Instituto Politécnico Nacional

Centro de Investigación en Computación

Enrutamiento semántico

TESIS

Que para obtener el grado de:

Doctorado en Ciencias de la Computación

P R E S E N T A :

M. en C. Miguel Rodríguez Martínez

Directores de tesis:

Dr. Rolando Menchaca Méndez

Dr. Miguel Jesús Torres Ruiz

México, D.F., enero de 2016

SIP-14 BIS

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 12:00 horas del día 2 del mes de diciembre de 2015 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"Enrutamiento semántico"

Presentada por el alumno:

RODRIGUEZ	**MARTÍNEZ**	**MIGUEL**
Apellido paterno	Apellido materno	Nombre(s)

Con registro:

A	1	2	0	4	3	7

aspirante de: **DOCTORADO EN CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Directores de tesis

Dr. Rolando Menchaca Méndez

Dr. Miguel Jesús Torres Ruiz

Dr. Grigori Sidorov

Dra. Nareli Cruz Cortés

Dr. Marco Antonio Moreno Ibarra

Dr. Rolando Quintero Téllez

PRESIDENTE DEL COLEGIO DE PROFESORES

Dr. Luis Alfonso Villa Vargas

INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACION EN COMPUTACION

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México DF el día 11 del mes de diciembre del año 2015, el (la) que suscribe Miguel Rodríguez Martínez alumno (a) del Programa de Doctorado en Ciencias de la Computación con número de registro A120437, adscrito al Centro de Investigación en Computación, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección del Dr. Rolando Menchaca Méndez y Dr. Miguel Jesús Torres Ruiz y cede los derechos del trabajo intitulado Enrutamiento Semántico, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección miguelonrm1@gmail.com. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Miguel Rodríguez Martínez

Nombre y firma

RESUMEN

Con la diversificación y masificación de las de contenido en la Internet actual, buscadores como Google, Yahoo, o Bing han tomado un papel central en el funcionamiento de la Internet. Sin embargo, debido a que todos estos buscadores trabajan sobre los subsistemas de enrutamiento y de resolución de nombres tradicionales, son inherentemente ineficientes debido a que cada consulta requiere de varios viajes a través de la red.

En este trabajo se propone una alternativa novedosa para la organización de los servicios de Internet (o de una red privada) que permite acceder directamente a la información solicitada por el usuario sin la necesidad de un buscador de información centralizado o de un servicio de nombres. Esta nueva organización de la red está basada en un único algoritmo distribuido que enruta las solicitudes de información hacia el nodo que almacena los objetos de datos más relevantes para dicha consulta. Los dos componentes principales del algoritmo, que hemos denominado de enrutamiento semántico, son un algoritmo que mapea objetos de datos y consultas a los nodos de un árbol *k*-ario de altura *h,* y un algoritmo de enrutamiento basado en etiquetas de prefijos que inducen un árbol *k*-ario sobre los nodos de la red.

En los resultados presentados se compara la efectividad del esquema propuesto como herramienta de recuperación de información contra la efectividad del esquema centralizado conocido como Locality Sensitive Hashing. Las métricas de desempeño usadas son el *precision* y *recall*.

ABSTRACT

With the diversification and massification of the sources of Internet content, the web search systems such as Google, Yahoo and Bing have gained a central role in the way the current Internet operates. However, the fact that these search systems work on top of the traditional routing and name resolution subsystems makes them inherently inefficient. This is because the traditional process of retrieving a data object requires traversing the Internet a number of times.

In this thesis, we propose a novel alternative for the organization of the services that implement the Internet (or a private network) that eliminates the use of a centralized search system and allows accessing relevant data objects directly. This new network organization is based on a single distributed algorithm that routes the request issued by the users towards the node in the network that contains the most relevant data objects. The proposed algorithm has two main components, an algorithm that maps data objects and queries to the nodes of a *k*-ary tree of height *h*, and a routing algorithm based on prefix tables that induces a *k*-ary tree over the de network nodes.

We present results where we compare the effectiveness of the proposed scheme as an information retrieval tool against that of a well-known centralized scheme known as Locality Sensitive Hashing. For our experiments we used precision and recall as performance metrics.

AGRADECIMIENTOS

Agradezco a Dios por acompañarme día a día e iluminarme en los momentos difíciles para salir adelante.

Gracias a mi esposa Libia Miriam Martínez de la Rosa por brindarme su apoyo incondicional, además de creer en mí durante el desarrollo y culminación de este trabajo.

Agradezco a mi padre Sebastián, mi madre Eustolia y mis hermanos por creer en mí en todo momento y por apoyarme durante toda mi vida.

Gracias a mi terapeuta Angélica Cortes Vázquez por haberme ayudado en el fortalecimiento de mi persona y ser para poder concluir con éxito esta meta en mi vida.

Un especial agradecimiento a mis directores de tesis, el Dr. Rolando Menchaca Méndez y Dr. Miguel J. Torres Ruiz, por su invaluable apoyo en la asesoría y dirección de esta tesis.

Agradezco a todos los profesores que me brindaron su valiosa enseñanza y su apoyo académico. ¡Gracias!

Mi más sincero agradecimiento al Instituto Politécnico Nacional y específicamente al Centro de Investigación en Computación por haberme permitido ser parte de su valiosa comunidad. Además, de haber abierto sus puertas para poder realizarme profesionalmente.

A mis amigos, por el apoyo y motivación que de ellos he recibido. Especialmente a mi amigo y compañero de grado Sergio Solano Pinedo.

A los sinodales por su dedicación y apoyo en la revisión y recomendaciones del presente trabajo.

Gracias al financiamiento del Instituto Politécnico Nacional, el Instituto México Estados Unidos de la Universidad de California (UC MEXUS) y el Consejo Nacional de Ciencia y Tecnología del México (CONACyT), ya que fue una parte muy importante para la realización de este trabajo.

DEDICATORIA

A Dios por darme la capacidad y oportunidad de poder alcanzar una meta más en mi vida.

A mi esposa Libia Miriam por brindarme su amor, comprensión y apoyo para alcanzar esta gran meta.

A mi padre Sebastián y mi madre Eustolia por quererme tanto y apoyarme en todo momento en mis decisiones.

A mis hermanos Juan, Ángel, Carlos, Rocío y Gabriel por su cariño y apoyo.

Índice

CAPÍTULO 3. Diseño y caracterización de la arquitectura de Internet basada en contenido

Índice de figuras

Índice de tablas

Capítulo 1

Definición del problema

Introducción

A finales de los años 60s y principios de los 70s, cuando las ideas fundamentales sobre las cuales subyace Internet fueron desarrolladas, la telefonía era el único ejemplo de una red de comunicaciones a gran escala, efectiva y exitosa. En este sentido y aunque la solución de comunicación ofrecida por TCP/IP era única y novedosa, el problema que resolvía era en esencia el de la telefonía, es decir, proporcionaba soporte a conversaciones punto a punto entre dos entidades.
Sin embargo, el mundo de las comunicaciones, así como el de las aplicaciones que hacen uso de ellas, ha cambiado dramáticamente desde entonces, por lo que los supuestos sobre los que Internet fue desarrollado ya no son completamente válidos. En particular, podemos mencionar:

- Actualmente, casi todo está disponible en línea y el Internet se ha convertido en la ventana de acceso al mundo. Por lo tanto, los patrones de acceso a los recursos contenidos en Internet han rebasado en mucho el antiguo paradigma de la comunicación punto a punto y extremo a extremo.
- Los avances en la codificación digital han hecho posible que Internet no sea únicamente una fuente para acceder a texto. Actualmente la mayor cantidad de información que transita a través de Internet son flujos de voz, imágenes y video.
- La Web se ha masificado y ha permitido que virtualmente cualquier persona creé, descubra y consuma recursos. Como resultado, enormes cantidades de nuevos contenidos son producidos anualmente.
- Los avances en el hardware han hecho posible conectar cualquier cosa al Internet: no sólo supercomputadoras y estaciones de trabajo, sino fábricas, infraestructuras municipales, teléfonos, carros, electrodomésticos, y hasta encendedores de la luz.
- Los buscadores de contenido como Google, Bing o Yahoo han pasado a formar parte fundamental del Internet. Actualmente, y gracias a estos servicios ya no es necesario conocer *a priori* el nombre del servidor que aloja el contenido que se desea acceder.

Aún y cuando el protocolo de Internet (IP) ha excedido todas las expectativas iniciales, fue diseñado primordialmente para soportar conversaciones entre dos puntos, lo que no lo hace particularmente eficiente como medio de distribuir contenido, aplicación que por mucho domina el tráfico actual de Internet.

La naturaleza punto a punto de Internet se ve reflejada en su funcionamiento basado en datagramas. Los datagramas IP únicamente pueden especificar puntos finales de comunicación, es decir, las direcciones IP de la fuente y del destino. En la presente propuesta se propone generalizar la arquitectura de Internet, removiendo esta restricción y permitiendo que el contenido almacenado en Internet pueda ser direccionado por medio de identificadores que reflejen el contenido semántico de los objetos de datos. Con este cambio, se busca que la arquitectura de Internet se

centre en los datos y no en sus contenedores, y además, llevar a cabo mejoras al funcionamiento, entre las cuales, podemos mencionar las siguientes:

- Las aplicaciones de Internet generalmente se diseñan en términos de la información, y no en base a dónde está almacenada dicha información. Con el modelo de Internet actual, se ha generalizado el uso de *middlewares* dependientes de la aplicación que son usados para mapear entre el modelo de la aplicación y de Internet. Con el modelo propuesto, las aplicaciones se podrán desarrollar directamente, eliminando los *middlewares*, así como la ineficiencia relacionada con ellos.
- La búsqueda de contenido en Internet involucra la participación de tres sistemas relativamente independientes, es decir, de un buscador como Google, de un servidor de nombres de dominio (DNS), así como de un conjunto heterogéneo de algoritmos de enrutamiento. Con una arquitectura como la propuesta en el presente documento, será posible integrar estas tres funciones en un mismo algoritmo, con las consiguientes mejoras en desempeño, robustez y escalabilidad.

Planteamiento del problema

Como se ha mencionado anteriormente, Internet fue originalmente diseñado para soportar comunicaciones punto a punto entre pares donde al menos uno de ellos conoce *a priori* la dirección de su contraparte, es decir, su dirección IP, el número de puerto y el protocolo de transporte utilizado. Este esquema era adecuado para el Internet de los años 80's porque el número de computadoras interconectadas era del orden de cientos de equipos, y porque los usuarios conocían de antemano los nombres de los servidores con los que generalmente interactuaban. Sin embargo, las condiciones en las que opera el Internet actual son radicalmente distintas:

- Debido principalmente al éxito de Internet y de su pila de protocolos, en la actualidad hay cientos de millones de computadoras conectadas a Internet. Más aún, debido a la tendencia actual de conectar todo (electrodomésticos, autos, teléfonos, etc.) a Internet, se espera que el número de dispositivos interconectados crezca aún más.
- Con el advenimiento de las redes sociales, de los blogs y de los sitios como youtube y flicker, así como la adopción generalizada de los sistemas de información Web, los usuarios de Internet, que también son generadores de contenido, han pasado de ser unos cientos a sobrepasar los cientos de millones.

Debido a esta masificación y diversificación de las fuentes de contenido, los buscadores como Google, Bing o Yahoo se han convertido en una parte fundamental del funcionamiento de Internet debido a que permiten a los usuarios realizar búsquedas de objetos de datos cuyo contenido es semánticamente cercano a una serie de palabras claves. Sin embargo, estos servicios de búsqueda se han construido sobre la estructura y protocolos actuales de Internet, lo que los hace altamente ineficientes. Como se muestra en la Figura 1.1, el proceso para encontrar y acceder a un objeto de datos (una página web, un video, una imagen, etc.) involucra a tres sistemas relativamente independientes entre sí, así como una serie de conexiones a puntos potencialmente distantes en la red. En el ejemplo mostrado en la Figura 1.1., puede observarse que es necesario realizar cuatro consultas a diferentes servidores localizados en la red (dos al servidor de nombres, uno a un servidor de búsquedas y otro al servidor que almacena el objeto de datos buscado), para poder acceder a un objeto de datos.

1. Enrutamiento lleva una consulta al DNS para resolver el URL de Google (----►).
2. DNS responde con la dir. IP de Google (----►).
3. Enrutamiento lleva una consulta a Google (-·-►).
4. Google responde con una lista de URLs. (-·-►).
5. Enrutamiento lleva una consulta al DNS para resolver el URL seleccionado (----►).
6. DNS responde con la dir. IP del sitio seleccionado (----►).
7. Enrutamiento lleva una consulta al sitio seleccionado (——►).
8. El sitio responde con el contenido direccionado por la URLs (——►).

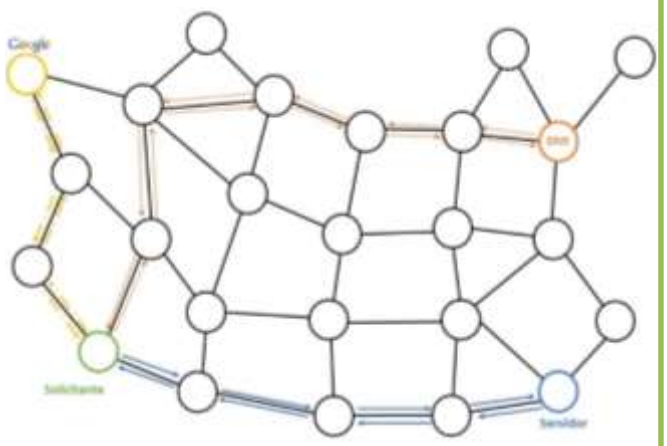


Fig. 1.1. Procedimiento para obtener información en el Internet actual.

La ineficiencia relacionada con los múltiples viajes a través de Internet para poder acceder a un objeto de datos impone severas restricciones a la escalabilidad de la red. Lo anterior, aunado a la problemática relacionada con los servidores centralizados que implementan tanto los servicios de resolución de nombres como de búsquedas, nos lleva a considerar que es necesario replantearnos la forma en cómo debe funcionar Internet.

En este sentido, se propone una nueva organización del Internet, que esté centrada en el contenido de los objetos de datos y no en las direcciones IP de los servidores que los alojan.

Para materializar el esquema anterior, uno de los problemas más importantes que se deben resolver es el de acceder de manera eficiente y efectiva a objetos de datos alojados en la Internet semánticamente relevantes.

Formalmente, desde el punto de vista de recuperación de información, el problema de recuperar el objeto "más relevante" de la Internet se puede formular de la siguiente manera: dada una red $G = (V, E)$ compuesta por un conjunto de nodos V interconectados por un conjunto de enlaces E, un conjunto de todas las posibles palabras clave K, un conjunto de objetos $O = \{o_1, o_2, \dots, o_n\}$ descrito por un conjunto de palabras clave $k_{o_i} \subseteq \mathrm{K}$, una consulta q que también puede ser descrita por un conjunto de palabras clave $k_{o_i} \subseteq \mathrm{K}$, una función $\lambda : O \to V$ que mapee objetos de datos a los nodos que los almacenan (o los genere en el caso de sensores) y una función de medida de similitud ρ: $\mathrm{K}^2 \times \mathrm{K}^2 \to [0,1]$ con 0 indicando similitud máxima y 1 indicando no similitud. Encontrar el objeto o_i y la componente de Internet $\lambda(o_i)$ tal que $\rho(k_q, k_{o_i})$ sea mínima para todos $o_i \in O$.

Una vez que se ha identificado el objeto de datos semánticamente relevante para una consulta, el segundo problema consiste en calcular una ruta eficiente desde $\lambda(o_i)$ hasta el nodo $u \in V$ donde fue generada la consulta.

Objetivo general:

- Diseñar, implementar y caracterizar experimentalmente un algoritmo de enrutamiento semántico capaz de recuperar objetos de datos almacenados en una red de computadoras con base en la semántica de los datos.

Objetivos específicos:

- Diseñar, implementar y caracterizar un protocolo distribuido de etiquetado que establezca un orden total sobre los nodos de la red.
- Diseñar, implementar y caracterizar un algoritmo distribuido de enrutamiento compacto basado en etiquetas compuestas de prefijos.
- Definir e implementar una nueva función de similitud semántica entre conceptos definidos en una ontología que sea asimétrica y que tome en cuenta el contenido de información de los conceptos.
- Definir e implementar un algoritmo que mapee objetos de datos especificados por una serie de palabras clave en un espacio *m*-dimensional donde cada una de las dimensiones represente conceptos de una ontología.
- Diseñar e implementar un algoritmo para embeber árboles k-arios de altura *h* en una híper-esfera de *n*-dimensiones.
- Definir e implementar una función hash semántica cuyo domino sean los objetos de datos y consultas especificadas por un conjunto de palabras claves y su co-dominio sea un espacio de etiquetas basadas en prefijos generadas a partir de un alfabeto Σ tal que $|\Sigma| = k$.
- Desarrollar una serie de experimentos para caracterizar la efectividad del sistema propuesta con base en su capacidad de recuperar información.

Justificación

Con la Internet actual, los viajes que se hacen a través de la red en busca de contenido, implican el uso de varios subsistemas centralizados como el servidor de nombres y el servidor de búsquedas, por lo que se tienen puntos centralizados potenciales de fallas. Además con el crecimiento masivo de internet, las tablas de enrutamiento empleadas por los principales protocolos usados en la actualidad, crecen de manera desmesurada provocando que se usen cada vez más recursos de cómputo para llevar a cabo un enrutamiento óptimo. Por otro lado, como se explicó anteriormente, el Internet actual está basado en la comunicación punto a punto, por lo que se convierte en un medio ineficiente para transportar contenido.

Dadas las condiciones anteriores, es indispensable empezar a desarrollar nuevas alternativas, quitando con ello la necesidad de tener servidores de nombres centralizados, servidores para hacer búsquedas de contenido en la red y reducir el uso de cómputo para llevar a cabo el enrutamiento. Lo anterior, con el fin de que el contenido almacenado en la red juegue el papel más importante al momento de crearlo y accederlo.

Con la implementación de la arquitectura propuesta, las aplicaciones diseñadas en términos de la información, se podrán desarrollar directamente, dejando a un lado el uso de *middlewares*, así como la ineficiencia relacionada con ellos.

Además, la arquitectura propuesta se puede adecuar para ser usada en sistemas basados en *Cloud Computing* [1] en la que el contenido es almacenado en la red y donde la localización física del mismo pasa a segundo término, pero, garantizando que dicho contenido siempre sea accesible a sus usuarios.

Organización de la tesis

La presente tesis está organizada de la siguiente manera:

- Capítulo 1. Se presenta el planteamiento del problema, se esboza una solución y se justifica la realización de la solución.
- Capítulo 2. En este capítulo se hace un estudio relacionado a los trabajos relevantes en enrutamiento compacto, redes orientadas a contenido, tablas hash, entre otros, con el objeto de conocer los avances que se tienen en cada campo.
- Capítulo 3. Aquí se presenta el diseño, implementación, y caracterización de la nueva arquitectura de Internet basada en contenido.
- Capítulo 4. Se presenta un análisis de desempeño de los algoritmos propuestos con base en los escenarios de evaluación.
- Capítulo 5. En este último capítulo se proporcionan las conclusiones finales y se realizan recomendaciones generales sobre la arquitectura realizada.

Capítulo 2

Estado del arte

2.1. Enrutamiento compacto

2.1.1. Enrutamiento escalable en MANETs usando etiquetas prefijo y funciones hash distribuidas [2]

2.1.1.1. Introducción

A pesar de los muchos avances recientes en hardware y software, proveer conectividad e información continua para personas y dispositivos durante su movimiento sigue siendo un gran reto. Se dice que, al menos en parte, esto es debido a los límites de escalabilidad de los protocolos de enrutamiento actuales para redes móviles ad hoc (MANET).

Una mirada más de cerca a los protocolos de enrutamiento para MANETs actualmente usados revela que las direcciones de los nodos usadas en la transmisión de paquetes de datos (e.g., direcciones MAC o IP) son realmente nombres, lo cuales son constantes y asignados a nodos independientemente de su localización en la MANET. Consecuentemente, las fuentes no tienen que usar un mapeo del nombre de un destino hacía su localización. Sin embargo, ya que los nombres son usados directamente en tablas de enrutamiento, tales tablas deben ser actualizadas conforme los nodos cambien sus localizaciones relativas en la red. Más allá, ya que una dirección de nodo no tiene correlación a su localización relativa en una MANET, la única manera de encontrar y establecer una ruta hacia un destino es a través de alguna forma de inundación manejada por la fuente o destino. La inundación llevada a cabo por el destino se usa en protocolos de enrutamiento proactivos (e.g., OLSR[3]) para diseminar ya sea distancias a todos los destinos o información del estado para todos los nodos, tal que los nodos puedan mantener un registro para cada destino. La inundación realizada por la fuente se usa en protocolos de enrutamiento bajo demanda (e.g., AODV[8], DSR[6]), en los cuales la fuente inunda una solicitud de ruta (RREQ) que es respondida por el destino o por nodos con rutas hacia él. Claramente, ninguna propuesta es eficiente para MANETs muy grandes o muy dinámicas, porque el alto uso de paquetes de señalización crece linealmente con respecto al número de destinos o enlaces.

El protocolo de enrutamiento prefijo sobre un conjunto de elementos (PROSE) usa un etiquetamiento prefijo auto organizado de nodos y el empleo de funciones hash distribuidas que van de identificadores de nodos hacia etiquetas prefijo para integrar eficientemente nombres, direccionamiento y enrutamiento en una MANET.
Los nodos usan señalización salto a salto para asignarse etiquetas prefijo denotando su localización relativa a un nodo raíz, elegido de acuerdo a un cubrimiento por amplitud de la red. Las etiquetas prefijo asignadas a los nodos definen implícitamente al menos una ruta entre cualesquiera dos nodos. Para permitir a las fuentes enrutar hacia los destinos, se construye y mantiene una tabla hash distribuida (DHT) en la red para almacenar el mapeo entre el identificador de cada destino (e.g., una dirección MAC o IP) y la etiqueta prefijo asignada al nodo. Cada destino usa una función hash común para mapear su identificador único hacia una etiqueta prefija pública, y envía su propio mapeo hacia la etiqueta prefija pública obtenida. El nodo con el parecido más cercano a la etiqueta prefijo pública se convierte en el *ancla* para ese nodo destino. El nodo fuente, usa la misma función

hash para enviar una solicitud al ancla de un destino, preguntando por la etiqueta prefijo actual del destino.

La diferencia clave entre PROSE y las demás propuestas basadas en agrupamiento o direcciones prefijo es que, los prefijos en PROSE son asignados usando una auto organización con búsqueda por amplitud de los nodos, más que el agrupamiento de nodos en conjuntos o grupos controlados por nodos especiales o linderos.

2.1.1.2. Funcionamiento de PROSE

El protocolo PROSE enruta paquetes de fuentes hacia destinos al asignar etiquetas prefijo a nodos, y el mapeo dinámico de identificadores de nodo únicos (NID) (e.g., direcciones MAC o IP) hacia etiquetas prefijo.

Además, construye y mantiene un grafo etiquetado acíclico dirigido (LDAG) con raíz en un nodo elegido de manera distribuida, el cual es llamado nodo raíz. La elección del nodo raíz es similar a la elección de la raíz en el algoritmo de un árbol de expansión distribuido. Se propagan paquetes de señalización de vecino a vecino desde la raíz hacia toda la red de misma manera como en la búsqueda por amplitud. Esto proporciona que cada nodo reciba una etiqueta prefijo del mismo nodo raíz. Un nodo anuncia a sus vecinos su propia etiqueta prefijo y las etiquetas prefijo que asigna a sus hijos en el LDAG, y almacena las etiquetas prefijo que escucha de sus vecinos. En su publicación los autores de PROSE asumen que un nodo informa a sus vecinos sólo la etiqueta prefijo más pequeña que es asignada por sus parientes. La figura 2.1 muestra un ejemplo de una MANET en la que el nodo *A* se elige como nodo raíz.

Claramente, una fuente debe saber la etiqueta prefijo de su destino intencionado para que el enrutamiento con prefijos sea efectivo. Sin embargo, almacenar todos lo mapeos de NIDs hacia etiquetas prefijo en uno (e.g., el núcleo) o sólo unos cuantos nodos podría provocar cuellos de botella y puntos de únicos de fallas. En consecuencia, los nodos construyen y mantienen una tabla hash distribuida (DHT) para almacenar los mapeos a través de toda la red, y llevar a cabo las operaciones de suscripción-publicación usando señalización de estado suave, el cual usa etiquetas prefijo para enrutar paquetes hacia nodos específicos.

Cada destino usa una función hash consistente que toma como entrada su NID, y regresa su etiqueta prefija pública, la cual establece la localización dónde su mapeo debería ser almacenado. El nodo con la etiqueta prefijo más parecida a la etiqueta prefijo se convierte en el *ancla* para ese destino. Posteriormente, un destino publica su presencia en la red al enviar su propio mapeo (i.e., NID y etiqueta prefijo) a su ancla (ver Figura 2.1. (a)). Para encontrar a un destino y suscribirse a él, una fuente usa la misma función hash consistente con el NID del destino como entrada. Entonces, la fuente envía una solicitud hacia la etiqueta prefijo del ancla resultante. El ancla del destino responde a la fuente con la etiqueta prefijo del destino, o reenvía la solicitud hacia el destino. Entonces, el destino responde directamente a la fuente. Una vez que el destino y la fuente conocen las etiquetas prefijo uno del otro ellos se pueden comunicar directamente (ver Figura 2.1. (c)). Como se puede ver en la figura, es claro que PROSE soporta enrutamiento múlticamino prefijo, i.e., se podrían usar múltiples caminos desde la fuente al destino. (e.g., nodos *A*, *B*, *E* y *G* tienen múltiples caminos prefijos hacia el nodo *K* en la Figura 2.1. (c)).

EL LDAG sirve como una estructura fuerte tolerante a la salida de nodos de la red, la cual por omisión asigna a un nodo con la etiqueta prefijo más parecida cuando una solicitud para una concordancia falla (cuando el nodo buscado no se encuentra). El algoritmo de enrutamiento que trabaja sobre el LDAG usa una estrategia greedy. Cuando encuentra un mínimo local, escoge el siguiente salto con el NID más bajo. Dado que los nodos tienen etiquetas prefijo, la opción del NID no afecta el camino hacia el destino. Un nodo usa la lógica de máxima similitud prefija del algoritmo de enrutamiento para escoger su siguiente salto.

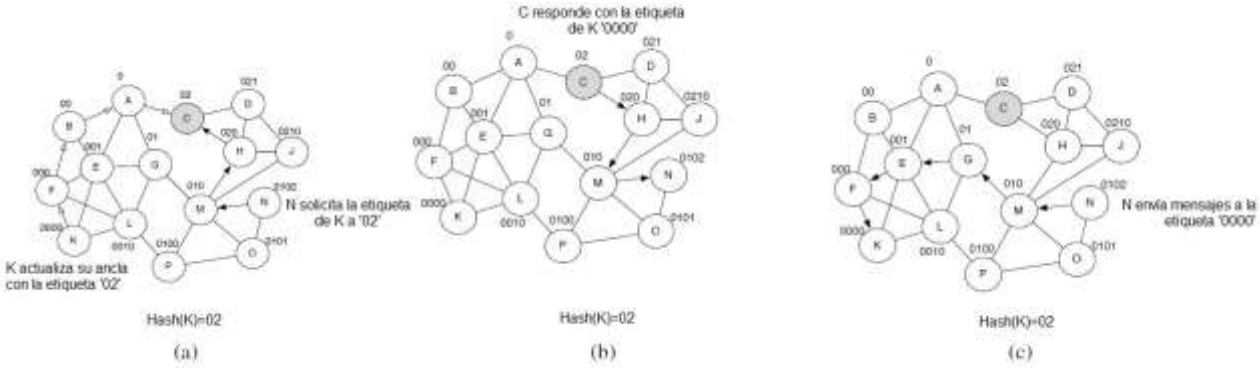


Fig. 2.1. Funcionamiento de PROSE: El nodo A es la raíz, nodo C es el ancla del nodo K; el nodo N solicita la etiqueta prefijo para el nodo K y entonces enrutar hacia él directamente.

2.1.1.3. Enrutamiento sobre etiquetas prefijo

2.1.1.3.1. *Estructura de las etiquetas prefijo*

Sea Σ el alfabeto que contiene un número finito de símbolos y Σ^* el conjunto de todas las cadenas sobre Σ tal que $|\Sigma| \geq 2$. Cada nodo etiqueta a sus enlaces para cada uno de sus vecinos con una letra ω de Σ. Si ω_i representa la letra asignada al *i-ésimo* enlace de cualquier nodo entonces, $\omega_i \mapsto \{\omega_i \in \Sigma | \omega_i \neq \omega_{i+1} \forall_i \leq d-1\}$, donde d es el grado del nodo. Por lo tanto, se asigna una unica letra a cada enlace que conecta a algún nodo con sus vecinos. La lógica etiquetadora etiqueta a cada nodo en el LDAG de manera parecida a la búsqueda por amplitud. Dado que el LDAG se puede organizar como un árbol $k-ary$, siendo k el grado del LDAG, a cada hijo (arriba de k) se le asigna una etiqueta prefijo Λ como se define a continuación:

Etiqueta prefijo: Una etiqueta prefija Λ para el nodo y es una palabra dentro de Σ^* tal que $\Lambda = \Lambda_{parent} \odot l'$, donde Λ_{parent} es la etiqueta prefijo obtenida del padre y $\odot$ es el operador de concatenación donde Λ_{parent} es concatenada con un único sufijo sobre k diferentes opciones de Σ para formar Λ.

Se deduce de la definición anterior que la etiqueta prefija Λ, de un nodo identifica únicamente al nodo en un LDAG dado. Las etiquetas prefijo de los nodos definen una relación *predecesor* en el LDAG, denotada por $\hookleftarrow$, tal que para dos nodos cualesquiera s y d en el LDAG:

1) $s \leftarrow d$: $s\ precede\ a\ d$. En este caso, y entonces se puede alcanzar a *d* al recorrer los subárboles de *s*.
2) $d \leftarrow s$: $d\ precede\ a\ s$. En este caso, y entonces se puede alcanzar a *d* yendo hacia arriba del flujo de *s* al recorrer los antecesores de *s*.
3) $s \approx d$: d y *s* son pares donde $|\Lambda_d| = |\Lambda_s|$ y ambos comparten un antecesor común. En este caso, d puede ser alcanzado al recorrer hacia arriba el antecesor común hasta que se mantenga $r \leftarrow s$, donde r es el antecesor común, y entonces recorrer hacia abajo del subárbol más parecido, mientras $r \leftarrow d$ se mantenga y *d* sea alcanzado. Por lo tanto, $d \approx s \Longrightarrow r \leftarrow s \Longrightarrow r \leftarrow d$.
4) $d \approx s$: Este es un caso especial del anterior, donde $|\Lambda_d| \neq |\Lambda_s|$ y el único antecesor común es el nodo raíz.

Los nodos construyen y mantienen un LDAG con raíz en un nodo elegido usando mensajes *hola* intercambiados entre los vecinos del uno vecindario. Cada mensaje *hola* específica el NID de la raíz, un número secuencial que se incrementa monotónicamente el cual es asignado por el nodo raíz, la etiqueta prefijo y el NID del nodo emisor, y una lista de tuplas con la asignación de etiquetas prefijo hacia NIDs. El algoritmo para la elección de la raíz escoge el nodo con el cubrimiento más grande hecho a un salto y rompe ataduras con el NID de la raíz más chica. En cuanto la red inicia a autoorganizarse con etiquetas prefijo, el proceso de etiquetamiento puede producir múltiples LDAGs, la etiqueta de cada uno de ellos se compara y la etiqueta más grande lexicográficamente se elige como el dominante.

Consideremos el ejemplo en la Figura 2.1 Las etiquetas prefijo para los nodos son asignadas sobre el alfabeto $\sum = 0, 1, 2$. A la raíz del árbol se le asigna una letra de $\sum$ y los nodos unidos a la raíz son etiquetados 00 y 01 sucesivamente. En el siguiente nivel, a cada nodo se le asigna una única etiqueta prefijo sobre el enlace combinado con el prefijo de su padre. Los nodos *E* y *M* son etiquetados con 001 y 010, respectivamente.

2.1.1.3.2. *Enrutamiento*

Para enrutar hacia un destino *d*, un nodo *s* escoge al enlace de cualquiera de sus vecinos ubicados a dos saltos de distancia, que ofrezca la longitud máxima de similitud al comparar la etiqueta prefijo de cada uno con la etiqueta prefijo del destino. Eso es simplemente la lógica de máxima similitud prefija, la cual selecciona el siguiente salto usando una estrategia greedy que considera a los vecinos ubicados a dos saltos de un nodo, y puede encontrar caminos más cortos que el enrutamiento tradicional basado en árbol prefijo al sacar ventaja de la diversidad de caminos de un LDAG comparado con un árbol prefijo. Por ejemplo, si existe una etiqueta en el dos-vecindario que es lexicográficamente más parecida al destino, se escoge al siguiente salto tal que el paquete se envía al nodo en lugar de enrutarlo a través del padre en el árbol prefijo. Se asegura que esta estrategia greedy no encuentra un mínimo local al seleccionar aleatoriamente el siguiente salto cuando todos los nodos ofrecen la misma similitud prefija.

A partir de la relación de predecesor inducida por las etiquetas prefijo en el LDAG de una MANET, un nodo dado selecciona su siguiente salto mediante su etiqueta prefijo de acuerdo a los cuatro posibles casos permitidos por la relación de predecesor. El siguiente lema muestra que PROSE converge al proporcionar rutas correctas.

Lema 3.2.1. *Las rutas basadas en prefijos construidas usando PROSE son correctas en una red conectada sin cambios en la topología.*

Demostración: Por la definición de las etiquetas prefijo, cada nodo tiene una única etiqueta prefijo, y el LDAG es libre de ciclos al construirse. Sin pérdida de generalidad, considere una fuente *s* y un destino *d*. Debido a que ellos pertenecen al LDAG, deben satisfacer uno de los cuatro casos de la relación de predecesor, y cada nodo que es elegido como siguiente salto hacia *d* iniciando en *s* debe satisfacer también la misma relación. Debido a que el LDAG es finito y no cambian, debe existir al menos un camino libre de ciclos desde *s* hasta *d* en el LDAG.

Sin embargo, la topología de una MANET cambia constantemente y los nodos se unen o dejan la red arbitrariamente. Esto resulta en etiquetas inconsistentes en el punto de la etiquetación. Para preservar la consistencia, se fuerza a un *etiquetado estricto* al ordenar las etiquetas prefijo usando números secuenciales.

Sea *L* que representa una tupla $(\mathrm{S}_{\mathrm{n}}^{\mathrm{a}}, \Lambda_{\mathrm{n}}^{\mathrm{a}})$, donde $\mathrm{S}_{\mathrm{n}}^{\mathrm{a}}$ denota el número secuencial que se origina en *a* y es reenviado por el nodo *n*, y $\Lambda_{\mathrm{n}}^{\mathrm{a}}$ denota la etiqueta prefija del nodo actual con respecto a *a*. *S* es un entero que se incrementa monótonamente, mientras Λ es una palabra en Σ^*. Se define un operador $\prec$ sobre el conjunto del par ordenado de identificadores en *L*. Si $\mathrm{L}_{\mathrm{x}}^{\mathrm{a}}$, $\mathrm{L}_{\mathrm{y}}^{\mathrm{a}}$ son dos tuplas entonces,

$$\left\{\mathrm{L}_{\mathrm{x}}^{\mathrm{a}} \prec \mathrm{L}_{\mathrm{y}}^{\mathrm{a}} \middle| \mathrm{L}_{\mathrm{x,y}}^{\mathrm{a}} \in \Sigma\right\} \tag{2.1}$$

$$\mathrm{if}\left\{\left(\mathrm{S}_{\mathrm{x}}^{\mathrm{a}} < \mathrm{S}_{\mathrm{y}}^{\mathrm{a}}\right) \vee \left[\left(\mathrm{S}_{\mathrm{x}}^{\mathrm{a}} = \mathrm{S}_{\mathrm{y}}^{\mathrm{a}}\right) \wedge \left(\Lambda_{\mathrm{x}}^{\mathrm{k}} \succ \Lambda_{\mathrm{y}}^{\mathrm{i}}\right)\right]\right\}$$

Para satisfacer la afirmación de que esta tupla establece un ordenamiento entre los nodos, se demuestra que el operador $\prec$ es antireflexivo y transitivo sobre el conjunto Σ^*.

Lema 3.2.2. *La relación* $\prec$ *es un ordenamiento parcial de tuplas sobre* Σ^*.

Demostración: La primera parte del lema es intuitiva, ya que dos tuplas son iguales si sus números secuenciales y las etiquetas prefijo son iguales. Por lo tanto, $\mathrm{L}_{\mathrm{x}}^{\mathrm{a}} \prec \mathrm{L}_{\mathrm{y}}^{\mathrm{a}}$ o $\mathrm{L}_{y}^{\mathrm{a}} \prec \mathrm{L}_{x}^{\mathrm{a}}$, sólo se puede mantener a uno y por lo tanto es anti reflexiva. Para probar que la transitividad se mantiene, considere tres etiquetas y para conveniencia se les llamaran $L_1, L_2, L_{3,}$ tal que $L_1 \prec L_2$ y $L_2 \prec L_3$. De la ecuación 1 se puede ver que, $S_1 < S_2 < S_3$ donde las S_s crecen de manera monótona. Esto implica que $S_1 < S_3$. Por lo tanto, si $S_1 \not< S_3$ entonces $L_1 \prec L_3$ sólo si $S_1 = S_3$ y $\Lambda_1 < \Lambda_3$ (la comparación para Λ está dentro del alfabeto). Debido a que se sabe que Λ se mantiene para la transitividad, la relación $\prec$ está en orden sobre $\sum^*$ y establece una relación sucesor-predecesor.

Una vez que se ha demostrado que las etiquetas prefijo secuenciadas preservan un ordenamiento, se muestra en un teorema más general que el algoritmo de enrutamiento en PROSE converge para caminos libres de ciclos, incluso cuando se tiene una topología dinámica.

Lema 3.2.3. *Un modelo de enrutamiento prefijo con números secuenciales enruta correctamente.*

Demostración: del lema 3.2.1., se sabe que todos los nodos tienen una etiqueta y que cada nodo comparte una relación con todos sus nodos a distancia uno. Del lema 3.2.2., se tiene que cada nodo etiquetado está ordenado con respecto a la longitud lexicográfica de la etiqueta prefijo y un número secuencial. Por lo tanto, si el algoritmo de enrutamiento enruta entre dos nodos, la estrategia *greedy* garantiza que mejora alguna métrica (conteo de salto, distancia, carga, etc.) hacia algún destino en cada salto. Por lo tanto, un modelo de enrutamiento prefijo con números secuenciales enruta correctamente, incluso cuando se presentan cambios topológicos.

2.1.1.4. Construcción de la Tabla Hash Distribuida (DHT)

Como se estableció anteriormente, para evitar cuellos de botella y puntos únicos de fallas, se deben diseminar los mapeos de NIDs a etiquetas prefijo en toda la red. El protocolo PROSE usa una función hash consistente para lograr tal objetivo. Cada destino publica su existencia al enviar una *solicitud de publicación* al nodo de quien su etiqueta prefijo tenga la mejor coincidencia con el resultado obtenido de aplicar la función hash al NID del destino, a este nodo se le llama el ancla del destino. Una solicitud de publicación es una tupla que consiste del NID y la etiqueta prefijo de algún nodo. Una fuente que está esperando para comunicarse con un destino, envía una *solicitud de suscripción* al nodo de quien su etiqueta prefijo tenga la mejor coincidencia con el resultado obtenido de aplicar la función hash al NID del destino intencionado (i.e., el ancla del destino). Una solicitud de suscripción consiste del NID, la etiqueta prefijo de la fuente y el NID del destino intencionado. El ancla puede responder la solicitud de la fuente o reenviar la solicitud hacia el destino, dependiendo de la naturaleza del intercambio entre la fuente-destino. En este apartado se asume que el ancla responde a la fuente de la solicitud.

El protocolo PROSE usa señalización de estado suave en la diseminación de solicitudes de publicación y suscripción, en el que las fuentes y los destinos son responsables del éxito de sus solicitudes, y las anclas trabajan sobre una base de mejor esfuerzo. Para ahorrar ancho de banda, se agregan las solicitudes hacia diferentes anclas en los mensajes *hola* intercambiados entre nodos. Para manejar los cambios en la topología de la red, los destinos actualizan sus anclas periódicamente, además, los eventos que cambian la etiqueta prefijo de un nodo también la actualizan. Una actualización refresca el mapeo almacenado en las anclas. Adicionalmente se mantienen temporizadores en los nodos fuentes y anclas para refrescar tales mapeos y asegurar que cualquier información vieja sea eliminada.

2.1.1.5. Adaptación de PROSE a cambios dinámicos en la red

Las etiquetas prefijo asignadas a los nodos se actualizan conforme la red cambia.

2.1.1.5.1 *Re-etiquetamiento en los nodos*

Los nodos adquieren una nueva etiqueta cuando se recuperan de algún cambio en la red. Dependiendo del tipo de nodo, la red se organiza de manera diferente. El evento de re-etiquetamiento se divide en uno de *unión* o uno de *abandono*. Un evento de *unión* se maneja de manera similar al proceso de etiquetamiento inicial. Cada nodo adquiere una etiqueta del nodo que es su predecesor inmediato en el punto de unión. Por otro lado, un evento de *abandono* se maneja de manera diferente para diferentes tipos de nodos. Si el nodo que abandona es un *nodo hoja*, entonces no se afectan más nodos por este evento. Sin embargo, si el nodo saliente es un nodo interno, entonces la adaptación es más complicada. Esto es principalmente debido a la construcción del LDAG, en el que las etiquetas del subárbol son derivadas de los prefijos o algún ancestro. Sin embargo, mientras los prefijos deben cambiar debido a la reposición, los sufijos permanecen únicos en el entero subárbol.

Para entender más a fondo cómo se maneja el evento en el que un nodo interno *abandona* la red, consideremos tres nodos x, y y z como se muestra en la Figura 2.2. Estos tres nodos tienen al mismo padre p y el LDAG entero tiene como raíz al nodo r. Sea $k\alpha p$ la etiqueta más reciente de p y sean $k\alpha px$, $k\alpha py$, $k\alpha pz$ las etiquetas más recientes de sus hijos. Además se asume que existe algún camino entre los tres subárboles tal que un nodo en cada subárbol puede alcanzar un nodo arbitrario en su subárbol hermano a través de un camino distinto hacia su padre. Si el nodo p fallara o abandonara, se llevan a cabo los siguientes pasos para asegurar que el resto de la red permanezca en gran parte no afectada:

- Cada hijo -$k\alpha px$, $k\alpha py$, $k\alpha pz$ inicia una rutina para descubrir sus nodos hermanos (o aquellos de los que sabía de su existencia) usando rutas distintas a la fallida.
- Conforme cada hijo descubra caminos, el hermano con el identificador global más bajo, digamos $k\alpha px$ se elige como una *raíz-secundaria* y adquiere la etiqueta prefijo de su padre perdido como su propia etiqueta.
- Una vez que se establece una *raíz-secundaria*, los hermanos restantes continúan usando su etiqueta prefijo sin modificaciones pero enrutan hacia sus hermanos de los demás subárboles usando rutas alternativas.
- Si un nodo estuviera por unirse a cualquiera de los nodos hermanos, digamos $k\alpha px$ el cual tiene caminos más cortos hacia todos sus hermanos entonces la raíz-secundaria renuncia a su posición y se restablece una nueva raíz. El nuevo nodo raíz envía una actualización con un número secuencial más grande para asegurar que sus caminos más cortos no son resultado de un ciclo en la red.

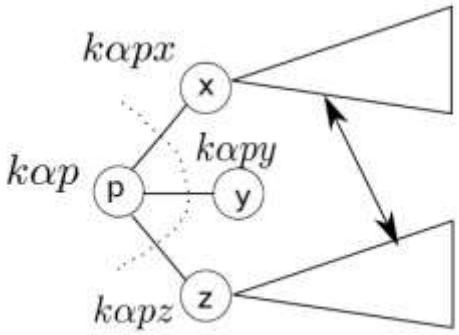


Fig. 2.2. Capacidad de recuperación de las etiquetas prefijo

2.1.1.5.2 *Re-etiquetado en el ancla*

Se asume que algún nodo con etiqueta $k\alpha x$ es el ancla designada para el nodo y en la red. El nodo y actualiza proactivamente a su ancla con su etiqueta actual. Si el nodo $k\alpha x$ falla o se mueve hacia otro lado en la red, o si no existe algún nodo con tal etiqueta, entonces se designa a un nodo en algún subárbol que tenga la máxima coincidencia prefija como el *ancla-secundaria.* Se debe notar que el ancla-secundaria está lógicamente en el camino hacia el ancla. Por ejemplo, si un nodo con etiqueta $k\alpha x$ se apagara y el prefijo de esta etiqueta fuera $k\alpha$, entonces el último nodo activo comprometido en proveer el servicio de directorio está en el subárbol α. En el escenario del peor caso, el modelo realiza respaldos en el nodo raíz, debido a que ningún otro nodo tiene un prefijo parecido al destino que se está buscando. Si otro nodo se une como un hijo al mismo padre y adquiere una etiqueta que se parece más al ancla, las actualizaciones proactivas del nodo expiran automáticamente al nodo padre el cual era el ancla y se enrutan hacia la nueva ancla.

En el caso de topologías cambiantes, reiniciar las etiquetas prefijo de un subárbol entero el costo puede ser alto, ya que se depende de la cantidad de nodos que existan en ese subárbol. En el momento en el que algún nodo hijo no sea capaz de alcanzar a cualquiera de sus hermanos, o volverse completamente desconectado, inicia un proceso de reetiquetamiento. El reetiquetamiento inicia con el prefijo de un subárbol conocido o con un nuevo LDAG con raíz en el hermano desconectado. Este proceso se determina al establecer un *temporizador-espera*. El temporizador-espera funciona por un cierto tiempo para permitir que el enlace se reconecte. Cuando el *temporizador-espera* expira asume la desconexión e inicia el proceso de reinicio de etiqueta.

