Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2015-002 January 31, 2015

Improved Caching Strategies for Publish/Subscribe Internet Networking

Kendra K. Beckler

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Improved Caching Strategies for

Publish/Subscribe Internet Networking

by

Kendra K. Beckler

S.B. M.I.T. 2009

Submitted to the Department of Electrical Engineering

and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

September 2014

Author: _____
Department of Electrical Engineering and Computer Science
September 2014

Certified by: _____
Karen Sollins, Principal Research Scientist, Thesis Supervisor
September 2014

Accepted by: _____
Prof. Dennis M. Freeman, Chairman, Masters of Engineering Thesis Committee

Improved Caching Strategies for Publish/Subscribe Internet Networking
by
Kendra K. Beckler
Submitted to the
Department of Electrical Engineering and Computer Science

September 2014

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

The systemic structure of TCP/IP is outdated; a new scheme for data transportation is needed in order to make the internet more adaptive to modern demands of mobility, information-driven demand, ever-increasing quantity of users and data, and performance requirements. While an information centric networking system addresses these issues, one required component for publish subscribe or content-addressed internet networking systems to work properly is an improved caching system. This allows the publish subscribe internet networking to dynamically route packets to mobile users, as an improvement over pure hierarchical or pure distributed caching systems, To this end, I proposed, implemented, and analyzed the workings of a superdomain caching system. The superdomain caching system is a hybrid of hierarchical and dynamic caching systems designed to continue reaping the benefits of the caching system for mobile users (who may move between neighboring domains in the midst of a network transaction) while minimizing the latency inherent in any distributed caching system to improve upon the content-addressed system.

Contents

List of Figures

1 Introduction

In the design of networks, traditionally local network storage has been rather limited in both capacity and utility. In the current Internet design, small amounts of storage are used for routing information and to provide extremely short-term buffering of packets, in support of reliable delivery. Long term and larger scale local storage as a network resource is one of the enabling concepts for the ideas now called "Information Centric Networking" (ICN). In this project we propose and evaluate a design for regional cooperative storage or cache management as an enhancement to the local cache models currently enabling ICN designs.

The packet architecture of the internet was designed in the 1960s and '70s in order to solve the problem of sharing computing resources of a few large supercomputers.

The unifying and common layer in the protocol stack is the Internet Protocol (IP). It provides best effort delivery of packets from one location to another. The most commonly used protocol that sits on top if IP is the Transmission Control Protocol (TCP), which layers ordered and reliable delivery of packets from a source to a destination, on top of IP. Another protocol that also sits on top of IP is UDP, which supports streams of packets, but without the ordering and reliability of TCP. The point that is central to our current work is that all these protocols have as a central tenet that communications traffic flows between a sender's and receiver's location or address, without any reference to what is being transported, only where it is coming from and where it is going.

Current network technology is defined on a point-to-point basis, using the physical location of the hosts as the fundamental unit. However, this does not coincide with modern usage of the technology, which is more concerned with the

data in question than where it is located. Large companies have many servers masquerading behind the same IP address to deal with millions of requests. Users have multiple devices, and they want their information to be available with equal ease on all of them; users also have mobile devices, laptops and PDAs and smartphones, and expect uninterrupted network access as they cross boundaries, whether of domain or of connection type.

There is a paradigm shift in the works, an emphasis on the *what* of network data instead of the *where*. This makes the content, the data, the fundamental unit of network transactions, without the user knowing the physical address for where it is stored. The common term for this set of ideas is "Information Centric Networking," or ICN. This content-based networking, paired by some designs with the publish/subscribe model of networking, is currently being designed to be a low-overhead network protocol. The publish/subscribe model will theoretically allow multiple distributed sources of a piece of content to be accessed equally, users to continue receiving data even if their mobile device changes transmission styles or crosses domain boundaries, and popular data to be cached around the world to minimize the needed bandwidth of its original source.

One aspect common to all publish/subscribe and content-addressed systems is the existence of a sophisticated caching system. They range from small caches attached to every router or node on the network to very large regional caches run by the local Internet Service Provider (ISP). All these systems provide authentication, integrity, and in most cases authorization control inherent in the objects themselves, thus relieving the system from having to depend on the original source of the information for those capabilities. Because if this, these systems can take advantage of widespread caching and in-network storage. These systems can utilize caching strategies to simultaneously minimize the

bandwidth used by the source of a popular piece of data and to decrease the time between a request for the data and the data delivery by storing a cached copy of the data within a few hops of the requester.

Information centric networking systems can utilize caching much more effectively and naturally than the current IP-based internet paradigm. The reason for this is that in the current internet, a source of a given piece of information can be sending it to two users proximal to each other at the same time, yet IP-routing metadata (session ID, packet number, etc.) does not label the contained information, meaning that the source has to separately send the packet to both of the users. However, in content-oriented internet architectures, this traffic need not be duplicated.

The specific question that I answer is the following. Assuming a futuristic information centric, publish/subscribe networking system, including a basic assumed system of local-area caching, how can the caching system be improved to increase performance?

This project evolved from the observation that cooperation and sharing of caches could provide enhanced performance when that cooperation remained reasonably local. My proposed solution to this question involves the creation of a concept that I call superdomains. A superdomain is a cooperative system of geographically proximal domains, each with their own rendezvous node (which handles routing of requests to nodes not in its domain) and local cache. In my system, all caches in a superdomain cooperate to effectively increase the cache hit rate for all of their proponent nodes' requests, without the increased lag from a fully distributed or hierarchical caching scheme. The geographical proximity of the domains minimizes the round-trip time of querying neighbor caches, and the geostationary nature of a superdomain makes it an easy location-based routing target, while not requiring users to stay put. The

challenge in this effort was, within the context of information centric networking, to improve upon traditional caching methods.

The rest of the thesis begins with a **Background** section, which serves as an overview of the information centric and caching technologies. Next, the **Related Works** section goes into more detail about many of the specific similar systems which have informed my thinking about the problem space. The **Design** section describes the systemic design of my superdomain system. Following that section, the **Experimentation** section describes the software I wrote as a proof of concept of the superdomain system. It also shows the direct results obtained from the experiment. The thesis concludes with some thoughts and overview in the **Conclusion** section; then the **Future Work** section describes some next steps to extend the work herein described.

The contribution of this work is a modular system for cache cooperation which can be added onto many different networking systems. This design improves user experience with response times in the majority of cases, at user-imperceptible overhead in response times.

2 Background

This chapter will elaborate on the high-level overview of the publish/subscribe and caching technology from Chapter 1. This chapter provides the reader with the necessary background in the two main research fields necessary to this thesis: caching and information centric networking.

2.1 Publish/Subscribe Technology

There are two leading designs of content-oriented networking technology in progress, Content-Centric Networking (CCN) and PURSUIT: Publish/Subscribe Internet Technology. They are competing possible future replacements for TCP/IP.

Both CCN and PURSUIT, as well as other proposed publish/subscribe variants, adhere to similar principles. Central to these efforts is that information or content of potential interest is named within the architecture. These names in turn allow for the separable announcement or publication of the information from the interest or request for the information by name. This, in turn, has led to these systems supporting their variants on publish/subscribe. Importantly, this allows for the decoupling of publishers from subscribers in space, time, and synchronization. Decoupling in space allows publishers and subscribers to know nothing about each other nor need to keep references to each other. Decoupling in time allows publishing, subscribing and data delivery to happen without requiring both publisher and subscriber to be simultaneously connected to the network. Decoupling in synchronization allows publishing and subscribing to happen concurrently and asynchronously without blocking on a response [11].

Eugster et al. wrote a survey paper which summarizes the common challenges

and approaches of publish/subscribe networking systems in current research[11]. This paper proposes that the common underpinning of the publish/subscribe variants is the full asynchronistic decoupling of publishers (content providers) and subscribers (content consumers) in time and space.

The basic interaction architecture in any publish/subscribe system is as follows: subscribers express interest in something (an event or a pattern of events), then a publisher causing an event (with the publish operation) that matches the subscriber's registered interest causes a notification to be sent to the subscriber. This requires the existence of a neutral third-party system which handles subscription requests and distributes information upon events occurring.

The three primary types of publish/subscribe systems are topic-based, content-based, and type-based. Topic-based publish/subscribe systems are distinguished by the use of keyword-identified groups, and users subscribe to a topic(i.e. the topic stock quotes), with implied subscriptions to all of its subtopics. Content-based publish/subscribe systems are distinguished by subscriptions based on the actual content (e.g. a specific picture or article). Type-based publish/subscribe systems are distinguished by subscriptions based on the content and also the structure of the information (i.e. information with content about stock quotes which has a structure of stock requests, which brokers use to express interest in buying stock).

According to Carzaniga et al.[4], the two critical parts of content-based internetworking are information-centric addressing and a publish/subscribe event notification system. Both pieces are integral to a true content-addressed networking paradigm. They classify publish/subscribe event notifications as a form of content-based communication, neatly fitting the two models together. Furthermore, they contend that on-demand content delivery is the most important (and sometimes only) primitive in these systems; alternate systems

use producer-initiated transmission, wherein the publisher of some information sends an interest packet, complete with call-back prefix, to effectively poll potential consumers, however this system in inefficient in network resource usage.

Carzaniga also touches upon the problem of accommodating both short-term and long-lived "interests" of the consumers. Most current content-based and publish/subscribe networks focus primarily on only one of those two.

2.1.1 CCN: Content-Centric Networking

CCN, also known as Named Data Networking (NDN), finds published items through transmission packets that are named "objects." Every request begins with an Interest packet. The user's node sends the Interest packet to the nearest node(s) of the network. Each node of the network has three data structures: the content store, the Pending Interest Table (or PIT), and the Forwarding Information Buffer (FIB). The content store contains storage for locally published data and a cache of seen data, stored by the name of the data. The PIT stores information about all Interest packets that are not yet resolved nor timed out. The FIB contains forwarding information about the many "faces," or connections to other network nodes; a local "strategy" determines how these multiple "faces" will be used for each Interest packet, defaulting to blanket multicast if needed. A node which receives an Interest packet will reply with a data packet if the named content is present in the content store, or log the Interest (and requesting network face) in its PIT and forward the Interest packet to other nodes it is connected to.

The nodes of the network which have seen this Interest packet blooms outwards until each either finds a dead end (in which case the Interest packets will time out after a set period of time) or a source of the requested data. Any Interest

packet which reaches the originator of the data or a cached copy of the data is successful. The publisher then sends the data back over the network by the same path, using the Pending Interest Table (which lists the request and the interface(s) from which the node received the Interest) as a trail of breadcrumbs back to the interested host.

Moreover, CCN ubiquitously caches Data packets. Any Data packet that arrives at any node which fits an entry in that node's Pending Interest Table is immediately stored in that node's Content Store. Items in the Content Store are replaced by new ones in either LRU (Least Recently Used) or LFU (Least Frequently Used) basis.

This strategy allows the most popular items on the network to be delivered to the requester very quickly but introduces a high rate of churn into the system. Since every node on the network behaves in this exact same fashion, less-popular items do not remain in any cache for very long and are quickly replaced while other items will be stored in an overabundance of caches all at once.

A benefit of CCN is the uncoordinated nature of decisions. Publishing and subscribing are decoupled in time and space. Nodes do not cooperate, so there is a minimum of network management overhead.

CCN would be greatly improved by the addition of more sophisticated caching strategies than simple LRU or LFU replacement. [17] [44]

2.1.2 PURSUIT: Publish/Subscribe Internet Technology

Some of the critical component parts of PURSUIT are described here. PURSUIT is a publish/subscribe internet technology built on the older PSIRP project. It is a content-oriented networking variant primarily differentiated by its use of a rendezvous system and scoping. The rendezvous system uses the

scopes to forward subscription requests to a publisher of that information using only identifiers on the data. The scope system is a recursive set of data objects which have rendezvous information for the published items. As published data objects themselves, scopes are treated just like information objects, but the data contained therein has rendezvous information for other data objects (some of which can be other scopes).

PURSUIT also is based on named objects that are made available independently of requests or interest, but is different from CCN in several key ways. First, the objects it is designed to support can be more abstract and complex than single packets. Its objects can be the kinds of objects of interest to humans and applications. Second, it separates the discovery of an object, or rendezvous, from the responding data delivery or "forwarding". Finally, the architecture includes the concept of scopes, domains within which objects are published.

2.1.2.1 Rendezvous System In the PURSUIT system, requests for data are forwarded to the local Rendezvous Node (RN), which is a component of a network-wide Rendezvous System. The goal of the Rendezvous System is to track where all data resides in the network through publishing of Scopes, specialized data items which list which data items are published and known about by a certain RN. The Rendezvous System finds which RNs have Scopes containing the requested data, then pass this information to the nearest (modularly separated) dedicated network authority, a Topology Manager (TM).

Each TM is a dedicated node of the network which tracks the connectivity of the network. Requests for a path from the Rendezvous System are phrased in the form of a graph search request (using the igraph module) from any of the sources of the published data to the requester, returning the shortest path across the network.

Most likely, if this system becomes implemented, the Internet Service Providers would be responsible for providing the TM and rendezvous services for the PURSUIT system. Caching would likely be handled by either the ISPs or by third-party entities, similar to how current CDNs function.

