NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

RAFAEL MARQUES SEQUEIRA

BSc in Computer Science

GENERAL PURPOSE SEARCH IN LARGE-SCALE UNSTRUCTURED OVERLAY NETWORKS

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
September, 2024

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

GENERAL PURPOSE SEARCH IN LARGE-SCALE UNSTRUCTURED OVERLAY NETWORKS

RAFAEL MARQUES SEQUEIRA

BSc in Computer Science

Adviser: João Carlos Antunes Leitão
Associate Professor, NOVA University Lisbon

Examination Committee

Chair: Bernardo Parente Coutinho Fernandes Toninho
Associate Professor, NOVA University Lisbon

Rapporteur: Francisco Almeida Maia
Affiliate Researcher at INESC TEC and Universidade do Minho

Member: João Carlos Antunes Leitão
Associate Professor, NOVA University Lisbon

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
September, 2024

General purpose search in large-scale unstructured overlay networks

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.1.29) [30].

To my family and friends, for their unconditional love and support.

Acknowledgements

This work would not have been possible without the support of those with whom I shared this journey. First and foremost, I would like to express my deepest gratitude to my adviser, João Leitão, for his invaluable guidance, patience, and support, which made this work possible.

I would also like to thank my colleagues and friends, such as Pedro Camponês, Pedro Ákos Costa, and all the other PhD students who worked alongside me. Their collaboration created a motivating and inspiring environment where I could freely exchange ideas and learn from their expertise. They were always available to answer my not-so-brilliant questions or offer personal advice, for which I am sincerely grateful.

Finally, I want to extend my heartfelt thanks to my family and close friends for their unwavering emotional support, wise life advice, and constant encouragement. Your presence helped me stay focused and motivated, even during the toughest moments.

Abstract

Centralized cloud computing solutions require complete trust in providers, which also constitute single points of failure for applications and users. Decentralized systems, such as the peer-to-peer data storage system InterPlanetary File System (IPFS), offer an alternative by distributing trust and responsibility among participants. While these systems have gained attention as a solution to the limitations of centralized cloud infrastructure, they still face significant challenges that can hinder broader adoption.

One of the key challenges is the efficiency of search mechanisms, which are crucial for locating resources (i.e., files) within the system. These mechanisms impact both the user experience and the scalability of the system. In peer-to-peer systems, searching is typically performed using distributed hash tables (DHTs) or by disseminating queries across unstructured overlays, with responses returned by peers storing the desired resources. In the latter case, which is the focus of this work, various strategies have been proposed to limit the search scope and guide queries toward peers that are more likely to possess the requested resources. These approaches seek to balance the overhead caused by messages, the success of resource discovery, and the search latency.

In this work, we propose a new indexing layer built on top of the existing IPFS unstructured overlay network, designed to share two types of information: indexes and a meta-indexes. This information is used to reduce search costs and latency while expanding the search horizon, thereby improving resource discovery efficiency. To validate the correctness and access the performance of our solution, we compared our prototype with the state-of-the-art IPFS search mechanism in an emulated distributed network, showing significant improvements in the latency, search cost and success rate when the DHT is not employed during searches.

Keywords: Peer-to-Peer Systems, Unstructured Overlay, Informed Search

Resumo

As soluções centralizadas oferecidas pela computação na nuvem exigem que se deposite total confiança nos seus provedores, tornando-se também assim pontos únicos de falha. Sistemas descentralizados, como o InterPlanetary File System (IPFS), que implementa uma rede de armazenamento de dados entre pares, oferecem uma alternativa ao distribuir a confiança e a responsabilidade entre os seus participantes. Embora estes sistemas tenham ganho popularidade como solução para as limitações da infraestrutura centralizada, enfrentam desafios significativos que constituem uma barreira à sua adoção em larga escala.

Um dos principais desafios é a eficiência dos mecanismos de pesquisa, fundamentais para localizar recursos (por exemplo, ficheiros) no sistema. Estes mecanismos afetam tanto a experiência do utilizador como a escalabilidade do sistema. Em sistemas entre pares, a pesquisa é tipicamente realizada através de tabelas de dispersão distribuídas (DHTs, do inglês *distributed hash tables*) ou pela disseminação de mensagens em redes sobrepostas não estruturadas, com respostas fornecidas pelos pares que possuem os recursos desejados. No caso da pesquisa em redes não estruturadas, foco deste trabalho, várias estratégias têm sido propostas para limitar o alcance da pesquisa e guiar as mensagens para os pares com maior probabilidade de possuir os recursos procurados. Estas abordagens procuram equilibrar o custo, a taxa de sucesso e a latência das pesquisas.

Neste trabalho, propomos a introdução de uma nova camada de indexação construída sobre a rede sobreposta não estruturada do IPFS, projetada para partilhar dois tipos de informação: índices e meta-índices. Esta informação é utilizada para reduzir os custos e a latência da pesquisa, mas também para expandir o seu horizonte, melhorando a eficiência na descoberta de recursos. Para validar a correção e avaliar o desempenho da nossa solução, comparamos o nosso protótipo com o mecanismo de pesquisa adotado pelo IPFS numa rede distribuída emulada. Os resultados mostram melhorias significativas na latência, no custo da pesquisa e na taxa de sucesso quando a DHT não é utilizada durante as pesquisas.

Palavras-chave: Sistemas entre pares, Rede não estruturada, Pesquisa informada

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Over the past decade, cloud computing has become the dominant deployment model for Internet applications and services. The fundamental principle of cloud computing is that resources are centralized in data centers managed by a central entity, which rents out portions of these resources to clients on an on-demand basis over the Internet [3]. This paradigm provides several advantages, such as a reliable and scalable environment where applications can dynamically expand their resource allocation as needed. Additionally, cloud computing shifts the responsibility of acquiring and managing infrastructure from clients to providers, who own and operate both the software and hardware. However, with increasing reliance on these centralized systems, concerns regarding trust and control have intensified. Trust is placed entirely in the hands of cloud providers, who typically exercise unilateral control over service terms, enjoy privileged access to user data, and effectively serve as a single point of failure [3].

In recent years, a movement known as Web3 [25] has emerged, challenging the centralized cloud model by advocating for decentralized approaches to the architecture of Internet applications and services. This movement has gained considerable attention due to its promise of democratizing access to data and computation, which has materialized in the development of decentralized systems. A prime example is the InterPlanetary File System (IPFS) [6], a peer-to-peer (P2P) data storage system, which serves as the foundation for the work presented in this thesis.

In P2P systems, participants (or peers) act as both clients and servers. These systems form and maintain overlay networks, which enable message exchange to coordinate activities and collaboratively deliver services that are typically utilized by all members of the system. The organization of peers within the network (i.e., how the overlay network is constructed and maintained) often has a direct impact on the implementation of these services.

Due to the large scale of P2P networks, it is impractical for each peer to maintain direct connections with every other peer. Instead, peers establish connections with a subset of participants, a concept known as partial membership [9]. The criteria for establishing these connections help categorize overlay networks. For example, structured overlays,

such as distributed hash tables (DHTs) [36, 42, 33], impose strict rules on connection patterns, while unstructured overlays rely on the random connections [47].

A key service in P2P networks is search, which allows participants to locate resources being available or shared by elements of the system. Search mechanisms are closely tied to the underlying structure of the overlay network. In structured overlays like DHTs, searches follow deterministic routing paths based on predefined rules to reach the peer responsible for the desired resource. In unstructured overlays, search queries propagate through the network, with peers possessing the relevant resource responding to queries. The openness of unstructured networks has led to the development of a variety of search strategies, each with its own policies for selecting peers or imposing limits on the horizon of the searches, making search a complex and compelling challenge [47, 24, 53].

Although IPFS utilizes a multi-level overlay network, incorporating both structured and unstructured overlays, this work will focus solely on the unstructured layer, where the primary mechanism for search in IPFS resides.

1.1 Motivation

For decentralized systems to become a viable alternative to centralized solutions, they must provide services of comparable quality. This is particularly challenging in P2P architectures, where decentralized control introduces significant and daunting obstacles. Overcoming these challenges is crucial if such systems are to fulfill their potential.

A suboptimal search mechanism can degrade the user experience by either failing to locate resources or introducing excessive delays. Moreover, inefficient searches that generate excessive network traffic can saturate the overlay network and overwhelm peers, ultimately compromising the system's scalability. Therefore, designing an efficient and scalable search service is essential to the success of decentralized systems.

On one hand, the emergence of Web3 has brought renewed attention to peer-to-peer systems like IPFS, once again making search a critical component. On the other hand, the Web3 ecosystem is still in its early stages, and systems like IPFS face several unresolved challenges. Specifically, IPFS's reliance on distributed hash tables (DHTs) for search, while effective in certain contexts, results in increased costs and latency. Additionally, it fails to fully leverage the potential of its unstructured overlay. These limitations present a valuable opportunity for exploring and improving unstructured search algorithms, with the goal of enhancing performance and reducing overhead in decentralized systems.

1.2 Contributions

The main contributions of this thesis can be summarized as follows:

- The design and implementation of a new layer built on top of an existing unstructured overlay network, along with a novel search mechanism that leverages the

information shared among peers to perform informed searches. Additionally, a second mechanism is introduced, which, through a collaborative process that leverages that same information, expands the search horizon without requiring nodes to further disseminate the query.

- The design and implementation of an enhanced version of the proposed solution, introducing a second layer of information (derived from the first), further optimizing the search mechanism.
- An experimental evaluation comparing both versions and operation modes of the proposed solution comparing, as a baseline, the search protocol of IPFS's unstructured overlay across multiple scenarios, validating the efficiency and effectiveness of our approach.

1.2.1 Publications

Part of the work presented in this dissertation resulted in the following publication [41]:

- *Pesquisa Descentralizada: Proposta de otimização da solução não estruturada do IPFS*

  Rafael Sequeira, Pedro Ákos Costa and João Leitão

  Proceedings of the 15th Simpósio de Informática (INForum 2024), Lisbon, Portugal, September 2024.

1.3 Document Structure

The remainder of this document is organized as follows:

Chapter 2 discusses the related work. We begin by presenting the concept of overlay networks in more detail, covering multiple architectures and exploring their characteristics, along with how they impact the system and their relation towards the objective of this work. We then review various search mechanisms, presenting and categorizing some classical approaches while evaluating the trade-offs between them. Finally, we examine two real systems in detail to provide a comprehensive view of search functionalities in decentralized systems.

Chapter 3 presents the starting point of our solution (the baseline search mechanism) and details the implementation of our proposed solution.

Chapter 4 outlines the methodology used in the experimental component of this work and presents the results, followed by a discussion of these results.

Chapter 5 concludes the thesis by summarizing the main findings and proposing directions for future work based on our solution.

2

Related Work

In this chapter, we present and discuss the work relevant to this thesis. We will consider a general system architecture, illustrated in Figure 2.1, which consists of an overlay network that supports various services, both of which are shaped by the system's intended application. Services enable systems to achieve some of their goals by providing adequate abstractions, and, being built on top of it, also affect the underlying network. Ultimately, search is a service integrated into a system, and so, depends on its overlay and must be tailored to its needs. However, even without considering this complex symbiosis between these layers, searching is a general and intricate topic by itself.

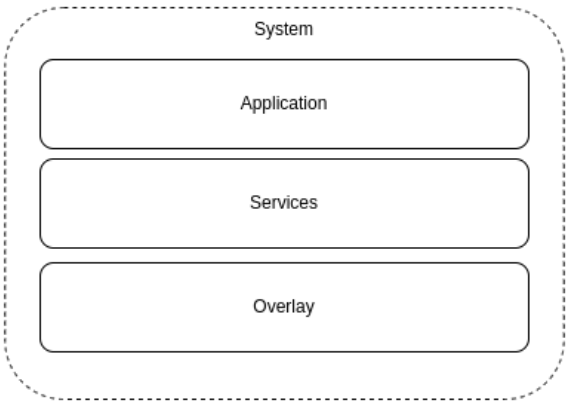


Figure 2.1: Generic system architecture

In the remainder of this chapter, we will explore the diversity inherent to searching but also present, at glance, the underlying mechanisms and how they interact with each other, culminating in some examples of real systems that leverage such services to achieve their objectives. The rest of this chapter is organized as follows:

In Section 2.1, we present the concept of overlay networks, which are part of the infrastructure, categorizing them according to their structure and hierarchical (or lack of) organization but also exploring their characteristics and how they affect the system and their relevance to our objectives.

In Section 2.2, we examine the gossip communication patterns which are usually employed in these systems to disseminate messages which include the ones used to

perform searches (i.e., queries).

After that, in Section 2.3, we study how searches can be performed in a decentralized way, how one can compare their performance defining a base to further explore them, and, lastly, we discuss some classical approaches discerning the existing trade-offs between them.

Finally, in Section 2.4 we will detail real systems to assist the reader in fully grasping the details of the operation of systems relying on decentralized search.

2.1 Overlay Networks

An overlay network is a logical network built on top of another network, usually abstracting the participants of the details of the underlying infrastructure. In peer-to-peer (P2P) systems, nodes (also referred to as peers) form these networks to facilitate communication, with P2P services being built directly on top of the overlay [46]. Applications then leverage these services to implement their desired functionalities.

Overlays are formed through a decentralized protocol known as membership protocol which establishes and maintains connections (links) between nodes. In large-scale systems, however, peers typically don't know all other peers in the system (known as full membership) but only a subset of nodes, which is usually referred to as partial membership [9]. The nodes within this subset are commonly referred to as neighbors, and the set of neighbors of a node constitute a partial view of that node.

The motivation behind adopting the partial memberships comes from the fact that, first, a full membership would require many resources from each node and the system would have limited scalability. For instance, each peer would have to store information about all others. As the system grows, the amount of information stored by each peer would increase linearly. Since peers have limited resources, a limit on how many peers can participate in the system would need to be imposed. Second, and more importantly, peers may join or leave the network as they please (and without further notice). The changes caused by this behavior are known as churn and, combined with other membership changes, greatly increase the cost of maintaining the full information at the system membership updated.

Finally, it is important to note that overlays in P2P systems are inherently dynamic. Even the partial view of each node is subject to constant change due to churn and other factors. These dynamics have a significant impact on the whole stack and will be discussed later.

The similarities between overlays and graphs are obvious. If nodes are perceived as vertices, the connections can be seen as edges and the overlay network as a graph. Furthermore, graph properties can be used as metrics to further describe and asset the properties of overlay networks, including studying their performance. We will consider the following graph properties [26]:

Connectivity Similar to a connected graph, each node in an overlay must have at least

one path to all other nodes. If this condition is not met, nodes or portions of the overlay can become isolated and are unable to receive messages from the rest and hence, collaborate.

Degree Distribution The degree of a node is defined as the number of connections established with other nodes. However, these connections may not be bidirectional, effectively forming a directed graph. In such cases, the number of directed edges is distinguished as **in-degree** and **out-degree**. The degree distribution provides a measure of node participation in message dissemination. Nodes with a high in-degree are highly reachable, while nodes with a high out-degree can efficiently contact numerous other nodes.

Average Shortest Path A path is the group of edges that a message must traverse to be delivered from one node to another. There may be several available paths to accomplish this. The average shortest path is the average of all smallest (in terms of length) paths between every pair of nodes and is closely related to the overlay diameter. It's important to note that this value should be as small as possible since having fewer intermediaries makes message dissemination more efficient.

Clustering Coefficient The clustering coefficient is used to infer the density of neighbor relations across the neighbors of a given node. It is obtained from the number of nodes' connections (the number of neighbors) divided by the maximum amount of links that could exist across those neighbors. High values of clustering can indicate that messages are not disseminated well in the overlay, producing many redundant messages. Furthermore, it is also related to fault tolerance properties of the graph since nodes that are tightly connected have less diversity in their pool of connections, and are more susceptible to becoming isolated.

How the neighbor relations are formed, may vary, and the strategy employed by protocol defines the network topology, which directly impacts the services implemented on top of the overlay.

Overlays are typically categorized as structured or unstructured. In this work, we include a third category, biased overlays, which we discuss in detail later.

2.1.1 Unstructured Overlay Network

In unstructured overlays, connections among peers are established randomly or without following strict rules. As a result, the processes of node arrival and departure are straightforward, and membership changes incur minimal overhead [34]. Consequently, these networks can tolerate high levels of churn and have low maintenance costs [5].

Unstructured overlays tend to form topologies with properties that make them resilient to node departures and failures. These networks typically remain functional (i.e., connected) even when a significant portion of nodes is removed, either randomly or selectively.

These characteristics make them particularly suitable for highly dynamic environments and systems such as Gnutella, which is discussed in Section 2.4.

However, the random nature of unstructured topologies poses challenges for services that cannot effectively leverage them. For example, reaching a specific node efficiently becomes difficult since there is no predictable structure to indicate which nodes are connected. Furthermore, when a node needs to communicate with a specific peer that is not directly connected, messages must be propagated through the network by gossiping, a mechanism explored in more detail in Section 2.2.

2.1.2 Structured Overlay Networks

Structured overlays try to implement more efficient communication by orchestrating a predetermined network topology. This is achieved by using nodes' identifiers to define their position on the network. Subsequently, messages can be routed to a known target by having each node forwarding them to nodes that are successively closer.

The topologies employed by structured overlays vary, with some systems adopting ring-like configurations [42], trees [33], or multidimensional spaces [36].

The main principle leveraged by this class of overlay can be extrapolated to other uses cases or P2P services. For instance, one can assign specific resources to predetermined nodes according to a resource identifier. For example, specific resources can be assigned to predetermined nodes based on a resource identifier. Any node that knows the identifier can also determine the location of the responsible node and access the resource. This concept forms the basis of a *Distributed Hash Table* (DHT), as implemented by systems such as CAN [36], Chord [42], and Kademlia [33], among others [54, 39]. Since peers can locate resources in the overlay *a priori*, DHTs effectively function as a distributed global index, enabling efficient resource discovery [47].

Nevertheless, structured overlay networks face additional challenges. When joining the network, a node must bootstrap itself into the correct position which produces a significant amount of messages since other peers must be a part of this process. Node departures and other membership changes are also more complex to handle since the topology may have to be rearranged in a very specific way that involves multiple (specific) nodes. Overall, structured overlays tend to deal worst with churn and are more susceptible to membership dynamics.

2.1.3 Biased Overlay Networks

The third category, which we refer to as biased overlays, seeks to combine the two categories previously discussed. The key idea is that peers The core idea is that peers, as in unstructured overlays, initially connect at random, however, after cementing themselves in the network, try to improve their position using some heuristics to select better neighbors. Moreover, by having their participants following this decentralized process, the network

slowly converges to a topology with desired properties that can benefit the communication efficiency or, more specifically, the services [46].

It is noteworthy that the resulting topology is less rigid than that of a structured overlay and is, potentially, constantly evolving towards better configurations. Additionally, when nodes join the network, their final position is not predetermined, nor is it known by other peers at any point. As a result, message dissemination operates similarly to unstructured overlays, but with improvements gained from the biased structure. However, nodes require time to improve overlay links (and their relative positions) considering some particular criteria, and if the network experiences high churn, the topology may never stabilize.

The heuristics vary according to the needs of the services or applications implemented on top of the network. For instance, one could bias the node connections through links with low latency. Such an example is presented in X-BOT[29], a protocol that produces biased overlays through generic efficiency criteria derived from a local oracle. Another example is Gia[8], a proposal to improve the Gnutella network, where the network is biased through high capacity nodes that are more capable of dealing with higher amounts of incoming traffic.

2.1.4 Multilevel Overlay Networks

In contrast to flat overlays, where all peers operate on a single level, multilevel overlays introduce multiple levels to enhance functionality, creating a multidimensional structure. Peers can participate in more than one level, with each level having its own overlay structure and distinct strategies for neighbor selection. These levels can be hierarchical, with peers at higher levels holding more responsibility than those at lower levels.

It is essential to distinguish between multilevel overlays and stacked overlays, as the latter are also commonly used in network architectures. In stacked overlays, different services are implemented independently across multiple layers, with no direct interaction between the levels. For example, SONs [12] use semantic overlays, where peers join the overlay that aligns with their specific interests. While searches can be performed in parallel across multiple semantic overlays, there is no direct relationship between them.

The motivation for adopting multilevel architectures often stems from the need to address scalability issues in flat designs, improve message dissemination, or increase flexibility. However, multilevel overlays are more complex, and systems that implement these architectures often have to deal with synergy challenges between levels.

CAN [36] is an example of a DHT that implements a multidimensional coordinate space. Nodes position themselves within randomly assigned zones and establish connections with nearby nodes for routing. These zones are dynamic and are reshaped when nodes join or leave the system. CAN introduces the concept of "realities," where nodes can join multiple coordinate spaces (realities), each with an independent set of neighbors. While nodes may have different positions in each reality, resources remain fixed across

all realities. This approach enables more efficient routing, as nodes can select the reality with the closest neighbor to the destination.

Nevertheless, in multilevel overlays, levels are often more diverse. One approach to exploit advantages the benefits of different overlay structures is to build an overlay where some peers form connections in a structured manner but also participate in an unstructured or biased overlay, creating a hybrid system.

