Proyecto MetroCel

Estimación de Performance en Enlaces GPRS-EDGE

Documentación de Proyecto de Grado
Ingeniería Eléctrica

Juan Andrés Negreira Marín
Javier Pereira Lucas
Santiago Pérez Inzaurralde

Tutor: Pablo Belzarena

Facultad de Ingeniería
Universidad de la República

Montevideo, Uruguay
Febrero 2007

Agradecimientos

A nuestras familias y amigos, que sin su apoyo incondicional no hubiésemos podido realizar la carrera.

A nuestro tutor, Pablo Belzarena, creador del proyecto, cuyo apoyo fue fundamental a lo largo del mismo.

A nuestros compañeros de generación, con los cuales compartimos muchos momentos dentro y fuera de la facultad y sin los cuales nuestra estadía por la misma no hubiese sido la misma.

A Federico Larroca y Pedro Casas, de los cuales tuvimos mucha colaboración a la hora de proporcionarnos información acerca del funcionamiento de MetroNet.

A Gabriel Gómez, cuyas gestiones a lo largo del proyecto han sido de muchísima ayuda.

Por último nos gustaría agradecer profundamente a la gente de ANTEL, que nos proporcionó el material necesario para poder llevar a cabo el proyecto satisfactoriamente.

Tabla de contenidos

Resumen

En este proyecto se presenta una herramienta para la estimación de performance en servicios de datos sobre enlaces celulares, más precisamente sobre los protocolos GPRS-EDGE[1]. Dicha herramienta surge como una extensión de un proyecto PDT realizado anteriormente en el IIE[2] denominado *MetroNet*, cuyo fin es el de medir la calidad de servicio en transmisiones online de audio y video.

En la primera etapa, como actividad introductoria al tema, se aborda el estudio de las distintas herramientas existentes en el área para la estimación de capacidades de cuello de botella y ancho de banda disponibles en redes de datos fijas y se implementan tres algoritmos seleccionados con el fin de ampliar las funcionalidades existentes en *MetroNet* hasta el momento.

En la segunda etapa se realiza un estudio de las características relevantes de los sistemas de comunicación de datos sobre tecnologías celulares, mas precisamente en tecnologías de acceso basadas en TDMA[3] (GSM[4] en particular). Con la base del conocimiento adquirido en la primera etapa se desarrolla e implementa un algoritmo propio, del cual se obtiene un análisis detallado del funcionamiento de la red GPRS-EDGE en la cuál funciona el sistema implementado. Para poder llevar a cabo las mediciones necesarias se implementa un sistema del tipo cliente - servidor el cual consta de un software corriendo en un servidor de Internet (ubicado en la LAN del IIE) y otro corriendo en un cliente. Este puede ser una computadora personal cuyo acceso a Internet sea a través de un módem GPRS-EDGE o bien un móvil celular que sea capaz de correr programas codificados en el lenguaje JAVA sobre la plataforma J2ME[5].

[1] GPRS: General Packet Radio Service EDGE: Enhanced Data Rates for GSM Evolution

[2] Instituto de Ingeniería Eléctrica

[3] Time Division Multiple Access

[4] Global System for Mobile Communications

[5] La plataforma J2ME es un lenguaje de programación que define una versión minimizada de la plataforma JAVA 2. Estas versión tiene el fin de ser utilizada para correr sobre dispositivos móviles cuya capacidad de procesamiento y memoria es sensiblemente inferior a la de una computadora personal de hoy en día, como por ejemplo móviles celulares y PDA'S

Como última actividad se desarrolla e implementa un sistema capaz de medir parámetros de performance en una conexión WAP[6]. Para esto se implementaron dos modalidades del software. La primera orientada a un usuario de aplicaciones WAP el cual desea tener un panorama del estado de su conexión. La segunda orientada a un operador de la red interesado en conocer el estado de la misma en un determinado momento.

Palabras Claves: GPRS-EDGE, GSM, WAP, performance.

[6]Wireless Access Protocol

Parte I

Actividades Introductorias

Capítulo 1

Introducción

1.1. Motivación

En las últimas décadas la evolución de las telecomunicaciones ha sido tal que hoy en día es prácticamente impensable que una persona, institución o empresa que pretenda estar en contacto con el resto del mundo no posea una conexión dedicada a Internet; el cual tiene un impacto profundo en el trabajo, el ocio y la adquisición de conocimientos. Gracias a él, millones de personas tienen acceso a una cantidad extensa y diversa de información y entretenimiento en línea.

Paralelamente con el crecimiento de Internet, de la mano del surgimiento de GSM, surgió un avance muy significativo en el área de la telefonía móvil, al punto tal que hoy en día el 30 % de la población global posee un móvil GSM [1].

Dada la notable demanda de los servicios mencionados anteriormente, no es extraño pensar que el acceso a Internet por medio de tecnologías celulares se encuentre en constante aumento. En este escenario se vuelve cada vez más necesario brindar acceso a redes de datos por medio de la telefonía celular a tasas de transmisión cada vez mas elevadas. Esto se refleja claramente en la evolución que han sufrido los protocolos de acceso a Internet en la telefonía celular, particularmente en GSM. Inicialmente GSM proveía acceso a datos a una tasa de $9,6$ kbps mediante el servicio orientado a circuitos denominado CSD[1], hasta llegar hoy en día a velocidades de transmisión de información en el entorno de los 240 kbps gracias al protocolo EDGE.

De los avances alcanzados en la calidad de las técnicas de transmisión de datos mencionados anteriormente, viene de la mano la incorporación de nuevos servicios sobre Internet orientados a clientes con móviles celulares,

[1] Circuit Switched Data

los cuales para su correcto desempeño demandan una calidad del enlace tal que si ésta no se cumple la performance de dichos servicios se ve seriamente afectada (ej. para poder brindar servicios de música online se necesita tener un ancho de banda disponible en el enlace tal que éste sea mayor o igual a la tasa de codificación del audio). Debido a éstos requerimientos es que existe una fuerte necesidad por parte de los ISP[2] de monitorear constantemente la calidad de sus enlaces con el fin de cumplir los SLA[3] establecidos con los clientes.

En el marco del cumplimiento de dichas funciones sobre enalces GPRS-EDGE es que surge como proyecto *MetroCel*, que pretende ser una herramienta útil para un proveedor de servicios que busca monitorear la calidad de los servicios de datos brindados a través de sus enlaces GPRS-EDGE.

1.2. Objetivos

El objetivo central de este proyecto es el de llevar a cabo la implementación de un sistema capaz de poder brindar la mayor cantidad de información posible acerca del comportamiento de los enlaces GPRS-EDGE de la red del operador en la cual se instala el mismo. La propuesta de investigación y desarrollo en esta área surge en el marco de nuestro proyecto de fin de carrera.

Dado que gran parte de los usuarios de servicios de datos sobre telefonía celular que acceden a los mismos directamente desde el móvil utilizan el protocolo WAP (ej. para leer las noticias o la transferencia de MMS) resultó intersante plantearse como otro de los objetivos para el proyecto el del estudio del funcionamiento de dicho protocolo.

1.3. Resumen del Proyecto

Se puede dividir el proyecto en cuatro tareas bien distintas. La primera, en la que se realiza el estudio de las técnicas existentes en el área para la medición de parametros de performance usuales (RTT, jitter, ancho de banda, entre otros) y la implementación de alguna de estas técnicas para ampliar las funcionalidades del software ya existente en *MetroNet*.

La segunda, en la que se realiza un estudio del funcionamiento del protocolo GPRS-EDGE y en base a ello y a los estudios realizados en la primera etapa se desarrolla e implementa un algoritmo que tiene por fin el del análisis cuantitativo y cualitativo acerca del funcionamiento de dicho protocolo desde el punto de vista práctico.

[2]Internet Service Provider
[3]Service Level Agreement

Como tercera tarea se lleva a cabo un estudio del funcionamiento del protocolo WAP y en base a ello se propone e implementa un sistema capaz de brindar indicadores de performance de una conexión establecida a través de dicho protocolo.

Por último se desarrolla una herramienta de software integradora, basada en la modalidad cliente - servidor, que reune todas las características mencionadas anteriormente y agrega funcionalidades de valor agregado, como ser el almacenamiento de datos de forma eficiente en una base de datos, comparación con resultados históricos por usuario y con la totalidad de los mismos (mediante visualizaciones numéricas y gráficas), integración con el proyecto PDT existente *MetroNet*. Además se re-organiza el software previamente existente mediante la aplicación de criterios de Ingeniería de Software.

1.3.1. Estudios iniciales en el área

Al plantearse un proyecto de estas características, en el que la información proviene de diversas fuentes, el primer desafío es realizar una búsqueda que permita obtener una visión general, el estado del arte y las tendencias de investigación en el mundo científico. La búsqueda de trabajos realizados en el área cumple con dos funciones básicas. Por un lado la de adquirir conocimiento acerca de cuáles son los indicadores de performance relevantes en las distintas topologías de las redes de datos mas comunes hoy en día. Por otro lado el de obtener un amplio espectro de conocimiento acerca de las técnicas relevantes utilizadas en el área para efectuar la medición de dichos indicadores de performance.

El primer desafío planteado para resolver en el proyecto fue el de encontrar herramientas útiles para poder hallar capacidades de cuello de botella en enlaces de redes fijas con el fin de incorporarlas como una nueva funcionalidad de *MetroNet*. Luego de haber efectuado los estudios en el área se decidió por la implementación de tres algoritmos; *Pathrate* [16], *Nettimer* [21] y *Pathchirp* [24].

Los primeros dos son algoritmos cuya función es la de determinar la capacidad de cuello de botella de un enlace. La elección de la implementación de dichos algoritmos radicó en que si bien ambos cumplen con el mismo objetivo, para llegar al mismo utilizan técnicas bastante distintas, por lo que se pensó que la implementación de dichas técnicas podría ampliar el espectro de herramientas para poder contar en un futuro al momento de desarrollar un algoritmo propio. Por otro lado el hecho de tener dos métodos distintos para efectuar la medición del mismo parámetro permite la posibilidad de establecer comparaciones entre ambos y, eventualmente, tener mas certeza en el resultado obtenido (intuitivamente, es mas probable que una medida sea

correcta si da lo mismo al calcularse mediante dos algoritmos que recorren caminos diferentes para hallarla).

El restante algoritmo tiene como función el de medir el ancho de banda disponible de un enlace, el cuál se encuentra acotado superiormente por la capacidad de cuello de botella del canal. Si bien inicialmente la idea no era la de implementar un algoritmo que efecutara la medición del ancho de banda disponible, se consideró oportuno implementarlo debido a que de éste modo se incorporarían mas funcionalidades al sistema. Por otro lado, desde el punto de vista de la adquisición de conocimientos en el área, dicho algoritmo estima el ancho de banda disponible en el enlace mediante técnicas bastante peculiares, por lo que a la hora de desarrollar el algoritmo propio se iba a poder contar con un potencial de herramientas mayor aún.

Luego de implementar dichos algoritmos, se incorporaron como una funcionalidad definitiva a *MetroNet*, por lo que seguidamente se procedió a probarlos en diferentes situaciones y topologías de red con el fin de evaluar su rendimiento en diferentes entornos.

1.3.2. Desarrollo e implementación del algorimto para el análisis de la performance de GPRS-EDGE

Esta actividad consta de cuatro etapas bien diferenciadas; el estudio del funcionamiento del protocolo GPRS-EDGE, el relevamiento del comportamiento del canal mediante pruebas para verificar el mecanismo de asignación del mismo, el desarrollo del algoritmo en función de los conocimientos teóricos y prácticos adquiridos y la etapa de implementación en el servidor de *MetroNet*. En este punto vale la pena comentar que si bien se pasaron por todas estas etapas, para poder llegar a la versión final del algoritmo fue necesario iterar varias veces dentro de las mismas.

Estudios teóricos necesarios

Durante esta etapa se consultó toda la bibiliografía disponible en el área, ya sea vía WEB o por medio del material bibiliográfico disponible.

Gracias a esta etapa se logró adquirir una comprensión tal del tema que fue posible, conjuntamente con los estudios y experiencia adquirida en la primera etapa del proyecto, el diseño y desarrollo de un algoritmo propio y específico para estudiar la performance de GPRS-EDGE para el transporte de datos sobre IP en enlaces celulares.

Relevamiento del canal

Con el fin de poder obtener una idea mas concisa de la forma en que trabaja el protocolo con el transcurso del tiempo (recordar que se trabaja sobre sistemas TDMA, por lo que no se dispone del canal todo el tiempo sino de a intervalos), se implementó un sistema que envía paquetes de prueba desde el servidor de *MetroNet* hacia el cliente de manera contínua por un período de tiempo dado. A partir de la información obtenida de los mismos, se puede ver la forma en que el protocolo asigna el canal.

Gracias a la realización de este experimento varias veces en distintas condiciones (horarios, celdas, nivel de potencia, y demás) se pudo adquirir una idea clara y concisa del funcionamiento del protocolo desde el punto de vista práctico.

Diseño y desarrollo del algoritmo

Luego de haber cumplido todas las etapas anteriores se procedió al diseño y desarrollo del algoritmo en sí. Los criterios utilizados fueron:

- Diseñar el algoritmo según las características intrínsecas del canal
- Obtener la mayor cantidad de información posible acerca del funcionamiento del protocolo
- Lograr un equilibrio razonable entre precisión obtenida y coste computacional

Teniendo en cuenta los criterios mencionados anteriormente se procedió a la formulación del algoritmo inicial, el cual fue ajustado durante la etapa de debagueo y pruebas del mismo hasta llegar a la version final, la que se encuentra en producción y pronta para ser probada desde el servidor de *MetroNet*.

Implementación en el servidor de MetroNet

Una vez que el algorimo fue probado y validado en situaciones dispares se procedio a la inclusión final dentro del sistema total, *MetroNet* + *MetroCel*. Una vez terminada esta etapa el mismo quedó pronto para ser utilizado desde cualquier cliente que posea acceso a Internet a través de GPRS-EDGE, ya sea un móvil celular o una computadora personal con disposición del módem correspondiente.

1.3.3. Desarrollo e implementación del sistema para la medición de indicadores de performance en conexiones wap

La actividad desarrollada durante esta etapa del proyecto consta en la implementación de un sistema capaz de brindar al usuario final y/o al operario de red del ISP en cuestión, resultados útiles acerca del comportamiento de la conexión WAP. En esta parte del proyecto se toman como válidas las siguientes hipótesis:

- Las pérdidas de performance más notables sufridas en la conexión se deben basicamente a problemas en la interfaz de aire
- El servidor que corre el sistema desarrollado se encuentra en la misma LAN que el proxy WAP

La idea básica que se utilizó para atacar el problema es que, en el punto donde se encuentra el servidor, las transacciones que se visualizan a nivel de red son básicamente paquetes TCP sobre IP; por lo tanto es posible aplicar toda la batería conocida de estudios de performance sobre redes TCP/IP para atacar el problema planteado.

1.4. Desarrollo de la Herramienta de Software Integradora

Una vez desarrolladas todas las herramientas del proyecto, se procedió a la integración definitiva con *MetroNet*. Si bien todo el software elaborado para *MetroNet* tenia una estructura lógica y cierto orden jerárquico, la misma no se veía reflejada a nivel de software a la hora de clasificar las diferentes clases en sus correspondientes paquetes[4]. Por lo tanto en esta etapa del proyecto las actividades se centraron en dos partes diferentes. La primera consistió en integrar todo el software de *MetroCel* a *MetroNet* para que quedara en producción funcionando como un todo. La segunda parte consistió, mediante la aplicación de los conceptos de Ingeniería de Software, en brindar un cuerpo físico a todo el software en su conjunto de modo tal que la estructura lógica que poseía el software de *MetroNet* antes de comenzar el proyecto finalmente se viera reflejada a nivel de clases en el sistema conjunto de *MetroCel* + *MetroNet*.

[4]Recordar que el lenguaje utilizado para codificar el software del proyecto fue JAVA

1.5. Organización de la Documentación

El próximo capítulo corresponde a las actividades introductorias desarrolladas en el tema de estimación de parámetros de performance en redes de telecomunicaciones.

Capítulo 2 - Estudio de Performance en Redes de Datos Fijas

En los dos capítulos siguientes se detallan y analizan los algoritmos y sistemas implementados para el análisis de redes de datos cuyo acceso al medio es mediante los protocolos GPRS-EDGE y el estudio de la performance del protocolo WAP.

Capítulo 3 - Análisis de Enlaces GPRS-EDGE

Capítulo 4 - Análisis de Performance de WAP

En los últimos dos capítulos se detalla la herramienta de software integradora desarrollada, así como la implmentación de la Ingeniería de Software de la misma.

Capítulo 5 - Descripción del Sistema

Capítulo 6 - Ingeniería de Software

Finalmente la documentación conlcuye con las conclusiones pertinentes.

Capítulo 7 - Conclusiones y Trabajo Futuro

Capítulo 2

Estudio de Performance en Redes de Datos Fijas

2.1. Descripción de la Actividad

En esta etapa del proyecto se realizó un estudio general acerca de las distintas teorías y técnicas existentes en el área de la medición de indicadores de performance en redes de datos. Seguidamente, se escogieron los artículos mas interesantes y se llevaron a cabo, con ciertas modificaciones pertinentes, algunas implementaciones en JAVA de determinados algoritmos de medición de parámetros típicos en área de estudio.

Como actividad conjunta se realizó un trabajo para la asignatura *Evaluación de Performance en Redes de Telecomunicaciones* que consistió en implementar un conocido algoritmo en el área de mediciones de capacidades de cuello de botella[1] en redes de datos, *Nettimer* [21]. Debido a su simpleza y practicidad, en dicho momento el lenguaje escojido para codificar el algoritmo fue *Matlab*. Como tarea específica para este proyecto, dicho algoritmo fue modificado y adaptado al lenguaje de JAVA, con el objetivo de poder ser integrado como herramienta definitiva dentro del software de *MetroNet*.

Además del mencionado algoritmo se implementaron el *Pathrate* y el *Pathchirp*. Al igual que el *Nettimer*, el *Pathrate* es un algoritmo que mide la capacidad de cuello de botella de un enlace. Por su lado, el *Pathchirp* es un algoritmo que mide el ancho de banda disponible[2] de un enlace.

La decisión de implementar dos algoritmos capaces de hallar la capacidad de cuello de botella de un enlace radicó en que si bien ambos tienen el

[1] Se entiende por capacidad de cuello de botella de un enlace como el máximo ancho de banda alcanzable entre terminales.

[2] Se entiende por ancho de banda disponible en un enlace como la tasa neta de transferencia de información en el mismo.

mismo fin, para llegar al mismo se recorren distintos caminos, tanto en el modo en que se envían los paquetes de prueba para efectuar las mediciones como en las técnicas aplicadas luego de terminada la prueba.

Dado que uno de los objetivos principales de esta etapa era el de familiarizarse con las distintas técnicas utilizadas en el ambiente para efectuar la estimación de parametros de performance de redes de datos, se pensó que la implementación de dichos algoritmos podría ampliar y fijar más el conocimiento que el grupo tenía hasta ese momento en el área. Por otro lado, tal como se mencionó en la sección 1, el hecho de tener dos métodos diferentes para estimar el mismo parámetro permite la posibilidad de tener mas certeza en el resultado obtenido.

La idea de implementar un algoritmo que efectúe la medición del ancho de banda disponible de un enlace, surge como iniciativa propia del grupo debido a que se consideró que tener la funcionalidad de conocer dicho parámetro podría ser de utilidad para un cliente del sistema. Sumado a esto, como fue mencionado en la sección 1, desde el punto de vista de la adquisición de conocimientos en el área, *Pathchirp* utiliza una metodología para estimar el ancho de banda del enlace de una forma muy distinta a como lo hacen el común de los algoritmos que estudian el área, por lo que se consideró que la implementación del mismo podría conllevar al enriquecimiento de conocimientos en el grupo en cuanto al manejo de este tipo de técnicas se trata.

2.2. Objetivos

Los objetivos de esta parte del proyecto eran, por un lado familiarizarse con las técnicas usuales de medición de indicadores de performance en redes de datos y, por otro, el desarrollo de nuevas funcionalidades para *MetroNet*.

2.3. Algoritmos Implementados

2.3.1. Pathrate

Como fue mencionado anteriormente, *Pathrate* es un algoritmo que halla la capacidad de cuello de botella de un enlace dado. La implementación del algoritmo está basada en [16].

En dicho artículo se discuten las ideas previas al mismo, en donde se estudian gran parte de los conceptos en cuanto a medición de la capacidad de cuello de botella se trata y se discuten las ventajas y desventajas de cada método, teniendo en cuenta las condiciones en las cuales dichos algoritmos son aplicados.

Dentro de las técnicas explicadas se hace énfasis en las técnicas *Packet Pair* y *Packet Train*. Dado que dichas técnicas son fundamentales para la implementación del algoritmo *Pathrate*, se explicarán a continuación más en detalle.

A. Consideraciones teóricas generales

Packet Pair

La técnica *Packet Pair* consiste simplemente en enviar paquetes de prueba de a pares. La diferencia de tiempos entre los paquetes del par debe ser lo suficientemente pequeña como para asegurarse que los paquetes salgan pegados del emisor. La figura 2.1 ilustra la técnia mencionada.

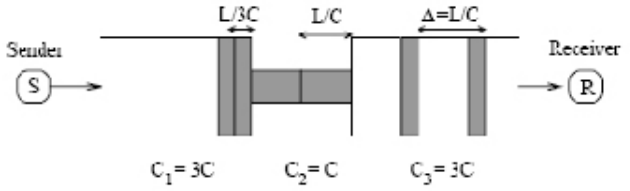


Figura 2.1: Ilustración de la técnica del *Packet Pair*, el ancho de cada enlace corresponde con su capacidad, el tamaño del paquete es L

En el caso de ausencia de *cross traffic*[3], los paquetes llegarán al receptor con cierta dispersión, tal como se muestra en la figura 2.1. En dicho caso, calculando la dispersión de los paquetes tenemos que la capacidad del canal es $C = \frac{L}{\Delta}$.

Sin embargo, en el caso de que exista *cross traffic*, el cálculo de la capacidad del canal no es tan trivial y la técnica *Packet Pair* puede producir resultados erróneos, tanto incrementando como decrementando la capacidad de cuello de botella real del enlace.

La razón de esto es que la distribución de las medidas del ancho de banda son multimodales y algunas modas locales, producidas debido a la prescencia de *cross traffic* pueden ser mayores a la moda de la capacidad de cuello de botella (CM). En los siguientes párrafos se estudian dichos fenómenos mas en profundidad.

Dado un camino con H saltos, definido por la secuencia de capacidades $\rho = \{C_0, C_1, \ldots, C_H\}$, se tiene que la dispersión inmediatamente después de

[3]Se entiende por cross traffic como el tráfico existente en la red que interfiere con los paquetes de prueba utilizados en el experimento

la salida de la fuente es $\Delta_0 = \tau_0 = \frac{L}{C_0}$. Generalizando, la capacidad luego del enlace i será Δ_i. Cuando el paquete llega al receptor, el mismo calcula un ancho de banda estimado $b = \frac{L}{\Delta_H}$. Dado que por lo general Δ_H es variable, si el experimento es repetido varias veces se tendrán valores de b según una distribución β dada.

En el caso que no exista *cross traffic*, la dispersión Δ_i no puede ser nunca menor que la dispersión del salto anterior, Δ_{i-1} y el retardo en la transmisión $\tau_i = \frac{L}{C_i}$ en el salto i, por lo tanto, $\Delta_i = max\{\Delta_{i-1}, \tau_i\}$. Haciendo la extensión a los H saltos se llega a que

$$\Delta_H = \max\{\tau_i\} = \frac{L}{C_n} = \tau_n \tag{2.1}$$

donde C_n y τ_n son la capacidad y el retardo de la transmisión del cuello de botella respectivamente. Por lo tanto, ante la ausencia de *cross traffic*, todos los estimativos del ancho de banda son iguales a la capacidad ($b = C_n = C$).

Ante la prescencia de *cross traffic*, los paquetes de prueba pueden experimentar un retardo adicional en las colas provocado por el mismo. Sean d_i^1 y d_i^2 los retardos en la cola del primer y segundo paquete de prueba en el salto i respectivamente. La dispersión luego del salto i viene dada por ecuación 2.2.

$$\Delta_i = \begin{cases} \tau_i + d_i^2, & \text{si } \tau_i + d_i^2 \geq \Delta_{i-1} \\ \Delta_{i-1} + \left(d_i^2 - d_i^1\right), & \text{en otro caso} \end{cases} \tag{2.2}$$

Dicho fenómeno se puede apreciar mas claramente observando la figura 2.2.

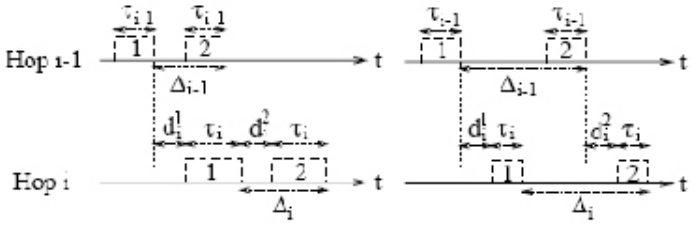


Figura 2.2: Los dos casos de la ecuación anterior

En el caso que $\tau_i + d_i^1 < \Delta_{i-1}$ y $d_i^2 < d_i^1$ entonces $\Delta_i < \Delta_{i-1}$. Dicho efecto puede causar que la capacidad de cuello de botella estimada sea mayor a la real, produciendo una estimación errónea. Esta observación significa que *la capaciadad no puede ser estimada simplemente a través de la dispersión mínima medida.*

En general, se puede afirmar que las estimaciones del ancho de banda que son menores que la CM son causadas por el *cross traffic* que interfiere con el *Packet Pair* y que las estimaciones de ancho de banda que son mayores que la CM son causadas en los los enlaces posteriores al cuello de botella cuando el primer paquete de prueba es retardado en mayor medida que el segundo.

Las observaciones realizadas anteriormente llevan a la explicación de la técnica *Packet Train*.

Packet Train

La técnica *Packet Train* no es otra cosa que una extensión de *Packet Pair* cuando el número de paquetes de prueba es $N > 2$. Por lo tanto, la idea es mandar N paquetes lo mas seguidamente posible de modo tal que a la salida del emisor los mismos salgan juntos. Luego, el receptor mide la dispersión total $\Delta(N)$ desde el el primer hasta el último paquete, calculando así el ancho de banda como

$$b(N) = \frac{(N-1) \cdot L}{\Delta(N)} \tag{2.3}$$

Tal como en el caso del *Packet Pair*, en el caso en que el *cross traffic* no esté presente, la estimación del ancho de banda será igual a la capacidad del canal.

Una gran utilidad que tiene la aplicación del *Packet Train* para medir la capacidad de punta a punta es cuando se tiene un enlace multicanal, es decir que para llegar desde el emisor al receptor existen k caminos distintos que son tomados por los paquetes de prueba. En tal situación, la capacidad del enlace puede ser calculada utilizando la técnica del *Packet Train* con un tren de paquetes de tamaño $N = k + 1$.

A priori, uno podría pensar que el *Packet Train* es una técnica más adecuada que la anterior debido a que es mas robusta al ruido aleatorio introducido por el *cross traffic*. Sin embargo, si bien la dispersión $\Delta(N)$ crece al aumentar L, también lo hace el ruido introducido por el *cross traffic*, por lo que es más probable que los paquetes del *cross traffic* interfieran con los paquetes de prueba *Packet Train*. Dicho fenomeno trae aparejado, por lo general, una sub estimación del ancho de banda total.

Las observaciones anteriores traen aparejado que el tamaño mas adecuado de N para medir la capacidad del canal es la de $N = 2$, por lo tanto cualquier medida que se realice para estimar la capacidad de un canal dado que no se lleve a cabo con la técnica *Packet Pair* tiene mas probabilidad de

verse afectada por el *cross traffic*.

Otra observación interesante es que, al aumentar N, la distribución del ancho de banda (β) tiende a convertirse en unimodal al mismo tiempo que la CM y las modas mayores a la CM ($PNCM$[4]) tienden a disminuir a costas de un aumento en las modas menores que la CM ($SNCM$[5]). Cuando N es lo suficientemente grande y $\beta(N)$ es unimodal, el centro de dicha moda es independiente de N. Dicho centro suele denominarse comunmente Tasa de dispersión asintótica (ADR). El hecho de que la ADR no dependa de N implica que la dispersión del *Packet Train* $\Delta(N)$ es proporcional a $N-1$, como se ver en la ecuación 2.3.

B. Implementación del Algoritmo

Como ya fue mencionado, *Pathrate* puede llegar a necesitar implementar *Packet Pair* y *Packet Train* para llegar a estimar la capacidad del cuello de botella de manera aceptable. Por lo tanto, resulta cómodo separar su implementación en dos fases.

Fase Uno

Como se explicó anteriormente, es más probable poder llegar a discriminar la CM utilizando la técnica *Packet Pair* que *Packet Train*. Por dicha razón es que la fase uno del algoritmo consiste en el envío de un gran número pares de paquetes para así tratar de "descubrir"todas las modas existentes de la distribución del ancho de banda, β.

En [16] se propone que para dicha etapa se utilicen alrededor de 2000 pares de paquetes con un tamaño del orden de 800 bytes (el tamaño del paquete queda determinado empíricamente, haciendo pruebas en redes en las cuales se tiene un grado de conocimiento alto para poder ver cual tamaño es el que se comporta mejor).

Luego de correr la prueba de pares de paquetes, se obtiene un histograma con la distribución del ancho de banda del canal. Un punto importante en este momento es que *el usuario debe elegir el ancho de la resolución del histograma*, tema que es clave para el correcto funcionamiento del algoritmo[6].

En el caso en que la distribución del ancho de banda sea unimodal (usual-

[4]Post-Narrow Common Mode.

[5]Sub-Narrow Common Mode.

[6]Otra alternativa, que no se propone en [16], es la de utilizar un kernel de convolución tal como se explica mas adelante en la sección 2.3.2. Con ésto se halla una distribución contínua, logrando una independencia del factor que influye en la elección del ancho del histograma.

mente pasa cuando los enlaces no estan demasiado congestionados), el proceso de medición termina y se elige como capacidad de cuello de botella al pico de la moda. En otro caso se pasa a la fase dos del algoritmo.

Fase Dos

Como se dijo anteriormente, a medida que va aumentando el tamaño de N, la distribución del ancho de banda, β, tiende a convertirse en unimodal (centrada en la ADR) a la vez que tanto la CM como las $PNCM$ tienden a desaparecer a costa de un aumento de las $SNCM$.

Considerando el intervalo $[\xi^-, \xi^+]$ como el que incluye a la única moda (se hace el experimento con un N tal que se asegura que la distribución de β tenga dichas características), entonces se dice que *la capacidad de la moda (CM) es el pico de la mínima moda en el histograma de la fase uno tal que es mayor a* ξ^+.

La razón de dicha afirmación es que cuando N es lo suficientemente grande como para que β sea unimodal, casi todos los trenes se ven afectado por el *cross traffic* por lo que $\xi^+ < C$. Además, dado que N es el tamaño de tren mínimo que hace que β sea unimodal, el intervalo de la moda aún es lo suficientemente grande como para cubrir todas las modas locales en la $SCDR$[7] de β entre el ADR y C.

En [16] se propone que en esta etapa se manden 400 trenes de paquetes de 1500 bytes de largo N_0. En caso que la distribución no sea unimodal, se repite el mismo experimento pero esta vez con un tren del doble de largo, es decir $2 \cdot N_0$. De éste modo se va iterando hasta conseguir una distribución unimodal. Cuando se llega a éste punto se mide ξ^+ y se procede como se explicó anteriormente.

C. Implementación propia del algoritmo

Si bien se trató de implementar un algoritmo lo mas parecido posible al que se propone en [16], se optó por hacerle algunas simplificaciones así como algunas adaptaciones teniendo en cuenta que en [16] se muestran resultados de performance del algoritmo en enlaces cuyas capacidades andan en el entorno de las decenas de Mbps y la mayor parte de nuestras medidas fueron realizadas en enlaces del orden de las centenas de kbps. Además de lo anterior, se probaron diferentes variantes en los parámetros del experimento para ver como influían en la performance del algoritmo. Las características más destacables del mismo serán expuestas conjuntamente con la de los otros algoritmos en la sección 2.4.

[7] Sub-Capacity Dispersion Range.

Esta implementación consiste en el envío de una tanda de *Packet Pair* (la mayoría de las pruebas fueron realizadas con corridas de entre 600 y 900 pares de paquetes). Seguidamente, el programa hace los cálculos pertinentes y, en caso de ser necesario, se hace la prueba de la fase dos con una tanda de *Packet Train* con la misma cantidad de trenes que en la primera fase pero duplicando la cantidad de paquetes por tren. En caso que la fase dos vuelva a dar multimodal, se repite el procedimiento. Por razones lógicas de programación dicha iteración se realiza un número limitado de veces; en caso que se exceda dicho límite la prueba se considera inválida. Igualmente, como adelanto de la sección 2.4, se comenta que en muy pocos casos hubo una real necesidad de implementar la fase dos del algoritmo ya que con los resultados de la fase uno por lo general se obtuvo una buena idea del ancho de banda del cuello de botella (obviamente tomando como dato que se sabía de antemano cual era el ancho de banda nominal del enlace).

2.3.2. Nettimer

Nettimer [21] es una herramienta presentada en Febrero de 2001 por integrantes del departamento de ciencias computacionales de la Universidad de Stanford con el objetivo de medir la capacidad del enlace cuello de botella de una conexión de banda ancha. Dicha herramienta es capaz de medir el ancho de banda del enlace cuello de botella en tiempo real.

El principio en el cual se basa Nettimer es la técnica *Packet Pair* la cual ya ha sido explicada. Como fue expuesto en la sección A de 2.3.1, los resultados obtenidos con la técnica de *Packet Pair* suelen estar distorsionados por la presencia de *cross traffic*. Para evitar esto, Nettimer implementa técnicas de filtrado que se detallan en la siguiente sección.

A. Técnicas de filtrado

En la figura 2.3 se muestra el gráfico Ancho de Banda con el cual se reciben los paquetes vs. Ancho de Banda con el cual son enviados.

Lo primero que se hace es filtrar los casos en los cuales el ancho de banda recibido es mayor que el ancho de banda enviado. Estos casos, identificados con la letra C en la figura 2.3, se deben a que luego del cuello de botella los paquetes se encuentran con otro enlace en el cual ya hay otros paquetes en su cola. Por este motivo los paquetes de prueba se vuelven a juntar, produciendo que el ancho de banda recibido parezca mayor que el ancho de banda del enlace cuello de botella, pudiendo superar inclusive al ancho de banda con el cual fueron enviados.

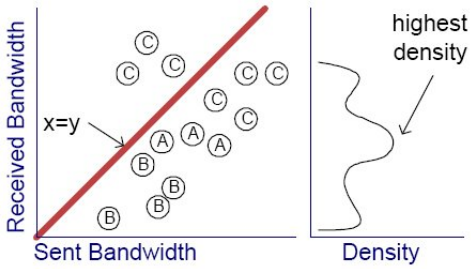


Figura 2.3: Anchos de banda recibidos vs enviados y función de densidad

Las muestras identificadas con la letra B en la figura 2.3 se corresponden con aquellos casos en los que los paquetes no se encolan juntos en el enlace cuello de botella debido a la presencia de *cross-traffic*. Esto causa que los paquetes sean separados más de lo que serían separados por el propio enlace, pudiéndose percibir un ancho de banda menor al de dicho enlace. Para filtrar este caso, se utiliza el principio de que los paquetes que están afectados por *cross traffic* tienden a no estar correlacionados unos con otros mientras que los paquetes que cumplen con las hipótesis de la técnica *Packet Pair*, estarían fuertemente correlacionados. Esto se debe a que el *cross traffic* tiene por lo general paquetes de tamaño aleatorio que llegan a los enlaces en un tiempo aleatorio desde el punto de vista de los paquetes de prueba.

Utilizando esta premisa, es posible calcular una función de densidad del ancho de banda recibido y elegir los puntos donde se maximiza dicha función. La función de densidad esta basada en un núcleo estimador (o kernel), $K(t)$ que cumple con la siguiente propiedad

$$\int_{-\infty}^{+\infty} K(t)\, dt = 1 \tag{2.4}$$

Sea x el ancho de banda recibido y x_i el ancho de banda de la muestra i, la función de densidad es entonces

$$d(x) = \frac{1}{N} \sum_{i=1}^{N} K\left(\frac{x - x_i}{c \cdot x}\right) \tag{2.5}$$

donde c es un coeficiente que controla la exactitud de la función densidad y N es el número de muestras x_i consideradas. Un valor apropiado de c para utilizar es 0.1. La cantidad de muestras N surge del compromiso de considerar las suficientes muestras como para que el algoritmo sea estable aunque a su vez ese valor no puede ser demasiado grande para que permita ejecutar el algoritmo en tiempo real reflejando la situación actual del enlace;

además un valor muy grande de N tiene un peso computacional grande. La función núcleo usada es

$$K(t) = \begin{cases} 1+t, & \text{si } -1 < t < 0 \\ 1-t, & \text{si } 0 > t > 1 \\ 0, & \text{en otro caso} \end{cases} \tag{2.6}$$

Si bien generalmente la función de densidad es un buen indicador del ancho de banda del enlace cuello de botella, en muchas ocasiones las muestras pueden ser enviadas con un ancho de banda menor al mismo, produciendo que los paquetes nunca sean encolados juntos en todo el enlace. Estas muestras, identificadas con la letra D en la figura 2.4, pueden dar un falso estimativo del ancho de banda.

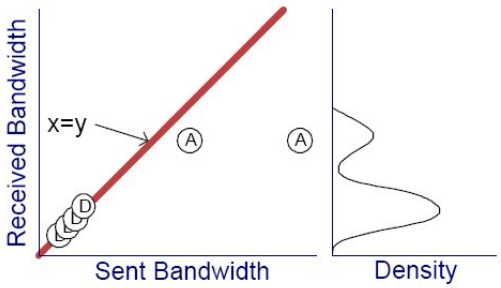


Figura 2.4: Gráfica luego del primer filtrado

Las muestras correspondientes al caso D son diferenciadas del resto debido a que se encuentran muy cerca de la recta *Ancho de banda recibido* = *Ancho de banda enviado*, mientras que las muestras que interesan considerar están en media bastante alejadas de dicha recta, por lo que dichas muestras son también filtradas.

Luego de filtrar estas últimas muestras, el máximo de la función de densidad $d(x)$ corresponde a la capacidad del enlace cuello de botella.

B. Implementación del Algoritmo

Nettimer puede ser implementado con dos posibles configuraciones. La primera en la cual se dispone de dos host que capturan paquetes de los cuales se registran los tiempos de envío y arribo. La segunda configuración solo permite que un host capture paquetes para registrar los tiempos de envío o arribo de paquetes.

Para la primer configuración un host hace las veces de servidor mientras que los clientes interesados en realizar las medidas descargan el software necesario de el. De esta forma ambos host estarían en condiciones de capturar los paquetes y realizar el registro de los tiempos de arribo y llegada. Con estos datos, el servidor será capaz de realizar el cálculo del ancho de banda del enlace cuello de botella aplicando los filtros explicados en el punto A. A esta técnica se le llama *Receiver Based Packet Pair (RBPP)*

Para el caso de que solo un host pueda capturar paquetes, existen dos posibles implementaciones de *Nettimer*. La primera de ellas consiste en que el host que permite capturar paquetes envíe tráfico y registre los tiempos de llegada de los Ack's de capas de transporte y capa de aplicación en lugar de utilizar los tiempos de arribo de los mismos paquetes de pruebas. La desventaja que presenta esta forma de implementar Nettimer es que es susceptible al *cross traffic* del camino de retorno. A esta técnica se la llama Sender *Based Packet Pair (SBPP)*

La otra solución para implementar *Nettimer* en el caso de que un solo host pueda capturar paquetes es la llamada *Receiver Only Packet Pair (ROPP)*, en la cual sólo se registran los tiempos de llegada de los paquetes. Con este mecanismo no se pueden utilizar todos los filtros que se definieron el la sección anterior debido a que no se tienen los tiempos entre paquetes enviados, por lo que tampoco se dispone del ancho de banda enviado.

C. Implementación propia del algoritmo

Para la implementación del software que calcula el ancho de banda del cuello de botella se tuvieron en cuenta los conceptos introducidos por *Pathrate* en cuanto a la técnica *Packet Pair* y la utilización de alguno de los filtros propuestos.

