Vítor Hugo Menino

Bachelor's Degree in Computer Science and Engineering

A Novel Approach to Load Balancing in P2P Overlay Networks for Edge Systems

Dissertation plan submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science

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

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

February, 2021

The work reported in this document was partially supported by
Fundação para a Ciência e Tecnologia through the Project NG-STORAGE (PTDC/CCI-INF/32038/2017)

Abstract

Edge computing aims at addressing some limitations of cloud computing by bringing computation towards the edge of the system, i.e., closer to the client. There is a panoply of devices that can be integrated into future edge computing platforms, from local data-centers and ISP points of presence to 5G towers and even multiple user devices like smartphones, laptops, and IoT devices. For all of these devices to communicate fruitfully, we need to build systems that enable the seamless interaction and cooperation among these diverse devices. However, creating and maintaining these systems is not trivial since there are numerous types of devices with different capacities. This resource heterogeneity has to be taken into account so that different types of machines contribute to the management of the distributed infrastructure differently, and the communications within the network are more efficient.

In this work, we plan to address the challenges identified above by exploring unstructured overlay networks, that have been shown to be possible to manage efficiently and in a fully decentralized way, while being highly robust to failures. To that end, we plan to devise a novel overlay, where the contribution of each device to maintain the system is translated to maintaining a specific number of overlay neighbors, being this number dynamically computed and adjusted—influenced on the local perception of each node regarding the average device capacity. To show the benefits of our proposal, we will devise a set of support subsystems that pave the way to build the next generation of edge-enabled systems. We will evaluate our proposal by comparing it with state of the art decentralized solutions by combining both simulation (to observe the performance of the solution in huge scale) and prototype deployments in realistic distributed infrastructures.

Keywords: peer-to-peer systems, edge computing, unstructured overlay networks, gossip protocols

Resumo

A computação de periferia visa abordar algumas limitações da computação em nuvem, trazendo a computação para mais perto do cliente. Há uma enorme variedade de dispositivos que podem ser integrados em futuras plataformas de computação de periferia, de data centers locais e pontos de presença de ISPs a torres 5G e até mesmo dispositivos de cliente, como smartphones, laptops e dispositivos IoT. Para que todos esses dispositivos comuniquem de forma proveitosa entre si, precisamos construir sistemas que possibilitem a interação e cooperação eficaz entre eles. No entanto, criar e manter esses sistemas não é trivial, uma vez que existem vários tipos de dispositivos com diferentes capacidades. Essa heterogeneidade de recursos deve ser levada em consideração para que diferentes tipos de máquinas contribuam para o gerenciamento da infraestrutura distribuída de forma distinta e as comunicações dentro da rede sejam mais eficientes.

Neste trabalho, pretendemos enfrentar os desafios identificados acima, explorando redes sobrepostas não estruturadas, que se têm mostrado possíveis de gerenciar de forma eficiente e totalmente descentralizada, sendo altamente resistentes a falhas. Para tal, planeamos conceber uma nova rede sobreposta, onde a contribuição de cada dispositivo para manter o sistema se traduz na manutenção de um número específico de vizinhos na rede, sendo este número calculado e ajustado dinamicamente – influenciado na percepção local de cada nó em relação à capacidade média dos dispositivos na rede. Para mostrar os benefícios de nossa proposta, iremos desenvolver um conjunto de subsistemas de suporte que abrem caminho para construir a próxima geração de sistemas para a computação de periferia. Avaliaremos nossa proposta comparando-a com soluções descentralizadas de última geração, combinando simulação (para observar o desempenho da solução em grande escala) e emulação de protótipos em infraestruturas distribuídas realistas.

Palavras-chave: sistemas entre-pares, computação de periferia, redes sobrepostas não estruturadas, protocolos de rumor

Contents

List of Figures

Chapter 1

Introduction

Nowadays, doing all computation on the cloud, i.e., inside data-centers distant from the end-user, is becoming insufficient mainly due to the high-latency access to these infrastructures [22]. Edge computing [33] aims at addressing this and other limitations of cloud computing by bringing computation closer to the end clients.

There is a panoply of devices that can be integrated into future edge computing platforms, from local data-centers and ISP points of presence to 5G towers and even multiple user devices like smartphones, laptops and IoT devices [22]. For all of these devices to communicate fruitfully, we need to build systems that enable the seamless interaction and cooperation among these diverse devices. However, creating and maintaining these systems is not trivial since there are numerous types of devices with different capacities. This resource heterogeneity has to be taken into account so that different types of machines contribute to the management of the distributed infrastructure differently, and the communications within the network are more efficient [22].

In this work, we plan to address this heterogeneity challenge by exploring unstructured overlay networks, that have been shown to be possible to manage efficiently and in a fully decentralized way, while being highly robust to failures. However, most of the work that has been done in this area deals with homogeneous networks, i.e., networks composed of nodes whose resources are mostly similar. Nonetheless, the system to manage devices with different capacities has to take into account their heterogeneous nature, therefore, it should be conceived thinking about heterogeneity.

1.1 Objective

In this dissertation, we plan to address device resource heterogeneity in edge computing. More specifically, we plan to address this problem by dynamically compute and adjust

the number of neighbors of each device at the overlay network layer, according to its local perception regarding its own capacity and the average capacity of devices in the system. We plan to do this at the overlay layer since this is a component essential to the effective tracking of membership in large scale systems that highly affects decentralized interactions.

Doing this arises some challenges. One of them is the challenge to keep the overlay proprieties sound when changing the number of neighbors of each device. For example, we have to make sure that the network does not become partitioned when we adjust the number of neighbors of the nodes. Another challenge that we face is the difficulty to infer the network average resources given that each node does not have direct access (i.e., is not directly connected) to every other node. In addition, we also face the challenge of devising an efficient support subsystem that catalyzes and illustrates the benefits of this approach when compared with current state of the art.

1.2 Contributions

The main contributions expected from this work are three-fold:

1. The design and implementation of a novel overlay, where the contribution of each device to maintain the system is translated to maintaining a different number of overlay neighbors, being this number dynamically computed and adjusted, influenced by the local perception of each node regarding the average device capacity.

2. The design and implementation of a set of support subsystems that pave the way to build the next generation of the edge-enabled systems, to show the benefits of our proposed overlay.

3. An experimental comparison of our proposal with state of the art decentralized solutions by combining both simulation (to observe the performance of the solution in huge scale) and prototype deployments in realistic distributed infrastructures.

1.2.1 Research Context

This dissertation is done under the context of a research project conducted by NOVA School of Science and Technology in association with Protocol Labs [16]. Protocol Labs is a company that focuses on the research, development and deployment of decentralized distributed systems. Some of their work includes libp2p [15], Filecoin [14] and IPFS [1].

