HELSINKI UNIVERSITY OF TECHNOLOGY
Faculty of Electronics, Communications and Automation
Department of Communications and Networking

Stefano Gagliardi

A Distributed Caching System in DTNs

Master's Thesis
Otaniemi, May 25, 2009

Supervisor: Professor Jörg Ott, Helsinki University of Technology
Instructor: Lic.Sc. Mikko Pitkänen, Helsinki University of Technology

HELSINKI UNIVERSITY OF TECHNOLOGY
Faculty of Electronics, Communications and Automation
Department of Communications and Networking

ABSTRACT OF THE MASTER'S THESIS

Author:	Stefano Gagliardi	
Title:	A Distributed Caching System in DTNs	
Date:	May 25, 2009	**Pages:** 12 + 71
Professorship:	Networking Technology	**Code:** S-38
Supervisor:	Professor Jörg Ott	
Instructor:	Lic.Sc. Mikko Pitkänen	

Many popular services in the Internet are based on a client-server architecture. This is not an optimal model in some cases. Especially, where the server becomes difficult to reach due to link or path failures or traffic overloadings. This Master's thesis presents a design, implementation and evaluation of a distributed caching system for optimization of content retrieval in Delay-Tolerant Networks (DTNs). The presented solution proposes pushing the content closer to the requesting clients, by caching it in alternative locations on the path between client and server. In this case, the clients are able to retrieve the content with lower latency and fewer network resource consumption. The approach is motivated by the decreasing price of storage in caches.

The first part of the thesis introduces the research problem on a general basis, and a literature overview is given to present the current research in the field of Web caching, proactive caching in Content Distribution Networks and caching in DTN environments. A design of the distributed caching system is then presented. Focus is to design a node that can participate in content caching. Theoretical approach is taken to motivate network topologies, routing protocols and distributions of the queries for simulations.

The experimental part of the thesis discusses the results of the simulations performed to evaluate the feasibility of the proposed solution. As a result, the presented distributed caching system is shown to increase the overall retrieval performance, as well as reduce the latency for obtaining the responses. In particular, caching policies that aggressively cache items always give the best results, however, with increasing storage costs. Moreover, the distribution of the queried items has a major impact on retrieval performance.

Keywords:	Caching, Delay-Tolerant Network (DTN), Simulation, Web, Content Distribution
Language:	English

HELSINKI UNIVERSITY OF TECHNOLOGY
Facoltà di ingegneria elettronica, delle telecomunicazioni e dell'automazione
Dipartimento di reti di telecomunicazioni

RIASSUNTO

Autore:	Stefano Gagliardi	
Titolo:	A Distributed Caching System in DTNs	
Data:	25 Maggio 2009	**Pagine:** 12 + 71
Indirizzo:	Reti di telecomunicazioni	**Codice:** S-38
Relatore:	Professor Jörg Ott	
Correlatore:	Lic.Sc. Mikko Pitkänen	

Il modello client-server su cui si basa la moderna Internet non è sempre un modello ottimale, ad esempio quando il server diventa difficilmente raggiungibile a causa di situazioni di guasto o sovraccaricamento. Questa tesi presenta la progettazione, implementazione e valutazione di un sistema di caching distribuito, per l'ottimizzazione dell'information retrieval in reti DTN. La soluzione proposta consiste nel trasportare i contenuti più vicino ai client, replicandoli in memorie cache distribuite. In questo modo il client può recuperare i contenuti richiesti con inferiore ritardo e più limitato utilizzo delle risorse di rete. Questa strategia è motivata dal decrescente costo delle memorie dati.

La prima parte di questa tesi é dedicata alla presentazione dei risultati delle ricerche nel campo della Web Caching, delle Content Distribution Networks e delle reti DTN. Segue una descrizione del sistema di caching distribuito, in particolare dei requisiti necessari per cui un nodo possa diventare un candidato a partecipare al processo di caching. Inoltre sono presentate le proprietà teoriche delle topologie di rete utilizzate, dei protocolli di routing, e delle distribuzioni delle query inoltrate dai clients.

La parte sperimentale di questa tesi consiste nella discussione dei risultati ottenuti tramite simulazioni effettuate allo scopo di valutare la fattibilità della soluzione proposta. I risultati mostrano che il sistema di caching distribuito contribuisce a migliorare le prestazioni generali, aumentando l'efficienza dell'information retrieval e diminuendo i ritardi necessari ad ottenere i dati richiesti. In particolare, strategie di caching aggressive portano sempre a migliori risultati, pur elevando i costi di memorizzazione delle informazioni.

Parole chiave:	Caching, Delay-Tolerant Network (DTN), Simulation, Web, Content Distribution
Lingua:	Inglese

.

Perillä on tuolla, edessämme jossain
Mennään, mutta ajetaan hiljempaa
Toivon, ettei matka loppuis ollenkaan
Tämä voi olla
koko elämämme ihanin päivä
Ajetaan hiljempaa
Toivon, että matka jatkuu, jatkuu vaan ...

Contents

List of Figures

Abbreviations and Acronyms

ACMS	Akamai Configuration Management System
ADU	Application Data Unit
CDN	Content Distribution Network
CP	Cache-Popular
CU	Cache-Unpopular
DNS	Domain Name System
DTN	Delay Tolerant Network
ExC	Extreme Caching
FC	First Contact
FM	Friendliness Metric
GUI	Graphic User Interface
ICP	Internet Cache Protocol
IP	Internet Protocol
KIC	K-copy Intelligent Caching
KRC	K-copy Random Caching
NC	No Caching
ONE	Opportunistic Networking Environment
PD	Pareto Distribution
PDF	Probability Density Function
QD	Quadratic Distribution
RIP	Routing Information Protocol
SaW	Spray-and-Wait
SRIP	Simplified RIP
TTL	Time to Live
UD	Uniform Distribution
URL	Uniform Resource Locator
WWW	World Wide Web

Chapter 1

Introduction

Today many popular services in the Internet are based on a client-server architecture. Whenever the Internet users decide to look for some content online (e.g., a simple text, webpage, image, video or other multimedia file), their web browsers act as a client and issue a query for a particular object. The query message is forwarded through the network towards the server, where the actual data is stored. The server receives the request and handles it, together with other requests from the other clients, and sends back a reply message (containing the requested item) to the requesting user. However, in some cases this model is not optimal; a single server located in a different geographical region than the users may become hard to reach within reasonable time. In fact, in some specific intervals of time frequency of queries for a specific content may create overloadings in the server, for example, the European basketball fans connecting simultaneously every morning to the NBA.com server in New York and checking the previous night's results. Moreover, link or path failures may create bottlenecks or even prevent the connections to the server and retrieving the contents.

Possible solutions to make content retrieval process more efficient have been suggested already since the early 1990s. This thesis add a contribution to the previous work. The goal is to optimize the content retrieval, and discuss the distribution of the contents to multiple locations throughout the network. In

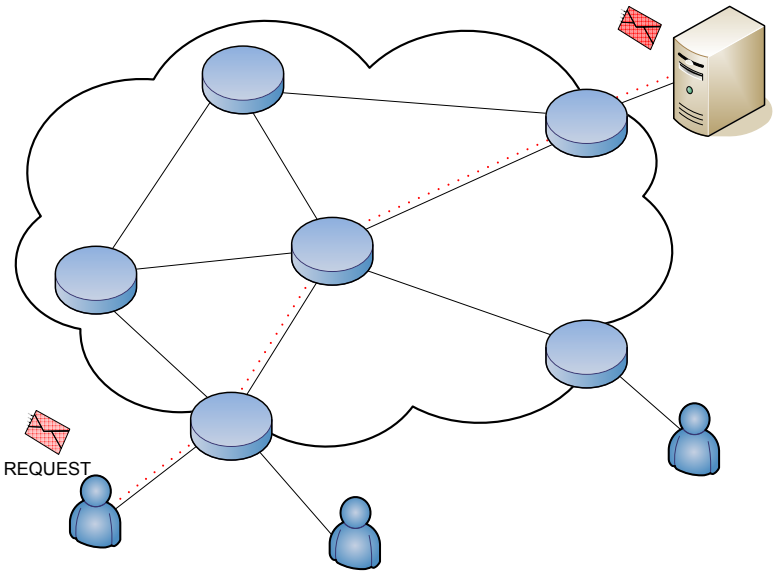


Figure 1.1: An example of a simple client-server content retrieval operation

particular, the intermediate nodes between the client and the server act as *caches*: they are allowed to temporarily store the contents of the messages that pass through them. This enables future requests for the same items to be handled by the caching nodes, which can directly send back a response to the clients without the need of invoking the server. This mechanism allows faster content retrieval process than without caching. Moreover, the mechanism leads to decreased resource consumption in terms of bandwidth by avoiding the transmission of data on the path from cache to server. Several caching schemes have been developed in order to reduce bandwidth usage and server load for the retrieval of Web documents [4]. As a result, Content Distribution Networks (CDNs) were born as a mechanism to support caching in the Internet. The CDNs comprise a high number of Web servers networked together and distributed over the Internet. The servers store copies of the Web contents. In this way, the contents are made available closer to the requesting clients. The clients do not have to know about caches, but can access the Internet as normally. This thesis presents a pull-based caching system that intelligently distributes the contents in the intermediate nodes to improve retrieval performance.

An illustrating example is shown in Figures 1.1 and 1.2. In the first case, the client queries the server for a particular data item and a response is sent back through three intermediate nodes. The second figure illustrates an optimized scenario, where a central cache node stores the requested item. Any query for the item from any other client arriving to the caching node can be handled without the need of redirecting the request to the server.

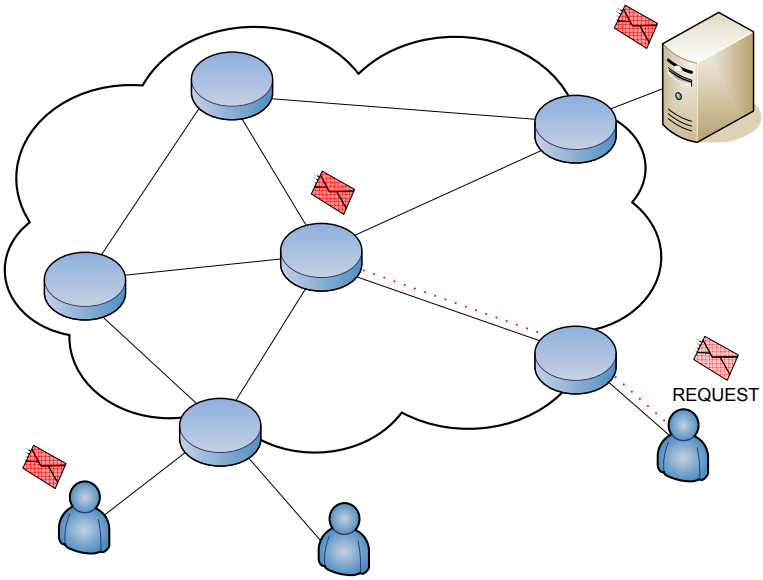


Figure 1.2: An optimized content retrieval operation from a cache node

It becomes a challenge to decide in which nodes it is feasible to cache the different items. An optimal distribution of items helps to cover the nodes with a high probability of finding responses from cache and leading to lower retrieval latency. Moreover, there should not be too many items cached around the network, since there is a cost for storage and it should not exceed the savings obtained by not letting the queries be processed directly by the server.

1.1 Problem Statement

This thesis addresses problems that arise from difficult access to resources, and an access infrastructure that does not optimize the access to shared items. A solution is proposed to push the content closer to the requesting client, by caching it in alternative locations. In this case, the requesting users are able to retrieve the content with lower latency and fewer network resource consumption. This approach is motivated by the decreasing price of storage, as well as high amount of bandwidth available for end users.

1.2 Goal and Scope of the Thesis

The main objective of this thesis is to evaluate performance improvement when introducing caching mechanisms in the intermediate nodes between client and server. The behavior of the network is analyzed, to investigate both the distribution and amount of cached items, as well as savings in the retrieval latency. The scope is limited to networks with static topology that transfer data in self-contained messages. This transfer model is adopted from Delay-Tolerant Networking (DTN) research [8]. The model allows to focus on caching at the application layer and enables to abstract out investigating the behavior of the network layer protocols such as IP.

1.3 Structure of the Thesis

Chapter 1 introduces distributed caching on a general basis. The chapter describes, defines and limits the research problem in this thesis and describes possible usage scenarios for discussed mechanisms.

Chapter 2 presents an overview about the related work that has been done in the past on this research topic. Web caching schemes in the Internet are presented in section 2.1, while the different push-based approaches for proactive caching in Content Distribution Networks (CDNs) are discussed

in section 2.2. Moreover, the caching schemes in the field of Delay-Tolerant Networks are presented in section 2.3.

Chapter 3 presents a distributed caching system; its design is carefully explained together with the requirements for a node to participate in content caching. In addition, different types of caching policies are introduced and compared. Furthermore, the following sections of chapter 3 present the theoretical properties of the used topologies, the routing protocols, and the query distributions.

Simulations are used to evaluate efficiency of the proposed caching system and the results are presented in chapter 4. The simulation environment is briefly introduced in section 4.1. The simulations are used to investigate the behavior of the system in different topologies, for several routing protocols and for different item distributions.

Finally, major findings and conclusions are drawn and explained in chapter 5 in order to have an overview of the results. Suggestions for future work and possible improvements to the presented system are summarized in chapter 6.

Chapter 2

Related Work

Caching is a wide research topic, and several caching schemes have been proposed in the literature. This chapter presents an overview on the work done in the past in this research area. In particular, Web caching systems and Content Distribution Networks (CDNs) are analyzed. Moreover, research on caching in Ad-hoc and Delay-Tolerant Networks (DTN) are summarized.

2.1 Web Caching Systems

In many types of network, including the World Wide Web (WWW), the exponentially growing amount of data transferred often leads to overloadings. For this reason, the need to minimize the user latencies and reduce the overall network traffic has been a major motivation to implement caching schemes in the Web [4],[28] starting from the early 1990s.

Harvest cache [6] was one of the first systems to be developed. It is a hierarchical mechanism based on the Internet Cache Protocol (ICP) [30]. Whenever a request is received for a missing object, the node forwards the request itself to his parent nodes, and so on via a remote procedure call, until the object is found, and retrieved from the node with a lower latency.

Cachemesh [29] is a distributed caching system built on top of the Harvest.

Its improvement to the precursor arises from the use of co-operative cache placement and cache routing mechanisms. In fact, nodes contain cache routing tables so that each of them can be addressed as the caching target of a subset of the resources to cache. The subset of the resources can then be retrieved through the routing tables themselves. In this way latency and search overheads can be dramatically reduced.

Alternatively, in Summary Cache [9] a compact summary of the cache directory of every other network caching element is kept in memory by each node. In case of requests for something locally missing, the query can be redirected properly to the relevant node - or to the server if there are no entries for that object.

Adaptive Web Caching [15] is a more complex caching system composed of a tight mesh of overlapping multicast groups. For each new request, the node checks its own cache and then the cache of all the other group members. If the look-up is not successful, the request is forwarded to another multicast group.

Access-driven Cache [31] aims to find the most prevalent access patterns to Web data, so that Web resources are divided into clusters, and nodes which frequently access them are grouped together. When the resource is not found the query is sent to the other members of the same information group.

Gwertzman et al. have proposed Geographical Push Caching [10], where data is dynamically mirrored on a geographical basis, e.g., cached items are kept close to the clients requesting them. Like Adaptive Caching, this system is capable to launch cache commands that may cross administrative boundaries. While there are many different Web caching schemes, with different architectures, deployment options, design techniques, the common problem is often the inter-cache communication, e.g., how to advertise efficiently the caching/removing actions to the other nodes [3] as well as the cache consistency [11].

2.2 Content Distribution Networks

In the modern Internet, with the proliferation of new Web servers all around the world, as well as the increase of the storage capabilities and the bandwidth resources, Content Distribution Networks (CDNs) were born. CDNs comprise a high number of Web servers networked together and widely distributed over the Internet. The servers cooperate transparently to the end user to distribute the Web content the user requested. The content is copied from the original target server to other servers, so that when a query is issued, the look-up mechanism automatically redirects it to the most appropriate location. BitTorrent [21] can be considered a CDN application that uses a peer-to-peer overlay network on top of the Internet for content delivery.

