de distinguir sistemas operativos a día de hoy, una identificación de AIMD indicaría seguramente un sistema operativo de Windows, puesto que es la β que se tendría por defecto, mientras que una $\beta = 0{,}7$ señalaría a un sistema de Linux, puesto que por defecto utilizan el algoritmo de Cubic.

5.3. Observaciones en un escenario real

Como se ha citado anteriormente, la identificación parte de un evento de retransmisión por *Timeout* que permitirá el estudio del parámetro β. Sin embargo, parte de la dificultad de este método de identificación consiste precisamente en que TCP intenta evitar que los eventos de retransmisión por *Timeout* se produzcan, utilizando para ello herramientas como confirmaciones selectivas y *fast retransmit*. De esta forma se evita la vuelta a la fase de *Slow Start*. A pesar de ello, esporádicamente ocurren este tipo de eventos. La idea es que aunque estos eventos no sean muy periódicos o ni siquiera se observen para todas las conexiones, bastará con que se puedan observar para un cierto número de conexiones de un mismo servidor, aunque el análisis se tenga que prolongar durante varias horas.

Este tipo de análisis se podrá realizar más fácilmente en conexiones con muchos Bytes, ya que el crecimiento no se vería afectado por no tener datos, así como en redes saturadas porque se producirán un mayor número de retransmisiones. En un escenario dónde las conexiones no sean muy largas, ni la red esté saturada, también se podrá llevar a cabo una identificación pero posiblemente, se necesitará de un tiempo mayor de observación puesto que se tendrán menos ocurrencias de retransmisiones por *Timeout*.

Pese a las dificultades mencionadas, se realiza un primer estudio de la identificación sobre tráfico de un escenario real que no cumple con las características de un escenario modelo. El objetivo de este estudio es mostrar cómo la identificación podrá ser realizada en escenarios reales aunque para obtener un número de conexiones aceptable se tenga que prolongar las capturas durante algo más de tiempo.

Con este objetivo se realiza un primer estudio para el tráfico capturado durante unas horas del día, 11/11/2013, en la Universidad Pública de Navarra. El escenario de captura es el mismo que se había descrito ya en anteriores capítulos, figura 3.5 capítulo 3.4. El estudio se comienza para una hora de tráfico ampliándolo posteriormente para una hora más de análisis, aunque los resultados mostrados en este capítulo son los de las dos horas de tráfico, puesto que para una hora se obtenían resultados similares pero con menos información.

5.3.1. Análisis de dos horas de tráfico 10-12h

Para la captura de tráfico se analiza el número de conexiones entrantes, es decir, de clientes externos con servidores de la universidad o cercanos a ella, pertenecientes a la red RedIris. Para obtener más información acerca de los servidores se realiza además pruebas de *fingerprinting* mediante NMAP [Nma] y p0f [Zal], con el fin de obtener más información sobre el posible sistema operativo de cada servidor, y en consecuencia de los posibles algoritmos de control de congestión que puedan estar utilizando. Durante las horas de análisis se destacan algunos servidores de la universidad con más tráfico, tabla 5.3.

Tabla 5.3: Top de servidores con más conexiones.

Conex	Servidor	Sistema operativo	Algoritmos
120772	130.206.159.234	Desconocido	Desconocido
90830	130.206.159.235	Linux 2.4-2.6	RENO, BIC, CUBIC, HighSpeed
12322	130.206.192.25	Linux 3.X	CUBIC, HTCP, HighSpeed
11844	130.206.192.34		
10005	130.206.159.227	Linux 2.6.X—3.X	BIC, CUBIC, HTCP, HighSpeed
9555	130.206.192.47	Linux 3.X	CUBIC, HTCP, HighSpeed
7480	130.206.192.55		
7084	130.206.193.20		
6983	130.206.192.15		
6777	130.206.163.10	Windows XP	AIMD

De los servidores mostrados anteriormente, algunos pertenecen a la red RedIris. Centrando el estudio solamente para aquellos servidores de la UPNA y para los que se observa algún evento de retransmisión por *Timeout*, seguido por

cumplirse al menos 3 RTTs del Slow Start, se tiene el siguiente top de servidores que pueden ser analizados, tabla 5.4.

Tabla 5.4: Top de servidores de la UPNA para los que se observa eventos de *Timeouts* seguidos por la fase del *Slow Start*.

Conex	Eventos	Servidor
120772	594	130.206.159.234
1188	304	130.206.159.48
90830	183	130.206.159.235
10005	103	130.206.159.227
1272	92	130.206.158.227
2794	44	130.206.158.247
755	39	130.206.159.49
789	20	130.206.159.239
490	14	130.206.169.245

Aunque en la tabla anterior, 5.4, se mostraban servidores para los que se habían contabilizado bastantes eventos de retransmisiones por *Timeout*, este primer estudio se va a centrar para los dos servidores principales para los que además se tenían muchas conexiones, 130.206.159.234 y 130.206.159.235.

En la figura 5.3 se muestran los histogramas de las β estimadas para los dos servidores para las dos horas de tráfico capturado, de 10-12 de la mañana.

A pesar de contar en un principio con bastantes eventos de *Timeout*, muchos tuvieron que ser desestimados para la estimación de β, ya que no se apreciaba que se hubiera alcanzado una ventana lo suficientemente alta como para ser comparada después de la fase del *Slow Start*. A pesar de ello, en el caso del servidor 130.206.159.234 parece que empieza a destacar un valor de β de $0{,}5$. Este valor podría corresponder a cualquier algoritmo, excepto el Cubic, porque el resto puede tomar este valor como medida conservadora.

En el caso del servidor 130.206.159.235, descartando los valores pequeños, se destaca la aparición de estimaciones de $\beta > 0{,}5$. Estos valores podrían corresponder al algoritmo BIC o incluso Highspeed. Para un mejor análisis se debería de continuar el estudio para tener un histograma con suficiente muestras.

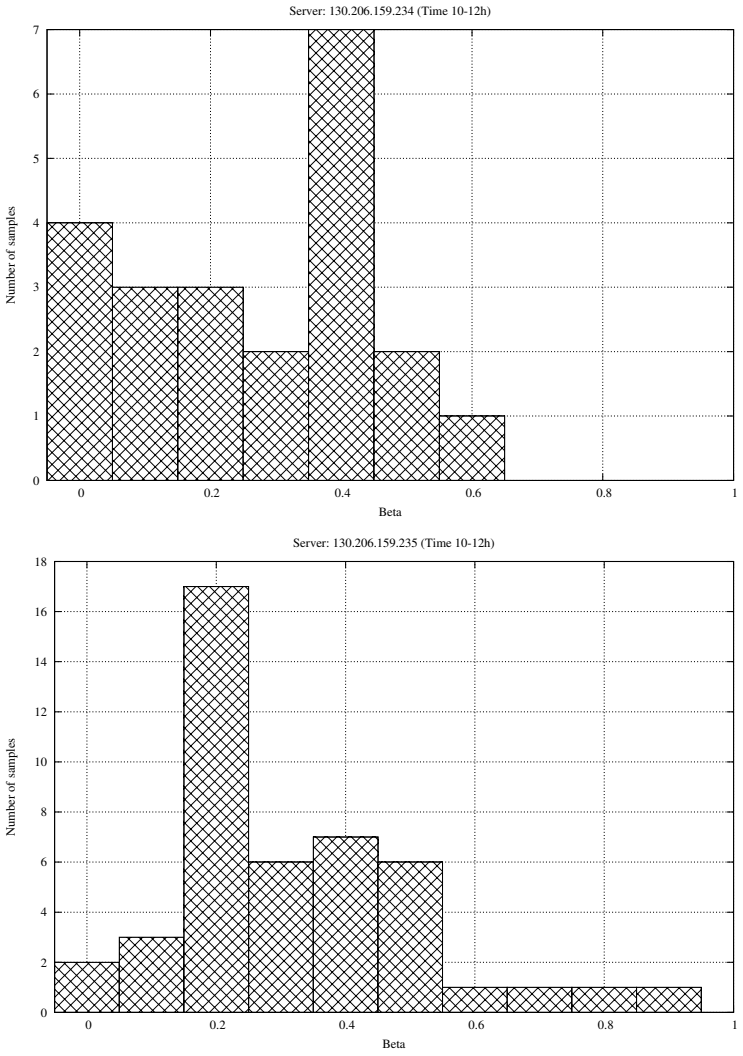


Figura 5.3: Histograma de β obtenidas para los servidores 130.206.159.234 y 130.206.159.235 de 10-12h de la mañana

En el histograma, se observan valores muy pequeños que no parecen corresponder a ningún algoritmo. Estos valores son ocasionados por comportamientos inesperados como por ejemplo, que aunque se haya salido de la fase del *Slow Start*, en el siguiente intervalo no se tengan datos que mandar y se manden me-

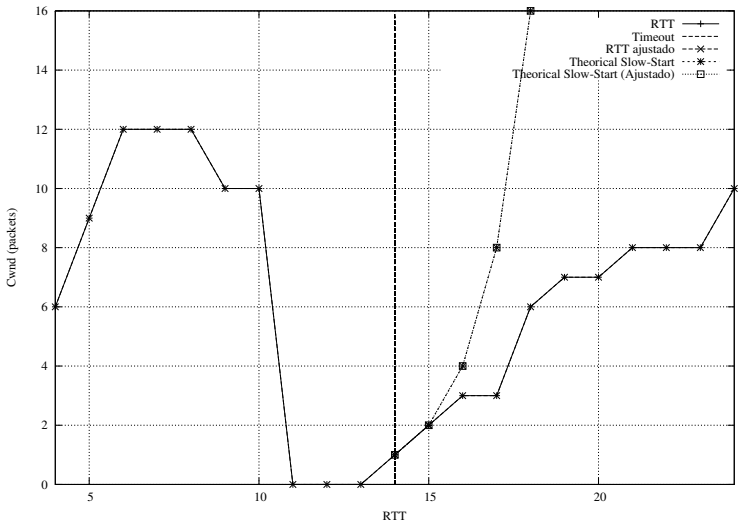


Figura 5.4: Traza de tráfico de un cliente externo a la UPNA y cuyo servidor era 130.206.159.235, para el cuál se calculaba una $\beta = 0{,}250000$

nos paquetes de los esperados. En este caso, se realizaría una subestimación de β. Este ejemplo se puede observar en la figura, 5.4. La figura es un ejemplo de una traza de un cliente externo con el servidor 130.206.159.235 para el cuál se calculaba una $\beta = 0{,}25$. En la figura se muestra la leyenda "Rtt Ajustado", que en este caso es igual al RTT fijo, en la cuál se calcula el número de paquetes para RTTs que comienzan a partir del paquete retransmitido por *Timeout*. El RTT fijo corresponde a los paquetes capturados en intervalos fijos de tiempo desde el inicio de la traza. En esta figura se aprecia como después de la tercera etapa de la fase de *Slow Start* se mandan 3 paquetes en lugar de los 8 si siguiera en esta fase, o 6 si la terminase.

Otros valores erróneos son debidos a que no se logra realizar una estimación buena debido a errores de cálculo en las fases de *Slow-Start* por problemas en la aproximación del RTT. En la figura 5.5, se muestra un ejemplo de este tipo.

Los errores causados por comportamientos inesperados se espera que representen un porcentaje pequeño y no tengan peso significativo en los histogramas al realizarse un estudio para un número mayor de estimaciones de β. De hecho, como se ha citado anteriormente, inicialmente se había realizado el estudio

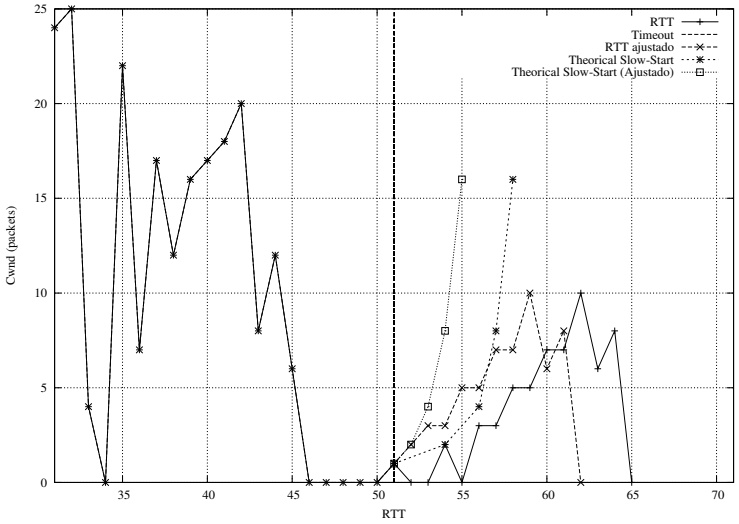


Figura 5.5: Traza de tráfico de un cliente externo a la UPNA y cuyo servidor era 130.206.159.235, para el cuál se calculaba una $\beta = 0{,}5$

para solamente una hora de tráfico. Al obtener pocas estimaciones de β con las que trabajar, se realizó el estudio de la hora siguiente. Al coger sólo una hora más se observaba como el número de eventos seguía aumentando, por lo que si se continuara el análisis se tendría el suficiente para la identificación. Además, algunos de los errores de las estimaciones de β pueden ser subsanados depurando el algoritmo en un escenario en el que se conozca de antemano cuál es el comportamiento esperado y por tanto se entienda cómo debería de haberse calculado. Con este objetivo, se evalúa el parámetro β para las conexiones de un escenario emulado dónde se tenga total conocimiento de los algoritmos y métricas de TCP utilizadas. Como línea futura se aplicará el algoritmo depurado a las trazas de un escenario real como el analizado el analizado en este apartado.

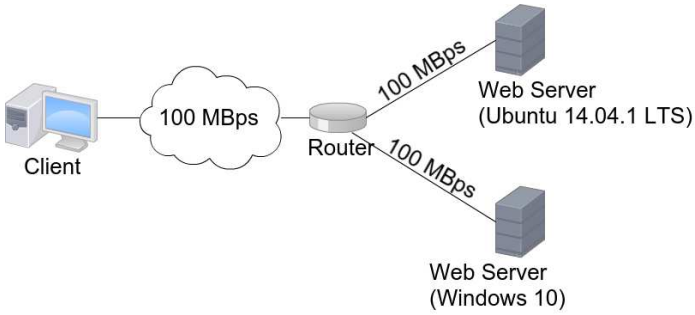


Figura 5.6: Escenario en el cuál se va a estudiar el parámetro β

5.4. Escenario emulado

En el escenario real académico analizado, se observaba cómo los principales servidores internos parecían ser servidores Linux, y si no se cambiaron específicamente en el sistema, los algoritmos de control, seguramente serían Bic o Cubic dependiendo de la novedad de la distribución de Linux utilizada.

Para poder perfeccionar el algoritmo de clasificación, en una primera fase, se debe de conocer realmente qué algoritmo implementan los servidores para decidir si se está clasificando correctamente o no los eventos observados. Además, mediante un escenario emulado se pueden probar diferentes algoritmos de control de congestión.

El escenario emulado para estudiar el comportamiento de β a través de la observación pasiva de las conexiones, es el que se describe en la figura 5.6.

El escenario cuenta con dos servidores Web instalados en diferentes sistemas operativos, uno en Linux, concretamente con la distribución de Ubuntu 14.04, y otra en Windows 10. Un cliente realizará peticiones HTTP hacia los servidores. Es en el router dónde se capturan las conexiones para su posterior análisis. De esta forma se captura en un punto cercano al servidor.

Se emularán diferentes configuraciones de red, en las que se varía tanto el RTT como diferentes pérdidas. Con el objetivo de tener bastantes eventos de Timeout con tan sólo unas pocas conexiones para cada uno de los algoritmos de control de congestión probados, se provocan pérdidas en el router modificando durante un intervalo de tiempo pequeño las de reglas del *Firewall*.

Esta situación tiene similitudes además, con una situación real. En un escenario real sería un caso de pérdidas debidas bien a que el router está saturado y tira paquetes, o bien por problemas en el interfaz.

Se han realizado pruebas para los dos escenarios que se describen en la tabla 5.5.

Tabla 5.5: Descripción de los escenarios emulados.

Escenario	RTT (ms)	Pérdidas
Entrenamiento	200	Pérdidas constantes cada 5s y de 100ms de duración
Validación	50	Perdidas de unos 30ms de duración producidas aleatoriamente cada cierto tiempo

El escenario de entrenamiento, que es el que tiene un RTT elevado, es el que se ha utilizado para el estudio y elaboración del algoritmo de clasificación según los valores obtenidos de β. Para este escenario se ha escogido un RTT alto así como intervalos claros de pérdidas para obtener rápidamente numerosos eventos de *Timeout* en las conexiones. Además, un RTT alto facilita el estudio visual de cómo son las funciones de crecimiento en cada fase, lo que permite reajustar la fase de identificación de *Timeouts* y el estimador de β.

El segundo escenario, el de validación, tiene unos valores de latencia más similares a los que se espera encontrar en un escenario real. Las pérdidas son también más variables para emular un escenario en el que un router pierda bloques de paquetes por saturación durante intervalos pequeños de tiempo.

En el primer escenario, además del algoritmo de control de congestión, se va a estudiar el efecto de parámetros de TCP que afectan directamente al rendimiento de las conexiones así como para el algoritmo de control de congestión. Estos parámetros son en el escalado de ventana, *Wscale*, y las confirmaciones selectivas *SACK*. Mediante el primero, servidores y clientes acuerdan en el inicio de la conexión utilizar ventanas de recepción mayores, por lo que a los servidores se les permite enviar un mayor número de paquetes. Al enviar más paquetes por intervalo se pueden dar mayor variación en las β observadas para aquellos algoritmos en los que β dependa del número de paquetes perdidos en intervalos

anteriores.

El segundo parámetro, *SACK*, también es anunciado en el inicio de la conexión. El objetivo de este parámetro es que se produzcan menos retransmisiones por Timeout ya que confirman paquetes que se han recibido posteriormente a uno pérdido. De esta forma se reenvía mediante *Fast Retransmit* los paquetes perdidos. Este parámetro provocará que se observen menos eventos de *Timeout* para un escenario con idéntico RTT y probabilidad de pérdidas.

Típicamente los sistemas operativos tienen activado ambos parámetros puesto que se consigue un rendimiento notablemente mayor. Por esta razón, el análisis y el algoritmo de clasificación se realizará para los datos observados en la segunda configuración.

5.4.1. Caracterización del escenario de entrenamiento

Para poder analizar los valores de β en un escenario lo más simple posible se realizan peticiones al servidor Web que contesta con un tamaño fijo de página de 27 MB. Como se ha citado anteriormente se prueban dos configuraciones, en la primera (configuración simplificada) con los parámetros de *Window Scale* (WSCALE) y *Selective Acknowledgement* (SACK) desactivados, y una segunda (configuración por defecto) con los parámetros típicos en los sistemas operativos y con los que se obtienen mejores rendimientos.

En ambas configuraciones se prueban diferentes algoritmos de congestión para cada servidor, para el servidor Linux se prueban: Reno, Cubic, HTCP, HighSpeed e Illinois. En el caso del servidor de Windows se prueban: NewReno y CTCP. Por cada configuración se recogen 100 peticiones realizadas por el cliente. Aunque pueden parecer pocas conexiones, al estar constantemente realizándose cortes en la red, se producen en cada conexión muchos eventos de *Timeouts* que permitirán analizar la β obtenida dentro de una misma conexión.

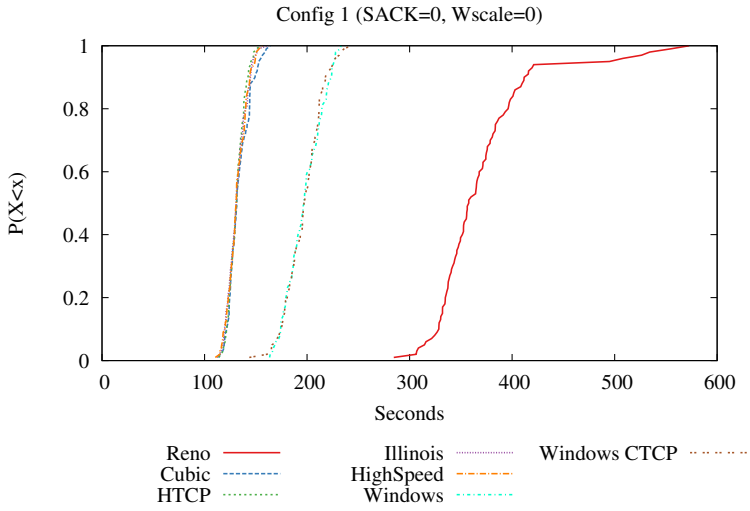
En la figura 5.7 se muestra las distribuciones acumuladas de la duración de las conexiones obtenidas para cada algoritmo en cada configuración. Al tratarse de un escenario emulado en el que el cliente siempre obtiene el mismo tamaño de página, en principio no debería de haber mucha variabilidad en las duraciones. Sin embargo, debido a los cortes se observa cómo algunas conexiones sufren

demora. Además se observa una importante diferencia entre los algoritmos de congestión empleados.