Part of the aforementioned research project consists on the design of a novel multi-level DHT. It is in this area that the contributions emerged by this dissertation can provide value. By using an unstructured overlay optimized for heterogeneity, we can enable the creation of cliques of heterogeneous nodes in close proximity that can share load and replicate data while being minimally susceptible to failures and churn, due to the unstructured nature of the overlay that connects them.

1.3 Document Structure

The rest of the document is organized as follows:

- Chapter 2 starts by presenting the relevant concepts that lay ground for the elaboration of the dissertation. More specifically, it explores the concepts of peer-to-peer, broadcast, overlays and their proprieties. In further sections of that chapter, we use these previously explained concepts to delve deeper into the problem that we are trying to solve and present the currently available solutions, exploring their advantages, disadvantages and how they can contribute to our work. Although numerous solutions were found, none address the problem in the way that we aim to—by dynamically adjusting the number neighbors of each node.
- Chapter 3 details the future work to be conducted, the strategy to achieve its objectives, the challenges that need to be addressed, and the proposed scheduling to accomplish it.

Chapter 2

Related Work

In this chapter, we are delving into the relevant concepts that lay ground for the elaboration of this dissertation, while presenting and analyzing related work. The chapter is structured as following: in Section 2.1 we introduce peer-to-peer; in Section 2.2 we present services that can be implemented on top of peer-to-peer; in Section 2.3 we delve into broadcast, its problems and performance metrics; in Section 2.4 we introduce the concept of overlays, the different architectures, its proprieties and protocols that implement them; in Section 2.5 we use the concepts explained in previous sections to explore how the heterogeneity problem can be tackled; finally, in Section 2.6 we reflect on the contributions that the solutions previously presented can provide to our work.

2.1 Peer-to-Peer

There are two main architectures for distributed computer systems: centralized and decentralized. Both architectures have advantages and disadvantages and their usefulness depends on the use case of the system.

Centralized distributed systems rely on a single logical component (usually named server) that has a special role in the system and that provides services to the other nodes (i.e., computers, usually called clients in this context) of the system. In such architectures, clients usually only interact with the server and never directly among them. Usually, these systems manage to achieve great performance, since its nodes delegate all coordination tasks (e.g., deciding which node is the current leader) to the single central server. On the other hand, these systems are highly dependent on the availability of the coordinator, i.e., if the central server fails, the whole system stops—which is a big disadvantage of centralized systems. In addition, centralized systems have scalability problems—if the network is large, it might be impossible for the central server to manage every client in

the network (i.e., every other node), since they can be numerous, distant from the central node, and joining and leaving the network on a very frequent basis. These limitations led to the appearance of systems implementing a different and decentralized architecture, i.e., decentralized systems.

Decentralized systems, on the other hand, do not rely on a single component that has a special role on the operation of the whole system. In such architectures, nodes communicate between themselves directly to execute the system logic: all information exchange and coordination is performed directly among the participants of the system. The biggest advantage of decentralized systems is that there is no single point of failure, which means that numerous nodes could fail and the system would still continue to operate. In addition, they are more scalable and there is no single component that is in total control of the system. However, decentralized systems often exhibit poorer overall performance than centralized systems. A popular example of decentralized systems are peer-to-peer systems [32], in which clients communicate between themselves and there is no central server.

One might ask—if centralized systems are theoretically faster, then why use peer-to-peer decentralized systems? As mentioned above, decentralized systems are usually more available (i.e., reliable against failures), more scalable, and make shared control over the system possible. There are a lot of cases where these features might be important—specially the last one. For example, a distributed system that is composed by replicas from many different banks should not have a centralized replica that coordinates it, because that would mean setting one bank as a coordinator of the system (i.e., control all the other banks), and if the coordinator bank would fail, the whole system would halt.

2.1.1 Peer-to-Peer Networks and Edge Computing

Peer-to-peer networks are regaining some popularity due to the emergence of edge computing [33]. Edge computing encompasses every computation done outside the cloud, i.e., outside the logical central computational unit. The rationale behind edge computing is moving the computation closer to the client so that, instead of all data generated by the clients being processed in the cloud (data-centers), some of that data can be processed on intermediary hardware (possibly including the client's device), in order to provide better latency, reliability and scalability [22], while avoiding saturating network links connecting clients to the cloud data-centers.

In the last years, two particular models emerged within the edge computing realm that have received significant attention—fog [25] and mist computing [10]. They are both seen as more specific instances of edge computing [22]. Fog computing aims to improve the overall performance of IoT applications by installing servers physically close to sensors that generate large amounts of data. These servers can preprocess data before sending it to the cloud data-centers, resulting in less load on these and on the network infrastructure in general. Beyond that, the fog servers can also reduce the latency observed

by the clients by doing time-sensible work and quickly replying to the client with any relevant time-sensible information (e.g., an alarm). The second model, mist computing, is an evolution of fog computing, where not only servers are installed nearby the clients and used for computation, but the clients themselves are also used to perform data-filtering computational work. Using the clients' hardware to do some computational work alleviates the load imposed on edge and cloud servers (and on the network), resulting in a better overall system performance and resource usage.

Peer-to-peer networks can be useful in edge computing in different ways. Firstly, because there are a lot of different nodes (i.e., computers) in the edge environment, managing them centrally becomes a daunting task, which is hard to perform in a timely fashion [4]. It is difficult mainly because in a centralized management system, the central component would be saturated with messages from all the nodes in the system. In addition to that, it is difficult because the network that encompasses those nodes can be wide, i.e., we can have nodes in remote locations, which makes it infeasible to timely manage changes in the configuration of the system. Using peer-to-peer networks to implement decentralized control solves these problems because the nodes usually only communicate with other nodes that are in their virtual vicinity [20], thereby not saturating a coordinator nor presenting unnecessary delay in the management communications.

One might suggest that for not saturating the coordinator we could have different hierarchies of coordination. This also constitutes a centralized architecture, since there is also one top-level coordinator that manages the whole network through the other coordinators [38]. However, such a solution does mitigate both problems mentioned above—node saturation and communication delay. Nonetheless, even such an optimized centralized solution ends up suffering other limitations. One of them is regarding the reliability—with a centralized system, we have a single point of failure (the top coordinator). This issue is avoided in most peer-to-peer decentralized systems because network management is a responsibility shared among the peers (i.e., nodes). The other problem is that in edge computing, nodes are usually not owned by the same entity—we can have nodes owned by different companies and individuals. That said, it is not wise to let a single coordinator (from one of those companies or individuals) manage the entire network, because that would involve having to trust that node for the whole system to keep operating.

2.1.2 Peer-to-Peer Architecture

When it comes to the system architecture at a typical peer-to-peer system, each of the nodes runs a stack of (usually) three protocols (which can be perceived as layers in their abstract peer-to-peer architecture): membership layer, service layer and application layer [20]. The layers within each node communicate with each other (to support the execution of the system logic) locally, and communicate with the equivalent layer of other nodes using message passing mechanisms that are sent through the network. Figure 2.1 illustrates the presented abstract peer-to-peer architecture, which is explained in detail

below.

