NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

RAFAEL MARQUES SEQUEIRA

Bachelor in Computer Science

GENERAL PURPOSE SEARCH IN LARGE-SCALE UNSTRUCTRED OVERLAY NETWORKS

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
February, 2022

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

GENERAL PURPOSE SEARCH IN LARGE-SCALE UNSTRUCTRED OVERLAY NETWORKS

RAFAEL MARQUES SEQUEIRA

Bachelor in Computer Science

Adviser: João Leitão
Assistant Professor, NOVA University Lisbon

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
February, 2022

Abstract

The centralized solutions provided by cloud computing require one to fully place trust in cloud providers that also act as a single point of failure for applications and users. Decentralized alternatives, where trust and responsibility are shared among those who participate in them, are presented as a solution to the limitations imposed by cloud infrastructures. However, decentralized alternatives require solutions and efficient services that enable users and application to fully take advantage of a decentralized network. Searching enables one to know the location of one or more desired resources in a network, and is often needed as a primary service or as a building block in decentralized systems. Unfortunately, how search is used in such systems varies from system to system, and search performance is heavily influenced by several factors, such as the topological organization of the network and resource popularity.

In this work, we propose a search mechanism that can be leveraged by a wide range of systems and applications, while maintaining its performance. To accomplish this, our search mechanism will enable arbitrary queries that are flexible enough to express the different resources existing in a network; by employing dynamically selected forwarding strategies that can be defined with fine-grained detail and take into consideration past activity information.

Keywords: Distributed, Search, Unstructured Overlay, Arbitrary Queries

Resumo

As soluções centralizadas fornecidas pelo *cloud computing* exigem que se deposite total confiança nos *cloud providers*, que também funcionam como um ponto único de falha para aplicações e utilizadores. Alternativas descentralizadas, onde a confiança e a responsabilidade são partilhadas entre os participantes, são apresentadas como uma solução para as limitações impostas pelas infra-estrutura da *cloud*. No entanto, estas alternativas necessitam de soluções e serviços eficientes que permitem aos utilizadores e às aplicações tirar partido da rede descentralizada. A pesquisa permite saber a localização de um ou mais recursos na rede, e é muitas vezes necessária como serviço primário ou como um bloco de construção em sistemas descentralizados. Infelizmente, como a pesquisa é utilizada varia de sistema para sistema, e o seu desempenho é fortemente influenciado por vários factores, tais como a organização topológica da rede e a popularidade dos recursos.

Neste trabalho, propomos um mecanismo de pesquisa que pode ser alavancado por uma vasta gama de sistemas e aplicações mantendo o bom desempenho. Para o conseguir, o nosso mecanismo de pesquisa irá permitir a utilização *queries* arbitrárias que são suficientemente flexíveis para descrever diferentes recursos; utilizaremos estratégias de propagação (que podem ser definidas ao detalhe) dinâmicas, com a sua selecção sendo efectuada com recurso a informação de actividades passadas

Palavras-chave: Pesquisa, Rede não estruturada, Queries arbitrários

Contents

List of Figures

1 Introduction

1.1 Motivation

We have been relying on cloud services to fulfill the ever-growing need for resources demanded by today's world applications. The key idea behind the cloud computing model is that the resources are clustered in data centers managed by a central entity that rents a fraction of them to clients in an on-demand fashion over the Internet[3]. This paradigm has, arguably, many advantages. For instance, one can benefit from a reliable environment and the available resources provide applications with some elasticity since they can expand the resources available to them "on the fly". Furthermore, the cloud also lifts the cost of acquiring and managing the infrastructure from its clients, providing them with both software and hardware that is owned and fully managed by cloud providers.

On the other hand, the cloud, even when considering that its services can be provided by multiple geo-replicated data centers, is inherently centralized. Because of this, we argue that it can come with a hidden cost: *trust*. Trust is fully placed on the providers, which can act as a single point of failure[3]. Additionally, with the rise of IoT and edge devices in general and their change from consumer to consumer/producer, cloud computing faces more challenges because of the huge amount of data generated that has to be shipped to the cloud through the network, and privacy concerns. Furthermore, with new use cases that produce a lot of data and which are time-sensitive, such as autonomous vehicles, the transmission of the data to the cloud may become a bottleneck. A paradigm known as *Edge Computing* has emerged motivated by these and other concerns[34].

To address the emerging limitations and drawbacks of the cloud computing paradigm, one can rely on alternatives where trust and responsibility are shared among those who participate and form a system, hence avoiding a single point of failure. Edge computing presents a unique opportunity to (re)explore decentralized collaborative processes since they address some of the problems that motivated its emergence and are present as prime examples of its usage. Decentralized systems like IPFS[6] have regained traction recently which can be explained by increasing users' privacy concerns and the popularization of cryptocurrencies that are expanding their functionalities beyond simple payment

systems.

However, one could not expect the success of these solutions if its services do not present a comparable quality to its centralized counterparts. Searching, which is the subject of this work, is a fundamental service for these systems since their goals often depend on it. For instance, a system may need to locate a set of capable nodes to delegate computations or a node that has the information needed to perform a given task.

1.2 Context

Search is a service that enables someone to know the location of one or more desired resources. These resources may vary, for instance one may be searching for a set of nodes that contain specific hardware specifications or a file containing a keyword in its description. Furthermore, beyond retrieving the location of something, a search also enables users to explore what is available and can serve as a fundamental build block for building other services.

How someone describes what aims to find plays an important role in the results returned by the search. Supporting highly generic descriptions increases the search and system's flexibility as a whole.

In order to find something, one must match this description with the available resources and return their locations. However, to maintain scalability, nodes in decentralized systems are unable to keep global records of all resources stored/owned by the system so the search processing usually involves finding a node that has or knows the location of resources that match such description.

A distributed global index, which can be materialized by a *Distributed Hash Table* (DHT), is a well-known solution to the problem of exact matching search, where users search for resources through their predefined unique identifier that acts as a key. The network is shaped according to a predetermined structure, with nodes and resources following strict rules that dictate their placement in the network. In the absence of global membership, which is roughly described as each node knowing of all other nodes, the node's placement refers to whom they need to be aware and usually interact with. In these structured networks, when one aims to find something, it issues a request that transverses the network contacting nodes that are progressively closer to the node that can fulfill the request. However, due to its rigid structure, they are very susceptible to churn (concurrent joining and departing of nodes). When nodes join or leave, it may be necessary to rearrange the network to maintain its structure. Additionally, if resources are allocated to nodes according to their identifier, these events may also trigger the reallocation of resources to new nodes. However, and perhaps more importantly, is the fact that search is limited to exact matching. This clearly limits the search flexibility and can make these solutions less desirable.