Akamai [26] is one of the biggest and most successful commercial CDNs with over 15'000 servers around the internet. To achieve network diversity and proximity to users, the servers are distributed in data centers and points of presence of all of the major Internet and communication carriers. These servers store the replicas of contents of Akamai Company's customers websites. Akamai also has its own DNS system to ensure fast delivery of the requested content by resolving the host name of the URL for the requested content to the IP address of the Akamai replica server that delivers the desired content to the user most quickly. The choice of the replica server to redirect to is made after extensive network and server measurements by the Akamai Configuration Management System (ACMS) [23], which is efficient, scalable and resistant to failures. In this way it is possible to modify internal services' configuration information with the certainty that the information is propagated to the replica servers within reasonable time. The system is quorum-based: whenever a publisher wants to transmit a new version of a configuration file, it contacts an ACMS storage point, and the submission is not acknowledged until the resource has been re-distributed to a majority of other storage points that have to *agree* on the submission. Once an agreement is reached, the data is made ready to be offered for distribution to clients.

This is done in order to provide a highly available and fault-tolerant management of configuration updates.

2.3 Caching in Ad-Hoc and Delay-Tolerant Networks

Caching becomes even more appealing in the case of Delay Tolerant Networks (DTNs) [16] where the end-to-end path from client to server may not exist at any given time, and the client has to retrieve the needed resources in some other way. The DTN model makes the communication possible when IP communication fails [8], by using *bundles* to carry application data in semantically self-contained application data units (ADUs). Basically, in DTNs the content retrieval works under the store-carry-and-forward paradigm, where messages may remain stored in a DTN router's buffer for a certain amount of time, and several replicated copies may be spread across several nodes. Effects of adding redundancy improved content retrieval performance in [19]; however, adding too much can lead to congestion and block the retrieval of less-popular resources, leading to fewer responses being delivered. The observation that resources remain stored in the buffers suggests to leverage the buffer storage by allowing the nodes to actively understand the resources and process the related requests at a higher application layer [20]. In fact, while a message waits in the transmission buffer, other requests for the same message can be issued, and if the node has an *intelligent* view of what he owns in his queue, it could respond to the query, just as a normal cache would do.

Pitkänen et al. [17] have also proposed a searching scheme in unreliable networking environments, with the result that popular contents will likely be found while items from the long tail only by chance. Moreover, the distance acts as a decisive factor for the success of the content retrieval. These characteristics lead to a need of implementing a caching scheme which can exploit both the topological information and the item's popularity as a metric.

DTN caching schemes suffer from security problems because the traditional

security mechanisms are not always applicable in scenarios without the assumption of end-to-end reliability [2]. Moreover, *fairness* has to be assumed for the users, because the whole mechanism works thanks to the donation of the intermediate node's resources for the system's use. If there are malicious or selfish nodes, the overall performance will degrade. A method to fight this problem is proposed in [24].

Seligman et al. [22] have proposed a distributed caching system to share the load in the network and relieve the server from over-exploitation. They propose a concept of *custody transfer* - the use of hop-by-hop reliability for enhancing end-to-end reliability and clearing retransmission buffers at senders as fast as possible. In simple words, whenever necessary, especially in the case of congestions, the aim is to try to prevent further traffic from flowing even when some outgoing connections are available. After rejecting the hypothesis of discarding messages, while still using it as a last choice, the proposed solution is to make the intermediate nodes act as *custodians*. Their task is to simply accept custody for messages, store them in the storage system, retrieve the messages at appropriate times, and keep track of where stored messages have been placed. A custodian attempts to avoid discarding any message for which it has taken custody, and if it gets congested it will stop accepting custody for subsequent messages. By doing so, the other nodes are aware that the custodian has made a promise of keeping the message, so they can freely discard or refuse the message itself, knowing that in case of a need they can easily retrieve it at anytime from the custodian. The challenge is then to determine which messages to store, and in which nodes: the message selection can be based on time, size, priority, while the node selection focuses on selecting alternative custodians near to the congested node (within a k-hop radius, using expanding ring search). The proposed approach uses migration cost metric that considers both the weighted summation of the normalized storage cost (available storage space) and transmission cost (latency, bandwidth, up/down schedules). Moreover, message fragmentation [18] could be used to take advantage of multiple buffers, where each fragment can be handled by different custodians individually.

Chuah et al. [7],[32] have proposed a caching scheme called *K-copy Intelligent Caching*. It is an enhancement of the *KRC, K-copy Random Caching*, where the item is simply cached in k random nodes. *KIC* implements the concept of *Friendliness Metric (FM)* which is calculated as a function of the observation interval and the number of unique nodes observed. The item is cached in the node which has the maximum *FM* in the path, and the action is repeated k times for different paths.

Cao et al. [5] have implemented a cooperative caching mechanism for Ad-hoc networks to save bandwidth and power as well as reduce delays by making nodes act as request-forwarding routers. Multiple nodes share and coordinate cached data to improve content delivery performance. Nodes pursue to cache different data items than their neighbors: it is then possible that nodes need to access some data from their neighbors instead of accessing it locally. Cao et al. propose two different approaches: caching the path to the data or caching the data. In the *CachePath* case, when nodes forward data, they keep the information (hop distance) between themselves and the source, but also between themselves and the destination. If successively new requests for the same data item arrive, the node knows whether the data source or the caching node is closer and forwards the request properly. Caching the path results in a reduction of the needed bandwidth and energy because of a need of fewer hops. However, the caching node may move or delete the data item, hence the cached path might be unreliable. The node then caches the data path only if the caching node is very close. Closeness is defined as a function of node's distance to the data source, distance to the caching node, route stability, and data update rate. On the other hand, in *CacheData* case, if the node finds out that the data is frequently accessed, it caches the data instead of caching the path. A careful usage is required, because of the storage space requirements. Cache consistency is based on time-to-live mechanism: if TTL expires, the node removes the data (or the path) from its cache. However, if a fresh copy passes by, the node updates the TTL field and eventually also the data item itself if it has been modified by the sender. The result is that if data size is small *CacheData* is optimal (no need of large

storage space), while if TTL is large *CachePath* is more convenient (as path information in static networks maintains reliable for a long time). *CachePath* performs better than *CacheData* when the size of available cache is small or the update rate is low. *CacheData* performs better in the opposite situation.

Chapter 3

Distributed Caching System for DTNs

In DTN environments, there is no centralized entity which has a control over the entire network and is able to have a complete overview of the network at any given time. The overview would be beneficial for deciding whether to cache or not a certain message in a certain node and for maintaining an optimal distribution of items around the network. Other schemes need to be developed; in these schemes every node doesn't know a priori information about the rest of the network, but can only exchange information with the surrounding nodes.

3.1 Design of the Caching System

3.1.1 Caching Signalling Mechanism

In this thesis, we implement a signalling mechanism, where each node broadcasts (at regular intervals) the caching information to its neighbors. It sends a list of the items it has cached, as well as another list called *Popularity*. This list is a registration for each different item of the number of times the item has passed by the node. In this way, each node can build a frequently

updated table containing the cached items in the node and its neighbors as well as the popularity of the items. These tables cover the sub-region of the network containing the node and its neighbors.

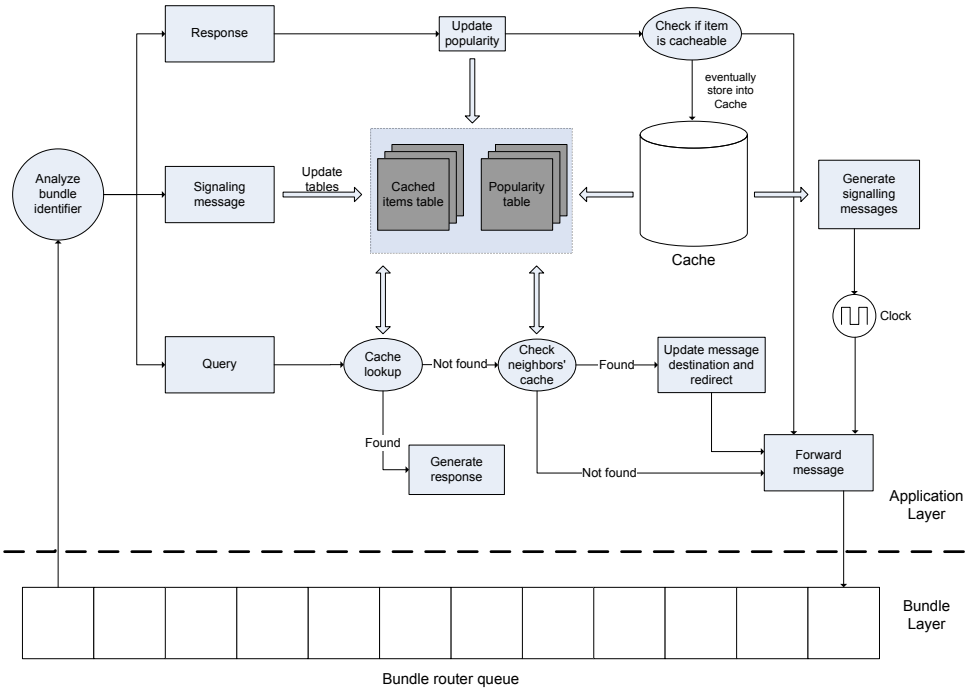


Figure 3.1: Processing flow for an incoming message at the application layer

Figure 3.1 illustrates the node's processing flow at the application layer, for different types of message bundles. If the incoming bundle is a *signalling* message, the node will update the popularity and the cached items' tables, without further forwarding of the message. If the incoming message is a *query*, the node performs first a lookup in its own cache to check if the content is stored locally. If the lookup operation is successful, the node stops further forwarding of the query message and generates a response to be sent to the requesting client. If the item is not cached locally, the node checks the table of its cached items if one of its neighbors is currently caching the requested item. If so, the destination of the query message is changed and the request is redirected to the neighbor node which has the item stored. If

none of the neighbors is caching the requested item, the query is forwarded as it is without modifications. If the incoming bundle is a *response* message, the node first updates its popularity table, increasing the counter corresponding to the item in the bundle. At this point, the node performs the operation of checking if the content is to be cached. The functionality for this operation is further explained in the following subsections. If the item yields positive decision in this operation, it will be stored in the local cache memory. Then, the cached items table will be updated and a new signalling message will be sent. The signalling message contains the new updated information about the cached items. After this operation, whether the item has been cached or not, the response message is forwarded to its destination. Moreover, signalling messages are generated by the node and sent to the neighbors at regular intervals. However, whenever an item is cached, a signalling message is sent immediately by automatically triggering the clock mechanism.

3.1.2 Caching Policies

Once each node has built its caching and popularity tables, it is ready to take a decision whether it is feasible to cache a certain item the next time it passes by. First, it is recommendable to wait for a certain *warmup time* before starting to cache, leaving the first few messages flow to the server. In this way, the popularity values are initialized correctly, avoiding the effect of sampling error when dealing with just a few samples which can lead to caching mistakes. Second, the node is not allowed to cache items that are already cached in it or in one of its neighbors. This is done to save storage space, because it allows the node to know exactly where the item is cached and the node can easily retrieve it without needing to waste extra resources. Furthermore, the item is cached in the node only if the node itself holds the maximum value of the popularity among his neighbors. This assures that the node really is the right place to cache. In addition, to avoid redundancy, the transit of a message on a server-client path is not allowed to lead to more than a single caching decision in an intermediate node.

It is possible that occasionally these rules are unintentionally broken, or simply the popularity values change during time and there is need for redrawing the *cache map*. For example, it can happen that there is a concurrent access to the cache system in two adjacent nodes at the same time (or within the interval between two consequent signalling messages) so that a message is cached twice. To deal with these situations, the node checks if it holds the max popularity for the items that the neighbors have cached after the delivery of each signalling message. If some discrepancies are found, the node caches the suggested item the next time it passes by, and sends a *Custody* message to the neighbor which used to hold it, telling it that the node keeps a copy of the message and that it can be deleted from the other cache where it has no longer reason to be stored.

When a node fulfils all the aforementioned requirements, it becomes a candidate for caching. The actual caching decision is then made considering the applied caching policy. In *Extreme Caching* (ExC), the message is always cached in the candidate node. This policy can lead to a large number of items cached in the network, and it is recommended for networks where the nodes have potentially unlimited storage space. Otherwise, in the case of high cost of storage resources this policy should be avoided.

The *Cache-Popular* (CP) policy is implemented in a way that the higher the item's popularity is - compared to the popularity of the other items in the node - the higher is the probability that it is cached. This is done in order to exploit the high frequency of access to the popular items and to offer a fast and reliable retrieval, as shown in work by Cao et al. [5].

On the other hand, as noticed in [17], in DTN environments retrieving the less popular items from the long tail becomes difficult, if not even impossible. To avoid this problem, the *Cache-Unpopular* (CU) policy is implemented - opposite to *Cache-Popular* - so that the least popular items are cached with the highest probability.

3.2 Topology Design

In the analysis part of this thesis, two different kinds of network topology are used. The first is called *Grid network topology* (Figure 3.2). In this scenario, a server is located in the center of the network and it is surrounded by cache nodes, while there are four clients standing at each corner.

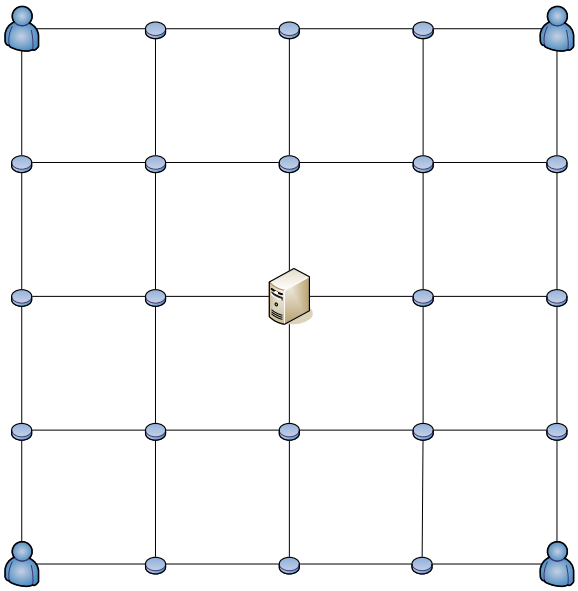


Figure 3.2: Grid network topology with $d = 4$ distance parameter, $\psi = 6$ shortest paths, and $n = 3.2$ average node degree

The topology is defined by a parameter d, which represents the hopcount distance between the clients. Consequently, there are $(d+1)^2$ total nodes in the network, with the $(d-1)^2$ in the middle, including the server, having node degree $n = 4$. Moreover, there are $4(d-1)$ edge caches with $n = 3$ and the clients have $n = 2$. The interesting property of this topology is that there are ψ different alternative shortest paths from each client to the server, each of which having d hop count length. ψ can be computed as:

$$\psi = \sum_{i=1}^{k} i \binom{2k-i-3}{k-3} \qquad \text{given} \quad k = \frac{d}{2} + 1 \quad \forall k \in \mathbb{N}, k \geq 3$$

This property can be exploited when using DTN routing protocols, where there are no routing tables and the path is not predetermined from source to destination, so the probability of reaching the target through one of the shortest paths can be increased in a controllable manner. The existence of multiple shortest paths is the motivation to choose the Grid topology.

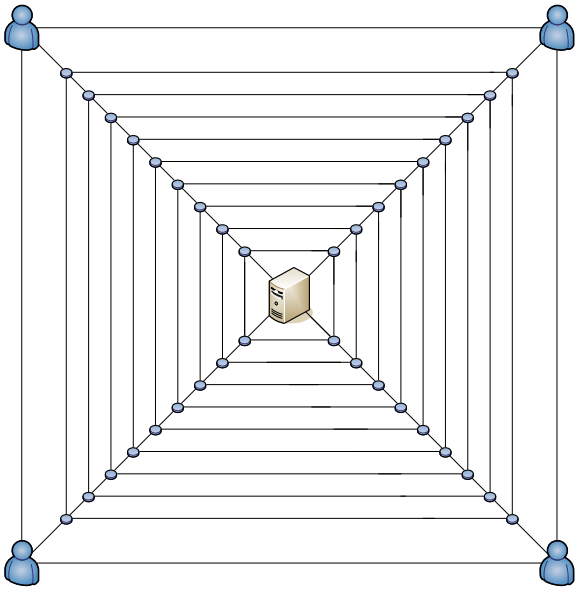


Figure 3.3: Ring network topology with $r = 10$ rings and node degree $n = 4$

The *Ring network topology* (Figure 3.3) represents a more uniform scenario, where the server is also located in the center, but is surrounded by r concentric rings of nodes. The caches are situated on the first $r - 1$ rings and have customizable node degree n, while the clients stand on the last ring, having node degree $n - 1$. However, in this topology there is only one shortest path from each client to the server - with distance r hops - which will be used only with probability $(\frac{1}{n-1})^r$. Ring topology allows to investigate the behavior of the system in networks with large path lengths. Moreover, a property of this topology is that any routing mistake increases the path length by only one hop.