Alveirinho et al. [2] present a materialization of this idea. Peers participate in a biased overlay that enables them to establish connections with similar peers based on predefined categories. Then, a subset of peers is selected as representative of each category and participates in a DHT. When performing a search, nodes use the DHT to locate a representative which, if unable to fulfill the search, forwards their message to an area in the biased overlay that is very likely to address its needs. The localized search in the biased overlay it's expected to deliver results with a high probability.

IPFS [6] (presented in more detail in Section 2.4) also combines an unstructured overlay with a structured one and both are leveraged by the search (and data exchange) protocol [38]. The search is initially performed in the unstructured overlay, however, if this approach fails, the node performs a lookup in the DHT to find additional providers of the content being looked for (this introduces some additional latency).

Unlike the examples presented thus far, hierarchical levels can also be used to leverage peer heterogeneity. Furthermore, if only a subset of peers at a given level participates in one service the burden of a large overlay is significantly reduced. Super-peers is a well-known approach that is inspired by these principles. It was adopted by multiple early file-sharing applications such as Gnutella (detailed in Section 2.4), which initially used a flat overlay but adopted it on later versions, and the FastTrack network. However, the latter utilized a closed-source and property protocol that is not well studied (a general description is available in several works [35, 15, 34]). In its simplest form, this type of system works as follows. Nodes are divided into two categories: super-peers and peers. Peers connect to one or more super-peers. Super-peers index the content of peers that connect directly to them (which is known as aggregated indexes [47]). Searches are performed by disseminating the messages in a super-peer unstructured overlay. The selection of super-peers takes a key role in performance improvement and network stability. Furthermore, the departure of super-peers has a greater impact on the network than normal nodes' departure, so they must have high availability. Further, to be able to handle the additional responsibilities, they also must be high-capacity nodes (for instance, have more processing power or bandwidth than other nodes). However, peer selection usually doesn't solve the problem completely and super-peers can become overloaded, disrupting the system operation [28]. Some works address this problem directly. For instance, SOSPNet [17] employs active load-balancing mechanisms while also relaxing the aggregated index, with super-peers only indexing partially (and selectively, according to how much past queries it resolved) the peers' content. Additionally, this proposal tries to improve search performance through biased connections between peers and super-peers to capitalize on

interest locality.

For the remainder of this work, our focus will be on unstructured overlays. However, we may also consider multilevel unstructured networks and biased networks, as their search processes are similar to those used in flat unstructured overlays and can be used interchangeably.

2.2 Gossip Dissemination

In gossip-based communication protocols [26], nodes collaboratively disseminate messages, as each node is only directly connected to a limited number of other nodes (neighbors), thus having only a partial view of the system. The process works as follows. A node selects n, a configurable value commonly known as *fanout*, of neighbors as gossip targets. The message is sent to all gossip targets which repeat the same steps when they receive a message for the first time. Since nodes relations can form cycles, mechanisms that guarantee that the same message is not processed twice must be employed. For instance, storing all received messages for a limited amount of time and discarding duplicates.

Gossip can follow several strategies when comes to sharing information. In push approaches, nodes share information immediately after receiving it while in pull approaches neighbors periodically request information that it's shared with them when the source of it receives the request. Intuitively, hybrid approaches combine the two, adopting the first to quickly disseminate information and the second to recover lost updates. [9]

The message dissemination can be limited, defining a horizon in a time-to-live (TTL) fashion. Each time a message is forwarded, a hop counter is incremented. When the counter reaches the value specified by the TTL, nodes stop forwarding the message.

Lastly, it's important to note the trade-offs between different parameters and strategies adopted in gossip. Higher fanout values guarantee better delivery probabilities but increase the number of redundant messages in the network. Similarly, high (or unlimited) TTL have more reliability but also produce more redundant messages. Finally, pull strategies require an extra round-trip per message since nodes have to actively request them which introduces extra latency and require more memory from nodes that have to store it in order to honor pull requests.

2.3 Decentralized Search

Locating a given resource in an overlay is an essential service of a distributed system. Peer-to-peer file-sharing applications, which are arguably the base of most of the work studied, need to allow their users to search for files to accomplish their primary objective. However, search is a general problem that extends beyond file sharing. For instance, service discovery [35] and P2P-Grid systems [53] are also heavily dependent on effective search mechanisms. With this in mind, we will use the generic term *resources* to denote the targets of a given search whenever possible.

Search is the process by which a node identifies the location of a resource. How a resource is ultimately retrieved is orthogonal to the search process and falls outside the scope of this work. However, in cases like IPFS [6] and Bitswap [38] (discussed in Section 2.4), the distinction between search and retrieval can be somewhat ambiguous. We will address this specific case in more detail later. Additionally, we will consider that a search is concluded when a sufficient number of resource locations is found, or when the search is deemed to have failed. Defining search failure can be challenging, as it is highly dependent on the search mechanism itself, and this will also be explored further ahead.

Resources must be indexed to be discoverable, and in unstructured (and biased) overlays, there are two fundamental approaches to achieve this.

One (arguably naive) strategy is for each node to store the location of all resources in the network. In this scenario, nodes could simply consult their local global indexes to find the desired resource. The global index could be updated slowly through a gossiping mechanism and shared when new nodes join, to prevent overwhelming the network. However, similar to the global membership issues discussed in Section 2.1, the cost of maintaining such indexes is prohibitively high, especially under conditions of high churn, and it imposes limitations on system scalability. Consequently, large-scale systems typically do not adopt this approach, instead preferring to rely on local indexes.

A more scalable strategy, analogous to partial membership schemes, avoids global knowledge of resource locations, instead, nodes maintain local indexes with a limited number of resources.

When using local indexes, a search begins at a node in the system (the source node) and follows a gossip communication pattern (as described in Section 2.2). The source node formulates a query describing the desired resource and sends this query in a message to its gossip targets. Typically, a push-based strategy is employed. Upon receiving a search message, each node attempts to match the query description with the resources listed in its local index. It is important to consider that the resources indexed by each node may var across different solutions using this strategy. In its simplest form, known as strict local indexing [47], nodes index only their own resources. This approach has the distinct advantage of allowing nodes to build and maintain their indexes independently, without needing to coordinate with other nodes. However, the reduced availability of information regarding resource locations can lead to a higher number of messages being generated during the search process.

Apart from super-peer architectures, which were already discussed in Section 2.1.4, nodes typically index their own resources but may also index resources belonging to other nodes (e.g., their overlay neighbors or resources previously requested by then). This allows them to assist a searching node in locating a resource even if they cannot directly fulfill the query themselves.

2.3.1 Query

Nodes use queries to describe the resources that are looking for. Generally, they should be expressive enough, so one can narrow the number of resources that match it but not prohibitive when it comes to exploring what resources are available without knowing fine (or all) details. However, the supported queries varies from system to system depending on their capabilities and use cases. For instance, to support complex queries where one is able to describe resources properties nodes have to index such properties.

Ultimately, some overlays are more fitted to support certain types of queries than others. Nevertheless, unstructured and biased overlays are undoubtedly more versatile in this regard because nodes can index resources in a wide variety of manners locally and this information doesn't impact the network itself since it's not used to define their location. This said, structured overlay networks are more efficient for exact search queries, when a unique identifier describes the resource being looked up.

In general, queries can be categorized as exact match, keyword-based, ranges and arbitrary, as detailed below.

2.3.1.1 Exact Match Queries

On exact match queries, resources are fully described by a unique identifier. Furthermore, in searches that employ these queries, the target resource is known in advance and often only one resource it's expected to fulfill it.

As said before, this type of query is naturally fitted to DHTs, already presented in Section 2.1, since they operate in a key-value fashion but also serve as a base to content-centric networking [22, 4]. They usually require that applications obtain the unique identifier of the resource they need to access by some out-of-band process.

2.3.1.2 Keyword-based Queries

Keyword-based queries come from the idea that resources can be described or categorized by a set of keywords. Regardless, resources can share keywords therefore multiple resources can match a single query.

Additionally, logic operators (like "AND", "OR" or "NOT") are usually combined with keywords to improve further the flexibility of the queries.

Unstructured overlays (in contrast to structured ones) naturally support this type of query since, as stated before, a node only has to organize the local indexed resources according to a set of categories. Furthermore, this process can be performed locally and, although desirable, nodes don't need to agree on the categories. Additionally, some search mechanisms (mainly informed ones, which will be explored later) leverage resource categorization to improve performance.

2.3.1.3 Range Queries

Range queries allow the source of a search to define upper or lower limits on quantifiable properties of resources. The properties vary, but the key concept is that any resource that has a value contained in the interval matches the query.

Similarly to keyword-based queries, unstructured overlays are more suitable to address this type of query. In comparison, performing range queries on a DHT it's particularly challenging, with solutions usually requiring the query to be subdivided into multiple lookup operations and segmenting the structured topology on a property basis [16, 32].

2.3.1.4 Arbitrary Queries

Arbitrary queries allow more fine-grained descriptions by enabling combinations of the previous queries types and enabling queries to contain properties of the resource.

These properties vary on what nodes are able to index which, ultimately, is also dependent on the type of system. For example, in the context of file-sharing applications, the file size could be considered as property and in the limit, even the content of a given file could be used to describe it. Furthermore, properties don't need to be directly obtained from the resource. In file-sharing systems, nodes may attribute names or descriptions to each of their resources which could be (fully or partially) considered when processing the query.

2.3.2 Search Termination

A search finishes when satisfactory results are found, which is usually defined through a predetermined amount of resources. When more than this goal is achieved, the search algorithm incurs in *overshooting* hurting its efficiency [21]. However, the search can be terminated even if this goal is not met. A first and unusual scenario, is when a query reaches all peers in the system which means that matching resources don't exist. Nevertheless, in unstructured and biased overlays, it is very difficult to ensure that all nodes were reached by the query since the topology and the number of nodes in the system is unknown [53]. Another instance is when a node is not able to forward a query further. One such case can happen if some mechanism is in place so that nodes are not visited by queries multiple times. However, a backtracking mechanism, where the query returns to the previous node and is forwarded to a new node, may be employed to fix this issue. Checking can also be used by some search techniques (mostly random walks and variations). In checking, nodes that participate in the search periodically ask the source node if the query must be forwarded further. Lastly, and perhaps more common, the message reaches a TTL-based horizon and stops being forwarded.

Results are usually returned in one of two ways. In the first, the message is back-forward following the reverse sequence of nodes that visited (path) to reach the query originator which might be useful to update nodes information (in informed searches) or

replicate the results across the path. In the second, a direct connection between the two nodes (the node having the resource and the query originator) is extablished to convey results.

2.3.3 Search Performance

The overall performance of a search is influenced by several factors, and search algorithms are highly diverse, often adopting fundamentally different approaches. As a result, comparing them objectively can be challenging.

Underlying factors, such as the characteristics of the overlay network (see Section 2.1), have a direct impact on message dissemination [27], ultimately affecting search performance. For example, in networks with a high clustering coefficient (where nodes share many neighbors), messages with a limited scope may struggle to reach new nodes, thereby negatively impacting search performance. On the other hand, in networks with a low diameter, messages can reach many nodes, or more importantly, the correct nodes, with relatively few hops. Unstructured overlays may also exhibit a power-law degree distribution, characterized by a few highly connected nodes and many with fewer connections. Early studies suggested that Gnutella exhibited power-law characteristics [31, 37, 1], but later research [43, 44, 21] argued that it instead had small-world properties (high clustering coefficient and low average shortest path). Regardless, power-law distributions are known to affect aspects such as system resilience and search performance [31, 1]. For example, messages tend to gravitate around well-connected nodes, increasing their load, generating more overhead due to duplication, and making it more difficult for queries to explore new nodes.

Apart from overlay properties, the distribution of resources across nodes is also a critical factor. Naturally, popular resources are easier to locate. Moreover, if the distribution is uneven, with some nodes storing many resources, certain informed search algorithms (presented in Section 2.3.5) may exploit this characteristic to improve their performance.

However, the distribution and the number of replicas for each resource can vary depending on the system. When evaluating search algorithms, various distribution models may be considered.

Some systems actively leverage replication to improve search performance. In this work, we define replication as the process of copying a resource to additional nodes, enabling them to serve the resource and answer future queries. A simple form of replication in file-sharing systems occurs when nodes that retrieve a file later act as providers for that file. Such strategy is partially employed by IPFS, as nodes which obtain a resource use it to fulfil searches on its unstructured overlay (although it does not happen in the structured). However, more sophisticated replication schemes have also been explored [46].

Even with all the aspects mentioned above, and considering that some algorithms may be better fitted for particular systems or overlays topologies, the following metrics [27, 28] can be used to compare different decentralized search algorithms:

Number of results A search algorithm has the ultimate goal of finding resources. The number of results is the total number of results returned by a search. However, it fails to capture the search performance since it's possible to have algorithms that return a great number of resources but are highly inefficient or have a high cost.

Query Recall Rate Closely related to the number of results, query recall rate is the percentage of results returned divided in relation to the total number of resources in the system that matches that query.

Query Dissemination Cost Message Cost of a search is measured in the total number of messages that were transmitted in the process. Usually, algorithms with high dissemination costs provide better recall rates but may not be viable since requiring a lot of effort from peers and place much stress on the network. Concurrent searches aggravate this effect.

Query Processing Rate The processing rate is the percentage of nodes in the system that participate in a search. Ideally, this value should be kept as low as possible without harming the recall rate.

Query Hit Rate The query hit rate is used to measure the performance of search algorithms. It is obtained from the number of resources found divided by the dissemination cost.

Latency Latency is the time interval from when the search starts until it finishes. It is important because usually, a user expects low response times and high latency directly impacts the perceived quality of a search. However, because of the distributed nature of searches, the results may be returned asynchronously, with nodes that matched queries returning their results early. In such cases, to ensure justice, the first match may be considered.

Search algorithms are usually described accordingly to how messages are propagated by the peers, being divided into two greater groups depending on how much information is known about the location of a resource when forwarding a message [24, 53]. We will adopt this approach to further classify the techniques presented below.

2.3.4 Uninformed Search

In uninformed (or blind) search mechanisms, no information about the resources or the overlay is considered when forwarding a message [46, 24]. These mechanisms are often praised for their simplicity in terms of both strategy and implementation. Since no additional information is required, nodes do not need to store extra data or exchange messages to obtain it. As a result, blind algorithms do not create additional overhead in the network. Furthermore, due to the absence of dependency on overlay-specific information,

they are expected to remain unaffected by changes in the network, maintaining consistent performance regardless of how dynamic the overlay is.

All techniques in this category fall into one of two fundamental strategies: flooding or random walks. However, some algorithms combine the two, forming what will be described as hybrid approaches.

2.3.4.1 Flooding

Flooding, sometimes referred to as Breadth First Search (BFS), is a classic search technique where a peer, unable to fulfill the query itself, forwards it with a fanout equal to its number of neighbors. Flooding the network has several advantages over other search algorithms. Its primary advantage is that many nodes are queried after only a few hops, with this number increasing exponentially. Consequently, low latency is expected, and resources are found with a high probability, providing a high recall rate. In fact, flooding is inherently deterministic: if the query reaches the entire network (i.e., all nodes receive the query), resources will be found, resulting in a 100% recall rate. However, this approach has a high dissemination cost and generates significant overhead. A vast number of messages are produced, and concurrent searches can saturate the network, overloading nodes that must process them [31]. Moreover, after only a few hops, depending on the overlay properties, many messages become redundant due to natural cycles formed between nodes, limiting the scalability of the system [5].

To partially mitigate this issue, flooding-based techniques usually impose a horizon on the search through a (previously discussed) TTL value. However, limiting the search scope means the search is no longer complete, as only a portion of the nodes are queried. Choosing an ideal TTL is challenging: too high a value creates unnecessary burden on the network, while too low a value harms the recall rate. Furthermore, the optimal TTL depends on the network topology and resource replication ratios [31].

Several variations of flooding have been proposed. To reduce the number of generated messages, **Normalized Flooding** and **Probabilistic Flooding** (also known as Modified-BFS or Teeming) [23, 5] limit the fanout of gossip messages. In the former, fanout is restricted to a predefined maximum, while in the latter, a percentage of random neighbors is selected to forward queries.

Other approaches attempt to dynamically control the TTL to resolve flooding issues without sacrificing recall rates. **Expanding Ring Search** (ERS) [5] performs several rounds of flooding. In each round, which is generally triggered by a timeout when queries are not replied, the search horizon is expanded by defining larger TTL values in the successive rounds. **Blocking Expanding Ring Search** [5] improves upon ERS by starting each round from where the previous stopped. Similarly, **Iterative Deepening** [52] accomplishes the same by making nodes at the border of one round store the query temporally. The source peer initiates another round when is not satisfied with the results and the nodes that participated in the previous round simply forward the request to border nodes that

initiate a new flooding round.

In the measurements performed in a Gnutella topology [5], it was found that BERS and ERS, have better latency than flooding and probabilistic flooding with BERS having slightly better results. But, in comparison, probabilistic flooding has a higher hit rate and produces fewer redundant messages. Such results can be explained by the fact that with each dissemination step executed for a query the number of nodes participating in the search increases exponentially and a lot of messages are created. In contrast, in probabilistic flooding, the number of reached nodes increases constantly and in a controlled manner. Furthermore, this characteristic may also cause ERS like algorithms to overshoot the number of results found.

Dynamic Querying [21] (DQ), adopted by Gnutella [44, 21], goes a step further by adjusting the TTL based on resource popularity. It works in two phases. The first aims to probe the popularity of a resource and consists in a flooding-based search with a small TTL value and partial fanout. If this optimistic first phase doesn't find the number of results, the second one takes in. It operates as follows. With the information already obtained in the probe phase, the source node infers the popularity of the resources and how many peers have been reached so far. This is used to estimate resource popularity. With it, the node infers how many more peers should be contacted to obtain the number of results. Knowing the degree of their neighbors, a node infers the TTL that should be used when contacting them. This value it's adjusted when the results of one neighbor are returned. The search ends when all peers were used or when the desired number of results was found.

However, in the Gnutella implementation, only nodes with a high degree (superior to 32) must use this mechanism. In [21] an evaluation of this method was performed using a snapshot of the two-layer Gnutella network (a detailed description of the network is provided in [43]) and the authors found that when performed by nodes with smaller degrees (between 7 and 9) the performance degrades with the number of results being unpredictable and often causing overshooting. This can be explained by small number neighbors used in the probing phase. Additionally, DQ was also compared to ERS with authors finding that has lower dissemination cost and mitigates overshooting problems of the latter but has a high latency caused by the conservative approach to define TTLs and the iterative process of contacting neighbors that aims to get partial results. An enchanted version of DQ was proposed in the same work, mitigating all the problems mentioned above by using a greedy second phase (with the higher TTLs that aim to retrieve all resources at each neighbor visited) and improved estimation methods to avoid overshooting.

2.3.4.2 Random Walk

Random Walk (RW) algorithms aim to minimize the number of messages generated during a search. To achieve this, each node involved in the search uses a fanout of one,

forwarding the message containing the query to a single neighbor, often referred to as a "walker". Since the next hop is chosen randomly, the walker traverses the overlay following a random path.

The main objective of RW is to reduce the dissemination cost, however, to achieve it a trade-off is made. Since the number of queried nodes grows slowly at a constant rate, searches are expected to have a significant high latency. This is aggravated if the resources are not well replicated in the network with walkers having to visit large fractions of nodes to find the desired results. Another aspect that must be taken into consideration is how RWs are susceptible to failures. If a node fails before forwarding the walk the search may stop prematurely, the same applies to overwhelmed nodes that may drop queries to improve their state. Furthermore, even without dropping the queries, the performance can be damaged since if queries are stored at a queue (or buffer) the latency of the search may increase [8].

To mitigate these problems, multiple parallel walkers can be deployed by the origin node. This search technique is known as **K-Walkers**. In this approach, the source node uses a fanout of k, the number of queries deployed (k) is configurable, but the nodes responsible to forward the queries use a fanout of one. Higher k values lead to faster searches since more nodes are reached in less time but also generate more load.

As with flooding-based searches, RW techniques typically employ a TTL value to limit the number of times that a walker can be forwarded. However, the TTL can be combined with checking that can also serve as a keep-alive mechanism to improve its reliability.

Authors of [31] stated that, in standard RW, message overhead is reduced by an order of magnitude when compared to ERS but at the cost of increasing delay (latency) by an order of magnitude. Measurements presented in the same work also showed that deploying 32 walkers reduced overhead by two orders of magnitude compared to flooding, although the number of hops required to find an object slightly increased.

2.3.4.3 Hybrid

Combining the two main groups of uninformed search algorithms already discussed (i.e., flooding and random walk) can be used to create algorithms that try to obtain the best of both worlds.

In **Local Flooding with k Independent Random Walks** [14], an initial local flood with a small, predefined TTL is performed. If satisfactory results are not found, each node that received the query in the first phase initiates a random walk.

The authors of [14] also performed experiments, measuring the number of reached nodes in a fixed number of messages, concluding that flooding (and normalized flooding) is ineffective in highly clustered networks (being outperformed by random walk based techniques) and that their algorithm slightly outperforms the k-walker in power-law networks (especially with high k-values). However, no other performance metrics, such as latency (which is expected to be high in RW-based algorithms), were presented.