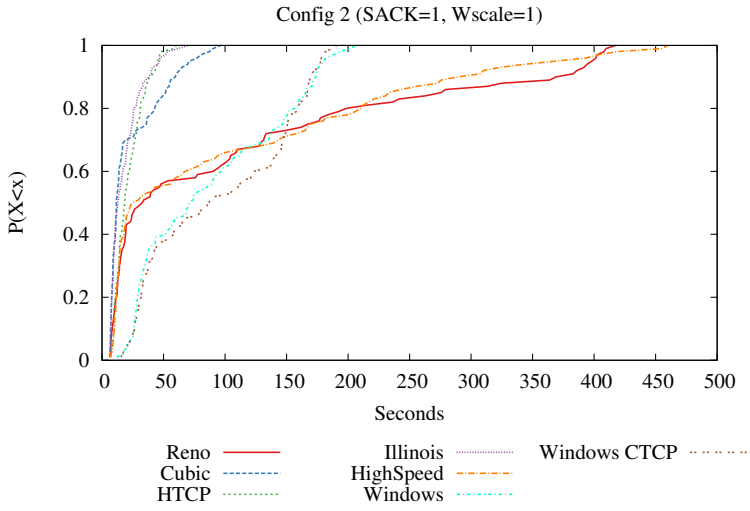
En el escenario de entrenamiento con la configuración simplificada, destaca la longitud de la duración de las conexiones con el algoritmo Reno frente al resto de algoritmos. Para poder comparar mejor el resto de algoritmos de esa configuración, se grafican las distribuciones en una gráfica a parte para el resto , figura 5.8.

Se puede apreciar cómo los algoritmos que obtienen mayor velocidad y por tanto la duración de las conexiones son más corta, son el Cubic, HTCP e Illinois. Estos escenarios son un ejemplo de cómo puede afectar el algoritmo de control de congestión en el rendimiento de las conexiones.

Si se estudia el comportamiento de los algoritmos a nivel del número de retransmisiones, figura 5.9, se observa cómo en el caso de la configuración por defecto, los algoritmos con un mayor número de paquetes retransmitidos son curiosamente los que obtenían menores duraciones. En estos casos, las retransmisiones son pérdidas que se recuperaban en su mayoría por *Fast Retransmist*, lo que consigue disminuir la duración de cada conexión. Además, este tipo de algoritmos permite enviar más debido a su función de crecimiento, figura 5.10, por lo que también se completar antes la petición.



Configuración simplificada



Configuración por defecto

Figura 5.7: Distribución acumulada de la duración de las conexiones por algoritmo para cada configuración

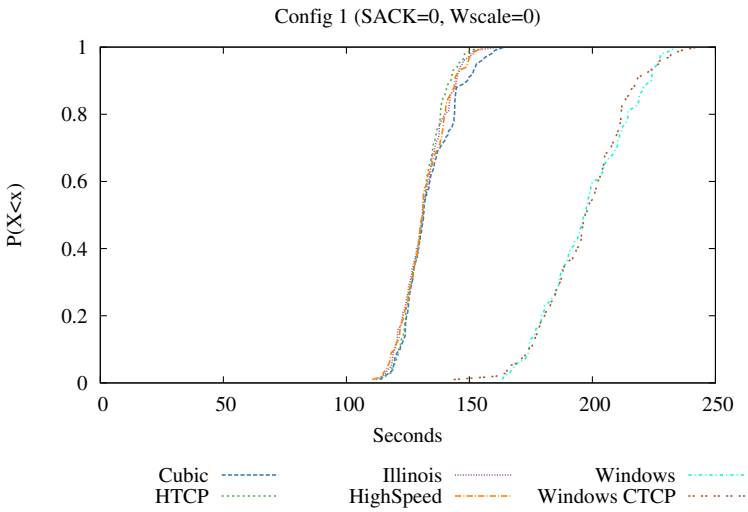
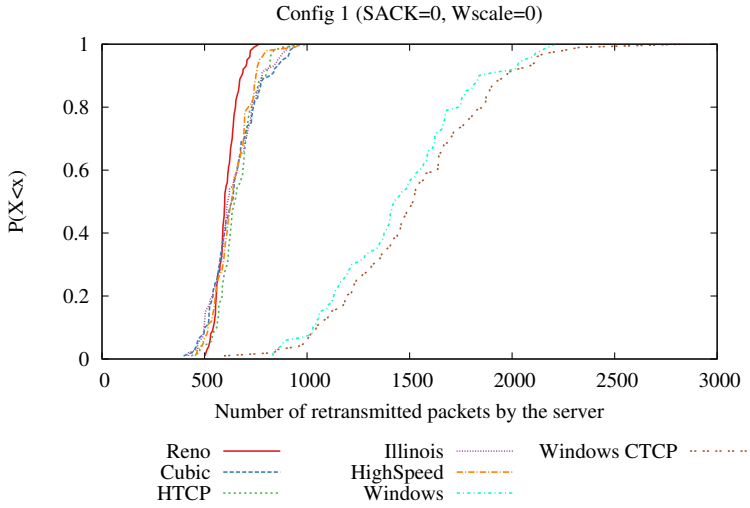
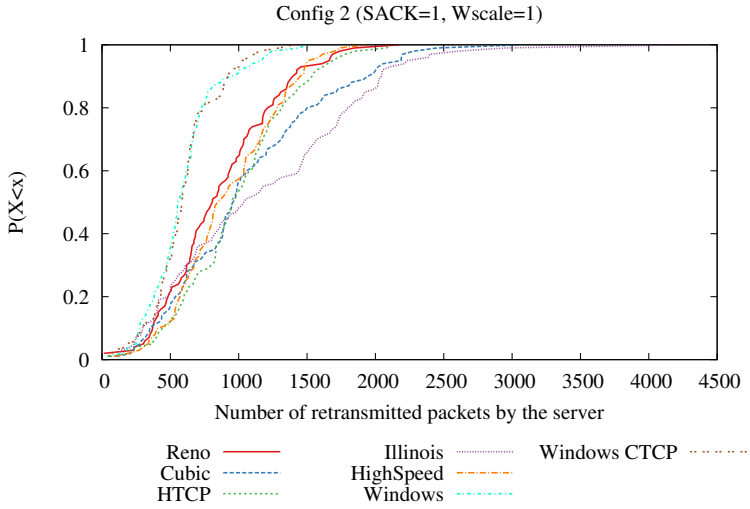


Figura 5.8: Distribución acumulada de la duración de las conexiones por algoritmo para la configuración simplificada quitando el algoritmo de Reno

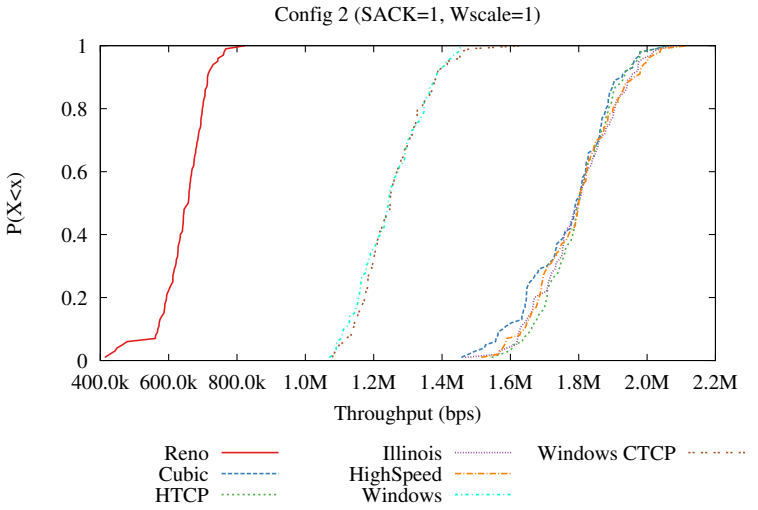


Configuración simplificada

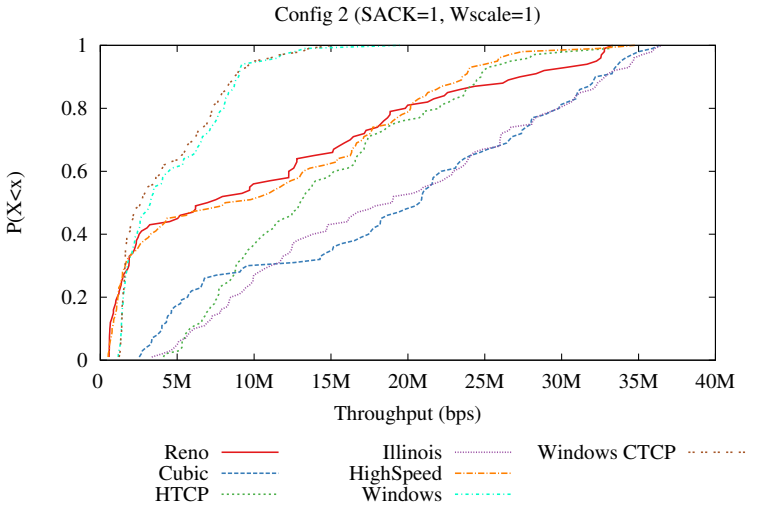


Configuración por defecto

Figura 5.9: Distribución acumulada del número de retransmisiones por algoritmo para cada configuración.



Configuración simplificada



Configuración por defecto

Figura 5.10: Distribución acumulada del throughput alcanzado por las conexiones.

5.4.1.1. Resultados de la configuración simplificada, (no SACK, no WScale)

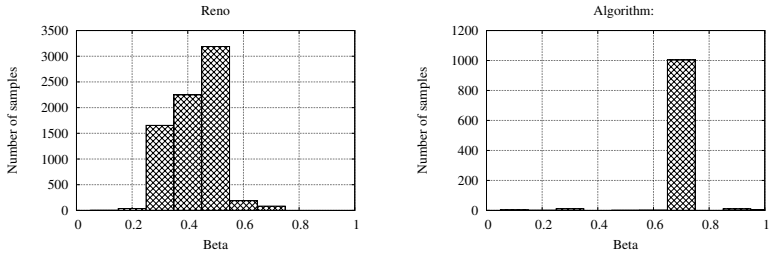
El objetivo del análisis para esta configuración del escenario es la obtención de valores de β los más puros posibles, sin que estén influidos por parámetros como el *SACK*. Para esta primera configuración se calculan las β observadas a partir de la identificación de eventos de *Timeouts* y el final de la fase posterior, *Slow Start*. En la tabla, 5.6 se muestran el número de eventos que se tienen en el escenario para cada algoritmo.

Tabla 5.6: Eventos de Timeouts que se tienen para cada algoritmo, escenario de entrenamiento, configuración simplificada.

Algoritmo	Num Eventos
Reno	7944
Cubic	1039
HTCP	1204
HighSpeed	1045
Illinois	1041
Windows	1247
CTCP	1153

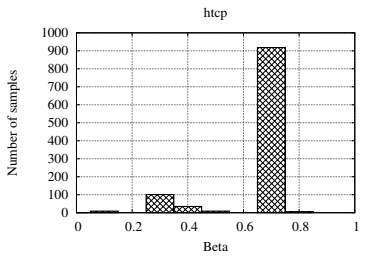
En la figura 5.11 se muestran los histogramas de las β obtenidas para cada algoritmo. En el caso del algoritmo Reno cuya β debería de ser $0{,}5$, en algunos casos se tiene valores de $0{,}571$ provocados porque en el último intervalo del *Slow Start* se salía con 8 paquetes en lugar de 7, puesto que 14 era el máximo número de paquetes enviados en un intervalo que se había alcanzado desde el evento anterior al *Timeout* siendo analizado.

Para comprender mejor por qué la desviación de las β observadas en cada caso se va analizar cada algoritmo de forma individual.

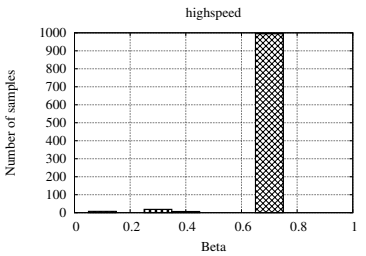


Reno

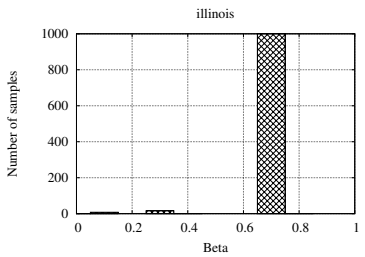
Cubic



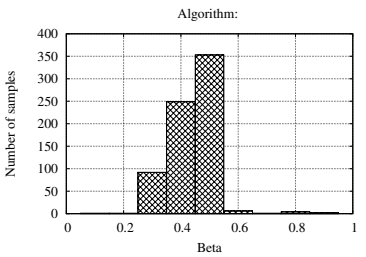
HTCP



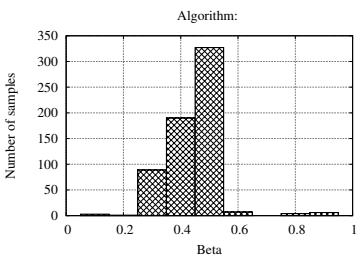
HighSpeed



Illinois



Windows



Windows CTCP

Figura 5.11: Histogramas de β obtenidas para cada algoritmo (escenario de configuración 1).

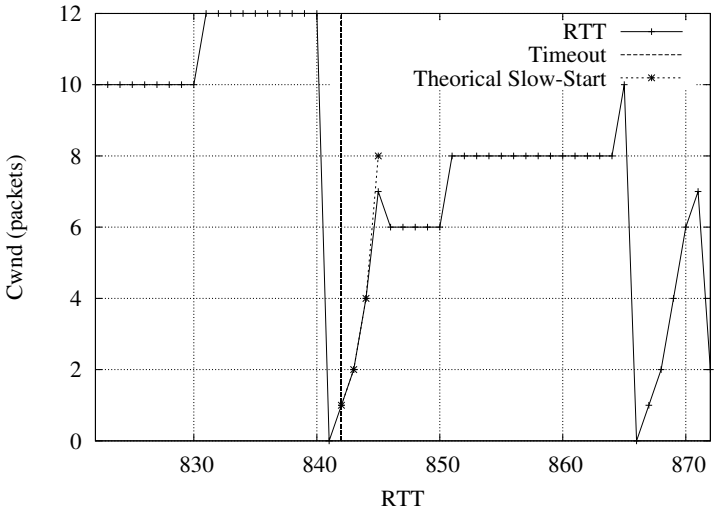


Figura 5.12: Ejemplos de traza del algoritmo Reno para la que se calcula una β de $0{,}33$ (Configuración simplificada).

Reno Al algoritmo de Reno por definición tendría que tener una $\beta = 0{,}5$. En los histogramas anteriores, figura 5.11, se observan algunos valores por debajo de $0{,}4$. Este tipo de valores son provocados por pequeños errores producidos en la etapa en la que se define el valor con el que ha acabado el *Slow Start*. Por ejemplo, en el caso de la traza mostrada en la figura 5.12 se aprecia que la fase del *Slow Start* termina enviando 7 paquetes en lugar de enviar 6, que eran los esperados puesto que el máximo enviado en un turno en la fase anterior era 12.

Utilizando este algoritmo de control de congestión, para una misma trazas se llegan a ver hasta 78 eventos de *Timeouts*, lo que explica por qué las conexiones duraban tanto. Al producirse tantos eventos seguidos, el número de paquetes enviados por el servidor en cada intervalo no termina de crecer y por eso hay tanta diferencia en cuánto a la duración de las conexiones de este algoritmo con respecto al resto. En la figura 5.13, se muestra un ejemplo de traza con dos eventos de *Timeouts* casi seguidos.

En el histograma anterior se observaban algunos valores de $0{,}7$ e incluso por encima. Curiosamente, aunque estos valores son pocos, todos se daban en el primer evento de Timeout que era observado en la traza. En la figura 5.14 se

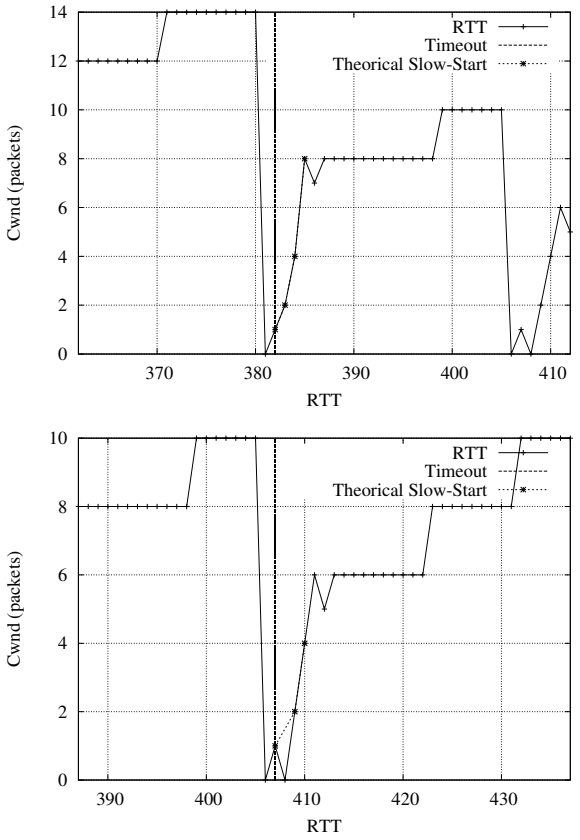


Figura 5.13: Ejemplos de una misma traza del algoritmo Reno con dos eventos de *Timeouts* muy seguidos, para la que se calcula una β de 0.57 y 0.6 respectivamente.

dibuja la distribución de β observada dependiendo del número de evento en cada conexión. Cómo para algunas trazas se llegan a observar muchos eventos sólo se dibujan las distribuciones obtenidas para los 10 primeros eventos de cada conexión.

Aunque las β obtenidas para el primer evento observado son en media ligeramente mayores, en la distribución anterior 5.14, se observa cómo no ocurre siempre, y en media se tendrá un valor cercano al teórico, $0{,}5$. Esta distribución muestra cómo el valor de β es independiente de cuándo tenga lugar el evento

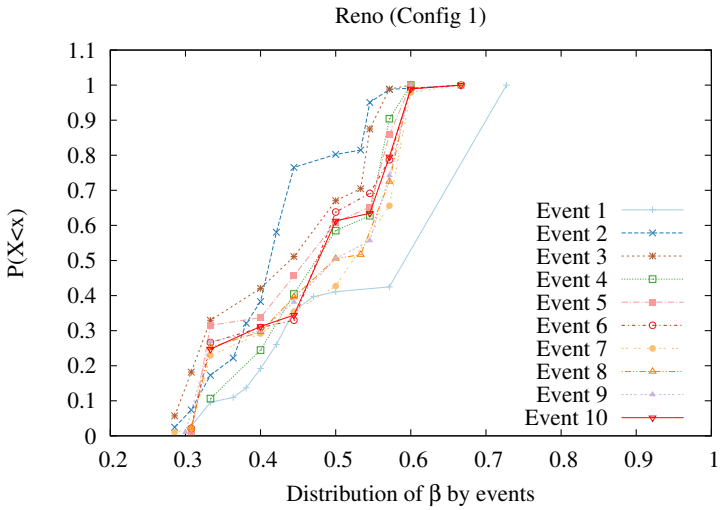


Figura 5.14: Distribución de β por eventos para el algoritmo de Reno (Configuración simplificada).

de *Timeout*.

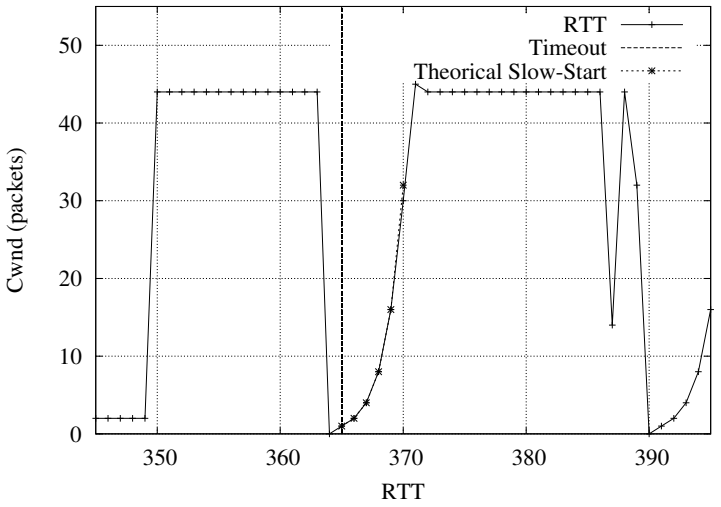


Figura 5.15: Ejemplo de traza para la que se calcula una $\beta > 0{,}7$ para el algorimo de Cubic (Configuración simplificada).

Cubic Para el algoritmo de Cubic se espera obtener valores de $\beta = 0{,}7$. Cómo se pueden producir errores en los cálculos lo normal sería tener valores aproximados a este valor. En el histograma mostrado al inicio de esta sección, figura 5.11, se apreciaba cómo la mayoría de valores estaban dentro del cubo de $0{,}7$.

Los valores del histograma mayores de $0{,}7$ son debidos a que se da por terminado la fase del *Slow Start* un intervalo más tarde del real, debido al error máximo permitido para su cálculo. En el ejemplo de la figura 5.14 se muestra una traza en la que ocurre este hecho. En dicha conexión, el máximo de paquetes alcanzado en el anterior evento al *Timeout* era de 44. El *Slow Start* se determina que finaliza con 45 paquetes y en el intervalo anterior se tenían 30, por eso el cociente entre el intervalo anterior y ese es de $1{,}5$. De esta forma la β se evalúa un intervalo después de lo que se debería. Como nuevamente los casos de este tipo son pocos en comparación con los valores buenos obtenidos, se mantiene el margen de error para determinar que se ha salido de la fase de *Slow Start*.