2.1.1.5.3 Temporizadores de actualización adaptables

En esta sección se describe un temporizador inteligente que determina si se necesita enviar una actualización dentro de cierto tiempo. Esta decisión se toma basándose en los cambios topológicos y etiquetas prefijo que ocurren dentro del vecindario del nodo. A cada nodo se le coloca un tiempo de espera (δ). Cuando el temporizador δ expira, el nodo transmite un mensaje *hola* con la carga correspondiente. Dentro de δ si un nodo detecta cambios topológicos, engloba tales cambios y

determina si un número de cambios cruzó un cierto umbral (T_δ). El número de cambios en el dos-vecindario determina por turnos al umbral. Se define otro umbral para el número de mapeos y una vez que se han detectado cambios en los mapeos los cuales son mayores que el umbral del mapeos (T_m), se envía una actualización con los nuevos mapeos. Si P_δ se define como la probabilidad del umbral topológico alcanzado y P_m es definido como la probabilidad del umbral de mapeo alcanzado, entonces la función global del temporizador se puede caracterizar como:

$$\Delta = \delta - \{P_\delta \cdot \delta_{T_\delta} + P_m \cdot \delta_{T_m}\} \qquad 2.2$$

2.1.1.6. Análisis de complejidad

Para el propósito del análisis, se modela una red MANET como un grafo $G(V, E)$ donde V es el conjunto de los vértices los cuales representan a los nodos en la red, E es el conjunto de las aristas las cuales corresponden a los enlaces físicos entre los nodos. Se asume que los enlaces son bidireccionales y que los costos de los enlaces no son negativos. Además se asume que G es un grafo geométrico aleatorio (RGG) [7] tal que los nodos se colocan uniformemente de manera aleatoria y uniforme unos de otros. Dos nodos están conectados si y solo si, ellos están dentro de un radio r uno del otro. El radio corresponde al radio del rango de los nodos físicos actuales. Se permite que los nodos se unan o dejen la red de manera arbitraria. El valor del radio del umbral de las distancias en el RGG se establece a $R_c = \sqrt{\left(\left(ln(n)\right) + 0(1)/(pi * n)\right)}$ [4] de modo que los nodos estén conectados. Esto da la propiedad [5] donde el diámetro de la red está dado por $\sqrt{(ln(n)/n)}$. Además, se puede aproximar el número de vecinos a distancia de un salto para cualquier nodo dado con la expresión $ln(n)$, donde n es el número de nodos en la red para todas las expresiones anteriores.

Para comprender la complejidad de la carga de trabajo acarreada en el protocolo PROSE, se inicia por identificar la aportación en la carga de señalización para cada mecanismo usado en el protocolo. Se calcula la carga de señalización en términos de mensajes transmitidos a través de toda la red.

Los mensajes *hola* usados en PROSE se envían periódicamente y sobre una base de manejo de evento entre los vecinos ubicados a un salto de distancia. En el modelo presentado en esta sección, se puede representar al número de vecinos ubicados a un salto de distancia como $ln(n)$ y por lo tanto el costo de esta fase, la amplitud de la red es $C_{p1} = \left(f_m(t - t_e) + f_m(t_e)\right)(n * ln(n) * \delta_h)$, donde n es el número de nodos en la red, δ_h es el tamaño del mensaje *hola* intercambiado y f_m es una función del modelo de movilidad elegido. La función f_m describe el número de eventos que provocan cambios en la topología. El primer término representa la frecuencia de las actualizaciones periódicas emitidas. Se debe notar que cada vez que se envía una actualización basada en evento, los temporizadores periódicos se reinician. El segundo componente captura puramente las actualizaciones manejadas por evento.

Cada nodo destino en la red envía un mensaje a su nodo ancla y refresca el mapeo almacenado. El temporizador periódico para esta actualización es mucho más grande que el temporizador en los mensajes de intercambio de vecino a vecino. Sin embargo, este componente, tiene dos casos distintos. En el escenario del peor caso cuando el ancla está situada en el lado diametralmente opuesto de la red, el mensaje viaja esa distancia. Por lo tanto, es una función del diámetro de la red. Un caso más común, es el camino más largo promedio que el mensaje tiene que viajar. Debido a las etiquetas prefijo, esta longitud de camino es menor. En el peor caso, el costo de esta parte de PROSE es:

$$C_{p2} = (n/\gamma)\big(f_m(t' - t'_e) + f_m(t'_e)\big)\big(\sqrt{n/ln(n)}\big) \qquad 2.3$$

donde γ reduce el número de nodos a los involucrados en el subárbol común. γ representa el factor de nodos dentro del subárbol común de nodos parecidos. La expresión en el caso promedio es $C_{p2} = (n/\gamma)\big(f_m(t' - t'_e) + f_m(t'_e)\big)ln_d(n)$, donde d es el grado promedio de la red.

El tercer componente de carga de señalización en PROSE corresponde a las solicitudes de suscripción, el cual involucra un intercambio que se incrementa constantemente basado en las fuentes de tráfico y en cualquier caso incrementa la expresión anterior en un factor pequeño constante $(\gamma + k)$.

Un componente adicional de carga incurre cuando ocurre un reinicio de etiquetas debido a componentes desconectados. Si la tasa de reinicios en la red se define como η entonces este componente se convierte $C_{reset} = (n\eta/\gamma)\big(\sqrt{n/ln(n)}\big)$, y en el peor caso y caso promedio $C_{reset} = (n\eta/\gamma)\big(ln_d(n)\big)$.

Para ver cómo PROSE ahorra en términos de carga de señalización comparado con esquemas de enrutamiento tradicional, ahora se comparan las expresiones anteriores con un protocolo de estado de los enlaces como OLSR, y un protocolo bajo demanda como AODV. El primer componente del total del costo en el caso del estado de los enlaces crece mucho más rápido que en PROSE, debido a que cada evento en cada enlace se propaga n^2 en lugar de $n * ln(n)$ veces. Incluso en un protocolo de estado de los enlaces el segundo componente C_{p2} no existe. Sin embargo, esta compensación es mucho más costosa para actualizaciones en el estado de los enlaces, debido a que la segunda parte de PROSE sólo incrementa la carga de señalización en un factor de $O(n)$. En el caso de los protocolos de enrutamiento bajo demanda, el movimiento constante de los nodos provoca que el segundo término del costo total se dispare. Los protocolos bajo demanda en el peor caso tienen que inundar la red completa y hacerlo para cada evento. Esto significa que la inundación alcanza a $\pi * \big(n/ln(n)\big)$ nodos en lugar de $\sqrt{n/ln(n)}$ nodos. Esto hace que el término C_{p2} sea de orden $O(n^2)$. La Figura 2.3., muestra que tan lento crece la complejidad de señalización en PROSE conforme el número de nodos en la red se incrementa, y es claro que supera la complejidad de los protocolos reactivos.

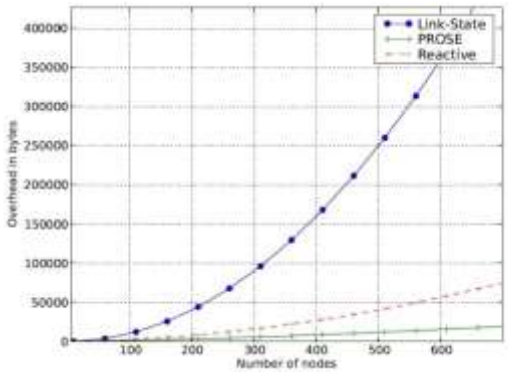


Fig. 2.3. Crecimiento asintótico de la carga de trabajo para diferentes protocolos

2.1.2 Enrutamiento integrado escalable usando etiquetas prefijo y tablas hash distribuidas para MANETs [9]

2.1.2.1 Introducción

El protocolo de enrutamiento con crecimiento automático (AIR), es la primera propuesta para enrutamiento unicast y multicast en MANETs en el cual el enrutamiento es automático; en el que las rutas desde cualquier origen a cualquier destino están implícitas en las etiquetas asignadas a los nodos, y en el que, los nodos retransmisores no necesitan saber la ruta completa hacia el destino.

AIR es la primera propuesta que elimina el método de inundación de cualquier paquete de control, enruta trafico unicast y multicast usando una estructura de enrutamiento integrada, y permite la adición o eliminación tanto de nodos como de enlaces, con impacto limitado para las etiquetas de los nodos y el estado de enrutamiento almacenado localmente. AIR usa señalización de vecino a vecino para etiquetar a cada nodo con una etiqueta prefijo que denota su posición en la MANET relativa al nodo raíz. El nodo raíz se elige con la ayuda de los mismos mensajes de señalización usados en la asignación de etiquetas. Como resultado, la tabla de enrutamiento de un nodo contiene sólo etiquetas prefijo de sus nodos vecinos, más que identificadores de posibles destinos.

El protocolo AIR construye y mantiene una tabla hash distribuida (DHT) que permite a un nodo fuente obtener la etiqueta prefija de algún destino sin utilizar el método de inundación. Los nodos usan una función hash consistente para mapear sus identificadores (los cuales son globalmente únicos) a las etiquetas prefijo de los nodos ancla, los cuales están a cargo de almacenar los mapeos entre un identificador de nodo y su correspondiente etiqueta prefijo. Cada destino publica su presencia en la red al aplicar la función hash a su propio identificador para obtener una etiqueta prefijo, y entonces enviar de manera unicast un mensaje *publish* con el mapeo hacia la etiqueta

prefija. El nodo que mejor coincida con la etiqueta prefijo del objetivo se convierte en el nodo ancla del destino y almacena el mapeo. Una fuente se suscribe a un destino simplemente al aplicar por primera vez la función hash al identificador del destino con la función hash consistente para obtener la etiqueta prefija del ancla, y entonces envía de manera unicast un mensaje *subscribe* hacia el ancla. Para el caso de transmisión multicast, los mensajes *publish* son grupos de solicitudes de unión de los receptores multicast que forman un árbol multicast compartido al forzar a los nodos receptores multicast y el grupo ancla para que se unan al grupo multicast. Una fuente multicast simplemente envía sus paquetes de datos multicast hacia el ancla del grupo objetivo multicast. Estos paquetes son enviados de manera multicast sobre el árbol multicast compartido después de que hayan alcanzado el primer nodo (el cual ya es un miembro del grupo).

2.1.2.2 Enrutamiento con crecimiento automático

El protocolo AIR enruta paquetes de nodos fuente a destinos unicast o multicast por medio de la asignación de etiquetas prefijo a nodos, y el mapeo dinámico de identificadores únicos de nodo (e.g. IP o direcciones MAC) a etiquetas prefijo. La base para la operación del protocolo AIR es el establecimiento distribuido del grafo acíclico dirigido etiquetado (LDAG) enraizado en algún nodo elegido. El nodo raíz es elegido tal que:

a) Cada nodo tiene asignado una etiqueta prefijo denotando la localización relativa del nodo con respecto a la raíz del LDAG,
b) las etiquetas prefijo de un nodo fuente y un destino definen una o múltiples rutas válidas entre dos nodos; y
c) la movilidad de los nodos, las fallas en los nodos o enlaces y la adición de nuevos nodos tienen un impacto limitado en las etiquetas prefijo ya asignadas a los otros nodos.

Este tipo de elección usada por el nodo raíz en el protocolo AIR, es similar a la elección de un nodo raíz en algoritmos de árboles expandidos distribuidos. Los mensajes *Hello* de vecino a vecino se usan para crear y mantener el LDAG. Un nodo anuncia a sus nodos vecinos su propia etiqueta prefijo y las etiquetas prefijo que él asigna a sus hijos en el LDAG, y almacena las etiquetas prefijo que escuchó de todos sus vecinos. Todos los mensajes *hello* transportan un número secuencial para establecer su actualidad. La naturaleza del estado suave de los mensajes *hello* se usa para actualizar las etiquetas de los nodos conforme cambie la topología. Los mensajes *hello* se propagan desde el nodo raíz de manera parecida a la búsqueda por amplitud y se propaga a través de la MANET. El mensaje *hello* enviado por un nodo también enlista a los identificadores y etiquetas prefijo de sus vecinos que se encuentran a un salto de distancia, así como los grupos multicast a los cuales el nodo está suscrito. Después de un tiempo, cada nodo conoce los identificadores y las etiquetas prefijo de sus vecinos que se encuentran a dos saltos de distancia. La Figura 2.4. (a) muestra un ejemplo de un LDAG construido con AIR en una MANET.

Claramente, un nodo fuente debe conocer la etiqueta prefijo del destino al cual quiere enrutar. Para descubrir rutas a nodos destinos, los nodos construyen y mantienen una DHT y publican- suscriben a los mapeos del identificador de nodo y sus etiquetas prefijo en esta DHT. Cada destino usa una función hash consistente (e.g SHA-I) que toma como entrada el identificador de nodo y devuelve una etiqueta prefijo. Si el identificador del nodo corresponde a una dirección multicast, entonces la función has devuelve una etiqueta prefijo de grupo. El nodo de quien la etiqueta coincide más a la etiqueta devuelta por la función hash, se convierte en el ancla de ese destino. Entonces, el destino publica su presencia en la red al enviar su propio mapeo (i.e. su identificador de nodo y su etiqueta prefijo) a su ancla (ver Figura 2.4.). Para encontrar un destino y suscribirse a él, un nodo fuente usa la misma función hash consistente con el identificador del nodo destino como entrada. El nodo fuente envía una solicitud de suscripción al nodo ancla resultante. Los nodos ancla reenvían solicitudes y tráfico de datos directamente a los destinos al buscar los mapeos almacenados localmente.

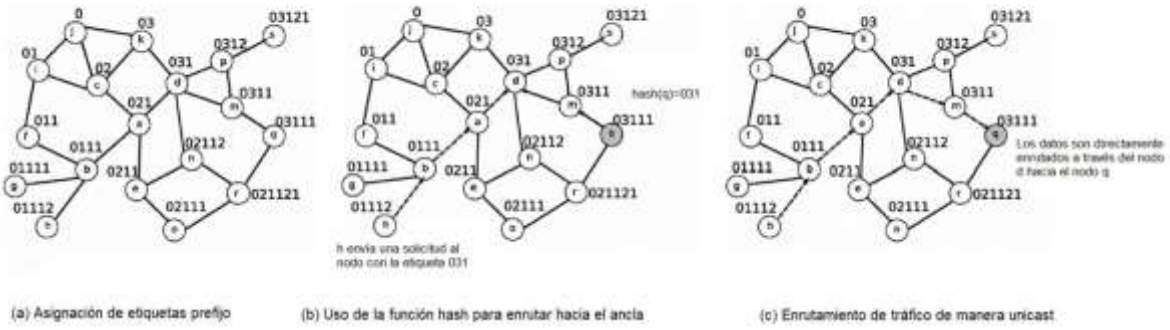


Fig. 2.4. Enrutamiento unicast mediante AIR.

En el caso de enrutamiento multicast, el establecimiento de árboles multicast iniciados por el receptor es muy similar a la manera en que se establecen rutas unicast. La etiqueta prefijo de grupo derivada de un identificador de grupo multicast sirve como el *ancla* de un grupo multicast, el cual toma el rol tradicional del núcleo [10], [11] en un grupo multicast. Para unirse a un grupo multicast, un receptor multicast primero determina si alguno de sus vecinos está en el grupo, si es así, envía una solicitud de unión a éste; de otra manera, el receptor simplemente aplica la función hash al identificador del grupo multicast para obtener la etiqueta prefija del grupo. Se envía la solicitud de unión hacia el nodo con la correspondencia más cercana a la etiqueta obtenida con la función hash anterior, el cual sirve como nodo ancla del grupo. Una solicitud de unión se contesta con una respuesta de unión por cualquier nodo que ya es parte del árbol compartido y se activa un camino inverso para que la respuesta de ruta sea reenviada y los nodos retransmisores se vuelvan parte del árbol multicast compartido para el grupo.

2.1.2.3. Detalles del protocolo AIR

2.1.2.3.1. Información almacenada e intercambiada

Cada nodo mantiene una *tabla vecino* (NT) y una *tabla vecindario a dos saltos* (TNT). La NT de un nodo *i* contiene un registro para cada vecino inmediato del nodo *i*, y en cada registro se establece el identificador y la etiqueta prefija de un vecino; el número secuencial más reciente recibido de un vecino, y una lista de grupos multicast a los cuales pertenece el nodo, con cada registro de la lista se indica si el nodo *i* fue seleccionado por su vecino para llegar al núcleo del grupo. La TNT del nodo *i* contiene un registro para cada vecino ubicado a dos saltos de distancia del nodo *i*, y cada entrada registra el identificador y etiqueta prefijo del nodo vecino (ubicado a dos saltos). Además, cada nodo mantiene una *tabla de grupo multicast*, que lista información para cada grupo multicast al cual el nodo vecino pertenece; la información almacenada para cada grupo incluye: la etiqueta prefijo e el identificador del grupo, número secuencial del grupo, y el siguiente salto hacia el núcleo del grupo.

Además, cada nodo mantiene una *caché de paquete* la cual lista información acerca de paquetes de datos multicast escuchados recientemente, y una caché de señalización que almacena y totaliza la información actualizada que será enviada en el siguiente paquete *Hello*.

El protocolo AIR usa un único tipo de mensaje de control llamado *Hello* para llevar a cabo la señalización. El paquete H*ello* enviado por un nodo *a* contiene la siguiente información:

a) Un identificador del LDAG que consiste del identificador de nodo raíz del LDAG y el último número secuencial creado por el nodo raíz,
b) la etiqueta prefijo e identificador del nodo,
c) el número secuencial creado por el nodo emisor,
d) la lista de los identificadores de nodo y etiquetas prefijo de los vecinos que se encuentran a distancia uno y dos saltos, y
e) una lista de información de grupo (GIL).

La GIL en un paquete *hello* consiste de una lista de uno o más registros que describen el estado del nodo emisor con respecto a un grupo multicast. Cada registro en la GIL especifica:

a. el identificador y etiqueta prefijo de grupo del grupo multicast
b. la acción que se está tomando para el grupo (solicitud de unión, respuesta de unión, solicitud de abandono, o miembro);
c. el identificador del nodo y etiqueta prefija del siguiente salto seleccionado para alcanzar el núcleo del grupo; y
d. el identificador de uno o más nodos para el cual la respuesta de unión está dirigida.

2.1.2.3.2. Estructura de las etiquetas prefijo

Sea Σ el alfabeto que contiene un número finito de símbolos y Σ^* el conjunto de todas las cadenas sobre Σ tal que $|\Sigma| \geq 2$. Cada nodo etiqueta sus enlaces hacia cada uno de sus vecinos con una letra ω de Σ. Si ω_i representa la letra asignada al i^{th} enlace de cualquier nodo entonces, $\omega_i \mapsto \{\omega_i \in \Sigma | \omega_i \neq \omega_{i+1} \forall_i \leq d-1\}$, donde d es el grado del nodo. Por lo tanto, se asigna una unica letra a cada enlace que conecta a algún nodo con sus vecinos. La lógica etiquetadora etiqueta a cada nodo en el LDAG de una manera por amplitud. Dado que el LDAG se puede organizar como un árbol $k-ary$, siendo k el grado del LDAG, a cada hijo (arriba de k) se asigna una etiqueta prefijo Λ como se define a continuación:

Etiqueta prefijo: Una etiqueta prefijo Λ para el nodo y es una palabra dentro de Σ^* tal que $\Lambda = \Lambda_{parent} \odot l'$, donde Λ_{parent} es la etiqueta prefijo obtenida del padre, ⊙ es el operador de concatenación y Λ_{parent} es concatenada con un único sufijo sobre k diferentes opciones de Σ para formar Λ.

Se deduce de la definición anterior que la etiqueta prefija de un nodo, Λ identifica únicamente al nodo en un LDAG dado.

Las etiquetas prefijas de los nodos definen una relación *predecesor* en el LDAG, denotada por ↩, tal que para dos nodos cualesquiera *s* y *d* en el LDAG:

5) $s \hookleftarrow d$: $s\ precede\ a\ d$. En este caso, *d* puede ser alcanzado al recorrer un camino prefijo de descendientes de *s* en la porción del LDAG con raíz en *s*.
6) $d \hookleftarrow s$: $d\ precede\ a\ s$. En este caso, *d* puede ser alcanzado flujo arriba de *s* al recorrer un camino prefijo de antecesores de *s*.
7) $s \approx d$: d y s No están directamente relacionados. En este caso, *d* puede ser alcanzado al recorrer el LDAG de *s* hacia arriba hasta el primer antecesor común de *d* y *s*, supongamos *A*, tal que se mantiene $A \hookleftarrow s$ y $A \hookleftarrow d$, y entonces se recorre hacia bajo de *A* para llegar a *d*.

La topología de una MANET cambia constantemente y los nodos se unen o abandonan la red arbitrariamente, lo cual resulta en etiquetas inconsistentes mientras los nodos las están actualizando. El protocolo AIR fuerza un etiquetado estricto al ordenar etiquetas prefijas usando números secuenciales. Sea *L* que representa una tupla (S_n^a, Λ_n^a), donde S_n^a denota el número secuencial que se origina en *a* y es reenviado por el nodo *n*, y Λ_n^a denota la etiqueta prefijo del nodo actual con respecto a *a*. *S* es un entero que se incrementa de manera monótona, mientras Λ es una palabra en Σ^*. Se define un operador ≺ sobre el conjunto del par ordenado de indentificadores en *L*. Si L_x^a, L_y^a son dos tuplas entonces,

$$\{L_x^a \prec L_y^a | L_{x,y}^a \in \Sigma\} \quad 2.4$$
$$if\{(S_x^a < S_y^a) \vee [(S_x^a = S_y^a) \wedge (\Lambda_x^k \succ \Lambda_y^i)]\}$$

Se puede ver que el operador $\prec$ es antireflexivo y transitivo sobre el conjunto Σ^*, y que estas etiquetas prefijas pueden ser establecidas en orden entre los nodos.

2.1.2.3.3. Elección de la raíz y asignación de etiquetas prefijo

Se elige a un nodo raíz de manera distribuida usando mensajes *Hello*. Las ataduras entre dos nodos se rompen por sus identificadores de nodo, por ejemplo. Si un nodo no está etiquetado, busca en su tabla de vecinos para determinar si algún nodo está etiquetado. Si no hay alguno etiquetado y su temporizador local para etiquetado expira, el nodo se elige a él mismo como nodo raíz.

El nodo raíz auto elegido posteriormente, envía mensajes *Hello* con su propia etiqueta y asigna etiquetas a los nodos del vecindario.

El nodo con el valor de identificador más bajo impone el ordenamiento entre los nodos en el LDAG. En el trabajo presentado por simplicidad, sólo se consideran etiquetas prefijas asignadas por un único nodo raíz.

2.1.2.3.4. Enrutamiento en el protocolo AIR

Para enrutar hacia un destino *d*, un nodo *s* escoge el enlace a cualquiera de sus vecinos ubicados a dos saltos de distancia que ofrezca la longitud máxima de la etiqueta prefijo la cual coincida con la etiqueta prefijo del destino. Esto es simplemente la lógica que usa el prefijo con la máxima coincidencia, la cual selecciona el siguiente salto usando la estrategia voraz que considera el dos vecindario de un nodo, y puede encontrar caminos más cortos que el enrutamiento tradicional de árbol prefijo, al provocar el enriquecimiento de la diversidad en caminos de un LDAG comparada con un árbol prefijo. Por ejemplo, si existe una etiqueta en el dos vecindario que está lexicográficamente más cercano al destino, se escoge a ese nodo tal que el paquete es renviado a él en lugar de enrutar por medio del padre del árbol prefijo. Se asegura que está técnica no encuentra un mínimo local al seleccionar aleatoriamente el siguiente salto cuando todos los nodos ofrecen el mismo nivel de coincidencia en el prefijo.

2.1.2.3.5. Construir la DHT (Tabla Hash Distribuida)

Para evitar cuellos de botella y puntos únicos de fallas, los mapeos de identificadores de nodos a etiquetas prefijo se distribuyen dentro de la red. Los nodos publican o se suscriben a destinos unicast y multicast usando el mismo LDAG.

1) *Publicación de destinos:* Cada nodo publica el mapeo entre su identificador de nodo y su etiqueta prefijo al nodo ancla. El nodo con la etiqueta prefijo que tenga la mayor

coincidencia con la etiqueta prefijo publicada en la solicitud, se convierte en el ancla designada para el mapeo, y esa información se almacena. La frecuencia de los mensajes de actualización es controlada por los cambios en la topología del vecindario de un nodo. Cada nodo mantiene también, una lista de etiquetas prefijas asignadas a él y su padre correspondiente. Si el nodo detecta un cambio en su etiqueta prefija, entonces envía una actualización a su ancla.

2) *Suscripción a destinos:* Las fuentes unicast activas se suscriben a destinos unicast al aplicar la función hash primero al identificador de nodo para obtener la etiqueta del ancla. La solicitud de suscripción unicast (comúnmente el primer paquete de datos) recorre el camino correspondiente al nodo ancla, el cual puede reenviar hacia el destino. Los receptores multicast se suscriben a grupos multicast de la misma manera mostrada anteriormente. Sin embargo, las suscripciones multicast hechas por los receptores son resueltas por el primer miembro de grupo multicast, que fue quien recibió la solicitud de unión. El ancla de un grupo multicast es aquel nodo que su etiqueta prefija tenga la mayor coincidencia con la etiqueta prefija del grupo declarada en una solicitud de suscripción para el grupo. Las fuentes enrutan hacia el ancla de un destino unicast o multicast, escogiendo el siguiente salto el cual su etiqueta prefijo coincida más con la etiqueta prefijo del ancla.

2.1.2.3.6. Temporizadores adaptables

Los mensajes *Hello* se transmiten periódicamente por un nodo a todos sus vecinos inmediatos. La frecuencia de los mensajes *Hello* se determina por la naturaleza de las actualizaciones que tienen que llevarse a cabo. Un nodo mantiene dos valores de expiración; un tiempo de expiración largo (LT) y un tiempo de expiración corto (ST). Las actualizaciones se agregan a la información del vecindario o GIL hasta que el tiempo de expiración termina. Cualquier evento que cambie el salto siguiente hacia el núcleo de un grupo (para el cual el nodo está activo), activa el tiempo de expiración corto. Cuando el ST termina, el nodo envía su mensaje *Hello* con las actualizaciones agregadas almacenadas en su caché de señalización, y el nodo lleva a cabo los pasos necesarios para mantener arboles multicast. El tiempo de expiración largo se activa cuando la topología es relativamente estática y se necesitan pocos mensajes para mantener los arboles multicast de varios grupos. Estos temporizadores aseguran que la frecuencia con la que un nodo transmite sus mensajes *Hello* sea en función de los cambios de la topología alrededor del nodo.

2.1.2.3.7. Nodos dinámicos

Los nodos adquieren nuevas etiquetas cuando se reponen ellos mismos en la red. Mientras los cambios a los nodos que se unen por el marco son triviales, los cambios en las etiquetas prefijas de nodos internos son más elaborados. Cuando un nodo intenta encontrar caminos hacia sus pares en el evento que un padre se mueve o falla. Si él encuentra un camino válido a todos sus pares,

entonces emulan al nodo padre hasta que se termine un temporizador de reetiquetamiento. Esto previene la eliminación de flujos de paquetes existentes. Además, cada nodo mantiene su etiqueta vieja por un periodo de tiempo completo después del reetiquetamiento para evitar ciclos o eliminaciones de paquetes.
Los cambios a etiquetas de las anclas o núcleos no afectan la topología, ya que el nodo que tenga tanto la etiqueta prefijo más cercana a la etiqueta prefijo del ancla o etiqueta prefijo de grupo de la anterior ancla, automáticamente se asigna a él mismo como el ancla o núcleo.

2.1.2.3.8. Grupos dinámicos

La membresía de grupo se mantiene usando simple lógica de estado suave a través del intercambio periódico de mensajes *Hello.* Aunque cada mensaje *Hello* especifica información de control en la GIL para uno o múltiples grupos multicast, los pasos llevados a cabo por un grupo multicast son independientes de otros grupos.

Un receptor interesado en unirse a un grupo multicast particular aplica la función hash al nombre del grupo para obtener la correspondiente etiqueta prefijo. Entonces, envía mensajes de solicitud de unión al grupo, en la GIL de su mensaje *Hello* establece el identificador y etiqueta prefijo del nodo vecino que elige junto con una de sus rutas implícitas a la etiqueta prefijo del grupo. El nodo núcleo de un grupo multicast G recibe una solicitud de unión para ese grupo en el mensaje *Hello* de un vecino *x*. Entonces, envía un mensaje con el registro de respuesta de unión guardada en la GIL para el grupo G y establece el identificador del nodo *x* como el receptor de la respuesta. Un nodo que es miembro del grupo multicast lleva a cabo la misma acción cuando recibe una solicitud de unión para ese grupo dentro del mensaje *Hello* de un vecino, estableciendo que el nodo es el siguiente salto al núcleo del grupo. El núcleo de un grupo G se convierte en un miembro del grupo e inicia el envío de un registro de “miembro” para el grupo G en la GIL de sus mensajes *Hello* después de recibir la primera solicitud de unión para el grupo.

Si el nodo *x* no es un miembro del grupo G y recibe una solicitud de unión de un vecino estableciendo que él es el nodo siguiente al núcleo del grupo G, entonces el nodo *x* envía su propia solicitud de unión establecida en la GIL de su siguiente mensaje *Hello*. Su solicitud de unión establece su opción de siguiente salto al núcleo de G junto con una de sus rutas implícitas al núcleo.

Un registro “miembro” para el grupo G en la GIL de un mensaje *Hello* de un nodo, sirve como una actualización del estado multicast para todos sus vecinos. Además, ayuda a nodos a decidir cuándo reenviar paquetes de datos multicast. Un receptor multicast que no sirve como reenviador y elige abandonar el grupo simplemente deja de incluir el registro para el grupo G en sus mensajes *Hello*.

2.1.3 PASTRY: enrutamiento y localización de objetos descentralizado escalable para sistemas a gran escala punto a punto [12]

2.1.3.1 Introducción

Las aplicaciones de internet punto a punto se han popularizado recientemente a través de aplicaciones de compartición de archivos como Napster, Gnutella y FreeNet. Los sistemas punto a punto tienen muchos aspectos técnicos interesantes como: control descentralizado, auto organización, adaptabilidad y escalabilidad. Estos sistemas se pueden caracterizar como sistemas distribuidos en los cuales todos los nodos tienen capacidades y responsabilidades idénticas, y toda la comunicación es simétrica.

Uno de los problemas clave en aplicaciones punto a punto a gran escala, es proveer algoritmos eficientes para localización de objetos y enrutamiento dentro de la red.
Pastry es un modelo de enrutamiento y localización de objetos genérico punto a punto. Además es completamente descentralizado, tolerante a fallas, escalable y confiable.
Pastry está pensado como una subcapa general para la construcción de una variedad de aplicaciones de internet punto a punto, tales como compartición global de archivos, almacenamiento de archivos, comunicación grupal y sistema de nombres.

En una red del tipo Pastry, cada nodo tiene un identificador numérico único (*nodeId*). Cuando se presenta con un mensaje y una clave numérica, un nodo Pastry enruta eficientemente hacia el nodo con el *nodeId* que sea el más cercano numéricamente a la clave, entre todos los nodos Pastry activos. El número de pasos en el enrutamiento es $O(logN)$, donde N es el número de nodos Pastry en la red.

Pastry toma en cuenta la localización en la red; la cual busca minimizar la distancia del viaje del mensaje, de acuerdo a la métrica de proximidad escalar, como el número de saltos de enrutamiento IP. Cada nodo Pastry almacena a sus vecinos inmediatos y, notifica a la aplicación de la llegada de nuevos nodos, fallas y recuperaciones.

2.1.3.2 Diseño de PASTRY

Un sistema Pastry es un red superpuesta de nodos auto organizada, donde cada nodo enruta solicitudes de clientes e interactúa con instancias locales de una o más aplicaciones.
A cada nodo en la red Pastry se le asigna un identificador de nodo (*nodeId*) de 128-bits. El *nodeId* se usa para indicar la posición del nodo en un espacio de identificadores de nodos circular, el cual tiene como rango de 0 a $2^{128} - 1$. El nodeId se asigna aleatoriamente cuando un nodo se une al sistema.

Se asume que los identificadores de nodos son generados tal que, el conjunto resultante es uniformemente distribuido en el espacio de identificadores. Por ejemplo, los *nodeIds* se podrían generar al calcular una función hash criptográfica a la llave pública del nodo o a su dirección IP.

Pastry puede enrutar hacia el nodo numéricamente más cercano a una clave dada en menos de $\lceil \log_{2^b} N \rceil$ pasos bajo operación normal (b es un parámetro de configuración con un valor típico de 4).

Para el propósito del enrutamiento, los *nodeIds* y claves son pensados como una secuencia de dígitos de base 2^b. Pastry enruta mensajes al nodo de quien su nodeId sea el más cercano numéricamente a la clave dada. Esto se consigue mediante el procedimiento siguiente. En cada paso de enrutamiento, un nodo normalmente reenvía el mensaje hacia un nodo de quien su *nodeId* comparte con la clave un prefijo que es al menos un digito (o b bits) más grande que el prefijo que la clave comparte con el id del nodo actual. Si no se conoce a tal nodo, el mensaje es reenviado hacia un nodo de quien su *nodeId* comparte un prefijo con la clave tanto como el nodo actual, pero es numéricamente más cercano a la clave que el id del nodo actual. Para respaldar este procedimiento se describe el estado de enrutamiento que cada nodo mantiene en una red Pastry.

2.1.3.2.1 Estado del nodo Pastry

Cada nodo Pastry mantiene una *tabla de enrutamiento*, un *conjunto de vecindario* y un *conjunto hoja*. La *tabla de enrutamiento* de un nodo, R, se organiza en $\lceil \log_{2^b} N \rceil$ renglones con $2^b - 1$ entradas cada uno. Las $2^b - 1$ entradas en cada renglón n de la tabla de enrutamiento se refiere a que un nodo de quien su nodeId comparte el nodeId con el nodo actual en los primeros n digitos, pero de quien su $n + 1$ésimo digito tiene uno de los $2^b - 1$ valores posibles más que el $n + 1$ésimo digito en el id del nodo actual.

El *conjunto vecindario* M contiene *nodeIds* y direcciones IP de los $|M|$nodos más cercanos (de acuerdo a la métrica de proximidad) al nodo actual. El conjunto vecindario no se usa normalmente en el enrutamiento de mensajes; es útil para mantener propiedades de localización. El conjunto hoja L es el conjunto de nodos con los $|L|/2$ numéricamente más cercanos nodeIds más largos, y los $|L|/2$ nodos con los numéricamente más cercanos nodeIds más pequeños, relativos al nodeId del nodo actual.

2.1.3.2.2 Enrutamiento

El proceso de enrutamiento se ejecuta siempre que un mensaje con clave D llega a algún nodo con nodeId A.

Dado un mensaje, el nodo primero que la clave caiga dentro del rango de los *nodeIds* cubiertos por su conjunto hoja. Si es así, el mensaje se reenvía directamente hacia el nodo destino, específicamente el nodo en el conjunto hoja de quien su *nodeId* es más cercano a la clave

(posiblemente el nodo actual).Si la clave no está cubierta por el conjunto hoja, entonces se usa la tabla de enrutamiento y se reenvía el mensaje hacia un nodo que comparta un prefijo común con la clave en al menos uno o más dígitos. En ciertos casos, es posible que la entrada apropiada en la tabla de enrutamiento esté vacía o que el nodo asociado no sea alcanzable, en tal caso se reenvía el mensaje hacia un nodo que comparta un prefijo con la clave en al menos la misma cantidad de dígitos que el nodo actual, y que sea numéricamente más cercano a la clave que el id del nodo actual.

Tal nodo debe estar en el conjunto hoja a menos que el mensaje ya haya llegado al nodo con el *nodeId* numéricamente más cercano. Y, a menos que los $|L|/2$ nodos adyacentes en el conjunto hoja hayan fallado simultáneamente, al menos uno de estos nodos debe estar activo.

2.1.3.2.3 Auto organización y adaptabilidad

Adición de un nodo. Cuando se agrega un nodo, necesita inicializar el estado de sus tablas, y entonces informar a otros nodos de su presencia. Se asume que el nuevo nodo inicialmente conoce a un nodo Pastry cercano A, de acuerdo a la métrica de proximidad, que ya es parte del sistema. Tal nodo puede ser localizado automáticamente, por ejemplo, usando "expansión de anillo" IP multicast, o ser obtenido por el administrador del sistema a través de canales externos.

Se asume que el *nodeId* del nuevo nodo es X, entonces solicita al nodo A enrutar un mensaje especial de "unión" con la clave igual a X. Como cualquier mensaje Pastry enruta el mensaje de unión al nodo existente Z de quien si id es numéricamente cercano a X.

En respuesta a la solicitud de unión, los nodos A, Z, y todos los nodos encontrados en el camino de A a Z envían sus estados de tablas a X. El nuevo nodo X inspecciona esta información y podría solicitar estados de nodos adicionales, y entonces inicializa el estado de sus tablas. Finalmente, X informa a cualquier nodo que necesite estar consciente de su llegada.

Ya que se asume que el nodo A está próximo a l nuevo nodo X, el conjunto vecindario de A es el conjunto vecindario de X. Además, Z tiene el nodeId más cercano a X, por lo tanto su conjunto hoja es la base para el conjunto hoja de X.

Finalmente, X transmite una copia de su estado resultante a cada uno de los nodos encontrados en su conjunto vecindario, conjunto hoja, y tabla de enrutamiento. El costo total para la unión de un nodo, en términos de número de mensajes intercambiados, es $O\left(\log_{2^b} N\right)$.

Partida de un nodo. Los nodos en una red Pastry pueden fallar o dejar la red sin avisar. Para reemplazar un nodo que falló en el conjunto hoja, su vecino en el espacio *nodeId* contacta al nodo activo con el índice más largo en el lado del nodo que falló, y entonces pregunta por su tabla hoja a tal nodo. Por ejemplo, si L_i falló para $\lfloor |L|/2 \rfloor < i < 0$, se solicita el conjunto hoja de $L_{-\lfloor |L|/2 \rfloor}$. Sea el conjunto hoja recibido L'. Este conjunto se traslapa con el conjunto hoja L del nodo actual y

contiene nodos con ids cercanos no presentes en L. Entre estos nodos nuevos, se elige al apropiado para insertarlo en L, verificando que el nodo esté activo al contactarlo.

La falla de un nodo que aparece en un la tabla de enrutamiento de otro nodo se detecta cuando ese nodo intenta contactar al nodo que falló y no responde.

Para reparar una entrada fallida en una tabla de enrutamiento R_l^d, un nodo primero contacta al nodo referido a través de otra entrada R_l^i, $i \neq d$ del mismo renglón, y pregunta por la entrada de ese nodo para R_l^d.

2.1.3.2.4 Localización

La proximidad de red en la noción de Pastry está basada en una métrica de proximidad escalar, como el número de saltos de enrutamiento IP o distancia geográfica. Se asume que la aplicación provee una función que permite a cada nodo Pastry determinar la distancia de un nodo con una dirección IP dada. Se asume que el espacio de proximidad definido por la métrica de proximidad elegida es Euclidiano.

Localización en la tabla de enrutamiento. La propiedad que se desea mantener es que todas las entradas de las tablas de enrutamiento se refieran a un nodo que sea cercano al nodo actual, de acuerdo a la métrica de proximidad, entre todos los nodos activos con un prefijo apropiado para la entrada.

Localización de ruta. Las entradas en la tabla de enrutamiento de cada nodo Pastry se eligen para estar cerca del nodo actual, de acuerdo a la métrica de proximidad, entre todos los nodos con el *nodeId* prefijo deseado. Como resultado, en cada paso de enrutamiento, se reenvía un mensaje a un nodo relativamente cerca al *nodeId* que comparte un prefijo común más grande o es numéricamente más cercano a la clave más que el nodo actual. Es decir, en cada paso el mensaje se mueve más cerca hacia el destino en el espacio de los *nodeIds*, mientras viaja la distancia menos posible en el espacio de proximidad.

Ya que sólo se usa información local, Pastry minimiza la distancia del siguiente paso de enrutamiento sin detectar la dirección global. Este procedimiento claramente no garantiza que se elija al camino más corto entre origen y destino.

Localizar el más cercano entre k nodos. Pastry naturalmente enruta un mensaje con la clave dada hacia el nodo activo con el *nodeId* más cercano numéricamente, y así asegura que el mensaje alcanza uno de los *k* nodos siempre y cuando uno de ellos esté activo.
Pastry usa una heurística para tratar el problema de prefijos disparejos (cuando se podrían omitir nodos con un prefijo diferente al de la clave). La heurística está basada en estimar la densidad de los *nodeIds* en el espacio de los *nodeIds* usando información local. Basada en esta estimación, la heurística detecta cuándo un mensaje se aproxima al conjunto de los *k* nodos numéricamente más

cercanos, y entonces cambia a direcciones numéricamente más cercanas para localizar a la réplica más cercana.

2.1.3.2.5 Fallas arbitrarias de nodos

Una red Pastry puede tener fallas en nodos, donde un nodo con fallas podría continuar funcionando, pero comportándose incorrectamente o incluso maliciosamente. En el momento en que se presente una falla o un comportamiento malicioso en un algún nodo, la solicitud que haga podría ser repetida varias veces por el cliente, hasta que se elija una ruta para evitar el nodo malo. Este es el mecanismo implícito que se tiene en este protocolo, ya que las fallas en los nodos o nodos maliciosos no se tratan de manera directa.

2.1.4. Tapestry: una capa adaptable de escala global para el desarrollo de servicios [13]

2.1.4.1. Introducción

Tapestry es una capa de red punto a punto que provee alto rendimiento, escalable, y enrutamiento de mensajes independiente de su localización hacia puntos finales cercanos, usando sólo recursos localizados. El enfoque de enrutamiento trae con él, el deseo de eficiencia: minimizar la latencia de los mensajes y maximizar el rendimiento de los mensajes. De este modo, Tapestry explota la localización en el enrutamiento de mensajes para destinos móviles tales como réplicas de objetos.

Tapestry usa algoritmos adaptables con estado suave para mantener la tolerancia a fallas para enfrentar el cambio de membresía de nodos y fallas en la red. Su arquitectura es modular. Esta Interfaz de Programación de Aplicaciones (IAP) permite a los desarrolladores trabajar y extender la funcionalidad de la capa cuando la funcionalidad básica es insuficiente.

2.1.4.2. Algoritmos de TAPESTRY

2.1.4.2.1. El API de DOLR (Decentralized Object Location and Routing)

Tapestry provee una interfaz de comunicaciones de tipo datagrama, con mecanismos adicionales para manipular las localizaciones de los objetos.

Los nodos en Tapestry participan en el sistema y se les asigna *nodeIDs* aleatoriamente uniforme de un espacio de identificadores grande. Se puede alojar a más de un nodo por un nodo físico. A los

puntos finales específicos de la aplicación se le asignan *Identificadores Únicos Globales* (GUIDs, Globally Unique Identifiers), seleccionados del mismo espacio de identificadores. Tapestry actualmente usa un espacio de identificadores de valores de 160-bits con un radio definido globalmente (e.g., hexadecimal, el cual produce identificadores de 40-digitos). Tapestry asume que los *nodeIDs* y GUIDs son aproximadamente equitativamente distribuidos en el espacio de nombres, el cual se puede obtener al usar un algoritmo que implemente una función hash como SHA-1 [15]. Se dice que un nodo N tiene un *nodeID* N_{id}, y un objeto O tiene un GUID O_G.
Ya que la eficiencia del sistema Tapestry generalmente mejora con el tamaño de la red, es una ventaja para múltiples aplicaciones compartir una única red superpuesta Tapestry grande.

Para habilitar la coexistencia de la aplicación, cada mensaje contiene un identificador específico de aplicación, A_{id}, el cual se usa para seleccionar un proceso, o aplicación para la entrega de un mensaje en el destino (similar al rol de un puerto en TCP/IP).

Dadas las definiciones anteriores, se establecen las cuatro partes de la API de red de DOLR de la siguiente manera:

1. PUBLISHOBJECT(O_G, A_{id}): Se publica o pone a disposición al objeto O en el nodo local. Esta llamada es de mejor esfuerzo, y no recibe confirmación.
2. UNPUBLISHOBJECT(O_G, A_{id}): Intenta por mejor esfuerzo remover la localización de mapeos para O.
3. ROUTETOOBJECT(O_G, A_{id}): Enruta mensajes hacia la localización de un objeto con GUID O_G.
4. ROUTETONODE($N, A_{id}, Exact$): Enruta mensajes hacia la aplicación A_{id} del nodo N. "Exact" especifica si el ID del destino necesita ser emparejado exactamente para entregar los datos.

2.1.4.2.2. Enrutamiento y localización de objetos

Tapestry mapea dinámicamente cada identificador G hacia un único nodo activo, llamado la *raíz* del identificador o G_R. Si existe un nodo N con $N_{id} = G$, entonces este nodo es la raíz de G. Para entregar mensajes, cada nodo mantiene una tabla de enrutamiento que consiste de nodeIDs y direcciones IP de los nodos con los cuales se comunica (vecinos del nodo). Cuando se enruta hacia G_R, los mensajes se reenvían a través de los enlaces de los vecinos de quien sus IDs son progresivamente más cercanos (i.e., prefijos parecidos más grandes) a G en el espacio de los IDs.

1) Malla de enrutamiento: Tapestry usa tablas locales en cada nodo, llamadas *mapas de vecino*, para enrutar mensajes de la capa hacia el ID destino digito por digito (e.g., 4***⟹42**⟹42A⟹42AD, donde los *s representan comodines). Esta propuesta es similar al enrutamiento del prefijo más largo usado por la asignación de direcciones IP CIDR [16]. Un nodo N tiene un mapa de vecindario con múltiples niveles, donde cada nivel contiene enlaces a nodos que coinciden con un digito en una posición del prefijo del ID, y contiene un número de entradas igual a la base del ID. La i-ésima entrada primaria en el j-ésimo nivel es el ID y la localización del nodo más cercano que inicia con el prefijo (N, $j-1$)+"i" (e.g., la novena entrada para el cuarto nivel para el nodo $325AE$ es el nodo

más cercano con un ID que inicie con 3259). Esta es la propuesta del "nodo más cercano" que provee las propiedades de localización de Tapestry. La figura 2.5 muestra algunos de los enlaces salientes de un nodo.

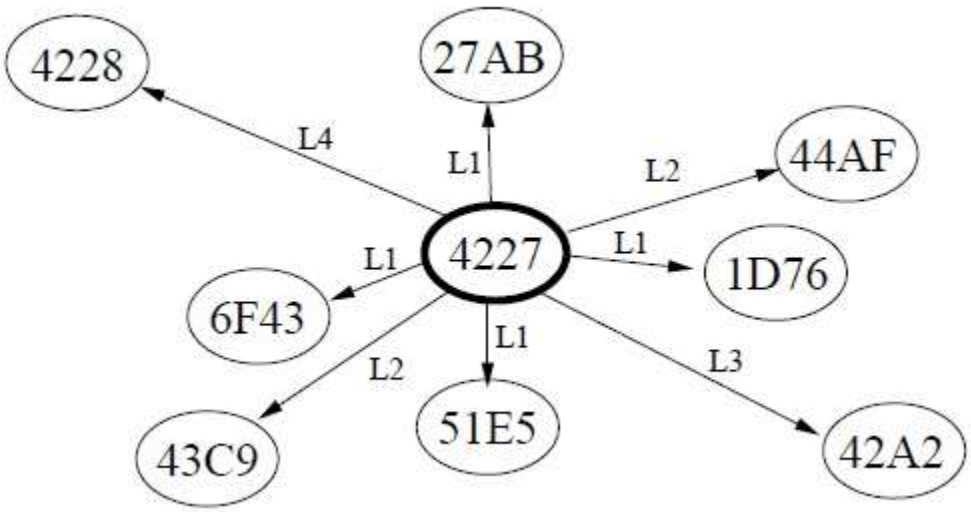


Figura 2.5. *Malla de enrutamiento de Pastry desde la perspectiva de un único nodo.* Los enlaces salientes del vecino apuntan hacia nodos con un prefijo emparejado común. Las entradas para niveles más altos coinciden con más dígitos. Todos estos enlaces forman la tabla de enrutamiento local.