El software recupera de una base de datos los valores correspondientes a los tiempos de envío y arribo de los paquetes de prueba (éste último medido en el cliente). Luego, conociendo el tamaño de los paquetes de prueba se procede a calcular

$$b_1 = \frac{s_1}{t_n^1 - t_n^0} \text{ y } b_0 = \frac{s_1}{t_0^1 - t_0^0} \tag{2.7}$$

donde b_1 y b_0 son los anchos de banda recibido y enviado respectivamente y s_1 es el tamaño del paquete de prueba.

A partir de estos valores se procede al filtrado de los mismos. Serán filtrados todos aquellas muestras que cumplan que $b_1 > b_0$. Es decir, las muestras correspondientes a aquellos paquetes que, habiendo sido enviados

lo suficientemente rápido como para no tener *cross traffic* entre ellos, llegaron a un enlace después del cuello de botella el cual ya tenía paquetes encolados e hizo que los paquetes de prueba se juntaran en el tiempo más de la distancia con la cual fueron enviados. Dicho fenómeno trae aparejado un aumento del ancho de banda recibido.

También se filtran aquellos paquetes que no forman colas en el trayecto, ya que para estos casos, la diferencia entre tiempos de envío y arribos de paquetes es la misma y por lo tanto $b_1 \approx b_0$. Como criterio se toman estos casos como aquellos donde $b_0 - \delta < b_1 < b_0 + \delta$. Juntando este filtro con el anterior, se filtran todas aquellas muestras que cumplan $b_1 > b_0 + \delta$

Una vez filtradas todas las muestras que cumplen la condición anterior y siguiendo con la premisa de que las muestras que no sean afectadas por *cross traffic* van a estar mucho más correlacionadas unas con otras que aquellas muestras que si se vean afectadas, se utiliza una función de densidad que nos permite visualizar el punto de máxima correlación entre muestras. La función de densidad utilizada es la propuesta por *Nettimer*

$$d(x) = \frac{1}{N} \sum_{i=1}^{N} K\left(\frac{x - x_i}{c \cdot x}\right) \tag{2.8}$$

donde

$$K(t) = \begin{cases} 1 + t, & \text{si } -1 < t < 0 \\ 1 - t, & \text{si } 0 > t > 1 \\ 0, & \text{en otro caso} \end{cases} \tag{2.9}$$

x es el ancho de banda recibido y x_i el ancho de banda de la muestra i, el coeficiente c que controla la exactitud de la función densidad vale 0.10 y N es el numero de muestras x_i consideradas luego de la aplicación del primer filtro. De esta manera, el máximo de la función de densidad será el valor hallado del ancho de banda del enlace cuello de botella.

2.3.3. Pathchirp

Pathchirp es una herramienta que estima el ancho de banda disponible en un enlace, la cual fue propuesta en [24].

El algoritmo que propone se basa en mediciones realizadas a partir de tráfico que él mismo genera, con el cual logra congestionar el enlace en cuestión. Dicho tráfico consta de trenes de paquetes a los que llama *chirps* en donde la tasa de envío de tráfico aumenta de manera exponencial para cada paquete.

A. Nociones básicas

Se supondrá como hipótesis que se tiene una ruta entre dos terminales donde hay múltiples encolamientos, por este motivo se modela la ruta como una serie de nodos que almacenan y envían, cada uno con determinada tasa de servicio y todos con sistemas de encolamiento FIFO[8].

Para hallar el ancho de banda uno de los terminales genera tráfico, que consiste en m trenes de N paquetes donde varía el tiempo entre envíos de paquetes sucesivos de forma exponencial. Se fijan para esto dos parámetros, el tiempo entre los dos últimos paquetes, T, y el factor de propagación, γ.

De este modo el tiempo entre los dos primeros paquetes del tren será $T \cdot \gamma^{N-2}$ y este tiempo se divide entre γ para cada paquete. Esto se repite sucesivas veces, dejándose un tiempo τ entre trenes.

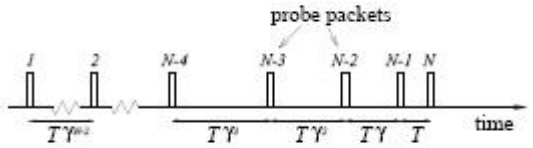


Figura 2.5: Separación de tiempos entre paquetes

A partir de los datos obtenidos, la información que se utilizará será solamente el retardo relativo entre paquetes, por lo que no se requiere sincronización entre relojes.

Sea $q_k^{(m)}$[9] el retardo de encolamiento del paquete k, $t_k^{(m)}$ el instante en el cual se envía el paquete k y $\Delta_k^{(m)}$ el intervalo entre los paquetes k y $k+1$, entonces

$$\Delta_k^{(m)} = T \cdot \gamma^{N-1-k} \tag{2.10}$$

Si los paquetes enviados son de P bytes se define la tasa instantánea de envío como

$$R_k^{(m)} = P/\Delta_k^{(m)} \tag{2.11}$$

Se puede suponer que en el caso de *cross traffic* de tasa constante se tiene

[8]First In First Out.

[9]Si los relojes de ambos terminales no están sincronizados pero son estables, podemos suponer que la diferencia entre las marcas de tiempo de ambos es igual al retardo de encolamiento más una constante

que

$$q_k^{(m)} = 0, \text{ si } B\left[t_1^{(m)}, t_N^{(m)}\right] \geq R_k \tag{2.12}$$

$$q_k^{(m)} > q_{k-1}^{(m)}, \text{ en otro caso} \tag{2.13}$$

siendo $B\left[t_1^{(m)}, t_N^{(m)}\right]$ el ancho de banda disponible del enlace. A partir de esto se puede hacer una estimación de dicho ancho de banda, diciendo que éste es igual a R_{k^*}, siendo k^* el paquete a partir del cual los retardos de encolamiento comienzan a crecer.

Para levantar la hipótesis de *cross traffic* de tasa constante (lo cual evidentemente simplifica la realidad) se supondrá que los retardos de encolamiento no varían en forma monótona, sino que pueden aumentar y disminuir debido a la presencia de *cross traffic* en ráfagas.

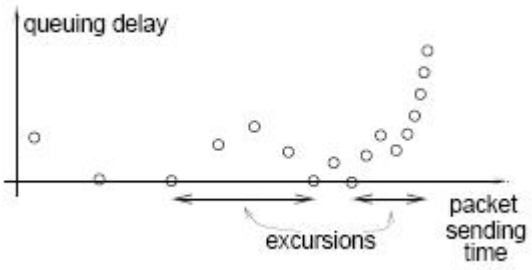


Figura 2.6: Evolución de los retardos en las colas

Se observa en la figura 2.6 que el diagrama se divide en *excursiones* que consisten en evoluciones desde cero, con retardos de colas crecientes por varios paquetes consecutivos debido a ráfagas de *cross traffic*. Se nota que las primeras excusiones terminan con un retorno a cero de los retardos, debido a que la tasa de envío no supera aún la capacidad de cuello de botella del enlace y en ausencia de *cross traffic* las colas logran volver al mínimo. Por otro lado la última generalmente termina con un contínuo aumento de los retardos en las colas, esto se debe a que la tasa de envío es superior a la capacidad del enlace cuello de botella y por lo tanto se está saturando el enlace.

A partir de la forma de dicha gráfica es que se estima el ancho de banda disponible por paquete $E_k^{(m)}$ y luego se hace un promedio de estas estimaciones ponderado por los tiempos entre envíos.

$$D^{(m)} = \frac{\sum_{k=1}^{N-1} E_k^{(m)} \Delta_k}{\sum_{k=1}^{N-1} \Delta_k} \tag{2.14}$$

Finalmente se promedian los $D^{(m)}$ hallando de este modo la estimación correspondiente del ancho de banda disponible.

B. Segmentación

Para hallar el ancho de banda estimado por paquete $E_k^{(m)}$ es necesario dividir la gráfica de retardos de encolamiento en excursiones, teniendo en cuenta que cada muestra puede o no pertenecer a una excursión.

Se supondrá que un retardo de encolamiento creciente significa que la tasa de envío instantánea es mayor que el ancho de banda disponible en dicho momento mientras que un retardo decreciente significa lo contrario. Esto se puede expresar como

$$E_k^{(m)} \geq R_k, \text{ si } q_k^{(m)} \geq q_{k+1}^{(m)} \tag{2.15}$$

$$E_k^{(m)} \leq R_k, \text{ en otro caso} \tag{2.16}$$

En un escenario con un solo salto se tiene que la ecuación 2.15 es exactamente verdadera, mientras que la ecuación 2.16 no necesariamente siempre lo es. Por ejemplo, en el caso que los paquetes k y $k+1$ sean enviados muy espaciados en el tiempo, el hecho de que $q_k^{(m)} < q_{k+1}^{(m)}$ no aporta información a cerca de $E_k^{(m)}$ ya que es necesario introducir congestión en la red.

Por este motivo es que se realiza la separación en excursiones, para luego aplicar la ecuación (2.16) solamente dentro de estas regiones. Intuitivamente si $q_k^{(m)}$ aumenta y se mantiene mayor que 0 por varios paquetes consecutivos, estos paquetes pertenecen todos al mismo *busy period*[10] de un enlace congestionado en la ruta. Al estar congestionando la ruta es de esperarse que $q_k^{(m)} < q_{k+1}^{(m)}$ validando de este modo la ecuación 2.16.

A continuación se verá cómo se hallan dichas excursiones. Cada paquete i que cumpla que $q_i^{(m)} < q_{i+1}^{(m)}$ es un candidato a empezar una excursión. Se define el final de dicha excursión como el primer paquete j que cumple

$$q(j) - q(i) < \frac{\max_{i \leq k \leq j}[q(k) - q(i)]}{F} \tag{2.17}$$

donde F es un parámetro del algoritmo llamado *decrease factor* o factor de reducción. De este modo en j el retardo de encolamiento relativo a $q(i)$ ha decrecido por un factor F con respecto al máximo retardo relativo entre los paquetes comprendidos entre i y j. Además de cumplir esta condición, se impone un largo mínimo de excursión a través de otro parámetro L llamado *busy period threshold*. Por lo tanto si $j - i > L$ todos los paquetes entre i y

[10] intervalo de tiempo en el cual la cola no está vacía

j forman una excursión.

Generalmente la última excursión no termina, ya que se están enviando paquetes a una tasa mayor que la capacidad del enlace y por lo tanto los retardos son cada vez mayores. En este caso se toma el último paquete como fin de la excursión y luego ésta será tratada de una forma especial.

C. Estimación por paquete

Una vez realizada la segmentación identificando excursiones se estima el ancho de banda disponible por paquete. Para ello se clasifica cada paquete en una de las siguientes tres categorías.

Caso (a): Si k pertenece a una excursión que termina y además $q_k^{(m)} \leq q_{k+1}^{(m)}$ entonces se define

$$E_k^{(m)} = R_k \tag{2.18}$$

lo cual satisface la ecuación (2.16).

Caso (b): Si k pertenece a la última excursión y ésta no termina, entonces se tiene que

$$E_k^{(m)} = R_l, \forall\ k > l \tag{2.19}$$

siendo l el comienzo de la excursión.

Se observa que en la última excursión la tasa de envío puede ser mucho mayor que la capacidad del enlace, por este motivo no es adecuado utilizar la ecuación 2.18 en este caso. Cabe notar de todos modos que si un paquete k cumple que $q_k^{(m)} > q_{k+1}^{(m)}$ entonces de acuerdo a la ecuación 2.15 se tiene que $E_k^{(m)} > R_k > R_l$ por lo que la ecuación 2.19 es una subestimación de $E_k^{(m)}$ para dicho k.

Caso (c): Para los restantes paquetes se define

$$E_k^{(m)} = R_l \tag{2.20}$$

Esto incluye todos los paquetes que no pertenecen a ninguna excursión y a aquellos que pertenecen a una excursión pero el retardo de encolamiento es decreciente. En el caso que la última excursión no termine, se define $l = N - 1$.

D. Detalles de implementación

Los paquetes de prueba que utiliza el algoritmo son paquetes UDP, los parámetros son el tamaño de los paquetes P, el factor de propagación γ, el factor de decrecimiento F, el umbral L y el tiempo entre trenes de paquetes τ.

En [24] se propone que los paquetes viajen en un solo sentido y los datos sean procesados en el receptor para prevenirse de interferencias en canales que no son *full-dúplex*. En la implementación realizada los paquetes son rebotados para guardar la información en la base de datos, procesándolos en el terminal que envía y recibe los paquetes.

Otra consideración [24] es que descarta los paquetes que se encolan en el cliente antes de ser procesados, lo cual puede deberse a que éste está ocupado con otro proceso. Cuando esto ocurre, las marcas de tiempo que realiza el cliente hacen que el algoritmo pueda confundir este encolamiento con encolamientos en la ruta; para eviarlo se propone utilizar un umbral d. Cuando se detectan dos marcas de tiempo entre las cuales hay una separación menor que dicho umbral se supondrá que los paquetes se encolaron en el cliente y por lo tanto el tren no será tenido en cuenta.

Esto no fue implementado debido a que el umbral d propuesto es de 30 μs y las marcas de tiempo impuestas por el sistema implementado tienen una resolución de 1 ms.

Por último vale aclarar que el algoritmo también desecha aquellos trenes que pierden paquetes.

2.4. Resultados Obtenidos

Las pruebas realizadas fueron llevadas a cabo en dos escenarios bien distintos. Por un lado se implementó una red local con enlaces nominales de 10 Mbps, cuyo esquema se muestra en la figura 2.7.

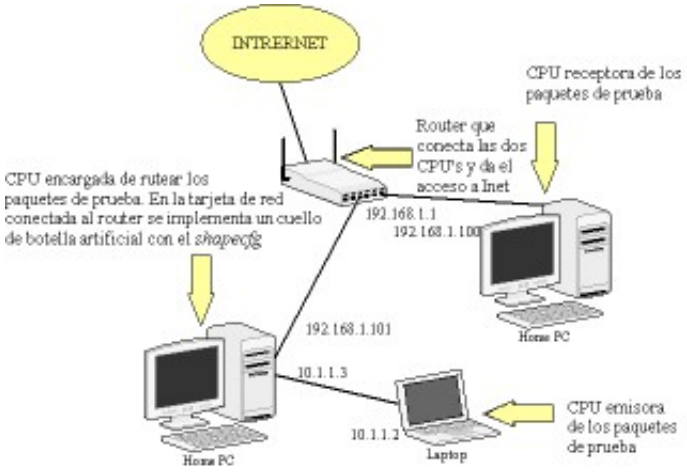


Figura 2.7: Esquema de la red armada para implementar alguna de las pruebas

Mediante el comando *shapecfg* de Linux, se implementó un cuello de botella de capacidad variable. Esto permitió poder comprobar la performance de los algoritmos en una red cuya capacidad era relativamente alta comparada con los enlaces sobre la red pública que se encontraban disponibles. En este escenario se hicieron pruebas con y sin tráfico, aunque cabe destacar que el tráfico fue introducido artificialmente, por lo que dichas medidas no son un reflejo del todo fiel de una situacion normal en enlaces con capacidades del orden de los Mbps.

El otro escenario de pruebas consisitió en correr los experimentos a través de Internet. Si bien la mayoría de las pruebas fueron realizadas entre el servidor de *MetroNet* y ADSL residenciales, también se realizaron experimentos entre dos ADSL, entre ADSL y un módem V.90 de 56kbps y entre ADSL y un Internet Class. Las pruebas fueron realizadas con y sin tráfico forzado por los terminales del enlace así como a diferentes horas del día con el propósito de ver si en algún momento la actividad de la red pública provocaba que el cuello de botella se trasladara al backbone en vez de a uno de los terminales del enlace, como es de esperar cuando se manejan enlaces del tipo anteriormente mencionados.

A continuación se muestran los resultados obtenidos para las pruebas más interesantes y representativas, así como los comentarios pertinentes al

respecto.

2.4.1. Caso de enlace entre un módem V.90 y un ADSL residencial

En el caso de este enlace en particular se tuvo un inconveniente que a priori no estaba considerado. Inicialmente los paquetes de prueba para los algoritmos se enviaban con un relleno que consistía de un timestamp más una cadena de caracteres idénticos hasta completar el tamaño del paquete dado para la prueba en cuestión.

Al efectuar las mediciones con *Nettimer* y *Pathrate* se observó que la capacidad del enlace era bastante mayor a la esperada. A priori se esperaría que la capacidad del enlace ronde los 56kbps (impuesta por la capa física utilizada por el módem V.90). Lo interesante es que al efectuar las pruebas se observó que la capacidad del enlace rondaba los 100kbps.

La explicación a este fenómomeno es que cuando el módem procesa datos cuya tasa de redundancia es significativa, utiliza un algoritmo de compresión con el propósito de aumentar el ancho de banda aparente de la tranferencia. Para solucionar este pequeño inconveniente se modificó el enviador de paquetes rellenándolo con el mismo timestamp de antes más una secuencia de caracteres pseudoaleatoria.

Una vez sorteado dicho inconveniente se repitieron los experimentos nuevamente pudiendo comprobar como efectivamente la capacidad del enlace disminuyó al orden de los 50kbps.

A modo de ejemplo, en las figuras 2.8 y 2.9 se muestra la performance del *Pathrate* con el enviador de paquetes modificado. Como se puede observar en la figura 2.8 luego de realizada la fase uno existen varios candidatos para determinar la CM por lo que se procede a realizar la fase dos del algoritmo, dando como resultado lo que se muestra en la figura 2.9. En este caso se alcanzó una distribución relativamente unimodal para un tamaño de tren de prueba de 10 paquetes.

Luego de comparar las dos distribuciones se puede observar que la menor moda candidato de la fase uno mayor que el ADR obtenido en la fase dos se encuentra en aproximadamente 43 kbps, lo que es una capacidad razonable para un enlace con las características mencionadas anteriormente.

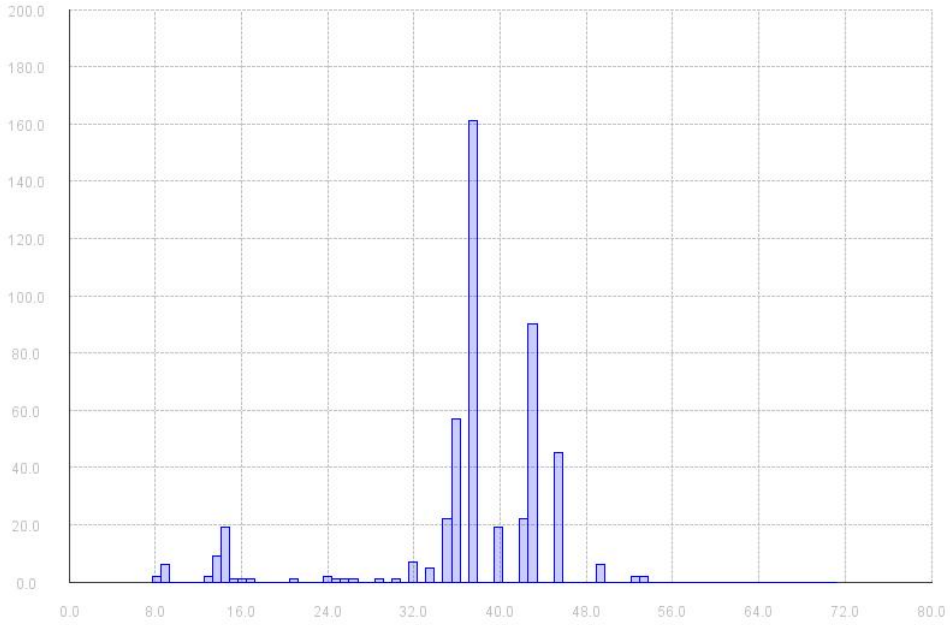


Figura 2.8: Fase uno del Pathrate con cross trafic intoducido en el enlace

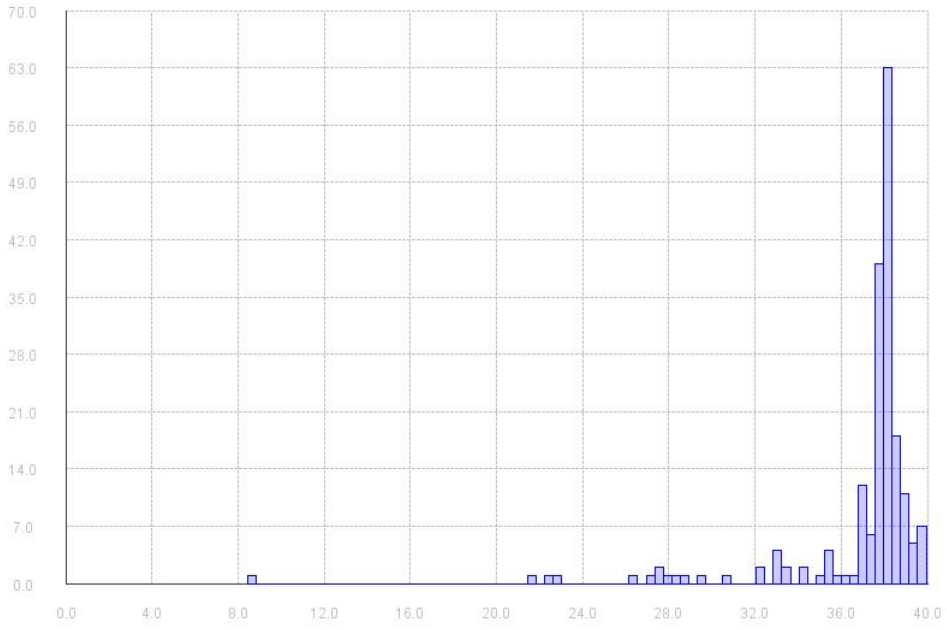


Figura 2.9: Fase dos del Pathrate con cross trafic intoducido en el enlace

Para el caso de *Nettimer* se realizaron pruebas el mismo día a la misma hora dando como resultado que para un tamaño de paquete de 800 bytes la capacidad del cuello de botella resultó de 35.5 kbps mientras que para un tamaño de paquete de 1400 bytes la misma resultó de 42 kbps. Esto refleja que cuando se utilizó un tamaño de paquete cercano al MTU los resultados obtenidos de ambos algoritmos son similares. Este comportamiento se comprobó a lo largo de todos los experimentos realizados.

2.4.2. Caso de enlace entre dos ADSL residenciales

En éste tipo de enlaces se realizaron varias pruebas variando por lo general los tamaños de los paquetes, trenes y demás. Como fue mencionado anteriormente es de esperar que en este caso las capacidades del cuello de botella del enlace ronden los 128kbps.

De las figuras 2.10 a 2.13 se muestra, a modo de ejemplo, experimentos realizados con *Pathrate* y *Nettimer* el mismo día a la mimsa hora con y sin tráfico introducido artificialmente por el emisor de los datos.

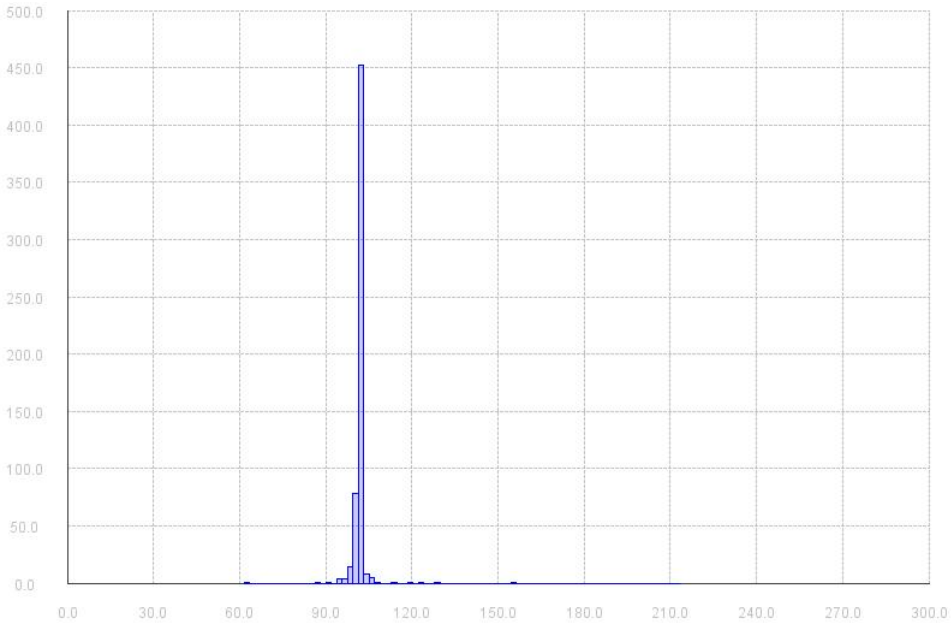


Figura 2.10: Fase uno del Pathrate sin tráfico

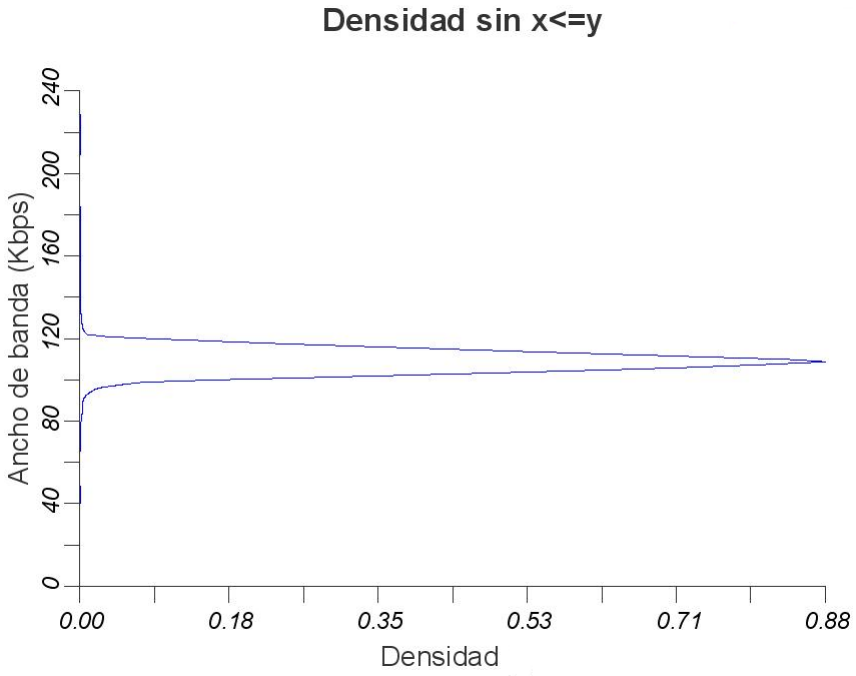


Figura 2.11: Nettimer sin tráfico

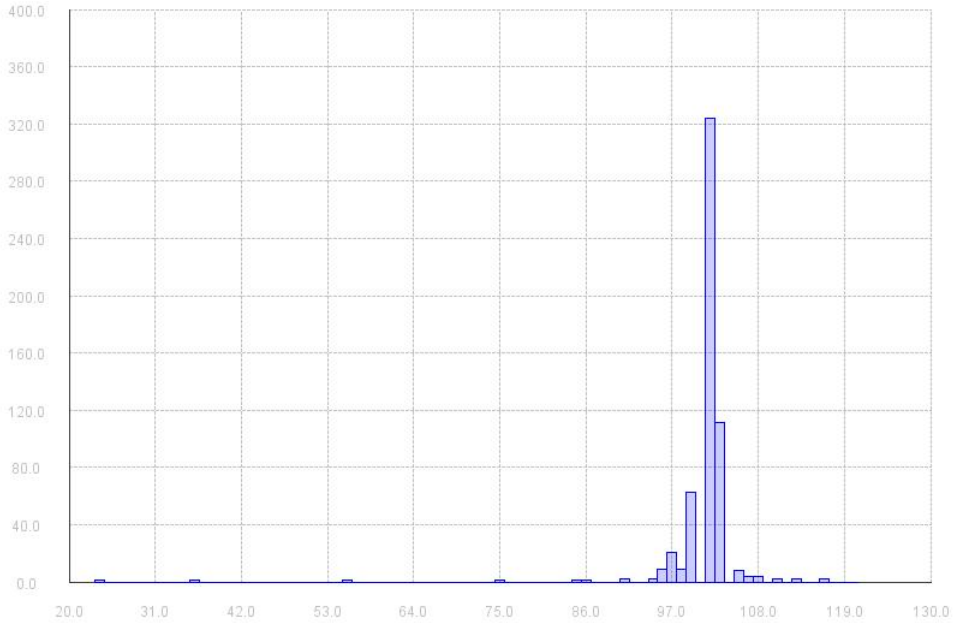


Figura 2.12: Fase uno del Pathrate con tráfico

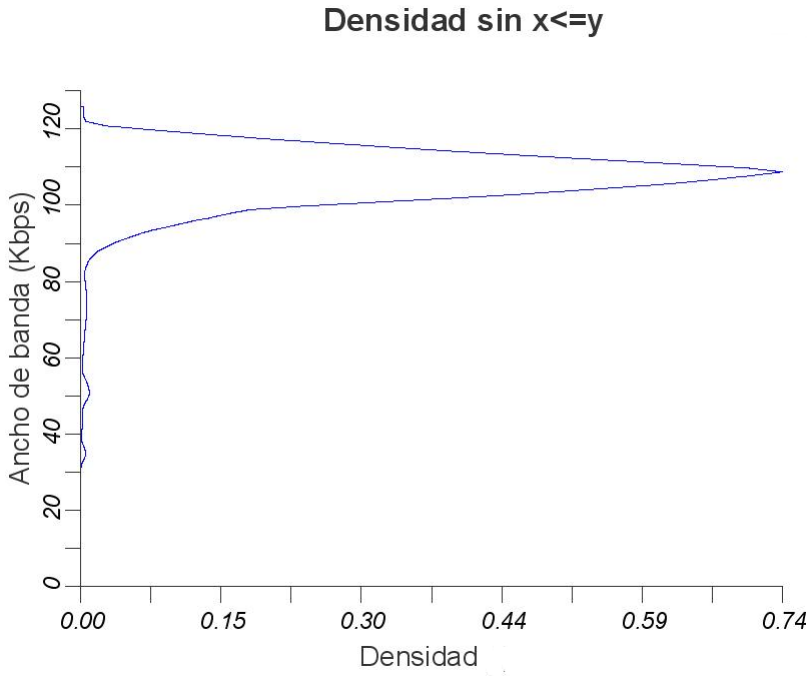


Figura 2.13: Nettimer con tráfico

De las figuras 2.10 y 2.11 se desprende que cuando no hay *cross traffic* introducido artificialmente, en ambos algoritmos la capacidad obtenida se encuentra entre los 105 y 110 kbps, valores considerados aceptables teniendo en cuenta que la capacidad nominal es de 128 kbps (que además no se tiene total certeza de que sea la existente realmente). En el caso de presencia de *cross traffic* se tiene que las capacidades registradas en ambos algoritmos son levemente menores que en el primer caso sin llegar a ser dichas diferencias demasiado importantes, tal como se aprecia en las figuras 2.12 y 2.13. Como observación extra se puede apreciar cómo en la función densidad obtenida al aplicar *Nettimer* aparecen pequeñas modas introducidas debido a la prescencia de *cross traffic*, esto también se puede apreciar en el histograma de *Pathrate* al ver como el mismo se encuentra más distribuido.

Vale la pena destacar que durante todos los experimentos realizados en este tipo de enlaces se observó un comportamiento similar al descripto anteriormente.

En las figuras 2.14 y 2.15 se muestran los retardos en las colas obtenidos al correr *Pathchirp* en el enlace con y sin presencia de trafico introducido artificialmente en la red.

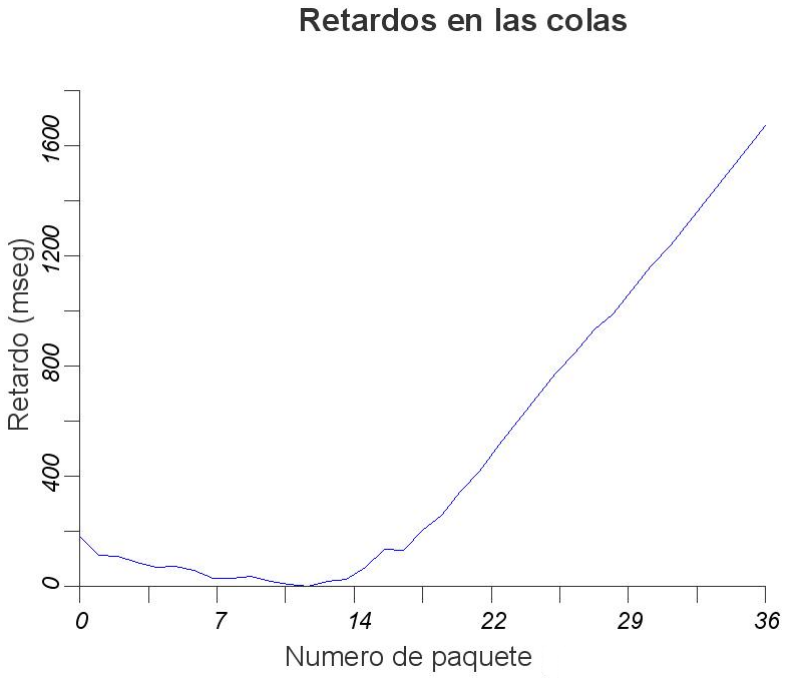


Figura 2.14: Pathchirp sin tráfico

Al correr el algoritmo se obtiene que el ancho de banda disponible sin tráfico es de 110 kbps mientras que el ancho de banda disponible cuando se introduce tráfico en la red es de 69 kbps. Los resultados de este experimento tienen cierta coherencia teniendo en cuenta que ante la ausencia de *cross traffic* el ancho de banda disponible tendría que ser levemente inferior a la capacidad de cuello de botella del enlace, mientras que ante la presencia de *cross traffic* el mismo tendría que bajar sensiblemente.

Otra observación interesante es ver qué sucede cuando aparece un encolamiento grande de paquetes en un punto de la red posterior al cuello de botella. Para esta situación se observa que *Nettimer* funcionó correctamente pero al realizar la prueba del *Packet Pair* para *Pathrate* la gráfica no resultó ser unimodal. Tras realizar la prueba de *Packet Train* observamos que se pierden muchos paquetes y por lo tanto se hace difícil vislumbrar la capacidad del enlace. De este modo se conlcuye que *Nettimer* es más robusto ante los encolamientos posteriores al cuello de botella, ya que para sortear este problema aplica un filtro a los datos obtenidos en vez de realizar un nuevo experimento.

Figura 2.15: Pathchirp con tráfico

2.4.3. Caso de enlace entre un Internet Class y un ADSL residencial

Los experimentos llevados a cabo en dicha situación fueron implementados de modo tal que el emisor de los paquetes fuera el terminal cuyo acceso a Internet de subida era el de mayor capacidad, o sea el Internet Class. De este modo se pudo comprobar la performance de los algoritmos cuando el cuello de botella en teoría era del orden de los 512 kbps. Además, se pudo hacer la prueba de intentar ver si el ancho de banda disponible pasaba en algún momento del día a estar limitado por el *backbone* de la red, para esto se efectuaron pruebas con las mismas características a diferentes horas.

Luego de haber hecho las mismas pruebas durante varios días a diferentes horas se pudo llegar a la conclusión que el ancho de banda disponible siempre estuvo limitado por uno de los dos terminales involucrados.

2.4.4. Caso de enlace entre el servidor de MetroNet y diferentes tipos de clientes

Luego de concluída la etapa de desarrollo y debagueo de los diferentes algoritmos se decidió incorporar definitivamente dichas funcionalidades al servidor de *MetroNet*. Una vez realizada la integración se procedió a la utilización de los algoritmos desde clientes con distintos tipos de acceso a Internet. Las características de los resultados obtenidos fueron similares a las presentados anteriormente.

2.5. Conclusiones

Una vez concluida esta etapa se alcanzaron los objetivos principales de la misma. Por un lado se logró familiarizar con el software y el funcionamiento de *MetroNet*. Por otro lado se adquirieron conocimientos acerca de las técnicas más comunes de estimación de parámetros de performance en el área de estudio.

Si bien los resultados generales obtenidos fueron buenos, se observó que al ir aumentando la capacidad de los enlaces en los que se probaron los algoritmos el valor obtenido por los mismos se aparta de la capacidad nominal del canal. Dicho efecto es debido a que al efectuar pruebas en enlaces de capacidades altas las diferencias de tiempos de arribos entre paquetes disminuye, lo que provoca que la incertidumbre en los cálculos aumente[11]. Si bien dicho fenómeno es una limitante del sistema se optó por no detenerse demasiado en este punto debido a que el objetivo principal del proyecto es el de estimar parámetros de performance sobe enlaces GPRS-EDGE, cuya capacidad máxima es sensiblemente menor a las capacidades tratadas en esta etapa.

[11] Por más información sobre este tema ver el punto B de la sección B.

Parte II

Estudios de Performance de los Protocolos GPRS-EDGE

Capítulo 3

Análisis de Enlaces GPRS-EDGE

3.1. Conceptos Introductorios

3.1.1. Descripción de la actividad

Durante esta etapa del proyecto se llevó a cabo el diseño e implementación de un algoritmo pensado específicamente para el análisis de la performance sobre enlaces GPRS-EDGE.

Previamente, para poder cumplir con dicho fin, fue necesario atravesar por varias etapas que lógicamente se tienen que cumplir en cualquier proceso de desarrollo e investigación.

- Estudios acerca del funcionamiento de la red GSM-GPRS
- Lectura de publicaciones sobre proyectos similares en el área
- Relevamiento del comportamiento del canal para verificar el ajuste de la teoría a la práctica
- Desarrollo del algoritmo teniendo en cuenta las características particulares del enlace
- Debagueo y ajuste del algoritmo mediante pruebas en enlaces reales
- Integración del algoritmo en su versión final al software de *MetroNet*

3.1.2. Objetivo

Si bien en un principio la actividad a realizar durante esta etapa no tenía un objetivo demasiado concreto, debido al carecer de un conocimiento previo cabal del problema, el objetivo inicial planteado para la misma era obtener la mayor cantidad posible de información acerca del funcionamiento del protocolo GPRS-EDGE.

Cuando se hace referencia al comportamiento de la asignación del canal de datos hay que tener en cuenta que nos encontramos estudiando un medio de acceso a la información que utiliza un sistema de asignación del canal en el tiempo. Esto significa que uno no dispone necesariamente del canal para transferir información cuando lo precisa, sino que el sistema le asigna recursos teniendo en cuenta la cantidad de recursos físicos en la radiobase que sirve al cliente (cantidad de *time slots* disponibles para la transferencia de datos en la radiobase) y la cantidad de usuarios en la radiobase en cuestión que pretenden transferir datos en el mismo instante que el cliente, es decir, con cuántos usuarios se tienen que compartir los recursos disponibles en el sistema.

Por lo tanto, teniendo en cuenta el marco del problema, se plantearon como objetivos a tratar de resolver por el algoritmo los que se listan a continuación.

- Capacidad instantánea disponible
- Throughput total
- Throughput en el período en que se está utilizando el canal de forma activa
- Porcentaje de tiempo de actividad en el canal
- Porcentaje de paquetes perdidos durante la prueba
- RTT entre el servidor y el cliente
- Jitter en las mediciones de RTT tomadas

A. Capacidad instantánea disponible

Es la capacidad de cuello de botella del canal asignada por el sistema en el momento de realizar la transferencia de datos. Se toma como hipótiesis del problema que la capacidad existente en la parte fija de la red es mucho mayor que en la red de acceso. Es decir que la capacidad de cuello de botella se da en la transmisión de los datos en la interfaz de aire. Dicha hipótesis viene fundamentada por el hecho de que en cualquier red interna de un ISP las capacidades de transmisión rondan las centenas de Mbps o incluso los Gbps. Por otro lado, la salida a Internet disponible por el servidor del IIE es del orden de los Mbps mientras que, como máximo, la capacidad alcanzada en la interfaz de aire puede ser de 240 kbps, lo que corresponde a disponer de 4 *time slots* con una codificación EDGE del tipo MCS-9.

Por las razones mencionadas anteriormente se puede decir que la capacidad del canal queda determinada en la red de acceso, la cual se caracteriza por las siguientes variables.

Multiclass Slot	(DL Slots)	(UL Slots)	(Slots Activos)
1	1	1	2
2	2	1	3
3	2	2	3
4	3	1	4
5	2	2	4
6	3	2	4
7	3	3	4
8	4	1	5
9	3	2	5
10	4	2	5

Tabla 3.1: Clasificación de terminales según su Multiclass Slot Type.

