Universidad Andrés Bello

FACULTAD DE INGENIERÍA
INGENIERÍA CIVIL INFORMÁTICA

Una arquitectura caché para redes ICN basada en comportamiento de usuario

Tesis para optar al grado de ingeniero civil en informática.

Autor:
Mathias Nicolas Velilla Brandau.

Profesores guía:
Carlos Gomez-Pantoja
Miguel Gutierrez

Santiago de Chile, Chile.

Junio, 2017

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Motivación

El nacimiento del Internet en el año 1958 en EE.UU. a través de ARPA (*Advanced Researchs Proyects Agency*) tuvo como objetivo la comunicación entre diversas entidades con propósitos investigativos. La aparición de la conmutación por paquetes implementadas en los mecanismos de comunicación sentó las bases para el desarrollo de esta nueva tecnología, la cual hoy en día ha revolucionado la informática y las comunicaciones como ninguna otra cosa convirtiéndose en una herramienta de índole mundial. Un mecanismo el cual nos permite diseminar información de manera inmediata, generando un medio de colaboración e interacción entre las personas y los ordenadores, desconociendo su ubicación física.

Durante los años 2000 - 2008 el uso diario del Internet en las personas americanas rondaba entre un 60 % y 70 % [23]. Este crecimiento en el uso de la Internet, conlleva también a un aumento de la información presente en la web, dando como resultado el desarrollo de aplicaciones las cuales deben interactuar con un gran numero de usuarios y analizar una sobresaliente cantidad de información. Por este motivo Internet ha sido impulsado por demandas de aplicaciones cada vez más emergentes y capacidades de las nuevas redes de comunicación, ha convertirse en un mosaico arquitectónico que resulta en una creciente complejidad y vulnerabilidades imprescindibles, entregando violaciones de capas, proliferación de subcapas, y la erosión del modelo de extremo a extremo. Es bajo esta eventualidad que todos los cambios efectuados concluyen en un aumento de la complejidad, lo cual se traduce en una Internet osificada. Es por esta razón que los problemas anteriormente señalados no se encuentran directamente relacionados con los protocolos o mecanismo específicos del Internet actual. Más bien son causados esencialmente por la incapacidad de integrar nuevos mecanismos. Esto

quiere decir que los problemas son causados por la arquitectura de Internet y podrían ser resueltos con un nuevo diseño de arquitectura de Internet [24].

La creciente demanda de una distribución altamente escalable y eficiente del contenido ha motivado el desarrollo de futuras arquitecturas de Internet basadas en objetos de datos nombrados (NDO, por sus siglas en inglés), por ejemplo, páginas web, videos, documentos u otras informaciones. El enfoque de estas arquitecturas se conoce comúnmente como redes centradas en información (ICN). Por el contrario, las redes actuales se centran en el *host*, donde la comunicación se basa en *hosts* nombrados, por ejemplo, servidores web, PC, portátiles, teléfonos móviles y otros dispositivos [6].

Las arquitecturas ICN aprovechan el almacenamiento en red para el almacenamiento en *caché*, la comunicación multilateral a través de la replicación y los modelos de interacción que desacoplan a los remitentes y receptores. El objetivo común es lograr una distribución eficiente y confiable de los contenidos en donde cada uno de estos es nombrado única e independientemente desde la ubicación del productor (servidor), facilitando el almacenamiento en *caché* en los nodos intermediarios. Así, por ejemplo, los consumidores solicitarán contenidos enviando el denominado paquete de interés, el cual lleva el nombre del contenido, mientras que el productor o cualquier nodo que mantenga una copia del contenido puede responder a la petición realizada [6].

A su vez, los usuarios ya antes mencionados como consumidores, realizan peticiones en la web, las cuales siguen una conducta dinámica caracterizándose por un elevado sesgo entre los diferentes conjuntos de peticiones. En otras palabras, dentro del universo de peticiones generadas existen conjuntos de peticiones que son regularmente solicitadas por los consumidores en intervalos de tiempos distintos, por lo contrario, otras peticiones escasamente son solicitadas. Dicho lo anterior, también existen situaciones dentro de un intervalo acotado de tiempo, donde surgen explosivamente peticiones las que se caracterizan por poseer un contenido en común, las cuales nacen por el desarrollo de un evento de interés popular teniendo como resultado un aumento sustancial en la demanda generada a los nodos, teniendo como consecuencia latencia y congestión en las redes.

1.2. Desafíos

Los desafíos que trae consigo llevar a cabo el proyecto de titulo, son los de aprender a investigar, comprender nuevos conceptos, realizar una ingeniera inversa de como funciona el simulador y la de plantear una nueva arquitectura *caché* que ayude a mejorar la eficiencia del *caché* y consigo la red.

1.3. Contribución de la tesis

El aporte entregado por el proyecto de título, se puede identificar inicialmente por la creación y diseño de una nueva arquitectura de memoria *caché* para los nodos de las redes *ICN* (*Information Centric Network*), considerando el comportamiento de los usuarios por medio del tráfico de red. En segundo lugar, se debe incorporar una estrategia de *caché* (políticas de admisión, desalojo y reemplazo) y que será implementada en un simulador denominado *SIM* (*Named Data Networking Simulator*), todo esto con el fin de mejorar resultados de eficiencia de la memoria *caché* (*Hit's, Miss*) en comparación con otras arquitecturas *caché* que se encuentran por defecto dentro del simulador (LFU, FIFO). No obstante, a continuación se detallarán las contribuciones efectuadas por este trabajo:

- Diseño de una nueva arquitectura *caché* bajo el paradigma de las redes ICN, que contenga una subdivisión de tres grupos capaces de retener diferentes tipos de peticiones (intereses) de modo que la eficiencia del nodo se vea afectada positivamente. En cuanto a la división de la memoria *caché*, el primer segmento se encarga del almacenamiento de peticiones del tipo ráfaga, la segunda de guardar las de tipo permanente y para la última sección, la recaudación de peticiones de tipo variable.

- Creación de una topología de redes ICN, utilizando el simulador ndnSIM, la cual contenga dentro de sus nodos la arquitectura *caché* anteriormente señalada con la finalidad de inyectar trafico de datos en base al comportamiento usuario para la obtención de resultados empíricos respecto a la nueva arquitectura *caché* diseñada.

1.4. Estructura de la tesis

A continuación se realizará una breve reseña de cómo se estructura el siguiente trabajo.

El capítulo 2 abarca la descripción del problema, en el se define el objetivo general, específicos, hipótesis y los alcances que contiene el proyecto de título, incluyendo también la descripción de la metodología de trabajo utilizada para el progreso del trabajo.

El capítulo 3 posee el marco teórico, en el cual se definen diferentes conceptos esenciales para el entendimiento del proyecto del título. Dicho lo anterior se encuentran los siguiente conceptos: Redes centradas en la información (ICN), ndnSIM, aplicaciones web de gran escala, el *caché*, estructura y políticas de reemplazo del *caché* existentes, finalizando con el comportamiento del usuario de manera donde se explica la ley de Zipf y se detallan los tipos de peticiones de usuario.

En el capítulo 4 se presenta una revisión bibliográfica de los trabajos realizados respecto a las arquitecturas *caché* y las políticas de *caché* implementadas en redes ICN, además del revisión de trabajos dentro del simulador ndnSIM, todo esto con el fin de diseñar una arquitectura de *caché* que no exista en la literatura.

Capítulo 2

Descripción del problema

2.1. Contextualización

NDN es una arquitectura de red basada en contenido perteneciente al paradigma ICN, que tiene como objetivo desplazar la actual arquitectura del Internet. Logrando así aumentar diversidad de contenido en red, reducir la congestión en red, tiempos de respuesta y carga del servidor.

Para hacer lo anteriormente señalado la arquitectura de red NDN consta de nodos intermediarios que poseen memoria *caché*. Las que guardan entradas que están ligadas a una respuesta pre-computada, que fueron realizadas por las peticiones de los usuarios. Así cuando una nueva petición de un usuario sea enviada, si el nodo contiene en su memoria *caché* la respuesta pre-computada de la petición esta sera devuelta al usuario.

Como la memoria *caché* tiene limite de capacidad esta cuenta con algoritmos denominados políticas, los que permiten mantener actualizada su información y por ende generar mayor cantidad de *hit's*, lo que se traduce en un mejor rendimiento de *caché*. También la memoria *caché* posee múltiples esquemas que permiten un mejor funcionamiento de esta. Por otro lado las peticiones realizadas por los usuarios en el Internet poseen patrones de comportamiento que varían en el tiempo y en su frecuencia. Alguno de estos patrones son: Ráfagas, Permanentes, Periódicas.

El proyecto de título tiene como objetivo proponer una arquitectura *caché* considerando el comportamiento del usuario, para los nodos de la arquitectura de red NDN y implementarlo en el simulador de red NDNSIM. La arquitectura constara de asignar un espacio del *caché* la cual se dividirá en tres secciones de consultas (ráfaga,

permanente y dinámicas).

El resultado de la solución propuesta, sera contrastada con estrategias que vienen por defecto en el simulador NDNSIM y nombradas en el estado del arte del este trabajo. Analizando así la tasa de *hit* de los experimentos realizados y verificando la eficacia de la arquitectura *caché* propuesta.

2.2. El problema

2.2.1. Declaración del problema

La cantidad de información presente en la red del Internet es de proporciones inimaginables. Información que es almacenada en servidores y solicitada por los usuarios en forma de consultas y de manera constante, para luego ser procesada y satisfacer la respuesta del usuario. Para reducir la carga del servidor producida por la búsqueda de la respuesta en *back-end*, se propone la utilización de una red basada en contenidos, específicamente la arquitectura de red NDNSIM, la cual contiene una interconexión de nodos caché.

El *caché* tiene un almacenamiento limitado, es por eso que es importante mantenerlo actualizado respecto a las consultas de los usuarios, para así tener una mayor tasa de *hit's* y reflejada en una menor carga del servidor. Para lograr esto, es que se utilizan esquemas de *caché* y también las llamadas políticas de admisión, reemplazo y eliminación las cuales ayudan a mantener el almacenamiento caché actualizado.

En los trabajos mencionados en la sección 4, específicamente el trabajo [37] en donde exponen una modificación tanto del esquema del *caché* y a las políticas de reemplazo, con el fin de mejorar las métricas propuestas. O bien trabajos como el de [25] en el cual desarrollan nuevos paquetes.

También se tiene que tener en consideración que las consultas de los usuarios arrastran un patrón de comportamiento que varia según el tiempo y su frecuencia. Comportamiento que provoca congestión en red(Cuello de botella), sobre carga de servidores, aumento en tiempo de respuesta, disminución de la eficiencia del *caché*, fallas en nodos, etc.

2.2.2. Diagrama de Ishikawa

En la Fig.2.1 se muestra el diagrama de Ishikawa con el fin de comprender los problemas relacionados con la ineficiencia del *caché*.

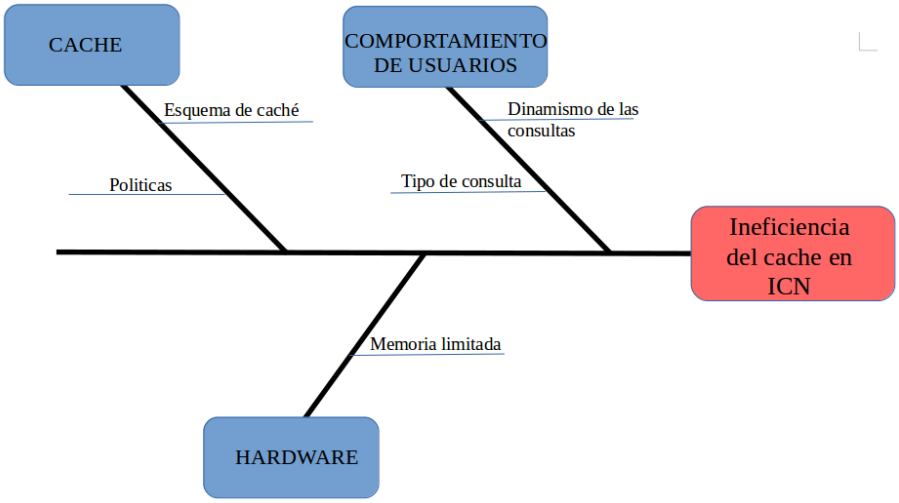

Figura 2.1: Diagrama de Ishikawa. Fuente: Elaboración propia.