La figura 2.6. Muestra el camino que un mensaje debe tomar a través de la infraestructura. El enrutador para el $n-ésimo$ salto comparte un prefijo de longitud $\geq n$ con el ID del destino; así, para enrutar, Tapestry busca en su $(n+1)ésimo$ nivel del mapa para la entrada que coincida con el siguiente digito en ID del destino. Este método garantiza que cualquier nodo existente en el sistema sea alcanzado en a lo más $log_{\beta}N$ pasos lógicos, en un sistema con un espacio de nombres de tamaño N, IDs de base β, y sumiendo mapas de vecindario consistentes. Cuando un digito no puede ser emparejado, Tapestry busca a un digito "cercano" en la tabla de enrutamiento; a este mecanismo se le llama *enrutamiento suplente (surrogate routing)* [14], donde cada ID no existente es mapeado a algún nodo activo con un ID similar.

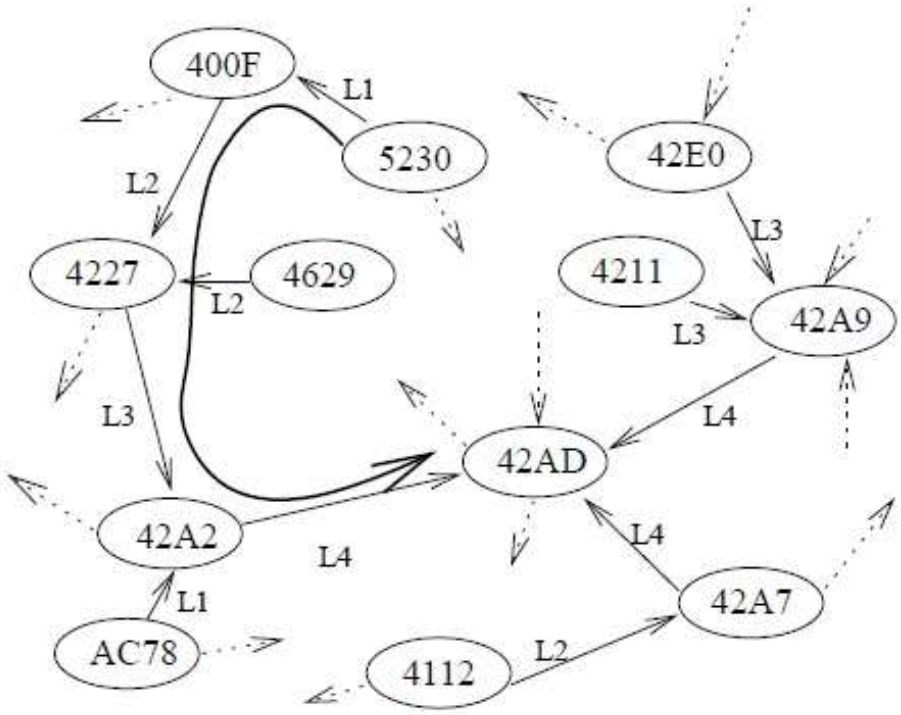


Figura 2.6. *Camino de un mensaje.* El camino tomado por un mensaje originado en el nodo 5320 destinado al nodo $42AD$ en una malla Tapestry.

El reto en el ambiente de una red dinámica es continuar enrutando confiablemente incluso cuando los enlaces intermedios están cambiando o fallan. Para ayudar a proveer adaptabilidad, se explota la diversidad de los caminos de la red en forma de caminos de enrutamiento redundantes. Los enlaces primarios de los vecinos mostrados en la Figura 2.5 son aumentados por enlaces de respaldo, que comparten el mismo prefijo. En el *n*-ésimo nivel de enrutamiento, los c enlaces de vecinos difieren sólo en el *n*-ésimo digito. Existen $c \times \beta$ apuntadores en un nivel, y el tamaño total del mapa del vecindario es $c \times \beta \times log_\beta N$. Cada nodo además almacena referencias de regreso hacia otros nodos que apuntan a él. El número total de tales entradas es $c \times \beta \times log_\beta N$.

2) Localización y publicación de objeto: Como se mostró anteriormente, cada identificador G tiene una raíz de nodo única G_R asignada por el proceso de enrutamiento. Cada raíz de nodo hereda un árbol de expansión único para enrutar con mensajes desde nodos hojas, atravesando nodos intermedios para enrutar hacia la raíz. Se utiliza esta propiedad para asignar objetos, al distribuir información de directorio de estado suave a través de los nodos (incluyendo la raíz del objeto).

Un servidor S, que almacena un onjecto O (con GUID, O_G, y la raíz, O_R), periódicamente anuncia o *publica* este objeto al enrutar un mensaje de publicación hacia O_R(ver Figura 2.7.). En general, el nodeID de O_R es diferente de O_G; $O_R$es el *único* nodo [2] alcanzado a través de enrutamiento suplente al hacer llamadas sucesivas a NEXTHOP$(^*, O_G)$. Cada nodo a lo largo del camino de publicación almacena un apuntador de mapeo, $< O_G, S >$, en lugar de una copia del objeto. Cuando existen réplicas de un objeto en servidores separados, cada servidor publica su copia. Los nodos

Tapestry almacenan la localización de mapeos para réplicas de objetos de acuerdo al orden de latencia de red a partir de ellos.

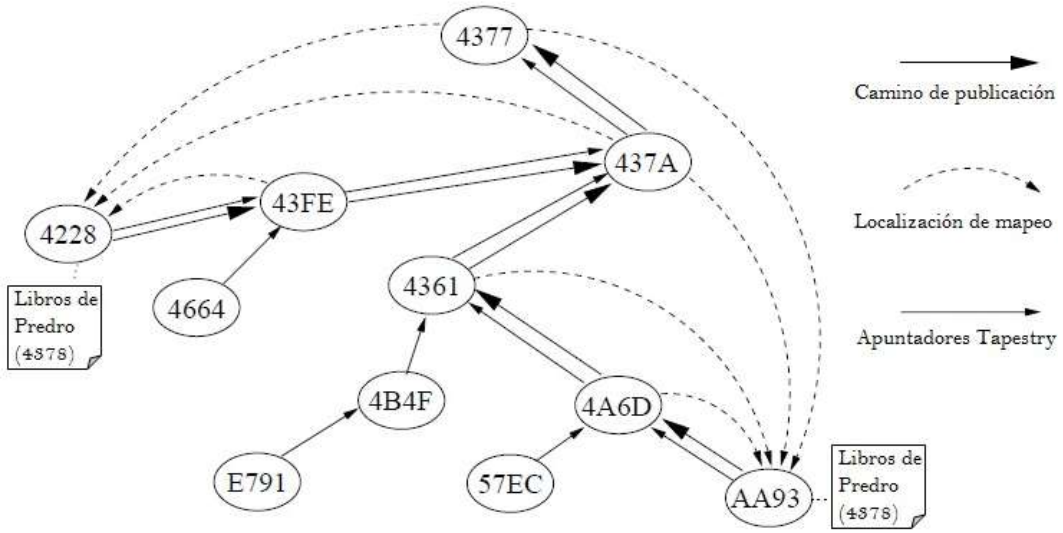


Figura 2.7. *Ejemplo de publicación de un Pastry.* Se publican dos copias de un objeto (4378) a su nodo raíz 4377. Los mensajes de publicación enrutan hacia la raíz, depositando un apuntador de localización del objeto en cada salto encontrado a lo largo del camino.

Un cliente localiza a O al enrutar un mensaje a O_R(ver figura 2.8). Cada nodo en el camino verifica si tiene un mapeo de localización para O. Si es así, redirecciona el mensaje a S. De otra manera, reenvía posteriormente el mensaje a O_R(garantizando tener un mapeo de localización).

Cada salto hacia la raíz reduce el número de nodos que satisfacen la restricción del prefijo del siguiente salto en un factor de la base del identificador. Los mensajes enviados hacia un destino desde dos nodos cercanos generalmente cruzaran los caminos más rápidamente debido a que: cada salto incrementa la longitud del prefijo requerido para el siguiente salto; el camino hacia la raíz es una función sólo del ID del destino, no del nodeID de la fuente; y los saltos hacia los vecinos se eligen por la localidad de la red, la cual es transitiva. Así, mientras más cercano (en distancia de red) sea un cliente a un objeto, más rápido sus consultas probablemente cruzaran los caminos con el camino de publicación del objeto, y más rápido alcanzaran al objeto. Ya que los nodos ordenan los apuntadores a objetos por distancia a ellos mismos, las solicitudes son enrutadas hacia réplicas de objetos cercanas.

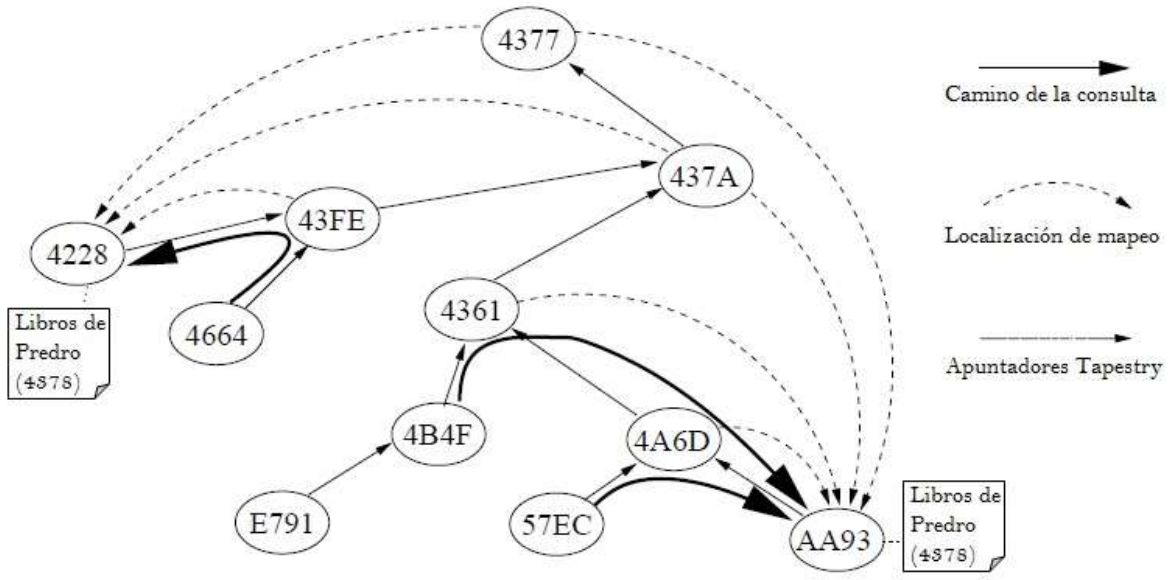


Figura 2.8. *Ejemplo de enrutamiento hacia un objeto Tapestry.* Varios nodos envían mensajes hacia el objeto 4378 desde diferentes puntos en la red. Los mensajes se enrutan hacia el nodo raíz de 4378. Cuando se intersectan en el camino de publicación, siguen el apuntador de localización hacia la copia más cercana del objeto.

2.1.4.2.3. Algoritmos de nodos dinámicos

Tapestry incluye un número de mecanismos para mantener la consistencia de las tablas de enrutamiento y asegurar disponibilidad. La mayoría de los mensajes de control descritos aquí requieren reconocimientos, y son retransmitidos donde sean requeridos.

1) Inserción de nodo: Existen cuatro componentes para insertar un nuevo nodo N en una red Tapestry:

a) *Necesidad de saber.* Los nodos son notificados acerca de N, debido a que N llena una entrada nula en sus tablas de enrutamiento.
b) N debe convertirse en la nueva raíz de objeto para objetos existentes. Las referencias a esos objetos deben ser movidas a N para mantener la disponibilidad de los objetos.
c) Los algoritmos deben construir una tabla de enrutamiento cercanamente óptima para N.
d) Los nodos cercanos a N son notificados y podrían considerar usar a N en sus tablas de enrutamiento como una optimización.

La inserción de un nodo inicia en el suplente S de N (el nodo "raíz" que N_{id} mapea en la red actual). S encuentra a p, la longitud del prefijo más largo que su ID comparte con N_{id}. S envía un mensaje *Multicast Reconocido (Acknowledged Multicast)* para alcanzar al conjunto de todos los nodos existentes que comparten el mismo prefijo, al atravesar un árbol basado en sus nodeIDs. Conforme

los nodos reciben el mensaje, agregan a N a sus tablas de enrutamiento y transfieren referencias de apuntadores enraizados localmente, completando puntos (a) y (b).

Los nodos alcanzados por el mensaje multicast, contactan a N y se convierten en un *conjunto vecino* inicial usado en la construcción de su tabla de enrutamiento. N realiza una búsqueda iterativa del vecino más cercano iniciando con el nivel de enrutamiento p. N usa el conjunto vecino para llenar el nivel de enrutamiento p, recorta la lista hacia los *k* nodos más cercanos, y solicita a estos *k* nodos enviar sus apuntadores de regreso. El conjunto resultante contiene a todos los nodos que apuntan a cualquiera de los *k* nodos en el nivel de enrutamiento previo, y se convierte en el conjunto vecino siguiente. N entonces decrementa a p, y repite el proceso hasta que todos los niveles estén completados. Esto completa el punto (c). Los nodos contactados durante el algoritmo iterativo usan a N para optimizar sus tablas de enrutamiento donde sea aplicable, completando el punto (d).

Para asegurar que los nodos que se están insertando en la red simultáneamente, no fallen al notificarse uno al otro a cerca de su existencia, cada nodo A en el proceso multicast guarda el estado de cada nodo B que aún está en proceso multicast hacia abajo de uno de sus vecinos. Este estado es usado para decirle a cada nodo C con A en su árbol multicast a cerca de B. Adicionalmente, el mensaje multicast incluye una lista de huecos en la nueva tabla de enrutamiento del nodo. Los nodos checan sus tablas con la nueva tabla de enrutamiento y notifican al nuevo nodo de entradas para llenar tales huecos.

2) Eliminación voluntaria del nodo: Si un nodo N abandona la red Tapestry voluntariamente, avisa al conjunto D de nodos en los apuntadores de regreso de N de su intención, junto con un nodo de reemplazo para cada nivel de enrutamiento de su propia tabla de enrutamiento. Los nodos notificados envían tráfico de republicación de objeto a N y su reemplazo. Mientras tanto, N enruta referencias para objetos enraizados localmente a sus nuevas raíces, y avisa a los nodos en D cuando termina.

3) Eliminación de un nodo de manera involuntaria: En una red dinámica y propensa a fallas como Internet, los nodos generalmente se salen de la red, debido a que los nodos o los enlaces fallan o existen particiones en la red, y podría entrar y salir muchas veces en un intervalo corto de tiempo. Tapestry mejora la disponibilidad de los objetos y el enrutamiento en tal ambiente, al construir redundancia en tablas de enrutamiento y referencias de localización de objeto.

Para mantener la disponibilidad y redundancia, los nodos usan balizas periódicas para detectar enlaces salientes y fallas de nodos. Tales eventos ejecutan la reparación de la malla e inician la redistribución y replicación de las referencias de localización de objetos. Además, el proceso de reparación es acrecentado por re-publicación de estado suave de las referencias de los objetos.

2.1.5. Red escalable accesible por contenido [17]

2.1.5.1. Introducción

Una tabla hash es una estructura de datos que mapea eficientemente claves a valores y sirve como un bloque principal en la implementación de sistemas de software. Se supone que muchos sistemas distribuidos a grande escala podrían beneficiarse de igual manera de la funcionalidad de las tablas hash. Se usa el término *Red accesible por contenido (CAN)* para describir tal tabla hash.

Los diseños de igual a igual (peer to peer) emplean enormes cantidades de recursos, sin requerir de planificadores centralizados o grandes inversiones en hardware, ancho de banda, o espacio de almacenamiento. Por definición, la compartición de archivos en sistemas punto a punto podrían originar nuevos modelos de distribución de contenido para aplicaciones tales como: distribución de software, compartición de archivos, y entrega de contenido web estático.

Desafortunadamente, la mayoría de los diseños actuales de los sistemas punto a punto no son escalables. Por ejemplo, en Napster [20] un servidor central almacena el índice de todos los archivos disponibles dentro de la comunidad de usuarios Napster. Para obtener un archivo, un usuario hace una solicitud a este servidor central usando el nombre conocido del archivo deseado y, obtiene la dirección IP de la computadora del usuario que almacena el archivo solicitado. Entonces, aunque Napster es un modelo de comunicación punto a punto para transferir el archivo actual, el proceso para localizar un archivo es aún muy centralizado. Esto lo hace caro (escalar el directorio central) y vulnerable (existe un único punto de falla). La red Gnutella [19] está un paso más adelante, ya que descentraliza el proceso de localización de archivos. Los usuarios en la red Gnutella se auto organizan en un malla a nivel de aplicación en la cual las solicitudes por algún archivo son inundadas dentro de algún cierto alcance. La inundación en cada solicitud es claramente no escalable y, debido a que la inundación debe ser cortada en alguna parte, podrá fallar al encontrar contenido que está actualmente en el sistema.

El punto central para cualquier sistema punto a punto es el esquema de indexado usado para mapear nombres de archivos (ya sea algún nombre bien conocido o alguno descubierto por medio de algún mecanismo externo) a su localización en el sistema. Es decir, el proceso de transferencia de archivos punto a punto es por naturaleza escalable, pero la parte difícil es encontrar el par a partir del cual obtener el archivo. De este modo, un sistema punto a punto escalable requiere, por lo menos, un mecanismo de indexado escalable.

Como se dijo anteriormente, una red CAN se parece a una tabla hash; las operaciones básicas realizadas en un red CAN son la inserción, búsqueda y eliminación de pares (clave, valor). Cada nodo CAN almacena una parte (llamada *zona*) de la tabla hash completa. Además, un nodo mantiene información acerca de un pequeño número de zonas adyacentes en la tabla. Las solicitudes (inserción, búsqueda, o eliminación) hacia una clave particular son enrutadas por nodos CAN intermedios hacia el nodo CAN de quien su zona contenga tal clave.

A diferencia de sistemas como DNS o enrutamiento IP, el diseño de CAN no impone ninguna forma de estructura de nombres jerárquica rígida para obtener escalabilidad. El diseño se puede implementar totalmente en la capa de aplicación.

2.1.5.2. Diseño

El diseño se centra en un espacio virtual de coordenadas cartesianas *d*-dimensional sobre un *d*-torus. Este espacio de coordenadas es completamente lógico y no tiene relación con algún sistema de coordenadas físico. En cualquier punto del tiempo, el espacio completo de coordenadas es dinámicamente dividido entre todos los nodos en el sistema, tal que cada nodo posee su zona individual y distinta dentro todo el espacio. Por ejemplo, en la figura 2.9 se muestra un espacio de coordenadas 2-dimensional $[0,1] \times [0,1]$ dividido entre 5 nodos CAN.

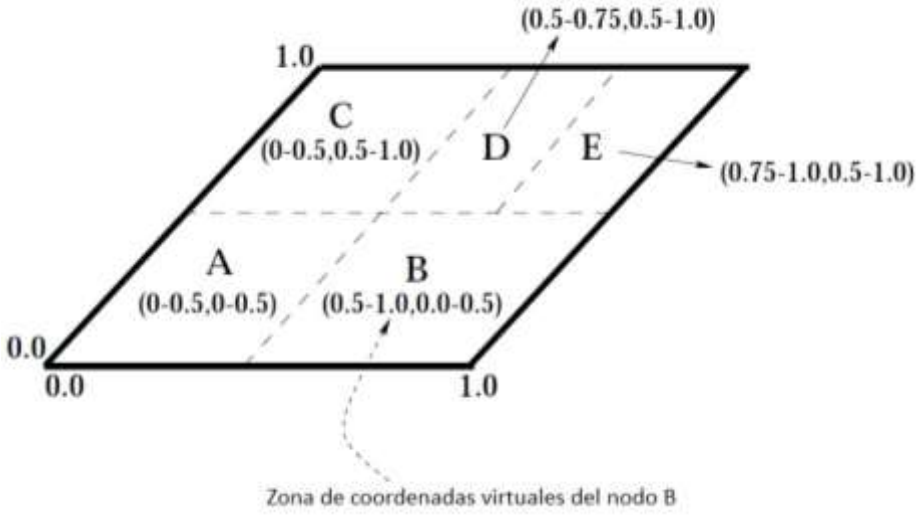


Figura 2.9. Ejemplo de espacio 2-d con 5 nodos

Este espacio de coordenadas virtuales se usa para almacenar pares (clave, valor) de la siguiente manera: para almacenar un par (K_1, V_1), la clave K_1 se mapea de manera determinista en un punto P en el espacio de coordenadas usando una función hash uniforme. El par (clave, valor) correspondiente, es almacenado en el nodo que pertenece a la zona en el cual el punto P cae. Para obtener un una entrada correspondiente a la clave K_1, cualquier nodo puede aplicar la misma función hash determinista para mapear K_1 en el punto P y, entonces obtener el valor correspondiente a partir del punto P. Si el punto P no está poseído por el nodo solicitante o sus vecinos inmediatos, la solicitud debe ser enrutada hacia la infraestructura CAN hasta que alcance al nodo que pertenece a la zona en el cual el punto P cae. Por lo tanto, el enrutamiento eficiente es un aspecto crítico de una red CAN.

Los nodos en la red CAN se auto organizan dentro de una red superpuesta que representa al espacio de coordenadas virtuales. Un nodo aprende y mantiene las direcciones IP de los nodos que tienen

zonas adjuntas a la de él. Este conjunto de vecinos intermedios en el espacio de coordenadas sirve como una tabla de enrutamiento de coordenadas que habilita el enrutamiento entre puntos arbitrarios en este espacio.

2.1.5.2.1 Enrutamiento en una red CAN

Intuitivamente, el enrutamiento en una red accesible por contenido funciona al seguir el camino directo hacia el espacio cartesiano a partir de las coordenadas de la fuente hasta el destino.

Un nodo CAN mantiene una tabla de enrutamiento de coordenadas que contiene la dirección IP y la zona de coordenadas virtuales de cada uno de sus vecinos inmediatos en el espacio de las coordenadas. En un espacio de coordenadas *d*-dimensional, dos nodos son vecinos si sus coordenadas se empalman a lo largo de $d-1$ dimensiones y colinda con una dimensión. Este estado estrictamente local del vecino, es suficiente para enrutar entre dos puntos arbitrarios en el espacio ya que, un mensaje CAN incluye las coordenadas del destino. Usando su conjunto de coordenadas de sus vecinos, un nodo enruta mensajes hacia su destino simplemente al reenviar de manera voraz (*greedy*) hacia el vecino con coordenadas más cercanas a las coordenadas del destino. Para un espacio d dimensional dividido en n zonas iguales, la longitud promedio del camino de enrutamiento es de $(d/4)(n^{1/d})$ saltos y cada nodo mantiene un tamaño de vecindario de $2d$. Estos resultados escalables significan que para un espacio d dimensional, se puede incrementar el número de nodos (y por lo tanto zonas) sin incrementar el estado por nodo, mientras que el promedio de la longitud del camino se incrementa en $O(n^{1/d})$.

Se debe notar que existen muchos caminos diferentes entre dos puntos en el espacio y por lo tanto, incluso si uno o más de los vecinos de un nodo, falla, un nodo automáticamente puede enrutar a través del siguiente camino mejor disponible.

Si a pesar de esto, un nodo pierde a todos sus vecinos en una cierta dirección, y los mecanismos de reparación aún no han reconstruido el hueco en el espacio de coordenadas, entonces el reenvío voraz podría fallar. En este caso, un nodo podría usar una búsqueda de expansión de anillo (usando inundación controlada sobre la malla unicast CAN superpuesta) para localizar un nodo que esté más cercano al destino. El mensaje entonces es reenviado a este nodo más cercano, y a partir del cual se continúa con el reenvío voraz.

2.1.5.2.2. Construcción de CAN

Como se describió anteriormente, el espacio completo de CAN se divide entre los nodos presentes en el sistema. Para permitir que el sistema CAN crezca de manera incremental, un nuevo nodo que se une al sistema debe ser asignado a su propia porción del espacio de coordenadas. Esto se hace

al dividir la zona asignada a un nodo existente por la mitad, reteniendo la mitad y la otra mitad manejada por el nuevo nodo.

El proceso se lleva a cabo en tres pasos:

1. Primero, el nuevo nodo debe encontrar un nodo existente en el sistema CAN.
2. Después, usando los mecanismos de enrutamiento de CAN, debe encontrar un nodo a quien se dividirá su zona.
3. Finalmente, los vecinos de una zona dividida deben ser notificados tal que, el enrutamiento pueda incluir al nuevo nodo.

2.1.5.2.2.1. *Arranque*

Un nuevo nodo CAN primero descubre la dirección IP de cualquier nodo presente en el sistema. El funcionamiento de un sistema CAN no depende de los detalles de cómo se hace, pero en CAN se usa el mismo mecanismo de arranque como en YOID [18].

Al igual que en [18] se asume que un sistema CAN tiene un sistema de nombres de dominio, y que éste resuelve una dirección IP de uno o más nodos iniciador CAN. Un nodo iniciador mantiene una lista parcial de nodos CAN lo cuales cree que están presentes en el sistema. Las técnicas simples para obtener esta lista razonablemente se encuentran descritas en [18].

Para unirse a un sistema CAN, un nuevo nodo busca el nombre de dominio CAN en el DNS para obtener la dirección IP de un nodo iniciador. El nodo iniciador entonces suministra las direcciones IP de varios nodos elegidos aleatoriamente presentes en el sistema.

2.1.5.2.2.2. *Encontrar una zona*

El nuevo nodo entonces aleatoriamente elige un punto *P* en el espacio y envía una solicitud de UNIÓN destinada para el punto *P*. El mensaje es enviado en el sistema CAN por medio de cualquier nodo CAN existente. Cada nodo CAN entonces usa el mecanismo de enrutamiento para reenviar el mensaje, hasta que alcanza a la zona del nodo donde *P* cae.
El nodo ocupante actual entonces divide su zona a la mitad y asigna la otra mitad al nuevo nodo. La división se hace al asumir un cierto orden de las dimensiones en decisión junto con la dimensión de una zona la cual será dividida, tal que las zonas puedan ser reincorporadas cuando un nodo abandona el sistema.

Para un espacio 2 dimensional, una zona primero sería dividida a lo largo de la dimensión X, y después el eje Y. Los pares (clave, valor) de la mitad de la zona que será entregada, son también, transferidos al nuevo nodo.

2.1.5.2.2.3. *Unión al enrutamiento*

Una vez que se ha obtenido la zona, el nuevo nodo aprende las direcciones IP de su conjunto vecino de coordenadas a partir del ocupante previo. Este conjunto, es un subconjunto de los vecinos ocupantes anteriores, más el ocupante actual. De la misma manera, el ocupante anterior actualiza su conjunto vecino para eliminar a los nodos que ya no son sus vecinos. Finalmente, se debe avisar a los nuevos y anteriores vecinos de esta reasignación de espacio. Cada nodo en el sistema envía un mensaje de actualización inmediata, seguido por reenvíos periódicos, con su zona actual asignada, a todos sus vecinos. Estas actualizaciones de estado suave aseguran que todos sus vecinos rápidamente conozcan a cerca del cambio y que actualicen sus conjuntos vecino correctamente. La adición de un nuevo nodo sólo afecta a un número pequeño de nodos existentes en una localidad muy pequeña del espacio de coordenadas. El número de vecinos que un nodo mantiene, depende sólo de la dimensión del espacio de coordenadas, y es independiente del número total de nodos en el sistema. Así, la inserción de un nodo afecta sólo a $O(número\ de\ dimensiones)$ nodos existentes, lo cual es importante para sistemas CAN con un gran número de nodos.

2.1.5.3 Salida de un nodo, recuperación y mantenimiento de un sistema CAN

Cuando los nodos dejan un sistema CAN, se necesita asegurar que las zonas que ocuparon son retomadas por los nodos restantes. El procedimiento normal para llevar a cabo esto es, que un nodo explícitamente entregue su zona y la base de datos asociada (clave, valor) a uno de sus vecinos. Si la zona de uno de los vecinos puede ser incorporada con la zona del nodo que está abandonando se puede producir una única zona válida, entonces se lleva a cabo. Si no, entonces la zona es entregada al vecino de quien su zona actual sea la más pequeña, y ese nodo manejará temporalmente ambas zonas.

El sistema CAN además, necesita ser robusto para nodos o fallas en la red, donde uno o más nodos simplemente se vuelven inalcanzables. Esto se maneja a través de un algoritmo que inmediatamente toma el control para asegurar que alguno de los vecinos del nodo que falló, tome el control de la zona. Sin embargo en este caso, los pares (clave, valor) del nodo saliente se pierden, hasta que el estado se refresca por los poseedores de los datos.

Bajo condiciones normales, un nodo envía mensajes de actualización periódica a cada uno de sus vecinos, dándoles las coordenadas de su zona y una lista de sus vecinos y sus coordenadas de zona. La ausencia prolongada de un mensaje de actualización de un vecino quiere decir que existe una falla.

Una vez que un nodo ha decidido que su vecino ha muerto, inicia el mecanismo de toma de control e inicia la ejecución de un temporizador para la toma de control. Cada vecino del nodo fallido hará esto de manera independiente, con el temporizador inicializado en proporción al volumen de su

zona. Cuando un temporizador expira, un nodo envía un mensaje de TOMA DE CONTROL transmitiendo su volumen de zona a todos los vecinos del nodo fallido.

Un nodo al recibir un mensaje de TOMA DE CONTROL, cancela su temporizador si el volumen de zona en el mensaje es más pequeño que su volumen de zona, o responde con su propio mensaje de TOMA DE CONTROL. De esta manera, se elige a un nodo eficientemente del vecindario el cual está activo y tiene un volumen de zona pequeño.
Bajo ciertos escenarios de fallas que involucran las fallas de múltiples nodos adyacentes, es imposible que un nodo detecte una falla, a menos que la mitad de los vecinos de un nodo fallido aún estén activos. Si el nodo toma el control de otra zona bajo estas circunstancias, es posible que el estado del sistema CAN se vuelva inconsistente. En tales casos, antes de lanzar el mecanismo de reparación, el nodo lleva a cabo una búsqueda por expansión de anillo para cualesquiera nodos situados más allá de la región de la falla y por lo tanto, al final reconstruye el estado del vecindario apropiado para iniciar una toma de control segura.

Finalmente, el procedimiento normal de salida y el algoritmo de toma de control inmediato puede resultar en un nodo que mantenga más de una zona. Para prevenir la fragmentación repetida del espacio, se ejecuta en segundo plano un algoritmo de reasignación de zona para asegurar que el sistema CAN tienda a tener una zona para cada nodo.

2.2. Redes orientadas a contenido

2.2.1. Optimización de tablas de enrutamiento semántico [21]

2.2.1.1. Introducción

En el futuro, los usuarios de Internet preferirán acceder a los recursos de la red basados en sus intenciones más que usando direcciones IP o nombres de recursos de red.
Una *Red Enrutada Semánticamente (RES)* enruta mensajes basándose en su significado en lugar de sus direcciones IP, y puede satisfacer razonablemente las necesidades del usuario. Para enviar un mensaje a un nodo particular de la red, el mensaje es cargado con una *clave semántica,* la cual tiene significado similar a la descripción del destino deseado, entonces el destino puede responder de regreso a la fuente del mensaje. Una RES puede implementarse por una red interconectada de *enrutadores semánticos* los cuales comparan la similitud entre las claves de los mensajes y la descripción de los recursos almacenados en las *tablas de enrutamiento semántico* para resolver el siguiente salto destino (*búsqueda semántica*). Esta RES podría ser una red superpuesta por encima de los protocolos de Internet e IP. Los mensajes semánticos pueden ser cargas de datos en paquetes IP y, la conectividad entre enrutadores semánticos (hosts de Internet) puede ser sobre TCP/UDP expandiéndose a través de varios enrutadores IP en Internet. Se pueden usar nombres URI (Uniform

Resource Identifier) como direcciones físicas subyacentes para permitir localización física de los múltiples recursos, los cuales podrían ser un destino distinto de la RES, y SOAP se puede usar como el transporte de mensajes sobre TCP.

El enrutamiento semántico y las RES tienen varias aplicaciones y ventajas. Cuando una red tipo RES está disponible, la búsqueda de recursos en una grid es posible sin servicios de indexación/directorio. La naturaleza distribuida de las RES, da mejor escalabilidad y cubrimiento par la búsqueda/descubrimiento de recursos en una grid. Los usuarios no necesitan saber la descripción exacta (palabras claves que coincidan) de los datos/servicios o su esquema, simplemente pueden enviar una solicitud de "retrollamada" con la clave semántica apropiada, la cual describe los servicios/datos deseados. Los enrutadores semánticos toleran la variabilidad de las claves usadas siempre y cuando las claves contengan el mismo significado e información suficiente para desambiguar el significado. Esta clase de sistema de obtención semántica de datos es novedoso porque las redes accesibles por contenido [22], sistemas de nombres intencional [23], o redes de búsqueda distribuida [24,25], disponibles sólo soportan enrutamiento/búsquedas con claves de comparación exactas.

El verdadero reto en una RES es proveer rendimiento en el enrutamiento escalable, en términos de éxito de enrutamiento extremo a extremo y tiempo de respuesta bajo, para cualquier clave significativa. La calidad de la tabla de enrutamiento determina el rendimiento del enrutamiento, el cual en su momento afecta la tasa *recall* en las búsquedas y el tiempo de respuesta, cuando se usa una RES como sistema de obtención semántica de datos. La tabla de enrutamiento necesita ser optimizada para mejorar el rendimiento del enrutamiento. Desde esta perspectiva las propuestas disponibles [22-25], no intentan mejorar la calidad del contenido de la tabla de enrutamiento (o la estructura de datos de la hash) y los mecanismos de enrutamiento semántico [26] sugeridos no son expansibles, y requieren de mecanismos adicionales para mejorar el rendimiento del enrutamiento.

La búsqueda semántica es un proceso intensivo de recursos, por lo tanto los enrutadores semánticos sólo soportan pequeñas tablas de enrutamiento teniendo limitado el número de entradas clave-destino y servicio limitado en la tasa de mensajes de tráfico. Este problema de escalabilidad se trata al usar un número grande de enrutadores pequeños, en lugar de un número pequeño de enrutadores grandes. Para hacer efectiva la tabla de enrutamiento semántico se aplican varias técnicas que: (1) mejoran la calidad de su contenido; (2) e imparten un comportamiento auto organizado para la RES tal que se mejora el rendimiento de enrutamiento con el tiempo y uso.

2.2.1.2 Diseño de la Red Enrutada Semánticamente (RES): principios básicos y retos

2.2.1.2.1 Modelo abstracto de la RES y concepto de Enrutamiento Semántico

Una RES consiste de nodos enrutadores semánticos y nodos recursos que representan los recursos de la red. Los enrutadores almacenan las direcciones de todos los recursos y nodos enrutadores registrados a él para el reenvío de mensajes. Un nodo recurso envía mensajes a otro nodo recurso o enrutador vía el nodo enrutador al cual está registrado. La resolución de rutas (*búsqueda semántica*) involucra elegir un único o unos cuantos destinos de reenvío, basados en cual descriptor de destino D_i está más cercano (más similar) al descriptor clave del mensaje "K". Se denota a la métrica de similitud semántica entre D_i y "K" por la notación $K \rightarrow D_i$. Si $K \rightarrow D_i > K \rightarrow D_K$ entonces el mensaje será reenviado al destino que tenga la descripción D_i.

2.2.1.2.2 Estructura de datos del descriptor semántico: Motivación y retos

Si se elige una organización de red que agrupe destinos basándose en qué tan similares son sus descriptores, entonces esto permite reemplazar múltiples entradas de mapeos clave-destino (renglones de la tabla de enrutamiento) con una única entrada (renglón). Sin embargo, esto requiere de identificar a todos los destinos de quienes sus descriptores son similares entre ellos. Para la resolución de rutas, también se necesita la misma capacidad de similitud semántica. Entonces se necesita: (1) diseñar una *estructura de datos tratable computacionalmente y escalable semánticamente* para representar el significado; y (2) diseñar un método α para comparar dos estructuras de datos descriptor y generar una métrica de similitud semántica. Esta estructura de datos se usará para definir la clave semántica y los descriptores destino, y el método de comparación está sujeto al diseño de la estructura. Algunas propuestas tradicionales para capturar, transportar y comparar significados usando propuestas de red léxicas [27], ontológicas [28], y semánticas [29] tiene limitación en una aplicación de tipo RES. Estos modelos no escalan semánticamente, lo cual quiere decir ya sea que no pueden capturar correctamente y comparar moderadamente significados complejos que son requeridos, o son computacionalmente caros para un enrutador.

2.2.1.2.3 Organización de red: Motivación y retos

Otros problemas tratan con la organización y topología de la red los cuales tienen efectos en la escalabilidad. La topología jerárquica fue suficiente para contrarrestar el problema de la escalabilidad en redes IP porque las direcciones IPv4/v6 tienen rango limitado y el enrutador podría soportar tablas de enrutamiento más grandes. Sin embargo, la organización jerárquica es insuficiente para contrarrestar los retos de escalabilidad en una RES debido a que el espacio de enrutamiento semántico es bastante grande mientras que las tablas de enrutamiento tienen que ser más pequeñas. El rango del espacio de enrutamiento, el cual representa todos los posibles mapeos entre todos los puntos en espacio de direcciones y los puntos en el espacio del destino actual, es el producto de: (i) el número de todos los recursos posibles que tienen que ser individualmente direccionados (espacio de destinos) y, (ii) el número de posibles maneras para describir cada recurso (espacio de direcciones clave). Ya que los usuarios probablemente usan múltiples descripciones para direccionar un recurso, la dirección combinatoria resultante es extremadamente grande comparada con el espacio de direcciones IP. De la misma manera, el

número de objetos semánticos (nodos destino) es probablemente de magnitud de varios ordenes más grande que los destinos IP posibles, debido a que la disponibilidad de la RES apoya a aplicaciones la cuales necesitan de un gran número de objetos semánticos individuales. Por lo tanto el reto es cómo generar una topología de red escalable alterna que: (1) cubra el espacio grande de enrutamiento semántico para producir una tasa más alta de éxito de enrutamiento (porcentaje de mensajes que alcanzan satisfactoriamente a los destinos para una variedad de combinaciones clave-destino), (2) limite el tiempo de respuesta al limitar el número de saltos de enrutamiento semántico para un tamaño dado de tabla de enrutamiento; (3) minimice la congestión; y (4) minimice el número de enrutadores.

2.2.1.3 Descriptor semántico y comparación de similitud

2.2.1.3.1 Estructura de datos del descriptor para conceptos complejos

El significado es un *concepto* en la mente humana. Los seres humanos componen *conceptos complejos* (e.g., "Estación del tiempo en Chicago") en sus mentes, usando una red asociativa (conocida como "red semántica" [29]) de conceptos elementales (e.g., "Estación de tiempo", "Chicago"). Entonces se puede usar una red semántica como estructura de datos del descriptor semántico para representar/almacenar, transmitir y comparar el significado intencionado como fue entendido por los usuarios humanos. Se propone representar la red semántica por un conjunto plano simple de conceptos elementales. En este esquema el descriptor D_i de un concepto complejo es representado por un conjunto de conceptos elementales: $\{C_{i1}, C_{i2}, \dots, C_{i3}\}$.

2.2.1.3.2 Comparación de similitud de conceptos complejos

El método para comparar similitud semántica entre dos descriptores está basado en similitudes semánticas entre sus conceptos elementales. Por ejemplo, cuando $D_1 = \{C_{11}, C_{12}\}$ y $D_2 = \{C_{21}, C_{22}\}$, entonces $D_1 \to D_2$ es representado por un conjunto ordenado $S_{12} = \{s_1, s_2, \dots, s_i, \dots, s_n\}$, donde $s_1 > s_2 > \cdots s_i$, y cada elemento s_i pertenece a un conjunto $\{C_{11} \to C_{21}, C_{11} \to C_{22}, C_{12} \to C_{21}, C_{12} \to C_{22}\}$. Suponer que $D_1 \to D_2 = S_{12} = \{5,5,4,3,1,1,0\}$ y $D_3 \to D_4 = S_{34} = \{5,5,4,4,3,1,1,0\}$. El conjunto de elementos con prefijo común de los dos conjuntos ordenados S_{12} y S_{34} es $\{5,5,4\}$. Una vez que se quiten los elementos $\{5,5,4\}$ de S_{12}, el conjunto ordenado de elementos restantes es $S'_{12} = \{3,1,1,0\}$. De la misma manera $S'_{34} = \{4,3,1,1,0\}$. Los elementos máximos de los conjuntos S'_{12} y S'_{34} son 3 y 4 respectivamente. Como $3 < 4$, por lo tanto se asegura que $D_1 \to D_2 < D_3 \to D_4$. Aquí las relaciones más cercanas son las que más influencian la comparación. Usar un conjunto plano de conceptos elementales es una elección práctica debido a que barato. Después, se representa la estructura de datos del concepto elemental (e.g., C_{11}) y la técnica de comparación de similitud asociada (e.g., $C_{1i} \to C_{2j}$), la cual proporciona suficiente expresividad.

2.2.1.3.3 Estructura de datos para conceptos elementales

Los seres humanos conceptualizan y recuerdan objetos por sus atributos y relaciones asociadas a la suposición. Por lo tanto, se usa una estructura de datos conocida como *concept lattice* [12,13] para representar conceptos elementales en término de atributos.

Un *concept lattice* está compuesto por múltiples conceptos unitarios, cada uno de los cuales es representado por un par {O, A} donde "O" denota un conjunto de objetos los cuales tienen todos sus atributos pertenecientes a un conjunto denotado por "A". Por lo tanto, el significado de un objeto perteneciente a O puede ser expresado por esta representación de concepto unitario {O, A}.

Cuando el significado de un atributo puede ser comunicado des ambiguamente por un conjunto más pequeño de símbolos alternativos (representaciones léxicas) bajo todos los posibles contextos, entonces son considerados como atributos de más bajo nivel o primitivos. De otra manera los atributos requieren una descomposición mayor en el *concept lattice*. Los atributos primitivos son representados por representaciones léxicas.

2.2.1.3.4 Comparación de similitud de conceptos elementales

Los autores de RES modificaron la técnica sugerida en [30,31] para comparar dos conceptos *lattice*. La métrica de similitud es derivada basándose en cuántos nodos en los dos conceptos *lattice* comparten elementos comunes del conjunto de objetos "O" y el conjunto de atributos "A". Primero, un conjunto candidato de objetos y atributos que son comunes para ambos conjuntos *lattice* es identificado (paso 1). Después, para cada objeto común (y atributo) en este conjunto candidato, los conceptos más específicos son extraídos de ambos conjuntos *lattice* (paso 2). Estos conceptos unitarios candidatos extraídos son comparados entre ellos mismos para comprobar si empatan o no (paso 3). Un emparejamiento indica que hay al menos un objeto común y un atributo común entre los dos conceptos unitarios $\{O_i, A_i\}$ y $\{O_k, A_k\}$ pertenecientes a estos dos conceptos *lattice*. El número total de emparejamientos encontrados es normalizado (paso 4) por el número total de todos los pares de conceptos extraídos considerados para emparejarlos.

2.2.1.4 Mecanismo de organización de red

2.2.1.4.1 Principios clave de la topología

La topología de red es caracterizada por una pequeña longitud de camino esperada (distancias en saltos) y un coeficiente agrupamiento grande (probabilidad de que dos nodos estén conectados si tienen un nodo peer común) [32]. Esta topología es generada si una fracción muy pequeña de las aristas en un grafo uniforme *lattice* son desconectados para un nodo el cual está lejos.

2.1.5.4.2 Comportamiento de los nodos RES y generación de la pequeña topología del mundo

Los nodos enrutador y recurso tienen comportamiento dinámico (como un agente) en virtud de que ellos se interconectan rápidamente para organizar un topología pequeña del mundo. Los nodos recurso tienen un descriptor único llamado *descripción de recurso*, mientas que los enrutadores tienen un número pequeño de múltiples descriptores no similares, llamados *intereses del enrutador*, los cuales están semánticamente separados uno del otro. Los enrutadores prefieren registrar sólo a los nodos enrutadores y recursos en su tabla de enrutamiento, de quienes sus descripciones sean cercanas a sus propios intereses. Mientras que un recurso prefiere registrase a un único enrutador de quien uno de sus intereses es el más cercano semánticamente a su descripción. Cada renglón de la tabla de enrutamiento tiene múltiples columnas para permitir el reenvío de mensajes hacia múltiples destinos los cuales incrementan efectivamente el coeficiente de agrupamiento. Para incrementar más este coeficiente, se debe elegir reenviar mensajes a destinos en los *t* renglones más similares de la tabla de enrutamiento. Esto puede duplicar los mensajes, por lo tanto los enrutadores detectan y desechan los mensajes duplicados.

Se puede visualizar un *espacio de enrutamiento semántico* (basado en una escala ordinal) donde un nodo físico RES pueda ser representado como un nodo virtual del *grafo de enrutamiento semántico.* Un nodo recurso es representado como un nodo virtual único del grafo, mientras que un nodo enrutador es representado por múltiples nodos virtuales del grafo, uno para cada interés del descriptor. Los nodos virtuales del grafo son conectados por aristas uni/bidireccionales dependiendo de la conectividad física entre los respectivos nodos RES. La distancia entre los nodos virtuales del grafo varía dependiendo de la similitud semántica entre las descripciones/intereses de los nodos RES, entonces la preferencia para conectarse a nodos similares genera aristas cortas entre sus nodos virtuales para formar grupos en una estructura *lattice*. La diversidad en los intereses de los enrutadores permite a sus nodos virtuales ser parte de múltiples grupos, lo cual genera efectivamente aristas largas en el grafo de enrutamiento, haciendo un grafo pequeño que representa al del mundo.

2.2.1.4.3 Algoritmo de agrupamiento de nodos

Para identificar a los nodos con descripciones similares, un enrutador envía periódicamente mensajes de consulta, uno para cada descripción de interés, junto con un umbral de comparación de similitud. Los nodos de quienes sus descripciones sean similares a la clave de consulta (la métrica de similitud es mayor que el umbral) responde al enrutador y el enrutador los registra. Si la tabla de enrutamiento aún tiene espacio, el enrutador expande progresivamente su horizonte al enviar consultas con un umbral más pequeño. De la misma manera, un nodo recurso envía consultas para identificar al enrutador con más parecido para registrase.

2.2.1.4.4 Política de vaciado de registros en la tabla de enrutamiento

Cada enrutador tiene un número de registros y columnas en su tabla de enrutamiento. Por lo tanto, los enrutadores semánticos se esfuerzan por mejorar la eficiencia de la calidad de espacio del contenido de la tabla de enrutamiento sobre el tiempo, al remplazar las entradas de la tabla de enrutamiento con otras más relevantes, tal que los mensajes sean correctamente encaminados hacia el destino a través de la ruta más corta posible. Las direcciones de los destinos en las columnas de la tabla de enrutamiento son conservadas o eliminadas basándose en la relevancia de la descripción del nodo destino con respecto a la clave del renglón. La dirección de quien su descripción está más lejos semánticamente de clave del renglón, es eliminada. De la misma manera, el renglón candidato por eliminar, es elegido basándose en el interés relativo del enrutador, al comparar la similitud entre la clave del renglón y los descriptores de interés del enrutador.