2.3.5 Informed Search

Informed search mechanisms attempt to improve the search performance by using information to guide the search query toward nodes that are more likely to fulfill it. When nodes intelligently select which neighbors to forward queries to, the query hit rate is expected to increase. However, these algorithms are generally more complex and demand that nodes store and process information about their neighbors. In some approaches, this information is disseminated proactively, adding communication overhead to the network.

The information used to infer which neighbors best suited for answering a query ranges from simple metrics, such as the number of connections (degree) of the neighbors, to more sophisticated, such as similarity between nodes and search queries. Because of this, we will to further categorize the techniques accordingly to how this information is gathered.

2.3.5.1 Reactive Gathering

In reactive gathering, nodes collect information from previous search results, leveraging past experiences to improve future queries. This is a powerful concept because it generally does not introduce additional network overhead. However, the knowledge of a node is increased slowly and nodes that joined the system recently experience worse performance than well-established nodes.

A rather simple example of the concept of this category of search mechanisms is **Directed BFS** [52]. Briefly, it limits the fanout when forwarding a query (like in Normalized-flooding or Modified-BFS) but instead of choosing the gossip targets at random, employs simple heuristics to try to infer which neighbors are more likely to return good performance. The fanout and the heuristics may vary. For instance, a node could forward queries to neighbors that returned the most results in the last ten queries.

Intelligent Search [23], also known as Intelligent-BFS, employs a very similar strategy but uses previously answered-to queries to select which neighbors should receive the search message. Briefly, each node records the most recent query replies for each neighbor, which are obtained when the answer is provided (following the reverse path). In this context, a query is a set of keywords used to describe the resource. When forwarding, the query is compared to the learned data employing a distance (or similarity) function to obtain a subset of neighbors that are closer to the current query. Additionally, a random neighbor is added to the selected gossip targets to mitigate the risk of forwarding loops, where queries circulate between the same set of peers

However, as discussed in [49], Intelligent-BFS has some shortcomings. First and most importantly, improvements in the search are dependent on the assumption that nodes are specialized in specific subjects, which may be false. Second, the algorithm also lacks negative feedback, meaning that it doesn't adapt well when resources are deleted or nodes leave the network. Ultimately, the same authors found that Intelligent-BFS succeeds in

reducing the number of messages when compared to flooding, however, the query hit rate is only slightly improved when compared to Modified-BFS.

Another well-known technique is called **Adaptive Probabilistic Search** (APS) [48], that works as follows. Each node stores a table per neighbor. In these tables, each entry is an object seen by the node and has a counter that stores how many times that object was requested or forwarded by the neighbor. These values are then used to calculate the probability of choosing that node when searching or forwarding queries to a given file. When a search is initiated, the source node sends k-walkers to k probabilistic selected neighbors for that file. When a node has to forward a walker, uses the same approach to choose the neighbor which will receive the walker. The tables are updated optimistically or pessimistically when walkers follow the reverse path back. Additionally, each node also keeps a temporary state that stores each search processed, if more than a walker is received by this node the second one is terminated.

A drawback of this approach is that only files that already have been seen can take advantage of it, implying in the process that queries use some identifier (for instance, file names) to these files instead of more powerful ways to describe them like already seen keyword-based queries.

AntSearch [51] implements a very similar strategy to Dynamic Query, but the second phase is biased towards neighbors that return good results in the past to exclude free-riders from the searches. Shortly, free riders are nodes that benefit from peer-to-peer systems without contributing to them. For instance, nodes that participate in file-sharing systems in order to obtain files but never share files of their own nor help other nodes in locating source for tiles that they are looking for. The main idea behind excluding them, it's that since these nodes don't index many resources locally their contribution is expected to be small. However, and even when a node doesn't address many queries directly, doesn't mean that its neighbors don't. Taking this into consideration, in the second phase nodes are selected probabilistically according to their hit ratio (called a pheromone) and the average pheromones of their neighbors in a weighted fashion. These values are obtained in direct ping messages (being stored at each node in a pheromone table) but also be updated through the search process.

2.3.5.2 Proactive Gathering

In proactive information gathering, nodes share information used to perform search independently of the search process. This implies that, even if the node never performed a search or forwarded a query message, it still participates in the process to (at some extent) create the information to do so. Furthermore, the search process doesn't improve only with the number of searches performed, which may be seen as an advantage when implemented in dynamic networks. However, the messages exchanged to maintain this information adds additional overhead. For instance, the joining or departing of a node can trigger an update in this information in its neighbors and the joining process usually

implies the exchange of some messages to build the nodes' local knowledge.

An early approach to informed searches it's called **Routing indexes** (RI) [11]. Briefly, each node maintains a table that stores data about its neighbors and enables it to infer how likely each neighbor is to return good search results. In its simplest form, neighbors have associated the number of documents of a specific topic. However, more complex data structures can be applied, the two presented in the original work take into consideration the number of hops between the node and the documents instead of a simple counter.

One key perk of RIs is how the information about the documents is updated. When a node establishes a new connection with another node it must inform the new neighbor about how many documents (and their topics) can be reached through it. The new neighbor, upon receiving this information must update its own RI and inform its neighbors. One can see how this process causes a chain reaction of updates, even if a horizon is employed or the updates' policy it's relaxed (which can be done by not advertising minor changes or delaying the updates) a non-trivial amount of update messages are expected. This is aggravated if the network suffers from high levels of churn, other membership changes or if nodes change the documents that have available locally.

Finally, another relevant aspect to take into account it's the fact that cycles have to be addressed directly either by avoiding, detecting, and recovering from them or by limiting the horizon of the update. This introduces further overhead and is aggravated by network with a high clustering coefficient.

Gia [8], a P2P file-sharing system built on top of Gnutella that aims to improve scalability by exploiting node heterogeneity, takes advantage of this approach. Different from other approaches, Gia tackles the search problem on multiple fronts. Several changes are proposed. Firstly, all nodes index the content of their neighbors. Second, instead of flooding, the search is performed using a biased random walk through well-connected nodes that are limited by a TTL and return results following the reverse path. The motivation is simple, since nodes index the resources of their neighbors, nodes with a high degree will, naturally, index more content and therefore have a higher probability of fulfilling the query. This strategy slightly reassembles the super-peer architecture. However, high degree nodes may not be able to handle the extra load. Lastly, and to address this issue, Gia proposes two different mechanisms. First, the overlay is biased through nodes with high capacity. The capacity is defined as a representation of how many queries a node can handle per second and is obtained from the node's specification (for instance, bandwidth). This mechanism ensures that weaker nodes are connected to more powerful ones, turning them into well-connected nodes. Second, a flow control mechanism enables nodes to actively advertise to their neighbors how many queries are willing to accept through a token system. Nodes only perform searches through neighbors that attributed their tokens. The distribution of tokens to neighbors is adjusted to the rate of queries that the node can handle. The main motivation to adopt an active mechanism instead of a reactive one comes from the fact that the second, where nodes drop queries when are overloaded, would cause the problems already presented in Section 2.3.4.2.

The flow control mechanism also serves as an incentive for nodes to advertise their true capacity since nodes with higher capacity have access to more tokens to perform their searches.

2.4 Case Studies

As stated before, search is a service integrated into a system. In order to enable one to see how it is integrated but also shaped by the overall system, in this section we will present two examples. The motivation to choose each one varies, but both were deployed and adopted by considerable amounts of users.

2.4.1 Gnutella

Gnutella was an early fully decentralized file-sharing network. It is specified as a protocol, being implemented by several clients that share the same network. Because of this, the behavior of peers that are connected through different clients may vary slightly. Furthermore, the protocol specification enables clients to implement their own extensions (that can be used to add custom features) and is purposely loose when describing certain mechanisms. Ultimately, we will present Gnutella as a system, considering the specifications of two different versions to describe it and studying the network as a whole. The motivation to present this system as an example comes from several factors.

1. In its prime, was very popular, forming a large-scale overlay with 1.3 million peers in February 2005 [44].

2. Due to the open nature of the protocol, a lot of work was published around it. The network was well characterized through several surveys and, even when is not the prime subject, manifold articles present search mechanisms to improve it, or at least, study their performance in some Gnutella topology.

3. Its architecture changed, swapping a flat unstructured overlay to multilevel super-peer one which may present some insight about the impact of the overlay in the system services.

In Gnutella 0.4 [18], peers form a flat unstructured overlay establishing connections with a set of neighbors. Messages are propagated in the overlay by flooding with a limited TTL (with recommended values between 1 and 7 hops) and contain a randomly generated identifier and a type. Responses maintain the same identifier and are forwarded backward using the same path. The identifier it's used to detected cycles and reduce duplicates. When a node receives two or more messages with the same identifier and type, removes the duplicates from the network by not forwarding them. Furthermore, messages can be a *Ping*, a *Pong*, a *Query*, a *QueryHit*, or a *Push*, with Ping/Pong and Query/QueryHit forming pairs where the first element is a request and the second a response for that given

request. When nodes receive a response that doesn't match an already seen request (for instance, a QueryHit with a given identifier without having received a Query with the same identifier) also stop forwarding it. Nodes learn about their neighbors and discover new nodes by periodically sending Ping messages to them. Upon receiving a Ping message, nodes must respond with a Pong that contains some information about themselves. The TTL used in Pings may vary, if the TTL is equal to 1 are perceived as "direct" pings and if with the TTL equals 2 are known as "browsing" pings. Searching is performed via Query messages. When a node receives a query, tries to match a set of keywords with its resources, if is successful responds with a QueryHit that must contain enough information to a node retrieve the data.

Motivated by scalability issues, Gnutella 0.6 [19] version, also known as the second version of Gnutella, changes the overlay to a super-peer architecture. Nodes can be leaf nodes or ultrapeers. Leaf nodes only establish connections with a few ultrapeers and never serve as relays between ultrapeers' communication. Ultrapeers, effectively, form a top-level overlay by establishing relations with other ultrapeers and using it to forward queries on behalf of leaf peers. The number of super-peers in the network is dynamic, changing according to the network needs, and any node that meets a set of requirements can become one. These requirements ensure that nodes can fulfil the role and take into consideration the node capacity (for instance, bandwidth and CPU speed) and their uptime. When a leaf peer connects to an ultrapeer shares its Query Route table which is, in practice, an index of the keywords that describe all of its files. When an ultrapeer receives a query checks all the tables of the leaf nodes and forwards the query to those who are likely to fulfil it. The exact search mechanism used is not specified, however, some works state that it employs Dynamic Querying [44, 21], which was already presented in Section 2.3.

Several works have explored the Gnutella network in an attempt to characterize it. Both [40] and [37] studied an early version of the network (before the adoption of the super-peer architecture). In [37] is stated that the networks had a small diameter, with most of the nodes being less than 7 hops away from each other and was very dynamic, with 40% of nodes connected to the network leaving in less than 4 hours. The degree distribution had power-law characteristics but, as the same authors point out, the degree of each node it's defined locally by limiting the number of connections and deciding a policy to which other nodes a client should be connected. In two different observations that were performed some months apart, was discovered that the network shifted from a power-law topology across all nodes to one where only nodes with more than 10 connections can be characterized as such. In [40] the power-law distribution of the Gnutella network is further explored, analyzing the network robustness and concluding that it can handle a great number of random disconnections (around 60% of random nodes) without being fragmented. However, targeting nodes with higher degrees tells a different story, with only 4% of the best-connected peers being disconnected from the network the overlay was divided into several small "pieces". Furthermore, the same work also describes the duration of a session (considering only sessions with less than 12 hours) concluding that

in median each session was around 1 hour. The peers' heterogeneity is also studied, peers were found to vary a lot (between 3 and 5 orders of magnitude) in internet speed and latency, time in the system, and amount of files shared.

The multilevel network was also studied. In [43, 44] the network top-level (the super-peers overlay) is described as having a small diameter because of the high degree of ultrapeers and is stated that it doesn't exhibit a power-law topology but a small-world one. Furthermore, in [44] the authors argue that such results in previous works are explained by distorted snapshots that are a consequence of slow crawlers. The same work found that each ultrapeer prefers to serve between 30 and 45 leaf peers and maintain around 30 ultrapeers neighbors which form spikes in the degree distribution. Leaf peers are mostly connected to 3 or fewer ultrapeers. The authors argue that the degree distribution (both for leaves and ultrapeers) is heavily influenced by the desired number of neighbors defined in the two more popular clients at the time (which were found to be used between 94% and 96% of the nodes), however, one must not forget that churn and membership changes are an obstacle to maintain constant degree values. When it comes to the shortest paths, it's stated that the introduction of super-peers (and considering that the network growth significantly) didn't change the shortest-paths values. The measurement revealed that in the ultrapeer network most nodes (65%) are reachable within 4 hops while considering the whole network this values increases to 5, shortly followed by 6 hops (both around 50% of the nodes). Finally, was found that the network is highly resilient, maintaining 90% of nodes connected after removing 85% random nodes and, perhaps more surprisingly, 75% of the nodes connected after removing the top 50% peers with the highest degree.

2.4.2 InterPlanetary File System

The InterPlanetary File System (IPFS) [6] is a distributed, peer-to-peer file system that enables its users, who act as nodes across two overlay networks, to share, search, and retrieve resources.

IPFS has garnered significant attention in recent years and is under active development. It can be applied to a wide range of use cases [50], and several projects have integrated it as part of their technology stack.

In IPFS, resources are modeled using a Merkle Directed Acyclic Graph (Merkle DAG). Merkle DAGs are similar to Merkle trees, but unlike the latter, they do not require balancing. Additionally, the nodes in a Merkle DAG can have multiple parents and are capable of storing data. The data, referred to as a *block*, varies in size between 256 KiB and 1 MiB. Consequently, large files are typically split into multiple blocks, which are then organized into a DAG. The root node of this DAG links to child nodes that contain the payload. Each node in the DAG is uniquely identified by a *content identifier* (CID), which is derived from the multi-hash of its content (i.e., its payload and/or references to other child nodes). A multi-hash is a cryptographic hash that includes both the data hash and metadata about the hash function and the data type.

Several key features of IPFS arise from this data modeling approach. For instance, resources in IPFS are immutable since any modification results in a new CID, necessitating propagation of this change throughout the DAG. Additionally, searches within IPFS may require recursive sub-searches that traverse the DAG structure to retrieve complete resources.

Peers within the IPFS network are identified by a *PeerID*, which is a multi-hash of the peer's public key. These peers participate in two distinct overlay networks, which share connections but differ in architecture and purpose. The first overlay is a DHT based on Kademlia [33], where peers publish and retrieve pointers to nodes capable of providing specific CIDs (*provider records*) and lists of addresses associated with specific PeerIDs (*peer records*). Nodes in this DHT operate in one of two modes: server or client. Server nodes, typically those that are publicly reachable via the Internet, maintain the network by participating in routing and serving content, while client nodes primarily perform requests without contributing to the network's maintenance. The second overlay is unstructured and is utilized by a protocol called Bitswap [38], which handles resource searching and data transfer. Unlike the DHT, where explicit publishing is required, Bitswap functions as a decentralized caching mechanism. When a peer retrieves a resource via Bitswap, it temporarily stores the resource and makes it available to other peers upon request. This mechanism increases resource availability and is expected to improve the likelihood of locating popular resources. Bitswap will be explored in greater detail in Section 3.2, as its search functionality forms the foundation of the solution presented in this work.

To locate resources in IPFS, peers initially perform a flooding search via Bitswap, restricted to a horizon of one (i.e., the queries are not propagated further in the network by the contacted neighbors). If the search is successful, the querying node that received a positive response proceeds to request the resource from the responding neighbor, which then transfers the resource. Upon receiving the resource, if the DAG contains additional blocks, the querying node either requests the remaining blocks from the same neighbor or initiates a new search if the neighbor cannot provide them.

In cases where the search fails or takes too long, peers resort to the DHT to locate the desired CID, making the DHT a fallback mechanism. Resource discovery in the DHT occurs in two phases. First, the searching node queries the DHT to identify a provider of the desired CID. If the provider's address is not found in the local cache, the DHT is queried again to retrieve the address, enabling the node to establish a direct connection with the provider. Once the address is obtained, the content transfer is performed via Bitswap.

Recently, studies have emerged examining the IPFS network. However, most of the existing research primarily focuses on the structured overlay (the DHT).

In [10], an analysis of two weeks' worth of logs from one of the most popular IPFS gateways (nodes that convert HTTP requests into IPFS requests) revealed that the frequency of requested CIDs follows a typical Zipfian distribution. Furthermore, by querying the DHT for these requested CIDs and analyzing the associated providers, the study found

that the requested CIDs were not well replicated within the DHT. An analysis of the records from these same providers revealed that only a small percentage of all CIDs were well replicated, and that a small percentage of providers served a disproportionately large number of CIDs. This phenomenon is attributed by the authors to the need for explicit action to replicate content in the DHT, a requirement that, as previously noted, is absent in Bitswap, which functions as a cache. This raises concerns regarding the balance of the structured overlay. Finally, by approximating the geolocation of providers' IP addresses, the study confirmed that most content is not retrieved from local providers, highlighting the global scale of the IPFS network.

In another study, the authors of [20] presented a comprehensive overview of the IPFS network, with a focus on its structured overlay protocol, highlighting discrepancies between the official white paper[6] and the implementation's source code at the time of the research. Through an empirical study, the authors found that IPFS nodes typically maintain around 900 connections, with 76.40% of these connections enabled by the DHT protocol. By crawling the overlay over a six-day period, they identified 309,404 unique PeerIDs. However, only a small percentage (6.44%) of these nodes accepted incoming connections, suggesting that a large proportion of IPFS nodes are behind NATs and may be operated by private individuals.

2.5 Discussion and Summary

In this chapter, we presented a comprehensive overview of search methods based on state-of-the-art literature.

We began by introducing the concept of overlay networks, which nodes use to disseminate messages, and discussed their organizational structures. Overlays were categorized based on how nodes arrange themselves, and we examined the implications of multilevel architectures.

A key insight highlighted was that the choice of search methods is intrinsically linked to the type of overlay network employed. This relationship must be carefully considered when designing search algorithms. Certain limitations, such as the restriction of searches in DHTs to exact-match queries based on unique resource identifiers, arise directly from the underlying overlay structure.

The discussion then focused on search techniques in unstructured overlays, which are central to this work. We analyzed various aspects of search, such as query formulation and message forwarding strategies, and reviewed several well-established search algorithms.

Our exploration revealed that while some techniques are more efficient than others, each involves trade-offs. For instance, flooding-based techniques achieve high recall rates but incur significant dissemination overhead, impacting scalability. Conversely, random walk (RW)-based approaches reduce overhead but at the expense of increased search latency.

Informed search algorithms aim to achieve better trade-offs by enabling nodes to use additional information to make smarter decisions about where to route queries. These algorithms vary in how they gather and utilize such information.

We explored two key strategies for information gathering and maintenance: reactive and proactive approaches. Reactive methods incur lower maintenance costs but are less effective in dynamic networks. On the other hand, proactive methods, though requiring greater effort to build and maintain search indexes, offer advantages in highly dynamic environments.

Lastly, we examined two relevant systems. The first represents a classic example extensively studied in the literature. The second, gaining increasing relevance, is central to the solution proposed in this thesis.

IPFS, a peer-to-peer file system, utilizes two overlay networks: a structured overlay and an unstructured overlay. Search and content retrieval are initially carried out through a flood-based protocol that also serves as a caching mechanism, improving resource availability. However, this approach is limited to one hop (i.e., only direct neighbors are queried) and incurs significant overhead due to its reliance on flooding. This issue is further exacerbated by the large number of neighbors maintained by IPFS nodes. When the initial search fails, peers resort to the structured overlay network. In the structured overlay, where only explicitly published content is available, peers perform a two-phase search: first, to identify a provider of the desired content, and second, to retrieve the provider's address (if not already cached) to establish a direct connection.

In the next chapter, we introduce our proposed solution: an informed search algorithm for the Bitswap protocol. This solution incorporates a new layer within the unstructured overlay, enabling nodes to proactively share information and thereby improving search efficiency.

3

Optimizing Search through Indexing

In this chapter, we present our solution for unstructured search. Rather than relying on a standalone search algorithm, our approach combines multiple mechanisms that leverage their synergy to overall improve the effectiveness of decentralized search. The remainder of this chapter is structured as follows:

In Section 3.1, we define the system model.

In Section 3.2, we describe and provide insights into the system and protocol that serves as the foundation of this work.

In Section 3.3, we introduce the core of our solution, outlining the fundamental mechanisms that underpin the main contribution of this work.

In Section 3.4, we expand upon these foundational mechanisms to develop a more refined and optimized version of our solution.

Finally, in Section 3.5, we present an overview of the implementation of our prototype, detailing multiple components and their integration within the existing system.

3.1 System Model

We consider the system model to represent a distributed scenario in which nodes, each with limited resources (such as CPU, memory, and disk), interact via message exchange over the *Internet*.

In this model, the nodes must be part of a simple **file-sharing P2P system** operating on an unstructured overlay network formed through a partial membership protocol. The network facilitates message exchange between established nodes and their neighbors, the set of directly connected nodes of a given node. However, no message forwarding capabilities are assumed. Message delivery is ensured to occur in order at the transport layer, utilizing either TCP or QUIC protocols.