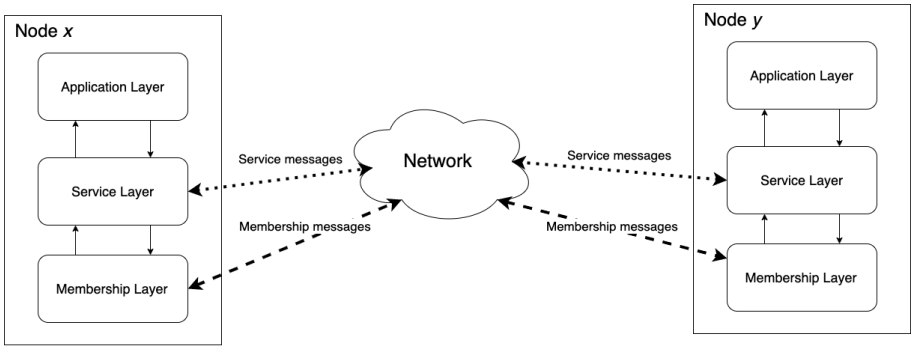


Figure 2.1: Peer-to-peer node layered architecture.

The membership layer is responsible for maintaining the neighbors of each node, i.e, the other nodes in the system that each node knows about and with whom it exchanges information directly. This layer communicates with other nodes' membership layers in order to maintain each node's local and partial view of the system membership (i.e., which neighbors each node has) [20]. This layer is going to be analyzed in depth on Section 2.4.

The service layer is responsible for providing higher level services to the application layer taking advantage of the membership layer. In other words, this layer uses the local view of the system provided by the membership layer to send messages to the node's neighbors in order to perform some function for the application layer (e.g., broadcast a message or identify some particular node) [20]. This layer is going to be analyzed in depth on Section 2.2.

The application layer uses the service layer (and potentially the membership layer) to implement a distributed application. It is in this layer that the application logic resides [20]. For example, if we were to implement a distributed file system where each node hosts part of the contents of the file system, this layer would be responsible for the application logic which involves exposing the operations over files and directories but not operations such as data/message dissemination (because that is typically implemented at the service layer) and membership management (because that is a responsibility of the membership layer).

2.2 Peer-to-Peer Services

There are numerous services that can be implemented in the service layer. As mentioned above, these services provide functionalities for the application layer and usually involve the exchange of data (in the form of messages) between the nodes.

One of those services is streaming. In this type of service, a flow of data is sent by one node through the other nodes in order for the multiple receivers to be able to receive that flow of data and feed the application layer with it. Another common service is shared storage, which aims at sharing (e.g., splitting) data between the nodes for the

client application to enable sharing the access to files by multiple parties. One example of an application that can be built on top of it is a shared file system [37], as mentioned above.

The service layer is also used to implement routing solutions. These solutions aim at discovering and providing efficient and up-to-date routing paths for the messages to be sent through, among individual parties. This enables point-to-point exchange of information at the application layer.

Lastly, the service layer can implement a broadcast solution, which aims at disseminating messages across all nodes of the network. We are going to focus on broadcast in Section 2.3.

One of the building blocks that is used to implement many of the services mentioned above is gossip [20]. Gossip is a class of decentralized protocols that are many times employed to disseminate information through nodes with a confirmable probabilistic assurance. They are also employed to synchronize replicas. Due to their probabilistic essence, gossip protocols usually rely on parameters that enable the control over their probabilistic guarantees and cost.

In a nutshell, gossip works as follows. When a process (i.e., the software that a node runs) wants to disseminate a message, it picks t (the fanout) random nodes from its neighbors and sends the message to those nodes. When a process receives a message for the first time, it simply repeats the process (possibly avoiding sending the message back to the sender) [20].

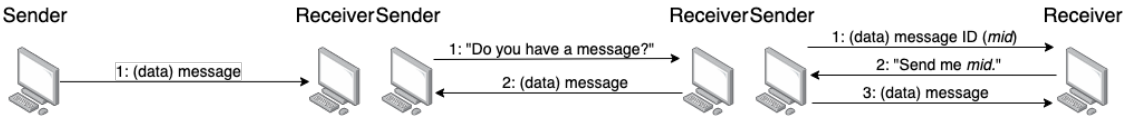
In their most simple form, gossip algorithms have two parameters: the fanout and the maximum rounds [21]. The fanout [21] consists on the number of neighbors that each neighbor disseminates a message to. The fanout has an impact on the probability that every node receives the message. For this probability to be high, the fanout should be $t >= ln(n)$, t being the fanout and n being the total number of nodes in the network [7]. That said, the higher the fanout, the more likely it is that all nodes receive (and then deliver to the application layer) the message being disseminated. On the other hand, a higher fanout results in more traffic on the network. Thus, the chosen fanout for the gossip implementation depends on the goals for the system, i.e., what is most important—high probability of total dissemination (i.e., all nodes receiving the disseminated message) or lowering the bandwidth consumption.

The maximum rounds parameter corresponds to the maximum number of times that a message can be retransmitted by nodes [21]. One way to control this is by using a round-number that is carried by messages. When a message is first transmitted (from the original sender), it carries a round-value of zero. This value is increased every time that a node retransmits the message. When a node receives a message for the first time and its round number is equal or higher than the maximum rounds parameter, the node does not retransmit the message. This round-value can be seen as a TTL (Time-to-live) of the message being disseminated. Many simple implementations of gossip dissemination protocols set this value to infinite.

Gossip interactions among pairs of nodes can be conducted in three different ways: eager push gossip (also known as push gossip), pull gossip, and lazy push gossip [21]. Eager push gossip [21] consists in the original behavior as described previously, where a node sends the payload message to t (fanout) random neighbors once it receives it for the first time. This is the faster communication mode, although it can be expensive in terms of traffic, specially if payload messages are big—due to the redundant nature of gossip.

An alternative to this is to employ pull gossip [21], where the nodes which do not yet have the payload message ask t random nodes for it. If one of those nodes has it, then it sends the payload message back to the node who asked. This solution produces less traffic than eager push gossip if the payload messages are big, because the number of redundant payload messages being transferred is not as high. However, if the payload messages are small (specifically, if they are smaller than the "ask" messages), then this solution ends up creating more traffic due to the fact that now we have—in practice—two messages being transferred between each pair of nodes (the payload message and the "ask" message). Pull gossip is also usually the slowest solution, since the nodes that do not have the payload message are the ones who have to ask for it to get it, and since they do not know when there are new messages, they end up setting a periodical timer to ask for new messages—a timer that cannot be very frequent because otherwise the traffic would be too much, and it would not be worth it to use a pull approach.