Another specialized server employed by the ISP in each domain, or local network, is a cache server with large amounts of storage capacity. Each cache server has full agency as a machine on the network, able to subscribe to data and to publish data items or scopes via its local RN. The cache servers can also pre-cache any new releases from popular publishers for minimal network impact upon the software release date. In addition, they can opportunistically cache data requested by nearby users. These cache servers will be able to utilize more sophisticated cache replacement strategies than simple LRU or LFU replacement in order to efficiently manage hundreds of terabytes of storage and anticipate their domain's users' needs.

One advantage of the PURSUIT system is separating these functions into modular groups handling communication (Rendezvous), the graph structure of the network (Topology Management), and data storage (Caching). However, the caching system is underdeveloped despite being an integral part of this triumverate. We have defined some goals of this caching system next, and later will suggest some strategies which will improve caching in PURSUIT and other publish/subscribe networking systems. [18] [39]

2.1.2.2 Scopes Scopes in PURSUIT are essentially abbreviated advertisements for information items that a given Topology Manager knows about, higher abstraction level and more complex rendezvous information. Scopes themselves are information items, and therefore scopes may list other scopes as information items to which the scope's host TM can route requests.

By each TM advertising its scope(s) to neighbors in the network, requests can be routed by following the trail of scopes back to a source of the information.[26]

2.2 Caching Terminology and Background

Caching is very important to any networking system. Caching allows network traffic for popular items to use only edge networks, clearing some traffic from the congested core and keeping servers of popular information from overloading. Caching also provides some duplication of data and resiliency to network failure on the part of the data originator.

Here is some basic caching-related terminology that are useful reminders, as these will be used freely hereafter.

- **Collaborative caching system:** a system by which multiple individual cache servers communicate with each other to increase caching potential, yet appear to the user/requester as if it were a single cache.
- **Cache hit:** a request to a cache server which can be fulfilled by the cache server (in the case of collaborative caching systems, a request which can be fulfilled by any machine within the caching system, not just one server).
- **Cache miss:** a request to the cache server which cannot be fulfilled by the cache server and must be requested from the original source (in the case of collaborative caching systems, a request which cannot be fulfilled by any machine in the caching system).
- **Cache hit ratio:** the ratio of cache hits to cache misses, a standard indicator of quality for a caching system.
- **LRU:** "least recently used," a cache replacement policy which deletes the

least recently requested cache entries when memory must be freed.

- **LFU:** "least frequently used," a cache replacement policy which deletes the cache entries requested the fewest times per period.

2.2.1 Initial Cache Seeding

With sufficient prior request data, a cache can be seeded with the most commonly requested items in the training set, which is more effective at predicting users' needs and generating a higher rate of cache hits than a cache which starts empty, or only seeded with a few specific items chosen by the system administrator. However, this strategy for initially filling a cache works on a training set of data, which is difficult to impossible to acquire for the internet packets requested by a specific campus or neighborhood. Initial cache seeding improves latency for many predictable queries, such as to the newspaper or popular search results, but would likely not be tailored to the neighborhood's specific interests.

In addition, the decision unit may mark items for deletion or re-subscription based on the time of day, release cycle for a specific publisher of data, or other criteria. This is an extension of the pre-caching discussed in Trossen's Conceptual Architecture document [39]. That pre-caching strategy only allows for soon-to-be-released large, popular software to have caches around the world to ease the burden on global internet traffic right when it is released; their pre-caching allows the majority of user requests to be routed to a nearby cache server instead of requiring all data packets to be sent from the far-away authoritative server.

Trossen's Conceptual Architecture[39] is concerned primarily with pre-loading the cache at its initial startup. However, this pre-caching strategy can be

extended to the notion of dynamically pre-loading into each cache server information items for anticipated requests. This processing is done by a decision unit which watches network traffic and strategically precaches information items in caches all around the network (or a subsection of the network). Centralizing the strategic caching decision-making unit has the advantage of coordinating cache efforts and preventing unnecessary duplication of cached copies.

2.2.2 Opportunistic Caching Versus Pre-Caching

There are two types of cache decision-making which can benefit a publish/subscribe information networking system, opportunistic caching and pre-caching.

Opportunistic caching is the standard method for filling caches. In opportunistic caching, the cache software intercepts data requested by a user, choosing whether or not to cache that data locally. Opportunistic caching decision-making systems do not choose whether or not to cache an item until and unless a user request prompts it to do so.

Pre-caching is a technique where caches are loaded with certain data before it is requested by any users. Some instances of pre-caching require human intervention to accomplish, while others[12] are under development to automaticaly predict requests and cache them. A sample use of pre-caching is to pre-warm caches around the world with a new Microsoft Windows Service Pack release shortly before the release goes public in expectation of heavy load. Doing so would noticeably decrease the impact of the release on the worldwide internet.

2.2.3 Cache Filling Strategies

Using caching in information centric systems is critical to system functionality. These systems' efficiency hinges on not needing to request data from the original source every time. Since publish/subscribe systems are content-based rather than location-based, caching of raw data, in the form of individual packets or information items, can become an integral and fundamental part of the system.

In the current internet, some companies (such as Content Delivery Networks, or CDNs, see subsubsection 3.3.2) have servers around the world or cache their own data, such as the responses to common web search queries. However, these cases usually cache an entire web page, the full list of top search results for that query, data for one specific user, or otherwise do not fully utilize the flexibility that caching on the level of an individual packet or information item would afford. This happens because the current internet is strongly location-based and packets are classified by what location they are being sent to rather than by what information they contain.

Content-oriented internet systems, however, are content-based systems instead of location-based. This has the benefit of all data being labelled as part of a certain piece of information, which can be more easily reused for multiple users or applications of the same information. In order for this to be optimally useful, the cache should be seeded with a reasonable set of information items before startup, and also the decision unit must dynamically choose items to cache based on incoming requests and some minimal available external information. The PURSUIT system has in place a mechanism for intentionally pre-caching large popular downloads, as well as a mechanism for cooperative dynamic caching, which are discussed next.

2.2.4 Existing Cache Replacement Strategies

Understanding how an individual cache functions, and how it chooses which items to store or replace, informs our decision-making about how caches should communicate with each other. There are many excellent references for the current state of research on cache replacement strategies, including [20], [21], [22], and [26].

The two common cache replacement strategies proposed for existing publish/subscribe architectures (namely, PURSUIT and CCN) are LRU and LFU. Each of LRU and LFU has many variations in the literature. Each has its limitations; the variants are designed to address those.

LRU, or Least Recently Used replacement, maintains age bits on the data and when a new item needs to be added to a full cache, it replaces the least recently used information item. The main limitation of LRU is that it does not account for the volume of queries to a target. Sites visited regularly but not frequently fall out of the cache, while one-off requests stay in the cache for much longer than they should (for example, a user downloading a large quantity of personal files would cause the cache to be flooded with useless entries).

There are many LRU variants, including the following few. These show us the variety of potential lower-level caching strategies which might be used as components of a larger caching system. In LRU/2, the penultimate access times are used to run LRU, and the item with the least recent second-to-last access time is replaced. LRU/2 extends to an arbitrary number of accesses before the most recent, and this collection of strategies is called LRU/K, each replacing the kth most recent information item. In SLRU (segmented LRU), new items are placed on a probationary cache ordered from most to least recently accessed; if an item is accessed again while still in this cache, then it is placed in the

protected cache. When necessary, the least recently accessed in the protected cache is downgraded to the most recently accessed slot of the probationary queue. Items are purged when they are at the least recently accessed end of the probationary queue and a new item is added to the cache. FBR incorporates elements of LFU (below) by separating the cache into new, middle, and old sections and, when an item is accessed and it is not in the new (most recently used) section, then incrementing a reference count for the item. When replacement needs to happen, the block in the old section with the smallest reference count is purged.

LFU, or Least Frequently Used replacement, maintains a counter on the data of number of access times for each item. When a new item needs to be added to a full cache, in replaces the item with the fewest accesses. One main limitations of LFU are the additional complexity (logarithmic) to add or remove an entry or to reorder entries in the heap. The other main limitation of LFU is that recent pages are not given priority. Stale entries which received many hits some time ago are kept in the cache over those which received a few hits very recently.

Adaptive Replacement caching uses a LRU cache and a LFU cache on the same data, each with a ghost list tracking recently evicted entries. Cache misses which are in a ghost entry will increase the size of the ghost entry's associated cache and evict an item from the other cache.

Depending on the needs of the network, different caching strategies may be more or less successful. According to Rajahalme et al., internet packets have an optimal 15 minute time to live (TTL) in a cache – packets which are highly requested are likely to have already been requested again in that time span within the service domain, and packets which have not been requested again in that time span are likely candidates for replacement. This temporal locality property indicates that most queries previously submitted will be submitted

again in a short time frame. Using this expectation, a few modifications to these caching strategies emerge as potential improvements on LRU or LFU performance. [26]

2.2.5 Dynamic Caching

While most simple caches replace items based on simple frequency or recency criteria, cost-based criteria can be a beneficial addition to a more advanced cache[22].

With a basic LRU cache, each request inserts at the front of the queue (cache hits moved up from further back in the queue, cache misses inserted at the front of the queue). With a basic LFU cache, a separate list is kept of the candidate items for the queue and the number of requests for them. When one of these candidate items is requested as many times as the least-requested item that is cached, it gets inserted into the cache.

According to Ozcan, when miss costs are not necessarily uniform, cost-based caching methods are very effective. These methods frequently extend LRU such that items which have a high cost to cache misses are less likely to be replaced than low cost items. Cost-aware caching can take into account latency, power, or bandwidth consumed by cache misses as well as penalize the replacement of selected items or categories.

Miss costs in packet caching could increase based on a long delay to the source server, downtime for the source server or authoritative rendezvous node, or size of files or packets. [22]

A system for cooperative dynamic caching for PURSUIT has already been developed [12]. The decision-making algorithm for this system intelligently precaches information items in chosen caches around the network based on

request patterns and expected user interests in various sections of the network.

In most cases, dynamic caching would be a threat to the security of the information. Security and naming of information items in publish/subscribe systems is the focus of much research. The common belief, embodied in Zooko's Triangle, is that names can be no more than two of: human-understandable, secure, and unique. While self-authenticating names, created from a hash of the document's contents, name, and/or other identifying information, are commonly held to be a standard of the publish/subscribe systems, they are not human-understandable. However, this may be a benefit, as network architects cannot guarantee any structure of human-understandable naming that will remain constant in the long term; if we insist on that structure with the introduction of our new publish/subscribe paradigm, it is likely to need to change, which will upset internet functionality [34].

Another important problem that our content-oriented caching model can help with is network management. Network management is another important problem in the current internet structure, as only local domains have human maintainers. Problems in the global internet, such as widespread congestion, cycles in routing tables which cross ISPs, and non-localized related failures can rarely be detected, let alone be fixed. Yet keeping vast quantities of data for network management, and having it be available non-locally in case of network failure, is a great challenge for network architects [35], but they might be able to improve their strategies with the cooperative caching model able to store more data about the network in the area.

Distributed Hash Tables (DHTs) are efficient structures for cataloging and retrieving information by name, with requirements similar to the caching requirements of publish/subscribe internet. DHTs are typically used for a hash table instead of as a routing guide across the internet, but many of the lessons

learned by the engineers working on DHTs are relevant to publish/subscribe caching strategies. Data management in a DHT is discussed in [5]. Churn, and the associated problems caused in DHTs by hosts frequently joining and leaving the network, is discussed in [28], with particular attention to methods for re-balancing cached copies to accommodate cache servers which enter and leave the network periodically instead of expecting constant uptime.

2.2.6 Collaborative Caching Systems

A collaborative caching system is a system in which multiple caches cooperate to improve cache hit rates. Elements of a collaborative caching system include storage management and coordination overhead. Caches in a collabrative caching system cannot make independent decisions, since these decisions affect the other caches in the system. The most important caching decisions are what to store and what to throw out when the cache is full and also how to optimize and organize across the collaborating caches.

The two main types of organization for collaborative caching are hierarchical caching and distributed caching. These have been covered in subsubsection 2.2.5, but both are briefly defined below.

Hierarchical caching is a cache system which has a tree structure. All user requests are first sent to the local cache (leaf node of the hierarchical cache tree). Any requests that cannot be answered by that cache are passed to the parent node's cache. This continues until either there is a cache hit or until the cache at the root node has a cache miss. While this system provides a high rate of cache hits, latency for unusual requests is very long as each subsequent cache miss passes the request up the tree.

Distributed caching is a cache system in which every cache communicates with

every other cache in the system whenever it fields a request that it cannot answer. While this system provides a high rate of cache hits, worst-case latency is determined by the slowest intra-cache RTT (round trip time), as the requestor cache must wait for every reply before informing the user of a hit or miss[30].

This chapter has provided an overview of critical terms and concepts, within the fields of caching and publish/subscribe or information centric networking. The next chapter will go into more detail about specific systems and projects.

3 Related Work

The problem space that this thesis operates within is that of caching in information centric and publish/subscribe networking systems. This section will briefly describe some of the research projects in similar problem spaces. First, the specifics of individual information centric or publish/subscribe networking systems will be described, since those represent the general space of options and alternatives for working within content-oriented networking systems. Next is selected other projects which are working on other pieces of the information centric networking space. The next major component of my project is caching, so some overviews of works on caching are discussed in Section 3.3, both in TCP/IP and in information centric systems. The last section covers miscellaneous projects, such as distributed hash tables, routing, and network management. While these miscellaneous projects affect this research less, they are still useful works for any network engineering project.