Within this system, nodes utilize message exchange to perform basic tasks, such as querying neighbors about the ownership of resources and retrieving said resources if owned by one of them. Nodes can join or leave the system at any time without any notice, including due to crash-faults.

Finally, nodes must be able to open new connections to other nodes in the network if enough information is provided, such as a public IP address or, in an IPFS-based system, a unique ID known as *PeerID*.

3.2 Baseline: Bitswap

In Chapter 2, we introduced a state-of-the-art system that has recently gained significant attention: the *InterPlanetary File System* (IPFS). As previously discussed, IPFS employs an uninformed, flood-based search mechanism over its unstructured overlay network, leveraging a search and block exchange protocol known as *Bitswap* [38]. Bitswap plays a pivotal role in IPFS, serving not only as the primary search mechanism and the sole protocol enabling content transfer, but also as a caching mechanism. Despite its central role, the search mechanism used by Bitswap can be considered relatively unsophisticated (as discussed previously).

Given the relevance of IPFS, and the fact that Bitswap has potential for optimization while aligning with our system model (outlined in Section 3.1), we have chosen Bitswap as the foundational system for the development of our solution.

In the remainder of this section, we provide a brief overview of Bitswap and insights to aid in understanding the proposed solution.

In a nutshell, Bitswap constructs an unstructured overlay network where nodes initiate searches by flooding their immediate neighbors (limited to one hop). Due to the structure of resources in IPFS, which are organized as Merkle DAGs, a search often involves a sequence of recursive (sub-)searches. For instance, a node seeking a resource may first retrieve the root of the DAG, followed by each referenced DAG node, and so on.

To optimize this process, Bitswap introduces the concept of *sessions*. Sessions group neighbors likely to be helpful for subsequent searches, based on the assumption that a neighbor possessing part of a resource is likely to have other parts of the same resource.

Each search begins with the creation of a session and the addition of members, which is achieved by flooding neighboring nodes with a WANT-HAVE message. This message, containing a CID (i.e., content identifier, as presented previously in Chapter 2), prompts a response with a HAVE message if the recipient node has the requested resource, or a DONT-HAVE message otherwise. However, nodes running more recent protocol versions may omit sending a DONT-HAVE message, in which case the querying node infers the lack of the resource after a timeout.

Additionally, CIDs involved in the flood are added to a list (referred to as the wantlist) until the corresponding resource is retrieved. This wantlist is sent as a WANT-HAVE message whenever a connection is established with a new node.

Any neighbor that responds with a HAVE message is added to the session. Within the session, one of the neighbors then receives a WANT-BLOCK, meaning that it should send the block identified by the CID in BLOCK message, which contains the data. Once the searching node receives the BLOCK message, it sends a CANCEL message to all neighbors

that were previously sent a WANT-HAVE or WANT-BLOCK, indicating that is no longer searching for that resource.

This process is repeated for each resource within the session. However, specific events may trigger the addition of new nodes to the session:

1. **No neighbor in the session can fulfil a WANT message.** This occurs when all neighbors respond with DONT-HAVE messages, or when the session becomes empty after evicting all neighbors following a series of consecutive DONT-HAVE responses (16 by default).

2. **No block is received within a specified time interval.** Initially, this interval is set to 1 second (by default) but is recalculated based on the latency of previous responses, if any, or progressively increased if no responses have been received.

3. **The session duration exceeds 1 minute** (default value).

In such cases, a DHT search is initiated for an active CID (i.e., a CID corresponding to a resource that has not yet been found) within the session. If the CID has not already been flooded (i.e., is not in the wantlist), it is flooded. However, to reduce the load placed on the DHT, for the first and second events, the DHT search is only performed on the first occurrence (i.e., when the consecutive count is zero). This counter resets once a block is received by the session. In the third case, the session always searches for a random active CID (i.e., a CID that has not yet been found). Nodes discovered via the DHT are added to the session, and the search continues as described previously.

3.3 Proposed Solution

The cornerstone of our solution is the introduction of a novel indexing layer within the unstructured network. In this layer, nodes proactively share information about the resources they can provide. This information is stored and leveraged by interconnected mechanisms. The first mechanism, designed to broaden the horizon of searches, enables nodes to assist their neighbors without further disseminating queries across the overlay. The second mechanism, aimed at reducing the number of messages and improving efficiency, enhances the current Bitswap search protocol by enabling nodes to perform informed searches.

Our solution will be presented iteratively in the following subsections. We will emphasize how each component integrates into the overall solution and explain the rationale behind the design decisions. The structure of this section is as follows:

First, in Subsection 3.3.1, we introduce the indexing layer and the information shared within it.

Next, in Subsection 3.3.2, we present source sharing, which allows nodes to assist neighbors in their searches.

Finally, in Subsection 3.3.3, we describe the initial version of the informed search algorithm.

3.3.1 Indexing Layer

In our solution, each node constructs and maintains a partial view of its neighbors, referred to as *close neighbors*, effectively creating a new overlay on top of the unstructured overlay network. These views are asymmetrical to avoid the overhead of additional coordination, such as exchanging extra messages to manage the addition or removal of neighbors from each node's respective view.

The view is built by incrementally adding new neighbors until a configurable maximum is reached. Once this limit is attained, the view is managed reactively, meaning it only changes in response to external events, such as a neighbor disconnecting or leaving the system. The dynamic nature of the network ensures that the view is continuously repopulated as nodes connect and disconnect.

To promote stability, we prioritize maintaining connections to close neighbors using IPFS's native mechanisms. However, the duration for which a neighbor remains in the view is not assumed or guaranteed.

The purpose of this overlay is to facilitate the exchange of information regarding the resources each node can provide. This information, referred to as the *index*, is a set of content identifiers (CIDs) for each resource (block) that a node possesses and thus can provide. Nodes transmit their complete index (i.e., all owned resources) to a close neighbor only once, when the node is first added to the view. Subsequent changes, occurring within a configurable time interval, are aggregated and sent as index updates. These updates may include additions when new resources are acquired or deletions when resources are no longer available. Batching these updates reduces the number of messages exchanged, and transmitting only the changes minimizes message size.

Since views are asymmetrical, nodes accept indexes from any neighbor, even if that neighbor is not included in their own view. Upon receipt, the index and its origin (i.e., the PeerID of the node that sent it) are stored as long as the connection to the neighbor remains active.

The repository of indexes (referred to as the indexstore, or simply indexes, in the rest of this chapter) can be consulted to retrieve the source of a specific resource. The utilization of this information will be detailed below.

3.3.2 Source Sharing

Indexes provide nodes with information, within a certain level of confidence, about the CIDs their neighbors can serve. The source sharing mechanism leverages this information to assist neighboring nodes in their searches, thereby mitigating the one-hop search horizon limitation imposed by Bitswap. Unlike traditional search mechanisms, this

process is achieved without increasing the Time-To-Live (TTL) or further disseminating search queries.

To facilitate this, we introduce a new message type called the SOURCE message. The SOURCE message, which serves as an alternative to the HAVE message typically sent in response to a WANT-HAVE query, contains the CID of a resource and PeerIDs of potential providers (or sources) of that resource.

Algorithm 1: Handling Received WANT-HAVE queries

Input: peer (sender), cid
Result: Responds with either *have*, *dont-have*, or *source* message

```
   /* Check if the block repository contains the CID                    */
1  if blockstore.contains(cid) then
2  |  response ← (cid, have);
   /* Check if the index repository contains the CID                    */
3  else if indexstore.contains(cid) then
   |  /* Fetch up to 10 random sources from the indexstore              */
4  |  sources ← indexstore.getN(10, cid);
5  |  response ← (cid, sources);
6  else
7  |  response ← (cid, dont-have);
8  end
   /* Send the response message to the sender peer                      */
9  send(peer, response);
```

Algorithm 1 illustrates, in simplified form, how nodes handle WANT-HAVE queries in our solution. Upon receiving a WANT-HAVE message, nodes first check whether they can provide the requested resource (Alg. 1, lines 1 to 2) and respond accordingly. The HAVE message is prioritized over the SOURCE message due to its smaller size and more direct nature. A HAVE message allows immediate resource retrieval, while a SOURCE message may require additional steps to establish connections with providers, as the source may not be directly connected to the node that initiated the search.

If the node cannot provide the requested resource, it checks the indexes of its close neighbors and sends their PeerIDs as potential sources (Alg. 1, lines 3 to 5). A configurable limit on the number of sources per SOURCE message is enforced to manage message size and avoid overshooting (e.g., providing too many results for a query). If the limit is reached, sources are selected randomly to ensure diversity, and, thereby reducing the likelihood of the origin node receiving duplicate sources from multiple neighbors.

If the resource cannot be found, the node responds with a DONT-HAVE message.

When a node receives a SOURCE message, it first checks whether any of the listed sources are already connected. It then attempts to establish connections with those that are not. Given the possibility of receiving duplicate sources, during this process, the node temporarily stores the PeerIDs along with the CID of the resources they are able to provide.

Once a connection with a source is established, the resources it offers are reported to the Bitswap session as a HAVE message from that source. This information is also cached temporarily to avoid multiple HAVE reports from the same source, which could lead to the resource being requested multiple times by the session. Since the resource is treated as a HAVE, the searching node can directly request it using a WANT-BLOCK message, bypassing the need for a WANT-HAVE query. This optimization reduces the number of messages and minimizes round-trip time (RTT).

It is important to note that indexes may occasionally be outdated, meaning that the sources provided by a neighbor may no longer possess the resource. In such cases, the source will respond to a WANT-BLOCK request with a DONT-HAVE message. When this occurs, the Bitswap session removes that neighbor from the search, allowing the search to continue as usual.

3.3.3 Informed Search

Similar to the source sharing mechanism, the informed search algorithm also utilizes the indexes to guide the search process. This approach helps reduce unnecessary flooding of queries to neighboring nodes.

Algorithm 2: Informed Search Algorithm

Input: *cid* of the desired resource
Result: Sends search queries for the requested resource

```
   /* Check if the local index repository contains the requested CID  */
 1 if indexstore.contains(cid) then
 2 |   sources ← indexstore.get(cid);
 3 |   send (sources[0], (cid, want-block));
 4 |   foreach source in sources[1:] do
 5 |   |   send (source, (cid, want-have));
 6 |   end
 7 else
 8 |   foreach neighbor in neighbors do
 9 |   |   send (neighbor, (cid, want-have));
10 |   end
11 end
```

Algorithm 2 illustrates the informed search process. Instead of flooding the network immediately, as is common in Bitswap, nodes first consult the indexes of their neighboring peers (Alg. 2, line 1). If a match is found in the local index, the node optimistically sends a WANT-BLOCK message to one of the sources (Alg. 2, line 3), an approach that, if successful, minimizes the round-trip time (RTT) by avoiding unnecessary additional requests.

Simultaneously, WANT-HAVE messages are sent to other potential sources (Alg. 2, line 4) to provide redundancy, ensuring a higher chance of success if the initial WANT-BLOCK request fails or if the index information is outdated.

Finally, if no matches are found in the local index, the node falls back to broadcasting WANT-HAVE messages to all its neighbors (Alg. 2, line 7). However, it is important to note that, due to the source sharing mechanism, these neighbors can respond with sources from their own indexes, effectively extending the search beyond the immediate neighborhood. This allows the search to reach a broader horizon, increasing the likelihood of finding the requested resources.

3.4 Proposed Enhanced Solution

In this section, we build upon the core ideas presented in the previous section, introducing significant enhancements to both the indexing layer and the informed search mechanism.

We'll begin, in Section 3.4.1, by introducing the concept of the meta-index, a novel approach that consolidates the indexes received by a node into a compressed form using a Bloom filter [7].

After that, in Section 3.4.2, we'll dive into the improvements we've made to the informed search mechanism, including a closer look at its two operational modes: the filtering mode and lookup mode.

3.4.1 Meta-indexing

The first enhancement in our protocol is the introduction of *meta-indexes*. A meta-index aggregates the information from individual indexes received by a node (i.e., the indexstore) in the form of a *Bloom filter*.

In essence, a Bloom filter [7] is a probabilistic data structure known for its space efficiency. The primary advantage of employing Bloom filters stems from this attribute. Meta-indexes are anticipated to be both larger and grow more rapidly than individual indexes, as their size increases in proportion to the combined growth of all the individual indexes they encapsulate. However, Bloom filters can produce *false positives*, and often there are trade-offs between accuracy and size.

In our approach, nodes dynamically adjust the size of the Bloom filter based on the number of entries in the index repository. The goal is to maintain a reasonable level of accuracy while minimizing the size of the messages exchanged during sharing. As a consequence, the meta-index size varies across nodes. Nevertheless, this also results in scenarios where nodes must rebuild the meta-index when it reaches its capacity. Since Bloom filters typically do not support deletions, this occurs not only when space runs out but also when index updates include deletions.

Nodes share their meta-indexes with close neighbors through the indexing layer, leveraging the already established view and thus avoiding additional coordination. However, these meta-indexes are shared at more conservative time intervals. The rationale behind this policy is twofold. First, despite the use of Bloom filters, meta-indexes are expected to be larger than individual indexes, which may impose a heavier load on the network.

Second, reducing the frequency of meta-index updates mitigates the computational cost of rebuilding the Bloom filter, allowing nodes to perform this operation less frequently.

Similar to individual indexes, meta-indexes are shared only when changes occur and nodes accept and store meta-indexes from any neighbor, maintaining this information for as long as the connection persists. However, meta-indexes lack the ability to support partial updates, meaning that when a meta-index changes, it is shared in its entirety with close neighbors.

Unlike indexes, obtaining a source for the desired resource through meta-indexes is not possible since the node that shared it may not own the resource. Therefore, in the event of a match, the querying node can only that that its neighbor possesses information about the specific entry. Essentially, from the querying node's perspective, this means that the neighbor knows who might have the resource. We will explore the impact of this below.

3.4.2 Enhanced Informed Search

The final expansion to our protocol introduces meta-indexes to the informed search mechanism, enabling searches to leverage an additional layer of information: the neighborhood of a neighbor. We developed two different modes in which this information may be used.

Algorithm 3: Enhanced Informed Search Algorithm (Filtering Mode)

Input: *cid* of the desired resource
Result: Sends search queries for the requested resource

```
/* Check if the local index repository contains the requested CID */
if indexstore.contains(cid) then
    sources ← indexstore.get(cid);
    send (sources[0], (cid, want-block));
    foreach source in sources[1:] do
        send (source, (cid, want-have));
    end
else
    foreach neighbor in neighbors do
        if !metaIndexstore.hasPeer (neighbor) or !metaIndexstore.peerHas
         (neighbor, cid) then
            send (neighbor, (cid, want-have));
        end
    end
end
```

The first mode, called *filtering mode*, directly addresses the challenge of meta-indexes potentially producing false positives, though taking into consideration false negatives never occur. This mode aims to minimize the fanout of flooding while using meta-indexes to avoid contacting neighbors unlikely to contribute to the search.

Algorithm 3 shows how this search process functions. As in the standard informed search (Algorithm 2), local indexes are consulted, and if a match is found, flooding is avoided (Alg. 3, lines 1 to 6). However, if no match is found, instead of querying all neighbors, the node consults the meta-indexes (Alg. 3, line 9) and only contacts neighbors that either have not shared their meta-index or whose meta-index indicates a potential match (Alg. 3, line 10).

Algorithm 4: Enhanced Informed Search Algorithm (Lookup Mode)

Input: *cid* of the desired resource
Result: Sends search queries for the requested resource

```
   /* Check if the local index repository contains the requested CID  */
 1 if indexstore.contains(cid) then
 2 |   sources ← indexstore.get(cid);
 3 |   send (sources[0], (cid, want-block));
 4 |   foreach source in sources[1:] do
 5 |   |   send (source, (cid, want-have));
 6 |   end
   /* Check if the local meta-index repository contains the requested
      CID                                                              */
 7 else if metaIndexstore.contains(cid) then
 8 |   sources ← metaIndexstore.get(cid);
 9 |   foreach source in sources do
10 |   |   send (source, (cid, want-have));
11 |   end
12 else
13 |   foreach neighbor in neighbors do
14 |   |   send (neighbor, (cid, want-have));
15 |   end
16 end
```

The second mode, referred to as *lookup mode*, leverages the second-order information in a more straight forward manner, contacting directly neighbors that are believed (due to meta-indexes matches) to have the resource in their neighborhood.

Algorithm 4 demonstrates how this search is conducted. As with the previous approach, the first layer of information (i.e., the indexes) is queried, and any node that has a match receives the query. If the indexes do not return a match, meta-indexes are consulted (Alg. 4, line 7). If a meta-index match is found, flooding is avoided, and a WANT-HAVE message is sent to nodes that have the resource listed in their meta-index (Alg. 4, lines 8 to 11). These nodes, through the source-sharing mechanism (discussed in Section 3.3), are expected to return sources from their own indexes. If no match is obtained, the search proceeds with the standard informed search approach, broadcasting the query to all neighbors.

3.5 Implementation

To evaluate the proposed solution, whose results are presented in Chapter 4.2, we developed a prototype that replaces the existing Bitswap implementation (used as the baseline in our experiments) within the IPFS stack. This prototype integrates seamlessly with the overall IPFS system components.

Our development process began with an in-depth analysis of the Go source code for Bitswap and, to some extent, the IPFS system and its related components. This examination provided critical insights into their internal mechanisms, which informed the design and construction of our prototype. A high-level summary of the key insights gathered during the implementation process is outlined below.

3.5.1 Bitswap

The Bitswap source code is divided primarily into two main packages: `server` and `client`, each of which contains multiple sub-packages and files. In addition to these, other important packages, such as the `message` package, play a significant role by defining and implementing message structures and their auxiliary logic, however, we will focus mostly in the main two.

The coordination between these packages and the IPFS network is facilitated through the main Bitswap file (`bitswap.go`), which is invoked by the IPFS system to handle external events, such as receiving messages from peers in the network.

3.5.1.1 Server Package

The `server` package implements the engine responsible for responding to requests from neighboring nodes (e.g., want-haves or want-blocks). The server engine performs this by consulting the blockstore, an external component that stores the resources a node can serve, enabling lookups and retrieval of content.

3.5.1.2 Client Package

On the other hand, the `client` package is responsible for executing search operations. As outlined in Section 3.2, the client implements the concept of a session, which is utilized by the search algorithm. The session behavior itself, including tracking timeouts and invoking the DHT if necessary, is implemented by `session.go`. This component constantly updates a subcomponent responsible for subsequent search processes (`sessionwantsender.go`).

However, given that multiple concurrent sessions may run simultaneously, session management plays a crucial role in the behavior of the client. The logic for managing sessions is spread across various packages and files. For example, `sessionmanager.go` is responsible for creating and removing sessions, as well as ensuring the correct delivery of information (e.g., notifications that a peer has sent a have message for a given CID) to the

appropriate session. This is facilitated by the `sessioninterestmanager.go` component, which tracks the CIDs that each session is interested in.

Since peers are shared among sessions, the client also implements peer management behaviors, such as protecting connections to peers in which at least one session has an interest. This functionality is managed by `sessionpeermanager.go`. Additionally, mechanisms are in place to ensure that sessions do not send duplicate messages (e.g., multiple `want-have` messages for the same CID originating from different sessions). These mechanisms are managed by `peermanager.go` and `peerwantmanager.go`. These components are also responsible for most of the message-sending behavior, such as flooding a message if a session requests it. Message multiplexing is handled by `messagequeue.go`.

3.5.2 Prototype

The proposed solution integrates into both the client and server packages, modifying existing components, which we will detail below. However, it also introduces a new package (named `idx`) containing the foundational components of our solution. It is important to note that due to the highly concurrent nature of Bitswap, most of this new components support concurrent operations and operate in a concurrent fashion.

3.5.2.1 Indexing Manager

The indexing manager (`manager.go`) is responsible for managing the indexing layer comprehensively, handling both indexes and meta-indexes. It shares meta-indexes and adjusts the size of the close neighbors' view and the intervals at which indexing information is shared, according to the configuration. This manager builds and maintains the membership, handling changes in node membership and ensuring (to some degree) connection stability by leveraging Bitswap's functionality.

Additionally, it processes index (and meta-index) messages, updating their respective repositories: the `indexstore` (`indexstore.go`) and `bloomstore` (`bloomstore.go`). The `indexstore` and `bloomstore` are responsible for tracking indexes and meta-indexes, respectively, received by a node. Both components map indexes to PeerIDs and support operations like adding, removing, and consulting indexes (or meta-indexes). The consultation function returns a set of all or a limited number of random PeerIDs that have a matching index. Additionally, the `indexstore` supports partial updates to indexes.

3.5.2.2 Sending Index Messages

The manager periodically shares the node's indexing information if new resources are acquired. This process is facilitated by Bitswap's peer manager and its message-sending stack. The node's current index is retrieved from a dedicated component, `indexer.go`, which is updated via a pub/sub-like mechanism integrated with the blockstore. This mechanism, implemented through modifications to the blockstore, notifies the `indexer` of newly added resources.

The meta-index, which is utilized throughout the stack, is directly obtained from the `indexstore`. It represents its content as a Bloom filter, generated using the external IPFS module `bbloom`. The size of the Bloom filter dynamically adjusts based on the number of entries and is recalculated when updates to the index occur. To reduce overhead, a caching mechanism delays updates, allowing changes to be batched and processed efficiently.