Finally, the other—more sophisticated—communication method is lazy push gossip [21]. In this method, the nodes that receive the payload message for the first time send a payload message ID (*mid*) to t random nodes. If those nodes do not possess that payload message yet, they reply asking for the payload message. Then, the payload message is sent to them. This communication mode is faster than pull gossip and if the payload messages are big, it is faster than eager push gossip too. However, it is the communication mode with more communication steps, which adds complexity to the system. In Figure 2.2, we can see a diagram illustrating each of the communication modes explained above.



(a) Eager push gossip. (b) Pull gossip. (c) Lazy push gossip.

Figure 2.2: Diagrams of three different gossip communication modes.

There are numerous variants of gossip—all with different advantages and disadvantages. All these variants use different fanout and/or communication modes, some even combining multiple communication modes at once [17]. The simplest gossip strategy is flood [20]. In this strategy, eager push is used as communication mode and the fanout is always the same as the number of neighbors of each node. Flood produces a lot of traffic, but it assures the highest probability (out of all strategies) for every node to receive the

disseminated message. Other commonly used strategy is Anti-Entropy [31]. It uses a fanout of 1 and pull as the communication mode (executed by every node in the system periodically). This strategy is widely used for replicating non time-sensitive data in distributed databases [31]. It is extremely slow but does not use a lot of bandwidth (due to very low traffic). The random-walk strategy [20] also uses a fanout of 1 but uses eager push as the communication mode (usually with a maximum number of retransmissions per message, i.e., with a maximum rounds parameter set to some value below infinite). This strategy is useful to search content in the network. For example, if a node wants to search for a file in the other nodes of the network, it can send a message in a random-walk so that if the message reaches a node that has that file, that node replies to the searcher node saying that it has the file.

Some of these variants can be used to implement various gossip-based services that are going to be thoroughly studied in this work. In the next section, we are going to delve into some of these.

2.3 Broadcast

The broadcast problem can be synthesized as follows: a process needs to transmit the same message m to n other processes (where n is every process in the system including itself). There are several different formulations of this problem. Some of these have weaker guarantees, like the best-effort broadcast (e.g., IP Multicast [6]). Others have strong proprieties that provide strong guarantees, like the reliable broadcast problem [12].

2.3.1 Reliable Broadcast Problem

As the name suggests, the reliable broadcast problem [12] is the strongest problem. It is, hence, a stronger version of the best-effort broadcast problem. Reliable broadcast states that if a correct process i delivers message m, then every process j has to deliver message m at some point in time.

One solution to solve this problem is to use flood [20]. As described above (in Section 2.2), when using flood, when a node receives a message for the first time, it sends that message to all of its neighbors, i.e., all the nodes that that node knows about. This way, we are flooding the network with the message, so that all nodes receive it (and, consequently, deliver it to the application layer). Although flooding the network is effective (and it is a very simple and easy to implement solution), it creates a lot of redundant messages in practice, which leads to a lot of traffic and, consequently, high bandwidth usage. For that reason, flood is not an optimal solution for broadcasting messages in the real world. Furthermore, if a connected node is temporarily disconnected from the system (i.e., has no valid neighbor) it might not receive messages flooded in the network, and an additional mechanism, such as anti-entropy [31], is required, increasing the cost of the protocol. Since we cannot solve the reliable broadcast problem without such a high

load of redundant messages, we have to relax the reliable broadcast problem so that we can implement solutions that work in practice. That is where the probabilistic broadcast problem comes into play.

2.3.2 Probabilistic Broadcast Problem

The probabilistic broadcast problem is a relaxed version of the reliable broadcast problem. In the probabilistic broadcast problem, if a message is delivered by some correct process, then it is delivered by every correct process with a configurable high probability [13]. This means that, when implementing a solution to this problem, we can choose how likely it is that all nodes in the network receive the broadcasted message, effectively controlling the trade-off between communication cost and reliability. This is relevant since there is a direct correlation between the likelihood of all nodes to receive the broadcasted messages and the number of redundant messages traversing the network during the process. That means that if we relax the likelihood of all nodes receiving the message, then we can send less redundant messages and, consequently, generate less traffic.

Epidemic broadcast is a family of protocols that use gossip to solve the probabilistic broadcast problem. The communication mode used is eager push and $t >= ln(n) \wedge (t < n)$, t being the fanout and n being the total number of nodes in the network [20]. It is called epidemic broadcast because the message is spread on an epidemic fashion, where each node "infects", i.e., sends the message to t (fanout) other nodes [18]. The probability of all nodes receiving the broadcasted message is configurable by changing the fanout used by the algorithm [13]. A higher fanout will result in a higher probability of all nodes receiving the broadcasted message but will also increase the number of redundant messages [21]. A lower fanout does the opposite. When implementing epidemic broadcast algorithms, the challenge is to find the harmony between the likelihood of all nodes receiving the broadcasted message versus generating a lot of redundant traffic, i.e., choosing the right fanout.

According to [7], the probability p of all n nodes to receive the broadcasted message using an epidemic broadcast protocol with a fanout t is:

$$p = 1/(1 + n * e^{-t}) \tag{2.1}$$

2.3.2.1 The Beauty and the Beast of Epidemic Broadcast

Redundancy can be seen as the Beauty and the Beast of epidemic broadcast. On one hand, it can be good because since messages can be lost in the network (while in transit between two nodes), redundancy will make sure that—with a high probability—everyone will receive the message, because it will organically compensate for the lost messages in the network.

On the other hand, redundancy has a dark side, because on average each process will receive the broadcasted message t (fanout) number of times (from different processes).

This adds a lot of redundant traffic to the network and can have a big impact on a network composed by low bandwidth devices. Furthermore, processing such redundant messages also has a cost in terms of CPU usage. The total cost of messages of an epidemic broadcast protocol with a fanout t on a network with n nodes is $t * n$.

2.3.2.2 Performance Metrics

There are a series of metrics that can be used in order to evaluate and compare epidemic broadcast protocols. These metrics can also be generalized to evaluate the performance of other classes of gossip-based protocols.

Reliability Reliability in this context can be defined as the percentage of nodes of the system that received (and then delivered to the application layer) a broadcasted message [21]. A broadcast reliability of 100%—where all the nodes receive and deliver the broadcasted message—indicates that atomic broadcast was achieved [13]. Even though we usually aim at probabilities very close to 100%, unless flooding is used (i.e., the fanout is equal to the number of neighbors), with epidemic broadcast we can never be sure that we are going to achieve such a reliability—due to the probabilistic nature of epidemic protocols. A higher fanout usually leads to a higher reliability and a lower fanout usually leads to a lower reliability [13].