3.1 Specific Content-Oriented and Publish/Subscribe Networking Systems

3.1.1 PSIRP, PURSUIT, and COSMON

PSIRP, the predecessor of PURSUIT, uses a similar rendezvous architecture[26]. This report documents the design for registering and locating network objects within a flat space for identifiers, the rendezvous service model. The rendezvous service model involves object owners registering (publishing) their objects in any rendezvous node; then these objects are requested by users and found through lookup in the Rendezvous Overlay Structure, wherein rendezvous nodes have overlay links to other rendezvous nodes (which later become scopes in

PURSUIT). Global lookup prioritizes locality, attempting to route traffic between nearby edge networks without adding load to the high-traffic core of the system.

Deliverable D2.2: Conceptual Architecture

The full conceptual architecture behind the PURSUIT project is described in [39]. One key architectural element is the idea of scopes. A scope is an information item in the PURSUIT system which consists of a list of information items that a certain rendezvous node (administrative node) knows how to find in the system. Since scopes themselves are information items, a scope can include other scopes in its list. Scopes get advertised to other rendezvous nodes just like any other named data item.

PURSUIT is both a content-oriented networking system and a publish/subscribe networking system. Addressing is done on information items (including scopes), and nodes can subscribe to published information items, which will update the subscribers' copies of the information items whenever they change. Publishing an information item allows nodes to begin subscribing to it; advertising a published information item tells other rendezvous nodes where to find it.

Network traffic in PURSUIT first uses Domain-local Rendezvous of traffic to and through the user's local rendezvous node. If the local rendezvous node knows the location of the sought information item, it handles the forwarding of the request. If not, then the traffic gets routed through the Global Rendezvous system.

Section 5.2 of [39] addresses the conceptual architecture behind the use of caching in PURSUIT. One component is Storage Replication, a mechanism to select information items to store in various nodes of the network to improve forwarding traffic. The implementation of this is covered in more detail in [12].

The other component to caching in PURSUIT is the In-Network Opportunistic Caching, which allows a local rendezvous node to cache information items as users request them. While [12] is a start, it is insufficient, as that work primarily deals with a high-level algorithm for where in the network should replicate storage of very popular items; it does not handle the opportunistic caching angle.

COntent-Switched MObile and access Networks (COSMON) Proposal

COSMON is a proposed project which is a spin-off of PURSUIT, focused on mobility concerns[19]. COSMON proposed to use a content-switching architecture to natively and seamlessly support the modern mobile users using in-network caching. Caching is a core component of any content-oriented architecture which attempts to support migratory users. In this proposal, their ideas for in-network caching are informed by the success of Content Distribution (or Delivery) Networks, commonly called CDNs, over the past decade by bringing content closer to the end users, reducing latency of delivery to the end user and minimizing traffic concentration near popular content providers.

3.1.2 CCN and NDN

A different take on content-centric networking was developed by the Palo Alto Research Center. While both CCN and NDN started from this project and share a code base, the two projects have recently been focusing on different ideas and improvements. For my purposes, these two projects can be considered one and the same.

A Reality Check for Content Centric Networking

While applauding the paradigm shift from host-centric to content-centric networking and interesting systemic designs, in this paper, the researchers

analyzed the hardware requirements for deploying a large-scale CCN system[23]. However, the practical impact of large-scale deployment of CCN would be insupportable by today's routing technology. Small-scale deployments are feasible, but more study must be done on the interconnection of multiple routers and experiments on them using realistic network traffic. Large-scale deployments will be more feasible in the future, once the router hardware has improved.

Named Data Networking (NDN) Project

The Named Data Networking Project (NDN) is identical in architecture to that of CCN, employing interest packets from the subscribers, and the triadic structure of the Forwarding Information Base, Pending Interest Table, and Content Store to handle forwarding of requests, retracing data packets back to the requester, and caching[44]. One key feature of this project is that it has security features built into the data, as each transmitted piece of data is signed, together with its name, ensuring a secure binding. Another key feature is the hierarchical naming scheme, which results in fast name lookup.

Routing Policies in Named Data Networking

In the current internet, the Border Gateway Protocol (BGP) routes traffic between domains based on policies rather than shortest path or lowest cost algorithms. Due to competing internet service providers, the internet is grouped into sets of routers which employ an internally consistent set of policies and which each are under a single administrative entity. These entities are called Autonomous Systems (AS).

With the future paradigm shift to content-addressed networking, there is likely to be a corresponding shift in the administrative organizations. While there is likely to be some form of groups of routers forming ASes, how they interact

with each other is yet to be determined. DiBenedetto et al. explore some possible ways that BGP and ASes might evolve come the paradigm shift to content-centric internet[9].

3.1.3 HTTP System

HTTP as the Narrow Waist of the Future Internet

Rather than a complete overhaul of the fundamental building blocks of the internet, some researchers are extending the existing internet protocols, with a focus on HTTP, to provide support for publish/subscribe systems within the existing internet structure[25]. In this model, HTTP takes over the middle layers of the current communication protocol (the "narrow waist") while allowing users to subscribe to content natively. Given the present existence of a massive and ubiquitous HTTP infrastructure, as well as the recent surge in HTTP traffic, modifications to HTTP for publish/subscribe capability would require minimal infrastructure changes compared to other potential publish/subscribe architectures. Since this approach involves few changes from the way the modern internet works, any caching systems would be those that can work in TCP/IP.

3.2 Other Projects in Content-Oriented Networking Systems

Naming in Content-Oriented Architectures

The three characteristics of an entity in naming schemes in content-oriented architectures are: its name in the scheme, its real-world identity, and its public key[13]. The various proposed naming schemes do not connect all three of these

together. Human-readable naming schemes connect the name to the real-world identity of the entity but require an external authority to match those to the public key of the entity, for cryptographic security. An alternate naming scheme is to use self-certifying, cryptographically secure names. This binds the name of the entity to the public key, but an external authority is needed to connect this to the real-world identity. While some schemes attempt to bind an entity's name to both its real-world identity and its key, this leaves a security hole without augmentation by third parties for the name to real-world identity binding.

VoCCN: Voice-over Content-Centric Networks

The problems surrounding streaming media in publish/subscribe internet, including the necessity and ability to drop slow packets while using Voice-over CCN for streaming audio is shown in [16]. Live streaming of audio, including the flexibility to ignore dropped packets and continue transmitting and receiving, is one key type of information request which content-centric networks need to handle.

3.3 Caching

Analysis of Web Caching Architectures: Hierarchical and Distributed Caching

The caching problems in the current internet are surveyed in [30], comparing the strengths and weaknesses of hierarchical versus distributed caching systems. In these cases, the caching systems compared are structured, fully hierarchical and fully distributed cache systems.

In fully hierarchical caching, with three levels of caching, in the case of a cache miss at the local cache, the request is forwarded to the state level cache. In the case of another cache miss, the request is forwarded to the regional level cache.

In the case of another cache miss, the reuqest is forwarded to the national cache server. At this point, if another cache miss takes place, the request is forwarded to the original source of the item.

In fully distributed caching, all caches are predetermined (not ad-hoc) and mutually peers, but there is no central organization (such as in a DHT or the Chord system). Each cache fills its buffer opportunistically based on requests from users in its local region, so there is no collaborative assignment of data to each cache. In the case of a cache miss at the local cache, the local cache queries every single other peer. It must wait to hear from all peer caches before forwarding the request to the original source, if all caches report a cache miss.

Hierarchical caching has lower connection times than distributed caching. Distributed caching, however, has lower transmission times than hierarchical caching. This is hypothesized to be because most of the network traffic is on the less-congested edges of the network. Since total latency is the sum of the connection and transmission times, both hierarchical and distributed caching systems can be subject to high latency.

Hierarchical caching architectures reduce the expected network distance to retrieve a file, effectively implements multicast at the application level, and has minimal administrative overhead compared to distributed caching. However, hierarchical caching systems also have problems with hot spots and high peaks of load. Distributed caching architectures perform very well in highly-interconnected areas. In addition, distributed caching also has smaller disk requirements than top-level caches in hierarchical caching; also, it shares load well for the system as a whole, as opposed to the hot spots that can be created by a hierarchical caching system. However, distributed caching systems cannot be effectively modified to be large-scale, due to large network distances and bandwidth and administrative overhead.

Caching Techniques for Large Scale Web Search Engines

While cache requests for large scale web search engines (Google, Yahoo!, Bing, etc.) and cache requests for objects a user is interested in are different problems, they exhibit similar characteristics. Both are complex caching problems which need to efficiently access data from a very large possibility space of requests.

Many different cache filling, replacement, and eviction mechanisms, as well as metrics for evaluating them, are discussed in this paper[22]. Some of these are summarized in subsubsection 2.2.5. Also discussed is cache freshness, or mechanisms for ensuring that the items stored in a cache are still correct. The primary focus is on various cost-based algorithms. These cost-aware mechanisms are extended to cost-aware caching policies.

On Caching Search Engine Query Results

One major caching problem in the current internet is the problem of caching search engine query results, as popular queries can be requested frequently by different users and recomputing the results every time would be wasteful of computing resources[20]. From analyses of popular search engines, patterns of use of internet computing resources can be seen. There is a strong locality of queries, as approximately one third of search queries are duplicates (from other users or even from the same user). In their study, 1,639 out of 927,010 queries followed within 100 queries after an identical query. Additionally, 14,075 queries followed within 1,000 queries after an identical query, and in 68,618 instances, the time between identical queries was less than 10,000 queries. Even a single, non-cooperative, small cache that holds only the previous 10,000 queries (using LRU replacement) would have a hit rate greater than 7%.

3.3.1 Caching in Content-Oriented and Publish/Subscribe Networking Systems

Caching in content-based publish/subscribe systems

Performance evaluations for certain caching mechanisms in publish/subscribe internet technology are described in [36], focusing on planned prior caching using an intelligent decision-making network-wide program to place caches rather than opportunistic caching. While information delivery for active subscribers at the time of a publish operation is guaranteed, subscribers can also request information that was published prior to the subscription event. In these cases, caching may be utilized to increase performance.

The two caching policies discussed are basic caching, in which every candidate node along the network may choose to opportunistically cache an information item which it receives, and leaf caching, in which only leaf brokers can cache requests when they have multiple users subscribing to the same item.

3.3.2 Content Distribution Networks

A Content Propagation Metric for Efficient Content Distribution

Content Distribution Networks are the closest thing the current internet has to a widely available caching system of popular data. This paper proposes a novel metric by which to judge how effective CDNs are at information management in distributing content to competing consumers[24]. The three sources of bandwidth in the network are (a) the original content publisher/distributor, (b) other clients/peers, and (c) in-network caches. Any networking protocol needs to consider all three sources of information and network traffic, due to the highly varying needs of different kinds of traffic (for instance, streaming video

from official YouTube servers and ad-hoc peer-to-peer protocols, which have greatly differing bandwidth requirements).

The researchers found that a hybrid, peer-assisted architecture for content distribution maximizes system-wide available bandwidth by routing some traffic along the edges of the network.

4 Design

The goal of this project was to design, implement, and test a new strategy for utilizing the principles of caching more efficiently in a information centric information networking system. To this end, I created a superdomain system to utilize multiple regional caches to increase hit rates while not being penalized by the greatly increased lag caused by fully distributive or fully hierarchical. This project is designed specifically within the context of the PURSUIT architecture, but the superdomain system is designed to be a module which could benefit a variety of networking systems.

The details of the design are explained in this section. First is a description of many of my goals for designing a caching system, many of which informed the superdomain system. Next is a discussion of caching's role in a system with many mobile users. Native support of user mobility is a strength of information centric caching systems. Following that, I briefly describe the existing research being done for PURSUIT for intelligent pre-caching. The rest of the chapter is specifically on the superdomain system design.

4.1 Caching Goals

When designing a caching system for publish/subscribe internet technology, the goals for the system fall into the following categories:

- Availability of data
- Network performance (high throughput, high speed, low latency)
- Reliability and resiliency to failures
- Security (confidentiality, availability, integrity of data)

- Differentiation of systemic failures from intentional deletion

I will now elaborate on each category, as well at the bearing each has on this project.

4.1.1 Availability of data

The main goal for the availability of data is for any published data to be able to be found in the network and transmitted back to the requester, including in cases where the original source has unexpectedly gone offline. An additional specific concern is for network management information to be available for a network which has gone down in order for the managing networks and ISPs to diagnose the problem.

Caching greatly helps availability of data by providing additional distributed sources of the information items. Intelligent caching systems can utilize network management data to predict users' information wishes and cache those information items in advance of the users' requests. If this happens, and the original source experiences downtime, the users may experience uninterrupted service from the cached copies.

4.1.2 Network performance

One of the incentives to switch to publish/subscribe internetworking is an expectation of improved performance, eliminating much of the excess traffic, duplicate traffic for popular items, and suboptimal routing pathing. Among the performance-oriented goals for publish/subscribe internet, we desire that popular data is not taxing on the network, that there is a good response rate for items new to the network (and especially that soon-to-be-released popular items are ready for the demand by being pre-cached), that cache servers avoid wasting

much cache space with spam, trash, or unnecessary copies, and that cache servers vary their strategies statistically (much like ethernet cool-off times vary statistically by machine so not all machines will attempt to re-send simultaneously after a network blockage). In addition, an important overarching goal is for this sytem to be reasonably self-organizing without constant communication.