2.3. Objetivos

2.3.1. Objetivo general

Optimizar la eficiencia del *caché* en una arquitectura NDN por medio del diseño de una arquitectura *caché* basada en comportamiento de usuario.

2.3.2. Objetivos específicos

Para lograr el objetivo general de la tesis se requiere cumplir con los siguientes objetivos específicos.

- Analizar patrones de comportamiento.
- Proponer una solución *caché* para una red basada en contenido con una arquitectura NDN.
- Realizar una evaluación experimental .

2.3.3. Alcance de los objetivos

El proyecto de título se limita a la propuesta y experimentación de una arquitectura *caché* para la red basada en contenido NDN. Donde se utilizará un paquete ndnSIM de código abierto, que implementa la pila de protocolos NDN para el simulador de red NS-3, para ejecutar simulaciones.

Este trabajo se enfoca principalmente en el *caché* y en utilizar las políticas tanto de reemplazo, admisión y eliminación (LFU, LRU, RR, FIFO). Las cuales serán implementadas en cada sección de *caché* según corresponda al comportamiento de usuario. Definiendo parámetros de entrada y una topología especifica. Con el fin de obtener resultados respecto a la tasa de *hit*. Por eso como trabajo futuro queda analizar como afecta el comportamiento del borde de la red, en la arquitectura *caché* propuesta.

Situación actual	**Objetivo específico**	**Resultado esperado**	**Métrica**	**Criterio de éxito**
Comportamiento dinámico y variable en el tiempo de las consultas de los usuarios.	Analizar patrones de comportamiento.	Conjuntos de trabajos que señalen los distintos tipos de patrones de comportamiento.	Número de patrones definidos (P_n)	$P_n \geq 3$
Variados trabajos que buscan mejorar la eficiencia del *caché* algunos de ellos presentados en el estado del arte.	Proponer una solución *caché* para una red basada en contenido con una arquitectura NDN.	Esquema o política *caché*.	Numero patrones usados en la solución (SP_n)	$SP_n \geq 2$
Esquemas y políticas *caché* (Admisión y reemplazo) ya presentes en el simulador (LFU,,FIFO, RR).	Realizar una evaluación experimental de la solución *caché* propuesta.	Gráficos con resultados obtenidos.	Tasa de *hit's*(T_h) Numero contexto(N_c)	$T_h \geq 5\,\%$; $N_c \geq 2$.

Cuadro 2.1: Alineamiento de objetivos

Capítulo 3

Marco teórico

3.1. Information Centric Networking

ICN (*Information-Centric Networking*) es un paradigma que pretende motivar la transición arquitectónica de la actual arquitectura centrada en el *host* a centrada en la información, para difundir de manera eficiente y flexible la enorme información generada por una variedad de aplicaciones. En esta dirección de investigación, se han propuesto muchos enfoques durante los últimos años, de los cuales para el desarrollo de este trabajo se centrara en NDN.(Figura 3.1)

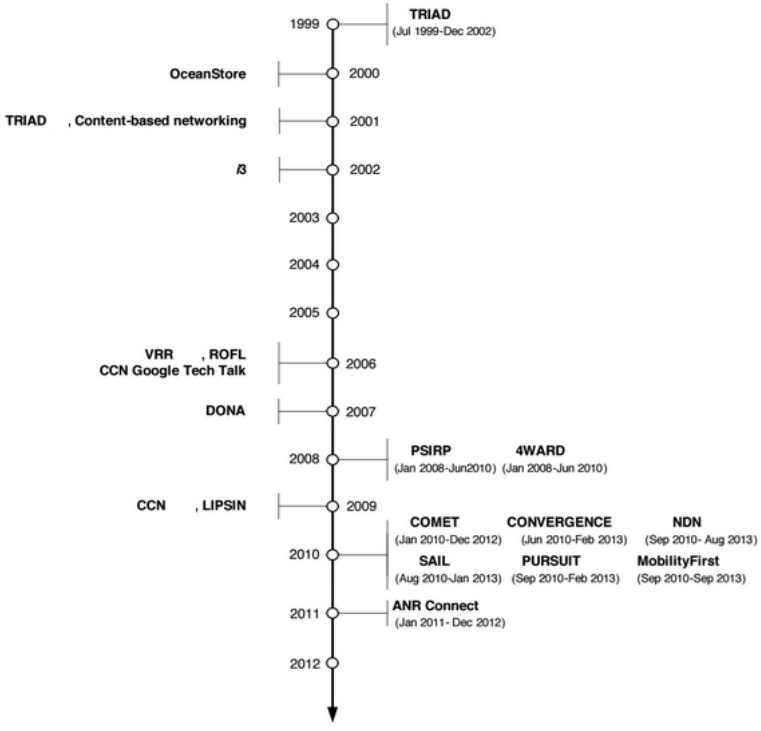

Figura 3.1: Cronología de los principales hitos de ICN [42].

Named Data Networking (NDN) es una de las arquitecturas futuras para el Internet, con años de investigación sobre el uso de la red y de los problemas no resueltos en arquitecturas contemporáneas del Internet como IP [2][1]. La arquitectura NDN tiene su origen en el proyecto *Content-Centric Networking* (CCN) que Van Jacobson divulgo por primera vez en el año 2006 en una charla tecnológica de Google [17].

El proyecto NDN financiado por EE.UU desarrolla aun más la arquitectura CCN anteriormente mencionada, reconfigurando la pila de protocolos del Internet intercambiando la cintura delgada de la arquitectura de Internet por el de datos nombrados, empleando diversas tecnologías de red por debajo de la cintura para la conectividad, incluyendo IP. También, NDN con el fin de optimizar el uso de recursos, contiene una capa de estrategia media que se ubica entre la capa de datos nombrada y las tecnologías de red, mientras que una capa de seguridad lleva a cabo las funcionalidades de seguridad directamente a los datos nombrados [42](Fig. 3.2).

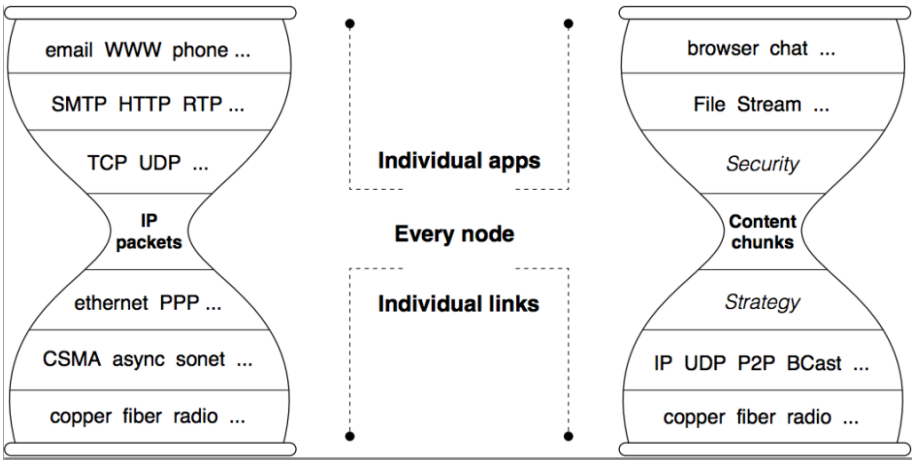

Figura 3.2: Arquitecturas de reloj de arena de Internet y NDN [3].

Dicho lo anterior, la principal abstracción de ICN es el NDO (*Named Data Objects*), la que se define como la unidad de datos direccionales que se encuentra dentro de una red centrada en la información, siendo capaces de representar una colección de bytes o una información en específico. Así mismo cada uno de los objetos de datos existentes posee un nombre vinculado a el, logrando verse como fragmentos de datos nombrados sin semántica alguna y caracterizándose por su granularidad la cual varía desde el tamaño del paquete hasta los objetos completos [20][6].

Los objetos de datos con nombre se desligan de la ubicación, la aplicación, el almacenamiento y los medios de transporte, dando paso al almacenamiento en *caché* y un bajo costo y ubicua replicación en la red, esperando como resultado una mayor eficiencia y seguridad, una mayor escalabilidad respecto a la demanda de información/ancho de banda y una superior robustez en escenarios de comunicación desafiantes [20].

La arquitectura NDN se compone de dos tipos de NDO, los paquetes de interés (*Interest Packet*) y los paquetes de datos (*Data Packet*), presentes(Fig. 3.3), siendo el nombre de estos de tipo jerárquico, similares a una URL, como por ejemplo /aueb.gr/ai/main.html. Los cuales son enviados por dos tipos de usuarios: el primero denominado consumidor, entidad encargada de enviar una solicitud de un NDO a la red (*Interest*), y por otra parte un productor, que es la entidad encargada publicar NDO's en la red, los cuales serán solicitados por un consumidor y enviados (*Data*).

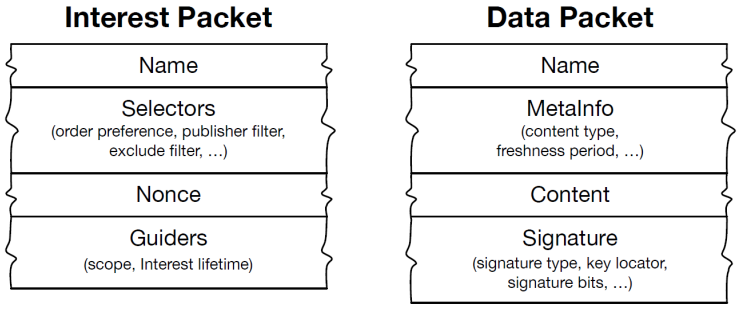

Figura 3.3: Paquetes NDN.

Adicionalmente esta arquitectura también cuenta con nodos intermediarios entre los consumidores y los productores denominados CR (*Content Routers*), los cuales le entregan a la red un aumento en la replicación de información y consigo una mayor resolución de consultas. Los nodos contienen tres estructuras de datos principales (Fig. 3.4):

- ***Forwarding Information Base* (FIB):** El FIB se utiliza para reenviar paquetes de interés hacia fuentes potenciales de datos coincidentes. Es casi idéntico a un IP FIB, excepto que permite una lista de interfaces salientes en lugar de una sola. Esto refleja el hecho de que CCN no está restringido a reenvío en un árbol de expansión. Permite múltiples fuentes de datos y puede consultarlas todas en paralelo [18].

- **Almacén de contenido (*Content Store*, CS):** El almacenamiento en *caché* de los NDO es una parte integral del servicio ICN. Todos los nodos potencialmente tienen *cachés*, incluyendo nodos en redes de infraestructura de ejecución de operadores y redes domésticas dirigidas por usuarios, así como terminales móviles. Las solicitudes de NDO pueden ser satisfechas por cualquier nodo que contenga una copia en su *caché*. ICN combina así el almacenamiento en *caché* en el borde de la red, como en P2P y otras redes de superposición, con almacenamiento en *caché* en la red (por ejemplo, *cachés* de web transparentes). El almacenamiento en *caché* es genérico, es decir, es independiente de la aplicación y se aplica a todos los proveedores de contenido, incluido el contenido generado por el usuario [18].

- ***Pending Interest Table* (PIT):** El PIT realiza un seguimiento de los Intereses enviados hacia arriba hacia la(s) fuente(es) de contenido de forma que los datos devueltos puedan ser enviados aguas abajo a su(s) solicitante(s) [18].

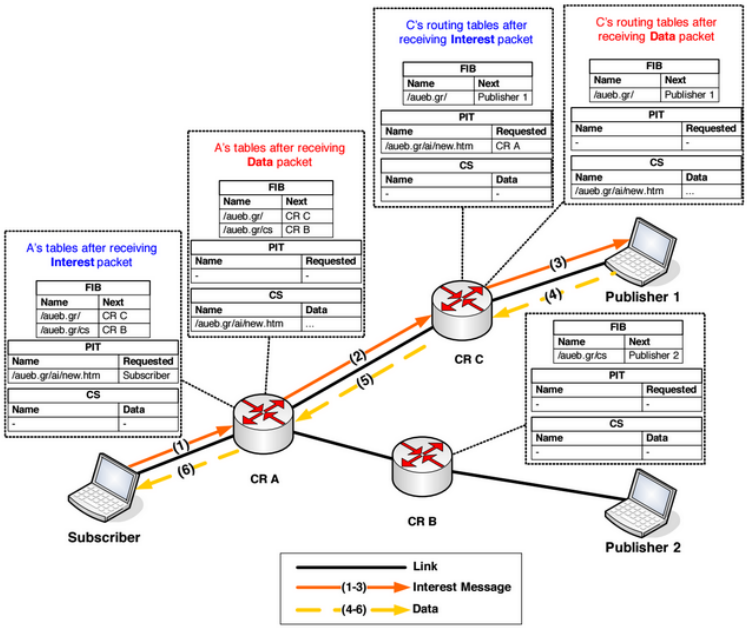

Figura 3.4: Arquitectura NDN, CR significa Content Router, FIB para Forwarding Information Base, PIT para Pending Interest Table y CS para Content Store. Fuente: A Survey of Information-Centric Networking Research [42].

