NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

JACINTA LOPES DE SOUSA

BSc in Computer Science

AFFINITY BASED OVERLAYS FOR DECENTRALIZED SYSTEMS

Dissertation Plan
MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
February, 2023

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

AFFINITY BASED OVERLAYS FOR DECENTRALIZED SYSTEMS

JACINTA LOPES DE SOUSA

BSc in Computer Science

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

Dissertation Plan
MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
February, 2023

Abstract

Currently, decentralized systems are rising in popularity, not only due to the Web3.0 paradigm and blockchains but also other systems such as the Inter Planetary File System (IPFS). Decentralized systems are scalable and can utilize resources that would otherwise be idle in the nodes, machines belonging to the system.

Peer-to-Peer systems are a category of decentralized systems that usually make use of overlay networks. These connect nodes and give them the means to find each other to communicate and cooperate, ensuring the correct operation of the system as a whole. Most overlay networks do not distinguish between nodes for their inherent differences and capabilities, which can lead to suboptimal virtual links, overlay topologies and workload distribution.

In this work, we propose a novel overlay network that can operate while taking into account the heterogeneity of the nodes in the network, be it related to capacity or content. The idea is to group nodes by affinity over some common properties, which can be either low level or high level. Low level properties include CPU architecture, presence of a GPU and link latency. The applications being executed by the node and the data stored are some examples of high level properties.

Finally, we plan to experimentally evaluate the devised solution by comparing it with the state-of-the-art overlay networks using emulation, along with proving its applicability in real systems with a set of pre-defined use cases.

Keywords: Overlay Networks, Decentralized Systems, Peer to Peer, Overlay Topology Management, Unstructured overlays

Resumo

Sistemas descentralizados têm ganho popularidade, não só devido ao paradigma Web3.0 como também a *Blockchains* e outros sistemas, como o *Inter Planetary File System (IPFS)*. Estes sistemas são escaláveis e podem utilizar recursos de máquinas do sistema, ou nós, que de outra forma estariam inutilizados.

Os sistemas entre-pares são uma categoria de sistemas descentralizados que frequentemente utilizam redes sobrepostas. Redes sobrepostas ligam nós e permitem que estes se encontrem para comunicar e cooperar, garantindo a operação correta do sistema. A maioria das redes sobrepostas não distingue entre nós tendo em conta as suas capacidades, podendo criar ligações virtuais, topologias de rede e distribuições de carga subótimas.

No presente documento, propomos uma rede sobreposta inovadora, que pode operar tendo em conta a heterogeneidade dos nós na rede, em termos de capacidade ou conteúdo. A ideia é agrupar nós por afinidade de acordo com algumas propriedades comuns, que podem ser de baixo nível ou de alto nível. Entre as propriedades de baixo nível podemos considerar a arquitetura do processador, a presença de placa gráfica ou a latência das ligações. As aplicações a serem executadas nesse nó, dados armazenados são exemplos de propriedades de alto nível.

Por fim, planeamos avaliar experimentalmente a solução desenvolvida, comparando-a, por meio de emulação, com as redes sobrepostas do estado da arte e demonstrando a sua aplicabilidade em sistemas reais, com um conjunto de casos de uso.

Palavras-chave: Redes Sobrepostas, Sistemas Decentralizados, Sistemas Entre-Pares, Gestão de Topologias, Redes Sobrepostas Não Estruturadas

Contents

List of Figures

Acronyms

CDN Content Distribution Networks *(p. 10)*

DHT Distributed Hash Table *(pp. 10, 17, 20, 23, 24)*
DOLR Distributed Object Location and Routing *(p. 17)*
DSHT Distributed Sloppy Hash Table *(p. 22)*

NAT Network Address Translation *(pp. 10, 11)*

P2P Peer-to-Peer *(pp. 7–14, 28)*

RTT Round-Trip-Time *(p. 23)*

TaRDIS Trustworthy and Resilient Decentralised Intelligence for Edge Systems *(p. 3)*
TCP Transmission Control Protocol *(p. 10)*
TTL Time-to-Live *(pp. 13–15, 22, 25, 27, 28)*

UDP User Datagram Protocol *(p. 10)*

VoIP Voice over IP *(p. 10)*

XOR Exclusive OR *(pp. 15, 19, 23)*

1 Introduction

The last few years have witnessed rapid technological developments and the rise of novel paradigms, such as Web3.0, a new version of the World Wide Web with a focus on decentralization integrating "Blockchain technology and token-based economy" [19], and edge computing, which emerged as an extension to the cloud computing paradigm as a strategy to reduce the latency of cloud computing by moving computation, resources, and storage closer to where they are requested typically, the edge of the network where clients are located and access and produce data [34]. These two new paradigms have decentralization in common.

Decentralized systems are computer systems, often heterogeneous, connected by a network in which processes and resources are spread across multiple computers. No computer is the sole authority, meaning that there is no central coordinating unit, often called a server, to control the system. In contrast, nodes in a decentralized system tend to have similar responsibilities and behaviours and cooperate to achieve a particular goal.

Decentralized systems have risen to popularity before, in the early 2000s, when Peer-To-Peer systems [1, 6] were popularized by Gnutella [16] and BitTorrent [11] which were themselves inspired by Napster [8], a partially decentralized file sharing application. Peer-to-Peer systems are decentralized systems in which computers, or nodes, take both the roles of a server and a client and share the load of the service amongst every node. Nowadays, these systems are again rising in popularity, not only due to blockchains [55, 58] but also other systems, such as the Inter Planetary File System (IPFS) [4].

Benefits of decentralized computing systems include the utilization of resources that would otherwise be idle in computers that are part of the system and the ease of scale, as the amount of computational resources available to support the operation of the system grows with the number of participants in the system [50].

Decentralized systems usually make use of overlay networks [1, 6], logical networks that operate at the application level above the IP networking suite. These connect devices and give them the means to find each other to communicate and cooperate to ensure the correct operation of the system as a whole.

1.1 Problem Statement

Most overlay networks do not distinguish between nodes for their different capabilities. Similar nodes may be found to be distant on the network so, when querying the overlay for nodes of similar capability to execute a specific task, the overhead will be large, as several queries will have to happen to request the task from specific nodes. If a node shares the load of work equally among its neighbours, some neighbours will take longer than others as they do not have the same capabilities. This is a consequence of the inherent network heterogeneity. Efficiency is lost with the communication overheads imposed by the overlay network, as the logical distance among nodes is not optimized with concern for the underlying physical network or any other high level criteria.

1.2 Objectives

There are many solutions for building and maintaining overlay networks, depending on the use case. Some have structure to ensure easier and faster look-up of information, which usually makes recovery times longer when the rate of concurrent joins and departures is high (churn [6]). Others form a random graph, which does not suffer from the same problem as the structured ones in highly dynamic environments, but generate greater overhead when dealing with queries and look-ups for information [1].

As mentioned previously, the nodes in the system can be heterogeneous, and in most cases they are very different from each other. This means that some modes can deal better with certain amounts of load than others, as they own (or have available) different amounts of computational resources. Most overlays do not take the inherent difference of the nodes into account when they are being created, instead assuming that all nodes have similar capacities [7, 53].

While there are solutions such as super-peers [9], in which a hierarchy is formed in the overlay and nodes with higher capacity take a larger and more central role within the overlay, these only make a single distinction of nodes with higher computing capability. It is not clear what can make a node a super-peer as compared to their peers in an overlay, since they most likely only have knowledge of the nodes directly connected to them and those may not be a representative sample of the whole system, to infer their relative computational capacity regarding the rest of the network.

Therefore, the goal of this work is to develop an overlay that can operate while taking into consideration the heterogeneity among nodes of the system, creating affinity groups of devices that share common properties. These properties are set by the developers or the applications and can be low level properties, such as, computation capacity, latency, network capacity or higher level properties, such as, the applications being executed in that node, etc. By having affinity groups defined, in which nodes with similar attributes are closer to one another, the location of nodes for a specific request/computation needs a single query. If a node that matches the search parameters is found, its neighbours

are most likely the next best matches. When a request requires many nodes to execute some function, if they have to coordinate and communicate with each other, the (physical) proximity between them is advantageous, as it lowers communication overhead.

1.3 Expected Contributions

The expected contributions of this work are the following:

- The implementation and evaluation of a new overlay, where the heterogeneity of nodes is taken into account to form affinity groups of nodes with common properties.
- An experimental comparison between the new overlay and other overlays that may provide a semblance of similarity to the proposed one.

1.4 Research Context

This work has been partially motivated by the Trustworthy and Resilient Decentralised Intelligence for Edge Systems (TaRDIS) [52] project, from the Community Research and Development Information Service (CORDIS), held by the European Union.

"The project TaRDIS focuses on supporting the correct and efficient development of applications for swarms and decentralized distributed systems, by combining a novel programming paradigm with a toolbox for supporting the development and executing of applications." [52]

1.5 Document Structure

The remainder of the document is organized as follows:

Chapter 2 presents relevant concepts that will act as groundwork for the work to be conducted in the future. It goes over the concepts of Peer-to-Peer, Overlays and their topology management techniques. This chapter presents available solutions that fall under the previously explained concepts, identifying their main advantages, disadvantages, and possible contributions to the work at hand.

Chapter 3 details the plan for the work to be conducted, the strategy, the main objectives, possible challenges that need addressing and a review of the tools that will be used to create the solution. This chapter also presents the calendarization of future activities organized as tasks.

2

Related Work

This chapter presents relevant concepts regarding overlay networks used in Peer-to-Peer systems and services. There is a review of widely used overlay networks along with the state of the art.

The structure of this chapter is as follows: Section 2.1 contextualizes Peer-to-Peer within Distributed Systems, focusing on Peer-to-Peer systems' architecture and services. In Section 2.2 we present a review of overlay networks, their classifications, and existing solutions. Strategies to manage and/or maintain the overlays' topology are discussed in Section 2.3. Lastly, Section 2.4 summarizes the previous sections and revises the insights provided by the discussed solutions and how they could be helpful for the work presented in this document.

2.1 Distributed Architectures and Peer-to-Peer Systems

A Distributed System is a collection of computer systems, which we will be referring to as nodes throughout this document, connected by a network, in which processes and resources are spread among different nodes. There are many ways to classify Distributed Systems: their use, e.g., high performance computing or pervasive computing, the logical organization of their software components, often described as software architecture, e.g., layered, object-based, resource-centred or event-based architectures, or system's architecture [50].