4.1.3 Reliability and resilience to failures

Reliability and resilience to failures is an important quality of publish/subscribe internet technology. Related goals include that data is still available when the authoritative source is down, and that original sources have enough information about the external published copies of their data to have a "second," a trusted authoritative source if the original goes down or for the remaining copies to be able to elect one.

Although reliability and resilience to failures can be addressed by other means, a good caching system does provide better reliability and resilience to failures. If a popular information item is cached, then if the publishing server is under heavy load or experiencing downtime, the data remains available.

4.1.4 Security

Publish/subscribe internet technology has many security concerns in attempting to address the privacy and integrity of information streaming to every subscriber. Some, but not all, of these concerns apply to storing the data in caches midway between the source and the subscriber. Various sources can be differentiated by the signature of the source.

Caches would only hamper the network if filled with junk and malicious

information. Ideally, cache servers can differentiate, blacklist, and/or delete malicious data and that no more than a small (set) percentage of a cache comes from any one source. This is rarely, if ever, done in current caching systems.

Most publish/subscribe systems utilize the fact that modern processors are fast enough to do packet-level signatures or encryption. This allows for source certification and increased security. Packet-level encryption and data signatures are an expectation of the system as a whole, including the data caches and useful for the caches to enact self-defense mechanisms.

However, opportunistic caching inherently fills caches based on the demands of subscribers, so limiting caching based on publishers is a bit backwards. Security is an ongoing, extremely complex problem, and this merits more investigation in the future.

4.1.5 Differentiate systemic failures from intentional deletion

In order to differentiate systemic failures from intentional deletions, all copies must get deleted when the authoritative source deletes yet at the same time the item must remain available while facing censorship or other forms of malicious takedown.

This is a difficult problem and beyond the scope of this project, but it makes for interesting future work.

4.1.6 Miscellaneous

Other goals for creating publish/subscribe networking which do not fit into the above categories include the following. These are beyond the scope of this project, but may be considered for future work.

First, how the cache servers handle data must be tailored to the type of data (i.e. streaming media, aggregating feed, or published text unlikely to change). For instance, certain types of data are expected to change frequently, such as stock market prices, news headlines, and twitter feeds; other types of data are expected to never be modified, such as major software releases, pictures, and ebooks. Data expected to change frequently could benefit from push notifications which alert subscribers to new modifications and updates, but this is unnecessary for static data.

In addition, the timeliness of the delivery of data is always critical in discussions of network architectures. Users demand low latency and high throughput. Intelligent caching could potentially provide fast and seamless service to users. Since users' perception of network latency and throughput is the most important qualitative properties of a network, this is always kept in mind while designing networking systems.

Also, cache servers should be able to pre-warm caches around the world tracking the sleep/work/play cycle of a time zone or to use time-based subscriptions to an item (either duration or refresh), and for the cache servers to have low power usage and cheap energy usage (for instance, Google dynamically moves computations around the country or around the world depending on the price of energy at that particular moment in each location).

Another way to improve caching will be to define a set of strategies for individual pieces of data, for them to choose if it gets cached outside of the original source, in a few places (either locally or widely distributed), or in as many places as possible. Combinations of the strategies are also possible. Different types of data will necessitate the use of different choices of how widespread the caching strategies should be. For instance, private data which will not need to be publicly accessed (and should not be) need not be stored

externally to the original publisher unless the user still requires access to the data if the original source unexpectedly crashes. Alternatively, most websites, videos, and other data items which are available to anyone should be able to be cached widely in case of the vast demand of popularity.

Additionally, updates to an already-released item should be atomic, i.e. updates appear in none or all copies. In some publish/subscribe systems, this is a mere option, a design choice that can be made. However, since the advent of publish/subscribe is tightly coupled with the networking architectural support for our multitudinous mobile devices, the atomicity of updates becomes important. However, atomicity is a great challenge, and require design trade-offs in space and time.

As an example, a user on their smartphone crosses wireless domain boundaries frequently, including while downloading an item. If multiple different sets of bits are available under the same unique identifier name (some with an update, some without), then the user may receive bits from conflicting versions of the information item when crossing wireless domain boundaries due to the different routers having conflicting local sources, some updated and some not. Therefore, atomic updates are important for supporting the mobile users.

4.2 Caching and Mobility

One of the major goals of any publish/subscribe system is for the system to seamlessly support mobile and wireless devices. Mobile devices are everywhere in today's networking environment and becoming more so; any new networking paradigm must handle them specifically.

Caching could become a great boon to publish/subscribe systems when a client is mobile. Requested data gets cached locally in response to any mobile device's

request. This easily handles the common case of multiple requests for the same data from the local network because the cached copy can be delivered quickly and efficiently. If the mobile device changes internet connectivity and domain before receiving the data, the data can no longer be delivered along its specified path. However, the cached copy remains in the local cache.

Using standard internet protocols, the mobile device would then request the same data item again, this time from its new local network. However, since there is a cached copy in an adjacent network (the previous local network for the mobile device), smart caching strategies can utilize the proximity of the other local networks in the network topology and can find this just-cached copy to send it to the mobile device in the new domain, while retaining efficiency very similar to if the mobile device had not switched domains. This would improve the standards of mobile computing greatly. The seamless switching of wireless domains for a mobile user is one of the strengths of my superdomain system explained in subsection 4.4.

Current caching systems do not have any built-in means to find data cached in adjacent domains (either to correlate data from a larger set of users to pre-cache likely requests, or to seamlessly support a mobile user roaming across a set of proximal domains). This is partially due to limitations placed by rivalries between ISPs.

What makes this work unique and particularly valuable is the fact that data can be found and delivered to the requestor from adjacent domains, without any additional effort on the mobile requestor's part.

4.3 Improving Caching and Storage Strategies

The purpose of my work in the field of caching and network architecture, specifically within the bounds of the publish/subscribe systems, is to improve caching and storage systems in PURSUIT system, although most internet architectures could benefit by the addition of my modular system. The goal of my system is to better support mobile users roaming across multiple domains (which are, by necessity, adjacent) and to provide a higher cache hit ratio than a single cache without adding the latency problems inherent in traditional distributed caching or hierarchical caching systems[30].

In the CCN architecture, intelligent precaching decision-making algorithms are difficult due to the inherent nature of the widely distributed small caches. The caches in CCN also have little to no cooperation. Additionally, in CCN, data is only sent in response to an interest, so in order to precache, the network must have drones at many edge points in the network which demand data based on the predictions of what to precache. This is not an effective way to use caching to benefit information-centric networking in a thoughtful manner which allows for planning and pre-caching of items which are expected to be requested. Therefore, I focus on the large caching facilities, especially in the PURSUIT system, but that could be used in other publish/subscribe systems or with the current networking standard of TCP/IP.

4.4 Superdomain Caching Design

4.4.1 Superdomains

"Superdomain" is a term I use to describe my hybrid system between distributed and hierarchical caching systems. I use the term superdomain to indicate a

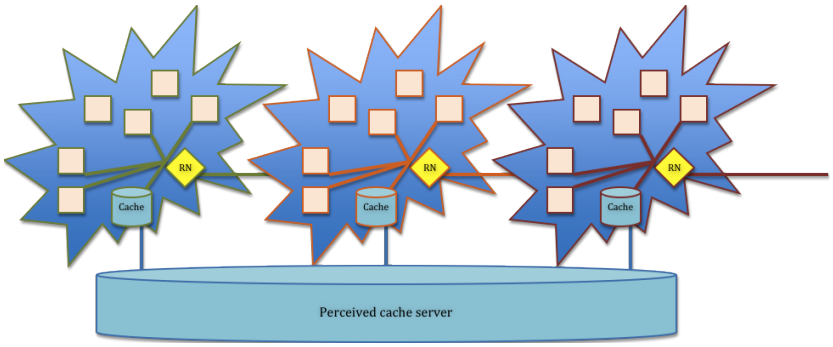


Figure 1: *These three domains shown comprise a superdomain. Each rendezvous node (RN) is connected to each user and a cache. From the perspective of the users and RN, the three separate caches in the different domains appear as one coherent, large cache server.*

small, geographically linked, set of domains, each of which includes a cache server. The cache servers within the superdomain behave as a distributed caching system, but since they are guaranteed to be all within a small geographical region (such as the Cambridge/Somerville area), there is only a short latency waiting for replies from other caches.

Opportunistic caching of information items in PURSUIT manifests as storage of a copy of the items in a local machine which receives all cacheable local requests for those items. Opportunistic in-network caching greatly increases performance for popular or recently requested items by redirecting the network request to the locally stored copy, eliminating the propagation time to and from the (non-local) authoritative source for the item as well as by avoiding the duplication of network traffic along that route.

Which items should be opportunistically cached are chosen by the decision-making units presented by Flegkas[12]. These cache servers then subscribe to the information item and republish it within their own scope.

The architecture where caches are subscribed to their cached information items allows any updates to the information items to be passed down to all copies, as

each publisher keeps a list of its subscribers and any update would get pushed down the subscription tree. This also minimizes duplication of updates to a common information item (such as software patches) and automates distribution.

In order to keep each cache server manageably sized but to provide to the users with the illusion of a far larger cache while maintaining low cache check latency, we divide the world into superdomains, each of which has its own distributed cache system, treated by the users as a single cache. These superdomains contain an intermediate number of individual domains[1], all under the control of a single ISP. Individual domains each need to have at least one cache server, even if it is only a placeholder. (Note that a house or apartment is viewed as one or a few nodes in a larger domain, not a domain in itself. All subscribers to a certain ISP in a city could define a domain, for example.) Each user treats all cached items in any cache in her superdomain as locally cached, see Figure 1.

Many modern networking systems, including information centric ones, aim to support widespread user mobility. Superdomain caching systems support this aim for two main reasons. First, mobile users moving between various domains must be supported by complex handovers between hotspots. Having fixed server locations which are richly interconnected with low latency eases these transitions. Second, while users are mobile, their mobility is limited and comparatively slow. Users walking or biking would take a long time to move from one superdomain to another; even users travelling via rapid transit (i.e. trains, planes, automobiles) would spend at least several minutes at a time within the realm of a given superdomain. In addition, recent research into cell phone use patterns shows that users, despite mobility, are often stationary. Fully 60% of a given user's mobile device activity is associated with the top two

[1]For example, a superdomain could consist of 3 individual domains or 10 individual domains (although this can vary based on the needs for a given situation as well as performance optimizations), allowing utilization of multiple cache servers without adding too much latency, but exact conditions will be discussed under experimentation.

cell phone towers that device is associated with[14].

As an example, if a non-local information item is requested from a cache by a user, this will cause a subscription event at the cache server. Then, even if the user moves between hotspots or domains inside the superdomain, the user's request will still be routed to that cache server's copy of the item.

We wish to distribute the burden of in-network caching within the bounds of the PURSUIT publish/subscribe system, yet give users and applications control over which information items are appropriate to cache. Towards this, the users can request an information item in two different ways. One way is (without caching) from the information item's home scope, i.e. /scope1/scope3/item1. The other way, with caching, would be to request the item directly from its domain's local cache's scope by prefixing a reserved scope in which only a local cache server is allowed to publish/advertise, i.e. /cache/scope1/scope3/item1. This base scope /cache would be reserved much like the current 10.x.x.x in IPv4 internet addressing. (Additionally, the base scope /local would be reserved for requests that should only be filled from within the local domain. These can be either served or rejected quickly, as the local rendezvous node only need check anything that was published to it by one of its users. The scope /local is required to be checked for any information item before the scope /cache.)

In the case of a request under the scope /cache, the local rendezvous node will direct the request to its local cache. If the requested object is not stored locally in the cache, the cache will use flood routing to check for the requested object in all peer caches in its superdomain. The caches must communicate with each other. This communication can happen via any networking means, including long-term publication/subscription relationships or an out-of-band, separate network. As long as superdomains contain a small number of domains which are geographically and topologically close to each other, this flood routing request

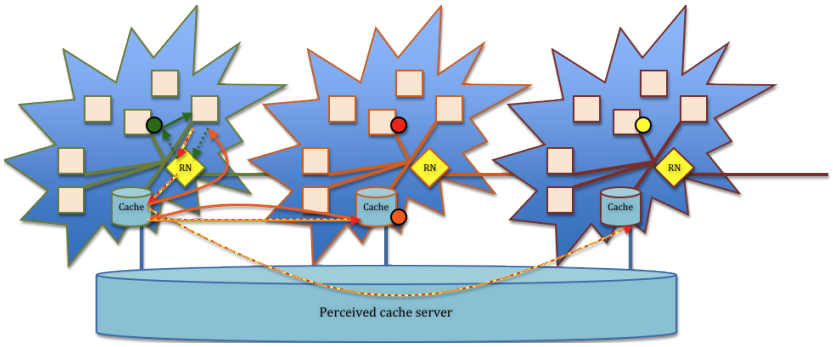


Figure 2: *The user requests 4 information items, the locations of which are shown by the color-coded circles. The requests are shown by arrows of the same color, with dotted lines for the outgoing requests and solid lines for the return path of the information items. The user first queries the /local scope for the items, the rendezvous node directs the request for the green object and it is delivered to the user with no intervention from the cache. None of the other three objects are found in the local domain's cache, so that cache uses flood routing to query its two peer caches for the objects (via the inter-cache communication system). A peer cache returns the information about the orange object to the cache, which sends it to the user. The other two objects are not found, and must be requested under the original rendezvous ID from the local rendezvous node for the request to be forwarded to the external network.*

will add only a small amount of latency to the system. If any peer caches in the superdomain have a copy of the requested information item, it is sent to the subscriber.

