Topología enriquecida de Internet

Tesis de grado
Ingeniería Electrónica

Marzo 2012

Tesista: Esteban Poggio

Tutor: Dr. José Ignacio Alvarez-Hamelin

Agradecimientos

Luego del esfuerzo realizado para terminar la presente tesis quiero agradecer a aquellas personas que de alguna manera estuvieron vinculadas a este trabajo:

En primer lugar, a los miembros del jurado: Lic. Jorge R. Busch, Ing. Luis Marrone y Dr. Juan Ignacio Giribet por haber aceptado la lectura y evaluación de este trabajo.

A José Ignacio, por aceptar ser el tutor de esta tesis y tenerme paciencia cuando las cosas no avanzaban; por guiarme y aconsejarme y también por incentivarme a seguir estudiando e investigando.

En lo personal también quiero agradecer a las siguientes personas:

A mis viejos, José María y Lucía, por darme la posibilidad de poder priorizar el estudio.

A mis hermanos, Anabella y Nacho, por estar siempre dispuestos a escucharme, aconsejarme y porque siempre creyeron en mi; y por sobre todas las cosas... por soportarme.

A mi compañera, Valentina, por ser la persona que siempre escuchó mis quejas sobre la Tesis, ahora sí llegó el tan ansiado momento. No puedo escribir en una línea lo que significó su aporte para llegar a este momento. ¡Te quiero hermosa!

A mis amigos, porque siempre creyeron en que este momento iba a llegar. Gracias por los momentos compartidos y por estar siempre.

A mis compañeros de facultad, por compartir tardes enteras de estudio, tenerme paciencia cuando no entendía algo y hacer más ameno el curso de esta carrera.

A mis compañeros de trabajo del Centro de Comunicación Científica de la UBA, porque siempre entendieron que mi prioridad era terminar los estudios.

II

Índice general

Índice de figuras

Capítulo 1

Introducción

El modelado de Internet surge debido a la necesidad de disponer de redes para la simulación de protocolos de comunicación. Por ejemplo, para verificar el funcionamiento de nuevos algoritmos de ruteo en redes artificiales que posean las mismas características que las reales. Para mejorar los modelos existentes se necesitan exploraciones más completas, por ejemplo agregando a la topología datos como el ancho de banda o la utilización de los enlaces.

El presente trabajo se centrará en poder extraer mapas anotados de Internet, es decir poder caracterizar redes como Internet mediante el relevamiento de su topología y propiedades características. Ejemplo de estas propiedades pueden ser: ancho de banda, demora, disponibilidad, caracterización estadística del tráfico, tasa de pérdidas, etc. Para poder cumplir con este objetivo, debemos realizar exploraciones a gran escala sobre Internet, con el fin de obtener un modelo representativo de la realidad.

Comenzaremos definiendo el concepto de tomografía de una red, que podría definirse como: "el estudio de las características internas de una red mediante mediciones externas", donde cada uno de estos conceptos significan lo siguiente:

- características internas: son determinadas propiedades de una red, como por ejemplo: la demora en cada enlace, la topología, el porcentaje de pérdida de paquetes en cada enlace.

- mediciones externas: se toman siempre entre dos nodos de la red, esto quiere decir que no se estudia un paquete mientras el mismo se encuentra viajando por la red; estos nodos se encuentran generalmente en la periferia de la red a estudiar.

- tomografía: se usa este término en analogía con el conjunto de técnicas aplicadas en el campo de la medicina, el que intenta ser un proceso de medición poco invasivo. Esta no invasividad de las mediciones debe entenderse de la siguiente manera: el tráfico inyectado en la red para realizar las mediciones es prácticamente despreciable frente al tráfico normal de datos.

Para señalar la utilidad de una tomografía de red, previamente debemos introducir el concepto de agregación. Basándonos en el ejemplo de la figura 1.1, podemos ver que el camino entre los nodos A y B atraviesa los enlaces x_1, x_2 y x_3.

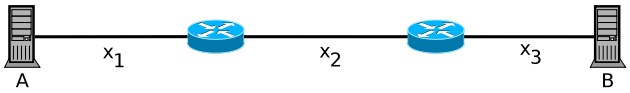


Figura 1.1: Concepto de agregación

La demora de un paquete entre A y B puede expresarse como en la ecuación 1.1, es decir, como la suma de la demora de cada uno de los enlaces que atraviesa. Si bien tomar la medición agregada (tiempo que tarda un paquete en llegar de A a B) es sencillo, tomar una medición individual de cada uno de los enlaces no lo es, dado que deberíamos tener control sobre cada uno de ellos.

$$t_{AB} = t_{x_1} + t_{x_2} + t_{x_3} \tag{1.1}$$

Podríamos decir que es muy difícil o imposible medir de forma directa los parámetros mencionados anteriormente (demora, topología, pérdida de paquetes), pero sí es posible medirlos de manera indirecta, mediante el cruce de información agregada. Retomando la utilidad de la tomografía de la red, podríamos decir que ésta consiste en el estudio de las propiedades internas (individuales) mediante mediciones externas (agregadas).

Las exploraciones se realizarán utilizando la red PlanetLab, de la cual la Universidad de Buenos Aires forma parte. En la sección 4.1 se describen las utilidades de esta plataforma y cómo es su funcionamiento. Para implementar las exploraciones a gran escala, las mismas se realizarán eligiendo 100 nodos de la red PlanetLab, realizando mediciones entre todos ellos.

El resto del trabajo se organiza de la siguiente manera: en el capítulo 2 se describe la situación actual de la temática. Luego, en el capítulo 3, se definen los conceptos fundamentales sobre los cuales se construye este trabajo. Posteriormente, en el capítulo 4, se presentan las herramientas utilizadas para la realización de las exploraciones y los resultados experimentales obtenidos. Finalmente, en el capítulo 5, se desarrollan las conclusiones finales y posibles estudios a futuro.

Capítulo 2

Estado del arte

En este capítulo se presenta una descripción de la situación actual del tema, basada en el análisis de la literatura existente hasta el momento (Diciembre de 2011) y de la revisión de los principales proyectos activos.

Para comenzar, en la introducción de la sección 2.1, se presenta la problemática del crecimiento del número de mediciones a medida que aumentan los nodos de una red. Luego, en las secciones 2.1.1, 2.1.2 y 2.1.3 se analizan diferentes trabajos, los que se encargan de estudiar cómo seleccionar rutas para efectuar mediciones sobre redes de forma eficiente.

Posteriormente, en la sección 2.1.4, se presenta una metodología para implementar mediciones a gran escala, denominada IDM (*Inter-packet Delay Measurement*). Este sistema fue pensado y desarrollado como una poderosa herramienta para investigadores, diseñado como una extensión de la ya conocida infraestructura DIMES [1].

Finalmente, en la sección 2.2, se presentan dos de los principales inconvenientes a resolver a la hora de efectuar exploraciones a gran escala sobre redes como Internet: los ruteadores que no responden a la expiración del TTL (*Time To Live*) y la resolución de *aliases*.

2.1. Marco teórico

Dados n nodos controlables, para medir todas las rutas posibles entre ellos, se necesitarían $n \cdot (n-1)$ mediciones, siendo n el número total de nodos, esto hace que, a medida que n crece, el sistema no sea escalable ya que se necesitarían en teoría, $O(n^2)$ mediciones. Por ejemplo, considerando el grafo

completo de 6 nodos de la figura 2.1, se necesitarían $6 \cdot 5 = 30$ mediciones. En la bibliografía actual, hay varios trabajos que se encargan de estudiar esta problemática, es decir, estudian la posibilidad de reducir el número de mediciones a partir de una adecuada selección de rutas a monitorear.

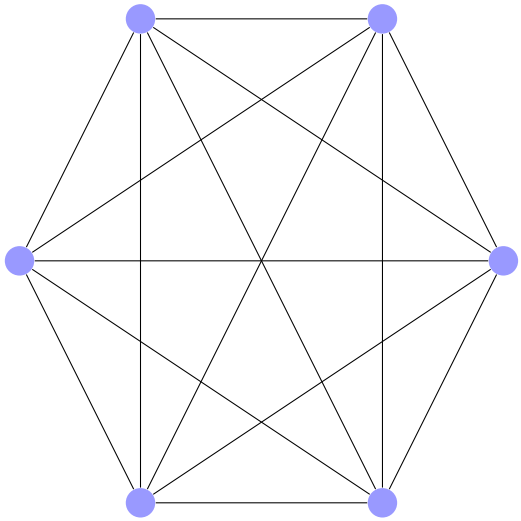


Figura 2.1: Grafo completo de 6 nodos.

2.1.1. Primer enfoque algebraico

Como es sabido, en redes como Internet, muchos enlaces son compartidos por diferentes rutas, por lo que la cantidad real de incógnitas es bastante menor a $O(n^2)$ (cota superior de mediciones, para el caso del grafo completo). Chen et al. [2, 3] demuestran que para un n suficientemente grande (por ejemplo, n>100), el número total de mediciones necesarias para poder obtener toda la información sobre el grafo, está acotada por $O(n \cdot log(n))$, valor que representa una cota superior, el cual fue obtenido a partir del estudio de redes simuladas y reales. Inicialmente, los autores se plantearon encontrar un conjunto mínimo de k caminos linealmente independientes que puedan describir completamente todos los $O(n^2)$ caminos y observaron que k crece relativamente lento como función de n.

El modelo matemático usado en este caso, parte de representar un camino del grafo por un vector $v \in \{0,1\}^s$, donde la j-ésima entrada v_j es 1 si el enlace j es parte del camino y cero en otro caso; s es la cantidad de enlaces que tiene el grafo y $r = O(n^2)$ es el número de caminos extremo a extremo.

Considerando todos los caminos de la red, hay r ecuaciones lineales. Agrupando estos caminos, se forma una matriz rectangular $G \in \{0,1\}^{r \times s}$. Cada fila de G representa un camino en la red:

$$G_{i,j} = \begin{cases} 1 & \text{si el enlace } j \text{ pertenece al camino } i \\ 0 & \text{en otro caso} \end{cases}$$

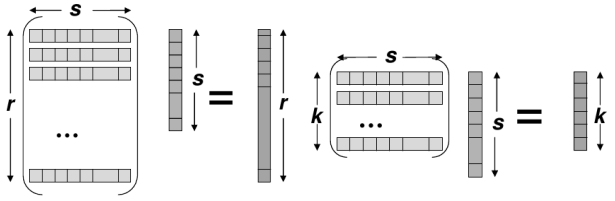
Luego, se puede escribir el sistema de ecuaciones lineales que relaciona las mediciones en los enlaces con las mediciones en los caminos como:

$$G \cdot x = b \tag{2.1}$$

donde,

- G: matriz de ruteo
- x: métrica, como por ejemplo pérdida de paquetes, demora, etc.
- b: valores medidos de la métrica

Normalmente, el número de caminos r es mucho mayor que el número de enlaces s (ver figura 2.2a). Esto sugiere que podríamos seleccionar s caminos a monitorear, usar las mediciones para resolver el sistema (obtener las variables x_i) e inferir el valor de las incógnitas para el resto de los caminos usando la ecuación (2.1).



(a) $Gx = b$ (b) $\overline{G}x_G = \overline{b}$

Figura 2.2: Representación del tamaño de la matriz.
(Imagen extraída de [3])

Generalmente la matriz de ruteo G presenta rango deficiente, es decir, $k < s$ (donde $k = rango(G)$), esto quiere decir que será imposible determinar la solución de algunas incógnitas a partir de la ecuación (2.1). Llevado esto a la terminología de los enlaces, estos enlaces serán no identificables. Para obtener sólo la información útil, se deben separar los enlaces identificables de los no identificables; esto es separar el espacio fila y el espacio nulo de G.

Para obtener los k caminos a monitorear, los autores proponen utilizar algoritmos con técnicas reveladoras de rango, como por ejemplo la descomposición QR [4].

En general la matriz G es rala; esto es, tiene unos pocos elementos distintos de cero por fila. Esta propiedad se aprovecha para acelerar el cálculo; los autores utilizan rutinas optimizadas de la biblioteca LAPACK [5] (*Linear Algebra Package*) para implementar las técnicas reveladoras de rango, donde se procesan varias filas a la vez. La complejidad del algoritmo es $O(rk^2)$, donde r es la cantidad total de rutas y k el rango de la matriz de ruteo R.

Luego, los autores ponen a prueba este trabajo sometiéndolo a diferentes experimentos sobre redes simuladas y reales obteniendo resultados apropiados. Su propuesta funciona en tiempo real, ofreciendo adaptación rápida a los cambios de topología (agregado y eliminación de caminos) y maneja los errores de medición de topología.

Llegado este punto, vale la pena aclarar que, los autores de este trabajo desarrollan esta metodología buscando propiedades de la red en rutas entre extremos de la misma, sin interiorizarse en las características individuales de los enlaces que componen una ruta. Para su estudio se basan en redes de *overlay*, es decir, redes que se construyen sobre otras redes. Los nodos de la red de *overlay* se pueden considerar como conectados por enlaces virtuales o lógicos, cada uno de los cuales corresponde a una ruta, tal vez a través de muchos enlaces físicos, en la red subyacente.

2.1.2. Enfoque estadístico

Siguiendo con la idea de cuántos caminos medir, posteriormente Chua et al. [6] se replantean el problema desde el punto de vista de predicción estadística. Muestran que las propiedades de la red de extremo a extremo se pueden predecir con exactitud en muchos casos, utilizando un conjunto mucho menor de caminos medidos (cuidadosamente seleccionados) de los necesarios para la recuperación exacta.

Los autores analizan las matrices de ruteo y observan que en la práctica poseen un rango efectivo[1] bajo. Esto significa que todos los caminos en la red

[1] Una matriz tiene rango efectivo q, si una pequeña perturbación es suficiente para hacerla de rango q pero es necesaria una gran perturbación para hacer la matriz de rango $q-1$.

pueden ser aproximadamente construidos a partir de un conjunto de caminos bastante más chico que el propio rango de G. La implicancia de esto es que ciertos enlaces en la red son más importantes para medir que otros.

Los autores muestran esta propiedad estudiando un caso real: la red de *Abilene*. Esta red, que se muestra en la figura 2.3, consiste en 11 nodos distribuidos a lo largo y a lo ancho de los Estados Unidos, red que cuenta con 30 enlaces, en este caso, considerados como dirigidos.

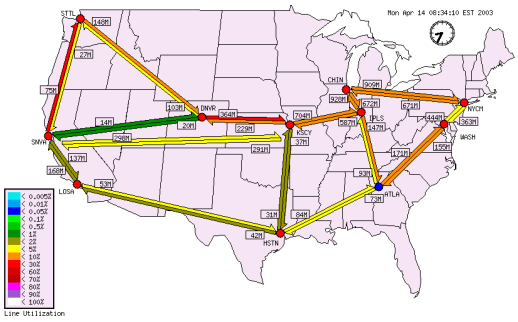


Figura 2.3: Mapa de la red *Abilene*.
(Imagen extraída de http://wn.com/abilene_network)

En la figura 2.4 se muestra el espectro de la matriz de ruteo G. El espectro de la matriz de ruteo está formado por los valores singulares[2] de la misma ordenados en forma decreciente (eje de ordenadas) en función de la cantidad de enlaces (eje de abscisas).

Generalizando, se puede ver el mismo comportamiento en diferentes redes, como se muestra en la figura 2.5, donde los valores singulares están normalizados por el mayor valor singular δ_i. Para cada red, se observa claramente, que se produce un quiebre significativo en el momento en que se superan el 20 % de los valores singulares, lo que indica una dimensión efectiva mucho

[2]La descomposición en valores singulares (SVD) de una matriz $M \in \mathbb{R}^{m \times n}$ es una descomposición de la forma $M = U \cdot \Delta \cdot V^T$ donde U y V son, respectivamente, matrices ortogonales $m \times m$ y $n \times n$ y Δ es una matriz rectangular diagonal $m \times n$ cuyos valores $\delta_i \geq \delta_{i+1}$ son los valores singulares. Una propiedad de esta descomposición es la relación $M \cdot v_i = \delta_i \cdot u_i$ entre las columnas u_i y v_i de U y V, respectivamente. La SVD de una matriz M está estrechamente relacionada con la descomposición en autovalores y autovectores de $M \cdot M^T$; en particular, la raiz cuadrada de los autovalores de $M \cdot M^T$ son los valores singulares de la matriz M.

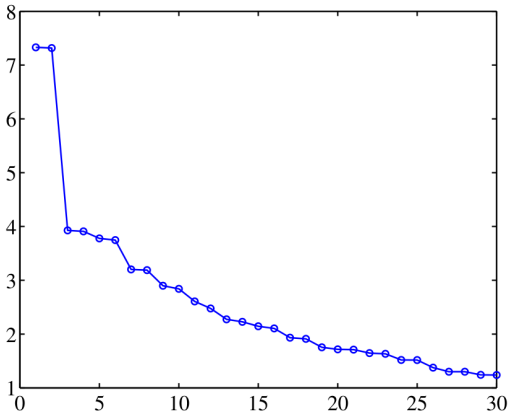


Figura 2.4: Espectro de la matriz de ruteo G de *Abilene*.
(Imagen extraída de [6])

menor que el rango de la matriz de ruteo. Los espectros de las matrices de ruteo que se muestran en la figura 2.5 fueron medidos por *Rocketfuel*[3].

Los autores describen un marco estadístico para predecir las propiedades extremo a extremo de la red a partir de los valores observados. Plantean $y = G \cdot x$ como la suma correspondiente de la métrica para el conjunto de n caminos, donde:

- x: métrica (como por ejemplo la demora o tasa de pérdida), de media μ y covarianza Σ
- G: matriz de ruteo
- y: valores medidos de la métrica

El objetivo es extraer información de una serie de condiciones globales de las rutas. Esto puede verse como un clásico problema de predicción estadística con muestreo. Si la calidad del estimador se mide en base al error cuadrático medio MSPE (*Mean Squared Prediction Error*), el mejor estimador es el de la media a posteriori. Sin embargo, suele restringirse a otra clase de posibles estimadores: los estimadores lineales. El BLP (*Best Lineal Predictor*) es un ideal que no se puede computar debido a que depende de μ (media de x). Estimando μ surge así el E-BLP (*Estimated* BLP), que será el mejor estimador entre todos los lineales.

[3] http://www.cs.washington.edu/research/networking/rocketfuel/

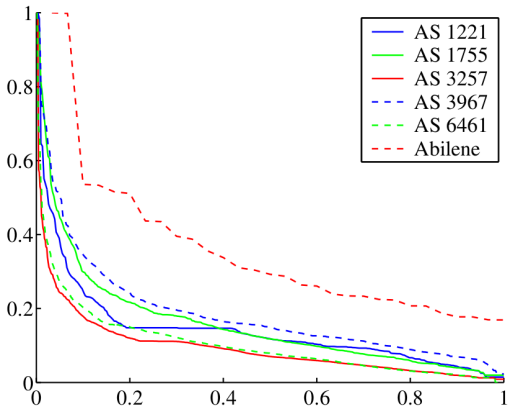


Figura 2.5: Espectros de matrices de ruteo.
(Imagen extraída de [6])

El E-BLP se puede ver como el resultado de los siguientes pasos:

1. Estimar x con la inversa generalizada de Moore-Penrose, donde se obtiene $\hat{x}$.

2. Mapear esta estimación a un vector de valores de rutas predichos: $\hat{y} = G \cdot \hat{x}$.

3. Predecir $l^T y$ por $l^T \hat{y}$.

Podría pensarse a priori, que la selección de los caminos a medir deberían ser los que minimicen el MSPE del estimador, pero esto no es realizable en la práctica, dado que es un problema NP-completo[4], por lo que no existen algoritmos eficientes para encontrar su solución. Se debe buscar una solución aproximada. Entonces, dada una matriz G y un entero k, se busca un subconjunto de k filas (o columnas) linealmente independientes de G que aproxime las primeras k dimensiones singulares y forme una submatriz bien condicionada[5].

[4]En teoría de la complejidad computacional, la clase de complejidad NP-completo es el subconjunto de los problemas de decisión en NP (*Nondeterministic Polynomial-time*) tal que todo problema en NP se puede reducir a cada uno de los problemas de NP-completo, mediante un algoritmo de complejidad polinomial. Informalmente, los problemas NP son aquellos en los que la única forma que hoy se conoce de determinar todas las instancias positivas (es decir aquellas que cumplen alguna propiedad) es verificar todas las instancias posibles. Contrariamente, en los problemas P (*deterministic Polynomial-time*) se pueden encontrar todas las instancias positivas en tiempo polinomial [7].