Con respecto al anterior algoritmo, Reno, se destaca la disminución del número de eventos de *Timeout* que se llegan a observar para una misma conexión, como máximo se observan 18. Análogamente que en el caso anterior se

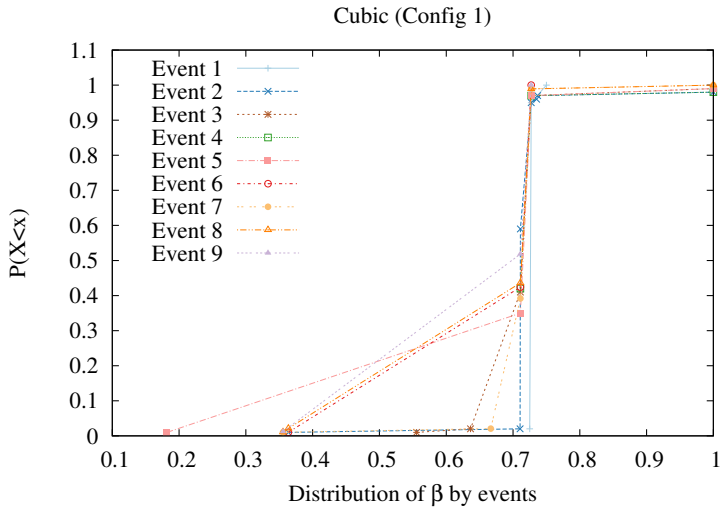


Figura 5.16: Distribución de β por eventos para el algoritmo de Cubic (Configuración simplificada).

grafica las distribuciones de β obtenidas dependiendo del número de evento, aunque sólo se muestran aquellos con al menos 100 muestras, que se observó en cada conexión, 5.16.

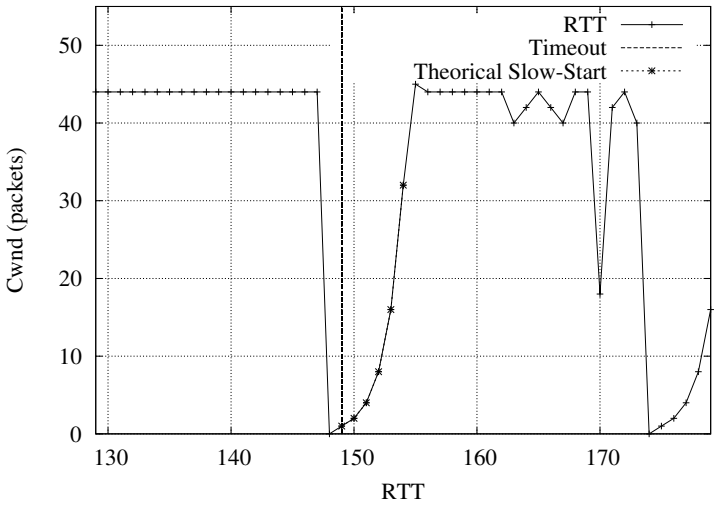


Figura 5.17: Ejemplo de traza para la que se calcula una $\beta = 0{,}7$ para el algorimo de HTCP (Configuración simplificada).

HTCP En el caso del algoritmo HTCP los valores esperados de β pueden ir desde $0{,}5$ hasta $0{,}8$. Como el escenario es muy estático, no hay parámetros que varíen el RTT percibido por el servidor, por ello es lógico que se tengan valores de β alrededor de una misma cifra, en este caso $0{,}7$.

En la figura 5.17 se muestra un ejemplo de una traza dónde el servidor utilizaba el protocolo HTCP.

Respecto al número de eventos de Timeouts que se llegan a observar en una conexión, el máximo es de 22, pero a partir del evento 10 el resto de eventos se producen en una sola conexión. Nuevamente, se estudia la distribución de las β obtenidas para cada evento 5.18, aunque sólo se muestran aquellos eventos con al menos 80 muestras. Se observa cómo el valor obtenido de β es independiente a cuándo ocurrió en la conexión.

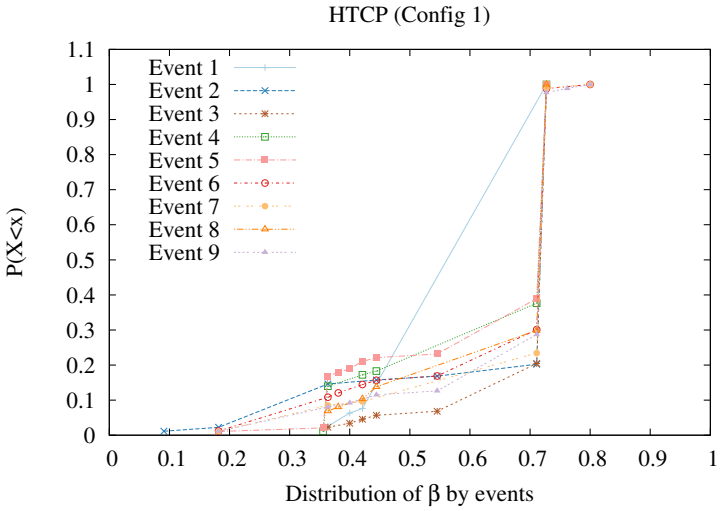


Figura 5.18: Distribución de β por eventos para el algoritmo de HTCP (Configuración simplificada).

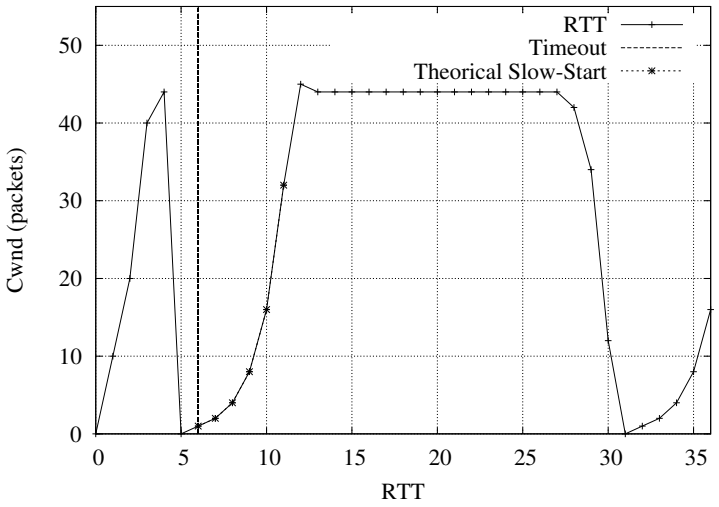


Figura 5.19: Ejemplo de traza para la que se calcula una $\beta = 0{,}7$ para el algorimo de HighSpeed (Configuración simplificada).

HighSpeed Los valores esperados de β deberían comprenderse entre $0{,}5$ y $0{,}9$, dependiendo del parámetro indicador de pérdidas, observado en los intervalos anteriores. Al tratarse de un escenario muy estable los valores obtenidos nuevamente se centran en su gran mayoría en $0{,}7$, como se apreciaba en la figura 5.11. En la figura 5.19 se muestra una traza de ejemplo para la cual se ha calculado $\beta = 0{,}7$.

Mediante este algoritmo de control de congestión y este escenario se obtienen trazas con hasta 17 eventos de *Timeouts* por conexión, aunque la mayoría no pasa de los 12. La distribución obtenida para cada β dependiendo del evento revela que no parece existir relación entre cuándo ocurre el evento y el valor de β observado.

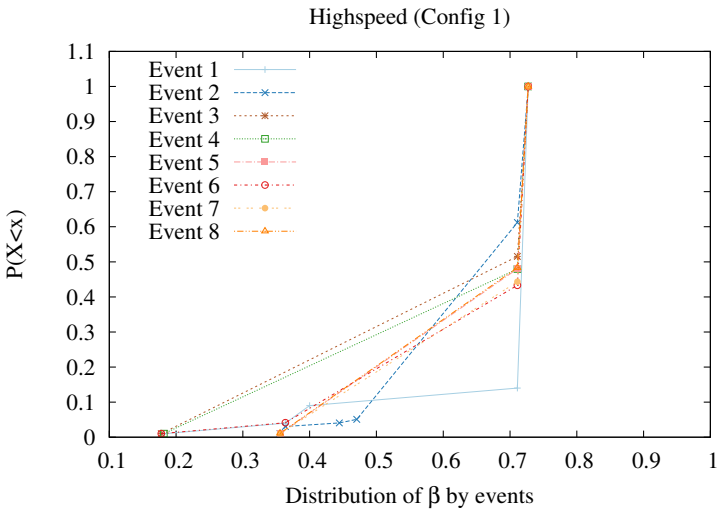


Figura 5.20: Distribución de β por eventos para el algoritmo de HighSpeed (Configuración simplificada).

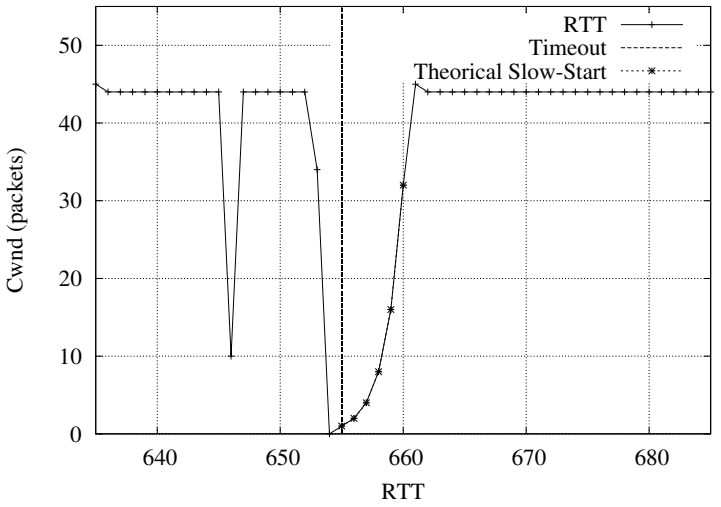


Figura 5.21: Ejemplo de traza para la que se calcula una $\beta = 0{,}7$ para el algorimo de Illinois (Configuración simplificada).

Illinois Este algoritmo tiene varios valores de β dependiendo del retardo de la red. Sin embargo, en el escenario emulado los valores obtenidos caen en su gran mayoría en la caja del $0{,}7$, figura 5.11. Este fenómeno se explica por la invariabilidad de las condiciones de la conexión debido a las características de esta configuración simple.

Al igual que para los casos anteriores se muestra una gráfica dónde se observa un *Timeout* y el crecimiento posterior en la fase *Slow Start*, 5.21

El análisis de las distribuciones de β dependiendo del número de evento en el que ocurrió en la traza no revela anomalías, figura 5.22. Al igual que para los casos anteriores, en la gráfica sólo se dibujan las distribuciones para los eventos con más muestras.

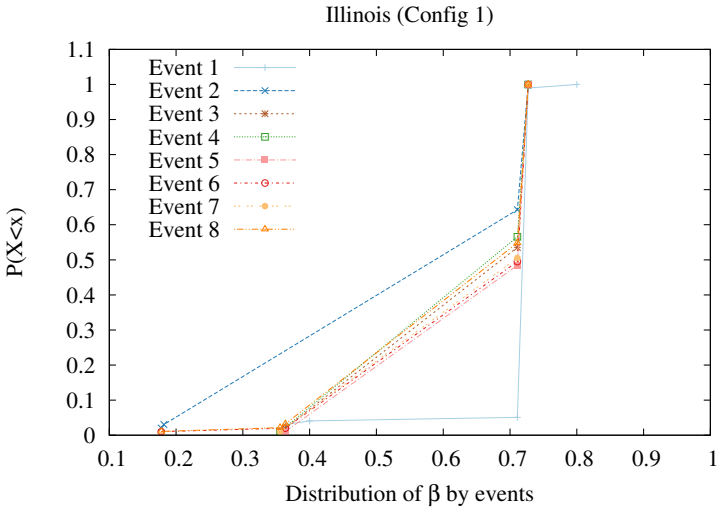


Figura 5.22: Distribución de β por eventos para el algoritmo de Illinois (Configuración simple).

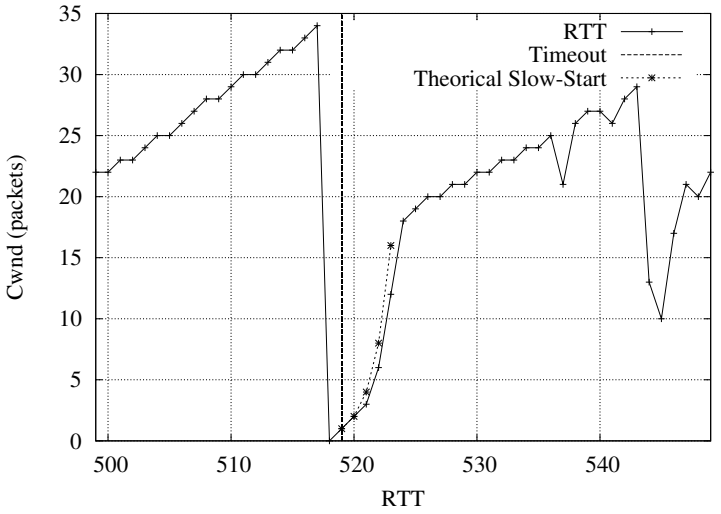


Figura 5.23: Ejemplo de traza para la que se calcula una $\beta = 0{,}5$ para el algorimo por defecto de Windows (Configuración simplificada).

Windows Para el caso del algoritmo por defecto de Windows, si se grafica de la misma forma que en los casos anteriores, se observa fácilmente cómo se sale de la fase de *Slow Start* cuándo $\beta = 0{,}5$. Ejemplo de traza obtenida con el servidor de Windows y la configuración simple, 5.23.

El análisis de distribuciones de β no revelaba anomalías para su estimación dependiendo del evento.

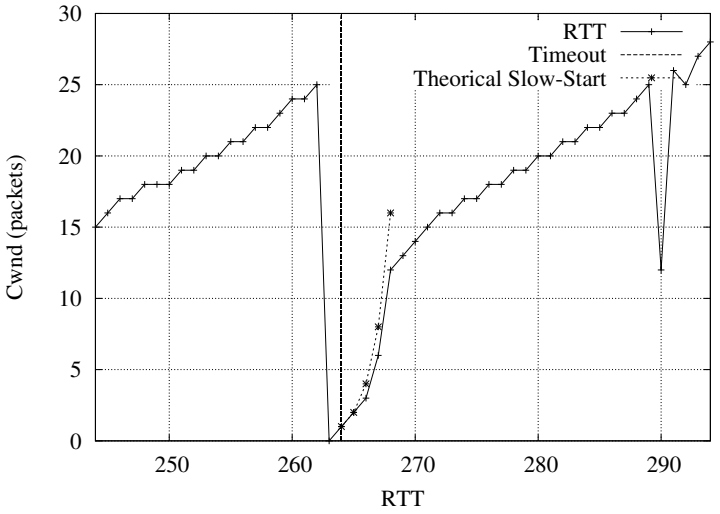


Figura 5.24: Ejemplo de traza para la que se calcula una $\beta = 0{,}5$ para el algorimo CTCP de Windows (Configuración simple).

WindowsCTCP Finalmente, se analiza el algoritmo CTCP impuesto en el servidor de Windows 10. Por defecto, estas distribuciones tienen activado el algoritmo de control de congestión de Reno, sin embargo modificando el registro en los casos de las distribuciones de windows 8 y 10, y a través de líneas de comandos en Windows 7, se puede cambiar el algoritmo para usar CTCP. Aunque la β esperada debería de ser la misma que la de AIMD, $0{,}5$, el crecimiento posterior podría variar ligeramente.

Ejemplo de traza obtenida cambiando el algoritmo del servidor de Windows por el CTCP y configuración simplificada, 5.24.

Similarmente al caso anterior, el análisis de distribuciones de β no revelaba anomalías para su estimación dependiendo del evento.

5.4.1.2. Configuración por defecto (SACK y WScale)

Las únicas diferencias con la configuración anterior es que se activan las opciones de SACK y WScale. Estas opciones deberían de aumentar el rendimiento de las conexiones, aunque como se puede ver en las distribuciones anteriores, figura 5.8, para algunos algoritmos no se observa tanta mejora como era de esperar.

En la tabla 5.7 se muestra de las 100 conexiones de cada algoritmo, el número de eventos de *Timeouts* válidos para analizar. En la tabla se observa una disminución considerable del número de eventos con respecto a la configuración anterior. La utilización de estos parámetros está activada por defecto con las nuevas versiones de los sistemas operativos por lo que en principio esta configuración se asemeja más a un escenerio real.

Tabla 5.7: Eventos de *Timeouts* que se tienen para cada algoritmo, escenario de entrenamiento, configuración por defecto.

Algoritmo	Num Eventos
Reno	488
Cubic	172
HTCP	81
HighSpeed	505
Illinois	47
Windows	528
CTCP	571

A continuación se muestran las distribuciones de β obtenidas para esta configuración del escenario, figura 5.25. Activando estos parámetros, SACK y Wscale, se permite más variabilidad de sucesos en las conexiones y por tanto, para aquellos algoritmos que β es variable se obtiene un mayor rango de valores de β. Sobre todo, con respecto a la configuración anterior, se observa mayor variabilidad para los algoritmos HTCP y Highspeed.

Análogamente al estudio realizado en el apartado anterior, a continuación se analiza los resultados obtenidos para cada algoritmo.

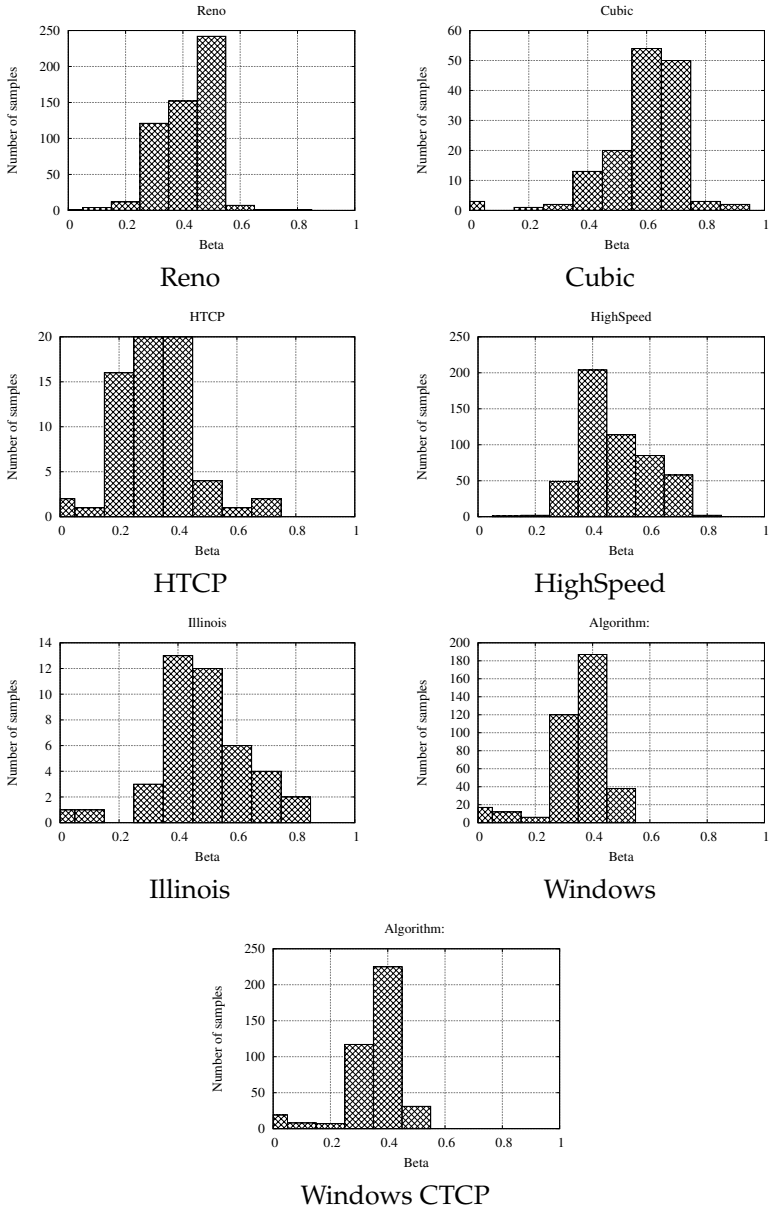


Figura 5.25: Histogramas de β obtenidas para cada algoritmo, escenario de entrenamiento, configuración por defecto.

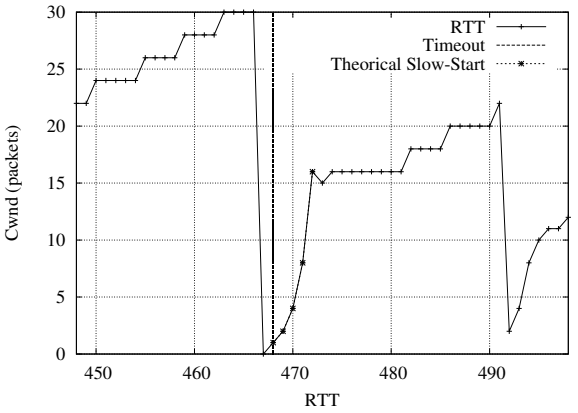


Figura 5.26: Ejemplos de traza del algoritmo Reno para la que se calcula una $\beta = 0{,}53$, escenario de entrenamiento, configuración por defecto.