3.5.2.3 Search Process

For search operations, the `server` package leverages the `indexstore` to respond to requests. If a resource is unavailable in the blockstore, the engine consults the `indexstore` to identify potential sources, which are returned via a source message if matches are found.

Similarly, both the `indexstore` and `bloomstore` (depending on the configuration and whether the solution is enhanced or not) are integrated into the client. These are integrated into the peer manager, where the flooding mechanism consults the `indexstore` to request peers directly in all solutions. The meta-index is used to send requests directly (in lookup mode) or to filter out peers during flooding (in filtering mode).

Additionally, the client includes a `middleware.go` component to optimize source message handling. This middleware establishes new connections and notifies active sessions when a potential provider becomes available. It also eliminates duplicate sources to enhance performance.

3.5.3 Source Code Availability

Is important to note that, while we provide an overview of the key components and their integration in this section, additional modifications to other (existing) components were also necessary. By analyzing the Git logs of the repository where the project is hosted (publicly available: `https://github.com/rmseq/boxo`), we found that a total of 37 files were changed, resulting in 4519 additions and 175 deletions.

3.6 Summary

In this chapter, we presented two novel solutions to enhance search in unstructured peer-to-peer networks, along with insights and implementation details of the prototype that embodies these solutions.

The first solution, which serves as the foundation of our approach, is built around two key mechanisms: a collaborative process called source sharing and an informed search algorithm. Both mechanisms rely on a newly introduced indexing layer, integrated into the existing overlay, to enable nodes to share information with a partial view of their neighbors. The first mechanism, source sharing, improves search efficiency by enabling nodes to assist their neighbors in locating resources, thereby overcoming the one-hop limitation imposed by Bitswap and increasing search success rates. The second mechanism, the informed search algorithm, uses the indexing layer to guide the search directly to the node that

holds the desired resource, thereby avoiding the need for costly flooding when an index match is found. This significantly reduces search costs and, as a result, decreases the overall network load.

The second solution builds upon the first by extending the information available to nodes through the sharing of a meta-index within the existing indexing layer. The meta-index provides nodes with information about their neighbors' indexes, and thus the resources available in neighbors of those neighbors (i.e., two-hop nodes). In the enhanced search mechanism, if a node does not find the desired information in its indexes, it can use the received meta-index to guide the search to the appropriate node, one that contains an entry for the resource in its index. This node then utilizes the source sharing mechanism to provide the source of the requested resource. This effectively expands the knowledge available to while nodes, which increases the likelihood of having matches and avoiding flooding.

4

Evaluation

In this chapter, we present our experimental evaluation. The rest of this chapter is organized as follows:

In Section 4.1, we begin by describing the environment in which the experiments were conducted. Following this, we outline the experimental methodology.

In Section 4.2, we present the results for each protocol. This section starts by explaining the metrics collected during the experiments and is subsequently divided into two subsections, each providing an overview of the results and analysis for one specific scenario.

4.1 Experimental Setup and Methodology

All the experiments reported in this chapter were performed on two machines of the *DI-Cluster*. [1] Each machine has the following configuration: 2x Intel Xeon Gold 6346 CPUs with a total of 32 cores and 64 threads, 128 GiB of DDR4 memory at 3200 MHz, 440 GB internal SSD storage, and a 2x10Gbps network connection.

We used Docker's swarm mode to emulate a distributed network. Each machine in the swarm hosted 250 Docker containers, with each container running a single IPFS node. To better mimic a real-world environment, we introduced artificial latency between each pair of nodes using Linux Traffic Control (Linux TC) [45], varying the latency randomly between 75 and 225 milliseconds. The latency attributed to each pair was maintained constant throughout all experiments.

We modified the IPFS configurations of the nodes participating in our experiments as follows. We set the routing type to *dhtserver*, disabling HTTP routing to ensure that every node contributes to the DHT. For transport configurations, we disabled all protocols (WebSocket, WebTransport, and Relay) except TCP and QUIC. Additionally, we disabled the AutoNAT service. Given our network's size of only 500 participants, we set the connection manager parameters *LowWater* and *HighWater* to 16 and 48, respectively. However, the connection manager only attempts to remove idle connections. Because the

[1] https://cluster.di.fct.unl.pt/

neighbors of the DHT and Bitswap (including our protocols) are shared, we adjusted the DHT configuration to efficiently reduce the number of neighbors for nodes by setting the *BucketSize* parameter to 10.

Before initiating the experiments, the containers were launched sequentially on each machine. Each container had access to a set of protocols and configurations. An experiment's configuration included the resources hosted by each node, referred to as replicas, and a list of resources to search for, referred to as the search list.

Replicas and search lists for each node were pre-generated based on popularity. A resource's popularity determined its number of replicas and search frequency (i.e. how often a resource is searched). To emulate resource popularity, we adopted two distributions: a Uniform distribution with a probability of 0.01 and a Zipfian distribution with $\alpha = 0.82$. While the former presents an ideal scenario of equally replicated and desired files, the latter is more representative of the real IPFS network [10]. For increased variability, we generated five different configurations for each distribution.

In our experiments, a resource's popularity directly influenced its number of replicas. However, the number of searches for resources was capped at 4000 for consistency across all experiments. This cap allowed comparison among experiments with different resource counts, affecting the sizes of indexes and meta-indexes shared between nodes in our solutions. Increasing the number of resources in the experiment decreased the number of searches performed for each resource. Additionally, since nodes that retrieve a resource subsequently act as hosts, experiments with more resources are inherently more challenging. We varied the number of resources in the system between 1000, 2000, and 3000.

In our use case, the type or content of resources does not affect search performance. However, we chose to employ subsets of images provided by the authors of [13] as resources.

An experiment starts with the setup phase, which begins by copying the binary containing the version of the protocol that will be tested and the node's configuration to each container. After that, the nodes are launched sequentially (with a brief period of 500 milliseconds between launches to avoid overloading the system) within each machine, but on both machines simultaneously. Afterward, we wait for 10 minutes to allow nodes to bootstrap, establish themselves, and form a stable overlay.

When the wait is over, the experiment transitions to the search phase. In this phase, nodes attempt to find and obtain the files in their search list sequentially. However, to avoid synchronizing all searches in the system while maintaining consistency among experiments, nodes introduce a pseudorandom wait before each search. This wait varies between 1 and 40 seconds for the first search and between 1 and 20 seconds for subsequent searches. Following the wait, nodes proceed to search for the next file in their list. If a node is unable to retrieve the file within 60 seconds, the search is considered a failure, and the node moves on to the next file in the list. This phase lasts for 20 minutes.

Finally, after that, the experiment enters the cleanup phase, where all nodes are stopped

similarly to how they were launched and wait for 3 minutes before the beginning of the next experiment.

4.2 Protocol Evaluation

This section presents a comprehensive experimental evaluation of our protocol. The primary objective is to assess its performance compared to a state-of-the-art solution (Bitswap). Additionally, we aim to compare different modes of our protocol and its initial iteration, as described in Chapter 3, to validate the improvements provided by the mechanisms. To achieve these goals, we subjected each protocol to two distinct scenarios.

We will start by delineating these scenarios and pertinent details regarding the parametrization of the protocols in Section 4.2.1.

After that, in Section 4.2.2, we will provide a succinct overview of the metrics collected.

Finally, in Sections 4.2.3 and 4.2.4, we will present the results of the protocols in the first and second scenarios, respectively.

4.2.1 Scenarios and Parametrization

As previously mentioned, Bitswap (the baseline) uses a Distributed Hash Table (DHT) as an additional mechanism to locate providers for a given resource. This mechanism, referred to as the fallback mechanism, is triggered when an initial search attempt, conducted via flooding, fails to locate the desired resource within a set time interval. If the fallback succeeds, it provides nodes with additional providers for the resource. However, even if the fallback mechanism succeeds, the search in the unstructured network continues. For a more detailed explanation, please refer to Section 3.2.

We opted to test the protocols under two distinct scenarios: one where the fallback mechanism is available to nodes and another where it is not. In the first scenario, we retained the default behavior, allowing all protocols to utilize the DHT-based fallback. In the second scenario, although the DHT remains active within IPFS to maintain the normal behavior of components outside the scope of this work, nodes are restricted from using it during searches. This setup ensures a more precise evaluation of the success rate of our source-sharing mechanism without interference from the fallback mechanism.

Lastly, besides the changes to the IPFS configuration, we used the same parametrization for all protocols, the default values. The only exception is the *ProvSearchDelay* parameter in our protocol. This parameter serves as the initial threshold to determine when a search is considered unsuccessful, prompting nodes to engage the fallback mechanism if enabled (as described earlier). In our protocol, this value is set to 10 seconds, while Bitswap maintains the default value of 1 second.

The rationale for this adjustment is based on the fact that by utilizing nodes with alternative responses (i.e., the source-sharing mechanism), searches may require additional time to yield positive results. Our earlier experiments indicated that the default value

unfairly penalized such protocols by prematurely declaring the flood unsuccessful before the source-sharing mechanism could fully take effect.

4.2.2 Considered Performance Metrics

To evaluate the performance of our approach, we collected the following key metrics:

Success Rate: The percentage of searches that successfully located at least one resource satisfying the query within 60 seconds.

Latency: The time taken (in seconds) for a search to complete. A search is deemed complete when the first byte of the requested resource is received.

Processing Rate: The percentage of nodes in the system that participated in a search. Nodes are considered participants if they sent or received at least one message related to the search during the experiment.

Number of Messages: The total number of messages sent by all nodes throughout the experiment.

However, it is important to provide additional context.

First, all metrics are derived from data collected at the Bitswap level, which means only the messages exchanged within the unstructured overlay (the focus of this work) are measured. Messages sent or received at the structured overlay (such as those involving the DHT) are excluded from the results.

Second, the searches performed in the experiments had a timeout of 60 seconds. This means that any search exceeding this duration is considered failed, and such searches are assigned a latency value of 60 seconds.

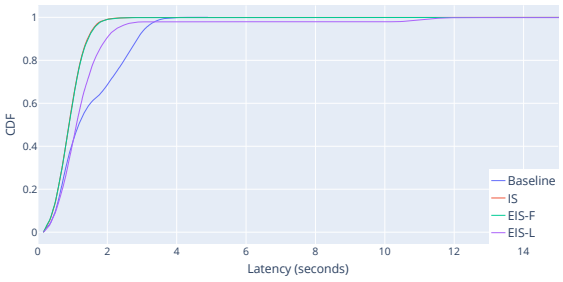
Finally, it is important to note that we do not differentiate between how a resource is found. In the first scenario, if a search locates the resource via the fallback mechanism, it is still considered successful under the success rate metric.

4.2.3 With fallback

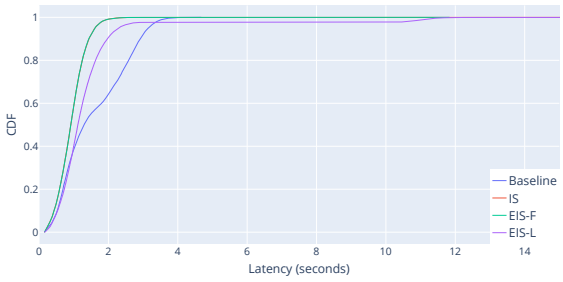
In this first scenario we maintained the DHT as a fallback mechanism for all protocols subjected to our experiments.

We purposely omitted the success rate from this scenario because all protocols performed successfully in nearly 100% of the searches (above 99.90%) in every experiment. Therefore, we will start by presenting the latency results. After that, we will provide the results for the processing rate. Finally, we will conclude with the number of messages.

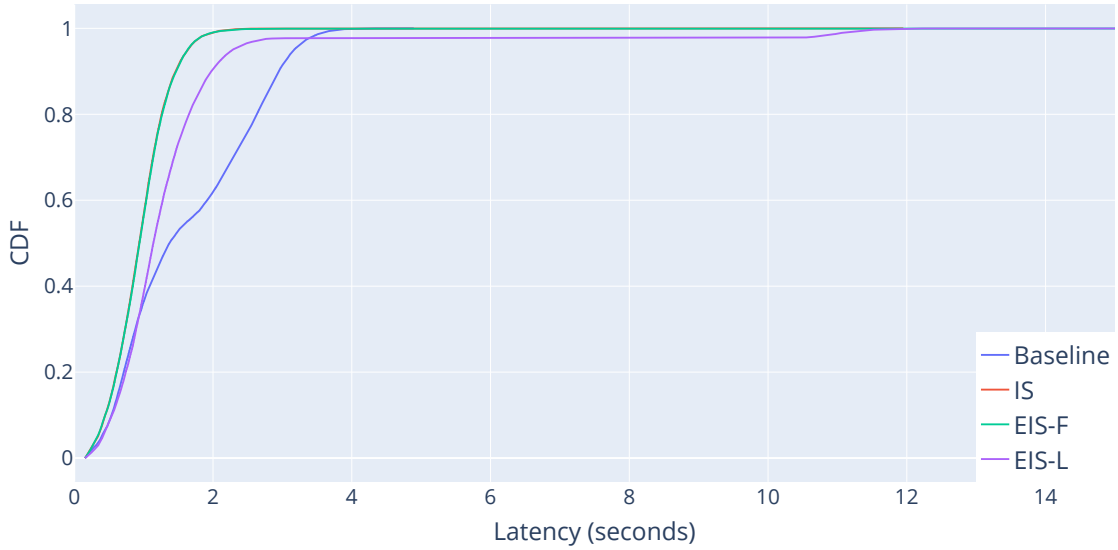
Figures 4.1 and 4.2 present the cumulative distribution function (CDF) of **latency** for each experiment. To enhance clarity, we have truncated the X-axis at 15 seconds, resulting in the omission of a small percentage of results, which will be addressed as necessary.



(a) 1000 resources



(b) 2000 resources



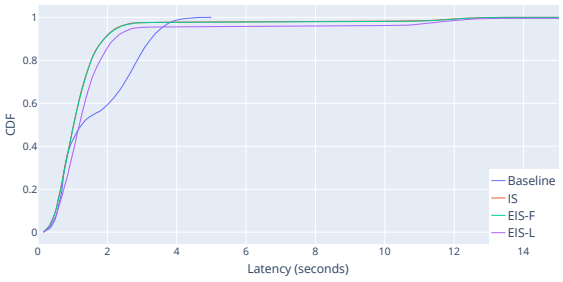
(c) 3000 resources

Figure 4.1: Latency CDF, Uniform distribution

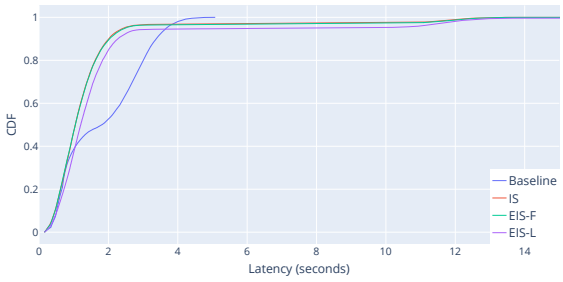
An initial observation reveals that, in terms of latency, both the *informed search* and *enhanced informed search* in *filtering mode* exhibit nearly identical values across all experiments. This similarity is expected; the only difference between the algorithms is that, in *filtering mode*, the *enhanced informed search* attempts to exclude additional neighbors from flooding by consulting the meta-indexes. To avoid redundancy, we will present the results of both algorithms together.

In Figure 4.1(c), which shows data from the Uniform distribution with 3000 resources, the *enhanced informed search* in *filtering mode* and the *informed search* (hereafter referred to as *EIS-F* and *IS*, respectively) have a steeper incline compared to both the *baseline* and the *enhanced informed search* in *lookup mode* (referred to as *EIS-L* hereafter). *EIS-F* and *IS* reach a slope close to the 99th percentile around the 2-second mark and achieve the median in less than 1 second (approximately 0.94 seconds).

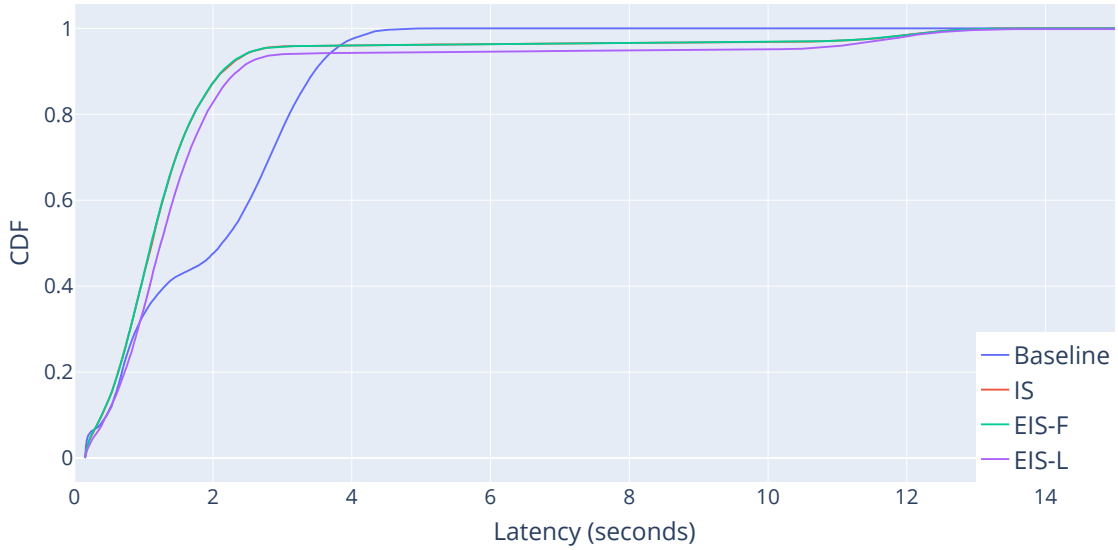
Conversely, *EIS-L* encounters a slope around the 97th percentile and only reaches the 99th percentile after 11 seconds, performing slightly worse than the baseline in approximately 2% of the slowest searches. The median value is approximately 1.14



(a) 1000 resources



(b) 2000 resources



(c) 3000 resources

Figure 4.2: Latency CDF, Zipfian distribution

seconds, with slightly inferior performance compared to the baseline in approximately 33% of the fastest searches (up to the 0.93-second mark).

For the baseline, the median value is around 1.37 seconds, reaching the 99th percentile around the 3.58-second mark. However, there is a noticeable decline in growth shortly after the median value.

Both Figure 4.1(a) and Figure 4.1(b) present a similar overall scenario. However, the informed search, the filtering mode of the enhanced version, and the baseline seem to benefit more from having fewer (and more replicated) resources.

In the 1000 and 2000 resource experiments, *EIS-F* and *IS* resolve approximately 60% and 57% of the searches under 1 second, respectively, compared to 56% in the 3000-resource experiment. The baseline performs better than *EIS-L* in approximately 36% and 42% of the fastest searches (in the 2000 and 1000 resource datasets, respectively), and the decline, although observed at roughly the same time (around the 1.5-second mark), affects fewer searches due to the steeper incline.

Figure 4.2 presents the latency of the experiments with the Zipfian distribution.

In general, solutions have more gradual inclines than those displayed in the Uniform distribution CDFs, indicating that they generally need more time to find and retrieve the first byte of the desired resources.

Figure 4.2(c) shows the latency data for the experiment with the Zipfian distribution and 3000 resources. Some previous observations still hold: the *EIS-F* has steeper inclines than other solutions and the smallest median value (roughly 1.12 seconds). The *baseline* performs slightly better than the *EIS-L* in approximately 31% of the fastest searches and the 5.8% slowest searches, displaying a decline in incline around the 1.5-second mark.

However, significant changes can be observed in this scenario. First, similar to what was previously seen in *EIS-L*, both the *EIS-F*) and *IS* fail to reach the 99th percentile before their slopes decline (which happens around the 95th percentile). Second, the baseline's decrease in performance around the 1.5-second mark is more pronounced and occurs before reaching the median value. The baseline's median latency of 2.3 seconds is relatively high compared to the 1.12 and 1.24 second values of the filtering and lookup modes, respectively. Finally, the baseline performs better than all other protocols in searches that take less than 0.28 seconds.

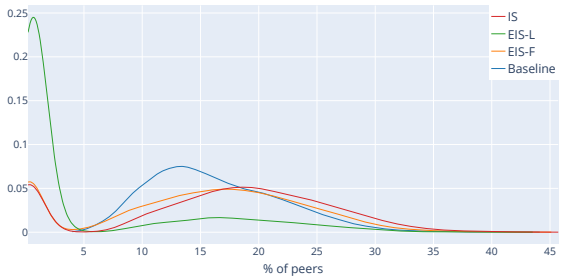
Similarly to what was observed in the Uniform distribution, Figures 4.2(a) and 4.2(b), which display the results of experiments with 1000 and 2000 resources respectively, show that fewer and more replicated resources benefit protocols in this metric. However, the baseline is more impacted by changes in the number of resources. In fact, it performs slightly better than *EIS-F* between the 0.75 and 0.85-second marks in the experiment with 1000 resources, and almost as well between the 0.7 and 0.8-second interval in the 2000 experiment.

There are two prevalent phenomena observed in all results: the baseline decrease around the 1.5-second mark and the slope observed in our solutions around the 95th percentile. Providing additional context may help readers better understand these phenomena.