- Clase del móvil del cliente en cuestión
- Potencia de la señal recibida desde la radiobase al móvil
- Configuración particular de la celda

Clase del móvil del cliente

La clase del móvil del cliente es una clasificación dentro de los tipos de terminales la cual determina cuántos *time slots* simultáneos puede utilizar el mismo para la transferencia de datos (se indican en forma separada tanto para el uplink como para el downlink). En la tabla 3.1 se listan los tipos de clases mas comunes.

A modo de ejemplo, se puede ver que un móvil de clase 10 puede utilizar hasta 4 *time slots* para recibir datos y hasta 2 *time slots* para enviar. Sin embargo, simultáneamente solo puede utilizar hasta 5 en su conjunto, es decir, puede trabajar en la modalidad $4DL + 1UL$ o $3DL + 2UL$ pero, por ejemplo, nunca en una modalidad del tipo $4DL + 2UL$.

Potencia de la señal

La potencia que recibe el móvil desde la radiobase es la determinante de la tasa efectiva por *time slot* a la que se transfieren los datos. Esto es así debido a que cuanto menor es la potencia de la señal, mayor es la probabilidad de que ocurran errores en los bits transferidos. Por dicho motivo cuanto menor es la potencia de la señal mas bits de redundancia son utilizados para poder reconstruir la información en el otr extremo de manera satisfactoria. Al ser necesario mandar mas bits de redundancia por bit de información transferido la tasa neta de información disminuye. Por lo tanto, cuanto menor es la potencia de la señal, menor va a ser la tasa de transferencia de información por *time slot*. En las tablas 3.2 y 3.3 se listan los diferentes

Code Scheme	BW por TS (en capa2, en kbps)
1	8.0
2	12.0
3	14.4
4	20.0

Tabla 3.2: Code Scheme en GPRS.

MCS	BW por TS (en capa2, en kbps)
1	8.8
2	11.2
3	14.8
4	17.6
5	22.4
6	29.6
7	44.8
8	54.4
9	59.2

Tabla 3.3: Code Scheme en EDGE.

esquemas de codificación posibles según el nivel de potencia registrado en el móvil.

Configuración particular de la celda

El proveedor del servicio puede elegir, según las características de la demanda de tráfico dadas en la celda en cuestión, la cantidad de *time slots* disponibles para la transferencia de datos en la misma. Por lo tanto, alguna de las variables de configuración de la celda pasan a ser la cantidad de *time slots* fijos asignados para la transferencia de datos (y si son EDGE o GPRS), la cantidad de *time slots* fijos asignados para el tráfico de voz y la cantidad de *time slots* on demand para datos. Por lo general en este último tipo de *time slot* se le da la prioridad al tráfico de voz, pero en caso que existan *time slots* osciosos para el tráfico de voz dichos recursos pueden ser utilizados para el tráfico de datos.

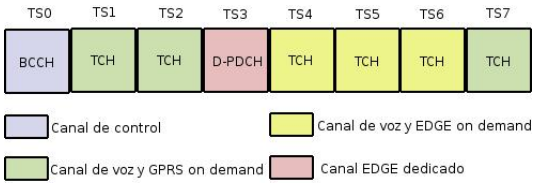


Figura 3.1: Configuración típica de una trama GSM.

B. Throughput total

El throughput total del experimento se define como la cantidad de datos transferidos sobre el tiempo total transcurrido durante dicha transferencia. Es un indicador más perceptible por el usuario que la capacidad del canal ya que si bien un usuario puede obtener altas capacidades de transferencia mientras se encuentra activamente utilizando el canal, hay ciertos momentos en que el canal se encuentra siendo utilizado por el resto de los usuarios y el usuario en cuestión se encuentra inactivo desde el punto de vista de la transferencia de datos. Dicho fenómeno provoca que la velocidad de transferencia real percibida por el usuario sea sensiblemente menor a la capacidad de transferencia obtenida por el mismo cuando se encuentra haciendo uso del canal. Esto es tenido en cuenta al momento de efectuar el cálculo del throughput total y, por lo tanto, el mismo es un indicador mas fiel de la velocidad de la conexión que dispone el cliente en dicho momento.

C. Throughput en el período en que se está utilizando el canal de forma activa

El throughput obtenido durante el período en que el cliente se encuentra utilizando activamente el canal se calcula como la cantidad de datos transferidos mientras el cliente estuvo utilizando de forma activa el canal sobre dicho tiempo. Es una medida que brinda una idea de la velocidad promedio que obtiene el cliente cuando se encuentra utilizando activamente el canal. Intuitivamente dicho valor se tiene que encontrar por debajo de la máxima capacidad registrada dentro de dicho período de actividad[1] y por encima del throughput total.

[1]Durante un mismo período de actividad no necesariamente tiene que haber una única capacidad registrada. A modo de ejemplo, si mientras el cliente se encuentra utilizando el canal de forma activa hay un cambio sustancial en la potencia recibida en el móvil, variará la velocidad de transmisión efectiva de los datos.

D. Porcentaje de tiempo de actividad en el canal

El porcentaje de tiempo de actividad en el canal se calcula como la cantidad total de tiempo en que el cliente se encuentra utilizando el mismo sobre el tiempo total transcurrido en la prueba. Dicho indicador tiene una estrecha relación con la cantidad de usuarios del servicio de datos que se encuentran utilizando activamente los mismos recursos que el cliente. Cuanto más grande sea dicho porcentaje menor cantidad de usuarios estarán compartiendo el canal.

$$\%Act = \frac{\sum T_{on}}{T_{tot}} \tag{3.1}$$

E. Porcentaje de paquetes perdidos durante la prueba

El porcentaje de paquetes perdidos durante la prueba es simplemente la cantidad de paquetes perdidos en la descarga hacia el cliente sobre la cantidad total de paquetes enviados, o lo que es lo mismo

$$\%Perd_{DL} = 100\left(1 - \frac{Paqs_{rec_{cli}}}{Paqs_{env_{tot}}}\right) \tag{3.2}$$

F. RTT

Es el tiempo que tarda un paquete en ir y venir entre dos equipos de red. Es un indicador del retardo que impone la red a los paquetes que viajan por ella. Este parámetro es importante debido a que diferentes aplicaciones tienen distintos requerimientos de retardos en la red.

G. Jitter

Es una medida de la variación del RTT. Se calcula como el promedio de la variación entre sucesivos registros de este parámetro. Es importante debido a que el mismo es relevante a la hora de brindar cierto tipo de servicios en tiempo real, como ser la voz. Si uno transmite un archivo de audio online puede admitir un retardo considerable en la transmisión, sin embargo si los paquetes llegan con retardo variable la calidad del servicio se ve claramente afectada. El Jitter es el indicador que cuantifica este fenómeno.

3.1.3. Descripción de los problemas presentados en el acceso

Tal como se mencionó anteriormente, cuando uno trata de estimar parámetros de performance sobre servicios orientados al transporte de datos IP sobre los protocolos GPRS-EDGE surgen problemas inherentes a las características del medio en que se encuentra trabajando.

La diferencia más notable que surge en comparación con las redes de acceso fijas tradicionales (ej. ADSL) es que, en estas últimas uno siempre tiene disponible el canal para realizar la transferencia de información cuando lo considere pertinente, mientras que en una red de acceso basada en la técnica de acceso múltiple por división de tiempo (sistemas TDMA) uno no dispone contínuamente del canal sino que transmite cada intervalos determinados de tiempo.

Además de este fenómeno intrínseco del medio de acceso se suma el hecho que dichos recursos deben ser compartidos con los diferentes usuarios que quieren acceder a servicios de datos, por lo que el fenómeno de carecer de la continuidad para disponer del canal cuando se necesite se ve incrementada por la prescencia de "competidores" por el canal que existan en la celda.

Otro de los problemas que trae aparejado la presencia de distintos usuarios para poder acceder a los recursos disponibles es que debido a la escasez de recursos en relación a la cantidad potencial de usuarios en una celda en un momento dado, el sobredimensionamiento existente en todo sistema de telecomunicaciones es mucho mayor en este tipo de sistemas. La probabilidad de querer acceder a los recursos en algún instante y que no se pueda por no disponer de la cantidad suficiente de los mismos no es tan despreciable como en otros medios de acceso típicamente utilizados por los usuarios de este tipo de servicios.

Por último, un problema no menor presentado típicamente en todos los sistemas de acceso al medio inalámbricos es la pérdida de información en el mismo. A continuación se comentan brevemente los problemas típicos causantes de la pérdida de información en el acceso.

Pérdida de camino

La pérdida de camino es la atenuación que impone el medio de acceso intrínsecamente. Para modelar dichas pérdidas se toma como modelo la atenuación en espacio libre, es decir que la atenuación aumenta con el cuadrado del producto de la frecuencia de trabajo (F_0) y la distancia entre la radiobase y el móvil (R_0).

El efecto obvio que tiene dicho factor sobre la performance del móvil es la pérdida de potencia y por lo tanto es uno de los principales factores causantes de transmisiones de información a velocidades bajas. Dicho fenómeno es más perceptible en zonas rurales donde la distancia entre radiobases es más considerable que en entornos urbanos.

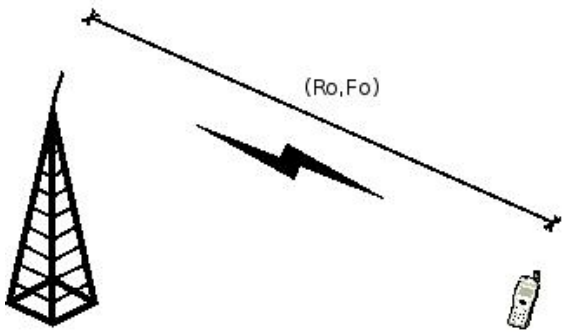


Figura 3.2: Esquema de radioenlace típico entre una radiobase y un móvil celular a distancia R_0 y frecuencia F_0.

Fast fading o fading de Rayleigh

Dicho fenómeno se aprecia principalmente en entornos urbanos y es debido a que, como es imaginable, el móvil no recibe señal solamente de un trayecto sino que el mismo se nutre de una señal compuesta por la adición de varias de éstas, las cuales pueden arribar al móvil con distinta fase y amplitud. Dependiendo de la fase con la que lleguen las diferentes señales la contribución de las mismas puede ser constructiva o desctructiva.

El *fast fading* es un fenómeno que produce cambios en la potencia de recepción ante relativamente poca movilidad del cliente. A modo de ejemplo, trabajando a un a frecuencia de 900 MHz recorriendo sólamente 30 cm se puede llegar a cambiar de fase de 0 a 2π.

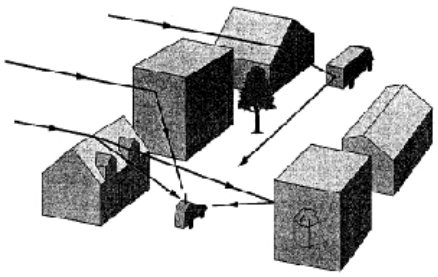


Figura 3.3: Situación típica de prescencia de fast fading en un entorno urbano.

Las señales componentes de la señal total tienen una distribución uniforme $U\ [0, 2\pi]$ (distribución de Rayleigh, de ahí su nombre). Este tipo de

distribución se asocia cuando no hay línea de vista entre la antena y el móvil.

Slow fading

El *slow fading* es un fenómeno producido debido a la presencia de grandes obstáculos, como lo son cerros, construcciones de gran porte, etc. El efecto producido es el de una variación del valor medio del *fast fading*. Tiene asociando una distribución logartítmica.

Dispersión en el tiempo

El fenómeno de dispersión en el tiempo, al igual que el *fast fading*, es debido a los caminos múltiples. Es producido principalmente por obstáculos lejanos, teniendo como principal característica la introducción de ISI[2]. Es un fenómeno preponderante en áreas rurales.

Time alignment

Ocurre cuando la información que envía el móvil llega a la radiobase fuera del *time slot* que le corresponde. Dicho fenómeno es el causante de la introducción de interferencia con el móvil que transmite en su *time slot* correspondiente. Este efecto ocurre preponderantemente cuando el móvil se encuentra lejos de la radio base.

El nivel de la señal recibida cambia con los problemas descritos anteriormente según se puede apreciar en la figura 3.4.

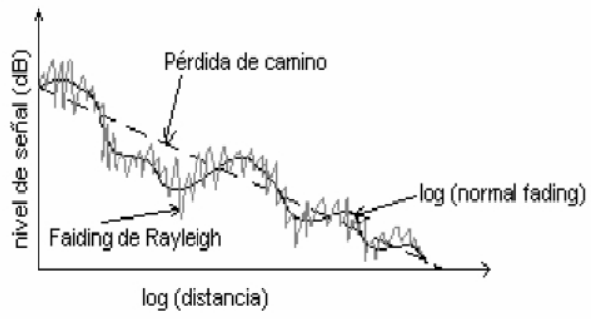


Figura 3.4: Nivel de la señal en el móvil vs la distancia entre la radio base y el móvil.

[2]Interferencia Inter - Simbólica.

3.2. Algoritmo Desarrollado

Descripción general

Para poder comprender mejor el funcionamiento del algoritmo desarrollado es conveniente visualizar al mismo como un todo compuesto por diferentes módulos. Cada módulo se comunica con el siguiente siguiendo un orden cronológico en el procesamiento de los datos lo que conlleva al resultado final. Dicha clasificación en módulos se puede ver como sigue.

1. Realización de la prueba física
2. Clasificación de las muestras obtenidas en intervalos de actividad
3. Procesamiento para cada intervalo de actividad
   A. Detección de las potenciales capacidades ocurridas
   B. Procesamiento de los datos para cada una de las capacidades detectadas
4. Cálculo de los indicadores de performance

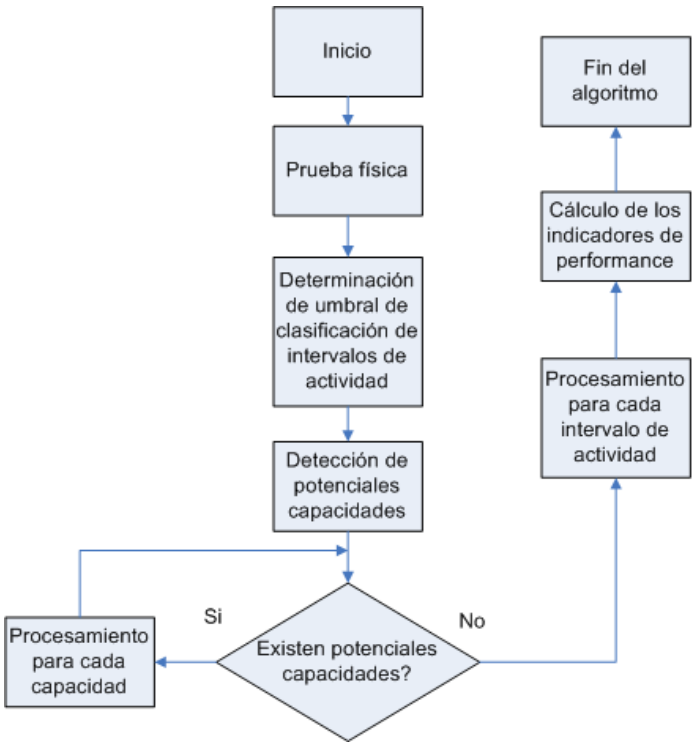


Figura 3.5: Diagrama de flujo del algoritmo.

A continuación se detalla el funcionamiento de cada uno de dichos módulos.

3.2.1. Realización de la prueba física

Este módulo consta de todo lo relacionado a la realización del experimento en sí. Las etapas para la determinación del experimento físico se pueden descomponer como sigue a continuación.

A. Elección de la forma de comunicación entre el cliente y el servidor

B. Elección de la prueba óptima

- tamaño del paquete de prueba
- cantidad de trenes de paquetes
- cantidad de paquetes por tren
- tiempo entre paquetes del mismo tren
- tiempo entre trenes de paquetes

A. Eleción de la forma de comunicación entre el cliente y el servidor

El funcionamiento de la prueba física se puede esquematizar como sigue.

1. Armado del paquete de prueba en el servidor
   - Identificación del paquete mediante un número de secuencia
   - Marcado del paquete mediante el tiempo de salida en el servidor
   - Introducción de bits de relleno para completar el tamaño del paquete
2. Envío del paquete de prueba hacia el cliente
3. Llegada del paquete al cliente
   - Marcado del paquete mediante el tiempo de llegada al cliente
4. Envío de la información hacia el servidor, se puede realizar de dos maneras
   a) Rebote de cada paquete por separado
   b) Envio de la información de todos los paquetes arribados al cliente en un solo paquete al finalizar la prueba mediante un paquete HTTP.
5. En el caso en que se rebota cada paquete por separado se puede marcar el mismo en el servidor mediante el tiempo de llegada a éste

A la hora de determinar cúal sería la modalidad de la prueba física a realizar en el algoritmo se tuvo especial cuidado en las etapas de armado del paquete de prueba y de la manera en que se envía la información desde el cliente al servidor.

En la etapa del armado del paquete de prueba surge la disyuntiva de si los mismos tienen que ser UDP o TCP. Por un lado se tiene que si se utilizan paquetes de prueba TCP, debido a las características intrínsecas del protocolo (establecimiento de conexión, control de flujo y congestión), es el mismo quien controla la manera en que se lleva a cabo la conexión, adaptándose a las características del enlace. Por otro lado al utilizar el protocolo UDP esto no sucede ya que el mismo es no orientado a conexión y por lo tanto los paquetes son enviados de forma independiente. Por estas razones es que el protocolo utilizado para el armado de los paquetes de prueba es UDP.

Con el fin de enviar la información desde el cliente hacia el servidor existen dos alternativas. Por un lado efectuar el rebote de cada paquete individualmente y por otro efectuar el envío de un solo paquete al final de la prueba conteniendo la información pertinente acerca de todos los paquetes arribados al cliente.

A continuación se verán cuales son las características, ventajas y desventajas de cada una de las alternativas planteados.

Rebote de cada paquete por separado

La técnica del rebote de cada paquete por separado se puede describir como sigue. Al arribar un paquete al cliente proveniente desde el servidor se obtiene la información útil del mismo (número de secuencia y tiempo de salida del servidor) y se arma un paquete UDP con dicha información, agregándole a la misma el tiempo de llegada del paquete al cliente. Seguidamente se rellena el paquete con caracteres aleatorios hasta completar un tamaño de paquete dado. Eventualmente al llegar cada paquete al servidor se puede volver a marcar el mismo con el tiempo de llegada para efectuar ciertos cálculos.

Esta técnica presenta como ***ventaja*** que, tomando como hipótesis válida que la capacidad de bajada del cliente siempre es mayor a la capacidad de subida del mismo, se puede asumir que el envío de paquetes desde el cliente al servidor se va a realizar siempre de manera contínua a una tasa mayor o igual que la capacidad de subida. Esto trae aparejado que, haciendo un marcaje en el paquete a la llegada al servidor, se pueden procesar los datos

de la prueba tanto para el downstream como para el upstream, por lo que con una única prueba se puede determinar la performance del enlace en ambos sentidos.

La ***desventaja*** presentada por esta técnica es que existen dos grandes tramos donde se puede perder información de la prueba, en el downlink y en el uplink. Si las pérdidas presentadas en el uplink son considerables, se puede llegar a la situación de que regresen al servidor una cantidad mucho menor de paquetes que la que partieron del mismo, provocando la invalidez de la prueba.

Envío de toda la información mediante un solo paquete http

La técnica del envío de toda la información de los paquetes arribados al cliente mediante un solo paquete HTTP consiste en que, al arribar cada paquete al cliente, se extrae del mismo la información útil (número de secuencia y tiempo de salida del servidor), almacenándose conjuntamente con la misma información de los paquetes previos. Una vez conlcuído el arribo de los paquetes al cliente, se construye un paquete HTTP con toda la información, la cual es enviada posteriormente al servidor. La razón por la que el envío se realiza mediante un paquete HTTP y no UDP radica en que si el envío se efectuase mediante un paquete de este último tipo, ante cualquier error acontecido en el upstream se perdería toda la información de la prueba, lo cual traería aparejado la invalidez de la misma. Por otro lado, mediante el envío de un paquete HTTP con toda la información, se puede aprovechar la confiabilidad inherente del protocolo para asegurar que en caso de que se pierda el paquete en el uplink se efectuarán retransmisiones del mismo hasta que finalmente arribe satisfactoriamente al servidor.

Esta técnica presenta como ***ventaja***, en comparación con la anteriormente expuesta que, al haber solamente un solo sentido con probabilidad de pérdida de paquetes (el downlink), se reduce considerablemente la cantidad de paquetes perdidos en la prueba. Gracias a esto en el servidor se cuenta con una cantidad más significativa de datos, lo que provoca un mejor relevamiento del comportamiento del canal, produciendo un mejor funcionamiento del algoritmo.

Por otro lado presenta como ***desventaja*** que, al contar solamente con los datos necesarios para correr el algoritmo para el downstream, se obtiene el análisis de la performance del enlace únicamente en dicho sentido. Para poder relevar el canal en el uplink es necesario entonces generar otra prueba donde el origen de los paquetes sea en el cliente.

Una vez planteadas las alternativas mencionadas anteriormete se procedió con su implementación. Mediante la realización de una gran cantidad de pruebas se comprobó que, al implementar el rebote de paquetes, las pérdidas sufridas en el uplink eran de un orden bastante considerable, por lo que la información recolectada finalmente en el servidor era sensiblemente menor a la enviada inicialmente. Dicho efecto provoca que no se tenga un relevamiento cabal del comportamiento del canal, lo que trae aparejado que los cálculos efectuados por el algoritmo no sean tan precisos como sería deseable.

Por otro lado, teniendo en cuenta que para la mayor parte de los servicios de datos que se brindan sobre telefonía celular (WAP, WEB, servicios online, etc) la performance de los mismos se ve significativamente mas afectada por la comunicación en el downlink que en el uplink. Se consideró entoncesque obtener la información del comportamiento del canal en el uplink, a costas de la pérdida de información en todo el canal, no era tan relevante en este punto, ya que lo que se quería comprobar realmente era la eficacia del algoritmo desarrollado y para eso es deseable contar con la mayor cantidad de datos disponibles.

Sin embargo, algunos de los resultados a los que se pretendía llegar con la implementación del algoritmo requieren que los paquetes sean rebotados en el cliente para luego, una vez recibidos por el servidor colocar una marca de tiempo y realizar medidas de cuanto tiempo le llevó al paquete desde que salió del servidor hasta que llegó nuevamente. Este es el caso del cálculo del RTT y el Jitter.

Dado que es necesario rebotar paquetes en el cliente para poder hallar algunos de los parámetros propuestos y que es deseable que la información de los paquetes que llegan al cliente se envíen en un paquete HTTP al final de la prueba para no perder información en el uplink, se tomo como primer medida implementar una solución híbrida, donde en el cliente se acumulaba información de todos los paquetes que se recibían durante la prueba para después enviarla toda junta mediante un paquete HTTP. Asímismo cada paquete era rebotado para poder contar con la información requerida para hacer algunos cálculos.

Luego de realizar esta implementación surgió otro problema. Dado que los paquetes salen del servidor a una tasa mayor a la capacidad del canal, éste está generalmente saturado durante la realización de la prueba. Es por esto que los tiempos que le lleva al paquete en ir del servidor al cliente está más afectado por la saturación del canal que por la caracteristica intrínceca del mismo. Lo mismo sucede con las pérdidas de paquetes.

Fue necesario entonces implementar la realización del experimento en dos etapas. Una primera etapa en la cual se realiza una prueba poco invasiva para el canal, en la cual los paquetes son rebotados en el cliente. A partir de esta prueba se calculan los valores del RTT, el Jitter y las pérdidas en el canal. Por otro lado se realiza una segunda etapa donde los paquetes no son rebotados sino que se envía toda la información de los paquetes al final de la prueba. Con la información de estos paquetes se aplica el procedimiento que se detalla en las próximas secciones para calcular el resto de los parámetros propuestos.

B. Elección de la prueba óptima

Una vez solucionada la forma de transmisión de la información entre el cliente y el servidor surge de la disyuntiva de cuáles son las características que tiene que tener la prueba tipo que saque el mejor provecho posible al algoritmo desarrollado. Los parámetros modificables para la realización de la prueba física son los que se listan a continuación.

- Cantidad de paquetes por tren
- Tiempo entre la partida de paquetes del mismo tren
- Cantidad de trenes de paquetes
- Tiempo entre trenes de paquetes
- Tamaño del paquete de prueba

La primer interrogante que uno tiene que resolver cuando se le plantea una situación de este tipo es la de conocer con certeza qué información se desea obtener mediante la realización de la prueba.

Para determinar la característica de una prueba tipo, se tienen que tener en cuenta las limitantes que los objetivos que se desean obtener de la realización de la prueba imponen sobre la misma (ver sección 3.1.2).

Dado que uno de los datos mas interesantes a determinar por el algoritmo es la determinación de la forma en que el sistema le asigna el canal al cliente (mediante la identificación de los tiempos de actividad e inactividad en el canal), es necesario poseer un relevamiento en el tiempo del comportamiento del canal de la manera más contínua posible. La manera mas lógica de lograr dicho objetivo es la de **enviar de manera contínua paquetes de prueba** de forma tal de que el servidor siempre tenga información para ser enviada hacia el cliente. Por otro lado también es deseable tener el mayor relevamiento posible dentro de cada tiempo de actividad, para lo que

es necesario contar con la mayor cantidad de mediciones efectuadas durante dichos intervalos de tiempo. Este objetivo se cumple enviando **paquetes del menor tamaño posible**.

Sin embargo, todo problema de ingeniería cuenta con ciertas disyuntivas y este no es la excepción. Para poder determinar con más precisión el tamaño de los paquetes de prueba es necesario efectuar un estudio de las incertidumbres que se obtienen en las mediciones, tema que se aborda a continuación.

Estudio de las incertidumbres en las mediciones

Sea ϖ el tamaño en `bytes` de un paquete de prueba y Δt la diferencia en `ms` entre el arribo de dos paquetes consecutivos al cliente. Por definición, el cálculo del ancho de banda en `kbps` viene dado por

$$Bw = \frac{8 \cdot \varpi}{\Delta t} \tag{3.3}$$

Por lo tanto, la expresión de la incertidumbre del ancho de banda viene dada por

$$\delta Bw = \frac{\partial Bw}{\partial \Delta t} \cdot \delta \Delta t + \frac{\partial Bw}{\partial \varpi} \cdot \delta \varpi \tag{3.4}$$

Dado que la incertidumbre asociada con el tamaño del paquete ($\delta\varpi$) es nula, se llega a que la expresión de la incertidumbre para el cálculo del ancho de banda viene dada por

$$\delta Bw = \frac{8 \cdot \varpi}{\Delta t^2} \cdot \delta \Delta t \tag{3.5}$$

Debido a que el lenguaje de programación utilizado (JAVA) impone una resolución de tiempo de 1 `ms`, la incertidumbre asociada a cada medición de tiempo es de $\pm$0,5 `ms`, por lo que la incertidumbre asociada a la diferencia de tiempos entre paquetes consecutivos, $\delta\Delta t$, es de $\pm$1 `ms`. Por lo tanto, sustituyendo en 3.5 se tiene que

$$\delta Bw = \frac{8 \cdot \varpi}{\Delta t^2} \tag{3.6}$$

De lo anterior resulta que la incertidumbre en la medición del ancho de banda aumenta linealmente con el tamaño del paquete de prueba pero a su vez disminuye cuadráticamente con el tiempo entre arribos de paquetes consecutivos (el cual, se relaciona de forma inversa con el tamaño del paquete de prueba), por lo que a priori no es trivial realizar una afirmación acerca del tamaño de paquete que minimiza la incertidumbre en la medición. Desarrollando aún más la expresión dada en la ecuación 3.6 podemos llegar a

la fórmula de la incertidumbre relativa en función del ancho de banda y el tamaño de paquete.

$$\frac{\delta Bw}{Bw} = \frac{Bw}{8 \cdot \varpi} \tag{3.7}$$

Para resolver la incógnita planteada se realiza el estudio para el peor de los casos, el cual se da al registrarse la medición del mayor ancho de banda posible. Suponiendo que se tienen cuatro *time slots* disponibles para el downlink y que se está utilizando una codificación EDGE del tipo MCS-9 (ver tabla 3.3) se tiene que el máximo ancho de banda teórico alcanzado en la bajada de datos es de 240 `kbps`. Para dicho ancho de banda, la variación de la incertidumbre relativa en el cálculo del ancho de banda con el tamaño del paquete se puede apreciar en la figura 3.6.

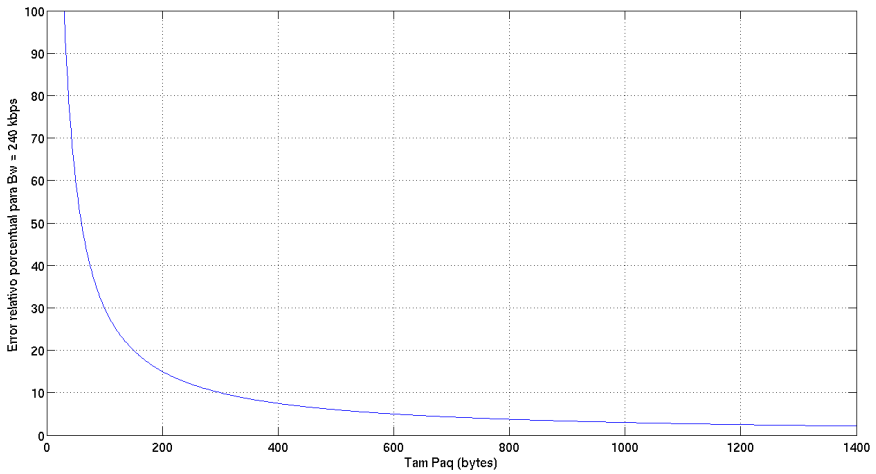


Figura 3.6: Variación de la incertidumbre relativa en el cálculo del ancho de banda con el tamaño de paquete de prueba utilizado cuando el $Bw = 240$.

Tal como se puede apreciar en la figura 3.6, al disminuir la incertidumbre relativa de forma inversa con el tamaño de paquete, **lo más conveniente es elegir un paquete de tamaño lo más grande posible**.

Esta idea se contrapone con la idea de elegir un paquete de tamaño lo más pequeño posible con el fin de poder obtener un relevamiento lo mas grande posible del canal. Por estas dos razones es que es necesario encontrar un equilibrio para poder tratar de satisfacer ambos criterios en la mayor medida de lo posible.

Tal como se puede observar en la figura 3.7, si se quiere tener un error relativo máximo en el peor de los casos del 20 % (caso en que el ancho de banda registrado es de 240 `kbps`) se tiene que trabajar con un tamaño de

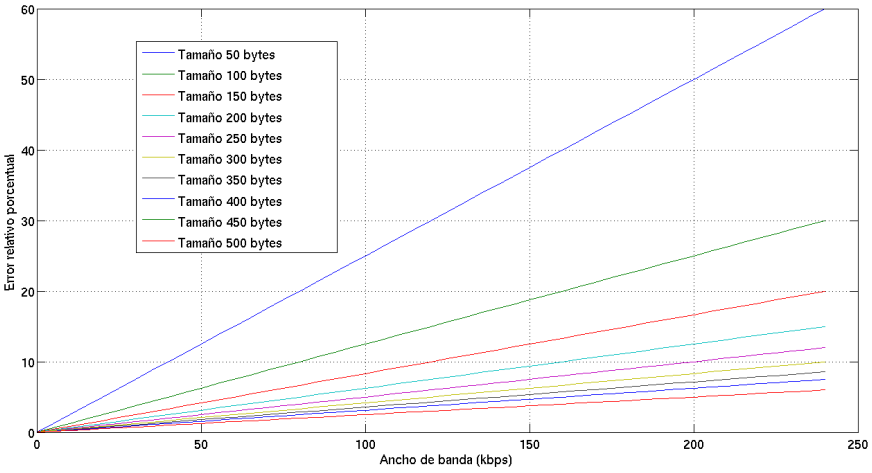


Figura 3.7: Incertidumbre relativa vs ancho de banda para paquetes de prueba de distintos tamaños.

paquete de 150 `bytes`.

Por otro lado, mediante resultados empíricos se obtuvo que, si se desea tener una cantidad de paquetes medianamente aceptable durante los períodos de actividad para servir de entrada al algoritmo, el tamaño del paquete de prueba no debe superar los 300 `bytes` (en esta situación el máximo error relativo es del 10 %, tal como se aprecia en la figura 3.7).

Por las razones expuestas anteriormente se **trabajó con tamaños de paquete en el intervalo de 150 a 300** `bytes`.

Una vez elegido el tamaño de los paquetes de prueba, es deseable que el tiempo entre los mismos sea tal que la tasa de envío supere la capacidad máxima del canal. Dado que la capacidad máxima del canal es 240 `kbps` para el caso que el sistema asigne cuatro time slots con una codificación EDGE del tipo MCS-9, entonces:

$$240kbps < \frac{8 \cdot \varpi}{\Delta t} \tag{3.8}$$

donde ϖ es el tamaño de los paquetes de prueba en *Bytes* y Δt el tiempo entre paquetes en ms. De acuerdo con los tamaños de paquetes elegidos se obtiene para el peor de los casos que

$$\Delta t < 5ms. \tag{3.9}$$

La elección de la cantidad de paquetes a enviar surge de realizar un balance entre mandar la suficiente cantidad de paquetes como para tener un

Parámetro de la prueba	Valor
Cantidad de paquetes por tren	$[1000, 1500]$
Tiempo entre paquetes del mismo tren (`ms`)	$[1, 5]$
Cantidad de trenes de paquetes	1
Tiempo entre trenes de paquetes	-
Tamaño del paquete de prueba (`bytes`)	$[150, 300]$

Tabla 3.4: Parámetros típicos utilizados en las pruebas del algoritmo.

relevamiento significativo del canal y mandar la mínima cantidad de paquetes debido a que el método en sí, al ser un método de estimación de performance del tipo activo, es bastante invasivo para la red, ya que busca saturar el canal durante la duración de la prueba. De dicho balance surgió que la cantidad de paquetes de la prueba deben ser entre 1000 y 1500.

Resta sólamente determinar la cantidad de trenes de pruebas y el tiempo entre sucesivos trenes. Por lo general algunas pruebas se realizan en trenes de paquetes para descongestionar el canal, ya que al mandar a una tasa mayor a la capacidad del canal, en algún momento las colas de los equipos de ruteo se llenan y comienzan a descartar paquetes. Sin embargo en este caso es deseable, por criterio de diseño, mantener siempre a lo largo de toda la prueba paquetes para enviar desde el servidor al móvil. Es por esto que se eligió el envío de un solo tren de paquetes, tratando de enviar los mismos a una tasa apenas mayor a la maxima capacidad del canal.

De acuerdo al análisis realizado anteriormente se obtiene que las características de una prueba tipo óptima serían las especificadas en la tabla 3.4.

3.2.2. Clasificación de los datos obtenidos en intervalos de actividad

El objetivo de este módulo es el de clasificar los datos obtenidos de la prueba física dentro de los intervalos de actividad correspondientes. Se entiende como momento de actividad del sistema el de un intervalo de tiempo tal que el cliente posee el canal para transmitir datos.

Para poder clasificar los datos obtenidos de la prueba física dentro de dichos intervalos es necesario hallar un umbral capaz de discriminar correctamente entre puntos pertenecientes a cada grupo, por lo tanto en éste punto surge la disyuntiva acerca de qué criterio seguir para la elección del umbral de decisión. Para ésto se evaluaron dos alternativas bien distintas.

- Umbral fijo independiente de la prueba en cuestión
- Umbral dependiente de la prueba realizada

A. Umbral fijo

El método de elección de un umbral fijo consta en elegir un valor para dicho umbral en función de resultados empíricos y fundamentación teórica. Dicha metodología tiene como ***ventaja*** la de que una vez determinado el valor del umbral no es necesario hallar un valor del mismo para cada prueba realizada, por lo que se ahorra coste computacional y tiempo de procesamiento de la prueba. Esto se traduce en que el cliente conoce más rápidamente el resultado del experimento realizado. La ***desventaja*** que presenta este método es que carece de robustez y la flexibilidad que posee un algorimo que determina el umbral de clasificación basandose en las características de cada prueba en particular.

Por estas razones es que para el diseño del algoritmo *se decidió hallar el umbral para cada prueba* mediante la aplicación de técnicas de reconocimiento de patrones.

B. Umbral particular

La metodología de hallar un umbral particular para cada prueba consiste en hacer una elección del valor del umbral de discriminación en función de las características propias de la prueba realizada. El área que se encarga de resolver esta clase de problemas es la del Reconocimiento de Patrones, mediante la aplicación de técnicas no supervisadas de agrupamiento (métodos automáticos de detección de características, sin interacción con una persona que "marque" ciertas características manualmente).

El algorimto elegido para resolver el problema planteado es el `K-mean` o `algoritmo de las medias móviles` [11], donde `k` es el némero de grupos a encontrar, el cual se supone conocido previamente. En el caso de estudio $k = 2$ (grupo de puntos pertenecientes a un período de actividad cualquiera y grupo de puntos pertenecientes a períodos de inactividad). El diagrama de flujo de dicho algoritmo se muestra en la figura 3.8.

Luego de concluída la ejecución del algoritmo, el umbral seleccionado para efectuar la discriminación pertinente viene dado por

$$Umb = \frac{\overline{g_1} + \overline{g_2}}{2} \tag{3.10}$$

donde $\overline{g_1}$ y $\overline{g_2}$ son las medias de los grupos 1 y 2 respectivamente.

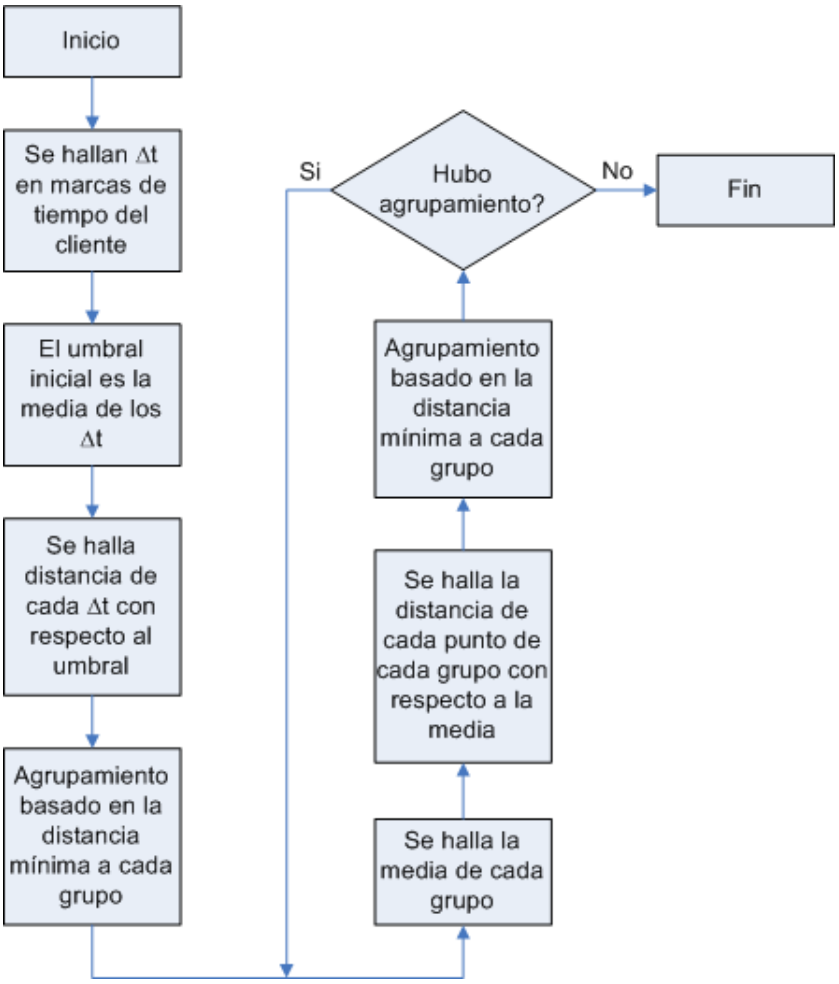


Figura 3.8: Diagrama de flujos del algoritmo de las medias móviles.

La elección del algoritmo de las medias móviles viene dada porque es un algoritmo de bajo coste computacional, de sencilla implementación, efectivo y que, por sobre todas las cosas, una vez implementado los resultados obtenidos mediante la aplicación del mismo fueron muy satisfactorios. A modo de ejemplo se muestra en la figura 3.9 el marcaje de los períodos de actividad realizado en una prueba real mediante la utilización del umbral hallado mediante el mencionado algoritmo.