Systems that use local indices, where each node usually only indexes its resources, can easily support richer descriptions and hence achieve a more general search since

they don't impact how messages are disseminated. In this approach, nodes don't need to connect to a strict set of other nodes and form an unstructured overlay. Furthermore, by enabling nodes to form connections in a loose manner, these solutions are usually more resilient to churn and node failures. However, there's no way to accurately predict where resources are located, and searching can be roughly described as asking every reachable node if they have something that matches the description. Whom nodes one chooses to contact is a root question of searching in this context since it can impact the search quality and performance but also the system since a lot of messages are generated which might overload nodes or at least will hinder scalability.

1.3 Expected Contributions

In this thesis, we expect to overcome some of the challenges presented. A more detailed plan will be presented in Chapter 3 but our main objective is to implement a search mechanism that achieves good performance while maintaining the capacity of being leveraged by a wide range of applications. From this, and considering our solution, the expected contributions are the following:

- A search mechanism that can adopt strategies assigned *ad hoc* to adapt to different environments and needs;
- A mechanism that leverages past searches, received queries, session information to choose the best search strategy;
- Improve the search mechanism employed by a state-of-art distributed system, exploring its caveats;
- An experimental evaluation to validate our solution in a range of environments, comparing it against other solutions.

1.4 Document structure

The remaining of this work is structured as follows:

Chapter 2 presents the previous work related to our objectives. Search is a intricate subject that is dependent on several aspects. Considering this, we will adopt a "bottom-up"approach starting from the network and finishing in full systems that were heavily impacted by this service. While doing so, we will discuss and categorize how one can describe resources and search mechanisms but also how evaluate its performance.

Chapter 3 describes our proposal and how we will evaluate and validate it. Additionally, also presents how we plan to achieve this in the following months.

2

Related Work

In this chapter, we survey the state-of-the-art related to our objectives. We will consider a system, illustrated in the Figure 3.1, as the composition of an (overlay) network that is leveraged by its services but also its purpose that is closely related to its application usage. The latter undeniably influence the first two layers, however, the reverse also applies. Services enable systems to achieve some of their goals by providing adequate abstractions, and, by 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 the two, searching is a general and intricate topic by itself. For instance, to find something, one needs to define it, and the resources that can be looked after also need to be indexed somehow.

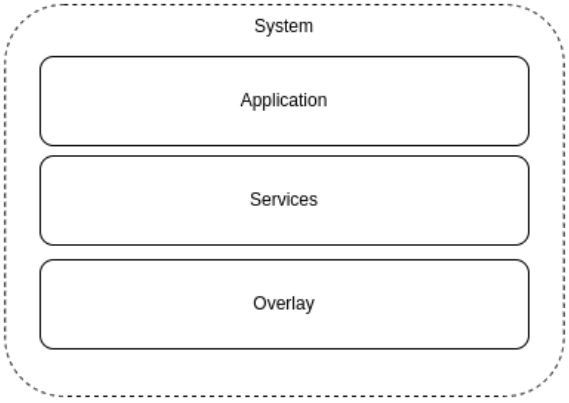

Figure 2.1: Generic system architecture

In the rest of this chapter, we will explore the diversity inherent to searching but also present a glance at the underlying mechanisms and how they interact with each other, culminating in some examples of real systems that leverage this service to achieve its objectives. It will be 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 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. After that, in Section 2.3, we study how searches are performed, how one can compare their performance defining a base to further explore them, and, lastly, some classical approaches are presented evaluating trade-offs between them. Finally, in Section 2.4 we will present real systems that, hopefully, will enable one to see the full picture.

2.1 Overlay

An overlay is a network built on top of another, usually abstracting the participants of the details of the underlying infrastructure. Moreover, nodes (or peers) in peer-to-peer systems form these networks to communicate with each other, P2P services are built directly on top of them, and applications use the services to implement their features.

Overlays are formed through a decentralized protocol known as membership protocol which establishes and maintains connections (links) between nodes. However, in large-scale systems peers usually don't know all other peers in the system (known as full membership) but only a subset of nodes, which is typically referred to as partial membership[8]. The nodes in the subset of a given node are typically referred to as neighbors. The motivation behind adopting the second strategy comes from the fact that, first, a full membership would require a large number of 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, 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 information updated. Overall, it's important to retain that overlays are usually very dynamic and even the partial membership of each node changes constantly. The effects of this impact 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 network as a graph. Furthermore, graph properties can be used as metrics to further describe them and study their performance. We will consider the ones that follows[21]:

Connectivity Similarly to a connected graph, each node in an overlay should have at least one path from each node to all other nodes. If this condition fails to be met, nodes or portions of the overlay 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,

forming effectively a directed graph. If so, the number of directed edges is distinguish in **in-degree** and **out-degree**. It provides a measure of how much a node participates in the message dissemination. Nodes with a high in-degree are very reachable while nodes with a high out-degree can easily contact a lot of other nodes.

Average Shortest Path A path is the group of edges that a message must transverse 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 by having fewer intermediaries the message dissemination is 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 a lot of redundant messages. Furthermore, it also is related to fault tolerance properties of the graph since if nodes 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 usually categorized as structured or unstructured. We will also include a third category, biased overlays.

2.1.1 Unstructured

In unstructured overlays, connections among peers are established at random or without imposing strict rules. As a direct consequence, the nodes' arrival and departure processes are simple and membership changes produce low overhead[28]. Therefore, these networks are known to be able to tolerate significant amounts of churn and have low maintenance costs[5].

This type of overlay is also known to form topologies with properties that provide them with high resilience against node departures and failures, usually remaining functional (connected) even when a big portion of nodes is removed from it at random or selectively. For these reasons, are particularly desired for highly dynamic environments like the edge and were adopted by early peer-to-peer file-sharing applications like Gnutella (presented in Section 2.4).