3.2. Caché

La cantidad de usuarios van en aumento día a día, por lo que las aplicaciones web con un gran número de usuarios tienen que ser capaces de soportar grandes volúmenes de consultas y transacciones de datos (millones), las cuales poseen una patrón de comportamiento definido, este tema se abordará en la sección 3.3. Una de las tecnologías que existe para apaciguar el acceso incesante a información de los servidores, es la de un lugar de almacenamiento destinado a archivar las consultas y su respuesta pre-computada, denominado *caché*.

El *caché* es una de las partes más importante dentro de los servicios web y motores de búsqueda [7],[5]. Principalmente el *caché* es un espacio de almacenamiento (memoria) de acceso rápido, en donde se retienen las consultas más reiteradas hechas por los usuarios y sus respuestas, entregando una gran cantidad de ventajas entre las que se encuentran:

- Reducir la latencia de acceso a los documentos, esto se debe a que los objetos que son accedidos con mayor frecuencia se encuentran almacenados en un nodo de *caché* cercano, lo que disminuye el tráfico en la red, permitiendo que los objetos que no están en *caché* puedan también ser recuperados con mayor rapidez [38].
- Reducir el consumo de ancho de banda, lo que permite disminuir la congestión y el tráfico en la red [38].
- Reducir la carga de trabajo en los servidores web [38].

Siendo las características deseables para una arquitectura *caché* las siguientes: robustez (capaz de no generar fallos o bloquearse), acceso rápido (disminuir latencia), eficiente, escalable, estable, adaptable y simple de implementar [38]. Por ello una de las finalidades más importantes en el *caché* es la de obtener el mayor número respecto a la tasa de *hit*, en otras palabras, conseguir que una gran parte de las consultas realizadas por el usuario, que ingresan al *caché* cuente con una respuesta pre-computada.

En la arquitectura NDN, el *caché* de los CR juega un rol importante al momento de satisfacer las peticiones del usuario (consumidor). A modo de que sin la presencia del *caché*, el paradigma de las redes ICN no sería válida, causando así que los productores (servidores), procesen un volumen elevado de consultas simultáneas. La importancia del *caché* en la arquitectura NDN de las redes ICN radica en que esta

requiere de una difuminación eficiente de la gran cantidad de consultas realizadas diariamente por lo usuarios, de manera que sean respondidas de forma rápida.

El *caché* es limitado, por lo que al completar su capacidad máxima es necesaria la implementación de algoritmos que permitan el control del almacenamiento. Estos algoritmos son las denominadas políticas, de las cuales existen tanto de admisión, reemplazo y desalojo. Dada la gran cantidad de consultas que generan los usuarios en el Internet es que se hace necesaria la creación de nuevas políticas que permitan una mejor eficiencia del *caché*.

3.2.1. Organización del Cache

En la arquitectura NDN el *caché* (*Content Store*) se encuentra presente en los distintos nodos (CR) incluyendo también a los consumidores y productores. Donde la entrada que posee un CS es una tupla <Name, Data> siendo *Name* el NDO *Interest* y *Data* el NDO que responde dicho *Interest*. Como se señaló en el punto anterior, el almacenamiento en *caché* es limitado, por lo que la entrada de tuplas deben ser constantemente variadas, dando la posibilidad del ingreso de nuevas entradas, las cuales tienen mayor probabilidad de aumentar la tasa de *hit* en *caché*. A continuación se definirán las políticas ya señaladas.

3.2.1.1. Política Admisión

Esta política es la encargada del administrar la entrada a la memoria *caché*, a través del bloqueo de entradas que no sean constantemente requeridas en el futuro, a raíz de que lo anterior producirá un aumento en la tasa de miss del *caché* [8], disminuyendo la eficiencia del nodo CR.

Esto ocurre cuando la memoria en *caché* se encuentra en su máxima capacidad, por lo que no es posible almacenar una nueva entrada, motivo por el cual se pone en marcha la política de desalojo respectiva para descartar una entrada del *caché* y ingresar la nueva entrada. Pero debido a que la nueva entrada puede generar escasos *hit's*, entonces se deben de llevar a cabo las políticas de admisión las cuales tomarán la decisión de si la nueva entrada es guardada en la memoria *caché* o bien se mantendrá la entrada propuesta para el desalojo de *caché*.

3.2.1.2. Política Desalojo o Reemplazo

Las política de desalojo o reemplazo se llevan a cabo cuando la memoria *caché* se encuentra llena, por lo que es necesario liberar espacio para dar oportunidad a otras entradas entrantes, principalmente la idea es descartar una entrada la cual posee una baja probabilidad de lograr ser un *hit* en *caché* [8].

Los objetos web, los NDO en el caso de la arquitectura NDN contienen diferentes variables que son tomadas en cuenta para decidir si salen o se quedan en *caché*, como lo son el tiempo de la última referencia, el tamaño, tiempo de expiración y el tiempo desde su última modificación.

Algunas de políticas de desalojo conocidas dentro de la literatura, son *Least Frequently Used* (LFU) y *Least Recentently Used* (LRU) [16][15][11]. Dentro del simulador ndnSIM se encontraran las ya anteriormente señalados algoritmos de reemplazo, además del algoritmo FIFO, y un cantidad no menor de variaciones de los algoritmos señalados, con su respectiva variable que permitirá decidir que entrada seleccionar.

3.2.1.3. Eliminación de entradas antiguas

Principalmente la política de eliminación de entradas antiguas son las encargadas de mantener el almacenamiento en memoria *caché* actualizado.

3.2.2. Estructura del caché

Anteriormente se vio que es posible caracterizar el *caché* según las políticas que utiliza este mismo, pero también es factible agrupar el *caché* en dos categorías: Estático y dinámico.

3.2.2.1. Caché Estático

Tal como dice su nombre, son *cachés* que no cuentan con variabilidad de contenido durante un intervalo de tiempo amplio, estos tipos de *caché* se basan generalmente en información acumulada históricamente y es actualizada a largo plazo.

Dentro del contexto de los motores de búsqueda, el *caché* estático aglomera en memoria las entradas más habituales, por medio del análisis de registros históricos de las entradas efectuada por los usuarios. Siendo algunas de las características a tomar

en cuenta para ser incorporada en el *caché*: la cantidad de usuarios que realizaron la misma consulta, la frecuencia de la consulta, proporción de clics hechos por los usuarios a las URL que entregaron como respuesta a la consulta, además de la estabilidad temporal de la frecuencia de una consulta.

Realizada una vez la captación de las entradas más habituales, serán guardadas en *caché* y actualizada oportunamente.

3.2.2.2. Caché Dinámico

Son *cachés* en donde el contenido varía constantemente, por lo tanto se trata de incorporar dentro del *caché* las entradas que tienen alta probabilidad de ser solicitadas en un futuro. Por lo tanto al ser una *caché* que varía en el tiempo, el contenido almacenado debe ser frecuentemente actualizado, por medio de políticas de desalojo o reemplazo. Inicialmente este tipo de *caché* esta vacía, para luego ir llenándose periódicamente a medida que se ingresan entradas en *caché* (tuplas) hasta llenarse y luego implementar las políticas de reemplazo para mantenerse actualizado.

3.2.3. Arquitectura de caché

3.2.3.1. Arquitectura de caché jerárquico

Son *cachés* que se asimilan a la estructura de un árbol, sus nodos *caché* se clasifican en padre e hijo. El *caché* padre no se comunica con sus hijo, pero el hijo si puede solicitar al padre. En esta arquitectura *caché* se sitúan nodos *caché* en múltiples niveles de red [31]. Esta arquitectura consta de cuatro niveles: inferior, institucional, regional y nacional. Los cuales van siendo recorridos en el orden respectivo a medida que la peticiones no son solucionadas por estas mismas. Si la petición no es solucionada por ningún nivel de *caché* entonces se redirecciona al servidor original para encontrar el documento solicitado, para luego insertar esta el documento solicitado en el nivel más bajo de la jerarquía y dejando copias en los nodos intermedios. De modo que si continúan las solicitudes de ese documento, esta seguirá escalando a lo alto de la jerarquía. En la Fig. 3.5 se observa la topología del *caché* jerárquico.

3.2.3.2. Arquitectura de caché distribuido

A diferencia del *caché* anteriormente señalado, el *caché* distribuido, solo cuenta con un nivel de jerarquía, el *caché* institucional. En donde sus nodos se encuentran conectados entre si para así ir a buscar el documentos a otros nodos que lo puedan

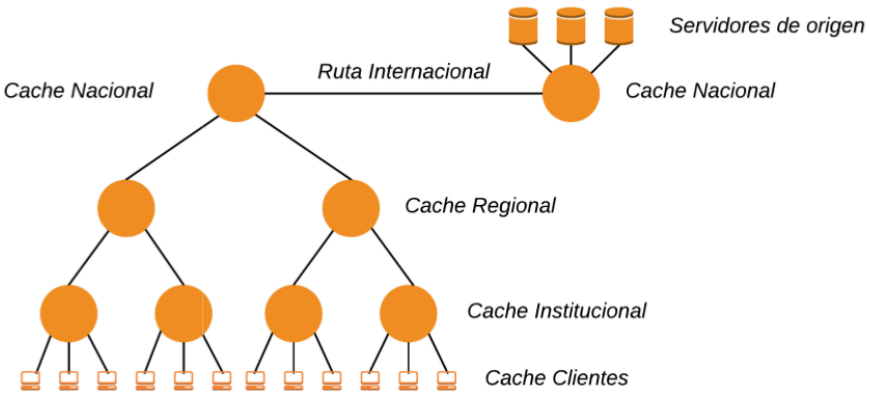

Figura 3.5: Topología de *caché* jerárquico. Fuente: Arquitectura para la distribución de documentos en un sistema distribuido a gran escala [35].

contener, en caso de generarse un *miss* en la búsqueda de alguno de estos(Ver Fig.). A continuación se describirán algunas ventajas del *caché* distribuido en contraste con el *caché* jerárquico según [26]: menor uso de espacio en disco, mayor balance de carga y tolerancia a fallos.

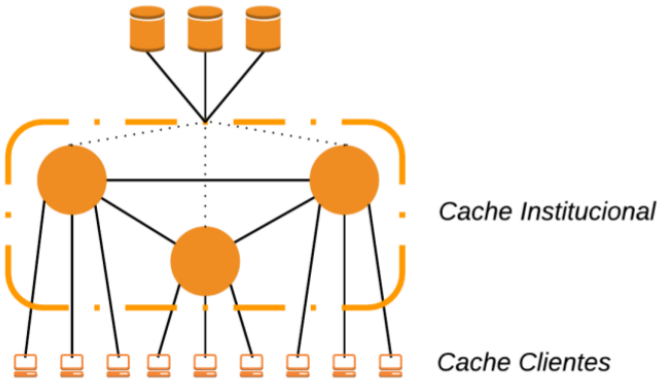

Figura 3.6: Topología de *caché* distribuido. Fuente: Arquitectura para la distribución de documentos en un sistema distribuido a gran escala [35].

3.2.3.3. Arquitectura de caché híbrida

Los *caché* híbridos son una combinación del *caché* jerárquico y el *caché* distribuido. Los nodos *caché* pueden cooperar con otros *cachés* que se encuentren en el mismo nivel de jerarquía o en un nivel superior a través del *caché* distribuido. En la Fig. 3.7 se observa la topología que puede seguir una arquitectura de *caché* híbrido.