Reno El histograma de las β obtenidas para el algoritmo de reno y la configuración normal, muestran una tendencia en $0{,}5$, figura 5.25. En el histograma se aprecian algunos errores, debido a que en algunos casos se estima que se termina antes de tiempo la fase *Slow Start* o bien porque se cuenta algún paquete más a la hora de hacer el cálculo.

En la figura 5.26 se muestra un ejemplo de traza observada en este escenario. Para esta traza se aprecia claramente el decrecimiento de la ventana a la mitad de su valor, tras la recuperación por *Slow Start* que seguía al evento de *Timeout*. Además el crecimiento lineal posterior también se puede intuir, aunque debido a que se produce un nuevo evento de *Timeout* no se aprecia un gran crecimiento.

Utilizando este algoritmo y esta configuración, se siguen obteniendo un gran número de eventos de *Timeouts* por conexión, se llegan a tener hasta 27 para alguna conexión. Las estimaciones que parecen tener más error son las β calculadas para el primer evento de la conexión, figura 5.27. No obstante estos errores no parecen suficientemente graves cómo para no tener en cuenta los valores de β calculados para el primer evento observado.

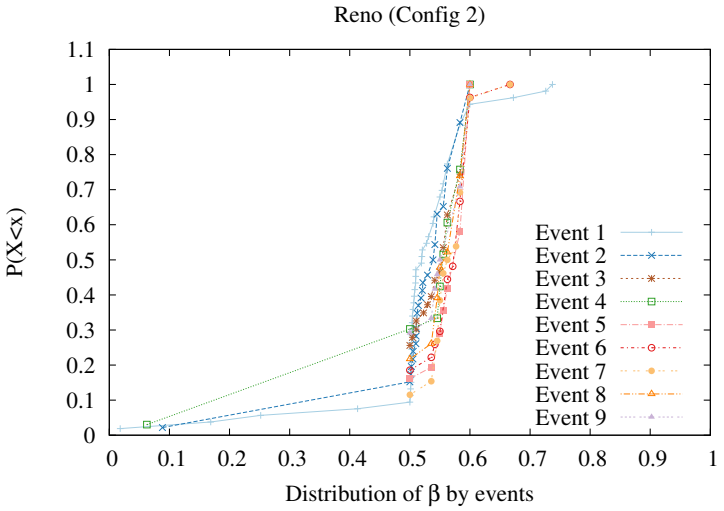


Figura 5.27: Distribución de β por eventos para el algoritmo de Reno, escenario de entrenamiento, configuración por defecto.

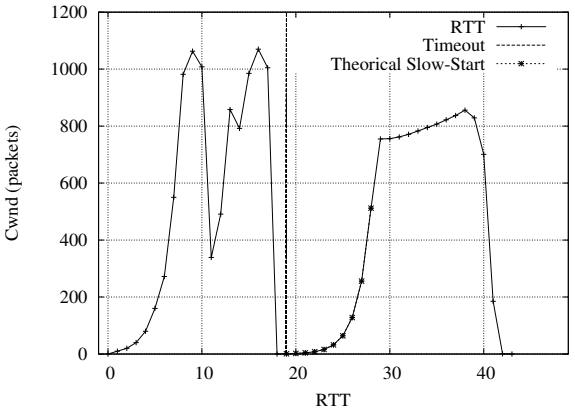


Figura 5.28: Ejemplos de traza del algoritmo Cubic para la que se calcula una $\beta = 0{,}7$, escenario de entrenamiento, configuración por defecto.

Cubic El histograma de las β obtenidas para el algoritmo de Cubic y la configuración por defecto, muestran una clara tendencia en $0{,}7$, figura 5.25, aunque hay una mayor variabilidad con respecto a la configuración simplificada. Estas desviaciones son normales puesto que al aumentar la variabilidad de sucesos también se aumentan los errores de cálculo por contabilizar de más algún paquete extra.

En la figura 5.26 se muestra un ejemplo de traza observada en este escenario. Aunque el tipo de crecimiento todavía no se está teniendo en cuenta, es interesante observar como después del *Slow Start* se tiene una función de crecimiento cóncava muy diferenciable del algoritmo AIMD.

Una diferencia notable, además del valor obtenido para β, con respecto al algoritmo de Reno con esta misma configuración es el número de eventos de *Timeout* por conexión, aunque sigue habiendo muchos en algunas ocasiones no son tantos como en el caso anterior. Como para los casos anteriores, se estudia la distribución de los valores estimados de β dependiendo del número de evento ocurrido en la traza, sin observarse anomalías.

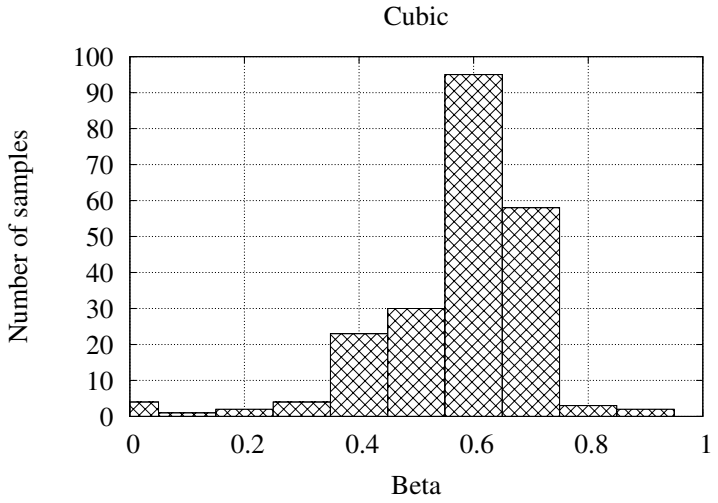


Figura 5.29: Histograma de cubic para 200 trazas (147 eventos de *Timeouts*). Escenario de entrenamiento, configuración por defecto.

Para observar si el histograma se mantiene para más conexiones, se realizan nuevamente 100 conexiones y se grafican los resultados de las β obtenidas conjuntamente con las anteriores, figura 5.29. Se observa como para este algoritmo los valores de β obtenidos se centran en el valor esperado.

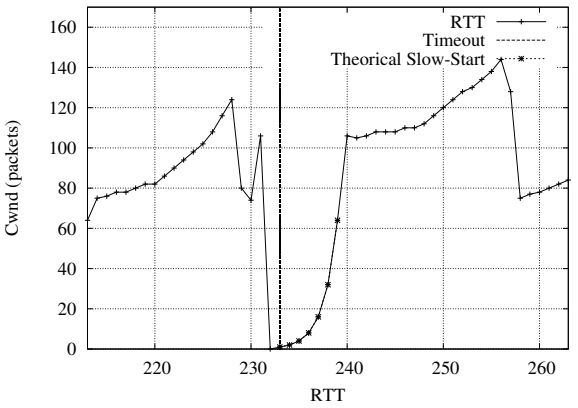


Figura 5.30: Ejemplos de traza del algoritmo HTCP para la que se calcula una $\beta = 0{,}85$, escenario de entrenamiento, configuración por defecto.

HTCP En este caso el algoritmo parece centrarse en valores de β más bien pequeños, en torno al $0{,}5$, figura 5.25. Estos valores tienen lógica puesto que la variabilidad de la β en este caso viene dada por el RTT. Al tratarse de un escenario con RTT grande parece lógico que el algoritmo trate de ser más conservador. No obstante, la variabilidad observada, pese a no ser muy grande será suficiente para distinguir este algoritmo del Reno.

Análogamente a otros casos se muestra cómo ejemplo una traza obtenida en el escenario con esta configuración, figura 5.26.

Para este caso se observan muchos menos eventos de *Timeouts* por conexión si se compara con los anteriores. Del estudio de la distribución de los valores de β obtenidos por eventos, se desprende conclusiones análogas a las anteriores, independencia del valor estimado de β con el número de evento observado en la conexión.

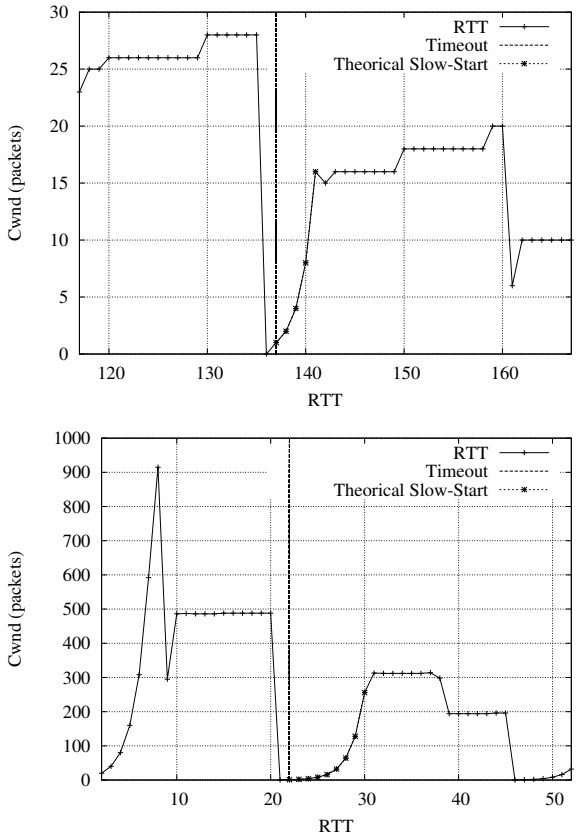


Figura 5.31: Ejemplos de traza del algoritmo HighSpeed para la que se calcula una $\beta = 0{,}5$ y $\beta = 0{,}64$ respectivamente, escenario de entrenamiento, configuración por defecto.

HighSpeed El algoritmo de HighSpeed parece tener valores más centrados en torno a $0{,}5$, lo que explicaría que se comportase parecido a Reno y por ello las conexiones tuvieran duraciones parecidas a las obtenidas con éste. Algunos ejemplos de trazas se muestran en la figura 5.31. Pese a que los valores de β parecen estar centrados en $0{,}5$, todavía se observan suficientes eventos para tener una distribución diferente a la observada con el algoritmo Reno, e incluso con el anterior analizado, HTCP.

Nuevamente, la distribución de los valores de β obtenidos en función del número de evento producido en la conexión, no refleja ninguna relación con la ocurrencia durante la conexión.

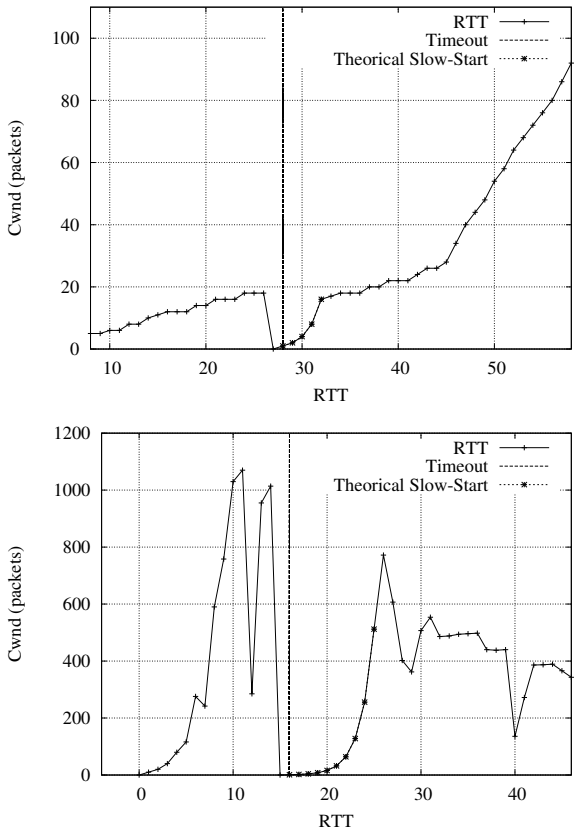


Figura 5.32: Ejemplos de traza del algoritmo Illinois para la que se calcula una $\beta = 0{,}89$ y $\beta = 0{,}5$ respectivamente, escenario de entrenamiento, configuración por defecto.

Illinois Este algoritmo puede alcanzar valores de β altos, hasta $0{,}9$. Sin embargo, en el escenario debido al alto RTT se muestra más conservador. Similarmente a los casos anteriores este algoritmo se distinguirá principalmente del Reno por la variabilidad en sus valores, en un escenario tan estático como el emulado ya se observa una tendencia de β a la dispersión de valores entre los posibles.

Algunos ejemplos de trazas obtenidas con el algoritmo de Illinois se muestran en la figura 5.32.

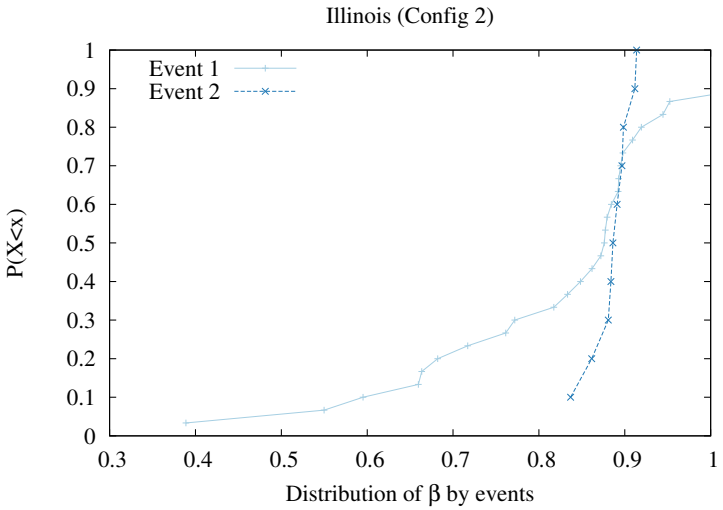


Figura 5.33: Distribución de β por eventos para el algoritmo Illinois, escenario de entrenamiento, configuración por defecto.

Siguiendo el proceso de los algoritmos anteriores, se grafica la distribución de las β por eventos, figura 5.33. Se puede observar como los primeros eventos que ocurren en las conexiones se estima una β más pequeña, mientras que para los segundos $\beta \sim 0{,}8$.

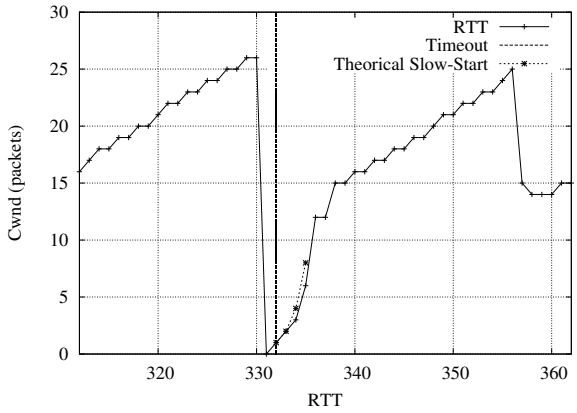


Figura 5.34: Ejemplos de traza del algoritmo Windows para la que se calcula una $\beta = 0{,}5$, escenario de entrenamiento, configuración por defecto.

Windows En el caso del servidor de Windows se esperaba obtener un histograma similar al obtenido para el algoritmo de Reno del servidor Linux. En este caso ambos servidores utilizan en realidad el mismo algoritmo de control de congestión. Las diferencias observadas entre uno y otro corresponden a diferencias en el valor de ventana utilizada por el servidor.

En la figura 5.34 se muestra un ejemplo de traza obtenida con el algoritmo de Windows.

Para este algoritmo se tiene en algunas conexiones un gran número de eventos de *Timeout*, lo que explica por qué algunas conexiones tardaban tanto en completarse. El estudio de la distribución de las estimaciones de β por eventos sigue reflejando independencia del valor estimado con respecto al número de evento observado en la conexión.

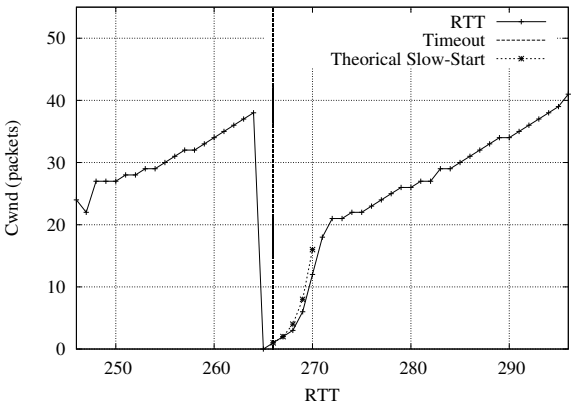


Figura 5.35: Ejemplos de traza del algoritmo Windows con CTCP para la que se calcula una $\beta = 0{,}5$, escenario de entrenamiento, configuración por defecto.

Windows CTCP Como ya se ha citado anteriormente, el algoritmo CTCP va a tener valores de β alrededor de $0{,}5$, aunque en la literatura se han propuesto ya variaciones para el cálculo de β en función del RTT, por defecto el algoritmo tiene la misma que NewReno. Lo que varía ligeramente es la función de crecimiento, por lo que es lógico la similitud entre los histogramas obtenidos de los dos algoritmos de Windows.

En la figura 5.35 se muestra un ejemplo de la β observada en una traza con el servidor Windows habiendo activado el algoritmo CTCP.

El estudio de la distribución de las estimaciones de β por eventos sigue reflejando independencia del valor estimado con respecto al número de evento observado en la conexión.

5.4.2. Comparación de todas las configuraciones

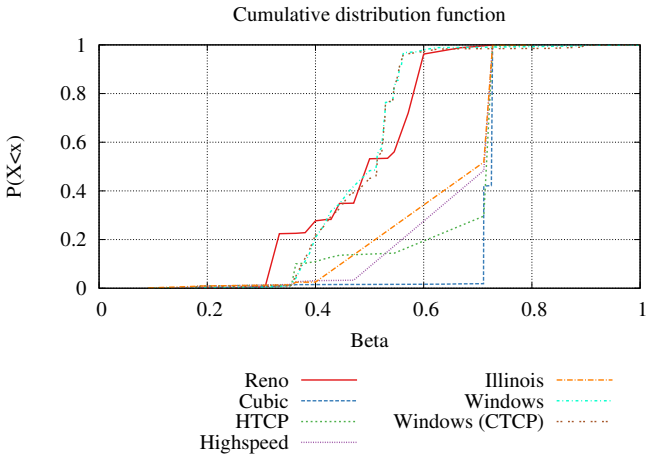
Hasta ahora se había analizado los valores de β de cada configuración de forma individual. El objetivo es distinguir los distintos algoritmos por lo que se compara las distribuciónes obtenidas para cada algoritmo independientemente del escenario, figura 5.36.

A partir de las distribuciones se observan diferencias entre las β obtenidas para cada algoritmo. Aún siendo un escenario dónde el RTT es estático se logra variabilidad de los valores de β para aquellos algoritmos que se podrían confundir con AIMD. Para un estudio más detallado se comparan las estadísticas de cada algoritmo para el caso del escenario por defecto, con el escalado de ventana y las confirmaciones selectivas activados, ya que este tipo de configuraciones serán las más probables en la realidad. Como para algunos algoritmos se tenían pocos eventos de *Timeouts* con tan sólo 100 conexiones, se capturan más trazas para esos casos y se calculan las estadísticas de todas las β obtenidas, 5.8.

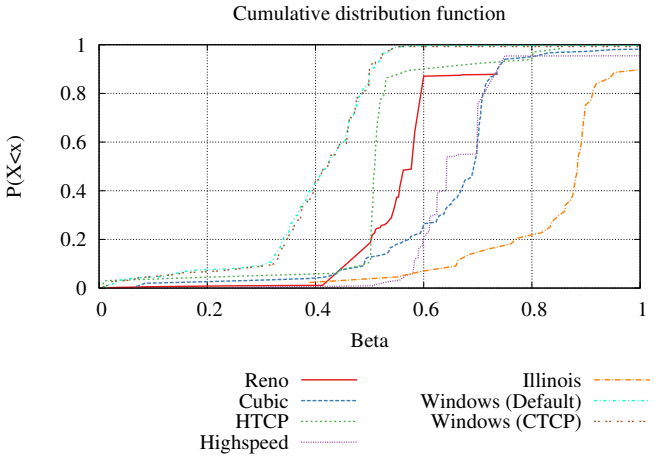
Tabla 5.8: Estadísticas de las trazas obtenidas del escenario de entrenamiento por algoritmo.

Algoritmo	Min	Máximo	Media	Var	p10	p90	Nº Timeouts
Reno	0.018	0.737	0.553	0.004	0.500	0.600	477
CTCP	0.007	0.556	0.403	0.012	0.324	0.500	410
Windows	0.007	0.600	0.401	0.013	0.312	0.500	381
Cubic	0.025	0.950	0.629	0.018	0.471	0.714	225
Highspeed	0.242	0.750	0.657	0.005	0.583	0.750	498
HTCP	0.007	0.900	0.544	0.025	0.432	0.799	218
Illinois	0.125	0.952	0.721	0.036	0.450	0.898	113

A partir de la tabla de las estadísticas, se puede describir un algoritmo simple de clasificación que permita distinguir entre los algoritmos estudiados a partir de recolectar unos pocos eventos de *Timeout* a los que le siga la fase de *Slow Start*. En el siguiente apartado se describe dicho algoritmo.