If no peer caches in the superdomain have a copy, the cache can either respond to the user with a failure notice (and then the user can rerequest the object using the original rendezvous ID, without the /cache prefix) or the cache can send a subscription request to its local rendezvous node for the same item, stripping the /cache prefix off of the rendezvous ID. The information item is delivered to the cache server and stored there, and then the cache server delivers the object, in turn, to the user.

In summary, superdomain caching answers my goals for a caching system, within the space of tradeoffs. Data availability is greatly improved by the distributed duplication that caching provides. Superdomain data availability is

improved over simple local caching, also, since caches near each other won't duplicate items that each other has cached. The idea behind creating superdomains was to improve network performance within the ability of a user to perceive it, where the average user cannot detect sufficiently low delay (thus the slight additional time for checking geographically nearby caches will not hamper the user's perception of network speed and latency).

Reliability and resiliency to failures is not greatly changed by this system, although the additional duplication of data does help provide some resiliency against nodes publishing data which are prone to have significant downtime. While security and differentiation of systemic failures from intentional deletion are not particularly affected by the superdomain caching system, they both provide significant and interesting avenues for future investigation.

The details of the design, from goals, through the choice of the information centric system and other current research in caching for the PURSUIT system, to the exact details of my system, have been described in this section. I built a simulation to test my concepts, to compare a system with only local opportunistic caching to my system with cooperative superdomain caching.

5 Experimentation

This chapter will describe the specific experiments that I ran to test my system. The system is ostensibly designed to fit into the PURSUIT system; it is being examined through that perspective. The code, however, is run in a simulation of the PURSUIT system. By abstracting away the lowest-level details of the exact networking protocol, I was better able to simulate wide distribution by simulating different delays in delivery.

The first simplification inherent in my simulation is that of data transit times and lag times. While the simulation I built does not run on many machines across the world, such as in Planetlab, response times are highly dependent on simulated distance between nodes and somewhat dependent on randomized lag. Since distance between machines is the primary determining factor in response time, this is an acceptable approximation.

The second simplification is a modification of the PURSUIT system - the use of an external, oracle routing system for names of objects. The current version of the PURSUIT code has not fully resolved dynamic rendezvous for requests among multiple domains. This was a key incentive to use a simulation. While the PURSUIT system is unlikely to ever have a routing system equal to that of an oracle system, I expect further research into the PURSUIT rendevous system to yield a good rendezvous system not much worse than an all-knowing oracle system. Furthermore, I expect the added time cost of routing algorithms and heuristic pathing to be small compared to response times and well within the margin of error from my randomized additional lag cost.

The third simplification is having each "node" of the network represent a domain, complete with multiple users and a local caching/rendezvous node. That each domain has a local cache server is an acceptable assumption, as

domains without a cache server are unaffected by my system - they do not reap the benefits of the superdomain caching system, but their behavior is essentially unchanged. Also, a domain can be represented as a single node if internally-filled requests are accounted for. Requests which can be filled by a local publisher or cached copy are filled in response time determined solely by the randomized lag, giving times in the tens of milliseconds.

The fourth simplification is a modification of the PURSUIT system. One distinctive feature of PURSUIT, compared to other publish/subscribe systems, is that it has the notion of active, long-term "subscriptions." Updates to the subscribed data are automatically sent to all subscribers, without need for a separate request. However, in the case of updates to data, the update would not be cached anywhere, so this sheds no light on the question of whether superdomain caching improves the caching system within the system. While the caches would update previous publications of cached objects, this might not affect the speed of the update delivery to end users. Currently, my system does not properly handle update publications.

Next, I explain the simulation I created and the scenarios I designed to test the superdomain caching system compared to a local-network-only caching system. My results and analysis of the data follow after that.

5.1 Experiment Design

The design of the experiment to test my system has two key parts - the architecture of the ICN simulation and the scenarios crafted to test the system.

5.1.1 Simulation Architecture

At the core of the simulated system are two key concepts, the "named" information object or InfoItem, and the node which has a location and communicates with its neighboring nodes.

The system I built is a information centric architecture, where the primary addressing scheme is on the InfoItem. However, I allowed nodes (caches) to also have purely local addresses, for the express purpose of allowing neighbor nodes to query caches while under superdomain caching. Since these local addresses are only shared with sister caches with a direct, local connection, and the names can be reused in other superdomains, this still fulfills one fundamental goal of the content-oriented architectures, location independence. The names are merely a shared moniker for a particular local connection and are merely used for convenience. This is the only instance in which node names are used by any nodes other than the rendezvous nodes (RNs). The issue of RN forwarding via purely InfoItem names is beyond the scope of this simulation, but that should minimally affect the behavior of the caching in the network.

Each node object (representative of a whole domain) keeps track of the list of items in its local publications and cache, its node ID and that of its local RN, and its list of neighbor node IDs. The two outward-facing functions that a node object has, by which the rest of the network can interact with it, are `request` and `subscribe`. Nodes may only request subscriptions to published objects. `request` automatically generates a subscription request from a simulated end user in the domain of that node, based on the probability densities of requests in the current world object. If the RN can then have the user subscribed to its cached copy of the information item (or a neighbor cache's copy, when using superdomain caching), it will do so. Therefore, any updates to the item will get

to the user via the chained subscription links. If the RN does not find a cached copy to use, `subscribe` is called by the RN, since one of its nodes has a request which cannot be filled by its cache or, only in the case of superdomain caching being on, neighbor caches. This will subscribe the RN's local cache to the original published object and subscribe the user to the local cache's copy of the object.

In this setup, any two nodes can talk to each other, as this represents the rendezvous servers of two independent domains talking to each other. When a request is generated, the following node transactions take place in the simulation:

1. The system randomly decides which node (representing a domain) will make the request.
2. The chosen node generates which information item will be requested according to the popularity distribution of InfoItems.
3. The chosen node requests the named data from itself, adding a local request round-trip time (RTT) to the cumulative request time. This represents the requesting user querying the local rendezvous node.
4. If the node finds the requested data item in the local domain, either in the list of local publications or in the local domain cache, the item is found and the request completed. Control goes back to the simulator to make a new request.
5. If the node does not find the requested data item locally, then the node must find an external copy. If superdomain caching is in use, the other nodes in the superdomain are sent a request for that item, and a within-superdomain RTT is added to the cumulative request time. If any node within the superdomain has the item on its locally published list or

in its local cache, the information item is found and the request completed. Control goes back to the simulator to make a new request.

6. If superdomain caching is not in use, or fails to find a copy of the InfoItem within the superdomain, then the original publisher of the data must be queried. The publisher of the information is queried in a global routing table. This is an approximation, as a perfect oracle routing system is not possible, but should not have a great effect on the total response time of the data.

7. The node sends a request for the data to the original publisher, and a request RTT related to the heuristic distance between the two domains is added to the cumulative request time. The item has now been found and the request completed. Control goes back to the simulator to make a new request.

The world object keeps track of a few global settings (such as the number of requestable items in its universe, number of nodes in the world object, and the max cache size). It also contains pointers to all contained node objects and the array of probabilities for each possible request, determined by my implementation of Zipf's Law or modified Zipf's Law.

One of the important elements of the design of the experiments is the pattern of subscriptions. InfoItems' popularity was chosen by Zipf's Law and a modified Zipf's Law.

Zipf's Law states that, for many types of data from social studies on human behavior, for any item i_n that occurs with probability p, the next-most popular item i_{n+1} occurs approximately half as often, with probability $p/2$. Some studies have shown the application of Zipf's Law to human behavioral patterns for internet requests.[31] However, other studies show that a heavy-tailed

distribution are more applicable to human behavioral patterns for internet requests.[6]

In light of this, and since Zipf's Law original allows a single node's cache (of only 10 items) to cover more than 99% of requests in the simulations, I also used a variation on Zipf's Law for two experimental scenarios, in which clusters of four same-probability items are at every level of Zipf's Law, which functions as a rough approximation of a middle ground between Zipf's Law and a heavy-tailed power-law distribution.

5.1.2 Tested Scenarios

The topology of the simulated network in all scenarios is a linear connection of five superdomains. Each node represents a domain, and each superdomain consists of a few nodes (domains) which are geographically close together. This represents five geographical areas spread out linearly in the topology of the core internet connections. It is linear to provide a variation in response times. Response times within a superdomain system are expected to be short, due to geographical location similarity, with simulated response times ranging from 15ms to 30ms. Response times to farther superdomain systems increase, with communication between superdomain 0 and superdomain 4 possibly taking up to 1000ms. Compared to the scale of these response times, an individual node checking its own cache or list of published items is assumed to be approximately 0. This is a sufficiently general representation of internet topology to be a reasonable approximation. Furthermore, the primary interest of this work is finding what percentage of data requests can be served in user-imperceptible time, so the structure of the network outside of a node's superdomain does not affect this query, as any requests which must be served by nodes outside of the requester's superdomain all occur in user-perceptible time. (Research suggests

that data response times below 100ms are undetectable by users[43].)

Four separate experiements were run, with all parameters and setup the same except for the two independent variables and some initial, controlled, randomness in the exact topology set up. The two key independent variables were: the existence of cooperative superdomain caching or not (both cases have local network caching), and the implementation of Zipf's Law as stated versus as applied to small clusters of InfoItems with the same probability of being requested. The overall response time of the request was the measurement of primary interest. Requests were also tallied to see what percentage of requests were served by within the local domain, within the superdomain if superdomain cache cooperation was in use, and cache system misses.

All scenarios iterate over 1,000,000 requests, 10,000 possible requestable items, a maximum cache capacity of 10 items, 5 rendezvous/forwarding nodes, 25 total nodes (4 non-rendezvous nodes per RN, and every node has neighbors of all other nodes in its rendezvous cluster). In all scenarios, which node publishes each InfoItem is randomized during initialization.

The first scenario uses the original Zipf's Law and no superdomain caching. If Zipf's Law is a good measure of human behavior for internet requests, this is a simulation similar to how information-centric networks would work currently.

The second scenario uses the original Zipf's Law and superdomain caching. If Zipf's Law is a good measure of human behavior for internet requests, this is a simulation similar to how superdomain caching would affect information-centric networks.

The third scenario uses the modified Zipf's Law and no superdomain caching. If Zipf's Law applies more to groups of items in its popularity model, rather than individual items, this is a simulation similar to how information-centric

networks would work currently.

The fourth scenario uses the modified Zipf's Law and superdomain caching. If Zipf's Law applies more to groups of items in its popularity model, rather than individual items, this is a simulation similar to how superdomain caching would affect information-centric networks.

Some factors in real-world networking were not handled in the simulation, but these are factors where variations would leave the results of the analysis almost entirely unaffected; while they must be considered for any real-world instantiation of this software, they are tangential to the investigation at hand, that is, whether superdomain caching systems improve the network over caching that occurs in the local domain only. Among these factors are the size of various InfoItems, transient variations in network topology (links going down, for instance, which are realistically duplicated and resilient to transient failures), and light traffic versus peak traffic. Other factors in real-world networking were not handled in the simulation, as they are beyond the scope of this project and require separate investigation; the assumption I make is that these problems can be solved

5.2 Experiment Results

I analyzed my raw results by parsing the log files generated during the simulation and then using the `numpy` and `matplotlib.pyplot` libraries for python. The full analysis script can be found in Appendix B.

While seeing the figures, keep in mind that there are no horizontal or vertical axis bars - the thick, dark lines are densely packed data points. Near 0 ms communication time between the nodes, there are two sets of data very near each other, for requests served via the local cache and those served by a

neighbor.

With the original Zipf's Law and no superdomain caching, an immense 99.8% of requests are served from the local cache. Only 0.2% of requests must come from subscriptions to the original publisher of the InfoItem. This is incredibly unbalanced, and if this were an accurate representation of internet traffic, then no cooperative caching would be necessary. However, with a pure Zipf's Law interpretation of request probabilities, the 9 most popular request items make up the top 99.8% of traffic. Since I believe that it is important to cache more than the top 9 internet items, I do not believe this model is sufficient. See Figure 3.

With the original Zipf's Law and superdomain caching, 22.5% of requests are served from the local cache, but less than 0.01% of requests must come from subscriptions to the original publisher of the InfoItem. All other requests are handled by neighbor caches. In the response time graph, the response times for local requests and neighbor requests are very close, as evidenced by the dense packing of dots in a line just barely above the 0 ms marker (representing the requests served within the local domain) and the dense packing of dots in the area between the local request times and approximately 50 ms. As the inherent locality of superdomains guarantees that neighbor domains are geographically close to each other, such as domains halfway across a city from each other, response times for a neighbor request are very fast. See Figure 4.