However, the random nature of the topology imposes additional difficulties on services that cannot efficiently leverage it. For instance, how a node can be reached efficiently is uncertain since one cannot predict to whom a given node is connected (where is located). Furthermore, when a node wants to communicate with someone that's not directly

connected to them, messages are propagated in the network by gossiping, which will be discussed in more detail in Section 2.2.

2.1.2 Structured

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

The topology employed by the overlay varies, with some systems adopting ring-like shapes[35], trees[27], or a multidimensional space[30].

The main principle leveraged by them 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. Every node which knows the identifier also knows where is located and can reach the node responsible for storing it. That's the key idea behind a *distributed hash table* (DHT) implemented by CAN[30], Chord[35] and Kademlia[27]. Since peers know how to locate a given resource in the whole overlay *a priori*, DHTs implement effectively a distributed global index that can be used to efficiently locate resources[39].

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 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

The third category, which we named biased overlay, tries to combine the two already discussed. The key idea is that peers (like in unstructured overlays) connect at random firstly but after cementing themselves in the network, try to improve their position by using some heuristic 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.

However, is important to note that the achieved topology is less rigid than the one accomplished in structured overlays and it's constantly evolving. Furthermore, nodes joining the network don't know their final location in the network nor that location is known at any point by other peers. Because of this, message dissemination is performed similarly to unstructured overlays but with the improvements conceived by the network

itself. Additionally, nodes need some time to improve their position and if the network has huge amounts of churn the topology may never be stable.

The heuristics vary according to the needs of the services or applications implemented on top of it. For instance, one could bias the node connections through links with low latency. Such an example is presented in X-BOT[24], a protocol that produces biased overlays through generic efficiency criteria derived from a local oracle. Another example is Gia[7], 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

In contrast with flat overlays, where peers operate at an equal level, multilevel overlays adopt more than one level to improve their functionality building effectively a multidimensional overlay. Each level is composed of a set of peers who can participate in multiple levels simultaneously. Levels can be hierarchical (where peers on one level have more responsibility than others) and adopt different overlay structures, having different neighbors at each level and employing distinct strategies to select them.

However, it's important to note the difference between a multilevel overlay and multiple stacked overlays. The latter approach is also common and was proposed by some works like SONs[10]. The key difference between the two is that in stacked overlays one level doesn't impact the others, different services (even with similar purposes) are implemented in each overlay. Applications then leverage these independent services to implement their goals. In SONs, peers join overlays that are semantic close to them. Searches use similar mechanisms in each one but are, ultimately, not related since can be performed in parallel on multiple semantic overlays.

The motivation for adopting a multilevel architecture often comes from scalability issues of flat designs, to improve message dissemination or extended flexibility. However, multilevel overlays are more complex and systems that use this architectures often have to deal with synergy challenges between levels.

CAN[30] implements a DHT over a multidimensional coordinate space. Nodes position themselves at a random zone and establish connections with nearby nodes that are used for routing. Zones are dynamic, being reshaped when nodes join or leave the system if needed. Similar to other DHTs, resources are hosted at predefined zones. However, it introduces a concept particularly relevant for our discussion named realities. Nodes in CAN may join multiple realities. Each reality implements an independent coordinate space and, since nodes position themselves at random, have one position and an independent set of neighbors per reality. One could argue that each independent coordinate space forms a level. Furthermore, these multiple overlays are leveraged to route more efficiently. Contrary to nodes, resources share the same location between realities. When forwarding messages, a node chooses the reality that has the closest neighbor to the final

location reducing the path length.

Nevertheless, in multilevel overlays levels are often more diverse. An 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] presents a materialization of this idea. Peers participate in a biased overlay that effectively makes them establish connections with similar peers according to a set of 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 exchange) protocol[32]. The search is initially performed in the unstructured overlay, with peers that have part of the resources being placed in a search session. The search session is motivated by the fact that resources in IPFS can be subdivided into several interconnected but independently addressable chunks and a search involves retrieving multiple chunks. If the session becomes empty, meaning that all peers were purged from it, the structured overlay is used to populate it. Briefly, the node performs a lookup in the DHT to find providers for chunks. After that, the search continues normally in the session.

Different from the examples presented until now, *hierarchical levels* can be used to take advantage of peer heterogeneity. Furthermore, if only a subset of peers of 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 which is not well studied (a general description is provided in several works[29, 12, 28]). In its simplest form, 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 indices[39]). 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 anyway[23]. Some works address this problem directly. For instance, SOSPNet[14] employs active load balancing

mechanisms but also relaxes the aggregated index, with super-peers only indexing partially (and selectively, according to how much past queries it resolved) the peers' content. Additionally, tries to improve search performance through biased connections between peers and super-peers to capitalize on interest locality.

In the rest of this work, we will focus on unstructured overlays. Ultimately, we may also consider multilevel unstructured networks or biased networks because the search processes are analogous to the ones employed in flat unstructured overlays, being interchangeable between the two.

2.2 Gossip

In gossip-based communication patterns[21], nodes perform the dissemination of the messages in a collaborative process since nodes are only directly connected to a limited amount of other nodes (neighbors) and, consequently, only have 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. Since nodes relations can form cycles, mechanisms that guarantee that the same message is not processed twice are usually 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.[8]

The message dissemination can be limited, defining a horizon in a time-to-live (TTL) fashion. Each time a message it's forward a counter of hops it's incremented, when the counter reaches the limit defined by the TTL parameter nodes stop forwarding it.

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 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 let their users search for files to accomplish their primary objective. Nevertheless, searching is a general problem. For instance, service discovery[29] and

P2P-Grid systems[46] are also heavily dependent on it. Considering this, we will use the generic term *resources* whenever possible to denote the targets of a given search.

Search is the process through which a node can know the location of a resource. How a resource is retrieved is orthogonal and falls out of the scope of this work, however, in cases such as IPFS and Bitswap (detailed in Section 2.4) the gap between search and retrieval can be ambiguous. We will address this specific case in more detail later. Addionally, we will consider that a search finishes when a satisfactory amount of locations is obtained or when the search fails. However, the latter can be hard to define because it's highly dependent on the search mechanism and will also be explored later.