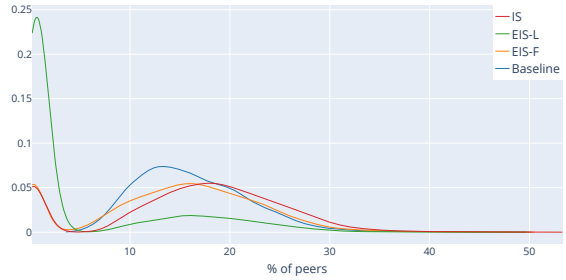
A key factor at play is that the baseline attempts to contact new neighbors and resorts to the fallback mechanism after 1 second (increasing this timeout linearly afterward), which can be described as a second phase of the search. Searches that take more than 1 second to be resolved by this algorithm are, for the most part, from resources that are not found in one of the neighbors. Both *IS* and *EIS* (in the two modes) resort to the same mechanisms (or enter the second phase) after 10 seconds (the timeout chosen for the experiments). The baseline tends to react faster to cases where the resource cannot be found in the first phase.

Figures 4.3 and 4.4 present the distributions of the **processing rate** across resource distributions and number of resources. At first glance, it is clear that informed solutions display a bimodal shape, while the baseline has a (slight) right-skewed shape. These results suggest that our protocol (in all its variants) has a relatively high probability of contacting a low percentage of peers while searching, whereas the baseline does not.

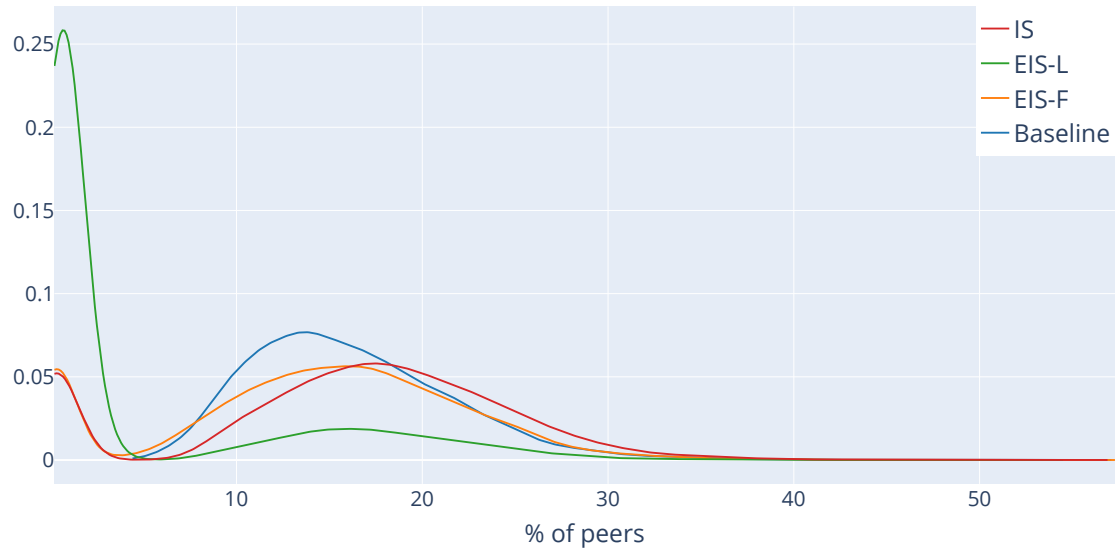
EIS-L has the highest probability of falling into this best-case scenario. In Figure 4.3(c), which shows the results of the Uniform distribution, it has a peak slightly above 0.25



(a) 1000 resources



(b) 2000 resources



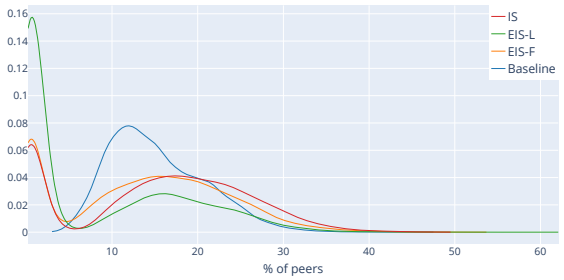
(c) 3000 resources

Figure 4.3: Processing rate distribution plot, Uniform distribution

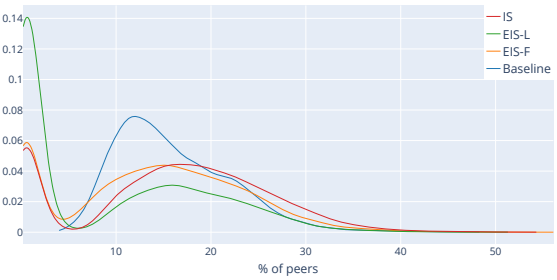
probability, making it 0.20 points higher than the peaks of both *EIS-F* and *IS*, which roughly share the same values. Figure 4.4(c) confirms that this trend continues in the Zipfian distribution, although the probability is significantly decreased.

This phenomenon can be explained by index (and meta-index in *EIS-L*) matches: if a search finds the location of a resource in the indexes, it avoids flooding and thus contacts fewer nodes. This occurs more frequently in *EIS-L* because it consults both the index and the meta-index to find the location of resources, whereas *EIS-F* and *IS* rely solely on the index.

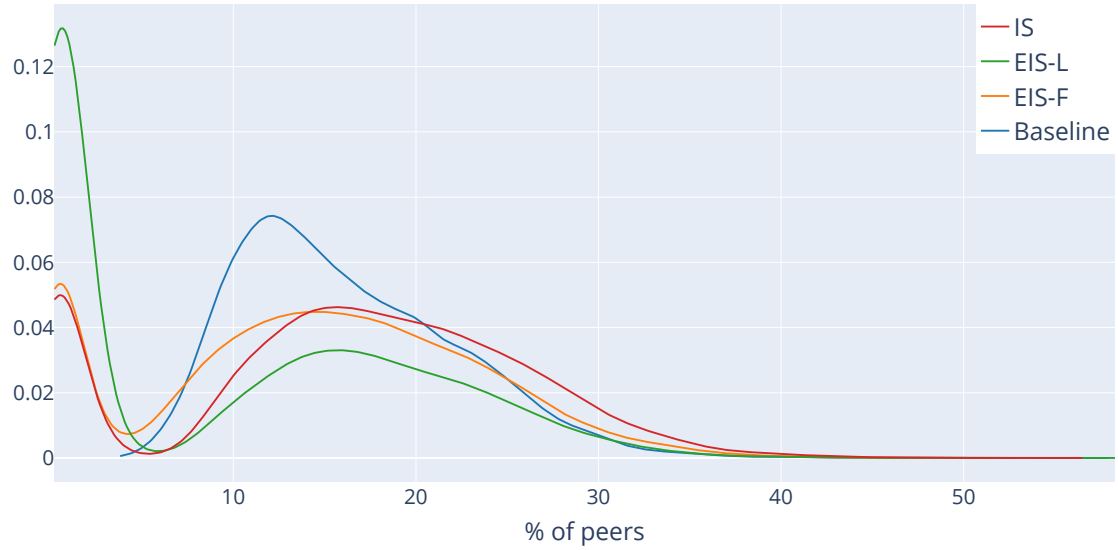
In cases where the search algorithms fail to leverage the index and at least a flood is performed, the results, which are represented in the second peaks, are mixed. On one hand, *IS* and *EIS-F* have, to some extent, a higher probability of contacting more peers than the baseline, suggesting that both can be worse than the baseline in the worst cases. *EIS-F* partially mitigates this, as evidenced by comparing the two, but the use of the meta-index as a filter fails to fully reverse this tendency. On the other hand, *EIS-L* performs no worse than the baseline in the vast majority of the searches, with the exception of two instances



(a) 1000 resources



(b) 2000 resources



(c) 3000 resources

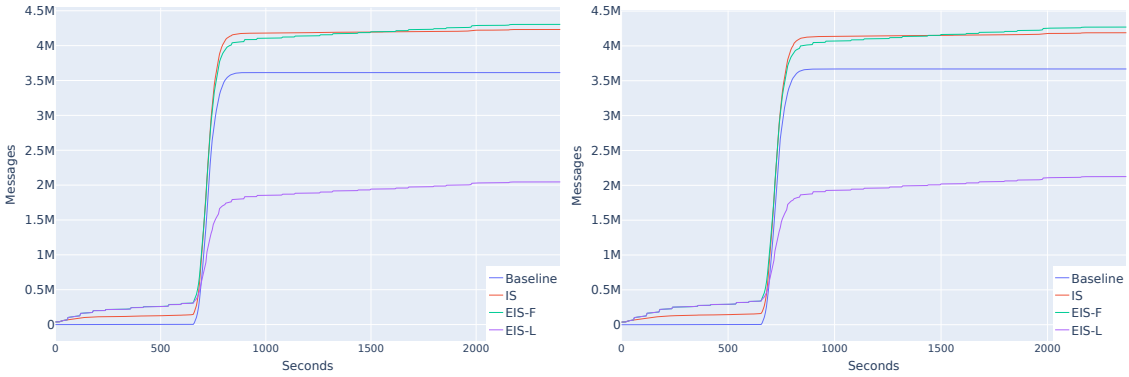
Figure 4.4: Processing rate distribution plot, Zipfian distribution

seen in Figure 4.4(c) and Figure 4.4(a).

A possible explanation for this is that because the flooding has a fanout equal to the number of neighbors, the membership size influences the processing rate, and the informed solutions tend to have a higher number of neighbors because they attempt to maintain connections to a configurable number of peers (15 in the experiments presented in this chapter) to establish a stable indexing overlay.

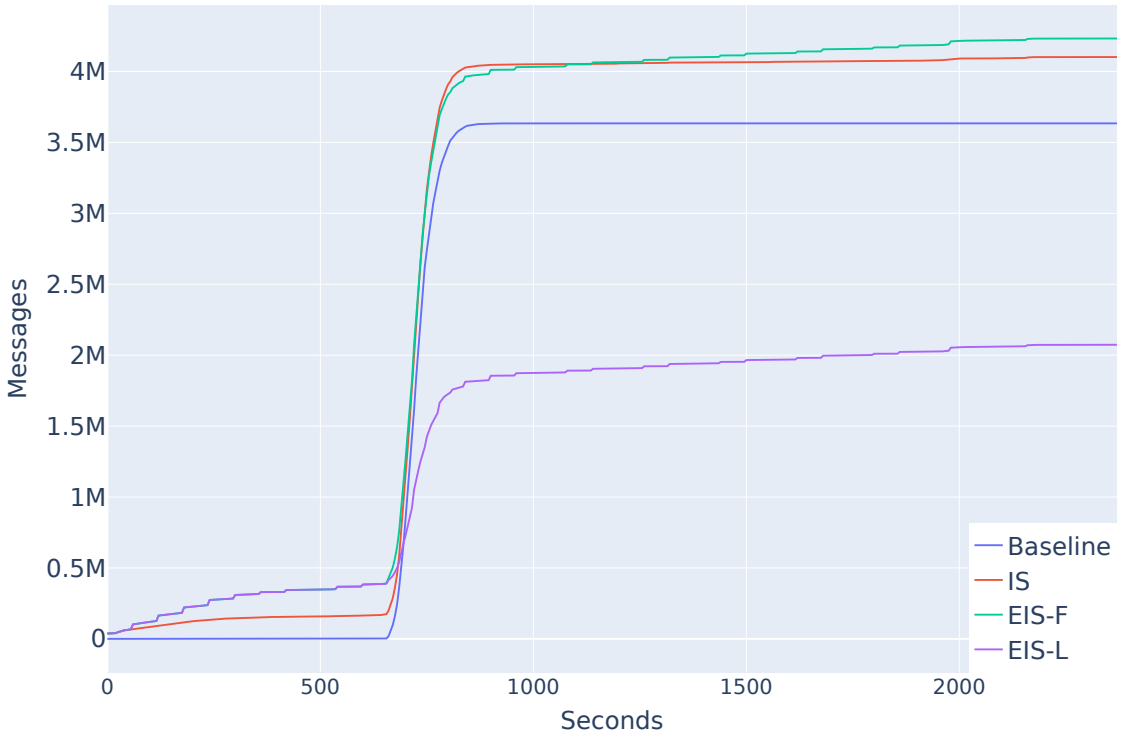
Another important metric for inferring the cost of the search is the number of messages sent. Although this metric is arguably related to the processing rate because protocols with high processing rates are more likely to exchange additional messages, it captures different aspects of the system's performance. First, the processing rate only captures the scenario during the search, failing to account for the cost of indexation, which mostly occurs outside the searches. Second, it is possible to contact a small number of nodes and still exchange a significant number of messages.

Figures 4.5 and 4.6 show the cumulative *number of messages* sent by the nodes in the experiments, aggregated in 5-second intervals. Initially, the number of messages sent by



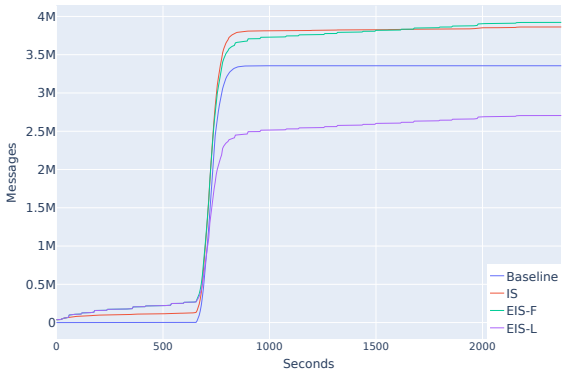
(a) 1000 resources

(b) 2000 resources

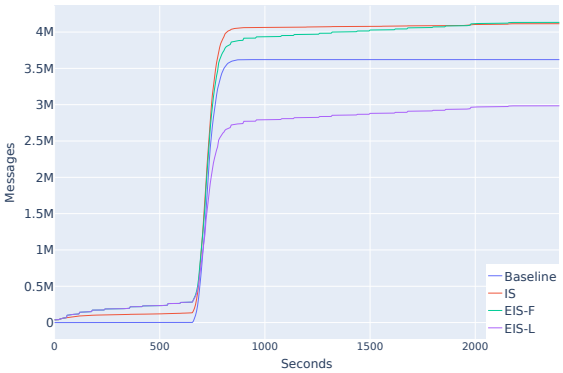


(c) 3000 resources

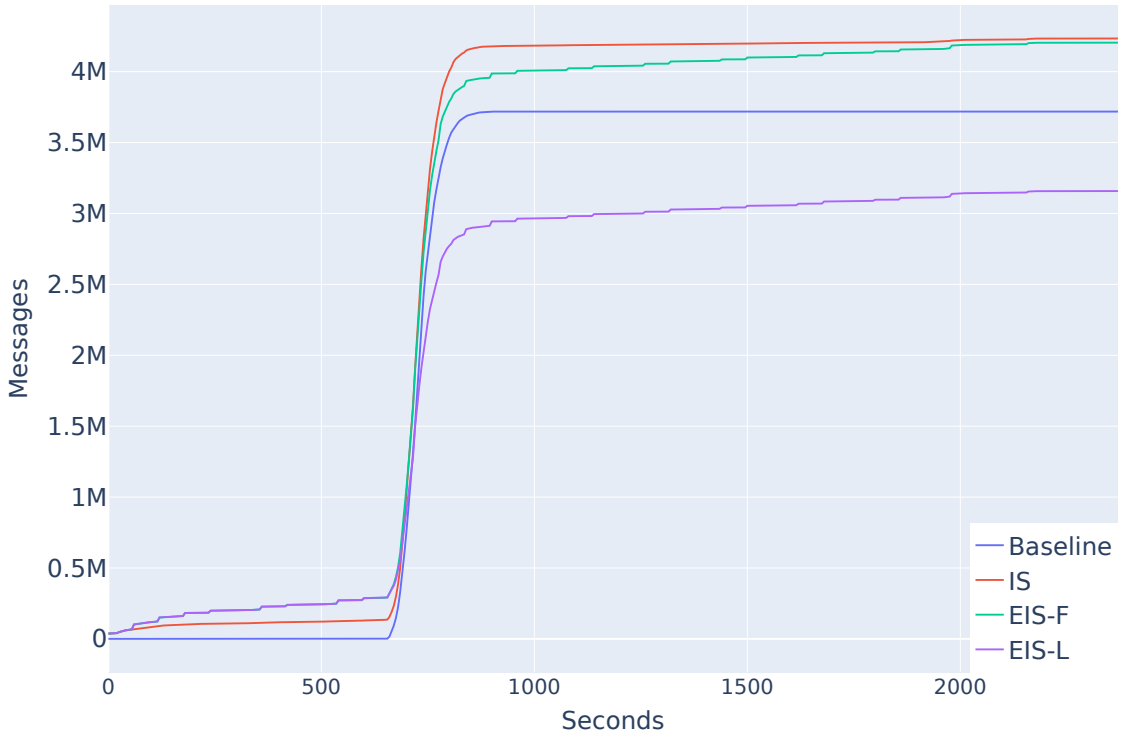
Figure 4.5: Cumulative number of messages, Uniform distribution



(a) 1000 resources



(b) 2000 resources



(c) 3000 resources

Figure 4.6: Cumulative number of messages, Zipfian distribution

the solutions that use the indexing layer increases. Every protocol experiences a sharp rise in the number of messages around the 650-second mark, which roughly corresponds to the point when the searches start. This trend continues for approximately 200 seconds before returning to the initial patterns.

The ever-growing number of messages in *IS* and *EIS* (in both modes) is caused by the indexing layer messages. Initially, nodes running our protocol need to send and receive information to build the local indexes. Once established in the network, nodes may continue receiving information to keep the indexes updated. Across all our experiments, the number of messages attributed to this phenomenon is relatively small compared to the search activity. However, it ultimately varies with the distribution and number of resources, and, as expected, between solutions.

Firstly, it is slightly higher in Uniform distributions, as shown in 4.5, than in Zipfian distributions, as can be seen in 4.6.

Secondly, it increases with the number of resources in the system, as observed by comparing Figures 4.5(a), 4.5(b) and 4.5(c) values before the search starts. This phenomenon seems to impact the uniform distribution disproportionately more, as the differences between Figures 4.6(a), 4.6(b) and 4.6(c) are smaller.

Lastly, solutions that use the meta-index have a higher number of these types of messages, which seems to continuously increase. This does not occur in solutions using only the index. The latter suggests that the meta-index is far more likely to require updates. A possible explanation is that, because the meta-index is composed of the indexes of close neighbors, a membership change triggers a meta-index update, which is then sent to all close neighbors. In contrast, an index update is only sent to the close neighbors if the resources that the node has changed.

A key idea behind our solutions is that the additional cost discussed above can be traded for reduced cost while searching. This trade-off appears to be achieved with varying degrees of success between different variants of our protocol and distributions. Another crucial aspect is how the cost is distributed over time. A burst of messages within a short time interval may overwhelm the network or its nodes, whereas the same cost spread over a longer interval requires fewer resources to be available at any one time.

Both *IS* and *EIS-F* fail to meet these expectations. They are more expensive than the baseline, and although at certain points during the searches the gap between them seems to diminish, by the end of the search, the gap has widened. This is presumably due to the cost of updating the indexes. Additionally, more expensive floods, as indicated by higher processing rates in some cases, may also negatively impact these solutions.

Comparing the two (*IS* and *EIS-F*) reveals that the filtering mechanism in *EIS-F* has subpar performance. Although it slightly reduces the cost while searching, particularly in Zipfian distributions where it performs better, the additional cost of maintaining the meta-index quickly catches up. After some time, the number of messages in *EIS-F* surpasses that of *IS*.

On the other hand, *EIS-L* effectively reduces the cost of the search, becoming the most

	1000	2000	3000
Baseline	72.15	64.98	63.47
IS	100.00	100.00	100.00
EIS-F	100.00	100.00	100.00
EIS-L	99.93	99.98	99.98

Table 4.1: Success Rate (%), Uniform distribution

	1000	2000	3000
Baseline	53.66	45.82	42.12
IS	99.87	99.69	99.57
EIS-F	99.87	99.62	99.53
EIS-L	99.65	99.46	99.37

Table 4.2: Success Rate (%), Zipfian distribution

efficient solution, with the cost spread over longer intervals. In the Uniform distribution, as shown in Figure 4.5 it can be seen that at the 900-second mark (approximately at the end of the search phrase), *EIS-L* has sent roughly half the number of messages compared to the baseline in all experiments. In the Zipfian distribution, depicted in Figure 4.6 the gap is reduced; however, the solution still maintains its positive results.

4.2.4 Without fallback

In this scenario, unlike the previous one, protocols were prevented from using the DHT as a fallback mechanism. As stated earlier, there are two main motivations for testing this scenario. First, we aim to assess the impact of the fallback mechanism on the protocols. Second, we intend to explore the effectiveness of the source sharing mechanism. Additionally, we seek to verify our assumption that the baseline protocol is heavily dependent on the fallback mechanism. If this dependency is confirmed, we aim to provide an alternative and fairer framework to evaluate our protocol.