With the modified Zipf's Law and no superdomain caching, 71.9% of requests are served from the local cache, and 28.1% must come from subscriptions to the original publisher of the InfoItem. Since there is such a wide distribution of the distance that responses must travel to fill requests, every portion of the possible response times is so densely packed with dots on the scatter plot that all sections of the possible response times are fully filled with dots. See Figure 5.

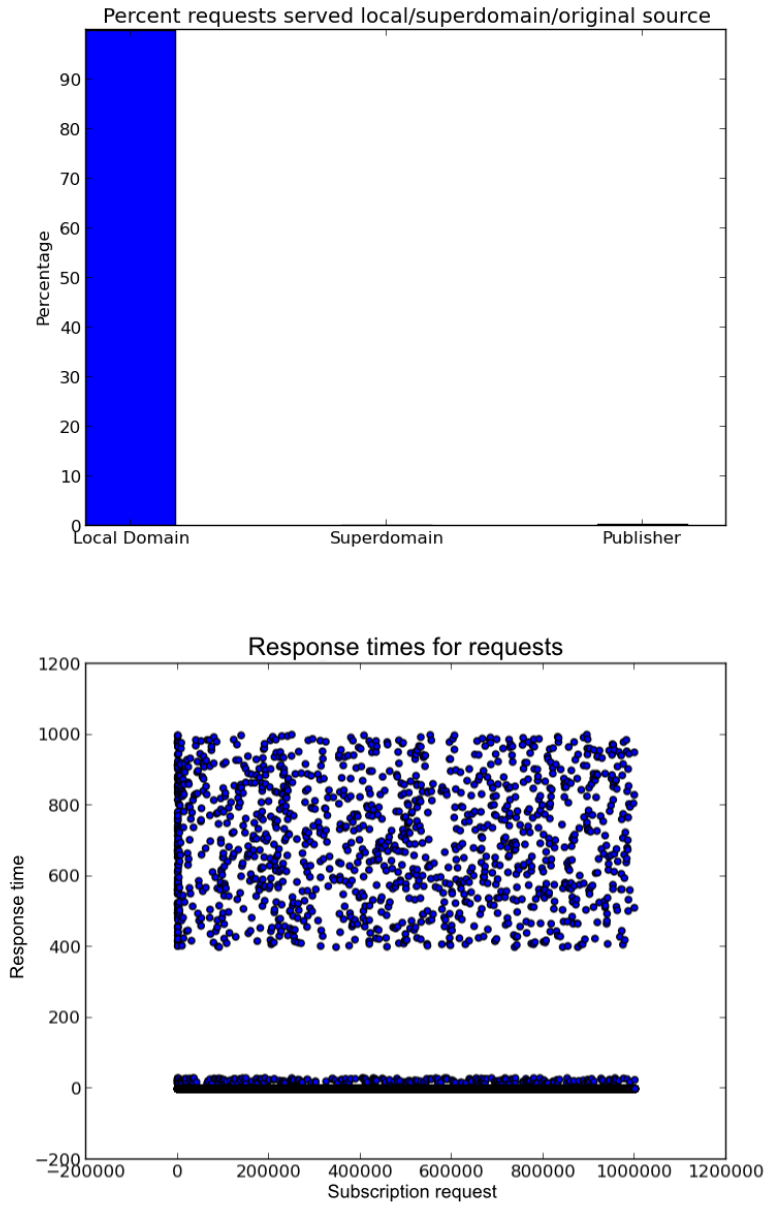
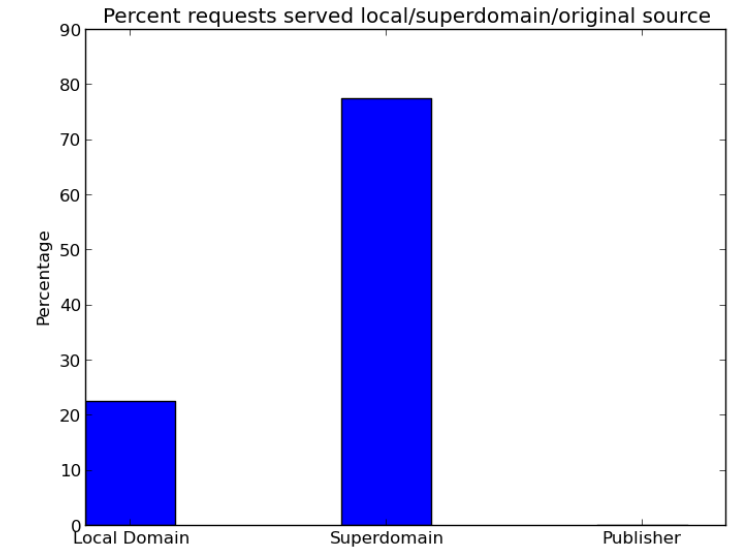


Figure 3: *Trial 1: Original Zipf's Law request likelihoods with no superdomain caching.*



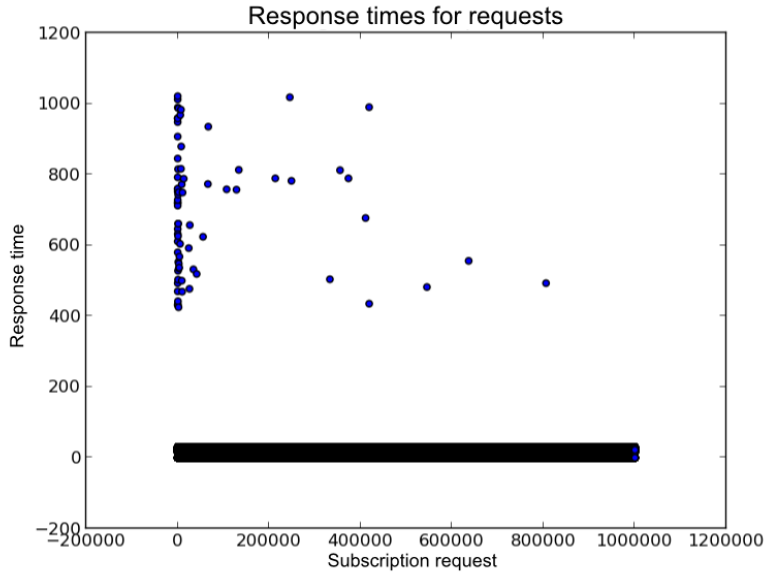
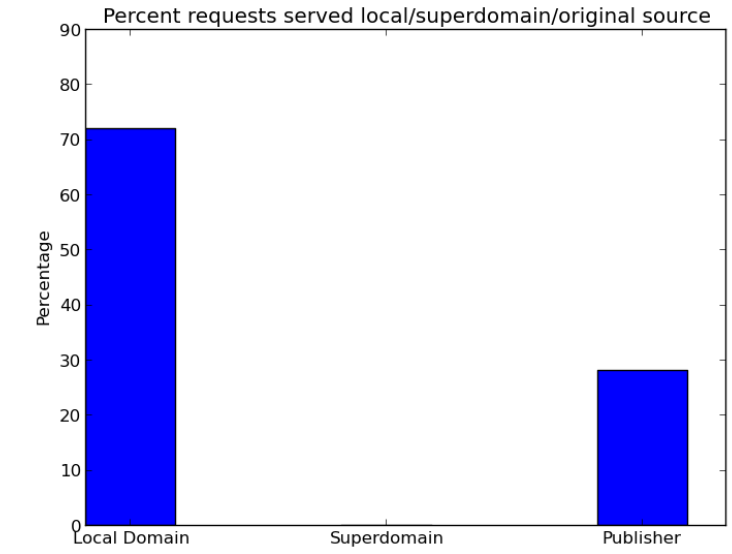


Figure 4: *Trial 2: Original Zipf's Law request likelihoods with superdomain caching.*



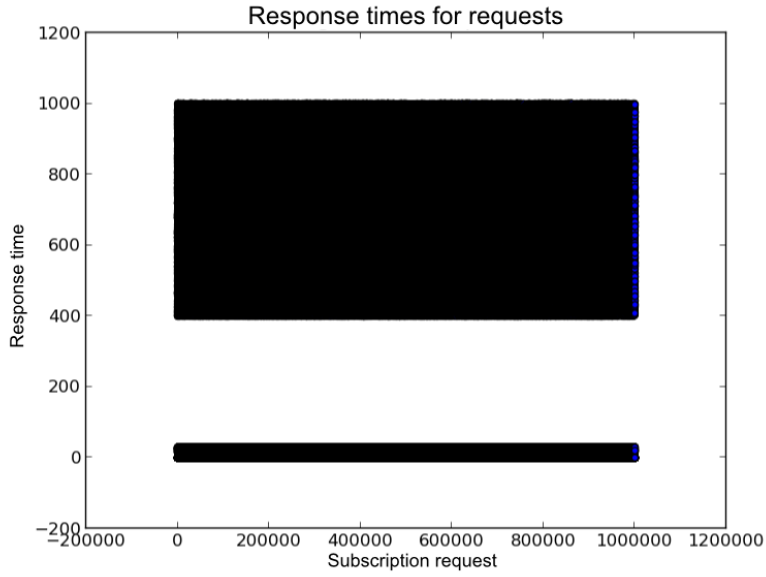
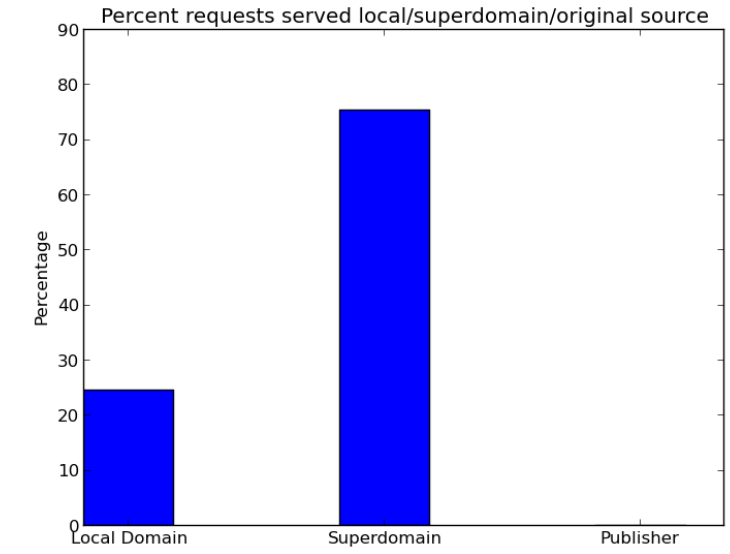


Figure 5: *Trial 3: Modified Zipf's Law request likelihoods with no superdomain caching.*



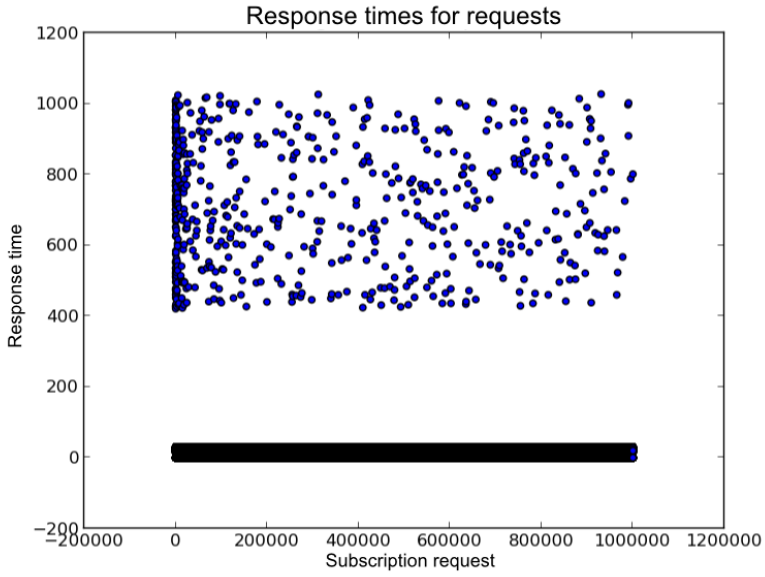


Figure 6: *Trial 4: Modified Zipf's Law request likelihoods with superdomain caching.*

With the modified Zipf's Law and superdomain caching, 24.6% of requests are served from the local cache. However, this is alleviated by the fact that only 0.06% of requests must come from subscriptions to the original publisher of the InfoItem. All other requests are handled by neighbor caches. In the response time graph, the response times for local requests and neighbor requests are very close, as evidenced by the dense packing of dots in a line just barely above the 0 ms marker (representing the requests served within the local domain) and the dense packing of dots in the area between the local request times and approximately 50 ms. As the inherent locality of superdomains guarantees that neighbor domains are geographically close to each other, such as domains halfway across a city from each other, response times for a neighbor request are very fast. See Figure 6.

The first analysis creates a graph of the percentage of requests served by each category of node: local node, a node within the superdomain system (if in use), and the node which originally published the InfoItem if a cached copy of the item is not found. The second analysis creates a scatter plot of the response times of all requests, over time.

The measurements of primary interest to me were the response time for requests and the percentage of requests that have response times below the threshold of human perception. One of the ultimate measures of the success of a networking architecture is that of qualitative user experience. Having a large percentage of requests delivered to the user below the threshold of user-perceived delay substantially affects the perceived quality of the networking architecture. Using superdomain caching systems decreases the cache hit percentage in the local cache but greatly increases the cache hit percentage in the geographical region with a response time below the threshold of what users will perceive as instantaneous delivery.