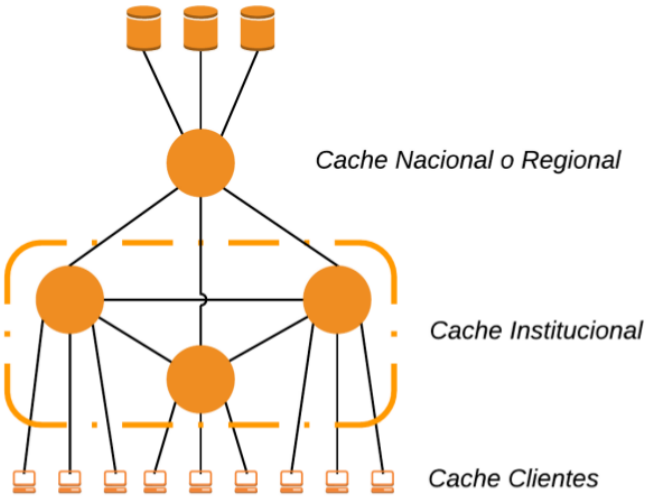

Figura 3.7: Topología de *caché* híbrida. Fuente: Arquitectura para la distribución de documentos en un sistema distribuido a gran escala [35].

3.3. Comportamiento de usuario

El comportamiento de usuario se puede definir como, un área que intenta encontrar características, patrones o conductas de los usuarios en algún contexto específico, como por ejemplo en las redes sociales, en algunos sistemas de compras, y otras. Esto es de importancia para las empresas en general, ya que pueden mejorar sus ofertas de productos o hacer sistemas más eficientes que consideren alguna conducta repetitiva de los usuarios.

3.3.1. Ley de Zipf

La llamada ley de Zipf, formulada en la década de 1940 por George Kingsley Zipf [45][46] es una ley empírica la cual establece que una cantidad menor de palabras son altamente utilizadas, mientras que existe un conjunto mayoritario de palabras que son raramente usadas. El comportamiento o bien patrón que tienen estas palabras siguen la fórmula:

$$\frac{1}{n} \tag{3.1}$$

Donde n corresponde a la n-ésima palabra ordenada por frecuencia decreciente. El comportamiento anteriormente señalado no solo ha sido estudiado para las palabras,

también ha sido analizado en el contexto de los motores de búsqueda, específicamente al comportamiento de las consultas, obteniendo como resultado que estas también siguen una distribución "zipfianas"[9][11][15][21][39][33][41].

3.3.2. Peticiones de usuarios

Los usuarios del Internet de ahora ya no son los mismos usuarios del Internet de años atrás, la información que necesitaban los primeros usuarios, ya no es la misma información que necesitan los usuarios de ahora y al igual que la moda, esta va variando en el tiempo. De la misma manera, las consultas generadas por los usuarios va cambiando en el tiempo, en función de distintas circunstancias que afectan el interés de los usuarios a la hora de realizar consultas, como lo son la economía, el deporte, desastres naturales, fechas importantes, temporada, eventos sociales y/o culturales, etc.

Teniendo en cuenta lo anterior y la frecuencia con la que se lleva a cabo cierto tipo de consulta, es que sera logrado encontrar patrones de conducta los cuales permiten clasificar los distintos tipos de consultas. Entre las clasificaciones de consultas más conocidas se encuentran las consultas periódicas, las consultas en ráfaga y las consultas permanentes, las que serán detalladas a continuación.

3.3.2.1. Consultas Periódicas

Corresponden a consultas que poseen un aumento en su frecuencia en un intervalo definido de tiempo, para luego disminuir hasta llegar a su frecuencia constante, todo esto de manera periódica (meses, semanas, días, horas, años, etc.). Dicha información permite anticipar la sobrecarga de los servidores, sin tener la necesidad de utilizar un algoritmo que permita la admisión en *caché*. A continuación en la figura se muestra un ejemplo de una consulta de tipo periódica, correspondiente a la consulta "18 de septiembre"

3.3.2.2. Consultas Ráfagas

Este tipo de consultas se caracteriza principalmente por poseer una frecuencia constante en el tiempo, la cual se ve afectada por un evento en específico, ya sea de índole social, económico, cultural, etc, llamando así el interés de los usuarios en general. Esto provoca que la frecuencia respecto a un tema en específico aumente

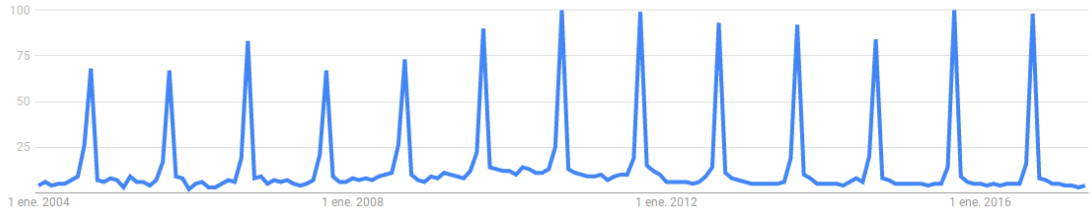

Figura 3.8: Frecuencia de consulta periódica: “18 de Septiembre” (2004-2016). Fuente: Google Trends.

bruscamente hasta alcanzar un *peack*, para luego ir reduciéndose a medida que pasa el tiempo, como consecuencia del desinterés que comienzan a tener los usuarios respecto al tema mencionado. Un ejemplo de este tipo de consultas corresponde a la “muerte de Fidel Castro”tal como se muestra en la Figura 3.9.

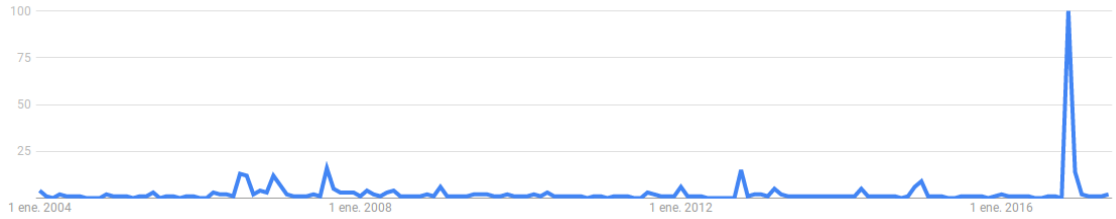

Figura 3.9: Frecuencia de consulta ráfaga: ”Muerte de Fidel Castro”(2004-2017). Fuente: Google Trends.

Además, las consultas en ráfaga no solo afectan a una consulta en específico, si no que también a todo conjunto de consultas que tengan algún tipo de relación con el evento en particular que llevó al alza en este tópico. En la Figura 3.10 se observa como también la consulta “Historia de Fidel Castro”posee un patrón ráfaga de la misma forma que la consulta “muerte de Fidel Castroz durante los mismos intervalos de tiempo.

Uno de los problemas más serios que producen estas consultas debido al aumento en el volumen de consultas solicitadas, es la generación de desbalance en los nodos con respuestas pre-computadas en *caché*, como también una congestión al acceso de la información en los servidores.

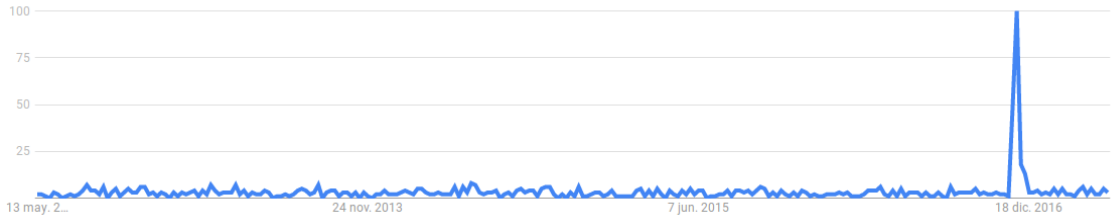

Figura 3.10: Frecuencia de consulta rafaga: “Historia de Fidel Castro”(2012-2017). Fuente: Google Trends.

3.3.2.3. Consultas Permanentes

Finalmente las consultas permanentes, son definidas como aquellas que poseen una continuidad en la frecuencia que son solicitadas en intervalos largo de tiempo. En simples palabras, son consultas en que el paso del tiempo no afecta abruptamente su frecuencia. En la Figura 3.11 se adjunta una consulta del tipo permanente la cual corresponde a "YouTube".

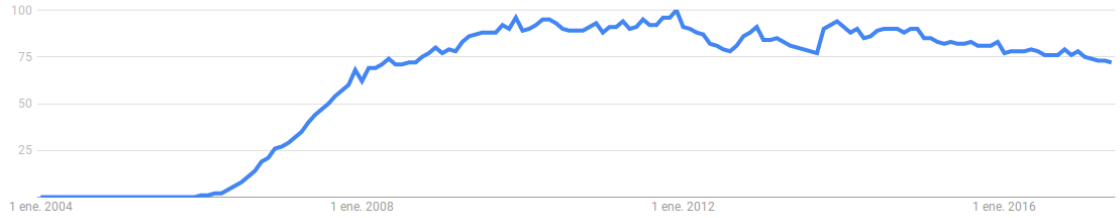

Figura 3.11: Frecuencia de consulta permanente: "YouTube"(2004-2017). Fuente: Google Trends.

Bajo este contexto, muchos son los trabajos encontrados en la literatura [44], [22], [27], [19], los cuales tienen como conclusión que destinar un espacio estático en *caché* para guardar las consultas permanentes, lo que permite aumentar el rendimiento del *caché*, mejorando la tasa *hit* de este mismo. Cabe recordar que las consultas permanente se obtienen de registros históricos, los cuales son analizados para determinar cuales son consultas permanente y cuales no, tomando en cuenta su frecuencia.

Capítulo 4

Estado del arte

4.1. NDNSIM

4.1.1. On Performance of Cache Policies in Named Data Networking [30]

En [30] se propone el diseño de una política de reemplazo de *caché* basada en la popularidad del contenido (CCP), la cual tiene en cuenta el factor de popularidad y una nueva estructura de datos denominada "Tabla de popularidad de contenido (CPT) sobre la base de retención del *Content Store* (CS), *Pending Interest Table* (PIT) y la *Forwarding Interest Base* (FIB). Con el fin de beneficiar las métricas de golpe de *caché*, carga del servidor y el promedio de rendimiento de la red en comparación con las políticas de *caché* LFU y LRU ya presentes en el simulador.

Para que esto sea posible, según [30] las estrategias *caché* deben poseer dinamismo en cuanto al contenido almacenado en *caché*, siguiendo los dos siguientes puntos:

- Ser capaz de adaptarse a los cambios rápidos en el modo de transmisión dinámica de red.
- No puede ser afectado por el contenido de la alternativa temporal.

De modo que las demandas anteriormente señaladas son saciadas por la política de reemplazo de *caché* basada de popularidad (CPP) propuesta.

4.1.1.1. Calculo de la popularidad de un contenido

CCP añade una nueva estructura de datos CPT, la cual contiene toda la información sobre la popularidad del contenido almacenada en CS, incluyendo el nombre del contenido, el *hit* en *caché*, la popularidad anterior y actual. Dicha estructura de datos sa va actualizando cada vez que termina un ciclo de conteo, obteniendo así el *ranking* actual de popularidad del contenido por medio del *hit* de *caché* y la popularidad anterior. A continuación se describe la formula de calculo para la popularidad del contenido:

$$P[i+1] = \frac{N[i] * \alpha + P[i]}{\alpha + 1} \tag{4.1}$$

$$\alpha = 1 + c * T \tag{4.2}$$

En donde P[i] corresponde a la popularidad del contenido en *caché* y N[i] los *hit's* del contenido en *caché* en el momento del conteo actual. En cuanto a α ($\alpha > 1$) es el coeficiente de peso de la popularidad del contenido, T el ciclo de conteo y C el coeficiente proporcional de α y T. Señalado lo anterior, para este trabajo desarrollado se considero que c = 0,5 y T = 8.

Pero para calcular la popularidad de un contenido, no solo se toma en cuenta el peso entre la popularidad anterior y actual, si no que también se considera la influencia de la ultima red *hot pot*. Logrando así por medio de la expansión de 4.1 la obtención de 4.3.

$$P[i+1] = \frac{N[i] * \alpha + P[i]}{\alpha + 1} = \frac{N[i] * \alpha}{\alpha + 1} + \frac{N[i-1] * \alpha}{(\alpha + 1)^2} + \frac{N[i-2] * \alpha}{(\alpha + 1)^3} + ... \tag{4.3}$$

Llegando así a la conclusión de que con el tiempo, el numero de *hit's* anterior posee una influencia menor en la popularidad de l contenido, lo que refleja de mejor manera la popularidad del contenido en la red actual. Obteniendo resultados como los del siguiente ejemplo, pudiéndose ver que el segundo contenido tiene una popularidad anterior mucho mayor que la primera pero con un menor *caché* actual.