This work will discuss systems by following the system's architecture classification, which relates to the placement and interaction between software components across multiple machines. Each architecture has its advantages and disadvantages, which must be analysed when considering the system being built and its uses.

Centralized Architectures, also called layered architectures, divide nodes into two roles: clients and servers. The understanding and management of the complexity of the system is easier if one considers that clients request services from servers and wait for a response [48].

The nodes performing the server role are responsible for the bulk of the work to provide the service, that being either information processing, computation or storage,

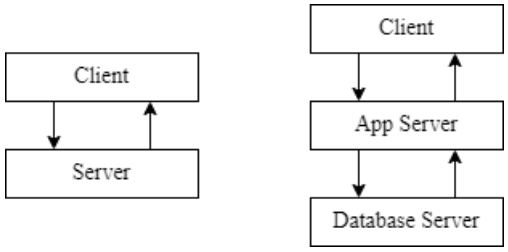

Figure 2.1: Layered Architectures: Simple client-server to the left and three-tiered architecture to the right.

and as such, they are the central components of the system. This architecture relies on communication happening only between clients and servers. A client requests a service to the server and waits for its response. A server receives many requests from different clients and services them.

A node can perform both client and server roles for different sets of nodes. This leads us to the layered term used to coin this architecture. Clients only see the layer of servers, often abstracted to the illusion of a single server, that they interact with and have no knowledge of other clients. The servers can, however, interact with another layer of servers acting themselves as clients.

Figure 2.1 is a visual example of the abstraction created by layered architectures. In the three-tiered architecture, seen on the right side of Figure 2.1, the first layer represents the clients that request some service of the Application (App) Server. Assuming the App Server layer is only responsible for the processing and computation of information, there can be another layer below it, from which the App server requests services, like a Database Server. The client only sees the abstraction shown on the left, one central component acting as server that provides the service they request, even if there are multiple servers and multiple layers of servers. The server can act as a client to other servers, lower in the architecture.

Software components are distributed vertically over different nodes. Since specific nodes perform specific roles, their underlying machines can be tailored for the role they execute. Control of data and its security is a responsibility of the servers, which comprise a much lower number of nodes than the clients. These servers can suffer from strain, since they have to attend to a high number of clients and, usually, perform the bulk of the computation [50]. Servers, even when replicated, can become the central point of failure of the system. Scaling these systems to accommodate for growth is often a matter of upgrading the physical machines for faster and more efficient ones, this is often referred to as *scaling up*.

Decentralized Architectures, are those in which all nodes perform **both** roles: client and server. Communication happens between all nodes, as they act like clients requesting some service of other nodes and as servers when services provided with their own resources, the ones they are sharing with the system, are requested from them. Nodes are autonomous,

their participation in the system is determined locally as there is no central point to coordinate and control them.

This distribution of software components is horizontal, since all nodes have the same logical components and are equal from a high-level perspective. The behaviour of the system is symmetrical, also a term used to name this architecture, and determined by the actions of all the nodes belonging to the system [50].

Each node keeps a share of the system's complete dataset or resource pool, they typically store metadata to facilitate search and information about some other nodes in the system. Usually, each node also keeps a set of logical connections to a subset of other nodes in the system, named neighbours. Nodes can also keep information about their neighbours and use it to adapt to changes in the network, when the joining and leaving/failure of nodes is frequent.

These systems are very scalable and can grow to thousands of nodes, often heterogeneous in terms of hardware and software, harnessing the power of computers all over the Internet without incurring in excessive operational and monetary cost [56].

In the specific case in which the resource being shared is data, an important advantage of decentralized architectures is that owners have full control over their own data, by storing it in their own machine. They may also have the possibility of anonymity, by publishing the data in the system and joining the pool of nodes that share the responsibility of storing it, becoming indistinguishable from the others. These systems are, however, hard to control and secure, since there is no central management component and resources are spread all across the system.

Communication happens between any combination of nodes, potentially being routed through other nodes belonging to the system. Updates to information are not instantaneous since they have to be disseminated across the network to the nodes involved. This makes the system's communication model more complex than that of a centralized system.

Hybrid Architectures are a mix of the two previously described architectures. They harness their advantages or bypass their disadvantages. An example of this would be to have a decentralized system in which nodes share their resources but with the addition of a logically centralized component that would control and monitor the service, allowing for a better lookup strategy with global indexing.

This work focuses on a class of decentralized or hybrid systems, **Peer-to-Peer (P2P) systems**, which will be presented in greater detail in the following sections.

2.1.1 Peer-to-Peer Systems

"P2P is not a new paradigm and in fact has already been applied in the original Internet's design, for example, in basic Internet routing" [1]. The P2P approach has been most notably used for resource location by building overlay networks, such as Gnutella [16], Freenet [10] or Pastry [47], on top of the IP network [56].

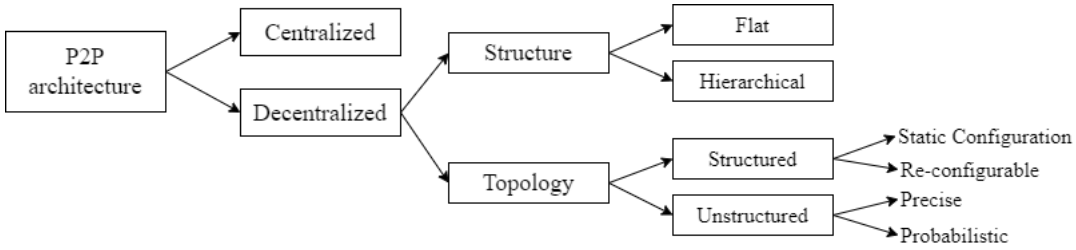

Figure 2.2: Classification of P2P Systems, extracted from [56].

Overlay networks are logical or virtual networks constructed in the application layer, they provide connectivity, routing, message passing, and functionalities to the system. There is a wide range of algorithms and architectures for overlay networks, some of which will be discussed in more detail in Section 2.2.

Nodes are addressable and share part of their resources, some of which can be processing power, storage capacity, and/or network link capacity. All nodes are used to provide the service and content to all other nodes. P2P systems are known to take a decentralized architecture, although there are a few that use the hybrid approach, like Napster [8] one of the first and most popular file sharing applications to explore the peer-to-peer paradigm.

In principle, these applications could operate over the physical networking layer, but using an overlay has the advantage of supporting application specific requirements and identifiers along with providing the possibility of additional services that support network maintenance, authentication, trust, etc [1].

2.1.1.1 Peer-to-Peer Architectures

As described in the previous section, P2P systems can fall under two distributed system classifications. The variety under the paradigm requires a new classification. Figure 2.2 displays the classification presented by [56], dividing the possible architectures into Centralized and Decentralized.

Centralized P2P Systems fall under the hybrid distributed system classification described in Section 2.1. There is one or more central servers that can serve numerous purposes: authentication of nodes, entry point for the network, indexation lists, resource locators, and schedulers for the access to those resources.

The flow usually starts with a client requesting something from the central server and, upon receiving the response, querying specific nodes within the overlay network, the decentralized part of the system.

Central servers, while useful for control and management of the system, often become possible points of failure, and bottlenecks, when clients or request rise in number.

Decentralized P2P Systems match the decentralized classification described in Section 2.1. Figure 2.2 divides them according to structure and topology.

A system's structure can be flat, where load and functionality are distributed evenly across all participating nodes. Most of the existing decentralized P2P systems follow this structure. On the other hand, nodes can belong to different layers, and the system would follow a hierarchical structure. The load in this case is often different depending on the layer in which a node finds itself. A common example of hierarchical structure are super-peer systems, in which nodes are separated into two ranks. The higher ranking node, called super-peer, is meant to handle more load and provide additional functionality for the system as opposed to lower ranking nodes, which mostly act as clients to the higher rank nodes.

The topology classification relates to whether the logical network disposition of the nodes is generated on a priori structure or not. The distinction is made by way of querying for an object and how data is mapped, or not, onto the nodes. Basically, a structured P2P system is one in which nodes are connected to each other following rules and restrictions to form a specific graph topology, or in which resources are mapped onto nodes and the resource lookup is made by following that global mapping strategy. In unstructured P2P systems, nodes are connected at random. This distinction is further explained in Section 2.2.

2.1.2 Peer-to-Peer Services

There are many uses for Peer-To-Peer systems, they can be divided in a simplified way into two categories relating to what is being shared: computational resources or data [56].

Resource sharing enables users to take advantage of available resources (CPU cycles, disk storage and bandwidth capacity) within the network. The P2P paradigm harnesses underused resources to perform tasks that would require a much more expensive machine to complete in a centralized way. Data storage devices are used to build a wide area storage network and bring data closer to users [56].

In data sharing applications, users are allowed to access, modify or exchange data stored in other machines or their own in a flexible manner. The difference between data sharing and sharing data storage as a resource is the following: Sharing data storage is simply storing the user's own content on another machine, while data sharing involves access privileges and strategies to propagate the changes made to the files by any user.

With this division we can discern two uses for P2P already, one is digital content sharing, e.g., Gnutella [16], BitTorrent [11], IPFS [4], and the other is computation, most notably scientific computation like the SETI@home project [57]. Another appliance of the P2P paradigm would be to have distributed databases.

Media streaming is also a service that can be provided by a P2P system. One or more peers that have all or part of the requested media can supply it to the requesting peers, who will in turn become suppliers for other requesting peers. Since every node is contributing with their storage and network bandwidth, the system's capacity is amplified when compared to a client-server architecture [24].

Messaging and gaming can take advantage of decentralized architectures as well, if there is enough information locally one can simply bypass the communication with a central server and communicate directly with the destination peer. In messaging, that would be the receiver of the message. In gaming, that would be other peers that need the complex models involving 3D graphics, in which the processing necessary is shared among nodes [56].

Also within the communication realm, there are Voice over IP (VoIP) systems, an example is Skype [49], which previously used a Distributed Hash Table (DHT) to translate from user to IP. Currently, it uses a centralized server for authentication, and super-peers to help set up the connection between two clients, if one or both of them are behind Network Address Translation (NAT)s or firewalls. Once that connection was established, communication is solely happening between two or three nodes (in a tunnel-like way using the super-peer) [3].

2.2 Overlay Networks

An overlay network is a logical or virtual network that is built on top of an existing network, be it physical or logical. This isn't a novel concept, nor is it only applied to P2P.