Relative Message Redundancy (RMR) The relative message redundancy is a metric that captures the message overhead in an epidemic broadcast (or other gossip-based broadcast mechanisms) [17]. It is equal to:

$$(m/(n-1)) - 1 \tag{2.2}$$

m being the total number of messages exchanged during the broadcast and n being the number of nodes that received and delivered the broadcasted message [17]. This metric is only applicable when $m >= 2$. Usually, the optimal RMR value is zero [17]. A value of zero would mean that there was exactly one payload message exchanged per receiver in the system. That would mean that we spent exactly the bandwidth necessary to propagate the message to all nodes—not wasting any more than necessary. A high value of RMR indicates that a lot of redundant traffic was generated and, therefore, a high amount of bandwidth was unnecessarily consumed. Higher fanouts usually lead to higher RMR and lower fanouts usually lead to lower RMR [17]. When using pure gossip (such as simple epidemic broadcast) to disseminate a message, RMR tends to $t - 1$, t being the fanout [21].

Control messages (like the ones used on the pull gossip strategy) are not considered in this metric, i.e., they do not count towards the value of m. There are two main reasons for this. The first one is the fact that control messages are usually significantly smaller than the actual payload messages—otherwise we would not

be using a communication mode that relies on control messages. Therefore, the control messages do not have a substantial impact on the total generated traffic and, consequently, can be ignored.

It is worth noting that it is easy to have a high reliability with a high RMR (by having a very high fanout) and it is also easy to have a low reliability with a low RMR (by having a very low fanout) [17]. Nevertheless, it is challenging to have a high reliability with a low RMR—which is the ultimate goal. The challenge lies on finding the fanout that provides the best balance (according to our priorities) between a high-enough reliability and a low-enough RMR.

Last Delivery Hop (LDH) The last delivery hop metric [21] measures the maximum number of hops performed by a delivered message, i.e., the maximum number of nodes that a message went through before being delivered to the application layer. This value is usually highly dependent of the network diameter, i.e., how wide the network can be in terms of paths between nodes [21]. For example, in a network where the shortest amount of links between two nodes is d, assuming an atomic broadcast started by one of those nodes, the LDH of that broadcast will never be smaller than d.

Latency The latency of an epidemic broadcast is given by the difference between the time upon which the last process delivers the message and the time upon which the first process (the original broadcaster) broadcasts the message [20]. Usually, we strive for low latency. However, the obtained latency is not solely dependent on the epidemic broadcast protocol—it is also dependent on the network itself (e.g., the latency between nodes, the node relative locations). If every path between the nodes has the same latency (which is very unlikely in practice), we can compute the broadcast latency by multiplying LDH by the link latency between two nodes (also known as *per node latency*) [21].

2.4 Overlays

An overlay is a logical network on top of another network (e.g., the physical network). In an overlay, nodes define (logical) neighboring relationships between them. These neighboring relationships can be materialized as links. This set of links and nodes constitutes an overlay, which can be seen as a graph, composed of nodes and links established between them. This overlay is then used by applications or services to propagate messages between the nodes. The set of neighbors of each node (i.e., the nodes to whom a node shares an overlay link with) is called the partial view of the node [20]. In other words, the partial view of a node n can be seen as the set of nodes that are known by n.

An overlay (or graph) can either be undirected or directed, i.e., having asymmetric links or exclusively symmetric links, respectively. In a symmetric overlay, if node a

has node b in its partial view (i.e., as neighbor), then node b has node a in its partial view. When using symmetric views, a node is always sure of the nodes that have him as neighbor and that can be useful in some cases. For example, if the overlay is symmetric and a node's partial view is not empty, then that node is sure that is not disconnected from the network, i.e., that other nodes have it in their partial views. However, symmetric partial views are usually harder and costlier to maintain.

Contrarily to using a partial view, we could use a global view membership system. This would mean that each node would know every single node in the network [9]. Using this strategy, we would not need an overlay, since all nodes could interact with every other node (as long as they were connected to the physical network). That raises the question: why bother with creating and managing an overlay when we can have all nodes communicating with all other nodes? The answer is simple—when we are dealing with a large system (i.e., a system with a very high number of nodes), the set of nodes in the system is not static. Large systems are often dynamic because processes might be added at all times (to deal with additional load, for example) and other processes might leave the system anytime (e.g., if they crash). If every node (i.e., process) contains a global membership, that would mean that that node would have to keep up with every addition and removal to the system—which might not be realistic, because nodes can be far away from each other (thereby presenting high latencies) and that would lead to unacceptably high bookkeeping costs [23]. This could easily result in network saturation and total system inoperability.

There are a number of proprieties that overlays must own in order to support a high level of fault tolerance and fast message dissemination, i.e., for epidemic broadcast to be effectively implemented on top of them. These proprieties are useful to ensure the correctness of the overlay (when it comes to connectivity and accuracy) but also to measure the quality of the partial views—in terms of fault tolerance, message dissemination efficiency, etc. The most relevant of these proprieties are:

Connectivity The connectivity of an overlay indicates whether all nodes in the network are connected, i.e., if there is a path between every correct node a to every correct node b. For an overlay to be correct, it has to be connected. In other words, no node can be isolated from the other nodes. If a network is not connected, some nodes will not be able to communicate with the remaining elements of the system [21].

Accuracy The accuracy of a node is given by the number of correct nodes in its partial view divided by the total number of nodes in its partial view [18]. By correct, we mean nodes that did not fail, i.e., that are running. Optimally, we would want that partial views would have an accuracy of 1 (100%) at all times, but that is obviously impossible because nodes can crash and the partial views of the nodes that had those as neighbors take some time to identify such failure events. Therefore, we aim to have an accuracy as close to 1 as possible. The accuracy of an overlay (or graph) is defined by the average of the accuracies of all the correct nodes in it [18].

For an overlay to be correct, its accuracy has to eventually be equal to 1 (if no nodes are removed nor added for some amount of time).

Having a high overlay accuracy is very important for epidemic dissemination. If the accuracy is low, it means that nodes have a lot of faulty nodes in their partial views and when disseminating a message to t random neighbors, some of those t nodes will not contribute to the effective dissemination of the message, decreasing the broadcast reliability (unless we use a higher t in order to mask this issue) [21].

Degree Distribution The *out-degree* of a node is given by the number of nodes that are present in its partial view [21], i.e, its number of neighbors. The *in-degree* of a node is given by the number of nodes that have that node as neighbor [21].

The *out-degree* of a node is useful to infer the node's contribution to the membership protocol, i.e., how important that node is in maintaining the overlay connected [21]. For example, a node with an *out-degree* of three is less relevant in maintaining the overlay than a node with an *out-degree* of ten, because it has the responsibility of managing and disseminating the messages to more nodes. If a node with a high *out-degree* crashes, it has an increased probability of affecting overall overlay proprieties such as the connectivity.