Resources have to be indexed to be discovered and, in unstructured (and biased) overlays, are two base approaches to do so. A first (and arguably naive) possible strategy is storing at each node the location of all resources in the network. If so, to search, nodes would simple consult their (local) global indices and obtain the desired answer. The global index could be shared on nodes' arrival and updated slowly by gossiping to not overwhelm the network. However, similarly to the global membership problems presented in Section 2.1, the cost of maintaining such indices would be too large (which would be aggravated by churn) and would impose a limit on the system scalability. As a consequence of this, large-scale systems usually don't adopt this approach but instead prefer local indices.

A second strategy to tackle this problem, like partial memberships, does not use global knowledge about resources location. Nodes keep strict local indices[39] with a limited amount of resources, varying between only the resources of the node itself and resources of a limited horizon of other nodes (for instance, neighbors). Ideally, this value should be kept low to avoid some of the problems mentioned above.

Using local strict indices, searches start at a node of the system (source node) and are performed following a gossip communication pattern (see Section 2.2). Moreover, when the source node wishes to find some resource describes it in a query and sends a message containing it to the gossip targets. Usually a push strategy is employed. Upon the arrival of a search message, a node tries to fulfill it by matching the description provided in the query with their indices.

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 details. However, it 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.

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.

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[18, 4].

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 with 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. Besides, 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[13, 26].

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.

A search finishes when satisfactory results are found, which is usually defined through a predetermined amount of resources. When more than this goal is obtained, the search algorithm incurs in *overshooting* hurting its efficiency[17]. However, the search can be terminated even if this goal is not met. A first, and unusual scenario, is when a node visits 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 visited since the topology and the number of nodes in the system is unknown[46]. Another instance is when a node is not able to forward a query further. One such case can happen if some mechanism assures that nodes are not visited 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 backforward following the reverse sequence of nodes that visited (path) to reach its final destination 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 (source and destination) it's opened *ad hoc*.

2.3.2 Search Performance

The overall performance of a search is influenced by several factors and search algorithms are very diverse, adopting fundamentally different approaches, hence, comparing them objectively is challenging.

Underlying aspects such as the overlay characteristics (see Section 2.1) directly impact the message dissemination[22], ultimately influencing search performance. For instance, if the network has a high clustering coefficient (nodes share a lot of neighbors), messages with a limited scope may have difficulty reaching new nodes which directly impacts search performance. Furthermore, in networks with low diameter, messages reach a lot

of nodes or, more importantly, the right nodes in a low amount of hops. Unstructured overlays may also exhibit a power-law degree distribution, which is characterized by the presence of a small number of nodes with a very high degree and a lot of nodes with a small one. Some early studies found that Gnutella displayed power-law characteristics[25, 31, 1], but it was contested by later works[36, 37, 17] that ague that it has small-world properties (a high cluster coefficient and low average shortest path) instead. Anyhow, power-law distributions are known to affect, among other things like system resilience, search performance[25, 1]. For instance, messages gravitate around well-connected nodes which increases their load, creates more overhead due to duplication, and difficult queries exploring new nodes.

Unrelated to overlay properties but also relevant, is how the resources are distributed across the nodes. Innately, it is easier to find popular resources. Furthermore, if the distribution is not uniform and some nodes have a huge amount of resources some informed search algorithms (see Section 2.3.4) may exploit this characteristic to improve their performance. However, this distribution and the amount of replicas of each resource varies in a system base. When evaluating search algorithms several distributions can be adopted.

Another relevant aspect and closely related to file-sharing applications is replication. Naturally, when some node retrieves a file start to index it. Nevertheless, there have been several proposals to improve search performance in this type of system through proactive replication[38, 25]. Further, files content doesn't have to be necessarily replicated, making a set of selected nodes only store the index instead. Some search algorithms adopt a well-known approach that consists in indexing the resources of every node in a 1-hop distance[7] and super-peers index the content of normal peers. However, active replication falls out of the scope of this work and this subject will be studied in the overlay context or in a search algorithm basis.

Even with all the aspects mentioned above, and considering that some algorithms may be better fitted systems or overlays with determined conditions, the following metrics[22, 23] can be used to compare different 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 by the total 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. This effect is aggravated by concurrent searches.

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 major groups depending on how much information is known about the location of a resource when forwarding a message[20, 46]. We will adopt this approach to further classify the techniques presented below.

2.3.3 Uninformed Search

In uninformed (or blind) search mechanisms, no information about the resources or the overlay is considered when forwarding a message. These mechanisms are usually praised for their simplicity regarding both their strategy and their implementation. Since no additional information is used, nodes are not required to store additional data or exchange messages to obtain it. Considering this, blind algorithms don't cause extra overhead in the network. Furthermore, for similar reasons, they are expected to be unaffected by network changes maintaining their performance independently of how dynamic is the overlay.

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

2.3.3.1 Flooding

Flooding sometimes refereed to as Breath First Search (BFS), is a classic search technique where a peer, upon the impossibility of fulfil the query himself, forwards it with a fanout equal to its number of neighbors. Flooding the network has several advantages over other search algorithms but its main characteristics come from the fact that a lot of nodes are

queried after a few hops and this value increases exponentially. Because of this, low latency is expected and resources are found with a high probability, having a high recall. In fact, flooding is inheriting deterministic since if the query reach the whole network (with all nodes receiving the query) resources will be found and the recall will be 100%. However, it has a high dissemination cost and lot of overhead is produced. A huge amount of messages are generated, concurrent searches can saturate the network and overload nodes that have to process it[25]. Most importantly, after reduced number of hops, depending on the overlay properties, these messages are mostly redundant because of natural cycles that are formed between nodes, which harms the scalability of the system[5].

To partially mitigate this problem, flooding-based techniques usually impose a horizon on the search through a, already mentioned, TTL. Nevertheless, limiting the search scope implies that it is no longer complete since only a portion of the nodes are queried and choosing the ideal TTL is challenging. Too high values may cause unnecessary burden in the network while too low values harm the recall percentage. Further, the optimal TTL also changes with the network topology and resource replication ratios[25].

Several variations of flooding were proposed. To further reduce the number of messages generated, **Normalized Flooding** and **Probabilistic Flooding** (also known as Modified-BFS or Teeming)[19, 5] limit the fanout of the gossip messages. In the first, this value is upper limited to a predefined amount and in the latter a percentage of random neighbors is used to forward the queries.