For example, *routing overlay networks* support alternative routing strategies with no application-level processing, these are used for end-system broadcast, resilient overlay networks and deploying experimental versions of IP. Content Distribution Networks (CDN)s "act as a Web hosting service, providing an infrastructure for distributing and replicating the Web documents of multiple sites across the Internet" [50], are also overlays by definition [45]. In this work, we will focus our approach in overlays built at the application layer, often used in P2P systems.

Nodes in an overlay network are connected via logical links, that represent neighbouring relationships and can abstract the path between neighbours, be it a single or multiple underlying network links. An overlay network therefore consists of a set of distributed nodes and the links that connect them, and can be interpreted as a graph. Nodes often keep information regarding their neighbours in one or more local sets, called views, depending on the protocol. The protocols must then guarantee that eventually when a node joins it will add nodes to its neighbour set and in turn be added to other nodes' neighbour sets, and when a node leaves it will be removed from the sets it was part of.

Overlays rely on the underlying network for basic networking functions, like routing and forwarding messages. Most overlays today are built at the application layer level, over the IP networking suite, using either User Datagram Protocol (UDP) or Transmission Control Protocol (TCP). While relying on it, they deal with some of the underlying network's shortcomings, while offering new features for the application without the need to change the physical routers [1, 53].

An ideal overlay would be striving to optimize a set of requirements. Starting with the **efficiency** of the system, it often relates with the attempt to minimize the number of

hops for a lookup or for activities that have to be conducted among participants. Another requirement is **scalability**, in which the goal is to achieve a network with numerous peers without significant performance degradation. The lack of centralized control and frequent system membership changes require nodes to **self-organize** in the presence of network instability or churn, towards a stable configuration. "Churn is a metric that is especially pertinent for P2P overlay systems. Churn pertains to the rate of arrivals and departures in the system." [53]

Since nodes and links can fail or nodes can leave the network at any time, there is a need to replicate resources. Redundancy is necessary for **fault-tolerance**. Lastly, but most importantly, P2P systems in general and overlay networks in particular depend on the **cooperation** of the participants. There must be trust in other nodes regarding their behaviour [1].

Compared to centralized solutions or those that require change in routers, overlay networks offer some advantages:

- Since an overlay network does not require changes on existing hardware, it can be grown node by node, being **incrementally deployed** as needed.
- Routing and forwarding decisions can utilize metrics, taking application-specific concerns into account and **adapting** to them, unlike the underlying network, which must be general purpose.
- With a sufficient number of nodes in the overlay, an overlay may be able to offer multiple independent paths to the same destination, which would make the system more **robust** to node or network failures.
- They can support specific application semantics and identifiers, along with providing **additional features** not supported by the Internet.

No advantages come without their counterparts, some challenges and limitations of overlay networks include:

- The most used underlay protocol, IP, does not provide universal end-to-end connectivity. Firewalls and NAT devices pose **reachability issues**. Special solutions are needed to overcome them [14, 28].
- Their **management** may prove cumbersome if there are more than one administrative domains. An administrator is often removed from the actual physical devices, which in turn requires advanced techniques to detect failures or suspicious behaviour of nodes.
- The nodes that constitute the overlay are heterogeneous and cannot route and forward packets as efficiently as dedicated devices, i.e., routers. This fact leads to **overhead**, which is heightened when considering that overlays may not have information about the underlying topology to optimize their routing [35, 23].

P2P and overlay networks go hand in hand, and one of the most important goals is to locate resources shared by nodes. There are several types of search queries to lookup resources: Exact match search, keyword search or arbitrary search queries [31]. *Exact match search* queries are those in which the node has the knowledge of the desired resource's unique identifier and queries using that identifier. *Keyword search* queries in which the nodes are looking for a resource or resources that best match one, or a set of, keywords. *Arbitrary search* queries are more general and describe resources being looked for using one or more (non-unique) properties of the resources, e.g., type of file, size, etc.

Each overlay network can be defined by the decisions made on the following six design aspects:

Identifier space The choice of a virtual identifier space to map nodes and/or resources, in which there may be a closeness metric between identifiers, is important for the preservation of application semantics, message routing and is independent of physical location.

Mapping of nodes and/or resources into a unique virtual identifier from the defined identifier space.

Structure relates to the topology of the virtual network, which can be interpreted as a graph. Relevant properties include the graph's structure, and a node's minimum and maximum in/out degree, the direction of links, etc.

Routing strategy is the basic service to route a request from a node to another, or a lookup for either nodes or resources within the system.

Maintenance and management strategies are the means through which the system ensures the structure's integrity and/or network's connectivity when nodes join or leave.

There are several types of network overlays and different ways of classifying them. They can be divided by generations, often seen when describing file sharing systems. The first generation combines decentralized and centralized architectures, e.g., Napster [8]. A second generation's system is fully decentralized, e.g., Gnutella [16] and BitTorrent [11], and, when security or anonymization is added, like in Freenet [10], it is sometimes referred to as the third generation.

Another more common classification is to divide them into structured, unstructured or, occasionally, hybrid (a combination of both). The difference between the two first designs lies in the topology that the collection of links among nodes generates [56]. In the following sections, we will explain these in more detail and provide examples of overlays that fall under their category.

2.2.1 Unstructured Overlay Networks

In unstructured overlays, "a node relies only on its adjacent nodes for delivery of messages to other nodes in the overlay." [53] The structure formed by unstructured overlays is fundamentally random, usually generating scale-free or power-law random graphs. They are considered the first generation of P2P overlays.

A peer's list of neighbours is constructed in an ad hoc fashion [50]. When a new peer joins an unstructured P2P system, it will often contact a well-known contact node to receive a list of other peers in the system. This list will be used to find more peers or to replace non-responsive ones, so there is the need for the list to be updated periodically.

The primary feature of unstructured peer-to-peer overlay networks is that allocation of resources, such as file storage, is unrelated to overlay topology. There is no mapping of object identifiers to certain nodes that will store them. As a result, mechanisms for resource location are either random or centralized.

By having a centralized index server, to which peers, upon joining the overlay, send an index of files they keep locally, the server will then perform queries on behalf of the peers. While it provides guarantees for completeness and can be used to circumvent the end-to-end reachability problem described in Section 2.2, from a fault tolerance perspective it will constitute a single point of failure, not to mention the need for management and maintenance of it by some organization.

If opting for a fully decentralized unstructured overlay, the other option is using random search, usually flooding, gossiping or random walks [6, 31]. Flooding requires nodes to keep indexes of the resources they share with the system. When a query is made to a node, if it does not contain any resource matching the query, it will forward it to all of its neighbours. The query is propagated in the network, and nodes whose data matches will usually send the result directly to the node who requested the data. This technique strains the network significantly, as it often happens when flooding is involved. It is a method that avoids centralized/single point of failure and allows for keyword and arbitrary search queries, but it does not do well with locating unpopular/rare resources.

To lower the overhead on the network, some approaches instead rely on the use of random walks with an incremental Time-to-Live (TTL) value. As with flooding above, each node keeps an index of resources it shares with others, but when receiving a query, that does not match any of the data it stores, it will pick a single neighbour at random and forward the query to it at the same time decrementing the TTL value in the query. If the TTL reaches zero and the node who receives the query does not contain the data queried for, it will either notify the client or it will drop the message. Eventually, the client will time out and resend the query with a higher TTL value.

This method has its advantages, when a matching object is found, the node will reply to the client and the query will not be forwarded, therefore there will be no unnecessary overhead in the network while forwarding an already answered query. It is however a very slow and inefficient way of locating an item. Not only does the client may have to

query several times with higher TTL values, nodes probed can happen to be the same as they are chosen and random and queries are considered as independent of one another. It does, however, work for popular items in a few hops, much like flooding would, although with a lower communication cost.

As locating items as efficiently and quickly as possible is an important necessity, there are some heuristic key-based routing techniques. Freenet [10] is an example of a peer-to-peer system that leverages this technique, and it will be explained in more detail in Section 2.2.1.2.

While the location of items within the network is not ideal, there are advantages to unstructured overlays:

- Since there is no specified requirement for structure or topology, unstructured overlays are **resilient** to concurrent join and departure of nodes potentially at high rates (often mentioned as churn) [24, 40].

- Unstructured overlays usually **support complex search**, since upon receiving a query they check their file metadata for a match, they support searches with various keywords and arbitrary complex descriptions of resources [56].

Unstructured P2P networks have their own limitations. Their major problem is, as mentioned before, low search efficiency, especially for unpopular resources of which there are few copies in the system. Search for such a resource may lead to large-scale flooding or many repeated random walks with higher and higher TTL values. There is no upper bound on the hops needed to reach an object [24]. They do not exploit the physical network's properties and topology, which leads to a topology mismatch between the two networks and results in the suboptimal links of the overlay network, lowering their efficiency [22].

In the following sections, we provide a description of several peer-to-peer systems whoe design relies on unstructured overlay networks.

2.2.1.1 Gnutella

Gnutella [16] is an open, decentralized membership and search protocol, mostly used for file sharing [25, 46]. It was designed with the following goals in mind: performance, scalability, reliability, and the ability to operate in a dynamic environment with high churn [46].

The Gnutella protocol does not have an identifier space, nor does it map resources to specific nodes. Instead, nodes make their resources available when they join the network. When other nodes download that resource, they can also make it available to the system [5].

The overlay forms a random graph, while nodes join the network by connecting to one of several known contact nodes, they will quickly create connections to other nodes within the network and build their neighbours set. In later versions of Gnutella, there is a change regarding this structure mostly motivated by the scalability of query routing on the overlay, this will be discussed in Section 2.2.3. Links between nodes are unidirectional, and there

are no bounds to a node's in or out degree, meaning the sets do not have a minimum or maximum number of elements in their neighbour set for ideal overlay connectivity.

As for the routing strategy, the queries are flooded through the network with a defined maximum number of hops, a TTL value. The interpretation of those queries is done by the node that received them, and it will search for the resource locally while trying to match the keyword(s) to one or a mix of the following: the resource's title, its metadata, the resource itself, etc.

There is no maintenance of any topology, if a node finds itself disconnected it must simply redo the joining steps to enter the overlay network again.

Flooding queries, while necessary to find the resources, causes a lot of strain to the network. It scales poorly with as the number of nodes in the system rises along with the number of requests done by them.