[5]Que la matriz esté bien condicionada garantiza que el sistema lineal resultante podrá resolverse digitalmente con un error numérico razonable.

La complejidad de cálculo del estimador E-BLP es $O(n_p^2 n_e)$ donde n_p y n_e son el número total de caminos y enlaces del grafo, respectivamente. Este cálculo está dominado por la descomposición en valores singulares de la matriz de ruteo G.

Los resultados experimentales que muestran los autores confirman que es posible conseguir resultados con un error relativo menor al 10 % monitoreando solamente entre un 20 % y un 30 % del número de caminos mínimos requeridos para obtener error nulo. Podría pensarse que los autores de este trabajo optimizan el anterior [3], buscando un número menor de rutas a medir, a partir de un compromiso entre cantidad de mediciones y precisión, basándose en una elección cuidadosa de las rutas sobre las cuales efectuarán las mediciones.

2.1.3. Segundo enfoque algebraico

Si bien el enfoque de este trabajo no está en el análisis de la selección de rutas a medir como en los casos mencionados previamente, Shavitt et al. [8] presentan otro enfoque algebraico que permite calcular características adicionales (como por ejemplo: distancias[6]) de una red que no fueron medidas explícitamente. Para llevar a cabo esto, cuentan con estaciones de medición denominadas *Tracers* ubicadas dentro de la red. Estos *Tracers* periódicamente informan sus mediciones a clientes, los cuales se encargarán de hacer un mapa de las distancias estimadas de la red. Aunque a priori parecería obvio elegir la herramienta `traceroute`[7] para llevar a cabo las mediciones, esto no es así por diferentes motivos:

1. El RTT (*round trip time* o tiempo de ida y vuelta) devuelto por el `traceroute` se considera sólo como una estimación grosera del RTT real.

[6] Este término es usado genéricamente, para denotar tanto al retardo de propagación como la demora, el número de saltos, o cualquier otra característica medible.

[7] Aplicación que sirve para descubrir la ruta que sigue un paquete entre dos nodos de una red IP (*Internet Protocol*). Se basa en un parámetro denominado TTL (*time to live* o tiempo a vivir) el cual es decrementado por cada nodo por el que atraviesa el paquete. El nodo en el cual el TTL se agota lo elimina y envía un mensaje al nodo origen ICMP (*Internet Control Message Protocol*) tipo 11 (*timeexceed*). De esta manera, enviando paquetes con distintos TTL, el origen puede ir recibiendo respuestas del primer nodo de la ruta, del segundo, etc. y así descubrir la ruta. Por cada valor del TTL se hacen habitualmente tres pruebas.

2. La sobrecarga de la red es significativamente mayor a la alternativa obvia del `traceroute`, el `ping`[8].

Por lo expresado anteriormente, una manera simple y efectiva que los autores proponen para medir la demora entre 2 extremos es hacerlo mediante la aplicación `ping`. La idea principal de este enfoque es que, usando las rutas medidas, se pueden identificar nodos a través de los cuales pasan rutas entre varios *Tracers*. A estos nodos, los autores los denominan *crossing points*. Lo expresado anteriormente se puede ver en la figura 2.6.

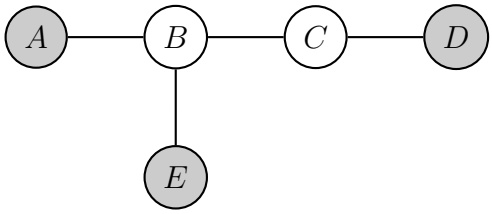


Figura 2.6: Red de ejemplo de 5 nodos.
(Imagen construída a partir de [8])

- A, B, C, D, E: son nodos.
- En A, D y E hay ubicados *Tracers*.
- B: es un *crossing point*.
- AB, EB, BCD: son segmentos.

De esta manera, el problema que resuelven en este trabajo es el siguiente: dado un conjunto de mediciones de extremo a extremo entre *Tracers* con sus rutas asociadas, encuentran todos los posibles segmentos[9] o grupos de segmentos consecutivos cuyas longitudes se pueden derivar.

A los fines de hacer la explicación más sencilla, se suponen diferentes características en la definición del modelo, como por ejemplo: la red es modelada

[8]Aplicación que sirve para determinar si existe una ruta hasta un determinado nodo, y en caso afirmativo obtener el RTT hasta y desde el nodo en cuestión. Se implementa en base a los mensajes ICMP tipo 8 (*echo request*) y tipo 0 (*echo reply*). Ver [9].

[9]Por definición, un segmento es un máximo sub-camino de una ruta, cuyos extremos son *Tracers* o *crossing points*, donde dentro de ese sub-camino no se incluye ningún *crossing point*.

como un grafo no orientado y las rutas entre *Tracers* se consideran cuasi estáticas. Es importante señalar que tanto la estructura del grafo como el tamaño del mismo son desconocidos por el algoritmo; se utiliza sólo con fines de análisis.

Con respecto al algoritmo, lo único conocido por éste, es el conjunto de mediciones extremo a extremo. Para poder obtener resultados consistentes, se deben tener en cuenta las siguientes consideraciones:

1. Interpretar las mediciones: esto hace referencia a cómo van a definirse las variables, por ejemplo: unidireccionales o bidireccionales. Una vez identificados los *crossing points*, las variables se escriben por segmentos, dado que hay enlaces individuales que siempre aparecen juntos, por lo que no podrán resolverse.

2. Segmentación y escritura de las ecuaciones: con las definiciones de las variables puede escribirse un sistema de ecuaciones lineales que describan las mediciones, de forma similar al que ha sido definido en la ecuación 2.1. El lado izquierdo de cada ecuación será la suma de todas las variables correspondientes a los enlaces que aparezcan en un camino medido. El lado derecho de cada ecuación será el valor de la característica medida, por ejemplo, la demora.

$$G \cdot x = b$$

   Sabiendo que:

   - g_{ij}=1 si el segmento j aparece en el camino i medido y g_{ij}=0 en otro caso.
   - x_j: representa una variable, j=1,...,m.
   - b_i: representa una medición, i=1,...,n.
   - m es la cantidad de segmentos luego de identificar los *crossing points*.
   - n es el número de mediciones.

3. Resolver tanto como sea posible: el sistema está generalmente subdefinido para algunas variables y sobredefinido para otras. Para hallar la solución, se transforma en un nuevo sistema que aisla todo lo que es resoluble. Como en los trabajos presentados previamente, esto se hace triangulando la matriz G, en este caso a través del método de eliminación de Gauss.

4. Complejidad de los diferentes pasos:

   - Identificación de *crossing points* y segmentos: $O(n)$
   - Escribir las ecuaciones: $O(nm)$
   - Triangulación por eliminación de Gauss: $O(nmm')$
   - Segmentos resolubles: menos de $O(nm'^2)$
   - Calcular longitud de los segmentos: $O(m^2)$

   siendo m' el rango de la matriz $\tilde{G}$.

Este método fue experimentado sobre redes reales como Internet y se ha obtenido que a mayor cantidad de *Tracers* más distancias descubre el algoritmo. Las ganancias sobre mediciones adicionales obtenidas fueron sustanciales, van desde el 70 % al 214 %.

Características comunes

Los trabajos presentados en las secciones 2.1.1, 2.1.2 y 2.1.3 poseen varias características comunes. Todos buscan métodos eficientes para obtener o estimar parámetros de la red, a partir de un número reducido de mediciones; para llevar a cabo lo anterior se basan en un modelo matricial de la red para despejar las incógnitas buscadas. Para hallar la solución, en todos los trabajos presentados, está implícita la triangulación de la matriz de ruteo con diferentes técnicas del álgebra lineal.

2.1.4. Mediciones a gran escala

Por otra parte, Allalouf et al. [10] presentan el diseño de un sistema para llevar a cabo mediciones de QoPC (*Quality of Path Characteristics*) a gran escala. Las características de los caminos que son recorridos por los paquetes de las aplicaciones son de gran importancia. Éstas determinan la idoneidad de los caminos para los requisitos de las aplicaciones, como por ejemplo: ancho de banda disponible, *jitter*, reordenamiento de paquetes, demora, etc.

En general, para este tipo de mediciones, es preferible utilizar métodos de medición en una sola dirección (*one way*) por sobre los métodos convencionales (ida y vuelta), ya que los mismos son menos susceptibles a la medición de ruido. El sistema IDM (*Inter-packet Delay Measurement*) fue diseñado y desarrollado por los autores para ser una poderosa herramienta para los investigadores, la cual permite realizar mediciones en una sola dirección usando

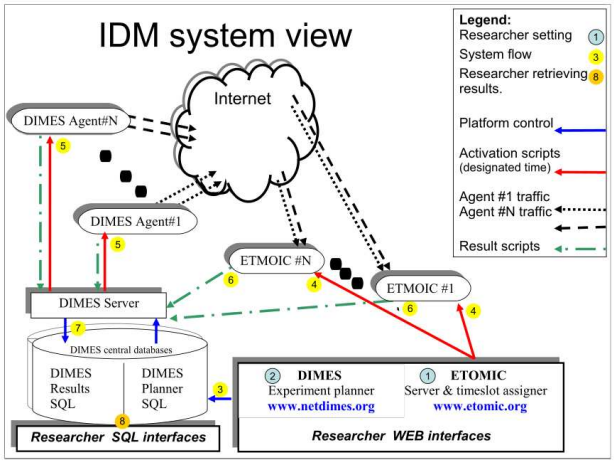


Figura 2.7: Esquema de funcionamiento del sistema IDM.
(Imagen extraída de [11])

trenes de paquetes emisores y receptores para determinar el QoPC. El sistema IDM fue diseñado como un extensión a la infraestructura de DIMES [1]. Como conjunto de receptores, los autores utilizan los servidores ETOMIC [11] (*European Traffic Observatory Measurement Infraestructure*) debido a su capacidad para medir el tiempo de llegada de paquetes con una precisión de microsegundos.

En la figura 2.7 se presenta un diagrama de flujo con la preparación, ejecución y la recuperación de los resultados del experimento IDM. Este procedimiento puede dividirse en varios pasos:

- Paso 1: dado que la infraestructura ETOMIC no permite la ejecución de más de un experimento en forma simultánea, deberá reservarse un espacio de tiempo a través de la interfaz web[10].
- Paso 2: el siguiente paso será solicitar la autorización correspondiente para realizar el experimento en el sitio web de DIMES[11].
- Paso 3: cuando el experimento es aprobado (este procedimiento se hace en forma manual), el mismo es almacenado en la base de datos central de DIMES.
- Paso 4: al tiempo seleccionado, los servidores ETOMIC comenzarán a ejecutar los agentes.

[10] www.etomic.org
[11] www.netdimes.org

- Paso 5: los servidores ETOMIC transferirán a los agentes DIMES designados los *scripts* a ejecutar.
- Paso 6: cuando termina la medición los resultados obtenidos son analizados por los agentes ETOMIC almacenando la información relevante.
- Paso 7,8: los resultados relevantes son transferidos a la base de datos central de DIMES, donde posteriormente podrán ser evaluados por los investigadores.

2.2. Problemas prácticos

Una vez definida la metodología para llevar a cabo la exploración de la red, la misma conlleva a tomar decisiones a la hora de procesar los datos obtenidos. De esta manera, luego de recolectar las mediciones de las rutas de extremo a extremo (mediante el uso de `traceroute`), el siguiente paso es procesar el conjunto de datos para construir un mapa de Internet. Esta tarea implica, al menos, dos pasos importantes: primero, la resolución de las identidades de los ruteadores que no responden, es decir, los ruteadores que responden al `traceroute` con un '*' para un determinado número de saltos; y segundo, resolver los *aliases* de los ruteadores que poseen varias direcciones IP.

2.2.1. Ruteadores que no responden

El primer problema ocurre cuando los rutedores tienen configurado no responder la expiración del TTL, por lo que muestran en su salida un '*'. Para resolver esto, Bilir et al. [12] proponen comparar todos los pares de caminos (por ejemplo p_1 y p_2) con ruteadores que no responden, y asignarle a estos ruteadores, la misma dirección IP de la siguiente manera:

- Suponiendo que p_1 y p_2 contienen un '*' entre dos ruteadores conocidos, si las entradas correspondientes al '*' tienen el mismo ruteador en el salto posterior e inferior para ambos caminos (p_1 y p_2) que tienen el mismo destino final, los autores consideran cada uno de esos ruteadores que no responden como el mismo ruteador y le asignan un único nombre, por ejemplo *ur.1* en este caso. Este es el caso que se muestra en la figura 2.8a.
- Suponiendo que p_1 y p_2 contienen dos '*' consecutivos entre dos ruteadores conocidos, de forma similar al caso anterior, los autores proponen agrupar estos ruteadores y darle el mismo nombre a los ruteadores en

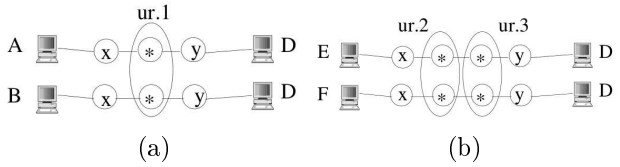


Figura 2.8: Resolución de rutedores que no responden.
(Imagen extraída de [12])

el mismo grupo si el grupo tiene el mismo ruteador en el salto posterior e inferior, donde el destino final es el mismo. Esto se muestra en la figura 2.8b.

- Los autores proponen descartar las trazas que tienen más de dos '*' consecutivos.

2.2.2. Resolución de *aliases*

Los ruteadores poseen múltiples interfaces cada una con una dirección IP diferente. De esta manera, un mismo ruteador podría aparecer en diferentes caminos con diferentes direcciones. Una de los primeros trabajos donde se estudió esta problemática fue en el desarrollado por Govindan y Tangmunarunkit [13].

Para resolver este problema, se busca identificar las direcciones IP que pertenecen a un mismo ruteador y combinarlas en un único nodo resultante para representarlo en el mapa de la topología. En el trabajo [14], Gunes y Sarac proponen un novedoso enfoque para solucionar este problema, llamado APAR (*Analytical and Probe-based Alias Resolver*). Dado un conjunto de caminos medidos, APAR usa las prácticas comunes de asignación de direcciones IP (ver [15]) para inferir los *aliases*. APAR incluye dos pasos: primero, análisis de las direcciones IP en el conjunto de los caminos medidos con el fin de identificar posibles subredes; y segundo, usar las subredes identificadas para resolver los *aliases*.

Capítulo 3

Fundamentos

En este capítulo se desarrollan los conceptos fundamentales sobre los cuales se construye el presente trabajo y la metodología implementada para su desarrollo. En primer lugar, en la sección 3.1, se introduce el problema de resolución de *aliases* en los ruteadores, problema que encuentra extensamente desarrollado en el trabajo [14]. A partir de lo explicado en el mencionado trabajo, se define un procedimiento de resolución que permite inferir alias de forma correcta.

Posteriormente, en la sección 3.2, se expone un simple ejemplo de una topología de red a partir del cual se muestra el sistema de ecuaciones lineales a resolver $G \cdot x = b$; de forma similar a lo encontrado en los trabajos [2, 3, 6]. Partiendo de este sistema, se presenta una transformación lineal que permite aumentar la cantidad de ceros de la matriz de ruteo G, obteniendo una matriz diferencias denominada G', la que permitirá acelerar el tiempo de cálculo de la solución. Luego se muestra el proceso de construcción de la matriz de ruteo diferencias G', a partir de las observaciones realizadas, su respectiva solución y cómo a partir de ésta, se construye el grafo de la red para su posterior análisis.

Finalmente, en la sección 3.3, se presenta un amplio panorama de los diferentes tipos de estudios que podrían efectuarse a partir de la información almacenada en la base de datos, especificando detalladamente cuáles serán nuestros casos de estudio.

3.1. Resolución de *aliases*

Como se mencionó anteriormente en la sección 2.2, uno de los principales problemas a la hora de procesar los datos es poder agrupar las direcciones IP pertenecientes a un mismo ruteador. Este problema es conocido en la bibliografía como *IP alias resolution* [14]. Los ruteadores poseen múltiples interfaces, cada una de ellas con una dirección IP diferente. De esta manera, el objetivo será identificar las direcciones IP que pertenecen a un mismo ruteador y combinarlas en un único vértice (o nodo) en el mapa de la topología resultante.

3.1.1. Asignación de direcciones IP

El mecanismo de asignación de direcciones IP sigue lineamientos generales definidos en RFC 2050 [15]. Básicamente, direcciones IP pertenecientes a un mismo dominio de red o ISP (*Internet Service Provider*) son divididas en rangos de subredes para optimizar el uso de las mismas. Cada subred tiene una dirección de red y cada interfaz, perteneciente a un ruteador o a una estación terminal dentro de la subred, obtiene su dirección IP a partir del rango de dirección de red asignado a la subred.

En general, hasta N interfaces pueden ser conectadas usando una subred $/x$ donde $x = 32 - \lceil log_2(N+2) \rceil$. Los primeros x bits de la dirección IP asignada determinan la dirección de red y los últimos $32 - x$ bits identifican las direcciones utilizables dentro de la subred.

3.1.2. Nomenclatura utilizada

En el desarrollo de este capítulo se utilizará una nomenclatura similar a la encontrada en [14]. A continuación se definen algunos conceptos que se utilizarán más adelante.

Definición (Grafo de ruteadores). Se define $G = (V, E)$ a un grafo de ruteadores, donde V representa un conjunto de vértices o nodos (por ejemplo: ruteadores y estaciones terminales) y E representa un conjunto de aristas (por ejemplo: enlaces de comunicación) conectando vértices en V. Cada vértice $v \epsilon V$ tiene una o más interfaces (i^v_1,i^v_2,...,$i^v_{grado(v)}$) donde $grado(v)$ representa el número de interfaces de v. Cada interfaz i^v_e de un vértice v tiene una dirección IP i^v_e, la cual es única en G. Una arista $e \epsilon E$ conecta dos vértices adyacentes v y w mediante la conexión de las interfaces i^v_e de v y i^w_f de w.

Definición (trace). La función $trace(p, r)$ devuelve como salida una lista de interfaces (una por cada vértice) empezando por p y terminando con r, esto es: $trace(p, r) = (i_g^p, ..., i_h^r)$. Se implementará a partir una herramienta similar al `traceroute`[1]. Cada valor de dirección de interfaz tendrá asociado un tiempo (RTT) y un número de salto (TTL).

Definición (Subred). Una $subred/x$ describe una red cuya dirección es $subred$ y cuya máscara de subred es de longitud x. Desde un punto de vista práctico, debido al tamaño limitado del espacio de direcciones IP (menos que 2^{32} direcciones), el tamaño de V es limitado. Esto indica que dadas dos direcciones IP, i_e^p y i_f^r, las mismas pueden estar dentro de la misma subred ó en dos subredes diferentes, basándose en la longitud x de la máscara de subred.

Por ejemplo, para una determinada subred que tiene como dirección de red 192.168.0.0/28, habrá 4 bits para identificar las direcciones IP individuales. Estos 4 bits permiten la identificación de 16 direcciones IP distribuidas de la siguiente manera:

- 1 dirección de red: 192.168.0.0
- 1 dirección *broadcast*: 192.168.0.15
- 14 direcciones IP válidas para asignación a dispositivos: 192.168.0.1-14

Ejemplo de cálculo

A continuación se presentan dos ejemplos donde se muestra cómo a partir de una dirección IP de un dispositivo y su máscara correspondiente se obtiene la dirección de subred. Esta operación, la cual podría pensarse como una operación lógica `AND` bit a bit, efectuada para diferentes direcciones IP, nos da la información necesaria para decidir si dos direcciones pertenecen o no a una misma subred.

Caso 1: 192.168.0.4/28 y 192.168.0.12/28. En este caso, vemos en la tabla 3.1 luego de efectuar la operación lógica, que ambas direcciones IP pertenecen a la misma subred: 192.168.0.0/28.

Caso 2: 192.168.0.4/28 y 192.168.0.42/28. En este segundo caso, vemos en la tabla 3.2 luego de efectuar la operación lógica, que estas direcciones IP pertenecen a diferentes subredes: 192.168.0.0/28 y 192.168.0.32/28.