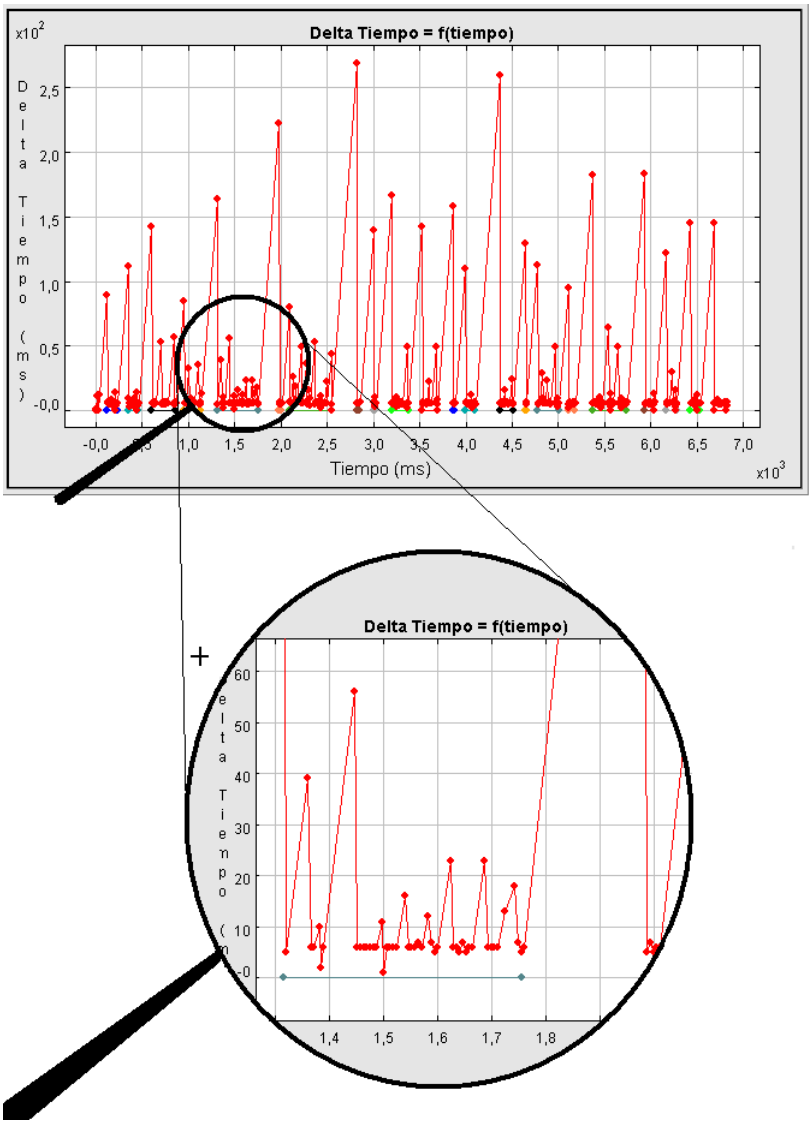


Figura 3.9: Identificación de los períodos de actividad mediante el umbral hallado con el algoritmo de las medias (marcas coloreadas en la base del gráfico).

3.2.3. Procesamiento para cada intervalo de actividad

Luego de detectados los intervalos de actividad de la prueba mediante la aplicación del umbral hallado mediante el algoritmo de las medias móviles, el algoritmo desarrollado procede al estudio de los intervalos de actividad, repitiendo el procesamiento para cada intervalo por igual. El diagrama de flujo del módulo de procesamiento de intervalos se muestra en la figura 3.10.

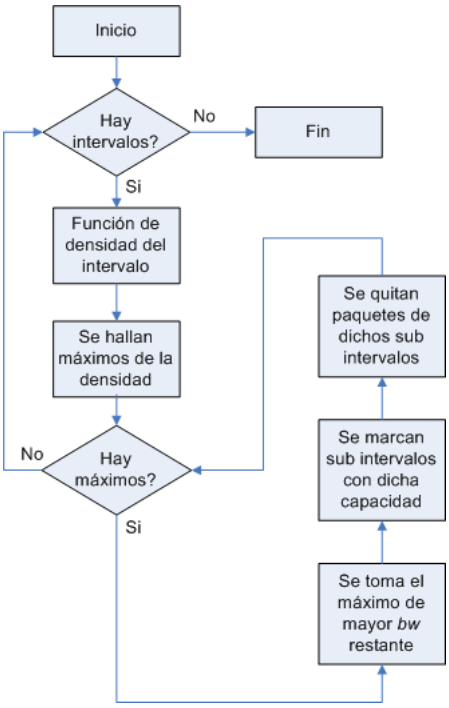


Figura 3.10: Diagrama de flujo del módulo de procesamiento de intervalos.

A. Detección de las potenciales capacidades ocurridas

La primera etapa de procesamiento que sufre un intervalo detectado es la de la detección de las potenciales capacidades ocurridas. Para ésto se halla la función densidad de las muestras del intervalo y, seguidamente, los máximos relativos de dicha función. Para la estimación de las capacidades ocurridas existen varias alternativas diferentes, las cuales se pueden clasificar según los siguientes criterios

- Estimar potenciales capacidades mediante la realización de un histograma
- Estimadores de densidad basados en núcleos

Estimar potenciales capacidades mediante la realización de un histograma

Para el caso de estudio en particular la estimación de las potenciales capacidades mediante la utilización de un histograma consiste en dividir el espectro de anchos de banda posibles (por ejemplo de 0 a 240 `kbps`) en una cantidad finita de intervalos. El ancho de dicho intervalo es lo que se denomina como la resolución del histograma. Para la determinación de la re-

solución del histograma existen varias recomendaciones comunes como, por ejemplo, la *regla de Sturgess* [18]. Seguidamente se clasifica cada ancho de banda registrado dentro de su intervalo correspondiente. Por ejemplo, si la resolución del histograma es de 10 `kbps` y se registra un ancho de banda de 154 `kbps` se incrementa en uno la altura correspondiente a la barra cuya base se asocia al intervalo correspondiete a 150 - 160 `kbps`. De esta forma la superficie de cada barra es proporcional a la cantidad de anchos de banda registrados dentro del rango de dicha barra.

Una vez conformado el histograma se procede a hallar los máximos del mismo. Dado que el histograma es una función discreta, para determinar dichos puntos se pueden registrar los cambios en las áreas de las barras. Cuando se registra un cambio positivo y luego uno negativo se dice que dentro del intervalo intermedio del cambio de áreas ocurre el máximo relativo. Aplicando dicho procedimiento para todos los intervalos registrados dentro del histograma se detectan las potenciales capacidades del intervalo de actividad en cuestión. Como se vió en el capítulo 2, un algoritmo muy popular que utiliza dicho procedimiento es *Pathrate* [16].

Este método para determinar las potenciales capacidades tiene como ***ventaja*** el ser una metodología que insume bajo coste computacional y, por lo tanto, se obtiene una mayor rapidez a la hora de procesar los datos. Por otro lado presenta como ***desventaja*** el hecho de que al no tener una metodología concisa y práctica para determinar con certeza la resolución del histograma, al llegar al momento de la determinación del mismo se cae dentro de una zona de cierta inseguridad al respecto. Por otro lado, el resultado devuelto no es un número en sí mismo sino que es un intervalo en el cual se registró una cantidad máxima de ocurrencias relativas.

Por las ***desventajas*** presentadas anteriormente del método de estimación de potenciales capacidades a través de la realización de un histograma es que **se descartó de plano la idea de implementarlo como módulo en el algoritmo desarrollado**.

Estimadores de densidad basados en núcleos

La idea de hallar la función densidad de las muestras consiste en, dada una sucesión finita de n muestras $X_1, \ldots, X_n$, estimar la función densidad $f(t)$ en un punto t por medio del estimador[3].

$$\hat{f(t)} = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{t - X_i}{h}), \tag{3.11}$$

[3]Por mas detalles ver [13] y [12]

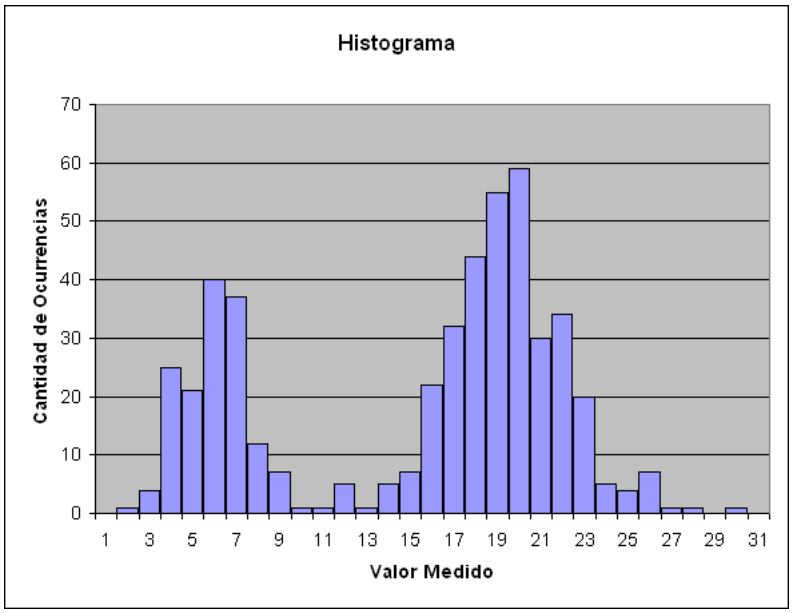


Figura 3.11: Ejemplo de un histograma típico.

donde K es una función llamada núcleo (o kernel) que cumple que $\int_{-\infty}^{\infty} K(t)dt = 1$ y h la ventana del núcleo.

Luego de obtenida la función densidad de las muestras se procede a hallar los máximos registrados en la misma, siendo estos las potenciales capacidades del sistema. A continuación se muestra el método que, dada una función de densidad cualquiera, devuelve las potenciales capacidades registradas.

```
public ArrayList<Double> getPicos(ArrayList<Double> densidad){
   ArrayList<Double> picos = new ArrayList<Double>();
   pendienteAnterior = dens.get(1) - dens.get(0);
   for(int i = 1; i < dens.size() - 1; i++){
      pendiente = densidad.get(i+1) - densidad.get(i);
      if(pendiente*pendienteAnterior <= 0 && (pendienteAnterior > 0 || pendiente <
         pico = (this.factorBWmax*i*this.BWmax)/(this.largoVector);
         picos.add(pico);
      }
      pendienteAnterior = pendiente;
   }
   return picos;
}
```

Algunos de los nucleos más utilizados son los que se listan en la tabla

Núcleo	**Fórmula**
Triangular	$1 - signo(t)tI_{\|t\|<1}$
Epanechnikov	$\frac{3}{4}(1-t^2)I_{\|t\|<1}$
Gaussiano	$\frac{1}{\sqrt{2\pi}}e^{\frac{-t^2}{2}}$
Uniforme	$\frac{1}{2}I_{\|t\|<1}$
Cuadratica	$\frac{15}{16}(1-t^2)^2I_{\|t\|<1}$

Tabla 3.5: Comparación entre los diferentes núcleos estudiados.

3.5

Donde

$$I_{|t|<1} = \begin{cases} 1 & \text{si } |t| < 1 \\ 0 & \text{en caso contrario} \end{cases}$$

En la figura 3.12 se muestran gráficamente los núlceos triangular, gaussiano y de Epanechnikov que son los que se tuvieron en consideración para la implementación del del algoritmo. El motivo por el cual se consideraron estos núcleos y no otros, es por ser los más eficientes y más estudiados de los núcleos listados en la tabla 3.5. En [26] se realiza una comparación entre los diferentes núcleos y se menciona que el de Epanechnikov es el más eficiente en la minimización del error cuadrático integrado esperado (o promedio) asintótico entre la función estimada con el núcleo y la función real.

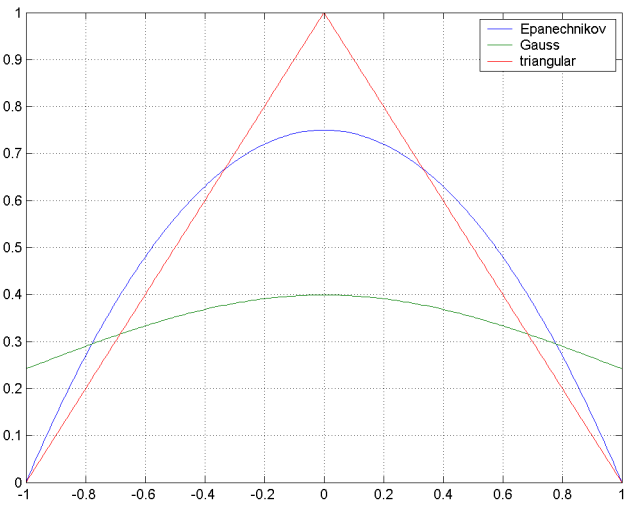


Figura 3.12: Núcleos de convolución estudiados.

Tal como se puede apreciar en la figura 3.12, el núcleo triangular tiene

como propiedad de ser el que decae mas rápido, teniendo en cuenta en menor medida a su entorno en el momento para estimar la función densidad. El que posee una pendiente intermedia es el núcleo de Epanechnikov y finalmente, el de pendiente mas suavizada resulta ser el núcleo gaussiano.

Luego de efectuar la implementación de los tres núcleos mencionados anteriormente se procedió a comprobar los resultados obtenidos a través de las pruebas físicas realizadas con cada uno de ellos, confirmando los conceptos teóricos expuestos anteriormente. En las mismas se observó que el núcleo de Epanechnikov es el que minimiza la cantidad de "falsas capacidades". Esto significa que las capacidades detectadas mediante la aplicación de dicho núcleo son las que tienen mayor probabilidad de ser finalmente capacidades reales dentro del intervalo de actividad estudiado, lo que también conlleva a una menor carga computacional en el módulo posterior (al detectar una menor cantidad de "capacidades falsas" el módulo posterior itera para una menor cantidad de capacidades).

Por las razones mencionadas anteriormente **se decidió que el núcleo de Epanechnikov sea el estimador implementado** en el módulo de procesamiento de capacidades del algoritmo desarrollado.

El parámetro h de la función de densidad determina el ancho de la ventana y nivel de suavizamiento de la función. La elección de éste parámetro es tan o más importante que la elección del núcleo, tal como se analiza en [27]. Cuanto más pequeño es el valor de h más irregular y más parecida a un histograma es la función de densidad. Por otro lado cuanto mayor es el valor de la ventana, más suave y menos definida es la forma de la función hallada. En la búsqueda del mejor tamaño de ventana se trata de minimizar el error absoluto o minimizar el error cuadrático de la estimación respecto a la función real, pero en este caso no se cuenta con la función real, sino con medidas experimentales de los valores a los cuales se quiere aproximar la función. Con base en la minimización de alguno de estos errores, se propone en [29] diversas metodologías que tratan de determinar el tamaño óptimo de ventana. Utilizando la clásica regla del tipo rule-of-thumb, la cual consiste en utilizar una distribución de referencia conocida, como ser la distribución normal, se puede estimar el valor de h_{optimo}, cuyo valor es:

$$h_{optimo} = (\frac{3}{8})^{-1/5} \delta_k \pi^{1/10} s_n n^{-1/5} \tag{3.12}$$

donde s_n represenata la desviación estandar de los puntos medidos

$$s_n = \left(\frac{1}{n} \sum_{i=1}^{n} (X_i - \overline{X})^2 \right)^{1/2} \tag{3.13}$$

y

$$\delta_k = \left(\frac{\int_{-\infty}^{\infty} K^2(t)dt}{(\int_{-\infty}^{\infty} t^2 K(t)dt)^2} \right)^{1/5} \tag{3.14}$$

Para el caso particular del núcleo de Epanechnikov $\delta_k = (15)^{1/5}$. Sustituyendo a δ_k para dicho núcleo en la ecuación 3.12 se obtiene el resultado de h_{optimo}:

$$h_{optimo} = (40)^{1/5} \pi^{1/10} s_n n^{-1/5} \tag{3.15}$$

B. Procesamiento de los datos para cada una de las capacidades detectadas

Luego de hallados los potenciales candidatos a ser capacidades reales se procede a evaluar si dichos valores son efectivamente capacidades reales o no.

El primer paso que se sigue en esta etapa del módulo es la de determinar el criterio para definir si un par de paquetes consecutivos se encuentran "muy separados" en el tiempo o no. Para ello se halla un umbral de discriminación Δt_0 que sea un fiel representante de la idea de la cercanía en el tiempo de dos paquetes dentro del intervalo de actividad en estudio.

Determinación del Δt_0

Para poder determinar un valor adecuado de Δt_0 existen dos metodologías distintas.

- Elegir un Δt_0 fijo, independiente del intervalo a analizar
- Calcular un Δt_0 para cada intervalo en particular

Elegir un Δt_0 fijo, independiente del intervalo a analizar consiste en, a través de resultados empíricos, determinar un tiempo razonable con el cual se sea capaz de decidir si dos paquetes consecutivos se encuentran "muy juntos" o no. Dicho criterio tiene como ***ventaja*** el hecho que una vez determinado dicho valor se ahorran cálculos cada vez que se procesa un intervalo de actividad dado. Por otro lado presenta la ***desventaja*** de ser una metodología poco flexible ante las características particulares de los intervalos de actividad. Por ejemplo, en una prueba con una muy buena relación portadora - interferente es esperable que dentro del intervalo de actividad los paquetes consecutivos arriben a una tasa mayor que cuando la prueba se lleva a cabo con una relación portadora - interferente mala. Por esta razón resulta claramente evidente la impracticidad de realizar la elección de un valor de Δt_0 fijo independiente de la prueba realizada.

El **criterio utilizado** para la determinación del valor de Δt_0 es el de *hallar la media de las diferencias de tiempo entre paquetes consecutivos dentro del intervalo de actividad y multiplicarla por un factor de amplificación dado*. La idea de la elección de dicho valor radica en que si se tiene sólamente en cuenta como umbral de decisión a la media de las diferentcias de tiempo entre paquetes consecutivos se está considerando como válida, en promedio, sólamente a la mitad de los paquetes del intervalo analizado. Además si se admite que dentro de un período de actividad exista presencia de *cross traffic* o que el sistema registre una *medición inválida* por una falla del mismo es probable que la diferencia de tiempo entre dos paquetes sea mayor que la media de la diferencia de tiempo entre paquetes consecutivos y, sin embargo, dichos paquetes pueden llegar a pertenecer al mismo intervalo de capacidad. Tal como se puede apreciar en la figura 3.13 es común que dentro de un intervalo de tiempo en el que los paquetes arriban a tasa constante (y por lo tanto marcan claramente una capacidad ocurrida en dicho intervalo) ocurran ciertas mediciones inválidas (ya sea debido a la presencia de *cross traffic*, defecto del hardware encargado de procesar los paquetes u otro) que, claramente, tienen que ser incluídas dentro del intervalo en que se registra la capacidad preponderante de la totalidad de los paquetes. A continuación

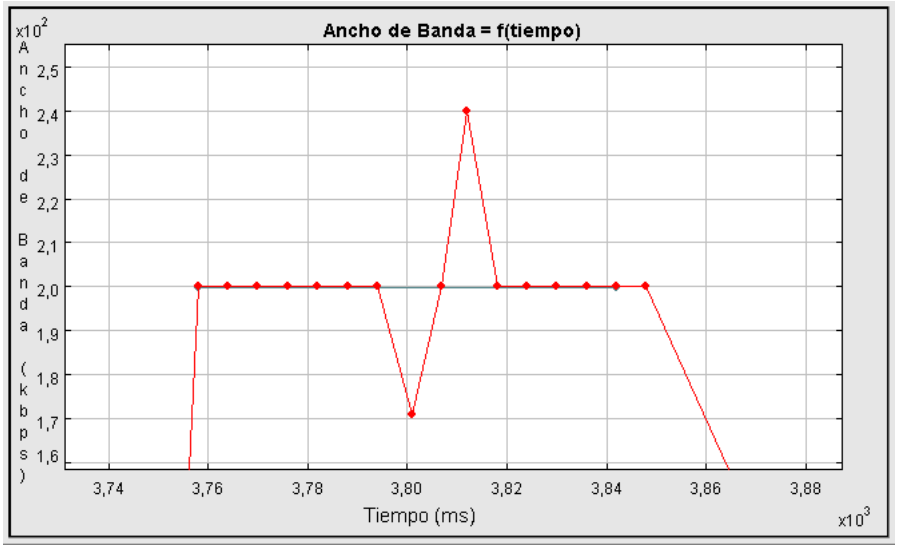


Figura 3.13: Ejemplo real de un caso en que dentro de una ráfaga de paquetes que arriban a tasa constante se registra una medición inválida o alterada por tráfico ajeno al de la prueba.

se muestra el código del algoritmo encargado de hallar Δt_0.

```
private double getDeltaT(ArrayList<Double> paqsDelTON){
   double factor = 3; //Factor de amplificación
   ArrayList<Double> deltasT = new ArrayList()<Double>;
   for(int i = 0; i < paqsDelTON.size() - 1; i++){
      deltasT.add(paqsDelTON.get(i + 1) - paqsDelTON.get(i));
   }
   return promedio(deltasT, factor);
}
```

Luego de la determinación del Δt_0 del intervalo de actividad en análisis se procede a determinar las capacidades ocurridas durante dicho período de tiempo. Para realizar el procedimiento se asumen la siguientes hipótesis.

Hipótesis 1 *Defectos en la medición del sistema no ocurren en la progresión temporal de manera tal que pueda identificarse una determinada capacidad durante un intervalo de tiempo dado debido exclusivamente a dicho efecto.*

Hipótesis 2 *El cuello de botella de la red ocurre en el acceso, por lo que no existe encolamiento en un punto posterior de la red (teniendo en cuenta el downstream).*

Las hipótesis 1 y 2 aseguran que no puede ocurrir una capacidad ficticia mayor a la capacidad de mayor ancho de banda registrado en la prueba. Basándose en dichas hipótesis, se puede realizar la siguiente proposición.

Proposición 1 *El primer candidato a ser capacidad (ordenados de mayor a menor), que cumpla con las condiciones establecidas en el dominio del tiempo para serlo es, con seguridad, capacidad del intervalo de actividad en cuestión.*

Basándose en la proposición efectuada, el algoritmo toma la capacidad candidata de mayor ancho de banda y verifica si se cumplen las condiciones en el tiempo necesarias para afirmar que durante un intervalo de tiempo dado dicho candidato fue efectivamente una capacidad del intervalo de actividad en análisis.

Las condiciones que se tienen que cumplir en el tiempo para poder afirmar que una capacidad candidata es efectivamente una capacidad del sistema son:

- Que la diferencia de tiempos entre paquetes consecutivos que componen el intervalo hallado sea de, como máximo, Δt_0
- Que al menos tres paquetes compongan dicho intervalo temporal

- Un punto puede llegar a pertenecer a un intervalo temporal de la capacidad en estudio si y solo si la misma pertenece al rango de incertidumbre asociada al cálculo de dicha capacidad

La primera de las condiciones ya fue justificada anteriormente. La segunda viene dada simplemente como una condición para establecer la cantidad mínima de paquetes que deben componer un intervalo de capacidades válida. La tercera se tiene en cuenta de manera de considerar que el cálculo de una capacidad puntual tiene una incertidumbre asociada (tal como se explicó en el estudio re incertidumbres realizado en éste capítulo) y la misma debe ser considerada para establecer a qué capacidad pertenece dicho punto.

Una vez recorrido todo el intervalo de actividad en estudio para la capacidad dada, se procede a la eliminación de las muestras pertenecientes a los eventuales intervalos de tiempos donde se detectó la capacidad en cuestión[4]. El proceso se repite mientras queden al menos tres puntos dentro del intervalo de actividad o hasta que no queden mas capacidades por analizar.

Al finalizar con dicho procedimiento se almacenan todas las capacidades válidas en dicho intervalo de tiempo y se repite todo el proceso con el siguiente intervalo de actividad. Se puede observar en la figura 3.14 que algunas muestras del Ancho de Banda instantáneo superan el máximo teórico del enlace estudiado. Se considera que estas muestras corresponden a paquetes que por algún motivo no fueron procesados al momento de su arribo. Esto provoca que el ancho de banda medido entre un paquete que sufre dicho fenómeno y el arrivado anteriormente sea menor al real. Asimismo el ancho de banda registrado entre el paquete procesado a destiempo y el posterior es mayor al real. Igualmente, tal como se explicó en el punto B de la sección 3.2.3, el algoritmo es inmune a dicho fenómeno.

Cuando ya no quedan mas intervalos de actividad por analizar en la prueba se procede con el siguiente módulo.

[4]Notar que en dicho intervalo de tiempos pueden existir puntos que no posean la capacidad en cuestión. También notar que, luego de finalizado el proceso, pueden quedar puntos que no se pudieron asociar a capacidad alguna, considerándose ruido en el intervalo que se analiza.

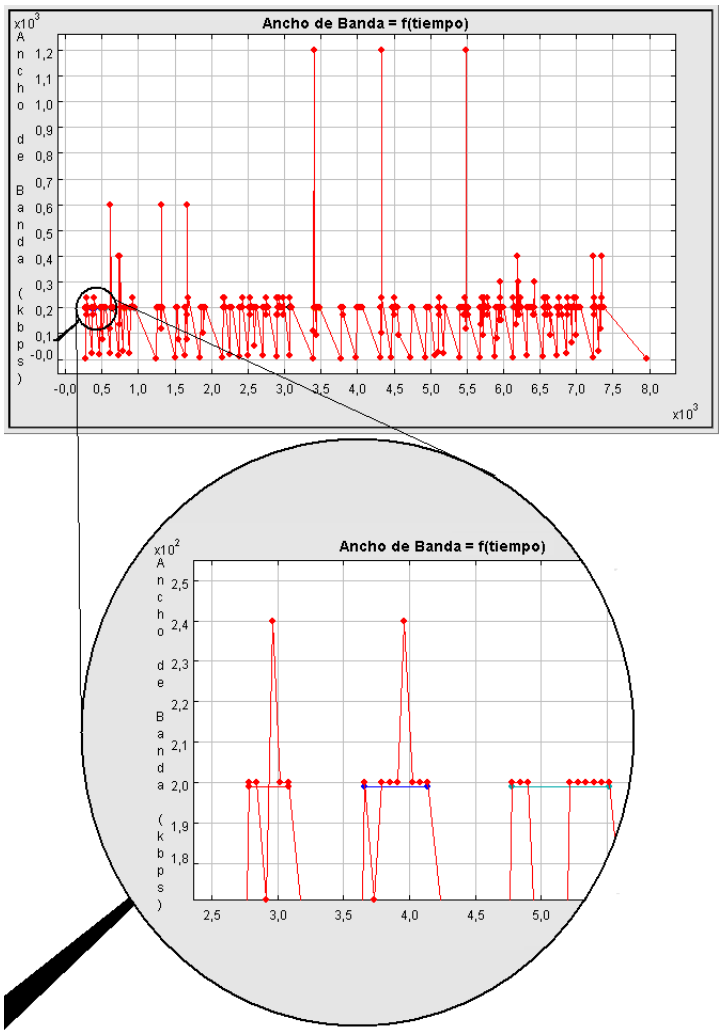


Figura 3.14: Identificación de de las capacidades de cada período de actividad.

3.2.4. Cálculo de los indicadores de performance

Una vez terminado el procesamiento de los datos se procede a calcular los indicadores de performance devueltos por el algoritmo.

Paquetes perdidos en el downlink

Se calcula como la cantidad de paquetes arrivados al cliente sobre la cantidad total de paquetes enviados desde el servidor (en porcentaje).

$$\%Perd_{DL} = 100\left(1 - \frac{Paqs_{rec_{cli}}}{Paqs_{env_{tot}}}\right) \tag{3.16}$$

Capacidad media de la prueba

Se calcula como la sumatoria de cada capacidad hallada durante la prueba, ponderada por el intervalo de tiempo durante la cuál ocurrió, sobre la sumatoria de dichos intervalos de tiempo.

$$\overline{C} = \frac{\sum_i C_i \Delta t_i}{\sum_i \Delta t_i} \tag{3.17}$$

Throughput global de la prueba

Se halla como la cantidad total de datos arrivados al cliente sobre el intervalo de tiempo total transcurrido en la prueba.

Duración media de un intervalo de actividad

Se calcula como el tiempo medio de duración de un intervalo de actividad registrado en la prueba.

Tamaño medio de un intervalo de actividad

Se calcula como la cantidad promedio de bytes de cada intervalo de actividad

Throughput medio de los intervalos de actividad

Se halla como la media de la razón entre el tamaño de un intervalo de actividad y la duración del mismo.

Porcentaje de tiempo en que el cliente se encuentra utilizando el canal

Se calcula como la cantidad total de tiempo en que el sistema estuvo activo sobre la duración total de la prueba.

3.3. Performance del Algoritmo

En la presente sección se analiza el funcionamiento del algoritmo, estudiando su performance al enfrentarse ante diversas situaciones. Con dicho fin se realizaron distintas pruebas en diferentes escenarios. Se muestran a continuación las observaciones más destacadas.

3.3.1. Detección de intervalos de actividad

Una vez llevadas a cabo una gran cantidad de pruebas se observaron diferentes escenarios. Los mismos pueden provocar distintos comportamientos en el funcionamineto del módulo de identificación de intervalos de actividad del algoritmo. Dado que esta etapa es la primera clasificación de las muestras que realiza el algoritmo, en caso de que la misma no funcione como es debido, se compromete seriamente el funcionamiento posterior del resto de los módulos. Por esta razón, es crucial que el mencionado módulo sea lo mas robusto posible ante diferentes patrones de tráfico.

En un principio el algoritmo fue diseñado para poder diferenciar los intervalos de inactividad impuestos por la asignación de recursos de la red de las diferencias de tiempos entre paquetes consecutivos pertenecientes a un mismo intervalo de actividad. Sin embargo, existen factores externos a la prueba, como son el *cross traffic* en el cliente, incertidumbres en las mediciones, fallas en el procesamiento de los paquetes y ruido en general que pueden comprometer la discriminación deseada. Por ejemplo, ante presencia importante de *cross traffic* en el cliente, las diferencias de tiempo entre paquetes de prueba consecutivos pueden llegar a ser considerables, por lo que la discriminación de dicho tiempo con respecto a un período de inactividad introducido por la red se puede dificultar seriamente. Debido a esto es que es deseable que las pruebas se realicen en ausencia de *cross traffic* en el momento de la misma.

Un caso particular en las pruebas es que durante el transcurso de las mismas no existan períodos de inactividad, lo cual sucede en el caso de disponer del enlace en forma exclusiva. En este caso, a la hora de tratar de diferenciar los períodos de inactividad, al no haberlos, se efectúa una detección incorrecta de dichos períodos, identificando grandes diferencias de tiempo entre paquetes consecutivos como períodos de inactividad introducidos por la red. Esto provoca que los falsos intervalos de actividad detectados por el módulo contengan una cantidad muy pequeña de paquetes, lo que conlleva a que las etapas posteriores no puedan identificar las capacidades de la conexión correctamente. En esta situación, lo deseable sería identificar a los paquetes de la prueba como pertenecientes a un solo intervalo de actividad y entregárselos al módulo siguiente sin procesamiento alguno. Para solucionar dicho inconveniente se decidió fijar un umbral mínimo de detección de períodos de inactividad en el sistema. Dado que el tiempo entre arribos de paquetes depende del **tamaño** de los mismos y del **ancho de banda instantáneo** registrado en la prueba, se halló el umbral mínimo del experimento en base a dichos parámetros.

Luego de realizadas una cantidad de pruebas considerables, se encontró que

el ancho de banda instantáneo entre paquetes consecutivos pertenecientes a un mismo período de actividad nunca fue menor a 30 `kbps`, por lo que cualquier ancho de banda instantáneo menor que dicho valor puede asumirse que es provocado por una diferencia entre paquetes consecutivos pertenecienes a diferentes intervalos de actividad. Por otro lado, se determinó experimentalmente que el ancho de banda instantáneo entre dos paquetes consecutivos pertenecientes a distintos períodos de actividad nunca es mayor a 10 `kbps`. Con el propósito de tener un margen de error menor, se propuso que si un par de paquetes consecutivos produce un ancho de banda instantáneo menor a 20 `kbps` es debido a que ambos pertenecen a distintos intervalos de actividad. Siguiendo este razonamiento es que se calcula el menor umbral posible para cada prueba en particular. Siendo ϖ el tamaño en `bits` de los paquetes de prueba se tiene que el umbral mínimo de decisión viene dado por la ecuación 3.18.

$$\Delta t_0 = \frac{\varpi}{20} \tag{3.18}$$

Por lo tanto, el umbral final utilizado para determinar los tiempos de inactividad viene dado por la ecuación 3.19

$$Umb = max\left\{\Delta t_0, Umb_{Kmean}\right\} \tag{3.19}$$

donde Umb_{Kmean} es el umbral hallado por el algoritmo `k-mean` explicado en la sección 3.8 del presente capítulo.

A modo de ejemplo, en las figuras 3.15 y 3.16, se muestran los casos de una prueba típica en la cual existen intervalos de inactividad claramente marcados y una prueba particular en la que no existen intervalos de inactividad y que por lo tanto de no ser por la aplicación de la determinación del umbral mínimo el resultado de la misma hubiese sido inválido.

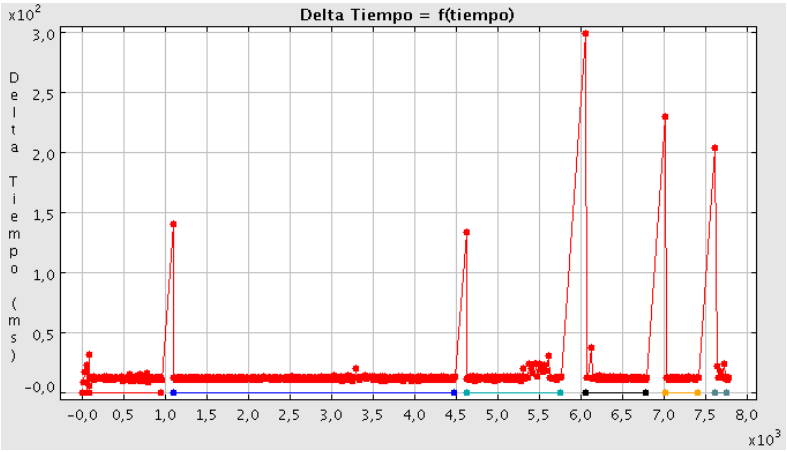


Figura 3.15: Prueba en la cual existen intervalos de inactividad claramente marcados.

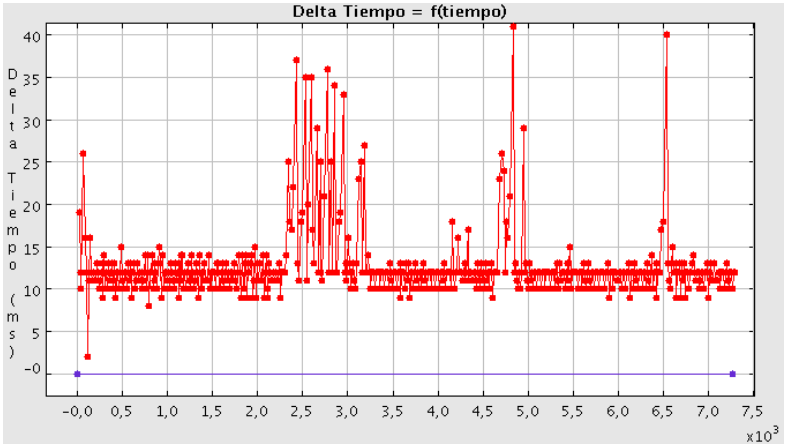


Figura 3.16: Prueba en la cual no existen intervalos de inactividad.

En el caso que no se aplique el criterio del umbral mínimo se puede apreciar en la figura 3.17 como se identificarían una cantidad muy grande de períodos de actividad "falsos", lo que puede provocar que los módulos siguientes de la prueba no funcionen correctamente y que, por lo tanto, el resultado de la prueba sea inválido.

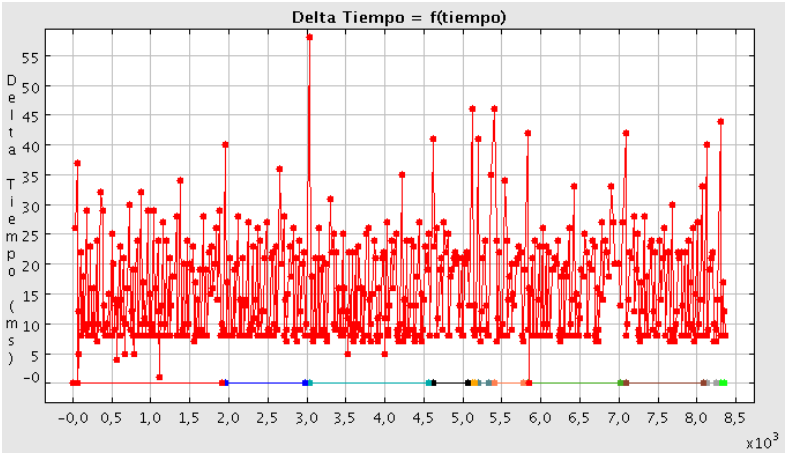


Figura 3.17: Prueba en la cual de no ser por la aplicación del criterio del umbral mínimo se detectan intervalos de actividad de más.

3.3.2. Detección de las capacidades definitivas del sistema

Tal como se explicó en el punto 3.2.3 del presente capítulo, luego de clasificados los períodos de actividad del experimento se procede con la estimación de las potenciales capacidades ocurridas en cada uno, estimando

la función de densidad de las muestras mediante la aplicación del núcleo de Epanechnikov. Seguidamente, se procede a identificar cuales de dichas potenciales capacidades son efectivamente capacidades del sistema.

El propósito de esta sección es el de mostrar la performance del algoritmo al momento de identificar las capacidades definitivas mediante la aplicación de los criterios anteriormente expuestos en el presente capítulo.

En las figuras 3.18, 3.19, 3.20 y 3.21 se muestra la performance del algoritmo en el momento de detectar las capacidades finales del sistema ante diferentes patrones de tráfico.

En la figura 3.18 se muestra una situación típica a la que se enfrenta el algoritmo al tratar de determinar una capacidad final en la prueba. Tal como se puede apreciar en la misma, en el intervalo estudiado una gran cantidad de muestras registran el mismo ancho de banda. Sin embargo, la presencia de ciertas muestras aisladas que se alejan de dicho valor podrían, a priori, provocar que no se identifique el intervalo completo como perteneciente a la capacidad marcada (identificada con color azul en el gráfico). A pesar de esto, debido a las condiciones temporales vistas en el punto B de la sección 3.2.3, se puede ver como el algoritmo identifica que dichos puntos aislados son, con gran probabilidad, provocados debido a ruido en el sistema o a registros de anchos de banda instantáneos en el sistema los cuales no cumplen con las condiciones necesarias como para demarcar claramente una capacidad. Por estas razones es que el algoritmo ante situaciones como estas decide que en todo el intervalo en cuestión la capacidad registrada es la misma.

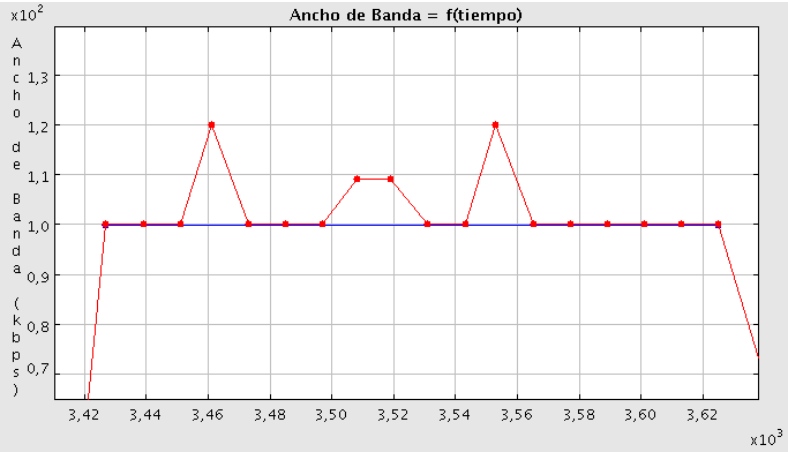


Figura 3.18: Detección de capacidades ante la presencia de poco ruido.