Other approaches try to dynamically control the TTL to overcome the flooding issues without lowering recall rate. **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**[45] 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 when a new round is performed 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[17] (DQ) technique, adopted by Gnutella[37, 17], goes a step further in selecting the right TTL by adjusting its value according to the popularity of

the desired resources. 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 desired 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[17] an evaluation of this method was performed using a snapshot of the two-layer Gnutella network (a detailed description of the network is provided here[36]) 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.

2.3.3.2 Random Walk

Random Walk (RW) algorithms try to maintain the number of messages generated while searching to a minimum. To accomplish this, all nodes involved in the search use a fanout of one when forwarding the message containing the query, which is usually referred to as a walker. Since the gossip target in each node is chosen at random, the walker transverses 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 huge latency. This is aggravated if the resources are not well replicated in the network with walkers having to visit a lot 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[7].

To overcome these problems, several parallel walkers can be deployed by the origin node. This search technique is known as **K-Walkers**. In other words, 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.

Like in flooding-based searches, RW techniques usually utilize a TTL value to limit the number of times that a walker can be forward. However, the TTL can be combined with checking that can also serve as a keep-alive mechanism to improve its reliability.

Authors of [25] stated that, in standard RW, message overhead is reduced by an order of magnitude when compared to ERS but the delay (latency) is also increased an order of magnitude. In the measurements performed in the same work, was found that 32 walkers compared to flooding decreased the overhead by 2 orders of magnitude but the number of hops needed to find an object also slightly increased.

2.3.3.3 Hybrid

Combining the two main groups already seen is can be used to create algorithms that try to obtain the best of both worlds.

In **local flooding with k independent random walks**[11], an initial (local) flood with a small and predefined TTL is performed. After that, in absence of good search results, each node that received the query in the first phase initiates a random walk.

The authors of [11] 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, like latency that is expected to be high in RW-based algorithms, are presented.

2.3.4 Informed Search

Informed search mechanisms try to improve the search performance by using information to guide the search query toward nodes that are believed to better fulfill it. If each node that forwards a query chooses wisely to whom the message is sent, the query hit rate is expected to increase. However, informed search algorithms are usually more complex and require that, at least, nodes store some information to select neighbors. Furthermore, this information has to be processed and sometimes disseminated in an proactive fashion which increments the number of messages in the network.

The information used to infer which neighbors are more fitted to answer the query varies from simple metrics like the degree of the neighbors to similarity between nodes and a given query. Because of this, we will to further categorize the techniques accordingly to how this information is gathered.

2.3.4.1 Reactive Gathering

In reactive gathering, nodes mainly collect information from previous search results. Obtaining information in a reactive fashion it's a powerful concept since it usually doesn't introduce additional overhead in the network. 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**[45]. 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. One example is using neighbors that returned the greatest number of results in the past 10 queries.

Intelligent Search[19], which is sometimes called Intelligent-BFS, employs a very similar strategy but uses previously answer queries to select which neighbors should receive the search message. Briefly, each node keeps the most recent query replies for each of its neighbors, 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 problems caused by possible cycles (where a set of peers forwards messages only between them).

However, some problems associated with intelligent-BFS were presented in [41]. 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 lacks adaptation to resources deletion or node departures since it doesn't employ a negative feedback strategy to ensure the quality of the information. 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 and rather simple technique is called **Adaptive Probabilistic Search** (APS)[40], it 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.

One of the drawbacks 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[44] 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. 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.

2.3.4.2 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 forward a query message, has (at some extent) 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 Indices** (RI)[9]. 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 must inform the new neighbor about how many documents (and their topics) can be reached through them. 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 it's applied 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 it's 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 thing 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[7], a P2P file-sharing system built on top of Gnutella that aims to improve scalability by exploiting node heterogeneity. Different from other approaches, Gia tackles the search problem on several fronts. Several changes are proposed. Firstly, all nodes index the content of their neighbors in a strategy that is known as one-hop replication. 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 2.3.3.2. 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 Systems

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 of 2005[37].

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[15], peers form a flat unstructured overlay establishing connections with a set of neighbors. Messages are propagated in the overlay by flooding with a limited scope (with recommended values between 1 and 7 hops) and contain a randomly generated identifier, a type, and a TTL. 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[16] 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 small number of 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 fulfill 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 fulfill it. The exact search mechanism used is not specified, however, some works state that it employs Dynamic Querying[37, 17], which was already presented in Section 2.3.

Several works have explored the Gnutella network in an attempt to characterize it. Both [33] and [31] studied an early version of the network (before the adoption of the super-peer architecture). In [31] 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 [33] 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 [36, 37] 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 [37] 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 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

InterPlanetary File System (IPFS)[6] is a distributed peer-to-peer file system that enables its users, that act as nodes in two overlays, to share, search and retrieve objects.

IPFS is an interesting case study because recently has gained wide interest and is actively developed. It can be leveraged to a wide range of usages[42] and several projects have adopted it as part of their stack.

The system works as follows. Objects are divided into chunks that vary in size between 256KiB and 1MiB. Each chunk of an object is called a *block* and it's uniquely identified by a *content identifier* (CID) that is obtained from the hash of its content. Blocks may be associated (for instance, in the case of a big file that was divided into several chunks or a directory with multiple files) using a Merkle Directed Acyclic Graph (Merkle DAG). Merkle DAGs are similar to Merkle Trees but without the need to be balanced and with each node (of the graph) having the possibility to store data. DAGs are, ultimately, composed of a set of blocks CIDs (or other DAGs) and a CID.

Some features of IPFS are directly derived from this approach to objects. For instance, the content linked in a DAG is immutable, since if it was changed their CID would change which would lead to a change in the DAG CID and the same content shares a CID which can be used for deduplication purposes since a block a DAG (or a block) can have multiple parents.

Objects are retrieved through a protocol named *Bitswap*[32] which uses the unstructured overlay to flood the neighbors with an exact match query (named WANT-HAVE) that includes the root CID of the DAG. Since the DAG can link a set of children blocks a session is created with the nodes that return positive results (through a HAVE message). The key idea is that nodes that have the DAG are likely to also have their children, if they exist, and can be used in the next searches to limit the number of messages generated by the flooding. However, the node of DAG that contains the CIDs has to be retrieved first. The source node replies to HAVE messages with a WANT-BLOCK message and the content is transferred. When a block is received, the source node send a CANCEL message to any peer that has received an WANT-HAVE containing its identifier. The iterative process continues, with the source including the multiple children CIDs in the new query and

flooding it to session members, until there are no more blocks to retrieve.