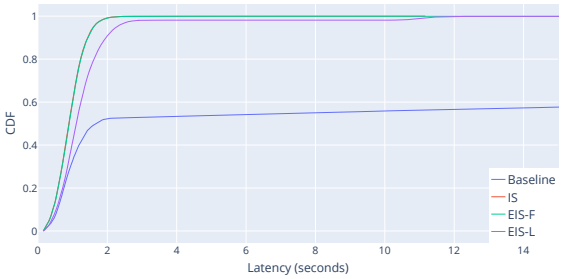
Tables 4.1 and 4.2 show the **success rate** in the two resource distributions used in the experiments. This metric was omitted from the previous scenario because every protocol performed successfully nearly 100% (above 99.90%) of the searches in every experiment. However, as can be seen, disabling the DHT leads to significantly different results.

Our protocol, across all its variants, achieves almost 100% success regardless of the distribution and amount of resources, thanks to the source-sharing mechanism. In comparison, the baseline, which lacks such a mechanism, fails to achieve this goal in approximately 28% to 58% of the searches, depending on the scenario.

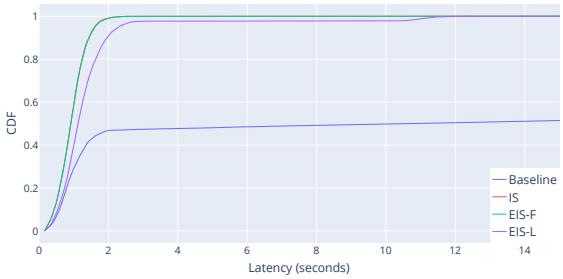
These results suggest that, at least in our experiments, source sharing effectively extends the reach of the searches and greatly reduces the need for a fallback mechanism

in protocols that employ it. On the other hand, the baseline appears to rely heavily on the DHT to complete a significant percentage of searches.

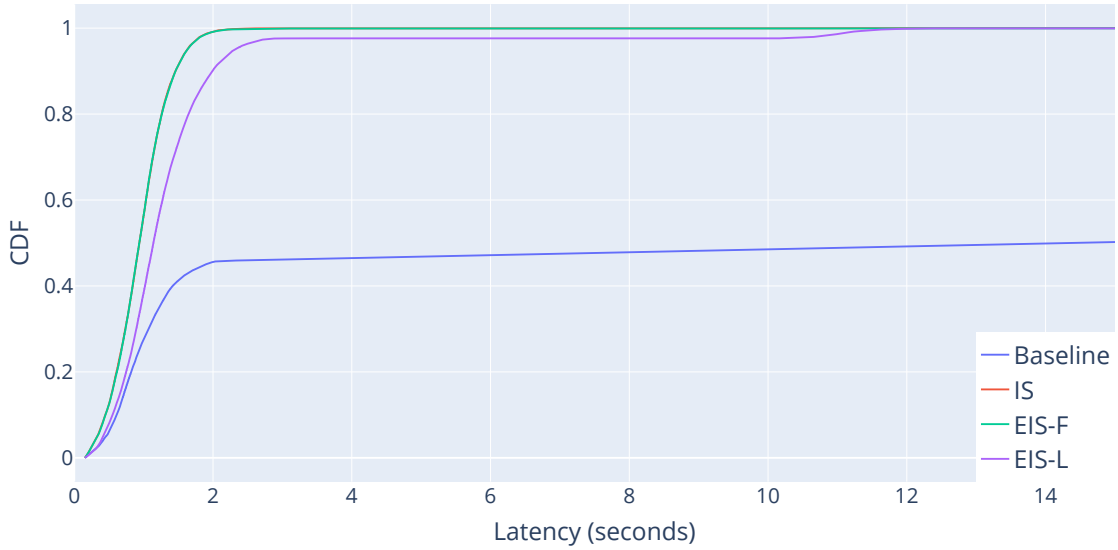
Similarly to other metrics, in the Zipfian distribution, presented in Table 4.2, every protocol performs worse than in the Uniform distribution (shown in 4.1). Additionally, increasing the number of resources slightly reduces the success rate.



(a) 1000 resources



(b) 2000 resources



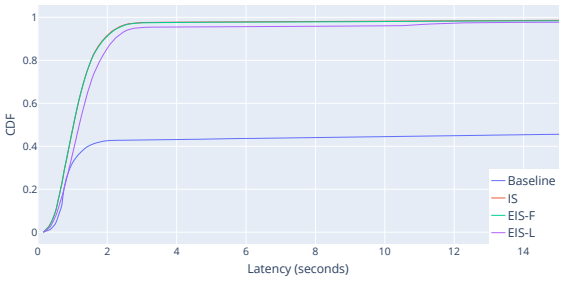
(c) 3000 resources

Figure 4.7: Latency CDF, Uniform distribution

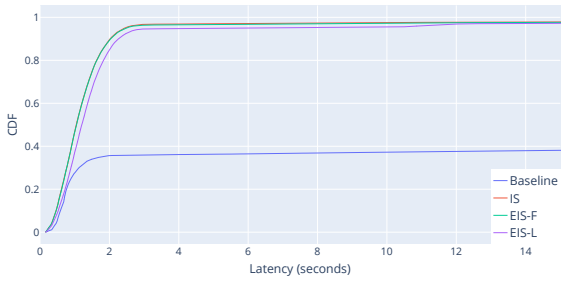
Figures 4.7 and 4.8 show the **latency** CDFs. As before, we cropped the X-axis of the figures at 15 seconds to enhance clarity, which omits results beyond this value. It is important to note that failed searches were assigned a latency of 60 seconds, which is the timeout value used in our experiments.

A comparison between Figure 4.7 and Figure 4.1 (presented previously, also reporting latency) shows that in the Uniform distributions, there were no significant changes in our solution, regardless of the variant. The lack of the fallback mechanism did not lead to a significant increase in latency, reaffirming the effectiveness of our protocol without it.

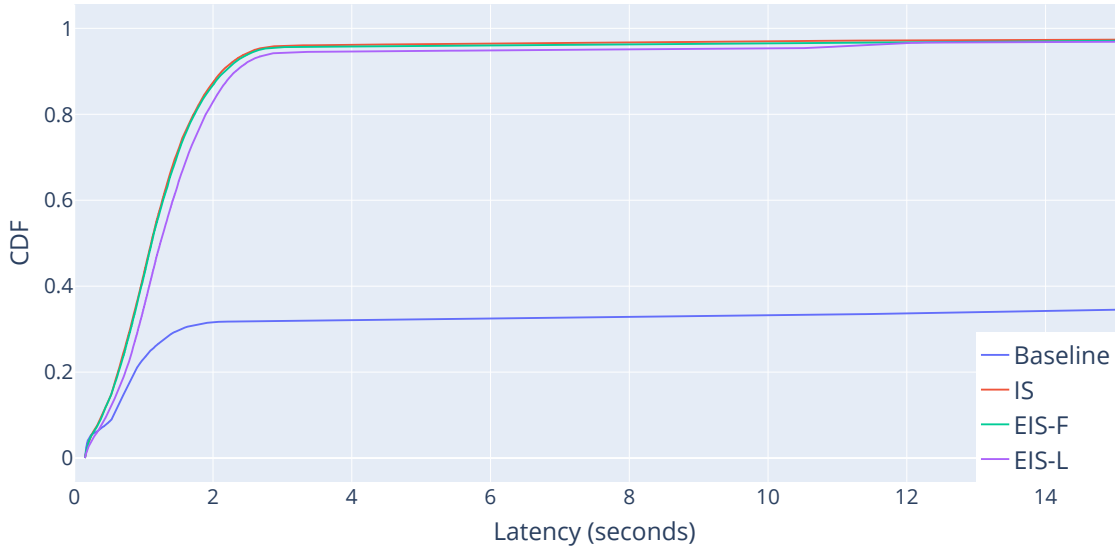
In contrast, the baseline's growth stalls at the two-second mark, with a small percentage of searches exhibiting progressively higher latencies until they reach the timeout, resulting



(a) 1000 resources



(b) 2000 resources



(c) 3000 resources

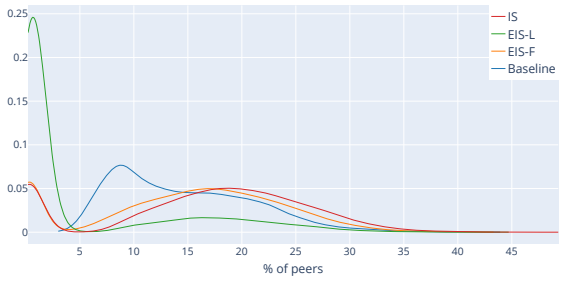
Figure 4.8: Latency CDF, Zipfian distribution

in failed searches.

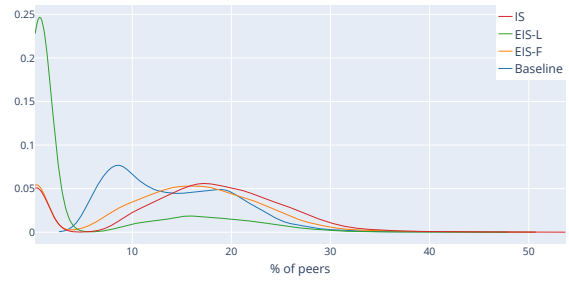
Figure 4.8 present a slightly different scenario using a Zipfian distribution for resource popularity. The baseline follows the same trend, although fewer searches are completed within two seconds, resulting in a worse performance. The solutions employing the source sharing mechanism display strong similarities to the values observed in the same experiment with the fallback mechanism available (presented in Figure 4.2). However, in this case, they fail to reach the 99th percentile in less than 15 seconds, though they eventually do so as indicated by the success rate values.

It is possible that, in this distribution, smaller timeouts between floods or a more sophisticated approach to managing timeouts could reduce latency in these cases.

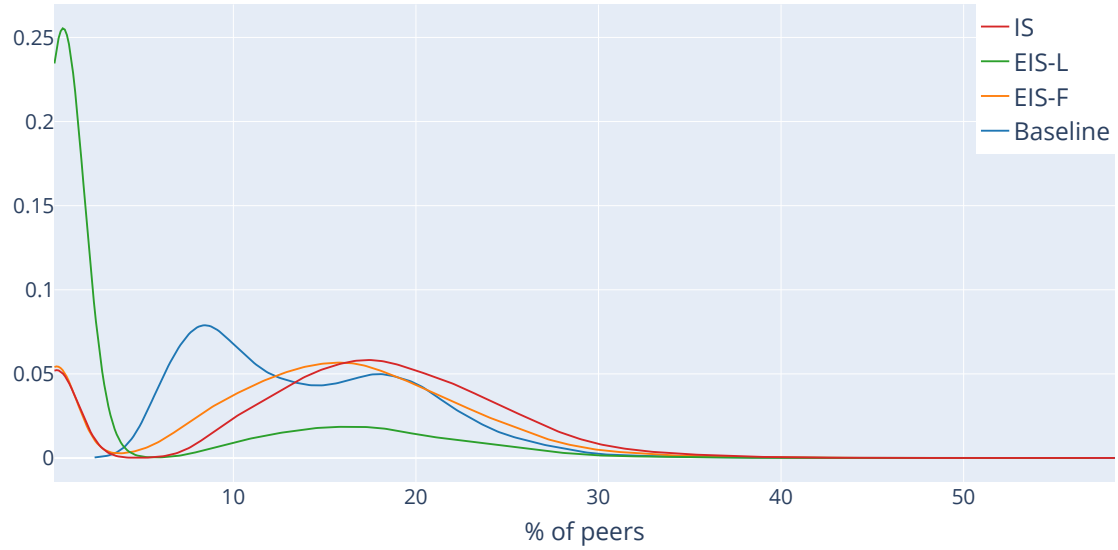
Figures 4.9 and 4.10 show the **processing rates** of the experiments without using the DHT as a fallback mechanism. Similar to other metrics presented in this analysis, our solutions (IS, EIS-F, and EIS-L) exhibit an overall similar scenario to the experiments with the fallback mechanism. We consider this a positive result, though it does not provide additional insights beyond what was already observed.



(a) 1000 resources



(b) 2000 resources

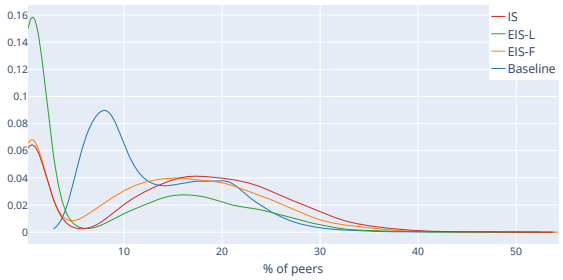


(c) 3000 resources

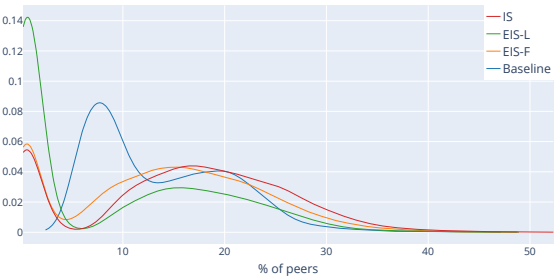
Figure 4.9: Processing rate distribution plot, Uniform distribution

In contrast, the baseline processing rates are lower without the DHT as a fallback, with peaks that fall close to the 7% mark across all experiments and resource quantities. At first glance, these results might seem positive; however, it is essential to consider the impact of failed searches on this metric. For instance, if nodes repeatedly fail to find a resource, and considering that searches in the experiments are performed sequentially, the search phase of the experiments will be prolonged. Consequently, the connection manager may have a more pronounced effect on the number of neighbors each node has, leading to fewer neighbors and thus lower processing rates. Moreover, even accounting for such effects, it is debatable how fair it is to compare algorithms with significantly different success rates. For example, if a node does not contact other nodes while searching, it would have a processing rate of 0%; however, it would fail to find 100% of the resources.

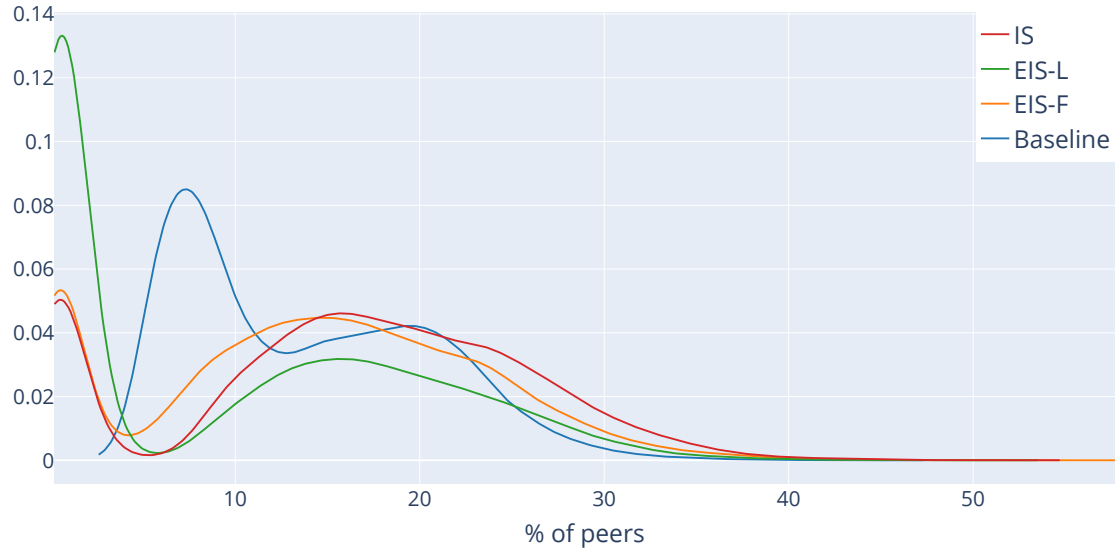
Finally, Figures 4.11 and 4.12 depict the cumulative **number of messages** sent during the experiment. Similar to the previous observations, *IS* and *EIS* in both modes exhibit comparable results to the experiments with the DHT. This suggests that removing the fallback mechanism does not significantly increase the number of messages sent by the



(a) 1000 resources



(b) 2000 resources



(c) 3000 resources

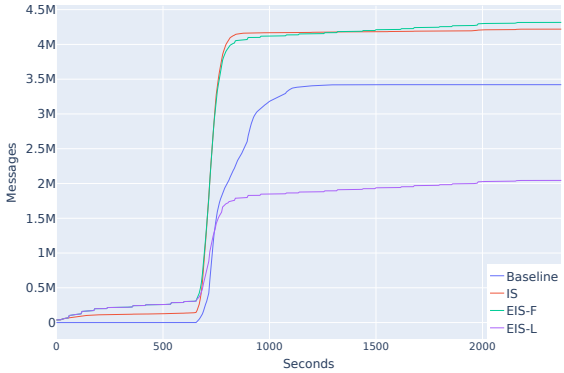
Figure 4.10: Processing rate distribution plot, Zipfian distribution

protocols, indicating, again, that using the DHT as a fallback mechanism may not be necessary in such solutions. Despite a reduced gap, *EIS-L* remains the most efficient protocol in terms of message count in our experiments.

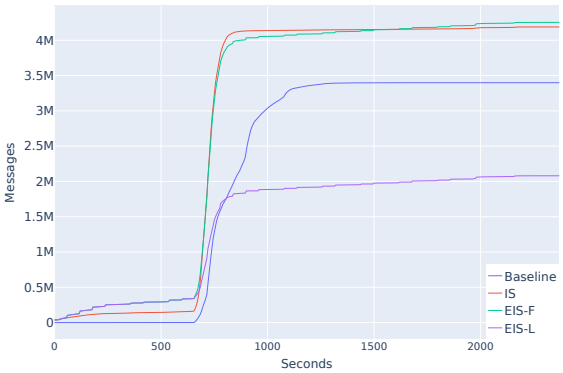
In contrast, the baseline shows different outcomes. Firstly, considering that the protocol does not send messages at this layer after searching, it is evident that it spends more time of the experiment in the search phase, as indicated by where the number of messages stabilizes. Secondly, although it sends fewer messages compared to experiments with the fallback mechanism available, this might be influenced by the phenomenon hypothesized in the processing rate analysis, and the same criticism applies.

4.3 Summary

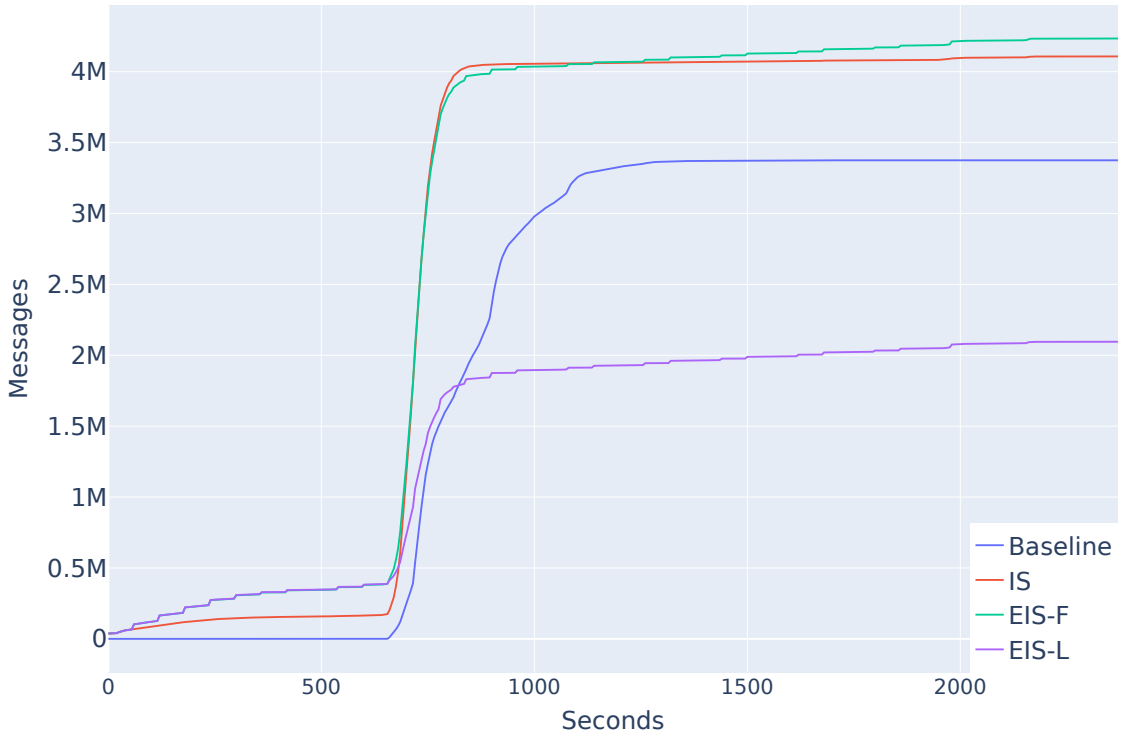
In this chapter, we conducted an experimental evaluation of the two search mechanisms proposed in this thesis. We compared the performance of these solutions with each other as well as with the baseline search mechanism currently used in IPFS. The results confirmed