Otra situación similar a la de la figura 3.18 es la que se muestra en la figura 3.19. En este caso, debido a que la presencia de muestras separadas del valor medio es mayor, se podría pensar que el algoritmo no es capaz de determinar claramente una capacidad en el período de tiempo en estudio. Sin embargo, por las razones mencionadas en el caso anterior y, teniendo en cuenta el intervalo de incertidumbre en el cálculo del ancho de banda instantáneo asociado a cada muestra analizado en el punto B de la sección B, se puede ver como el algoritmo identifica a la moda de capacidad preponderante en el intervalo de tiempo como capacidad definitiva en dicho intervalo, siendo el mismo inmune al *cross traffic* registrado en las mediciones.

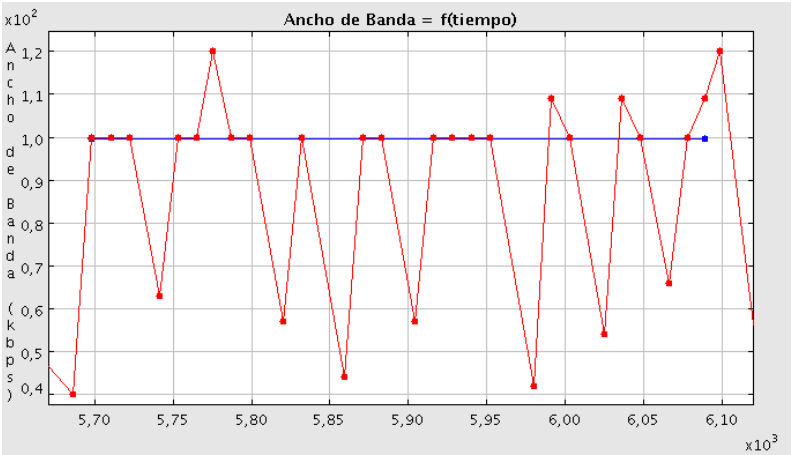


Figura 3.19: Detección de capacidades ante la presencia importante de ruido.

Otro caso particular que se puede dar al realizar una prueba es el presentado en la figura 3.20. En la misma se puede apreciar como el ancho de banda instantáneo medido oscila entre 120 y 109 `kbps`. Este fenómeno es debido probablemente a un problema en el marcado de los tiempos de llegada de los paquetes al cliente. Particularmente en este caso la prueba se realizó con un tamaño de paquetes de 300 `bytes` y las diferencias de tiempo registradas fueron de 22 y 20 `ms`. Sin embargo, dadas las características de la prueba, es factible suponer que en realidad los paquetes arrivaron a una tasa constante, intermedia entre 22 y 20 `ms`, por lo que la capacidad real en el intervalo no es ni 120 ni 109 `kbps`, sino una intermedia entre ambas. Como se puede apreciar, mediante la aplicación del núcleo de Epanechnikov para realizar la estimación de la capacidad registrada, se detecta que la capacidad real en el intervalo de tiempo es de aproximadamente 113 `kbps`, por lo que el algoritmo se comporta de manera satisfactoria ante esta situación.

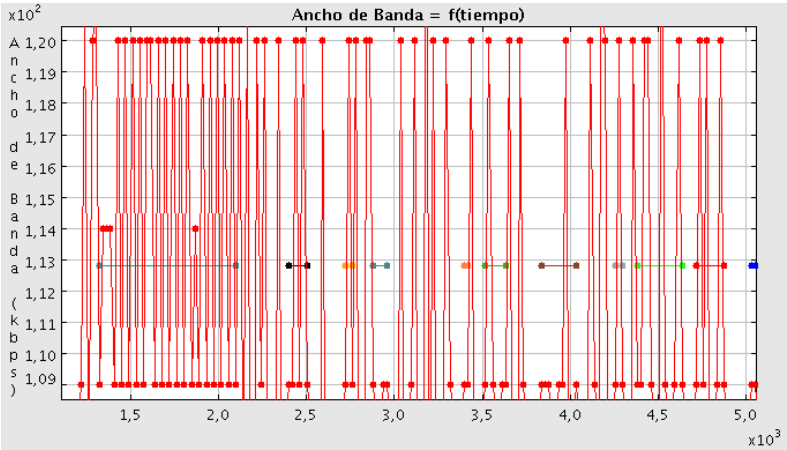


Figura 3.20: Detección de capacidades ante la presencia de anchos de banda instantáneos oscilantes.

Por último, en la figura 3.21, se puede apreciar como, ante una situación severa de ruido, en la prueba en cuestión no es posible determinar facilmente cual es la capacidad del intervalo en estudio. Por criterio de diseño, se prefirió que ante situaciones similares a ésta el sistema no identifique capacidad alguna ya que en estos casos estimar un resultado tiene una gran fuente de error, lo que puede provocar que la capacidad media calculada de la prueba se vea distorsionada.

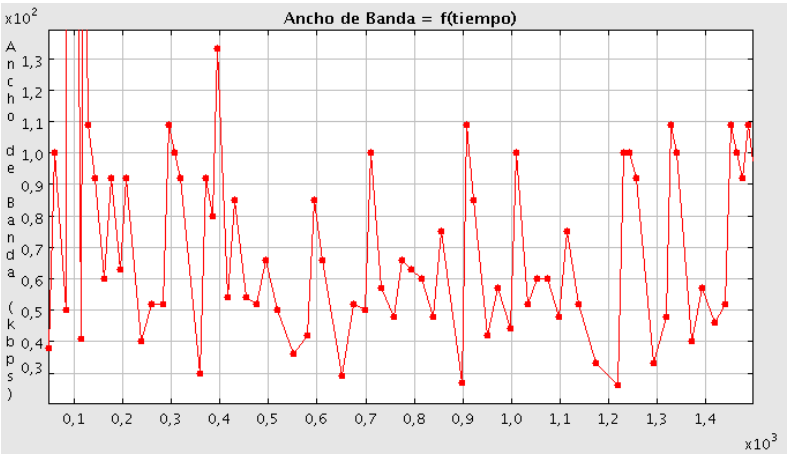


Figura 3.21: Situación en que la intensidad de ruido es tan grande que el sistema es incapaz de identificar una capacidad.

3.4. Resultados Obtenidos

3.4.1. Generalidades

Durante la etapa de realización de pruebas se efectuaron las mismas en diferentes circunstancias. Las variables consideradas para ello fueron el *lugar de la prueba (celda de servicio)*, la *potencia de la señal*, *momento en que se efectuaron las mismas* y el *tipo de móvil*. Las pruebas realizadas fueron hechas mediante un móvil/módem modelo Nokia 6230 provisto por la empresa ANTEL y un móvil/módem modelo Motorola V360. Asimismo las pruebas se realizaron por medio de un Applet[5], utilizando los móviles como módem conectado al PC, como un Midlet[6] en el propio teléfono.

Celda de servicio

La idea de efectuar pruebas en diferentes celdas viene dada por la razón de realizar experimentos con distintas configuraciones a nivel de radio y distintos niveles de tráfico, tanto de voz como de datos. Es de esperarse que no se obtengan los mismos resultados al efectuar pruebas en una radio base de un entorno urbano que en una de un área rural o poco habitada.

Potencia de la señal

El propósito de realizar pruebas bajo distintos niveles de potencia es el de observar el fenómeno de cambio de tipo de codificación de la información para su posterior transferencia en función del nivel de potencia. Si bien no se disponía del equipamiento necesario para testear la potencia de los experimentos, para la estimación del nivel de la misma se basó en las barras de potencia mostradas por el móvil.

Horario

La idea de efectuar pruebas en distintos horarios es la de observar la variación de los resultados obtenidos en función la demanda de tráfico, la cual es de esperarse que cambie según la hora del día.

Móvil

Como se mencionó anteriormente, las pruebas se llevaron a cabo tanto con un móvil modelo Nokia 6230, como con un móvil modelo Motorola V360. Mediante la realización de pruebas con diferentes dispositivos se logra

[5]Un Applet es un programa implementado en JAVA que corre sobre un navegador WEB en el cliente.

[6]Un Midlet es un programa implementado en JAVA para dispositivos de baja capacidad de procesamiento.

- Comprobar el efecto de la utilización de móviles de clase diferente al usufructuar los recursos de la red

- Comprobar la robustez del sistema ante potenciales problemas introducidos por particularidades de un dispositivo dado (ej. utilización de buffers de almacenamiento de paquetes, memoria caché, etc)

- Ampliar el espectro de pruebas a realizar

3.4.2. Pruebas realizadas y resultados obtenidos

Los lugares elegidos para realizar la gran mayoría de las pruebas fueron Punta Carretas, Playa Hermosa, Malvín, La Blanqueada y Palermo. A continuación se detalla un análisis de los resultados obtenidos para los parámetros calculados mas relevantes.

Capacidad media

En las figuras 3.22, 3.23 y 3.24 se muestran los resultados obtenidos de la capacidad media del canal en distintos experimentos realizados con el transcurso del tiempo en las celdas de Playa Hermosa, Punta Carretas y Palermo con el móvil modelo Nokia 6320.

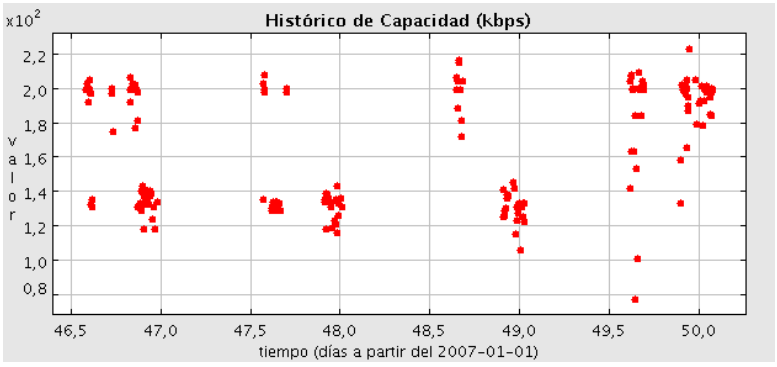


Figura 3.22: Capacidad media del canal en kbps contra días transcurridos en la celda de Playa Hermosa con el móvil modelo Nokia 6320.

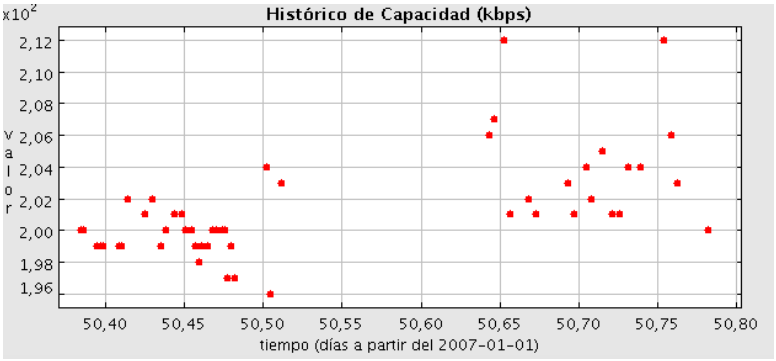


Figura 3.23: Capacidad media del canal en kbps contra días transcurridos en la celda de Punta Carretas con el móvil modelo Nokia 6320.

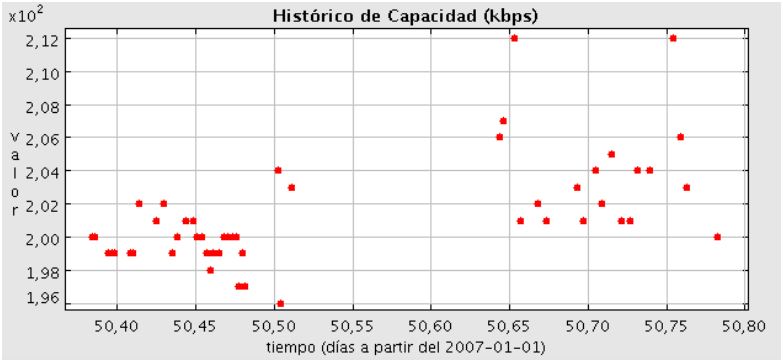


Figura 3.24: Capacidad media del canal en kbps contra días transcurridos en la celda de Palermo con el móvil modelo Nokia 6320.

Tal como se puede observar en las gráficas 3.22, 3.23 y 3.24 ante condiciones normales de tráfico en la celda las capacidades medias en los enlaces oscilan entre los 190 y 220 kbps. De estos resultados se puede conlcuir que el móvil por lo general obtiene para el downstream entre tres y cuatro *time slots* con code scheme variable entre MCS-6 (29.6 kbps por time slot) y MCS-9 (59.2 kbps por time slot).

En la figura 3.22 se observa que el patrón de las capacidades medias obtenidas en los primeros tres días de experimentos realizados por la tarde son sensiblemente mayores a las capacidades medias obtenidas por la noche. Esto se debe a que las pruebas fueron realizadas durante el fin de semana largo de carnaval. No es de extrañar que en un balneario, durante una semana de este tipo, el tráfico cursado por la noche aumente de forma considerable con respecto al tráfico de la tarde. Además, con seguridad, las celdas de la zona están dimensionadas para soportar un volumen de tráfico menor al

Lugar de estudio	Capacidad media (kbps)	Capacidad mínima (kbps)	Capacidad máxima (kbps)
Pta. Carretas	200	167	212
Palermo	202	187	224
P. Hermosa - Tarde	196	142	216
P. Hermosa - Noche	142	115	203

Tabla 3.6: Capacidades registradas por localización con el móvil Nokia 6230.

registrado en estos días por la noche ya que la cantidad de habitantes en la zona aumenta sensiblemente. Esto trae aparejado que, al tener disponibles menos *time slots* para hacer uso del servicio de datos, la capacidad media obtenida disminuya claramente. Confirmando más aún dicho fenómeno se puede observar que el último día de pruebas mostradas, correspondiente a la noche del Martes - madrugada del Miércoles, las condiciones de tráfico se vieron un poco normalizadas por lo que las capacidades obtenidas por la noche empiezan a comportarse como la media normal de la zona.

Los valores de las capacidades medias obtenidos por celda se pueden apreciar en la tabla 3.6.

En la figura 3.25 se muestran los resultados obtenidos de la capacidad media del canal en distintos experimentos realizados en las zonas de Malvín (color rojo) y Punta Carretas (color verde) con el móvil modelo Motorola V360.

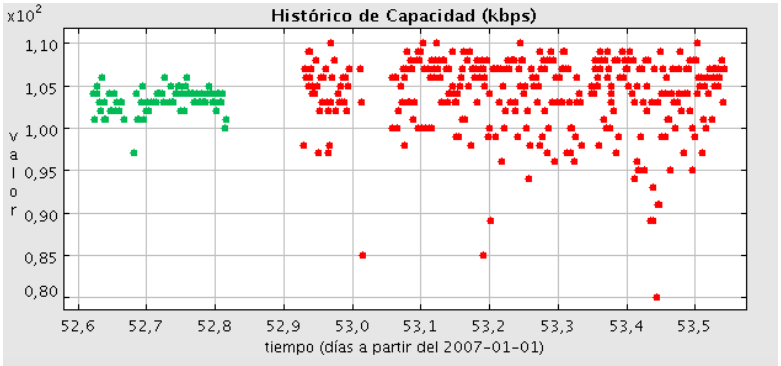


Figura 3.25: Capacidad media del canal en kbps contra días transcurridos en las zonas de Malvín y Punta Carretas con el móvil modelo Motorola V360.

Tal como se puede apreciar en la figura 3.25, las capacidades obtenidas con las pruebas llevadas a cabo con el móvil Motorola son sensiblemente menores a las registradas con el Nokia. Mediante la realización de pruebas

Lugar de estudio	Capacidad media (kbps)	Capacidad mínima (kbps)	Capacidad máxima (kbps)
Pta. Carretas	103	59	125
Malvín	104	55	150

Tabla 3.7: Capacidades registradas por localización con el móvil Motorola V360.

Móvil utilizado	Throughput medio (kbps)	Throughput mínimo (kbps)	Throughput máximo (kbps)	Relación thput/cap (%)
Nokia	98	31	119	49
Motorola	91	37	104	87

Tabla 3.8: Throughputs registradas en la zona de Malvín con los diferentes móviles.

en distintos horarios y zonas de la capital (Malvín, Punta Carretas y La Blanqueada), se descartó la idea de que las bajas capacidades registradas en las mediciones con dicho móvil fueran debido a factores como la celda de servicio o el horario en el que se llevó a cabo la prueba. Luego de descartar dichas hipótesis se comprobó que dicho comportamiento es atribuido al efecto de la utilización de un móvil distinto en la red. Sin embargo, como se aprecia en la figura 3.25, existe una leve diferencia en las capacidades registradas según la zona y el horario.

Las capacidades media, mínima y máxima registradas con dicho móvil en las zonas correspondientes se muestran en la tabla 3.7

Throughput medio

Durante las pruebas realizadas se pudo observar que, si bien las capacidades medidas del móvil Nokia son sensiblemente mayores a las del Motorola, dicho fenómeno no se ve reflejado de igual manera a la hora de hallar el throughput medio de la conexión. Si bien el throughput promedio medido con el Nokia es superior al medido con el Motorola, la relación throughput/capacidad es mucho mayor en este último. Dicha observación se puede visualizar mas claramente en la tabla 3.8.

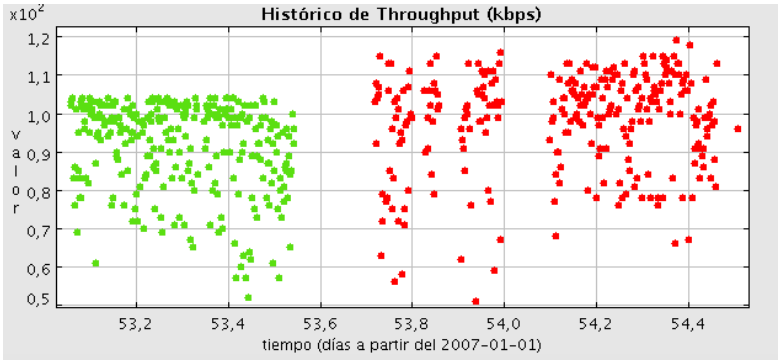


Figura 3.26: Throughputs (kbps) registrados con el Motorola (verde) y el Nokia (rojo).

De lo anteriormente expuesto se puede llegar a la conclusión de que el Motorola, si bien utiliza menos recursos de la red que el Nokia, los aprovecha de una mejor manera. Este fenómeno también puede ser visto a través del porcentaje de actividad, el cual es un indicador del aprovechamiento de los recursos de la red. La figura 3.27 confirma esta teoría.

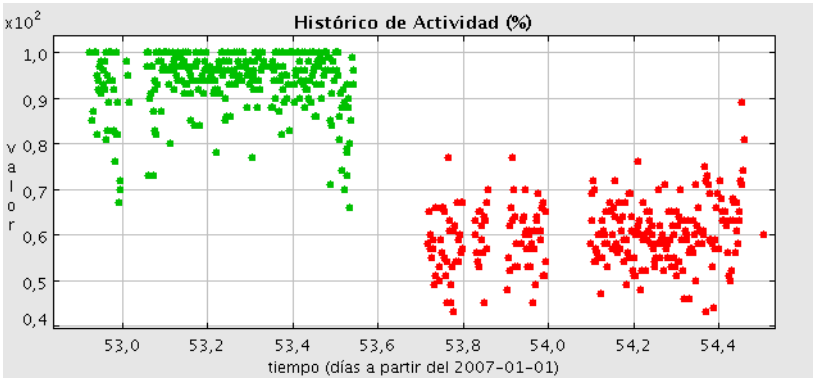


Figura 3.27: Porcentajes de actividad del Motorola (verde) y el Nokia (rojo).

RTT

Durante las pruebas realizadas, se pudo observar que los retardos registrados en la red son un tanto altos si se piensa en ciertas aplicaciones que comunmente corren sobre IP en redes fijas, como aplicaciones en tiempo real.

En la figura 3.28 se pueden apreciar los RTT medios calculados por el algoritmo en cada prueba realizada en las zonas de Malvín y Punta Carretas con el móvil Motorola. Igualmente se observó el mismo patrón de medidas en las distintas zonas de análisis.

Móvil utilizado	RTT medio (ms)	RTT mínimo (ms)
Motorola V360	897	709
Nokia 6230	1329	833

Tabla 3.9: RTT's registrados en la zona de Punta Carretas con los móviles utilizados.

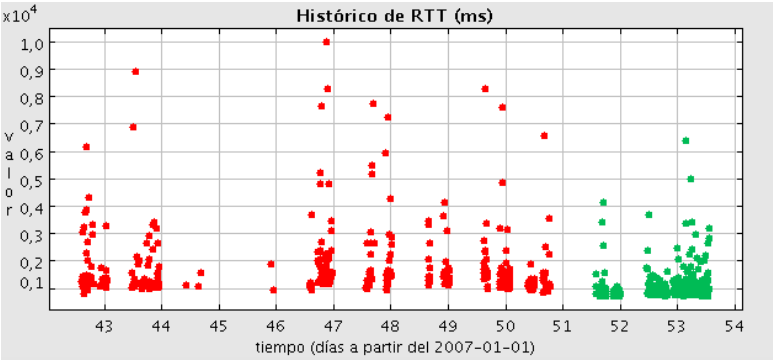


Figura 3.28: RTT (ms) registrado en la totalidad de las pruebas realizadas.

En las tablas 3.9, 3.10 y 3.11 se muestran los valores mínimo y medio del RTT en diferentes situaciones. Tal como se puede apreciar en la tabla 3.9, existen ciertas diferencias en el RTT registrado al realizar pruebas con distintos móviles y en zonas diferentes.

De la tabla 3.9 se puede apreciar la influencia del cambio del móvil al realizar experimentos en una misma zona de trabajo y en franjas horarias similares. Luego de realizadas una gran cantidad de mediciones se puede observar que el retardo introducido al utilizar el móvil Nokia es aproximadamente 400 `ms` mayor con respecto al Motorola.

En la tabla 3.10 se muestra la influencia que tiene la zona de la prueba en la medición del RTT. Se puede apreciar que ante eventuales cambios en la zona de servicio el RTT varía ligeramente. Esto puede probablemente ser atribuído a la congestión existente en cada una de las zonas en la franja horaria de realización de las pruebas.

De la tabla 3.11 se aprecia un análisis similar al de la tabla 3.10 con la salvedad de que las pruebas fueron realizadas con el móvil Nokia. Como se ve en este punto las diferencias entre ambas zonas de estudio son muy considerables, lo que puede ser atribuído tanto al período de realización de

Zona de estudio	RTT medio (ms)	RTT mínimo (ms)
Punta Carretas	897	709
Malvín	1013	721
La Blanqueada	1107	755

Tabla 3.10: RTT's registrados utilizando el móvil Motorola en distintas zonas de la capital.

Zona de estudio	RTT medio (ms)	RTT mínimo (ms)
Playa Hermosa	2376	920
Punta Carretas	1329	833

Tabla 3.11: RTT's registrados utilizando el móvil Nokia en distintas zonas del país.

las pruebas[7] como a la las características particulares de cada zona, dado que en la zona de Playa Hermosa la potencia registrada por el móvil es inferior a los niveles registrados en las distintas zonas de Montevideo.

Jitter

De las pruebas realizadas se llega a la conclusión, como era de esperarse, que el Jitter registrado en la red tiene un estrecho vínculo con el RTT medido en la misma, de modo tal que si uno aumenta el otro también lo hace.

En la figura 3.29 se puede apreciar que el Jitter registrado en las mediciones nunca fue menor a los 20 ms, llegando a valores máximos del orden de los 90 ms. Este rango de valores puede traer como consecuencia que ciertos servicios que requieren de un bajo Jitter para su correcto funcionamiento, como ser Push To Talk, se vean degradados considerablemente.

[7]Recordar del punto 3.4.2 de ésta sección que las pruebas realizadas en Playa Hermosa fueron en su mayoría durante el feriado de Carnaval, momento en el cual la cantidad de tráfico en la zona aumentó considerablemente.

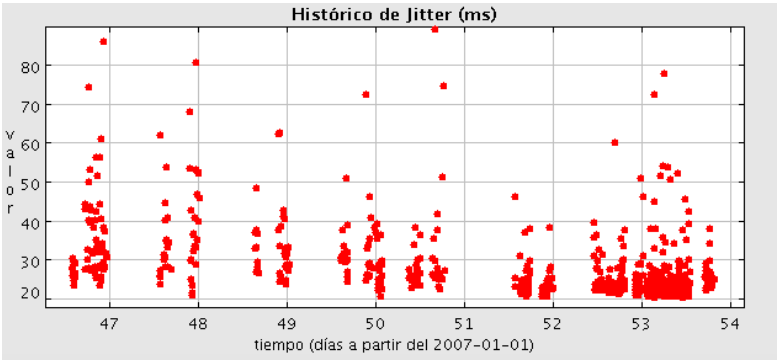


Figura 3.29: Jitter (ms) registrado en la totalidad de las pruebas realizadas.

Pérdidas

Tal como se puede apreciar en la figura 3.30 el grueso de las pérdidas registradas se encuentran concentradas entre el 0 y el 4 % de los paquetes enviados. Sin embargo, en varias pruebas las mismas llegaron a superar ampliamente el 10 %. Una observación interesante es que, ante un mismo escenario, en pruebas muy cercanas en el tiempo los valores registrados pueden llegar a ser muy dispares, lo que remarca la dinámica de la red.

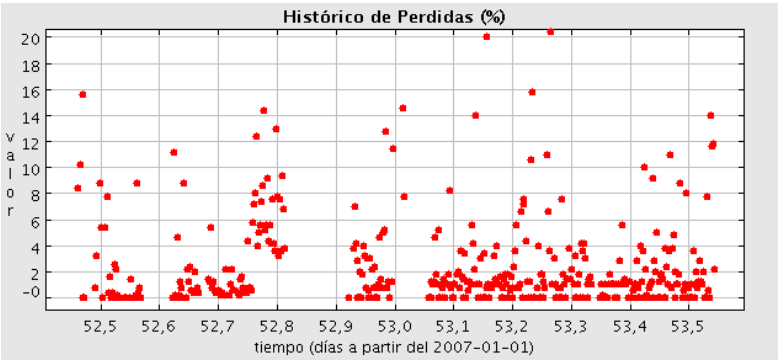


Figura 3.30: Pérdidas registradas en las pruebas realizadas.

3.5. Conclusiones

Se desarrolló e implementó un algoritmo propio capaz de analizar las características del canal y obtener una gran cantidad de indicadores de performance caracterizadores del funcionamiento de la red.

El algoritmo desarrollado tiene como característica principal el de estar fuertemente orientado al funcionamiento del protocolo en estudio. Esto trae como resultado la gran cantidad de información que se puede extraer a partir de la utilización del mismo.

El algoritmo fue testeado bajo diferentes circunstancias, como lo son la zona de servicio, el horario, el móvil empleado y la calidad de la señal. Gracias a esto fue posible comprobar el funcionamiento del mismo, así como obtener una base de datos con una cantidad de experimentos registrados considerable, los cuales avalan el desempeño de éste.

Durante las primeras pruebas realizadas se fue ajustando el funcionamineto del algoritmo de modo tal que, luego de obtener su versión estable, permitió realizar el estudio de las conexiones GPRS/EDGE tal cual fue descrito en la sección 3.4.2. En las mismas se pudo observar la influencia de los factores mencionados en el párrafo anterior. Como observación particular, vale la pena mencionar la influencia del móvil en el resultado, el cual delimita notablemente las condiciones de performance en la conexión, de una manera aún mayor a la esperada en un principio.

Capítulo 4

Análisis de Performance de WAP

4.1. Introducción

El crecimiento en el acceso a Internet de los últimos años de la década de los 90, conjuntamente con el aumento en la utilización de dispositivos móviles como ser PDAS[1], notebooks y teléfonos celulares, ha evidenciado la necesidad de que las personas pudieran estar conectadas a la red todo el tiempo y en cualquier lugar, utilizando como acceso al medio estos dispositivos.

Tradicionalmente el protocolo TCP/IP ha sido la pila de protocolos por excelencia en la cual se basa la red de Internet que hoy conocemos, debido a su robustez y escalabilidad. Sin embargo las redes inalámbricas tienen la característica de tener una tasa de errores de bit alta, ancho de banda reducido, poca disponibilidad de los recursos y diferencia en los canales de descarga y subida. Estas características hacen que sea necesario adaptar el protocolo TCP de manera que el nuevo protocolo sea específico para enlaces con los rasgos antes mencionados.

Dada la capacidad de procesamiento reducida y la carencia gráfica que tienen los terminales móviles en comparación a los PCs, así como la ineficiencia de protocolos como HTTP[2] en enlaces de bajo ancho de banda, fue necesario redefinir un protocolo de aplicación que permita la navegación WEB. Las soluciones que surgieron con más fuerza fueron WAP[3] y NTT DoCoMo's i-mode.

WAP es un estándar propuesto por el WAP Forum, integrado por fabricantes y proveedores de telefonía móvil. Consiste en una pila de protocolos

[1] PDA: Personal Digital Assistant
[2] Hypertext Transfer Protocol
[3] Wireless Aplication Protocol

que permiten la navegación WEB de forma eficiente por medio de un proxy WAP ubicado en la red celular.

NTT DoCoMo's i-mode es una solución propietaria de la empresa japonesa NTT DoCoMo, que se diferencia de WAP en la tecnología que utiliza y en que no es un estándar.

Hoy en día, entre los servicios de valor agregado que permiten brindar los teléfonos celulares del mercado a los proveedores, la navegación con WAP es el mas utilizado y es por eso que se ha decidido dedicarle un estudio detallado no solo al protocolo en sí sino a la performance del mismo.

4.2. Objetivo

Si bien la evaluación de performance en redes de computadoras es un tema bastante conocido en el ámbito de las redes de datos, en el área de los celulares no ha sido del todo estudiado, lo cual motivó la implementación de una aplicación que permita evaluar la performance de los protocolos utilizados por WAP, específicamente la variante del protocolo TCP que utiliza.

El objetivo propuesto era el de implementar una aplicación que permitiera brindar parámetros objetivos de calidad de servicio de navegación con el protocolo WAP, ya sea para un usuario del servicio como para un proveedor que desee conocer el estado actual de su red.

Entre los parámetros de calidad que se pretendían calcular, se destacan los siguientes:

- RTT
- Thoughput
- Goodput
- Porcentaje de actividad del canal
- Porcentaje de retransmisiones

Para poder obtener un resultado experimental de la calidad de la conexión WAP, es necesario poder realizar medidas, ya sea en forma pasiva o activa, del tráfico intercambiado entre ambos extremos de la conexión. Es por esto que la captura de datos se convierte en el primer objetivo a sortear al momento de buscar una solución al problema.

4.3. Solución al Problema Planteado

4.3.1. Captura de datos

A. Captura en la LAN del proxy WAP

Consideraciones generales

El primer desafío planteado a la hora de cumplir con esta etapa del proyecto es el de diseñar una herramienta mediante la cual un operario de la red sea capaz de monitorear el estado de las conexiónes WAP activas. Complementariamente con dicho objetivo se implementaron funcionalidades de valor agregado a dicha herramienta, a saber

- Aplicación de filtros a nivel de paquetes (dirección IP y puerto)
- Monitoreo del enlace durante un tiempo determinado
- Monitoreo del enlace hasta capturar una cantidad determinada de paquetes
- Visualización de capturas anteriores
- Visualización de la evolución histórica de los parámetros de performance calculados tanto de un usuario en particular como de una ubicación dada

Mediante la implementación de las funcionalidades mencionadas el sistema diseñado puede ser utilizado por medio de operarios del proveedor de servicios WAP de manera de poder monitorear parte de interés de la red en cuestión. De este modo se pueden atender reclamos de una manera mas eficiente, tener un control de la situación actual e histórica de la red, diagnosticar problemas y preever inconvenientes futuros.

Implementación de la solución

La solución encontrada consiste en capturar los paquetes de la conexión TCP entre el proxy WAP y el terminal o un grupo de terminales. El ordenador encargado de efectuar la captura deberá estar ubicado en la misma LAN que el proxy WAP de modo tal de poder capturar los paquetes antes que sean traducidos por el mismo. En la figura 4.1 se muestra este escenario. Finalmente a partir de los datos capturados[4] se realizan los cálculos pertinentes, los cuales son explicados en el punto 4.3.3.

[4]Ver punto 4.3.2.

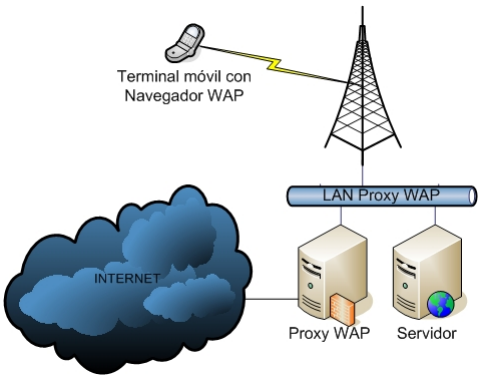


Figura 4.1: Escenario para el caso de un operador que quiere monitorear su red.

B. Captura en el servidor

Consideraciones generales

El segundo aspecto a abarcar dentro de esta etapa del proyecto es el de implementar un sistema mediante el cual un usuario de WAP pueda determinar el estado de su conexión actual. Dadas las características del problema a abordar, se plantearon las siguientes limitantes.

1. El resultado de la prueba debía poder ser interpretado por un usuario sin formación técnica en el área
2. Cualquier usuario WAP debía poder realizar la prueba, independientemente de las características del móvil en cuestión (es decir no se podían imponer limitaciones específicas sobre el móvil como, por ejemplo, el de poder interpretar programas codificados en JAVA)
3. El sistema debía ser implementado de manera tal que se pudieran probar los algoritmos de estimación de performance de manera satisfactoria, sin necesidad de ubicar el sistema de estimación dentro de la red del proveedor de servicios

Una primer propuesta fue la de capturar paquetes directamente en el móvil, pero luego de un estudio detallado de la viabilidad de la solución, resultó que no iba a ser fácilmente implementable en el lenguaje JAVA debido a las limitaciones de la versión *Micro Edition* y a la falta de librerías adecuadas para la captura de paquetes en dispositivos móviles. Otra alternativa al problema planteado era la de implementar la captura en el móvil mediante el lenguaje de programación de bajo nivel *Symbian*. Dicha opción fue descartada luego de hacer una evaluación del tiempo requerido para poder codificar una aplicación de tal magnitud en este lenguaje. Otro punto importante que implicó descartar las alternativas mencionadas anteriormente es que ambas soluciones no satisfacen el requerimiento de que cualquier

usuario WAP debía poder realizar la prueba, independientemente de las características del móvil que se utilice para la navegación.

Como se menciónó anteriormente, los enlaces celulares tienen la característica de tener más pérdidas, más jitter y latencias más altas que los enlaces en redes fijas. En [9] se hace un estudio de la relación entre las pérdidas y las latencias para una conexión a Internet en el tramo cableado y el inalámbrico, y se muestra que las pérdidas en el tramo inalámbrico dominan practicamente las pérdidas de la conexión. Es por esto que para evaluar la performance en un enlace celular es posible estimar la calidad de la conexión extremo a extremo y adjudicarle las características de la performance estimada al enlace inalámbrico.

Por lo mencionado anteriormente es posible capturar los paquetes en el servidor que brinda el servicio y evaluar la performance de la conexión extremo a extremo tal como se muestra en la figura 4.2. La *hipótesis fundamental* que se tiene que asumir en este punto es que no se almacena información en el proxy WAP. Este punto del problema es realmente crucial porque, en caso contrario, las pérdidas notables de performance podrían no ser detectadas por el sistema de monitoreo de la conexión. Cabe recordar que, al tratarse de conexiones TCP, es muy intuitivo suponer que ante condiciones desfavorables del estado de la conexión las retransmisiones de paquetes ocurridas en el enlace aumenten significativamente. Si conjuntamente con el punto anterior se tiene en cuenta que las pérdidas significativas de performance ocurren en la interfaz de aire y, por lo tanto, en el tramo del enlace correspondiente entre el cliente y el proxy WAP, en caso de que el proxy mantuviera en *caché*[5] la información recientemente intercambiada con el cliente, ante un pedido de retransmisión del mismo, el proxy podría perfectamente retransmitirle la información almacenada en *caché* y por lo tanto dicha retransmisión no sería detectada por el sistema de monitoreo.

Dado que el funcionamiento del proxy WAP depende del fabricante y de las opciones que habilite el operador, para validar la hipótesis de que el proxy no acumula información fue necesario consultar con personal técnico de la empresa ANTEL, el cual nos confirmó dicha hipótesis y, por lo tanto, dio vía libre a la implementación del sistema.

Implementación de la solución

Dadas las consideraciones generales mencionadas anteriormente se consideró que la solución a implementar debería de satisfacer algunos requerimientos básicos.

[5]El caché es un tipo de memoria dinámica utilizada comumente para minimizar la cantidad de operaciones en el sistema particular en el que se implementa.

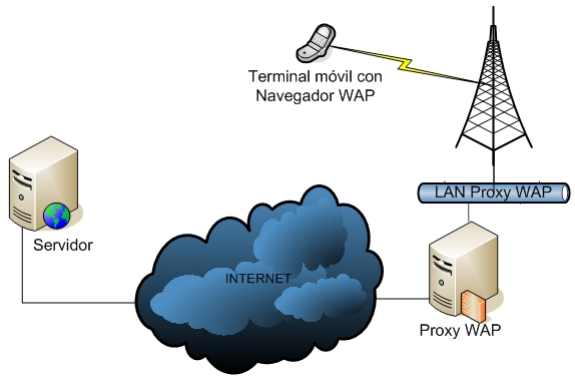


Figura 4.2: Escenario para el caso de suscriptor que quiere monitoriar su conexión.

- Las mediciones a efectuar se deberían hacer de un modo pasivo, es decir sin introducción de tráfico de prueba en el enlace.

- El tipo de conexión establecida entre el cliente y el servidor debería ser del estilo de las mas comunmente cursadas en servicios del tipo WAP con el propósito que de el test realizado se obtengan resultados representativos de una conexión WAP standard.

- Con el fin de incorporar la herramienta al software existente de *MetroNet* la misma debería ser capaz de correr en el servidor del sistema mencionado.

Una vez impuestas todas las condicionantes mencionadas se prosiguió con la determinación del tipo de sistema mediante el cual se iba a establecer el intercambio de datos entre el cliente y el servidor. Dadas las características intrínsecas del protocolo WAP resultó natural que el sistema a implementar consistiera en un conjunto de páginas WML mediante la cual se generara una cantidad de tráfico suficiente como para sacar datos estadísticamente considerables acerca del estado de la conexión actual del cliente. Para efectuar el armado de la estructura de la serie de páginas se consultó al personal de ANTEL acerca de cuales eran las características generales de las páginas mas frecuentemente utilizadas por sus clientes. Una vez zanjado dicho problema se procedió con la codificación de una serie de páginas WML con las características deseadas.

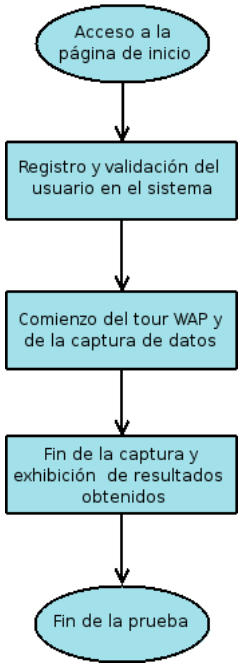


Figura 4.3: Flujo de eventos al utilizar el sistema de estimación de performance de la conexión WAP.

Tal como se muestra en la figura 4.3, el flujo de eventos del sistema consiste en las etapas de

1. Registro del usuario en el sistema
2. Validación del usuario
3. Comienzo de captura de datos
4. Comienzo del tour WAP
5. Fin del tour y de la captura de datos
6. Devolución de los resultados de la prueba

Los puntos más delicados en la implementación del sistema son la captura de datos, la manipulación de los datos capturados y el cálculo de los resultados obtenidos. Estos aspectos se explican a continuación en los puntos 4.3.2, 4.3.3 y 4.3.4 respectivamente.