The *in-degree* of a node evidences how well known a node is in the network. It also has a direct correlation with the number of redundant messages that that node might receive—a higher *in-degree* results in a higher probability of that node receiving a high number of redundant messages [21], when messages are being broadcasted through the overlay network.

In the case of a homogeneous network (i.e., a network where all nodes have the same resources and links possess the same proprieties), all processes should have a similar number of *out-degree* (and *in-degree*), so that the traffic load is well distributed between them [18]. It is also worth mentioning that in a symmetric graph/overlay, a node's *in-degree* is equal to its *out-degree*.

Average Path Length A path from node a to node b is the composed by the edges that a message goes through to travel from a to b. The length of the path from a to b is given by the number of such edges. The average path length of an overlay/graph is equal to the average of the shortest paths between all pairs of different nodes in the overlay[1]. The average path length should be low, so that a message does not have to pass through a lot of nodes in order to reach its destination, which adds latency [18]. This metric is closely related to the overlay diameter [21]—a wider network results in messages having to go through numerous edges to go from one side to the other of the overlay.

[1] This propriety is only worth measuring if the network is connected. Otherwise, at least one node will have an infinite shortest path length to all other nodes—which will result in an infinite average path length, by definition.

Clustering Coefficient The clustering coefficient of a node is equal to the number of edges between that node and its neighbors divided by the total number of possible edges between all of them [18]. It results in a number between zero and one. A clustering coefficient close to one means that a node shares a lot of the same neighbors with its neighbors. The clustering coefficient of the graph/overlay is equal to the average of the clustering coefficients of all nodes. An overlay with a high clustering coefficient is bad for two reasons: i) it will produce more redundant messages for nodes, since they have essentially the same neighbors as their neighbors; ii) it lowers the fault tolerance of the overlay because if most of the nodes share the same neighbors as their neighbors, then those nodes do not have a lot of neighbors outside their clusters, which means that the inter-cluster links are scarce—if they are broken, the overlay can more likely become disconnected. Moreover, a high clustering coefficient usually results on a high average path length [21], which is usually an undesirable propriety.

As mentioned in Section 2.1.2, the membership layer is responsible for managing the partial view. This partial view has to be frequently updated so that the nodes that left the network are removed from it (to improve accuracy) and the nodes that entered the network become known by (some) other nodes.

Overlay networks can be divided in two groups: structured and unstructured. They both have different characteristics and have been employed for different use-cases. In the following sections we are going to delve into them.

2.4.1 Structured Overlays

Even though this dissertation focuses on unstructured overlays, it is worth discussing structured ones. A structured overlay is an overlay whose topology has proprieties known *a priori* [20]. Each node in the overlay is usually identified by a unique identifier. The neighbors of a node are defined (at least partially) considering the identifier of nodes, such that the overlay topology has a global organization that can be exploited. For instance, Chord [34] structures nodes in a ring where they are organized in increasing order of their identifiers (assuming that the identifier space is circular). This ring topology is then exploited to easily navigate the overlay to locate particular nodes considering their identifiers.

Knowing some topology proprieties allows us to implement mechanisms that operate on top of the overlay in a much more efficient way. Thereby, we can use structured overlays to implement specific abstractions and functionalities more efficiently than with unstructured ones [34].

One example of a problem that can be very efficiently solved with a structured overlay is the exact location problem [20]. This problem can be defined informally as finding an object (i.e., a resource, such as a file) by its unique ID. If we used an unstructured network, we would be blindly searching for the object across random nodes. However, with a

structured network, we can take advantage of a scheme to attribute the responsibility of hosting that resource to a particular node, and then find it faster and more efficiently, i.e., with fewer messages. This can be done by hosting each object in the nodes (or node) whose identifiers are closest to the object ID.

On the other hand, structured overlay networks have an important drawback, which is the slower convergence when compared to unstructured overlays. Due to having a strict structure, structured overlays are not as flexible when membership changes happen, since nodes that join or leave the network trigger a mandatory restructuring of the overlay for it to enforce its topological proprieties [20]. Besides this, structured overlays are not as fault tolerant as unstructured overlays—a node can quickly become isolated if a relevant part of its neighbors crash [20]. However, most implementations try to minimize these problems (specially the lack of fault tolerance) with some algorithm tweaks, as we are going to discern below.

There are many protocols to build and manage a structured overlay. Chord [34] and Kademlia [26] are two of the most well known, which we will discern in more detail.

2.4.1.1 Chord

The Chord [34] protocol main functionality is—given an object identifier, find the node responsible for managing that object. To achieve this, Chord structures nodes in a ring–like topology, each node being responsible for an interval of object identifiers. Each of the nodes has a unique ID, and they are organized based on that ID. Each node is responsible for the data objects whose identifiers fall in the interval between that node's predecessor on the ring and its own identifier.

A simple and naive routing solution would be for each node to have the next node in the ring in its view (i.e., node n would know the IP address and port of $successor(n)$). That way, each node would only know one other node (its successor). Hence, whenever a node would receive an operation to find an object with a specific identifier, it would check if that node was the one that managed it, and if it was not, it would send the operation to its successor. When the message finally arrived to the node that manages the object, that node would reply to the client.

Chord proposes a more complex routing solution in order to deal with the fault tolerance and efficiency problems generated by the simple solution above. Essentially, it works by each node n maintaining in its view a set of nodes whose IDs are $neighID = n + 2^{i-1} \wedge i > 0 \wedge neighID < maxID$, $neighID$ being the ID of neighbor and $maxID$ being the maximum possible identifier. In this solution, in case the node receiving the search query has in its view a node with ID equal to the identifier being searched, it redirects the query to that node. Otherwise, the node receiving the search query redirects that query to the successor of the known closest preceding node of the node that manages the object. Effectively, this halves the distance to the target node at each routing step.

With this solution, each node stores information about m nodes when the total number

of nodes in the network is 2^m. This number is not very high, as it increases logarithmically with the total number of nodes in the overlay. However, it is high enough to make the overlay robust against nodes failing and broken links. The fact that each node has neighbors well spread through the overlay also helps with quickly locating the searched objects (since a low number of hops is often sufficient to find the node that manages the target object).

It is also worth mentioning that Chord proposes a periodical stabilization process with the goal of converging the overlay in case nodes recently joined or left the network. In this process, nodes update their successors and predecessors in order to match the correct structure of the overlay. This cyclic stabilization process presents a trade-off between spending a lot of bandwidth (by doing very frequent stabilization processes) to have a correct overlay at (almost) all times or spending little bandwidth and risk having an often inconsistent overlay.

2.4.1.2 Kademlia

Kademlia [26] is a protocol that builds and maintains a structured overlay that is widely used. Although the number of expected hops for each search query is $O(log(n))$ (similar to Chord), Kademlia presents some clear advantages when compared to Chord. Some of these advantages are the fact that configuration information spreads automatically as a side effect of search queries. Besides that, Kademlia uses parallel asynchronous search queries in order to be fault-tolerant to nodes that have possibly crashed mid-execution.