2.2.1.2 Freenet

Freenet is "an adaptive peer-to-peer network application that permits the publication, replication, and retrieval of data while protecting the anonymity of both the authors and readers." [10] The main concerns addressed were the design of this system are privacy and availability.

This protocol relies on an identifier space that is composed of all the possible hashes created by a selected hash function.

The generation of a node identifier starts with the hashing of a seed, which is then forwarded to nodes belonging to the network, who will hash random seeds themselves and Exclusive OR (XOR) them with the received hash. Hashing the resulting XOR and forwarding it to a random neighbour, until the TTL value is hit, and the final hash is returned to the joining node, as its identifier. Resource identifiers are created by hashing the XOR of the hashes of two public keys together, the first one is attained deterministically from a short text description of the resource, called the *keyword-signed key* and the second one is the *signed-subspace key*, a randomly generated key to identify the user's namespace. The private key pairs of both these keys are used to sign the resource.

There is no structure imposed on the logical network, it is much akin to a random graph. Links are symmetric and while there is a maximum value for the number of routing table entries, there is no enforcement over the number of connections a node should have.

As for the routing strategy, Freenet does something different from the majority of unstructured overlays. Routing is based on the knowledge acquired by the node as queries are sent and processed. Each node maintains a routing table, which is updated whenever a query is forwarded by them and eventually backtracks to the original requestor with the reply. The node will record the hash of the resource requested and the node through which the request found fruition. The routing is made according to the resource identifiers, if there is a similar identifier within the table, the query will be forwarded that way. The propagation backtracks if a loop is created while forwarding the query message. A node,

upon receiving a query message with a previously seen request ID, will send a request failed warning to the sender, who will then choose another node to send the request to.

2.2.1.3 HyParView

HyParView, short for Hybrid Partial View membership protocol, "supports gossip-based broadcast that ensures high levels of reliability even in the presence of high rates of node failure." [32] The main concern addressed by this protocol was the recovery time of neighbouring sets and their desired properties by application level message broadcast protocols.

HyParView does not define identifier semantics for nodes, nor does it map resources to identifiers. There is no concept of logical topology, a norm for unstructured overlays. Links between nodes are symmetrical, and there are bounds imposed on the cardinality of the neighbour sets. As for the routing strategy, a gossip or epidemic protocol is used.

Gossip dissemination protocols work by selecting a number t, known as the fanout parameter, of neighbours and sending the request or message to those t neighbours, and so on. If it is equal to the total number of neighbours, it is the same as a flooding protocol. To combat the network overhead while keeping the resilience provided by the redundancy of flooding, the fanout parameter needs to be lower than the number of neighbours but high enough to ensure resilience (higher than three is the rule of thumb).

HyParView works by having two peer sets per node. The first set, called active view, stores the information about the symmetric neighbouring relations of the node, as the usual sets kept by other overlays. This set is maintained by a *reactive strategy*, which means it only changes when there is a node join, departure or failure. It remains unaltered if the system is stable. Whenever a departure or a failure happens, it will replace it with a node stored on the other set, the passive view, which keeps a larger sample of nodes for this purpose. The passive view is kept using a *cyclic strategy*, in which, periodically, a node will arrange a subset of nodes of both their views, include itself and forward that information using a random walk to some other node within the network.

The active view is smaller than most neighbour sets, because its maintenance is assured by the larger passive view. The fanout parameter is usually set to the maximum number of neighbours minus one, the sender of the message to be disseminated.

2.2.2 Structured Overlay Networks

Structured overlays are overlays “in which nodes cooperatively maintain routing information about how to reach all nodes in the overlay” [53]. The topology of the network formed is tightly controlled and content/data is stored in specific node to make queries more efficient, based on the identifiers of content and nodes.

These are typically based on the notion of a semantic-free index [53], one that does not capture data semantics and is not readable for humans as opposed to keywords,

document names and keys. A semantic free index usually corresponds to the index by a hash mechanism [21].

This index will work as a key for key-based routing. Each node belonging to the overlay is assigned an identifier from the same set of all possible hash values and is responsible for storing data associated with a specific subset of keys, in the form of a *(Key, Value)* pair [50]. These keys are attained through the hashing of a specific item, a subset of its properties or its content.

Peer-to-peer systems using key based routing are called Distributed Object Location and Routing (DOLR) systems, a specific instance of a DOLR is a DHT, in which identifiers and keys are computed using consistent hashing. Most structured overlays that exist implement a DHT [2].

The overlay network needs to support scalable storage and retrieval of (key, value) pairs [53]. Compared to unstructured overlays, structured overlays provide a maximum number of hops until the object is found in the overlay. This is important when the object of search has low popularity or low frequency of searches. "In order to provide deterministic routing, peers are placed into a virtualized address space, the overlay is organized into a specific geometry, and a converging distance function over the combined object and node identifier space is defined for the routing forwarding algorithm." [7]

In terms of routing, each peer has a local routing table to be used by the forwarding algorithm. It is initialized when a peer joins the overlay, using a specified bootstrap procedure, and it is changed periodically as part of overlay maintenance.

Nodes are organized in an overlay that adheres to a specific, deterministic topology: a ring, a binary tree, a grid, etc. This topology is used to efficiently look up data.

Structured designs can be differentiated from one another with the following characteristics, as described by [6]:

Maximum number of hops taken by a request to reach its destination node, given an overlay of N peers.

Identifier space organization can be flat or hierarchical, usually uniformly distributed.

Next hop criteria relates to the convergence of routing to the correct destination. Often referred to as distance metric. The routing algorithm must strive to shorten the distance between the current position and the desired destination. A few examples are XOR metric, linear distance in a ring, Euclidean distance, etc.

Geometry refers to the graph properties of the overlay, like node degree.

Maintenance of the overlay upon joins and departures of nodes to the overlay. It can be active, or delay the correction of the topology for when it is needed.

Locality and Topology awareness to deal with the network mismatch problem, in which an overlay does not take into account the underlying network and uses it for efficiency

purposes, designs can form neighbouring relationships based on the proximity of nodes in the underlay.

While it is important to have a limit on the number of hops made by a query until the item is found, for performance issues, this way of addressing items does not support advanced queries, if there is a need to search by keyword or filter by any property, this sort of overlay does not easily allow for that.

These overlays may exhibit poor performance in highly dynamic environments in which churn, the concurrent entry and departure or failure of multiple nodes will affect the overlay greatly, plays a big part. Namely, because it will take more time than an unstructured one to recover, since it needs to ensure the links formed do not violate the constraints and the overall graph structure is maintained and that takes more effort and time. Keyword searches are more prevalent than exact-match queries (these are the only queries allowed in DHTs), and most queries are usually for the popular objects, not for rare ones, which somewhat benefits unstructured overlays [9].

In the following sections, we discuss the design o some of the more popular structured overlay networks found in the literature.

2.2.2.1 Chord

"The Chord protocol is a decentralized distributed hash table algorithm for connecting the peers of a P2P network together." [53] It works much like a key-value store, where there is the necessity of a key to retrieve a value. The main concern addressed by Chord "is the efficient location of the node that stores a desired data item." [51]

Chord's identifier space is given by the range of the consistent hashing function selected for the protocol. A node identifier is generated by hashing the node's IP address and port, with resources being mapped into the identifier space by their hashed key. *(Key, Value)* pairs are then stored in the node whose identifier is the same or, if there is no such node, immediately after to the hashed key. Whenever a new peer joins, if there are keys that should belong to them, they will be transferred from the current holder.

Chord forms a ring-like structure along the identifier space. Nodes do not need the knowledge of every other node in the system. In fact for correct routing, nodes need only to have a connection to the node whose identifier directly follows their own, often named successor.

For efficiency, nodes keep a routing table in which each position stores the identifier of the node that succeeds the node's identifier by at least 2^{i-1}, with i being the index of the entry at the table. Whenever a node receives a lookup request, it will check the routing table for the lowest closest identifier and forward it to that node. If the routing table is updated often enough, lookup complexity will be $O(\log n)$.

Every node periodically runs a function to learn about the new nodes, as well as a function to update the routing table. To deal with failures, nodes keep a list of possible successors, to replace the successor in case it does not respond.

2.2.2.2 Kademlia

Kademlia [42] is a "peer-to-peer system to combine provable consistency and performance, latency-minimizing routing, and a symmetric, unidirectional topology. Kademlia furthermore introduces a concurrency parameter that lets people trade a constant factor in bandwidth for asynchronous lowest-latency hop selection and delay-free recovery." [42]

The identifier space is a range from 0 to 2^x, the ID is an x-bit long string. Nodes are given uniformly random identifiers within the identifier space. The identifiers given to the resources work as a key to the resource and are defined by the application running over Kademlia. The resources act as values. They can be assigned at random, as well as being a hash of some of the resource's features, e.g. title, a keyword string, metadata, or even context [12], for a better load distribution.

The logical structure formed by the nodes is a binary tree, in which the bits of the identifier separate sets of nodes, known in the Kademlia protocol as buckets. Each node keeps sets according to the proximity of the nodes to the node itself. This proximity is measured by the XOR metric. If there are two nodes, each with an identifier, one simply has to XOR the identifiers together to learn the difference between them. Further away nodes will have different bits in more significant positions and, therefore, the distance given by the XOR will be greater. Nodes with similar identifiers, when compared, will have a smaller XOR metric.

The tree structure of the identifiers is not representative, however, of the overlay structure. Connections between nodes are not made according to proximity of its own identifier, or to the notion of binary tree. A node will keep a routing table with the entrances divided by the distance defined by the XOR metric, with more entries for closer prefix identifiers. This routing table keeps sets of k nodes.

Whenever a node receives a request for a resource, it will check if it has said resource stored locally. If it does not, it will check its routing table for the k closest identifier matches to the key and return them to the requestor. The requestor node will then query each of the k nodes for the key needed, if they return it, the procedure is done. If not, they will return a set of k nodes with the closest identifiers to the key that they possessed in their lists of contacts.

Requests routed and answered by a node will be useful to update its routing table. If non-responding nodes are found during this procedure, they can be replaced with newly found ones that match the same distance or those kept in cache. Whenever new nodes are found, but the respective entries in the table are full, they will be cached.