4.3.2. Implementación de la captura

Para hacer posible la implementación de la captura se plantearon e implementaron inicialmente dos alternativas diferentes con el fin de evaluar sus

características y así decidir cuál solución era la mas viable para ser implementada en el sistema final. Las mismas eran

- Captura mediante la interacción JAVA - `tcpdump`
- Captura mediante la utilización de librerías particulares

Captura mediante la interacción JAVA - tcpdump

Esta solución consiste en, mediante la ejecución de código en JAVA, la manipulación de la aplicación `tcpump`[6], disponible en cualquier sistema operativo basado en UNIX. Dicha alternativa presenta como *ventaja* que se pueden aprovechar todas las funcionalidades de dicha aplicación. Además, dado que es una aplicación altamente probada y aceptada en el ambiente, la misma presenta gran robustez y confiabilidad a la hora de ser utilizada. Otra de las ventajas presentadas por dicha alternativa es que no se necesita de instalación de librería particular alguna para que el sistema funcione correctamente. La *desventaja* mas notable que presenta dicha solución es que el sistema pasa a depender del funcionamiento de un programa externo, por lo que el mismo no estaría diseñado al cien por ciento en JAVA. Por otro lado una potencial interacción con el `tcpdump` implicaría que el sistema operativo en el cual se debe instalar el sistema tenga que estar basado en sistemas UNIX, por lo que de éste modo se pierde una de las grandes ventajas que tiene el codificar un programa en JAVA, es decir, el de ser multiplataforma.

Captura mediante la utilización de librerías particulares

La otra alternativa planteada para atacar el problema de la captura de datos es la de la instalación de una librería particular de JAVA mediante la cual se permita la implementación de un capturador de paquetes. La alternativa estudiada fue la de instalar la librería `jpcap`[7] de JAVA. Dicha librería trabaja sobre la librería `libpcap` (disponible en cualquier sistema operativo) que es la encargada de manejar las instrucciónes del sistema operativo con el fin de efectuar la captura de los paquetes. La librería `libpcap` es utilizada por reconocidos programas capturadores de datos en el área, como el `tcpdump` o el `ehtereal`[8]. La *ventaja* que presenta implementar esta solución es que el programa en su conjunto pasa a estar codificado al 100 % en JAVA, por lo que el mismo se transforma en una herramienta compacta y multiplataforma. Además se puede personalizar las funciones del sniffer propio acomodándolo a los requerimientos de la aplicación para la cuál fue diseñado. Para adaptar el capturador diseñado a los requerimientos del problema se reescribió la clase `PacketHandler.java` provista por el `jpcap` de

[6] Ver [5].
[7] Ver [3].
[8] Ver [4].

modo tal que el mismo devolviera la información necesaria en el formato requerido para satisfacer los requerimientos de la aplicación deseada. Parte de la recodificación del código se muestra a continuación.

```
if (packet instanceof TCPPacket){
   TCPPacket tcpPacket = (TCPPacket)packet;
   byte[] data = tcpPacket.getTCPData();
   String srcHost = tcpPacket.getSourceAddress();
   String dstHost = tcpPacket.getDestinationAddress();
   int srcPort = tcpPacket.getSourcePort();
   int dstPort = tcpPacket.getDestinationPort();
   long ack = tcpPacket.getAcknowledgementNumber();
   long seq = tcpPacket.getSequenceNumber();
   int win = tcpPacket.getWindowSize();
   String tiempo = ((tcpPacket.getTimeval()).toString()).split("s")[0];
   int tamanioTCP = tcpPacket.getPayloadDataLength();
   int tamanioIP = tamanioTCP + tcpPacket.getIpHeaderLength() +
   tcpPacket.getTcpHeaderLength();
   boolean esSyn = tcpPacket.isSyn();
   boolean esFin = tcpPacket.isFin();
   boolean esPsh = tcpPacket.isPsh();
   boolean esRst = tcpPacket.isRst();
   boolean esUrg = tcpPacket.isUrg();
   boolean esAck = tcpPacket.isAck();
}
String s = tiempo+"::"+srcHost+"::"+srcPort+"::"+dstHost+"::"+dstPort+"::";
String info = "Datos";
if(esSyn == true){
   bandera = "S::";
}
if(esPsh == true){
   bandera = "P::";
}
if(esFin == true){
   bandera = "F::";
}
if(esRst == true){
   bandera = "R::";
}
s = s +bandera;
if(esAck == true){
   s = s +tamanioIP+"::"+tamanioTCP+"::"+seq+"::"+ack;
}
else{
```

```
   s = s +tamanioIP+"::"+tamanioTCP+"::"+seq+"::"+"-1";
}
if(esUrg == true){
   s = s +"::"+win+"::"+"::U::"+info;
}
else{
   s = s +"::"+win+"::"+"-1::"+info;
}
```

La alternativa presentada tiene como mayor desventaja el hecho de que para permitir el correcto funcionamiento del programa es necesario instalar previamente la librería `jpcap` con todos sus paquetes asociados, hecho que si bien no es del todo práctico tampoco es demasiado tedioso.

Luego de haber implementado ambas alternativas se decidió que lo mas adecuado para satisfacer los requerimientos del problema de una manera eficiente era la implementación de la segunda solución planteada.

4.3.3. Manipulación de datos capturados

Una vez que se tienen paquetes capturados por el sistema, se procede a identificar las diferentes conexiones TCP dentro de la captura. Cada conexión se identifica biunívocamente con un número de puerto diferente. Una vez separadas las diferentes conexiones, se identifican dentro de cada una de ellas aquellos paquetes que pertenecen al estableciemiento, a la finalización y al intercambio de información de paquetes TCP entre los distintos hosts. Para realizar esta discriminación es necesario observar las banderas del encabezado TCP de los paquetes capturados. También se extraen otros datos de las cabeceras de TCP e IP para el posterior procesamiento de cada paquete. La clasificación de cada conexión en sus diferentes etapas se efectúa según el siguiente criterio.

Establecimiento de la conexión

Viene identificada por el handshake de 3 vías de TCP. Se compone de los paquetes con las banderas SYN, SYN ACK y ACK. En caso de que el establecimiento no sea el ideal (es decir por medio de los tres paquetes mencionados) se incluyen dentro de la etapa del establecimiento de la conexión todos los paquetes necesarios para llevar a cabo el mismo.

Transferencia de información

Los paquetes identificados como pertenecientes a la transferencia de información de la conexión son los correspondientes al rango que comprende desde el primer paquete identificado luego del establecimiento de la conexión hasta el último antes de que se inicie la finalización de la misma.

Finalización de la conexión

La finalización de la conexión viene dada por la presencia de paquetes con la bandera de FIN o RST según la situación particular de la misma. Esta etapa de la conexión la componen todos los paquetes desde que aparece el primero con alguna de las banderas antes mencionadas hasta que ya no quedan más paquetes capturados de dicha conexión.

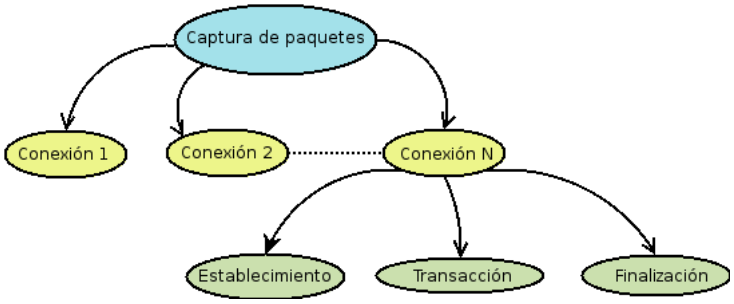


Figura 4.4: Clasificación de la captura de paquetes.

Para cada etapa de las conexiones TCP se analizan los diferentes parámetros y se realizan cálculos correspondientes a partir de los mismos.

4.3.4. Cálculo de parametros objetivos de calidad de la conexión

Establecimiento de conexión TCP

Para cada establecimiento de conexiones TCP se analiza si el establecimiento de conexión de 3 vías se realizó correctamente, si hubieron retrasmisiones en algún sentido o si no se pudo establecer la conexión. Cada establecimiento se lo clasifica de la siguiente manera:

- óptimo: si no hubieron retrasmisiones de paquetes y el establecimiento se llevó a cabo
- con errores leves: si hubieron retrasmisiones en alguno de los 2 sentidos pero el establecimiento se llevó a cabo
- fallido: en el caso que no se pudo llevar a cabo el establecimiento de la conexión.

En las figuras 4.5 y 4.6 se observa esta clasificación para los casos en que la captura se realice en la LAN del proxy WAP o en un servidor de Internet respectivamente. En función de la cantidad de establecimientos de cada una de las 3 categorías se hace una primera aproximación de la calidad del enlace. En [9] se realiza un análisis similar, capturando los paquetes entre la interfaz G_i (GPRS Interface) y la red de Internet, analizando el tráfico TCP de una conexión extremo a extremo, sin la presencia del proxy WAP.

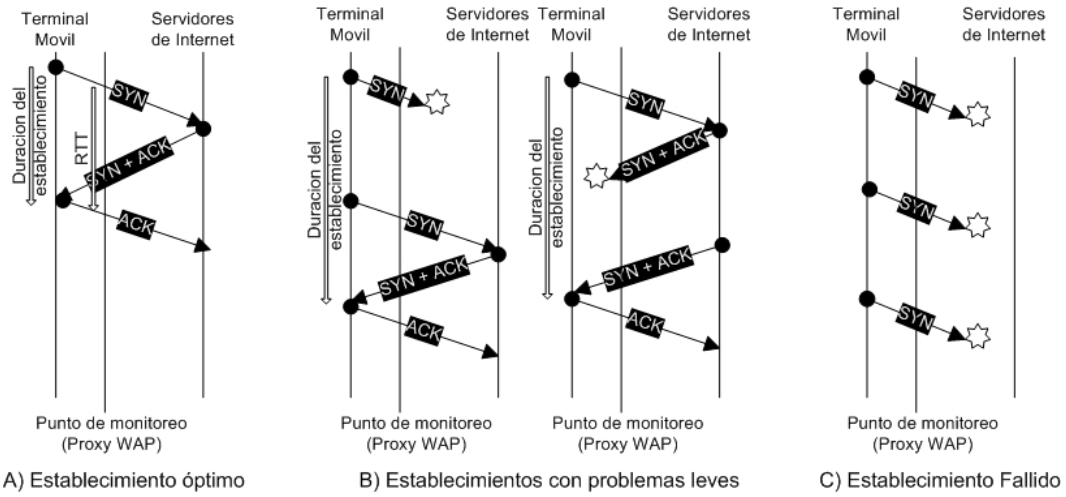


Figura 4.5: Clasificación de establecimientos de conexión TCP cuando se captura en el PROXY WAP.

RTT

En [19] se describen 2 procedimientos para realizar la estimación del RTT con la captura de los paquetes de la conexión TCP.

Método SYN ACK

El mencionado método consiste en tomar los paquetes del establecimiento de la conexión TCP, los cuales tienen un tamaño conocido, y medir la diferencia de tiempo entre que parte la confirmación del pedido de conexión por medio de la bandera SYN ACK en el encabezado de TCP, y el arribo del reconocimiento del mismo, identificado por la bandera ACK. El proceso mencionado se puede apreciar gráficamente en la figura 4.6. Esta técnica conlleva una estimación del RTT adecuada cuando se cumplen las siguientes condiciones.

- La transmisión del paquete SYN ACK y la confirmación del mismo no son retrasadas por algún motivo
- Ni el paquete SYN ACK ni el ACK se pierden en el trayecto

Método `Slow Start`

El segundo método consiste en estimar el RTT a partir de aquellos paquetes pertenecientes a la etapa de `Slow Start`[9] de la conexión TCP. Luego de

[9] Ver [2].

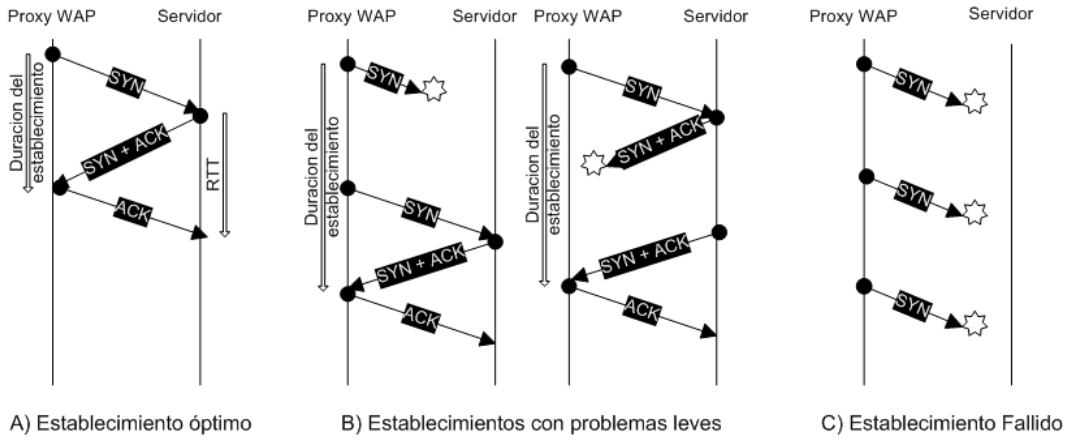


Figura 4.6: Clasificación de establecimientos de conexión TCP para capturas en un servidor de Internet.

que el establecimiento de la conexión TCP concluye, la transferencia de los datos se encuentra gobernada por el algoritmo `Slow Start` de TCP. Durante este período, el enviador de los datos incrementa la ventana de congestión por un `Maximun Segment Size` (MSS) por cada ACK recibido que confirma los datos. La idea básica de la estimación por medio del método del `Slow Start` es que la diferencia de tiempos entre las dos primeras ráfagas de datos son aproximadamente iguales al RTT de la conexión, lo cual se puede apreciar gráficamente en la figura 4.7.

Dado que en [19] se concluye que el método mas efectivo y simple para efectuar la medición del RTT es el primero de los mencionados, se optó por incorporarlo al sistema. Para mayor robustez del resultado, se realiza el cálculo del RTT de cada uno de los establecimientos de conexión a lo largo de la prueba y se realiza un promedio de todos los valores hallados. El método implementado para efectuar dicha estimación se encuentra en la clase `EstablecimientoTCP.java` y básicamente funciona como se muestra a continuación.

```
public double rttSA(){
   Paquete paq;
   for(int i = 0; i < paquetes.size(); i++){
      if(paquetes.get(i).getFlag().equals(BanderasTCP.SYN) &&
      paquetes.get(i).getAck() != -1){
         tSynAck = paquetes.get(i).getTiempo();
         seqSynAck = paquetes.get(i).getNroSecuencia();
         tamanioSynAck = paquetes.get(i).getTamanioDatosEfectivos();
      }
      else if(paquetes.get(i).getAck() == seqSynAck + 1 + tamanioSynAck){
         tAck = paquetes.get(i).getTiempo();
         break;
      }
   }
   return (tAck - tSynAck)*1000;
}
```

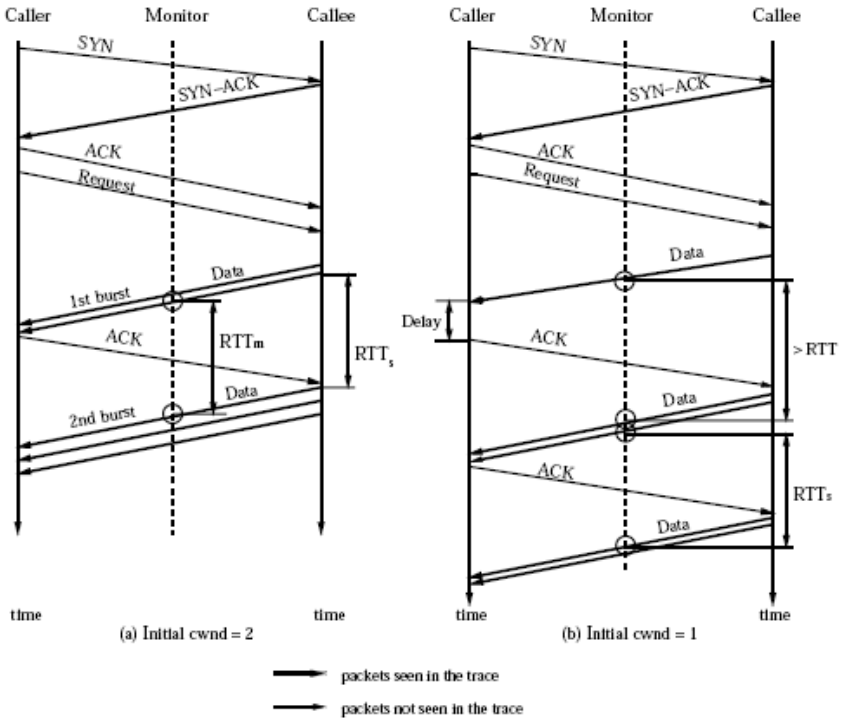


Figura 4.7: Cálculo del RTT por el método `Slow Start`.

Porcentaje de actividad del canal

Es necesario para el cálculo de otros parámetros como ser el `thoughput` o el `goodput` el poder identificar los intervalos en los cuales hay actividad, es decir en los cuales efectivamente hay intercambio de paquetes. Para esto se asocia el tipo de tráfico estudiado al de un modelo del tipo ON-OFF. En dicho modelo un tiempo ON (T_{ON}) se asocia a un intervalo de tiempo durante el cual hay actividad en el canal, mientras que un tiempo OFF (T_{OFF}) se asocia al intervalo de tiempo durante el cual el canal se encuentra inactivo. Intuitivamente, para el caso de conexiones WAP, se puede asociar un T_{ON} al intervalo de tiempo en que el cliente se encuentra descargando información de la red, mientras que un T_{OFF} se asocia al intervalo de tiempo en el cual el cliente se encuentra interpretando la información descargada. Con el fin de poder discriminar entre un T_{ON} y T_{OFF} se implmentó el *algoritmo de las medias móviles* visto en el punto B de la sección 3.2.2.

Una vez identificados los intervalos de tiempo T_{ON}, se procede al cálculo del porcentaje de actividad del canal de la siguiente manera

$$\%Act = 100\frac{T_{ON}}{T_{tot}} \tag{4.1}$$

siendo T_{tot} el tiempo desde que aparece el primer pedido de establecimiento de conexión y el último paquete de la última liberación capturados.

Vale la pena recalcar que el porcentaje de actividad no es un parámetro indicativo de la calidad de la conexión debido a que en los tiempos que se manejan para realizar el cálculo esta incluido el tiempo que le lleva a los usuarios mirar el contenido del sitio en el que se encunetra, por lo que dicho valor depende claramente del cliente en particular, careciendo de objetividad.

Throughput

Una vez que se tienen identificados los intervalos de actividad, se halla el `throughput` de la conexión, el cual se calcula como la relación entre la cantidad de bytes de información trasmitida en cada T_{ON} y la duración del mismo

$$Thput(kbps) = 8\frac{T_{ON}(bytes)}{T_{ON}(mseg)}$$

Finalmente se calcula el `throughput` medio y máximo durante toda la prueba.

Goodput

El goodput por su parte se calcula de manera análoga, donde el cociente se realiza entre la carga útil, es decir sin los encabezados de capa 4, y el intervalo de tiempo T_{ON}.

Retrasmisiones

Se analizan las retrasmisiones que se producen a lo largo de la conexión tanto en el downstream como en el upstream y se realiza una estadística de las pérdidas en ambos sentidos. Para ello se analizan aquellos paquetes que tienen los mismos números de secuencia[10]. El método encargado de realizar dicho cálculo es `analizoRetransmisiones`, perteneciente a la clase `TransferenciaTCP.java`.

```
public double[] analizoRetransmisiones(){
   Paquete paq;
   for(int i = 0; i < paquetes.size(); i++){
      paq = paquetes.get(i);
      if((paq.getHostOrigen()).equals(servidor)){
         if(paqsServidor.contains(paq)){
            retransmisionesServidor ++;
            totalPaqServ ++;
         }
         else{
            paqsServidor.add(paq);
            totalPaqServ ++;
         }
      }
      else{
         if(paqsCliente.contains(paq)){
            retransmisionesCliente ++;
            totalPaqCli ++;
         }
         else{
            paqsCliente.add(paq);
            totalPaqCli ++;
         }
      }
   }
resultados[0] = (retransmisionesServidor/totalPaqServ)*100;
resultados[1] = (retransmisionesCliente/totalPaqCli)*100;
return resultados;
}
```

[10]Por mas información se puede consultar [10].

4.3.5. Valoración de los resultados obtenidos

Una vez que se obtienen todos los datos necesarios para calificar el estado de la conexión son guardados en una base de datos. Para cada parámetro se calcula la media y la desviación estandar de los registros históricos y se compara con la prueba actual. Si el valor actual de un parámetro cae en un entorno de ancho igual a la desviación estándar alrededor de la media, se le da al mismo una valoración de normal. Si en cambio cae fuera del rango normal, dependiendo si es mayor o menor que las cotas del intervalo se lo valora como bueno o malo. Esta comparación se efectúa tanto a nivel general de todas las pruebas realizadas en el histórico como a nivel de cada usuario en particular, teniendo en cuenta solamente las pruebas realizadas por el mismo.

4.4. Resultados Obtenidos

4.4.1. Generalidades

Al igual que en el caso del algoritmo explicado en el capítulo 3, durante esta etapa se efectuaron pruebas en diferentes circunstancias. Las variables consideradas para ello fueron el *lugar de la prueba (celda de servicio)*, la *potencia de la señal*, el *horario* y el *móvil* utilizado.

4.4.2. Pruebas realizadas y resultados obtenidos

RTT

En la figura 4.8 se muestran algunos de los resultados obtenidos del cálculo del RTT mediante el método `Syn Ack`. Tal como se puede apreciar en la misma, los resultados son muy variados. Generalmente los mismos se encuentran en una franja entre 500 y 1600 `ms`.

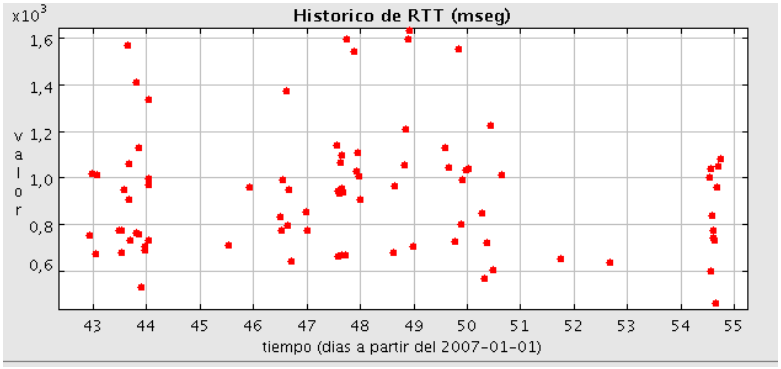


Figura 4.8: RTT hallado por el método `Syn Ack` en las pruebas realizadas.

Lugar de estudio	**RTT medio (ms)**	**RTT mínimo (ms)**	**RTT máximo (ms)**
`Malvín`	819	461	1052
`Palermo`	895	531	1569
`P. Hermosa`	1134	642	5277

Tabla 4.1: RTT obtenido en las diferentes zonas de estudio.

En la tabla 4.1 se muestran los resultados del RTT obtenidos en las distintas zonas de estudio.

De la tabla 4.1 puede observarse que el RTT en la zona de Playa Hermosa es sensiblemente superior al medido en el resto de las localizaciones de Montevideo. Recordando los conceptos explicados en la parte de resultados del capítulo anterior, una justificación posible a dicho fenómeno es que el grueso de las pruebas realizadas en la zona de Playa Hermosa fueron hechas durante el feriado de carnaval, período en el cual la congestión en las celdas de servicio eran considerables debido a que, seguramente, las mismas no se encuentran dimensionadas para alojar grandes picos de tráfico como los registrados en épocas del estilo. En este punto vale la pena destacar que, dado que para realizar pruebas se necesita la interacción de una persona, es posible establecer un mapeo entre la calidad percibida por el usuario en el momento de realizar las pruebas y el resultado obtenido. Es de esperarse que cuanto mayor sea el valor del RTT, peor será la calidad de la navegación vía WAP. Esto puede justificarse teniendo en cuenta que existe una estrecha relación entre el RTT y el tiempo de respuesta del servidor ante la petición de información. Esto trae aparejado que al realizar consultas por páginas WAP, cuanto mayor sea el RTT mas tiempo deberá esperar el usuario a que se le devuelva la página por lo que peor será la calidad percibida de la conexión. En este aspecto, se señala que la calidad percibida en el momento de realizar la mayoría de las pruebas en la zona de Playa Hermosa la calidad de navegación fue sensiblemente inferior a la percibida al navegar en las zonas de Montevideo, lo que concuerda con los resultados obtenidos para el RTT.

Un aspecto interesante a considerar es el de la comparación entre el RTT obtenido mediante el método `Syn Ack` y el hallado mediante el algoritmo explicado en el capítulo 3. En la figura 4.9 se muestran dichas comparaciones por zona de estudio. Tal como se puede apreciar en la misma, las mediciones realizadas para las zonas de Malvín y Palermo son relativamente similares, mientras que las realizadas en la zona de Playa Hermosa difieren significativamente. Una posible explicación para justificar el hecho de que siempre los RTT hallados por el algoritmo hayan sido mayores que mediante el método `Syn Ack` es que, para éste último, se utilizan para efectuar la me-

Tipo de calificación	RTT hallado (ms)
`Buena`	< 404
`Mala`	> 1536
`Normal`	entre 404 y 1536

Tabla 4.2: Calificaciones obtenidas en comparación al histórico global según el resultado obtenido del RTT.

dición los paquetes del establecimiento de conexión de TCP, los cuales tienen un tamaño ligeramente menor a los utilizados para medir el RTT mediante el algoritmo. Otro factor que puede llegar a afectar es que el método `Syn Ack` halla el RTT de manera aproximada siempre que se cumplan con las hipótesis mencionadas en 4.3.4.

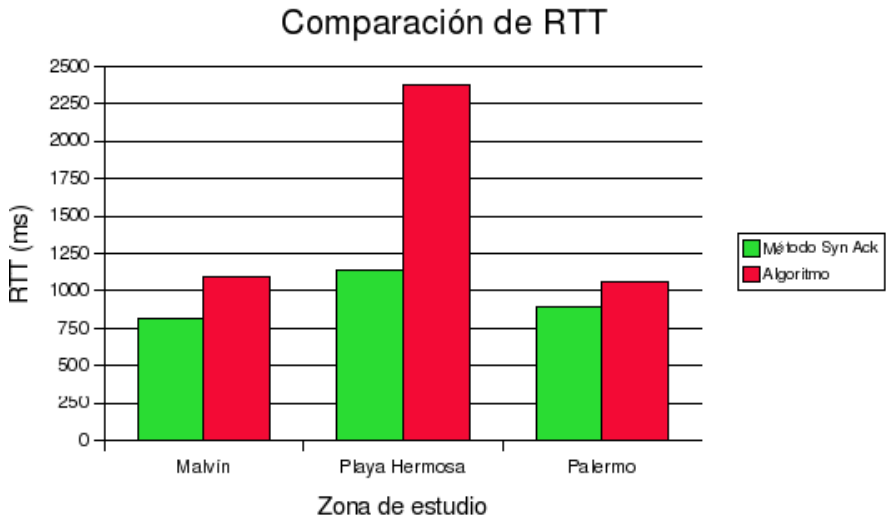


Figura 4.9: Comparación de los RTT hallados mediante el método `Syn Ack` y el algoritmo desarrollado.

A nivel global, la media y desviación estandar del RTT se encuentran en el entorno de los 970 y 566 `ms` respectivamente, por lo que, al realizar una prueba se la calificará como buena, mala o normal según la tabla 4.2.

Throughput

En la figura 4.10 se puede apreciar el throughput medio de las conexiones TCP ocurridas en cada prueba realizada. En la misma se puede observar que si bien se registran valores de hasta 60 `kbps`, la mayor parte de los registros se encuentran en la franja comprendida entre los 2 y 15 `kbps`. Vale la pena recalcar que el throughput hallado viene modelado por el protocolo TCP,

Tipo de calificación	Throughput hallado (kbps)
`Buena`	> 16
`Mala`	< 2
`Normal`	entre 2 y 16

Tabla 4.3: Calificaciones obtenidas en comparación al histórico global según el resultado obtenido del throughput.

por lo que el mismo no tiene porqué registrar valores altos en relación a la capacidad teórica del canal sino que es un indicador de la velociadad media a la cual se transfieren los datos mediante el mencionado protocolo, el cual como es sabido no explota al máximo los recursos del canal una vez establecida la conexión. De aquí que los valores hallados sean relativamente bajos en comparación con la máxima capacidad teórica disponible en el canal.

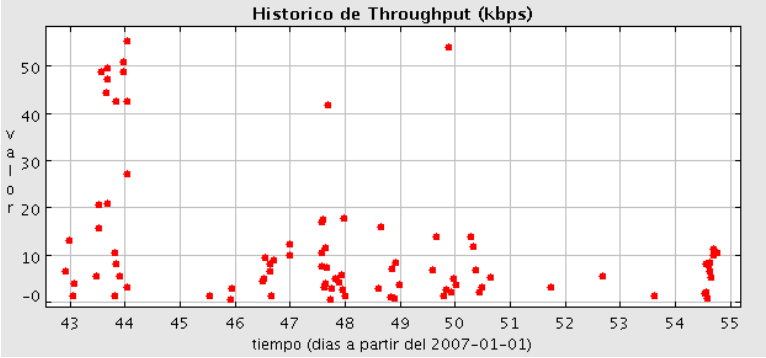


Figura 4.10: Throughput obtenido mediante la realización de las pruebas efectuadas.

A nivel global, la media y desviación estandar del `throughput` se encuentran en el entorno de los 9 y 7 `kbps` respectivamente, por lo que, al realizar una prueba se la calificará como buena, mala o normal según la tabla 4.3.

Retransmisiones

Las retransmisiones ocurridas en la conexión son un indicador muy importante del estado de la misma. Ante una mayor cantidad de retransmisiones, mayor es la pérdida de paquetes en la conexión, por lo tanto para transmitir una cantidad de información dada se requiere más tiempo y mayor cantidad de datos transferidos. En el momento de realizar los experimentos se comprobaron estos conceptos notándose que cuanto peor era la calidad de la conexión mas retransmisiones registraba el sistema de medición. Experi-

mentalmente dicho fenómeno se percibe claramente al momento de tratar de bajar imágenes, situación que en los peores casos se vuelve bastante tediosa. En las figuras 4.11 y 4.12 se muestran las retransmisiones efectuadas tanto por el cliente como por el servidor. En las mismas se puede observar que en la gran mayoría de las pruebas no existen retransmisiones de paquetes de ninguna parte involucrada en la conexión. Sin embargo, existen algunas pruebas cuyo porcentaje de retransmisiones fue considerable (superior al 10 %), lo que representan conexiones con problemas severos de comunicación.

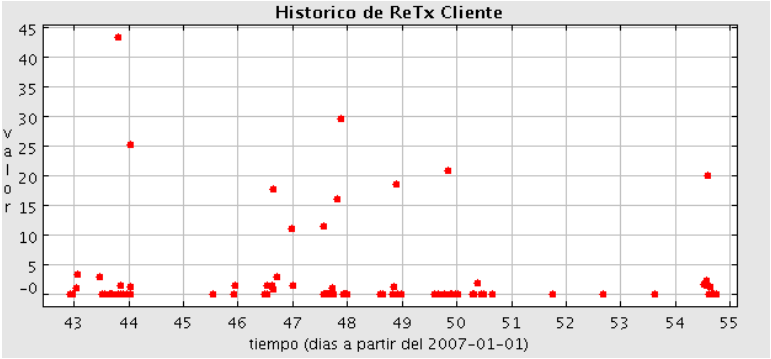


Figura 4.11: Porcentaje de retransmisiones de paquetes por parte del cliente.

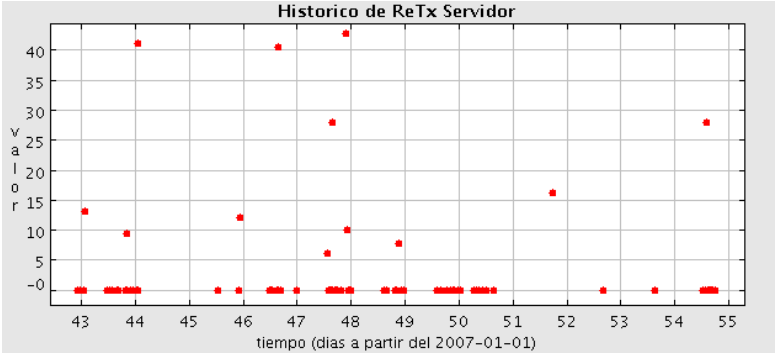


Figura 4.12: Porcentaje de retransmisiones de paquetes por parte del servidor.

4.4.3. Interpretación de los resultados

Luego de haber realizado una gran cantidad de pruebas bajo las distintas condiciones mencionadas anteriormente y relacionando los resultados obtenidos con la calidad percibida del enlace al recorrer el portal de pruebas se observó que los parámetros más significativos para describir la calidad de la conexión son el `RTT`, las `retransmisiones de paquetes` y el `throughput`

de la conexión.

4.5. Conclusiones

Se desarrolló una herramienta que estima la performance de WAP sobre el protocolo GPRS-EDGE que brinda al usuario una idea acerca del estado actual de su conexión en función de comparaciones con resultados previos tanto de dicho usuario como con la totalidad de usuarios del sistema.

La presencia del proxy WAP introduce incertidumbre a la hora de capturar tráfico en un servidor ubicado en Internet. Es por esto que se desarrolló un módulo de la aplicación pensado para correr en la misma LAN que el proxy WAP, el cual no pudo ser probado ya que no se tiene acceso a dicha LAN.

Por otro lado pese a la incertidumbre mencionada se piensa que los resultados obtenidos con el sistema de captura en un servidor en Internet son buenos de acuero a lo esperado.

4.6. Trabajo a Futuro

En la estimación de calidad de la navegación por medio del protocolo WAP, se realizó una estimación del estado de la conexión actual en función de promedios estadísticos de pruebas realizadas previamente.

De todas formas se entiende que la comparación con valores históricos no es la mejor solución ya que en el caso que los resultados son malos desde el punto de vista de un usuario tipo que utiliza el sistema, pero los resultados históricos son igualmente malos, la conexión va a valorarse como normal, o incluso como buena en un caso extremo. Con esto se quiere decir que porque la prueba actual haya dado mejor o peor que el promedio de las pruebas históricas no necesariamente se puede afirmar que la prueba actual es buena, normal o mala.

La solución a este problema es que, por medio de un sistema de aprendizaje y la valoración de los usuarios (MOS[11]) se pueda entrenar al sistema para que aprenda cuando una conexión es verdaderamente buena, normal o mala.

Esta solución no se llevó a la práctica ya que de hacerlo se traspasarían los plazos del proyecto.

[11] Mean Opinion Score

Parte III

Descripción de las Herramientas de Software

Capítulo 5

Descripción del Sistema

5.1. Generalidades

A lo largo del proyecto se fueron implementando aplicaciones que permiten monitorear el estado de conexiones físicas como ser enlaces GPRS-EDGE, conexiones ADSL, vía módem V.92, etc. y la performance de protocolos como TCP adaptado para WAP sobre GPRS. Para llevar esto a cabo fue necesaria la implementación de un sistema de comunicación del tipo servidor/cliente.

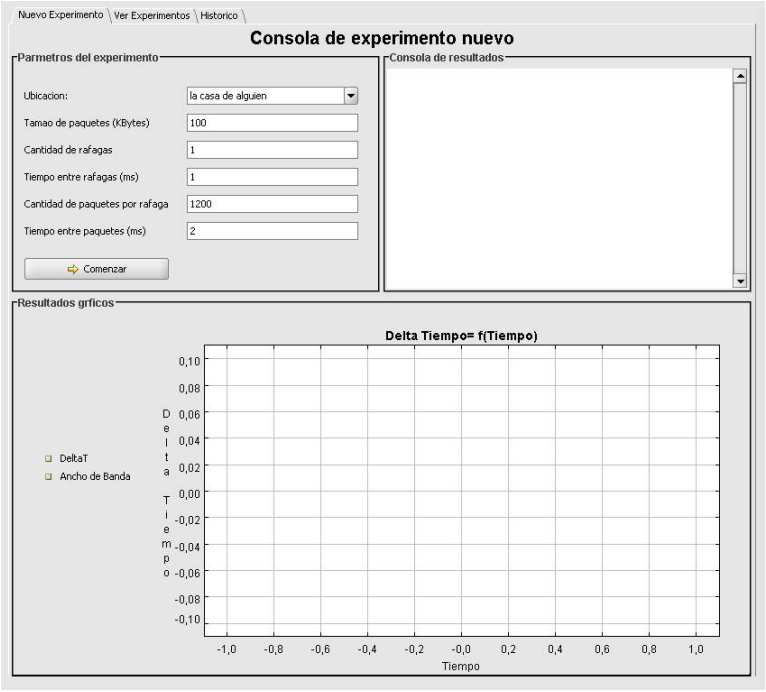
Era restricción del proyecto que el servidor fuera un Tomcat corriendo sobre una plataforma Linux (Mandrake 10.0), en la cual se encontraba en funcionamiento la versión anterior del software de *MetroNet*. También era restricción del proyecto que todo el sistema debía tener una cierta coherencia con lo ya implementado, donde se utilizarían las mismas bases de datos, se mantendría la estructura básica del software y se reutilizaría todo lo que fuera necesario. En este sentido, en el servidor hay un servlet escrito en JAVA que es el encargado de recibir órdenes del cliente y ejecutar todas las aplicaciónes que corran en el servidor, el cual se ha ido adaptando a las necesidades de las nuevas aplicaciones.

Del lado del cliente se abre un abanico de opciones diferentes dependiendo de qué aplicación se trata y a quién esta orientada. Desde el punto de vista de un operador de red, todas las aplicaciones se implementaron por medio de un *Applet* escrito en JAVA, el cual es una aplicación que corre sobre un navegador WEB en el cliente, como ser Internet Explorer o Firefox entre otros, la cual se descarga del servidor cada vez que se requiere su uso. Es por esto que el cliente debe tener la JVM[1] instalada en su equipo. Dado que se descarga cada vez que se requiere su uso, es deseable que no sea una aplicación muy pesada tanto desde el punto de vista del tamaño del archivo a descargar como de recursos que consume al ejecutarse. Es por esto que

[1] Java Virtual Machine

el *Applet* solamente hace de interfaz gráfica entre el cliente y el sistema. El grueso de los cálculos se realizan en el servidor para no sobrecargar el *Applet*, implementando en el mismo lo mínimo imprescindible para que el sistema funcione de manera eficiente.

En el caso de un usuario que desea monitoriar el estado de su conexión (ej. ADSL), se mantuvo la filosofía del *Applet* como interfaz gráfica. Lo mismo se hizo para el caso de un usuario que desea testear la calidad de su enlace GPRS utilizando un modem GPRS-EDGE o un teléfono móvil como módem conectado al PC. En la figura 5.1 se observa la interfaz gráfica para el caso de la aplicación para estimar la performance de GPRS.