Content name	Previous popularity	Current cache hit	Current popularity
/campus/images/v1.mpg	12.08	20	18.68
/campus/images/v2.mpg	21.28	15	16.05
/campus/music/m1.mp3	22.14	30	28.69

Figura 4.1: Tabla de popularidad de contenido. Fuente: On Performance of Cache Policies in Named Data Networking [30]

4.1.1.2. Estrategia de reemplazo

La estrategia de reemplazo del presente trabajo consta de calcular la diferencia entre el valor de umbral del tamaño de *caché* y el numero de contenido en *caché*, obteniendo así el tamaño de espacio de *caché* restante. Por lo tanto si un paquete de datos es mayor que el resto del espacio, el reemplazo ocurrirá, siendo el contenido con la menor popularidad el seleccionado para su reemplazo y eliminando también su registro en CPT. Para que luego el nuevo contenido llegado, se almacene en *caché* se registre el CPT al mismo tiempo.

4.1.1.3. Simulación y evaluación

Para lograr cuantificar la efectividad de CCP se utilizo el paquete ndnSIM de código abierto, que implementa la pila de protocolos NDN para el simulador de red NS-3 para ejecutar simulaciones. Se implementara CCP dentro de ndnSIM para evaluar la efectividad de CCP respecto a otras estrategias presentes en el simulador (LFU, LRU). Las métricas utilizadas para cuantificar el rendimiento son la proporción de *caché hit*, el promedio de rendimiento de la red y carga del servidor. Se asume que los usuarios expresan intereses según una tarifa de promedio constante con un intervalo de tiempo aleatorio entre dos intereses consecutivos según una distribución uniforme.

Para la simulación se utilizaron 20 *routers* de contenido, los usuarios se distribuyen en el borde de la red y solo existe un servidor para satisfacer los intereses solicitados. En la Fig 4.2 se puede observar parte de la topología utilizada. Se utilizaron 200 objetos de contenidos diferentes en la red. La capacidad del *caché* equivale a un porcentaje que oscila entre el 20 % y 60 % de la cantidad total de objetos de contenido en la red y el tiempo definido para la simulación es de 300s. Además de asumir un ancho

de banda de *backhaul* de 100Mbps y un tamaño de *caché* que varia entre 20 y 400Kbits.

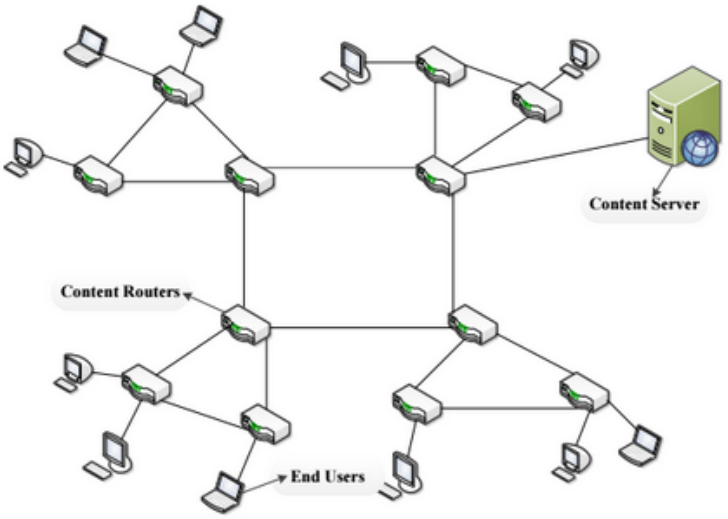

Figura 4.2: Sección de la topología de la red. Fuente: On Performance of Cache Policies in Named Data Networking [30]

4.1.1.4. Resultados

A continuación se presentaran los resultados obtenidos respecto a las pruebas efectuadas a CCP en el simulador ndnSIM.

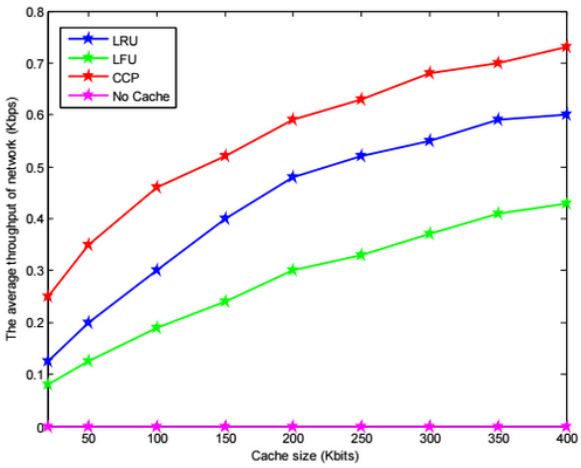

Figura 4.3: Proporción de *hit* en *caché* VS Tamaño *caché*. Fuente: On Performance of Cache Policies in Named Data Networking [30]

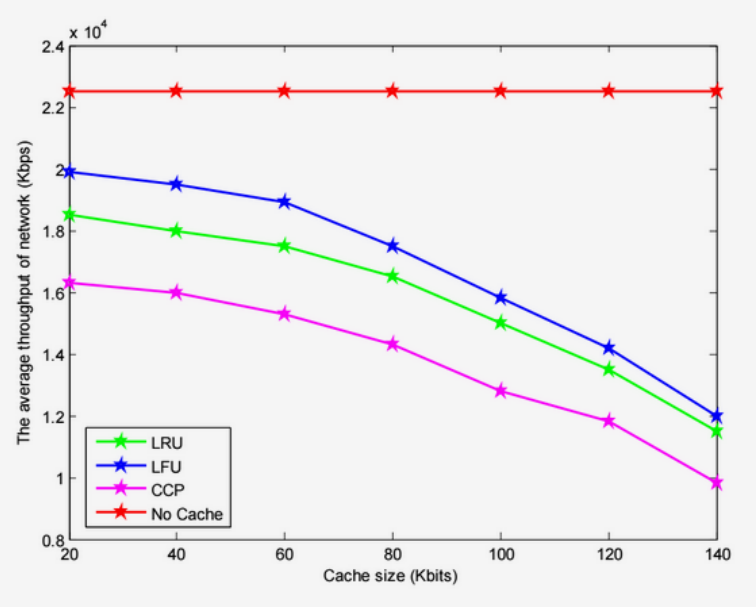

Figura 4.4: Rendimiento medio de la red VS Tamaño *caché*. Fuente: On Performance of Cache Policies in Named Data Networking [30]

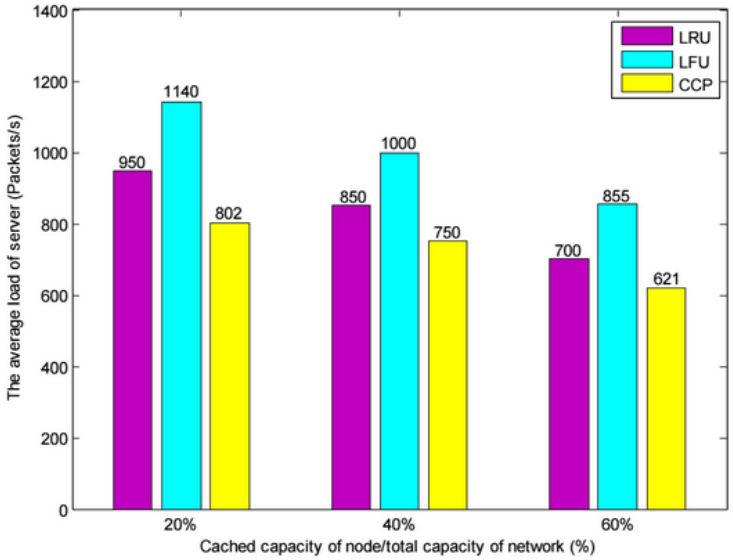

Figura 4.5: Carga media del servidor VS Capacidad *caché*. Fuente: On Performance of Cache Policies in Named Data Networking [30]

4.1.2. Performance of probabilistic caching and cache replacement policies for content-centric networks [37].

En este trabajo se habla principalmente entre la diferencia de lo que es un esquema de *caché* y una política de reemplazo, entregando detalles de cada uno de estos. El trabajo [37] tiene como objetivo el estudio del esquema de *caché* probabilístico por medio de la simulación por computadora (ndnSIM) respecto a la utilización de diversas políticas de reemplazo de *caché* (LRU, LFU y *Randomly Replace*), esto debido a la simpleza del esquema de *caché* probabilístico en comparación con otros esquemas *caché* los cuales agregan una complejidad significativa y gastos generales en cuanto respecto a los sistema de almacenamiento en *caché*. Basándose [37] en que el comportamiento del esquema *caché* probabilístico nunca han sido estudiada, y solo teniendo en cuenta el uso de la política de reemplazo LRU y de la cual no se definió un criterio que entregara un valor decente de la probabilidad de *caché* así como su limitación practica.

Con respecto a lo definido en el trabajo [37] se habla de dos tipos de esquemas de *caché*. El primero denominado esquema de *caché* universal(*Always*) donde $p = 1$ y se caracteriza por el uso de una política de decisión de almacenamiento en *caché* en cada uno de los *router*(CR).Lo que si bien puede distribuir rápidamente el contenido en la red, se tiene evidencia de que este tipo de esquema puede poner replicas de los mismos objetos de contenido en múltiples *routers* (CR),lo que degrada el rendimiento del *caché* en la red provocando bajas tasas de *hit's* en *caché* en los *routers* intermedios.

El segundo esquema denominado esquema de *caché* probabilístico *prob*(p) nombrado en [29][32] y [13]. Consiste en en que cada enrutador CCN almacena en *caché* aleatoriamente un objeto de contenido que lo atraviese, basándose en una probabilidad de *caché* definida como p, donde $0 < p < 1$. Este tipo de esquema solo ha sido probado solo con una política de reemplazo de *caché* LRU con $p = 0,1$ siendo el valor más bajo de p que se ha usado[4]. Por lo que en este trabajo se propone reducir aun más el valor de p para:

1)Disminuir la probabilidad de *caché* p en Prob (p) reduce la probabilidad de que múltiples *routers* CCN en una ruta de entrega almacenen en *caché* el mismo objeto de contenido en una entrega de contenido. 2) Disminuir la probabilidad de *caché* p de Prob (p) resulta en una duración más larga del estado inicial de los sistemas de almacenamiento en *caché*.

La puesta en marcha del trabajo descrito consta de la evaluación de los dos esquemas de *caché*(universal, Prob(p)) cuando son puestos a trabajar con las políticas de reemplazo LRU, LFU y RR. Se utilizo ndnSIM el cual es un simulador basado en NS-3 modelando así el enrutamiento del intercambio de paquetes de interés y paquetes de datos los cuales poseen un tamaño fijo igual para todos.
Para la probabilidad de almacenamiento en *caché* p se asignaron los siguiente valores (1, 0,7, 0,3 y 0,01). Ademas el CS de cada nodo comienza vació (arranque frio) y su tamaño varia entre el 1 % al 10 % de la población de contenido y se uso un algoritmo de Dijkstra para calcular el camino más corto (numero de saltos) para llegar al proveedor de contenido.

En cuanto a los solicitantes y proveedores de contenidos, el primero sigue una distribución Poisson con una media de 50 peticiones y el segunda consta de 1.000 objetos de contenidos diferentes. La simulación tuvo un tiempo de ejecución de 10.000 segundos y un calentamiento de 4.000 segundos.

La topología utilizada para representar la simulación fueron dos:

- **Red en cascada**
  Red de longitud fija con cinco enrutadores Fig. 4.6, de la cual se consideraron dos casos de estudios, con un solicitantes de contenidos y con múltiples solicitantes de contenido.

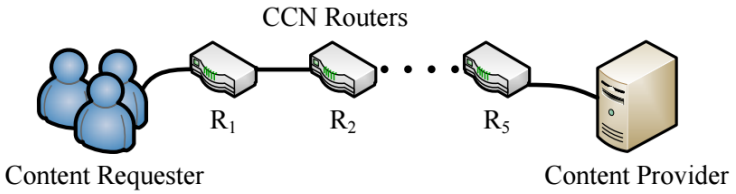

Figura 4.6: Red cascada usada en la simulación. Fuente: Performance of probabilistic caching and cache replacement policies for content-centric networks [37]

- **SINET4**
  Topología híbrida de una red de malla y un numero de redes con topología estrella. Esta topología consta de ocho nodos centrales y 42 nodos de borde.

Las métricas a considerar para los resultados son los siguientes, carga del servidor, *round-trip hop*, *caché hit rate*, comportamiento instantáneo.