Configuración simplificada



Configuración por defecto

Figura 5.36: Función de distribución acumulada de β obtenidas para cada algoritmo en el escenario de entrenamiento.

5.4.3. Clasificador

En el estudio en el escenario de entrenamiento y la configuración por defecto, se ha observado que para distinguir algunos algoritmos basta con el valor de la media. Más concretamente, una media por encima o por debajo de $0{,}6$ sirve para aislar dos grupos de algoritmos; uno compuesto por $\{Reno, HTCP\}$ y otro con el resto de algoritmos, $\{Cubic, Highspeed, HTCP, Illinois\}$. Se debe destacar que en el primer grupo se clasificarán también los algoritmos utilizados por Windows.

Aunque en el caso de los algoritmos Highspeed, HTCP e Illinois se pueden tener valores de $\beta = 0{,}5$, debido a que también se pueden tener valores por encima en media la β se espera que esté por encima de este valor.

Para distinguir entre los algoritmos de cada grupo el parámetro que se puede tener en cuenta es el percentil 90, $p90$. El algoritmo HTCP puede tomar valores mayores que $0{,}5$ y Reno en principio no debería de superar ese valor. Para compensar posibles errores de la aproximación, se toma como umbral $0{,}65$.

En el caso del otro grupo, $\{Cubic, Highspeed, Illinois\}$, en principio el más fácil de diferenciar debería de ser Cubic, puesto que tiene un valor fijo de $\beta = 0{,}7$. Al igual que para el caso de Reno, se utiliza el percentil 90 con un umbral por encima del valor esperado por los posibles errores derivados de las aproximaciones, $0{,}75$. La diferencia principal entre Highspeed e Illinois es que el último parece alcanzar valores mayores que el primero, por lo que se utiliza el percentil 90 con un umbral de $0{,}8$ para diferenciarlos.

A continuación se muestra el algoritmo de clasificación explicado en pseudocódigo.

Data: $Mean_\beta$,$P90_\beta$
Result: Algorithm identified: Reno,Cubic,HTCP,HighSpeed,Illinois
if $Mean_\beta <= 0{,}6$ **then**
  if $P90_\beta < 0{,}65$ **then**
    return Reno;
  else
    return HTCP;
  end
else
  if $P90_\beta <= 0{,}8$ **then**
    if $P90_\beta < 0{,}75$ **then**
      return Cubic;
    else
      return HighSpeed;
    end
  else
    return Illinois;
  end
end

Algorithm 1: Clasificador para los algoritmos de control de congestión.

El clasificador procesará los paquetes de una o varias conexiones TCP en la que un emisor utiliza un algoritmo de control de congestión V, que es desconocido. A partir de las medidas que realice generará una salida M, que es el algoritmo de control de congestión indicado por el clasificador. A continuación se define la notación que se va a utilizar para el resto del capítulo.

***Definición* 5.1** *El conjunto de posibles algoritmos de congestión se define por Ω y sus elementos: Reno r, Cubic c, HTCP h, Highspeed s, Illinois i.*

$$\Omega = \{r, c, h, s, i\}$$

V y M toman valores en Ω.

***Definición* 5.2** *Se define como experimento, i, a la clasificación del conjunto de valores β obtenidos para una o varias conexiones TCP de un mismo emisor, que utiliza el algoritmo de control de congestión V y cuyo resultado de clasificarlo es M.*

Si el resultado del clasificador m, para un algoritmo observado es igual al valor correcto v, es una identificación correcta. En caso contrario es una identificación incorrecta.

Un conjunto de n clasificaciones permite estimar de forma experimental la probabilidad de clasificar correctamente cada algoritmo, a partir del número de clasificados correctamente y el número total de pruebas realizadas.

***Definición* 5.3** *Dado un conjunto de experimentos de clasificación, realizados con trazas de varios algoritmos de congestión, llamamos $N(m, v)$ al número de experimentos en los que un algoritmo v ha sido clasificado como m.*

***Definición* 5.4** *La probabilidad de que una traza etiquetada como v tenga como resultado clasificarse como m puede estimarse como: $P(M = m|V = v) = \frac{N(m,v)}{\sum_{i\in\Omega} N(i,v)}$*

Para evaluar las prestaciones del clasificador se definen una serie de métricas. La precisión del algoritmo se define como:

***Definición* 5.5** *La precisión del algoritmo,P, se define:*

$$P = \frac{1}{card(\Omega)} \sum_{i\in V} P(M = i|V = i)$$

Finalmente, la distribución de la variable V una vez que se tiene un resultado del clasificador sería la $P(V = v|M = m)$.

***Definición* 5.6** *La probabilidad de que una traza clasificada como m, provenga de un valor v se define por: $P(M = m|V = v) = \frac{N(m,v)}{\sum_{i\in\Omega} N(m,i)}$*

Una vez se ha definido el algoritmo de clasificación se ha probado primero sobre las propios eventos obtenidos en el escenario de entrenamiento, para probar después su efectividad con nuevas conexiones obtenidas en el mismo. Posteriormente se ha probado sobre un nuevo escenario, escenario de validación. En las siguientes secciones se analizan los resultados para las clasificaciones obtenidas.

5.4.3.1. Clasificación de trazas para el escenario de entrenamiento

En primer lugar se analizan los resultados de la clasificación para las conexiones del escenario de entrenamiento. Para cada conjunto de trazas de cada algoritmo se realizan experimentos en los que se cogen aleatoriamente diferentes números de estimaciones de β. Para cada experimento se calcula la media, $Mean_\beta$, y el percentil 90, $P90_\beta$, de las muestras de las estimaciones de β. Los valores obtenidos de la media y el percentil serán usados para clasificar cada entrada con un algoritmo. Estos experimentos son realizados varias veces dependiendo del número de estimaciones de β que se tuvieran en total.

De esta forma, por ejemplo, un experimento sería calcular la media y el percentil 90 de 10 estimaciones de β, escogidas aleatoriamente entre las trazas del servidor configurado con el algoritmo Cubic. Estos parámetros serían la entrada para el clasificador y como resultado se obtendría una etiqueta con el algoritmo clasificado para ese experimento. El mismo experimento se repitirá más veces cogiendo aleatoriamente otras estimaciones de β entre las trazas.

En la tabla 5.9, se muestra la probabilidad de clasificar correctamente cada algoritmo, $P(M = v|V = v)$, es decir, la probabilidad de clasificar el valor real de v con una muestra de salida del algoritmo que sea ese valor, $M = v$. Se puede observar que en el caso de coger tan sólo 10 muestras de β en el algoritmo Cubic, de los 10 experimentos realizados, tan sólo 4 han sido clasificados correctamente. Sin embargo, cogiendo 20 muestras, lo cuál sigue siendo un número pequeño, se logra clasificar el 100 % de todos los casos.

La tabla 5.10 muestra la precisión, P, alcanzada del clasificador dependiendo del número de estimaciones, β, empleados. Se observa que pese a no obtener buenos resultados de forma individual para algunos algoritmos analizando 10 eventos, la precisión total es bastante buena, del 78.3 %. Los altos porcentajes indican que los valores escogidos como umbrales en el clasificador, parecen bastante precisos.

En la figura 5.37 se muestran los porcentajes de clasificación correctamente, $True$, así como los erróneos, $False$.

Una información interesante para analizar son los falsos positivos, $False$. Si se analiza por cuál algoritmo es clasificado erróneamente se puede tener información de lo que pueda estar sucediendo, figura 5.38. En la figura se puede

Tabla 5.9: Resultados para 5 pruebas con diferentes eventos cada una.

Experimento	Algoritmo	$P(V = v \mid M = v)$
10 Experimentos, 10 Muestras de β	AIMD(Linux)	1
	AIMD (Windows)	1
	CUBIC	0.4
	HTCP	0.7
	Highspeed	0.6
	Illinois	1
5 Experimentos, 20 Muestras de β	AIMD(Linux)	1
	AIMD (Windows)	1
	CUBIC	1
	HTCP	1
	Highspeed	1
	Illinois	1

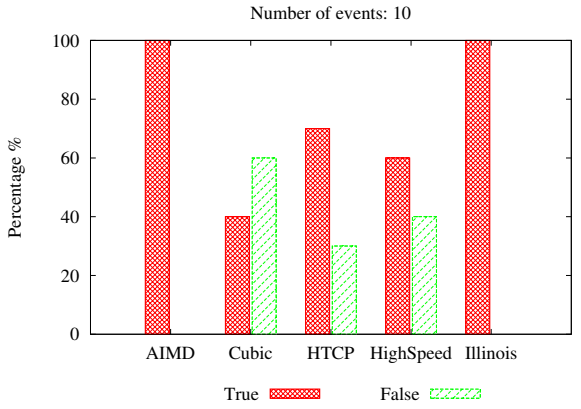
Tabla 5.10: Porcentaje de exactitud dependiendo del número de eventos observados.

Muestras de β	Nº Pruebas	Precisión
10	10	78.3 %
20	5	100 %

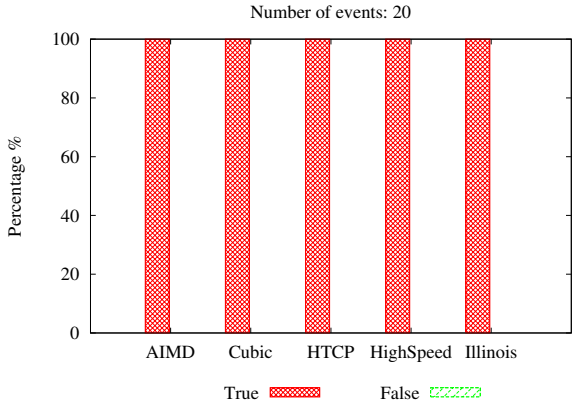
observar como en el caso de intentar clasificar utilizando sólo 10 estimaciones de β para el algoritmo de Cubic, algunas veces se tendrá de resultado que se trata del algoritmo de HTCP, Highspeed o incluso Illinois, pero ninguno como AIMD. Esto es debido a que al estimar la media con tan pocos valores se obtiene unos valores algo por encima de lo esperado. Análogamente ocurre para otros algoritmos.

En la figura 5.39 se muestra la probabilidad de clasificar para una muestra, v, como v u otro algoritmo, es decir lo que se ha definido como $P(M = m|V = v)$. En esta gráfica se aprecia nuevamente como los algoritmos de Cubic, Highspeed y HTCP se confunden entre ellos al intentar clasificarlos con pocas estimaciones de β.

Desde otro punto de vista se puede estudiar la probabilidad de habiendo obtenido una clasificación de m para una muestra, realmente esta fuera m u otro algoritmo, figura 5.40. Esta figura se diferencia de la anterior porque representa



10 Muestras de β



20 Muestras de β

Figura 5.37: Histograma obtenido cogiendo estimaciones de β aleatorios de las trazas en el escenario de entrenamiento.

el punto de vista del resultado. Por ejemplo, en la figura anterior 5.39, para el algoritmo de Reno se tenía una probabilidad de 1 utilizando 10 estimaciones, ya que todas las trazas etiquetadas como Reno eran clasificadas correctamente. Sin embargo, en esta figura 5.40, para el caso de 10 estimaciones, Reno tiene una probabilidad de $0{,}9$ ya que al clasificar el algoritmo de HTCP en un experimento se obtuvo el valor de Reno.

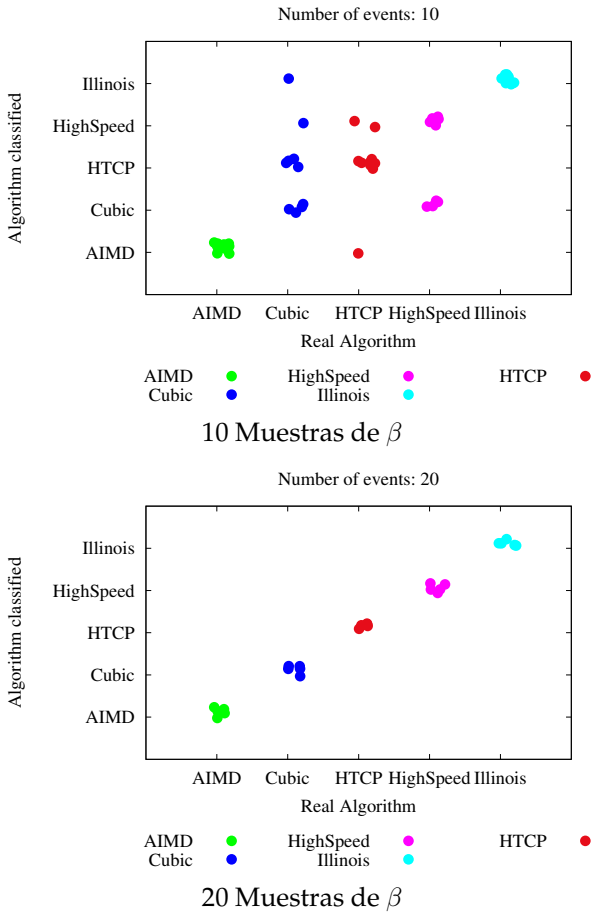


Figura 5.38: Clasificación obtenida cogiendo estimaciones de β aleatorios de las trazas en el escenario de entrenamiento.

Como era esperado, la clasificación obtenida de aplicar el algoritmo sobre eventos aleatorios del escenario, con los que se definieron los umbrales del algoritmo de clasificación, dan unos buenos resultados a partir de un número razonable de eventos con los que calcular la media y el percentil 90. A continuación se analizará si la situación se mantiene para nuevas trazas capturadas en el mismo escenario y la misma configuración.

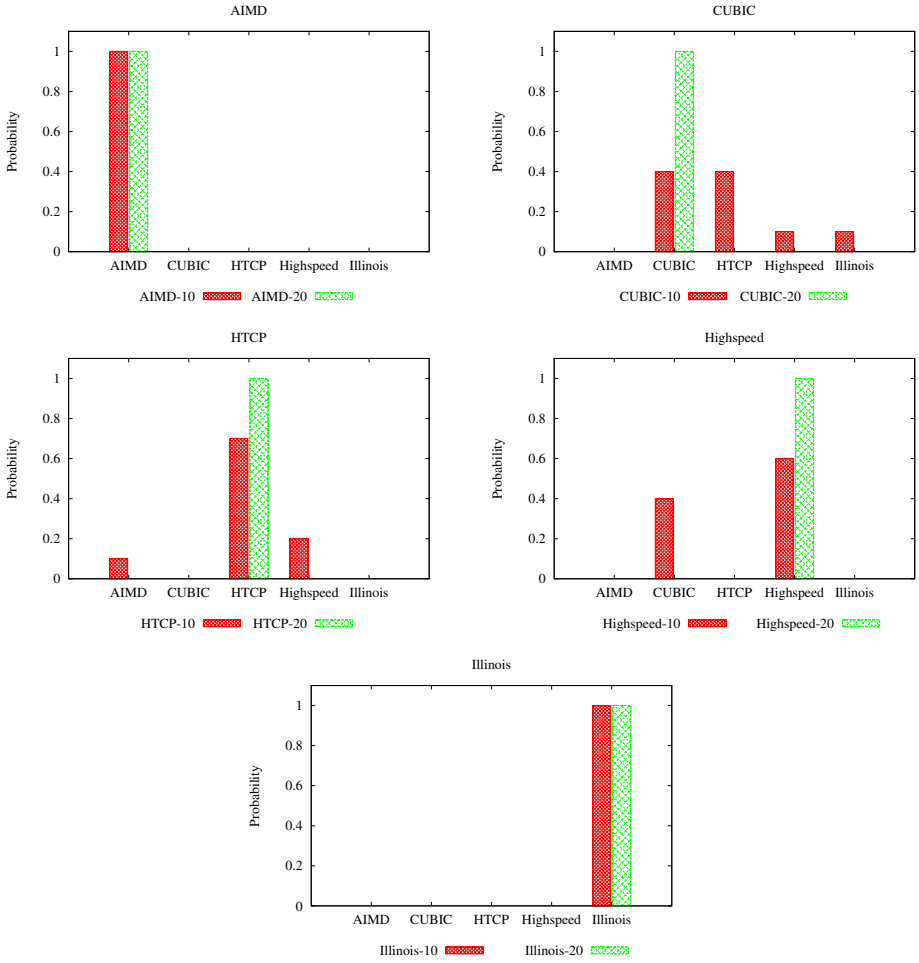


Figura 5.39: Probabilidad de clasificación, $P(M = m|V = v)$, para cada algoritmo. Escenario de entrenamiento.

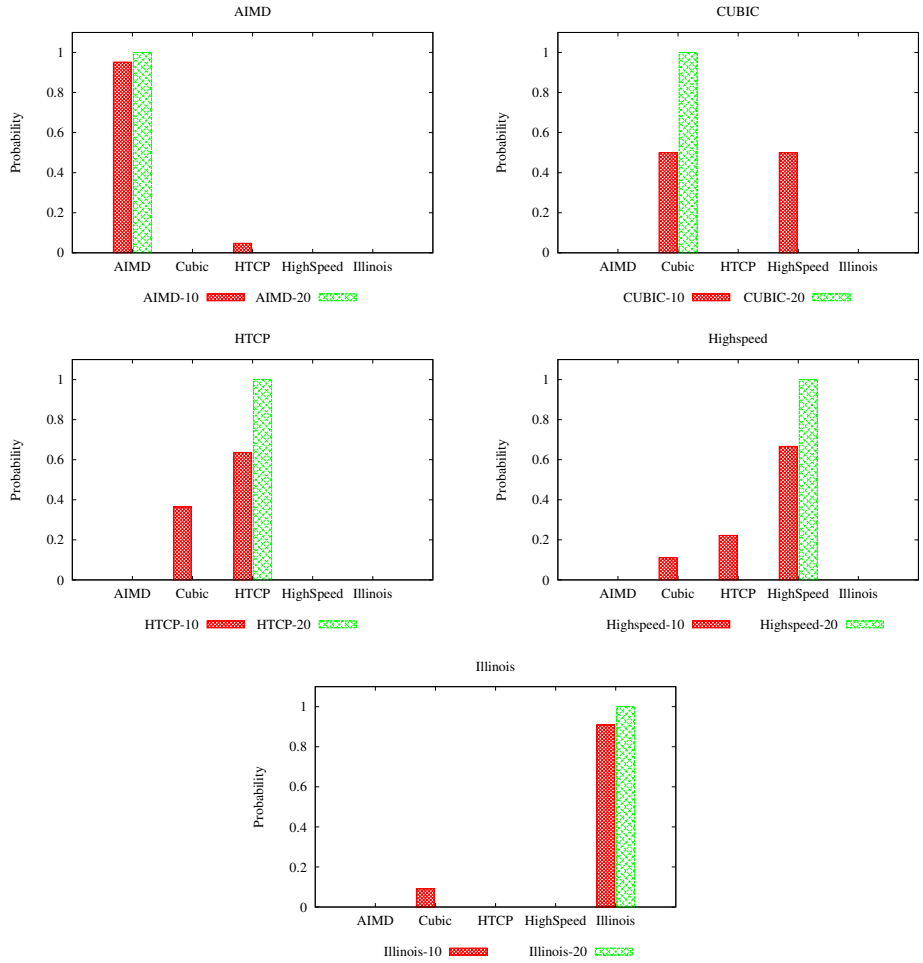


Figura 5.40: Probabilidad de si se obtiene un resultado m corresponda al algoritmo m, $P(V = v|M = m)$. Escenario de entrenamiento.

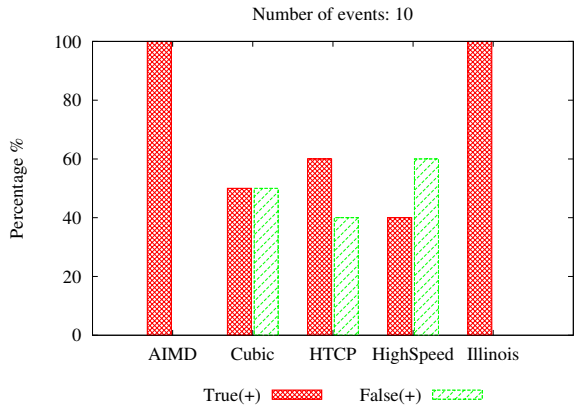
Nuevas trazas obtenidas en el escenario de entrenamiento Como se ha citado anteriormente, se va a comprobar la precisión del algoritmo de clasificación para nuevas conexiones capturadas en el mismo escenario. El objetivo es comprobar que los umbrales utilizados se mantienen estables al menos en las mismas condiciones. La única diferencia con respecto a la configuración anterior, es que las peticiones del cliente hacia el servidor se realizan más asiduamente, lo que provoca una mayor saturación en la red y en el servidor. Los experimentos se repiten análogamente al apartado anterior, pero procesando exclusivamente los eventos obtenidos en las nuevas trazas.

En la tabla 5.11 se muestra la probabilidad de clasificación correcta, $P(M = v|V = v)$, obtenida para cada algoritmo y conjunto de experimentos. En este caso, utilizando 20 estimaciones de β no se logra clasificar correctamente todos los algoritmos, cómo ocurría anteriormente, sin embargo, la probabilidad de clasificación individual sigue siendo bastante alta, en el caso peor del $0{,}8$.