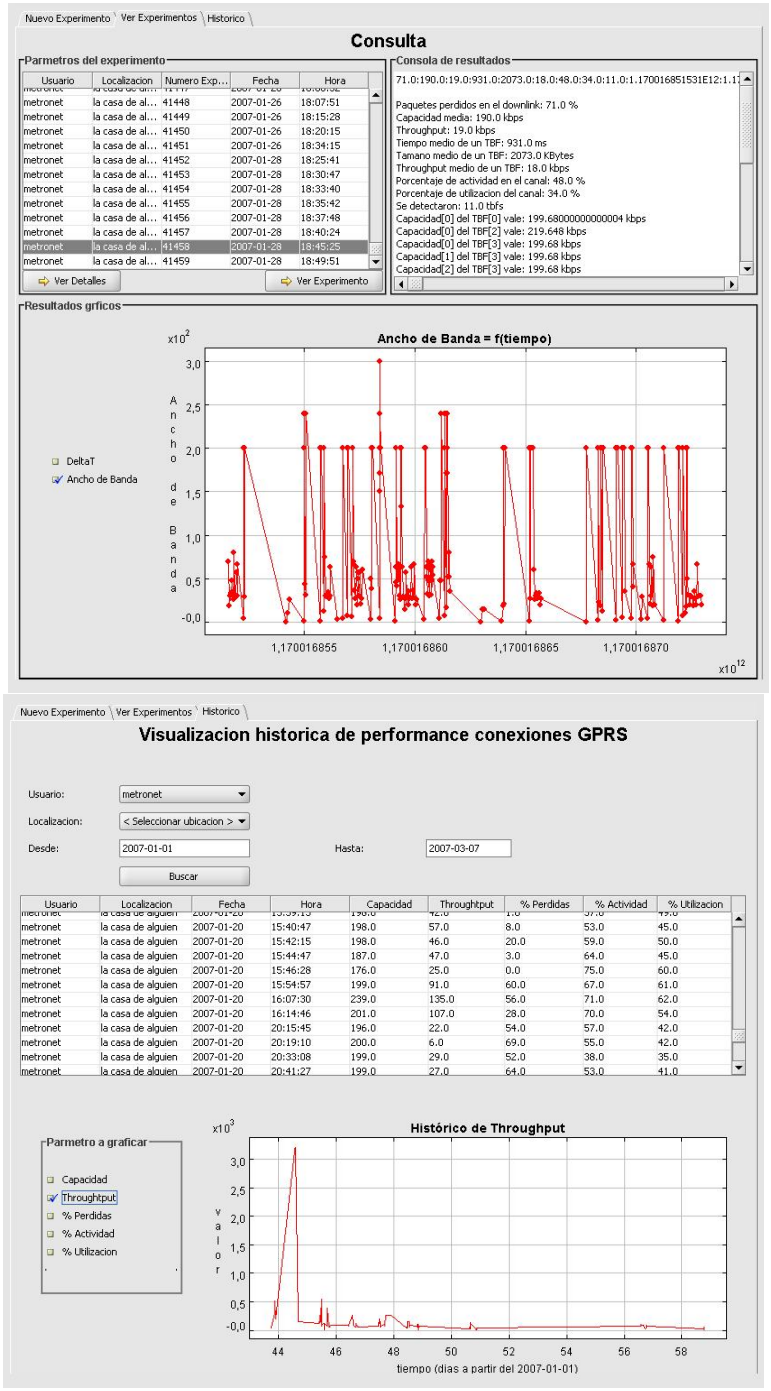


Figura 5.1: *Applet* del sistema para la estimación de performance en GPRS

Para el caso del usuario que desea monitoriar la calidad de su enlace

GPRS a través de su dispositivo móvil, se implementó un *Midlet*, el cual es un programa escrito en la version MicroEdition de JAVA que permite la comunicación con el servlet de manera similar a como lo hace el *Applet*. Si bien se mantuvo el lenguaje de programación, la versión MicroEdition de JAVA es bastante mas limitada que la edición estandar, lo cual trajo algunas complicaciones extra. Asimismo, dada la capacidad limitada que tiene un dispositivo donde corre esta aplicación fue necesario limitar al máximo las instrucciónes que corren en el *Midlet*. En la figura 5.2 se muestra el *Midlet* implementado corriendo en el Emulador de celulares JAVA.

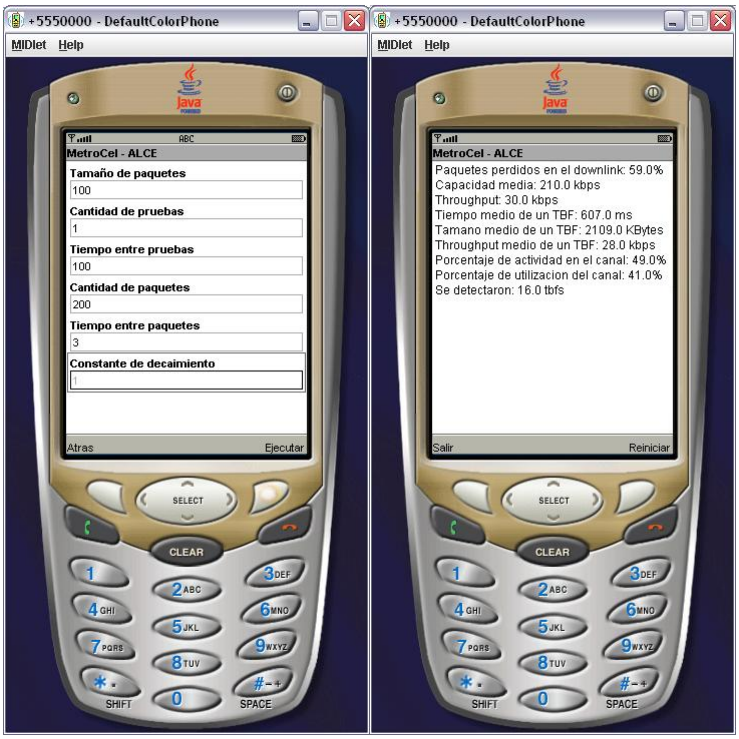


Figura 5.2: *Midlet* para la estimación de performance de WAP

Finalmente se implementó un portal WAP escrito en codigo WML[2], el cual es una implementación del estandar XML[3]. Una vez que se ingresa al sitio, el sistema comienza a capturar el intercambio de paquetes entre el cliente y el servidor donde se encuentra el portal. El sitio consiste en una sucesión de páginas WML las cuales el usuario va recorriendo con el fin de generar tráfico para poder realizar los cálculos posteriores, en donde en la última página se devuelve un resumen de los resultados hallados en la prueba conjuntamente con una valoración de la performance de TCP. En la figura 5.3 se muestra

[2] WML: Wireless Markup Language
[3] XML: eXtensible Markup Language

la pantalla de bienvenida al portal WAP implementado corriendo en un navegador emulador de celulares.

Figura 5.3: *Midlet* para la estimación de performance de WAP

5.2. Sincronización Entre el Servidor y el Cliente

A lo largo de las pruebas realizadas hay una contínua comunicación entre el servidor y el cliente. Si bien el servidor es el encargado de efectuar los cálculos necesarios para poder realizar una estimación de lo que se quiera medir, es el cliente quien por medio de intercambio de información le da la orden de qué paso seguir en cada instancia. El protocolo elegido para el intercambio de información entre el servidor y el cliente en todos los casos es HTTP[4].

5.3. Paquetes de Prueba

Los paquetes de prueba para los sistemas que requieren la inyección de tráfico para realizar la estimación de calidad son paquetes UDP. Dentro de

[4]HyprText Transfer Protocol

los paquetes UDP se imprime información útil como ser el número de paquete para poder llevar un control sobre las perdidas, un número de ráfaga por si los paquetes se envían en ráfagas, y los timestamps correspondientes a los tiempos de partidas y llegadas de los paquetes.

5.4. Comunicación Desde Redes Fijas

Para generar el tráfico UDP la edición estandar de JAVA ofrece, dentro de sus librerías, la clase `DatagramSocket`. Esta implementación tiene la importante ventaja de que es posible controlar los puertos origen y destino de los paquetes enviados, lo cual es imprescindible para que la comunicación sea posible.

Dinámica de intercambio de paquetes

Dado que en este tipo de enlaces no se producen pérdidas a no ser por saturación, se envían los paquetes en ráfagas a una tasa mayor que la capacidad del canal como para poder saturar el enlace, y con un tiempo entre rafagas lo suficientemente grande como para permitir la descongestión del canal. Una vez que llegan los paquetes se les imprime una marca de tiempo correspondiente a la llegada del cliente y son rebotados. En el servidor se recibe el paquete rebotado y se guarda la información de cada paquete para el futuro procesamiento.

5.5. Comunicación a Través de la Red Celular

Las conexiones celulares hoy en día toman una IP de un rango privado y hacen NAT[5] con el propósito de salir a la red pública (Internet). Por este motivo no se puede establecer la comunicación del mismo modo que se hace para redes fijas, ya que la IP destino de los paquetes UDP que salen del servidor no pueden ser de un rango privado. Por este motivo no se pueden enviar los paquetes desde el servidor directamente hacia la IP del móvil, por lo que los mismos deben ser enviados hacia la IP del router que hace NAT y buscar la forma de que este equipo redireccione los paquetes hacia el móvil en cuestión.

La solución encontrada inicialmente fue la de iniciar la comunicación por parte del cliente controlando tanto el puerto de origen como el de destino, de manera de realizar el envío de paquetes desde el servidor utilizando dichos puertos. La dinámica de dicha comunicación se describe como sigue

[5]NAT: Network Address Traslation

1. Mediante una negociación previa se acuerda la pareja de puertos X e Y a utilizarse durante la comunicación entre ambas partes
2. El cliente envía paquetes de prueba hacia el servidor con puerto origen X y puerto destino Y con el fin de crear una entrada en memoria en la tabla de NAT
3. El servidor inicia la prueba utilzando como puerto origen Y y como puerto destino X de modo tal que el router que realiza el NAT sepa rutear adecuadamente los paquetes hacia el cliente

Sin embargo, en el caso de los paquetes de prueba UDP, esto no pudo ser implementado de dicha manera por dos restricciones existentes:

- En el caso de acceder desde el *Applet*, la utilización de `DatagramSockets` solo permite generar una instancia de paquete por puerto, entonces no se puede enviar y recibir en el mismo puerto porque se requieren dos instancias diferentes.
- En el caso de acceder desde el *Midlet*, es necesario tener en cuenta que en la versión Micro Edition de JAVA no cuenta con la clase `DatagramSocket`, por lo que fue necesario utilizar la clase `Datagram`. La misma presenta la desventaja con respecto a la anterior que no permite elegir ni conocer el puerto origen de la transmisión de paquetes.

Lo que se hizo entonces fue comenzar desde el cliente el envío de unos paquetes de entrenamiento con el puerto al cual se van a recibir posteriormente los paquetes de prueba, de manera que quede vinculado en la tabla de NAT del router en cuestión el puerto utilizado con el dispositivo móvil. En el servidor se capturan estos paquetes y se observan de qué puerto e IP vienen. Luego de la etapa de aprendizaje de los puertos a utilizar, se cierra en el móvil la instancia del paquete usada y se le consulta al servidor por uno de los puertos utilizados con el fin de poder saber en que puerto escuchar los paquetes provenientes del mismo. En el caso de la implementación por *Applet*, este último paso no se aplica ya que el puerto de envío se elije en el programa por lo que ya se tiene conocimiento de su valor. La dinámica de dicha comunicación se resume como sigue

1. Se negocia un número de puerto en el cual va a escuchar el servidor
2. Se envían datagramas de entrenamiento desde el cliente hacia el servidor en el puerto elegido en 1
3. El servidor recive dichos datagramas y determina el puerto origen del cual provienen
4. El servidor le comunica al cliente dicho puerto con el fin de que el mismo espere los paquetes de prueba en el puerto con dicho valor

5. Comienza la prueba

En la figura 5.4 se muestra un resumen de la dinámica del intercambio de paquetes a lo largo de una prueba para la estimación de performance en GPRS.

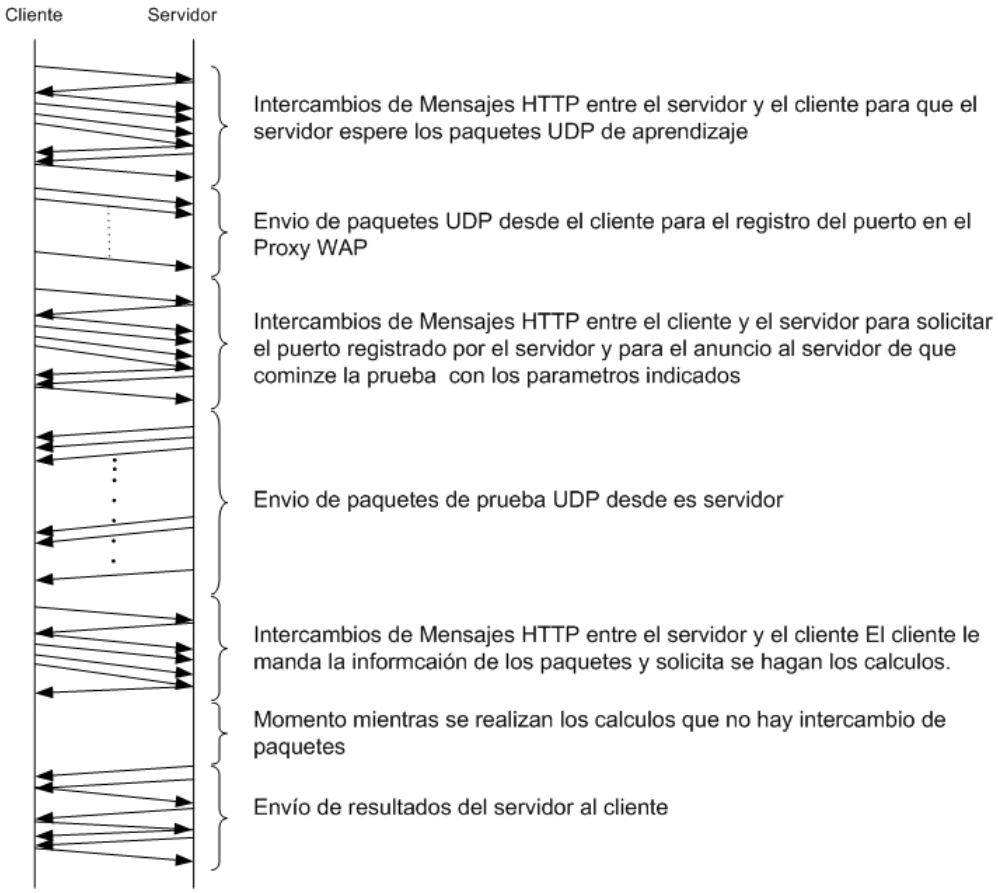


Figura 5.4: Evolución del sistema

Dinámica de intercambio de paquetes

Dado que los sistemas celulares presentan una gran cantidad de pérdidas en el acceso, la solución de rebotar paquetes hace que exista la posibilidad de perder información al momento de rebotar. Es por esto que para las aplicaciones que corren en el móvil en lugar de rebotar cada paquete se van guardando los datos de los paquetes que llegan, junto con las marcas de tiempo con que salen del servidor y con que llegan al cliente. Se manda toda la información junta concatenada al final de la prueba mediante la utilización del protocolo HTTP lo que asegura que la información llegue al servidor para poder guardar los datos y procesarlos después.

Capítulo 6

Ingeniería de Software

6.1. Introducción

Para el diseño y análisis de la arquitectura de software a implementar se tuvieron en cuenta tres requerimientos principales.

Adaptación al sistema ya existente

El software a desarrollar y por ende su arquitectura, debía acoplarse al software ya existente, perteneciente al proyecto *MetroNet*. La versión de *MetroNet* en producción aportaba documentación [8] acerca de sus casos de uso, del funcionamiento dinámico del sistema y de la estructura con la cual había sido implementado el software. Sin embargo, las clases no estaban organizadas en paquetes que reflejaran dicha estructura. Por lo tanto, como primer paso en el armado de la arquitectura del software, se consideraron las clases desarrolladas por *MetroNet* y se las organizó en paquetes, de acuerdo a la documentación existente.

Modelo de desarrollo y adaptación a cambios de requerimientos

Debían resolverse satisfactoriamente los requerimientos del nuevo software a desarrollar, y esto debía resolverse con una flexibilidad tal que permitiera cambios en el funcionamiento del sistema si durante el desarrollo del mismo ésto se evaluaba como necesario. Sobre este punto cabe recordar el modelo de espiral para desarrollo de software, el cual implica que se itere reiteradas veces por cada una de las etapas hasta llegar a una solución definitiva. Las etapas de desarrollo son:

1. Relevamiento de requerimientos: se busca establecer qué problemas debe resolver el software a desarrollar. Como consecuencia de este análisis se desprenden los casos de uso que se verán en la siguiente sección.

2. Análisis de funcionamiento: en esta etapa se modela una abstracción del sistema, buscando identificar los actores involucrados y elaborando un diagrama conceptual.

3. Diseño: Se elabora un modelo de paquetes y clases, implementando los resultados del análisis ya desarrollado.

4. Codificación: En esta etapa se implementa el diseño, codificando las clases y métodos.

5. Testeo: Se prueban los métodos implementados, verificando que el funcionamiento se corresponda con los casos de uso propuestos.

En el modelo de espiral, una vez cumplida la etapa de testeo de una iteración, se pasa nuevamente a la etapa de requerimientos, verificando cuáles de éstos se cumplen y si se hace como era de esperarse. En el presente desarrollo en particular el desafío era aún mayor, ya que los requerimientos cambiaban a medida que se avanzaba en el mismo.

Crecimiento futuro

Por último era necesario que tras dejar el software en producción, el mismo pudiera ser ampliado con facilidad. Para ello, a parte de dejar a punto toda la documentación correspondiente (parte de la cual se verá a continuación), es necesario que el diseño propuesto sea fácilmente adaptable o ampliable según nuevos requerimientos que surjan en el futuro.

6.2. Casos de Uso

En esta sección se describen los casos de uso que debe cumplir el sistema de acuerdo a los requerimientos de *MetroCel*. Los casos de uso exclusivos del funcionamiento del software de *MetroNet* no serán cubiertos por estar ya incluídos en su documentación.

Se identifican como actores externos al sistema los usuarios, que hacen las pruebas de evaluación de performance para evaluar sus propios enlaces, y los operarios de la red del ISP, que hacen o recuperan evaluaciones de performance para monitorear el funcionamiento de la red.

Además se encuentran dos lugares bien distintos en donde se ejecuta el software desarrollado: en un servidor de Internet, y en un servidor en el núcleo de la red del ISP.

6.2.1. Validar usuario

- Descrpción: Permite que un usuario se valide contra el sistema.

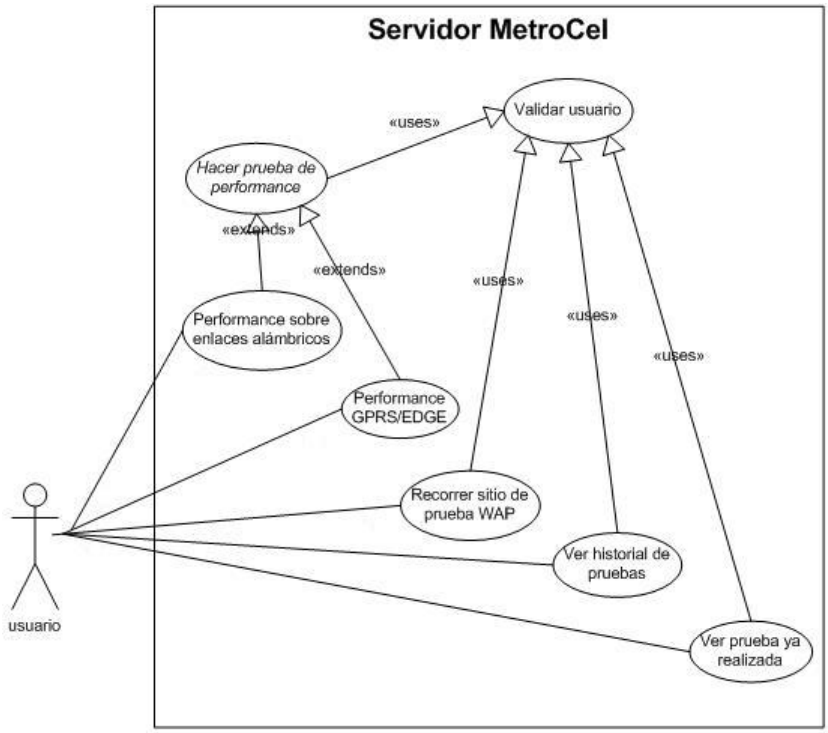


Figura 6.1: Diagrama de casos de uso a ejecutarse en el servidor de MetroCel.

- Actores: Usuario y sistema
- Flujo normal de eventos:
  1. El usuario ingresa nombre y contraseña.
  2. El sistema verifica que los datos sean correctos.
  3. Si el sistema valida los datos el usuario ingresa al sistema.
- Flujo alternativo:

  3b. Si los datos ingresados no son válidos, se muestra un mensaje de error.
- Postcondiciones: El usuario ingresa al sistema.

6.2.2. Hacer prueba sobre redes fijas

- Descripción: Permite al usuario realizar una prueba de performance sobre redes fijas.
- Actores: Usuario y sistema

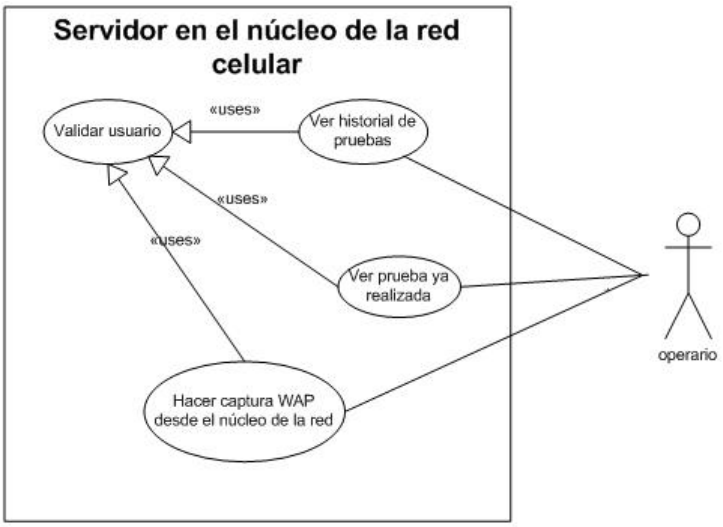


Figura 6.2: Diagrama de casos de uso a ejecutarse en un servidor de la red del ISP.

- Precondiciones: El usuario debe haber sido validado por el sistema.
- Flujo normal de eventos:
  1. El usuario elige el tipo de prueba a realizar.
  2. El sistema muestra las distintas localizaciones registradas por el usuario.
  3. El usuario selecciona su localización actual y el tipo de conexión.
  4. El usuario selecciona comenzar la prueba.
  5. El sistema realiza la prueba seleccionada y muestra los resultados obtenidos.
- Flujo alternativo:
  4'. El usuario selecciona opciones avanzadas
  5'. El sistema despliega una lista de parámetros a determinar.
  6'. El usuario ingresa los parámetros correspondientes.
  7'. Vuelve al punto 4 del flujo normal.
- Postcondiciones: El usuario obtiene un resultado que caracteriza su enlace.

6.2.3. Recorrer sitio de prueba WAP

- Descripción: El usuario recorre un sitio de prueba y éste le indica la performance de su conexión WAP.
- Actores: Usuario y sistema
- Precondiciones: El usuario debe haber sido validado por el sistema.
- Flujo normal de eventos:
  1. El usuario recorre el sitio de prueba WAP
  2. El sistema muestra como resultado en la última página las características de la conexión WAP que estableció el usuario
- Postcondiciones: El usuario obtiene un resultado que caracteriza la performance de su conexión WAP.

6.2.4. Hacer captura WAP desde el núcleo de la red

- Descripción: Permite al operario de la red obtener capturas del tráfico WAP, designando los filtros que desee.
- Actores: Operario de la red y sistema.
- Precondiciones: El operario debe haber sido validado por el sistema.
- Flujo normal de eventos:
  1. El operario selecciona su ubicación y opcionalmente puede elegir los filtros que desee.
  2. El sistema valida los filtros elegidos y en caso de ser correctos comienza la captura.
  3. Una vez finalizada la captura, el sistema muestra los resultados obtenidos.
- Flujo alternativo:
  2b. En caso de no ser válidos los filtros elegidos, el sistema muestra un mensaje de error.
- Postcondiciones: El operario obtiene un informe sobre las capturas realizadas.

6.2.5. Hacer prueba para evaluar la performance de enlaces GPRS-EDGE

- Descripción: Permite al usuario obtener informe sobre las estadísticas de su enlace GPRS o EDGE.
- Actores: Usuario y sistema.
- Precondiciones: El usuario debe haber sido validado por el sistema.
- Flujo normal de eventos:
  1. El usuario selecciona su ubicación y elige los parámetros de la prueba.
  2. El usuario solicita al sistema comenzar la prueba.
  3. Al terminar la prueba, el sistema muestra los resultados obtenidos.
- Postcondiciones: El usuario obtiene estadísticas sobre su enlace.

6.2.6. Ver historial de pruebas

- Descripción: Permite al operario obtener estadísticas de las pruebas hechas con anterioridad.
- Actores: Opeario de la red y sistema.
- Precondiciones: El operario debe haber sido validado por el sistema.
- Flujo normal de eventos:
  1. El operario selecciona el usuario de quién quiere ver la prueba o la localización que quiere estudiar.
  2. Si seleccionó un usuario, el sistema muestra las localiaciones disponibles para el mismo, en caso de haber seleccionado una localización, el sistema muestra los usuarios que hicieron pruebas en ella.
  3. El operario selecciona el parámetro restante, además puede seleccionar si lo desea fecha de inicio y fin del historial a visualizar.
  4. El sistema lista los experimentos que cumplen con las características especificadas.
  5. El operario selecciona el parámetro a graficar.
  6. El sistema muestra en forma gráfica la evolución del parámetro seleccionado.
- Postcondiciones: El operario obtiene la evolución de los parámetros mas destacados de la red.

6.2.7. Ver prueba ya realizada

- Descripción: Permite al operario de la red ver en detalle una prueba hecha con anterioridad.
- Actores: Opeario de la red y sistema.
- Precondiciones: El operario debe haber sido validado por el sistema.
- Flujo normal de eventos:
  1. El sistema lista los experimentos existentes.
  2. El operario selecciona el experimento que desea observar.
  3. El sistema muestra los resultados del experimento seleccionado.
  4. El operario puede seleccionar un parámetro para graficar.
  5. En ese caso, el sistema grafica el parámetro seleccionado.
- Postcondiciones: El operario obtiene el resultado de la prueba seleccionada.

6.3. Análisis y Diseño

En esta sección se describe la arquitectura del software implementado, mencionando su planificación y sus principales paquetes. Además se describe en detalle la composición de los módulos más importantes.

6.3.1. Arquitectura general del sistema

Para la arquitectura del sistema se hizo inicialmente una separación por capas: *Cliente*, *Negocio* y *Datos*.

La capa *Cliente* incluye todos los módulos que interactúan con el usuario, así como tambien aquellas clases que deben ejecutarse en el terminal del cliente.

La capa *Negocio* incluye todas las clases y módulos que tienen la función de resolver la escencia del problema.

La capa *Datos* tiene la función de interactuar con todos los medios de persistencia de datos que vayan a utilizarse, por ejemplo bases de datos, archivos de texto plano o XML.

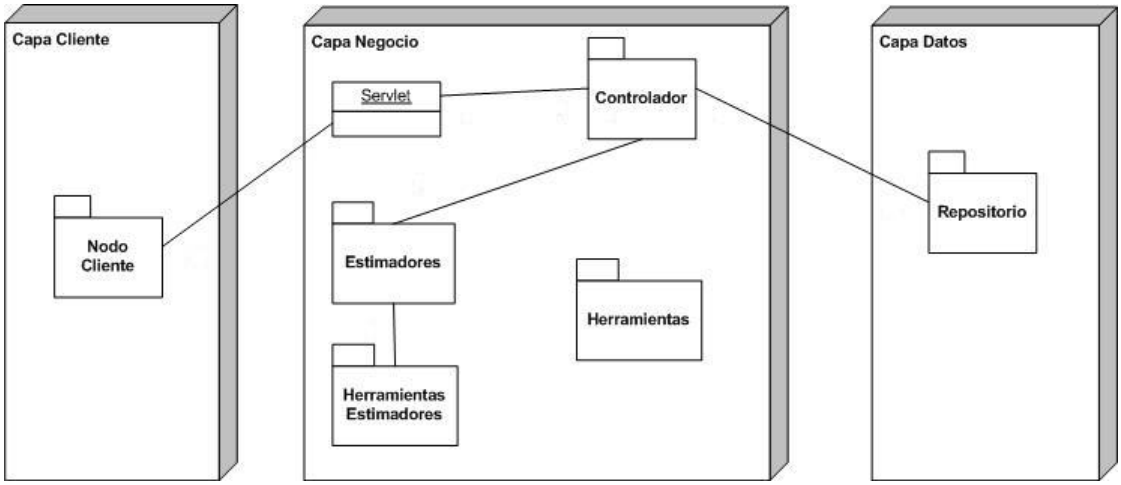


Figura 6.3: Arquitectura de MetroCel.

6.3.2. Capa Cliente

La capa *Cliente* consiste en las distintas interfaces que se le presentan al usuario para utilizar las aplicaciones del sistema. Dichas aplicaciones estarán diferenciadas segun funcionalidad: prueba sobre redes fijas, captura de tráfico WAP y evaluación de enlaces GPRS-EDGE.

Las dos alternativas más evidentes que surgían en esta etapa eran elaborar dichas interfaces utilizando JSP[1] o mediante el empleo de *Applets*. Dado que además de la interacción con el servidor era necesario que en el terminal del cliente se ejecutaran operaciones como el rebote de paquetes UDP, se optó por la segunda opción planteada, ya que es la tecnología que permite implementar dichos requerimientos. Además es necesario tener en cuenta que en el software desarrollado por *MetroNet* implementaba esto, por lo que desarrollar la misma tecnología favorecía la compatibilidad.

Pruebas en redes fijas

Fue ampliada la clase *IFExperimento* que había sido desarrollada por *MetroNet*, para que soportara los nuevos algoritmos. Además se utiliza la clase *Rebotador*, que tiene la función de rebotar los paquetes UDP que llegan al terminal del cliente.

[1] Java Server Pages

Captura de tráfico WAP

Para esta funcionalidad se creó un *Applet* con la clase *IFCapturaWap*, la cual cumple todas las funcionalidades relacionadas con el estudio de performance WAP: captura de tráfico, estudio de historial de resultados, y observación de las pruebas hechas con anterioridad.

Evaluación de enlaces GPRS-EDGE

La clase *IFGPRSPerform* implementa un *Applet* que cubre las funcionalidades ofrecidas para el estudio de performance sobre este tipo de enlaces. Además se utiliza la clase *Rebotador*, que cumple la misma funcionalidad que en el caso de las redes fijas. Este *Applet* ofrece la posibilidad de realizar una prueba de evaluación de performance cuando se utiliza un celular como módem. Además se puede, como en el caso anterior, visualizar pruebas hechas con anterioridad o estudiar la evolución histórica de resultados.

Otra opción para esta funcionalidad es la de utilizar un *Midlet*, desarrollado para que desde aquellos celulares que soporten JAVA (J2ME) se puedan realizar las pruebas sin necesidad de conectar el móvil a una computadora. En el desarrollo del *Midlet* se creó un archivo *jar* que concentra las clases utilizadas: *IFMovil*, *RebotadorME* y *EnviadorME*

6.3.3. Capa Negocio

En esta capa se concentra el núcleo de la aplicación. En la figura 6.3 se pueden observar los principales paquetes de la misma. Inicialmente se planteó una división en paquetes dentro de la capa, definiéndolos segun función y operaciones que se realizan.

El Servlet tiene la función de servir como nexo de comunicación entre las capas de *Cliente* y *Negocio*, cumpliendo un rol de *fachada*, ya que es la clase que ofrece todas las funcionalidades disponibles en la capa.

El paquete *Controlador* tiene la función de coordinar todo el funcionamiento del sistema. Se encarga de la comunicación entre todas las capas y utiliza los demás paquetes de su capa para resolver los pedidos de la capa *Cliente*. El objetivo de este paquete es el de centralizar el control de la aplicación.

En el paquete *Estimadores* se encuentra la lógica para realizar las operaciones necesarias que permiten calcular los parámetros deseados. Este paquete a su vez se divide (como será explicado más adelante) en distintos módulos según las características de los parámetros que se desean medir. La funcionalidad central de este paquete es la de aplicar algoritmos sobre las

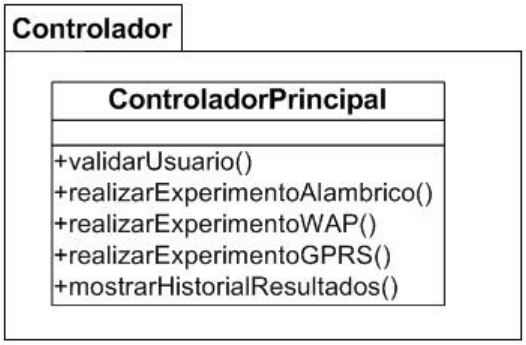


Figura 6.4: Paquete Controlador.

muestras que se hayan obtenido.

En el paquete *Herramientas Estimadores* se pueden encontrar todas aquellas operaciones que no se refieran al cálculo en si de los parámetros, pero que si son necesarios para realizar la pruebas adecuadas. Incluye las funciones de generación de tráfico sobre la red y de modelado de datos.

El paquete *Herramientas*, si bien en la figura 6.3 no se muestra vinculado con ningún otro paquete, sirve a todo el resto de la aplicación, ya que su función es la de proveer métodos y campos que puedan ser utilizados por clases de otros paquetes.

Paquete Controlador

Se implementó una única clase con los métodos necesarios para invocar todas las operaciones del sistema. Presenta diversos constructores que inicializan la clase de modo tal que se pueda especializar en cierta funcionalidad.

A efectos de tener una correcta coordinación entre capas, se buscó por un lado que ésta clase provea los métodos que necesita invocar el Servlet y por otro lado se propuso que sea la única que genere interacciones con la capa *Datos*, de modo tal de hacer un uso correcto y ordenado de la base de datos.

Paquete Estimadores

Este paquete se encuentra dividido en tres módulos, uno para cada tipo de estimación que se puede realizar por la aplicación. El módulo *Calidad de servicio* no fue desarrollado en esta etapa, sino que fue heredado del *software* original (*MetroNet*). Provee herramientas que permiten estimar la calidad

de servicio percibida por el usuario al ver contenido multimedia por Internet (por ejemplo video). El módulo *Caracterśticas de enlace* provee herramientas para calcular parámetros objetivos de enlaces, como por ejemplo capacidad o ancho de banda. Aquí se concentran las herramientas desarrolladas para estimaciones sobre redes fijas y sobre enlaces GPRS-EDGE. En el módulo WAP residen las herramientas de estimación de performance de dicho protocolo.

Cada uno de los módulos tienen implementando un controlador, que coordina el funcionamiento del mismo y ofrece las funcionalidades que pueden ser provistas por él.

Es pertinente hacer una distinción entre el módulo WAP y el resto de los contenidos en el paquete. El módulo *Calidad de servicio* fue desarrollado completamente por *MetroNet*, a la vez que en el módulo *Características de enlace* residen clases desarrolladas tanto por *MetroNet* como por *MetroCel*. Es por este motivo que para mantener una compatibilidad se modeló la estructura de clases acorde a lo desarrollado por *MetroNet*, respetando tanto las herencias, los parámetros utilizados para citar métodos y el esquema de funcionamiento de los módulos. En el caso de WAP en cambio, se optó por proponer una estructura distinta. Los factores que impulsaron a dicha propuesta fueron los siguientes

- Las pruebas a realizar en el módulo WAP serían de características marcadamente diferentes a las realizadas en los otros módulos. Esto se ve reflejado en la forma de generar tráfico, en la forma de capturarlo y el tipo de tráfico que se genera[2].
- Se buscaba favorecer la reutilización en otros módulos del código generado, por lo que se buscó que las clases desarrolladas fueran lo más genéricas posibles. Por este motivo se implementó un módulo TCP (será descripto en detalle en la sección 6.3.3) cuyo funcionamiento no se limita a las conexiones WAP.

De este modo se puede observar que aunque el módulo WAP mantiene similitudes en su arquitectura, gran parte de las operaciones necesarias para estimar la performance de dicho protocolo se encuentran en el módulo TCP.

- **Calidad de servicio**: Este módulo fue implementado por el proyecto *MetroNet*, por lo que no se entrará en detalle sobre su funcionamiento.
- **Caracterśticas de enlace**: Este módulo tiene como controlador a la clase *MedioActual* y posee una colección de estimadores, que son capaces de calcular diversos tipos de parámetros de la red. En el este diseño

[2]Recordar que para el caso de WAP, los paquetes que se intercambian son TCP, resultantes de una conexión generada por un browser externo a la aplicación

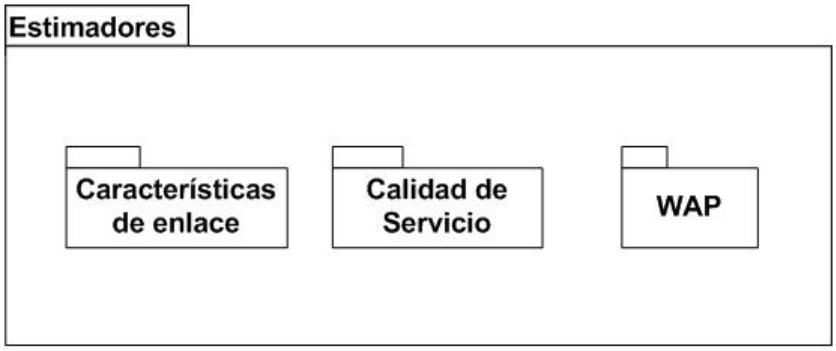


Figura 6.5: Paquete Estimadores.

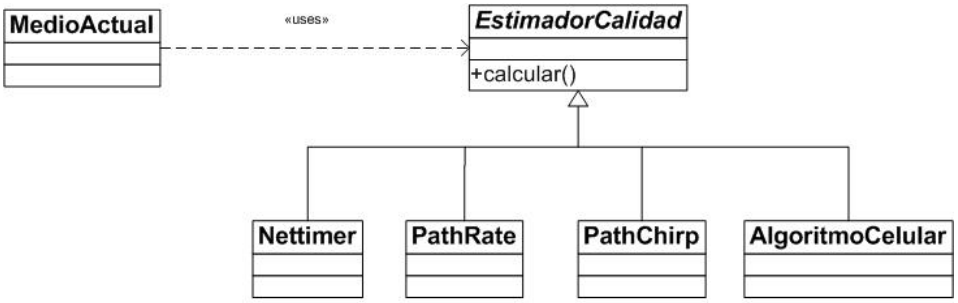


Figura 6.6: Modulo características de enlace.

se presentan como estimadores de enlaces en arquitectura de redes fijas implementaciones propias de los algoritmos *Nettimer*, *Pathrate* y *Pathchirp*, cada uno de los cuales se representa con una clase. Para estimar performance sobre enlaces GPRS-EDGE se diseñó un algoritmo, el cual también se representa por una clase. Todas estas clases heredan de una clase abstracta llamada *EstimadorCalidad*, de la cual deberán heredar todos los estimadores que se agregen en un futuro. La figura 6.6 muestra las clases de este módulo.

- **wap**: El controlador de ese módulo es la clase *MedioWap*, el cual utiliza la clase *TrazaTCP* para estimar la performance. Este módulo está relacionado con el módulo *TCP* del paquete *Herramientas estimadores*. En la figura 6.7 puede observarse un esquema del mismo.

Paquete Herramientas Estimadores

Este paquete proporciona las herramientas necesarias por los estimadores para poder hacer las pruebas que éstos requieren y modelar los datos en

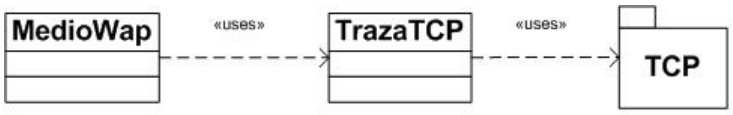


Figura 6.7: Modulo WAP.

forma adecuada. Se distinguen cuatro grandes módulos: *Envío tráfico* tiene la finalidad de generar el tráfico de prueba, con las características requieridas por el estimador que vaya a utilizarse. El módulo *Captura tráfico* contiene las clases necesarias para realizar la captura de dicho tráfico, y extraer de él la información necesaria. El módulo *Resultado* está diseñado para representar unidades de datos que puedan ser facilmente transportables por la aplicación, en particular a la hora de persistir datos. Finalmente el módulo TCP tiene las clases correspondientes para poder modelar dicho tipo de tráfico.

Nuevamente en este paquete puede apreciarse que existe una diferencia en la arquitectura implementada para las pruebas de performance de WAP con respecto a los demás tipos de prueba. En este caso se puede observar adicionalmente que al utilizar la librería *jpcap* para efectuar las capturas de tráfico TCP, fue necesario diseñar una estructura de clases que se adaptara a la misma. A continuación se expone el diseño implementado

- Inicialmente se creó la clase *Sniffer*, la cual estaría encargada de administrar las capturas de paquetes. Dado que una vez finalizada la prueba era necesario indicar que se finalizara la captura, se implementó el patrón *singleton*, el cual impone que exista una única instancia de la clase en la aplicación y ésta pueda ser invocada a través de un método estático. De este modo, la clase brinda la posibilidad de iniciar y terminar una captura en el momento que se desee.

- Los capturadores de tráfico debían correr en hilos independientes, para no interrumpir la ejecución de la aplicación mientras se realiza la captura. Pero por otra parte los mismos debían heredar de una clase de la librería *jpcap*, *PacketCapture*. La primer opción que se manejó fue la de hacer que las clases heredaran de dicha clase y que a su vez implementaran la interfaz *Runnable*. Sin embargo, dicha solución dificultaba la operación de interrumpir una captura en el momento que fuera necesario. Por este motivo se creó una clase *HiloCapturador*, que hereda de la clase *Thread*. La misma tiene una instancia de la clase *Capturador*, que hereda de *PacketCapture*. *Sniffer* contiene entonces una colección de *HiloCapturador* y a través de dicha clase es posible iniciar y terminar la captura realizada por *Capturador*.