4.1.2.1. Resultados

Redes en Cascada Los resultados entregan que la carga del servidor disminuye como una función del tamaño CS para todos los casos. En el caso de Prob(p) + LRU reduce la carga de servidor a medida que p disminuye p, como lo fue con Prob(0,01) + LRU que logro un 20 % de mejora respecto a Always + LRU. Caso contrario es Prob(p) + LFU y Always + LFU, donde la carga del servidor aumenta a medida que se disminuye p, debido ha que los *routers* con LFU pueden acumular objetos obsoletos. Por otro lado Prob(p) + RR no produce ningún cambio significativo en los resultados en términos de la reducción de carga del servidor, respecto a los valores entregado por p, siendo los resultados para Always + RR y Prob(p) + RR son casi idénticos.

En cuanto a los resultados obtenidos a la métrica de distancia de salto de ida y vuelta, entregando que Always + LRU es inferior al de Always + RR en el primer caso de estudios. Ya en el segundo Always + RR es idéntico. Por ultimo se analizo la tasa de *hit* en *caché*, de los cual se concluye que Always + LFU entrega las mejores tasas de éxito para todos los nodos. Seguido de Prob(0,01) + LRU y Always + LRU respectivamente.

SINET4 Para la carga del servidor, los resultados entregados son similares a los entregados en la anterior topología. Ahora, en lo que respecta a la distancia de salto de ida y vuelta para diferentes esquemas de *caché* y política, se observar en los gráficos entregados que según el porcentaje de tamaño del *caché* Always y Prob(0.03, 0.07, 0.01)+ RR tiene el mejor resultado entregado, seguido de Always + LRU.

En cuanto al porcentaje de *hit* de los nodos, para el sector del núcleo de la topología, se obtuvo que Prob(0.01) + LRU tiene el mayor porcentaje de *hit*, cercano al 40 %. Por otro lado el sector del borde de la topología entrega que Prob(0.01) + LRU entrega el mayor resultado con un 65 %, seguido de Always + LFU con un 63 %.

4.1.3. Cache Sharing Using Bloom Filters in Named Data Networking [25]

Se define una paquete resumen (*summary packet*) usando *Bloom Filter* y un método para compartir el resumen. Esto cuando un paquete de datos es recibido, el *routers* toma la decisión de si guardar o no el dato, dependiendo de el resultado de la consulta en *caché* de los resúmenes de los *routers* vecinos. Si alguno de los *routers* vecinos tiene el dato, entonces el dato no tiene que ser necesariamente guardado. De otra manera, si un paquete de interés es recibido por un *routers* entonces este mismo puede reenviarlos a sus *routers* vecino analizando sus resúmenes. Todo esto con el fin de aumentar la diversidad de contenido en *caché* en NDN. En este trabajo se utilizo una topología de 6 nodos Sprint PoP Fig. 4.7 Mientras que los ajustes de los parámetros para la simulación se encuentran descritos en la Fig.4.8

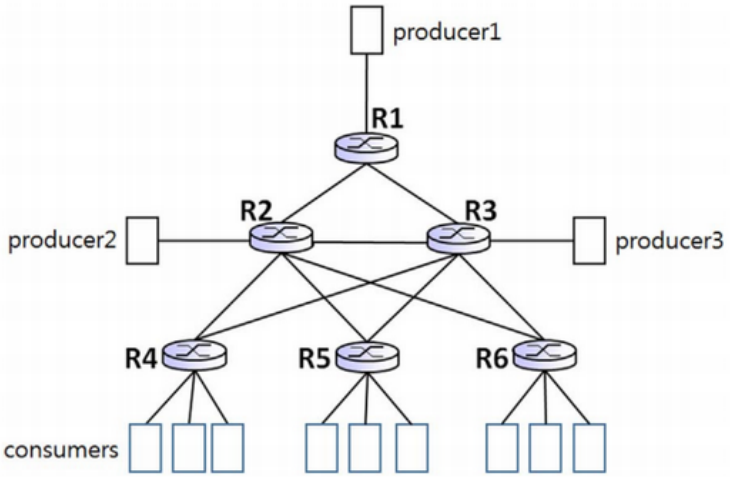

Figura 4.7: Topología 6-node Sprint PoP. Fuente: [36]

Parameter setting.

Parameter	Default value	Range
request rate (λ)	100 *Interests*/s	uniform, Poisson
node capacity	100 contents	
content size	1 KB	
catalog size	900	[600,1500]
Zipf parameter (α)	0.7	[0.5, 0.9]
Bloom filter size	1024 bits	
number of BF indices (k)	2	

Figura 4.8: Ajuste de parámetros. Fuente: Cache Sharing Using Bloom Filters in Named Data Networking [25]

Se aplican las políticas de reemplazo LRU y LFU correspondientes al simulador, siendo LRU-SS y LFU-SS las políticas con resumen compartido. Ademas se utilizo la política LCProb ya mencionada en [43] la cual deja copias de contenido en el camino de la ruta con una cierta probabilidad. El conjunto de métricas para la evaluación el rendimiento de los esquemas corresponden a: Éxito en *caché* (*caché hit*), diversidad de contenido y el tiempo de entrega del contenido.

Los resultados obtenidos por el trabajo se clasifican según el tipo de distribución utilizada (Uniforme y Poisson):

Cache Hit's (%) En cuanto a la métrica de *hit's* de *caché* la política que alcanzo un mayor porcentaje fue la de LFU-SS de la distribución uniforme, la que tuvo un comienzo acierto del 65 % con 200 contenidos, para luego ir descendiendo hasta llegar a un 43 % con 500 contenidos. La que obtuvo peor resultado fue la política LRU de la distribución Poisson.

Diversidad de contenido(%) Los mejores resultados los tuvo la política LRU-SS de la distribución Poisson, con un resultado que va entre un 60 % para luego aumentar a un 68 % aproximadamente en un intervalo de 200 a 500 contenidos respectivamente. El peor rendimiento lo tuvo la política LRU de la distribución Poisson.

Tiempo de entrega del contenido(ms) La política que entrego mejor resultado es la de LRU de la distribución uniforme con un tiempo mínimo de 7,6 para luego aumentar a 10,5, con 200 y 500 contenidos respectivamente. El peor resultado lo dio LFU-SS de la distribución Poisson, con una máxima de 9.5 aprox.

Parámetros Zipf (α) También en este trabajo se obtuvieron análisis del parámetro α de zipf que van desde 0,5 a 0,9 y al igual que los resultados anteriores, están clasificados según el tipo de distribución utilizada. Los Resultados fueron los siguientes:

- **Cache Hit's (%):** Si bien LFU-SS de la distribución uniforme con $\alpha = 0,5$ obtuvo el mejor resultado en la primera instancia con un total de 43 % aprox., a medida que α aumenta LFU-SS y LRU-SS de la misma distribución llegan al mismo resultado de 65 % de *hit's*.

- **Diversidad de contenido(%):** El mejor resultado lo tuvo sin duda la política LRU-SS de la distribución uniforme partiendo con un 66 % para luego seguir manteniéndose hasta llegar a un 65 %.
- **Tiempo de entrega del contenido(ms):** Respecto a esta métrica, el mejor resultado lo tiene la política LRU de la distribución uniforme partiendo con 10,3 ms aproximadamente llegando a un tiempo de 7,6 ms. La peor política es la de LFU-SS tanto en la distribución uniforme y Poisson.

4.2. Arquitecturas de Caches

4.2.1. Two level caching technique for improving result ranking [34].

En el siguiente trabajo [34] se propone un esquema el cual reduce los requerimientos de entrada/salida y el computo, aumentando la escalabilidad de los motores de búsqueda web sin modificar el *ranking* de este. Es una arquitectura de *caché* que se compone de dos niveles que mezcla de forma paralela el *caché* de listas invertidas y el *caché* de resultados.

En el *caché* de listas invertida se quiere evitar el acceso al disco el cual genera retraso, a través de la mantención en memoria de una lista la cual contiene documentos web's ligados a un término de la consulta. Por lo que al llegar una petición con un idéntico termino, esta tiene la posibilidad de ser respondida por la lista previamente almacenada. Además en este *caché* se incluye una técnica para el almacenamiento de las listas invertidas, la cual consiste en ordenar la lista invertida respecto la frecuencia con la que aparece el termino en cada documento, de modo que solo se usaran los termino que tengan alta frecuencia. Por eso es que la lista no se recorre completamente, además estas listas se dividen en bloques de documentos los que contienen los términos de la misma frecuencia.

Por otro lado el *caché* de resultados, consiste en la utilización de una lista de documentos asociados a la consulta, en la cual para cada uno de estos documentos se guarda, el titulo, la URL y 250 caracteres.

El *caché* propuesto en el trabajo [34](Fig. **??**) combina lo mejor de los dos *cachés* anteriormente mencionados, y cada vez que se lleva a cabo una petición se revisa si se encuentra en el *caché* de resultado, si la petición es encontrada (*hit*) entonces se responde inmediatamente y si no, entonces se revisa si se encuentra en el *caché* de listas invertidas, por lo tanto si se genera *hit* se mitiga el numero de acceso al disco pero si se genera *miss* entonces se es necesario acceder a disco a encontrar la respuesta.

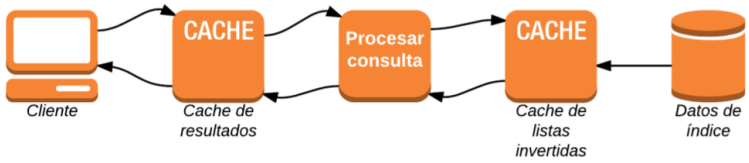

Figura 4.9: Esquema *caché* de dos niveles. Fuente: Two level caching technique for improving result ranking [34]

Como conclusión, los resultados obtenidos de los experimentos realizados en el motor de búsqueda "TodoBR"dan cuenta que el esquema de *caché* utilizado es un 36 % mayor a que si solo se usara el *caché* de resultado, y 52 % en comparación con el *caché* de listas invertidas respecto al *throughput*.

4.2.2. Time-based query classification and its application for page rank [12].

Este trabajo consta de la clasificación de las paginas y consultas con el fin de mejorar el resultado de *ranking*. En la Fig. 4.10 se puede observar como son clasificadas las consultas según el tipo que correspondan respecto a su temporalidad (permanentes, ráfaga, periódicas y ráfaga múltiple).

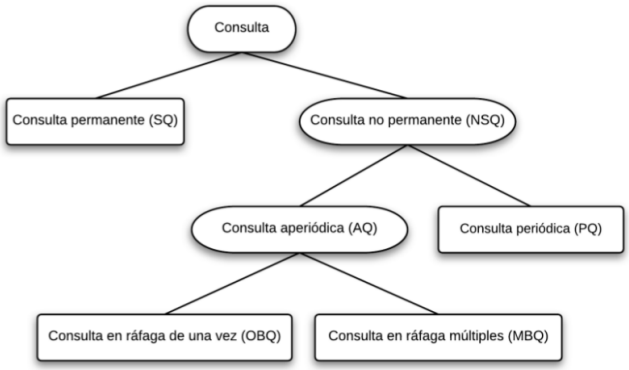

Figura 4.10: Clasificación de consultas basadas en el tiempo. Fuente: Time-based query classification and its application for page rank [12].

Para llevar a cabo lo anteriormente señalado. Primero se desarrolla un algoritmo que es el encargado de clasificar la categoría a la que corresponde una consulta según el registro de las peticiones pasadas. En segundo lugar se usa un modelo dinámico teórico el cual establece el *rank* de las paginas respecto al tiempo de la publicación en la web y la información temporal contenida en la categoría de la consulta.

Entendido lo anterior, la clasificación se lleva a cabo de la siguiente manera. Cuando llega una consulta nueva el algoritmo desarrollado, clasifica la consulta según corresponda (permanente, periódica, ráfaga o ráfaga múltiple). Luego es utilizado el modelo dinámico propuesto (*time-sensitive*), que entrega como salida el *rank* de la pagina y la cual puede variar según la clasificación que fue designada la consulta entrante. Este método de *ranking* usados, superan a los del método *baseline ranking*.

4.3. Organización del Caché

4.3.1. Políticas de desalojo

Según lo leído en la literatura específicamente en los trabajos de [40][28] etas políticas se pueden clasificar en:

4.3.1.1. Basada en la frecuencia

Son estrategias que tienen como base la utilización de la frecuencia (tamaño y/o costo) con la que es llamado un objeto, para decidir si reemplazar o no un objeto en *caché*. Resultando así una mayor cantidad de *hit* en *caché*. Una de estas estrategias basadas en frecuencia es:

- ***Least Frequently Used*** **(LFU)**
  LFU Posee una complejidad algorítmica de $O(\log(n))$ y su función es la de eliminar la entrada menos frecuente referenciada. Pero LFU posee problemas, debido a que si un objeto posee una cantidad de frecuencia muy alta, y luego se convierte en impopular, entonces para que salga de *caché* debe pasar un largo tiempo ha esto se le denomina corrupción de *caché* [40].