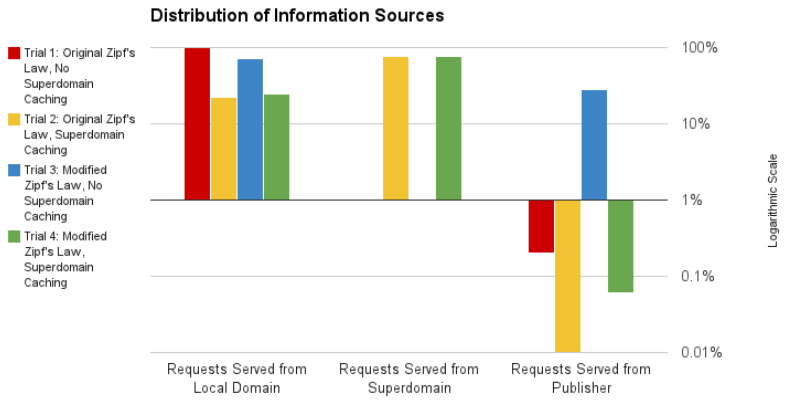


Figure 7: *Logarithmic scale comparison of the percentage of the subscription requests served locally, by superdomain, and by the original publisher.*

In order to demonstrate the differences between trials of what percentage of requests are served from where, I did a logarithmic comparison. Modifying Zipf's Law has a direct impact on how many requests must be served from the original publisher, as a greater variety of items is requested, but otherwise has minimal impact on the efficacy of the superdomain caching. Having superdomain caching causes fewer requests to be served locally but almost all requests to be served within the superdomain (a geographically close area), therefore response times are still short. With superdomain caching, a very small percentage of the time (less than 0.1% in both cases here) must the subscription request be sent all the way to the original publisher of the information, causing longer response times. See Figure 7.

In this chapter, the architecture of the ICN simulation code and the tested scenarios were explained. Then, I showed the raw results produced by my analysis script. These results will be discussed in the following section.

6 Conclusion

Towards the goal of improving information-centric systems, specifically the PURSUIT system, I investigated improvements to the associated caching system. I devised a superdomain caching system to provide the cache hit benefits of traditional cooperative caching schemes (hierarchical and distributed caching) without the large associated cost to lag and response time. I built a simulation to test this theory, and the results supported my key claim, which is that a superdomain caching system, a small-scale cooperative caching system strongly tied to nearby geography, will greatly increase cache hit rates within user-imperceptible response time.

Next, I return to the initial claim that such a system benefits internetworking systems by providing a greatly increased hit rate without sacrificing human-perceptible performance.

The two key statistics that the superdomain caching system affects are the effective cache hit rate and the overall latency cost of the additional system. As expected, the cache miss rate (the case in which the original source of the Information Item must receive a subscribe request) decreased substantially when using cooperative caching.

Traditional forms of cooperative caching, hierarchical and distributed caching, also both substantially decrease the cache miss rate. However, the problem with these traditional forms of cooperative caching is the greatly increased latency. The insight on which this work is based is that geography can be used advantageously to design locally distributed cooperative caching for significantly improved performance, as well as improved support for mobility. Furthermore, this can be achieved with little or no disruption to the current communications models in ICN. By coordinating cooperative caching with knowledge of the

geographical network layout, short response times to neighbor caches can be enforced (and must be, if this system is to function as intended).

On the graphs of response times for the randomly generated requests, in accordance with Zipf's Law or a modified heavy-tailed version of Zipf's Law, it is immediately apparent that the density of response times greater than 100ms is far greater in the cases with only local caching, as compared to those with superdomain caching. However, the density of response times around 50ms is far greater in the cases with superdomain caching, as compared to those with local-only caching.

Is this trade-off worthwhile? If, as I claim, users cannot perceive the overhead for checking neighbor caches, then it is. If not, then local caches should all optimize for the most popular items, rather than cooperating.

Research into the threshold of human-perceptible computer interactions shows that ideal latency is less than 100ms.[43] As long as the overhead from the superdomain system does not cross this threshold, the network of neighbor caches should be perceived by the user as if they were all the local cache; if this is the case, then a superdomain caching scheme increases the cache hit rate without sacrificing perceived latency. The overhead cost of using the superdomain system would vary in CCN versus in PURSUIT. The objects being requested and retrieved in PURSUIT are large, complete objects, while the objects being requested and retrieved in CCN are small fragments of a higher level objects. This will affect the overhead cost of using the superdomain system.

In all of the simulations, queries to neighbor caches did not cause the overall latency to become greater than 100ms in the case of a cache hit. In cases of a cache miss, the latency would already generally be greater than 100ms. However, the same logic applies - since the portion of the latency due to queries

to neighbor caches was under 100ms, it is not a perceptibly different lag than otherwise.

Having analyzed my data, I showed that the latency overhead of a superdomain caching system is lower than the ability of users to perceive it. Thus my system is vindicated as a system that provides a much greater cache hit rate without perceptibly sacrificing performance.

7 Future Work

In this final section, some possible avenues for future work are described. However, the problem space surrounding the next generation of networking technology is vast and these are only a few select steps which could better inform future research in internetworking. I will discuss some of these avenues for future research from the high level perspective of networking research towards the more specific systems that are relevant to the niche of this thesis, namely caching in publish/subscribe and information centric networks.

One major component of any internetworking technology is security. This thesis does not handle the complex problems which come with any adequate treatment of the problem of network security. However, information privacy continues to grow as a concern, as users put more personal information on the internet. Security frequently competes with transmission streamlining (for instance, through caching of Information Items) as a concern in network engineering design.

Other major components of networking design, such as reliability and resilience to failures, are important problems for ongoing research, not only in information centric networking and other futuristic networking research but also in the current IP-based internet. Improved caching, such as my proposed system, can make a great contribution to the reliability and resilience to failures of a network.

Two important problems for future research into publish/subscribe systems are both problems of differentiation. First, how do you differentiate systemic failures from intentional deletions? Second, how do you differentiate the different types of traffic on the internet? Publish/subscribe systems could greatly benefit from some added structure to inform the type of data (is it expected to change relatively frequently, such as a feed for a social media site,

or is it relatively static data, such as a photograph?), or the desired distribution pattern (is this private and/or streaming data not needing to be cached, such as a VoIP call, or is this expected to be popularly requested, such as a security update to a widespread piece of software?). This applies to caching in that the questions of which data items should be preserved in the cache or flushed from the cache. In the case of intentional deletion, the cached copies should be flushed as soon as possible. However, in the case of systemic failures, it becomes much more important that the cached copies be preserved, so that the data is still available during the failure.

In addition, within publish/subscribe systems, there may be ways of intelligently improving the operation of caches. For instance, any items which have expiration times (common in web pages and useful for information items which expect frequent or important modifications) can be forced to be immediately deleted from the cache at the expiration time, freeing up more space in the cache. However, some data items can utilize the stale version in the cache (even if not quite up to date, it is still useful and better than nothing) and other data items cannot utilize the stale version in the cache because a stale version is a waste.

Another way to improve caching will likely be to pre-warm the caches with data that is expected to be used, based on the day of the week and time zone. During the work day, business-related data items should be cached, while in the afternoon and evening data relevant to games, media, and other entertainment should be the focus of the local cache. During the nighttime hours, most of the processors serving the cache could be powered down to save on energy costs, while the cache stores backup data, network management data, and other similar items that must have a backup but need not take up storage space in any caches with more active users.

These, as well as many other problems in the space, must be researched as internetworking technology moves forward.

8 References

[1] M. Ahmed, F. Huici, A. Jahanpanah, "Enabling Dynamic Network Processing with ClickOS," SIGCOMM '12, August 13-17, 2012, Helsinki, Finland.

[2] M. Busari and C. Williamson, "ProWGen: A Synthetic Workload Generation Tool for Simulation Evaluation of Web Proxy Caches," *Computer Networks*, Vol. 38, No. 6, pp. 779-794, June 2002.

[3] G. Carofiglio, M. Gallo, L. Muscariello, "Bandwidth and Storage Sharing Performance in Information Centric Networking," ICN '11, August 19, 2011, Toronto, Ontario, Canada.

[4] A. Carzanigo, M. Papalini, A. L. Wolf, "Content-Based Publish/Subscribe Networking and Information-Centric Networking," ICN '11, August 19, 2011, Toronto, Ontario, Canada.

[5] J. Cates, "Robust and Efficient Data Management for a Distributed Hash Table," MEng thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 2003.

[6] M. E. Crovella and A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," In *IEEE/ACM Transactions on Networking*, Vol 5, Number 6, pages 835-846, December 1997.

[7] M. D'Ambrosio, C. Dannewitz, H. Karl, V. Vercellone, "MDHT: A Hierarchical Name Resolution Service for Information-centric Networks," ICN '11, August 19, 2011, Toronto, Ontario, Canada.

[8] F. Dabek, E. Brunskill, M. Frans Kaashoek, et al, "Building Peer-to-peer Systems With Chord, a Distributed Lookup Service," In: HOTOS '01 Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, Washington, DC, USA, 2001.

[9] S. DiBenedetto, C. Papadopoulous, D. Massey, "Routing Policies in Named Data Networking," ICN '11, August 19, 2011, Toronto, Ontario, Canada.

[10] R. Escriva, B. Wong, E. G. Sirer, "HyperDex: A Distributed, Searchable Key-Value Store," SIGCOMM '12, August 13-17, 2012, Helsinki, Finland.

[11] P. Eugster, P. Felber, R. Guerraoui, A.-M. Kermarrec, "The Many Faces of Publish/Subscribe," In: ACM Computing Surveys, Vol. 35, No. 2, June 2003, pp. 114-131.

[12] Flegkas, P. et al, "Storage Replication in Information-Centric Networking," 2011 IEEE ICON

[13] A. Ghodsi et al., “Naming in Content-Oriented Architectures,” SIGCOMM ICN ’11, August 19, 2011, Toronto, Ontario, Canada.

[14] S. Isaacman et al., ”Human Mobility Modeling at Metropolitan Scales,” MobiSys’12, June 25-29, 2012, Low Wood Bay, Lake District, UK.

[15] S. Iyer, R. Zhang, N. McKeown, “Routers with a Single Stage of Buffering,” In: SIGCOMM ’02, Pittsburgh, Pennsylvania, USA, 2002.

[16] V. Jacobson, D. K. Smetters, N. H. Briggs, et al, “VoCCN: Voice-over Content-Centric Networks,” In: ReArch ’09, December 1, 2009, Rome, Italy.

[17] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, “Networking Named Content,” In: CoNEXT ’09: Proceedings of the 5th international conference on Emerging networking experiments and technologies, pp. 1-12. ACM, New York, NY, USA, 2009.

[18] A. Karila, D. Lagutin, K. Visala, et al, “PURSUIT: Publish Subscribe Internet Technology Deliverable D2.1,” www.fp7-pursuit.eu, accessed 20 June 2011.

[19] A. Karila et al, “COntent-Switched MObile and access Networks,” proposal submitted for EU FP7 project status, January 17, 2012

[20] Markatos, Evangelos P., “On Caching Search Engine Query Results,” 5th International Web Caching and Content Delivery Workshop, May 2000.

[21] Megiddo and Modha, “ARC: A Self-Tuning, Low Overhead Replacement Cache,” in proceedings of the 2003 Conference on File and Storage Technologies, 2003.

[22] Ozcan, Rifat, “Caching Techniques for Large Scale Web Search Engines,” PhD dissertation, Bilkent University, September, 2011.

[23] D. Perino, M. Varvello, “A Reality Check for Content Centric Networking,” ICN ’11, August 19, 2011, Toronto, Ontario, Canada.

[24] R. S. Peterson, B. Wong, E. G. Sirer, “A Content Propagation Metric for Efficient Content Distribution,” SIGCOMM ’11.

[25] L. Popa, A. Ghodsi, I. Stoica, “HTTP as the Narrow Waist of the Future Internet,” Hotnets ’10, October 20-21, 2010, Monterey, CA, USA.

[26] Rajahalme, Sarela, Visala, Riihijarvi, “Inter-Domain Rendezvous Service Architecture PSIRP Technical Report #TR09-003,” 30 December 2009.

[27] S. Rhea, B.-G. Chun, J. Kubiatowicz, S. Shenker, “Fixing the Embarrassing Slowness of OpenDHT on PlanetLab,” In: Proceedings of the Second Workshop on Real, Large Distributed Systems (WORLDS 05), Berkeley, California, USA, 2005.

[28] S. Rhea, D. Geels, T. Roscoe, J. Kubiatowicz, "Handling Churn in a DHT," In: Proceedings of the USENIX Annual Technical Conference, June 2004.

[29] L. Rizzo, "pgmcc: a TCP-friendly single-rate multicast congestion control scheme," In: SIGCOMM '00, Stockholm, Sweden, 2000.

[30] P. Rodriguez, C. Spanner, E. W. Biersack, "Analysis of Web Caching Architectures: Hierarchical and Distributed Caching," In: IEEE/ACM Transactions of Networking, Vol. 9, No. 4, August 2001.

[31] L. Shi et al., "Modeling Web Objects' Popularity," Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou, 18-21 August 2005.

[32] L. Shi et al., "Popularity-based Selective Markov Model," Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, 2004 IEEE.