2.2.1.4.5 algoritmos de reorganización de la tabla de enrutamiento

Reasignación de las entradas columna: Cuando se elimina a un destino de algún renglón, se intenta reasignarlo en otro renglón el cual es el siguiente posible mejor lugar para él y el cual está vacío. La verificación del espacio para el registro se hace con la siguiente lógica. Si hay un espacio en la columna libre, entonces la prueba de espacio regresa “true”. Si no hay espacio libre en la columna entonces se hace una verificación para comprobar si el destino que se está reasignando probablemente será eliminado desde este renglón en un futuro cercano o no. Si sucede que el destino reasignado probablemente sea eliminado en el futuro cercano, entonces se considera el siguiente renglón que mejor coincida. La verificación para la posible eliminación en el futuro se hace al comparar la relevancia de la descripción del nodo destino reasignado con la descripción del nodo destino el cual ya está en ese renglón. Esta prueba de “relevancia” es llevada a cabo con respecto a la clave del renglón que está siendo considerado.

Si la prueba de espacio devuelve “true” para un renglón el cual de hecho no tiene un espacio libre en la columna, entonces el destino menos relevante es eliminado de este renglón para hacer espacio para el destino el cual está siendo reasignado. Esto conduce a una recursión de reasignación de registros en la columna. Para identificar el mejor renglón que empate, sólo son considerados los renglones los cuales no han sufrido una eliminación de destino hasta aquí, en el árbol de recursión actual.

Reasignación de un renglón completo: Cuando un renglón entero es eliminado (para hacer espacio para un nuevo renglón, para que un enrutador pueda expandir su horizonte), se hace un intento de reasignar todos sus destinos usando el algoritmo de reasignación de entradas columna.

2.2.2. Algoritmo de búsqueda Difusa para redes P2P estructuradas basadas en una matriz semántica multidimensional. [33]

2.2.2.1 Introducción

El algoritmo de búsquedas centralizado en Internet, como Google y Yahoo, tienen muchos problemas en capacidad de datos, eficiencia en búsqueda, tolerancia a fallas, y no son capaces de cumplir con la demanda del usuario. La tecnología P2P, está enfocada en hacer uso de recursos masivos de Internet de manera razonable y eficiente. Y la red P2P, tiene el reto de proveer el servicio de búsqueda eficiente cuando los usuarios están distribuidos ampliamente y en un gran número, el comportamiento de los nodos no es controlable y la capacidad de cómputo y el ancho de banda de las redes no es uniforme. Existen modelos de búsquedas en P2P los cuales son búsquedas por índices centralizados, inundación, y THD (Tablas Hash Distribuidas). Las THD han dominado las aplicaciones de búsqueda. Sin embargo, las redes P2P basadas en la tecnología THD sólo pueden proveer la búsqueda precisa con palabras clave únicas, pero no soportan búsquedas multi-clave, rango de búsqueda, unión de búsqueda, búsqueda de similitud y otras maneras de búsqueda semántica, lo cual es debido a que la operación de la función Hash lleva a cabo una relación de mapeo precisa entre el nombre de recurso y el ID del nodo.

Existen dos tipos de algoritmos para resolver los problemas de búsqueda semántica en redes P2P estructuradas: el Algoritmo de mejora del mecanismo de enrutamiento y el algoritmo de mejora de caché. En el primer algoritmo, la solicitud de búsqueda se transmite a cierta parte de los nodos vecinos, no a todos los nodos, por lo tanto el tráfico se decrementará y los resultados se pueden satisfacer. Caché de contenido el cual es una manera de copiar los recursos compartidos hacia algunos nodos con carga ligera con el propósito de aligerar la carga pesada de los nodos. Sin embargo, los algoritmos mencionados anteriormente simplemente hace algunas mejoras con recursos locales para reducir las búsquedas a ciegas, y eso no es suficiente para mejorar la eficiencia de las búsquedas semánticas y el ámbito.

En el presente apartado se describe a FSA-MDSM, un algoritmo de búsqueda difusa basado en matriz semántica multi-dimensional. El algoritmo propone maneras novedosas de selección de palabras semánticas claves y maneras de constituir vectores semánticos, nuevas medios de compresión para matrices semánticas multi-dimensionales, y nuevas maneras de agrupamiento de nodos semánticamente similares. El algoritmo puede aumentar altamente la eficiencia y precisión de búsquedas semánticas para las redes P2P basadas en THD.

2.2.2.2 Composición y optimización de la matriz de vectores semánticos

Con el propósito de comparar con la similitud semántica de los recursos, primeramente, se deberían de extraer los caracteres del recurso, y pueden ser expresados en una forma semántica cierta.

De acuerdo al Modelo del Espacio Vectorial (VSM, por su siglas en inglés vector space model), el recurso compartido es considerado como un vector en un espacio de vectores multi-dimensional y está compuesto de una serie de términos, de quien su orden de ocurrencia es irrelevante, pero se debería establecer un valor de peso para cada término basado en la frecuencia de uso normalmente del término. Y la combinación de términos y sus pesos pueden ser usados para describir al recurso, tal que el problema de la representación y el emparejamiento de los recursos podrían ser cambiados en la representación y el emparejamiento de los vectores.

2.2.2.2.1 Algoritmo para la construcción del vector semántico

En primer lugar, se puede extraer un número de palabras clave en el recurso con la ayuda de un diccionario semántico. Por ejemplo, en el famoso WordNet, un peso correspondiente w_i debería de agregarse a las palabras clave a_i de acuerdo a un algoritmo seguro, así este recurso es transformado en un número de tuplas (a_i, a_w). $\{a_1, a_2, \dots, a_m\}$ el cual es el conjunto de palabras clave que representan al contenido del recurso y puede ser visto como un sistema de coordenadas m-dimensional y los pesos $w_1, w_2, \dots, w_m$ son los valores de las coordenadas correspondientes. Cada recurso A puede ser mapeado en un vector de caracteres en tal espacio del vector.

$$A = (a_1 w_1, a_2 w_2, \dots, a_m w_m) \qquad 2.5$$

Como en la frecuencia de aparición, especialmente la importancia de cada palabra clave es diferente en el diccionario semántico, se debería de dar un peso a cada palabra la cual representa la importancia de las palabras clave, tal que el cálculo de la similitud del vector semántico podría ser más razonable y preciso. En general, los pesos de las palabras clave deberían cumplir dos requerimientos básicos: primero, reflejar las características del recurso representado; segundo, hacer obvia la diferencia entre los recursos. Por lo tanto, la conclusión puede ser conseguida tal que la importancia de las palabras clave es proporcional a su KF (Keyword Frecuency, frecuencia de palabras clave) en un recurso, e inversamente proporcional al DF (Resource Frecuency, frecuencia del recurso) en la biblioteca de texto de entrenamiento, aquí el recurso se refiere a aquellos recurso que contienen la palabra clave. Se puede combinar el KF y el DF y se puede construir una función de peso en una forma logarítmica de la manera siguiente.

$$w_i = KF_i \times log\left(0.00625 + \frac{M}{m_j}\right) \qquad 2.6$$

En la Formula 2, KF_i representa la frecuencia de aparición de la palabra clave en el recurso actual, M es el número total de recursos en el recurso de entrenamiento, m_j es el número de recursos que contienen la palabra clave, y K_i, 0.00625 es el factor de regulación.

Ya que el número de palabras clave extraídas de diferentes recursos podrían no ser las mismas, la longitud del vector en construcción tampoco es la misma, la cual resultará en falta de comparabilidad entre los dos vectores. Con el propósito de que los caracteres del recurso sean representados más eficientemente y contribuyan al cálculo de la similitud, se necesita normalizar los vectores caracter antes de calcular la similitud semántica. El proceso de normalización puede eliminar los impactos adversos de la longitud del vector inconsistente. Y de hecho la siguiente función de peso normalizada es usada para calcular el peso de cada palabra clave.

$$w_i' = \frac{KF_i \times log\left(0.00625 + \frac{M}{m_j}\right)}{\sqrt{\sum_{j=1}^{N}\left(KF_i \times log\left(0.00625 + \frac{M}{m_j}\right)\right)^2}} \qquad 2.7$$

De esta manera, cada recurso A_i puede generar un vector de caracteres m-dimensional $(a_{i1}, a_{i2}, \dots, a_{im})$, entonces los vectores de caracteres de n recursos pueden formar un espacio $n \times m$ dimensional para el vector semántico, los cuales son combinados con la Formula 2, resultando en la siguiente matriz de vectores semánticos.

$$A = \begin{bmatrix} a_{11}w_{11}' & a_{12}w_{12}' & a_{13}w_{13}' & \cdots & a_{1m}w_{1m}' \\ a_{21}w_{21}' & a_{22}w_{22}' & a_{23}w_{23}' & \cdots & a_{2m}w_{2m}' \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1}w_{n1}' & a_{n2}w_{n2}' & a_{n3}w_{n3}' & \cdots & a_{nm}w_{nm}' \end{bmatrix} \qquad 2.8$$

2.2.2.2.2 El algoritmo de optimización de la matriz de vectores semánticos.

La debilidad más grande de VSM es que, como el ámbito de la matriz de vectores es enorme, la velocidad de cálculo de la similitud semántica es influenciada. Además, existen múltiples elementos de ruido en la matriz de vectores, los cuales interfieren la correlación de las palabras clave, así la precisión del cálculo de la similitud semántica se decrementa. Por lo que también en el presente apartado se muestra un algoritmo para optimizar la matriz VSM.

En primer lugar, hacer varias transformaciones elementales a la matriz A, calcular su rango, y excluir a todo los vectores cero de la matriz A. Después A se transforma en una nueva matriz A', y calcular todos los valores de los caracteres de la matriz A', y usar el valor anterior del caracter r más largo para hacer la transformación a la matriz A', entonces conseguir una matriz similar A_r de A', $(r < \lambda s)$. λ es el factor de regulación, el cual se usa para ajustar el valor de r, $0 < \lambda < 1$. Cuando λ toma el valor más grande, la similitud entre A_r y A es grande, pero el volumen de A_r no se decrementa efectivamente, y cuando λ toma un valor más pequeño, la similitud entre A_r y A es pequeña, pero el espacio ocupado por A_r se decrementa enormemente. Antes de que se inicie la

ejecución del algoritmo FSA-MDSM, se usa el diccionario semántico provisto por WordNet para procesar todo los elementos del vector en A_r por última vez, y excluye a los vectores semánticos repetido por la transformación de la matriz. Las palabras populares expiradas y elementos ruido también son eliminados. A través de rumbo de procesamiento anterior, se consigue la matriz optimizada final A'', la cual es mostrada a continuación.

$$A'' = \begin{bmatrix} a'_{11}w'_{11} & a'_{12}w'_{12} & 0 & a'_{14}w'_{14} & \dots 0 \dots & a'_{1m}w'_{1m} \\ a'_{21}w'_{21} & 0 & a'_{23}w'_{23} & \dots 0 \dots 0 & a'_{2m}w'_{2m} \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ a'_{r1}w'_{r1} & a'_{r2}w'_{r2} & \dots 0 \dots a'_{r8}w'_{r8} & \dots 0 \dots a'_{rm}w'_{rm} \end{bmatrix} \quad 2.9$$

El elemento 0 en la matriz A'' denota a los elementos ruido que son excluidos de la matriz A'. Entonces la matriz A'' es más compacta que la matriz A', y ocupa menos espacio de almacenamiento, lo cual es benéfico para elevar la velocidad de cálculo de la similitud semántica y reduce el costo. Cada recurso en la matriz A'' necesita m palabras clave para delegarse mejor, lo cual puede denotar la relatividad entre los recursos y entre las palabras clave con costos bajos.

2.2.2.2.3 Cálculo de la similitud semántica

Un importante contenido del algoritmo FSA-MDSM es cómo el grado de similitud entre dos vectores semánticos. Se asume que hay dos vectores semánticos A y B, las palabras clave de A (incluyendo pesos) son $a_1, a_2, \dots, a_m$, y las palabras clave de B (incluyendo pesos) son $b_1, b_2, \dots, b_m$, entonces la similitud entre dos vectores semánticos se puede expresar por varias colecciones de $S(a_i, b_j)(1 \le i \le m, 1 \le j \le n)$, y se puede conseguir una matriz de comparación semántica $m \times n$

$$Fit(A,B) = \begin{bmatrix} S(a_1,b_1) & S(a_1,b_2) & \dots & S(a_1,b_n) \\ \dots\dots\dots\dots\dots\dots\dots\dots \\ S(a_m,b_1) & S(a_m,b_2) & \dots & S(a_m,b_n) \end{bmatrix} \quad 2.10$$

En el algoritmo, se usa el coseno del ángulo entre dos vectores semánticos para calificar sus grados de similitud, y de cuerdo al contenido de $Fit(A,B)$, se deriva la siguiente fórmula para el cálculo de similitud:

$$\cos\phi = \frac{\sum_{i=1,j=1}^{m,n} a_i b_j}{\sqrt{\sum_{i=1}^{m} a_i^2}\sqrt{\sum_{j=1}^{n} b_j^2}} \quad 2.11$$

En la Fórmula 7, a_i y b_i son elementos de los vectores semánticos A y B respectivamente, y cada vector semántico está compuesto por n elementos. Se puede calcular el grado de similitud de dos vectores a través una comparación de interpolación uno a uno. $\cos\phi \in [0,1]$, cuando $\cos\phi = 0$, los dos vectores son completamente ortogonales, lo cual quiere decir que no hay similitud entre los dos vectores; cuando $\cos\phi = 1$, los dos vectores están completamente superpuestos uno del otro, lo cual quiere decir que están completamente emparejados. Usualmente, cuando $\cos\phi$ es mayor a

cierto umbral, se considera que el vector A y B coinciden, lo cual cumple con los requerimientos del algoritmo de búsquedas difusas y compensa los flujos de búsqueda de la red P2P.

2.2.3. Enrutamiento semántico para búsquedas efectivas en bibliotecas digitales distribuidas [34]

2.2.3.1 Introducción

En años recientes, la constante integración y las mejoras en recursos computacionales y de telecomunicaciones, junto con el considerable derrame en costos de digitalización, han fomentado el desarrollo de sistemas, los cuales son capaces de electrónicamente de almacenar, acceder y difundir a través de la Web, una gran cantidad de documentos digitales y datos multimedia. En tal mar de información electrónica, el usuario puede fácilmente perderse en su esfuerzo por encontrar la información que requiere. Por estas razones, el concepto de *Biblioteca Digital (BD)* se ha convertido en algo crucial: exactamente como una biblioteca física, una BD contiene una colección de documentos que está a la disposición de los usuarios. Las BDs más avanzadas que están disponibles en la actualidad tienen las siguientes características (entre otras): (i) los documentos no están limitados a sólo texto, pero también son comúnmente más expresados en formatos semi-estructurados, como XML; (ii) provienen de diferentes fuentes, usualmente disponibles en la Web, y son *heterogéneos* con respecto a las estructuras adoptadas usadas para su representación, pero relacionadas por los contenidos de los que tratan; (iii) la arquitectura señalada es más a menudo *distribuida* sobre un número de nodos (peers), cada uno, por ejemplo, administrando colecciones de documentos específicos.

Junto con los propios documentos, una buena generación siguiente de BD debería ofrecer un conjunto entero de sistemas y servicios designados para ayudar a usuarios a encontrar fácilmente la información que están buscando y accederla. De hecho, consultar y acceder a información de una BD de manera distribuida y heterogénea de manera efectiva y eficiente, requiere de concebir una completa serie de técnicas en varias áreas sinérgicas. Por ejemplo, consideremos la Figura 2.10 para mostrar un escenario de una porción de una BD distribuida conteniendo datos sobre aplicaciones. Cada nodo (peer) que compone la red BD (DL peer en la figura 2.10) está enriquecido con un esquema que representa el dominio de intereses del nodo peer, y los mapeos semánticos (representados por líneas gruesas grises), están localmente establecidos entre los esquemas de los nodos peer [35], [36], [37]. Con el propósito de hacer una consulta, un nodo peer en la BD usa su propio esquema para la formulación de la consulta, y sus mapeos son usados para reformular la consulta sobre sus vecinos inmediatos, y así sucesivamente. Por consiguiente, las respuestas a consultas pueden venir de cualquier nodo peer en la BD que esté conectado a través de un camino semántico de mapeos [38]. En tal configuración, responder efectivamente a una consulta significa propagarla hacia los nodos peer, los cuales sean semánticamente mejor adecuados para responder, las necesidades de los usuarios. Sin embargo, no es siempre conveniente que un nodo peer

propague una consulta hacia todos los otros nodos peer. En particular, una consulta hecha a algún nodo peer BD debería ser reenviada a los nodos peer más relevantes que ofrezcan resultados semánticamente relacionados entre sus vecinos inmediatos, y así sucesivamente.

En el trabajo del presente apartado, los autores se concentran en concebir técnicas que permitan un enrutamiento efectivo de consultas en un ambiente distribuido, las cuales creen que pueden ser de suma importancia para proveer consultas en la siguiente generación de BDs, identificando los documentos más relevantes (y porciones de documentos) en la red.

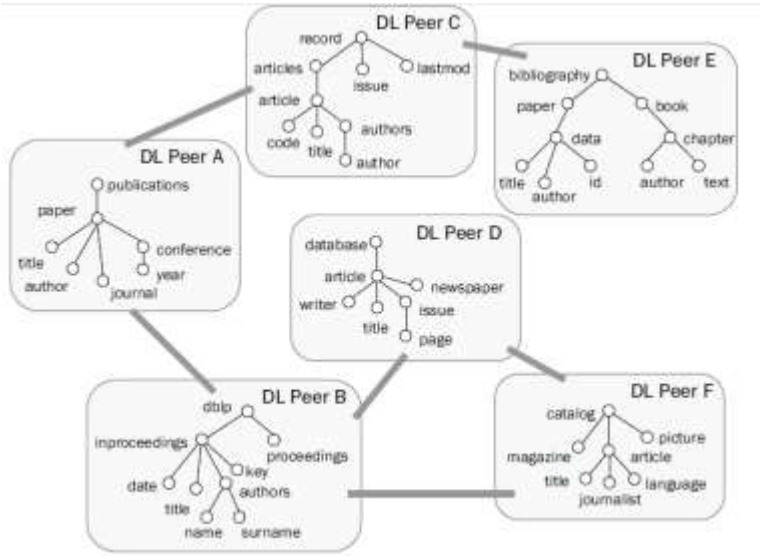


Figura 2.10. Ejemplo de una pequeña bibliografía distribuida de una BD.

2.2.3.2 Enrutamiento semántico por mapeos

Para llevar a cabo el enrutamiento semántico, los autores se basan en la noción de las subredes como en [40], y proponen el mecanismo de *enrutamiento semántico por mapeo*, donde la elección de los mejores nodos peer respondedores en la BD está basada en la información semántica acerca del contenido de los nodos peer. Se depende de los *mapeos semánticos* (originalmente descrito en [41] para un ambiente heterogéneo centralizado) los cuales los nodos peer establecen entre su esquema y la de sus vecinos, al llevar a cabo operaciones de emparejamiento de esquemas opuestos. Por medio de estos mapeos, cada concepto del esquema del nodo peer está asociado a los conceptos más similares de los esquemas de los vecinos y cada una de estas asociaciones está caracterizada por una calificación numérica, perteneciente al intervalo $[0,1]$ y que cuantifica el nivel de aproximación semántica al moverse del primer al segundo concepto. En el escenario se considera que, una consulta originada desde un nodo peer dado está siempre expresada en términos de su esquema de referencia. Si el enrutamiento estuviera limitado al conocimiento semántico que cada

nodo peer tiene en su vecindario, cada consulta que llega a un nodo peer sería reenviada a los vecinos que tienen las calificaciones más altas para los conceptos requeridos, debido a que estos nodos peer tienen la probabilidad más alta para producir resultados correctos.

2.2.3.2.1 *De calificaciones de mapeos a información resumida*

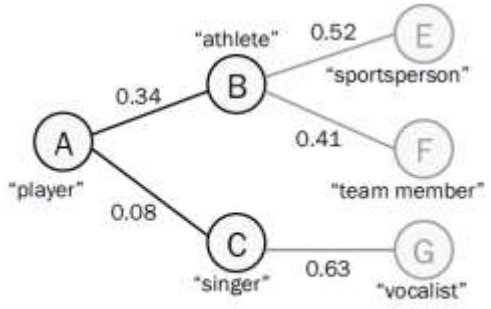
Un buen mecanismo de enrutamiento no debería estar limitado sólo a la explotación de información acerca de sus vecinos. De hecho, en la selección de vecinos, cada nodo peer también debería considerar la capacidad de aproximación de los nodos peer pertenecientes a las subredes enrutados por sus vecinos, tanto como la consulta sea probablemente propagada a esas subredes. Idealmente, sería deseable para cada nodo peer, calcular el mapeo semántico con cada otro peer de la BD, tal que esta información podría ser explotada en el proceso de enrutamiento. Sin embargo, una propuesta de esta clase es claramente no aplicable a un contexto de BD distribuida de la vida real, debido a la cantidad excesiva de datos que se almacena a causa del potencialmente muy grande número de nodos peer.

En la propuesta del presente apartado, cada nodo peer crea y mantiene información acumulativa, sintetizando las capacidades de aproximación de las subredes completas enrutadas por cada uno de sus vecinos. Esta información resumida, es calculada por cada nodo peer al combinar apropiadamente las calificaciones de mapeos semánticos hacia sus vecinos con la información resumida que cada vecino tiene a cerca de su propia subred. Una vez que se ha computarizado dicha información de la misma manera, se consigue que el conocimiento acerca de los mapeos sea propagado a lo largo de toda la BD, y cada nodo peer pueda aprender acerca de todos los otros nodos peer sin estar directamente conectados o interactuando entre ellos. Además, con el propósito de evitar la presencia de caminos cíclicos en la propagación de actualizaciones, cuando un nodo peer se conecta a la red posiblemente se puede adoptar un mecanismo para la detección de ciclos basado en identificadores globales únicos [40]. Para obtener la información acumulativa, se aplican dos diferentes tipos de operaciones, llamadas *agregación* y *composición*, a las calificaciones de los mapeos originales. Tales operaciones se muestran a través del siguiente ejemplo. Consideramos el escenario de la BD de la figura 2.11-a. El nodo peer B está conectado al nodo peer E y F además del nodo A, y entonces la calificación para el mapeo que asocia a los conceptos “player” y “athlete” debe ser revisado considerando la subred de B. Para lograr este fin, el nodo peer A calcula su calificación semántica hacia B, al *componer* la calificación de similitud entre “player” y “athlete” (i.e. 0.34) con una calificación obtenida a partir del punto B, indicando qué tan bien se puede aproximar al concepto “athlete” en la subred incluyendo el nodo peer E y F. Esta última calificación es calculada por el nodo peer B al *agregar* las calificaciones caracterizando sus mapeos para el concepto “athlete” hacia los vecinos E y F. Específicamente, estos mapeos involucran conceptos como “sportperson” (nodo E) y “team member” (nodo F) con dos calificaciones 0.52 y 0.41, respectivamente. El nodo peer B envía el resultado acumulado hacia A, el cual lo compone con su calificación del mapeo “player”-“athlete” y, obtiene una calificación final expresando qué tan bien el concepto “player” puede ser semánticamente aproximado por la subred enrutada en el nodo peer B. Similarmente, la calificación para “player” (nodo A) hacia el nodo C

debe ser calculada considerando la subred del nodo peer C, i.e., el nodo peer G. Entonces, la calificación para “player” hacia el nodo peer C, es calculada por el nodo peer A, al *componer* la calificación de similitud entre “player” y “singer” (0.08) con la acumulación de las calificaciones, caracterizando los mapeos del nodo peer C hacia sus vecinos (i.e. sólo el nodo peer G), que corresponde sólo a la calificación 0.63.

2.2.3.2.2 Índices de enrutamiento semántico

Para mantener la información acerca de las calificaciones de mapeos, cada nodo peer BD posee una estructura de datos llamada Índice de Enrutamiento Semántico (IES). El índice está representado por una matriz y, por cada nodo peer del sistema, las filas están asociadas a los vecinos del nodo, mientras que las columnas se refieren a los conceptos de su esquema.



(a) Mapeos semánticos

	sport	team	coach	player
B	0.19	0.44	0.32	0.28
C	0.02	0.08	0	0.05

(b) IES del nodo peer A

Figura 2.11. Ejemplo de enrutamiento semántico.

Tal estructura de datos se ejemplifica en la figura 2.11-b para el nodo peer A, de quien se supone el esquema incluye sólo cuatro conceptos: “sport”, “team”, “coach” y “player”. Como se puede ver, las columnas de la matriz están asociadas a los conceptos del esquema del nodo peer A, mientras que sus renglones están asociados a los vecinos de A.

La idea es que cada celda de la matriz almacene una calificación representando qué tanto se aproxima semánticamente un concepto asociado a esa columna, por la subred enrutada por el vecino asociado a un renglón dado.

Las calificaciones almacenadas en los índices son calculadas por los nodos involucrados en una manera incremental, en la base de las conexiones peer hacia el sistema y siguiendo su desarrollo. Específicamente, cuando se ingresa al sistema, cada nodo peer solicita a su vecindario, información de sus mapeos previos y puede consecuentemente calcular sus calificaciones llevando a cabo las operaciones de agregación y composición. La secuencia de operaciones que se llevan a cabo se específica por el protocolo detallado en [39].

2.3 Redes Centradas en Información (*Information Centric Networking*)

2.3.1 Introducción

La creciente demanda de distribución eficiente y altamente escalable de contenido ha motivado el desarrollo de arquitecturas de Internet del futuro basadas en objetos de datos nombrados (named data objects, NDOs), por ejemplo, páginas web, videos, documentos, u otras piezas de información. La propuesta de estas arquitecturas es llamada comúnmente red centrada en información (ICN). En contraste, las redes actuales están centradas en servidor donde la comunicación está basada en servidores nombrados, por ejemplo, servidores web, PCs, laptopts, dispositivos móviles, y otros dispositivos.

Las arquitecturas de ICN hacen uso de almacenamiento en la red para memoria caché, comunicación múltiple a través de la replicación, y modelos de interacción que dividen emisores y receptores. La meta común es conseguir distribución de contenido confiable y eficiente al proveer una plataforma general para servicios de comunicación que en la actualidad sólo están disponibles en sistemas dedicados tales como redes peer to peer (P2P) y redes de distribución de contenido registrado.

En el actual apartado se presentan los siguientes enfoques:

- Data-Oriented Network Architecture (DONA) [42]
- Content-Centric Networking (CCN) [43], actualmente dentro del proyecto Named Data Networking (NDN).
- Publish-Subscribe Internet Routing Paradigm (PSIRP) [44]
- Network of Information (NetInf) del proyecto de diseño para la Internet del (4WARD) [45].

El propósito de las propuestas anteriores es desarrollar una arquitectura de red que esté mejor adecuada para acceder y distribuir eficientemente al contenido y que haga frente a desconexiones, perturbaciones, y efectos de aumentos de acceso en el servicio de comunicación. La comunicación es manejada por los receptores que están solicitando los NDOs. Los emisores ponen los NDOs a disposición de los receptores al publicar los objetos.

Como se ilustra en la figura 2.12, la red puede satisfacer las solicitudes de clientes con datos desde cualquier fuente que posea una copia del objeto, habilitando memoria cache eficiente e independiente de la aplicación como parte del servicio de la red. La integridad de los datos entregados es establecida independiente del host que está entregándolos, lo cual podría no ser confiable.

En el análisis hecho en este apartado a las propuestas anteriores se usan los términos *información*, *contenido*, y *datos* de manera indistinta.

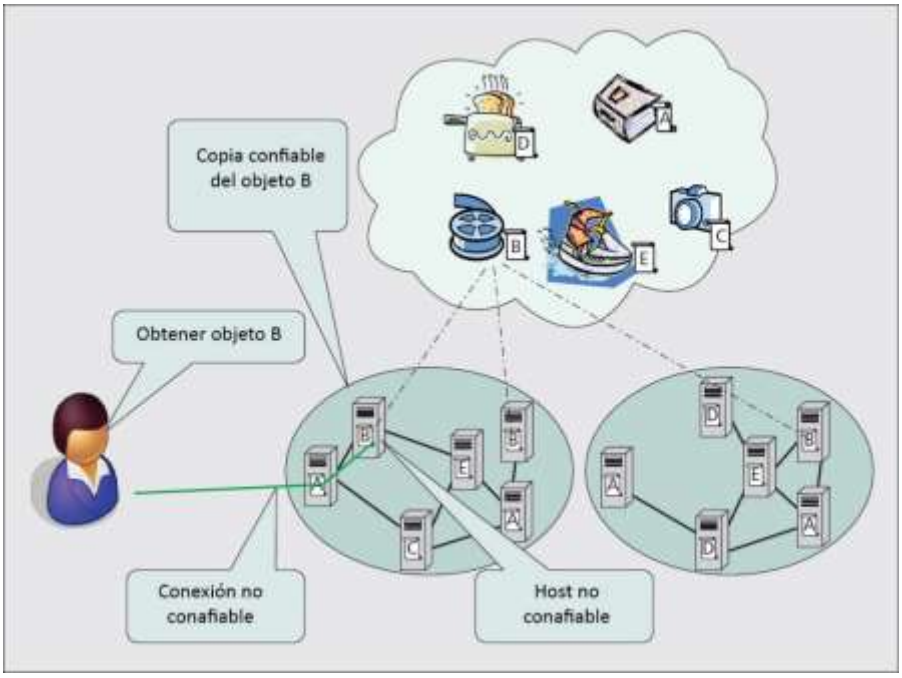


Figura 2.12. Modelo de comunicación de ICN: lado del cliente.

2.3.2 Propuesta de red centrada en información

En esta sección se describen los principales componentes comunes a los diseños más propuestos en sus opciones de diseño respectivas, y se explica las ventajas que motivan al desarrollo de la propuesta de ICN.

2.3.2.1 Componentes principales de ICN

Objetos de datos nombrados (Named Data Objects) — La principal abstracción de ICN son los NDOs. Por ejemplo, páginas web, documentos, películas, fotos, canciones, así como streaming y medios interactivos; en otras palabras, todos los tipos de objetos que se almacenan y se acceden a través de computadoras. El NDO es independiente de localización, método de almacenamiento, aplicación, y método de transporte. También quiere decir que cualesquiera dos copias de un NDO son para todos los propósitos equivalentes; por ejemplo, cualquier nodo que posea una copia puede

suministrarla a algún solicitante. El nivel de detalle de los NDOs varía entre las propuestas en el tamaño del paquete a objetos completos.

Desde una perspectiva de red, estos objetos se pueden ver como piezas de datos nombradas sin semántica, aunque algunos diseños de ICN tienen como modelo de abstracción de información que incluye representaciones múltiples para el mismo objeto. Algunos ejemplos son diferentes tipos de codificación o diferentes grabaciones de una pieza de música. Podría haber meta datos asociados a los NDOs, tales como autor, fecha de creación, u otro dato a cerca de la información representada.

Asignación de nombres y Seguridad — Asignar nombres a objetos de datos es tan importante para ICN como lo es nombrar servidores para la Internet actual. Fundamentalmente, ICN requiere nombres únicos para NDOs individuales, ya que los nombres son usados para identificar objetos independientemente de su localización o poseedor. Es importante establecer un enlace verificable entre el objeto y su nombre (integridad del nombre-dato) tal que un receptor pueda estar seguro de que los bits recibidos representa al objeto nombrado (autenticidad del objeto). La información acerca de la procedencia de un objeto (i.e., quien lo generó o publicó) es también util asociarla con el nombre.

Existen maneras diferentes de usar nombres y criptografía para conseguir las funciones deseadas [46], y existen diferentes maneras para administrar espacios de nombres de manera que correspondiente.

Se han propuestos principalmente dos modelos para nombres: uno con un nombre de espacios jerárquico y otro plano. El modelo jerárquico tiene una estructura similar a las URLs actuales, donde la jerarquía tiene como raíz a un prefijo publicador. La jerarquía permite la agregación de información de enrutamiento, mejora la escalabilidad de los sistemas de enrutamiento. En algunos casos, los nombres son *human-readable*, lo cual la hace posible para que los humanos manualmente escriban directamente el nombre, y, para otros casos, evalúa la relación entre un nombre y lo que el usuario quiere.

El otro modelo de nombre es *self-certifying*, lo cual quiere decir que la integridad nombre-dato del objeto puede ser verificada sin la necesidad de una infraestructura de llave pública u otra entidad para primero establecer confianza en la llave. Self-certification es llevado a cabo al unir el resultado de la función hash del contenido cercano al nombre del objeto. Esto se puede hacer al empotrar directamente el resultado de la función hash del contenido en el nombre. Otra opción es una unión indirecta, la cual empotra la llave pública del publicador en el nombre y firma el resultado de la función hash del contenido con la llave secreta correspondiente. Los nombres resultantes son típicamente no jerárquicos, o planos, aunque el campo publicador provee una estructura que puede ser usada para enrutamiento.

Existen compensaciones delicadas en el diseño de para nombres en ICN que afectan al enrutamiento y seguridad. Los nombres en self-certifying no son leíbles por el humano o jerárquicos. Sin embargo, pueden proveer alguna estructura usada en la agregación, por ejemplo, la parte de un nombre correspondiente a un publicador. Sin self certification, como se mencionó anteriormente, la

infraestructura depende de un PKI (Public Key Infraestructure) para su operación, lo cual podría considerarse como una desventaja mayor.

Interfaz de programación de aplicaciones (aplication programming interface, API) – La interfaz de programación de aplicaciones de ICN (API) está definida en términos de solicitud y entrega de NDOs. La fuente/productor pone a un NDO disponible a otros al publicarlos en la red. Un cliente/consumidor solicita un NDO por su nombre (acción llamada obtener, interés, solicitar, encontrar, o subscribir). La operación anterior es en la mayoría de los diseños de ICN una operación síncrona de un tiempo. Sin embargo, algunas propuestas como PSIRP se basan en un modelo publicar/subscribir-como, donde el cliente registra una subscripción y se le notifica cuando algo esté disponible.

Ambas operación usan el nombre del objeto como parámetro principal. Adicionalmente, algunas propuestas suportan parámetros complementarios. Por ejemplo, en la propuesta CURLING [47] soportan preferencias de localización para el ámbito y filtro de una publicación o solicitud.

Enrutamiento y reenvío – Existen dos propuestas generales en enfoques de ICN para manejar enrutamiento, ambas dependen fuertemente en las propiedades del espacio de nombres del objeto, en particular, si los nombres son agregables o no. Se distingue entre dos fases de enrutamiento:

- Enrutamiento de solicitudes de NDO
- Enrutamiento de NDO de regreso al solicitante

La primera propuesta usa el servicio de resolución de nombres (NSR) que almacena enlaces de un nombre de objeto a localizadores basados en topología apuntando a las correspondientes localizaciones de almacenamiento en la red. Esta propuesta tiene tres fases de enrutamiento conceptual:

- Enrutar el mensaje de solicitud al nodo NRS responsable donde el nombre del objeto es traducido en una o múltiples direcciones fuente.
- Enrutar el mensaje de solicitud a las dirección(es) fuente
- Enrutar los datos de la(s) fuente(s) al solicitante

Todas las fases pueden potencialmente usar diferentes algoritmos de enrutamiento. Se podría usar un método de enrutamiento basado en nombres para la primera fase. En la segunda y tercera fase se podría usar enrutamiento basado en topología como el protocolo de Internet (IP).

La segunda propuesta general enruta directamente el mensaje de solicitud del solicitante a una o múltiples fuentes de datos en la red basándose en el nombre del objeto solicitado. El algoritmo de enrutamiento usado para esta propuesta depende mucho de las propiedades del espacio de nombres. Después de que la fuente haya recibido el mensaje de solicitud, los datos son enrutados de regreso al solicitante, igual que en la fase 2 de la propuesta basada en NRS.

Memoria caché – Almacenamiento para guardar NDOs es una parte integral del servicio de ICN. Todos los nodos potencialmente poseen memoria caché, incluyendo a nodos en redes de infraestructura de operador y redes de casa de usuario, así como terminales móviles. Las solicitudes de NDOs pueden ser resueltas por cualquier nodo que posea una copia en su memoria caché. ICN entonces combina la memoria cache en la red, como en P2P y otras redes superpuestas, con memoria caché dentro de la red (e.g., memorias caché transparentes a la web). La memoria caché es genérica, es decir, es independiente de la aplicación y aplica a todos los proveedores de contenido, incluyendo contenido generado por el usuario.

2.3.2.2 Ventajas de la propuesta ICN

Distribución eficiente de contenido es una de las ventajas de la propuesta ICN, pero por sí sola no es suficiente para motivar el cambio a una nueva infraestructura.

Distribución de contenido escalable y de costo eficiente – De acuerdo a predicciones recientes, el tráfico IP global se incrementará en un factor de cuatro de 2010 a 2015, aproximándose a los 80 exabytes en 2015. Específicamente, trafico global de datos móviles se espera que crezca 26 veces entre 2010 y 2015. Esto es principalmente atribuido a varias formas de video (TV, video sobre demanda [VoD], video de Internet, y P2P) que continuara siendo aproximadamente 90 por ciento del tráfico global del consumidor alrededor del 2015.
La creciente demanda de replicación y distribución masiva de grandes cantidades de recursos condujeron a dos desarrollos principales: redes P2P y redes de distribución de contenido (Content Distribution Networks, CDNs).

Ambas propuestas representan una tendencia hacia un modelo de comunicación basado en contenido: identificadores uniformes de recurso (URIs) y nombres DNS son interpretados de tal manera que permita acceder a copias de contenido almacenadas en caché en la red. Aún existe un número problemas por resolver: selección de peer P2P sub-óptimo que guía a un tráfico inter proveedor caro, y la incapacidad para efectivamente incrementar el uso de almacenamiento en la red para reducir la sobrecarga para ambos escenarios, P2P y CDN.

Mirando en la necesidad de distribución de contenido escalable y eficiente, la pregunta es: si los usuarios y agentes de usuarios están más interesados en acceder a contenido nombrado, a pesar de los localizadores del punto final, ¿existe una manera, arquitecturalmente hablando, de abordar estos requerimientos que no requieran correcciones individuales para dominios y arquitecturas específicas? ICN es el intento por resolver esta pregunta de manera afirmativa.

Asignación de nombres única y persistente – La mayoría de los URIs de contenido en la red actual son de hecho localizadores de objetos los cuales, después de la resolución DNS, exhiben la dirección IP de un servidor web que atiende solicitudes al resolver la parte local del URI. Como resultado, el enlace nombre-objeto puede ser fácilmente roto, por ejemplo, cuando un objeto es movido, el sitio

cambia de dominio, o el sitio por alguna razón es inalcanzable. Además, si las réplicas del mismo objeto son colocadas en diferentes servidores web, ellas serán accesibles usando diferentes URIs, y esencialmente aparecen como objetos diferentes al sistema.

La propuesta ICN resuelves estos problemas con la asignación de nombres única y persistente de NDOs, y con su modelo de servicio que separa a los productores de los consumidores.

Modelo de seguridad – La seguridad de red actual protege el canal de comunicación entre un cliente y un servidor usando seguridad en la capa de transporte (Transport Layer Security, TLS) o una técnica similar. Este modelo de seguridad quiere que el cliente confíe en que el servidor entregue información correcta en el canal. El modelo de seguridad de ICN, en contraste, provee integridad en nombre-datos y verificación de origen de los NDOs, independiente de la fuente directa. El modelo habilita memoria caché ubicua con la autenticidad e integridad del nombre-datos retenida, algo que el modelo actual no provee.

Movilidad y multihoming (conexión a más de una red) – La naturaleza basada en host de las redes actuales generan que la movilidad y multihoming de los nodos y redes produzca un problema para administrar conexiones end-to-end y elegir cual camino o interfaz usar para esas conexiones.
La propuesta ICN no tiene conexiones end-to-end que requieran esta clase de administración de conexión. Entonces el problema se convierte mucho más simple. Un cliente que está moviéndose sólo continúa emitiendo solicitudes de NDOs en un nuevo acceso. Las solicitudes en el nuevo acceso son potencialmente atendidas desde una fuente diferente, en lugar de necesitar mantener una conexión hacia la fuente previa. Un cliente con múltiples conexiones puede similarmente elegir enviar una solicitud a cualquier, varios, todos los accesos.

Tolerante a perturbaciones – Comunicación end-to-end con sesiones de transporte a servidores origen a menudo es difícil de conseguir en redes con conectividad dispersa, movilidad alta, y alteraciones. Cuando las sesiones del protocolo de aplicación son limitadas a sesiones de transporte, ellas fallaran tan rápido como las sesiones de transporte fallen.
Muchas aplicaciones no requieren comunicación continua con caminos end-to-end. Si el objetivo primario es acceder a los objetos de datos, ICN con su memoria caché en la red puede ofrecer propuestas guardar-y-reenviar similar a la arquitectura de red de transportes de datos (Data Transport Networking, DTN) [48] con su concepto de capa de convergencia para transporte hop-by-hop. Esto puede proveer mejor confiabilidad y mejore rendimiento al hacer uso de transporte hop-by-hop optimizado y memoria caché en la red.

2.3.3 Arquitecturas de red centradas en información

En esta sección se introduce y se ilustra cuatro propuestas ICN en un nivel alto con el propósito de proveer un entendimiento general de ellas antes de entrar en a detalle.

2.3.3.1 Arquitectura de Red Orientada a Datos

En la arquitectura de red orientada a datos (Data-Oriented Network Architecture, DONA), los NDOs son publicados en la red por las fuentes. Los nodos que están autorizados para servir datos, se registran en la infraestructura de resolución la cual consiste de manejadores de resolución (RHs). Las solicitudes son enrutadas por su nombre hacia el apropiado RH, como se ilustra en la figura 2.13, pasos 1-4. Los son enviados de regreso como respuesta, ya sea a través del camino inverso RH (pasos 5-8), habilitando la caché, o sobre una ruta más directa (paso 9). Los proveedores de contenido pueden llevar a cabo un registro tipo comodín de su representante en el RH, tal que las consultas sean dirigidas a ellos sin la necesidad de objetos específicos. Además es posible registrar los nombres de los NDOs antes de que el contenido del DNO sea cread y puesto disponible. Comandos de registro tienen tiempos de expiración. Cuan el tiempo de expiración es alcanzado, el registro necesita ser renovado. La infraestructura de resolución RH enruta las solicitudes por nombre de manera jerárquica e intenta encontrar un copia del contenido lo más cercano al cliente. El proceso anycast de resolución de nombre de DONA permite un soporte limpio para los middleboxes impuestos en la red (e.g., firewalls, proxies).

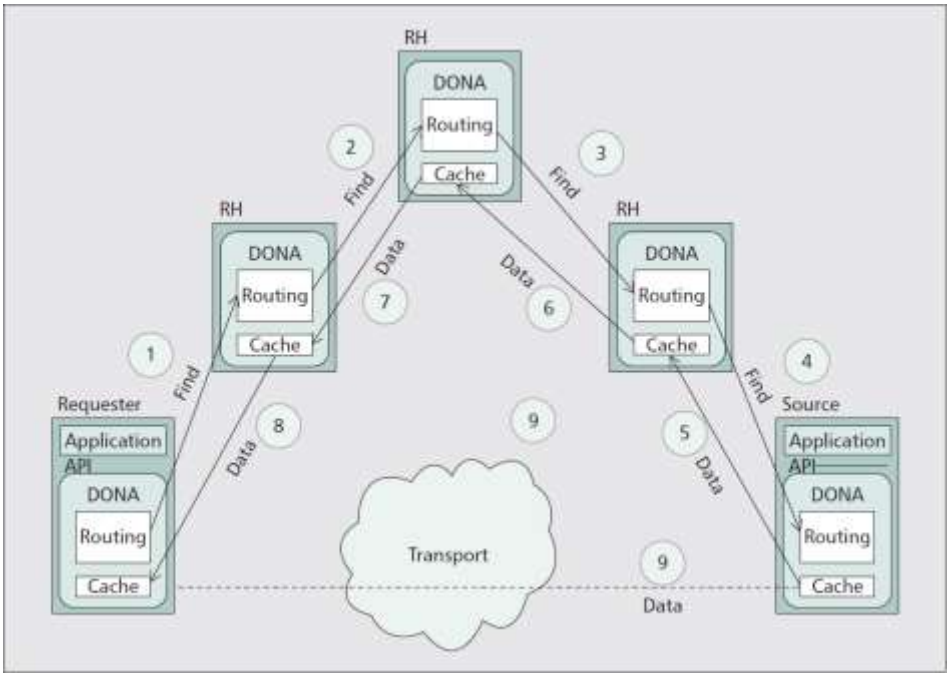


Figura 2.13. Visión de DONA cuando existe memoria caché en todos los manejadores de resolución (RHs).

2.3.3.2 Red Centrada en Contenido

En la red centrada en contenido (Content-Centric Networking, CCN), los NDOs son publicados en los nodos, y los protocolos de enrutamiento son empleados para distribuir información acerca de la localización del NDO. El enrutamiento en CCN puede hacer uso de la agregación a través de un

escenario de nombres jerárquico. La seguridad de un NDO se consigue a través de criptografía de llave pública. La confianza en las llaves puede ser establecida a través de diferentes medios, como cadenas de certificados PKI-like basados en la jerarquía de nombres, o información provista por una amigo. Los solicitudes (paquetes de interés) por un NDO son reenvidas hacia una localización de un publicador, como se ilustra en la figura 2.13, pasos 1-3. Un enrutador de CCN mantiene una tabla de interés pendiente (PIT) para solicitudes reenviadas salientes, la cual habilita agregación en la solicitud; es decir, un enrutador de CCN normalmente no reenviaría una solicitud para un NDO específico cuando se haya enviado recientemente una solicitud para ese NDO. La PIT mantiene el estado para todos los intereses y los mapea a la interfaz de red donde corresponden las solicitudes que haya recibido. Los datos son enrutados de regreso en el camino de solicitud inverso usando este estado (pasos 4-6). La red CCN tiene soporte para memoria cache en camino: los NDOs que un enrutador CCN recibe (en respuesta a solicitudes) puede ser almacenados en caché tal que las solicitudes subsecuentes recibidas para el mismo objeto puedan ser resueltas usado esa caché (como se ilustra en los pasos 7-8 de la figura 2.14). Desde una perspectiva del nodo CCN, existe un balance de solicitudes y respuestas; es decir, cada solicitud enviada es contestada por una respuesta (o no respuesta). Los nodos CCN pueden usar diferentes estrategias para las solicitudes, paso de (re) transmisión y selección de interface, dependiendo de la configuración local, rendimiento de la red observado, y otros factores.

El proyecto NDN continua con el desarrollo de la propuesta CCN. En ella se provee un escenario de nombres independiente de la topología y tiene enrutamiento explorador greedy para una mejor escalabilidad en el enrutamiento.