4.3.1.2. Basada en los reciente

- ***Least Recententle Used* (LRU)**
  Una de las políticas de reemplazo más populares, todo gracias a su sencillez y muy bien catalogado rendimiento en diferentes escenarios [40] y con una complejidad de $O(1)$. La política LFU desaloja la entrada menos referenciada recientemente en *caché*. Cuando una cantidad grande de usuarios poseen un interés en común de objetos web, entonce se dice que existe una alta localidad temporal, escenario en donde la política LFU se destaca.

- ***Probability Driven Cache* (PDC)[21]**
  Modelo probabilístico desarrollado para motores de búsqueda, debido ha que estos mismos guardan los resultados en memoria *caché*. En [21] se explica que los motores de búsqueda entregan una enorme cantidad de paginas de resultados, de las cuales solo la primera es visitada por la mayoría de os usuarios. Por lo tanto almacenar el resto de paginas no es eficiente. Para mitigar esto, el modelo propuesto prioriza las paginas que están en *caché* respecto al numero de usuarios que actualmente esta navegando las paginas que se dieron como resultados de la consulta.

  La política almacena las paginas de resultados que son visitadas actualmente respecto a la consulta y reemplaza o desaloja los resultados que no son relevantes o bien no poseen un flujo de visitas importante en aquel instante. Entregando como resultado un aumento del 50 % respecto a la política LRU.

4.3.1.3. Basada en los frecuente/reciente

- ***Hyper-G***
  Política que combina lo mejor de tres políticas existentes (LRU, LFU, *Size*), utilizando cada una de estas políticas en el orden respectivo, siempre y cuando se generen conflictos con objetos que cumplen el mismo criterio para el desalojo de este. La complejidad del la política corresponde ha $O(\log(n))$ y fue introducida primeramente en el trabajo de [4].

4.3.1.4. Basada en tamaño del objeto

- ***Size***
  Su Complejidad es de $O(\log(n))$ y se caracteriza por el desalojo prioritario de entradas de mayor tamaño, por medio de colas de prioridad. En muchos casos el tamaño de los objetos web suelen ser fijos, por lo que en estos casos, se toma en cuenta una variable temporal la cual ayuda para el desalojo [40].

4.3.1.5. Basada en funciones

Tal como dice su nombre son estrategias las cuales se basan en la utilización de funciones, y parámetros tales como tiempo, tamaño, costo, latencia, frecuencia, entre otros. Para determinar si un objeto es desalojado o no, algunas de estas estrategias son:

- ***Least Unified Value*(LUV)**
  Política que unifica las ya conocidas políticas LRU y LFU. Por medio de la utilización de la formula 4.4 la que entrega un valor V(i) a cada documento , para esto esto la política utiliza la información de los objetos referenciados recientemente y la frecuencia de estos.

$$V(i) = W(i) * p(i) \tag{4.4}$$

  $W(i)$ corresponde al costo estimado de ir a buscar el objeto al servidor y $p(i)$ a la probabilidad de que un objeto i sea referenciado en el futuro. Para calcular $W(i)$ se utiliza la función la descrita a continuación

$$W(i) = \frac{c_i}{s_i} \tag{4.5}$$

  de la cual c_i representa el costo y s_i el tamaño del objeto. Por otro lado para calcular $p(i)$ se utiliza la siguiente función:

$$p(i) = \sum_{k=1}^{f_i} F(t_c - t_k) \tag{4.6}$$

  donde t_c y t_k son el tiempo actual y el tiempo en que se referencia el documento en un instante k respectivamente. De 4.6 surge también 4.7 el cual entrega el valor de λ y permite a LUV determinar medida que λ es cercano a 1 entonces

se da más énfasis a la información reciente, de lo contrario si λ es cercano a 0 se enfatiza más la información de la frecuencia.

$$F(x) = \left(\frac{1}{2}\right)^{\lambda x}, \text{ donde } 0 \leq \lambda \leq 1 \tag{4.7}$$

LUV reemplaza el documento de menor valor, el que cambia solo cuando se accede a el. La complejidad de LUV es de $O(\log(n))$ y fue nombrado primeramente en el trabajo de [10].

- ***Hybrid***

  Por otro lado se encuentra la política de desalojo *Hybrid* la cual reemplaza los documentos respecto al tiempo de conexión al servidor para acceder a el (c_s), el ancho de banda con el servidor (b_s), el numero de veces referenciado el documento en *caché* (f_i) y el tamaño del documento (s_i). Para determinar que documento ha reemplazar se utiliza la siguiente función:

  $$v_i = \left(c_s + \frac{W_b}{b_s}\right) * \frac{(f_i)^{W_n}}{s_i} \tag{4.8}$$

  donde ademas de las variables mencionadas anteriormente se agregan W_n y W_b que son constantes que determinan la importancia relativa de las variables f_i y b_s respectivamente.

  Según lo descrito en 4.8 la política de desalojo *Hybrid* tiene menores probabilidades de reemplazar los documentos que poseen un ancho de banda de conexión bajo, el tamaño del documento es pequeño, es referenciado varias veces y/o el tiempo para conectarse al servidor que lo contiene es alto. La complejidad de la política es de $O(\log(n))$.

4.3.1.6. Asignación al azar

LRU-S: Política nombrada en el trabajo de , y corresponde a la utilización de la densidad normalizada de un objeto i correspondiente ha $d_i = \frac{S_m in}{S_i}$, donde $S_m in$ corresponde al tamaño mínimo de los N objetos almacenados en *caché*. Por lo tanto LRU-S varia de lugar los objetos i según el valor entregado por d_i. La complejidad de LRU-S es de $O(1)$.

4.3.2. Políticas de admisión

4.3.2.1. Admission Policies for Caches of Search Engine Results [8].

Política de admisión propuesta en el trabajo de [8] la que busca clasificar si una consulta es frecuente o infrecuente antes de que esta entre en *caché* por medio de un estimador. El diseño de la política, consiste en dividir el *caché* en dos. La primera sección están las consultas que la política predice podrían tener *hitś*, esta sección es denominada *caché* controlado (CC). La segunda sección es denominada *caché* no controlado (UC) y es donde se envían las consultas que no fueron aceptadas por la política.

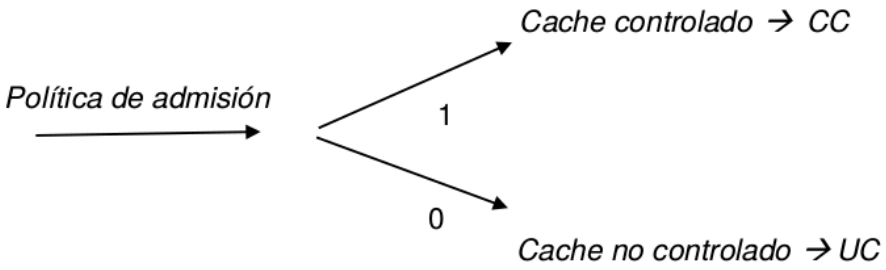

Figura 4.11: Clasificación de consultas según política. Fuente: Admission Policies for Caches of Search Engine Results [8].

Para lograr entrar al *caché* controlado se toma en cuenta la política de admisión basada en umbral K. Existen dos, el primero es una umbral estático y respecto al numero de palabras, y el segundo corresponde a un umbral dinámico el cual toma en cuenta lo caracteres no alfanuméricos en la consulta, de modo que una consulta con varios caracteres alfanuméricos no llegara a ser popular. Los resultados demuestran una mejora del 21 % respecto ha *Least Recentently Used* (LRU) y un 4 % en relación a *Static Dynamic Cache* (SDC) de un total de 100.000 consultas.

4.3.2.2. TinyLFU: A highly efficient cache admission policy

Política propuesta en [14], la cual busca aumentar la eficacia del *caché*. Esta política se basa en la frecuencia aproximada y consiste en en tener almacenadas las estadísticas de las frecuencia de los ítems, con el objetivo de decidir si permitir la nueva entrada, tomando en cuenta al candidato a desalojar en *caché* la cual se le denomina "víctima".

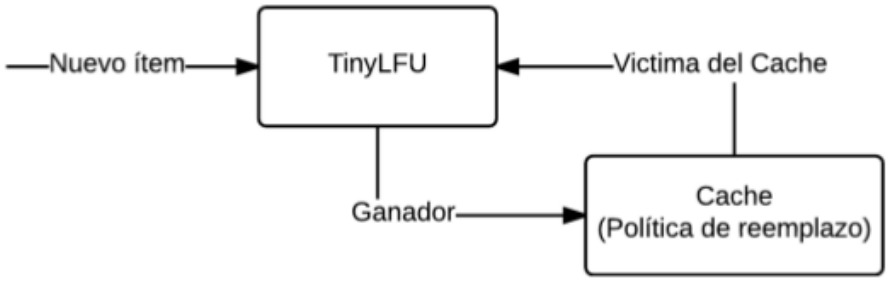

Figura 4.12: Arquitectura TinyLFU. Fuente:.[14]

Debido ha que almacenar la estadística de las frecuencia de los ítems tiene un gran costo, es necesario llevar a cabo técnicas de recuento aproximado para las frecuencias con el fin de disminuir el consumo de memoria. Una técnica para esto es la denominada *Counting Bloom Filter* (CBF).

El *Bloom Filter* es una estructura con datos probabilísticos lo cual permite saber de manera eficiente si un dato existe o no en un conjunto. Por lo que el *Counting Bloom Filter* es un tipo de *Bloom Filter*. Para cada política utilizada (LRU, Random, TLRU (LRU incorpora TinyLFU), TRandom (Random incorpora TinyLFU), TLFU (LFU incorpora TinyLFU), W-TinyLFU, LIRS y ARC). Las conclusiones obtenidos por esta demuestran que la tasa de *hit* en las que utilizan esta política de admisión es mejor en comparación con las otras, destacándose entre estas TLRU y W-TinyLFU.

Capítulo 5

Propuesta

Para lo anterior, la solución propuesta en este trabajo consiste en desarrollar una arquitectura de caché para los nodos de una red NDN, tomando en cuenta el comportamiento dinámico de los usuarios. Para ello se utilizara el simulador ndnSIM el que ayudara a recrear una red basada en contenidos y obtener los resultados de la arquitectura caché propuesta.

5.1. Escenario

5.1.1. Topologia y enlaces

El conjunto de nodos interconectados entre si o dicho de otra manera la topologia utilizada para llevar a cabo este trabajo, es descrita como una matriz de tres por tres, la cual se encuentra dentro del la categoría de topologia de árbol. El enlace entre los nodos sigue un protocolo PPP (Point-to-Point Protocol). La representación gráfica de la topologia se puede apreciar en la Fig. 5.1

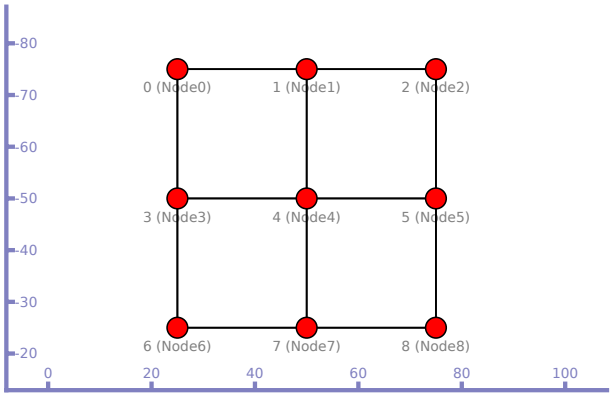

Figura 5.1: Topologia utilizada. Fuente: Elaboración propia

A continuación, en la Fig, se puede apreciar parte del código principal (main) en el cual se realiza la lectura de un archivo de texto, que contiene la estructura y los datos necesarios para crear la topologia usada en la simulación. Esto gracias a la clase 'AnnotatedTopologyReader'.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/ndnSIM-module.h"

namespace ns3 {

int
main(int argc, char* argv[]) // main
{
CommandLine cmd;
cmd.Parse(argc, argv);

AnnotatedTopologyReader topologyReader("", 25);
topologyReader.SetFileName("extensions/topo-grid-3x3.txt"); // Se
   llama al archivo
topologyReader.Read(); // Lecura del archivo con la topologia.
```

Listing 5.1: Codigo fuente: main

Por otro lado en la Fig. se encuentra un fragmento del archivo de texto leído 'topo-grid-3x3-.txt' en donde se pueden configurar la estructura y los parámetros para el trafico de datos.