3.3 Routing Protocols

The query messages from client to server and the relative responses from server (or cache) to client are routed in the network using zero-knowledge, i.e., *opportunistic* DTN routing protocols. Under this assumption, the nodes do not need to know any information about the network topology and can forward the messages according to a predetermined self-controlled mechanism. Three different DTN routing protocols are used in the simulations: *First Contact*, *Epidemic*, and *Binary Spray-and-Wait Routing*.

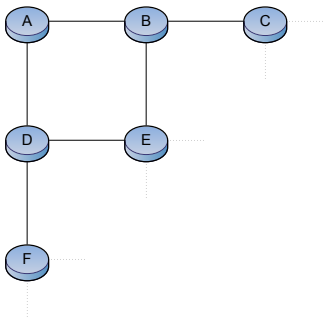
In *First Contact* routing [12], each node randomly chooses one of its neighbors and forwards to it a single copy of the message. If all interfaces of the node are busy, the message is held in the forwarding buffer until the first contact becomes available. To prevent loops, a node is not allowed to forward the message to a node where the message has already passed by. The message can be forwarded to a dead end and get lost. *First Contact* simply forwards the message to a randomly chosen neighbor in the case of static topologies.

In *Epidemic* routing [27] each node forwards a copy of the message to all its neighbors. In this way, the routing protocol eventually achieves very high message delivery success, but at the cost of flooding the network with an exponentially increasing number of duplicate messages. In fact, in a well connected topology with node degree n and destination node k hops away from the source, n^k messages are generated for a single query or response, and the message reaches every region of the network, which is not necessarily desired. In order to limit resource consumption, it is possible to limit the maximum hop count or the TTL for every message to avoid excess number of copies.

Binary Spray-and-Wait routing [25], *sprays* in the source node L copies of the messages into the network, and then *waits* until one of these reaches the destination. In more detail, the source of a message initially starts with L copies; it then forwards to its first contact neighbor (with no copies) $\lfloor \frac{L}{2} \rfloor$ copies and keeps $\lceil \frac{L}{2} \rceil$ for itself. Subsequently, it forwards half of the copies it has kept for himself (i.e. $\lfloor \frac{L}{4} \rfloor$) to the following node, and so on, until one

single copy is left. In order to reach the destination node k hops away, the starting value of L is expected to be at least equal to 2^k with fixed topology. The use of this protocol is a good compromise that can exploit the advantages of both *Epidemic* and *First Contact* routing.

However, the caching system has to be tested also in the simpler scenario where the shortest paths from source to destination can be discovered by using traditional routing protocols. For this purpose, we have implemented *SRIP*, a simplified version of RIP [14]. Every node has a routing table, containing the mapping between the message destination and the next hop node where the message has to be forwarded. A simplifying assumption is made: the routing tables are fixed and static, and known by the nodes from the start of the simulations.



Destination	Next hop
A	local
B	B
C	B
D	D
E	B or D
F	D

Figure 3.4: Example of *SRIP* routing table for node A

One peculiarity of *SRIP* is depicted in Figure 3.4. Whenever there are more than one shortest paths between two nodes, like in the Grid topology, the use of a predetermined routing table can lead to the use of only one of them, eventually preventing the messages from ever passing through certain nodes. To solve this problem, the node is free to choose the next hop randomly, in order to avoid the use of the same path every time and balance the load in the network. For example, there are two equal shortest paths from node A to node E (A-B-E and A-D-E) so A can decide to forward the message either through B or D. However, in the other cases (like the path from A to C) the path is unambiguous and strictly determined.

3.4 Query Distributions

In the simulation model of this thesis each server stores n items permanently. The items are labeled with increasing integer numbers from 1 to n and the clients issue queries for their retrieval. The query process is then a random selection of one of the items, each of which can be selected with a certain probability. The probability is modeled according to a predetermined probability distribution function (PDF). Three different PDFs are implemented: *Uniform*, *Quadratic* and *Pareto* distribution.

In *Uniform* distribution (*UD*) all items are queried with the same probability. As a consequence, the probability p_i of the i-th item being queried will simply be (Figure 3.5):

$$p_i = \frac{1}{n} \qquad \forall i \in \{1...n\}$$

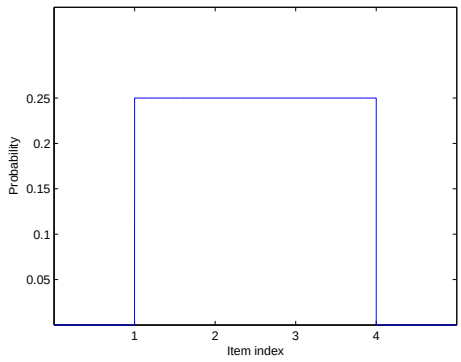


Figure 3.5: PDF of *Uniform* distribution with $n = 4$ items

It is also relevant to implement a model where some items are more likely to be requested than the others, as the content of the items is more interesting to a larger number of users. Consequently, in *Quadratic* distribution (*QD*) the items are queried with varying probabilities, so that the i-th item has a higher probability of being queried than the $(i+1)$-th. In particular the formula used is (Figure 3.6):

$$p_i = \frac{(n-i+1)^2}{\sum_{i=1}^{n}(n-i+1)^2} \qquad \forall i \in \{1...n\}$$

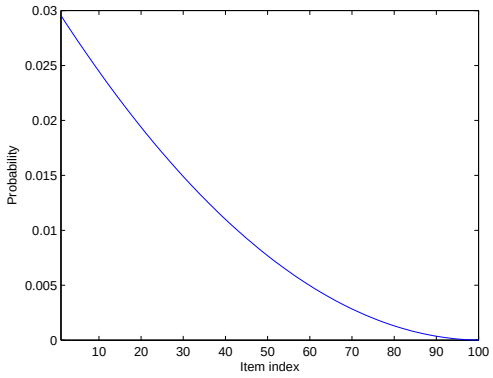


Figure 3.6: PDF of *Quadratic* distribution with $n = 100$ items

Pareto distribution (*PD*) is an enhancement of the *Quadratic*, and is used for better modeling of the real-life distribution of items in the Internet according to the Long Tail model discussed in [1]. The probability p_i of the i-th item being queried is obtained by (Figure 3.7):

$$p_i = \frac{\frac{k}{i^{(k+1)}}}{\sum_{i=1}^{n} \frac{k}{i^{(k+1)}}} \qquad \forall i \in \{1...n\}$$

where k is a positive integer-valued parameter.

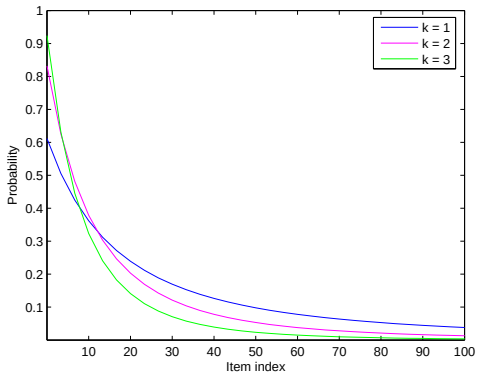


Figure 3.7: PDF of *Pareto* distribution with $n = 100$ items for three different values of parameter k

Chapter 4

Simulations

In this chapter simulation results are discussed to evaluate feasibility of the proposed solution. We combine simulations for different network topologies, routing protocols, query distributions and caching policies. We evaluate the performance of the distributed caching system by comparing it to the performance of traditional scenarios with no caching mechanisms.

4.1 Simulation Environment

All the simulations for the purpose of this thesis are performed by using the ONE - Opportunistic Networking Environment - simulator [13]. The ONE is a simulation environment written in Java that is capable of modeling either fixed or mobile network topology, given a settings file. All the nodes in the simulation can generate, query and exchange messages using the aforementioned DTN routing algorithms. The message passing can be monitored in real-time in the graphical user interface (GUI). For the simulations of this thesis, extensions to the source code had to be made. A system for metadata exchange for caching purposes was implemented, as well as a caching probability function to determine if caching items in the cache node. Moreover, the queries were given the functionality to be modeled according to different distributions. Finally, *SRIP* routing protocol was implemented.

4.2 Definition of Performance Metrics

First Contact routing is used to compare different caching mechanisms with four different values of average inter-query time α from 1 to 4 minutes. Inter-query time is defined as the amount of time each client waits after a query before issuing the next one. This value is uniformly distributed in the interval $[\alpha - 30s, \alpha + 30s]$. The approach allows to analyze the behavior of the system with different traffic intensities in the network. With small values of α there are many queries (and messages) traveling in the network, while there is less traffic (because of longer time between messages) when α is higher. The measurement unit for time is the *SimClock* attribute in the ONE simulator, which is expressed in seconds since the start of the simulation.

Binary Spray-and-Wait routing is used to compare different performance obtained by varying the number L of query messages distributed to the network. As stated in [25], it is expected that a value of $L = 2^d$ guarantees the reachability of nodes d hops away from the source. For this reason, a subset of values of L (always powers of two) is used to explore the behavior of the system in the interval $L \in [2^{x_{min}}, 2^{x_{max}}]$ where $x_{min}, d, x_{max} \in \mathbb{N}$ and $x_{min} < d < x_{max}$. For *Binary Spray-and-Wait* routing, the average inter-query time is always given value $\alpha = 1$ minute and the standard deviation is $\sigma = 30s$.

Efficiency η is defined as the ratio of responses which finally reach back to the clients, against the number of queries issued. It is straightforward that in the best possible scenario, where all the queries are satisfied and responded, the value is $\eta = 1$.

Delay τ is calculated as the amount of time elapsed between the moment the query is issued from the client until the corresponding response is received back to the client. The delay is registered only for the messages which reach back to the client, so efficiency has to be considered as well when evaluating the delay. The measurement unit is also simulation seconds.

Hopcount κ is the number of hops the response needs to pass by, only considering its way back from the server (or the cache) to the requesting client.

Again, the measurement is only for the messages which reach back to the client, so efficiency needs to be considered again.

Simulations are run 10 times for each combination of parameters, and average values are drawn. This allows to decrease the influence of the random variables used. In every graph, the average values are plotted. Every simulation lasts 28'000 *SimClock* seconds, which is roughly equal to an 8 hours working day. All parameters of the simulations are resumed in Table 4.1.

Simulation parameters	**Value**
Network topology	[Grid; Ring]
Parameter d (Grid topology)	4
Parameter r (Ring topology)	10
Average node degree	4
Number of clients	4
Number of servers	1
Total number of nodes (Grid topology)	25
Total number of nodes (Ring topology)	41
Query distribution	[uniform; quadratic; pareto]
Parameter k for Pareto distribution	1
Routing protocol	[FC; SaW; SRIP]
Average inter-query time (FC and SRIP)	[1; 2; 3; 4] min
Average inter-query time (SaW)	90 s
Inter-query time standard deviation	30 s
Parameter L for SAW (Grid topology)	[8; 16; 32; 64]
Parameter L for SAW (Ring topology)	[2^{10} - 2^{20}]
Cache policy	[NC; CP; CU, ExC]
Cache size	15 Mb
Number n of items	100
Simulation length	28k s (8 hours)
Warmup time	1000 s
Signalling message clock interval	60 s
Forward buffer size	5 Mb
Link capacity	5 Mbps
Message TTL	60 s
Message size	1 Mb
Number of simulation runs	10

Table 4.1: Parameters used in the simulations

4.3 Analysis of Grid Topology Scenario

Simulations are performed first in Grid topology with parameter $d = 4$. The topology is similar to the one presented in Figure 3.2. The server has $n = 100$ different items in its static memory. The clients are willing to request these resources. In particular, the client in the top left corner and the one in the bottom right corner issue queries for the items labeled with an odd number, while the clients in the top right and bottom left corners request the even-numbered items. Retrieval performance is first analyzed in the case where the intermediate nodes just forward the messages towards their destination (*No Caching*), and then compared to the cases where the caching policies presented in subsection 3.1.2 are applied.

4.3.1 Evaluation of First Contact Routing

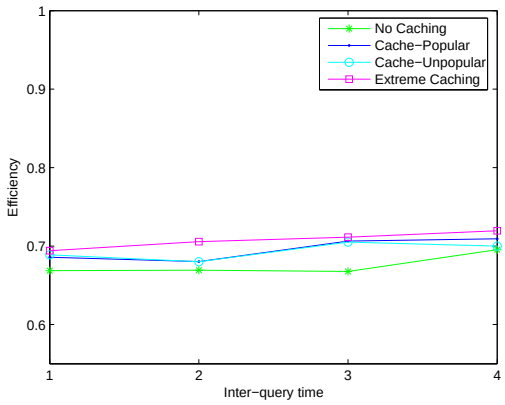


Figure 4.1: Efficiency η versus inter-query time α for Grid topology with parameter $d = 4$, *First Contact* routing and *Uniform* distribution

Uniform Distribution

First, efficiency is measured and the results are presented in Figure 4.1. The figure shows that over 65% of the queries successfully reach the server and the requested items are delivered back to the clients, in a scenario where caching

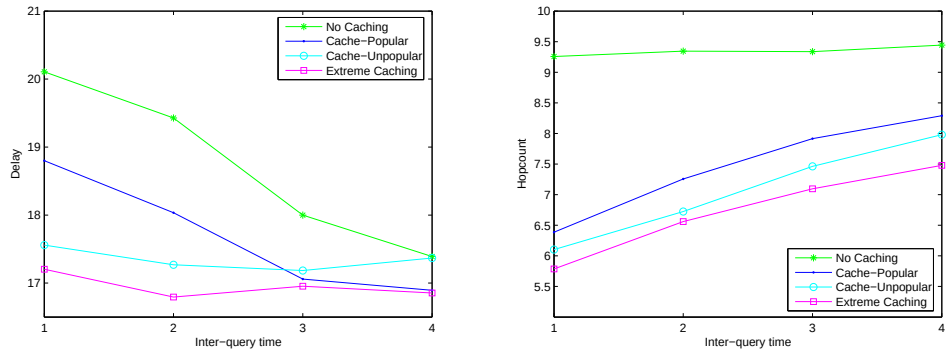


Figure 4.2: Delay τ (left) and hopcount κ (right) versus inter-query time α for Grid topology with parameter $d = 4$, *First Contact* routing and *Uniform* distribution

is not applied in the nodes. However, adding a caching system improves efficiency by only few percentages for the *Cache-Popular* and *Cache-Unpopular*, and up to $\eta \simeq 70\%$ on average for *Extreme Caching*. Another interesting observation is that performance keeps relatively stable in all traffic conditions without caching, while it seems that when applying *Extreme Caching* retrieval efficiency increases when the inter-query time is higher. This is explained by the fact that when there is less traffic, there is an higher probability for the messages of successfully reaching their destination without being queued too long and expire their TTL.

On the other hand, delay and hopcount distance of the reply are clearly decreased when using a caching mechanism (Figure 4.2). The delay in the *NC* case decreases when the inter-query time is high, and the same behavior is obtained when applying *CP* policy, however with not a particular improvement (just 6% on average). This is probably due to the longer waiting times in the forwarding queues caused by the higher amount of messages in the network when α is small. *Cache-Unpopular* and *Extreme Caching* instead keep their delay performance stable when varying the inter-query time, with *ExC* producing delays smaller than *CU* in the order of 500 *ms* on average.

The hopcount distance of replies is also decreased, as 2 to 3 fewer hops on average are needed to return a response when applying a caching policy

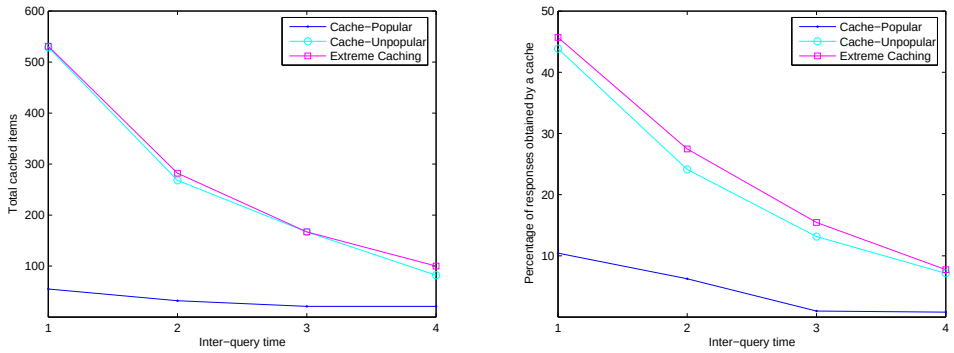


Figure 4.3: Total number of cached items (left) and percentage of responses obtained by a cache (right) versus inter-query time α for Grid topology with parameter $d = 4$, *First Contact* routing and *Uniform* distribution

(Figure 4.2 right). However, the hopcount increases for *CP*, *CU* and *ExC* policies when the inter-query time increases while it keeps stable in the scenario without caching. This is explained (Figure 4.3 left) by the fact that when there is less traffic in the network (higher inter-query time) there are less items cached and consequently it is less probable that the requests find the cached item on their path. As a consequence, in these cases it is more probable that the request needs to be anyway forwarded up to the server, but when α is small a higher percentage of query/response transmissions are handled by the intermediate caches without invoking the server (Figure 4.3 right). However, the results show that a very small percentage of queries is responded by the caches when applying *Cache-Popular* and even when applying more aggressive policies still more than half of the requests have to be redirected to the server. This happens when the distribution of the queries is uniform, because it is statistically very rare that a request issued for the same item happens to be routed through the same path when we are using a routing protocol like *FC* which basically routes randomly. For the same reasons, *Cache-Popular* fails to obtain a significative improvement in all performance tests, as it tends to behave very similarly to the *NC*. This happens mostly because the node tries to cache the most popular item between a large collection of equally popular items, so the choice made is not always correct and consistent.