More relevant to the context of this work, libp2p [39],"a modular system of protocols, specifications, and libraries that enable the development of peer-to-peer network applications" [38], has an implementation of Kademlia [17] that allows for neighbour filtering by setting up a function to decide which connections are kept in the local routing table of the node. This library allows introducing application specific preferences to the logic that governs which peers to keep in the routing table when multiple alternatives exists,

whereas Kademlia originally was proposed to always give preference to maintain node identifiers in routing tables for nodes that are known for a longer time. As far as we know this approach has not been widely used, with the single example of its use being in the prototype of Bacalhau [54], a decentralized system to support distributed computations.

2.2.2.3 Optimization Technologies

Identifiers in the identifier space are often attributed randomly, by hashing a property of the node in question like IP, or IP and Port, so they do not account for the locality of the nodes.

Locality is important because by using any metric of closeness that relates to the underlying network, we are effectively lowering the routing overhead of the overlay network protocol, since there can be less unnecessary hops between nodes [43].

There are several works that address this problem, using the following techniques:

Peer Selection optimizes an additional property, e.g., latency, between nodes by having nodes prefer to store information about other nodes that minimize or maximize the additional property. Used in Coral [15], which will be explained in Section 2.2.3.3.

Coordinate System Transformation refers to the transformation of an existing physical coordinate system into a logical identifier space to be used by the DHT and that maintains the same properties as the original space. This technique is used in [18]

Identifier Manipulation involves separating the identifier into two parts, global and local. The global part will encompass the most significant bits of the identifier and is used to encode locality, while the local part will be randomly generated. [43, 20] focus on this technique.

2.2.3 Semi-structured Overlay Networks

This section describes a third class of overlay that combines aspects of both structured and unstructured overlay networks. These, as described in Chapter 1 and briefly mentioned in Section 2.1.1.1, take the heterogeneity of nodes into consideration and organize them into a hierarchical structure. Better nodes, according to some metric, are higher in the hierarchy.

This metric is usually computational power, nodes with more powerful being called super-peers or supernodes and usually form a network amongst each other. Less powerful nodes, called clients, connect to one or more super-peers [50]. Typically, each super-peer maintains a consolidated index of all the regular peers that are attached to it. All communication to and from a client node, goes through the super-peer to which it is connected.

Figure 2.3 displays an example of what a hierarchical overlay could look like, clients represented by the smaller black circles are connected to super-peers, the larger lighter

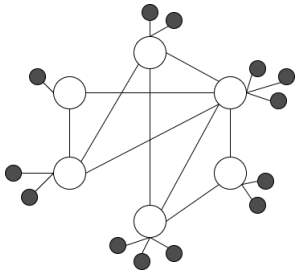

Figure 2.3: A simple example of a hierarchical overlay network, in which the network formed between super-peers is unstructured.

circles, which in turn are all connected to one another. The network formed between super-peers can be either structured or unstructured.

The routing of a resource lookup starts when the client node queries its super-peer. The super-peer will then check its directory, containing an index of the resources kept by itself and all other client nodes connected to it. If the resource is held by any client of that super-peer, it will simply return the information so that the requestor can form a connection with the resource holder. If no such information is found, the super-peer will then check for any information it has on other super-peers and query them for the resource, they will search within their clients and, if the resource exists, return information for the super-peer to provide to the requestor client, so that a connection is made between requestor and resource holder [56, 40].

This approach is particularly good if the network formed by the super-peers is unstructured and the lookup method is flooding. The network scales by reducing the number of nodes on the network involved in message handling and routing, causing less strain, as well as, reducing the actual traffic among them, with lookups and requests first performed within the super-peer's range of clients. It may still be limited by the flooding mechanism used for communications across super-peers. Moreover, the super-peer approach makes a binary decision about a node's capacity. It is either a super-peer or it is not, that does not capture the full heterogeneity of the network. [40]

There is, however, the problem of selecting which nodes are eligible to become super-peers, as there is no global view of the system and each node's capabilities. There are two ways to address this problem: static and dynamic. [56] In static super-peer selection, once a peer is chosen as supernode by local configuration. Whereas a dynamic selection uses heuristics, such as, sufficient online duration, sufficient bandwidth, free of firewall, etc.

While super-peer networks are efficient in message dissemination, they become less tolerant to node failures. Should a super-peer fail, all its clients would have to find a replacement super-peer and this new super-peer would have to update its index to account for the new clients.

2.2.3.1 Gnutella 0.6

As described previously in Section 2.2.1.1, all nodes in the Gnutella [16] protocol were connected to each other at random and, while that worked for users with broadband connections, those with slow modems suffered, with the flooding of requests. That problem was approached by organizing the network hierarchically.

The latest version, Gnutella 0.6, uses the notion of super-peers, for nodes with better bandwidth connectivity, to help the routing performance of the network. [16, 40] It still uses flooding to disseminate messages across the network of super-peers as the overlay formed between them is **unstructured**.

Peer selection is still a problem as the only solution for this is for the client to choose if their node is able to be a super-peer or not, which does not bode well.

2.2.3.2 Overnesia

Overnesia [36] is a resilient overlay network protocol for virtual super-peers. It takes the lower fault tolerance challenge and builds a network of virtual super-peers, each composed by multiple physical nodes that act as replicas to achieve fault tolerance. Virtual super-peers form an **unstructured** overlay amongst each other, and the nodes within them form a fully connected graph with each other.

Nodes inside a virtual peer or cluster keep information about every other node in the cluster and its size. Clusters attempt to keep their size within a certain range, either by collapsing and migrating into other clusters, if they do not have enough nodes, or splitting and forming two clusters, if they surpass the maximum size.

Besides cluster size control, Overnesia promotes the creation of multiple, distinct, inter virtual super-peer connections. These make the overlay robust to node and link failures, reduce clustering among different virtual super-peers, and allow queries to be disseminated more efficiently.

Message dissemination can use flooding with the awareness of the virtual super-peers, meaning the clusters only respond to queries from other clusters, and if a node within the cluster receives a query, it avoids forwarding it to the cluster members. Again, as with most flooding techniques mentioned, this one makes use of the TTL value. Overnesia also works under virtual super-peer aware gossip, which can be improved by combining random walks, as described in [36].

2.2.3.3 Coral

Coral is "a latency-optimized hierarchical indexing infrastructure based on a novel abstraction called a Distributed Sloppy Hash Table (DSHT)." [15] It uses DSHTs to find nameservers within close topological range of the clients' networks, proxies keeping certain objects in cache, and close by nodes. A DSHT is a distributed table that allows for several values per key. This is used to map various keys to Coral nodes' addresses.

Much like the DHTs described in Section 2.2.2, the node's identifier is an x-bit long string. It is created by hashing the IP address of the node. Node proximity is defined by the XOR distance between the identifiers, similar to Kademlia [42].

Coral nodes belong to several overlays called clusters, one global and others divided by a maximum network Round-Trip-Time (RTT). Each of these clusters will form a level, and there is a hierarchy between levels. Coral nodes first join the global cluster and are only allowed to join a non-global cluster if their RTT value to a percentage of the nodes within said cluster is below a desired value.

These clusters have a layered hierarchy among them, lower layers allow for higher latencies than the ones above. This organization allows for a reduction of lookup latency, as nodes start by querying higher layers (physically closer nodes) for information, and then progress to lower layers, if those nodes do not provide the requested resource.

Nodes collect information from other nodes from normal lookups, since all messages contain RTT values, cluster membership and estimated size. Periodically, nodes change their cluster membership based on the collected information.

2.2.4 Discussion

The last sections have described the various categories of overlay networks, their advantages, and disadvantages, and reviewed some examples. Most semi-structured overlay networks, being hierarchical, only view node heterogeneity as a binary variable. They give good insights on how enforcing structure works over both structured and unstructured overlays. In structured overlays, the structure is based on the nodes' identifiers, which do not provide information about the nodes as they are often hashes of some property, e.g. IP or IP and Port.

Structured overlays are not the ideal solution to the problem described in Chapter 1, as the idea is to be able to have complex search, with logical operators and over several properties, and these usually only allow for efficient exact match search, since their routing is done based on identifiers.

Since unstructured networks often use flooding, gossip or random walks, they allow for keyword and arbitrary queries which are necessary for our work. It is also easy to add constraints to them and bias them to account for some property or target structure, as will be discussed in the next section.

2.3 Overlay Topology Management

In this section, we discuss more approaches for managing and leveraging the topology of overlay networks.

Most structured overlays do topology management by default, as they have to ensure the structure involving the node identifiers is kept. While there are implementations to

which more restrictions regarding the connections are added, e.g., [17], they are not often used. As discussed in Section 2.2.4, these are not the focus of this work.

Unstructured overlays, however, can be enhanced with some sort of management. Topology management can happen within the communication and membership protocol, or on a layer above, the service/application layer [33, 50].

2.3.1 Service Layer Management

The overlay layer will expose its links to the layer above, the service layer, which will provide feedback about the links' uses to the lower layer.

By [31]'s classification, we can either **embed** a more efficient topology over the overlay layer exploiting the underlying links, like a spanning tree, e.g, Plumtree [27], MON [37], or to **enrich** the underlying overlay, by adding more links between nodes maintained by the service protocol and picked according to the defined needs, [29, 26].

While having the management delegated to the Service layer allows for service specific optimizations and usage, it will need to have knowledge and concerns of the overlay, often mixing with the actual service. It may still suffer from overhead since the underlying layer does not have the concern to pick better links, and some approaches rely on the links defined by it.

2.3.2 Overlay Layer Management

"The fundamental idea behind topology management at the overlay layer, is to directly manipulate the protocol that builds and maintains the unstructured overlay, as to ensure that the neighbouring associations established among peers result in an overlay that owns a set of properties that benefit P2P services operating on top of them." [31] This management approach leaves the concern of the properties needed to the overlay, so that the service layer on top does not need to ensure any of them or have any knowledge. It works by giving preference to certain links that match the specific criteria, e.g., latency, bandwidth, or ensuring a topological indicator limit, e.g., shortest path, clustering coefficient.

For such, [31] divides the approaches into two strategies: Bias and Control. The Bias approach does not enforce constraints on the nodes' connections, their neighbouring relations. It works by improving the random overlay formed, by iteratively and continuously swapping links to improve defined criteria. Control adds constraints to the neighbouring relations, so that no link between two nodes violates said constraints. Unlike DHTs, these do not have to do solely with the node identifier. And these constraints are hard to predict a priori. [33]

It is hard to maintain the system's connectivity by following the Control approach. Recovery from churn also becomes harder, as the most immediate neighbouring relationships may be ones that violate the constraints. Bias overcomes churn fairly easily, but if done in a naïve fashion, it can affect the network connectivity.