Peers that respond with DONT-HAVE message to a predefined value of query messages are removed from the session. New peers are added to the session by sending periodically a WANT-HAVE to a random neighbor. However, if the session becomes empty, meaning that none of the neighbors have the blocks, Bitswap performs a lookup in the structured overlay to find new peers to the session. The structured network is, effectively, a Kademlia[27] DHT and, as seen previously, nodes store (or index) blocks that are attributed to them according to their identifier. In comparison with Bitswap, the DHT may offer higher latency since each search requires taking approximately $log\ n$ (with n being the number of nodes in the system) steps, while in flooding resources may be found almost immediately if the neighbors have it. The authors of [32] performed some measurements that indicate that Bitswap outperforms the DHT, reducing the latency by 30% on average, however, a mesh topology with only 30 nodes was used so source nodes are directly connected to nodes that host the objects.

To avoid free-riders, IPFS also employs a credit strategy where nodes are incentivized to exchange blocks in order to improve their reputation. If a node doesn't have any blocks to share, may accept blocks from others to participate more actively in the system, which also serves as a replication strategy.

In the baseline protocol, neighbors don't forward the query to their neighbors so the search is very limited. To improve the recall rate, the same authors added normalized flooding (where the fanout is limited to a predefined degree) with a horizon defined through a TTL value. Additionally, a reactive gathering strategy was also proposed and tested. Nodes store the queries received from other peers in a registry, hoping that they were able to retrieve the blocks described. When a peer wants to search for a CID consults their registry, sending a WANT-BLOCK message to a predefined number of peers that requested it in the past if possible instead of flooding the network. The baseline mechanism is used as a fallback.

2.5 Discussion

Although some search techniques are more efficient than others, searching usually involves some trade-offs. For instance, searches in DHTs are usually limited to exact match queries, where one is only allowed to search for an object using its unique identifier. Unstructured overlays naturally support keyword-based, range, and arbitrary queries which are more flexible but the location of a given resource is unknown *a priori*.

Even when considering only searches in unstructured networks, nodes disseminate search messages in the network according to a predetermined strategy and each strategy has its advantages and disadvantages. For instance, flooding-based techniques achieve a good recall rate but have high dissemination cost, which harms the system's scalability. In contrast, RW-based solutions produce a small overhead but increase search latency. Furthermore, random walks can have limited reliability in very dynamic environments

since messages can be lost and the number of walkers is never incremented in the search process.

Nevertheless, how effective a strategy often varies on a resource base, and its performance is impacted by overlay properties (for instance, clustering coefficient or degree distribution) that affect the query propagation. Ideally, search mechanisms must adapt to these factors. Some search techniques such as Dynamic Querying try to achieve better trade-offs by imposing a limit on the search horizon on a search basis, however, how queries are forward by nodes never changes. Informed search techniques go a step further, usually choosing a set of nodes to forward the queries according to previously obtained information, but the same also applies.

We believe the strategy used to propagate must be considered to achieve better performance. If nodes are allowed to define *ad hoc* strategies on a search (or neighbor) basis, the search mechanism can be tailored (making more fitted trade-offs) to the conditions that impact its performance.

2.6 Summary

In this chapter, we presented a complete vision of searching based on the state of art found.

Firstly, we introduced the concept of overlay networks used by nodes to disseminate messages and how they can be described. Furthermore, we categorized them according to how nodes organize themselves in the network and discussed multilevel architectures.

Following, we explored search in unstructured overlays which is the main subject of this work. We concluded that search is the composition of several topics like queries and message forwarding techniques. We explored them and presented some well-known search algorithms.

Finally, two relevant systems were presented. The first is a classic example with multiple articles exploring its properties. The second is becoming more relevant and will serve as a base for our future work.

In the next chapter, we will describe our solution and how to validate it. Both the solution and the evaluation draw inspiration from the work studied and presented in this chapter. Furthermore, we will use some of the search techniques studied as a comparison.

3

Planning

In this chapter, we will detail the main aspects of our solution and how to evaluate it. In Section 3.1 we clarify our proposed solution, subdividing it into several parts. In Section 3.2 a description of how we will evaluate the solution presented is provided. Furthermore, a set of algorithms and modifications needed for a fair comparison are also covered. Finally, in Section 3.3 we present a work plan that will be followed to accomplish our goals.

3.1 Proposed Solution

As mentioned in Chapter 1, our goal is to explore and improve general search techniques. In Chapter 2, we presented a state-of-art system that has gained some traction recently, the InterPlanetary File System. IPFS already has a search mechanism incorporated in a search and block exchange protocol called Bitswap, however, its queries have very limited flexibility only allowing its users to search for identifiers (using exact match queries). Furthermore, in IPFS, objects are divided into blocks and connected through a Merkle DAG which, ultimately, leads to the adoption of search sessions by Bitswap. To the best of our knowledge, this concept was not been previously explored and presents a good opportunity to develop new approaches. Considering this, the proposed solution will use IPFS and Bitswap as a base.

The main challenge of implementing an efficient search mechanism is that, as we have seen before, its performance is influenced by several factors. Furthermore, a general search by definition has to be applicable to different use cases. We will take this into consideration and expand upon it in our work but, as a consequence, our solution will be presented in an incremental way going through several main steps. In the rest of this section, we will further detail each of the main steps in our research agenda.

3.1.1 Arbitrary Queries

In Chapter 2 we explored different types of overlays, concluding that some types are more fitted to support complex queries. IPFS already has an unstructured overlay in its

multilevel architecture, however, Bitswap doesn't leverage it to exploit the advantages of local indices. Searching using more versatile descriptions aligns directly with the subject and goals of this work and, because of this, we plan to extend Bitswap queries to support arbitrary queries.

3.1.2 Generic Strategies

The currently deployed version of Bitswap (that we will refer to as baseline) uses a single hop lookup which is essentially best effort as the main strategy, messages are only forwarded to neighbors and not further propagated to other nodes. Proposals were made to expand its functionalities in this regard[32].