[1] El funcionamiento de esta herramienta se detalla en la sección 4.3.2

	Notación decimal con puntos	Equivalencia binaria de 32 bits
IP	192.168.0.4	11000000 10101000 00000000 00000100
Máscara	255.255.255.240	11111111 11111111 11111111 11110000
Subred	192.168.0.0	11000000 10101000 00000000 00000000
IP	192.168.0.12	11000000 10101000 00000000 00001100
Máscara	255.255.255.240	11111111 11111111 11111111 11110000
Subred	192.168.0.0	11000000 10101000 00000000 00000000

Tabla 3.1: Caso 1

	Notación decimal con puntos	Equivalencia binaria de 32 bits
IP	192.168.0.4	11000000 10101000 00000000 00000100
Máscara	255.255.255.240	11111111 11111111 11111111 11110000
Subred	192.168.0.0	11000000 10101000 00000000 00000000
IP	192.168.0.42	11000000 10101000 00000000 00101010
Máscara	255.255.255.240	11111111 11111111 11111111 11110000
Subred	192.168.0.32	11000000 10101000 00000000 00100000

Tabla 3.2: Caso 2

Definición (**a** $\overset{x}{\longleftrightarrow}$ **b**)**.** Dadas dos direcciones de interfaces $a = i_e^p$, $b = i_f^r$, y una longitud x de máscara de subred, esta operación lógica devolverá *VERDADERO* si a y b pertenecen a la misma $subred/x$. Para otro caso, devolverá *FALSO*.

3.1.3. Identificación de subredes correspondientes a enlaces

La subred más chica en Internet es construida mediante enlaces punto a punto que conectan dos dispositivos. Una $subred/30$ es definida y usada para asignar direcciones IP a las interfaces en este tipo de redes. Subredes más grandes (con máscara /29 o mayores) no son consideradas, debido a que causan un desaprovechamiento de direcciones IP. Entonces, una $subred/30$ tendrá 2 bits para identificar direcciones IP, asignadas de la siguiente manera:

- Identificación de red: 1.
- Identificación de dirección *broadcast*: 1.
- Direcciones IP válidas para asignación a dispositivos: 2.

La metodología utilizada para la identificación de subredes correspondientes a enlaces se basa en aplicar, para cada dirección IP de una interfaz observada, una máscara /30 y almacenar esa información en una estructura de datos, la cual se define en la ecuación 3.2. Si bien al usar máscaras de esta longitud perdemos la posibilidad de identificar subredes más grandes, cabe destacar que, como se verá en la sección 3.1.4, no se introducirán errores al inferir un alias.

La asignación de direcciones IP en enlaces punto a punto puede ser utilizada para identificar segmentos de ruta simétricos en los caminos medidos. Dado un conjunto de mediciones, se compararán segmentos de diferentes caminos buscando direcciones IP, $a = i_e^p$ y $b = i_f^r$, evaluando la operación lógica $(a \overset{x}{\longleftrightarrow} b)$, donde $x = 30$. Una vez encontrada la coincidencia, deberán alinearse los caminos, para poder inferir correctamente los *aliases*[2]. Esto se explicará usando como base el ejemplo de la figura 3.1.

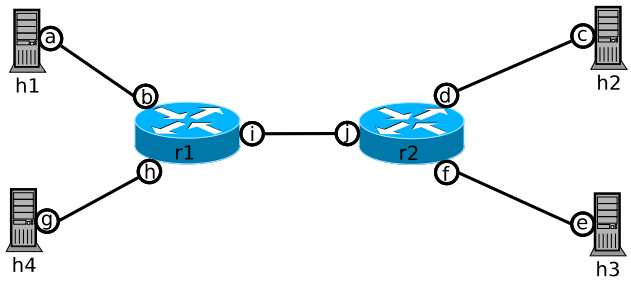


Figura 3.1: Red de 4 estaciones terminales y 2 ruteadores.

3.1.4. Identificación de *aliases*

Considerando la topología del ejemplo de la figura 3.1 donde h_1, h_2, h_3 y h_4 son estaciones terminales y r_1 y r_2 ruteadores, se observa que las letras minúsculas $a, b, ..., j$ representan direcciones IP de las interfaces. Se supone que se obtuvieron las siguientes mediciones: $trace(h_1, h_3) = (a, b, j, e)$ y $trace(h_2, h_4) = (c, d, i, g)$. Comparando una a una las direcciones IP de ambos

[2]En el presente trabajo, el análisis se basará en subredes con máscara de longitud 30, pudiéndose extender el análisis para máscaras menores.

caminos, se obtiene el siguiente resultado:

$$\begin{cases} (a \overset{30}{\longleftrightarrow} c) & = & FALSO \\ (a \overset{30}{\longleftrightarrow} d) & = & FALSO \\ \quad\quad ... & & \\ (j \overset{30}{\longleftrightarrow} i) & = & VERDADERO \\ \quad\quad ... & & \\ (e \overset{30}{\longleftrightarrow} g) & = & FALSO \end{cases}$$

Luego de evaluar todas las posibilidades, sólo en $(j \overset{30}{\longleftrightarrow} i)$ el resultado es $VERDADERO$. Siguiendo la misma metodología planteada en [14], a partir de esta relación de subredes pueden alinearse los caminos como se muestra en la figura 3.2.

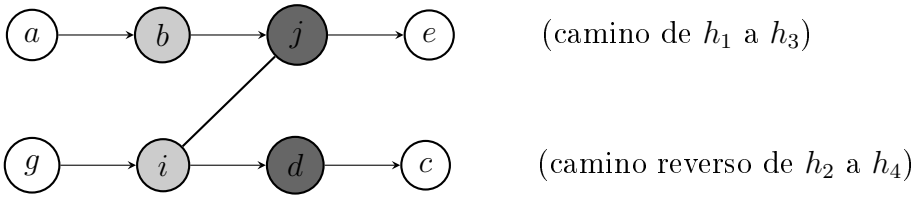


Figura 3.2: Alineación de caminos.

A partir de la alineación, se identifican dos pares de alias (b, i) y (d, j). Cabe aclarar que al usar máscaras de longitud 30, no hay posibilidad de cometer errores al inferir un alias. Sin embargo, deben hacerce algunas aclaraciones:

1. Deberá verificarse que cada elemento del alias inferido no sea parte del camino del otro elemento. Esto podría llegar a suceder si se encontrase algún *loop* de ruteo.

2. En el momento de la alineación de caminos podría darse el caso de inferir un alias donde una de sus interfaces no haya respondido, resultando $(i_j^v, *)$[3]; este alias deberá descartarse.

3.1.5. Procedimiento seguido para la resolución

El procedimiento efectuado para la resolución de *aliases* se basó a partir de los conceptos desarrollados en el trabajo [14]. Las mediciones que hemos

[3]Como se mencionó en la sección 2.2 aquellos ruteadores que tuvieren configurado no responder la expiración del TTL, mostrarán en su salida un '*'

realizado facilitan esta metodología de resolución, debido a que contamos con los caminos de ida y vuelta entre dos estaciones terminales h_i, h_j. En términos de la nomenclatura utilizada, se efectuaron mediciones $trace(h_i, h_j)$ y $trace(h_j, h_i)$. A continuación se detalla el procedimiento seguido para identificar los *aliases* a partir de los datos obtenidos:

1. Se almacenan las salidas de los diferentes $trace(h_i, h_j)$ en una estructura de datos[4] con el siguiente formato:

$$\begin{aligned} traces \;=\; & \{s_1 : [\ldots, i_e^p, i_f^r, \ldots], \\ & \;\; s_2 : [\ldots, i_e^p, i_g^s, \ldots], \\ & \;\; \vdots \\ & \;\; s_n : [\ldots, i_j^t, i_k^u, \ldots]\} \end{aligned} \tag{3.1}$$

   donde cada s_i es una secuencia de direcciones IP construida a partir de los datos medidos y observados, donde $s_i \neq s_j$.

2. Para cada dirección IP de las interfaces que componen una secuencia s_i se obtiene la dirección de subred y se almacena en una estructura de datos con el siguiente formato:

$$\begin{aligned} subredes \;=\; & \{subred_1/30 : [\ldots, s_i, s_j, \ldots] \\ & \;\; subred_2/30 : [\ldots, s_k, s_l, \ldots] \\ & \;\; \vdots \\ & \;\; subred_n/30 : [\ldots, s_i, s_n, \ldots]\} \end{aligned} \tag{3.2}$$

   la cantidad de elementos de cada $subred/30$ puede ir desde 1 hasta n, siendo n el número total de secuencias diferentes que incluyen esa subred.

3. Se itera sobre las subredes que tienen una cantidad de elementos mayores o iguales que 2, alineando las salidas de los $trace(h_i, h_j)$ observados

[4]Los diccionarios, también llamados matrices asociativas, deben su nombre a que son colecciones que relacionan una clave y un valor. Como clave puede usarse cualquier valor inmutable: números, cadenas, *booleanos*, tuplas, ... pero no listas o diccionarios, dado que son mutables. Esto es así porque los diccionarios se implementan como tablas *hash*, y al momento de introducir un nuevo par clave-valor en el diccionario, se calcula el *hash* de la clave para después poder encontrar la entrada correspondiente rápidamente. Las tablas *hash* proveen tiempo constante de búsqueda promedio $O(1)$, sin importar el número de elementos en la tabla, es decir que, esta propiedad garantiza una manera eficiente de buscar elementos. Se implementó en el lenguaje de programación Python (para más información, ver http://python.org/).

como se explicó en la sección 3.1.4, almacenando lo inferido en una estructura de datos con el siguiente formato:

$$\begin{aligned} \textit{vértices} \;=\; \{&v_1 : [\ldots, i_e^p, \ldots], \\ &v_2 : [\ldots, i_f^r, \ldots], \\ &v_3 : [\ldots, i_k^u, \ldots], \\ &\vdots \\ &v_n : [\ldots, i_j^t, \ldots]\} \end{aligned} \tag{3.3}$$

donde la cantidad de elementos de cada v_i puede ir desde 2 hasta $grado(v_i)$. En este punto, vale la pena aclarar, que una i_e^p sólo puede pertenecer a un v_i.

4. Una vez finalizada la resolución de *aliases*, se identifican aquellos vértices que sólo poseen una interfaz (como por ejemplo, todos los h_i). Como resultado se obtienen v_i con cantidad de elementos desde 1 hasta $grado(v_i)$

Criterio adoptado sobre las secuencias de salida

En una primera instancia de análisis sobre los datos obtenidos se tomaron las siguientes decisiones:

- Aquellos ruteadores que tuvieren configurado no responder la expiración del TTL, al ejecutar $trace(h_i, h_j)$ se obtendrá como respuesta para el valor de la dirección IP de la interfaz en cuestión '*'. Cuando se presenten estos casos, se reemplazará la secuencia de salida $trace(h_i, h_j) = (..., e, \text{'*'}, f, ...)$ por $trace(h_i, h_j) = (..., e, f, ...)$ suponiendo que e y f están conectados virtualmente[5].

- Si se observara como respuesta para el valor de una interfaz, una dirección IP perteneciente a rangos reservados, se procederá a operar como en el ítem anterior, es decir, se considerará un '*'. Los rangos reservados de direcciones IP comúnmente utilizados son los siguientes:

  - 10.0.0.0/8: 10.0.0.0 a 10.255.255.255
  - 172.16.0.0/12: 172.16.0.0 a 172.31.255.255
  - 192.168.0.0/16: 192.168.0.0 a 192.168.255.255

  la información completa puede encontrarse en el IANA[6] (*Internet Assigned Numbers Authority*).

[5] Este concepto se desarrollará en la sección 3.2.3
[6] http://www.iana.org/

3.2. Matriz de ruteo

A partir del ejemplo sencillo de la figura 3.3, se intenta describir cómo, a partir de una topología de red, puede transformarse el sistema en un modelo matricial. En este caso, el objetivo es calcular la demora en los enlaces x_1, x_2 y x_3, a partir de las mediciones de tiempos entre A, B y C. Se puede plantear el siguiente sistema de ecuaciones:

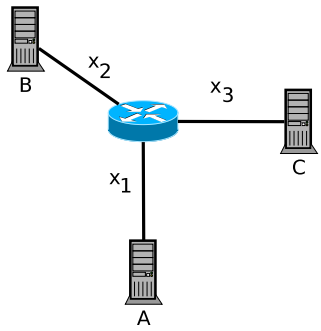


Figura 3.3: Red de 3 estaciones terminales y 1 ruteador.

$$\begin{cases} AB &= x_1 + x_2 \\ AC &= x_1 + x_3 \\ BC &= x_2 + x_3 \end{cases}$$

Empleando notación matricial, se puede expresar el mismo sistema de la siguiente manera:

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} AB \\ AC \\ BC \end{pmatrix}$$

$$G \cdot x = b \tag{3.4}$$

donde:

- G: matriz de ruteo, indica qué enlaces están incluídos en cada ruta, esto puede expresarse como:

$$G_{i,j} = \begin{cases} 1 & \text{si el enlace } j \text{ pertenece al camino } i \\ 0 & \text{en otro caso} \end{cases}$$

- x: incógnitas, valores individuales de los enlaces
- b: datos, mediciones agregadas de las rutas

En este ejemplo simple, hay 3 incógnitas y 3 ecuaciones independientes, con lo cual el sistema resulta compatible determinado y las incógnitas se pueden calcular. Pero en el caso general, G tiene rango deficiente, lo cual es equivalente a decir que hay menos ecuaciones independientes que incógnitas. En ese caso el sistema será indeterminado, por lo que la solución obtenida será aproximada.

3.2.1. Resolución del sistema $G \cdot x = b$

A partir de las características observadas en la matriz de ruteo G (matriz rala y sobredimensionada), donde $G \in \mathbb{R}^{m \times n}$, se buscó un método apropiado que sea capaz de resolver este sistema de ecuaciones líneales. El método utilizado se denomina LSQR [16] y se encuentra implementado en el módulo `scipy`[7] de `Python`.

Este método devuelve una solución por cuadrados mínimos del sistema $G \cdot x = b$ ó $min||b - G \cdot x||^2$, donde la matriz G puede ser cuadrada o rectangular (sobredeterminada o subdeterminada) y tener cualquier rango.

A grandes rasgos, LSQR utiliza un método iterativo para aproximar la solución. El número de iteraciones necesarias para alcanzar una cierta precisión depende fuertemente de la escala del problema. Por este motivo, escalas pobres de las filas o columnas de G deben evitarse, cuando fuere posible.

Preacondicionando la matriz G es otra manera de reducir el número de iteraciones. Si es posible resolver un sistema afín $M \cdot x = b$ eficientemente, donde M se aproxima a G de alguna manera útil (por ejemplo, $M - G$ tiene bajo rango o sus elementos son pequeños en relación a los de G), LSQR podría converger más rápidamente resolviendo el sistema $A \cdot M^{-1} \cdot z = b$, después de la cual x puede ser recuperado resolviendo $M \cdot x = z$.

3.2.2. Estructura de la matriz de ruteo

En la figura 3.4 se muestra una topología conocida para ilustrar el procedimiento de construcción de la matriz de ruteo a partir de datos obtenidos $trace(h_i, h_j)$.

[7] http://www.scipy.org/

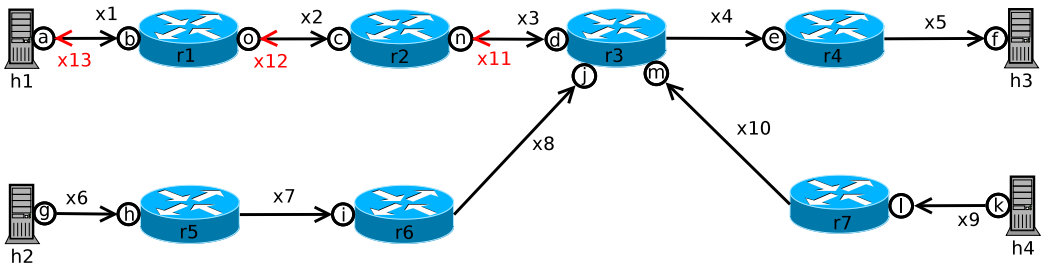


Figura 3.4: Red de 4 estaciones terminales y 7 ruteadores.

Considerando como origen a la estación terminal h_1 en el primer caso, a la estación terminal h_2 en el segundo caso, y como destino a la estación terminal h_3 en ambos casos, y posteriormente h_4 como origen y h_1 como destino; se obtienen como secuencias de salida los siguientes resultados: $trace(h_1, h_3) = (a, b, c, d, e, f)$, $trace(h_2, h_3) = (g, h, i, j, e, f)$ y $trace(h_4, h_1) = (k, l, m, n, o, a)$; como se muestra en las tablas 3.3, 3.4 y 3.5.

$trace = (h_1, h_3)$		
ttl	IP	tiempo
	a	0
1	b	t_{b1}
2	c	t_{c1}
3	d	t_{d1}
4	e	t_{e1}
5	f	t_{f1}

Tabla 3.3: Salida

$trace = (h_2, h_3)$		
ttl	IP	tiempo
	g	0
1	h	t_{h2}
2	i	t_{i2}
3	j	t_{j2}
4	e	t_{e2}
5	f	t_{f2}

Tabla 3.4: Salida

$trace = (h_4, h_1)$		
ttl	IP	tiempo
	k	0
1	l	t_{l3}
2	m	t_{m3}
3	n	t_{n3}
4	o	t_{o3}
5	a	t_{a3}

Tabla 3.5: Salida

Suponiendo que los enlaces son simétricos y que los paquetes siguen el mismo camino tanto a la ida como a la vuelta, de manera similar a lo desarrollado en [2, 3, 6], puede plantearse el siguiente sistema de ecuaciones lineales:

$$\begin{cases} x_1 & & & & & & & & = & t_{b1}/2 \\ x_1 & + & x_2 & & & & & & = & t_{c1}/2 \\ x_1 & + & x_2 & + & x_3 & & & & = & t_{d1}/2 \\ x_1 & + & x_2 & + & x_3 & + & x_4 & & = & t_{e1}/2 \\ x_1 & + & x_2 & + & x_3 & + & x_4 & + & x_5 & = & t_{f1}/2 \end{cases} \tag{3.5}$$

$$\begin{cases} & & & & x_6 & & & & = & t_{h2}/2 \\ & & & & x_6 & + & x_7 & & = & t_{i2}/2 \\ & & & & x_6 & + & x_7 & + x_8 & = & t_{j2}/2 \\ x_4 & + & & & x_6 & + & x_7 & + x_8 & = & t_{e2}/2 \\ x_4 & + & x_5 & + x_6 & + & x_7 & + & x_8 & = & t_{f2}/2 \end{cases} \tag{3.6}$$

$$\begin{cases} x_9 & & & & & & & & = & t_{l3}/2 \\ x_9 & + & x_{10} & & & & & & = & t_{m3}/2 \\ x_9 & + & x_{10} & + & x_{11} & & & & = & t_{n3}/2 \\ x_9 & + & x_{10} & + & x_{11} & + & x_{12} & & = & t_{o3}/2 \\ x_9 & + & x_{10} & + & x_{11} & + & x_{12} & + x_{13} & = & t_{a3}/2 \end{cases} \tag{3.7}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \end{pmatrix} = \begin{pmatrix} t_{b1}/2 \\ t_{c1}/2 \\ t_{d1}/2 \\ t_{e1}/2 \\ t_{f1}/2 \\ t_{h2}/2 \\ t_{i2}/2 \\ t_{j2}/2 \\ t_{e2}/2 \\ t_{f2}/2 \\ t_{l3}/2 \\ t_{m3}/2 \\ t_{n3}/2 \\ t_{o3}/2 \\ t_{a3}/2 \end{pmatrix}$$

$$G \cdot x = b \tag{3.8}$$

En general, la matriz de ruteo G, es una matriz rala. Observando las propiedades de esta matriz, se propone aplicar una transformación lineal que aumente la cantidad de ceros de G. Esta transformación lineal consta de ir restando las ecuaciones a medida que analizamos un mismo $trace(h_i, h_j)$, por ejemplo, tomamos las primeras dos ecuaciones del sistema 3.5:

$$\begin{cases} x_1 & & = & t_{b1}/2 \\ x_1 & + x_2 & = & t_{c1}/2 \end{cases} \tag{3.9}$$

si restamos ambas ecuaciones:

$$x_1 + x_2 - x_1 = t_{c1}/2 - t_{b1}/2 \tag{3.10}$$

obtenemos como resultado:

$$x_2 = (t_{c1} - t_{b1})/2 \tag{3.11}$$

Para el sistema de ecuaciones 3.5:

$$\begin{cases} x_1 & & & & & = & t_{b1}/2 \\ & x_2 & & & & = & (t_{c1} - t_{b1})/2 \\ & & x_3 & & & = & (t_{d1} - t_{c1})/2 \\ & & & x_4 & & = & (t_{e1} - t_{d1})/2 \\ & & & & x_5 & = & (t_{f1} - t_{e1})/2 \end{cases} \tag{3.12}$$

para el sistema de ecuaciones 3.6:

$$\begin{cases} & & x_6 & & & = & t_{h2}/2 \\ & & & x_7 & & = & (t_{i2} - t_{h2})/2 \\ & & & & x_8 & = & (t_{j2} - t_{i2})/2 \\ x_4 & & & & & = & (t_{e2} - t_{j2})/2 \\ & x_5 & & & & = & (t_{f2} - t_{e2})/2 \end{cases} \tag{3.13}$$

y para el sistema de ecuaciones 3.7:

$$\begin{cases} x_9 & & & & & = & t_{l3}/2 \\ & x_{10} & & & & = & (t_{m3} - t_{l3})/2 \\ & & x_{11} & & & = & (t_{n3} - t_{m3})/2 \\ & & & x_{12} & & = & (t_{o3} - t_{n3})/2 \\ & & & & x_{13} & = & (t_{a3} - t_{o3})/2 \end{cases} \tag{3.14}$$

Como resultado obtenemos la siguiente matriz:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \end{pmatrix} = \begin{pmatrix} t_{b1}/2 \\ (t_{c1} - t_{b1})/2 \\ (t_{d1} - t_{c1})/2 \\ (t_{e1} - t_{d1})/2 \\ (t_{f1} - t_{e1})/2 \\ t_{h2}/2 \\ (t_{i2} - t_{h2})/2 \\ (t_{j2} - t_{i2})/2 \\ (t_{e2} - t_{j2})/2 \\ (t_{f2} - t_{e2})/2 \\ t_{l3}/2 \\ (t_{m3} - t_{l3})/2 \\ (t_{n3} - t_{m3})/2 \\ (t_{o3} - t_{n3})/2 \\ (t_{a3} - t_{o3})/2 \end{pmatrix}$$

la cual podría pensarse como una matriz diferencias. En esta matriz diferencias obtenida, se evidencia el aumento de los elementos nulos de la misma. Luego de aplicar la transformación lineal, se obtiene un sistema equivalente, definido según la ecuación 3.15:

$$G' \cdot x = b' \tag{3.15}$$

Si bien el método LSQR definido en la sección 3.2.1 puede aplicarse directamente sobre la matriz de ruteo original, una de las ventajas de la transformación lineal que hemos definido en 3.10 es que permitirá por un lado, acelerar el tiempo de cálculo de la solución y por otro, obtener una solución más precisa; considerando, en este caso, la precisión como el error cuadrático medio.