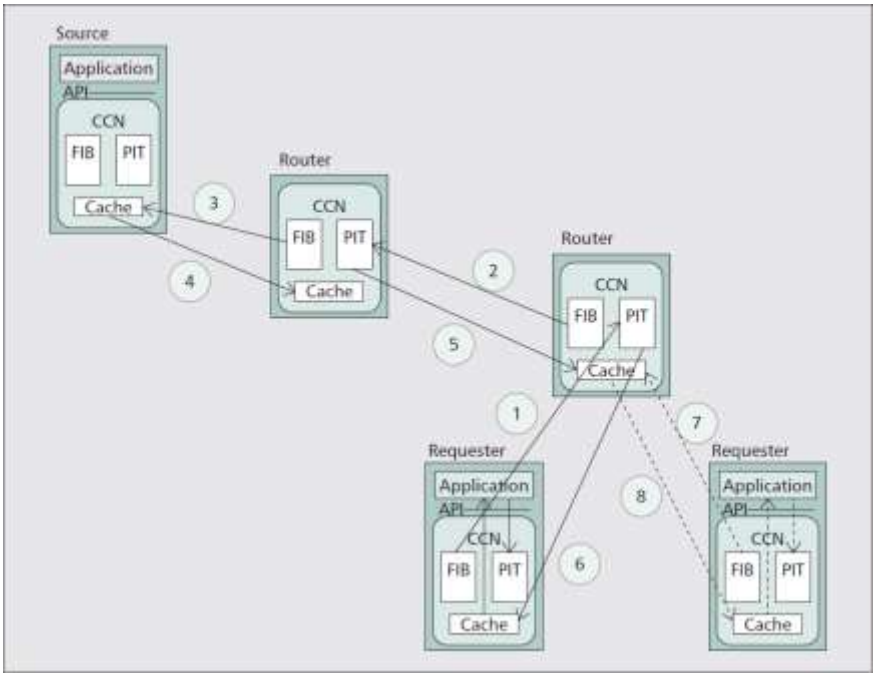


Figura 2.14. Visión general de CCN.

2.3.3.3 Paradigma de Enrutamiento de Internet Publicar-Subscribir (PSIRP)

En este paradigma, los NDOs son también publicados en la red por las fuentes de los NDOs como se ilustra en el paso 1 de la figura 2.15. La publicación pertenece a un ámbito con nombres particular. Los receptores se pueden suscribir a los NDOs (paso 2). Las publicaciones y suscripciones son emparejas por sistema rendezvous (paso 3). La solicitud de suscripción especifica el ámbito del identificador (scope identifier, SI) y el identificador rendezvous (rendezvous identifier, IR) junto con el nombre del NDO deseado. Los identificadores son entradas para un procedimiento de emparejamiento que tiene como resultado un identificador de reenvío (forwarding identifier, FI), el cual es enviado a la fuente del NDO (paso 4) tal que así pueda iniciar el reenvío de datos (pasos 5-7). El FI consiste de un filtro de Bloom que los enrutadores usan para seleccionar las interfaces sobre las cuales reenvían un NDO. Esto quiere decir que los enrutadores no necesitan mantener el estado de reenvío. El uso de filtros de Bloom producen un cierto número de falsos positivos; en este caso esto significa reenviar en algunas interfaces donde no existen receptores.

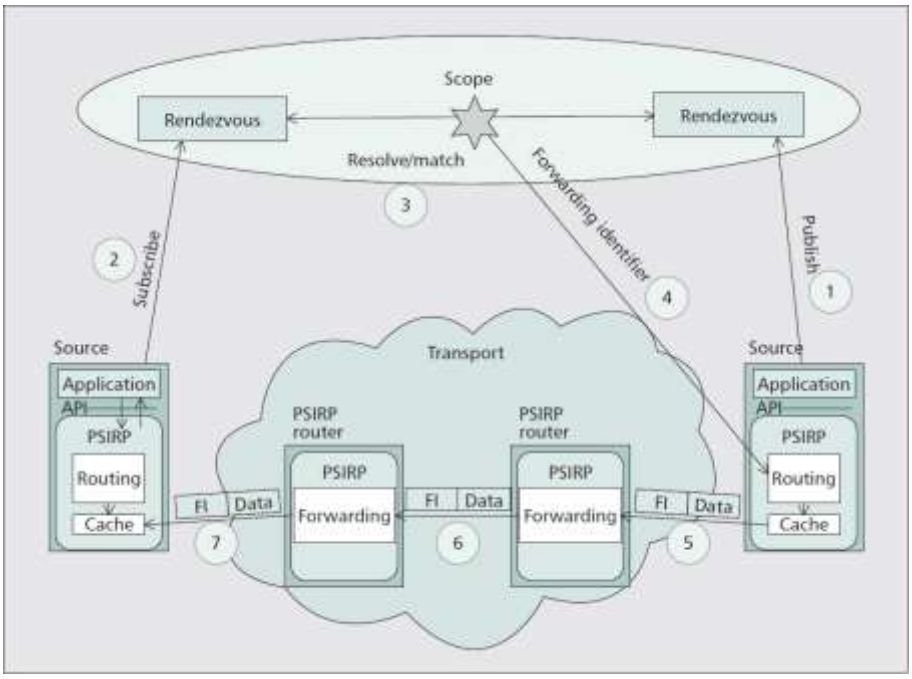


Figura 2.15. Visión general de PSIRP.

2.3.3.4 Red de Información (Network of Information, Netinf)

NetInf ofrece dos modelos para obtener NDOs, por medio de resolución de nombres y enrutamiento basado en nombres, de esta manera permitiendo la adaptación a diferentes ambientes de red. En NetInf, dependiendo del modelo usado en la red local, las fuentes publican NDOs al registrar un nombre/localizador enlazándolo con un servicio de resolución de nombres (name resolution service, NRS), o anunciando información de enrutamiento en un protocolo de enrutamiento. Un nodo de

NetInf que posea una copia de un NDO (incluyendo caché en red y terminales de usuario) puede opcionalmente registrar su copia con un NRS, agregando así un nuevo enlace al nombre/localizador. Si un NRS está disponible, un receptor puede primero resolver un nombre de NDO de un conjunto de localizadores disponibles y puede subsecuentemente obtener una copia de los datos de la mejor fuente disponible, como se ilustra en los pasos 1-4 de la figura 2.16. Alternativamente, el receptor puede enviar directamente una solicitud GET con el nombre del NDO, el cual será reenviado hacia alguna copia NDO disponible usando enrutamiento basado en nombres (paso 5-8 en la figura). Tan pronto sea alcanzada la copia, los datos serán enviados de regreso al receptor. Los dos modelos son mezclados en una propuesta hibrida enrutamiento/resolución donde un sistema global de resolución provee mapeos en la forma de indicios de enrutamiento que habilitan la agregación de información de enrutamiento.

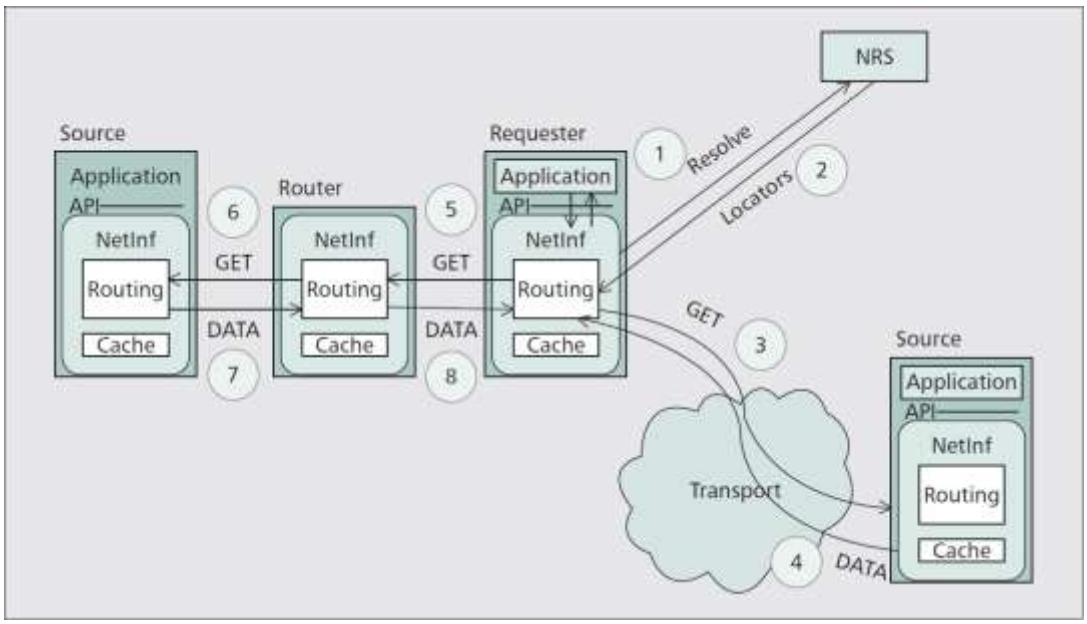


Figura 2.16. Visión general de NetInf.

2.3.4 Opciones de diseño y compensaciones

2.3.4.1 Asignación de nombres y seguridad para objetos de datos

En la propuesta DONA se asignan nombres a los NDOs con un espacio de nombres plano en la forma $P:L$, donde P es el campo principal globalmente único, el cual contiene la función hash criptográfica de la llave pública del publicador, y L es la etiqueta única del objeto. Cuando P identifica al editor (y no al dueño), que está republicando el mismo contenido por un editor diferente (e.g., por una caché en red) generalmente resulta en un nombre diferente para el mismo contenido.

El espacio de nombres en CCN es jerárquico con el propósito de conseguir mejor escalabilidad de enrutamiento a través de la agregación nombres prefijo. Los nombres tienen como raíz un prefijo único para cada editor. El prefijo del editor hace posible que los clientes construyan nombres válidos para datos que aún no existen, y los editores pueden responder con datos generados dinámicamente. Los nombres en CCN son usados para ambos propósitos información de nombres y enrutamiento. La granularidad de los nombres es muy fina: se asignan nombres a paquetes únicos.

PSIRP (Publush Subscribe Internet routing Paradigm) hace uso de dos tipos de nombres: identificadores rendezvous e identificadores de ámbito; ambos corresponden a un espacio de nombres plano. Los identificadores rendezvous (junto con los identificadores de ámbito) asignan nombres a los NDOs. Los NDOs son mapeados a puntos rendezvous, los cuales son usados para establecer contacto entre editores y suscriptores. PSIRP también usa identificadores de reenvío, los cuales son empleados por la estructura de reenvío para transportar datos después de haber establecido contacto en un punto rendezvous. Los identificadores de reenvío no son nombres para NDOs; ellos son temporales e identifican un camino del editor al suscriptor.

NetInf generalmente emplea un espacio de nombres plano [49] con alguna estructura similar al espacio de nombres de DONA. Con el propósito de adecuarse a diferentes requerimientos de desarrollo de ICN, NetInf distingue entre un formato de nombres común (que todos los nodos deben entender) y mecanismos de nombres semánticos y validación de enlace objeto-nombre.

El formato de nombres común de NetInf [50] está basado en contener resúmenes hash en el nombre, y se usan diferentes modelos de funciones hash. El resumen hash del poseedor de la llave pública (PK) también puede ser contenido en el nombre para soportar datos dinámicos. Los nombres en NetInf pueden ser transformados en diferentes representaciones, incluyendo una representación URI y una representación binaria.

DONA, NetInf y PSIRP usan espacios de nombres planos. Los tres esquemas pueden checar la integridad del nombre-dato solamente basándose en el nombre del dato (i.e., sin requerir medios externos como PKI). Esta propiedad es llamada nombres auto-certificables. Para conseguir esto para datos estáticos, la función hash criptográfica del contenido puede ser incluido como una etiqueta del objeto. Para datos dinámicos, la integridad nombre-dato es obtenida al proveer una firma de la hash del contenido como metadato con el NDO, firmado por la clave pública correspondiente al hash en el campo autenticador del ID. De esta manera, el identificador de objeto es limitado de manera segura a los datos y también permite que los datos que aún no existen sean manejados. Usar identificadores auto-certificables es una compensación intencionada entre las propiedades de integridad del nombre-dato deseables y legibilidad humana tanto como los identificadores contengan un hash criptográfica. Como resultado, potencialmente se requieren medios adicionales para enlazar de manera segura nombres legibles al humano en el nivel de aplicación a esos identificadores. Se podría argumentar que muchos de los URLs en la Internet actual también sufren del mismo problema de no ser amigables con el humano. Por otro lado, los identificadores auto-verificables permiten revisar si los datos recibidos coinciden con el identificador usado en la solicitud de datos sin requerir una PKI, los cuales simplifican el modelo de seguridad y lo hace más confiable.

Los nombres en CCC típicamente no contienen la PK del editor (o su hash criptográfica). La hash del contenido estático es también típicamente no explícitamente parte del nombre usado por los solicitantes. Mientras esto mejora la legibilidad para el humano, complica la auto-certificación. La integridad de datos es además conseguida al firmar el contenido con la llave secreta del editor, pero la confianza en la llave para firmar siempre necesita ser establecida usando medios externos ya que no existe enlace directo entre la llave y el nombre del NDO. CCN soporta múltiples diferentes maneras para verificar la confianza la llave, tales como experiencia directa, información provista por amigos, un directorio de llaves en las que se confía, o una PKI global.

2.3.4.2 Interfaz de Programación de Aplicación (API)

La API para ICN es fundamentalmente diferentes de la API de socket de TCP/IP actual. La anterior está diseñada para establecer un canal de comunicación entre dos puntos en localizaciones específicas.

La API de ICN está diseñada para permitir que las aplicaciones soliciten un NDO desde la red sin tener ningún conocimiento de la localización del objeto y la propuesta de recuperación (e.g., descarga desde múltiples fuentes).

Las llamadas a la API principal de todas las propuestas de ICN presentadas son equivalentes a una llamada de publicar y obtener. Sin embargo, estas llamadas abordan diferentes entidades subyacentes a la red. En PSIRP y NetInf, las publicaciones son abordadas por el sistema rendezvous y NRS, respectivamente, para registrar nuevos nombres, resultando en las entradas de enlace correspondientes. En CCN y DONA, la publicación es usada para llenar las tablas de enrutamiento de los enrutadores/RH de contenido. De la misma manera, en NetInf y PSIRP, las llamadas de obtención son dirigidas al sistema resolución/rendezvous, seguidas por un segundo paso para obtener los datos de la fuente NDO. Sin embargo, este segundo paso es típicamente oculto de la API, así como los localizadores e indicios de enrutamiento son típicamente no expuestos a las aplicaciones ICN. EN CCN y DONA, la llamada de obtención es manejada directamente por los enrutadores/RH. NetInf es una propuesta hibrida ya que puede enviar un mensaje de publicación/obtención al NRS para resolución de nombre así como directamente a un enrutador basado en nombres.

2.3.4.3 Resolución de nombres y enrutamiento

En ICN existen dos funciones clave que resolución de nombres y enrutamiento deben conseguir cuando hay una solicitud para un NDO específico. La primera es encontrar un nodo que tenga una copia del NDO y entregar la solicitud a ese nodo (i.e., enrutamiento de solicitudes de NDO). La segunda es encontrar un camino de regreso desde ese nodo al solicitante sobre el cual el NDO puede

ser entregado (i.e., enrutamiento de NDOs). Una manera para hacer esto es a través de resolución de nombres, lo cual quiere decir que se requiere un servicio de resolución, y se devuelve uno o más localizadores de capa más baja. Estos localizadores pueden usarse para obtener el objeto, usando un protocolo como HTTP o IP directo. Una alternativa es directamente reenviar la solicitud hacia una copia del objeto en la red basándose en el nombre del objeto sin resolver primero el nombre del objeto en algún localizador de capa más baja. Esta propuesta es referida a menudo como enrutamiento basado en nombres. Se debe notar que la resolución de nombres debería también incluir algunos pasos que involucren enrutamiento basado en nombres, como cuando se usa un sistema de resolución de nombres basado en DHT.

DONA usa enrutamiento basado en nombres para enrutar la consulta por medio de RHs hacia una copia del NDO solicitado. Los nodos que están autorizados para servir datos usan la primitiva $REGISTER(P{:}\,L)$ para registrar un dato en un RH. Cada dominio/editor tiene un RH. Para resolver un nombre, se usa la función $FIND(P{:}\,L)$. Ambas primitivas permiten usar comodines en lugar de P o L. Los RHs son organizados en una estructura jerárquica. Cada solicitud que un RH no puede manejar en reenviado a su RH padre. El RH intenta encontrar una copia del contenido más cercano al cliente. Una vez que se encuentra una copia, se regresan los datos al cliente, potencialmente a través del camino de la solicitud RH, como se muestra en la figura 2.13, cuando el RH realiza almacenamiento en caché. De otra manera, los datos se pueden regresar directamente al cliente.

CCN usa enrutamiento basado en nombres. Los clientes preguntan por un objeto de datos al enviar paquetes de interés, los cuales son enrutados hacia el editor del nombre prefijo usando coincidencia de prefijo más largo en la base de información de reenvío (FIB) de cada nodo. La FIB puede ser construida usando protocolos de enrutamiento similares a aquellos usados en la Internet actual. Los nodos CCN guardan el estado para cada solicitud saliente en la tabla de intereses pendientes (PIT, Fig. 3). Esto hace que la agregación sea posible, i.e., cuando el mismo nodo recibe múltiples solicitudes por el mismo NDO, sólo la primera es reenviada hacia la fuente. Cuando una copia del objeto de datos es encontrado en el camino, un paquete de datos conteniendo el objeto solicitado es enviado de regreso en el camino inverso al cliente (todos los nodos a lo largo del camino almacenan en caché una copia del objeto). El camino inverso es encontrado usando el estado que el paquete de interés ha dejado en los nodos.

PSIRP usa un modelo de resolución donde el que resuelve es llamado el punto *rendezvous*. El camino de regreso de datos hacia el cliente puede, potencialmente, tomar un camino diferente al comino de resolución/*rendezvous*. El punto *rendezvous* no tiene que estar en el camino hacia el editor o mantener una copia de los datos. Los datos son reenviados usando un modelo de enrutamiento de fuente llamado zFilters: un filtro de Bloom que describe a la ruta es construido por el punto *rendezvous* y usado para reenviar paquetes desde la fuente seleccionada al destino. El filtro Bloom es adjuntado al paquete, y contiene todos los nombres de los enlaces que tienen que ser seguidos. La propuesta del filtro Bloom permite que la longitud del paquete sea compensada en contra de recursos de la red desperdiciados. Un filtro Bloom más grande da menos falsos positivos, esto resulta en menos paquetes que son reenviados en enlaces sin receptor.

NetInf representa una arquitectura hibrida que soporta resolución de nombres así como enrutamiento basado en nombres para devolver objetos de datos. NetInf soporta una amplia variedad de servicios de resolución de nombres que tienen flexibilidad y escalabilidad para proveer de un amplio rango de tipos de red. El mismo mecanismo que funciona en una red ad hoc pequeña podría no funcionar/escalar en una red global núcleo. NetInf define una interface interdominio para resolución de nombres y enrutamiento que permite usar diferentes mecanismos en diferentes partes de la red. Dos mecanismos de resolución de nombres explicito se ha desarrollado hasta el momento. El primero es llamado tabla hash distribuida multinivel (*Multilevel Distributed Hash Table*, MDHT) [51]. Este método usa una jerarquía de servicios de resolución empotrados topológicamente, tablas hash potencialmente distribuidas (DHTs), para la habilitar escalabilidad y resolución consciente de localización de espacio de nombres plano. Adicionalmente, NetInf provee un propuesta NRS llamada construcción de localizador tardío (*Late Locator Construction*, LLC) [52] que se enfoca en manejar topologías de red altamente dinámicas, incluyendo redes móviles grandes. La propuesta de resolución de nombres de NetInf permite una evolución regular a partir de la Internet actual. El NRS puede resolver nombres de objeto en tradicionales URLs, las cuales se pueden recuperar usando el protocolo HTTP existente.

Comparando las alternativas, se puede notar que la propuesta basada en resolución de nombres tiene la ventaja que el caché no está limitado a las copias en el camino de reenvío. En su lugar, los NRS pueden administrar y proveer localizadores para cualquier copia de objeto disponible, incluyendo los de caché en la red y las copias almacenadas en los dispositivos de usuario que puedan ser accedidos, se podría decir, en escenarios locales. Adicionalmente, una propuesta basada en NRS puede simplificar la migración tanto que las capas inferiores de enrutamiento y reenvío no tenga que ser modificadas. Por otro lado, el enrutamiento basado en nombres elimina el paso de resolución de nombre completamente, por lo tanto reduce potencialmente la latencia general y simplifica el proceso completo. Tampoco es claro cuánta agregación se puede hacer de manera eficiente en propuesta basada en NRS.

2.3.4.4 Caché

ICN en general toma ventada del almacenamiento en la red para proveer un servicio de transporte de mejor rendimiento y más robusto. Los NDOs se pueden almacenar en caché en el camino (e.g., como se hace de manera tradicional en la web), e ICN también puede hacer que los objetos en caché estén disponibles para solicitudes fuera del camino o registrarlos en un protocolo de enrutamiento o registrarlos en un servicio de resolución de nombres. Adicionalmente a almacenar en caché los NDOs, algunas propuestas de ICN hacen agregación en solicitudes, las cuales se pueden ver como una forma de caché de solicitudes.

En DONA, el almacenamiento en caché es inherente en la arquitectura. Cualquier RH también puede servir como caché. Para poblar su caché, el RH modifica la solicitud FIND tal que el NDO sea

regresado al RH antes de que sea entregado al solicitante. Cualquier caché puede responder a una solicitud FIND al devolver una copia almacenada en caché del NDO.

CCN puede almacenar ambas solicitudes (a través de su agregación de solicitudes) y objetos. CCN enruta una solicitud de datos hacia el editor, y hace uso de cualesquiera copias en caché a lo largo del camino. Un nodo de CCN puede guardar paquetes de interés recibidos en una tabla de interés pendiente y así suprimir el reenvío de solicitudes recibidas subsecuentemente para el mismo objeto si ya se ha enviado una solicitud por él. Las copias de objeto también pueden ser encontradas por una búsqueda local. Así como los paquetes simples son objetos atómicos en CCN, es posible que solo una parte de un objeto más grande sea almacenado en caché.

En PSIRP, el almacenamiento en caché es limitado al ámbito del punto *rendezvous* para el identificador asociado al objeto. Dentro de ese ámbito un objeto puede ser almacenado en múltiples caches.

NetInf también puede almacenar en caché objetos y solicitudes. Generalmente, el enrutamiento basado en nombres y resolución de nombres es empleado para encontrar opciones para el siguiente salto. Cuando un nodo recibe una solicitud GET, puede decidir emplear una estructura de tabla de interés pendiente para agregación en la solicitud. También puede decidir realizar una búsqueda en un NRS dedicado para cada interés recibido, tal que realizar agregación en la solicitud se convierte en una política de decisión. Existen dos maneras de hacer uso de una copia de un objeto almacenada en memoria caché: primero, cualquier copia puede ser encontrada directamente al consultar al NRS, siempre y cuando la copia esté explícitamente registrada ahí o haya sido descubierta por el NRS a través de otros medios (e.g., broadcast); segundo, la copia puede ser encontrada por el protocolo de transporte de NetInf consciente de caché en el camino hacia la localización conocida de una copia.

2.3.4.5 Transporte

Al hablar de transporte se refiere a dos conceptos:

- Los mecanismos fundamentales de reenvío de solicitud y respuesta.
- Funciones de protocolo de transporte como compartición de recursos (“control de congestión”), control de flujo, y confiabilidad.

La arquitectura DONA no pone mucho énfasis en transporte y parece depender de los protocolos de transporte existentes como TCP.

CCN define diferentes tipos de paquetes, paquetes de interés y paquetes de datos, representando a los elementos básicos de un protocolo. Un nodo que envía un paquete de interés a través de una de sus interfaces (llamadas faces en CCN) hacia (un conjunto de) nodos vecinos tiene alguna expectativa de recibir un paquete de datos correspondiente en un tiempo corto (i.e., los nodos CCN

operan bajo el principio de que existe un balance de paquetes de interés y datos). Los paquetes de interés y datos trabajan en el nivel de paquetes – La suposición es que los objetos más grandes serían representados por trozos individuales, y cada trozo puede ser accedido por un nombre único.

Este mecanismo fundamental es la base de CCN para realizar diferentes servicios que son convencionalmente considerados como “funciones de la capa de transporte”, como la transmisión confiable, flujo de control, y comunicación multi-camino. Depende esencialmente de una estrategia específica de un nodo para el cual los paquetes de interés en su interface deberían ser enviados y cómo comportarse en reacción a los paquetes de datos recibidos.

El mecanismo básico de reenvío de PSIRP está basado en filtros de Bloom como se describió anteriormente. PSIRP propone usar diferentes nombres para cada objeto para manejar control de flujo. Estos nombres son derivados algorítmicamente del nombre original, codificando la velocidad de recepción deseada. Otra opción es que los receptores publiquen comentarios de control de flujo bajo algún nombre derivado algorítmicamente para que el emisor posiblemente se suscriba.

NetInf define un conjunto de mensajes para solicitar recursos y responder esas solicitudes, por ejemplo, al devolver el objeto solicitado, o regresar un localizador o un indicio de redirección. Este protocolo es implementado por capas de convergencia; es decir, líneas concretas del protocolo para capa inferiores específicas. Podrían existir múltiples saltos involucrados en el reenvío como los mensajes de solicitud y respuesta, y cada salto puede potencialmente usar una capa de convergencia diferente.

La capa de convergencia usada en este esquema salto a salto implementa (o emplea) un protocolo de transporte que provee los mecanismos apropiados de compartición de recursos y confiabilidad para el camino de red correspondiente (segmento).

2.4.4.6 Movilidad

En esta sección se discuten tres tipos de movilidad y cómo se relacionan con redes centradas en información. El primer tipo es movilidad del cliente: un cliente se mueve durante o entre la solicitud de objetos de datos. El segundo tipo es movilidad del contenido: un objeto o conjunto de objetos cambia de localización. El tercer tipo es movilidad de red: cuando una red entera se mueve (e.g., una red de área de cuerpo o una red en un tren).

Una característica clave de redes centradas en información es que todas las copias son iguales. Para movilidad de cliente esto significa que cuando un cliente se mueve no hay necesidad de guardar una asociación hacia una copia específica viva. En su lugar, se pueden establecer nuevas asociaciones para copias alternativas cercanas a la nueva localización. Todas las propuestas de ICN discutidas pueden encontrar fácilmente una nueva copia localizada apropiadamente cuando un cliente se mueve.

Cuando el contenido (el editor) se mueve, la información de enrutamiento en la red necesita ser actualizada. Para propuestas basadas en NRS como NetInf, esto significa que se registra un nuevo localizador cuando un NDO es publicado con una nueva localización en la red. Para propuestas como CCN que usan enrutamiento basado en nombres y asignación de nombres jerárquica para agregar anuncios de red, la situación es más problemática. Si el agregado completo de los objetos está en movimiento, le anuncio de la nueva ruta necesita ser propagada y los registros viejos de enrutamiento necesitan ser reemplazados antes de que el enrutamiento converja, causando problemas similares como las vemos en redes IP actuales. En el caso cuando sólo partes de los objetos pertenecientes a un agregado se mueven hacia una nueva localización (e.g., un empleado de una compañía toma una laptop en un viaje), habrá también una necesidad de fragmentar las tablas de enrutamiento. En escenarios de red flexibles, esto podría vencer los beneficios de tener un espacio de nombres jerárquico. En PSIRP y DONA, la movilidad del contenido involucra actualizar el estado de enrutamiento en los nodos rendezvous y RH, respectivamente. Sin embargo, ambos no sufren del problema de agregación gracias a su espacio de nombres plano.

Mover una red entera puede causar una tormenta de actualización de enrutamiento/resolución, especialmente si la red que se está moviendo consiste de un conjunto heterogéneo de editores, como en un tren. Esto puede ser problemático para esquemas de enrutamiento basado en nombres así como propuestas basadas en NRS si ellas no permiten anuncios de rutas relativas o localizadores relativos. Por ejemplo, si los editores pueden expresar su localización como relativa a la localización de la red en movimiento, sólo la localización de la red necesita ser actualizada cuando se mueva, no todas las localizaciones de los objetos actualmente adjuntos a la red en movimiento. Como ejemplo de un sistema de enrutamiento/reenvío que soporte localizadores relativos es el sistema de resolución LLC de NetInf [52].

NetInf puede soportar los tres tipos de movilidad. Movilidad del proveedor de contenido/contenido es soportada a través del NRS. Cuando una copia de datos se mueve, este movimiento resulta en una actualización en el NRS para responder a la nueva localización de la red. Las actualizaciones de NRS son un estándar de operación en NetInf, lo cual puede ser llevado a cabo rápido y no resulta en tablas de búsqueda infladas. Para manejar una movilidad pesada del cliente depende de las tecnologías de transporte y reenvío. Por ejemplo, el NRS integrado y el sistema de enrutamiento/reenvío Global Information Network (GIN) [53] que nativamente soporta la movilidad del cliente sin aumentar las tablas de enrutamiento del cliente. La alternativa de NetInf el sistema de enrutamiento/reenvío LLC provee muy buen soporte para movilidad de red.

En PSIRP, los cliente sólo pueden cancelar su suscripción, cambiar de red, y volver a suscribirse. Un subárbol/camino nuevo será calculado por la capa de enrutamiento. Se permite uso de buffer y números secuenciales para la entrega continua. La movilidad en el proveedor de contenido es más compleja e involucra actualizado el estado de enrutamiento en los nodos *rendezvous*.

La movilidad del cliente en CCN es inherente. Un cliente puede cambiar a otra red y continuar expidiendo paquetes de interés. La capa de estrategia podría darse cuenta del cambio y re expedir todos los intereses pendientes, sin esperar por ellos. La movilidad del proveedor de contenido es

más compleja: un proveedor de contenido tendría que actualizar las tablas de enrutamiento de todos los nodos vecinos relevantes. Además, los proveedores de contenido en movimiento contaminan las tablas de enrutamiento con prefijos específicos, contrarrestando las ventajas de la agregación de prefijos.

2.4 Principios de obtención de texto basados en funciones hash

2.4.1. Introducción

Este apartado contribuye en el aspecto de búsquedas de similitud las cuales reciben más atención en obtención de información: el uso de funciones hash para acelerar las búsquedas de similitud. El paradigma de búsqueda basada en funciones hash se ha aplicado con gran éxito para las siguientes tareas:

- Detección de duplicados. Dado un (muy grande) corpus D, encontrar todos los documentos los cuales su similitud es cercana a uno [64, 65].
- Análisis de falsificación. Dado un documento candidato *d* y un (muy grande) corpus D, encontrar todos los documentos en *D* que contengan pasajes cercanamente idénticos a *d*.

Desde el punto de vista de obtención, obtención de texto basados en funciones hash es una tecnología incompleta. Claves hash idénticas no implican alta similitud pero indican una alta probabilidad de alta similitud. Este hecho sugiere una estrategia de solución para las tareas mencionadas: En el primer punto se construye un conjunto candidato $D_q \subset D, |D_q| \ll |D|$, por un método de obtención de texto basado en función hash; en el segundo punto D_q se investiga profundamente por un método completo.

El marco completo de obtención puede ser formalizado como sigue: Dado un conjunto $D = \{d_1, \dots, d_n\}$ de documentos cada uno de los cuales es descrito como un vector característico m-dimensional, $x \in R^m$, y una métrica de similitud, $\varphi : R^m \times R^m \to [0;1]$, con 0 y 1 indicando ninguna y máxima similitud respectivamente. φ podría depender de la norma l_2 o del ángulo entre dos vectores característicos. Para una consulta al documento d_q, representado por el vector característico X_{d_q}, y un umbral de similitud $\theta \in [0;1]$, es decir, se está interesado en los documentos del θ-vecindario $D_q \subseteq D$ de d_q, el cual es definido por la siguiente condición:

$$d \in D_q \iff \varphi\left(X_{d_q}, X_d\right)$$

Donde X_d denota al vector característico de *d*. Dentro de aplicaciones de obtención de información, los documentos son representados como vectores de términos con dimensión alta con $m > 10^4$, típicamente bajo el modelo del espacio vectorial. Se distingue entre los documentos reales, $d \in D$, y sus representaciones como vectores característicos, ya que uno y el mismo documento podría ser analizado bajo diferentes modelos, diferentes representaciones y diferentes métricas de similitud.

En aplicaciones con dimensiones bajas, es decir, $m < 10$, el problema de obtención puede ser resuelto eficientemente con métodos de particionamiento de espacio como archivos de malla, arboles-KD, o arboles-quad, así como con árboles indexados que dividen en partes a los datos como lo hacen los arboles-R, arboles-Rf o arboles-X. Para un *m* significativamente grande la construcción de D_q no puede ser mejor hecha que por un escaneo lineal en $O(|D|)$[69]. Sin embargo, si uno acepta un decremento en el *recall*, la búsqueda pude ser dramáticamente acelerada con funciones hash de similitud. La efectividad de aplicar funciones hash de similitud resulta del hecho de que el *recall* es controlado en términos del umbral θ de similitud para una métrica de similitud φ dada.

Para impulsar las ideas resaltadas vistas, se considera una representación de documento *m*-dimensional bajo el modelo del espacio vectorial con un escenario con pesos-*tf*. Una muy simple función hash de similitud h_φ, $h_\varphi : \{X_1, \dots, X_n\} \rightarrow N$, podría mapear cada término del vector *x* en un número único, $h_\varphi(X)$ totaliza el número de estos términos en *x* que inician con la letra "a". Si $h_\varphi(X_{d1}) = h_\varphi(X_{d2})$ se asume que d_1 y d_2 son similares.

Aunque el ejemplo es simple, ilustra el principio y los problemas de búsquedas de similitud basadas en funciones hash:

- Si h_φ es demasiada genérica afirmará que dos documentos bastante no similares son similares, es decir, regresara una gran cantidad de falsos positivos.
- Si h_φ es demasiada específica el entendimiento de similitud se volverá demasiado estrecho.

Si h_φ es diseñada con determinación y captura lo esencial de los vectores característicos, las solicitudes de búsqueda pueden ser resueltas casi en tiempo constante, independientemente de la dimensión de *x*.

2.4.1.1 Funciones hash perfectamente sensibles a la similitud

Primero, se quiere puntualizar que la búsqueda de similitud basada en función hash es un método de división del espacio. Segundo, es interesante notar que, al menos en teoría, se puede establecer un espacio perfecto de partes para búsquedas basadas en funciones hash, para un conjunto de documentos *D* y un umbral de similitud θ. Para hacer esto creible, la búsqueda de similitud basada en función hash se ha formulado como problema de cubrimiento de conjunto. Esta vista genérica difiere de las descripciones centradas en computación encontrada en la literatura relevante.

Se considera para este propósito dividir en partes el R^m dentro de regiones superpuestas tal que la similitud de dos puntos en la misma región es superior a θ, donde cada región es caracterizada por una única clave $k \in N$. Además, se considera una función hash que arroja múltiples valores, $h_\varphi^*: R^m \rightarrow P(N)$, la cual es perfectamente sensible a la similitud con respecto al umbral θ. $\forall d_1, d_2 \in D$:

$$\underbrace{\left(h_\varphi^*(X_{d_1}) \cap h_\varphi^*(X_{d_2})\right) \neq \emptyset}_{\alpha} \quad \Leftrightarrow \quad \underbrace{\varphi(X_{d_1}, X_{d_2}) > \theta}_{\beta} \tag{2.4.1}$$

Razones y utilización. h_φ^* asigna a cada vector característico X_d un conjunto pertenencia $N_d \in P(N)$ de la región de claves, mientras que dos conjuntos, N_{d_1}, N_{d_2}, comparten una clave si X_{d_1} y X_{d_2} tienen una región en común. La figura 2.17 sirve para ilustrar el concepto anterior.

Basados en h_φ^* se puede organizar los mapeos entre todas las regiones de claves $K, K := \bigcup_{d \in D} N_D$, y los documentos con la misma región clave, como tablas hash $h, h := K \rightarrow P(D)$. Basándose en el θ-vecindario se puede construir D_q de d_q con tiempo de ejecución $O(|D_q|)$:

$$D_q = \bigcup_{k \in h_\varphi^*(x_{d_q})} h(k) \tag{2.4.2}$$

Observe que h_φ^* lleva a cabo ambas; *precision* y *recall* perfecta. Para un conjunto D el cual es completamente conocido e invariante en tiempo tal que se pueda encontrar una función. Sin embargo, en la mayoría de los casos la relación de equivalencia de la ecuación (2.4.1), $\alpha \Leftrightarrow \beta$, no puede ser garantizada:

- $\not\Rightarrow$ Si β no es una conclusión de α, D_q contiene documentos que no pertenecen al θ-vecindario de d_q: la precisión es < 1.
- $\not\Rightarrow$ Si α no es una conclusión de β, D_q no contiene todos los documentos del θ-vecindario de d_q: el recall es < 1.

2.4.2. Métodos de búsqueda basados en funciones hash

A pesar del uso de sofisticadas estructuras de datos la búsqueda del vecino más cercano en *D* se degrada a una búsqueda lineal si la dimensión de los vectores característicos es de alrededor de 10 o más grande. Si se sacrifica exactitud, es decir, si se aceptan valores debajo de 1 para *precision* y *recall*, se puede evitar el cuello de botella en tiempo de ejecución al usar métodos de búsqueda basados en funciones hash. Éstas, son técnicas específicamente diseñadas para aproximar búsquedas del vecino (más) cercano dentro de tiempo de ejecución sublineal en la recopilación de tamaño $|D|$.

Hasta el momento sólo se han desarrollado pocos métodos de búsqueda basados en funciones hash, en particular proyección aleatoria (random projection), funciones hash sensibles a localidad (locality-sensitive hashing), y huellas difusas (fuzzy-fingerprinting).

Los métodos de búsqueda basados en funciones hash operan maneras para empotrar vectores con dimensiones altas en un espacio con menos dimensiones. El propósito buscado es la reducción en dimensión mientras se retenga tanta información de similitud como sea posible. En obtención de información se ha desarrollado tecnología de empotramiento para descubrir estructuras semánticas ocultas: la representación de un término con dimensión alta de un documento es empotrada en un espacio de concepto con dimensión baja. Algunas técnicas de transformación conocidas incluyen indexado semántico latente y sus variantes, análisis semántico latente probabilístico, re-escalamiento residual iterativo, y análisis de componente principal. El concepto representación proveerá un mejor *recall* en términos de expansión semántica de consultas proyectadas dentro del espacio del concepto.

El empotramiento es un paso vital dentro de la construcción de una función hash de similitud h_φ. Desafortunadamente, la tecnología de empotramiento semántica no se puede aplicar en esta conexión, la cual está basada en la naturaleza del caso de uso. Las búsquedas basadas en funciones hash se enfocan en lo que se llama aquí situaciones de obtención “abiertas”, mientras que el empotramiento semántico implica una situación de obtención “cerrada” o “semi-cerrada”.

Esta distinción está relacionada con el conocimiento que es compilado dentro de la función de obtención $\rho : \mathrm{Q} \times \mathrm{D} \rightarrow \mathrm{R}$, donde Q y D designan a la computadora representaciones de las consultas y documentos de los conjuntos *Q* y *D* respectivamente.

- *Situación de obtención abierta*. *Q* y *D* son desconocidos inicialmente. ρ depende del lenguaje de conceptos genérico tal como distribución de términos, frecuencia de términos, o longitud de la oración. Un ejemplo es el modelo del espacio vectorial junto con el coseno de la medida de similitud.
- *Situación de obtención cerrada. Q* y *D*, y por lo tanto Q y D son conocidos inicialmente. ρ modela dependencias semánticas encontradas en D con respecto a Q. Un ejemplo es una red neuronal autocodificada aplicada en la identificación de categorías en *D* [62].
- *Situación de obtención semi-cerrada. Q* es desconocido y *D* es conocido inicialmente. ρ modela dependencias semánticas de D y expande una consulta $q \in Q$ con respecto a la estructura encontrada. Un ejemplo es PLSI.

	Situación de obtención abierta	Obtención (semi-) cerrada
imparcial	Locality sensitive hashing [70] p-stable LSH [57] LSH forest [54]	MDS with cosine similarity [59] (latent semantic indexing) Non-metric MDS [67] PCA [66]

parcial	Fuzzy-fingerprinting [68] Vector approximation [69] Shingling [55]	Probabilistic LSI [63] Autoencoder NN [62] Iterative residual rescaling [71] Locality preserv. ind., LPI [61] Orthogonal LPI [58]

Tabla 2.1: Clasificación de paradigmas de empotramiento usadas para indexar en obtención de información.

Aquí, se propone el modelo mostrado en la Tabla 1 para clasificar métodos de empotramiento usados en obtención de información. El modelo distingue además si el dominio de conocimiento es explotado o no dentro del procedimiento de empotramiento (parcial contra imparcial). E.g., funciones hash sensibles a la localización trabajan con datos arbitrarios, mientras que huellas difusas así como shingling [55] explotan el hecho de que los datos empotrados son texto. Una discusión similar se aplica a MDS y LSI probabilístico.
Además de su restricción a obtención (semi-) cerrada la mayoría de los métodos de empotramiento en la columna derecha de la Tabla 2.1 no son escalables para grandes colecciones: emplean alguna forma de descomposición espectral, la cual es computacionalmente cara.

2.4.2.1. Principio genérico para la construcción de h_ρ

En esta sección se presenta una vista unificada de métodos de búsqueda basados en funciones hash al interpretarlos como instancias de un principio genérico de construcción, el cual compromete los siguientes pasos:

1. *Empotramiento*. Los vectores característicos *m*-dimensionales de los documentos en *D* son empotrados en un espacio bajo-dimensional, esforzados en tener mínima distorsión. Los vectores característicos *m*-dimensionales resultantes se parecerán en razón de distancia, al menos el orden de las parejas de distancias inter-documentos, tan cercanas como sea posible.
2. *Cuantificación*. Los componentes valorados realmente de los vectores característicos empotrados son mapeados sobre un número pequeño de valores.
3. *Codificación*. Desde los componentes *k* cuantificados se calcula un número único, el cual sirve como un código hash.

Algunos o todos estos pasos se pueden repetir para el mismo vector característico *x* con el propósito de obtener un *conjunto* de códigos hash para *x*.

2.4.2.2. Una vista unificada para funciones hash sensibles a la localidad y codificación utilizando lógica difusa

Funciones hash sensibles a la localidad (LSH) es una marco de trabajo genérico para la construcción de métodos de búsqueda basados en funciones hash. Para llevar a cabo el empotramiento, una función hash sensible a la localidad h_φ emplea una familia $\mathcal{H}_\varphi$ de funciones hash simples h, $h: R^m \to N$. Desde $\mathcal{H}_\varphi$ se escoge un conjunto de *k* funciones por una opción aleatoria independiente y uniformemente distribuida, donde cada función se usa para calcular un componente del empotramiento *y* de un vector original *x*. Se han propuesto varias familias de funciones hash $\mathcal{H}_\varphi$ [56, 57, 54]. En este apartado se enfocan en la propuesta de Datar [57], la cual mapea un vector característico *x* a un número real al calcular el producto punto $\mathrm{a}^{\mathrm{T}} \cdot x$. a es vector aleatorio del cual se escogen componentes desde una distribución de probabilidad α-estable.

La cuantificación se logra al dividir la línea del número real en intervalos equidistantes de amplitud *r* donde cada uno de los cuales tiene asignado un número natural único. El resultado del producto punto se identifica con el número de su intervalo asignado.

La codificación puede suceder en diferentes maneras y se hace típicamente al sumar; el cálculo de $h_\varphi^{(\rho)}$ para un conjunto ρ de vectores aleatorios $a_1, \dots, a_k$ se realiza de la manera siguiente:

$$h_\varphi^\rho(x) = \sum_{i=1}^{k} \left\lfloor \frac{a_i^T \cdot x + c}{r} \right\rfloor \qquad 2.12$$

Donde $c \in [0, r]$ es un offset aleatoriamente escogido de la línea del número real. Una función hash de múltiples valores repite los pasos señalados para diferentes conjuntos $\rho_1, \dots, \rho_l$ de vectores aleatorios.

La codificación utilizando lógica difusa (FF) es un método de búsqueda basado en funciones hash diseñado específicamente para obtención de información basada en texto. Su procedimiento de empotramiento señalado se puede entender como una abstracción del espacio vectorial y sucede al "resumir" un vector *x* de términos *m*-dimensional hacia *k* clases prefijas. Una clase prefija comprende todos los términos con el mismo prefijo; los componentes del vector característico *y* empotrado cuantifican las desviaciones esperadas normalizadas de las clases prefijas *k* escogidas.

La cuantificación se logra al aplicar un modelo de codificación usando lógica difusa, ρ, el cual proyecta las desviaciones exactas $y_1, \dots, y_k$ en el intervalo de desviación r tal que: $\rho: R \to \{0, \dots, r-1\}$

La codificación se lleva a cabo al calcular el número más pequeño en notación base r de las desviaciones calculadas usando lógica difusa; el cálculo de $h_\varphi^{(\rho)}$ para un particular modelo con lógica difusa particular ρ se lee:

$$h_{\varphi}^{(\rho)}(x) = \sum_{i=1}^{k} \rho(y_i) \cdot r^{i-1} \qquad 2.13$$

donde y_i es la desviación esperada normalizada de la *i*-ésima clase prefija en el vector de términos *x* original. De manera similar a LSH, una función hash de múltiples valores repite los pasos de cuantificación y codificación para diferentes modelos con lógica difusa $\rho_1, \ldots, \rho_l$.

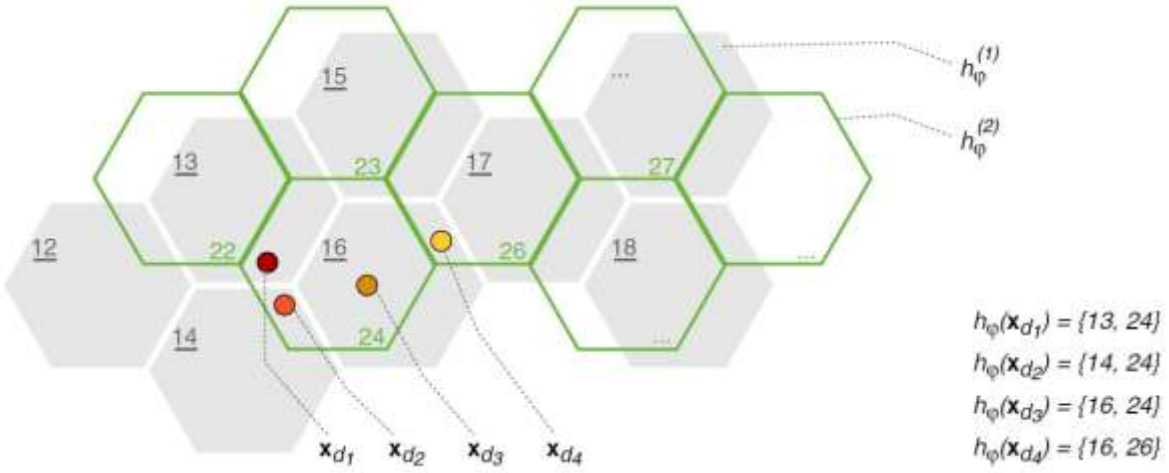


Figura 2.17: Un espacio dividido en regiones superpuestas, entendidas como dos mallas de hexágonos sombreados y con líneas. Cada región es caracterizada por una clave única; puntos en la misma región tienen una similitud de al menos θ. Una función hash de similitud h_{φ} en el nivel θ asigna un conjunto de la región de claves a un vector característico x_d, implicando la siguiente semántica: si y sólo si dos vectores característicos comparten una región clave se considera que ellos tienen una similitud de al menos θ. En el ejemplo $h_{\varphi}(x) = \{h_{\varphi}^{1}(x), h_{\varphi}^{2}(x)\}$ realiza ambas, una *precision* y un *recall* de 1. Para propósitos de legibilidad las regiones sombreadas son mostradas subrayadas.