This protocol has a lot of optimizations compared to simpler protocols (like Chord), but perhaps the most important one is that fact that it uses an XOR metric to measure distances between nodes. Because of that, Kademlia is able to use symmetric links between the nodes, which leads to a greater flexibility on the search queries redirection because—unlike Chord—Kademlia can redirect the search query to any node that is close to the node that should have the object. In fact, it can even redirect the search query to multiple appropriate nodes in parallel. This concurrency can be controlled by a concurrency parameter α that trades-off low bandwidth usage for low-latency and increased fault-tolerance.

2.4.2 Unstructured Overlays

An unstructured overlay is an overlay where the neighbors that each node has are selected at random [20]. Therefore, unlike structured overlays, an unstructured overlay has a random topology, i.e., we cannot infer proprieties about the topology of the overlay before it is actually built.

This type of overlay has a lower maintenance cost than structured overlays because there is no structure to be enforced [20]. Besides that, they tend to have better fault tolerance than structured overlays (as explained thoroughly in Section 2.4.1), mainly due to the fact that no strong topological proprieties have to be maintained when a node

updates its partial view, i.e., any node can have any random node in its view, unlike structured overlays, where each node has to have a very specific set of nodes in its partial view.

Due to its dynamic and low-cost adaptability and high tolerance to failures, unstructured overlays are useful for message dissemination (e.g., epidemic broadcast) [7]. They are also useful for replication of data across numerous nodes [24]. Another use case is system monitoring, where nodes monitor each other (e.g., their load, if they have not crashed) [36]. A different problem where using unstructured overlays is useful is the (generic) resource location problem, where a node has to find some resource in the network (e.g., files) matching a given set of proprieties (e.g., $size >= 256$) [24]. These are all problems where unstructured overlays are highly advisable (instead of structured ones) because solving these problems would not benefit from an existing structure, i.e., a structured overlay does not make solutions to these problems much more effective or efficient.

When it comes to building unstructured overlays, there are two strategies: cyclic and reactive. A cyclic strategy uses periodic timers to shuffle the partial views of nodes between them [20]. From the point of view of a node, it sets a periodic timer t and then every t units of time it will perform an operation where its neighbors may change (and it may also lead to updating the contents of some of its neighbors' partial view). This strategy is called cyclic because the only trigger for maintaining the overlay is given by periodic (i.e., cyclic) actions, i.e., it is not reactive upon some other event (like nodes joining the network).

2.4.2.1 Cyclon

Cyclon [36] is a cyclic, gossip-based membership management protocol (also named as peer-sampling protocol) that builds graphs (i.e., overlays) with low diameter, low clustering, highly symmetric node out-degrees, and that are highly resilient to massive node failures. The protocol is also capable of restoring randomness when numerous nodes fail, specially when compared to other (simpler) shuffling protocols. In [36], the authors presented an experimental analysis of a basic shuffling protocol for large networks (to demonstrate that shuffling is indeed a promising technique to build these protocols), and then presented an enhanced version of that basic shuffling protocol—Cyclon.

Cyclon employs an enhanced version of shuffling. Shuffling consists on (generally) two nodes switching neighbors between themselves, so that at the end of the shuffle operation, they both end up with a different partial view than they had when they started it. The number of neighbors shuffled by each node is called the shuffle length. Enhanced shuffling follows the same model as basic shuffling, with the difference that nodes do not randomly choose which neighbor to shuffle views with. Instead, upon shuffling, each node selects the neighbor whose identifier was created earliest. This is achieved by every node having, in its view, not only a list of nodes, but, instead, a list of pairs (node, age).

This way, each time a node shuffles, it increases the age of each neighbor and chooses the neighbor with the highest age to shuffle with (instead of a random one). Note that the age of an identifier is left unchanged when an identifier is sent to a peer during the shuffle. The other difference between Cyclon and basic shuffling protocols is in the *join* operation. Cyclon implements a sophisticated way of a node joining a network by only knowing one neighbor (contact node) without disrupting the randomness of the network, which is key to maintain the proprieties of unstructured overlays (such as high fault-tolerance).

In [36], the authors suggested a time interval between shuffles of 10s. However, for some applications, this might be too long, since the network can take minutes to recuperate from node failures (depending on the number of nodes that failed). That said, there is clear correlation between the bandwidth used and the speed of the protocol to detect and act upon failures. This trade-off between low-bandwidth usage and fast recovery might be one of the biggest draw-backs of this algorithm. This happens because the algorithm is exclusively cyclic, having no reactive strategy to failed nodes.

2.4.2.2 SCAMP

Unlike Cyclon, SCAMP [9] is a reactive gossip-based membership management protocol. A reactive strategy changes the membership of nodes in the overlay every time specific events are triggered—like nodes joining or leaving the network, instead of using periodic behaviors. SCAMP is also self-organizing, meaning that the size of the partial views is dynamically and locally adapted according to the number of nodes in the system, as to ensure reliability in large scale settings.

SCAMP works by having each new neighbor sending a subscription (i.e., presenting itself) to an arbitrary node that it already knows (contact node). This means that a node that enters the system starts with one node in its (partial) view. The node that receives the subscription, forwards it to all nodes on its partial view and then sends the subscription again to more c nodes of its partial view (c being a design parameter that determines the proportion of failures tolerated). When a node receives a forwarded subscription, it adds the subscriber to its partial view (if it is not already there) with a probability computed based on the number nodes already on its view. Else, it forwards the subscription to one of its existing neighbors. This probability makes the system configure itself towards (partial) views of average size $(c+1) * log(n)$, n being the total number of nodes in the system.

For failure detection, the protocol suggests two different techniques (that can be used simultaneously). The first one is for each node to save a different view (*in-view*) that contains the set of nodes that have that node in their partial views. That way, each node knows which nodes have it as neighbor. Then, when a node wants to leave the system, it just has to send a *leave* message to all nodes in its *in-view*. When a node receives a *leave* message, it removes the sender of the message from its view. The second technique is aimed to solve the problem of nodes that crash without warning (i.e., without sending the

leave messages). In this technique, nodes periodically send a heart-beat message to their neighbors in order to make sure that they are still alive. If a node does not answer the heart-beat for x times in a row, it is removed from the partial view of the node that sent the heart-beat. Both of these techniques are effective ways of maintaining a high overlay accuracy.

Besides being proven mathematically, the algorithm was also tested against a full membership protocol system in [9]. Using SCAMP, the view size of each node converged to an average size of $(c+1)*log(n)$. Because of this, when using the gossip algorithm with SCAMP, the fanout should be equal to the partial view size of each node, since a fanout $>= log(n)$ is necessary to achieve reliability. In terms of resilience to failures, SCAMP proved to be very reliable, with results very close to global membership knowledge—even when almost half of the nodes failed.