3.2.3. Construcción de G' a partir de datos observados

Podría decirse que la matriz diferencias obtenida en 3.15 representa un enfoque novedoso dentro de esta problemática. En el desarrollo de esta sección se presenta un método eficiente para construir la matriz de ruteo diferencias G'.

Teniendo en cuenta las aclaraciones realizadas previamente, se procede a analizar los datos obtenidos. Dado que el objetivo es construir un mapa de

ruteadores, se utilizará la información obtenida en la sección 3.1.4, referida a la resolución de *aliases*. Las observaciones que hemos realizado favorecen esta resolución, dado que en la mayoría de los casos, contamos con los caminos de ida y vuelta entre dos estaciones terminales.

Con el fin de facilitar la comprensión del objetivo buscado, se muestra cual sería el resultado del procedimiento de ejecución definido en 3.1.5 para la obtención de los vértices, utilizando el ejemplo de la figura 3.4:

$$\begin{aligned} vértices \;=\; \{&v_1 : [b, o], \\ &v_2 : [c, n], \\ &v_3 : [d, j, m], \\ &v_4 : [e], \\ &v_5 : [h], \\ &v_6 : [i], \\ &v_7 : [l], \\ &v_8 : [a], \\ &v_9 : [g], \\ &v_{10} : [f], \\ &v_{11} : [k]\} \end{aligned} \tag{3.16}$$

Incógnitas y mediciones

Se itera sobre los diferentes $trace(h_i, h_j)$ agrupando pares de interfaces consecutivas que pertenecen a una misma secuencia de salida s_i. Para cada valor de dirección IP de interfaz obtenido, se consulta a que vértice pertenece, concatenando estos valores para formar un enlace incógnita. Esto se ejemplifica a partir del grafo de la figura 3.4. Analizando el $trace(h_1, h_3) = (a, b, c, d, e, f)$ se observa lo siguiente:

$$\begin{cases} a \in v_8 \\ b \in v_1 \end{cases} \implies v_8|v_1 = x_1 \implies t(x_1) = [t_{b1}/2]$$

$$\begin{cases} b \in v_1 \\ c \in v_2 \end{cases} \implies v_2|v_1 = x_2 \implies t(x_2) = [(t_{c1} - t_{b1})/2]$$

donde el símbolo | denota la concatenación. Se procede de igual forma para todos los $trace(h_i, h_j)$ obtenidos.

Dado que en general los vértices poseen varias interfaces, una misma incógnita podría aparecer más de una vez. Cabe aclarar que sólo se definen una vez (por los dos primeros vértices que aparecen); cada vez que

se repita una incógnita, se almacenará su valor de tiempo. Analizando el $trace(h_1, h_3) = (a, b, c, d, e, f)$ se observa lo siguiente:

$$\left\{ \begin{array}{lcl} d & \in & v_3 \\ e & \in & v_4 \end{array} \right. \implies v_4|v_3 = x_4 \implies t(x_4) = [(t_{e1} - t_{d1})/2]$$

Analizando el $trace(h_2, h_3) = (g, h, i, j, e, f)$ se encuentra nuevamente x_4:

$$\left\{ \begin{array}{lcl} j & \in & v_3 \\ e & \in & v_4 \end{array} \right. \implies v_4|v_3 = x_4 \implies t(x_4) = [(t_{e1} - t_{d1})/2, (t_{e2} - t_{j2})/2]$$

esto quiere decir que, al encontrar un enlace ya definido, sólo se agregará su valor medido, formando un vector de tiempos medidos para esa incógnita.

Como resultado de este paso se obtiene lo siguiente:

$$\begin{array}{rcl} incógnitas & = & \{v_1|v_8 : x_1, \\ & & \ v_2|v_1 : x_2, \\ & & \ \vdots \\ & & \ v_j|v_i : x_n\} \end{array} \tag{3.17}$$

$$\begin{array}{rcl} mediciones & = & \{t(x_1) : [t_1(x_1), t_2(x_1), \ldots, t_x(x_1)] \\ & & \ t(x_2) : [t_1(x_2), t_2(x_2), \ldots, t_y(x_2)] \\ & & \ \vdots \\ & & \ t(x_r) : [t_1(x_r), t_2(x_r), \ldots, t_z(x_r)]\} \end{array} \tag{3.18}$$

Tiempos

Debido a que en redes como Internet el ruteo de paquetes por un mismo camino no está garantizado y que los paquetes que viajan en la red pueden sufrir diferentes retardos, procesamientos, etc., se observan al hacer las restas de los tiempos asociados a las interfaces valores negativos de los mismos. Esto puede expresarse de la siguiente manera:

$$t_{wj} - t_{vj} < 0 \tag{3.19}$$

dado que no tiene sentido pensar en tiempos negativos en estos casos de análisis, se deberá tomar alguna decisión a la hora de encontrarnos con ellos.

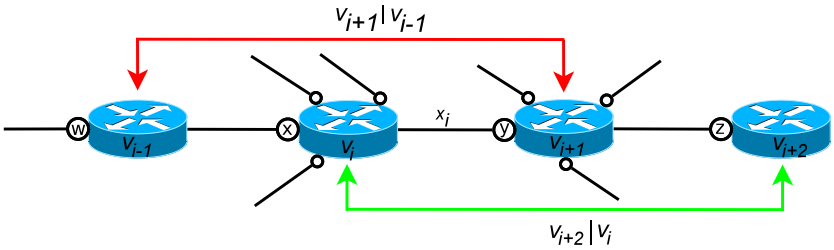


Figura 3.5: Análisis de tiempos negativos.

Teniendo en cuenta que el vector de mediciones para una incógnita posee varios valores de tiempos, se conservarán sólo aquellos que sean mayores que cero. Esto puede generar vectores nulos en aquellos casos en que todos los valores de tiempos sean negativos. A continuación, se enumeran las diferentes soluciones aplicadas:

1. **Enlace virtual:** este concepto puede encontrarse desarrollado en [2]. Podría decirse que los enlaces virtuales se utilizan cuando hay enlaces (incógnitas) que no pueden calcularse individualmente; estos enlaces también son denominados como no identificables. El procedimiento de creación de enlaces virtuales se explicará en base al ejemplo de la figura 3.5. En este caso, todos los tiempos $(t_{yu} - t_{xu})$ para x_i son negativos, por lo que el enlace incógnita $v_{i+1}|v_i = x_i$ deberá reemplazarse. Se buscarán posibilidades de reemplazo hacia vértices sucesores y predecesores, por lo que habrá dos candidatos: $v_{i+1}|v_{i-1}$ y $v_{i+2}|v_i$, tal como indican las flechas de la figura. En base a estas dos posibilidades se adoptará el siguiente criterio:

   - Se seleccionará el enlace virtual que posea un menor tiempo medido ($[t_{zu} - t_{xu}]$ ó $[t_{yu} - t_{wu}]$).
   - Si el enlace elegido no existiera, se creará, como se explicó anteriormente en 3.17.
   - Si el enlace elegido ya existe, se agregará el valor de tiempo, como se explicó anteriormente en 3.18.
   - Al finalizar el proceso de estos datos, el vértice v_i o v_{i+1} quedará absorbido en un enlace virtual, por lo que los enlaces que se conectan con el vértice en cuestión se reconectarán al vértice inmediato inferior.
   - De no encontrarse en ninguno de los sentidos al menos un tiempo positivo, se continua según siguiente punto.

2. **Reemplazo por el inverso:** en el proceso de armado de la matriz los enlaces (aristas del grafo) son considerados dirigidos, esto quiere decir que $e(p,r) \neq e(r,p)$. Si no hubiere solución para el punto anterior, se buscará si existe el enlace en el sentido inverso. En caso de que exista y que el mismo contara con valores de tiempos no negativos, se reemplazará; esto generará que $e(p,r) = e(r,p)$.

3. **Descarte:** si no hubiere solución posible según los puntos anteriores, se descartará esa medición.

3.2.4. Dimensiones de G'

Columnas

La cantidad de columnas n, que en este caso será equivalente a la cantidad de enlaces incógnitas, se obtendrá calculando la cantidad de claves del diccionario incognitas, el cual fue definido en 3.17.

Filas

La cantidad de filas m, que en este caso será equivalente a la cantidad de mediciones, se obtendrá sumando para cada clave $t(x_i)$ definido en 3.18 la longitud de su vector de tiempos asociado, resultanto $m = x + y + \ldots + z$.

3.2.5. Escritura de la matriz

Teniendo en cuenta que G' es una matriz rala, una vez conocidas sus dimensiones se escribirán los coeficientes no nulos en las coordenadas correspondientes. Para esto se recorren los diccionarios definidos en 3.17 y 3.18, construyéndose ternas de valores con el formato $[i, j, t_{lu}(x_i)]$, donde:

- i representa el número de fila de G' el cual se incrementará en 1 para cada valor medido.

- j representa el número de columna de G' el cual se incrementará en 1 para cada valor de clave diferente.

- $t_{lu}(x_i)$ representa el valor medido t_l para el enlace incógnita x_i el cual fue observado en la secuencia s_u.

El módulo `sparse` de `Python` brinda la posibilidad de trabajar con matrices de estas características. Para el ejemplo de la matriz 3.2.2 y en base

a lo mencionado anteriormente donde $m = 15$ y $n = 13$, la matriz G' rala resultante será:

$$\begin{array}{ll}
(0,0) & 1 \\
(1,1) & 1 \\
(2,2) & 1 \\
(3,3) & 1 \\
(4,4) & 1 \\
(5,5) & 1 \\
(6,6) & 1 \\
(7,7) & 1 \\
(8,3) & 1 \\
(9,4) & 1 \\
(10,8) & 1 \\
(11,9) & 1 \\
(12,10) & 1 \\
(13,11) & 1 \\
(14,12) & 1
\end{array}$$

debemos aclarar que `Python` comienza la indexación en el valor 0.

Para poder efectuar operaciones con matrices ralas, se debe convertir el vector de mediciones en este formato, el cual resultará de la siguiente manera:

$$\begin{array}{ll}
(0,0) & t_{b1}/2 \\
(1,0) & (t_{c1} - t_{b1})/2 \\
(2,0) & (t_{d1} - t_{c1})/2 \\
(3,0) & (t_{e1} - t_{d1})/2 \\
(4,0) & (t_{f1} - t_{e1})/2 \\
(5,0) & t_{h2}/2 \\
(6,0) & (t_{i2} - t_{h2})/2 \\
(7,0) & (t_{j2} - t_{i2})/2 \\
(8,0) & (t_{e2} - t_{j2})/2 \\
(9,0) & (t_{f2} - t_{e2})/2 \\
(10,0) & t_{l3}/2 \\
(11,0) & (t_{m3} - t_{l3})/2 \\
(12,0) & (t_{n3} - t_{m3})/2 \\
(13,0) & (t_{o3} - t_{n3})/2 \\
(14,0) & (t_{a3} - t_{o3})/2
\end{array}$$

donde las dimensiones de $b \in \mathbb{R}^{m\times 1}$.

Vector soluciones

Una vez que contamos con la matriz y las mediciones en el mismo formato, se ejecuta la función `lsqr`, la cual devuelve en un vector los valores de las incógnitas buscadas.

3.2.6. Grafo

Para representar los datos obtenidos se construirá un grafo pesado en un formato estandarizado. Este formato permitirá luego cargar el grafo obtenido en diferentes programas de análisis como: `Network Workbench`[8] y `Scilab`[9].

El grafo se representará en un archivo de texto, el cual consta de tres valores por fila, resultando finalmente un archivo de tres columnas: los primeros dos valores de cada fila representan un par de vértices que están conectados entre si; el tercer valor representará el peso, que en este caso será la solución x_i encontrada al resolver el sistema $G \cdot x = b$.

Nuevamente, utilizando el ejemplo de la figura 3.4, el grafo asociado presentaría la siguiente forma:

$$\begin{array}{lll} v_1 & v_8 & x_1 \\ v_2 & v_1 & x_2 \\ v_3 & v_2 & x_3 \\ v_4 & v_3 & x_4 \\ v_{10} & v_4 & x_5 \\ v_5 & v_9 & x_6 \\ v_6 & v_5 & x_7 \\ v_3 & v_6 & x_8 \\ v_7 & v_{11} & x_9 \\ v_3 & v_7 & x_{10} \\ v_2 & v_3 & x_{11} \\ v_1 & v_2 & x_{12} \\ v_8 & v_1 & x_{13} \end{array} \tag{3.20}$$

Para poder ser interpretados por los programas de análisis previamente mencionados, cada v_i deberá representarse por un identificador único el cual deberá ser un número natural. En base a la forma en que se construye la

[8] nwb.cns.iu.edu/
[9] www.scilab.org/

estructura de datos *vertices*, cada v_i podrá ser representado únivocamente por su valor de subíndice asociado.

3.3. Usos de la información almacenada

Como se mencionó en la sección 3.1.2 cada interfaz observada en $trace(p, r) = (i_g^p, ..., i_h^r)$ trae consigo un valor de RTT y un número de salto, lo que nos permitirá implementar diferentes vectores mediciones. Para poder llegar a construir un mapa anotado representativo de la realidad, las observaciones deben ser prolongadas en el tiempo. Esto, sumado a la propiedad de las redes como Internet donde un mismo flujo de datos entre un origen y un destino puede atravesar diferentes caminos, hace que la salida de un mismo $trace(p, r)$ en dos instantes de tiempo diferentes puedan llegar a arrojar distintos resultados.

Las mediciones realizadas durante un intervalo de tiempo prolongado permiten un gran almacenamiento de información. La misma podrá ser utilizada para calcular diferentes parámetros de estudio de redes, como por ejemplo: estadística de tráfico, demoras, ancho de banda, disponibilidad, tasa de pérdida, etc. Nuestros casos de estudio, por cuestiones de tiempo, se limitarán a la construcción de mapas de demoras con diferentes características de tiempos, los cuales se detallan en las secciones 3.3.1, 3.3.2 y 3.3.3.

3.3.1. Tiempos mínimos

Teniendo en cuenta lo mencionado anteriormente, se debe aclarar qué significa un tiempo mínimo. Dado un origen p y un destino r al ejecutar la función $trace(p, r)$ se almacenarán tantas secuencias s_i como salidas diferentes haya. Esto quiere decir que si a lo largo del tiempo de observación se encontraren cambios en las rutas, ya sea por balanceo de carga, cambios en la topología, etc., la salida de $trace(p, r)$ no será única, pudiéndose repetir cada secuencia s_i una cierta cantidad de veces. De esta manera, el tiempo mínimo asociado a una dirección IP de una interfaz, será el menor de todos los tiempos en el marco de una misma secuencia s_i.

Esto se explicará en base a la siguiente situación: luego de observaciones prolongadas en el tiempo, para $trace(h_1, h_2)$ se obtienen dos diferentes secuencias, las cuales se representan en las tablas 3.6 y 3.7, donde s_1 y s_2 se repiten u y v veces respectivamente. En base a lo observado se puede recons-

$trace = (h_1, h_2)$		
ttl	IP	tiempo
	a	0
1	b	$[t_{b1_1}, t_{b1_2}, ..., t_{b1_u}]$
2	c	$[t_{c1_1}, t_{c1_2}, ..., t_{c1_u}]$
3	d	$[t_{d1_1}, t_{d1_2}, ..., t_{d1_u}]$
4	e	$[t_{e1_1}, t_{e1_2}, ..., t_{e1_u}]$
5	f	$[t_{f1_1}, t_{f1_2}, ..., t_{f1_u}]$

Tabla 3.6: Salida de s_1

$trace = (h_1, h_2)$		
ttl	IP	tiempo
	a	0
1	b	$[t_{b2_1}, t_{b2_2}, ..., t_{b2_v}]$
2	c	$[t_{c2_1}, t_{c2_2}, ..., t_{c2_v}]$
3	g	$[t_{g2_1}, t_{g2_2}, ..., t_{g2_v}]$
4	h	$[t_{h2_1}, t_{h2_2}, ..., t_{h2_v}]$
5	f	$[t_{f2_1}, t_{f2_2}, ..., t_{f2_v}]$

Tabla 3.7: Salida de s_2

truir una topología como la de la figura 3.6, donde se observa claramente dos caminos posibles entre h_1 y h_2.

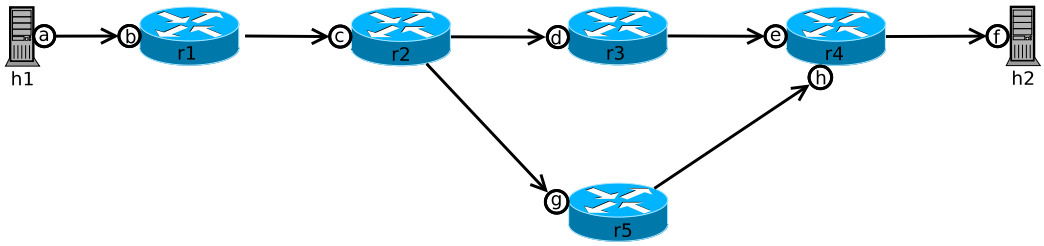


Figura 3.6: Red de 2 estaciones terminales y 5 ruteadores.

Durante el proceso de construcción de la matriz, en el caso de los tiempos mínimos, se obtendrá cada uno de ellos a partir de:

$$t_{qi_{\text{mín}}} = min[t_{qi_1}, t_{qi_2}, ..., t_{qi_u}] \tag{3.21}$$

donde:

- q representa una dirección IP de una interfaz.
- i representa una secuencia s_i determinada.

3.3.2. Tiempos mínimos modelados

Los ruteadores son máquinas de tiempo discreto, esto quiere decir que, cuando no hubiere ningún tipo de priorización de tráfico aplicado, procesarán los paquetes a medida que los vaya recibiendo con un tiempo aleatorio de atención, formando una cola de espera como se muestra en la figura 3.7.

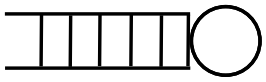


Figura 3.7: Cola de paquetes en un ruteador.

Si la cola estuviere siempre vacía, el tiempo de servicio o atención del ruteador puede modelarse en base a un tiempo que responde a una campana de Gauss. Dado que las mediciones se efectuaron durante un tiempo prolongado, se puede observar esta propiedad mencionada. En el gráfico de la figura 3.8 se muestra un histograma de datos observados (reales), con su correspondiente función de ajuste *gaussiana* y el error del mismo.