2.4.2.3. Propiedades del control de obtención

La propiedad más sobresaliente de la búsqueda basada en funciones hash, es la simplificación de una función de similitud continua φ al concepto binario "similar o no similar": se consideran a dos vectores característicos similares si sus claves hash son iguales; de otra manera son considerados como no similares. Esta implicación se generaliza en la Ecuación 1 mostrada al principio de este tema; la generalización pertenece a dos aspectos: *(i)* la relación de equivalencia se refiere a un umbral de semejanza θ, y *(ii)* la función hash h_{φ} es de valores múltiples.

Con los antecedentes de los métodos de búsqueda basados en funciones hash presentados, ahora se continua la discusión de *precision* y *recall* . Se observa que la probabilidad de colisión hash para dos vectores x_{d_1}, x_{d_2} se decrementa si el número *k* de funciones hash simples (LSH) o de clases

prefijas (FF) se incrementa. Cada función hash o cada clase prefija captura conocimiento adicional de *x* y por lo tanto aumenta el umbral de similitud θ. Esta consecuencia se puede descomponer con la siguiente formula, Propiedad 1:

"La longitud del código controla el precision"

Ser de múltiples valores es una condición necesaria para h_φ para lograr un *recall* de 1. Una función hash valuada escalar calcula una clave para un vector característico *x* en una vez, y por lo tanto, define un división rigurosa del espacio del vector característico. La Figura 2.17 ilustra esta conexión: la función hash valuada escalar $h_\varphi^{(1)}$ responsable por el seccionamiento sombreado, asigna diferentes claves a los vectores x_{d_1} y x_{d_2}, a pesar de su similitud alta (distancia corta). Con la función hash de múltiples valores $h_\varphi(x) = \{h_\varphi^1, h_\varphi^2\}$, la cual también considera el seccionamiento subrayado, la intersección $h_\varphi(x_{d_1}) \cap h_\varphi(x_{d_2})$ no es vacía, dando amplitud para inferir que $\varphi(x_{d_1}, x_{d_2}) > \theta$. De hecho, existe una relación monotónica entre el número de códigos hash y el *recall* obtenido, la cual puede ser descompuesta con la siguiente formula, Propiedad 2:

"La multiplicidad de códigos controla el *recall*"

Sin embargo, se tiene que pagar un precio, la mejora en el *recall* se obtiene con un decremento en el *precision*.

2.4.3. Optimización y empotramiento

El empotramiento del espacio vectorial en un espacio de menos dimensiones está vinculado inevitablemente con la pérdida de información. Mientras más pequeño sea el error de empotramiento, mejor es el *recall* y *precision* de la función hash construida, esto es porque la transformación afín en el paso 2 y 3 de la sección 2.4.2.1, la cual mapea un vector empotrado en un código hash, es conservara en distancia.

La sección inicia con una derivación del empotramiento óptimo global bajo la medida de similitud coseno, y entonces no cubre la inferioridad de su empotramiento comparado el empotramiento de la clase prefija de huella difusa. Esta observación se explica en la idea de los empotramientos *centrados en umbral*, para lo cual se introduce la base formal en la manera de nuevas estadísticas de error, llamadas *precision stress* y *recall stress* en un umbral de semejanza θ dado. Al extender la idea hacia matrices de similitud formadas con el umbral se muestra que tan óptimos se pueden desarrollar los empotramientos para funciones hash de similitud en situaciones de obtención cerradas.

2.4.3.1 Empotramientos globalmente óptimos

El escalamiento multidimensional *(multidimensional scaling* MDS) designa una clase de técnicas para empotrar un conjunto de objetos en un espacio valuado real de baja dimensión, llamado espacio de empotramiento. El error de empotramiento, también llamado "stress", se calcula a partir de las desviaciones entre las similitudes inter-objeto originales y las similitudes inter-objeto nuevas en el espacio de empotramiento.

Dados *n* objetos, su matriz de similitud, *S*, es una matriz asimétrica $n \times n$ de números reales positivos, de quien el (i, j)-ésimo registro cuantifica la similitud entre el objeto *i* y el objeto *j*. Cada objeto sea descrito por un *m*-dimensional vector característico $x \in R^m$, y sea X la matriz $m \times n$ compuesta por esos vectores.

Sin perder generalidad se asume que cada vector característico x es normalizado de acuerdo a la norma-l_2, i. e., $\|x\|_2 = 1$. Entonces, bajo la medida de similitud coseno, S es definida por la identidad $S = X^T X$, donde X^T especifica la matriz transpuesta de X.

Una propiedad importante de la medida de similitud coseno es que bajo la norma de Frobenius un empotrado óptimo de X puede ser directamente construido a partir de su valor de descomposición único (SVD, singular value decomposition). Con SVD se puede representar únicamente a una matriz arbitraria X como el producto de tres matrices:

$$X = U\Sigma V^T \qquad 2.14$$

U es una columna ortogonal $m \times r$ de la matriz, Σ es una diagonal $r \times r$ de la matriz con valores únicos de X, y V es un matriz $n \times r$. I.e., $U^T U = I$ y $VV^T = I$ donde I especifica la matriz identidad. Al usar estas propiedades, la matriz S puede ser reescrita bajo ambos puntos de vista de valor de descomposición único y el punto de vista de cómputo de similitud:

$$S = X^T X = (U\Sigma V^T)^T U\Sigma V^T \qquad 2.15$$
$$= \underbrace{V\Sigma^2 V^T}_{\text{SVD}} = \underbrace{(\Sigma V^T)^T (\Sigma V^T)}_{\text{Cómputo de similitud}}$$

ΣV^T representa un conjunto de puntos con los mismos inter-objetos similares al vector original. La naturaleza de la medida de similitud coseno implica la construcción directa de S y, en particular, las identidades $rank(S) = rank(X) = rank(\Sigma V^T)$. En cambio, si se restringe la cantidad de dimensiones del espacio de empotramiento a k, la matriz de similitud resultante $\hat{S}$ es también del rango de k. De acuerdo al teorema de Eckart-Young la rank-k aproximación óptima $\hat{S}^*$ de S bajo la norma de Frobenius puede ser obtenida desde el SVD de S, al restringir el producto de la matriz a los k valores únicos más grandes [60]:

$$\hat{S}^* = V_k \Sigma_k^2 V_k^T = \left(\Sigma_k V_k^T\right)^T \left(\Sigma_k V_k^T\right) \quad 2.16$$
$$\Longrightarrow \Sigma_k V_k^T = argmin \|S - Y^T Y\|_F$$
$$\left\{Y \middle|_{rank(Y)=k}^{columns\,(Y)=n,}\right\}$$

En la comunidad de obtención de información el empotramiento $Y_{SVD} \coloneqq \Sigma_k V_k^T$ de vectores de documento X es conocido como la representación en el espacio semántico latente así llamado, abarcado por los k conceptos. El proceso de empotramiento se convirtió popular bajo el nombre indexación semántica latente (LSI, latent semantic indexing) [59].

Observación 1. Una idea equivocada común es que LSI expresa a los vectores de documento dentro de un sub espacio con el propósito de representar similitud semántica. Más bien, LSI construye nuevas características para aproximar a las representaciones del documento original. Y, si la dimensión del espacio de empotramiento es elegida adecuadamente entonces, debido a la reducción de ruido y la eliminación de dependencias débiles, este empotramiento es capaz de direccionar problemas de obtención derivados del uso de sinónimos. Como consecuencia el rendimiento de la obtención se podría mejorar en aplicaciones de obtención semi cerradas. Hofmann argumenta de manera similar [63]: el principio de superposición detrás de LSI es incapaz de manejar polisemia.

2.4.3.2. La base de la búsqueda basada en hash: empotramientos centrados en umbral

Aunque el empotramiento Y_{SVD} minimiza el error de empotramiento de X, no es el mejor punto de inicio para construir códigos hash sensibles a la similitud. La razón principal es que MDS se esmera en alcanzar una minimización de la carga de trabajo global, mientras que los métodos de búsqueda basados en hash se concentran en las altas similitudes en S en primer lugar.[8] La naturaleza de esta propiedad es capturada por la siguiente definición, la cual está relacionada con la carga de un umbral específico de un empotramiento para los conceptos estadísticos de *precision* y *recall*. La figura 2.18 ilustra la definición.

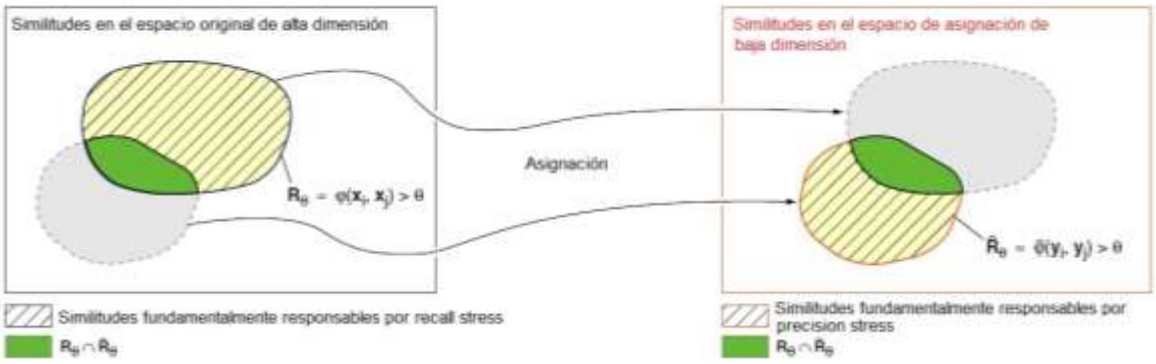


Figura 2.18. Si las representaciones del documento original, X, son empotradas en un espacio de baja dimensión, las representaciones del documento resultantes Y se parecen a las similitudes originales sólo imperfectamente. Dado un umbral particular θ, las similitudes del espacio original podrían ser intercambiadas de la parte de arriba de θ a la parte debajo de θ (área sombreada del lado izquierdo), de la parte debajo de θ a la parte de arriba de θ (área sombreada del lado derecho), o aún permanece en el intervalo $[\theta; 1]$ (área verde). Las similitudes en las áreas sombreadas son responsables de la mayor parte de la carga de empotramiento.

Definición 0.3.2.1 (precisión stress, recall stress) Sea D un conjunto de objetos y sea $\mathbf{X}$ y $\mathbf{Y}$ sus representaciones en el espacio n-dimensional y k-dimensional respectivamente, $\mathrm{k} < \mathrm{n}$. Además, sean $\varphi: \mathbf{X} \times \mathbf{X} \rightarrow [0; 1]$ y $\hat{\varphi}: \mathbf{Y} \times \mathbf{Y} \rightarrow [0; 1]$dos medidas de similitud, y sea $\theta \in [0; 1]$ un umbral de similitud.

θ define dos conjuntos de resultados, $\mathbf{R}_\theta$ y $\widehat{\mathbf{R}}_\theta$ los cuales comprenden pares $\{\mathrm{x_i}, \mathrm{x_j}\}$, $\mathrm{x_i}, \mathrm{x_j} \in \mathrm{D}$, de quienes sus respectivas representaciones en $\mathbf{X}$ y $\mathbf{Y}$ están por encima del umbral de similitud θ:

$$\{\mathrm{x_i}, \mathrm{x_j}\} \in \mathbf{R}_\theta \Leftrightarrow \varphi(\mathbb{x}_\mathrm{i}, \mathbb{x}_\mathrm{j}) > \theta, \qquad 2.17$$
$$\text{e igualmente: } \{\mathrm{x_i}, \mathrm{x_j}\} \in \widehat{\mathbf{R}}_\theta \Leftrightarrow \hat{\varphi}(\mathbb{y}_\mathrm{i}, \mathbb{y}_\mathrm{j}) > \theta$$

Entonces el conjunto de pares obtenido del espacio de empotramiento, $\widehat{\mathbf{R}}_\theta$, la carga de precisión en el umbral de similitud θ, $\mathrm{e_{p_\theta}}$:

$$e_{p_\theta} = \frac{\sum_{\{x_i, x_j\} \in \widehat{\boldsymbol{R}}_\theta} \left(\varphi(\mathbb{x}_i, \mathbb{x}_j) - \hat{\varphi}(\mathbb{y}_i, \mathbb{y}_j)\right)^2}{\sum_{\{x_i, x_j\} \in \widehat{\boldsymbol{R}}_\theta} \left(\varphi(\mathbb{x}_i, \mathbb{x}_j)\right)^2} \qquad 2.18$$

Así mismo, el conjunto de los pares similares en el espacio original, $\boldsymbol{R}_\theta$, define el recall stress en el umbral de similitud θ, e_{r_θ}:

$$e_{r_\theta} = \frac{\sum_{\{x_i, x_j\} \in R_\theta} \left(\varphi(\mathbb{x}_i, \mathbb{x}_j) - \hat{\varphi}(\mathbb{y}_i, \mathbb{y}_j)\right)^2}{\sum_{\{x_i, x_j\} \in \boldsymbol{R}_\theta} \left(\varphi(\mathbb{x}_i, \mathbb{x}_j)\right)^2} \qquad 2.19$$

Observación 2. *Precision stress* y *recall stress* de un empotramiento Y son estadísticas que dicen algo acerca del máximo *precision* y *recall* que se puede obtener con códigos hash de similitud construidos a partir de Y. Mientras más grande sea el *precision stress* mayor es la probabilidad de que se afirme que dos vectores empotrados, $\mathbb{y}_i, \mathbb{y}_j$, son similares a través de su similitud en el espacio original, $s_{ij} = \varphi(\mathbb{x}_i, \mathbb{x}_j)$.

Para los tres empotramientos, Y_{SVD}, Y_{FF}, y Y_{LSH}, obtenidos del óptimo MDS, fuzzy-fingerprinting, y LSH respectivamente, se analizaron las medidas de *precision stress* y *recall stress* en varios umbrales de similitud y con diferentes corpus. Los resultados reflejan el comportamiento pronosticado:

1. Debido a su generalidad (independencia de dominio), el empotramiento de LSH es consistentemente peor que el empotramiento de la clase prefija de fuzzy-fingerprinting.
2. En algún punto de paro par, el rendimiento de obtención del empotramiento de la clase prefija arroja el empotramiento óptimo de MDS.

Observación 3. Para la mayoría de las recientes tareas de obtención se puede aceptar un *precision stress*, ya que el análisis de similitud exacta necesaria necesita que se lleve a cabo sólo para una pequeña fracción $|D_q|/|D|$ de todos los documentos. Se debe recordar que los métodos de construcción para los métodos de búsqueda basados en hash proveen maneras suficientes para ajustar de manera fina la compensación entre el *precision stress*, e_p, y el *recall stress*, e_r.

Capítulo 3

Diseño y caracterización de la arquitectura de Internet basada en contenido

3.1 Arquitectura propuesta para Enrutamiento Semántico

3.1.1. Descripción general

El propósito del enrutamiento semántico es eliminar la necesidad de tener tres sistemas independientes, montados uno sobre otro, para poder recuperar información almacenada en la Internet. La propuesta consiste en distribuir los objetos de datos (o referencias a los mismos) entre los nodos de la red, de forma tal que objetos similares se encuentren localizados en nodos cercanos de la red. De esta forma, en lugar de requerir varios viajes a través de la red para resolver una consulta, el algoritmo de enrutamiento semántico puede dirigir la consulta hacia el nodo que contiene información la información más relevante.

Como se muestra en la Figura 3.1, las consultas se encaminan salto a salto, brincando siembre al nodo vecino que contiene información más relevante para la consulta. El algoritmo de enrutamiento garantiza que el máximo local alcanzado, será también el máximo global y por lo tanto el nodo que contiene a los objetos de datos que son semánticamente más cercanos a la consulta. Como se detalla en secciones posteriores, tanto los objetos de datos como las consultas son especificadas por medio de un conjunto de palabras clave tomadas de conceptos de una ontología de domino.

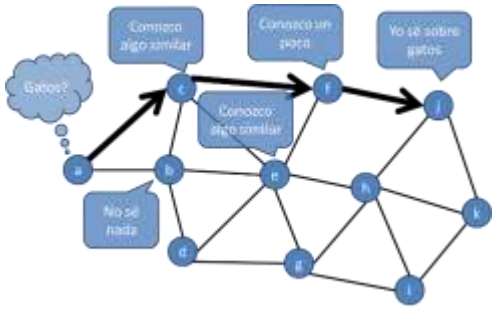


Figura 3.1. Funcionamiento de una red que utiliza enrutamiento semántico. Las consultas son encaminadas salto a salto, hacia el nodo que tiene la información más relevante.

Para responder a la consulta, el nodo que contiene a los objetos de datos más relevantes utiliza el camino inverso seguido por la consulta, para transmitir dichos objetos de datos. Lo anterior se muestra en la Figura 3.2, donde se puede observar que el nodo *j* utiliza el camino computesto por los nodos *jfca* para entregar los obetos de datos al nodo *a* que fue quien inició la consulta.

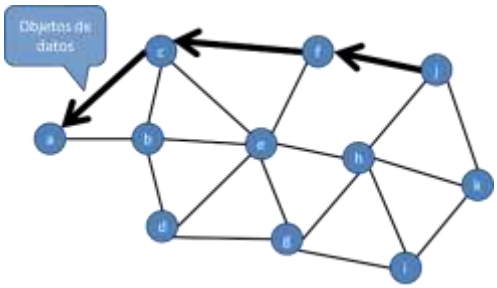


Figura 3.2. Los objetos de datos pueden ser entregados al destino siguiendo el camino inverso recorrido por la consulta.

Para implementar la funcionalidad anterior, se pronen tanto un esquema de enrutamiento compacto basado en etiquetas, como una función hash semántica. Ambos algoritmos definen funciones cuyo co-domino es un conjunto de etiquetas basadas en prefijos que son usadas para asociar a los objetos de datos con los nodos de la red que los alojarán. En términos simples, un objeto de datos con etiqueta l_o se almacenara en el nodo de la red cuya etiqueta posea el prefijo común más largo.

La función hash semántica toma como parámetros una ontología *O*, que define el contexto de las búsquedas, así como objetos de datos definidos por medio de un conjunto de palabras clave, y devuelve una etiqueta basada en prefijos de un nodo de un árbol *k*-ario de altura *h*. La principal propiedad con que debe contar la función hash semántica es que si dos objetos de datos son semánticamente similares, dada la conceptualización definida por la ontología, entonces las etiquetas que les son asignadas deben pertenecer a nodos cercanos en el árbol *k*-ario. En este caso la cercanía se define en térinos de la distancia en saltos entre los nodos del árbol. La Figura 3.3 ilustra este comportamiento donde puede observarse que los objetos de datos “cats” y “lions “son mapeado a nodos cercanos en el árbol *k*-ario. Por otro lado, el objeto “silver”es mapeado a un nodo alejado de los dos anteriores. Un aspecto importante a notar, es que la disntacia en saltos también entre nodos del árbol está relacionada con la distancia entre las etiquetas que le son asignadas a dichos nodos. La medida de distancia entre etiquetas propuesta está basada en la longitud de las etiquetas mismas y la longitud de su prefijo común.

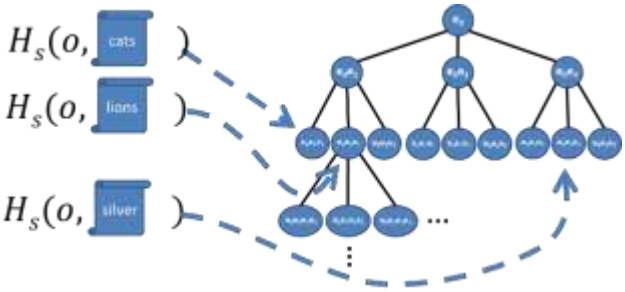


Figura 3.3. Ejemplo del comportamiento de una función hash semántica. Objetos similares deben ser mapeados a nodos cercanos en el árbol, mientras que objetos diferentes deben ser mapeados a nodos lejanos.

El procedimiento para calcular la etiqueta que le corresponde a un objeto de datos consta de dos etapas. En la primera etapa, un objeto de datos especificado por un conjunto de palabras clave $obj = \{w_1, w_2, \ldots, w_n\}$ es mapeado a la superficie de una híper-esfera de radio uno, que se encuentra en un espacio *d* dimensional cuya base está compuesta por conceptos de la ontología. Con el propósito de que los conceptos efectivamente definan una base del espacio *d* dimensional,

se eligen de forma tal que cada uno de ellos sea independiente de los otros, es decir, que no puedan ser expresado en términos de los otros. De esta forma, se cumplirá que ninguno de los vectores que se seleccionan como base del espacio *d* dimensional tampoco puedan ser expresados como una combinación lineal de los vectores restantes.

Una vez definida la base del espacio *d* dimensional, los objetos de datos son mapeados a la superficie de la híper-esfera por medio del cálculo de la similitud semántica de cada una de sus palabras claves, con los conceptos que definen el espacio *d* dimensional. Así, cada palabra clave contribuye al valor de la componente correspondiente a un concepto dado en proporción con la similitud semántica de la palabra y dicho concepto. De esta forma, los objetos de datos se pueden representar por medio de un vector de magnitud 1 y cuyos ángulos están definidos en términos de la *semántica* de dichos objetos. Esta idea se presenta de manera esquemática en la Figura 3.4, donde se muestra un ejemplo en el que un objeto de datos definido por *n* palabras claves es mapeado a un espacio tridimensional cuya base está compuesta por los conceptos C_B, C_C, C_D definidos en la ontología de contexto.

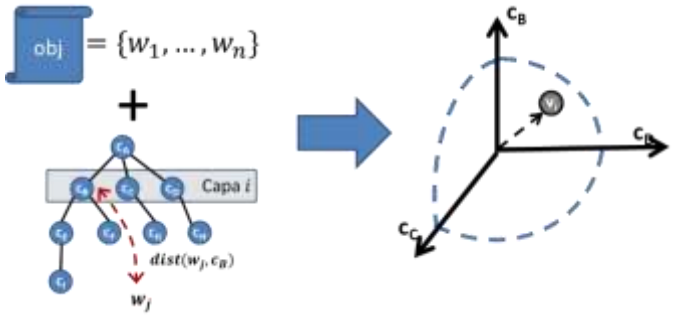


Figura 3.4. Fase 1 del proceso de cómputo de la función hash semántica. Los objetos de datos son mapeados a la superficie contenida en el primer cuadrante de una híper-esfera de radio 1.

Una vez que se ha obtenido el vector característico del objeto de datos, la segunda fase consiste en asignarle una etiqueta basada en prefijos. El primer paso de esta segunda fase consiste en etiquetar el árbol *k*-ario de altura *h* comenzando por la raíz y siguendo las relaciónes padre hijo definidas por las aristas del árbol. Las etiquetas están compuestas por caracteres obetnidos de un álfabeto Σ de *k* símbolos. La etiqueta de la raíz está compuesta por un único símbolo, mientras que las etiquetas de los nodos internos y de los nodos hoja se obtienen por medio de la concatenación de la etiqueta de su nodo padre con un caracter tomado de Σ que debe ser diferente de los caracteres asignados a cualquiera de sus nodos hermano. Así un nodo u con padre v cuya etiqueta sea e_v, se le asignara la etiqueta $e_v x$ donde $x \in \Sigma$, con $x \neq y$ para cualquier etiqueta $e_v y$ asignada a cualquiera de los nodos hermanos de u.

El segundo paso consiste en asignar regiones de la superficie de la híper esfera a cada uno de los nodos del árbol *k*-ario. Para esto, a cada nodo del árbol se le asigna un vector unitario, comenzando por el nodo raíz que es colocado en la posición $(1, \frac{\pi}{4}, \dots, \frac{\pi}{4})$. El resto de los nodos del árbol se posicionan en la superficie de la híper-esfera de manera uniforme a lo largo de anillos con centro en

la posición del nodo raíz. El proposito es distribuir los nodos del árbol de la manera lo más homogénea posible y que el diagrama de Voronoi inducido por las posiciones de los nodos en la superficie de la hiper-esfera esté compuesto de regiones del mismo tamaño. De esta forma, estas regiones de Voronoi son asignadas a su nodo semilla correspondiente. Cuando llega un nuevo documento, la función hash propuesta le asigna la etiqueta del nodo semilla correspondiente al punto donde el objeto haya sido mapeado.

Es importante notar, que no es necesario calcular el diagrama de Voronoi completo para asignar etiquetas a objetos de datos debido a que dado el vector característico del objeto, es suficiente encontrar a la semilla más cercana cuya etiqueta le será asignada al objeto de datos. La Figura 3.5 muestra un ejemplo donde el nodo raíz (en negro) es "sembrado"en la posición $(1, \frac{\pi}{4}, \frac{\pi}{4})$, mientras que sus nodos hijos son sembrados sobre un circunferencia con centro en $(1, \frac{\pi}{4}, \frac{\pi}{4})$.

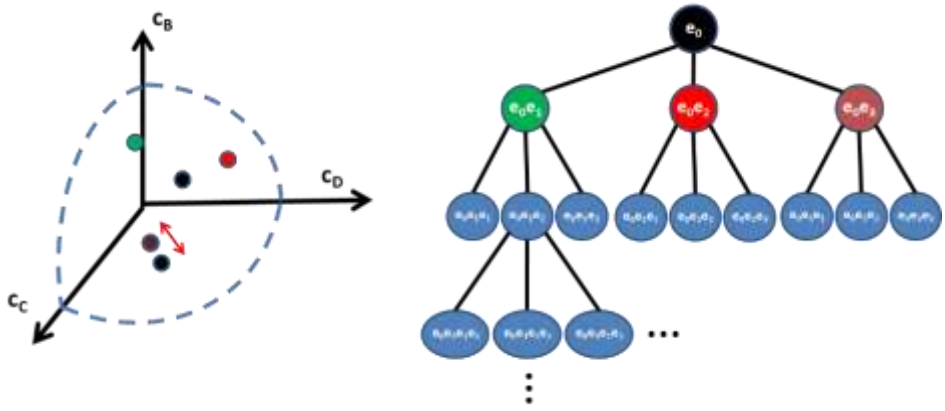


Figura 3.5. Fase 2 del proceso de cómputo de la función hash semántica. Los objetos de datos reciben la etiqueta del nodo del árbol k-ario más cercano.

Como se mencionó anteriormente, los nodos de la red también son etiquetados usando etiquetas basadas en prefijos. Para esto, se utiliza un algoritmo de elección distribuida de líder para seleccionar al nodo que fungirá como núcleo de la red. Una vez que se ha elegido al núcleo de la red, éste inicia una búsqueda por amplitud que establece apuntadores padre-hijo entre todos los nodos que componen la red. Además de los apuntadores padre-hijo, la búsqueda por amplitud asigna etiquetas basadas en prefijos de manera completamente análoga la forma en que le fueron asignadas a los nodos del árbol *k*-ario. Es importante notar que aún y cuando no existe ninguna relación entre las etiquetas que le son asignadas a los objetos de datos y a los nodos de la red, el hecho de que el espacio de las etiquetas sea el mismo, provee de una forma efectiva y eficiente para asignar objetos de datos a nodos de la red que consiste en simplemente almacenar los objetos de datos en el nodo de la red que tenga la etiqueta más cercana a la etiqueta de los objetos de datos. Este concepto se muestra de manera gráfica en la Figura 3.6 donde se puede observar que tanto la red de computadoras como los objetos de datos son mapeados al espacio de las etiquetas de

prefijos que se le asignan a un árbol *k*-ario de altura *h* a partir de los caracteres obtenidos de un alfabeto Σ con $|\Sigma| = k$.

Ya que se han descrito los dos principales componentes, es posible mostrar el funcionamiento del esquema de enrutamiento semántico propuesto. Sea $G = (V, E)$ un grafo que representa la topología de una red de computadoras donde V es el conjunto de nodos y E es el conjunto de enlaces que conectan a una computadora con otra, y sea $l_G^{\Sigma}: V \rightarrow L$ una función que asigna etiquetas de prefijos a los nodos de la red usando caracteres definidos en un alfabeto Σ. Sea $h_O^{\Sigma}: D \rightarrow L$ una función que mapea objetos de datos $o \in D$ definidos por medio de un conjunto de palabras clave a etiquetas basadas en prefijos obtenidos de etiquetar un árbol *k*-ario de altura *h* donde O es una ontología que define el contexto temático de los objetos de datos. El proceso de almacenar un objeto de datos en la red es el siguiente. El objeto o se almacena en el nodo $i \in V$ tal que:

$$i = \underset{j \in V}{\operatorname{argmin}}\{\, dist(h_O^{\Sigma}(o), l_G^{\Sigma}(j))\} \qquad 3.1$$

Donde $dist$ es una función de distancia basada en la longitud del prefijo común entre las etiquetas $h_O^{\Sigma}(o)$ y $l_G^{\Sigma}(j)$, así como sus longitudes absolutas medidas en términos del número de caracteres que las componen. Es importante remarcar, que la ecuación anterior se calcula de manera distribuida *enrutando* hacia la etiqueta $h_O^{\Sigma}(o)$.

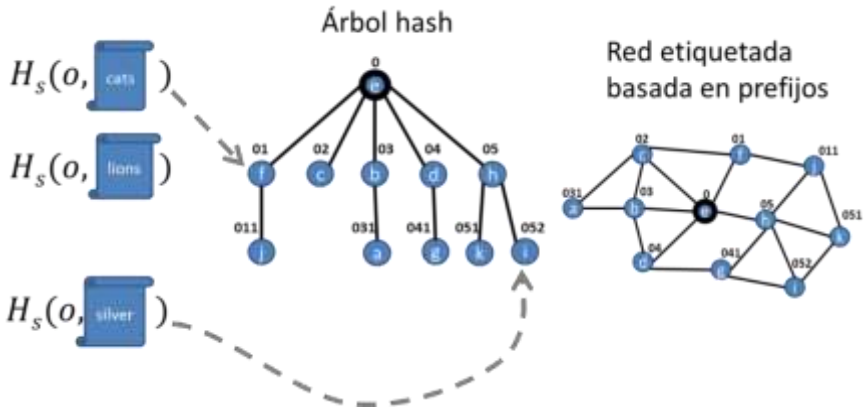


Figura 3.6. Funciones hash en el espacio de etiquetas. La función que asigna etiquetas a nodos, y la función hash semántica comparten el mismo co-domino que consiste del espacio de las etiquetas generadas a partir de un alfabeto Σ con $|\Sigma| = k$.

Para recuperar objetos de datos por medio de una consulta $q = \{w_1, w_2, \ldots, w_n\}$ definida con base a un conjunto de palabras clave, un nodo $u \in V$ calcula la función hash de dicha consulta para obtener su etiqueta $h_O^{\Sigma}(q)$. Ahora, para llegar al nodo $v \in V$ que contiene los objetos de datos más relevantes para la consulta se tiene que calcular la siguiente ecuación.

$$v = \underset{j \in V}{\operatorname{argmin}}\{\, dist(h_O^{\Sigma}(q), l_G^{\Sigma}(j))\} \qquad 3.2$$

Lo cual, se puede realizar de manera distribuida al enrutar hacia la etiqueta $h_O^{\Sigma}(q)$. Como puede observarse, una consulta $q = \{w_1, w_2, \dots, w_n\}$ es encaminada salto a salto hacia nodos con etiquetas cada vez más similares y por lo tanto hacia nodos con información cada vez más relevante para la consulta. Este panorama, es exactamente el descrito en la Figura 3.1.

En las siguientes secciones se presentarán los detalles técnicos de los algoritmos utilizados para implementar la funcionalidad aquí descrita.

3.1.2. Enrutamiento basado en etiquetas

Una parte importante para llevar a cabo el enrutamiento semántico es poder hacer enrutamiento de paquetes de datos de un origen a un destino. En nuestra propuesta de enrutamiento, hacemos uso del esquema de enrutamiento compacto en la que se tiene la ventaja de que sus tablas de enrutamiento crecen de manera independiente al tamaño de la red, en nuestro caso, tienen como tamaño máximo el grado máximo en la red. El precio que se tiene que pagar es que no se usan los caminos más cortos, como en el enrutamiento usado por la Internet actual, en su lugar este modelo de enrutamiento busca minimizar el *Stretch*, es decir, reducir el resultado de dividir la distancia del camino obtenido con el protocolo de enrutamiento compacto entre la distancia obtenida para ese mismo camino con el protocolo de enrutamiento óptimo o enrutamiento del camino más corto (e.g., OLSR [3]). Entonces, nuestro protocolo tiene un *Stretch* de $O(logn)$.

3.1.2.1. Funcionamiento del protocolo de enrutamiento compacto

Nuestro protocolo usa como identificador único, ya sea direcciones IP o MAC, con el propósito de poder identificar de manera única a cada nodo en la red. Para poder llevar a cabo el enrutamiento el protocolo construye y mantiene un grafo etiquetado acíclico dirigido (LDAG, por sus siglas en inglés) cuya raíz es un nodo elegido de forma distribuida usando el algoritmo del grandulón (*bully algorithm*). De manera similar como en un árbol de expansión distribuido, la elección del nodo raíz es realizada en la operación del protocolo. Para asegurar que todos los nodos en la red tengan al mismo nodo como raíz se propagan paquetes de señalización hasta que se llega al consenso de aceptar al mismo nodo como raíz (Figura 3.7 (a)). Al mismo tiempo que se propagan estos paquetes, el nodo raíz va formando un árbol, y anuncia su etiqueta, así como las etiquetas que asigna a sus hijos inmediatos. Además, almacena las etiquetas prefijas que escucha de sus vecinos. Al término de este proceso todos los nodos conocen su etiqueta así como las etiquetas de sus vecinos (padres, hermanos, e hijos). Después de haber etiquetado la red, las direcciones IP o MAC dejan de usarse, ya que ahora el identificador único es la etiqueta del nodo. Las direcciones IP o MAC aún pueden ser útiles en caso de un re-etiquetado en la red (poner referencia a la parte de reetiquetado).

Para que un nodo fuente se puede comunicar con un nodo destino, en esta etapa, se debe de especificar la etiqueta prefija del nodo destino. Una vez que el nodo destino y fuente conocen las etiquetas prefijas uno del otro, ellos se pueden comunicar directamente (Figura 3.7 (b)).

EL LDAG sirve como una estructura fuerte tolerante a la salida de nodos de la red, la cual por omisión asigna a un nodo con la etiqueta prefija más parecida cuando una solicitud para una concordancia falla (cuando el nodo buscado no se encuentra). El algoritmo de enrutamiento que trabaja sobre el LDAG usa una estrategia *greedy*. Cuando encuentra un mínimo local, escoge el siguiente salto con la dirección IP o MAC más pequeña. Dado que los nodos tienen etiquetas prefijas, la opción del identificador único no afecta el camino hacia el destino. Los nodos usan la lógica de máxima similitud prefija en el algoritmo de enrutamiento para escoger su siguiente salto.

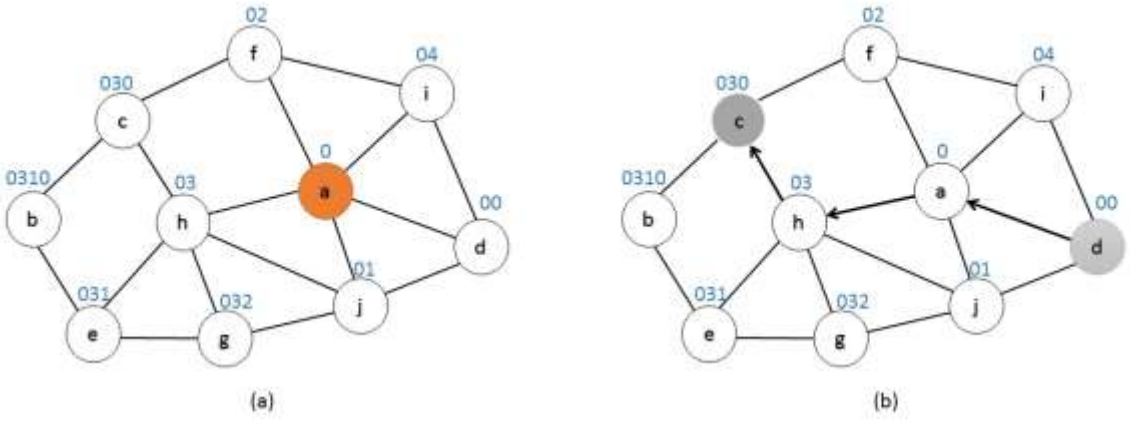


Figura 3.7. Enrutamiento con etiquetas. (a) Elección de nodo raíz. (b) una vez que los nodos c y d conocen sus etiquetas, pueden compartir paquetes de datos.

3.1.2.2. Estructura de las etiquetas prefijas

Para un alfabeto Σ que contiene un número finito de símbolos y Σ^* el conjunto de todas las cadenas sobre Σ tal que $|\Sigma| \geq 2$. Cada nodo asigna una etiqueta a cada uno de sus hijos con una letra ω de Σ. Si ω_i representa la letra asignada al *i-ésimo* hijo de cualquier nodo entonces, $\omega_i \mapsto \{\omega_i \in \Sigma | \omega_i \neq \omega_{i+1} \forall_i \leq d-1\}$, donde d es el grado del nodo. Por lo tanto, se asigna una unica letra a cada hijo de cada nodo. La lógica etiquetadora etiqueta a cada nodo en el LDAG de manera parecida a la búsqueda por amplitud. Dado que el LDAG se puede organizar como un árbol $k-ario$, siendo k el grado del LDAG, a cada hijo se le asigna una etiqueta prefija Λ como se define a continuación:

Etiqueta prefija: Una etiqueta prefija Λ para el nodo y es una palabra dentro de Σ^* tal que $\Lambda = \Lambda_{parent} \odot l'$, donde Λ_{parent} es la etiqueta prefija obtenida del padre y $\odot$ es el operador de

concatenación donde Λ_{parent} es concatenada con un único sufijo sobre k diferentes opciones de Σ para formar Λ.
Se deduce de la definición anterior que la etiqueta prefija Λ, de un nodo identifica únicamente al nodo en un LDAG dado. Las etiquetas prefijas de los nodos definen una relación *predecesor* en el LDAG, denotada por $\hookleftarrow$, tal que para dos nodos cualesquiera s y d en el LDAG:

1) $s \hookleftarrow d$: $s\ precede\ a\ d$. En este caso, y entonces se puede alcanzar a d al recorrer los subárboles de s.
2) $d \hookleftarrow s$: $d\ precede\ a\ s$. En este caso, y entonces se puede alcanzar a d yendo hacia arriba del flujo de s al recorrer los antecesores de s.
3) $s \approx d$: d y s son pares donde $|\Lambda_d| = |\Lambda_s|$ y ambos comparten un antecesor común. En este caso, d puede ser alcanzado al recorrer hacia arriba el antecesor común hasta que se mantenga $r \hookleftarrow s$, donde r es el antecesor común, y entonces recorrer hacia abajo del subárbol más parecido, mientras $r \hookleftarrow d$ se mantenga y d sea alcanzado. Por lo tanto, $d \approx s \Longrightarrow r \hookleftarrow s \Longrightarrow r \hookleftarrow d$.
4) $d \approx s$: Este es un caso especial del anterior, donde $|\Lambda_d| \neq |\Lambda_s|$ y el único antecesor común es el nodo raíz.

Los nodos construyen y mantienen un LDAG con raíz en un nodo elegido usando mensajes *hola* intercambiados entre los vecinos del uno vecindario. Cada mensaje *hola* especifica el identificador de la raíz, un número secuencial que se incrementa monótonamente el cual es asignado por el nodo raíz, la etiqueta prefijo y el identificador del nodo emisor, y una lista de tuplas con la asignación de etiquetas prefijas hacia cada uno de sus hijos. El algoritmo para la elección de la raíz escoge el nodo con el cubrimiento más grande hecho a un salto y rompe ataduras con el NID de la raíz más chica. En cuanto la red inicia a autoorganizarse con etiquetas prefijo, el proceso de etiquetamiento puede producir múltiples LDAGs, la etiqueta de cada uno de ellos se compara y la etiqueta más grande lexicográficamente se elige como el dominante.

Consideremos el ejemplo en la Figura 3.3 Las etiquetas prefijas para los nodos son asignadas sobre el alfabeto $\sum = 0, 1, 2, 3$. A la raíz del árbol se le asigna una letra de $\sum$ y los nodos unidos a la raíz son etiquetados con 00 y 01 sucesivamente. En el siguiente nivel, a cada nodo se le asigna una única etiqueta prefija combinada con el prefijo de su padre.

3.1.2.3. Enrutamiento

Para llevar a cabo en el enrutamiento de paquetes de datos de una fuente *s* a un destino *d*, un nodo elige el enlace hacia sus vecinos el cual le ofrezca la longitud máxima de similitud al comparar la etiqueta prefija de cada uno con la etiqueta del destino. Esta lógica de similitud máxima en etiquetas prefijas es la que nos permite ir avanzando en el camino hasta llegar al destino, además, selecciona el salto siguiente usando una estrategia *greedy*, con la cual permite buscar caminos más cortos que el enrutamiento tradicional basado en árbol prefijo, al sacar ventaja de la diversidad de caminos de un LDAG comparado con el árbol prefijo. Se asegura que esta estrategia *greedy* no encuentra un mínimo local al seleccionar aleatoriamente el siguiente salto cuando todos los nodos ofrecen la misma similitud prefija.

A partir de la relación de predecesor inducida por las etiquetas prefijas en el LDAG de una MANET, un nodo dado selecciona su siguiente salto mediante su etiqueta prefija de acuerdo a los cuatro posibles casos permitidos por la relación de predecesor.

3.1.3. Medida de similitud semántica para recuperación de información basada en taxonomías

La medida calcula la similitud semántica entre términos de una ontología dada, considerando el IC (*Information Content*) de cada uno. El enfoque principal de ésta es hacia el área de recuperación de información (IR).

Nuestra medida de similitud responde a la pregunta ¿En qué medida es el concepto A del tipo del concepto B? En lugar de responder a la pregunta que la mayoría de las medidas de similitud existentes plantean "¿Qué tan similar es el concepto A con el concepto B?". Por ejemplo, en la ontología de la Figura 3.8, con nuestra medida respondemos qué tan Águila Real es un Perro, y de la otra manera sería ¿qué tan parecida es un Águila Real a un Perro?

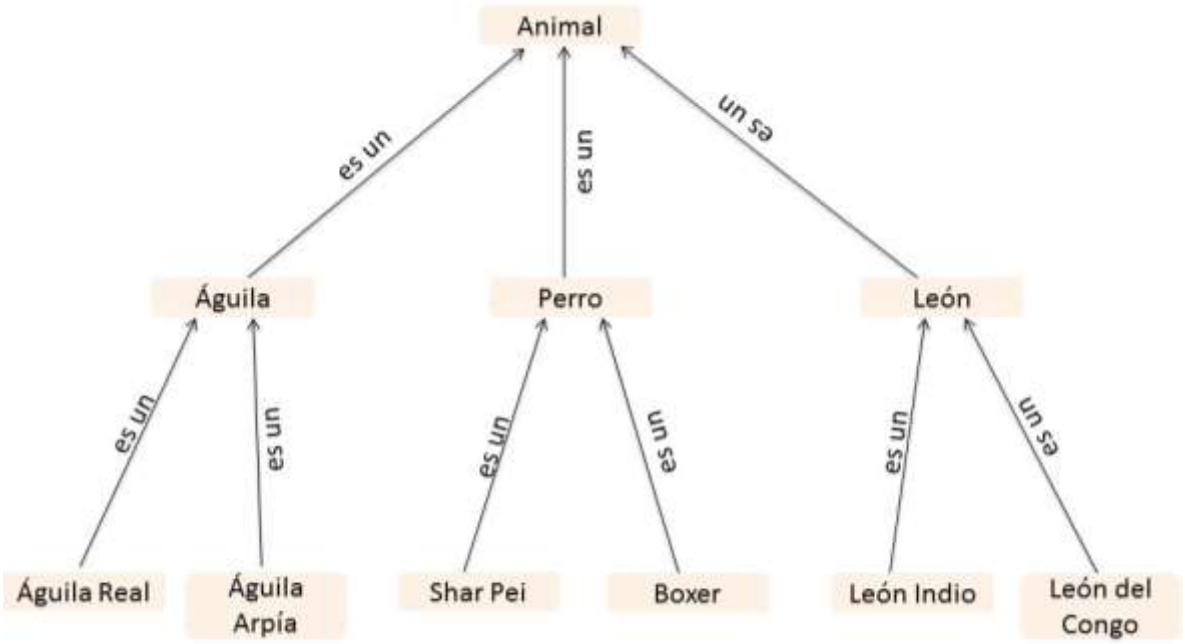


Figura 3.8. Ejemplo de ontología.

Es importante hacer notar, que la medida propuesta en este trabajo está específicamente diseñada para taxonomías donde la única relación definida dentro los conceptos es del tipo "es-un". El primer paso para el cálculo de la similitud entre dos conceptos de la ontología consiste en asignar peso a

las aristas definidas dentro de la ontología. Para esto, hemos definido dos alternativas, una simétrica y otra asimétrica. Ambas se muestran a continuación.

3.1.3.1 Medida de similitud semántica usando pesos simétricos.

Dada una ontología $O = (C, R)$ donde C representa al conjunto de conceptos y R representa al conjunto de relaciones "es-un" definidas entre los concetps de la ontología, la medida simétrica asigna pesos a todas las aristas $e \in R$ usando la fórmula:

$$w_{e=(u,v)} = e^{-|IC(u)-IC(v)|} \tag{3.1}$$

Donde $IC(u)$ es una función que asigna una cantidad de contenido de información a cada uno de los conceptos $u \in C$ definidos en la ontología O.

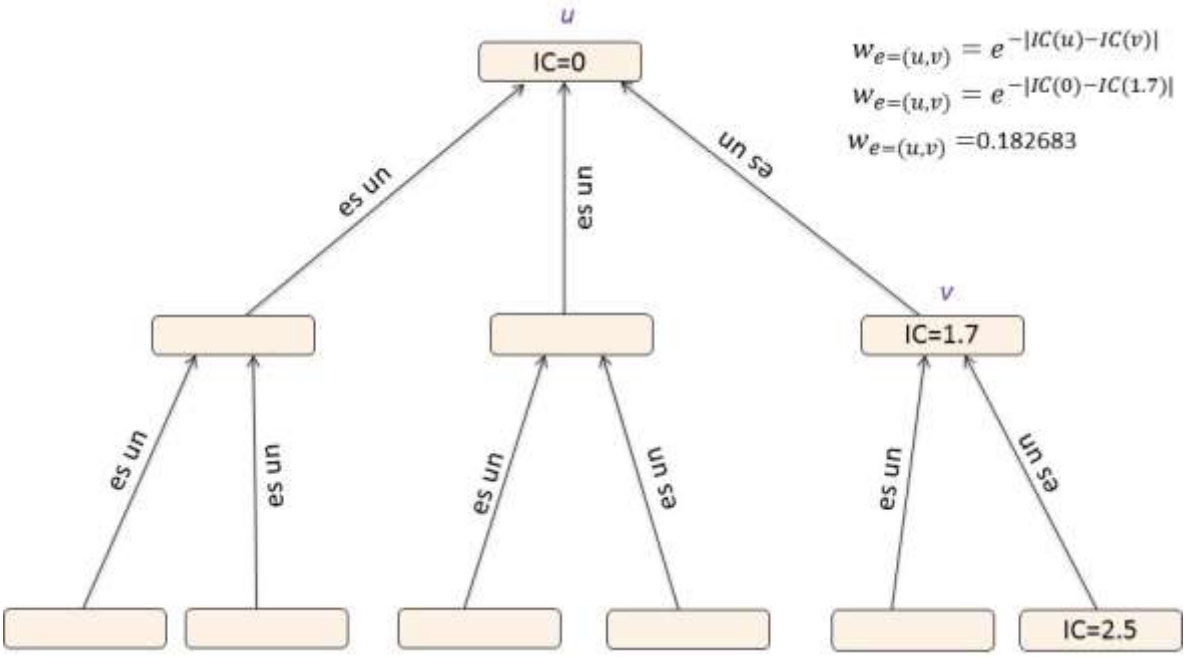


Figura 3.9. Cálculo de aristas para medida con pesos simétricos

La Figura 3.9 muestra un ejemplo de la utilización de la fórmula anterior donde se calcula el peso de la arista que une a los nodos u y v. Este cálculo se aplica para la asignación de peso a cada una de las aristas de la taxonomía. Como se puede observar, para esta versión de medida de similitud, el peso de la arista (u, v) es el mismo que el de la arista (v, u).