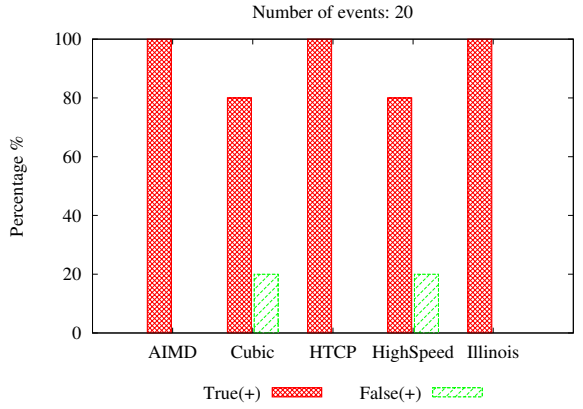
Tabla 5.11: Resultados para 5 pruebas con diferentes números de estimaciones cada una para nuevas trazas en el escenario de entrenamiento.

Experimento	Algoritmo	$P(M = v\|V = v)$
10 Experimentos, 10 Muestras de β	AIMD(Linux)	1
	CUBIC	0.5
	HTCP	0.6
	Highspeed	0.4
	Illinois	1
5 Experimentos, 20 Muestras de β	AIMD(Linux)	1
	CUBIC	0.8
	HTCP	1
	Highspeed	0.8
	Illinois	1

En la figura 5.41, se muestra el porcentaje de experimentos bien y mal clasificados para las estimaciones de β en las nuevas trazas. Los únicos algoritmos que no consiguen clasificarse correctamente para todos los experimentos, ni siquiera cogiendo 20 estimaciones de β, son Cubic y Highspeed. En el clasificador, Los umbrales para diferenciar uno de otro son bastante cercanos. Es normal que, a no ser que Highspeed se comporte siempre de forma muy estable con un valor



10 Muestras de β



20 Muestras de β

Figura 5.41: Histograma obtenido cogiendo cogiendo estimaciones de β aleatorias para nuevas trazas obtenidas en el escenario de entrenamiento.

por ejemplo de $0{,}5$, al coger pocos valores para promediar y realizar el percentil, se puedan confundir uno con otro. De hecho, en la figura 5.42, se puede observar como en el caso de 20 estimaciones, los falsos positivos de ambos algoritmos son confundidos por el contrario.

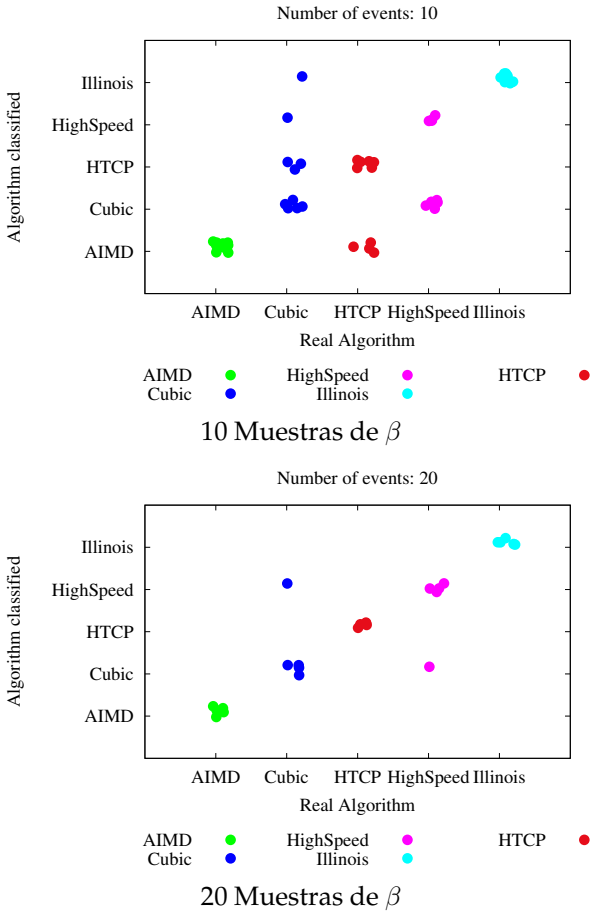


Figura 5.42: Clasificación obtenida cogiendo estimaciones de β aletarios para nuevas trazas en el escenario de entrenamiento.

La tabla 5.12 muestra los valores de precisión obtenidos con diferentes números de muestras para los experimentos realizados. El porcentaje de clasificación nuevamente en este caso es bastante alto, de un 92 % con tan sólo 20 estimaciones de β.

El porcentaje de clasificación para cada algoritmo dependiendo del número de estimaciones de β se muestra en la figura 5.43. Algunos algoritmos mejoran notablemente sus porcentajes al tener en cuenta más estimaciones de β, por

Tabla 5.12: Porcentaje de exactitud dependiendo del número de estimaciones de β observadas para nuevas trazas. Escenario de entrenamiento.

Eventos	Num Experimentos	Precisión
10	10	70 %
20	5	92 %

ejemplo HTCP consigue clasificarse correctamente para todos los experimentos con 20 estimaciones mientras que con 10 tenía una probabilidad del 60 %.

En la gráfica 5.44 se puede apreciar cómo en este caso, si se cogieran 20 estimaciones de β, la probabilidad de obtener un resultado de AIMD y en realidad ser otro algoritmo sería de 0. Como ya se ha citado, para este escenario los únicos algoritmos con alguna incertidumbre serían los algoritmos de Cubic y Highspeed, y para ello la probabilidad de error sería para ambos baja, en torno al 20 %.

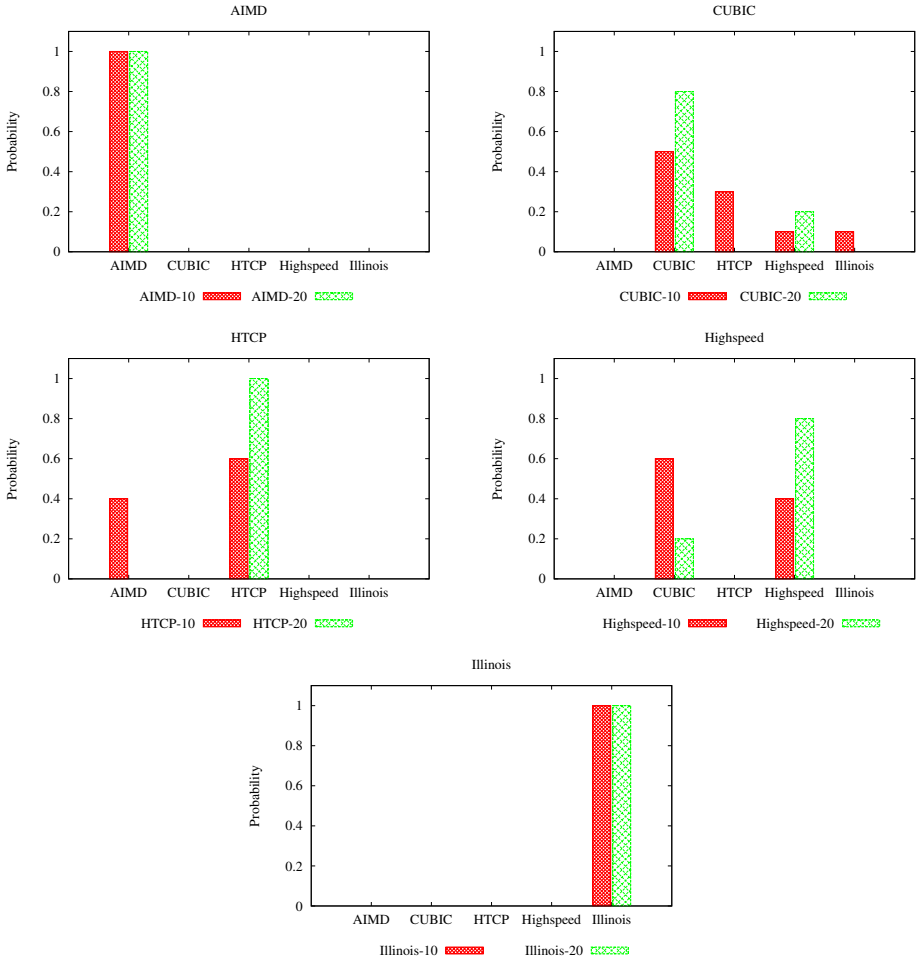


Figura 5.43: Probabilidad de clasificación, $P(M = m|V = v)$, para cada algoritmo. Escenario de entrenamiento, nuevas trazas.

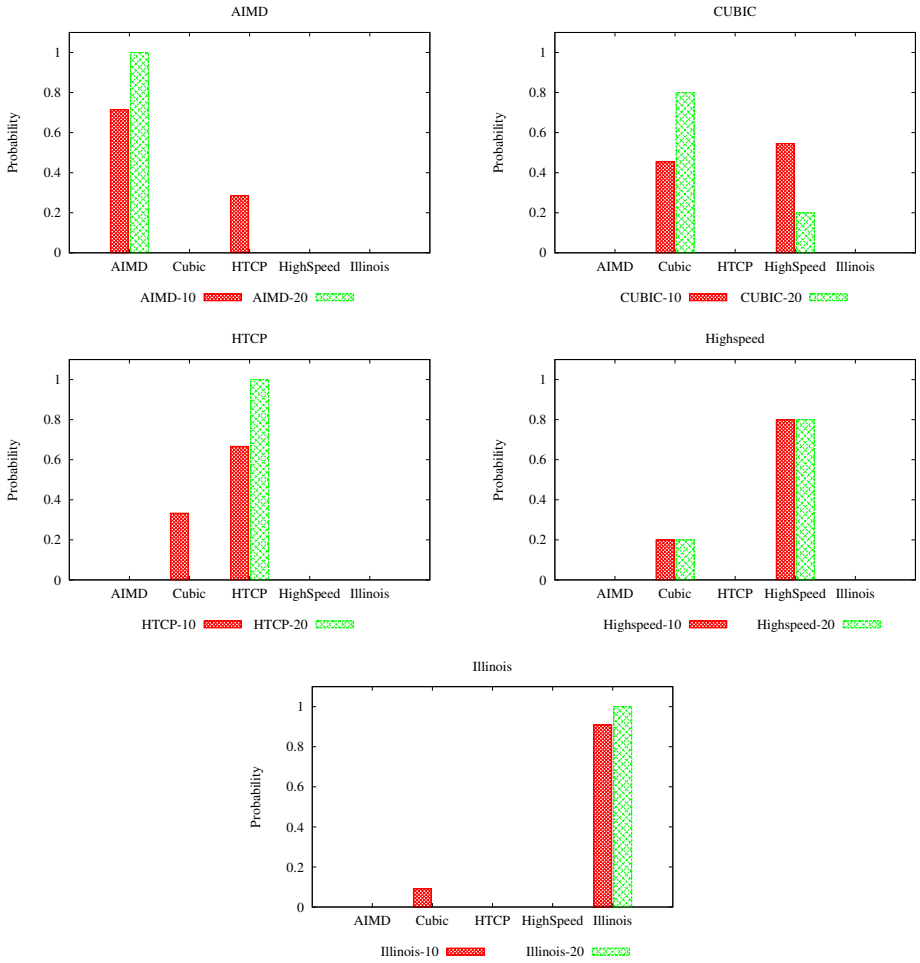


Figura 5.44: Probabilidad de si se obtiene un resultado m corresponda al algoritmo m, $P(V = v|M = m)$. Escenario de entrenamiento, nuevas trazas.

5.4.3.2. Clasificación de trazas para el escenario de validación

Este escenario ya había sido descrito anteriormente, tabla 5.5. El objetivo de realizar los experimentos sobre trazas obtenidas en un escenario con un RTT mucho más bajo y unas pérdidas más variables, es obtener unas condiciones más aproximadas a las posibles en un escenario real. Las pérdidas siguen siendo muy elevadas para compararse con un caso real, pero el único objetivo de éstas es tener un mayor número de eventos rápidamente sobre los que realizar las estadísticas. En un caso real, probablemente, se debería de monitorizar durante más tiempo para obtener un número razonable de estimaciones de β.

Sobre este escenario se realizan los mismos experimentos que en el apartado anterior. Sobre la totalidad de estimaciones de β capturadas para cada algoritmo se escogen un número de ellos de forma aleatoria, 10 ó 20, y se calcula la media y el percentil 90. Estos experimentos son reptidos varias veces dependiendo del número de eventos disponibles. En la tabla 5.45, se muestra el porcentaje de clasificación obtenido para la totalidad de experimentos con cada número de estimaciones de β empleadas.

Para el caso de 10 estimaciones de β, al contar con tan pocos valores las medias y percentiles no son estables por lo que la clasificación es lógico que no sea tan buena. En la tabla 5.13, se muestra la probabilidad de la correcta clasificación, $P(V = v|M = v)$, para cada algoritmo.

En 5.45, se apreciaba cómo no se consigue clasificar en ninguno de los casos el algoritmo Highspeed. La información de clasificación de 5.46, revela que en todos los casos este algoritmo es clasificado como HTCP. Este error es debido a que la media obtenida era más pequeña de lo esperado por lo que el primer filtro lo agrupa en la clasificación del grupo $\{Reno, HTCP\}$. En un escenario real se espera que este algoritmo tenga valores mayores con el fin de obtener un mejor rendimiento, por lo que no debería de repetirse un valor de media tan bajo.

A pesar de no lograrse una total clasificación al realizar la media y el percentil 90 utilizando 20 estimaciones de β, se tienen unos porcentajes altos, tabla 5.14.

Las gráficas 5.47 muestran rápidamente que los resulados de intentar clasificar en este caso los experimentos de Highspeed se tendrá una clasificación con probabilidades similares de AIMD o HTCP.

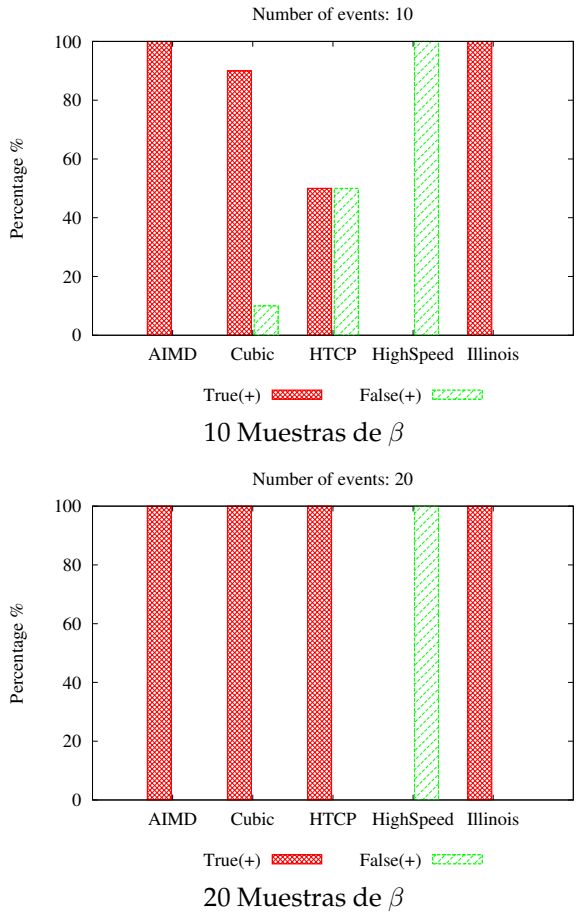


Figura 5.45: Histograma obtenido cogiendo eventos aletarios para nuevas trazas obtenidas en el escenario 2.

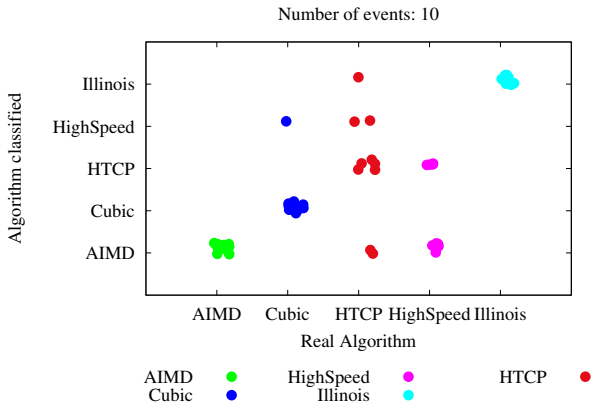
Además de que Highspeed no se va a lograr clasificar correctamente, además tomando sólo 10 estimaciones de β, en algunos casos se obtendrá como resultado este valor cuándo en realidad se trataba de HTCP (probabilidad de 30 %), o AIMD (probabilidad de 20 %). Sin embargo, al tener en cuenta más eventos nuevamente estos porcentajes disminuyen ligeramente, figura 5.48.

Tabla 5.13: Resultados para 5 pruebas con diferentes eventos cada una.

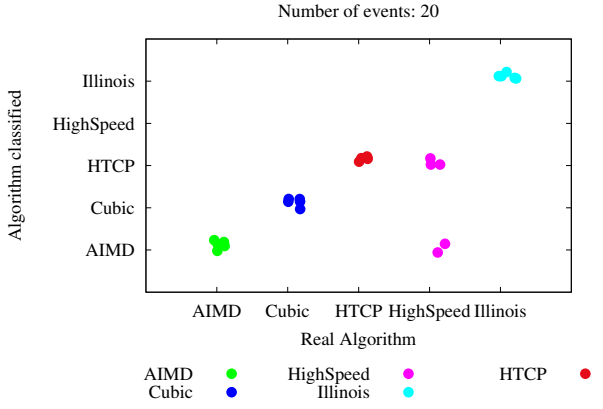
Experimento	Algoritmo	$P(M = v\|V = v)$
10 Experimentos, 10 Muestras de β	AIMD(Linux)	1
	AIMD (Windows)	1
	CUBIC	0.9
	HTCP	0.5
	Highspeed	0
	Illinois	1
5 Experimentos, 20 Muestras de β	AIMD(Linux)	1
	AIMD (Windows)	1
	CUBIC	1
	HTCP	1
	Highspeed	0
	Illinois	1

Tabla 5.14: Porcentaje de exactitud dependiendo del número de estimaciones de β observadas, escenario de validación.

Porcentaje de exactitud		
Eventos	Num Experimentos	Precisión
10	10	73.3 %
20	5	83.3 %



10 Muestras de β



20 Muestras de β

Figura 5.46: Clasificación obtenida cogiendo estimaciones de β aleatorias de las trazas del escenario de validación.

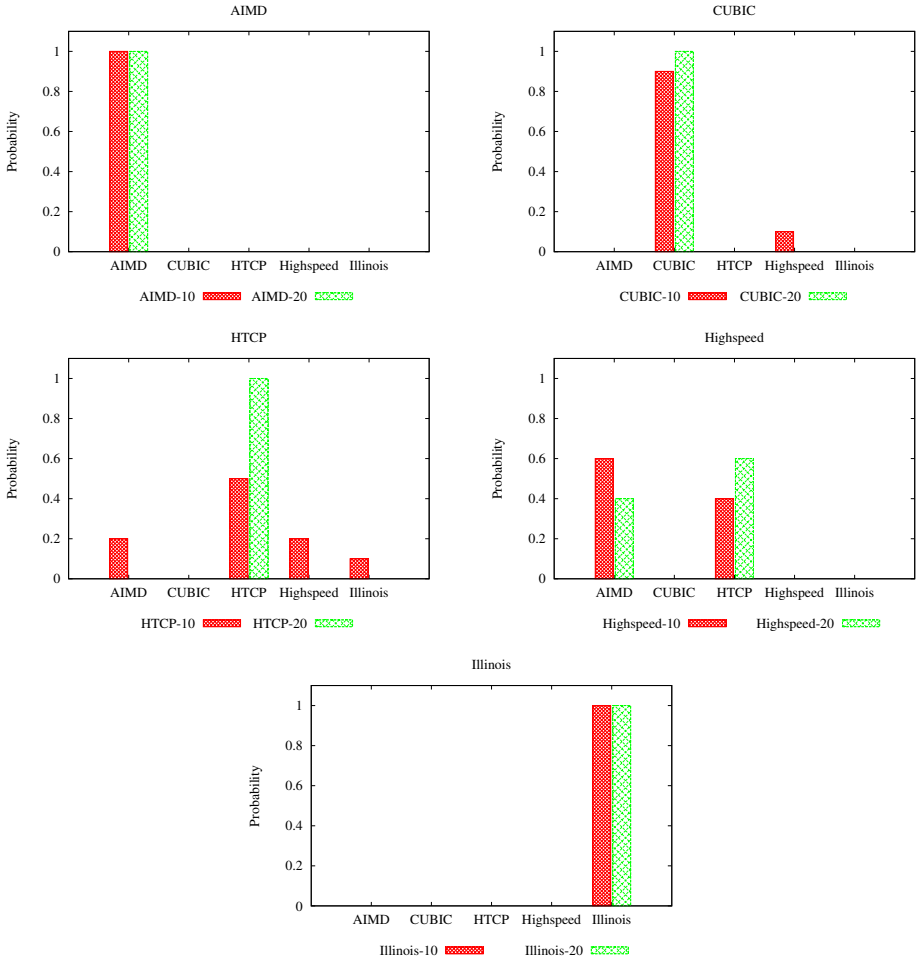


Figura 5.47: Probabilidad de clasificación, $P(M = m|V = v)$, para cada algoritmo. Escenario de validación.