[33] A. Shieh, E. G. Sirer, F. B. Schneider, "NetQuery: A Knowledge Plane for Reasoning about Network Properties," SIGCOMM '11.

[34] D. Smetters, V. Jacobson, "Securing Network Content," In: *Architecture*, pp. 1-7. Retrieved from http://www.parc.com/content/attachments/securing-network-content-tr.pdf

[35] K. R. Sollins, "An Architecture for Network Management," In: ReArch '09, December 1, 2009, Rome, Italy.

[36] V. Sourlas, G. Paschos, P. Flegkas, L. Tassiulas, "Caching in content-based publish/subscribe systems," In: IEEE GLOBECOMM 2009 proceedings.

[37] I. Stoica, R. Morris, D. Karger, et al, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," In: SIGCOMM '01, San Diego, California, USA, 2001.

[38] S. Sundaresan, N. Magharei, N. Feamster, R. Teixeira, "Accelerating Last-Mile Web Performance with Popularity-Based Prefetching," SIGCOMM '12, August 13-17, 2012, Helsinki, Finland.

[39] D. Trossen, G. Parisis, K. Visala, et al, "PURSUIT: Publish Subscribe Internet Technology Deliverable D2.2," www.fp7-pursuit.eu, accessed 17 July 2011.

[40] J. M. Wang, B. Bensaou, "Progressive Caching in CCN," In: IEEE GLOBECOMM 2012 proceedings.

[41] X. Wang and K. Goseva-Popstojanova, "Modeling Web Request and Session Level Arrivals," 2009 IEEE International Conference on Advanced Information Networking and Applications.

[42] T. Wolf et al., "Choice as a Principle in Network Architecture," SIGCOMM '12, August 13-17, Helsinki, Finland.

[43] A. Ya-li Wong and M. Seltzer, "Operating System Support for Multi-User, Remote, Graphical Interaction," Proceedings of 2000 USENIX Annual Technical Conference, San Diego, California, USA, June 18-23, 2000.

[44] L. Zhang, D. Estrin, J. Burke, et al, "Named Data Networking (NDN) Project," pp. 1-26. Retrieved from www.named-data.net/ndn-proj.pdf

Appendix A: Simulation Code

```
import random
import math
import os

#These initial declarations change with every experiment run
CACHING = 'ON'
GLOBAL_ITER = 1000000
GLOBAL_LOG = 'thesis/output_caching_2'
GLOBAL_TIMES = 'thesis/time_caching_2'
GLOBAL_PUBS = {}

class World:
    def __init__(self):
        self.size_of_info_items = 1000
        self.num_requestable_items = 10000
        self.max_cache_size = 10
        self.request_array = list(range(self.num_requestable_items))

        #these two lines are for straight Zipf's Law
        #self.request_probs = [0.5**(n+1) for n in self.request_array]
        #self.request_probs[-1] *= 2 # to make probabilities sum to 1

        #these lines are for modified Zipf's Law, clusters of 4 items
        self.request_probs = [(0.5**((n/4)+1))/4.0
                              for n in self.request_array]
        self.request_probs[-1] *= 2
        self.request_probs[-2] *= 2
        self.request_probs[-3] *= 2
        self.request_probs[-4] *= 2

        self.request_cum_probs = [0] * self.num_requestable_items
        cumulative = 0
        for i in self.request_array:
            cumulative += self.request_probs[i]
            self.request_cum_probs[i] = cumulative
        f = open(GLOBAL_LOG, 'a')
        f.write("\n\n\nWorld initialization complete.\n")
        f.close()
    def initialize_network(self):
        self.num_rns = 5
        self.num_nodes = 25
        self.num_nodes_per_rn = self.num_nodes / self.num_rns
        self.list_of_node_ids = list(range(self.num_nodes +
```

```
                                        self.num_rns))
        self.list_of_nodes = []
        self.split = self.num_nodes / self.num_rns
        temp = -1
        for i in range(self.num_nodes):
            if i % self.num_nodes_per_rn == 0:
                self.list_of_nodes.append(RN(self, i, [], i))
                temp = i
            else:
                neighs = []
                for j in range(temp, temp + self.split):
                    neighs.append(j)
                self.list_of_nodes.append(Node(self, i,
                                               neighs, temp))
        for i in range(self.num_requestable_items):
            ran = random.randint(0, self.num_nodes - 1)
            GLOBAL_PUBS[i] = ran
            self.list_of_nodes[ran].pub_list.append(i)
        f = open(GLOBAL_LOG, 'a')
        f.write("\nNetwork initialization complete.\n")
        f.close()
    def choose_random_node_to_request(self):
        #needs to NOT be a RN
        ran = random.randint(0, self.num_nodes - 1)
        picked = self.list_of_nodes[ran]
        if picked.node_id == picked.local_rn:
            return self.choose_random_node_to_request()
        else:
            return picked
    def ping_time(self, n1, n2):
        #all in milliseconds
        #this will, sadly, have to be mostly hardcoded
        node1 = self.list_of_nodes[n1]
        node2 = self.list_of_nodes[n2]
        temp = (abs(node1.local_rn - node2.local_rn) + 1)
        if temp == 1:
            m1 = 15
            m2 = 30
        elif temp == 2:
            m1 = 50
            m2 = 75
        elif temp == 3:
            m1 = 100
            m2 = 150
        elif temp == 4:
            m1 = 200
```

```
            m2 = 400
        else:
            m1 = 400
            m2 = 1000
        r = random.randint(m1, m2)
        return r

#every node here is actually a caching node representing a small network
class Node:
    def __init__(self, world, node_id, neighbor_list, local_rn):
        self.pub_list = []
        self.cache_list = []
        self.node_id = node_id
        self.neighbor_list = neighbor_list
        self.local_rn = local_rn
    def choose_request(self, world):
        #helper function to choose which item you'll request,
        #according to Zipf's Law
        ran = random.random()
        req = -1
        for i in list(range(w.num_requestable_items)):
            if ran <= world.request_cum_probs[i]:
                req = i
                break
        self.log_request(world, req)
        return req
    def log_request(self, world, req):
        # helper function to notate the creation of a request
        f = open(GLOBAL_LOG, 'a')
        f.write("Node %(node)d is requesting InfoItem %(item)d\n"
                % {'node': self.node_id, 'item': req})
        f.close()
    def check_cache(self, world, request, origin):
        if (request in self.pub_list):
            if origin == self.node_id:
                f = open(GLOBAL_LOG, 'a')
                f.write("Self %d pub_list hit\n" % self.node_id)
                f.close()
                return 0
            else:
                f = open(GLOBAL_LOG, 'a')
                f.write("Neighbor %d pub_list hit\n" % self.node_id)
                f.close()
                return 0
        elif (request in self.cache_list):
            #since we're doing LRU caching, need to push to back of list
```

```
            #because it got used
            self.cache_list.remove(request)
            self.cache_list.append(request)
            if origin == self.node_id:
                f = open(GLOBAL_LOG, 'a')
                f.write("Self %d cache hit\n" % self.node_id)
                f.close()
            else:
                f = open(GLOBAL_LOG, 'a')
                f.write("Neighbor %d cache hit\n" % self.node_id)
                f.close()
            return 0
        else:
            #these are unnecessary but possible additions to the log
            #f = open(GLOBAL_LOG, 'a')
            #f.write("Node %d pub_list/cache miss\n" % self.node_id)
            #f.close()
            #print "cache miss at node " + str(self.node_id)
            return -1
    def request(self, world):
        #look up in own publications first, then own cache
        #then ask neighbor caches if CACHING == 'ON'
        #will eventually return a 0
        t = 0
        request = self.choose_request(world)
        if self.check_cache(world, request, self.node_id) >= 0:
            f2 = open(GLOBAL_TIMES, 'a')
            f2.write(str(0) + "\n")
            f2.close()
            return 0
        if CACHING == 'ON':
            t = world.ping_time(self.node_id, self.node_id)
            #this is functionally the same as self to neigh
            for neigh in self.neighbor_list:
                if world.list_of_nodes[neigh].check_cache(
                    world, request, self.node_id) >= 0:
                    f2 = open(GLOBAL_TIMES, 'a')
                    f2.write(str(t) + "\n")
                    f2.close()
                    return 0
        val = (world.list_of_nodes[self.local_rn]).forward(
            world, request, self.node_id, t)
        #sanity checks
        if val < 0:
            "request error 1: forward passed error"
            return val
```

```
        elif val !=request:
            "request error 2: val not request"
            return val
        else:
            # OPPORTUNISTIC CACHING!
            self.cache_list.append(val)
            if len(self.cache_list) > world.max_cache_size:
                self.cache_list = self.cache_list[1:]
                #using LRU
            return val
    def subscribe(self, world, request, origin):
        #no need for long-term subscriptions here
        #prior functions have already verified that the node
        #contains the request, but let's double check
        if GLOBAL_PUBS[request] != self.node_id:
            print "subscribe error 1: self = " + str(self.node_id) +
                "; request = " + str(request) + "; origin = " + str(origin)
            return -1
        if not (request in self.pub_list):
            print "subscribe error 2: self = " + str(self.node_id) +
                "; request = " + str(request) + "; origin = " + str(origin)
            return -1
        f = open(GLOBAL_LOG, 'a')
        f.write("Node %(sub)d subscribes to InfoItem %(item)d from" +
                " Publisher %(node)d\n"
                % {'sub': origin, 'item': request, 'node': self.node_id})
        f.close()
        return request
#since individual networks are represented by single "nodes,"
#the RN only ever needs to pass missed requests
class RN(Node):
    def forward(self, world, request, origin, time):
        #look up where the request is hosted in saved forwarding table
        #ignore the case where this is not known, irrelevant
        #pass on request to the publisher
        value = GLOBAL_PUBS[request]
        node = world.list_of_nodes[value]
        t = world.ping_time(origin, value) + time
        f2 = open(GLOBAL_TIMES, 'a')
        f2.write(str(t) + "\n")
        f2.close()
        return node.subscribe(world, request, origin)

w = World()
w.initialize_network()
for i in list(range(GLOBAL_ITER)): #should increase
```

```
node = w.choose_random_node_to_request()
if (node.request(w) < 0):
    print "ERROR"
```

Appendix B: Analysis Code

```
import numpy as np
import matplotlib.pyplot as plt

times_base = range(1000000)
times_caching = []
times_no_caching = []

f_t_caching = open('thesis/time_caching_2')
f_t_no_caching = open('thesis/time_no_caching_2')

for line_t_caching in f_t_caching:
    num = int(line_t_caching)
    times_caching.append(num)
for line_t_no_caching in f_t_no_caching:
    num = int(line_t_no_caching)
    times_no_caching.append(num)

f_t_caching.close()
f_t_no_caching.close()

total_queries = 1000000
self_hit_caching = 0
neigh_hit_caching = 0
miss_caching = 0
self_hit_no_caching = 0
miss_no_caching = 0

f_caching = open('thesis/output_caching_2')
f_no_caching = open('thesis/output_no_caching_2')

i = 0
for line_caching in f_caching:
    if "Self" in line_caching:
        self_hit_caching += 1
        i += 1
    elif "Neighbor" in line_caching:
        neigh_hit_caching += 1
        i += 1
    elif "Publisher" in line_caching:
        miss_caching += 1
        i += 1
    elif "cache hit" in line_caching:
        if times_caching[i] == 0:
```

```
            self_hit_caching += 1
        else:
            neigh_hit_caching += 1
        i += 1

for line_no_caching in f_no_caching:
    if "Self" in line_no_caching:
        self_hit_no_caching += 1
    elif "Publisher" in line_no_caching:
        miss_no_caching += 1
    elif "cache hit" in line_no_caching:
        self_hit_no_caching +=1

f_caching.close()
f_no_caching.close()

def barplt(self_hit, neigh_hit, miss, filename):
    heights = [0, 0, 0]
    heights[0] = ( self_hit * 100.0 ) / ( total_queries )
    heights[1] = ( neigh_hit * 100.0 ) / ( total_queries )
    heights[2] = ( miss * 100.0 ) / ( total_queries )
    print heights
    if not sum(heights) == 100:
        print "ERROR"
        print sum(heights)
    ind = np.arange(3)
    width = 0.35
    plt.bar(ind, heights, width)
    plt.title('Percent requests served local/superdomain/original source')
    plt.ylabel('Percentage')
    plt.xticks(ind+width/2., ('Local Domain', 'Superdomain', 'Publisher') )
    plt.yticks(np.arange(0,100,10))
    plt.savefig(filename)
    return

#then do scatter plot of time and ping time from the time file

def scatterplt(times_array, filename):
    plt.scatter(times_base, times_array)
    plt.title('Ping times for requests')
    plt.ylabel('Ping Time')
    plt.xlabel('Time')
    plt.savefig(filename)
    return

#now make the pretty graphs
```

```
#IMPORTANT - PYPLOT LIKES TO NOT CLEAR ITS MEMORY
#ONLY RUN ONE PLOT COMMAND AT A TIME
#barplt(self_hit_caching, neigh_hit_caching, miss_caching,
       #'thesisfigs/bar_caching_2')
#barplt(self_hit_no_caching, 0, miss_no_caching,
       #'thesisfigs/bar_no_caching_2')
#scatterplt(times_caching, 'thesisfigs/times_caching_2')
scatterplt(times_no_caching, 'thesisfigs/times_no_caching_2')
```