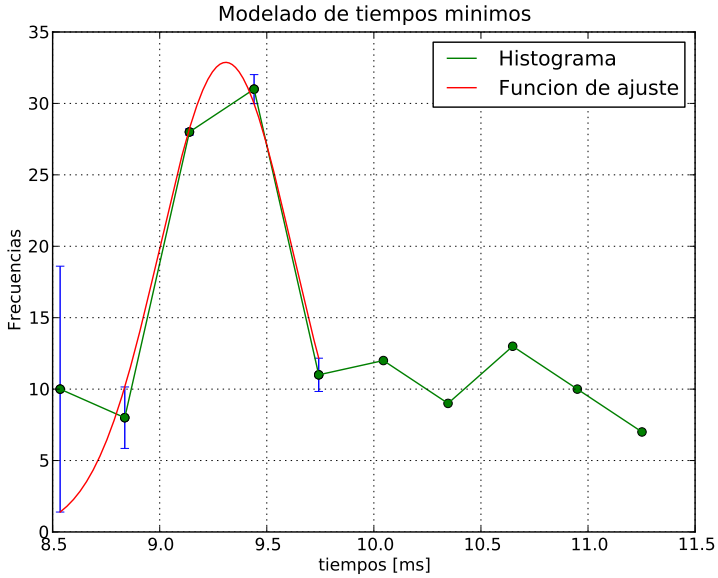


Figura 3.8: Tratamiento estadístico de datos observados.

La modelización del tiempo mínimo tiene como objetivo encontrar un nuevo tiempo mínimo que se ajuste con mayor precisión a la realidad del tiempo de servicio de los ruteadores. Este valor se obtendrá a partir de los parámetros característicos de la función de ajuste *gaussiana*: $\hat{t}_{\text{mín}} = \hat{\mu}$. Para el caso tomado de ejemplo en la figura 3.8, el menor de los tiempos de servicio encontrado fue: $t_{\text{mín}} = 8,53ms$ y el tiempo mínimo modelado resultó: $\hat{t}_{\text{mín}} = 9,31ms$. A continuación se detalla cómo se construye el histograma en base a los datos observados y cómo se realiza el ajuste.

Histograma

A partir del vector de datos obtenido se construirá un histograma con el fin de modelizar estadísticamente el tiempo mínimo. Dado un vector de observaciones $[t_{qi_1}, t_{qi_2}, ..., t_{qi_u}]$, la función histograma $\mathcal{M}(t)$ recibirá como parámetros dicho vector y el número de *bins*, el cual es calculado según la ecuación 3.22.

$$bins = \left\lfloor \frac{\delta}{\frac{a_m}{50}} \right\rfloor \tag{3.22}$$

donde:

- $a_M = arg_{\text{máx}}(t_{qi_1}, t_{qi_2}, ..., t_{qi_u})$
- $a_m = arg_{\text{mín}}(t_{qi_1}, t_{qi_2}, ..., t_{qi_u})$
- $\delta = a_M - a_n$

y devolverá un conjunto de pares de valores denominados frecuencia de repetición e intervalo de tiempo.

Estimación

La estimación del histograma se realizará mediante una función de Gauss, definida según la ecuación 3.23. La implementación se realiza con el módulo `optimize`, el cual pertenece a `scipy`; ambos dentro del contexto de `Python`.

$$\hat{\mathcal{N}}(t) = \hat{a} \cdot e^{\frac{(t-\hat{\mu})^2}{2 \cdot \hat{\sigma}^2}} \tag{3.23}$$

Los parámetros de la función $\hat{\mathcal{N}}(t)$ a estimar serán:

$$\begin{cases} \hat{a}\text{: amplitud estimada} \\ \hat{\mu}\text{: valor medio estimado} \\ \hat{\sigma}\text{: desvío estándar estimado} \end{cases}$$

La función de error, la cual mide la distancia a la función de ajuste, queda definida por la ecuación 3.24.

$$\varepsilon(t) = \hat{\mathcal{N}}(t) - \mathcal{M}(t) \tag{3.24}$$

donde $\mathcal{M}(t)$ representa los valores del histograma calculado, esto deberá ser, un valor de frecuencia de repetición y un valor de tiempo.

Una de las condiciones para que el algoritmo de optimización comience su ejecución, es que los valores a estimar tengan un valor inicial, los cuales fueron elegidos de la siguiente manera:

$$\begin{cases} \hat{a}_0\text{: frecuencia máxima observada} \\ \hat{\mu}_0\text{: valor correspondiente para la frecuencia máxima} \\ \hat{\sigma}_0 = \hat{\mu}_0 - arg_{\text{mín}} \end{cases}$$

donde $arg_{\text{mín}}$ es el valor mínimo de los valores de abscisas representados en el histograma.

A partir del histograma calculado, se define un intervalo de ajuste tomando como límite inferior al índice de la abscisa $\hat{\mu}_0$ incrementado en 1 y como límite superior al índice de la mayor abscisa del histograma. Tomando como referencia el ejemplo de la figura 3.8, estos valores son $lim_{\text{ínf}} = 5$ y $lim_{\text{sup}} = 10$.

Para los valores dentro del intervalo definido se realiza el ajuste *gaussiano*, obteniendo diferentes funciones de ajuste en base a la longitud del intervalo elegido; esto quiere decir que el ajuste se realiza con un histograma reducido. Para el ejemplo de la figura 3.8 se deberían obtener seis ajustes diferentes, donde:

$$\begin{cases} \text{Cantidad de muestras para } \hat{\mathcal{N}}_1(t)\text{:} & 5, [8,53 : 9,73]ms \\ \text{Cantidad de muestras para } \hat{\mathcal{N}}_2(t)\text{:} & 6, [8,53 : 10,11]ms \\ \text{Cantidad de muestras para } \hat{\mathcal{N}}_3(t)\text{:} & 7, [8,53 : 10,37]ms \\ \text{Cantidad de muestras para } \hat{\mathcal{N}}_4(t)\text{:} & 8, [8,53 : 10,63]ms \\ \text{Cantidad de muestras para } \hat{\mathcal{N}}_5(t)\text{:} & 9, [8,53 : 10,92]ms \\ \text{Cantidad de muestras para } \hat{\mathcal{N}}_6(t)\text{:} & 10, [8,53 : 11,27]ms \end{cases}$$

La decisión sobre que $\hat{\mathcal{N}}_k(t)$ ajusta con mayor exactitud se toma en base al menor coeficiente de variación obtenido ρ en cada ajuste realizado, definido según la ecuación 3.25.

$$\rho_{\%} = 100 \cdot \frac{\hat{\sigma}}{\hat{\mu}} \tag{3.25}$$

Nuevamente, tomando como referencia el ejemplo de la figura 3.8, el menor valor de ρ_k se obtiene para $k = 1$. De esta manera la función de ajuste sólo es representada en el intervalo correspondiente a $\hat{\mathcal{N}}_1(t)$.

Casos particulares

En aquellos casos donde se presenten alguna de las condiciones que se mencionan a continuación no podrá realizarse el histograma, por lo tanto se considerará como tiempo mínimo al menor de los tiempos observados.

- Cuando la cantidad de muestras observadas sea menor a un límite l, donde $l = 10$.
- Cuando la frecuencia máxima observada se encuentre en el primer valor de tiempos del histograma.
- Cuando la frecuencia máxima observada se encuentre en el máximo valor de tiempos del histograma.
- Cuando $\hat{\mu}$ devuelva valores negativos (el ajuste se encontrará en la parte descendente de la campana de Gauss).

3.3.3. Tiempos medios

Utilizando la información presentada en las tablas 3.6 y 3.7, otro caso de estudio será para el valor promedio de los tiempos observados, calculado de la siguiente manera:

$$\overline{t}_{qi} = \frac{1}{u} \sum_{j=1}^{u} t_{qi_j} \tag{3.26}$$

donde, al igual que lo explicado en 3.21:

- q representa una dirección IP de una interfaz.
- i representa una secuencia s_i determinada.

Para este caso de estudio propuesto, se esperaría observar el tiempo medio de espera en cada enlace. Siendo conscientes de que este modelo es extremadamente simple, el agregado de complejidad excede el marco de este trabajo, pero tenemos en cuenta que, para comenzar un análisis más profundo, habría que estudiar las distribuciones de los tiempos en las colas de los ruteadores.

Capítulo 4

Despliegue de los experimentos

Para llevar a cabo las exploraciones de Internet se utilizó la red PlanetLab[1], en conjunto con la plataforma FLAME [17] (*Flexible Lightweight A Measurement Environment*), entorno que fue desarrollado por el LNCC[2] (*Laboratório Nacional de Computação Científica*).

A continuación, en las secciones 4.1 y 4.2, se describe brevemente en qué consiste la red PlanetLab y la plataforma FLAME, respectivamente. Cabe destacar que hoy en día, la Universidad de Buenos Aires forma parte de la red PlanetLab, siendo el Dr. José Ignacio Alvarez-Hamelin el responsable científico de la misma, esto facilita la gestión de recursos al momento de diseñar un plan de exploraciones. También es importante señalar que se mantiene un contacto fluído con los desarrolladores de la plataforma FLAME.

Luego, en la sección 4.3, se explica la metodología implementada para la realización de las observaciones, incluyendo aquellos casos donde los resultados no fueron los esperados. Finalmente, en la sección 4.4, se presentan los resultados obtenidos explicando previamente los parámetros de análisis de grafos más significativos para el análisis y la validación de modelos, como por ejemplo: grado, *strength*, coeficiente de *clustering* y grado medio de los vecinos. Dentro de los resultados presentados, se incluye la visualización del grafo obtenido y un breve análisis sobre los resultados obtenidos.

[1] http://www.planet-lab.org
[2] http://www.lncc.br/

4.1. PlanetLab

PlanetLab es una red mundial de investigación que ha sido diseñada para apoyar el desarrollo de nuevos servicios en redes académicas avanzadas. Este proyecto nace en el año 2003, liderado por la Universidad de Princeton en Estados Unidos, y se lleva a cabo gracias a la suma de un gran número de servidores distribuidos a través de las redes académicas del mundo, los que a su vez, forman un laboratorio computacional a escala planetaria; esta característica origina su nombre.

En el conjunto de servidores que componen la red de PlanetLab se pueden desarrollar, instalar y ejecutar aplicaciones en un entorno de prueba desplegado sobre una red con condiciones reales. Podría decirse que, desde comienzos del año 2003, más de 1000 investigadores de las principales instituciones académicas mundiales y laboratorios de investigación han utilizado PlanetLab para desarrollar nuevas tecnologías para almacenamiento distribuido, mapeo de red, sistemas *peer-to-peer*, tablas de *hash* distribuidas, y otras aplicaciones.

Actualmente, el proyecto PlanetLab cuenta con 1077 nodos[3] (también llamados estaciones terminales o simplemente servidores) en 531 sitios diferentes, distribuídos por todo el mundo como se muestra en la figura 4.1.

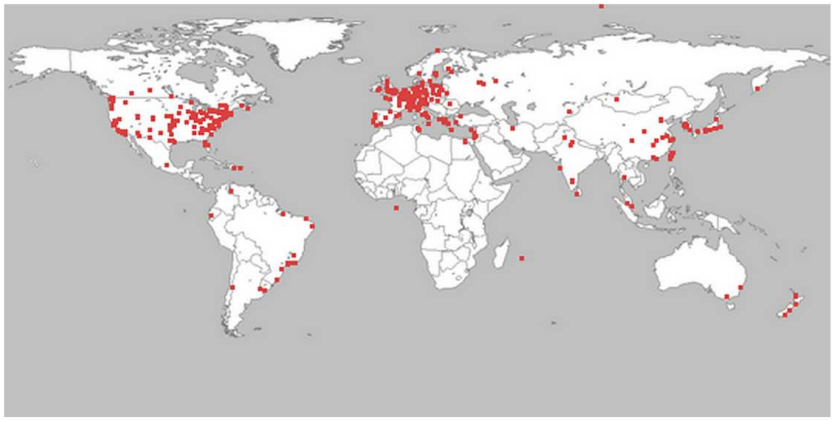


Figura 4.1: Ubicación geográfica de los nodos de PlanetLab.
(Imagen extraída de www.planet-lab.org)

[3]Dado que PlanetLab utiliza esta terminología, a partir de esta sección, los términos nodo y estación terminal se usarán como sinónimos.

Si bien la cantidad de nodos de la red PlanetLab es del orden de los 1000 nodos, uno de los requerimientos, a la hora de presentar una solicitud por parte de una organización para unirse a la red, es que en la misma debe haber por lo menos dos nodos. Aclarado esto, los diferentes nodos elegidos para realizar las exploraciones no pertenecerán nunca a la misma organización; dado que no aportarían información útil, por encontrarse ambos en la misma red. De esta manera, podría decirse que, para la realización de las exploraciones, se utilizará aproximadamente un quinto de la red PlanetLab.

4.1.1. Funcionamiento y acceso a los nodos

El sistema operativo que utilizan todos los nodos de PlanetLab, es `Linux`, específicamente `Fedora Core`[4] de `Red Hat`[5]. Los usuarios de PlanetLab obtienen acceso, debidamente autenticado e independiente de otros usuarios, a los recursos de un nodo en forma de espacios virtuales, denominados *slices*.

Sobre estos espacios, el usuario tiene total autonomía. Se crea un *slice* en cada nodo asociado al mismo. Sobre estos nodos se pueden instalar las aplicaciones necesarias para correr diferentes experimentos. Dado que cada espacio virtual o *slice*, funciona en modo seguro y en un ambiente aislado del resto de los espacios virtuales de otros usuarios; el usuario de un *slice* obtiene ciertos privilegios de súper usuario (*root*) de cada servidor, lo que le permite crear nuevos usuarios, controlar los servicios, instalar nuevos paquetes, etc.

Dado que los recursos son compartidos por la comunidad científica, el derecho de uso de los espacios virtuales o *slices* son concedidos por un rango de tiempo limitado (que puede durar desde una semana hasta varios meses), el cual puede ser renovado periódicamente con previa justificación.

Es importante mencionar que, un usuario por el sólo hecho de registrarse, no puede acceder a la creación de espacios virtuales o *slices*; éstos deberán ser creados por el responsable de la institución académica.

4.1.2. Implementación

Se creó el *slice* denominado *uba-flame* y se asociaron, aproximadamente, 200 nodos al azar que se encontraren disponibles al momento de la creación del *slice*. Se eligió un número de nodos bastante mayor al necesario para poder

[4] http://fedoraproject.org/
[5] http://www.redhat.com/

intercambiar nodos en caso de que alguno no se hallare disponible al momento de realizar las exploraciones. Luego de esperar el tiempo recomendado por PlanetLab para acceder a los nodos asociados una vez creado el *slice*, se procedió a la instalación del agente *flame-agent* en los nodos correspondientes, según la documentación brindada por el LNCC[6] (ver apéndice A.2.1). Para la realización de esta tarea, se utilizó una herramienta recomendada por PlanetLab, la cual permite conexiones en paralelo y ejecución de sentencias de forma simultánea. Este utilitario se conoce con el nombre de `pssh`[7].

4.2. FLAME

Esta plataforma fue desarrollada para la rápida implementación de herramientas activas de medición de parámetros de redes, herramientas basadas en el envío de sondas (paquetes con el único objetivo de realizar mediciones) entre nodos de la red, lo que posibilita la determinación de las propiedades de la red a lo largo de los caminos entre estos nodos.

El entorno FLAME está basado en la distribución de agentes de medición, denominados *flame-agent*, entre los nodos de la red a estudiar. Cada agente envía y/o recibe paquetes sonda en respuesta a comandos ejecutados en una unidad central de control, denominada *flame-console*. Los agentes reportan los datos de las mediciones al administrador central, denominado *flame-manager*, el cual almacena estos datos de forma estandarizada en un repositorio central, típicamente una base de datos SQL (*Structured Query Language*), lo que simplifica la gestión y el análisis de dichos datos. La comunicación entre los tres componentes de FLAME se implementa en base al protocolo XMPP (*Extensible Messaging and Presence Protocol*). La figura 4.2 muestra la arquitectura de FLAME.

Como ya se mencionó previamente, FLAME está orientado a la medición de parámetros de redes. Dicho entorno puede desplegarse sobre los nodos de PlanetLab (ver cómo en el apéndice A.2.1). En particular, esta herramienta brinda la posibilidad de construir paquetes-sonda a medida, posibilitando realizar pruebas no estándar con paquetes ICMP (Internet Control Message Protocol [9]). Vale la pena destacar que todas las herramientas de FLAME, están basadas en el lenguaje de programación `lua`[8].

[6] http://wiki.martin.lncc.br/instalacao-flame-planetlab-en
[7] http://www.theether.org/pssh
[8] http://www.lua.org/

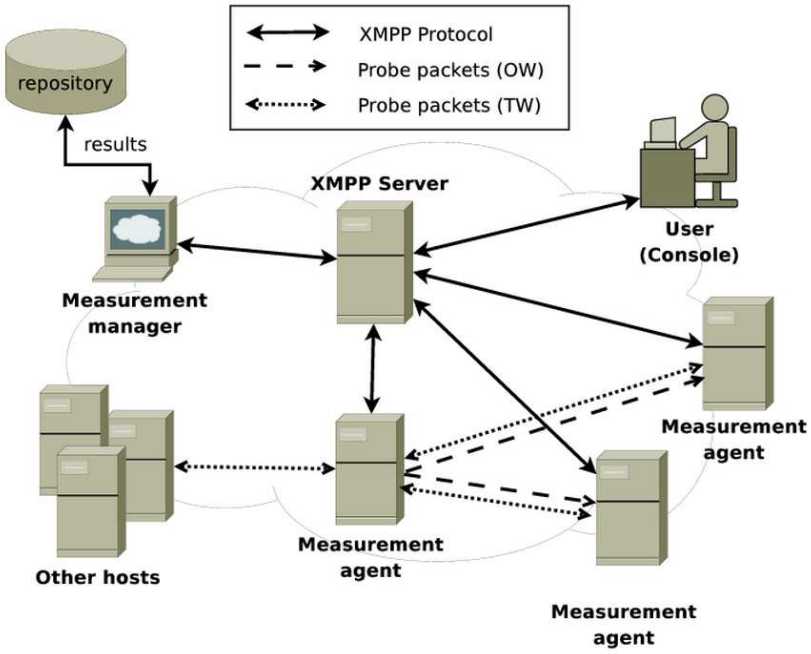


Figura 4.2: Arquitectura de la plataforma FLAME.
(Imagen extraída de http://martin.lncc.br/)

En el apéndice A.1 se muestra cómo está conformada la base de datos donde se almacenan los datos obtenidos, como así también las características técnicas del servidor donde se encuentra alojada.

4.3. Procedimiento de medición

En esta sección se detalla el procedimiento efectuado para poder obtener el conjunto de exploraciones buscado. Dentro de este contexto descrito, se incluye la explicación de un primer intento fallido para la obtención de las exploraciones y posteriormente, la solución encontrada.

4.3.1. Objetivo

El objetivo de las exploraciones es poder realizar una implementación a gran escala durante un intervalo de tiempo prolongado. Se busca poder hacer observaciones entre 100 nodos (todos contra todos) durante 24 horas, donde el paquete sonda utilizado sea ejecutado en los agentes 5 veces por hora, dando como resultado un total de 120 mediciones por nodo.

4.3.2. Paquete sonda propuesto

El paquete sonda utilizado está basado en un *script* escrito en el lenguaje de programación `lua`, el cual ha sido desarrollado por investigadores del LNCC. Este *script*, denominado *flame-trace*, fue modificado en base a nuestras necesidades. En el apéndice A.3.1 se muestra el código fuente del mismo.

Algunas de sus principales características que podemos mencionar, son las siguientes:

- máximo número de saltos permitido: 30
- número de pruebas: 3
- tiempo entre pruebas: 100 ms
- *timeout* para pruebas sin respuesta: 1 s
- protocolo: UDP

Comparando este paquete sonda con el tradicional `traceroute`, las diferencias que se señalan son las siguientes: mientras que en el `traceroute` tradicional se hacen tres pruebas antes de incrementar el TTL esperando en cada una de ellas un *timeout* para los casos en que no haya respuesta, en este paquete sonda se intenta lanzar un tren de 30 pruebas (incrementando el TTL en 1) equiespaciadas en 100ms repetiendo este procedimiento 3 veces, esperando que estos paquetes experimenten todos los mismos fenómemos, como por ejemplo: retardo, tiempo de procesamiento, congestión, etc. Pensando en términos de programación, podría decirse que se invierten los ciclos *for* anidados con respecto al `traceroute` tradicional.