Así, la similitud entre dos conceptos $u, v \in C$ será inversamente proporcional al peso del camino más ligero que los une en la ontología O. Es decir:

$$sim(u,v) = e^{-c\sum_{e_i \in P_u^v} w_{e_i}}$$

Donde P_u^v es el camino de peso mínimo que conecta a los conceptos u y v. Además, cabe mencionar que para la sumatoria se consideran los pesos w_{e_i} asignados a las aristas $e_i \in P_u^v$ por medio de la ecuación anterior.

3.1.3.2 Medida de similitud semántica usando pesos asimétricos.

Para realizar el cálculo de las aristas en esta versión, nosotros creemos que es de vital importancia poder conocer cuando se pierde especificidad al intentar responder a solicitud de recuperación de información. Entonces, tomando como ejemplo la ontología de la Figura 3.8, es distinto responder "¿Qué tan Águila Real es un Animal?" que "¿Qué tan Animal es un Águila Real?". Es decir, si vemos en la ontología, cuando calculamos el camino de Águila Real a Animal, podemos darnos cuenta que estamos perdiendo especificidad, por lo cual creemos que esta pérdida debe tener un costo mayor, teniendo así, aristas con un peso mayor. En caso contrario, se gana especificidad, y por lo tanto, tendremos aristas de menor peso. Entonces, nosotros calculamos dos valores para una arista, para la arista u, v se usa la expresión 3.1, y para la arista *v*, *u* donde creemos que se debe penalizar la pérdida de información, usamos la expresión:

$$w_{e=(u,v)} = e^{IC(v)-IC(u)} \tag{3.3}$$

En la Figura 3.10 se muestra un ejemplo del cálculo del peso de las aristas (u,v) y (v,u).

La manera de usar los diferentes valores de las aristas, descritas anteriormente, depende del cálculo de la similitud, dado que no es lo mismo calcular *sim(u,v)* a *sim(v,u)*. Además, usamos el término MICA (*Most Informative Common Ancestor*) para que el camino en el árbol entre *u* y *v* sea de menor longitud que si se usara la raíz de la ontología para enlazar a *u* y *v*. Entonces, para calcular la similitud entre *u* y *v* usamos la siguiente expresión:

$$sim(u,v) = e^{-c\sum_{e_i \in P_u^{MICA}} w_{e_i}} + e^{-c\sum_{e_i \in P_{MICA}^v} w_{e_i}} \tag{3.4}$$

Que está compuesta por la suma de dos exponenciales. La primera que contempla el costo de transitar desde el concepto u al ancesto común más informativo (MICA), y la segunda del ancestro común más informativo al concepto v.

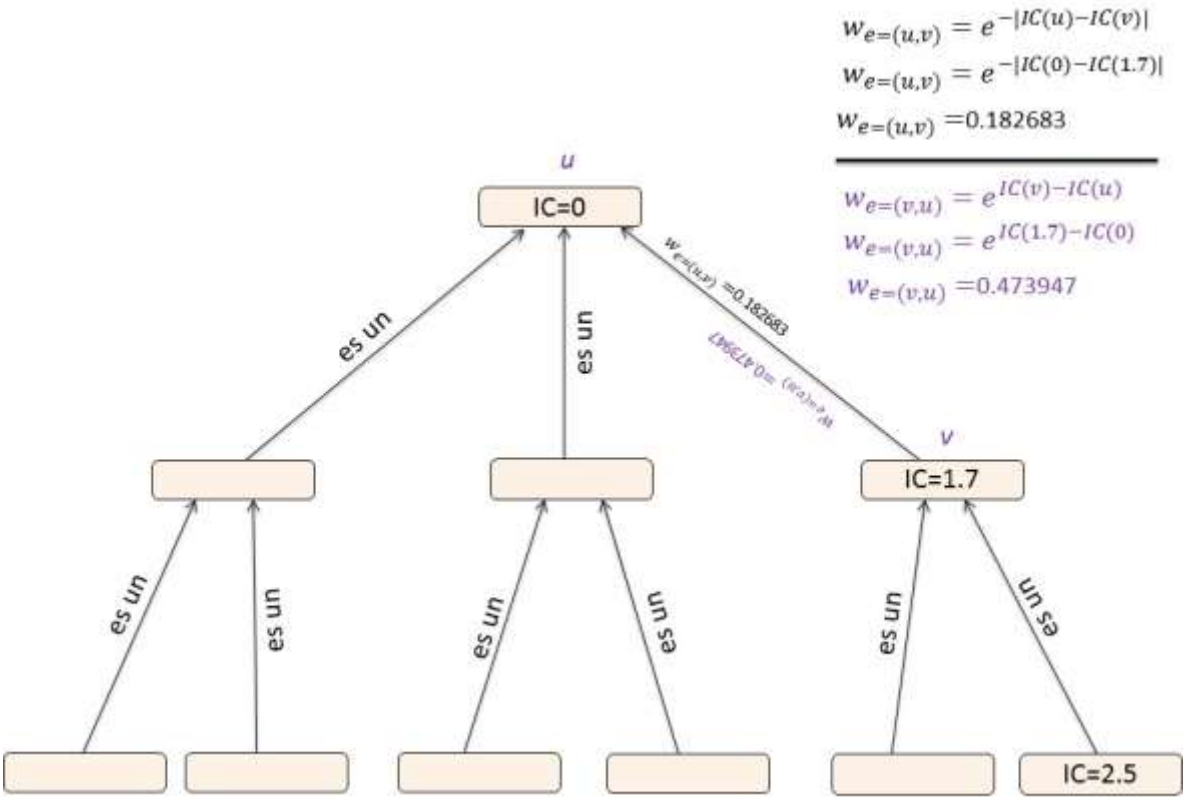


Figura 3.10. Cálculo de diferentes pesos de las aristas.

3.1.4. Similitud entre documentos y consultas

En secciones anteriores hemos definido la manera de calcular la similitud entre un par de conceptos definidos en una ontología. En esta sección presentamos una metodología para calcular similitud entre conjuntos de conceptos. En otras palabras el objetivo es definir una función $d_O: 2^C \times 2^C \rightarrow \mathbb{R}$ que mapeé al producto cartesiano del conjunto potencia del conjunto de conceptos consigo mismo, al espacio de los números reales, La función debe ser tal que valores cercanos a cero indiquen que ambos conjuntos de conceptos son similares, mientras que valores lejanos a cero indiquen que ambos conjuntos son diferentes.

De esta forma será posible calcular la distancia entre un objeto de datos $obj = \{w_1, w_2, \dots, w_n\}$ especificado por un conjunto de palabras clave tomadas de la ontología O, y una consulta $q = \{w_1, w_2, \dots, w_n,\}$ también especificada por medio de un conjunto de palabras clave.

El primer paso de este procedimiento consiste en mapear documentos y consultas a un espacio m-dimensional, donde cada una de las componentes esté especificada por la similitud de las palabras

clave a un concepto base. Este mapeo también servirá para asignar etiquetas de forma tal que objetos que tengan distancias semánticas cercanas obtengan etiquetas cercanas.

3.1.5. Mapeo de documentos os y consultas a un hiperespacio R^m

Para poder asignar etiquetas a los documentos hacemos una representación de los documentos en un espacio multidimensional R^m, para así poder representar cuánto es que un documento habla sobre cierta área del conocimiento, de esta manera obtenemos su vector característico en ese espacio multidimensional.

3.1.4.1. Determinación de las componentes del espacio *m*-dimensional.

Para determinar qué conceptos serán usados como la base del espacio vectorial, aplicamos el algoritmo de búsqueda por amplitud (BFS, *Breadth First Search*) a una ontología *O* para obtener una partición de los conceptos con base en la distancia en saltos de cada uno de ellos al concepto raíz. Dependiendo del número de dimensiones deseado se elige una capa que contenga un número de conceptos similar. Adicionalmente, es necesario que los conceptos contenidos en la capa seleccionada no compartan relaciones entre sí. El objetivo es que dichos conceptos sean independientes entre sí, y por lo tanto que no sea posible definir uno de ellos en términos de los demás. Desde el punto de vista geométrico, esto es equivalente a requerir que los conceptos sean una base del espacio vectorial y por lo tanto una de sus componentes no pueda ser obtenida por medio de una combinación lineal de las demás. La Figura 3.11 muestra un ejemplo de un vector característico en un espacio de tres dimensiones, donde se puede ver que el vector está más recargado hacia la componente C_B.

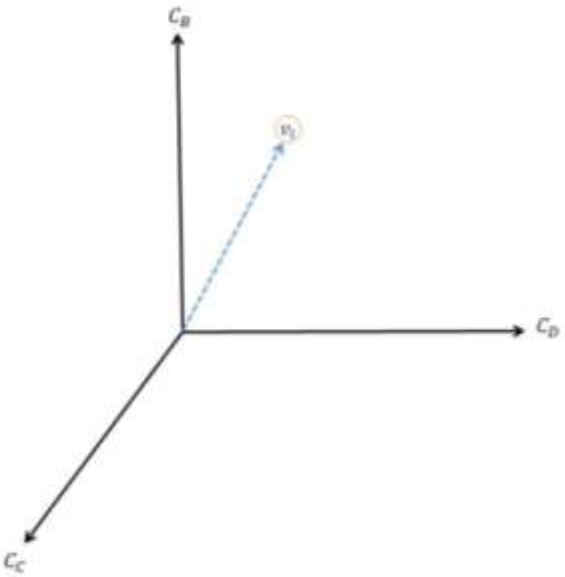


Figura 3.11. Ejemplo de vector característico. Vector característico con valor mayor en la componente C_B, con lo que podemos decir que el documento dado se refiere más a dicha componente.

3.1.4.2. Cálculo del vector característico.

Una vez que hemos definido la base de nuestro espacio *m*-dimensional, el siguiente paso es definir una función que a cada documento $d_i = \{k_1, k_2, \dots, k_n\}$ le asigne un punto en el espacio *m*-dimensional. Como hemos mencionado, un documento d_i está especificado por medio de una o varias palabras clave k_i pertenecientes a la ontología *O*. El vector de un documento/consulta es calculado con la siguiente formula:

$$v_i = \gamma(d_i, O) = \frac{\left(c_{d_i}^{c_B}, c_{d_i}^{c_C}, c_{d_i}^{c_D}, \dots\right)}{\left|\left(c_{d_i}^{c_B}, c_{d_i}^{c_C}, c_{d_i}^{c_D}, \dots\right)\right|} \qquad 3.5$$

donde

$$c_{d_i}^{c_X} = \sum_{j=1}^{n} e^{-c \times sim(k_j, c_X)} \qquad 3.6$$

k_n es cada una de las palabras clave pertenecientes al documento/consulta de quien se requiere su vector, C_x toma el valor de cada una de las componentes obtenidas en el BFS, c es una constante que sirve para amplificar o atenuar las diferencias entre los documentos y sim es la medida de similitud semántica propuesta. El denominador de la expresión anterior es una constante de normalización que sirve para que la distancia al origen de todos los puntos generados sea 1. De esta forma, tanto objetos de datos, como consultas, son mapeados a la superficie que se encuentra en el primer cuadrante de una hiperesfera de radio 1 y que está centrada en el origen.

Una vez que hemos obtenido el vector característico de una consulta o un objeto de datos, la similitud entre un par de ellos puede ser definida fácilmente en términos, ya sea de su distancia coseno [72], o de su distancia Euclidiana.

El siguiente paso en la metodología consiste en asignar etiquetas a diferentes regiones de la superficie de esta hiperesfera. Estas regiones deben ser convexas para que en general, puntos que se encuentren dentro de la misma región tengan una distancia pequeña.

3.1.6. Mapeo de un documento en un hiperespacio R^m a un árbol $K-ario$

Se propone un mapeo entre nodos y puntos en la *n*-esfera basándonos en el procedimiento que se describe a continuación.

1.- Indexar a los nodos del árbol usando palabras *k*-arias de longitud *h*.

Con profundidad $h \leq q$ un nodo ***n*** de un árbol de orden k tendrá un número de índice:

$$\alpha(\boldsymbol{n}) = \sum_{i=0}^{h-1} a_i k^i \qquad 3.7$$

Este índice es una dirección sobre un código k-ario etiquetando el camino que toma para alcanzar al nodo n desde la raíz. El coeficiente a_i podría tomar cualquier valor de 0 a *k-1* dependiendo de la rama la cual ha sido elegida desde un nodo en el nivel *i* hacia un nodo el nivel *i+1*.

2.- Emplear el índice para mapear los nodos como raíces del círculo unitario complejo.

Los nodos son distribuidos en raíces de círculos complejos concéntricos, con el origen correspondiente a la raíz del árbol. Los nodos del nivel *h* son mapeados a raíces de un círculo de radio 1. Sea *z* un número complejo, entonces:

$$z^q = 1 \qquad 3.8$$

es el círculo complejo con *q* raíces, correspondiente al número de nodos a partir del nivel *h* del árbol:

$$q = k^h \qquad 3.9$$

Este círculo complejo tiene *q* raíces:

$$\omega_l = e^{i2\pi l/q} \qquad 3.10$$

Con $= 0, \dots, q-1$

3.- Escalar cada raíz por un factor el cual es proporcional a la profundidad del nodo.

Estas raíces son escalables por un factor h/d, donde *h* es el nivel del nodo y d es la profundidad del árbol:

$$\omega_l{}' = \frac{h}{d}\omega_l \qquad 3.11$$

4.- Llevar a cabo una proyección estereográfica del plano complejo a una esfera en $\mathbb{R}^3$

La asignación de los nodos en las raíces de círculos complejos puede ser considerada como un mapeo sobre una 1-esfera en $\mathbb{C} \cong \mathbb{R}^2$. Usaremos esta 1-esfera compleja como una plantilla para proyectar los puntos en la superficie de una 2-esfera de una dimensión más alta.

Consideremos la esfera de Riemann $S_R = \{(x_1, x_2, x_3)\epsilon\mathbb{R}^3 | x_1^2 + x_2^2 + x_3^2 = r^2, r \in [0,1]\}$. Sea el corte del plano complejo S_R exactamente en el ecuador con el origen complejo alineado con el origen de la esfera. Es decir:

$$\mathbb{C} = \{z | z = x_1 + ix_2;\ (x_1, x_2, 0)\epsilon\mathbb{R}^3\} \qquad 3.13$$

Tomar el punto complejo $z = u + iv$, entonces la proyección P: $(u, v) \rightarrow (x_1, x_2, x_3)$ es

$$(x_1, x_2, x_3) = \left(\frac{2u}{u^2 + v^2 + 1}, \frac{2v}{u^2 + v^2 + 1}, \frac{u^2 + v^2 - 1}{u^2 + v^2 + 1}\right) \qquad 3.14$$

Y el mapeo inverso P^{-1}: $(x_1, x_2, x_3) \rightarrow (u, v)$:

$$(u, v) = \left(\frac{x_1}{1 - x_3}, \frac{x_2}{1 - x_3}\right) \qquad 3.15$$

Una característica de este mapeo es que proyecta todos los puntos fuera del círculo complejo unitario sobre el hemisferio norte y todos los puntos los cuales están adentro del hemisferio sur. Particularmente el origen complejo es proyectado al polo sur S y los puntos en el infinito dentro del polo norte N.

5.- Mapear los puntos de una esfera sobre $\mathbb{R}^3$ a una esfera de dimensión $m > 3$.

Los pasos 3 y 4 son aplicados sucesivamente para conseguir un mapeo para dimensiones más altas: Primero re-escalar el punto sobre la *m*-esfera por el factor h/d y entonces proyectarlo sobre la *(m+1)*-esfera. Esto es repetido hasta que el punto mapeado en la esfera $\mathbb{R}^{n-1}$ es alcanzado.

La proyección estereográfica del plano complejo sobre la esfera de Riemann puede ser generalizada para proyectar una *(m-1)*-esfera dentro de una *m*-esfera ϕ: $(x_1, x_2, \dots, x_m) \rightarrow (x_1, x_2, \dots, x_{m+1})$:

$$(x_1, x_2, \dots, x_{m+1}) = \left(\frac{2x_1}{s^2 + 1}, \dots, \frac{2x_m}{s^2 + 1}, \frac{s^2 - 1}{s^2 + 1}\right) \qquad 3.16$$

donde $s^2 = x_1^2 + \cdots + x_m^2$.La inversa ϕ^{-1}: $(x_1, x_2, \dots, x_{m+1}) \rightarrow (x_1, x_2, \dots, x_m)$ está definida por:

$$(x_1, x_2, \dots, x_m) = \left(\frac{x_1}{1 - x_{m+1}}, \dots, \frac{x_m}{1 - x_{m+1}}\right) \qquad 3.17$$

6.- Mover le punto mapeado de la *(n-1)*-esfera fuera de la *n*-esfera.

Un punto dentro de la esfera sobre $\mathbb{R}^{n-1}$ es convertido al agregar el vector de desplazamiento $(\delta_1, \cdots, \delta_m)$:

$$(x'_1, \cdots, x'_m) = (x_1 + \delta_1, \cdots, x_m + \delta_m) \qquad 3.18$$

donde $\delta_j = \gamma_j \frac{s^2+1}{2}$

con $s^2 = \frac{\sqrt{2}+1}{\sqrt{2}-1}$ y factor $\gamma_j = \frac{1}{\sqrt{2^j}}$, $\gamma_m = \frac{1}{\sqrt{2^{m-1}}}$

El propósito de este cambio es sacar el punto de la *(n-1)*-esfera tal que el punto es llevado al hemisferio norte sobre la *n*-esfera sobre la siguiente proyección. Particularmente, una *(n-1)*-esfera es proyectada a la superficie correspondiente al primer cuadrante de la esfera de $\mathbb{R}^n$ y es centrada sobre el punto

$$\mathbf{x_0} = \left(1, \frac{\pi}{4}, \ldots, \frac{\pi}{4}\right) \qquad 3.19$$

en $\mathbb{R}^n$

7.- Finalmente, proyectar el punto cambiado en $\mathbb{R}^{n-1}$ en el primer cuadrante de la esfera sobre $\mathbb{R}^n$.

De esta manera, cada uno de los nodos el árbol *k*-ario es asignado a un punto semilla del primer cuadrante de la superficie de la hiperesfera de *m* dimensiones, radio uno y centro en el origen. Así, el procedimiento para asignar una etiqueta a un documento o consulta consiste en encontrar el punto semilla más cercano y simplemente asignar la etiqueta correspondiente a dicha semilla.

Capítulo 4

Pruebas y Resultados

4.1 Pruebas

En este capítulo se presentan los resultados de una serie de pruebas aplicadas a nuestro proceso de almacenamiento y recuperación de información, los cuales, fueron descritos en el capítulo anterior. Las pruebas consisten en almacenar un conjunto sintético de documentos en el árbol *k*-ario, para después recuperar información mediante un conjunto de consultas, también generadas de manera sintética. Las métricas de desempeño utilizadas son el *precion* y *recall* debido a que son estándares de facto para evaluar el desempeño de sistemas de almacenamiento y recuperación de información.

Para evaluar el proceso de recuperación de documentos, se usó como ontología de entrada a Gene Ontology (Sección 4.2), de la que se construyó un escenario que comprende cien mil documentos especificados por medio de entre 11 y 17 conceptos o palabras clave de la ontología. El número de palabras clave de cada documento es entero uniformemente distribuido entre 11 y 17.

Una vez que se determinó el número de palabras clave, se procedió a elegir entre los conceptos de la ontología usando un proceso aleatorio donde se elige un concepto con la misma probabilidad sin utilizar reemplazo. Este método es igualmente usado para generar las consultas, ya que contienen conceptos de la misma ontología.

Como se vio en el capítulo anterior, nuestro proceso hace uso de las medidas de similitud semántica propuestas con el que obtenemos el vector característico *m* dimensional, y con el mapeo de ese vector en el árbol *k-ario*, obtenemos su etiqueta correspondiente. Una vez obtenido el vector característico, utilizamos la distancia coseno para saber qué tan similares son el vector de una consulta con cada vector de cada documento. De esta manera usamos una organización de los valores calculados durante el proceso de recuperación de documentos como la mostrada en la tabla 4.1.

Número de Documento	Vector Característico	Etiqueta	Similitud Coseno
Documento 1	(x, y, … , z)	0120	[0-1]

Tabla 4.1. Manera de organizar los elementos calculados durante el proceso de recuperación de documentos.

La similitud coseno registrada en la tabla anterior la usamos para calcular qué tan parecidos son los vectores de los documentos con respecto a una consulta dada. Es decir, se obtiene la similitud coseno de todos los documentos con respecto a cada consulta.

Para llevar a cabo nuestras pruebas usamos el parámetro R_0, el cual define la cantidad de elementos menores a ese parámetro, y que nos permite calcular *precision* y *recall*. Con el uso de tal parámetro podemos obtener los R_0 documentos relevantes y los R_0 documentos recuperados, los cuales son empleados en las fórmulas:

$$Precision = \frac{|\{documentos\ relevantes\} \cap \{documentos\ recuperados\}|}{|\{documentos\ recuperados\}|} \qquad 4.1$$

$$Recall = \frac{|\{documentos\ relevantes\} \cap \{documentos\ recuperados\}|}{|\{documentos\ relevantes\}|} \quad 4.2$$

Dado que en nuestras pruebas proponemos un escenario de 100 mil documentos y 200 consultas. Una vez que se ha calculado la similitud coseno de cada consulta a todos los documentos, se calcula un promedio para poder obtener el *precision* y *recall* promedio.

Además, calculamos la Medida F para conocer el balance del *precision* y *recall*:

$$F_1 = 2 \cdot \frac{precision\ \cdot recall}{precision + recall}$$

.

La cual es conocida como medida F_1, dado que el *precision* y *recall* son pesados uniformemente.

Con el propósito de tener una referencia que nos permita evaluar de manera comparativa el desempeño del algoritmo propuesto, se usó al algoritmo de Locality Sensitive Hashing para almacenar y recuperar información bajo los mismos escenarios y métricas empleadas en la técnica propuesta.

A continuación, y debido a que es un elemento central de nuestros experimentos, se presenta una breve descripción de las principales características de la ontología *Gene Ontology.*

4.1.2 Gene Ontology

La Ontología Genética (Gene Ontology [GO]) empleada en nuestras pruebas, provee un vocabulario controlado que describe el gen y los atributos del producto génico en cualquier organismo.

El proyecto GO ha desarrollado tres ontologías estructuradas que describen a los productos genéticos en términos de sus procesos biológicos, componentes celulares y funciones moleculares de una manera independiente, es decir, asigna una ontología para cada una de ellas. Existen tres aspectos separados para este proyecto: primero, el desarrollo y mantenimiento de las ontologías por si mismas; segundo, la anotación de productos genéticos, lo cual implica hacer asociaciones entre las ontologías y los genes y los productos genéticos en las bases de datos colaboradoras; y tercero, el desarrollo de herramientas que faciliten la creación, mantenimiento y uso de ontologías.

4.2 Resultados

Para la evaluación de nuestro proceso de recuperación de documento tenemos el escenario mostrado en la tabla 4.2. En el que podemos ver que los valores de orden y profundidad del árbol varían con el propósito de evaluar el desempeño en la asignación de etiquetas iguales a contenido similar.

Cantidad de consultas	200
Cantidad de documentos	100 mil
Ontología	Gene Ontology
Orden *k* del árbol *K*-ario	3, 5, 7
Profundidad *d* del árbol K-ario	3, 5, 7
Espacio *m* dimensional	3 dimensiones
Parámetro R_0	5, 10

Tabla 4.2. Escenario de evaluación.

El escenario anterior fue utilizado de la misma manera para el algoritmo de LSH (Locality Sensitve Hashing).

A continuación se muestran los resultados en donde se varía la profundidad y orden del árbol. La implementación de LSH fue desarrollada por Joren Six llamada TarsosLSH [73].En los resultados podemos ver que nuestro proceso de recuperación de información usando la medida con pesos asimétricos es comparado con TarsosLSH con sus diferentes medidas de distancia para comparar dos vectores en el espacio *m*-dimensional, como lo son: LS1, LS2, y COS. LS1 usa la distancia de haming. LS2 usa distancia Euclidiana. Y, COS usa distancia coseno.

Además, variamos el parámetro R_0, el cual nos indica la cantidad de documentos recuperados por consulta. En general se puede ver que al aumentar los valores de k y d, se disminuye el área de las regiones de la hiperesfera que reciben una misma etiqueta y por lo tanto el número de documentos que serán alojados en un nodo. De esta forma, la probabilidad de que todos los documentos relevantes hayan adquirido la misma etiqueta también disminuye, lo que se ve reflejado en una reducción en los valores del *precision* y el *recall*.

Los resultados están divididos por *precision*, *recalll*, y Medida F. Lo anterior con el propósito de poder visualizar de una manera clara el comportamiento de nuestra propuesta con respecto a LSH y sus variantes.

En las gráficas 4.1 – 4.6 se fijó a k=3 y se varió el valor de la profundidad *d* del árbol.

En las figuras 4.1 y 4.2 se puede ver el resultado para el *precision*, y en donde se puede notar que nuestro proceso de recuperación de información usando la medida con pesos asimétricos, proporciona mejores resultados que LSH y sus variantes. Además, se puede notar que al intentar usar un valor grande para *d*, la implementación de LSH no fue capaz de proporcionar un resultado.

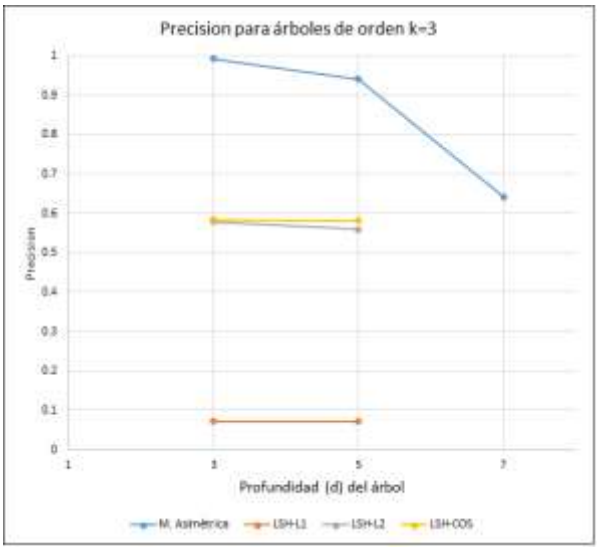


Figura 4.1. *Precision* para $R_0 = 5$ con *k=5* y variando a *d.*

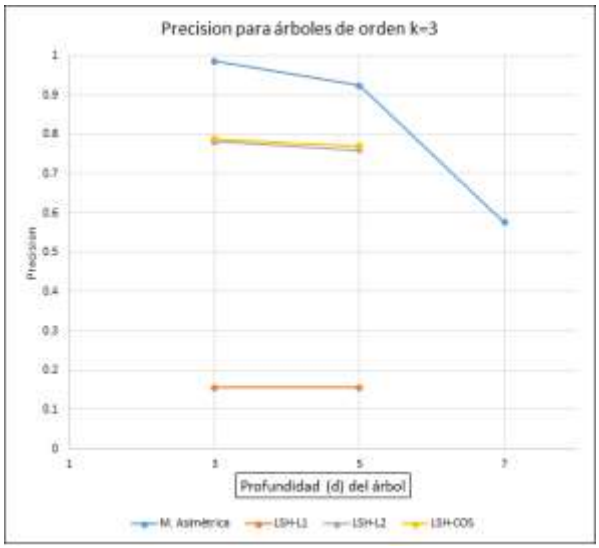


Figura 4.2. *Precision* para $R_0 = 10$ con *k=5* y variando a *d.*

En las gráficas 4.3 y 4.4 se muestran los resultados para *recall*, en donde de igual manera, nuestro proceso de recuperación es superior a LSH.

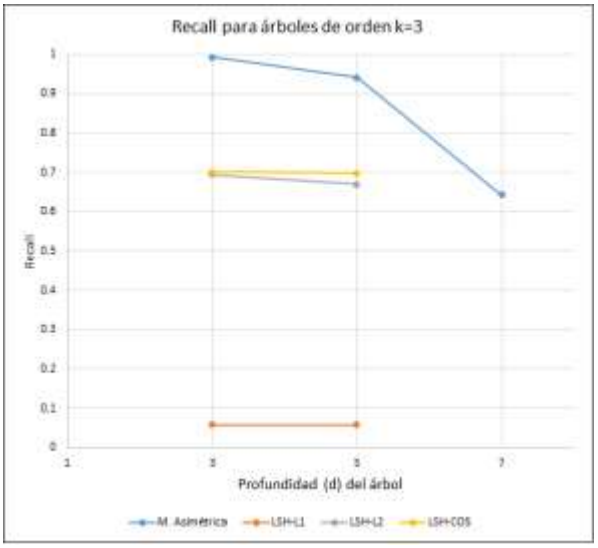


Figura 4.3. *Recall* para $R_0 = 5$ con *k=3* y variando a *d.*

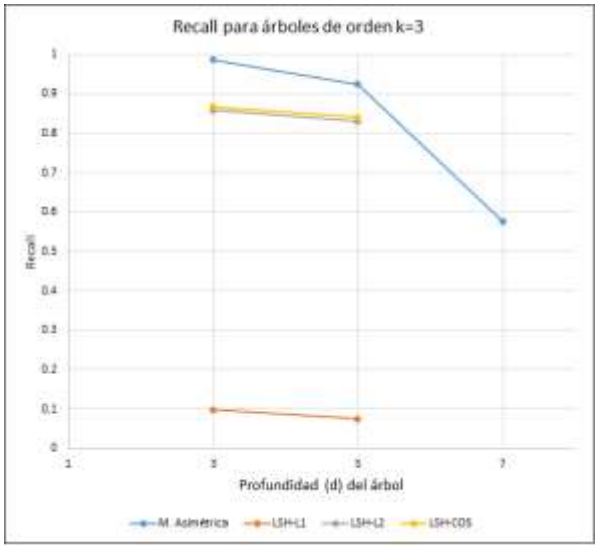


Figura 4.4. *Recall* para $R_0 = 10$ con *k=3* y variando a *d.*

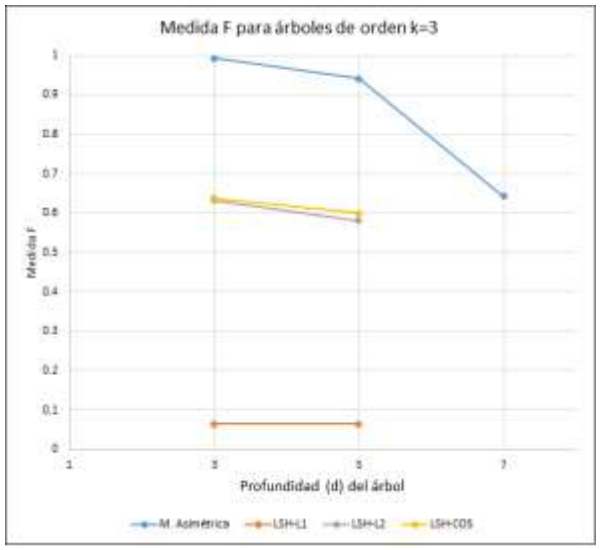


Figura 4.5. Medida F para $R_0 = 5$ con *k=3* y variando a *d.*

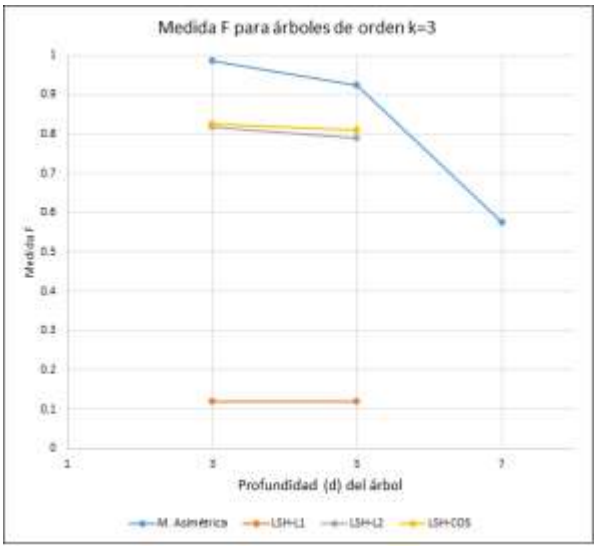


Figura 4.6. Medida F para $R_0 = 10$ con *k=3* y variando a *d.*

En las gráficas 4.7 – 4.12 se fijó a *k=5* y se varió el valor de la profundidad *d* del árbol. En ellas podemos ver que para un árbol de orden *k=5* y profundidad *d=3* la propuesta de LSH en sus diferentes versiones sí obtuvo resultados, pero aun así por debajo de nuestra propuesta. Además, podemos notar que para *precision*, *recall* y medida F, LSH no pudo escalar para árboles con profundidad *d* mayor a 3.

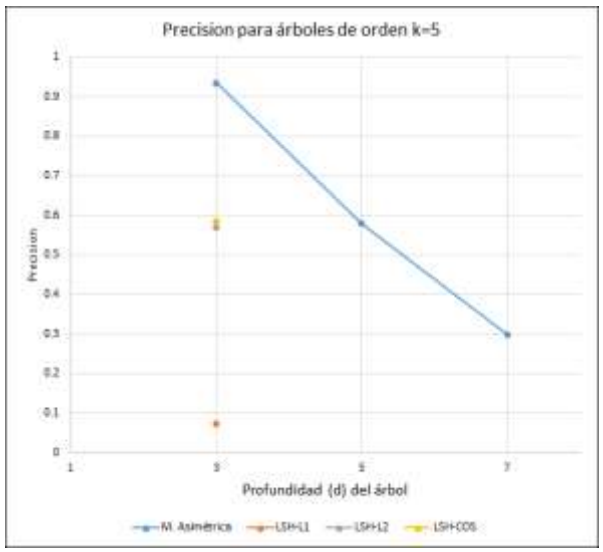


Figura 4.7. *Recall* para $R_0 = 5$ con *k=5* y variando a *d.*

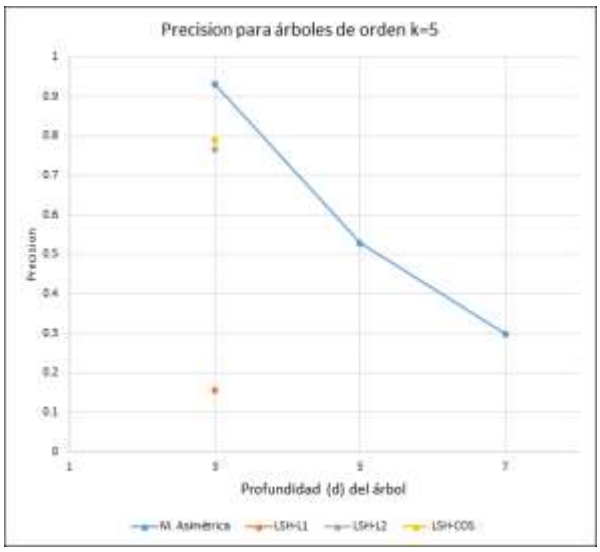


Figura 4.8. *Recall* para $R_0 = 10$ con *k=5* y variando a *d.*

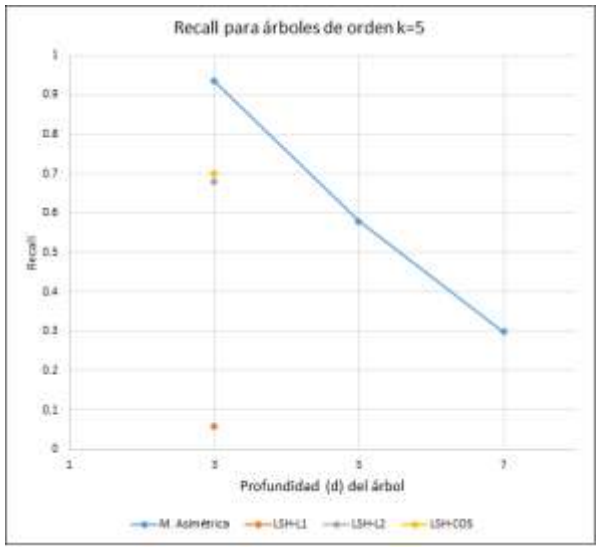


Figura 4.9. *Precision* para $R_0 = 5$ con *k=5* y variando a *d.*

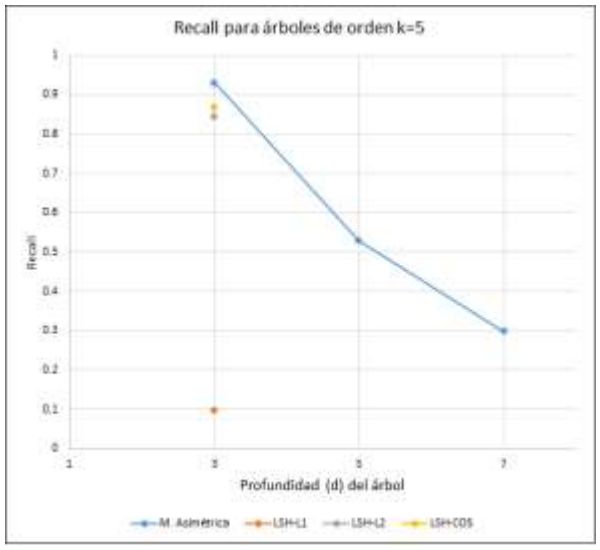


Figura 4.10. *Precision* para $R_0 = 10$ con *k=5* y variando a *d.*

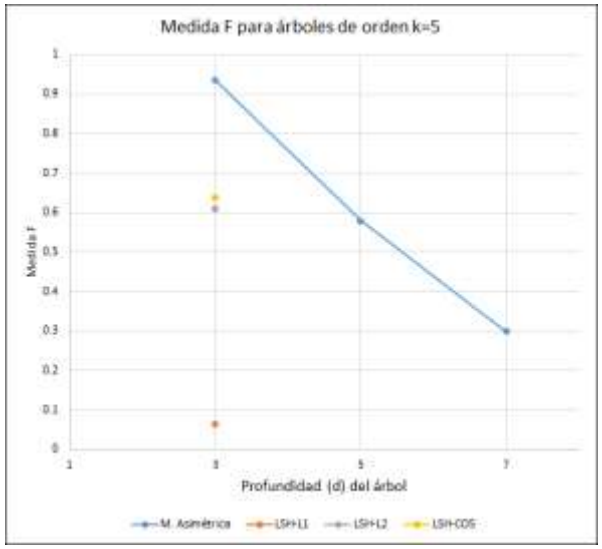


Figura 4.11. Medida F para $R_0 = 5$ con *k=5* y variando a *d.*

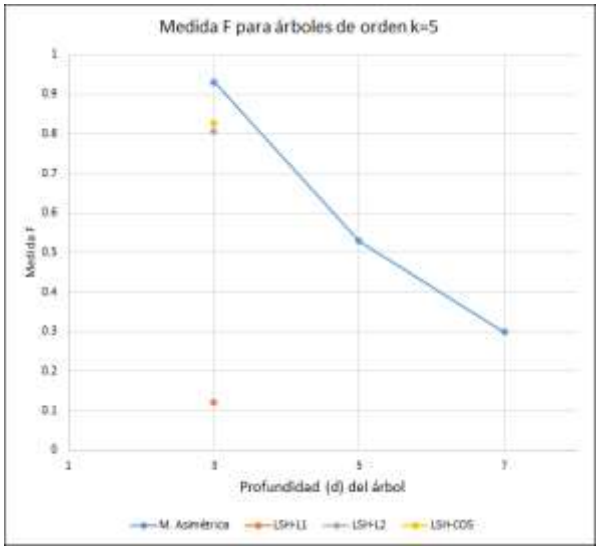


Figura 4.12. Medida F para $R_0 = 10$ con *k=5* y variando a *d.*

Las resultados para el orden del árbol fijado en *k=7* y la profundidad *d* variada, se pueden visualizar en las gráficas 4.13 – 4.18. Como se mencionó anteriormente, con el incremento de los valores de *k* y *d*, se reduce el área cubierta por cada nodo en la superficie de la hiperesfera dando como resultado que el un nodo sea representante de menos documentos. Lo anterior se puede ver reflejado para *precision* (Figura 4.13), *recall* (Figura 4.14) y medida F (4.15).

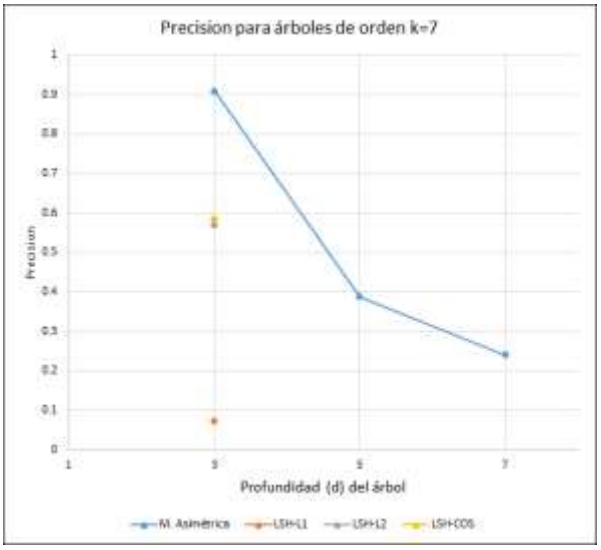


Figura 4.13. *Precision* para $R_0 = 5$ con *k=7* y variando a *d.*

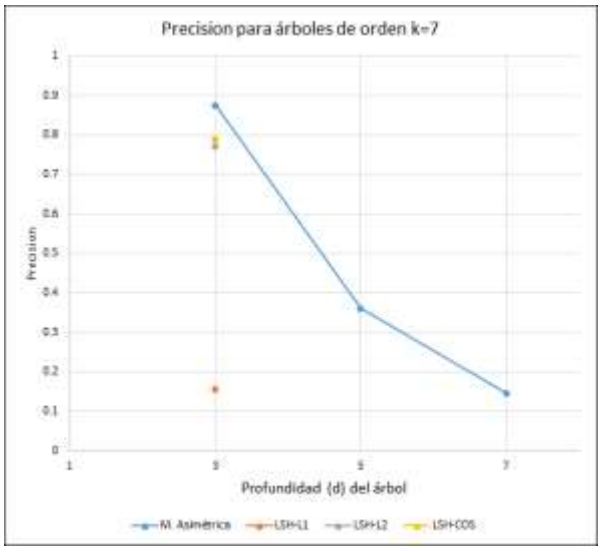


Figura 4.14. *Precision* para $R_0 = 10$ con *k=7* y variando a *d.*

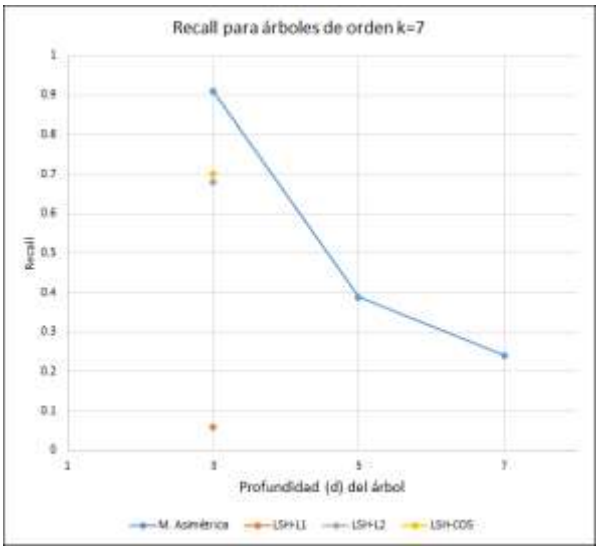


Figura 4.15. *Recall* para $R_0 = 5$ con *k=7* y variando a *d.*

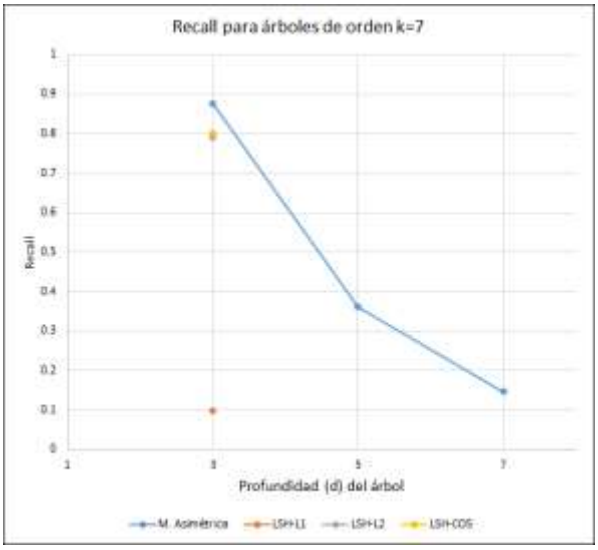


Figura 4.16. *Recall* para $R_0 = 10$ con *k=7* y variando a *d.*

La propuesta de LSH sólo puede proporcionar resultados para un árbol de *k=7* y *d=3*, mientras que con nuestra propuesta de recuperación podemos obtener información aún y cuando el orden y profundidad del árbol se incrementa.