(a) 1000 resources

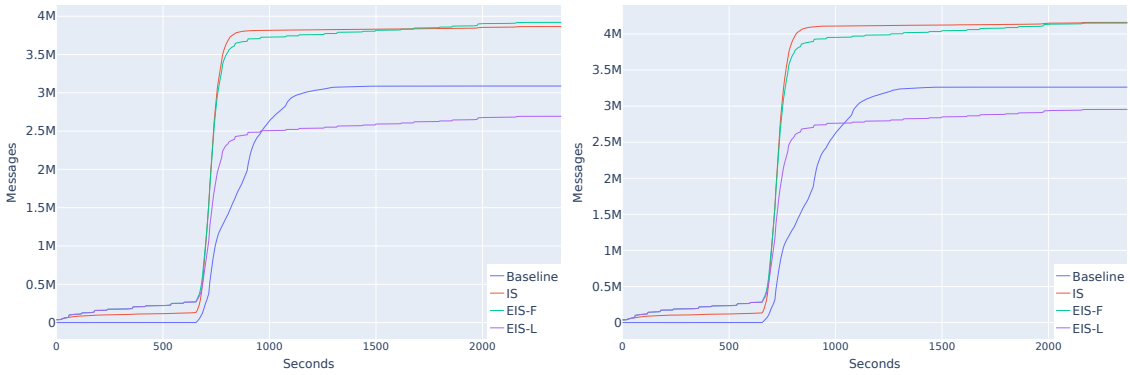


(b) 2000 resources



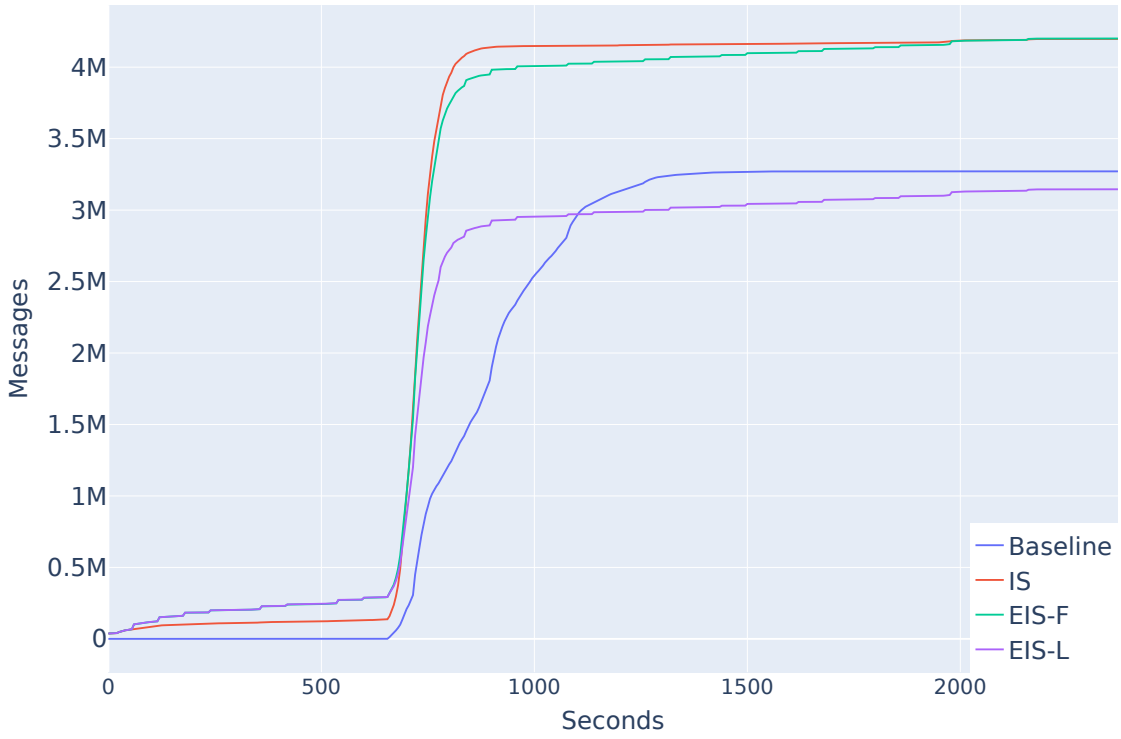
(c) 3000 resources

Figure 4.11: Cumulative number of messages, Uniform distribution



(a) 1000 resources

(b) 2000 resources



(c) 3000 resources

Figure 4.12: Cumulative number of messages, Zipfian distribution

the effectiveness of our approach, especially the enhanced version in lookup mode, which showed notable improvements in search latency, message overhead, processing efficiency, and success rates in scenarios where the DHT was not utilized. These outcomes highlight the potential of our approach to reduce dependency on the DHT for efficient resource discovery.

5

Conclusions and Future Work

5.1 Conclusion

The resurgence of decentralization, exemplified by the development and adoption of systems like IPFS, has brought peer-to-peer (P2P) architectures back into focus. However, despite their growing popularity, these systems face significant challenges, largely due to their highly distributed nature. One of the most critical challenges lies in implementing efficient services, such as search algorithms. The design of these services is often deeply interconnected with other architectural elements, such as the overlay network topology. Effective search functionality is crucial not only for enhancing user experience but also for ensuring the system's scalability and overall viability.

In this thesis, we explored the foundational components of P2P systems, including the construction and maintenance of overlay networks. We examined how these networks, with their varied architectures, facilitate peer communication and collaboration. Additionally, we analyzed the core principles underlying search mechanisms in P2P networks, categorizing and reviewing classical algorithms and strategies. We also investigated two widely-used systems, one classical and one more recent, highlighting how their search functionalities integrate into the broader system architecture.

Building on these insights, we proposed a solution that introduces a new layer, allowing peers to share information with some of their neighbors in the form of an index. This index contains details about the resources a node can provide, and is extended by a meta-index that indicates what resources their connected peers may offer and, ultimately, what the node may help to provide. To enable this, we introduced a mechanism that leverages this information, allowing nodes to collaborate in searches by using the received indexes to suggest potential sources for the resources a neighbor is searching for, thus increasing the search horizon.

Finally, we developed an informed search algorithm that guides queries toward useful peers rather than flooding the network, reducing the overall cost of search. Our solution was introduced iteratively: first with only first-order information, and later with significant enhancements made possible by second-order information, which is leveraged in one of

two modes. We implemented our solution on IPFS by modifying and expanding modules of its stack, most notably Bitswap, to seamlessly integrate our approach into the existing system.

Our experimental evaluation, conducted through the emulation of a network similar to IPFS, compared three versions of our solution with the baseline unstructured search solution of IPFS. The results validated the efficacy of our approach, particularly the enhanced version operating in lookup mode. This version demonstrated significant improvements in latency, the number of messages sent during searches, processing rate, and the success rate when the DHT was not used. These findings confirm the potential of our approach to reduce reliance on the DHT.

5.2 Future Work

In this section, we outline potential future developments for our proposed solution, as well as new approaches that could build on this work. Additionally, we suggest further evaluation scenarios to gain a deeper understanding of the implications and performance of our system.

Biased indexing layer Currently, the indexing layer allows each node to add a fixed number of randomly selected neighbors to its partial view, resulting in an unstructured overlay. However, as highlighted in Chapter 2, biased overlays can improve communication efficiency. An interesting direction for future research would be to introduce biased selection for determining which nodes exchange information. For instance, nodes with more resources or nodes that receive more indexes could be favored when forming connections. This would help in optimizing resource dissemination, as larger or more comprehensive indexes would provide greater utility.

Sophisticated timeouts In the current prototype, as mentioned in Chapter 4 the proposed search algorithms utilize a static timeout of 10 seconds to determine when a search has failed, after which a second attempt is made. The baseline mechanism uses a shorter default timeout of 1 second, allowing it to respond more quickly. A potential improvement would be to implement dynamic timeout mechanisms that adjust based on whether matches are found in the indexes or meta-indexes. By adapting the timeout based on the presence or absence of results, the system could respond more efficiently to worst-case search scenarios.

Broader search horizon An intriguing avenue for future research would involve extending the use of indexes and meta-indexes in the source-sharing process, further broadening the search horizon. Meta-indexes could be leveraged to improve search results, particularly in networks with longer average shortest paths. However, we note that the risk to increase the overhead for maintaining the indexing information can grow beyond acceptable.

More dynamic scenarios To better evaluate the performance of the proposed solution, it would be valuable to test the system in more realistic scenarios. This includes using larger networks with more nodes and longer evaluation periods. Additionally, while the current evaluation considers index updates when nodes acquire new resources, it would be important to examine scenarios where files are also removed from the system during searches.

Bibliography

[1] L. A. Adamic et al. "Search in power-law networks". In: *Physical review E* 64.4 (2001), p. 046135 (cit. on p. 14).

[2] J. Alveirinho et al. "Flexible and efficient resource location in large-scale systems". In: *Proceedings of the 4th ACM/SIGOPS Workshop on Large-Scale Distributed Systems and Middleware, LADIS 2010* (2010), pp. 55–60. DOI: 10.1145/1859184.1859199 (cit. on p. 9).

[3] M. Armbrust et al. "A View of Cloud Computing". In: *Commun. ACM* 53.4 (2010-04), pp. 50–58. ISSN: 0001-0782. DOI: 10.1145/1721654.1721672. URL: https://doi.org/10.1145/1721654.1721672 (cit. on p. 1).

[4] O. Ascigil et al. "A native content discovery mechanism for the information-centric networks". In: *ICN 2017 - Proceedings of the 4th ACM Conference on Information Centric Networking* (2017-09), pp. 145–155. DOI: 10.1145/3125719.3125734 (cit. on p. 12).

[5] H. Barjini et al. "Shortcoming, problems and analytical comparison for flooding-based search techniques in unstructured P2P networks". In: *Peer-to-Peer Networking and Applications* 5 (1 2012-03), pp. 1–13. ISSN: 19366442. DOI: 10.1007/S12083-011-0101-Y/FIGURES/13. URL: https://link.springer.com/article/10.1007/s12083-011-0101-y (cit. on pp. 6, 16, 17).

[6] J. Benet. "IPFS - Content Addressed, Versioned, P2P File System". In: (2014-07). URL: https://arxiv.org/abs/1407.3561v1 (cit. on pp. 1, 9, 11, 24, 26).

[7] H. Cai, P. Ge, and J. Wang. "Applications of bloom filters in peer-to-peer systems: Issues and questions". In: *2008 International Conference on Networking, Architecture, and Storage*. IEEE. 2008, pp. 97–103 (cit. on p. 34).

[8] Y. Chawathe et al. "Making Gnutella-like P2P Systems Scalable". In: *Computer Communication Review* 33 (4 2003), pp. 407–418. ISSN: 01464833. DOI: 10.1145/863955.864000 (cit. on pp. 8, 18, 21).

[9] P. Á. Costa. "Practical aggregation in the edge". MA thesis. 2018 (cit. on pp. 1, 5, 10).

[10] P. Á. Costa, J. Leitão, and Y. Psaras. "Studying the workload of a fully decentralized Web3 system: IPFS". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 13909 LNCS (2022-12), pp. 20–36. ISSN: 16113349. DOI: 10.1007/978-3-031-35260-7_2. URL: https://arxiv.org/abs/2212.07375v1 (cit. on pp. 25, 42).

[11] A. Crespo and H. Garcia-Molina. "Routing indices for peer-to-peer systems". In: *Proceedings - International Conference on Distributed Computing Systems* (2002), pp. 23–32. DOI: 10.1109/ICDCS.2002.1022239 (cit. on p. 21).

[12] A. Crespo and H. Garcia-Molina. "Semantic overlay networks for P2P systems". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3601 LNAI (2005), pp. 1–13. ISSN: 03029743. DOI: 10.1007/11574781_1. URL: https://www.researchgate.net/publication/2565172_Semantic_Overlay_Networks_for_P2P_Systems (cit. on p. 8).

[13] A. Q. Csail, A. Torralba, and V. St. "Recognizing Indoor Scenes". In: () (cit. on p. 42).

[14] R. Dorrigiv, A. Lopez-Ortiz, and P. Pralat. "Search algorithms for unstructured peer-to-peer networks". In: *32nd IEEE Conference on Local Computer Networks (LCN 2007)*. IEEE. 2007, pp. 343–352 (cit. on p. 18).

[15] "FastTrack". In: *Securing Im and P2P Applications for the Enterprise* (2005-01), pp. 319–357. DOI: 10.1016/B978-159749017-7/50017-3 (cit. on p. 9).

[16] J. Gao and P. Steenkiste. "An adaptive protocol for efficient support of range queries in DHT-based systems". In: *Proceedings of the 12th IEEE International Conference on Network Protocols, 2004. ICNP 2004.* 2004, pp. 239–250. DOI: 10.1109/ICNP.2004.1348114 (cit. on p. 13).

[17] P. Garbacki, D. H. Epema, and M. V. Steen. "Optimizing peer relationships in a super-peer network". In: *Proceedings - International Conference on Distributed Computing Systems* (2007). DOI: 10.1109/ICDCS.2007.126 (cit. on p. 9).

[18] *Gnutella - Stable - 0.4*. URL: http://rfc-gnutella.sourceforge.net/developer/stable/index.html#t3-2-3 (cit. on p. 22).

[19] *Gnutella Protocol Development*. URL: http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html (cit. on p. 23).

[20] S. Henningsen et al. "Mapping the interplanetary filesystem". In: *2020 IFIP Networking Conference (Networking)*. IEEE. 2020, pp. 289–297 (cit. on p. 26).

[21] J. Hongbo and J. Shudong. "Exploiting dynamic querying like flooding techniques in unstructured peer-to-peer networks". In: *Proceedings - International Conference on Network Protocols, ICNP* 2005 (2005), pp. 122–131. ISSN: 10921648. DOI: 10.1109/ICNP.2005.17 (cit. on pp. 13, 14, 17, 23).

[22] V. Jacobson et al. "Networking named content". In: *CoNEXT'09 - Proceedings of the 2009 ACM Conference on Emerging Networking Experiments and Technologies* (2009), pp. 1–12. DOI: 10.1145/1658939.1658941 (cit. on p. 12).

[23] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. "A Local Search Mechanism for Peer-to-Peer Networks". In: (2002) (cit. on pp. 16, 19).

[24] E. Khatibi and M. Sharifi. "Resource discovery mechanisms in pure unstructured peer-to-peer systems: a comprehensive survey". In: *Peer-to-Peer Networking and Applications* 14.2 (2021), pp. 729–746 (cit. on pp. 2, 15).

[25] G. Korpal and D. Scott. "Decentralization and web3 technologies". In: *Authorea Preprints* (2023) (cit. on p. 1).

[26] J. Leitão. "Gossip-based broadcast protocols". MA thesis. Master's thesis, University of Lisbon, 2007 (cit. on pp. 5, 10).

[27] J. Leitão. "Topology management for unstructured overlay networks". PhD thesis. 2012 (cit. on p. 14).

[28] J. Leitão and L. Rodrigues. "Overnesia: A resilient overlay network for virtual super-peers". In: *Proceedings of the IEEE Symposium on Reliable Distributed Systems* 2014-January (2014), pp. 281–290. ISSN: 10609857. DOI: 10.1109/SRDS.2014.40 (cit. on pp. 9, 14).

[29] J. Leitão et al. "X-BOT: A protocol for resilient optimization of unstructured overlay networks". In: *IEEE Transactions on Parallel and Distributed Systems* 23 (11 2012), pp. 2175–2188. ISSN: 10459219. DOI: 10.1109/TPDS.2012.29 (cit. on p. 8).

[30] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/main/template.pdf (cit. on p. i).

[31] Q. Lv et al. "Search and replication in unstructured peer-to-peer networks". In: *Proceedings of the 16th international conference on Supercomputing*. 2002, pp. 84–95 (cit. on pp. 14, 16, 18).

[32] V. March and Y. M. Teo. "Multi-Attribute Range Queries on Read-Only DHT". In: *Proceedings of 15th International Conference on Computer Communications and Networks*. 2006, pp. 419–424. DOI: 10.1109/ICCCN.2006.286312 (cit. on p. 13).

[33] P. Maymounkov and D. Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2429 (2002), pp. 53–65. ISSN: 16113349. DOI: 10.1007/3-540-45748-8_5. URL: https://link.springer.com/chapter/10.1007/3-540-45748-8_5 (cit. on pp. 2, 7, 25).

[34] Mehr-Un-Nisa et al. "Comparative analysis of unstructured P2P file sharing networks". In: *PervasiveHealth: Pervasive Computing Technologies for Healthcare* (2019-04), pp. 148–153. ISSN: 21531633. DOI: 10.1145/3325917.3325952 (cit. on pp. 6, 9).

[35] E. Meshkova et al. "A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks". In: *Computer Networks* 52.11 (2008-08), pp. 2097–2128. ISSN: 1389-1286. DOI: 10.1016/J.COMNET.2008.03.006 (cit. on pp. 9, 10).

[36] S. Ratnasamy et al. "A Scalable Content-Addressable Network". In: *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '01* (2001). DOI: 10.1145/383059 (cit. on pp. 2, 7, 8).

[37] M. Ripeanu. "Peer-to-peer architecture case study: Gnutella network". In: *Proceedings first international conference on peer-to-peer computing*. IEEE. 2001, pp. 99–100 (cit. on pp. 14, 23).

[38] A. D. L. Rocha, D. Dias, and Y. Psaras. "Accelerating Content Routing with Bitswap: A multi-path file transfer protocol in IPFS and Filecoin". In: () (cit. on pp. 9, 11, 25, 29).

[39] A. Rowstron and P. Druschel. "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems". In: *Middleware 2001: IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12–16, 2001 Proceedings 2*. Springer. 2001, pp. 329–350 (cit. on p. 7).

[40] S. Saroiu, P. K. Gummadi, and S. D. Gribble. "Measurement study of peer-to-peer file sharing systems". In: *Multimedia computing and networking 2002*. Vol. 4673. International Society for Optics and Photonics. 2001, pp. 156–170 (cit. on p. 23).

[41] R. Sequeira, P. Á. Costa, and J. Leitão. "Pesquisa Descentralizada: Proposta de otimização da solução não estruturada do IPFS". In: () (cit. on p. 3).

[42] I. Stoica et al. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications". In: (2001). DOI: 10.1145/964723. URL: http://pdos.lcs.mit.edu/chord/ (cit. on pp. 2, 7).

[43] D. Stutzbach and R. Rejaie. "Characterizing the two-tier Gnutella topology". In: *ACM SIGMETRICS Performance Evaluation Review* 33.1 (2005), pp. 402–403 (cit. on pp. 14, 17, 24).

[44] D. Stutzbach, R. Rejaie, and S. Sen. "Characterizing unstructured overlay topologies in modern P2P file-sharing systems". In: *IEEE/ACM Transactions on Networking (TON)* 16 (2 2008-04), pp. 267–280. ISSN: 10636692. DOI: 10.1109/TNET.2007.900406. URL: https://dl.acm.org/doi/abs/10.1109/TNET.2007.900406 (cit. on pp. 14, 17, 22–24).

[45] *TC(8) — Linux manual page*. URL: https://man7.org/linux/man-pages/man8/tc.8.html (cit. on p. 41).

[46] S. M. Thampi et al. "Survey of search and replication schemes in unstructured p2p networks". In: *arXiv preprint arXiv:1008.1629* (2010) (cit. on pp. 5, 8, 14, 15).

[47] A. S. Tigelaar, D. Hiemstra, and D. Trieschnigg. "Peer-to-peer information retrieval: An overview". In: *ACM Transactions on Information Systems (TOIS)* 30.2 (2012), pp. 1–34 (cit. on pp. 2, 7, 9, 11).

[48] D. Tsoumakos and N. Roussopoulos. "Adaptive probabilistic search for peer-to-peer networks". In: *Proceedings - 3rd International Conference on Peer-to-Peer Computing, P2P 2003* (2003), pp. 102–109. DOI: 10.1109/PTP.2003.1231509 (cit. on p. 20).

[49] D. Tsoumakos and N. Roussopoulos. "Analysis and comparison of p2p search methods". In: *Proceedings of the 1st international conference on Scalable information systems*. 2006, 25–es (cit. on p. 19).

[50] *Usage ideas and examples*. URL: https://docs.ipfs.io/concepts/usage-ideas-examples/ (cit. on p. 24).

[51] C. J. Wu, K. H. Yang, and J. M. Ho. "AntSearch: An ant search algorithm in unstructured peer-to-peer networks". In: *Proceedings - International Symposium on Computers and Communications* (2006), pp. 429–434. ISSN: 15301346. DOI: 10.1109/ISCC.2006.38 (cit. on p. 20).

[52] B. Yang and H. Garcia-Molina. "Improving search in peer-to-peer networks". In: *Proceedings - International Conference on Distributed Computing Systems* (2002), pp. 5–14. DOI: 10.1109/ICDCS.2002.1022237 (cit. on pp. 16, 19).

[53] J. Zarrin, R. L. Aguiar, and J. P. Barraca. "Resource discovery for distributed computing systems: A comprehensive survey". In: *Journal of Parallel and Distributed Computing* 113 (2018-03), pp. 127–166. ISSN: 0743-7315. DOI: 10.1016/J.JPDC.2017.11.010 (cit. on pp. 2, 10, 13, 15).

[54] B. Y. Zhao et al. "Tapestry: A resilient global-scale overlay for service deployment". In: *IEEE Journal on selected areas in communications* 22.1 (2004), pp. 41–53 (cit. on p. 7).