Por otro lado, se utiliza el protocolo UDP en lugar de ICMP, intentando garantizar, por lo menos en el viaje de ida, que todos los paquetes siguen el mismo camino; basándonos en que todos los paquetes enviados pertenecen al mismo flujo de datos. Dado que las respuestas serán mensajes ICMP tipo 11 (*time exceeded*), no puede hacerse ninguna afirmación sobre los caminos que recorren estos paquetes en el viaje de vuelta.

En las últimas líneas del paquete sonda *flame-trace*, se puede apreciar lo siguiente:

```
...
80 local targets = {'h1', 'h2', ..., 'h100'}
81 for k,v in ipairs(targets)
82 do flame_Trace{target=v}
83 end
```

al momento de invocar este *script* en cada *flame-agent*, el ciclo *for* de la línea 81 ejecutará de a un destino por vez. Esto quiere decir que, no pasará al siguiente nodo h_{i+1} hasta no finalizar las 3 mediciones completas con el anterior h_i.

4.3.3. Ejecución

Una vez definido el paquete sonda a ejecutar, instalados los agentes en los nodos de PlanetLab y teniendo disponible el repositorio de datos del administrador central *flame-manager*, se lanzan las mediciones a través de la unidad central de control *flame-console*.

Dado que las mediciones deben ser prolongadas en el tiempo, es necesario automatizar este procedimiento. Debido a que *flame-console* fue instalado en un sistema operativo `Debian GNU/Linux`[9], esto pudo implementarse a través de `cron`. Se configuró esta herramienta para que periódicamente (cada 12 minutos) invoque a *flame-console*, al cual es necesario pasarle como parámetros el nodo o la lista de nodos a los que se conectará, como así también el *script* que ejecutará en los mismos, en este caso *flame-trace* y el archivo de configuración de la herramienta, el cual se puede ver en el apéndice A.4.2.

Se puede decir que *flame-console* es una función que depende de tres parámetros:

$$\textit{flame-console(c, a, s)} \begin{cases} c & \text{configuración de la herramienta} \\ a & \text{nodo o lista de nodos (direcciones IP)} \\ s & \textit{script} \text{ a ejecutar en } a \end{cases} \tag{4.1}$$

Vale la pena aclarar en este punto que, a diferencia de lo que ocurre con el *script flame-trace* en cada nodo, al lanzar en la unidad de control *flame-console*, siendo a una lista de nodos, las conexiones con los agentes se establecen en forma simultánea.

[9] www.debian.org

Intento fallido de medición

Una vez que se encuentra montada toda la infraestructura en funcionamiento, están dadas las condiciones para comenzar a ejecutar las mediciones. El primer intento de ejecución fue fallido. Luego de esperar un tiempo prudencial de ejecución, al verificar en el administrador central *flame-manager*, el número de experimentos era mucho menor al esperado.

La falla se debe a que el paquete sonda no funciona tal cual se expicó en 4.3.2. No es posible lanzar el tren de 30 pruebas y aguardar las respuestas correspondientes, si no que se hace una prueba, se espera su respuesta o agotar el *timeout* y luego se incrementa el TTL hasta obtener la respuesta de la dirección destino; esto quiere decir que no siempre se llegará hasta 30 saltos. Finalmente, este procedimiento se repite tres veces.

Como peor caso, suponiendo que desde un nodo se ejecuta *flame-trace* y ninguno de los 99 destinos responde, el tiempo total de espera que lleva este proceso de ejecución estará dado por la expresión 4.2:

$$\begin{aligned} tiempo_{ejecucion} &= 99\,nodos \cdot 30\,saltos \cdot 3\,pruebas \cdot 1s\,(timeout) = 8910\,s \\ tiempo_{ejecucion} &= 2,475\,horas \end{aligned} \tag{4.2}$$

donde en este cálculo no se tuvo en cuenta el tiempo entre pruebas.

Teniendo en cuenta que el objetivo es ejecutar este ciclo cada 12 minutos, esta opción tal como se presentó no es viable. A continuación se comenta brevemente dónde se halla la limitación y luego cuál fue la solución adoptada.

Limitación

La limitación se encuentra en una biblioteca donde se define una función que utiliza *flame-trace* al momento de lanzar una prueba, la cual llama a otra función, donde la misma es bloqueante, esto quiere decir que no sale del ciclo *while* definido hasta tanto no obtener la respuesta o agotar el *timeout*.

Mediciones con éxito

Para poder reducir el tiempo de ejecución, se crearon en PlanetLab 5 *slices* denomimados de la siguiente manera: *uba-flame-1*, *uba-flame-2*, *uba-flame-3*, *uba-flame-4* y *uba-flame-5*; donde a cada uno de ellos se asociaron

exactamente los mismos nodos que en el caso anterior[10].

En base a los 5 *slices* creados en PlanetLab, se debe diseñar nuevamente el esquema de ejecución adoptado para cumplir con el objetivo propuesto. El esquema de ejecución es el siguiente:

$$\text{Minuto 0} \begin{cases} \textit{flame-console}(c_1,\ a,\ s_1) \\ \textit{flame-console}(c_2,\ a,\ s_2) \\ \textit{flame-console}(c_3,\ a,\ s_3) \\ \textit{flame-console}(c_4,\ a,\ s_4) \\ \textit{flame-console}(c_5,\ a,\ s_5) \end{cases} \tag{4.3}$$

$$\text{Minuto 12} \begin{cases} \textit{flame-console}(c_1,\ a,\ s_5) \\ \textit{flame-console}(c_2,\ a,\ s_1) \\ \textit{flame-console}(c_3,\ a,\ s_2) \\ \textit{flame-console}(c_4,\ a,\ s_3) \\ \textit{flame-console}(c_5,\ a,\ s_4) \end{cases} \tag{4.4}$$

$$\text{Minuto 24} \begin{cases} \textit{flame-console}(c_1,\ a,\ s_4) \\ \textit{flame-console}(c_2,\ a,\ s_5) \\ \textit{flame-console}(c_3,\ a,\ s_1) \\ \textit{flame-console}(c_4,\ a,\ s_2) \\ \textit{flame-console}(c_5,\ a,\ s_3) \end{cases} \tag{4.5}$$

$$\text{Minuto 36} \begin{cases} \textit{flame-console}(c_1,\ a,\ s_3) \\ \textit{flame-console}(c_2,\ a,\ s_4) \\ \textit{flame-console}(c_3,\ a,\ s_5) \\ \textit{flame-console}(c_4,\ a,\ s_1) \\ \textit{flame-console}(c_5,\ a,\ s_2) \end{cases} \tag{4.6}$$

$$\text{Minuto 48} \begin{cases} \textit{flame-console}(c_1,\ a,\ s_2) \\ \textit{flame-console}(c_2,\ a,\ s_3) \\ \textit{flame-console}(c_3,\ a,\ s_4) \\ \textit{flame-console}(c_4,\ a,\ s_5) \\ \textit{flame-console}(c_5,\ a,\ s_1) \end{cases} \tag{4.7}$$

donde la información referida a c_i, a y s_j es la siguiente:

- c_i: contiene la información de cada *slice*, en el apéndice A.4.3 se puede ver la configuración detallada de cada archivo c_i, pero podríamos pensar simplemente que cada valor de subíndice hace referencia al *slice uba-flame-i* correspondiente.

[10]Dado que los *slices* son diferentes, hubo que reinstalar el agente *flame-agent* en cada uno de los nodos para cada uno de los *slices*. Esto se realizó de la misma manera que en el caso de un sólo *slice*, explicado en la sección 4.1.2.

- $a = [h_1, h_2, \ldots, h_{100}]$ es una lista con las direcciones IP de los nodos o estaciones terminales asociados a cada *slice*. Como en todos los *slices* se asociaron los mismos nodos, a no varía en cada ejecución.
- s_j: es el paquete sonda *flame-trace* ejecutado en los nodos de PlanetLab. El cuerpo de este *script* es el mismo en todos los casos, la diferencia se presenta sólo en una línea; la línea 80 presentada en la sección 4.3.2, donde cada s_j contiene lo siguiente:
  - $s_1 = [h_1, h_2, \ldots, h_{20}]$
  - $s_2 = [h_{21}, h_{22}, \ldots, h_{40}]$
  - $s_3 = [h_{41}, h_{42}, \ldots, h_{60}]$
  - $s_4 = [h_{61}, h_{62}, \ldots, h_{80}]$
  - $s_5 = [h_{81}, h_{82}, \ldots, h_{100}]$

De esta manera, en base a la distribución adoptada para efectuar las mediciones, el tiempo de ejecución queda determinado por la expresión 4.8:

$$\begin{aligned} tiempo_{ejecucion} &= 20\, nodos \cdot 30\, saltos \cdot 3\, pruebas \cdot 1s\ (timeout) = 1800\, s \\ tiempo_{ejecucion} &= 0,5\, horas \end{aligned} \tag{4.8}$$

donde se observa que el tiempo de ejecución para el peor caso se ha reducido notablemente. Las observaciones se han realizado bajo estas condiciones.

Acceso a los datos

Una vez completadas las observaciones, los datos se encuentran en su totalidad almacenados en la base de datos del administrador central *flame-manager*. Se puede dar comienzo a la contrucción de las estructuras de datos definidas en 3.1.5, para comenzar el análisis y procesamiento de los datos obtenidos.

4.4. Resultados experimentales

En esta sección se presentan los resultados obtenidos, explicando previamente los parámetros de análisis de grafos comúnmente usados para el análisis y la validación de modelos, tomando como referencia los trabajos [18] y [19]. Para cada parámetro de análisis se muestran los resultados obtenidos según lo propuesto en 3.3; tiempos mínimos (t_{min}), tiempos mínimos modelados (t_{mod}) y tiempos medios (t_{med}). En base al grafo obtenido para cada caso, se calcularon los parámetros de análisis con el programa `Network Workbench`.

4.4.1. Distribución de grado

Ampliando la definición realizada en la sección 3.1.2, se podría decir que las propiedades topológicas de un grafo quedan totalmente definidas en su matriz de adyacencias A, cuyos elementos a_{ij} son 1 si un enlace conecta el vértice i con el vértice j; y 0 en otro caso. Los índices i,j van desde 1 hasta N, donde N es el tamaño del grafo. Para los casos tratados en el presente trabajo, se adopta la siguiente convención: $a_{ii} = 0$. De esta manera el grado k_i de un vértice queda definido por la ecuación 4.9, donde k_i representa en número de vecinos del vértice i.

$$k_i = \sum_j a_{ij} \tag{4.9}$$

Una caracterización natural de las propiedades estadísticas de los grafos es proporcionado por la probabilidad $P(k)$ que cualquier vértice dado tenga grado k. De esta manera, la distribución de grados se define según la expresión 4.10:

$$P(k) = \frac{1}{n} \sum_{\forall i / k_i = k} 1 \tag{4.10}$$

donde n es el número total de vértices y k_i es el grado del vértice i.

Muchos estudios han revelado que los grafos muestran una distribución de probabilidad $P(k)$ con cola pesada, la cual en muchos casos puede aproximarse con precisión por una distribución con comportamiento ley de potencias, esto es $P(k) \sim k^{-\gamma}$, donde $2 \leq \gamma \leq 3$. Esto ha llevado a la introducción de la clase de redes de escala libre [20].

En la figura 4.3 se representa la distribución de grado $P(k)$ en función del grado k para los diferentes casos de análisis propuestos. Como referencia, también se grafica la curva $f(x)$, con el fin de evidenciar que las diferentes distribuciones $P(k)$ siguen una pendiente cercana a -2. De esta manera, se puede deducir que todas las distribuciones responden a una distribución ley de potencias.

4.4.2. Distribución de *strength*

Para grafos pesados puede extenderse el concepto de matriz de adyacencias, en este caso denominada W, cuyo elemento w_{ij} es el peso de la arista que

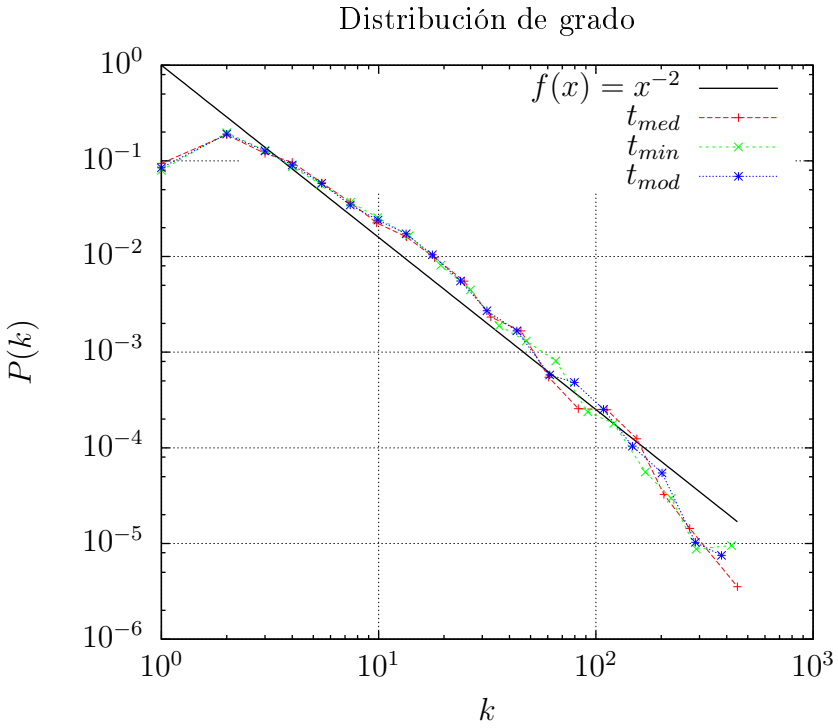


Figura 4.3: Distribución de grado para los casos de análisis.

une los vértices i y j (y $w_{ij} = 0$ si los vértices i y j no están conectados). De esta manera el *strength* s_i queda definido por la ecuación 4.11.

$$s_i = \sum_j w_{ij} \tag{4.11}$$

La distribución de *strength* se define según la expresión 4.12:

$$P(s) = \frac{1}{n} \cdot \frac{1}{\Delta} \sum_{\forall i / s_i \in [s \pm \Delta/2]} 1 \tag{4.12}$$

donde n es el número total de vértices, s_i es el *strength* del vértice i y Δ es un valor variable que depende del *strength* s. Debido a que el *strength* s no posee un valor discreto, se definen intervalos $[s - \Delta/2, s + \Delta/2]$ y se divide por el ancho del mismo para representar la distribución de probabilidad. Para la representación gráfica, se utiliza la técnica de *log bining*, donde la misma consiste básicamente en ir aumentando el tamaño de los intervalos de forma tal que en la escala logarítmica aparezcan equiespaciados.

De forma similar a $P(k)$, $P(s)$ representa la probabilidad de que un vértice tenga *strength* s. En diferentes trabajos ([21], [22], [23]) se ha encontrado que $P(s)$ responde a una distribución con cola pesada y con un comportamiento de ley de potencias.

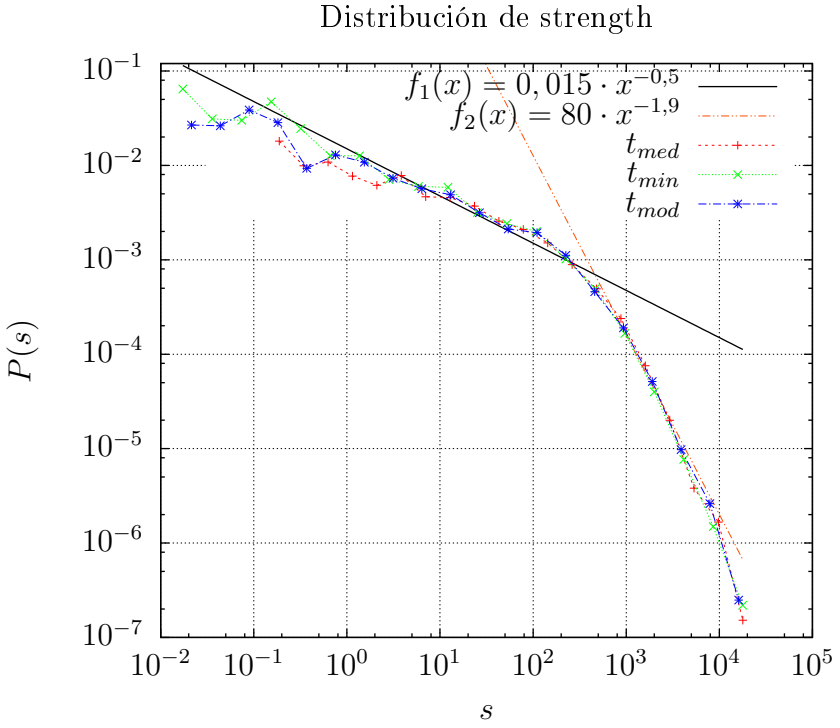


Figura 4.4: Distribución de *strength* para los casos de análisis.

Análogamente a lo mencionado para la distribución de grado, en la figura 4.4 se observa la distribución de *strength* $P(s)$ en función del *strength* s, para los diferentes casos de análisis propuestos. En este caso, podría dividirse el gráfico en dos partes, usando para cada una de ellas una función de referencia: $f_1(x)$ y $f_2(x)$, respectivamente.

Por un lado, sobre la parte izquierda del gráfico, se puede observar que la distribución sigue un comportamiento ley de potencias con una pendiente cercana a $-0,5$ ($f_1(x)$). Por otro lado, en el intervalo de valores de *strength* [10^2,10^3] se observa un punto de quiebre en la distribución, conocido generalmente como *cut off*. En base a esto, podría decirse que la cola de la distribución sigue una distribución ley de potencias con diferente pendiente,

en este caso, cercana a $-1,9$ ($f_2(x)$). Este comportamiento podría producirse por efecto del problema de tamaños finitos.

Relación *strength*-grado

En la figura 4.5, se muestra la relación encontrada entre el *strength* y el grado de los vértices para los diferentes casos de análisis, donde se observa una fuerte correlación entre ambos parámetros. Esta fuerte relación encontrada, servirá para explicar el comportamiento de los parámetros explicados en las secciones 4.4.3 y 4.4.4.

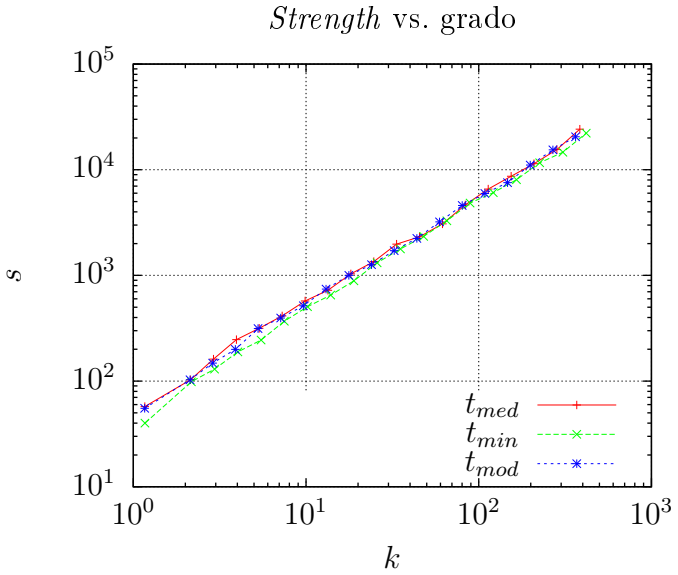


Figura 4.5: *Strength* s en función del grado k.

4.4.3. Distribución del grado medio de los vecinos

Otra importante fuente de información es observar cómo los vértices están conectados. Para ello, se define el grado medio de los vecinos de un vértice de grado k, el cual queda determinado según la ecuación 4.13.

$$k_{nn,i} = \frac{1}{k_i} \sum_{j=1}^{N} a_{ij} k_j \tag{4.13}$$

Para grafos pesados puede hacerce una generalización de lo expresado anteriormente:

$$k^w_{nn,i} = \frac{1}{s_i} \sum_{j=1}^{N} w_{ij} k_j \tag{4.14}$$

Una vez promediado sobre la clase de vértices con conectividad k, la distribución del grado medio de los vecinos k_{nn} puede expresarse según la ecuación 4.15:

$$k_{nn}(k) = \sum_{k'} k' P(k'|k) \tag{4.15}$$

proporcionando una prueba sobre la función de correlación de grado. Si los grados de los vértices vecinos están descorrelacionados, $P(k'|k)$ es sólo una función de k' y por lo tanto $k_{nn}(k)$ es una constante. Si en lugar de utilizar $k_{nn,i}$ se reemplaza por $k^w_{nn,i}$, se obtiene la distribución pesada del grado medio de los vecinos, denominada $k^w_{nn}(k)$.