As has been stated before, the DHT is used as a fallback mechanism to populate the search session. However, while searching for blocks in the structured overlay, one has to go through a considerable amount of nodes to retrieve them. Reducing the dependence of the search mechanism in the DHT may improve the search latency. This can be achieved by improving the recall rate. We plan to enable the queries to be further propagated in Bitswap.

Nevertheless, there are several strategies to propagate queries, however, based on our research (summarized in Chapter 2) their results vary according to several factors like the popularity of the object or the overlay topology itself. To address this, and allow our search to be generic while maintaining efficiency, we plan to further extend queries to enable the source node to define a specific search strategy that will be executed by nodes receiving the query (potentially over some conditions).

In the limit, this can be done on a per node basis, allowing our solution to adapt to a wide range of conditions. To leverage the session, some strategies may describe a fined-grained behavior that can be composed of several searches. For instance, one can define a strategy that tells a node to search for the root DAG and also all blocks linked by it (triggered automatically when the root node is found to void extra RTTs).

3.1.3 Dynamic Strategy Selection

Several adaptive and dynamic search techniques have been adopted to improve search algorithms. In its simplest form, some algorithms adapt the TTL according to the (estimated) popularity of an object but the informed search mechanism goes a step further, choosing specific nodes to forward a query. To leverage the solution presented in Section 3.1.2, we want to expand on this concept. We plan to enable source nodes to use strategies dynamically. Since searches in one session are intrinsically connected, we will use session parameters to enable the source node to infer the right strategy according to the feedback of previous searches performed on that session. Furthermore, and as mentioned in [32], IPFS nodes have the possibility to collect (in a reactive fashion) information from previous searches and received queries from other nodes. We believe that the reputation system may be also used to infer more information about the neighbors.

3.1.4 Self-evaluated State

Some of the work studied used peer heterogeneity in terms of capacity (their attributes) and the state (how overloaded are) to improve search performance. We believe that this concept could be applied to our solution, however, in our approach, we plan to adopt it locally, with nodes self-evaluating their state and their capacity, instead of employing strategies like biased networks and active load balancing. A challenge associated with this approach is the free-riders problem. However, Bitswap already employs a reputation system and it may be used as an incentive to nodes perform the search the best they can. This objective could be dropped if achieving the previous ones are more complex than expected and lead to unexpected delays.

3.2 Experimental Evaluation

In this section, we explore how we plan to evaluate our solution. To infer its quality, we plan to compare it to a diverse set of well-known algorithms explored and described in Chapter 2. However, to have a fair comparison, some of these algorithms have to be slightly modified. We will compare our solution to the ones that follow:

Baseline Bitswap Baseline Bitswap is an obvious candidate for comparison since our work is directly built on top of it. However, we will not consider the proposes presented in [32] because they employ strategies that will be explored by other algorithms used for the same purpose.

Probabilistic Flooding Flooding is a classical and well-known search technique. In Chapter 2 the problems associated flooding without restrictions may cause were presented. Based on this, we plan to use Probabilistic Flooding. However, the search session introduced in baseline Bitswap directly collides with this approach, since (after the first round) the search is limited to nodes in the session. We plan to maintain it, but enable nodes in the session to forward queries a configurable number of times (TTL) to a percentage of the neighbors in the session. Since the source node is not directly connected to them, the responses will be returned following the same path and the nodes that provide the answer to the source node will not be purged from the session.

K-Walkers Another fair candidate to serve as a comparison is a k-walker strategy because of its popularity and a well-known alternative to flooding-based solutions. However, selecting random k nodes from all the source neighbors to each block of an object is unfair. We plan to adapt it to only consider nodes that were added to the session while randomly assigning the walkers. Similar to flooding, responses will be returned following the same path and the horizon will be limited by a TTL parameter associated with each query.

APS To compare our algorithm with informed approaches we will use Adaptive Probabilistic Search. Since it adopts (biased) k-walkers as a strategy to forward queries, we will use the changes already presented. However, to ensure a fair comparison, we will consider the root DAG (instead of blocks) as entries in the table of each neighbor.

When it comes to performing experiments, one must remember that the overlay characteristics (for instance, degree distribution and clustering coefficient) heavily impact the message dissemination which, ultimately, impacts the search. This is usually considered by performing evaluations under different topologies [41, 25].

The effect of overlay dynamics is also important to be considered in our experimental evaluation. For instance, (reactive) informed algorithms may benefit from a stable membership since they can learn from past activity.

Finally, the replication rate (how many instances of objects exist) and distribution of this factor across objects that will be used as targets must also be considered.

Performing an evaluation considering all these factors is challenging. We plan to vary some of these parameters in a simulated environment. Briefly, a graph is generated according to several parameters that control its characteristics. After that, the behavior of each node is simulated according to the algorithm.

In this environment, we are mainly focused on perceiving the quality of each search approach in fundamentally different overlays and making a comparison between them. To do so, we will use the *search metrics* already presented in Section 2.3 but focus mostly in the **Query Recall Rate**, **The Query Dissemination Cost**, and the **Query Hit Rate**.

Additionally, we also plan to compare the algorithms in an emulated environment. The idea is that a set of nodes running real code (encapsulated in Docker containers) form an overlay like IPFS but on a smaller scale. Network conditions (latency, bandwidth) are simulated using tools such as Linux TC[43] to capture a realistic setting in the laboratory. Searches are then performed in this system and their performance is measured. We also plan to vary the replication and query parameters but the overlay properties will not be altered between experiments. Artificial latency and bandwidth will be introduced in some channels to consider peer heterogeneity. Beyond the metrics already mentioned, we also plan to measure the impact of the algorithms in the systems or, more specifically, in the nodes that are part of the system. To do so, we will use *node metrics* that are, in practice, the usage of the system resources (CPU, memory, bandwidth).

3.3 Work Plan

In this section we detail the identified tasks and their respective scheduling.