```
...
# node   comment      yPos     xPos
Node0    NA           3        1
Node1    NA           3        2
Node2    NA           3        3
Node3    NA           2        1
Node4    NA           2        2
Node5    NA           2        3
Node6    NA           1        1
Node7    NA           1        2
Node8    NA           1        3
...

# srcNode     dstNode     bandwidth    metric     delay    queue
Node0         Node1       1Mbps        1          10ms     10
Node0         Node3       1Mbps        1          10ms     10
Node1         Node2       1Mbps        1          10ms     10
Node1         Node4       1Mbps        1          10ms     10
Node2         Node5       1Mbps        1          10ms     10
Node3         Node4       1Mbps        1          10ms     10
Node3         Node6       1Mbps        1          10ms     10
Node4         Node5       1Mbps        1          10ms     10
Node4         Node7       1Mbps        1          10ms     10
Node5         Node8       1Mbps        1          10ms     10
Node6         Node7       1Mbps        1          10ms     10
Node7         Node8       1Mbps        1          10ms     10
```

En donde en la primera tabla donde se define los *routers* ubicándolos en un plano cartesiano (X,Y) dentro del simulador y una segunda tabla en la que se definen los enlaces y sus variables tales como el nodo salida(srcNode), nodo destino(dstNode), ancho de banda(bandwidth), métrica de enrutamiento(metric), retraso(delay) y cola(queue) la que ajusta el máximo de transmisión de paquetes en el enlace(*link*).

5.1.2. Arquitectura del caché

```
NodeContainer nodo1, nodo2, nodo3, nodo4, nodo5, nodo6 ,nodo7,
    consumerNodes;

// Se instancia la variable ndnHelper
ndn::StackHelper ndnHelper;

ndnHelper.SetOldContentStore("ns3::ndn::cs::Double", "MaxSize", "
    10000");

nodo1.Add(Names::Find<Node>("Node1"));// Se busca el nodo segun
    nombre
ndnHelper.Install(nodo1); // Se instala la politica de reemplazo en
    el nodo 1 (ROUTER)
.
.
.
ndnHelper.SetOldContentStore("ns3::ndn::cs::Nocache"); // Se define
    la politica de NO CACHE

consumerNodes.Add(Names::Find<Node>("Node0")); // Se busca el nodo
    segun nombre
Ptr<Node> producer = Names::Find<Node>("Node8");// Se busca el nodo
    segun nombre

ndnHelper.Install(consumerNodes); // Se instala la politica de
    reemplazo en el nodo CONSUMIDOR
ndnHelper.Install(producer); // Se instala la politica de reemplazo
    en el nodo PRODUCTOR
```

Listing 5.2: Caption

5.1.3. Consultas

5.1.3.1. Consumidor

```
  std::string prefix = "/prefix";

  ndn::AppHelper consumerHelper("ns3::ndn::ConsumerCbr"); // Se crea
      la instancia
  consumerHelper.SetPrefix(prefix);        // Seteo del prefijo
  consumerHelper.SetAttribute("Frequency", StringValue("1")); //
     Seteo de la frecuencia en que enviara
  // los intereses (1 por segundo).
  consumerHelper.Install(consumerNodes); // Se installa la
     aplicacion en uno o mas nodos.
```

Listing 5.3: Caption

5.1.3.2. Productor

```
ndn::AppHelper producerHelper("ns3::ndn::Producer"); // Se crea la instancia
producerHelper.SetPrefix(prefix); //Seteo del prefijo
producerHelper.SetAttribute("PayloadSize", StringValue("1024")); //
producerHelper.Install(producer); // Se instala la aplicacion en una o mas nodos.
```

Listing 5.4: Caption

5.2. Arquitectura cache

5.2.1. Instalación

Capítulo 6

Enfoques metodológico

Capítulo 7

Pruebas y resultados

Capítulo 8

Conclusiones

8.1. Trabajo Futuro

Bibliografía

[1] Future internet architectures – next phase (fia-np) (nsf13538). https://www.nsf.gov/pubs/2013/nsf13538/nsf13538.htm.

[2] Future internet architectures (fia) nsf10528. https://www.nsf.gov/pubs/2010/nsf10528/nsf10528.htm.

[3] Named data networking: Executive summary - named data networking (ndn). https://named-data.net/project/execsummary/.

[4] Marc Abrams, Charles R Standridge, Ghaleb Abdulla, Edward A Fox, and Stephen Williams. Removal policies in network caches for world-wide web documents. In *Acm sigcomm computer communication review*, volume 26, pages 293–305. ACM, 1996.

[5] Charu Aggarwal, Joel L Wolf, and Philip S. Yu. Caching on the world wide web. *IEEE Transactions on Knowledge and data Engineering*, 11(1):94–107, 1999.

[6] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Borje Ohlman. A survey of information-centric networking. *IEEE Communications Magazine*, 50(7), 2012.

[7] Ismail Sengor Altingovde, Rifat Ozcan, and Özgür Ulusoy. A cost-aware strategy for query result caching in web search engines. In *European Conference on Information Retrieval*, pages 628–636. Springer, 2009.

[8] Ricardo Baeza-Yate, Flavio Junqueira, Vassilis Plachouras, and Hans Friedrich Witschel. Admission policies for caches of search engine results. In *International Symposium on String Processing and Information Retrieval*, pages 74–85. Springer, 2007.

[9] Ricardo Baeza-Yates and Simon Jonassen. Modeling static caching in web search engines. In *European Conference on Information Retrieval*, pages 436–446. Springer, 2012.

[10] Hyokyung Bahn, Kern Koh, Sam H Noh, and SM Lyul. Efficient replacement of nonuniform objects in web caches. *Computer*, 35(6):65–73, 2002.

[11] Berkant Barla Cambazoglu, Flavio P Junqueira, Vassilis Plachouras, Scott Banachowski, Baoqiu Cui, Swee Lim, and Bill Bridge. A refreshing perspective of search engine caching. In *Proceedings of the 19th international conference on World wide web*, pages 181–190. ACM, 2010.

[12] Z Chen, H Yang, J Ma, J Lei, and H Gao. Time-based query classification and its application for page rank. *J Comput Info Sys*, 7:3149–3156, 2011.

[13] Kideok Cho, Munyoung Lee, Kunwoo Park, Ted Taekyoung Kwon, Yanghee Choi, and Sangheon Pack. Wave: Popularity-based and collaborative in-network caching for content-oriented networks. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, pages 316–321. IEEE, 2012.

[14] Gil Einziger and Roy Friedman. Tinylfu: A highly efficient cache admission policy. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pages 146–153. IEEE, 2014.

[15] Qingqing Gan and Torsten Suel. Improved techniques for result caching in web search engines. In *Proceedings of the 18th international conference on World wide web*, pages 431–440. ACM, 2009.

[16] Carlos Luis Gómez Pantoja. Servicios de cache distribuidos para motores de búsqueda web. 2014.

[17] Van Jacobson. A new way to look at networking. *Google Tech Talk*, 30, 2006.

[18] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12. ACM, 2009.

[19] Chetan Kumar and John B Norris. A new approach for a proxy-level web caching mechanism. *Decision Support Systems*, 46(1):52–60, 2008.

[20] Dirk Kutscher, Suyong Eum, Kostas Pentikousis, Ioannis Psaras, Daniel Corujo, Damien Saucez, T Schmidt, and Matthias Waehlisch. Information-centric networking (icn) research challenges. Technical report, 2016.

[21] Ronny Lempel and Shlomo Moran. Predictive caching and prefetching of query results in search engines. In *Proceedings of the 12th international conference on World Wide Web*, pages 19–28. ACM, 2003.

[22] Evangelos P. Markatos. On caching search engine query results. *Computer Communications*, 24(2):137–143, 2001.

[23] Karen Mossberger, Caroline J Tolbert, and Ramona S McNeal. *Digital citizenship: The Internet, society, and participation*. MIt Press, 2007.

[24] Paul Müller and Bernd Reuther. Future internet architecture–a service oriented approachfuture internet architecture–ein serviceorientierter ansatz. *it-Information Technology*, 50(6):383–389, 2009.

[25] Ju Hyoung Mun and Hyesook Lim. Cache sharing using bloom filters in named data networking. *Journal of Network and Computer Applications*, 2017.

[26] SV Nagaraj. *Web caching and its applications*, volume 772. Springer Science & Business Media, 2004.

[27] Rifat Ozcan, I Sengor Altingovde, B Barla Cambazoglu, Flavio P Junqueira, and ÖZgüR Ulusoy. A five-level static cache architecture for web search engines. *Information Processing & Management*, 48(5):828–840, 2012.

[28] Stefan Podlipnig and Laszlo Böszörmenyi. A survey of web cache replacement strategies. *ACM Computing Surveys (CSUR)*, 35(4):374–398, 2003.

[29] Ioannis Psaras, Wei Koong Chai, and George Pavlou. In-network cache management and resource allocation for information-centric networks. *IEEE Transactions on Parallel and Distributed Systems*, 25(11):2920–2931, 2014.

[30] Jianhua Ran, Na Lv, Ding Zhang, Yuanyuan Ma, and Zhenyong Xie. On performance of cache policies in named data networking. In *International Conference on Advanced Computer Science and Electronics Information*, pages 668–671, 2013.

[31] Pablo Rodriguez, Christian Spanner, and Ernst W Biersack. Analysis of web caching architectures: Hierarchical and distributed caching. *IEEE/ACM Transactions on Networking (TON)*, 9(4):404–418, 2001.

[32] Giuseppe Rossini and Dario Rossi. Evaluating ccn multi-path interest forwarding strategies. *Computer Communications*, 36(7):771–778, 2013.

[33] Paricia Correia Saraiva, Edleno Silva de Moura, Novio Ziviani, Wagner Meira, Rodrigo Fonseca, and Berthier Riberio-Neto. Rank-preserving two-level caching for scalable search engines. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 51–58. ACM, 2001.

[34] Amarjeet Singh, Mohd Hussain, and Rakesh Ranjan. Two level caching techniques for improving result ranking. *Bharati Vidyapeeth's Institute of Computer Applications and Management*, page 365, 2011.

[35] Víctor Jesús Sosa Sosa and Leandro Navarro Moldes. *Arquitectura para la distribución de documentos en un sistema distribuido a gran escala*. 2002.

[36] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring isp topologies with rocketfuel. *SIGCOMM Comput. Commun. Rev.*, 32(4):133–145, August 2002.

[37] Saran Tarnoi, Kalika Suksomboon, Wuttipong Kumwilaisak, and Yusheng Ji. Performance of probabilistic caching and cache replacement policies for content-centric networks. In *Local Computer Networks (LCN), 2014 IEEE 39th Conference on*, pages 99–106. IEEE, 2014.

[38] Jia Wang. A survey of web caching schemes for the internet. *ACM SIGCOMM Computer Communication Review*, 29(5):36–46, 1999.

[39] Jianguo Wang, Eric Lo, Man Lung Yiu, Jiancong Tong, Gang Wang, and Xiaoguang Liu. The impact of solid state drive on search engine cache management. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 693–702. ACM, 2013.

[40] Kin-Yeung Wong. Web cache replacement policies: a pragmatic approach. *IEEE Network*, 20(1):28–34, 2006.

[41] Yinglian Xie and David O'Hallaron. Locality in search engine queries and its implications for caching. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1238–1247. IEEE, 2002.

[42] George Xylomenos, Christopher N Ververidis, Vasilios A Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V Katsaros, and George C Polyzos. A survey of information-centric networking research. *IEEE Communications Surveys & Tutorials*, 16(2):1024–1049, 2014.

[43] Guoqiang Zhang, Yang Li, and Tao Lin. Caching in information centric networking: A survey. *Computer Networks*, 57(16):3128 – 3141, 2013. Information Centric Networking.

[44] Weizhe Zhang, Hui He, and Jianwei Ye. A two-level cache for distributed information retrieval in search engines. *The Scientific World Journal*, 2013, 2013.

[45] George Kingsley Zipf. *The psycho-biology of language: An introduction to dynamic philology.* Routledge, 2013.

[46] George Kingsley Zipf. *Human behavior and the principle of least effort: An introduction to human ecology.* Ravenio Books, 2016.