2.3.2.1 Gia

Gia's [9] goal is to create a scalable Gnutella-like P2P system that can handle much higher average query rates, as when faced with that, nodes tend to quickly overload and the system's performance becomes unsatisfactory.

Gia exploits heterogeneity, not by resorting to the super-peer idea, but by adapting the topology according to the node's capacity. The protocol dynamically adapts so that most nodes are within short reach of high capacity nodes, following a **bias** strategy.

While Gnutella uses a flooding-based search method to find resources, Gia replaces that with biased random walks. As mentioned before, random walks are less straining to the network and nodes since the query is sent to a node at random from the neighbour set of the current node until the TTL value reaches zero. These are, however, blind searches, as the node to redirect the query to is not picked based on any information of how likely it is to contain the resource. If the resource is not popular, finding it may need several random walks with increasing TTL values. The random walks are biased to direct queries towards high capacity nodes, which are typically the best to answer them if the resource looked for is popular.

There is a flow control scheme to avoid overloaded hot-spots, in which nodes are assigned tokens based on available capacity. Since Gia uses random walks, dropping packets if the node is overloaded is not ideal, so a sender is only allowed to send direct queries to a neighbour if it has been notified that it could via token. A node periodically assigns tokens to its neighbours, adapting the token allocation rate as needed depending on the number of queries. The number of tokens each node has to give out is related to its capacity. Neighbour assignment is also based on capacity. High capacity nodes are given more nodes for their outgoing queries, and have more tokens to give out to their neighbours than lower capacity ones.

All nodes maintain pointers to the resources offered by their immediate neighbours. The topology adaptation algorithm makes high capacity nodes have more connections. The one-hop replication guarantees that high capacity nodes are capable of providing answers to a greater number of queries.

Gia is not driven by any specific node characteristic [36], it simply distinguishes between nodes that are more capable of processing and routing queries.

2.3.2.2 X-BOT

X-BOT is a "protocol to Bias the Overlay Topology according to some target efficiency criteria X, for instance, to better match the topology of the underlying network." [35] It focuses on affecting the topology of the overlay by using the **Bias** strategy to improve the overlay, doing so in a decentralized manner without a priori knowledge. The protocol is inspired by HyParView [32], which has been described in Section 2.2.1.3, in the sense that it keeps two sets of nodes.

The symmetric neighbouring relations are stored in a smaller set, the active view, which is kept by reactive strategy, only changed if there are membership changes (joins and departures/failures of nodes). If there is the need to add a new node to the neighbour set, it resorts to the other larger set, the passive view, which keeps a random selection of nodes and is updated periodically by exchanging a subset of all nodes known by the current node (from both sets) with some other node found using a random walk.

There is the assumption that X-BOT has access to a local *Oracle*, which basically represents a function that numerically ranks links (the connections between two nodes, that can be more than a single link) between two nodes according to some metric, i.e., link latency, ISP or IP proximity calculations.

While HyParView strives for stability, by not changing its active view, unless to react to a membership change, X-BOT will relax said stability by periodically attempting to switch a neighbour for a better node belonging to the passive view. It focuses on maintaining network connectivity, without optimizing links for nodes whose active view is not full. The passive view is never biased, and it tries to keep a certain number of unbiased nodes within the active view.

To maintain the connection degree of all nodes, the optimization exchanges two existing links for two others that maximize the preference oracle [30, 33], either minimizing cost or maximizing efficiency.

2.3.2.3 T-MAN

T-Man is an algorithm "for creating a large class of overlay networks from scratch. The algorithm is highly configurable: the network to be created is defined compactly by a ranking method that is able to order sets of nodes according to the preference of any given base node to select them as neighbours." [23] The goal of the protocol is to reach a given target topology from a purely random overlay, so it follows the **bias** approach.

Initially, nodes form random links between one another. Each node maintains a fixed size set of neighbours, that will be ranked according to some preference. This preference is defined by a *ranking function*, which given a set of nodes returns the set ordered in a relevant order. They periodically exchange a random subset of their neighbours' indicators and its own with other nodes, favouring nodes that maximize the *ranking function*. A constant attempt at improving the set of neighbours by using the ranking function to discard lower preference nodes for higher preference nodes.

Routing and resource lookup follow the gossip dissemination strategy, in which a node, upon receiving a request, will forward it to t neighbours, t being the fanout value.

This protocol is highly configurable, as the overlay created over a defined amount of iterations is representative of the ranking function used. It allows for a variety of topologies and structures, so long as their description can be transformed into a function with the desired properties defined in [23].

T-Man does not ensure the connectivity of the overlay network, nor does it level the connection degree of each node, which can lead to heavy node strain on nodes with a high degree, since all routing goes through them, which may not be ideal if those nodes do not have the capacity for it.

2.3.2.4 Bias Layered Tree

Bias Layered Tree "builds a robust hierarchical tree-based topology connecting the cloud infrastructure and edge devices, in a way that takes into account their level and relative network proximity" [13].

It is similar to HyParView [32] in the sense that it has two sets storing other nodes' information, one that stores connected neighbours and tends to be smaller and another which stores potential neighbours. The first set, the active view, is organized differently, keeping a single identifier for the father of the node, a limited set of nodes that share the same height as the current node and all its children nodes.

Bias Layered tree uses the commonality of IP prefixes as distance criteria to bias nodes to form connections with (physically) closer nodes. When a node joins the network, it will contact the entry point, which is the root of the tree at the lowest height, zero. The root, upon receiving a join request, will create a request for the tree's information and forward it to any node belonging to its active view, using a biased random walk. This walk is biased in the way that a node, upon receiving the request and providing information regarding the tree, will pick from its active view the next node with the highest proximity to the joining node. The request will reach nodes of higher height if its TTL value reaches zero or if all nodes belonging to the current tree height have been explored. This request will be used by nodes that process it to update their knowledge of the system. The joining node will eventually receive the forwarded request with all the information. It will pick the closest node in terms of IP commonality from the closest lower height (a node's height is defined locally).

In sum, the Bias Layered Tree uses a **control** approach to decide a node's parent, and a **bias** approach for the siblings as it periodically runs a procedure to exchange siblings if there are siblings closer to it.

2.3.2.5 Overnesia

This protocol has been described in Section 2.2.3.2. It uses a **control** management strategy to ensure that the virtual super-peers have a limited size of super-peers. It is important that the virtual super-peers are well-connected and within defined ranges for the communication and dissemination protocols to be efficient. It controls the size using three procedures: Join, Divide and Collapse.

Join Procedure: When a node wishes to join the system, it will send a request to a known contact node. This request is forwarded using a random walk limited by the request's

TTL value. If the request finds a cluster that has not yet reached its target size, the node joins said cluster. If no such thing happens and the TTL value reaches 0, the node is added to the cluster where the request ended, no matter its size.

Divide Procedure: If several concurrent joins happen and the cluster's size surpasses a certain upper limit value, the node with the lowest identifier within the cluster will notice the violation of the limit and start the splitting of the cluster into two.

Collapse Procedure: When a cluster's size reaches a lower limit value, the nodes within it will notice and start the process to join another cluster.

These are not tight constraints, but they control the formation of connections between nodes, and allow for the protocol to keep their virtual super-peers composed by a limited number of nodes for replication.

2.3.3 Discussion

Following the overlay topology management categories defined by [31], there are two possible layers in which management can happen, Service and Overlay. As described in Section 2.3.1, management at the service layer is not adequate for our solution, as it mingles the service logic with overlay management logic and, while it does allow for service specific optimization, it comes at the cost of requiring that layer to have knowledge of the underlying overlay layer. For those reasons, no examples of said management were explored.

The other layer, management within the overlay, rids us of having another level of indirection and the possible overheads from the fact that the service layer cannot affect the overlay layer in full to completely optimize it. This management can be divided into two approaches, control and bias. These approaches are relevant to the work to be reported, and so are the insights provided by the works described previously.

2.4 Summary

Distributed systems' architecture can be classified as centralized, decentralized and hybrid, according to the way in which the components that offer services are divided. Decentralized solutions are those in which services are offered by every node in the network, P2P systems are an example of such.

P2P systems are often created by building overlay networks. Overlay networks are logical networks built on top of existing networks. They are comprised of nodes and the logical links between them, and can often be thought of as graphs.

As mentioned previously, structured overlay topologies are defined by following hard constraints regarding the node identifiers, which negatively affects their flexibility but allows for search services to be efficiently provided, e.g., application-level routing. And unstructured overlay networks are defined at random and are highly flexible, easy to

reconfigure when churning occurs. However, they cannot be leveraged efficiently to provide additional support to services. These are easier to bias and control to optimize for some criteria than the other categories (as noted by the considerable amount of work done to bias and control unstructured overlays [35, 23, 13, 9, 36] compared to others [17] which, while existing, are not used). Semi-structured overlays attempt to combine both paradigms to gain their respective advantages to some extent, often making use of hierarchical organizations.

As for overlay network managing techniques, two of the presented are relevant, control and bias. Control involves adding constraints to the link formation so that only links that obey the desired properties are created between nodes. These properties can be link-specific like latency, bandwidth, or simply regard the nodes at both ends and their properties. Bias allows for the overlay to take shape and simply biases the links, inciting swaps to optimize the desired target.

3

Future Work

This chapter details the future work proposed by this document. Section 3.1 revisits the problem introduced in Chapter 1, further detailing it. In Section 3.2, we present the intuition behind the proposed solution, its various steps, the challenges to overcome and, how we are expecting to do so. Section 3.3 presents the evaluation plan for our solution, the metrics to measure, other works that will serve as baseline for comparison, use cases to be implemented and employed in the evaluation. Finally, in Section 3.4, we provide a division of the planned work into tasks and a scheduling for the future.

3.1 Problem Description

As described in Chapter 1, most overlay networks do not consider heterogeneity, or consider it as a binary variable. Nodes with similar resources, which can be either low level, i.e, CPU architecture, GPU, link latency, or high level such as applications being executed by the node, data stored, may be found to be distant in the network, so when querying for a specific resource that requires the cooperation of said nodes, the overhead might be large. The requestor node will have to issue several queries to the nodes that hold the resources for the desired task, and if cooperation is necessary, the nodes with said resources will have to communicate amongst each other, which creates overhead if there are many overlay hops between them. Ideally, if a client requests for computation to be distributed among X nodes with a certain type of CPU, for example, one request to one node that matched said description would be enough. That node could forward the request to other nodes also matching the description, which would most likely be its neighbours or in close range of the initial node within the overlay, and they would organize and communicate to distribute and complete the request.