Al representar $k_{nn}(k)$ se pueden definir dos comportamientos (si se llegaren a observar pendientes marcadas en las curvas): comportamiento *assortative* si $k_{nn}(k)$ se incrementa con k, el cual indica que los vértices de mayor grado están preferentemente conectados con otros vértices de mayor grado; este último comportamiento es observado en algunas redes sociales; y comportamiento *disassortative* cuando $k_{nn}(k)$ decrece con k. Ambos conceptos se encuentran desarrollados en [24]. Si los vértices de grado elevado poseen un k_{nn} pequeño, entonces ellos actuarán de concentradores, implicando que los vértices de grado bajo tendrán como vecinos otros de grado elevado; este comportamiento se observa en las redes de AS (*Autonomous System*).

En las figuras 4.6, 4.7 y 4.8 se presentan las distribuciones del grado medio de los vecinos $k_{nn}(k)$ y $k^w_{nn}(k)$ (la cual indica que el grafo correspondiente es pesado) en función del grado k para los diferentes casos de análisis. Se observa que ambas curvas prácticamente no presentan pendiente, por lo que podríamos decir, basándonos en la curva $k_{nn}(k)$, que si uno elige un nodo al azar, el grado de sus vecinos no depende de su propio grado. Este mismo comportamiento se observa en la distribución $k^w_{nn}(k)$; esto se debe a la fuerte correlación encontrada entre grado y *strength*, la cual se muestra en la figura 4.5.

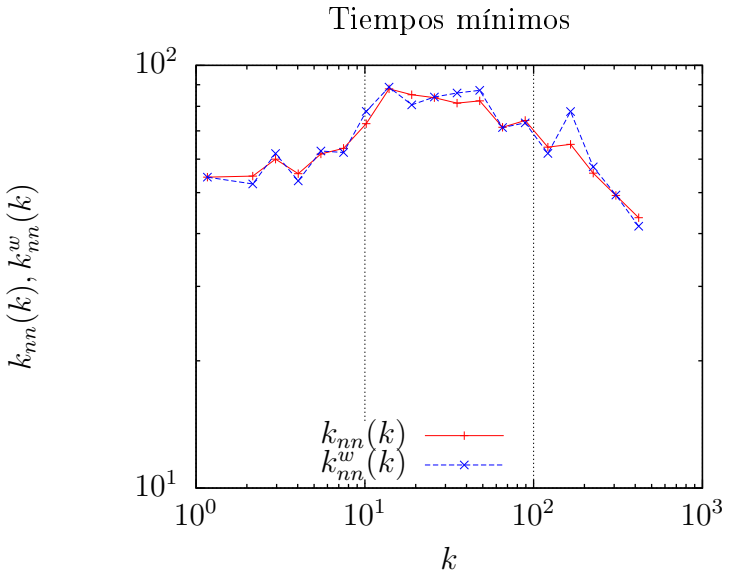


Figura 4.6: Distribución $k_{nn}(k)$ y $k^w_{nn}(k)$ para t_{min}.

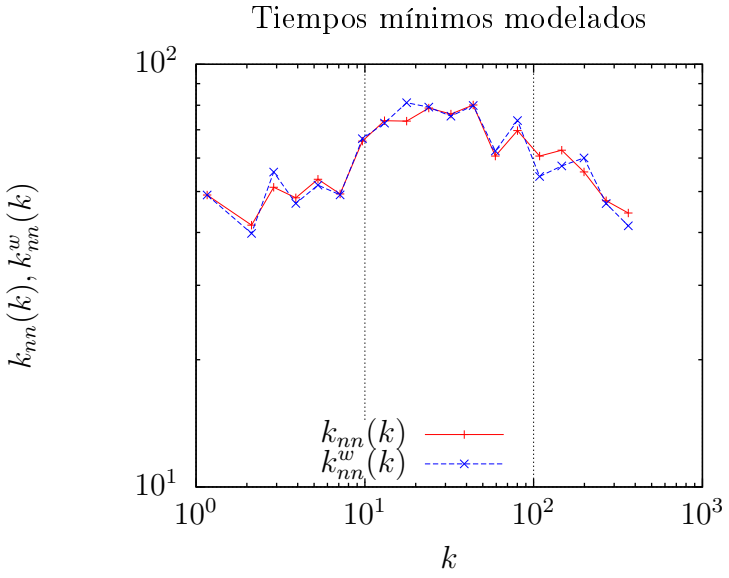


Figura 4.7: Distribución $k_{nn}(k)$ y $k^w_{nn}(k)$ para t_{mod}.

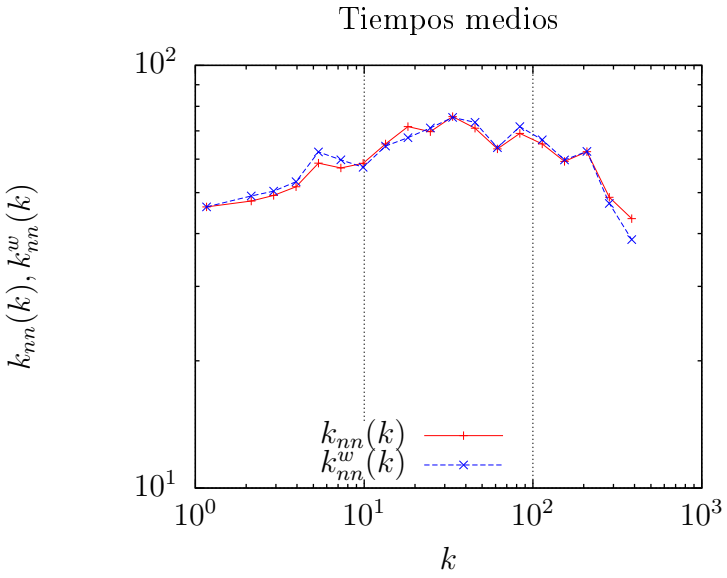


Figura 4.8: Distribución $k_{nn}(k)$ y $k^w_{nn}(k)$ para t_{med}.

4.4.4. Distribución del coeficiente de *clustering*

El coeficiente de *clustering* de un vértice i se define según la ecuación 4.16

$$c_i = \frac{1}{k_i(k_i-1)} \sum_{j,h} a_{ij}a_{ih}a_{jh} \tag{4.16}$$

y mide la interconexión de los vecinos del vértice i.

El coeficiente de *clustering* para grafos pesados se define según la ecuación 4.17.

$$c^w_i = \frac{1}{s_i(k_i-1)} \sum_{j,h} \frac{(w_{ij}+w_{ih})}{2} a_{ij}a_{ih}a_{jh} \tag{4.17}$$

donde esta cantidad combina la interconexión de los vecinos del vértice i con la intensidad (peso) de los enlaces que los interconectan.

La distribución del coeficiente de *clustering* indica qué probabilidad tienen los vecinos de un vértice de estar interconectados entre sí, la cual queda definida por la ecuación 4.18.

$$C(k) = \frac{1}{NP(k)} \sum_{i/k_i=k} c_i \tag{4.18}$$

De forma análoga a lo expresado para la distribución pesada para el grado medio de los vecinos, si en la ecuación 4.18 reemplazamos c_i por c_i^w obtenemos la distribución pesada para el coeficiente de *clustering*, denominada $C^w(k)$.

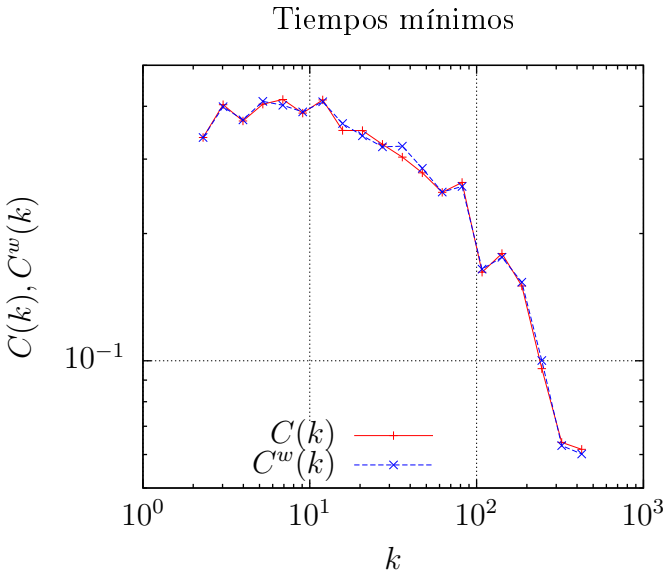


Figura 4.9: Distribución $C(k)$ y $C^w(k)$ para t_{min}.

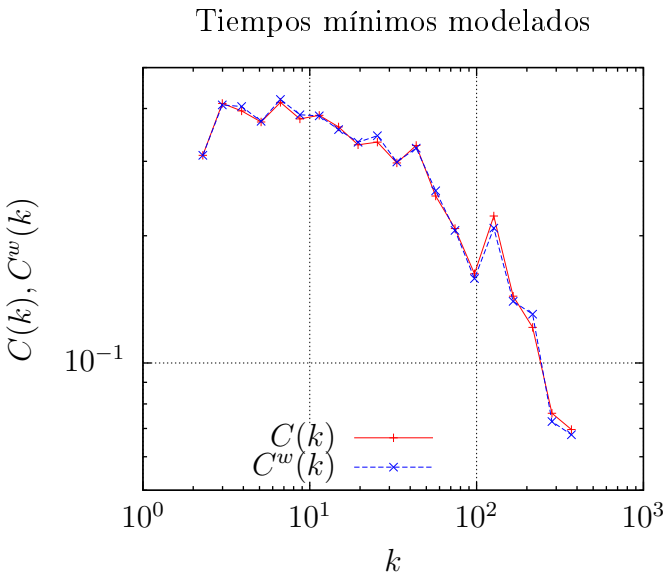


Figura 4.10: Distribución $C(k)$ y $C^w(k)$ para t_{mod}.

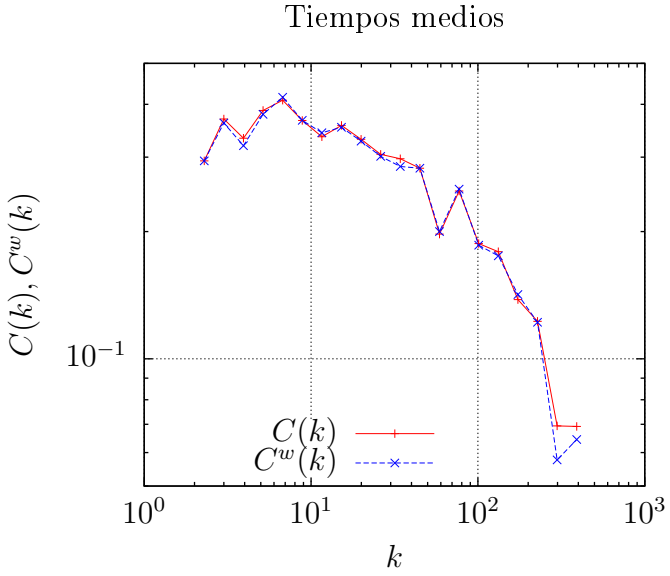


Figura 4.11: Distribución $C(k)$ y $C^w(k)$ para t_{med}.

En las figuras 4.9, 4.10 y 4.11 se presentan las distribuciones del coeficiente de *clustering* $C(k)$ y $C^w(k)$ (la cual indica que el grafo correspondiente es pesado) en función del grado k para los diferentes casos de análisis. Debido a la similitud encontrada entre las curvas $k_{nn}(k)$ y $k^w_{nn}(k)$, era esperable encontrar esta misma situación en este parámetro, dado que que existe una relación entre $C(k)$ y $k_{nn}(k)$, ya que para tener un elevado coeficiente de *clustering* es necesario que el grado de los vecinos sea elevado.

La similutud entre las curvas $C(k)$ y $C^w(k)$ se explica en base a la ya mencionada relación encontrada entre el *strength* y el grado (figura 4.5), donde se observa que la misma es prácticamente lineal. En estas curvas se evidencia nuevamente, dentro del intervalo [10^2,10^3], el punto de quiebre (*cut off*), causado por efecto del problema de tamaños finitos.

4.4.5. Visualización del grafo obtenido

Para la visualización de los grafos obtenidos, se utilizó la herramienta LaNet-vi[11] (*Large Networks visualization tool*), herramienta basada en la descomposición en k-núcleos [25]. A continuación, se introducen algunos conceptos necesarios para poder describir los resultados obtenidos.

[11] http://lanet-vi.soic.indiana.edu/

Definición (k**-núcleo**)**.** Un subgrafo $H = (C, E|C)$ inducido por el conjunto $C \subseteq V$ es un k-núcleo o un núcleo de orden k $sii \, \forall \, v \in C : grado_H(v)$ y H es el máximo subgrafo con esta propiedad.

Una forma de obtener la descomposición en k-núcleos es ir eliminando recursivamente todos los vértices de grado menor que k, hasta que todos los vértices restantes tengan grado mayor o igual a k.

Definición (**capa**)**.** Un vértice i tiene número de capa c (también conocido por su denominación en inglés: *shell index*), si dicho vértice pertenece al c-núcleo pero no al $(c+1)$-núcleo.

Definición (**clique**)**.** Un *clique* es un subgrafo donde cada vértice está conectado a cada uno de los vértices del grafo. Esto equivale a decir que el subgrafo inducido por V es un grafo completo.

En la figura 4.12 se visualiza el grafo obtenido para el caso t_{min}[12]. La idea de la visualización es ubicar en el centro la capa con el k_{max}-núcleo en forma de círculo, donde su diámetro es proporcional al número de elementos y luego las otras capas en circunferencias concéntricas que se alejan del k_{max}.

Sobre la derecha se muestra una escala de colores con el número de capa c_i, mientras que sobre la izquierda figura la escala logarítmica del grado de los vértices representado por el tamaño de los mismos. Es importante señalar que no se grafican todas las aristas, sino un porcentaje de las mismas elegidas con probabilidad uniforme. Los detalles completos de la visualización se pueden encontrar extensivamente descritos en [26].

En la figura 4.13, se muestra el mismo grafo visualizado en la figura 4.12 pero omitiendo los *cliques* del núcleo central.

Si bien se observan algunas conexiones hacia la capa central, también se observan características propias de los mapas de ruteadores. Hay una tendencia presente donde todas las capas están densamente pobladas y las conexiones (aristas) se producen principalmente entre capas.

[12]Teniendo en cuenta que, el proceso de construcción del grafo depende de los tiempos t_{min}, t_{mod} y t_{med} considerados; se obtienen tres visualizaciones diferentes. En esta sección, solamente se presenta el caso t_{min}.

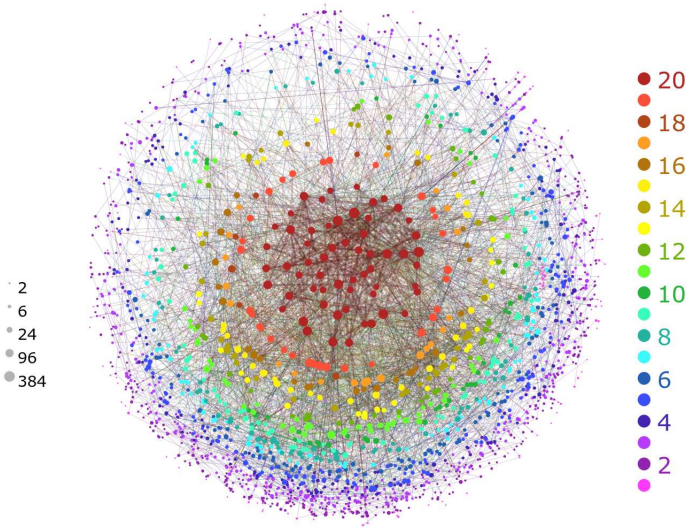


Figura 4.12: Visualización del grafo obtenido para t_{min} con LaNet-vi.

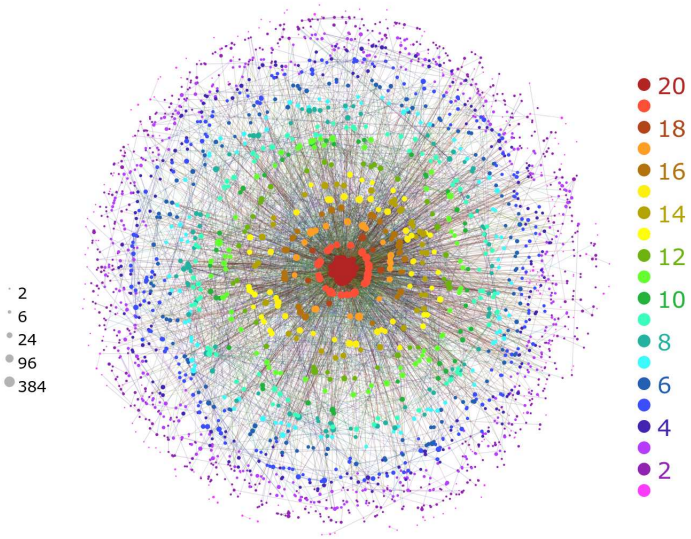


Figura 4.13: Visualización del grafo obtenido omitiendo *cliques* del núcleo central.

En ambas figuras, también se observan nodos de un tamaño considerable sobre la periferia, esta es otra característica de los mapas de ruteadores. El tamaño de estos nodos es un poco más reducido comparando con los encontrados en otros trabajos (por ejemplo: [18]), esto es debido a la cantidad limitada de fuentes desde donde se realizaron las exploraciones.

4.4.6. Discusión sobre los resultados obtenidos

En este sección se presenta un breve resumen incluyendo un análisis de los resultados obtenidos. A priori, era esperable encontrar un comportamiento diferente entre las distribuciones de los parámetros pesados y no pesados, pero debido a la relación mostrada en la figura 4.5 entre grado y *strength*, se explica el porqué de esta similitud en las distribuciones.

Por otra parte, debido al comportamiento cuasi estático observado en las curvas $k_{nn}(k)$ y $k^{w}_{nn}(k)$, podría decirse tanto para el grado como para el *strength* que, al elegir un nodo cualquiera al azar, ni el grado de sus vecinos ni el *strength* de sus vecinos, depende del grado del nodo seleccionado.

Si bien el modelo considerado para los tiempos medios es un modelo simple, resulta llamativo que, en las distribuciones de parámetros pesados, prácticamente no haya diferencias respecto de los casos para tiempos mínimos y tiempos mínimos modelados. En base a esta característica encontrada, podría pensarse que cada enlace es independiente de otro, dando a entender un correcto funcionamiento de la red. Por ejemplo, si un enlace se encuentra congestionado, no congestiona a los de su alrededor.

Finalmente, el grafo obtenido, visualizado a través de LaNet-vi (figuras 4.12 y 4.13), presenta características propias de los mapas de ruteadores de Internet.

Capítulo 5

Conclusiones

En la presente tesis se realizaron mediciones a gran escala, utilizando la red PlanetLab y tomando mediciones entre nodos en ambos sentidos; es decir que un mismo nodo fue considerado tanto origen como destino. Las mediciones realizadas de esta manera, facilitaron la resolución de *aliases*. Posteriormente, con los datos almacenados y el procesamiento adecuado, se construyeron mapas anotados de Internet para dos parámetros de interés: tiempos mínimos (absolutos y modelados) y tiempos medios (ver secciones 3.3 y 4.4). Es importante destacar que el procesamiento propuesto permite trabajar con redes de gran tamaño dada su baja complejidad.

Estos parámetros nos proporcionan una idea de diferentes características de la red. Por un lado, tanto el tiempo mínimo como el tiempo mínimo modelado, son indicadores de la longitud física de los enlaces. Por otra parte, los tiempos medios, son indicadores de la congestión de los enlaces.

Aunque a priori se esperaba observar una marcada diferencia entre el grado de los vecinos y el coeficiente de *clustering* tanto para los casos pesados como los no pesados, esto no sucedió. La justificación de esto se encuentra en la marcada correlación encontrada entre las distribuciones de grado y *strength* en función del grado.

Por otra parte, resulta llamativo que en las distribuciones del grado de los vecinos y coeficiente de *clustering* pesados prácticamente no haya diferencias entre tiempos medios y los casos para tiempos mínimos. Si bien el modelo considerado para el análisis de los tiempos medios es un modelo simple, podría pensarse que cada enlace es independiente de otro, dando a entender un correcto funcionamiento de la red. Por ejemplo, si un enlace se encuentra

saturado, el mismo no influye en la congestión de otros enlaces vecinos a los ruteadores de sus extremos.