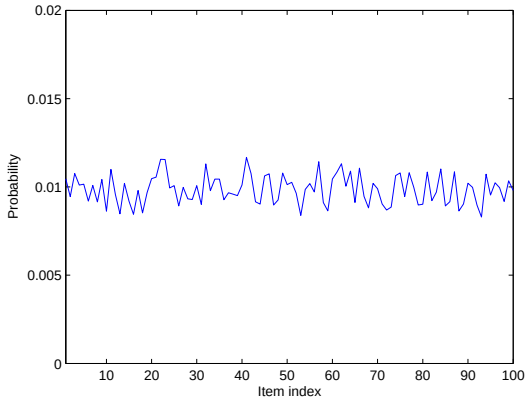


Figure 4.4: PDF of the responses received back for Grid topology with parameter $d = 4$, *First Contact* routing and *Uniform* query distribution

Figure 4.4 shows the distribution of the items that are successfully received back in the clients side. For better readability, the graph presents the average from all samples from all the caching policies. If plotted separately, *CP*, *CU* and *ExC* would have practically been overlapping, since they roughly have the same mean value and variance, so we can consider them as the same, and we can draw general conclusions. It is clear that the average of the probabilities equals the theoretical value of $p = \frac{1}{100}$ presented in Section 3.4, with just a small additional noise contribution. This means that, in general, if the distribution is uniform all items are received back with the same probability.

Quadratic and Pareto Distribution

When exploring behavior of the system in a case where the distribution of the items in the network is not uniform, performance tests give the same results as for the uniform distribution scenario. In fact, in both the quadratic and Pareto simulations (Figure 4.5), efficiency keeps more or less stable between 65% and 70%, with *Extreme Caching* performing better than *Cache-Popular* and *Cache-Unpopular*, regardless of the inter-query time.

However, when the distribution of the queries is not uniform, the delay is

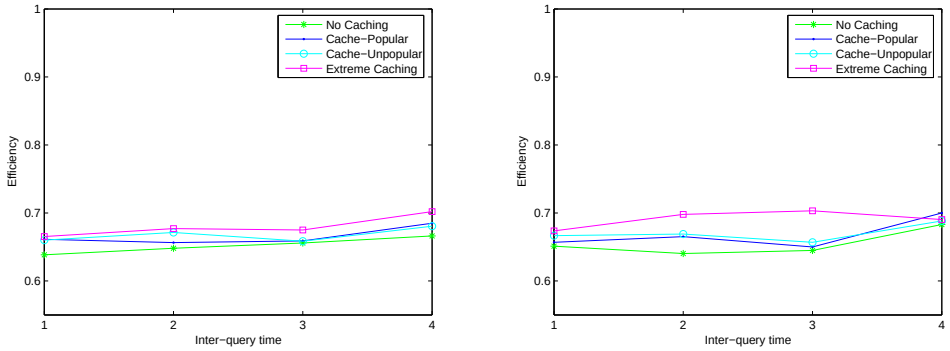


Figure 4.5: Efficiency η versus inter-query time α for Grid topology with parameter $d = 4$, *First Contact* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

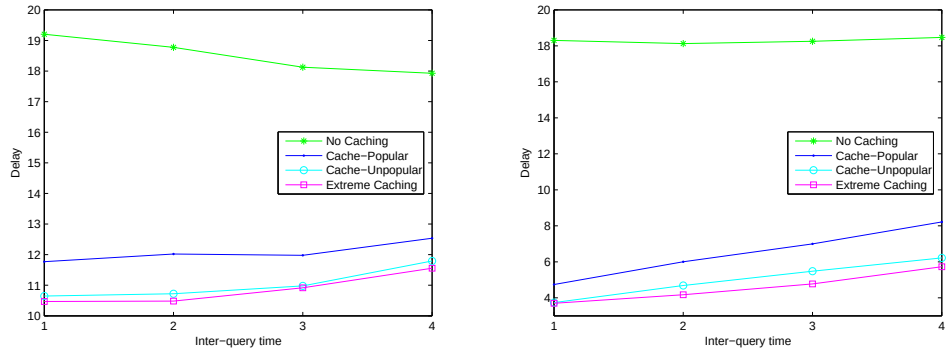


Figure 4.6: Delay τ versus inter-query time α for Grid topology with parameter $d = 4$, *First Contact* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

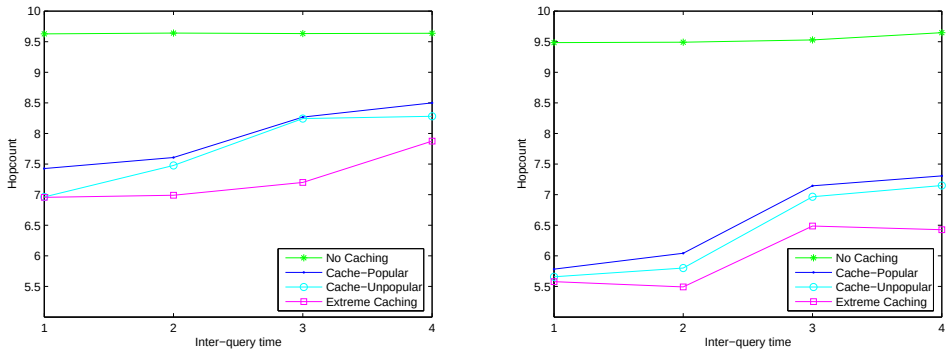


Figure 4.7: Hopcount κ versus inter-query time α for Grid topology with parameter $d = 4$, *First Contact* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

considerably reduced (Figure 4.6). In fact, while the delay for *NC* has the same average values than in the uniform distribution scenario, applying a caching policy reduces it to approximately one half for *QD* distribution and even to a third for *PD* distribution. Moreover, the delay in case of any caching policy increases with the increase of the inter-query time.

The hopcount results are again similar to the *UD* distribution scenario, as shown in Figure 4.7. The number of hops needed for retrieving a message to the client keeps stable for *No Caching* while it increases with increasing inter-query time for all the other cache policies.

The main difference between the uniform distribution scenario and the others stands in the amount of items cached and, as a consequence, the percentage of responses obtained directly from a cache (Figures 4.8 - 4.9). Since the distributions are not uniform, the popularity values in the intermediate nodes are registered correctly, a much easier task than ordering 100 equally probable items as was needed in the *UD* distribution case. For this reason, the caching decisions are performed in a very efficient way, as even 80% of the requests are sent by the cache nodes on average for Pareto distribution, with the need of caching only 30% of the total items needed for the uniform distribution. Quadratic distribution results are similar, but not so outstanding as in the Pareto case, probably because of a less steep popularity curve,

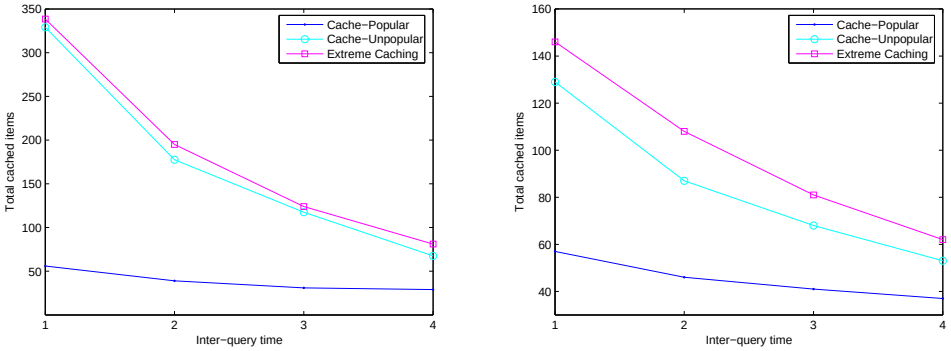


Figure 4.8: Total number of cached items versus inter-query time α for Grid topology with parameter $d = 4$, *First Contact* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

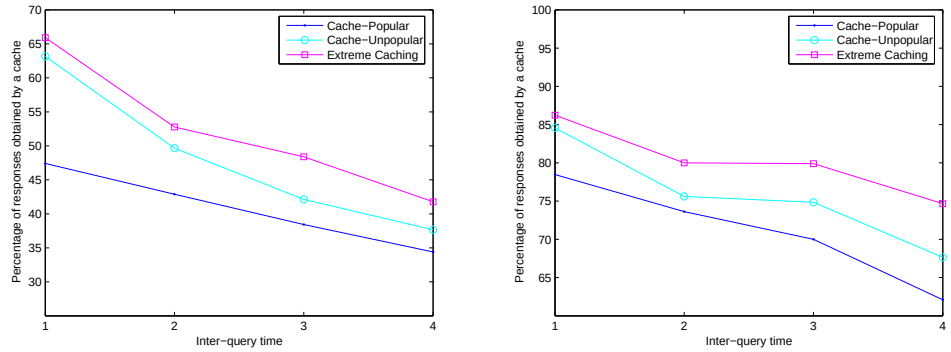


Figure 4.9: Percentage of responses obtained by a cache versus inter-query time α for Grid topology with parameter $d = 4$, *First Contact* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

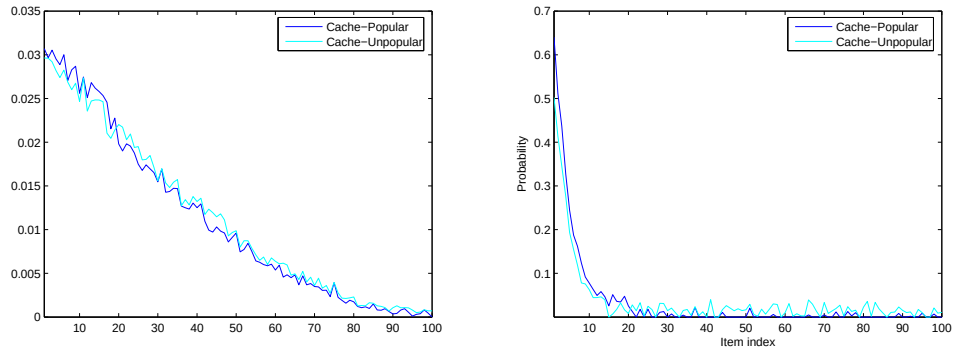


Figure 4.10: PDF of the responses received back for Grid topology with parameter $d = 4$, *First Contact* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

and consecutive items are likely to be misordered. However, just like in the uniform distribution case, the percentage of responses obtained from a cache node decreases with the increase of the average inter-query time, for both quadratic and Pareto distributions. This is, again, a consequence of the smaller number of items cached, that causes the need of redirecting more queries up to the server.

Figure 4.10 (left) presents the distribution of the responses received back, when the query distribution is quadratic. If we compare these values to the theoretical ones presented in Figure 3.6, we can see that the most popular items have even higher probability to be received, while for the least popular the probability is smaller than expected. This behavior is even more clear in the case of Pareto distribution (Figure 4.10 right). An interesting observation is that *Cache-Popular* results in higher probabilities of receiving items with a small identifier, while after a certain threshold ($n \simeq 20$) *Cache-Unpopular* starts to become the policy who has the higher probability of receiving the items, for both quadratic and Pareto cases. This is an explanation why *Cache-Unpopular* performs much better than *Cache-Popular*. In fact, while *CP* concentrates on caching only few popular items, *CU* exploits the long tail, which overall comprises a bigger portion of the items.

4.3.2 Evaluation of Spray-and-Wait Routing

Uniform Distribution

The simulations for *Spray-and-Wait* routing use values of $L = \{8, 16, 32, 64\}$, i.e., from 2^3 to 2^6. The optimal value for reaching nodes 4 hops away, as proposed in [25], is $L = 16$ in this Grid topology. As expected, efficiency in the *No Caching* case is really poor when $L = 8$, while it is acceptable (more than 70% of queries responded back, and slightly better than with *First Contact* routing) only when $L = 32$. Moreover, when the number of copies increases further, performance increases again, giving results not far from full 100% efficiency. Nevertheless, when applying a caching policy the performance is even higher. In fact, *Cache-Popular* gives an efficiency value which is two times better than *NC*, while *Extreme Caching* performs at $\eta = 55\%$ efficiency, both when $L = 8$. Starting from $L = 16$ the results again approach 100% efficiency (Figure 4.11).

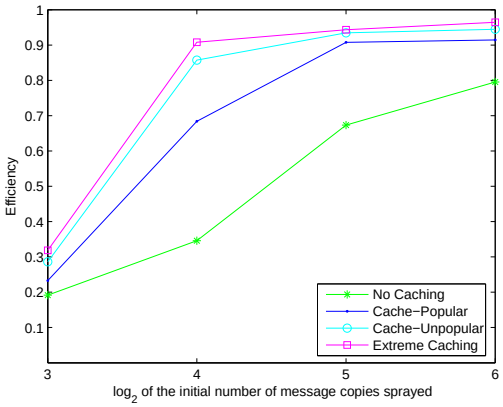


Figure 4.11: Efficiency η versus number L of message copies initially sprayed for Grid topology with parameter $d = 4$, *Binary Spray-and-Wait* routing and *Uniform* distribution

Good performance of caching in terms of retrieval success has its drawback as an increased delay (Figure 4.12 left). In fact, without caching the delay keeps more or less constant ($\tau \simeq 8.5$ seconds) when varying L. If *CP*, *CU* or *ExC* caching policies are implemented, the delay exponentially increases up

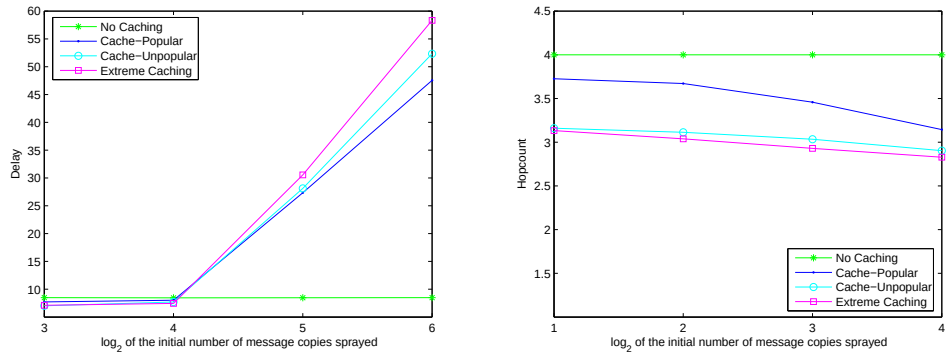


Figure 4.12: Delay τ (left) and hopcount κ (right) versus number L of message copies initially sprayed for Grid topology with parameter $d = 4$, *Binary Spray-and-Wait* routing and *Uniform* distribution

to $\tau = 50$ seconds on average with $L = 64$, which is rather unacceptable. This is caused by the very high number of distributed messages in the network, which creates longer queues in the forwarding buffers and additional latency.