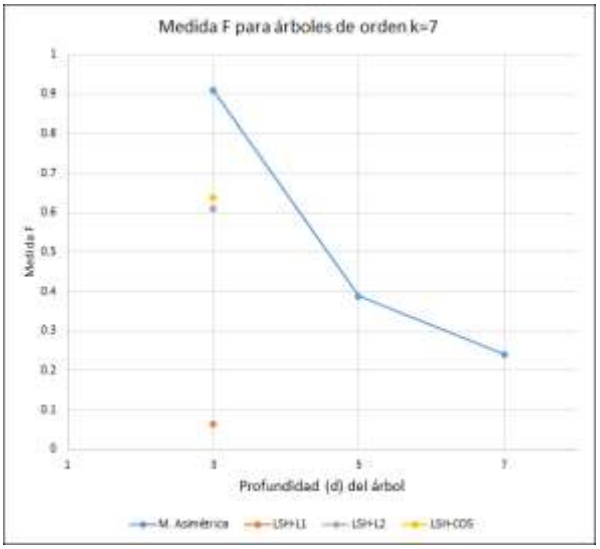


Figura 4.17. Medida F para $R_0 = 5$ con *k=7* y variando a *d.*

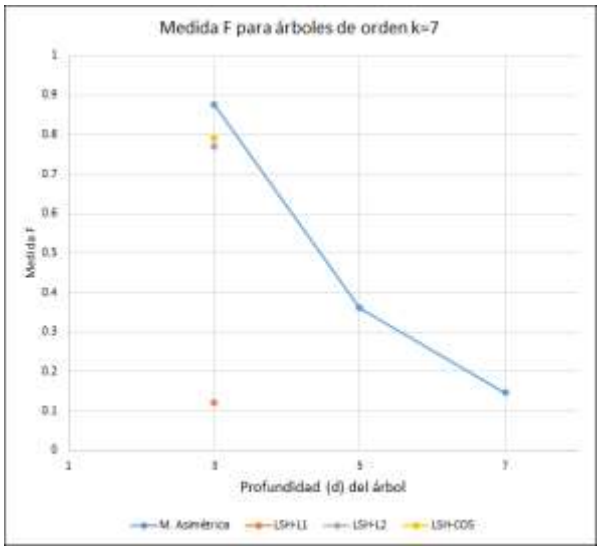


Figura 4.18. Medida F para $R_0 = 10$ con *k=7* y variando a *d.*

A continuación se muestran los resultados al dejar fija la profundidad del árbol y variar el orden del mismo. En las figuras 4.19 – 4.24, encontramos los resultados para *d=3* y *k* variado. En esta prueba se busca mostrar el comportamiento de nuestra propuesta y la propuesta de LSH cuando se varía el orden del árbol. Tanto en las gráficas de *precision* como *recall* nuestra propuesta proporciona mejores resultado que las variantes de LSH. Para este caso el algoritmo de LSH sí pudo escalar al tamaño de los árboles representados dado que son árboles pequeños.

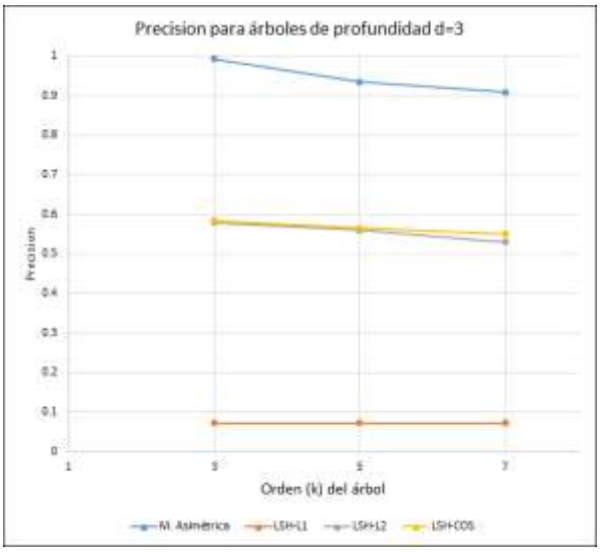


Figura 4.19. *Precision* para $R_0 = 5$ con *d=3* y variando a *k.*

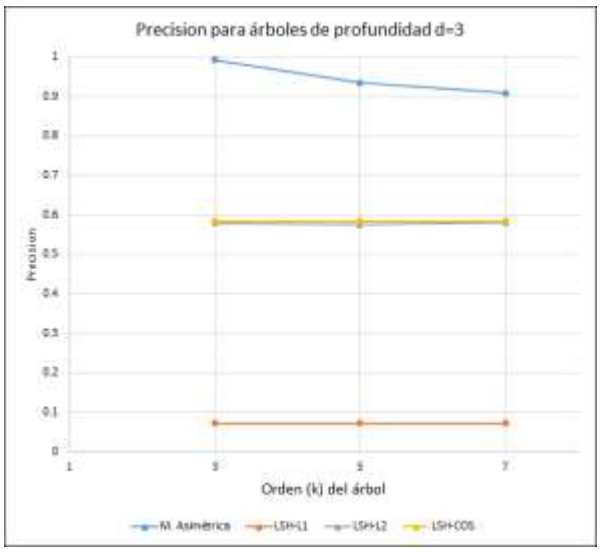


Figura 4.20. *Precision* para $R_0 = 10$ con *d=3* y variando a *k.*

Podemos apreciar que en los resultados para *recall* (Figura 4.21 y 4.22) nuestra propuesta obtiene mejores resultados de recuperación.

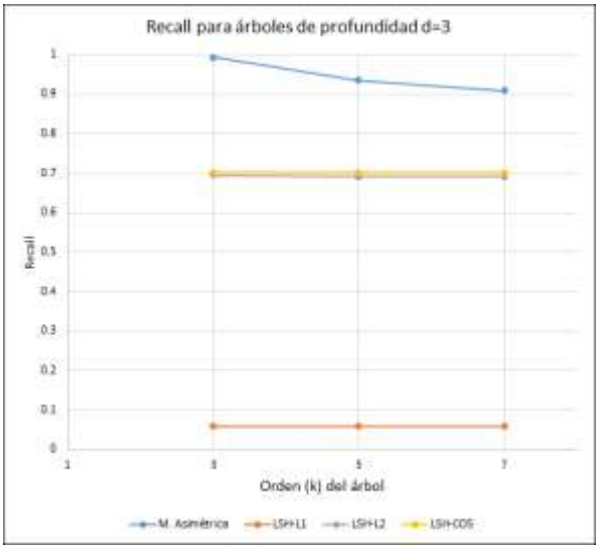


Figura 4.21. *Recall* para $R_0 = 5$ con *d=3* y variando a *k.*

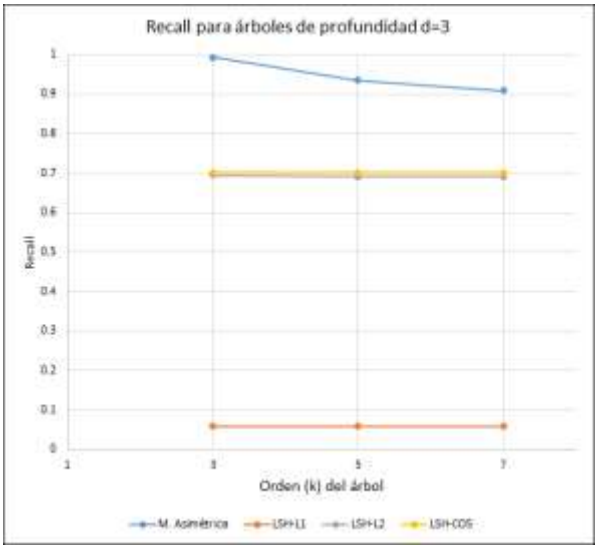


Figura 4.22. *Recall* para $R_0 = 10$ con *d=3* y variando a *k.*

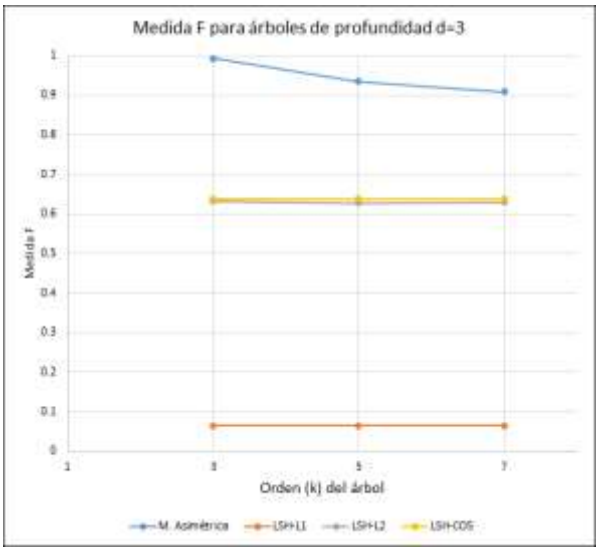


Figura 4.23. Medida F para $R_0 = 5$ con *d=3* y variando a *k.*

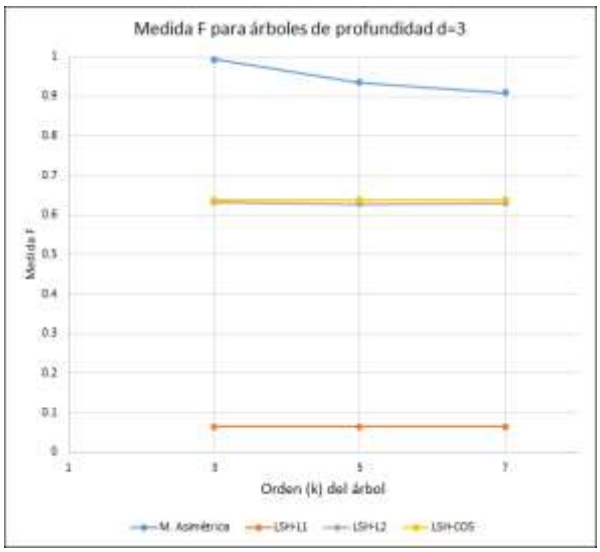


Figura 4.24. Medida F para $R_0 = 10$ con *d=3* y variando a *k.*

Para continuar con el análisis comparativo de nuestra propuesta con LSH, se fijó la profundidad del árbol en *d=5* y se varió el orden *k*, las gráficas para estos resultados los podemos encontrar en las figuras 4.25 – 4.30. En ellas podemos visualizar que nuestra propuesta continua proporcionando mejores resultados que los de LSH y sus variantes. Además, podemos notar que la implementación de LSH no escala para valores de orden *d* mayores a 3 cuando se fija a *d* en 5, sin en cambio, nuestra propuesta sigue recuperando información aún con esos cambios. Sin embargo, su rendimiento decae considerablemente cuando *d* es mayor.

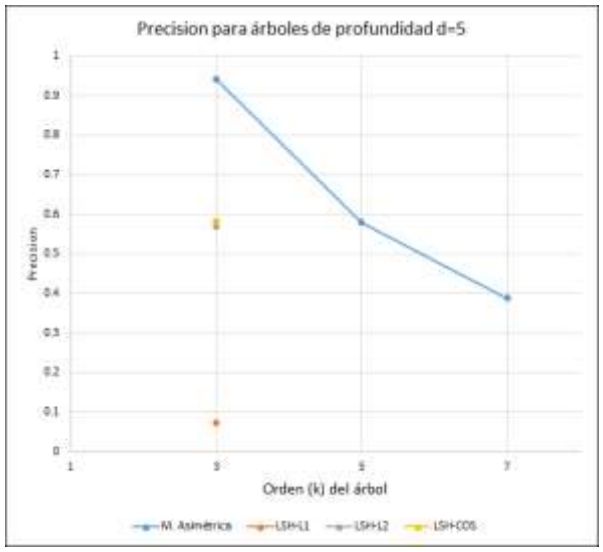


Figura 4.25. *Precision* para $R_0 = 5$ con *d=5* y variando a *k.*

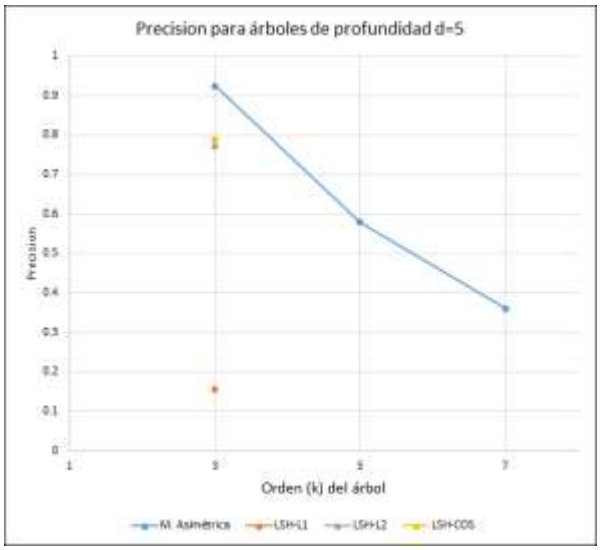


Figura 4.26. *Precision* para $R_0 = 10$ con *d=5* y variando a *k.*

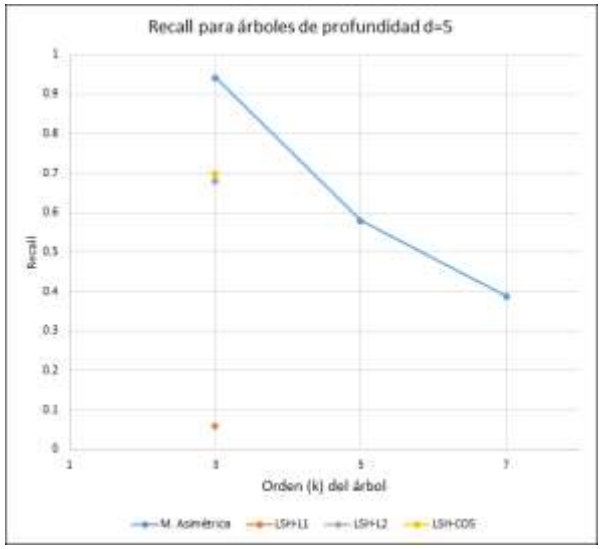


Figura 4.27. *Recall* para $R_0 = 5$ con *d=5* y variando a *k.*

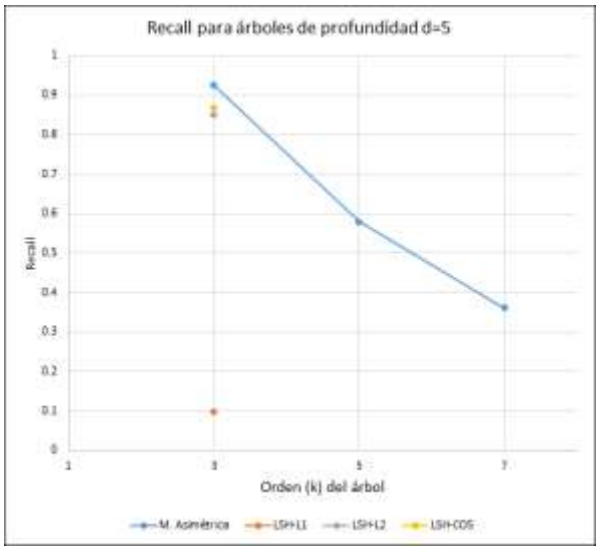


Figura 4.28. *Recall* para $R_0 = 10$ con *d=5* y variando a *k.*

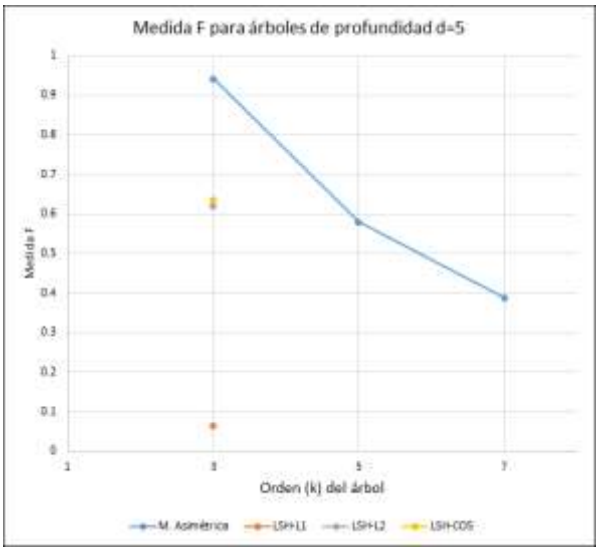


Figura 4.29. Medida F para $R_0 = 5$ con *d=5* y variando a *k.*

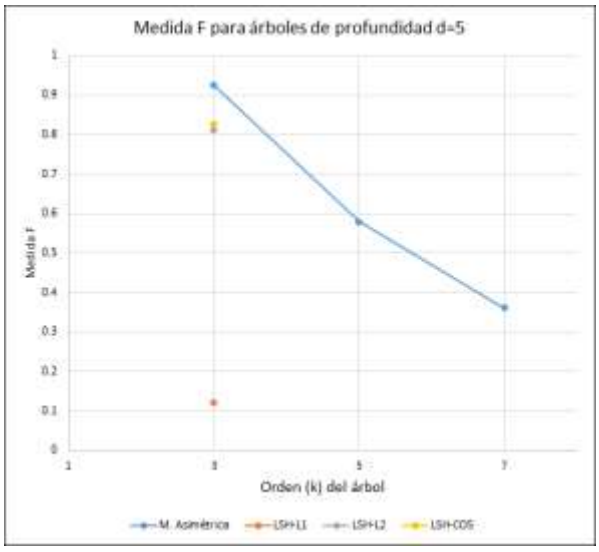


Figura 4.30. Medida F para $R_0 = 10$ con *d=5* y variando a *k.*

Finalmente, se fijó la profundidad en *d=7* y se varió el orden del árbol. En esta prueba se analizó el comportamiento de las propuestas de recuperación de información cuando los valores de *d* y *k* son grandes. En las gráficas 4.31 – 4.36, se puede notar que nuestra propuesta proporciona resultados de recuperación de aceptables a bajos mientras que con la propuesta de LSH no se pudo obtener resultados para estos valores de *k* y *d*. Tanto como para *precision* como *recall* los resultados son de medios a bajos para nuestra propuesta, esto debido a que el área cubierta por cada nodo en la superficie de la hiperespera es más pequeño y entonces representa a menos cantidad de documentos representados en el espacio.

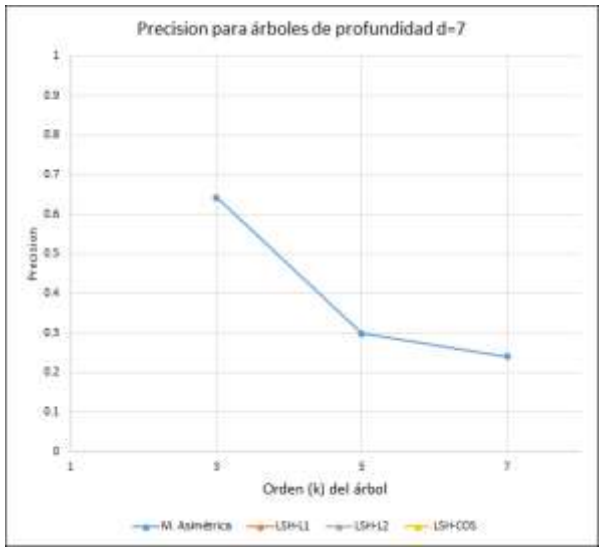


Figura 4.31. *Precision* para $R_0 = 5$ con *d=7* y variando a *k.*

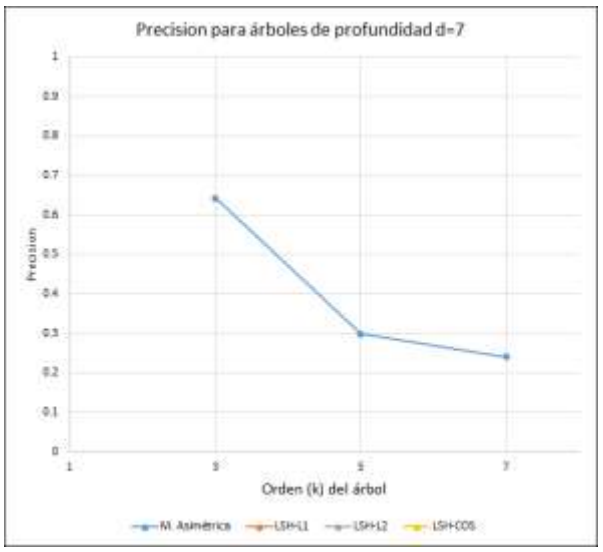


Figura 4.32. *Precision* para $R_0 = 10$ con *d=7* y variando a *k.*

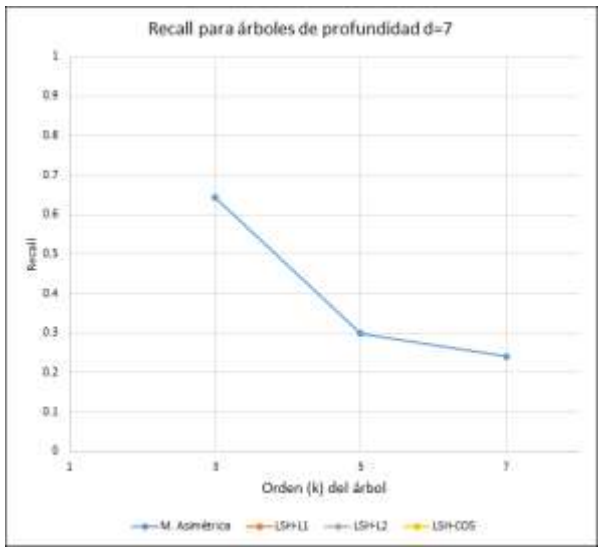


Figura 4.33. *Recall* para $R_0 = 5$ con *d=7* y variando a *k.*

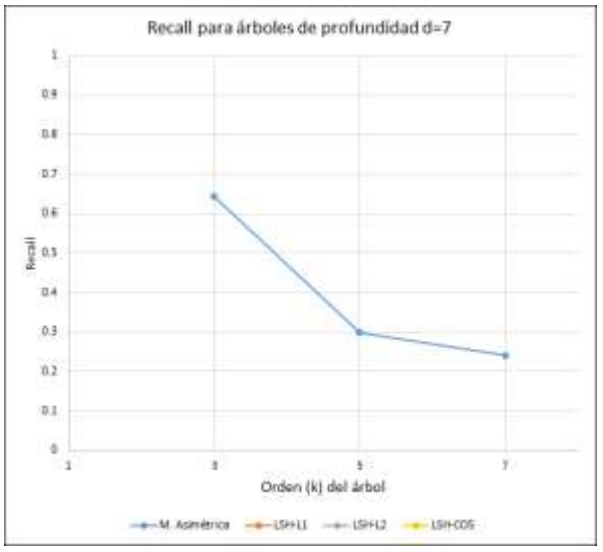


Figura 4.34. *Recall* para $R_0 = 10$ con *d=7* y variando a *k.*

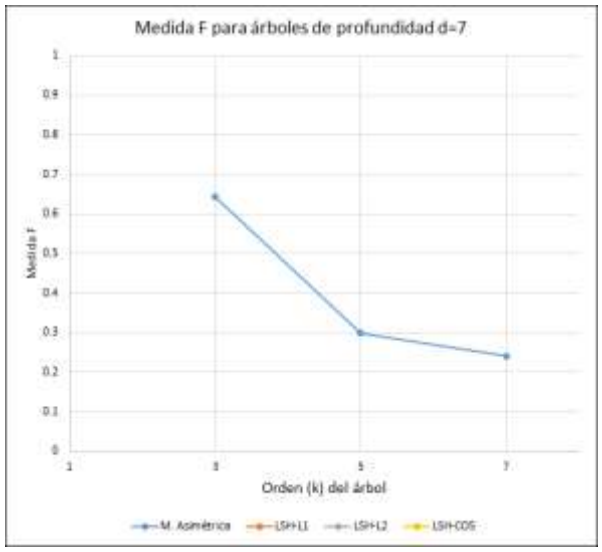


Figura 4.35. Medida F para $R_0 = 5$ con *d=7* y variando a *k.*

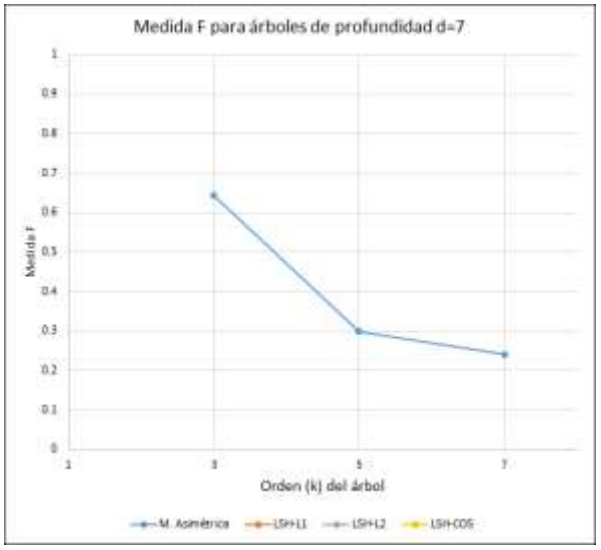


Figura 4.36. Medida F para $R_0 = 10$ con *d=7* y variando a *k.*

Capítulo 5

Conclusiones y Trabajo a futuro

5.1 Conclusiones

La creciente generación de contenido en la Internet actual, nos ha llevado a usar herramientas como Google para encontrar objetos de datos que sean relevantes a nuestras consultas. Lo anterior, ha generado una enorme dependencia de dicho servicio, el cual, al ser centralizado y estar bajo el control de una organización privada, no garantiza que siempre se tendrá acceso al contenido más relevante almacenado en Internet. Por otro lado, la conexión creciente de dispositivos a Internet conlleva a tener una red de una escala cada vez mayor lo que impone grandes retos a los servicios de internet actual. Por ejemplo, el enrutamiento tradicional requiere que las tablas de enrutamiento crezcan de manera proporcional al número de destinos, haciéndolo poco escalable. Este efecto se agrava por el hecho de que el proceso de acceder a un objeto de datos involucra la interacción de tres servicios independientes como lo son el enrutamiento, la resolución de nombres y la búsqueda de contenido. Por lo anterior, en esta tesis se presentó un esquema fundamentalmente nuevo para el funcionamiento del Internet en el cual, un único algoritmo es utilizado tanto para publicar objetos de datos en Internet, como para recuperarlos por medio de consultas semánticas basadas en palabras clave. Es importante remarcar, que este es el primer algoritmo que es capaz de proveer esa funcionalidad, sin importar si la red es fija, o dinámica como puede ser una red móvil ad hoc.

Como se ha mencionado, el cambio de paradigma propuesto en este trabajo, es eliminar la necesidad utilizar identificadores fijos para acceder computadoras, y en su lugar acceder directamente a los objetos de datos por medio de la semántica de la información que contienen. De esta manera, el medio físico donde son almacenados los objetos de datos se vuelve irrelevante y lo único que importa es su contenido. Esta línea de investigación está tomando auge cada vez mayor, ya que se prevé que la información generada en Internet siga creciendo indefinidamente. De esta manera surgen las redes orientadas a contenido, abordas en el presente trabajo en las Secciones 2.2 y 2.3. También es importante remarcar, que el presente trabajo constituye un paso hacia adelante con respecto a las redes orientadas a contenido, debido a que éstas se centran en realizar búsquedas textuales por el nombre de los objetos de datos, mientras que el presente trabajo realiza búsquedas semánticas.

A continuación se describen las principales conclusiones en torno a los resultados obtenidos en el presente trabajo de investigación, enmarcado en el ámbito de redes orientadas a contenido (CCN), con enfoque especial en enrutamiento semántico, haciendo uso de enrutamiento compacto y una novedosa función hash semántica que mapea objetos de datos a un espacio de etiquetas Σ.

El objetivo principal que planteó alcanzar esta tesis, fue diseñar, implementar y caracterizar experimentalmente un algoritmo de enrutamiento semántico capaz de recuperar objetos de datos almacenados en una red de computadoras con base en la semántica de los datos. Con este fin en el Capítulo 3 se describe nuestra arquitectura de enrutamiento semántico, la cual toma en cuenta la semántica de la información para poder brindar la información igual o similar a consultas hechas a la red.

Una vez desarrollada dicha arquitectura, se procedió a realizar pruebas con diferentes escenarios, con las que podemos concluir que nuestra arquitectura:

- Es fundamentalmente un nuevo tipo de red la cual actúa como un repositorio de datos completamente distribuido con servicios no centralizados, como lo son el servidor de nombres DNS, y el buscador de información. Lo cual la hace independiente del servicio de búsqueda. Además, soporta consultas tomando en cuenta la semántica de la misma. Por otra parte, se enfoca principalmente a la información almacenada en la red sin importar el lugar físico donde se encuentre.

  Bajo el esquema desarrollado, el proceso de publicar un nuevo objeto de datos en la red consiste en enrutar la solicitud de publicación hacía el nodo de la red que tenga la etiqueta de prefijo más cercana, a la etiqueta de prefijo asignada por la función hash semántica al objeto de datos. Por su parte, las consultas generadas por los usuarios son transformadas en etiquetadas basadas en prefijos y enrutadas de manera distribuida hacía los nodos que contienen información semánticamente relevante. Es importante notar, que el cálculo de las funciones hash se realiza en los extremos de la red, descargando totalmente a los dispositivos de red. Lo anterior es sumamente importante porque favorece la escalabilidad del sistema.
- El protocolo distribuido de elección de líder y etiquetado de la red establece un orden total sobre los nodos de la red con base en etiquetas basadas en prefijos y a las relaciones padre-hijo establecidas durante una búsqueda por amplitud que tiene como raíz al líder de la red. La relación de orden total es simplemente el orden lexicográfico de las etiquetas asignadas los nodos.
- El hecho de que todas las etiquetas asignadas a los nodos sean comparables, permite enrutar hacia cualquier nodo en la red debido a que siempre es posible determinar qué vecino tiene la etiqueta más cercana a la etiqueta del nodo destino. Más aún, debido a esta propiedad, es posible demostrar que las rutas calculadas por el protocolo de enrutamiento son libres de ciclos. Por otro lado, debido a que un nodo sólo necesita conocer las etiquetas de sus uno-vecinos, el tamaño de las tablas de enrutamiento es independiente del tamaño de la red por lo que el esquema propuesto puede considerarse como un algoritmo de enrutamiento compacto.
- La medida de distancia semántica desarrollada está específicamente diseñada para aplicaciones de Recuperación de Información (IR), ya que funciona de manera asimétrica, es decir, considerando el Contenido de Información (IC) de cada concepto. Además, tiene la capacidad de considerar la pérdida de especificidad al calcular la distancia semántica entre dos conceptos.
- Se diseñó una medida de distancia semántica entre conjuntos de conceptos. Dicha medida de distancia funciona por medio del mapeo de los conjuntos de conceptos a un espacio *m*-dimensional. El espacio *m*-dimensional está estructurado de forma tal que cada dimensión está especificada por otro conjunto de conceptos que tienen la propiedad de que su distancia al concepto raíz es la misma. Una vez que los conjuntos de conceptos han sido

mapeados a un punto del espacio, la distancia entre ambos conjuntos se calcula por medio de la distancia coseno entre los mismos.

- Se implementó un algoritmo que toma como entrada un árbol *k*-ario de altura *h* y empotra cada uno de los nodos que lo componen en la superficie de una hiperesfera de *m* dimensiones, de radio uno con centro en el origen. Este empotramiento se utiliza para asignar etiquetas de prefijo tanto a los objetos de datos como a las consultas. El procedimiento consiste en tomar las palabras clave que definen al objeto de datos o consulta, encontrar su vector característico en el espacio *m*-dimensional y después determinar el punto correspondiente al nodo empotrado más cercano.
- Al integrar los algoritmos que calculan el vector característico de objetos de datos (consultas) con el que asigna etiquetas basadas en prefijos a puntos de la superficie del primer cuadrante de la hiperesfera de radio uno, se completa el procedimiento que implementa la función hash semántica que toma como entrada palabras clave y devuelve etiquetas basadas en prefijos. La propiedad principal de esta función hash es que objetos de datos (consultas) con vectores característicos con distancia coseno baja, tenderán a tener etiquetas cercanas en el espacio de las etiquetas. De esta forma, los objetos de datos con contenido similar serán almacenados, ya sea en el mismo nodo de la red, o en nodos cercanos. De forma análoga, las consultas serán enrutadas hacia los nodos que contengan objetos de datos semánticamente similares a lo especificado por las palabras clave utilizadas en la consulta.
- Finalmente, una serie de experimentos muestran que los algoritmos presentados durante esta tesis son efectivos para recuperar información relevante a partir de consultas estructuradas en palabras clave. Lo anterior es particularmente cierto, cuando la estructura de árbol utilizada para almacenar los objetos de datos es de tamaño reducido. Esto se debe principalmente al hecho de que tener un mayor número de nodos reduce considerablemente el número de objetos de datos que son almacenados en cada uno de ellos. Además, con la comparación con el funcionamiento de la propuesta de LSH (Locality Sensitive Hashing), se puede ver que nuestro enfoque de recuperación de información obtiene un mejor rendimiento que LSH usando sus diferentes medidas de distancias (haming, Euclidiana, y coseno).

Otro aspecto importante a resaltar es que nuestra propuesta puede ser implementada también en servicios de almacenamiento en la nube, ya que comparten la característica de almacenar y recuperar información indistintamente del lugar donde se almacene/recupere. Más aún, dada la naturaleza totalmente distribuida de los algoritmos propuestos, es posible implementar nubes móviles compuestas de conjuntos dinámicos de nodos.

5.2 Aportaciones

Dentro de las principales contribuciones de este trabajo puntualizamos.

- Desarrollo de la primera arquitectura de red centrada en la semántica del contenido que combina algoritmos de enrutamiento eficientes y escalables con una función hash semántica para proveer enrutamiento semántico. A diferencia de los esquemas tradicionales que enrutan a direcciones IP, o a nombres en el caso de ICN, nuestra propuesta es capaz de encontrar el nodo en la red que almacena los objetos de datos semánticamente más relevantes para una consulta dada.
- Medidas de distancia semántica orientadas a la recuperación de información.
- Protocolo de enrutamiento compacto escalable desarrollado en el simulador de NS-2
- Una medida de similitud semántica basada entre conjuntos de conceptos. La mayoría de las medidas de similitud tradicionales están enfocadas en pares de conceptos.
- Una API desarrollada Java con métodos específicos para:
    - Carga de una ontología
    - Generación de consultas o documentos
    - Calcular el vector característico de consultas o documentos
    - Obtener la etiqueta de una consulta o documento

5.3 Limitaciones y trabajo a futuro

Una de las principales limitantes del trabajo es que la media de similitud semántica está específicamente diseñada para taxonomías. Aun y cuando esto es aceptable para aplicaciones de recuperación de información, es interesante extender los conceptos desarrollados aquí para permitir ontologías de topología arbitraría, así como con diferentes tipos de relaciones. En este mismo sentido, es importante realizar pruebas con otro tipo de ontologías como son SNOMED-CT y WORDNET.

Otra de las principales limitaciones del trabajo actual es que las palabras clave que definen tanto a los objetos de datos como a las consultas deben estar definidas como conceptos de la ontología. Por lo tanto, es importante incorporar un algoritmo de pre-procesamiento que sea capaz de asignar el mejor concepto de la ontología a la palabra clave introducida por el usuario. De esta forma, se eliminarán problemas relacionados con el hecho de que el usuario puede introducir palabras clave mal escritas y que por lo tanto no aparecerán en la ontología.

Desde el punto de vista del esquema de enrutamiento, es importante desarrollar nuevos algoritmos que no requieran la selección de un nodo líder o raíz. Lo anterior se debe a que las estructuras de enrutamiento generadas a partir de un único nodo tienen a concentrar el tráfico en las regiones de la red cercanas a él. Adicionalmente, es importante desarrollar nuevas estrategias que faciliten la replicación de objetos de datos a lo largo de la red y que además aprovechen el hecho de que existan diferentes réplicas de un objeto para mejorar el tiempo de respuesta y en general la escalabilidad del sistema.

Referencias

[1]. M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica et al., “Above the clouds: A berkeley view of cloud computing,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, 2009.

[2]. Sampath, D.; Garcia-Luna-Aceves, J.J. Sensor, Mesh and Ad Hoc Communications and Networks, 2009. SECON '09. 6th Annual IEEE Communications Society Conference on. 22-26 June 2009

[3]. T. Clausen and P. Jacquet. Optimized link state routing protocol (olsr). In RFC Editor, United States, 2003.

[4]. P. Gupta and P.R. Kumar. Critical power for asymptotic connectivity. Decision and Control, 1998. Proceedings of the 37th IEEE Conference on, 1:1106–1110 vol.1, 1998.

[5]. Xingde Jia. Wireless networks and random geometric graphs. Parallel Architectures, Algorithms and Networks, 2004. Proceedings. 7th International Symposium on, pages 575–579, May 2004.

[6]. David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In Mobile Computing, pages 153–181. Kluwer Academic Publishers, 1996.

[7]. Mathew Penrose. Random Geometric Graphs (Oxford Studies in Probability). Oxford University Press, USA, July 2003.

[8]. C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (aodv) routing. In RFC Editor, United States, 2003.

[9]. Garcia-Luna-Aceves, J.J. Mobile Adhoc and Sensor Systems, 2009. MASS '09. IEEE 6th International Conference on. 12-15 Oct. 2009.

[10]. T. Ballardie, P. Francis, and J. Crowcroft. Core based trees (cbt). SIGCOMM Comput. Commun. Rev., 23(4):85-95 , 1993.

[11]. J. 1. Garcia-Luna-Aceves and E. L. Madruga. The core-assisted mesh protocol. *IEEE Journal on Selected Areas in Communications,* 17(8):1380-1394, Aug 1999.

[12]. Antony Rowstron1 and Peter Druschel “Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems” Published in: Proceeding Middleware '01 Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg Pages 329-350

[13]. Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz, “Tapestry: A Resilient Global-scale Overlay for Service Deployment”, IEEE Journal on Selected Areas In Communications, VOL. 22, NO. 1, JANUARY 2004

[14]. B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, “Tapestry: An infrastructure for fault-tolerant wide-area location and routing,” Tech. Rep. CSD-01-1141, U. C. Berkeley, Apr 2001.

[15]. Matthew J. B. Robshaw, “MD2, MD4, MD5, SHA and other hash functions,” Tech. Rep. TR-101, RSA Labs, 1995, v. 4.0.

[16]. Y. Rekhter and T. Li, An Architecture for IP Address Allocation with CIDR, 1993, RFC 1518, http://www.isi.edu/in-notes/rfc1518.txt.

[17]. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker: “A scalable content-addressable network”. Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications. Pages 161-172.

[18]. P. Francis. Yoid: Extending the Internet Multicast Architecture. Unpublished paper, available at http://www.aciri.org/yoid/docs/index.html, Apr. 2000.

[19]. Gnutella. http://gnutella.wego.com.

[20]. Napster. http://www.napster.com.

[21]. Amitava Biswas, Suneil Mohan, Rabi Mahapatra “Optimization of Semantic Routing Table” Computer Communications and Networks, 2008. ICCCN '08. Proceedings of 17th International Conference on

[22]. Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Shenker, S., “A scalable content-addressable network”, SIGCOMM'01, 2001.

[23]. Adjie-Winoto, W., Schwartz, E., Balakrishnan, H., and Lilley, J., “The design and implementation of an intentional naming system”, 17th ACM Symposium on Operating Systems Principles (SOSP ’99), Published as Operating Systems Review, pp 34(5):186–201, 1999.
[24]. Waterhouse, S., Doolin, D.M., Kan, G. and Faybishenko, Y., “Distributed search in peer-to- peer networks”, IEEE Internet Computing, 2002.
[25]. Li, J., and Vuong, S., “A scalable semantic routing architecture for grid resource discovery”, 11th Int. Conference on Parallel and Distributed Systems, pp. 29-35, Vol. 1, July 2005.
[26]. Tempich, C., Staab, S., and Wranik, A., “'REMINDIN': Semantic Query Routing in Peer-to-Peer Networks Based on Social Metaphors”, 13th Int. World Wide Web Conference, pp. 640-649, 2004.
[27]. Yang, D. and Powers, D. M., “Measuring semantic similarity in the taxonomy of WordNet”, 28th Australasian Conference on Computer Science - Volume 38. 2005.
[28]. Hariri, B.H, Abolhassani, H and Khodaei, A, A new Structural Similarity Measure for Ontology Alignment. SWWS 2006, 36-42, 2006.
[29]. Lemaire, B. and Denhière, G., “Incremental Construction of an Associative Network from a Corpus”, 26th Annual Meeting of the Cognitive Science Society (CogSci'2004), pp 825-830, 2004.
[30]. Rajapske, R. and Denham, M., “Text retrival with more realistic concept matching and reinforcement learning”, Information Processing and Management, 42, pp. 1260-1275, 2006.
[31]. Rajapske, R, Denham, M., “Fast Access to Concepts in Concept Lattices via Bidirectioanl Associative Memory”, Neural Computation, 17, pp-2291-2300, 2005.
[32]. Watts, D.J., “Networks, Dynamics, and the Small-World Phenomenon”, The American Journal of Sociology, Vol. 105, No. 2, pp. 493-527, 1999.
[33]. Chuiwei Lu, “A Fuzzy Search Algorithm for Structured P2P Network Based on Multi-dimensional Semantic Matrix”, Journal of Networks, Vol 7, No 2 (2012), 377-384, Feb 2012
[34]. Federica Mandreoli, Riccardo Martoglia, Wilma Penzo, and Simona Sassatelli, “Semantic Routing for Effective Search in Heterogeneous and Distributed Digital Libraries”, IRCDL, page 67-70. DELOS: a Network of Excellence on Digital Libraries, (2007)
[35]. A. Halevy, Z. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov, “The Piazza Peer Data Management System,” IEEE TKDE, vol. 16, no. 7, pp. 787–798, July 2004.
[36]. W. Nejdl, B. Wolf, S. Staab, and J. Tane, “EDUTELLA: Searching and Annotating Resources within an RDF-based P2P Network.” in Proc. of WWW Intl. Workshop on the Semantic Web, 2002.
[37]. M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. Miller, and J. Mylopoulos, “The hyperion project: from data integration to data coordination,” SIGMOD Record, vol. 32, no. 3, pp. 53–58, 2003.
[38]. I. Tatarinov and A. Halevy, “Efficient Query Reformulation in Peer Data Management Systems,” in Proc. of SIGMOD, 2004.
[39]. F. Mandreoli, R. Martoglia, S. Sassatelli, and W. Penzo, “SRI: Exploiting Semantic Information for Effective Query Routing in a PDMS,” in Proc. of WIDM, 2006.
[40]. A. Crespo and H. Garcia-Molina, “Routing indices for peer-to-peer systems,” in Proc. of ICDCS, 2002.
[41]. F. Mandreoli, R. Martoglia, and P. Tiberio, “Approximate Query Answering for a Heterogeneous XML Document Base,” in Proc. of WISE, 2004.
[42]. T. Koponen et al., “A Data-Oriented (and Beyond) Network Architecture,” Proc. SIGCOMM ’07, Kyoto, Japan, Aug. 27–31, 2007.
[43]. V. Jacobson et al., “Networking Named Content,” Proc. CoNEXT, Rome, Italy, 2009, http://doi.acm.org/10.1145/1658939.1658941, pp. 1–12.
[44]. M. Ain et al., “D2.3 – Architecture Definition, Component Descriptions, and Requirements,” Deliverable, PSIRP 7th FP EU-funded project, Feb. 2009.
[45]. B. Ahlgren et al., “Second NetInf Architecture Description,” 4WARD EU FP7 Project, Deliverable D-6.2 v2.0, Apr. 2010, FP7-ICT-2007-1-216041- 4WARD / D-6.2, http://www.4ward-project.eu/.
[46]. A. Ghodsi et al., “Naming in Content-Oriented Architectures,” Proc. ACM SIGCOMM Wksp. Information-Centric Networking, Toronto, Canada, Aug. 2011.
[47]. W. Chai et al., “Curling: Content-Ubiquitous Resolution and Delivery Infrastructure for Next-Generation Services,” IEEE Commun. Mag., vol. 49, no. 3, 2011; http://discovery.ucl.ac.uk/1305428/, pp. 112–20.

[48]. K. Fall, “A Delay-Tolerant Network Architecture for Challenged Internets,” Proc. ACM SIGCOMM ’03, Karlsruhe, Germany, Aug. 25–29, 2003, pp. 27–36.
[49]. C. Dannewitz et al., “Secure Naming for A Network of Information,” Proc. 13th IEEE Global Internet Symp.’10, San Diego, CA, Mar. 2010.
[50]. S. Farrell et al., “Naming Things with Hashes,” IETF Internet draft draft-farrell-decade-ni, work in progress, Apr. 2012; http://tools.ietf.org/html/draft-farrell-decade-ni.
[51]. M. D’Ambrosio et al., “MDHT: A Hierarchical Name Resolution Service for Information-Centric Networks,” Proc. ACM SIGCOMM Wksp. Information-Centric Networking, Toronto, Ontario, Canada: ACM, 2011; http://doi.acm.org/10.1145/2018584.2018587, pp. 7–12.
[52]. A. Eriksson and B. Ohlman, “Scalable Object-to-Object Communication Over A Dynamic Global Network,” Proc. Future Network and Mobile-Summit ’10, June 2010.
[53]. M. D’Ambrosio et al., “Providing Data Dissemination Services in the Future Internet,” Proc. WTC ’08, New Orleans, LA, Dec. 2008, at IEEE GLOBECOM ’08.
[54]. M. Bawa, T. Condie, and P. Ganesan. LSH Forest: Self-Tuning Indexes for Similarity Search. In *WWW’05: Proc. of the 14th int. conference on World Wide Web*, 2005.
[55]. A. Broder, S. Glassman, M. Manasse, and G. Zweig. Syntactic Clustering of the Web. In Selected papers from the sixth int. conference on World Wide Web, 1997.
[56]. M. S. Charikar. Similarity Estimation Techniques from Rounding Algorithms. In STOC’02: Proc. of the thirty-fourth ACM symposium on theory of computing, 2002.
[57]. M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In SCG’04: Proc. of the twentieth symposium on computational geometry, 2004.
[58]. D. Cai and X. Hee. Orthogonal Locality Preserving Indexing. In Proc. of the 28th conference on Research and development in IR, 2005.
[59]. S. Deerwester, S. Dumais, T. Landauer, G. Furnas, and R. Harshman. Indexing by Latent Semantic Analysis. Journal of the American Society of Information Science, 41(6):391–407, 1990.
[60]. C. Eckart and G. Young. The Approximation of one Matrix by Another of Lower Rank. Psychometrika, 1:211–218, 1936.
[61]. X. He, D. Cai, H. Liu, and W.-Y. Ma. Locality Preserving Indexing for Document Representation. In Proc. of the 27th conference on research and development in IR, 2001.
[62]. G. Hinton and R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. Science, 313:504–507, 2006.
[63]. T. Hofmann. Unsupervised Learning by Probabilistic Latent Semantic Analysis. Machine Learning, 42:177–196, 2001.
[64]. H. Yang and J. Callan. Near-Duplicate Detection by Instance-level Constrained Clustering. In Proc. of the 29th conference on research and development in IR, 2006.
[65]. M. Henzinger. Finding Near-Duplicate Web Pages: a Large-Scale Evaluation of Algorithms. In Proc. of the 29th conference on research and development in IR, 2006.
[66]. I. Jolliffe. Principal Component Analysis. Springer, 1996.
[67]. J. Kruskal. Multidimensional Scaling by Optimizing Goodness of Fit to a Nonmetric Hypothesis. Psychometrika, 29(1), 1964.
[68]. B. Stein. Fuzzy-Fingerprints for Text-Based IR. In Proc. Of the 5th Int. Conference on Knowledge Management, Graz, Journal of Universal Computer Science, 2005.
[69]. R. Weber, H. Schek, and S. Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-dimensional Spaces. In Proc. of the 24th VLDB conference, 1998.
[70]. A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. In The VLDB Journal, 1999.
[71]. R. Ando and L. Lee. Iterative Residual Rescaling: An Analysis and Generalization of LSI. In Proc. 24th conference on research and development in IR, 2001.
[72]. Singhal, Amit. "Modern information retrieval: A brief overview." *IEEE Data Eng. Bull.* 24.4 (2001): 35-43.
[73]. Joren Six. TarsosLSH – Locality Sensitive Hashing (LSH) in Java. https://github.com/JorenSix/TarsosLSH