Debido a limitaciones encontradas en la plataforma FLAME, no se explotó toda la potencialidad que brinda PlanetLab, siendo utilizados, aproximadamente, un quinto de los nodos de esta plataforma.

En base al fluido contacto establecido con los desarrolladores de la plataforma FLAME, todos los inconvenientes que fuimos encontrando con esta herramienta han sido reportados. De esta manera, según lo conversado con los desarrolladores, se espera una nueva versión que permita un mejor funcionamiento al trabajar con un número mayor de nodos, como así también resolver el problema de la función bloqueante encontrada; el cual fue detallado en la sección 4.3.3. Llegado este punto, vale la pena decir que contamos con los conocimientos necesarios para realizar mediciones con una población mayor de nodos.

Trabajo futuro

A futuro se abren varias posibilidades para continuar con el estudio de esta temática. Por una lado, se podría ampliar aún más el número de exploraciones implementando mediciones hacia nodos no controlables, esto quiere decir que si bien se perdería la posibilidad de obtener mediciones entre orígenes y destinos, se podría obtener un mapa más representativo de la realidad en base a mediciones en un sólo sentido (conocidas como *one way*); por ejemplo, desde los 500 nodos de PlanetLab hacia cientos o miles de destinos en Internet. Es importante remarcar que los primeros saltos de un origen suelen ser idénticos para cualquier destino; por lo tanto detectarlos evitaría sobrecargar estos ruteadores con paquetes sonda cuando aumentamos la cantidad de destinos.

Por otro parte, se podría extender el estudio de los mapas anotados en función de otros parámetros de interés, como pueden ser: ancho de banda, disponibilidad, tipo de tráfico, tasa de pérdida, etc. También podría pensarse en la posibilidad de analizar con nuestra metodología, los datos brindados a la comunidad científica por CAIDA[1] y DIMES[2]; como así también estudiar la posibilidad de desarrollar modelos a partir de los datos obtenidos.

[1] http://www.caida.org
[2] http://www.netdimes.org

Apéndice A

Apéndice

A.1. Repositorio de datos

Para el almacenamiento de los datos provenientes de las exploraciones, se cuenta con un servidor ubicado físicamente dentro de la red de la Universidad de Buenos Aires, el cual posee las siguientes características técnicas:

- Procesador: AMD Athlon(tm) 64 X2 Dual Core Processor 5200+
- Memoria RAM: 4 GB
- Disco rígido: 160 GB
- Sistema operativo: Debian 5.0.8

En este servidor se ejecuta el componente correspondiente a *flame-manager*, es decir, la base de datos que recolecta la información proveniente de los agentes. En el apéndice A.2.2 se puede ver el detalle de la instalación del mismo, pero es importante señalar que, para el debido funcionamiento de este componente, se debe tener funcionando en el servidor una base de datos del estilo SQL (en este caso se utiliza `PostgreSQL 8.3`[1]) y un servidor XMPP (en un primer momento se utilizó `Prosody`[2] y finalmente se migró a `ejabberd`[3]).

A.1.1. Estructura de la base de datos

El motor de base de datos utilizado es `PostgreSQL 8.3`. La base de datos está compuesta por las siguientes tablas:

[1] http://www.postgresql.org/
[2] http://prosody.im/
[3] http://www.ejabberd.im/

```
- commands
- experiment_type
- experiments
- protocols
- results
- scripts
```

a los fines del presente trabajo, las tablas con información útil para nuestro propósito serán *experiments* y *results*; a continuación se detallan cuáles son los campos que conforman estas tablas.

Tabla *experiments*

La tabla *experiments* está formada por los siguientes campos:

Tabla *experiments*		
Columna	Tipo	Modificadores
experiment_id	*integer*	*not null default nextval('experiment_id'::regclass)*
date	*text*	
hour	*text*	
ip_from	*text*	
script_id	*integer*	

Tabla A.1: Campos de la tabla *experiments*

La tabla A.1 presenta el siguiente índice:

```
«pk_experiment» PRIMARY KEY, btree (experiment_id)
```

La tabla A.1 se utiliza principalmente para el control de los experimentos utilizando para este propósito las entradas `date`, `hour` y `experiment_id`. El campo `experiment_id` se va incrementando a medida que se ejecutan los paquetes sonda en los agentes.

Tabla *results*

La tabla *results* está formada por los siguientes campos:

Tabla *results*		
Columna	Tipo	Modificadores
result_id	*integer*	*not null default nextval('result_id'::regclass)*
protocol_id	*integer*	
type_id	*integer*	
experiment_id	*integer*	
loss	*boolean*	
packet_size	*integer*	
ip_resp	*text*	
start	*numeric(25,6)*	*not null*
finish	*numeric(25,6)*	*not null*
ttl	*integer*	
packet_size_rcv	*integer*	
icmp_error_msg	*text*	
ip_dest	*text*	
ip_from	*text*	

Tabla A.2: Campos de la tabla *results*

La tabla A.2 presenta el siguiente índice:

```
«pk_result» PRIMARY KEY, btree (result_id)
```

con las siguientes restricciones de llave foránea:

```
«fk_exp» FOREIGN KEY (experiment_id) REFERENCES experiments(experiment_id)
«fk_proto» FOREIGN KEY (protocol_id) REFERENCES protocols(protocol_id)
«fk_type» FOREIGN KEY (type_id) REFERENCES experiment_type(type_id)
```

En la tabla A.2 se almacenan los datos a partir de los cuales se construyen las secuencias definidas en 3.1. Los principales campos de esta tabla son:

- `ip_resp`: donde se almacenan las direcciones IP de los nodos intermedios que van respondiendo luego de ejecutar cada *flame-trace*.
- `ip_from`: dirección IP del nodo origen.
- `ip_dest`: dirección IP del nodo destino.
- `start`: marca de tiempo de envío del requerimiento.
- `finish`: marca de tiempo de llegada de la respuesta.

A.2. Puesta a punto de la plataforma FLAME

Para usar esta plataforma, el servicio XMPP debe estar funcionando correctamente sobre el servidor. Este servicio es una condición necesaria para el funcionamiento de los tres componentes de FLAME, donde cada una de las partes que componen la plataforma usará la misma configuración XMPP.

A.2.1. Instalación de *flame-agent* sobre PlanetLab

Los agentes, al estar funcionando sobre nodos de PlanetLab, utilizan como sistema operativo `Fedora 8`. Para la compilación de *flame-agent* es necesario instalar los siguientes paquetes: `g++`, `lua5.1`, `lua5.1`, `make` y `gloox`.

```
# yum install gcc-c++ lua make lua-devel
```

También debemos compilar e instalar la biblioteca `gloox`, después de descargar[4] el archivo, se debe ejecutar (dentro del mismo directorio):

```
# ./configure
# make
# make install
```

Luego, debemos crear un enlace simbólico a la biblioteca, para esto se ejecuta lo siguiente:

```
# ln -s /usr/local/lib/libgloox.so.7 /lib/libgloox.so.7
```

Finalmente, luego de descargar y descomprimir este componente, se compila ejecutando:

```
# g++ *.cpp  *.h -llua -lm -ldl -lgloox -lpthread \\
             -I/usr/include/lua -LLIBDIR -o flame
```

Al terminar la compilación, el archivo ejecutable será `./agent`. Para poder ejecutar el agente, se debe editar el archivo `flame.conf`, el cual se muestra en A.4.2, donde básicamente debemos especificar el servidor XMPP y el usuario común a todos los componentes. Este proceso se debe ejecutar con privilegios de administrador.

[4] http://wiki.martin.lncc.br/instalacao-flame-planetlab-en/file/gloox-0-9-9-9-tar.bz2

Finalmente, para levantar el proceso se debe ejecutar:

```
# ./agent -c conf/flame.conf
```

se debe tener en cuenta que para que este proceso se ejecute correctamente, el componente *flame-manager* debe estar ejecutándose previamente, caso contrario informará un error.

A.2.2. Instalación de *flame-manager*

Para la compilación de *flame-manager* es necesario instalar los siguientes paquetes: `libpqxx-dev` y `libgloox-dev`.

```
# aptitude install libgloox-dev libpqxx-dev
```

Luego de descargar este componente, dentro del mismo directorio se debe ejecutar:

```
aclocal
./autogen.sh
autoconf
automake
./configure
make
```

Al finalizar la compilación el archivo ejecutable será `./manager`. Nuevamente en `flame.conf`, el cual se muestra en A.4.1, se debe especificar el servidor XMPP y el usuario común a todos los componentes, teniendo en cuenta que en la configuración del componente *flame-manager*, también se debe incluir la información necesaria para conectarse a la base de datos.

Al descargar los componentes de FLAME se cuenta con la estructura de la base de datos que debemos introducir en el motor SQL. Para realizar esto, se ejecuta:

```
$ sudo -u postgres psql -f db-schema.sql
```

Finalmente, para levantar el proceso se debe ejecutar:

```
# ./manager -c conf/flame.conf
```

A.2.3. Instalación de *flame-console*

Para la compilación de *flame-console* es necesario instalar los siguientes paquetes: `lua5.1` y `libgloox-dev`.

```
# aptitude install lua5.1 libgloox-dev
```

Luego de descargar este componente, dentro del mismo directorio se debe ejecutar:

```
aclocal
./autogen.sh
autoconf
automake
./configure
make
```

Al finalizar la compilación el archivo ejecutable será `./console`. En este caso el archivo de configuración podrá ser el mismo que en el caso de *flame-agent* A.4.2. A modo de ejemplo, se muestra como debe ejecutarse este componente:

```
# ./console -a "return {'10.0.0.1','10.0.0.2'}"  \\
  -c conf/flame.conf -s examples/test_sendICMPTW2.lua
```

en este caso, se ejecutará el *script* `test_sendICMPTW2.lua` en los agentes cuyas direcciones IP son: 10.0.0.1 y 10.0.0.2.

A.3. *Scripts*

A continuación se muestra el código fuente de algunos de los *scripts* utilizados.

A.3.1. Paquete sonda: código fuente

```
1  -- Main function
2  function flame_Trace(param_table)
3   local target   = param_table.target or "127.0.0.1"
4   local maxhops  = param_table.maxhops or 30
5   local interval = param_table.interval or 100000
6   local npackets = param_table.npackets or 3
7   local protocol = param_table.protocol or "udp"
```

```
8   local size     = param_table.size or 40
9
10  -- Output header
11 print("traceroute to " ..  lamu.ip2name(param_table.target) .. " \\
12       (" ..  param_table.target .. "), " .. maxhops ..  \\
13        " hops max, " .. size .. " byte packets")
14
15  for packet = 1,npackets do
16   for hop=1, maxhops do
17    local last_addr
18    local output
19    local response
20    if protocol == "udp" then
21     response = lamp.sendUDPTW{ip = param_table.target, \\
22                     size = size, ttl = hop, timeout=1}
23    elseif protocol == "icmp" then
24     response = lamp.sendICMPTW{ip = param_table.target, \\
25                      size = size, ttl = hop, timeout=1}
26    elseif protocol == "tcp" then
27     response = lamp.sendTCPTW{ip = param_table.target, \\
28                     size = size, ttl = hop, timeout=1}
29    else
30     print("Invalid protocol")
31    end
32
33    if response then
34     if response.loss == lamu.err.ICMP_HOST_UNREACH then
35      print("DESTINATION UNREACHABLE " .. param_table.target)
36      return
37     elseif response.loss ~= lamu.err.REQUEST_TIMED_OUT then
38      local time = string.format("%.3f",(response.finish - \\
39                           response.start) * 1000) .. " ms"
40      if output then
41       if response.dstIP == last_addr then
42        output = output .. "  " .. time
43       else
44        output = output .. "  " .. lamu.ip2name(response.dstIP) \\
45                     .. " (" .. response.dstIP .. ")  " .. time
46       end
47      else
48       output = hop .. "  " .. lamu.ip2name(response.dstIP) .. " \\
```

```
49                          (" .. response.dstIP .. ") " .. time
50      end
51
52      last_addr = response.dstIP
53
54     else
55      if output then
56       output = output .. " * "
57      else
58       output = hop .. " * "
59      end
60     end
61    else
62     print("NETWORK UNREACHABLE(" .. param_table.target .. ")")
63     return
64    end
65    -- If current node is target... bullseye!
66    if (response.dstIP == param_table.target) then break end
67    print(output)
68   end
69
70   -- Wait time between probes
71   if (type(interval) == "number") then
72    lamu.sleep(interval);
73   elseif (type(interval) == "table") then
74    lamu.sleep(interval.func(interval.params))
75   end
76  end
77 end
78
79 -- Main function call
80 local targets = {‘h1’, ‘h2’, ..., ‘h100’}
81 for k,v in ipairs(targets)
82  do flame_Trace{target=v}
83 end
```

A.4. Archivos de configuración

A continuación, se muestran diferentes archivos de configuración utilizados a lo largo del desarrollo del presente trabajo.

A.4.1. *flame-manager*: 1 *slice*

En esta sección, se muestra el archivo de configuración de *flame-manager* para el caso de ejecución correspondiente a un *slice*. Se observa que la única diferencia con respecto a *flame-agent* y *flame-console*, se halla en la configuración de los parámetros de conexión de la base de datos.

```
-- ******* FLAME.CONF ******

-- * Server XMPP and a valid user / password
server = flame.uba.ar
user = uba-flame
pass = 123456

-- * DB connection options
db_host =       localhost
db_name =       flamedb
db_login =      postgres
db_psswd =      pepe1234

-- * MUC XMPP id (default = conference)
group = conference
```

A.4.2. *flame-console*: 1 *slice*

En esta sección, se muestra el archivo de configuración de *flame-console* para el caso de ejecución correspondiente a un *slice*. Este archivo corresponde al parámetro c_i definido en 4.1.

```
 1 -- ****** FLAME.CONF ******
 2
 3 -- * Server XMPP and a valid user / pass
 4 server =    flame.uba.ar
 5 user =      uba-flame
 6 pass =      123456
 7
 8
 9 -- * MUC XMPP id
10 group =  conference
```

A.4.3. *flame-console*: 5 *slices*

En esta sección, se muestran los diferentes archivos de configuración de *flame-console* para el caso de ejecución correspondiente a 5 *slices* de PlanetLab. Nuevamente, estos archivos corresponden al parámetro c definido en 4.1.

Slice uba-flame-1: c_1

```
-- ****** FLAME1.CONF ******

-- * Server XMPP and a valid user / pass
server =    flame.uba.ar
user =      uba-flame-1
pass =      123456


-- * MUC XMPP id
group =  conference
```

Slice uba-flame-2: c_2

```
-- ****** FLAME2.CONF ******

-- * Server XMPP and a valid user / pass
server =    flame.uba.ar
user =      uba-flame-2
pass =      123456


-- * MUC XMPP id
group =  conference
```

Slice uba-flame-3: c_3

```
-- ****** FLAME3.CONF ******

-- * Server XMPP and a valid user / pass
server =    flame.uba.ar
user =      uba-flame-3
pass =      123456
```

```
 8
 9 -- * MUC XMPP id
10 group =  conference
```

***Slice uba-flame-4*:** c_4

```
 1 -- ****** FLAME4.CONF ******
 2
 3 -- * Server XMPP and a valid user / pass
 4 server =    flame.uba.ar
 5 user =      uba-flame-4
 6 pass =      123456
 7
 8
 9 -- * MUC XMPP id
10 group =  conference
```

***Slice uba-flame-5*:** c_5

```
 1 -- ****** FLAME5.CONF ******
 2
 3 -- * Server XMPP and a valid user / pass
 4 server =    flame.uba.ar
 5 user =      uba-flame-5
 6 pass =      123456
 7
 8
 9 -- * MUC XMPP id
10 group =  conference
```

Bibliografía

[1] Y. Shavitt and E. Shir, "Dimes: let the internet measure itself," *Computer Communication Review*, vol. 35, pp. 71–74, 2005.

[2] Y. Chen, D. Bindel, and R. H. Katz, "Tomography-based overlay network monitoring," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, IMC '03, (New York, NY, USA), pp. 216–231, ACM, 2003.

[3] Y. Chen, D. Bindel, H. H. Song, and R. H. Katz, "An algebraic approach to practical and scalable overlay network monitoring," in *SIGCOMM*, pp. 55–66, 2004.

[4] G. H. Golub and C. F. Van Loan, *Matrix computations (3rd ed.)*. Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[5] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, third ed., 1999.

[6] D. B. Chua, E. D. Kolaczyk, and M. Crovella, "Efficient monitoring of end-to-end network properties," in *In Proc. of IEEE INFOCOM*, pp. 1701–1711, 2005.

[7] O. Goldreich, *P, NP, and NP-Completeness: The Basics of Computational Complexity*. P, NP, and NP-completeness: The Basics of Computational Complexity, Cambridge University Press, 2010.

[8] Y. Shavitt, X. Sun, A. Wool, and B. Yener, "Computing the unmeasured: An algebraic approach to internet mapping," in *in IEEE INFOCOM*, 2001.

[9] J. Postel, "RFC 792: Internet Control Message Protocol," Sept. 1981.

[10] M. Allalouf, E. Kaplan, and Y. Shavitt, "On the feasibility of a large scale distributed testbed for measuring quality of path characteristics in the internet," 2009.

[11] D. Morato, E. Magana, M. Izal, J. Aracil, F. Naranjo, F. Astiz, U. Alonso, I. Csabai, P. Haga, G. Simon, J. Steger, and G. Vattay, "The european traffic observatory measurement infraestructure (etomic): A testbed for universal active and passive measurements," *Testbeds and Research Infrastructures for the Development of Networks & Communities, International Conference on*, vol. 0, pp. 283–289, 2005.

[12] S. Bilir, K. Saraç, and T. Korkmaz, "Intersection characteristics of end-to-end internet paths and trees," in *International Conference on Network Protocols*, pp. 378–390, 2005.

[13] R. Govindan and H. Tangmunarunkit, "Heuristics for internet map discovery," *Proceedings IEEE INFOCOM 2000 Conference on Computer Communications Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies Cat No00CH37064*, vol. 3, pp. 1371–1380, 2000.

[14] M. H. Gunes and K. Sarac, "Resolving ip aliases in building traceroute-based internet maps," tech. rep., 2006.

[15] K. Hubbard, M. Kosters, D. Conrad, D. Karrenberg, and J. Postel, "Internet Registry IP Allocation Guidelines." RFC 2050 (Best Current Practice), November 1996.

[16] C. C. Paige and M. A. Saunders, "Lsqr: An algorithm for sparse linear equations and sparse least squares," vol. 0, pp. 43–71, 1982.

[17] A. Ziviani, A. T. A. Gomes, M. L. Kirszenblatt, and T. B. Cardozo, "Flame: Flexible lightweight active measurement environment," in *TRIDENTCOM*, pp. 526–541, 2010.

[18] J. I. Alvarez-Hamelin, "Taxonomía de los Modelos de Topología de Internet," in *Mecánica Computacional, Interdisciplinary Mathematical Methods*, vol. XXV, pp. 2597–2612, CIMEC, 2006.

[19] A. Barrat, M. Barthélemy, and A. Vespignani, "Modeling the evolution of weighted networks," *Phys. Rev. E*, vol. 70, p. 066149, 2004.

[20] A. L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, 1999.

[21] R. Guimera, S. Mossa, A. Turtschi, and L. A. Núñez-Amaral, "Structure and efficiency of the world-wide airport network," Tech. Rep. cond-mat/0312535, Dec 2003.

[22] A. Barrat, M. Barthelemy, R. Pastor-Satorras, and A. Vespignani, "The architecture of complex weighted networks," *PROC.NATL.ACAD.SCI.USA*, vol. 101, p. 3747, 2004.

[23] D. Garlaschelli, S. Battiston, M. Castri, V. D. P. Servedio, and G. Caldarelli, "The scale-free topology of market investments," 2003.

[24] M. E. J. Newman, "Assortative mixing in networks," *PHYS.REV.LETT.*, vol. 89, p. 208701, 2002.

[25] M. G. Beiró, J. I. Alvarez-Hamelin, and J. R. Busch, "A low complexity visualization tool that helps to perform complex systems analysis," *New Journal of Physics*, vol. 10, no. 12, p. 125003, 2008.

[26] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani, "Large scale networks fingerprinting and visualization using the k-core decomposition," in *Advances in Neural Information Processing Systems 18* (Y. Weiss, B. Schölkopf, and J. Platt, eds.), (Cambridge, MA), pp. 41–50, MIT Press, 2006.

Índice alfabético