Load distribution could also pose a problem. If the inherent heterogeneity of the network is not taken into consideration, distributing load equally may overload nodes with lower capacity. Ideally, nodes with higher capacity would receive higher loads.

3.2 Proposed Solution

The goal of this work is to develop an overlay network that can operate while taking into account the heterogeneity, be it related to capacity or content, of the nodes in the network. The idea is to group nodes by affinity over some common properties, which again can be either low level, i.e, CPU architecture, GPU, link latency, or high level such as applications being executed by the node, data stored.

In this section, we will further detail some design challenges of this work and the ways in which we plan to overcome them.

Influencing the overlay topology: There are many ways to manage the topology of an overlay. We will experiment with adding constraints to control the formation of neighbouring relationships or allowing for nodes to connect at random initially, but bias them to swap links and establish connections with nodes with higher similarity.

Maintaining the overlay's connectivity: While we intend to form groups of similar nodes in some defined property, if enforced too strictly, we could end up with several disconnected clusters of similar nodes. There needs to be leniency so that nodes that do not have much in common can still belong to the network and be connected to others, even if not as much as nodes in affinity groups. This requires testing and experimentation on the constraints applied, and on the chosen topology management strategy.

3.3 Evaluation

To be able to demonstrate the correct operation of the work to be developed, we need to test it and compare it with the following baselines: X-BOT [35], T-MAN [23], and Kademlia [42] with affinity based buckets.

We are going to conduct experimental comparison of our proposed solution with the state of the art, either by simulation or emulation, e.g., using PeerSim [44] or Kollaps [41], or by using a novel emulation system under development at NOVA LINCS. Preferably, we will use emulation, because running real code on an emulated environment provides more accurate results for real life implementations.

For the evaluation and testing of the solution, we describe the following use cases:

Application/Service Set: A node can run several applications and services. This use case bases affinity on the commonality of applications and/or services, e.g, publish-subscribe service, replication, DHT, running in each node [43]. This set is fairly static, as the node's services do not change often within its lifetime in the network. This use case is used to test and evaluate the initial solution because it is easier to implement than the others.

File-sharing: Distributed file-sharing is a common use of overlay networks. In this case, the affinity could be based on the popularity of the files or on the most searched files of each node. This use case serves specifically to compare our solution with Kademlia [42], and its affinity bucket based implementation.

Distributed Learning: This use case is based on computational sharing, the resource shared is static, like the CPU architecture, the presence of a GPU, link bandwidth, specialized hardware, e.g, Intel SGX. Affinity will be based on the set similarity of the properties. The main idea is to use these to be able to do distributed learning over a subset of nodes that match a certain requisite.

Publish-Subscribe: In this use case, we run a publish-subscribe system over our solution, and the affinity is going to be based on the commonality of topics shared between nodes. This is a challenging and relevant use case since the topics are dynamic (nodes can subscribe and unsubscribe at will) and the dissemination times of messages may be affected by the overlay topology.

The general evaluation will be based on:

Convergence Time: The time it takes the overlay to stabilize with no changes to the number of nodes within it, and how long it takes to stabilize post-churn.

Cost: The number of messages needed for the protocol to work.

Graph properties: These include the in-degree and out-degrees of each node, the clustering coefficient and the network's connectivity.

For each use case, there are additional metrics:

Application/Service Set: Each individual service's metrics will be considered along with the additional effort of each node, when it has to route messages that do not belong to any protocol it uses.

File-sharing: The number of resources that are returned by the system post given query, also called recall [31].

Distributed Learning: The hops/time it takes to find a node matching the necessary requisites, the nodes to coordinate to complete the task given to them, and the addition of both, for the overall time/hops it takes to complete a request.

Publish-Subscribe: Dissemination time per topic, which is the average time it takes a message to reach all the subscribers, after it has been published by a publisher regarding a topic.

The evaluation of the proposed work is divided into three phases. One, after the initial implementation of the solution, to compare it with the previously mentioned baselines,

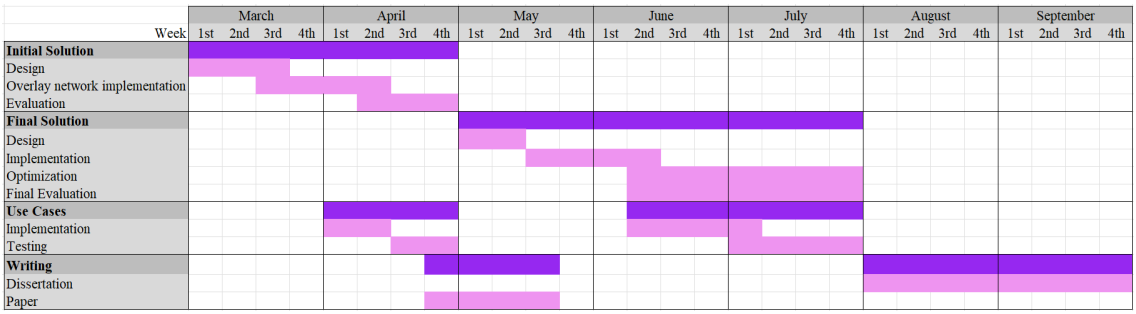

Figure 3.1: Gantt chart of the proposed work schedule

by using the use case that uses applications/services ran in each node as the properties for affinity.

The second and third happen at the same time. After the full proposal has been implemented, it will again be compared to the baselines and all the use cases defined will be tested over both the solution and the baselines.

3.4 Schedule

Figure 3.1 presents the expected work schedule divided into four sections.

Initial solution: It lasts eight weeks and consists of the design, implementation and simple evaluation of the initial solution.

Final Solution: Scheduled to occupy 12 weeks, consists of the design, implementation, and evaluation of a newly improved solution, along with the final testing.

Use Cases: Overlaps both the Initial and the Final Solution phase, it regards the implementation of specific use cases so that there is a better grasp of the performance of the solution.

Writing: Comprises the last eight weeks of the calendar for the writing of the dissertation and four weeks starting at the end of April for the elaboration of a paper to be submitted at a conference.

Bibliography

[1] K. Aberer et al. "The essence of P2P: a reference architecture for overlay networks". In: *Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*. 2005, pp. 11–20. DOI: 10.1109/P2P.2005.38 (cit. on pp. 1, 2, 7, 8, 10, 11).

[2] H. Balakrishnan et al. "Looking up data in P2P systems". In: *Communications of the ACM* (2003-02) (cit. on p. 17).

[3] S. A. Baset and H. G. Schulzrinne. "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol". In: *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*. 2006, pp. 1–11. DOI: 10.1109/INFOCOM.2006.312 (cit. on p. 10).

[4] J. Benet. "IPFS - Content Addressed, Versioned, P2P File System". In: *CoRR* abs/1407.3561 (2014). arXiv: 1407.3561. URL: http://arxiv.org/abs/1407.3561 (cit. on pp. 1, 9).

[5] F. Bordignon and G. Tolosa. "Gnutella: Distributed System for Information Storage and Searching Model Description". In: (2001-08) (cit. on p. 14).

[6] J. Buford, H. Yu, and E. Lua. *P2P Networking and Applications*. 2009-01. DOI: 10.1016/B978-0-12-374214-8.X0001-3 (cit. on pp. 1, 2, 13, 17).

[7] J. F. Buford and H. Yu. "Peer-to-Peer Networking and Applications: Synopsis and Research Directions". In: *Handbook of Peer-to-Peer Networking*. Ed. by X. Shen et al. Boston, MA: Springer US, 2010, pp. 3–45. ISBN: 978-0-387-09751-0. DOI: 10.1007/978-0-387-09751-0_1. URL: https://doi.org/10.1007/978-0-387-09751-0_1 (cit. on pp. 2, 17).

[8] B. Carlsson and R. Gustavsson. "The Rise and Fall of Napster - An Evolutionary Approach". In: *Active Media Technology*. 2001 (cit. on pp. 1, 8, 12).

[9] Y. Chawathe et al. "Making Gnutella-like P2P Systems Scalable". In: *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '03. Karlsruhe, Germany: Association for Computing Machinery, 2003, pp. 407–418. ISBN: 1581137354. DOI: 10.1145/863955.864000. URL: https://doi.org/10.1145/863955.864000 (cit. on pp. 2, 18, 25, 29).

[10] I. Clarke et al. "Freenet: A Distributed Anonymous Information Storage and Retrieval System". In: *Lecture Notes in Computer Science* 2009 (2001-03). DOI: 10.1007/3-540-44702-4_4 (cit. on pp. 7, 12, 14, 15).

[11] B. Cohen. "Incentives build robustness in BitTorrent". In: *Workshop on Economics of PeertoPeer systems* 6 (2003-06) (cit. on pp. 1, 9, 12).

[12] P. Costa, J. Leitão, and Y. Psaras. "Studying the workload of a fully decentralized Web3 system: IPFS". In: (2022-12). DOI: 10.48550/arXiv.2212.07375 (cit. on p. 19).

[13] P. Á. Costa, P. Fouto, and J. Leitão. "Overlay Networks for Edge Management". In: *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)*. 2020, pp. 1–10. DOI: 10.1109/NCA51143.2020.9306716 (cit. on pp. 27, 29).

[14] J. Dowling and A. H. Payberah. "Shuffling with a Croupier: Nat-Aware Peer-Sampling". In: *2012 IEEE 32nd International Conference on Distributed Computing Systems*. 2012, pp. 102–111. DOI: 10.1109/ICDCS.2012.19 (cit. on p. 11).

[15] M. J. Freedman, E. Freudenthal, and D. Mazières. "Democratizing Content Publication with Coral". In: *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*. NSDI'04. San Francisco, California: USENIX Association, 2004, p. 18 (cit. on pp. 20, 22).

[16] *Gnutella - A protocol for Revolution*. URL: https://rfc-gnutella.sourceforge.net/ (cit. on pp. 1, 7, 9, 12, 14, 22).

[17] *go-libp2p-kad-dht*. URL: https://pkg.go.dev/github.com/libp2p/go-libp2p-kad-dht#RoutingTableFilter (visited on 2023-02-04) (cit. on pp. 19, 24, 29).