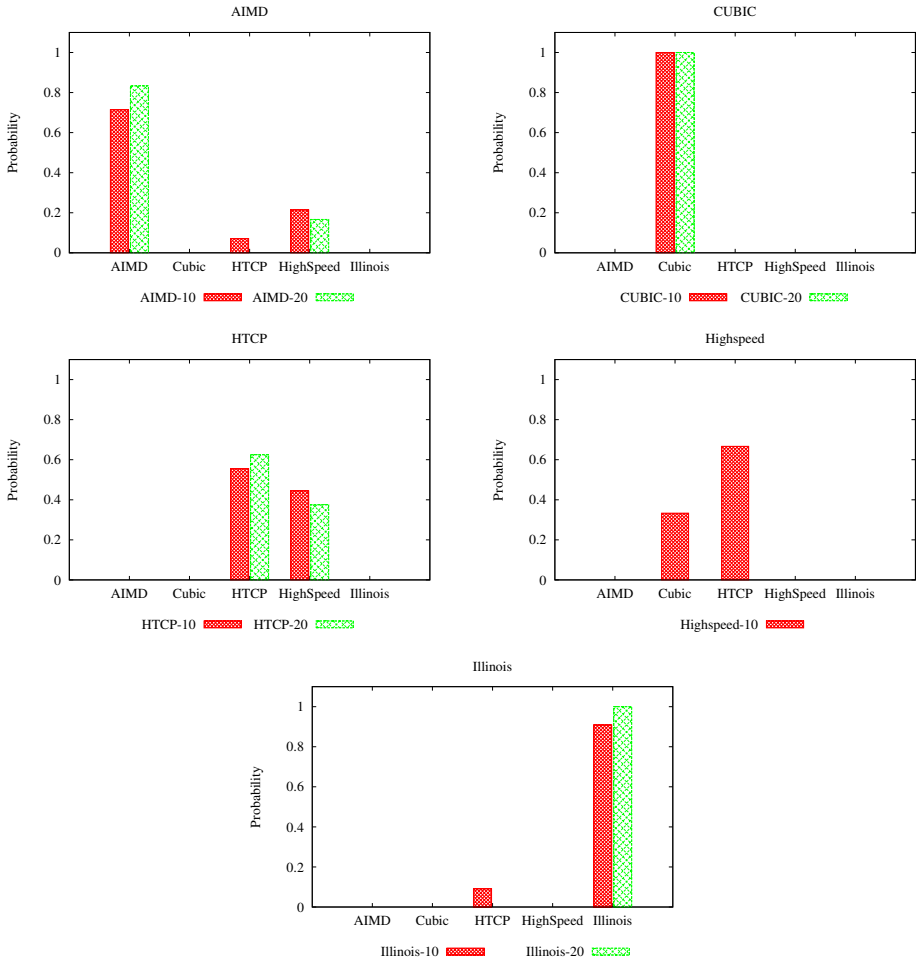


Figura 5.48: Probabilidad de si se obtiene un resultado m corresponda al algoritmo m, $P(V = v|M = m)$. Escenario de validación.

5.5. Conclusiones

Hasta ahora el enfoque del análisis pasivo de tráfico se había enfocado a parámetros directos que pueden afectar al rendimiento de un servicio, como son las disrupciones del servicio o bien el retardo de la red. Además de estos parámetros, el rendimiento de las conexiones también se ve afectado por el algoritmo de control de congestión empleado. El algoritmo de control de congestión define el comportamiento de TCP a la hora de enviar paquetes hacia el otro extremo intentando evitar congestionar la red.

La identificación del uso del algoritmo de control de congestión no es trivial, a no ser que se tenga un acceso directo al servidor, puesto que no se anuncia entre los parámetros del protocolo TCP. Mediante medidas activas se puede forzar al servidor a retransmitir por *Timeout* y por tanto se podría medir posteriormente como se ha decrementado la ventana. Sin embargo, siguiendo con las líneas de estudio de la tesis se prefiere un enfoque pasivo que permita la identificación del algoritmo utilizado.

El estudio de los diferentes algoritmos muestra como algunos de ellos se diferencian claramente por el valor de decremento, referido durante el capítulo como β, que tiene lugar después de un evento de retransmisión por *Timeout*. Estas retransmisiones se intentan evitar por los algoritmos de congestión, sin embargo, todavía siguen provocándose cada cierto tiempo, cómo se ha observado para unas trazas capturadas en un escenario real, la Universidad Pública de Navarra.

Mediante la utilización de un escenario emulado se ha realizado un estudio de la distribución de las β estimadas. A pesar de ser un escenario estático, donde el RTT no es variable y las retransmisiones son debidas a pérdidas del mismo tipo, se tienen unas distribuciones de β lo suficientemente variables para discernir un algoritmo de otro. A partir de estadísticas simples, como son la media y el percentil 90, se logran un porcentaje de clasificación para cada algoritmo alto, del 100 % tomando solamente 20 eventos para cada servidor.

El estudio presentado en este capítulo constituye un estudio inicial cuyo objetivo era comprobar si en un escenario real se podría identificar el algoritmo de control de congestión mediante el estudio del parámetro de decrecimiento. Este

primer análisis permite observar que aunque las retransmisiones por *Timeout* no son muy frecuentes, todavía se producen cada cierto tiempo, lo que permitiría estudiar este parámetro para todas las conexiones de clientes con un mismo servidor.

Como línea futura inmediata se pretende realizar todo el proceso de identificación presentado en el capítulo de forma *online*. Para ello, se pretende ir readaptando el RTT observado en una conexión para que, cuando se produzca una retransmisión se pueda calcular el parámetro β. De esta forma, este programa podría ser usado para ir monitorizando las conexiones con ciertos servidores e ir obteniendo valores de β. Cuando se obtuviera un número mínimo de estimaciones se realizaría la clasificación.

CAPÍTULO 6

Conclusiones

En la actualidad, las redes han adquirido un papel de omnipresencia en la sociedad. A lo largo del día, se realiza un uso constante de las mismas, tanto para ocio como profesionalmente. Hoy en día, cualquier empresa, por pequeña que sea, está disponible para sus clientes a través de páginas Web y redes sociales. Tal es la importancia de estar disponible las 24h ininterrumpidamente, que por seguridad, las grandes y medianas empresas, poseen servidores Web propios. Además de la Web, a nivel empresarial las redes son utilizadas para compartir recursos y servicios.

Directamente relacionado con la expansión de las redes, la comunidad científica se ha centrado en la búsqueda y perfeccionamiento de sistemas de monitorización para la detección de anomalías en las redes. Habitualmente, este tipo de sistemas han sido realizados con la colaboración de los sistemas a monitorizar, ya sea a través de peticiones de páginas HTTP, Pings, u otros tipos de sistemas más complejos. Este tipo de medidas se denominan activas y requieren de la inyección de nuevo tráfico en la red.

Un enfoque muy interesante, sobre todo para las redes grandes y con gran saturación en las que no se desea la inyección de nuevo tráfico, es el análisis pasivo del tráfico, tanto para la detección de problemas, como para la evaluación de métricas como la latencia de la red. A través del análisis pasivo del tráfico se pueden identificar problemas como configuraciones indebidas en las que se observa tráfico que no debería de seguir ese camino, saturaciones puntuales o

periódicas del tráfico, cierres inesperados de conexiones, etc.

Sin embargo, algunos problemas son más difíciles de abordar y requieren de estimaciones. A pesar de las ventajas, en la literatura no hay muchos estudios sobre el análisis pasivo de las redes de tráfico. La tesis se ha enfocado en estudiar en profundidad algunos de los problemas para el análisis de tráfico desde un punto de vista pasivo.

Las siguientes secciones son las conclusiones finales de la tesis, sección 6.1 y finalmente las líneas futuras 6.2.

6.1. Conclusiones

El objetivo principal de la tesis ha sido la evaluación y análisis de problemas derivados de la monitorización pasiva de las redes de tráfico de paquetes. Sin embargo, para algunos escenarios, la monitorización puede ser insuficiente para encontrar o entender el motivo de un comportamiento. Para estos casos, manteniendo el objetivo de no interferir en las redes a estudiar, se pueden usar simuladores. Para obtener conclusiones que puedan ser extrapolables al escenario real, las simulaciones se deberían de ajustar lo más fielmente posible a los escenarios reales. Esas simulaciones requerirán de un tráfico de entrada cuyas características de carga media, varianza y rafagosidad se asemejen con las del tráfico del escenario real. Este ha sido el primer problema analizado durante la tesis. Se ha propuesto una adaptación del algoritmo del Perlin Noise para la generación de tráfico auto-similiar. Las conclusiones extraídas de esta propuesta son las siguientes:

- La adaptación propuesta se puede describir por los parámetros utilizados en un proceso FGN: *Hurst H*, Media y Varianza.
- La precisión obtenida por el generador es comparable con la obtenida usando un generador RMD con los mismos parámetros.
- La gran ventaja de este generador es que permite generar tráfico de forma continua para ser usado directamente sin la necesidad de pre-generar y almacenar previamente las trazas.

El siguiente problema encarado durante la realización de la tesis ha sido la detección de interrupciones. Este es uno de los principales recursos demandados hoy en día en una red. No siempre es posible realizar una monitorización activa a través de las herramientas habituales, ya que requieren de alguna respuesta por parte de los servidores, y ya sea por problemas con *Firewalls*, o simplemente por no saturar más un servicio, no es posible realizar estas peticiones. Motivados por la relevancia de la detección, se ha realizado un estudio para elaborar un algoritmo capaz de detectar intervalos de interrupciones estudiando solamente el tráfico capturado en una red. El análisis se ha centrado para los servicios que utilizan el protocolo TCP, ya que este protocolo es utilizado por la mayoría de servicios que requieren monitorización. Las conclusiones obtenidas de este estudio son las siguientes:

- Se han obtenido modelos de conocimiento del efecto a nivel de paquete de tráfico de las interrupciones de servicios.
- Se ha elaborado un algoritmo con el que detectar cuándo se observa intervalos de tiempo de indisponibilidad de un servicio. De este algoritmo se destacan los siguientes puntos:
  - El algoritmo es sencillo, se basa en el uso de contadores simples de tipos de paquetes y de dos funciones de pertenencia para decidir si un intervalo es sospechoso o no de sufrir una interrupción de servicio.
  - El único requisito del algoritmo es que tenga la capacidad suficiente para capturar a la tasa de la red.

Otra problemática que se ha tenido en cuenta para la evaluación pasiva es la estimación de la latencia de la red. La latencia, o dicho de otro modo el RTT, es un valor importante para el análisis de una red, puesto que afecta directamente a la calidad de servicio percibida por el usuario. Normalmente, el RTT se calcula a través de herramientas activas, puesto que su estimación a través del análisis pasivo no es una tarea trivial. Esta tarea se complica todavía más para escenarios de redes grandes, ya que requiere tener en cuenta numerosos factores que pueden afectar al tráfico recibido: desórdenes, retransmisiones, pérdidas durante la captura, etc.

Se ha propuesto un método para la estimación pasiva del RTT para redes de alta velocidad. Las conclusiones extraídas de este estudio son las siguientes:

- Se puede construir un algoritmo basado en la premisa que una conexión no va a enviar más de lo que la ventana anunciada por el cliente le permite, aunque intentará aproximarse al máximo.
- El estimador *Timestamp* es preciso, pero se ha comprobado que en un escenario real no puede ser utilizado en muchos casos porque no es anunciado por alguno de los extremos.
- El algoritmo propuesto puede usarse conjuntamente con el del *Timestamp* para los casos en los que el segundo no sea anunciado.
- Las estimaciones del RTT propuestas han sido especialmente definidas para redes en las que no se llene su producto *bandwidth x delay*. Este requerimiento se cumple especialmente para redes con ancho de banda alto, como pueden ser las redes de centros de datos, que además son las redes objetivo de la tesis.
- En un escenario emulado, el rendimiento no era tan preciso como para el caso del *Timestamp*, pero los resultados eran lo suficientemente correctos como para ser utilizados.
- En el caso de un escenario real, la red de la Universidad Pública de Navarra, aunque no constituía el escenario idóneo para el algoritmo, se observaba como para RTTs dentro de la normalidad se obtenían unos errores asumibles.

La última prestación que se analiza es la identificación del algoritmo de control de congestión utilizado por el protocolo TCP. El algoritmo de control de congestión se utiliza internamente por cada extremo para decidir cuánto enviar en cada turno de RTT, intentando que la red no se sature. Al algoritmo básico le han seguido otras versiones cuyo objetivo era adptarse a redes con determinadas características. El rendimiento de las conexiones de una red, además de poder verse afectadas por saturación de las mismas, está influida por el algoritmo de control de congestión que utilice el servidor.

Las diferentes versiones de los algoritmos de control de congestión se diferencian en las funciones de crecimiento y en cómo de rápido se recuperan tras una pérdida considerada como grave. Las funciones de crecimiento pueden depender de distintos factores como el RTT, pérdidas anteriores, etc. Por esta razón no son fácilmente identificables. Sin embargo, es más intuitivo el parámetro que define hasta cuándo se recupera con un crecimiento exponencial después de una pérdida grave.

Una pérdida grave se identifica por una retransmisión por *Timeout*. Cuando salta un temporizador de este tipo se abandona la fase actual y se vuelve a la inicial, *Slow Start*. En esa fase se intenta alcanzar rápidamente el máximo de paquetes que se pueden enviar en un turno sin provocar retransmisiones. Esta fase tiene un crecimiento exponencial y se termina cuando se alcanza un umbral que depende del factor de decrecimiento, β, y del máximo alcanzado anteriormente. El parámetro β varía para cada algoritmo. La idea era que si se tienen muchas conexiones de un servidor con variabilidad de clientes, el parámetro β podría ser lo bastante informativo para diferenciar un algoritmo de otro.

Pese a que el parámetro β puede ser muy descriptivo, la identificación del control de congestión no es trivial. Al ser algo interno no hay ningún parámetro de TCP que sea intercambiado durante la conexión. Por otra parte, las pérdidas graves, es decir las retransmisiones por *Timeout*, son las que se intentan evitar en TCP, por lo que normalmente, no se deberían de observar muchas ocurrencias en las conexiones. El objetivo del análisis realizado en la tesis ha sido inferir en el tráfico capturado el valor de β y su clasificación. Del estudio de los algoritmos de control de congestión se llegan a las siguientes conclusiones:

- Del estudio de la identificación de las retransimisiones por *Timeout* y de la estimación del parámetro β para un escenario real, se observa que monitorizando las conexiones por un periodo corto de tiempo, se tendrán los suficientes eventos para identificar el algoritmo de control de congestión utilizado.

- El estudio de la estimación del parámetro β en un escenario emulado ha permitido estudiar las estadísticas de las estimaciones de β para servidores que utilizaban diferentes algoritmos de control de congestión.

- A través de las estadísticas de β se ha descrito un clasificador, que se ha probado para un escenario de entrenamiento y un escenario de validación.
  - En el escenario de entrenamiento se han logrado unos porcentajes de clasificación del 100 %.
  - El algoritmo de clasificación ha sido probado para un nuevo escenario, con unas características más próximas a un escenario real, obteniéndose unos porcentajes altos de clasificación, en torno al 83 %, con tan sólo 20 estimaciones de β.

En un escenario real, dónde se tienen clientes muy diversos para un mismo servidor, si se realizan estimaciones de β cada cierto tiempo para clientes aleatorios en diferentes días, probablemente la identificación del algoritmo sea más sencilla que la realizada en los escenarios emulados. Las condiciones de la red serían más variables por lo que para los algoritmos con β variable se apreciarían diferentes valores que permitirían su distinción. El estudio presentado constituye una primera aproximación que permite identificar los algoritmos de congestión más populares de forma pasiva.

6.2. Líneas futuras

Los resultados presentados en esta tesis abren nuevas líneas futuras de investigación que extiende el análisis pasivo de prestaciones a otros ámbitos de estudio y amplia algunos de los estudiados. A continuación se se presentan algunas de ellas.

Un trabajo inmediato es la optimización del algoritmo de detección de interrupciones propuesto en el capítulo 3. La implementación de este algoritmo requería conocer quiénes eran los servidores y clientes para los que se quería detectar la interrupción del servicio. En organizaciones grandes en ocasiones no es fácil obtener el listado de clientes que acceden a un determinado servicio. Aún cuando se disponen de listados, la aparición rápida de clientes o el movimiento de estos, provoca que los listados se queden obsoletos. Adaptando el algoritmo a un sistema capaz de diferenciar los servicios y clientes por sí solos, se podría

tener un sistema que se actualizara de forma independiente y no necesitara de listados.

Además, el método propuesto se basa en aplicar dos filtros en dos pasos; se podría aplicar algun método de inteligencia artificial para decidir con grados de incertidumbre si un intervalo puede ser sospechoso o no. Este criterio podría ayudar a disminuir los falsos positivos.

Respecto al estudio presentado en el capítulo 5, debido a la complejidad del problema, el algoritmo clasificador sólo tiene en cuenta uno de los parámetros que diferencia un algoritmo de control de congestión de otro, la β. El clasificador podría incluir parámetros descriptivos de la función de crecimiento que se observe tras la fase del *Slow Start*. La inclusión de estos parámetros permitiría una identificación para los algoritmos que pudiendo tener diferentes valores de β, sólo fuera observada uno.

Un proyecto próximo podría ser la identificación del algoritmo de congestión al vuelo. Debido a que la identificación requería de un análisis exhaustivo de cada una de las fases, la implementación del clasificador se ha realizado de manera modular cuyo parámetro de entrada son trazas de conexiones entre cliente y servidor. Al ser modular se podía rediseñar cada fase para un mejor ajuste y realizar un clasificador lo más limpio posible de estimaciones innecesarias. El paso a una implementación al vuelo no es trivial, porque se requiere de algoritmos que vayan calculando y reajustando algunos parámetros como el del RTT en cada turno.

El objetivo sería ir calculando los parámetros β para todas las conexiones e ir realizando distribuciones para cada servidor a la vez que el tráfico está siendo capturado. Cuándo se obtuvieran un número mínimo de estimaciones de β para un servidor se llamaría al algoritmo clasificador para obtener un resultado. De esta forma, se permitiría una identificación eficiente que podría ser probada en diferentes escenarios reales.

Creemos que este tipo de sistemas pueden ser de gran utilidad en grandes redes, en las que la elección de un algoritmo u otro pueda suponer un aumento importante de la velocidad de las conexiones.

La última línea futura está relacionada directamente con las redes móviles. Con la evolución de estas redes, cada día son más empresas que ofrecen a los

clientes potenciales desde sus aplicaciones ofertas e información. Estas redes están alcanzando una gran relevancia y por su relativa novedad, todavía no hay muchos estudios en los que se analice de forma pasiva los diferentes problemas a los que se enfrentan las aplicaciones a nivel de red. Habría que empezar a enfocar los estudios del tráfico pasivo a este tipo de escenarios.

CAPÍTULO 7

Lista de publicaciones

En este apartado se explica brevemente los artículos escritos durante la realización de la tesis. Cabe destacar que la tesis estaba inicialmente ligada al proyecto nacional **INSTINCT, Interdomain Strategies IN optical Core neTworks**. Este proyecto estaba coordinado por los grupos de investigación de la Universidad Pública de Navarra y la Universidad de Valladolid. Debido a los intereses del grupo de investigación, los objetivos de la tesis y los trabajos realizados durante el primer año, fueron virando finalmente hacia los objetivos presentados en esta memoria. Por esta razón, se presentan algunos artículos realizados durante la tesis que no están relacionados directamente con las líneas principales de investigación.

En las siguiente secciones, 7.1 y 7.2, se presentan respectivamente, los artículos relacionados directamente con las líneas de investigación y otros obtenidos como resultados de líneas secundarias.

7.1. Relacionadas con la tesis

En esta sección se describe cada uno de los artículos publicados relacionados con cada uno de los capítulos presentados en esta memoria.

- [PIM+12] Prieto, I.; Izal, M.; Morato, D. & Magaña, E. *Traffic generator using Perlin Noise Global*, Communications Conference (GLOBECOM), 2012 IEEE, 2012, 1847-1852. El artículo fue escrito como consecuencia del

desarrollo del algoritmo de generación de tráfico auto-similar explicado durante el capítulo 2.

- [PIM+15c] Prieto, I.; Izal, M.; Magaña, E. & Morato, D. *Detecting Disruption Periods on TCP Servers with Passive Packet Traffic analysis*, SOFTENG 2015, The First International Conference on Advances and Trends in Software Engineering, Best Paper Award, 2015. Este artículo fue realizado con el estudio presentado en el capítulo 3. En el mismo se describía el algoritmo y sus resultados para un escenario real.

- [PIM+15a] Prieto, I.; Izal, M.; Magaña, E. & Morato, D. Iaria (Ed.) *A Passive Traffic Algorithm for Detecting Unavailable Periods in TCP Services*, NetSer15v8n34, International Journal On Advances in Networks and Services, 2015, 182 - 191. Con la presentación del artículo anterior se recibió una invitación para presentar una versión extendida del algoritmo del capítulo 3. En esta nueva versión se probaba en un escenario emulado interrupciones originadas por diferentes causas. Las interrupciones eran etiquetadas y posteriormente comparadas con los resultados del algoritmo.