- El capturador implementado genera un evento al recibir un paquete. Dicho evento es atendido en la clase *PacketHandler*, la cual implementa la interfaz *PacketListener*, que pertenece a la librería *jpcap*. A través del método *packetArrived* se define cómo procesar cada paquete que es capturado en el servidor. A su vez esta clase almacena todos los datos de las capturas.

Por otra parte, para conservar los datos necesarios hasta el final de la prueba WAP (por ejemplo nombre del usuario, localización), éstos se almacenan en el *query string* de cada página WML desplegada[3]. Se optó por éste método ya que es sencillo de implementar, no sobrecarga al servidor de información y no hay posibilidades que no funcione en el browser de un celular, como podría ser el caso de utilizar *cookies*.

A continuación se describen los módulos del paquete

- **Envío tráfico**: Dos tareas deben realizarse en este módulo, coordinar el comienzo del envío de paquetes con el receptor correspondiente y generar y enviar el tráfico adecuado. La primera de ellas está representada por la clase abstracta *EnvioTraficoPrueba* y la segunda por la clase abstracta *Generador*. De *EnvioTraficoPrueba* heredan dos clases, *PaquetesPrueba* que es invocada cuando la prueba a realizar es de caracterśticas de enlace y *MedioSimulado* que es utilizado para las pruebas de calidad de servicio implementadas por *MetroNet*. Análogamente, de *Generador* heredan las clases *GeneradorUDP* para pruebas de caracterśticas de enlace y *GeneradorMedio* para pruebas de calidad de servicio. Por otro lado la clase *GeneradorPaginaWap* sirve de repositorio de páginas WAP, para cuando se explora el sitio de prueba. En la figura 6.8 se puede observar un esquema de las clases del módulo y la relación que existe entre ellas.

- **Captura tráfico**: Para las pruebas de caracterśticas de enlace y de calidad de servicio, la recepción de paquetes se hace a través de la clase *Receptor*. En este módulo se incluyen también las clases que permiten la captura de los paquetes TCP, como ya fue explicado. En la figura 6.9 se observa un esquema del módulo.

- **Resultado**: A efectos de transportar los resultados obtenidos por la aplicación y en particular para poder persistirlos, se tiene la interfaz *IResultado*, que establece los métodos básicos que deben tener las clases que la implementen, a efectos de cumplir con dichos requerimientos. En la aplicación desarrollada se implementaron las clases *ResultadoWap*, encargado de transportar los resultados obtenidos en experimentos de

[3] *query string* es la porción de una URL que contiene parámetros de búsqueda cuando se intenta desplegar una página web dinámica.

performance WAP, la clase *ResultadoGPRS*, análogo utilizado cuando se trata de un experimento de performance sobre dicho tipo de enlaces y *ResultadoBasico*, que es la representación de resultados para el caso de pruebas en redes fijas. Se optó por mantener incambiada la clase *ResultadoQoS*, desarrollada en el proyecto *MetroNet*. En la figura 6.10 se observa un esquema de la implementación de este módulo.

- TCP: Las clases de este módulo sirven para modelar el tráfico TCP. Se supone que un conjunto de paquetes TCP capturados (*trazaTCP*) contiene un conjunto de conexiones TCP, cada una de las cuales a su vez tiene un establecimiento, una transferencia y una liberación de conexión. Teniendo en cuenta esto se estructura el módulo segun la figura 6.11.

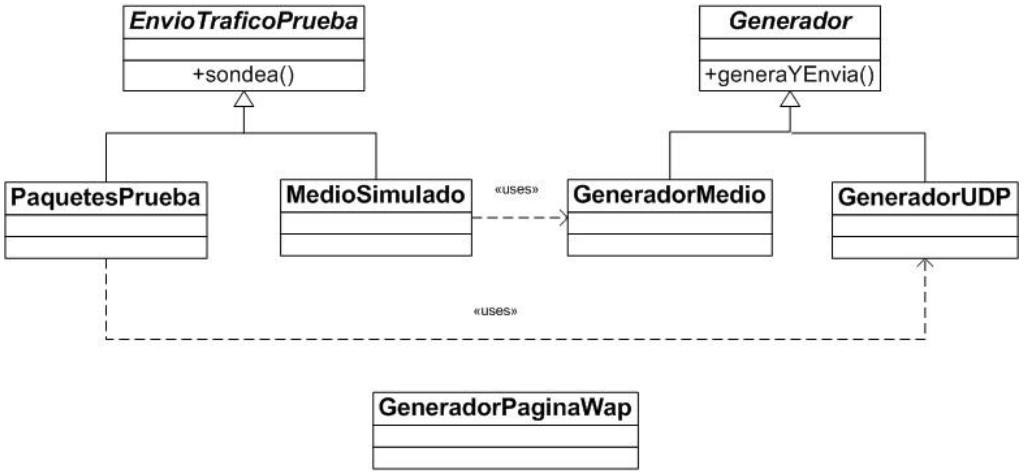


Figura 6.8: Modulo Envío tráfico.

6.3.4. Capa Datos

Esta capa tiene como funcionalidad la persistencia de datos, ya sea a través de bases de datos, archivos de texto plano o XML. Las operaciones que se definan en esta capa serán acordes a las necesidades del resto de la aplicación, ya sea para persistir datos o para recuperar datos ya guardados. En este diseño se considerará una base de datos como herramienta de almacenamiento de información, cuyo diseño será detallado más adelante.

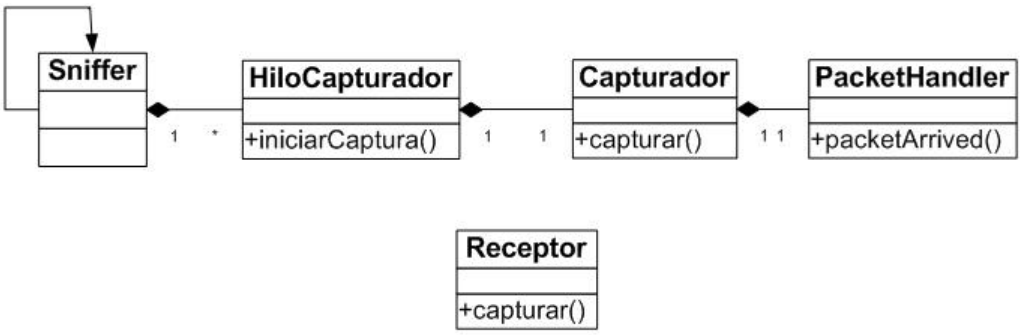


Figura 6.9: Modulo Captura tráfico.

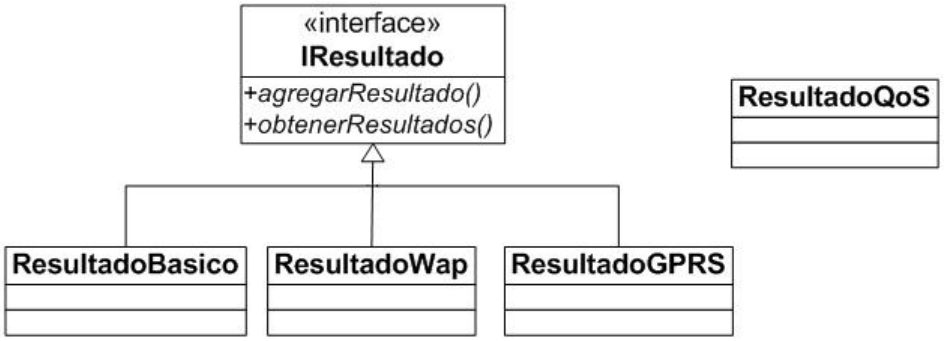


Figura 6.10: Modulo Resultado.

La clase DBMS

Las funciones de la capa están concentradas en la clase *DBMS*, la cual interactúa entre el controlador principal y la base de datos. En la figura 6.12 se observan las operaciones que ofrece.

Hay funcionalidades que preveen el acceso desde dos hilos distintos a la base de datos, lo cual requiere sincronización, especialmente teniendo en cuenta que uno debe ser después que termine el otro. Un ejemplo de esto son las pruebas de performance GPRS-EDGE que se efectuan desde el *Midlet*. En estos casos, para evitar que el *Midlet* almacene demasiados datos en memoria, conserva tan solo su número de experimento. Entonces cuando la prueba finaliza, se recuperan de la base de datos los parámetros con los cuales se efectuó la prueba, a efectos de utilizar dicha información cuando se aplique el algoritmo. Es necesario entonces asegurarse que al recuperar la información, ésta ya ha sido guardada correctamente en la base de datos. Para posibilitar este control se implementó el patrón *singleton* en la clase

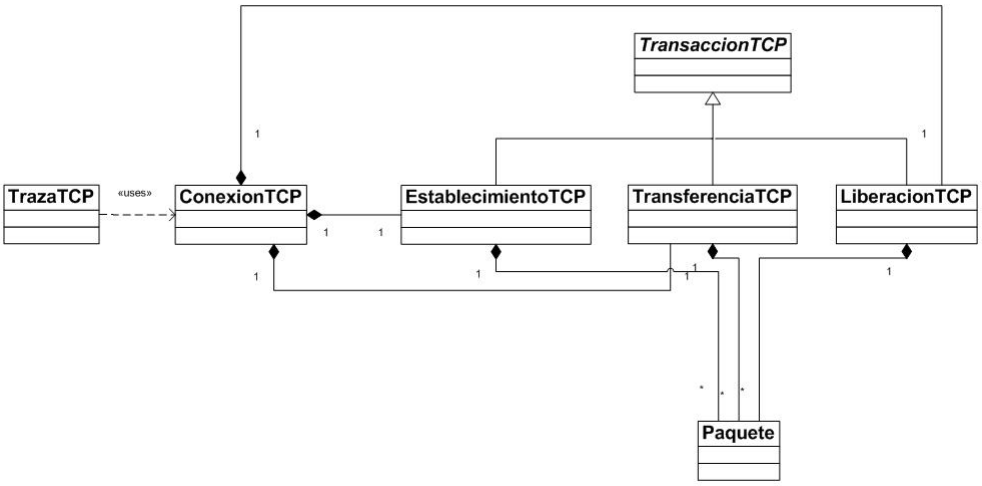


Figura 6.11: Modulo TCP.

DBMS, lo cual facilita los conotroles de sincronismo necesarios.

Esquema de almacenamiento

Para almacenar las pruebas realizadas se harán las siguientes consideraciones:

- Solamente los usuarios registrados en el sistema pueden realizar experimentos, por eso siempre es necesario validarse en el sistema antes de hacer prueba alguna.

- Toda la información generada al efectuarse un experimento es almacenada en tres instancias: creación del experimento, almacenamiento de las pruebas realizadas y almacenamiento de los resultados obtenidos.

- En la instancia de creación del experimento se guarda la información que detalla las características y tipo de prueba realizada. Como se observará, esta instancia se mapeará en la tabla *Experimento* de la base de datos.

- En la instancia de almacenamiento de las pruebas se guarda toda la información extraída de las capturas de tráfico realizadas. Esto se mapeará en la tabla *Pruebas*.

- En la instancia de almacenamiento de resultados se guardan, cuando es conveniente, los resultados obtenidos en el experimento. La tabla asociada a ésto es *Resultados*.

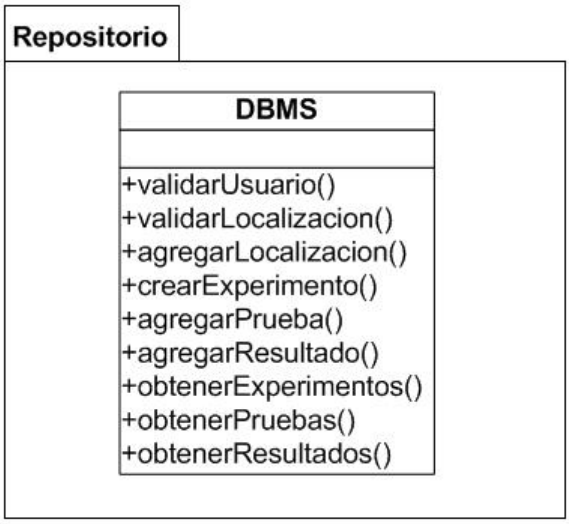


Figura 6.12: Repositorio de datos.

Diseño de la base de datos

La base de datos utilizada en el presente desarrollo está compartida con el proyecto *MetroNet*, por lo que hay tablas de la base de datos que no serán mencionadas aqui, por ser de uso exclusivo de dicho proyecto. Las tablas utilizadas se muestran en la figura 6.13 y se describen a continuación.

- **Usuarios**: Esta tabla tiene la función de registrar los usuarios que pueden acceder al sistema, asi como la contraseña que cada uno tiene para poder ingresar.

- **Localizaciones**: Registra todas las localizaciones de cada uno de los usuarios. No tiene definida una *primary key* ya que un usuario puede tener más de una localización registrada y más de un usuario puede compartir determinada localización.

- **Experimento**: En esta tabla son registrados todos los experimentos que se llevan a cabo por el sistema. Está vinculada con las tablas *Usuarios* y *Localizaciones* para validar estos datos. El identificador único que utiliza es el número de experimento. La columna *parametros* proporciona la información necesaria sobre el tráfico generado y el algoritmo aplicado, para que sea perfectamente identificada la prueba realizada.

- **Pruebas**: Aquí se guardan las capturas realizadas y por lo tanto la información necesaria para poder realizar las estimaciones.

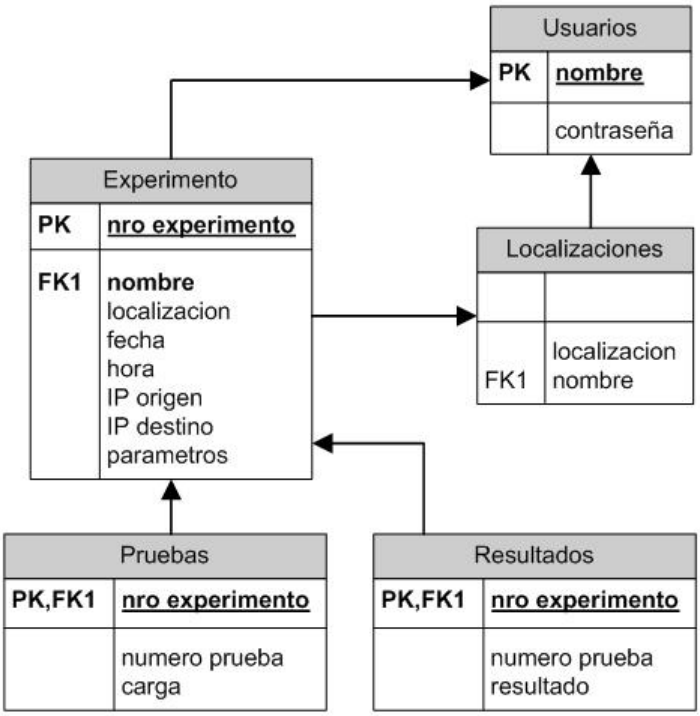


Figura 6.13: Estructura de la base de datos.

- **Resultados**: Si se considera necesario, se puede guardar el resultado del experimento realizado, ésto se realiza en esta tabla. La clase *DBMS* utiliza las clases que heredan de la interfaz *IResultado* para realizar esta operación. A su vez, al recuperar una serie de resultados, *DBMS* almacena los mismos en una clase que implemente dicha interfaz.

6.4. Comportamiento Dinámico del Sistema

En la siguiente sección se describe el funcionamiento dinámico de los casos de uso, explicando la interacción entre las clases descriptas anteriormente. No pretende ser una descripción exaustiva, sino que se menciona el rol de las principales clases que participan en cada uno de los casos de uso. Del mismo modo, los diagramas presentados están simplificados, mostrando solo en algunos casos (los mas pertientes) el mensaje que devuelven los métodos luego de ser invocados.

6.4.1. Validar usuario

1. El usuario ingresa nombre y contraseña en la interfaz que corresponda.

2. La interfaz de usuario envía un mensaje HTTP al servidor, pidiendo la autenticación del usuario.

3. El Servlet recibe dicha petición y hace un llamado al método validarUsuario del *ControladorPrincipal*.

4. El controlador principal ejecuta la operación validar usuario de *DBMS*, quien consulta con los datos recibidos la tabla *Usuarios* de la base de datos.

5. En caso de ser correctos los datos ingresados, *DBMS* responde afirmativamente a la consulta hecha por el *ControladorPrincipal*. En caso contrario le avisa que los datos no son válidos.

6. El *ControladorPrincipal* devuelve esta respuesta al Servlet, quien le envía la información a la interfaz de usuario.

7. En caso que los datos hayan sido validados, el usuario ingresa al sistema.

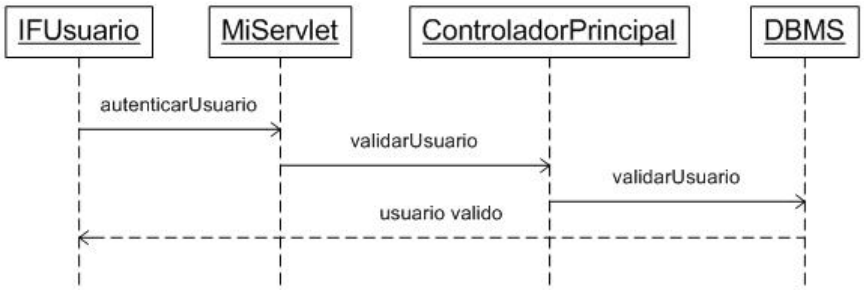


Figura 6.14: Secuencia del caso de uso Validar usuario.

6.4.2. Hacer prueba sobre redes fijas

1. Se supone que el usuario ya fue validado por el sistema.
2. *IFExperimento* lista los algoritmos disponibles.
3. El usuario selecciona el algoritmo con el cual quiere hacer el experimento.
4. *IFExperimento* envía un mensaje a *MiServlet* pidiendo las localizaciones disponibles del usuario.
5. *MiServlet* recibe dicho pedido y le pide a *ControladorPrincipal* una lista de las localizaciones del usuario.
6. *ControladorPrincipal* consulta a *DBMS* por dichas localizaciones.
7. *DBMS* interroga a la base de datos, obteniendo las localizaciones existentes en la tabla *Localizaciones* a nombre del usuario.
8. La lista obtenida es devuelta a *IFExperimento*, quien ofrece estas localizaciones para que el usuario seleccione su ubicación actual.
9. Si el usuario selecciona opciones avanzadas, *IFExperimento* lista los parámetros que pueden ser establecidos por el usuario, en caso contrario, el sistema ejecutará previamente una prueba de características de enlace y calculará a partir de ella los parámetros a utilizar.
10. Cuando el usuario selecciona comenzar la prueba, *IFExperimento* manda dicho mensaje a *MiServlet*, al mismo tiempo comienza a rebotar los paquetes UDP que llegan a través de la clase *Rebotador*.
11. Este pedido es transferido a *ControladorPrincipal*, el cual le indica a *MedioActual* que realice una prueba, detallandole el algoritmo a utilizar y los parámetros de la misma.
12. *MedioActual* comineza la prueba, dando la orden correspondiente a *PaquetesPrueba*.
13. *PaquetesPrueba* comienza la recepción y envío de paquetes UDP, a través de *GeneradorUDP* y *Receptor*. Los paquetes entretanto son rebotados en el cliente.
14. Una vez recibidos todos los paquetes, *ControladorPrincipal* le pasa a *DBMS* la información necesaria para que se cree el experimento y se guarden las capturas realizadas.
15. *ControladorPrincipal* pide a *MedioActual* que se procesen los datos y el resutado obtenido es transferido a *MiServlet* para que éste se los transfiera a *IFExperimento*.

16. *IFExperimento* muestra los datos al usuario y termina la prueba.

6.4.3. Recorrer sitio de prueba wap

1. El usuario debe previamente haber sido validado por el sistema.
2. Al comenzar la exploración del sitio, *MiServlet* recibe un pedido para iniciar la prueba.
3. *MiServlet* pasa dicho pedido a *ControladorPrincipal*, quien pide a *MedioWap* que se inicie una prueba.
4. *MedioWap* envía una orden a *Sniffer* para que sea creada una nueva captura.
5. *Sniffer* crea un nuevo *HiloCapturador* y le indica que inicie la captura.
6. *HiloCapturador* le indica a su *Capturador* asociado que inicie la captura.
7. *MedioWap* le solicita la primer página a *GeneradorPaginaWap* y se la devuelve a *ControladorPrincipal*, quien se la devuelve a *MiServlet* para que se la despliegue al usuario.
8. A medida que el usuario va explorando la página es *GeneradorPaginaWap* quien por pedido de *MedioWap* devuelve la página WAP correspondiente.
9. Al llegar el pedido de la última página del sitio, *ControladorPrincipal* le indica a *MedioWap* que terminó la prueba, quien hace que *Sniffer* finalice la captura correspondiente.
10. *MedioWap* devuelve a *ControladorPrincipal* una instancia de la clase *ResultadoWap* con los resultados de la prueba.
11. *ControladorPrincipal* pide a *DBMS* otra instancia de *ResultadoWap* que contenga los resultados históricos.
12. Se realiza la comparación entre el resultado actual y el histórico, por parte de *ResultadoWap*.
13. Es devuelto al cliente el resultado del experimento y la comparación con los datos históricos.

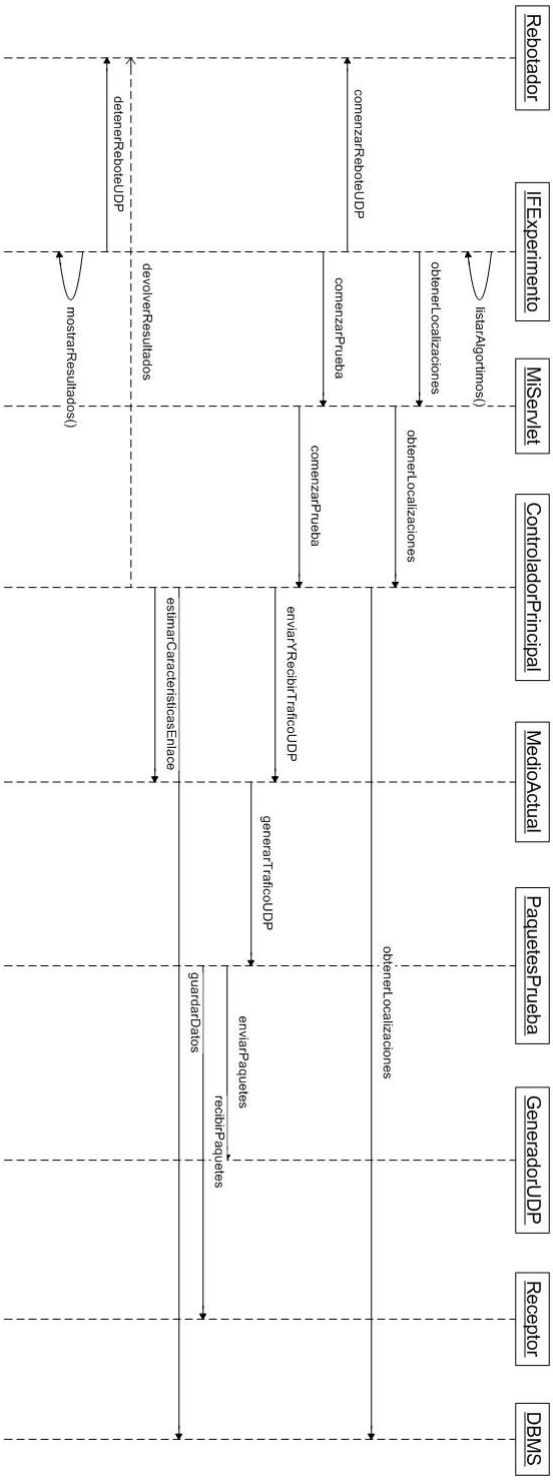


Figura 6.15: Secuencia del caso de uso Hacer prueba sobre enlaces alámbricos.

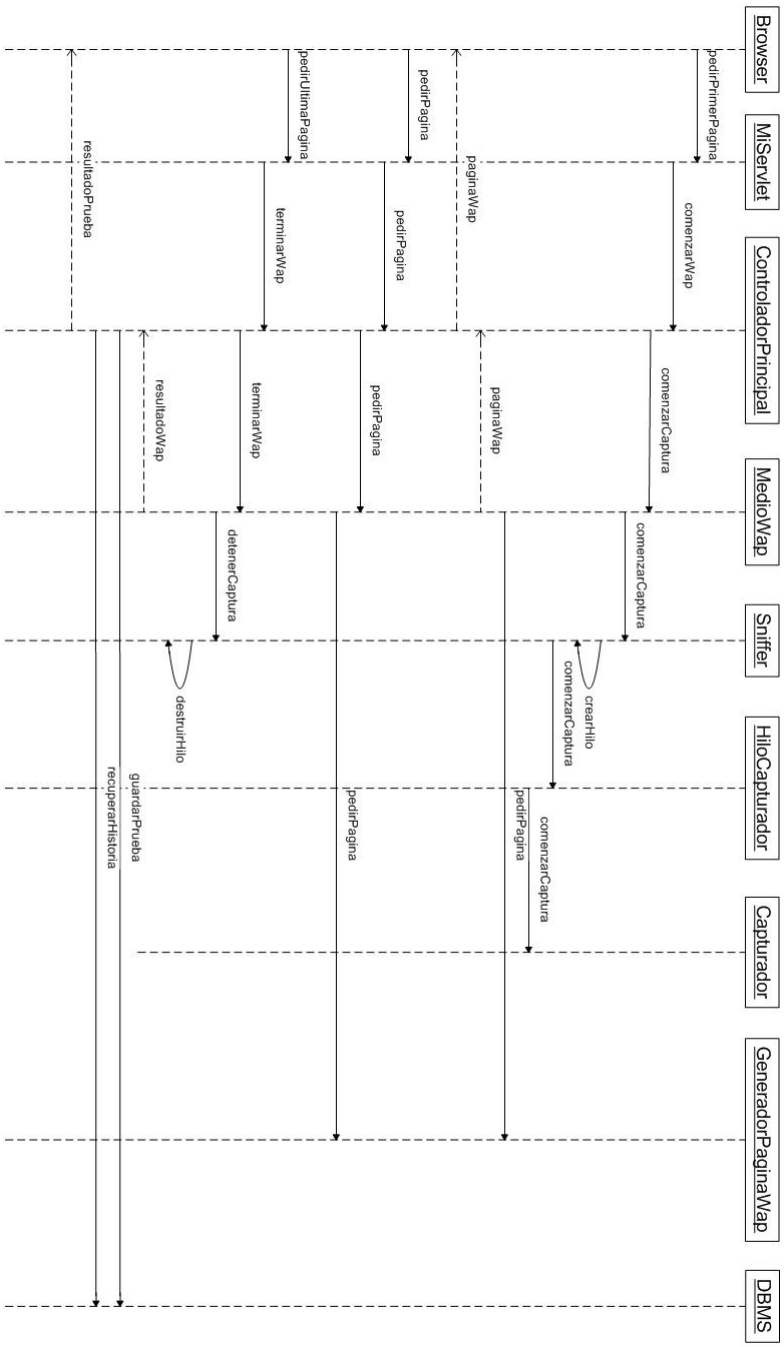


Figura 6.16: Secuencia del caso de uso recorrer sitio de prueba WAP.

6.4.4. Hacer captura WAP desde el núcleo de la red

1. El usuario debe previamente haber sido validado por el sistema.

2. El usuario ingresa en *IFCapturaWap* los filtros con los que desea realizar la captura.

3. *IFCapturaWap* envía dicha información a *MiServlet*, quien se la hace extensiva a *ControladorPrincipal*.

4. *ControladorPrincipal* le da la orden a *MedioWap* que inicie la captura con los filtros especificados.

5. Del mismo modo explicado en el caso de uso anterior, *MedioWap* da la orden de que se inicie la captura.

6. Una vez finalizada la captura, los datos se guardan en la base de datos y se recuperan los resultados históricos para hacer la comparación correspondiente.

7. *ControladorPrincipal* recibe los resultados y la comparación y se los hace llegar a *IFCapturaWap*.

8. *IFCapturaWap* muestra al usuario los resultados obtenidos.

6.4.5. Hacer prueba para evaluar la performance de enlaces GPRS-EDGE

1. El usuario debe previamente haber sido validado por el sistema.

2. *IFGPRSPerform* [4] obtiene y lista las localizaciones del usuario, del mismo modo que fue explicado en la sección 6.4.2.

3. El usuario selecciona los parámetros con los cuales desea hacer la prueba.

4. *IFGPRSPerform* envía esta información a *MiServlet*, quien se la transfiere a *ControladorPrincipal*.

5. El flujo de eventos de aquí en adelante es igual al explicado en la sección 6.4.2.

6.4.6. Ver historial de pruebas

1. El usuario debe previamente haber sido validado por el sistema.

[4] Esta misma prueba puede hacerse desde el móvil, utilizando el *Midlet*, el flujo de eventos en este caso es el mismo.

2. El usuario selecciona el criterio con el cual desea hacer la consulta (usuario, localización, fechas de inicio y fin).

3. Dicha consulta es enviada a *MiServlet* y transferida a *Controlador-Principal*.

4. *ControladorPrincipal* realiza la consulta correspondiente a *DBMS*.

5. *DBMS* encapsula el resultado de la consulta en la instancia de una clase que implemente la interfaz *IResultado*.

6. *ControladorPrincipal* extrae los datos convenientes y les da el formato para presentarlos al usuario.

7. El usuario recibe a traves de la interfaz en la que se encuentre una lista con los resulados de los experimentos que cumplan las condiciones seleccionadas.

Parte IV

Conclusiones y Trabajo Futuro

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

En el presente trabajo se realizó un estudio en el área de la evaluación de performance en redes de telecomunicaciones, enfocando dicho estudio en el área de transmisión de datos sobre los protocolos GPRS-EDGE. Como complemento se efectuó un estudio de la performance del protocolo WAP, una de las principales aplicaciones utilizadas sobre este tipo de enlaces.

Es importante resaltar que si bien el objetivo claro del proyecto era el estudio en dicha área, no estaba bien determinado a priori los alcances del mismo. Mas alla de esto, se logró desarrollar un conjunto de herramientas que redondean el estudio de los protocolos deseados, dando una visión amplia y clara sobre el funcionamiento de los mismos.

Tras haber efectuado un exahustivo estudio de las diversas técnicas de medición existentes en el área de estimación de parámetros de performance en redes de datos y tras haber analizado a fondo el funcionamiento de los protocolos GPRS-EDGE; se diseño, implementó y probó un algoritmo con el fin de estudiar en profundidad el comportamiento de dichos protocolos.

Si bien la base de herramientas utilizadas para el diseño del algoritmo son bien conocidas en el área, el producto final desarrollado tiene como característica la de resolver un problema poco tratado en la misma. El hecho de haber diseñado un algoritmo innovador y eficiente a la hora de resolver el problema dado, hacen que no solo los objetivos planteados al inicio del proyecto en esta etapa se vean ampliamante cumplidos sino que además se logró generar una gran base de conocimiento y perspectivas de investigación futura en el área tratada.

Por otra parte se desarrolló una herramienta capaz de medir la perfor-

mance del protocolo WAP, en la cual se obtuvieron resultados satisfactorios de acuerdo a lo esperado. Además se implementó un módulo que si bien no pudo ser probado en el lugar para el cual fue diseñado, es una herramienta de gran utilidad para monitorear las conexiones WAP en la red del proveedor.

Otra característica a resaltar del proyecto en general, es que las aplicaciones implementadas tienen la propiedad de estar prontas para quedar en producción para un cliente, una vez que se cuente con la infraestructura necesaria. Lejos de ser un prototipo el sistema se encuentra en producción en un servidor de la Facultad de Ingeniería de la Universidad de la República y puede ser probado por cualquier usuario con acceso a Internet.

7.2. Trabajo Futuro

El trabajo realizado deja abiertas las puertas al estudio específico de los enlaces GPRS-EDGE, dejando sentado un importante análisis sobre su funcionamiento y un algoritmo que analiza su comportamiento.

En el área de la estimación de performance de WAP, se plantea la posibilidad de implementar un algoritmo que vincule la calidad percibida por los usuarios que exploren el sitio de prueba con los parámetros medidos, logrando de este modo que el sistema aprenda a reconocer, a partir de las mediciones que actualmente se realizan, cuál es la calidad de la conexión analizada.

Finalmente se propone comparar los resultados obtenidos a través de dichas mediciones con las capturas que puedan realizarse en la red del proveedor de servicio, a través del módulo ya desarrollado. Adicionalmente se podría implementar un módulo que relacione y compare las mediciones hechas en dicho punto con las mediciones realizadas en un servidor de Internet.

Bibliografía

[1] Sitio web de la gsm association. http://www.gsmworld.com/.

[2] Rfc número 739 (tcp), 1981. http://www.faqs.org/rfcs/rfc793.html.

[3] Sitio web de la librería jpcap, 2004. http://sourceforge.net/projects/jpcap.

[4] Sitio web del ethereal, 2006. http://www.ethereal.com/.

[5] Sitio web del tcpdump, 2006. http://www.tcpdump.org/.

[6] Peter Arnby, Johan Hjelm, and Peter Stark. Wap 2.x architecture features, services and functions, 2005.

[7] Pablo Belzarena. Metrología en ip, estimación de capacidades, 2005.

[8] Pablo Belzarena, Pedro Casas, Federico Larroca, Laura Aspirot, and Víctor Gonzalez Barbone. Documentación del proyecto metronet, 2002.

[9] Peter Benko, Gabor Malicsko, and Andras Veres. A large-scale, passive analysis of end-to-end tcp performance over gprs, 2004.

[10] Peter Benko and Andras Veres. A passive method for estimating end to end tcp packet loss.

[11] Francisco José Cortijo Bon. Técnicas no supervisadas: Métodos de agrupamiento, 2001.

[12] Isabel Cañete. Estimación de densidades basada en núcleos: algunos elementos básicos, 2002.

[13] Isabel Cañete. Estimadores de densidad basados en núcleos: cómo hallarlos en la práctica, 2004.

[14] Rajiv Chacravorty, Joel Cartwright, and Ian Pratt. Practical experience with tcp over gprs.

[15] Javier García de Jalón, José Ignacio Rodríguez, Iñigo Mingo, Aitor Imaz, Alfonso Brazález, Alberto Larzabal, Jesús Calleja, and Jon García. Aprenda java como si estuviera en primero, 2000.

[16] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. What do packet dispersion techniques measure? In *INFOCOM*, 2001. http://citeseer.ist.psu.edu/516746.html.

[17] Allen Downey and Colby College. Using pathchar to estimate internet link characteristics.

[18] Wikipedia en español. Regla de sturgess. http://es.wikipedia.org/wiki/Regla_de_Sturgess.

[19] Hao Jiang and Constantinos Dovrolis. Passive estimation of tcp round-trip times, 2002.

[20] Kevin Lai and Mary Baker. Measuring link bandwidths using a deterministic model of packet delay.

[21] Kevin Lai and Mary Baker. Nettimer: A tool for measuring bottleneck link bandwidth. 2001. http://citeseer.ist.psu.edu/lai01nettimer.html.

[22] Irene C. Y. Ma and James Irvine. Characteristics of wap traffic, 2005.

[23] Sun Microsystems. Api de java 1.5.

[24] V. Ribeiro, R. Riedi, R. Baranuik, J.Ñavratil, and L. Cot. Pathchirp: Efficient available bandwidth estimation for network paths, 2003. http://citeseer.ist.psu.edu/ribeiro03pathchirp.html.

[25] Sergio Gálvez Rojas and Lucas Ortega Díaz. Java a tope: J2me.

[26] Elsa Servy, Cristina Cuesta, and Gonzalo Mary. Comparación del comportamiento de diversos estimadores basados en núcleos, 2004.

[27] B.W. Silverman. Density estimation for stadistics and data analysis, 1986.

[28] Andrew S. Tanenbaum. Redes de computadoras - 4ta edición, 2003.

[29] Berwin A. Turlach. Bandwidth selection in kernel density estimation: A review, 1993.

[30] Zhanping Yin and Victor Leung. A proxy architecture to enhance the performance of wap 2.0 by data compression, 2005.

Parte V

Apéndices

Apéndice A

Contenido del CD Adjunto

Bibliografía Y Referencias

En la carpeta `Bibiliografía` se encuentran en versión electrónica las publicaciones citadas en esta documentación, así como otras publicaciones utilizadas como material adiciona.

Documentación

En la carpeta `Documentación` se enuentra la presente versión de este documento en formato electrónico, así como el API del software desarrollado.

Software

En la carpeta `Software` se adjunta el proyecto codificado en JAVA, así como las librerías y software extra utilizado durante el proyecto

- Software de `Metronet/MetroCel`
- Librería `jpcap`
- `Apache Tomcat`
- `JDK 1.5`
- Simuladores de teléfonos celulares

Apéndice B

Dedicación

En todo proyecto de ingeniería de magnitud considerable es necesario establecer cronogramas de plazos y planificar tareas para cumplir con los objetivos intermedios y finales del mismo. En particular, el presente proyecto fue llevado a cabo en un lapso de 17 meses. Dado que los objetivos del proyecto se fueron planteando de acuerdo a la marcha del mismo y a los requerimientos surgidos durante su transcurso, no se efectuó una planificación del mismo de principio a fin si no que la misma fue efectuada mediante etapas.

En las siguientes secciones se muestra la tabla original planificada en el curso de Gestión de Proyecto [1] y la tabla de ejecución real del mismo.

[1] Curso de gestión dictado en el IIE previo al comienzo del proyecto.

B.1. Planificación

Nº	***Tarea***	***Fecha de inicio***	***Fecha de finalización***	***Horas por integrante***
1	Familiarización con el sistema de MetroNet	15/10/05	12/11/05	28
2	Implementación de algoritmos en redes fijas	13/11/05	11/12/05	28
3	Investigación acerca de parametros de performance en redes celulares	15/10/05	31/12/05	49
4	Estudiar protocolo WAP	15/10/05	31/12/05	49
5	Investigación de técnicas y requerimientos para efectuar mediciones sobre una red celular	1/01/06	16/03/06	245
6	Estudios en Java	1/01/06	26/02/06	140
7	Diseño e implementación del software	17/03/06	30/06/06	525
8	Evaluación de lo realizado y plan de trabajo	1/07/06	15/07/06	70

B.2. Ejecución Real

Nº	*Tarea*	*Fecha de inicio*	*Fecha de finalización*	*Horas por integrante*
1	Familiarización con el sistema de MetroNet	1/03/06	1/06/06	50
2	Implementación de algoritmos en redes fijas	13/11/05	1/03/06	180
3	Investigación acerca de parametros de performance en redes celulares	1/10/05	31/12/05	60
4	Estudiar protocolo WAP	1/10/05	31/12/05	100
5	Investigación de técnicas y requerimientos para efectuar mediciones sobre una red celular	1/06/06	16/10/06	100
6	Estudios en Java	1/01/06	26/02/06	80
7	Adaptación de algoritmos desarrollados para redes fijas al sistema de MetroNet	17/03/06	31/06/06	200
8	Implementación del sistema de estimación de parámetros en WAP	1/06/06	1/10/06	300
9	Diseño e implementación del algoritmo de estimación de performance sobre GPRS	17/03/06	15/01/07	400
10	Realzación de pruebas para validar los algoritmos desarrollados	15/12/06	25/02/07	250
11	Elaboración de la documentación final del proyecto	15/1/07	25/02/07	250

B.3. Evaluación de la Planificación y Conclusiones

Como primer conclusión importante, se vió que no era viable separar cronológicamente estudios teóricos de pruebas y prácticas que permitan verificar y entender dichos estudios. Se pudo comprobar la importancia de ejecutar ambas tareas en forma alternada, lo cual permite un entendimiento más rápido y efectivo de los temas que se están analizando.

Otro aspecto a tener en cuenta es que si bien el objetivo central del proyecto estaba enfocado a realizar mediciones sobre enlaces celulares y analizar su comportamiento, fue necesario invertir una muy importante dedicación de horas al desarrollo del software, con todos los aspectos que esto trae involucrado.

Un problema que se tuvo fue el de, en determinado momento, migrar el software desarrollado hasta entonces a la arquitectura de *MetroNet*, lo cual insumió mucho más tiempo del esperado.

Finalmente, hay que tener en cuenta que más alla de los desfasajes en el cronograma planteado originalmente, fue posible ejecutar todos los objetivos planteados desde el inicio, habiendo una importante coherencia entre los objetivos planteados inicialmente y el producto final.