[18] C. Gross et al. "Geodemlia: A robust peer-to-peer overlay supporting location-based search". In: *2012 IEEE 12th International Conference on Peer-to-Peer Computing (P2P)*. 2012, pp. 25–36. DOI: 10.1109/P2P.2012.6335806 (cit. on p. 20).

[19] C. Guan, D. Ding, and J. Guo. "Web3.0: A Review And Research Agenda". In: *2022 RIVF International Conference on Computing and Communication Technologies (RIVF)*. 2022, pp. 653–658. DOI: 10.1109/RIVF55975.2022.10013794 (cit. on p. 1).

[20] Y. Hassanzadeh Nazarabadi, A. Küpçü, and O. Ozkasap. "Decentralized and locality aware replication method for DHT-based P2P storage systems". In: *Future Generation Computer Systems* 84 (2018-02). DOI: 10.1016/j.future.2018.02.007 (cit. on p. 20).

[21] J. M. Hellerstein. "Toward Network Data Independence". In: *SIGMOD Rec.* 32.3 (2003-09), pp. 34–40. ISSN: 0163-5808. DOI: 10.1145/945721.945730. URL: https://doi.org/10.1145/945721.945730 (cit. on p. 17).

[22] H.-C. Hsiao, H. Liao, and C.-C. Huang. "Resolving the Topology Mismatch Problem in Unstructured Peer-to-Peer Networks". In: *IEEE Transactions on Parallel and Distributed Systems* 20.11 (2009), pp. 1668–1681. DOI: 10.1109/TPDS.2009.24 (cit. on p. 14).

[23] M. Jelasity, A. Montresor, and O. Babaoglu. "T-Man: Gossip-Based Fast Overlay Topology Construction". In: *Computer Networks* 53 (2009-04), pp. 2321–2339. DOI: 10.1016/j.comnet.2009.03.013 (cit. on pp. 11, 26, 29, 32).

[24] X. Jin and S.-H. G. Chan. "Unstructured Peer-to-Peer Network Architectures". In: *Handbook of Peer-to-Peer Networking*. Ed. by X. Shen et al. Boston, MA: Springer US, 2010, pp. 117–142. ISBN: 978-0-387-09751-0. DOI: 10.1007/978-0-387-09751-0_5. URL: https://doi.org/10.1007/978-0-387-09751-0_5 (cit. on pp. 9, 14).

[25] G. Kan. "Gnutella". In: *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. Ed. by A. Oram. 2001, pp. 62–79 (cit. on p. 14).

[26] A.-M. Kermarrec et al. "NAT-resilient Gossip Peer Sampling". In: 2009-06, pp. 360–367. DOI: 10.1109/ICDCS.2009.44 (cit. on p. 24).

[27] J. Leitão, J. Pereira, and L. Rodrigues. "Epidemic Broadcast Trees". In: *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems*. Beijing, China, 2007-09, pp. 301–310 (cit. on p. 24).

[28] J. Leitão, R. van Renesse, and L. Rodrigues. "Balancing Gossip Exchanges in Networks with Firewalls". In: *Proceedings of the 9th International Workshop on Peer-to-Peer Systems (IPTPS '10)*. San Jose, CA, U.S.A., 2010-04 (cit. on p. 11).

[29] J. Leitão, R. van Renesse, and L. Rodrigues. "Balancing Gossip Exchanges in Networks with Firewalls". In: *Proceedings of the 9th International Workshop on Peer-to-Peer Systems (IPTPS '10)*. San Jose, CA, U.S.A., 2010-04, (to appear) (cit. on p. 24).

[30] J. Leitão et al. "X-BOT: A Protocol for Resilient Optimization of Unstructured Overlays". In: *Proceedings of the 28th IEEE International Symposium on Reliable Distributed Systems*. Niagara Falls, New York, U.S.A., 2009-09, pp. 236–245 (cit. on p. 26).

[31] J. Leitão. "Topology management for unstructured overlay networks". In: *Technical University of Lisbon* (2012) (cit. on pp. 12, 13, 24, 28, 33).

[32] J. Leitão, J. Pereira, and L. Rodrigues. "HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast". In: 2007-07, pp. 419–429. ISBN: 0-7695-2855-4. DOI: 10.1109/DSN.2007.56 (cit. on pp. 16, 25, 27).

[33] J. Leitão et al. "On Adding Structure to Unstructured Overlay Networks". In: *Handbook of Peer-to-Peer Networking*. Ed. by X. Shen et al. Boston, MA: Springer US, 2010, pp. 327–365. ISBN: 978-0-387-09751-0. DOI: 10.1007/978-0-387-09751-0_13. URL: https://doi.org/10.1007/978-0-387-09751-0_13 (cit. on pp. 24, 26).

[34] J. Leitão et al. "Towards Enabling Novel Edge-Enabled Applications". In: *CoRR* abs/1805.06989 (2018). arXiv: 1805.06989. URL: http://arxiv.org/abs/1805.06989 (cit. on p. 1).

[35] J. Leitão et al. "X-BOT: A Protocol for Resilient Optimization of Unstructured Overlay Networks". In: *IEEE Transactions on Parallel and Distributed Systems* 99.PrePrints (2012). ISSN: 1045-9219. DOI: http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.29 (cit. on pp. 11, 25, 29, 32).

[36] J. C. A. Leitão and L. E. T. Rodrigues. "Overnesia: A Resilient Overlay Network for Virtual Super-Peers". In: *2014 IEEE 33rd International Symposium on Reliable Distributed Systems*. 2014, pp. 281–290. DOI: 10.1109/SRDS.2014.40 (cit. on pp. 22, 25, 29).

[37] J. Liang et al. "MON: On-demand Overlays for Distributed System Management". In: (2005-12) (cit. on p. 24).

[38] *Libp2p - IPFS Docs*. URL: https://docs.ipfs.tech/concepts/libp2p/ (visited on 2023-02-04) (cit. on p. 19).

[39] *libp2p: A Modular, P2P, Networking Library*. URL: https://libp2p.io/ (visited on 2023-02-04) (cit. on p. 19).

[40] E. K. Lua et al. "A survey and comparison of peer-to-peer overlay network schemes". In: *IEEE Communications Surveys & Tutorials* 7.2 (2005), pp. 72–93. DOI: 10.1109/COMST.2005.1610546 (cit. on pp. 14, 21, 22).

[41] M. Matos. "Kollaps/Thunderstorm: Reproducible Evaluation of Distributed Systems". In: *Distributed Applications and Interoperable Systems*. Ed. by A. Remke and V. Schiavoni. Cham: Springer International Publishing, 2020, pp. 121–128. ISBN: 978-3-030-50323-9 (cit. on p. 32).

[42] P. Maymounkov and D. Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". In: *Peer-to-Peer Systems*. Ed. by P. Druschel, F. Kaashoek, and A. Rowstron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 53–65. ISBN: 978-3-540-45748-0 (cit. on pp. 19, 23, 32, 33).

[43] J. Monteiro et al. "Enriching Kademlia by Partitioning". In: *Proceedings of the 1st Workshop on Decentralized Internet, Networks, Protocols, and Systems (DINPS'22)* (2022-07) (cit. on pp. 20, 32).

[44] A. Montresor and M. Jelasity. "PeerSim: A Scalable P2P Simulator". In: *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*. Seattle, WA, 2009-09, pp. 99–100 (cit. on p. 32).

[45] L. L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach*. 6th. 2019. ISBN: 0128103515. URL: https://book.systemsapproach.org (cit. on p. 10).

[46] M. Ripeanu. "Peer-to-Peer Architecture Case Study: Gnutella Network". In: (2002-03). DOI: 10.1109/P2P.2001.990433 (cit. on p. 14).

[47] A. Rowstron. "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems". In: vol. 2218. 2002-10. ISBN: 978-3-540-42800-8. DOI: 10.1007/3-540-45518-3_18 (cit. on p. 7).

[48] J. H. Saltzer and M. F. Kaashoek. *Principles of Computer System Design: An Introduction*. San Francisco: Morgan Kaufmann, 2009. ISBN: 978-0-12-374957-4. DOI: https://doi.org/10.1016/B978-0-12-374957-4.00003-7 (cit. on p. 5).

[49] *Skype*. URL: https://www.skype.com/ (cit. on p. 10).

[50] M. van Steen and A. Tanenbaum. *Distributed Systems: 4th edition*. 2023. ISBN: 978-90-815406-4-3. URL: https://distributed-systems.net (cit. on pp. 1, 5–7, 10, 13, 17, 20, 24).

[51] I. Stoica et al. "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications". In: *IEEE/ACM Trans. Netw.* 11.1 (2003-02), pp. 17–32. ISSN: 1063-6692. DOI: 10.1109/TNET.2002.808407. URL: https://doi.org/10.1109/TNET.2002.808407 (cit. on p. 18).

[52] *TaRDIS: Trustworthy and Resilient Decentralised Intelligence for Edge Systems*. URL: https://cordis.europa.eu/project/id/101093006 (visited on 2023-01-26) (cit. on p. 3).

[53] S. Tarkoma. *Overlay Networks - Toward Information Networking*. 2010-02. ISBN: 978-1-439-81371-3. DOI: 10.1201/9781439813737 (cit. on pp. 2, 10, 11, 13, 16–18).

[54] *The Bacalhau Project*. URL: https://docs.bacalhau.org/ (visited on 2023-02-07) (cit. on p. 20).

[55] M. Touloupou et al. "Towards a Framework for Understanding the Performance of Blockchains". In: *2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. 2021, pp. 47–48. DOI: 10.1109/BRAINS52497.2021.9569810 (cit. on p. 1).

[56] Q. Vu, M. Lupu, and B. Ooi. *Peer-to-Peer Computing - Principles and Applications*. 2010-01. ISBN: 978-3-642-03513-5. DOI: 10.1007/978-3-642-03514-2 (cit. on pp. 7–10, 12, 14, 21).

[57] *What is seti@home?* URL: https://setiathome.ssl.berkeley.edu/ (cit. on p. 9).

[58] Y. Xiao et al. "A Survey of Distributed Consensus Protocols for Blockchain Networks". In: *IEEE Communications Surveys & Tutorials* 22.2 (2020), pp. 1432–1465. DOI: 10.1109/COMST.2020.2969706 (cit. on p. 1).