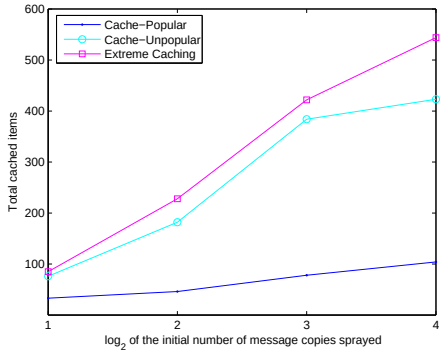


Figure 4.13: Total number of cached items versus number L of message copies initially sprayed for Grid topology with parameter $d = 4$, *Binary Spray-and-Wait* routing and *Uniform* distribution

On the other hand, performance in terms of hopcount is improved (Figure 4.12 right). As expected *NC* performs exactly with $\kappa = 4$ hops, which is the shortest path distance calculated in Section 3.2 regardless of the actual value of L . *Cache-Popular*, *Cache-Unpopular* and *Extreme Caching* can reduce the distance of about 1 hop, when L increases. This is due to the fact that it is more likely, with high values of L, that the query reaches a node

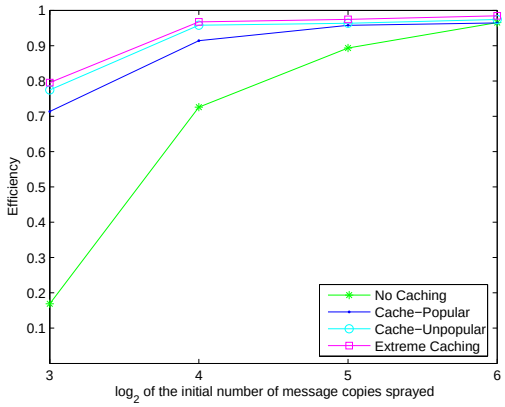


Figure 4.14: Efficiency η versus number L of message copies initially sprayed for Grid topology with parameter $d = 4$, *Binary Spray-and-Wait* routing and *Pareto* distribution

which has cached the desired item within a small distance from the client, as there are more cached items all around the network (Figure 4.13). Finally, the percentage of queries responded directly by the cache nodes does not change when varying L, it remains 57% on average.

Quadratic and Pareto Distribution

Quadratic and Pareto distribution behave similarly than with *First Contact* routing. When the query distribution is not uniform, the overall performance of the system is improved. However, there is not relevant difference between the quadratic and Pareto distribution results, hence only the graphs related to *PD* are shown.

Efficiency η with Pareto distribution is shown in Figure 4.14. The results are notably better than in the *UD* distribution case, as η is high already with $L = 16$. The reason is probably the same that we explained in the *FC* routing evaluation, as with uniform distribution the nodes try to cache the most popular item between a large collection of equally popular items, so the choice made is not always consistent, while correct decisions are made when the popularity rankings of the items are clear.

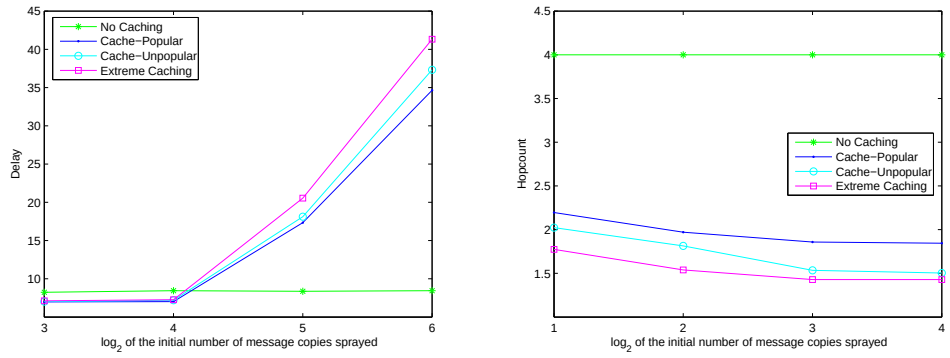


Figure 4.15: Delay τ (left) and hopcount κ (right) versus number L of message copies initially sprayed for Grid topology with parameter $d = 4$, *Binary Spray-and-Wait* routing and *Pareto* distribution

The delay and hopcount results are also similar to the uniform distribution case (Figure 4.15). Performance of *No Caching* keeps always stable, but when applying any cache policy the delay increases and the hopcount decreases when L is increased. However, delay is generally smaller than when we have uniform distribution. The hopcount performance is also better with *PD* distribution. Again, all these results are obtained without the need of caching a large amount of items in the nodes, opposite than with *UD* distribution, as shown in Figure 4.16.

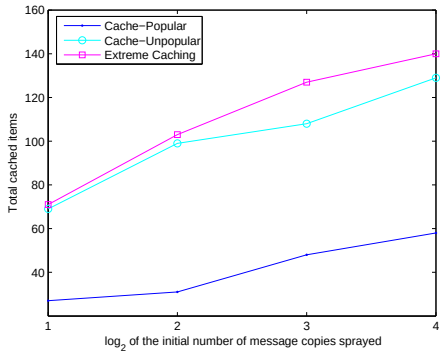


Figure 4.16: Total number of cached items versus number L of message copies initially sprayed for Grid topology with parameter $d = 4$, *Binary Spray-and-Wait* routing and *Pareto* distribution

Percentage of queries responded directly by the cache nodes does not change when varying L, with an average value of 91% for Pareto distribution. Additionally, the distribution of the items that have successfully been received by the clients is similar to the ones presented in Figures 4.4 and 4.10. From this point of view, there is no difference between *First Contact* and *Spray-and-Wait* routing.

4.3.3 Evaluation of Simplified RIP Routing

When routing the messages with a traditional routing protocol (under the simplified assumptions mentioned in Section 3.3) like *Simplified Routing Information Protocol* (SRIP), which assumes a predetermined routing table in every node, we have a single path between a client and a server and no topology changes. Hence, there are no packet losses, and all the queries and responses are successfully delivered to their final destination. As a consequence, our results show that efficiency is $\eta = 1$ for all query distributions and all caching policies when using *SRIP*. Moreover, since the paths are certain and determined, results show the presence of a strong direct relationship between delay and hopcount performance, as they both measure the distance of the responding node from the client, whether the measurement unit is time or space.

Uniform Distribution

Delay performance with a uniform query distribution is shown in Figure 4.17 (left). As expected, the delay keeps stable for *NC*, with an average value of $\tau \simeq 9$ seconds, when varying the inter-query time. Quite surprisingly, also *Cache-Popular*'s performance is stable, just few milliseconds better than *NC*. *Extreme Caching* gives the best results, with an average delay of only 6 seconds when inter-query time is $\alpha = 1$ minute. However, when the inter-query time increases, the delay increases as well, approaching the 9 seconds threshold value on the long term. *Cache-Unpopular*'s performance is in the case of *UD* distribution very similar to *ExC* policy case.

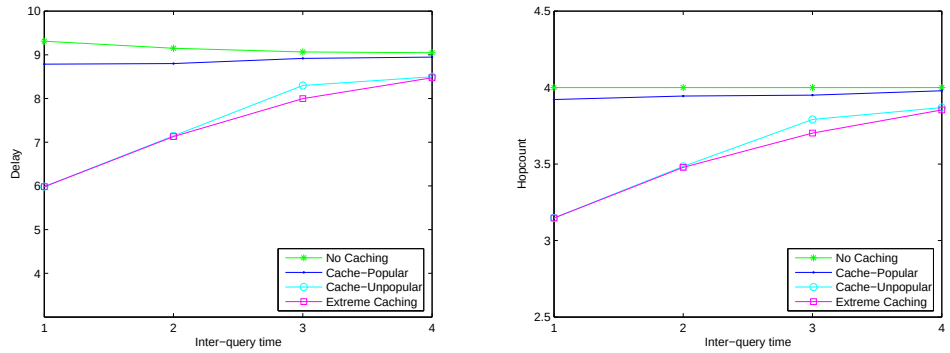


Figure 4.17: Delay τ (left) and hopcount κ (right) versus inter-query time α for Grid topology with parameter $d = 4$, *Simplified RIP* routing and *Uniform* distribution

Hopcount performance with *No Caching* policy is always $\kappa = 4$, the shortest path distance from the clients to the server. Similarly to the delay results, *Cache-Popular* doesn't give a significant performance improvement. Again, *Cache-Unpopular* and *Extreme Caching* behave very similarly, but can reduce the hopcount average value by only less than one hop with small values of α, while when the queries are more sparsely spaced the performance approaches the *NC* case. Hopcount results are shown in Figure 4.17 (right).

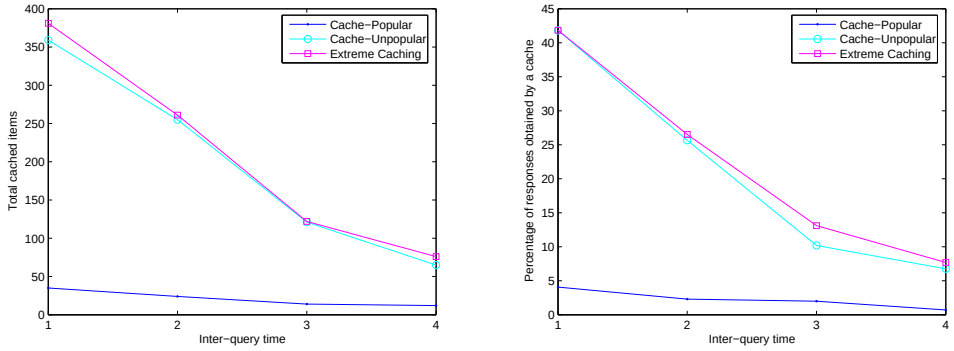


Figure 4.18: Total number of cached items (left) and percentage of responses obtained by a cache (right) versus inter-query time α for Grid topology with parameter $d = 4$, *Simplified RIP* routing and *Uniform* distribution

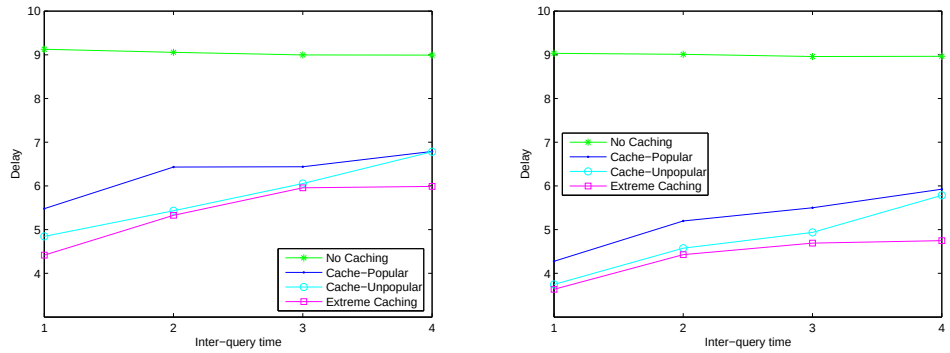


Figure 4.19: Delay τ versus inter-query time α for Grid topology with parameter $d = 4$, *Simplified RIP* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

Explanation for bad performance can be found in Figure 4.18. *Cache-Popular* fails to correctly initialize the popularity values, and caches only few items, resulting in percentage of responses from cache to approach 0%. *Cache-Unpopular* and *Extreme Caching*, while caching a high amount of items, cannot obtain more than 42% of responses from cache even in the best case with $\alpha = 1$ min. For all caching policies, when the inter-query time increases, the amount of cached items decreases, and the percentage of responses obtained by a cache decreases as well.

Quadratic and Pareto Distribution

Caching system performance improves significantly when the query distribution is not uniform. For both quadratic and Pareto distributions the delay in the *No Caching* case confirms the *UD* distribution results, and so does the hopcount performance, with $\tau \simeq 9$ seconds and $\kappa = 4$ (Figures 4.19 - 4.20). *Extreme Caching* is always the best performing caching policy with the minimum delay $\tau \simeq 4.3$ and $\tau \simeq 3.7$ seconds for *QD* and *PD* respectively when $\alpha = 1$ minute. Hopcount is also reduced to a small value of $\kappa = 2.7$ and $\kappa = 2.5$. *Cache-Unpopular* results are again very similar to *ExC* policy.

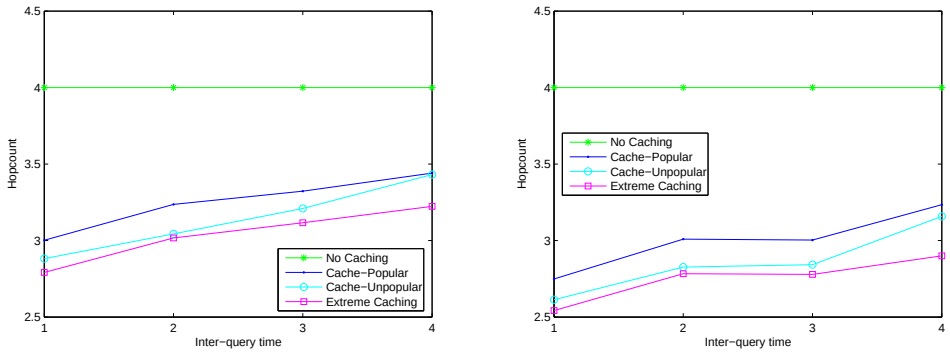


Figure 4.20: Hopcount κ versus inter-query time α for Grid topology with parameter $d = 4$, *Simplified RIP* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

However, in this case *Cache-Popular* performs sufficiently well, with a delay just about one second higher than with *Extreme Caching* on average (and less than 0.5 hops more) for both quadratic and Pareto distribution. As in the previous simulations, delay and hopcount increase with the increase of inter-query time α for all caching policies.

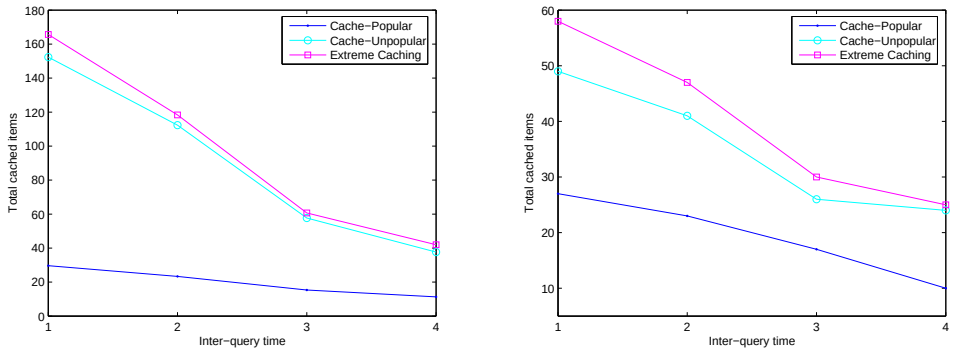


Figure 4.21: Total number of cached items versus inter-query time α for Grid topology with parameter $d = 4$, *Simplified RIP* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

Figures 4.21 and 4.22 explain these improved results. In fact, even if caching six times less items than in the *UD* distribution case, with a Pareto query distribution we can obtain up to 70% of responses from a cache. For quadratic distribution however, about a half of the total items for the *UD* case are

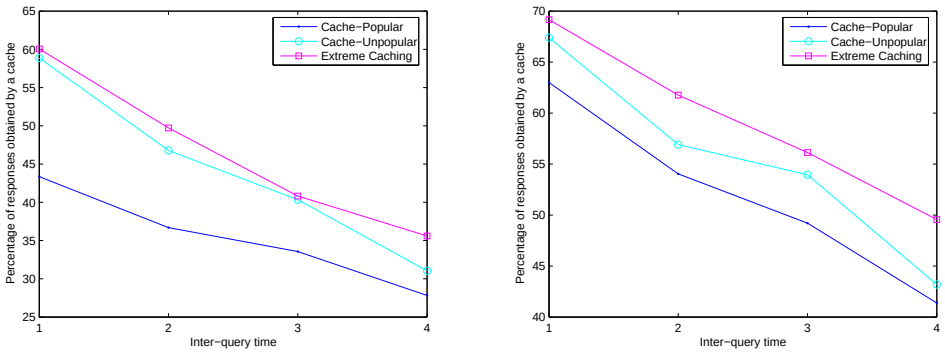


Figure 4.22: Percentage of responses obtained by a cache versus inter-query time α for Grid topology with parameter $d = 4$, *Simplified RIP* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

cached, resulting in almost 60% of responses from cache. Again, for all caching policies, the number of items cached and the percentage of responses obtained from cache both decrease with the increase of average inter-query time. Like in the previous results, *Extreme Caching* requires the highest amount of cached items, followed by *Cache-Unpopular*. *Cache-Popular* instead always tends to cache a very small amount of items.

4.4 Analysis of Ring Topology Scenario

In this section, simulations are performed in a Ring topology with $r = 10$ concentric rings and node degree $n = 4$, like the one presented in Figure 3.3. The server has in its static memory $n = 100$ different resources, that the clients are willing to request. Like in Grid topology, the client in the top left and the one in the bottom right corner request odd items, while the client in the top right corner and the one in the bottom left corner query for even-numbered items. Again, performance is first analyzed in the case where the intermediate nodes just forward the messages towards their destination, and then compared to the cases where *Cache-Popular* (or *Cache-Unpopular*) and *Extreme Caching* are applied. Simulation results show in several cases similar behavior than for Grid topology: for this reason, whenever not differently explained, the conclusions drawn in the previous section are valid. In particular, the performance of *ExC* policy is always better than *CU* and *CP*.

4.4.1 Evaluation of First Contact Routing

Uniform Distribution

Efficiency figures for *First Contact* routing in the ring topology show unacceptable performance (Figure 4.23). Simulations give results of $\eta = 15\%$ regardless of the inter-query time for *NC*. However, when applying any cache policy, there is no apparent improvement, and performance keeps stable.