- [PIM+15b] Prieto, I.; Izal, M.; Magaña, E. & Morato, D. *A Simple Method to Estimate RTT*, Internet 2015, Iaria, 2015. Este artículo fue escrito con el estudio del capítulo 4. En el mismo se describía el método explicado durante el capítulo y su análisis para un escenario emulado.

7.2. Realizadas durante la tesis

En esta sección se muestran algunos líneas secundarias de investigación durante la realización de la tesis que como resultados tuvieron algún artículo. Para cada uno de ellos se describe la motivación para realizar el estudio aunque no estuvieran fuera de las líneas principales de estudio.

- [DM14] De Miguel I., P. I. D. R. G. d. D. O. G.-J. S. M. E. F. A. I. M. M. N. M. D. L. R. *Strategies for the interconnection of heterogeneous optical networks*, Proceedings of the 16th International Conference on Transparent Optical Networks (ICTON 2014), paper We.B3.1, 2014. Este artículo surgió como

resultado del proyecto nacional **INSTINCT, Interdomain Strategies IN optical Core neTworks** en el que colaboraban la Universidad Pública de Navarra y la Universidad de Valladolid. Aunque finalmente este proyecto no ha tenido relación directa con el desarrollo de la tesis, fue realizado utilizando como entrada del simulador el generador de tráfico Perlin Noise, presentado en el artículo [PIM+12] y en el capítulo 2 de esta memoria.

- [PIM+15d] Prieto, I.; Izal, M.; Magaña, E. & Morato, D. *Midiendo retardos y pérdidas en las redes móviles de alta velocidad*. XXX Simposium Nacional de la Unión Científica Internacional de Radio (URSI), 2015. Aunque el análisis de las redes móviles no se enmarca dentro de la realización de la tesis, debido a la importancia que están adquieriendo hoy en día las redes móviles de alta velocidad, se estimó interesante realizar una primera aproximación de los retardos y pérdidas que se observan en este tipo de escenarios.

Bibliografía

[Aim] *Additive increase/multiplicative decrease, AIMD. RFC 2914*. URL: https://tools.ietf.org/html/rfc2914.

[Ber94] J. Beran. *Statistics for long-memory processes*. Monographs on statistics and applied probability, 61. Chapman & Hall, oct. de 1994. ISBN: 0412049015.

[BM04] S. Biyani y J. Martin. "A comparison of TCP-friendly congestion control protocols". En: *Computer Communications and Networks, 2004. ICCCN 2004. Proceedings. 13th International Conference on*. 2004, págs. 255-260. DOI: 10.1109/ICCCN.2004.1401641.

[BOP94] L. S. Brakmo, S. W. O'Malley y L. L. Peterson. "TCP Vegas: New Techniques for Congestion Detection and Avoidance". En: *SIGCOMM Comput. Commun. Rev.* 24.4 (oct. de 1994), págs. 24-35. ISSN: 0146-4833. DOI: 10.1145/190809.190317. URL: http://doi.acm.org/10.1145/190809.190317.

[BR03] Y. Bejerano y R. Rastogi. "Robust monitoring of link delays and faults in IP networks". En: *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*. Vol. 1. 2003, págs. 134-144. DOI: 10.1109/INFCOM.2003.1208666.

[BST+95] J. Beran, R. Sherman, M. Taqqu y W. Willinger. "Long-range dependence in variable-bit-rate video traffic". En: *Communications, IEEE Transactions on* 43.234 (1995), págs. 1566 -1579. ISSN: 0090-6778. DOI: 10.1109/26.380206.

[Cac] *CACTI, a complete network graphing solution.* 2004-2012. URL: \url{http://www.cacti.net/}[accessed:2014-12-29].

[CCK98] C. H. Choi, M. G. Choi y S. D. Kim. "CSMonitor: a visual client/server monitor for CORBA-based distributed applications". En: *Software Engineering Conference, 1998. Proceedings. 1998 Asia Pacific*. 1998, págs. 338-345. DOI: 10.1109/APSEC.1998.733738.

[CKC05] D. Chua, E. Kolaczyk y M. Crovella. "Efficient monitoring of end-to-end network properties". En: *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*. Vol. 3. 2005, págs. 1701-1711. DOI: 10.1109/INFCOM.2005.1498451.

[CMT98] K. Claffy, G. Miller y K. Thompson. "The nature of the beast: Recent traffic measurements from an Internet backbone". En: *International Networking Conference (INET) '98*. Geneva, Switzerland: The Internet Society, 1998, págs. 1-1.

[DC15] G. W. D. N. T. I. Y. O. Dirceu Cavendish Kazumi Kumazoe. "Congestion Avoidance TCP Improvements for Video Streaming". En: Iaria. 2015.

[DM14] P. I. D. R. G. d. D. O. G.-J. S. M. E. F. A. I. M. M. N. M. D. L. R. De Miguel I. "Strategies for the interconnection of heterogeneous optical networks". En: *Proceedings of the 16th International Conference on Transparent Optical Networks (ICTON 2014), paper We.B3.1*. 2014.

[FDM09] G. Fang, Z. Deng y H. Ma. "Network Traffic Monitoring Based on Mining Frequent Patterns". En: *Fuzzy Systems and Knowledge Discovery, 2009. FSKD '09. Sixth International Conference on*. Vol. 7. 2009, págs. 571-575. DOI: 10.1109/FSKD.2009.444.

[FM94] V. Frost y B. Melamed. "Traffic modeling for telecommunications networks". En: *Communications Magazine, IEEE* 32.3 (1994), págs. 70 -81. ISSN: 0163-6804. DOI: 10.1109/35.267444.

[GM04] L. A. Grieco y S. Mascolo. "Performance Evaluation and Comparison of Westwood+, New Reno, and Vegas TCP Congestion Control". En: *SIGCOMM Comput. Commun. Rev.* 34.2 (abr. de 2004), págs. 25-38. ISSN: 0146-4833. DOI: 10.1145/997150.997155. URL: http://doi.acm.org/10.1145/997150.997155.

[Gor96] J. Gordon. "Long range correlation in multiplexed pareto traffic". En: *Broadband Communications, 1996. Global Infrastructure for the Information Age. Proceedings of the International IFIP-IEEE Conference on*. 1996, págs. 28 -39. DOI: 10.1109/ICBC.1996.887780.

[HCT+14] R. Hofstede, P. Celeda, B. Trammell, I Drago, R. Sadre, A Sperotto y A Pras. "Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX". En: vol. PP. 99. 2014, págs. 1-1. DOI: 10.1109/COMST.2014.2321898.

[HDL+95] C. Huang, M. Devetsikiotis, I. Lambadaris y A. Kaye. "Fast simulation for self-similar traffic in ATM networks". En: *Communications, 1995. ICC '95 Seattle, 'Gateway to Globalization', 1995 IEEE International Conference on*. Vol. 1. 1995, 438 -444 vol.1. DOI: 10.1109/ICC.1995.525208.

[Hig] *HighSpeed TCP for Large Congestion Windows. RFC 3649*. URL: http://www.ietf.org/rfc/rfc3649.txt.

[HL96] D. Heyman y T. Lakshman. "Source models for VBR broadcast-video traffic". En: *Networking, IEEE/ACM Transactions on* 4.1 (1996), págs. 40 -48. ISSN: 1063-6692. DOI: 10.1109/90.503760.

[HRX08] S. Ha, I. Rhee y L. Xu. "CUBIC: A New TCP-friendly High-speed TCP Variant". En: *SIGOPS Oper. Syst. Rev.* 42.5 (jul. de 2008), págs. 64-74. ISSN: 0163-5980. DOI: 10.1145/1400097.1400105. URL: http://doi.acm.org/10.1145/1400097.1400105.

[Htc] *HTCP: TCP Congestion Control for High Bandwidth-Delay Product Paths*. URL: https://tools.ietf.org/html/draft-leith-tcp-htcp-03.

[IAM+06] M. Izal, J. Aracil, D. Morato y E. Magana. "Delay-throughput curves for timer-based OBS burstifiers with light load". En: *Lightwave Technology, Journal of* 24.1 (2006), págs. 277 -285. ISSN: 0733-8724. DOI: 10.1109/JLT.2005.860140.

[JD02] H. Jiang y C. Dovrolis. "Passive Estimation of TCP Round-trip Times". En: *SIGCOMM Comput. Commun. Rev.* 32.3 (jul. de 2002), págs. 75-88. ISSN: 0146-4833. DOI: 10.1145/571697.571725. URL: http://doi.acm.org/10.1145/571697.571725.

[JFW+09] E. Jammeh, M. Fleury, C. Wagner, H. Hagras y M. Ghanbari. "Interval Type-2 Fuzzy Logic Congestion Control for Video Streaming Across IP Networks". En: *Fuzzy Systems, IEEE Transactions on* 17.5 (2009), págs. 1123-1142. ISSN: 1063-6706. DOI: 10.1109/TFUZZ.2009.2023325.

[JID+04] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose y D. Towsley. "Inferring TCP connection characteristics through passive measurements". En: *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*. Vol. 3. 2004, 1582-1592 vol.3. DOI: 10.1109/INFCOM.2004.1354571.

[JLS97] P. Jelenkovic, A. Lazar y N. Semret. "The effect of multiple time scales and subexponentiality in MPEG video streams on queueing behavior". En: *Selected Areas in Communications, IEEE Journal on* 15.6 (1997), págs. 1052 -1071. ISSN: 0733-8716. DOI: 10.1109/49.611159.

[KSM94] S. M. Klivansky, C. Song y A. Mukherjee. *On Long-Range Dependence in NSFNET Traffic*. 1994.

[KY95] G. Klir y B. Yuan. *Fuzzy sets and fuzzy logic*. Vol. 4. Prentice Hall New Jersey, 1995.

[LBS06] S. Liu, T. Başar y R. Srikant. "TCP-Illinois: A Loss and Delay-based Congestion Control Algorithm for High-speed Networks". En: *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*. valuetools '06. Pisa, Italy: ACM, 2006. ISBN: 1-59593-504-5. DOI: 10.1145/1190095.1190166. URL: http://doi.acm.org/10.1145/1190095.1190166.

[LEW+95] W.-C. Lau, A. Erramilli, J. Wang y W. Willinger. "Self-similar traffic generation: the random midpoint displacement algorithm and its properties". En: *Communications, 1995. ICC '95 Seattle, 'Gateway to Globalization', 1995 IEEE International Conference on*. Vol. 1. 1995, 466 -472 vol.1. DOI: 10.1109/ICC.1995.525213.

[LHS+06] X. Liu, J. Heo, L. Sha y X. Zhu. "Adaptive Control of Multi-Tiered Web Applications Using Queueing Predictor". En: *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*. 2006, págs. 106-114. DOI: 10.1109/NOMS.2006.1687543.

[LL03] G. Lu y X. Li. "On the Correspondency Between TCP Acknowledgment Packet and Data Packet". En: *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*. IMC '03. Miami Beach, FL, USA: ACM, 2003, págs. 259-272. ISBN: 1-58113-773-7. DOI: 10.1145/948205.948239. URL: http://doi.acm.org/10.1145/948205.948239.

[LLH+08] D.-J. Lan, P. N. Liu, J. Hou, M. Ye y L. Liu. "Service-Enabled Automatic Framework for Testing and Tuning Multi-tier System". En: *e-Business Engineering, 2008. ICEBE '08. IEEE International Conference on*. 2008, págs. 79-86. DOI: 10.1109/ICEBE.2008.47.

[LTW+94] W. Leland, M. Taqqu, W. Willinger y D. Wilson. "On the self-similar nature of Ethernet traffic (extended version)". En: *Networking, IEEE/ACM Transactions on* 2.1 (1994), págs. 1 -15. ISSN: 1063-6692. DOI: 10.1109/90.282603.

[LY01] Y.-C. Lai y C.-L. Yao. "TCP congestion control algorithms and a performance comparison". En: *Computer Communications and Net-*

works, 2001. Proceedings. Tenth International Conference on. 2001, págs. 523-526. DOI: 10.1109/ICCCN.2001.956315.

[MCG+01] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi y R. Wang. "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links". En: *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*. MobiCom '01. Rome, Italy: ACM, 2001, págs. 287-297. ISBN: 1-58113-422-3. DOI: 10.1145/381677.381704. URL: http://doi.acm.org/10.1145/381677.381704.

[Mun] *MUNIN, networked resource monitoring tool*. 2003-2013. URL: \url{http://munin-monitoring.org/}[accessed:2015-01-15].

[Nag] *NAGIOS, a commercial-grade network flow data analysis solution*. 2009-2015. URL: \url{http://www.nagios.com/}[accessed:2015-01-30].

[Net] *Netem, Network Emulator*. URL: http://devresources.linux-foundation.org/shemminger/netem/LCA2005_paper.pdf.

[New] *NewReno*. URL: https://tools.ietf.org/html/rfc6582.

[Nma] *NMAP "Network Mapper"*. URL: https://nmap.org/.

[NR02] R. Narasimha y R. Rao. "Modeling variable bit rate video on wired and wireless networks using discrete-time self-similar systems". En: *Personal Wireless Communications, 2002 IEEE International Conference on*. 2002, págs. 290 -294. DOI: 10.1109/ICPWC.2002.1177295.

[Ost06] D. Ostry. "Synthesis of accurate fractional Gaussian noise by filtering". En: *Information Theory, IEEE Transactions on* 52.4 (2006), págs. 1609 -1623. ISSN: 0018-9448. DOI: 10.1109/TIT.2006.871036.

[Par97] Y. Park. "Systems monitoring using Petri nets". En: *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on*. Vol. 4. 1997, págs. 3245-3248. DOI: 10.1109/ICSMC.1997.633109.

[PHJ+02] E. Perrin, R. Harba, R. Jennane e I. Iribarren. "Fast and exact synthesis for 1-D fractional Brownian motion and fractional Gaussian noises". En: *Signal Processing Letters, IEEE* 9.11 (2002), págs. 382 -384. ISSN: 1070-9908. DOI: 10.1109/LSP.2002.805311.

[PIM+12] I. Prieto, M. Izal, D. Morato y E. Magaña. "Traffic generator using Perlin Noise". En: *Global Communications Conference (GLOBECOM), 2012 IEEE*. 2012, págs. 1847-1852. DOI: 10.1109/GLOCOM.2012.6503384.

[PIM+15a] I. Prieto, M. Izal, E. Magaña y D. Morato. "A Passive Traffic Algorithm for Detecting Unavailable Periods in TCP Services". En: *NetSer15v8n34, International Journal On Advances in Networks and Services*. Ed. por Iaria. 3&4. 2015, págs. 182 -191. URL: http://www.iariajournals.org/networks_and_services/netser_v8_n34_2015_paged.pdf.

[PIM+15b] I. Prieto, M. Izal, E. Magaña y D. Morato. "A Simple Method to Estimate RTT". En: *Internet 2015*. Iaria. Iaria, 2015.

[PIM+15c] I. Prieto, M. Izal, E. Magaña y D. Morato. "Detecting Disruption Periods on TCP Servers with Passive Packet Traffic analysis". En: Barcelona, España: SOFTENG 2015, The First International Conference on Advances y Trends in Software Engineering, Best Paper Award, 2015. URL: http://www.iaria.org/conferences2015/SOFTENG15.html.

[PIM+15d] I. Prieto, M. Izal, E. Magaña y D. Morato. "Midiendo retardos y pérdidas en las redes móviles de alta velocidad". En: *XXX Simposium Nacional de la Unión Científica Internacional de Radio (URSI)*. 2015.

[Rfc] *TCP Extensions for High Performance. RFC 1323.* URL: \url{https : //tools.ietf.org/html/rfc1323 } [accessed : 2015-05-29].

[SLK+11] D. Schatzmann, S. Leinen, J. Kögel y W. Mühlbauer. "FACT: Flow-Based Approach for Connectivity Tracking". English. En: *Passive and Active Measurement*. Ed. por N. Spring y G. Riley. Vol. 6579. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, págs. 214-223. ISBN: 978-3-642-19259-3. DOI: 10.1007/978-3-642-19260-9_22.

[Son12] G. Song. "The Study and Design of Network Traffic Monitoring Based on Socket". En: *Computational and Information Sciences (ICCIS), 2012 Fourth International Conference on*. 2012, págs. 845-848. DOI: 10.1109/ICCIS.2012.351.

[ST14] S. Sangolli y J. Thyagarajan. "An efficient congestion control scheme using cross-layered approach and comparison of TCP variants for mobile ad-hoc networks (MANETs)". En: *Networks Soft Computing (ICNSC), 2014 First International Conference on*. 2014, págs. 30-34. DOI: 10.1109/CNSC.2014.6906703.

[Str13] S. D. Strowes. "Passively measuring TCP round-trip times". En: *Communications of the ACM* 56.10 (2013), págs. 57-64.

[SW00] D. Sisalem y A. Wolisz. "MLDA: a TCP-friendly congestion control framework for heterogeneous multicast environments". En: *Quality of Service, 2000. IWQOS. 2000 Eighth International Workshop on*. 2000, págs. 65-74. DOI: 10.1109/IWQOS.2000.847939.

[SWW00] C. Steigner, J. Wilke e I. Wulff. "Integrated performance monitoring of client/server software". En: *Universal Multiservice Networks, 2000. ECUMN 2000. 1st European Conference on*. 2000, págs. 395-402. DOI: 10.1109/ECUMN.2000.880791.

[TAT11] A. Tachibana, S. Ano y M. Tsuru. "Selecting Measurement Paths for Efficient Network Monitoring and Diagnosis under Operational Constraints". En: *Intelligent Networking and Collaborative Systems (INCoS), 2011 Third International Conference on*. 2011, págs. 621-626. DOI: 10.1109/INCoS.2011.147.

[TDL98] T. Taralp, M. Devetsikiotis e I. Lambadaris. "Efficient fractional Gaussian noise generation using the spatial renewal process". En: *Communications, 1998. ICC 98. Conference Record.1998 IEEE International Conference on*. Vol. 3. 1998, 1456 -1460 vol.3. DOI: 10.1109/ICC.1998.683067.

[TSZ+05] K. Tan, J. Song, Q. Zhang y M. Sridharan. *A Compound TCP Approach for High-speed and Long Distance Networks*. Inf. téc. MSR-TR-2005-86. Microsoft Research, 2005, pág. 12. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=70189.

[V.J90] V.Jacobson. *Modified TCP Congestion Control and Avoidance Algorithms*. Inf. téc. 1990.

[VLL05] B. Veal, K. Li y D. Lowenthal. "New Methods for Passive Estimation of TCP Round-Trip Times". English. En: *Passive and Active Network Measurement*. Ed. por C. Dovrolis. Vol. 3431. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, págs. 121-134. ISBN: 978-3-540-25520-8. DOI: 10.1007/978-3-540-31966-5_10. URL: http://dx.doi.org/10.1007/978-3-540-31966-5_10.

[XHR04] L. Xu, K. Harfoush e I. Rhee. "Binary increase congestion control (BIC) for fast long-distance networks". En: *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*. Vol. 4. 2004, 2514-2524 vol.4. DOI: 10.1109/INFCOM.2004.1354672.

[YLX+11] P. Yang, W. Luo, L. Xu, J. Deogun e Y. Lu. "TCP Congestion Avoidance Algorithm Identification". En: *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*. ICDCS '11. Washington, DC, USA: IEEE Computer Society, 2011, págs. 310-321. ISBN: 978-0-7695-4364-2. DOI: 10.1109/ICDCS.2011.27. URL: http://dx.doi.org/10.1109/ICDCS.2011.27.

[YSL+14] P. Yang, J. Shao, W. Luo, L. Xu, J. Deogun e Y. Lu. "TCP Congestion Avoidance Algorithm Identification". En: *Networking, IEEE/ACM*

Transactions on 22.4 (2014), págs. 1311-1324. ISSN: 1063-6692. DOI: 10.1109/TNET.2013.2278271.

[Zab] *ZABBIX, the ultimate enterprise-level software designed for monitoring availability and performance of IT infrastructure components*. 2001-2014. URL: \url{http://www.zabbix.com}[accessed:2015-02-02].

[Zal] M. Zalewski. *p0f v3: passive fingerprinter*. URL: http://lcamtuf.coredump.cx/p0f3/.

[Zha86] L. Zhang. "Why TCP timers don't work well". En: *Proceedings of the ACM SIGCOMM conference on Communications architectures & protocols, 1986, Stowe, Vermont, United States August 5-7, 1986*. 1986, págs. 397-405. DOI: 10.1145/18172.18216. URL: http://doi.acm.org/10.1145/18172.18216.

Acrónimos

ACK	*Acknowledgement*.
AIMD	*Additive Increase/Multiplicative Decrease*.
CPD	*Centro de Procesamiento de Datos*.
FGN	*Fractional Gaussian Noise*.
HTTP	*Hypertext Transfer Protocol*.
ICMP	*Internet Control Message Protocol*.
IP	*Internet Protocol*.
LRD	*Long range dependence*.
NTP	*Network Time Protocol*.
OTT	*Over-The-Top*.

RMD	*Random Midpoint Displacement*.
RST	*Reset*.
RTT	*Round Trip Time*.
SACK	*Selective Acknowledgement*.
TCP	*Transport control Protocol*.
VBR	*Variable Bit Rate*.
WSCALE	*Window Scale*.
WWW	*World Wide Web*.