1. **Arbitrary Queries** (04/03/22 - 01/04/22)

a) Design and develop a modified version of IPFS object indexing (14/03/22 - 25/03/22)

b) Design and implement modifications to Bitswap queries to support arbitrary descriptions (14/03/22 - 25/03/22)

c) Validate our solution and its implementation experimentally (23/03/22 - 01/04/22)

2. **Generic Strategies** (14/03/22 - 25/04/22)

a) Design a simple query interface to accurately describe a wide set of strategies (14/03/22 - 15/04/22)

b) Expand the protocol with a module that honors the strategies defined in the queries (14/03/22 - 15/04/22)

c) Validate our solution and its implementation experimentally (13/04/22 - 25/04/22)

3. **Dynamic Strategy Selection** (26/04/22 - 07/06/22)

a) Design and develop a module to collect and store information from previous sessions and searches in a reactive fashion (26/04/22 - 09/05/22)

b) Implement *ad hoc* strategy selection based on the session and past activity information collected (10/05/22 - 30/05/22)

c) Validate our solution and its implementation experimentally (27/05/22 - 07/06/22)

4. **Self-evaluated State** (08/06/22 - 28/06/22)

a) Develop strategy selection based on the state of the local node (for instance, how overloaded is) (08/06/22 - 20/06/22)

b) Validate our solution and its implementation experimentally (17/06/22 - 28/06/22)

5. **Evaluation** (29/06/22 - 11/08/22)

a) Develop strategies that implement the search techniques proposed for comparison (29/06/22 - 21/07/22)

b) Evaluate solution and store metrics (29/06/22 - 21/07/22)

c) Test the proposed search techniques for comparison and store metrics (22/07/22 - 11/08/22)

6. **Document** (01/08/22 - 30/09/22)

a) Review the related work (01/08/22 - 08/08/22)

b) Write the document (09/08/22 - 30/09/22)

Figure 3.1 illustrates the scheduling of the tasks in the form of a Gantt chart.

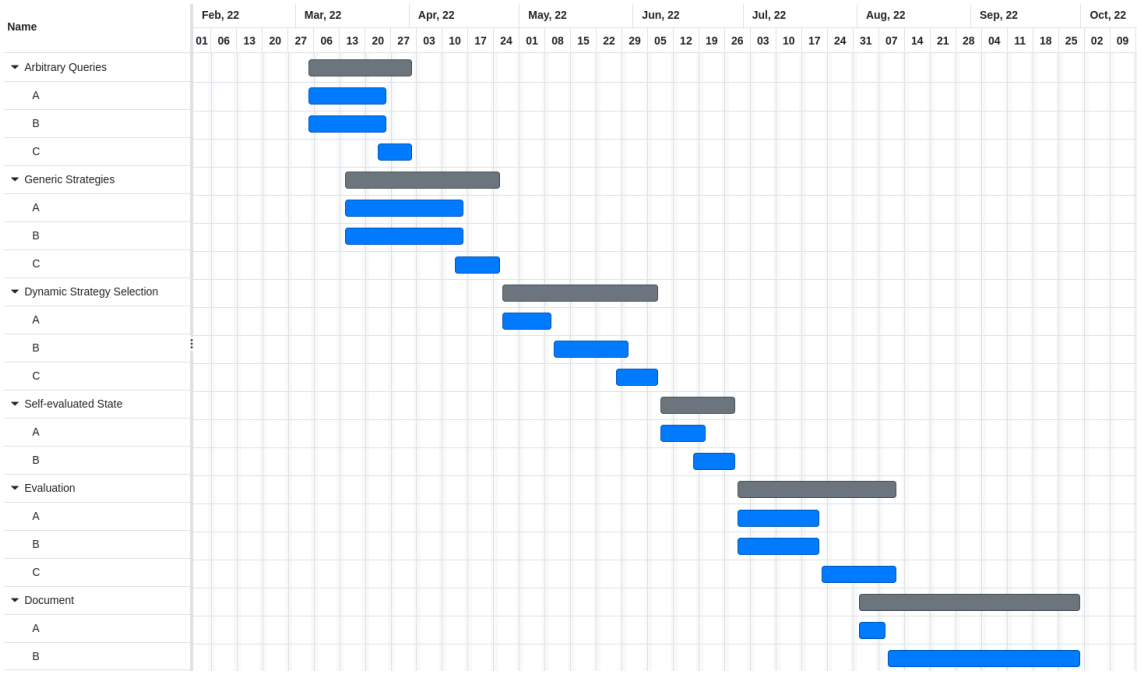

Figure 3.1: Gantt chart of the work plan

3.4 Summary

In this chapter, we presented our plan to research a novel search mechanism that will improve search flexibility and performance such that one may use arbitrary queries and dynamically selected strategies to find the location of resources. This work is performed in collaboration with Protocol Labs.

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 (Apr. 2010), 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* (Sept. 2017), 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 Mar. 2012), 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).

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

[7] 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, 14, 17, 21).

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

[9] 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. 20).

[10] 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).

[11] 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).

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

[13] 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. 12).

[14] 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).

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

[16] *Gnutella Protocol Development*. URL: http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html (cit. on p. 22).

[17] 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, 16, 17, 23).

[18] 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).

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

[20] 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 p. 15).

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

[22] J. Leitão. "Topology management for unstructured overlay networks". PhD thesis. 2012 (cit. on pp. 13, 14).

[23] 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).

[24] 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).

[25] 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, 30).

[26] 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. 12).

[27] 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. 7, 25).

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

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

[30] 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. 7, 8).

[31] 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).

[32] 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, 24, 25, 28, 29).

[33] 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).

[34] W. Shi et al. "Edge Computing: Vision and Challenges". In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. DOI: 10.1109/JIOT.2016.2579198 (cit. on p. 1).

[35] 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 p. 7).

[36] 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, 23).

[37] 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 Apr. 2008), 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, 16, 22, 23).

[43] *TC(8) — Linux manual page*. URL: https://man7.org/linux/man-pages/man8/tc.8.html (cit. on p. 30).

[38] S. M. Thampi et al. "Survey of search and replication schemes in unstructured p2p networks". In: *arXiv preprint arXiv:1008.1629* (2010) (cit. on p. 14).

[39] 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. 7, 9, 11).

[40] 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. 19).

[41] 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 pp. 19, 30).

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

[44] 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).

[45] 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).

[46] 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 (Mar. 2018), pp. 127–166. ISSN: 0743-7315. DOI: 10.1016/J.JPDC.2017.11.010 (cit. on pp. 11, 13, 15).