On the other hand, differently than Grid topology, delay performance keeps stable for every value of α in the *No Caching* case, while using a caching mechanism reduces the retrieval time (Figure 4.24 left). The same behavior is observable for hopcount performance (Figure 4.24 right). Again as the inter-query time increases, performance for delay and hopcount metrics suffers and closes to *NC* performance, as there are fewer cached items around the network (Figure 4.25 left) and less responses are handled by cache nodes (Figure 4.25 right), i.e., as most 30% with *ExC*.

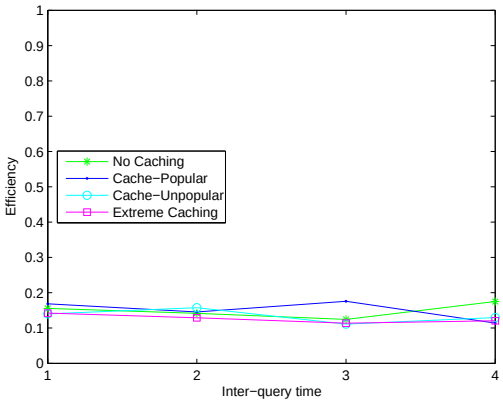


Figure 4.23: Efficiency η versus inter-query time for Ring topology with $r = 10$ rings and node degree $n = 4$, *First Contact* routing and *Uniform* distribution

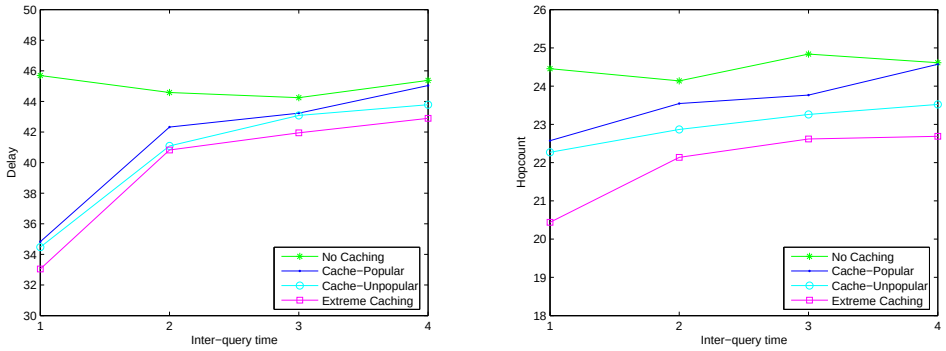


Figure 4.24: Delay τ (left) and hopcount κ (right) versus inter-query time for Ring topology with $r = 10$ rings and node degree $n = 4$, *First Contact* routing and *Uniform* distribution

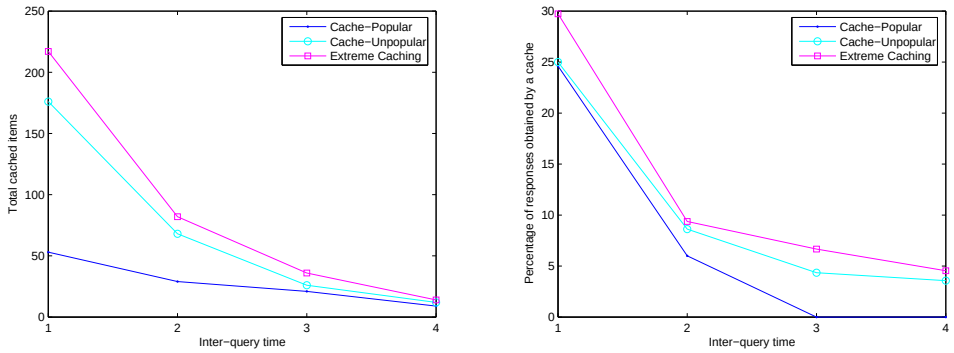


Figure 4.25: Total number of cached items (left) and percentage of responses obtained by a cache (right) versus inter-query time for Ring topology with $r = 10$ rings and node degree $n = 4$, *First Contact* routing and *Uniform* distribution

Quadratic and Pareto Distribution

First Contact routing performs again very poorly in the case of *No Caching* policy, as we have efficiency $\eta \simeq 13\%$ for all the values of α. However, differently than with *UD* distribution, performance is improved when applying the caching schemes. *Extreme Caching* performs about 6% better than *Cache-Popular*, and *Cache-Unpopular* shows performance between these cases. Quadratic distribution shows efficiency up to 43% for *ExC* (Figure 4.26 left), while the best result with Pareto distribution is 57% (Figure 4.26 right). Moreover, one observation stands out quite clearly: for all caching schemes efficiency increases when there is more traffic, i.e., with smaller values of α. The conclusion is again that when there are more messages travelling in the network, it is more probable that a nearby caching node has stored the requested message.

Delay in the case of *No Caching* has the same average value than in the *UD* distribution case, i.e., $\tau \simeq 45$ seconds for both quadratic and Pareto distribution. When applying cache mechanisms, the delay increases with the increase of the inter-query time. However, quadratic distribution leads to a delay of 15-20 seconds, while Pareto distribution even reduces delay down to 6 seconds on average (Figure 4.27).

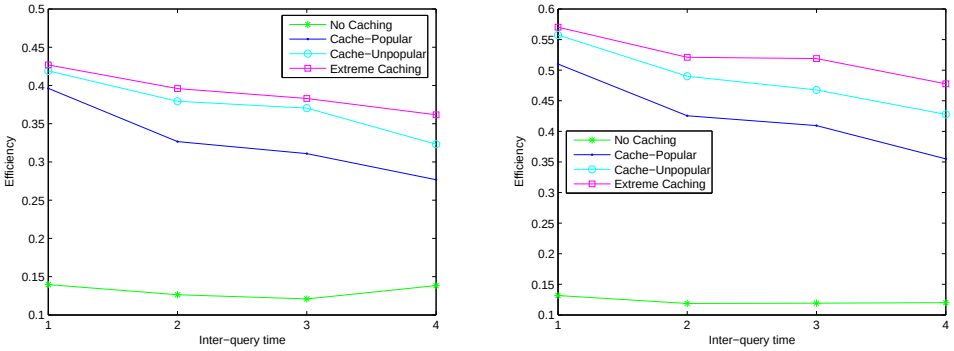


Figure 4.26: Efficiency η versus inter-query time for Ring topology with $r = 10$ rings and node degree $n = 4$, *First Contact* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

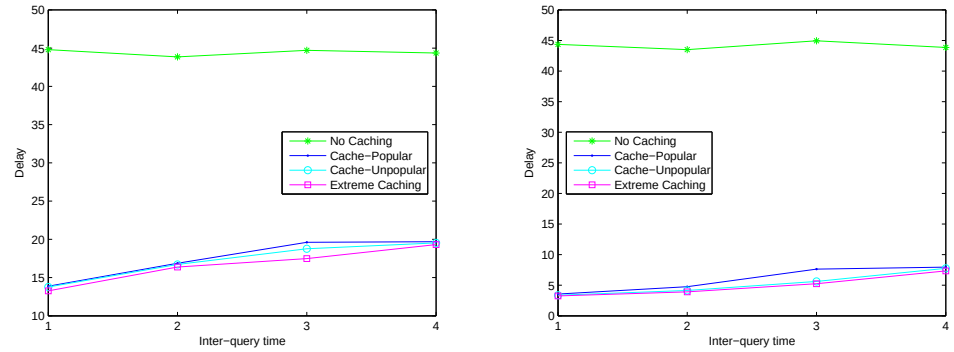


Figure 4.27: Delay τ versus inter-query time for Ring topology with $r = 10$ rings and node degree $n = 4$, *First Contact* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

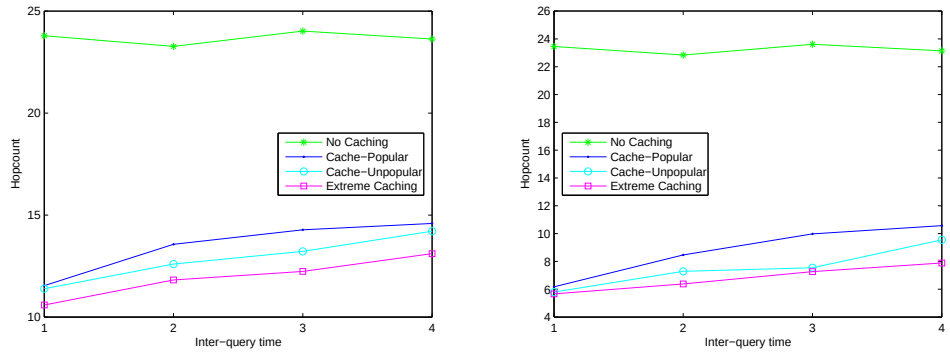


Figure 4.28: Hopcount κ versus inter-query time for Ring topology with $r = 10$ rings and node degree $n = 4$, *First Contact* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

Characteristics of hopcount performance are very similar. Hopcount is $\kappa \simeq 24$ for *NC* for both non-uniform distributions. The use of a caching systems reduces the hops needed to retrieve the response, with *Extreme Caching* having its best result with $\kappa \simeq 10$ for inter-query time $\alpha = 1$ minute and quadratic distribution (Figure 4.28 left), and $\kappa = 6$ for Pareto distribution (Figure 4.28 right).

Figure 4.29 shows the amount of cached items in the system. The results confirm the observations obtained with the Grid topology, where Pareto and quadratic distribution require to cache less items than with *UD* distribution. However, *QD* can provide up to 70% of responses obtained from a cache, while even 95% performance can be reached with *PD* distribution (Figure 4.30).

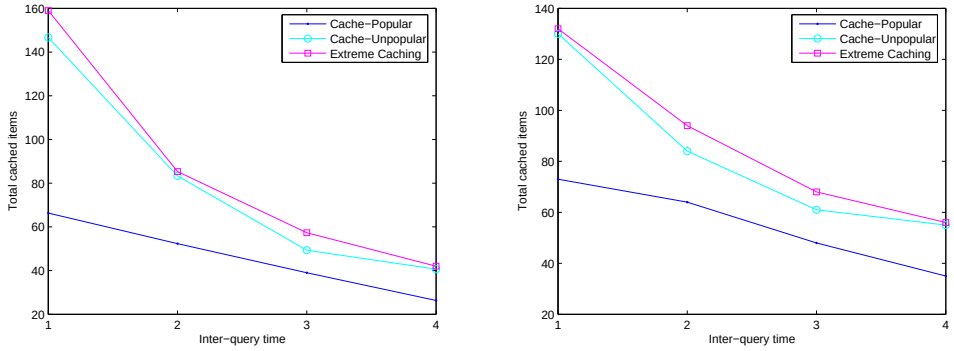


Figure 4.29: Total number of cached items versus inter-query time for Ring topology with $r = 10$ rings and node degree $n = 4$, *First Contact* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

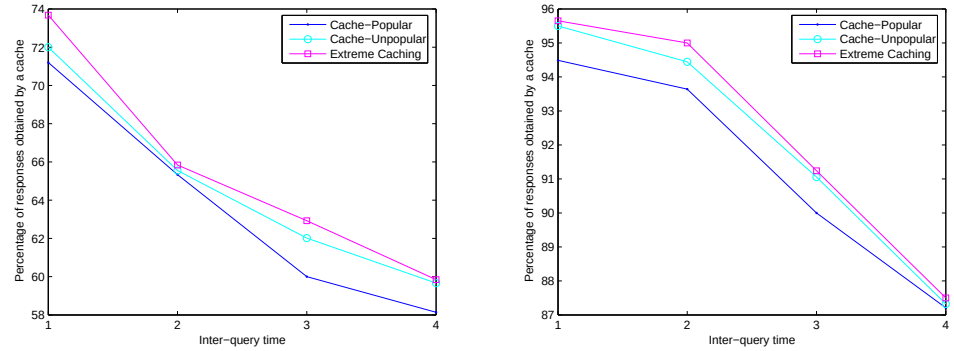


Figure 4.30: Percentage of responses obtained by a cache versus inter-query time for Ring topology with $r = 10$ rings and node degree $n = 4$, *First Contact* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

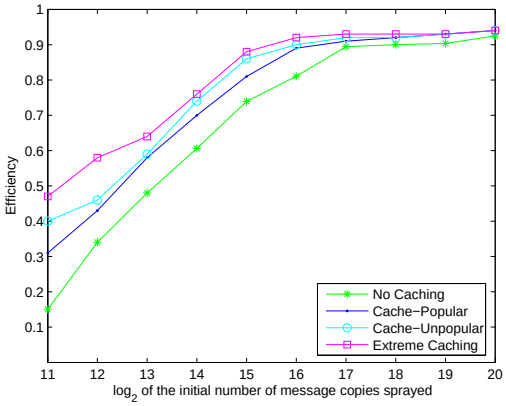


Figure 4.31: Efficiency η versus number L of message copies initially sprayed for Ring topology with $r = 10$ rings and node degree $n = 4$, *Binary Spray-and-Wait* routing and *Uniform* distribution

4.4.2 Evaluation of Spray-and-Wait Routing

Uniform Distribution

According to the paper by Spyropoulos et al. [25], implementing a *Binary Spray-and-Wait* routing mechanism with initial number of copies $L = 2^{10} = 2048$ should be enough to reach nodes within 10 hops distance, like in the Ring topology we use. However, our results indicate that only 0.002% of the queries reach back to the final target, i.e., an amount of data too small to have realistic analysis. This bad result is motivated by the fact, as explained in Section 3.2, that the probability of finding the shortest path is just $(\frac{1}{n-1})^r$. For this reason, values of L smaller or equal than 2^{10} are not considered, hence we are focusing on the interval $L \in [2^{11}, 2^{20}]$ acknowledging that these values are extremely high (up to a million messages sprayed for a single query): *Spray-and-Wait* fits poorly in static topologies with large path lengths.

Efficiency of *NC* policy increases, as expected, with the increase of L (Figure 4.31). However, while performance of *Cache Popular* and *Extreme Caching* is better, the improvement is just 10% and 20% on average, respectively for *CP* and *ExC*.

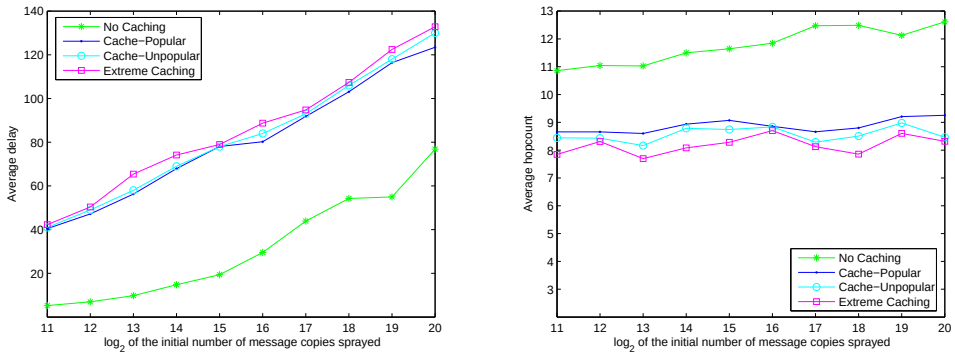


Figure 4.32: Delay τ (left) and hopcount κ (right) versus number L of message copies initially sprayed for Ring topology with $r = 10$ rings and node degree $n = 4$, *Binary Spray-and-Wait* routing and *Uniform* distribution

Differently than Grid topology, in this situation delay does not increase so rapidly in the *NC* case when increasing L (Figure 4.32 left). Nevertheless, when appliying *CP*, *CU* or *ExC* policies the behaviour is the same, with delays intolerably high, up to $\tau \simeq 130$ *SimClock* seconds. This depends on the very high number of messages travelling in the network, due to high values of L.

Hopcount performance κ is improved, as about two less hops are needed for *NC* than are requested in the case of *CP* policy and even less for *Extreme Caching* (Figure 4.32 right). This behavior is similar to what was observed with Grid topology, with the only difference that without caching, the average value of κ increases with the increase of L while in the other case (Figure 4.12 right) it was stable. One possible explanation is that with high values of L the probability of reaching the destination through longer paths is included, so that the average is influenced. However, when we explore the data further we can see that the majority of the entries have a hopcount value of $\kappa \simeq 11$.

Quadratic and Pareto Distribution

When the query distribution is not uniform, efficiency does not change considerably, so the results are comparable to Figure 4.31. However, delay and hopcount are notably reduced (Figures 4.33 - 4.34). In particular, delay is

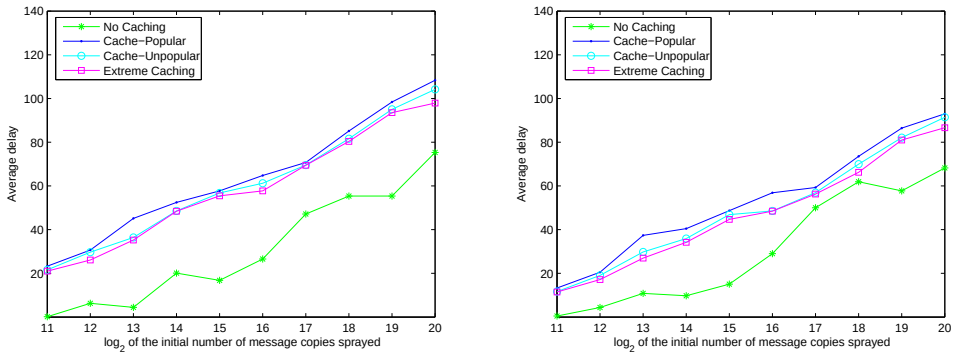


Figure 4.33: Delay τ versus number L of message copies initially sprayed for Ring topology with $r = 10$ rings and node degree $n = 4$, *Binary Spray-and-Wait* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

reduced by 20 seconds on average with quadratic distribution, and up to 30 seconds with Pareto distribution. The hops needed for delivering the responses are reduced to $\kappa = 6.5$ on average for QD and $\kappa \simeq 4$ for PD.

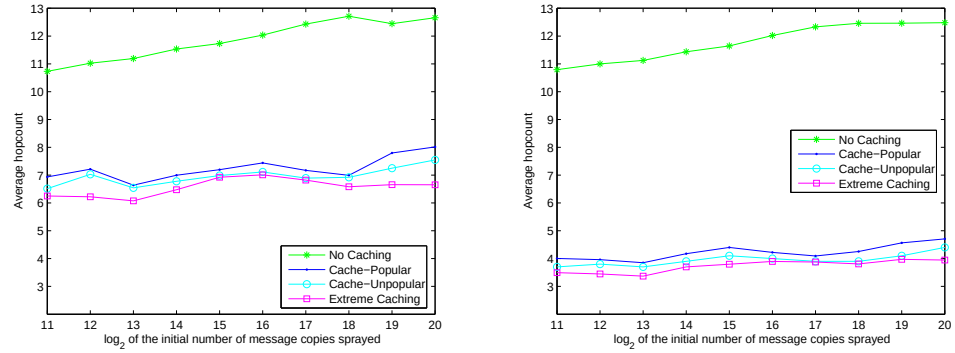


Figure 4.34: Hopcount κ versus number L of message copies initially sprayed for Ring topology with $r = 10$ rings and node degree $n = 4$, *Binary Spray-and-Wait* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

4.4.3 Evaluation of Simplified RIP Routing

Simulation results for *SRIP* routing in Ring topology confirm the results obtained in Section 4.3.3. In the *No Caching* case delay and hopcount keep stable for all inter-query time values, and all query distributions ($\tau = 25$ seconds and $\kappa = 10$ hops). On the other hand, when applying any caching mechanism, delay and hopcount increase with the increase of inter-query time. In particular, delay can be reduced to 17, 14 and 12 seconds (with *Extreme Caching*) for uniform, quadratic and Pareto distribution respectively, while hopcount is as small as 8.1, 6.5 and 5.1 hops in the three cases. Moreover, as mentioned previously for other simulations, Pareto and quadratic distribution require less items to be cached than uniform distribution, but they can provide a higher percentage of responses obtained from cache (in this case up to 56% for *PD* distribution). Detailed results are shown in Figures 4.35 - 4.40.

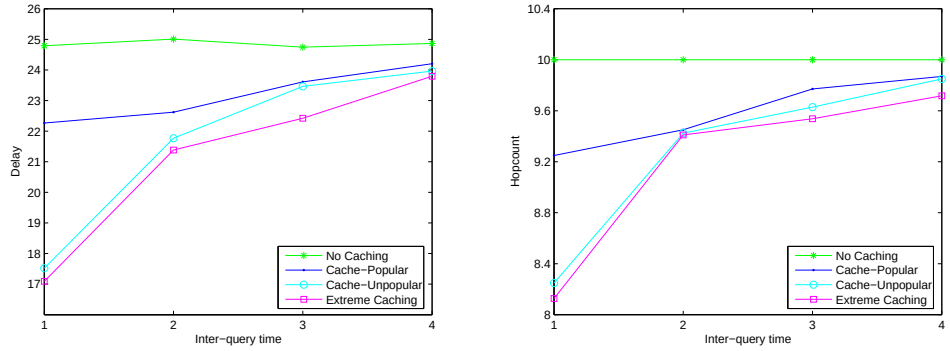


Figure 4.35: Delay τ (left) and hopcount κ (right) versus inter-query time for Ring topology with $r = 10$ rings and node degree $n = 4$, *Simplified RIP* routing and *Uniform* distribution

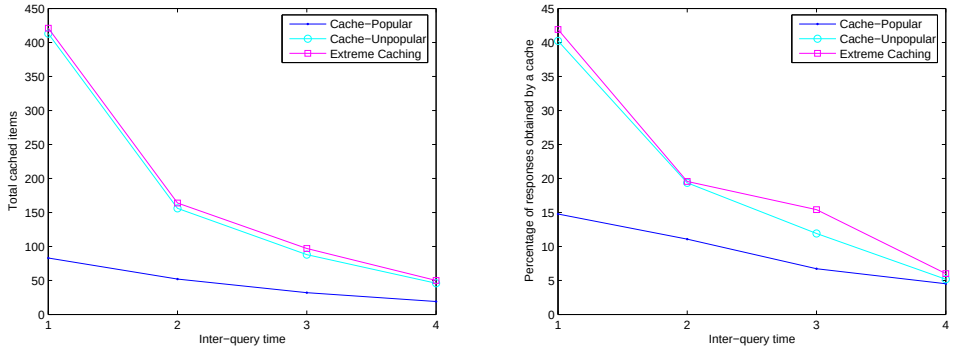


Figure 4.36: Total number of cached items (left) and percentage of responses obtained by a cache (right) versus inter-query time for Ring topology with $r = 10$ rings and node degree $n = 4$, *Simplified RIP* routing and *Uniform* distribution

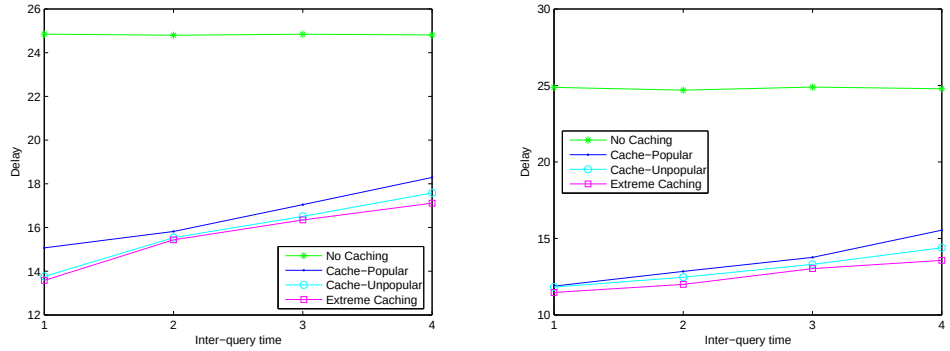


Figure 4.37: Delay τ versus inter-query time for Ring topology with $r = 10$ rings and node degree $n = 4$, *Simplified RIP* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

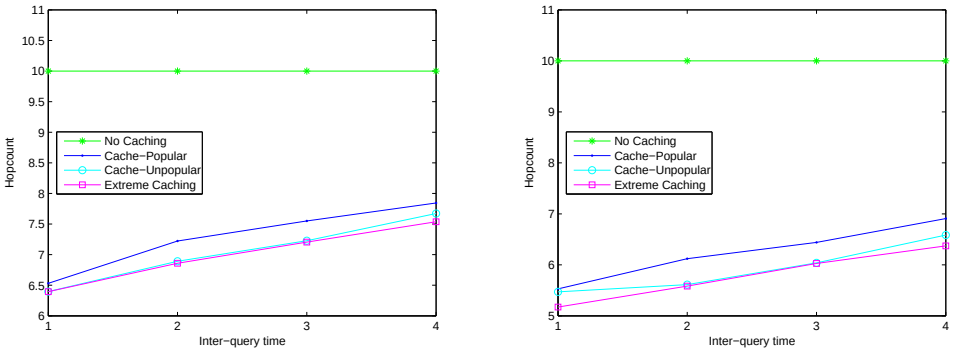


Figure 4.38: Hopcount κ versus inter-query time for Ring topology with $r = 10$ rings and node degree $n = 4$, *Simplified RIP* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

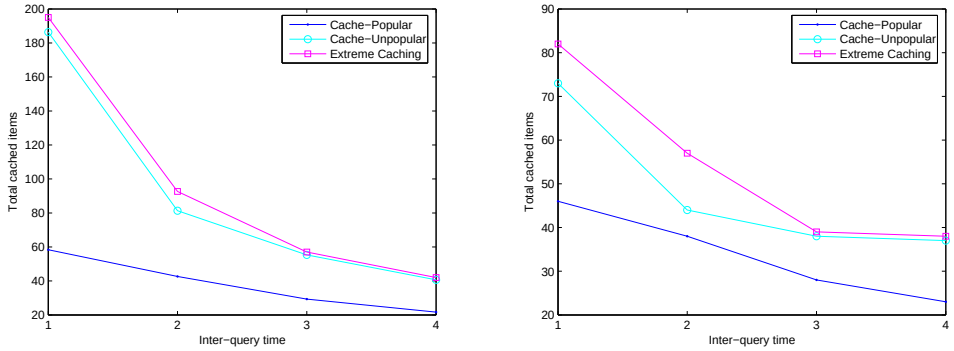


Figure 4.39: Total number of cached items versus inter-query time for Ring topology with $r = 10$ rings and node degree $n = 4$, *Simplified RIP* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

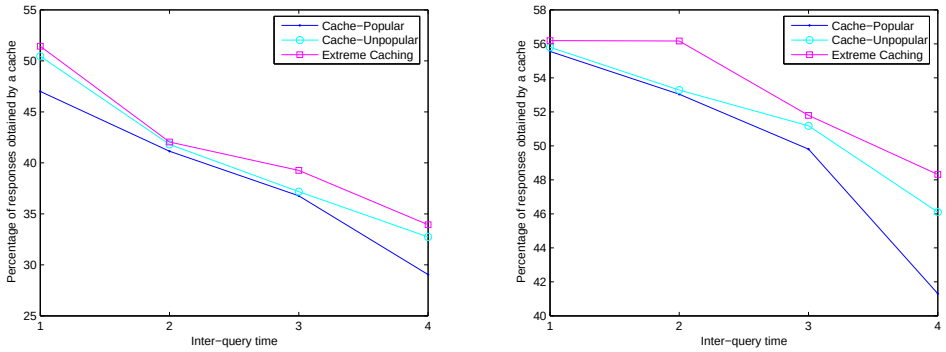


Figure 4.40: Percentage of responses obtained by a cache versus inter-query time for Ring topology with $r = 10$ rings and node degree $n = 4$, *Simplified RIP* routing and *Quadratic* distribution (left) or *Pareto* distribution (right)

4.5 Cache Maps

This section provides discussion on how the proposed system distributes the content in the network, i.e., which nodes are most probably chosen for caching. The following results are collected from the simulations that use the best performing cache policy (*Extreme Caching*) with Pareto query distribution in a Grid topology scenario. However, the behavior of other cache policies is similar from this point of view, even if they require less cached items on average. Similar results can be obtained also with a quadratic query distribution. On the other hand, uniform distribution requires a very high number of items cached, so that large amount of caching resources is required and this type of analysis is not reasonable. In the case of Ring topology, different clients do not share paths to the server and the requests that pass in an individual cache node are all coming from the same direction. As a consequence, the popularity values are the same for every intermediate node, so the items are cached with equal probability along the entire path.

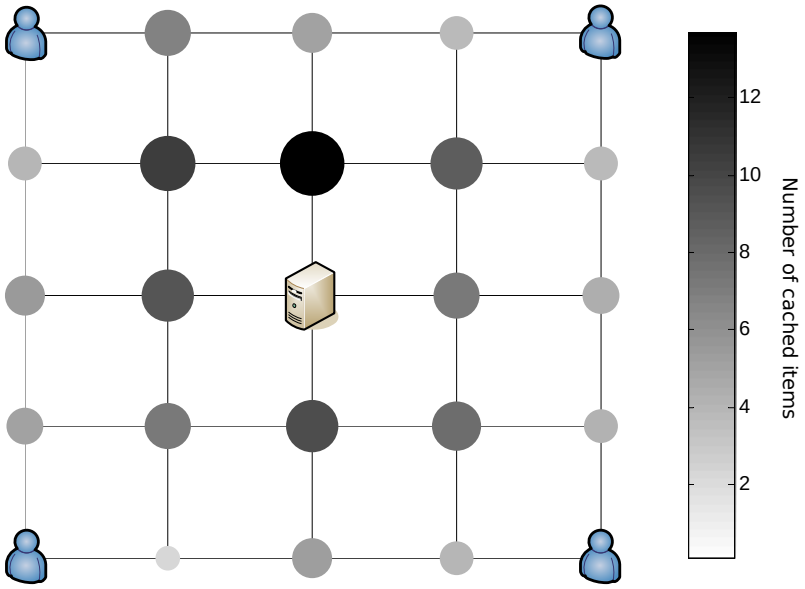


Figure 4.41: Distribution of cached items in the network, for Grid topology with parameter $d = 4$, *First Contact* routing, *Extreme Caching* policy and *Pareto* distribution

Figures 4.41 - 4.43 depict the network topology (as in Figure 3.2) and highlight the nodes, both with size and color of the glyph, that cache on average the highest amount of items. As explained in the previous sections, *First Contact* routing (Figure 4.41) requires to cache on average less items than *Binary Spray-and-Wait* (Figure 4.42). *SRIP* (Figure 4.43) minimizes the use of resources by exploiting the shortest paths, hence it requires the smallest amount of cached items. However, it stands out clearly that for every routing protocol used, the nodes which tend to be used mostly for caching are the ones closer to the server.

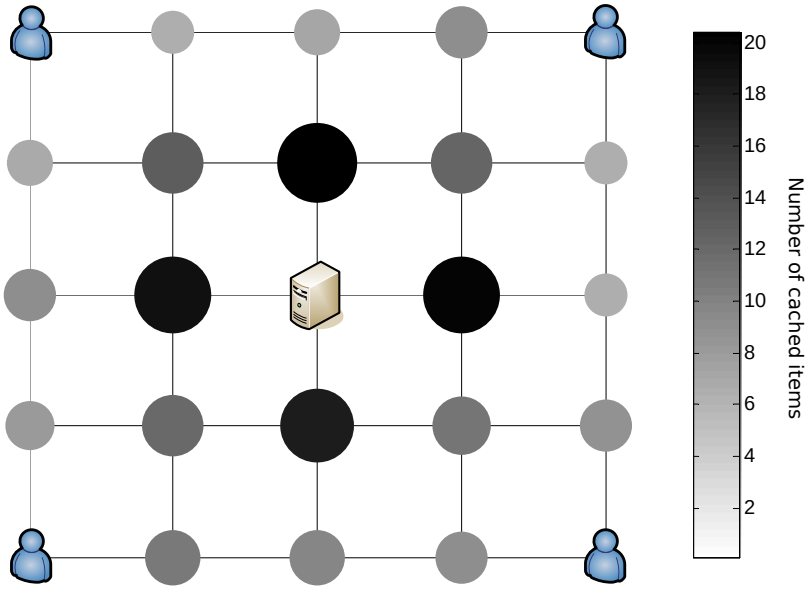


Figure 4.42: Distribution of cached items in the network, for Grid topology with parameter $d = 4$, *Spray-and-Wait* routing, *Extreme Caching* policy and *Pareto* distribution

In particular, the use of opportunistic DTN routing protocols results in the exploitation of all paths in all directions of the network. For this reason, the nodes within the same hop distance from the server are used for caching with the same probability. This probability remains higher as the hopcount distance to the server decreases. The behavior is particularly clear for *Spray-and-Wait*, where the cache map is symmetric and we can clearly identify different rings of nodes with equal amount of items cached.

On the other hand, the use of *SRIP* leads to a different behavior. In fact, the edge nodes are almost never used, while the central nodes within only one hop distance from the server are not used with the same probability. The nodes in the top-down direction cache a number of items notably higher than those in the left-right direction. This behavior is explained by the fact that those nodes are located on the last segment of the shortest path that is shared between clients that request the same items.

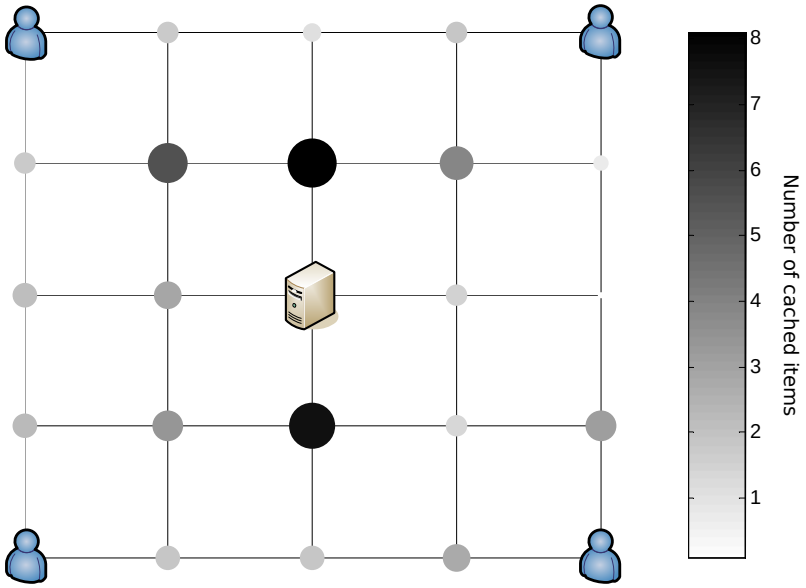


Figure 4.43: Distribution of cached items in the network, for Grid topology with parameter $d = 4$, *SRIP* routing, *Extreme Caching* policy and *Pareto* distribution

Chapter 5

Conclusions

A distributed caching system was presented in this thesis. Simulation results were shown to illustrate that the system helps to increase the overall retrieval performance, as well as reduce the delay and hopcount for obtaining the responses from the network. In particular, a policy that aggressively caches items (*Extreme Caching*) was shown to always give the best results. More conservative performance improvement is achieved by less aggressive policies (*Cache-Unpopular* and *Cache-Popular*). On the other hand, *Extreme Caching* results in high storage cost, as it requires the caching of high number of items around the network. For this reasons, this aggressive caching policy should be used only when the storage space is not a scarce resource. Moreover, when the probability distribution of the queries is uniform, applying caching does not yield to as good retrieval performance improvement as in the case of non-uniform query distributions. In latter case, Pareto distribution performs better than quadratic distribution. For example, a policy that caches popular items (*Cache-Popular*) together with uniform distribution of queries fails to obtain a significative improvement in all performance tests with DTN routing protocols, as it tends to behave similarly to the situation without caching. This happens mostly because the node tries to cache the most popular items between a large collection of equally popular items, so the caching decision is not always correct and consistent. Furthermore, when the query distribution is not uniform, caching less popular items

(*Cache-Unpopular*) performs much better than *Cache-Popular* in terms of increased efficiency and reduced delay. In fact, while *Cache-Popular* concentrates on caching only few popular items, *Cache-Unpopular* exploits the long tail, which overall comprises a bigger portion of the items.

First Contact routing protocol was used to explore performance of the caching system when random path discovery is used. This avoids distribution of multiple message copies in the network. However, it does not seem recommendable, as retrieval efficiency is mediocre at its best and applying caching mechanisms does not yield to considerable improvements. This result is seen in simulations with different topologies (Grid and Ring) which have varying node degrees. Moreover, retrieval performance in networks with large path lengths (Ring topology) is very poor, as only few requests are handled correctly. However, caching reduces the average delay of the responses. In this scenario, the delay has two different components. On one hand, the actual transmission delay is higher when there is more traffic in the network, because there are longer waiting times in the forwarding buffers. On the other hand, a higher number of cached items, together with increased traffic, helps to reduce retrieval delay by allowing the retrieval from intermediate cache nodes, which are physically closer to the requesting clients.

In the case of topologies with high number of shortest paths (Grid topology) and with uniform query distribution, caching does not give notable delay reduction and the traffic delay increase is very strong, while with non-uniform distributions the behavior is opposite. In networks with large path lengths (Ring topology), the absolute value of the delay is higher, so that the traffic component of the delay has only a small effect. The total number of items cached in the network decreases with the decreased amount of traffic in the network, and so does the percentage of responses obtained from cache nodes. However, the distance of replies, measured in terms of hopcount, increases with the decrease of the amount of traffic. For these reasons, single-copy routing with random walk (*First Contact*) should be used only where the retrieval latency is the primary goal, and it will be eventually possible to retransmit the request after a certain time if not successful.

Another routing protocol that spreads a limited number of message copies (*Binary Spray-and-Wait*) gives better retrieval performance results, especially when increasing the number of message copies, giving results not far from 100% efficiency. This allows to increase the explored portion of the network by varying the number of copies sprayed. Good performance of caching in terms of retrieval success has its drawback in terms of high delay, which exponentially increases when increasing the number of created copies. This is caused by the very high number of distributed messages in the network, especially in networks with large path lengths (Ring topology), which results in long queues in the forwarding buffers. For these reasons, *Binary Spray-and-Wait* should be used only when the priority is to retrieve correctly a large portion of the messages, in scenarios where the retrieval latency is not a key factor.

Epidemic routing allows to explore the entire network by flooding the query messages in a manner similar to broadcast. However, simulations showed that flooding presents the same performance improvement results as *Spray-and-Wait*, but it results in higher cost in terms of delay and amount of traffic. Hence, its use is not recommended.

The use of traditional routing schemes which use a single shortest path to destination (*SRIP*) gives the best overall performance, as delay and hopcount show the smallest values in all simulations, and retrieval success is always 100%. However, in a real-life DTN scenario it is not always possible to have the routing tables distributed in such a efficient way and, most importantly, the stability of the paths can vary. Nevertheless, at least some topological information should be used by the routing protocol, to efficiently forward queries and responses and to obtain acceptable performance.

Chapter 6

Future Work

A lot of research on caching has been done in the past, leading to numerous caching approaches. In this thesis we explore only some of them, and design and implement a distributed caching system. The proposed system serves as a basis for further work and improvements, where there is space for implementing new cache management mechanisms as networking technology evolves. For example, the popularity distribution of the queries appears to be a key factor for the caching performance. For this reason, new types of content distribution mechanism have to take into account the evolving distribution of items in the Internet, as well as alternative more sparse network topologies. Moreover, the models that describe clients' behavior can be improved, for example, to model more realistically the traffic and the inter-query times observed in the real Internet, with different peak traffic behavior at different times during the day, and for various time zones of the world. Furthermore, better congestion control techniques can be implemented, for example a mechanism of alternative custodians to deal with the problem of full forwarding buffers and caches, and mechanisms to handle the incoming requests when they exceed the link capacity.

Another important factor that an efficient caching system should take into account is the distance of the cache node from the client. In fact, a system based on popularity tends to cache the items in the nodes with the highest

traffic, which are typically near the server. In this case, the savings in terms of transmission cost are not always worth the cost of storage, and the system should be able to push the contents closer to the client. For this reason, nodes should be aware of their location in the network, and their distance both from the server and the client. The caching distance could serve as an additional input to the final caching decision, together with the content popularity. In addition, evaluating the tradeoff between the incurred storage cost and the retrieval performance improvement from caching seems an interesting challenge. This can help to gain deeper understanding about which caching policy has the best outcome.

Finally, a distributed caching system should also take into account special needs of mobile DTNs, where the use of opportunistic routing protocols is a more reasonable approach. Moreover, running the proposed caching system in the real Internet requires an overlay architecture which needs to be further investigated.

Bibliography

[1] Anderson, C. The long tail. *Wired Magazine 12*, 10 (2004).

[2] Asokan, N., Kostiainen, K., Ginzboorg, P., Ott, J., and Luo, C. Applicability of Identity-Based Cryptography for Disruption-Tolerant Networking. In *MobiOpp '07: Proceedings of the 1st international MobiSys workshop on Mobile opportunistic networking* (New York, NY, USA, 2007), ACM, pp. 52–56.

[3] Barish, G., and Obraczka, K. World Wide Web Caching: Trends and Techniques. *Communications Magazine 38* (May 2000), 178–184.

[4] Breslau, L., Cao, P., Fan, L., Phillips, G., and Shenker, S. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM* (1999), pp. 126–134.

[5] Cao, G., Yin, L., and Das, C. R. Cooperative Cache-Based Data Access in Ad Hoc Networks. *IEEE Computer Magazine 37*, 2 (2004), 32–39.

[6] Chankhunthod, A., Danzig, P. B., Neerdaels, C., Schwartz, M. F., and Worrell, K. J. A hierarchical internet object cache. In *In Proceedings of the 1996 Usenix Technical Conference* (January 1996), pp. 153–163.

[7] Chuah, M. C., and Yang, P. Performance Evaluation of Content-Based Information Retrieval Schemes for DTNs. Tech. rep., Department of Computer Science & Engineering, Lehigh University, 2008.

[8] Fall, K. A delay-tolerant network architecture for challenged internets. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2003), ACM, pp. 27–34.

[9] Fan, L., Cao, P., Almeida, J., and Broder, A. Z. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw. 8*, 3 (2000), 281–293.

[10] Gwertzman, J., and Seltzer, M. The case for geographical push-caching. pp. 51–55.

[11] Gwertzman, J., and Seltzer, M. World wide web cache consistency. In *Proceedings of the 1996 Usenix Technical Conference* (January 1996).

[12] Jain, S., Fall, K., and Patra, R. Routing in a delay tolerant network. *SIGCOMM Comput. Commun. Rev. 34*, 4 (2004), 145–158.

[13] Keränen, A., Ott, J., and Kärkkäinen, T. The one simulator for dtn protocol evaluation. In *SIMUTools '09: Proceeding of the 2nd International Conference on Simulation Tools and Techniques* (New York, NY, USA, 2009), ACM.

[14] Malkin, G. *Routing Information Protocol (RIP), version 2.* United States, 1998. RFC 2453.

[15] Michel, S., Nguyen, K., Rosenstein, A., Zhang, L., Floyd, S., and Jacobson, V. Adaptive web caching: towards a new global caching architecture. *Computer Networks and ISDN Systems 30*, 22-23 (1998), 2169 – 2177.

[16] Ott, J., and Pitkänen, M. J. DTN-Based Content Storage and Retrieval. In *First WoWMoM Workshop on Autonomic and Opportunistic Communications (AOC)* (2007).

[17] Pitkänen, M., Kärkkäinen, T., Greifenberg, J., and Ott, J. Searching for Content in Mobile DTNs. In *7th Annual IEEE Interna-*

tional Conference on Pervasive Computing and Communications PerCom (2009) (March 2009).

[18] Pitkänen, M. J., Keränen, A., and Ott, J. Message Fragmentation in Opportunistic DTNs. In *Second WoWMoM Workshop on Autonomic and Opportunistic Communications (AOC)* (2008).

[19] Pitkänen, M. J., and Ott, J. Redundancy and Distributed Caching in mobile DTNs. In *MobiArch '07: Proceedings of 2nd ACM/IEEE international workshop on Mobility in the evolving internet architecture* (New York, NY, USA, 2007), ACM, pp. 1–7.

[20] Pitkänen, M. J., and Ott, J. Enabling Opportunistic Storage for Mobile DTNs. *Elsevier Journal on Pervasive and Mobile Computing 4* (2008), 579–594.

[21] Pouwelse, J., Garbacki, P., Epema, D., and Sips, H. The bittorrent p2p file-sharing system: Measurements and analysis. In *In Proc. of the 4th International Workshop on Peer-to-Peer Systems (IPTPS'05)* (feb 2005), vol. 3640, LNCS, pp. 205–216.

[22] Seligman, M., Fall, K., and Mundur, P. Alternative custodians for Congestion Control in Delay Tolerant Networks. In *CHANTS '06: Proceedings of the 2006 SIGCOMM workshop on Challenged networks* (New York, NY, USA, 2006), ACM, pp. 229–236.

[23] Sherman, A., Lisiecki, P. A., Berkheimer, A., and Wein, J. Acms: the akamai configuration management system. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation* (Berkeley, CA, USA, 2005), USENIX Association, pp. 245–258.

[24] Solis, J., Asokan, N., Kostiainen, K., Ginzboorg, P., and Ott, J. Controlling Resource Hogs in Delay-Tolerant Networks. Tech. rep., Nokia Research Center, 2008.

[25] Spyropoulos, T., Psounis, K., and Raghavendra, C. S. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *WDTN '05: Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking* (New York, NY, USA, 2005), ACM, pp. 252–259.

[26] Su, A.-J., Choffnes, D. R., Kuzmanovic, A., and Bustamante, F. E. Drafting behind akamai (travelocity-based detouring). In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2006), ACM, pp. 435–446.

[27] Vahdat, A., and Becker, D. Epidemic routing for partially connected ad hoc networks. Tech. rep., Department of Computer Science, Duke University, Durham, NC 27708, 2000.

[28] Wang, J. A survey of web caching schemes for the internet. *ACM Computer Communication Review 29* (1999), 36–46.

[29] Wang, Z., and Crowcroft, J. Cachemesh: a distributed cache system for world wide web. In *Second International Web Caching Workshop* (1997).

[30] Wessels, D., and Claffy, K. *Internet Cache Protocol (ICP), version 2*. United States, 1997. RFC 2186.

[31] Yang, J., Wang, W., Muntz, R., and Wang, J. Access driven web caching. Tech. rep., UCLA, 1999.

[32] Yang, P., and Chuah, M. C. Performance Evaluations of Data-Centric Information Retrieval Schemes for DTNs. *Computer Networks 53*, 4 (2009), 541 – 555. Content Distribution Infrastructures for Community Networks.

Acknowledgements

Finally it came the day I have been waiting for a long time. Sometimes I wished it will never come, some other times I looked forward to it. This day, the day of my Graduation, is for sure marking a milestone in my life, splitting it into two parts. The future is yet to come, but what I certainly know is that I have enjoyed all my student life, absolutely nothing to regret, mostly because I was lucky that all the people I found on my way - from Italy to Finland - have been treating me the best way possible. I want to spend some lines of ink, while i shed a tear, to say thank you to everybody.

First of all, I would like to thank Professor JÖRG OTT for giving me the chance to join his research group and supervising my Master's Thesis. He has always been helpful and propositive, always with a smile on his face, really the best *Boss* possible. Additionally, I'd like to thank MIKKO PITKÄNEN for being my Instructor, always helpful to solve the daily problems with the work, and suggesting ideas, even from long distance and during the weekends. I am also grateful to TEEMU and all the other guys of the Lab for the precious help provided. But the person at TKK to whom I'll never be grateful enough is ANITA BISI who helped me not only with all the technical and bureaucratic stuff, but was spending her free time to come with me looking for houses to rent when I didn't have any. This is something I will never forget.

But if I am now here, at this momentum of my career, I have to give all to my family. My mum MARIA and my father ANTONIO have spent their life with the purpose of making mine better, always approving my good projects and denying what was good to deny, encouraging me to always improve and be the best possible, and also giving the chance and financial basis to take a plane and go to study in Finland. They have really been the best parents possible. I also thank for everything aunt ANGELINA, the most generous person in the world, my godfather TEMISTOCLE and nanny ROSARIA, for all they have always given me.

A really special thank goes to the second half of my heart, LAURA, who is simply the best girl in the world and even from thousand kilometers away has always given me the love I needed to go along. This thesis is dedicated to her, my beloved girlfriend, whose love made everything possible.

Also, a special mention for my friends I left in Italy but never stopped giving me support, and eventually came to visit me more than once: ALESSIA and LAURA T. for sharing with me a very special lasting friendship, LAURETTA, MARCO and FERRA for being precious buddies of travels and adventures; the old friends from high school DIEGO, DAVIDE, FEDE, IGOR and TRISO; all the schoolmates from the Politecnico di Milano especially LUCA with whom I have spent years in the lessons and labs; VALENTINA for being always so harshly sincere, and all the guys from the SCUDETTOWEB forum for constantly keeping me company night and day and not make me feel alone.

And last, but not least, I really want to give a big thank to all the people with whom I have shared the best moments of my two years in Finland. First of all, I have to thank RIIKKA who has been more than a special friend for me, always helping me when I was in need, even hosting me in her living room for long time when I didn't have any shelter. I also thank my best *kaverit*: FABIAN the best roommate and partymate, STEVAN for all the funny times spent together, EVA for always being such a nice company, SALLA the most reliable person I have ever met, VILLE the wise guy, MACIEJ, TINDARO, CARLOS, MAGDA, SYLVAIN and the other 300 Erasmus guys and girls I happened to meet in these months. Kiitos everybody, from the heart.

Helsinki, May 25th 2009

Stefano Gagliardi