On the other hand, Scamp presents a significant limitation—if a node was the last one to enter the system (and no nodes enter the system after it), it will ever only know one other node, i.e., its (partial) view size will be 1 forever. Even if a node was not the last one to enter, it can stay with a low view size for a long time if not many nodes enter the system right after it did, specially if the nodes that enter the system after that node do not choose a contact node nearby it. This means that this algorithm works well only if a lot of nodes are always entering/exiting the system. This happens due to the reactive nature of the algorithm—if there are no external changes to the overlay (i.e., no nodes joining or leaving) then the overlay will not change. Cyclon does not have this problem because no matter how many nodes enter or leave the network, it will always shuffle neighbors periodically.

2.4.2.3 HyParView

HyParView [18] is a membership protocol that supports gossip-based broadcast, which ensures high levels of reliability even when the rates of node failure are very high. This protocol uses two distinct partial views to ensure that high reliability is achieved, even when using a lower-than-average fanout size on the gossip protocol being used on top of it.

Many membership protocols with healing properties see the reliability of message broadcasts be seriously affected after heavy failures [18]. Even for protocols that are able to totally recover, it takes a long time to restore their desirable view properties when a very high number of nodes fail [18]. Furthermore, when using usual membership protocols, we often have to use high fanouts if we want to achieve high reliability, due to the lack of fault tolerance of these protocols [18]. HyParView claims to solve these problems.

The protocol works as follows. Each node maintains two views: i) small symmetric active view of size t+1, t being the fanout used in the upper (service) layer for the epidemic broadcast; ii) a bigger passive view of backup nodes that can be promoted to the active view when one of the nodes in the active view fails. Failures are detected using TCP as an

(unreliable) fault detector. The active view is maintained reactively and the passive view is maintained cyclically (by performing shuffles with other nodes, like Cyclon). When a node sends to its neighbor a set of nodes to be shuffled, it includes in the set some nodes from its active view as well. This is done to increase the probability of nodes having passive views with active nodes. By using these two views and TCP as a reliable transport, it is possible to use smaller fanouts, while maintaining a very high reliability, resulting in more cost-effective gossip protocols. Moreover, by using TCP as a reliable fault detector, this membership protocol has fast healing properties, quickly recovering from a high number of failures.

On a negative note, a lot of messages are used to maintain symmetric views (at least two times the number of needed messages to maintain asymmetric views). This might cause some unnecessary overhead. On the other hand, since the protocol provides us with the capability of using smaller epidemic broadcast fanouts (while providing the same levels of reliability), perhaps the number of messages spent to guarantee symmetric views ends up compensating the number of messages saved on the service layer (due to the lower fanout)—which is likely—specially if the overlay is fairly stable (i.e., if it has low churn[2]).

2.4.3 How to Adjust/Bias the Network Topology

Unstructured overlay networks—while being highly reliable and adaptable—can have performance caveats due to the randomness of the links that are established between the nodes [20]. This means that some connections between nodes might end up being inefficient (in terms of latency or other metrics) and that is not good, because it leads to an undesirable impact on the network efficiency. This is the problem that T-Man [11] and X-Bot [19] try to solve—how to optimize these networks, while maintaining some key proprieties.

Both of these protocols bias the topology of unstructured overlays according to some chosen criterion. The protocols are agnostic to the criterion chosen, i.e., any criterion can be used as long as it allows nodes to rank other nodes (in the case of T-Man) or links (in the case of X-Bot) based on it.

2.4.3.1 T-Man

T-Man [11] is an algorithm that can be applied on top of a random graph in order to bias its topology according to some criterion. Therefore, T-Man is capable of emerging structure out of a random graph, i.e., out of a pure unstructured overlay.

The protocol operates between the membership layer and the service layer, i.e., it uses the membership layer, and it presents itself as the membership layer to the service layer

[2]Churn is defined by the frequency to which nodes join and leave the network/overlay. An overlay with high churn is an overlay with a lot of nodes frequently joining and leaving, while an overlay with low churn is a stable overlay where nodes join and leave infrequently.

above. It runs cyclically. Every time interval t, each process picks the highest ranked node from its view (according to the ranking function/criterion provided) and sends to that node a set of nodes that includes its current view, itself and a sample from the random view on the membership layer. When a process receives a set of nodes, first, it replies with a set containing its current view, itself and a sample from the random view on the membership layer. Then it merges the received set with its own view. So that each process has always c nodes in its view, the merge operation merges both sets based on the ranking function, i.e., it ranks the nodes from both sets and chooses the highest ranked c nodes.

T-Man also proposes some optimizations to the solution described above. One of them is to forbid each node to communicate (in the context of T-Man) more than twice per interval t. That way, if a node tries to communicate with other that currently cannot communicate, it skips that node and tries with the next one. The authors in [11] affirm that this technique strikingly improves the convergence speed of T-Man.

Another proposed optimization is to reduce the payload of messages sent between nodes. This can be achieved by reducing the number of nodes sent in messages through inferring what nodes would definitely be discarded by the receiver of the set.

The periodic optimization of each node's views by applying the T-Man protocol makes the network topology more efficient (according to the criterion/ranking function used) over time. In [11], the authors construct a torus out of a random graph with 2500 nodes in just 15 cycles (with a view size of 20)—which is remarkable. On the other hand, although T-Man ensures that the topology of the overlay improves in a fast manner, it does so at the risk of allowing the topology to break (and the overlay to lose some of its desired proprieties) [4].

2.4.3.2 X-Bot

The goal X-Bot [19] is to optimize the overlay by minimizing the cost of the links between the nodes, while striving to maintain the same number of links as the original overlay network. The link cost function is parameterized and encapsulated in a companion oracle.

X-Bot allows us to use any efficiency criteria X, i.e., we can choose the oracle that we want for the costs of the links between neighbors. This means that we can choose latency as our cost indicator or any other characteristic of the path (like distance, monetary cost of infrastructure, etc.)—X-Bot is completely agnostic to the oracle. Besides this, X-Bot preserves several key proprieties of the overlay, like a low clustering coefficient and low overlay diameter.

The protocol is decentralized and relies on a 4-node coordinated optimization technique in order for those 4 nodes to switch links between them so that the least link cost is achieved. By always switching links between 4 nodes, the algorithm makes sure that most node degrees remain the same. This switch is made periodically, making sure that the overlay never stabilizes at some local minimum. In this technique, each node starts optimization rounds in which it tries to switch one node of its active view with one

(better) node in its passive view, and that node communicates with another node that communicates with another node. To avoid breaking some key proprieties of the overlay network (like the low clustering coefficient, low average path length, or connectivity), the algorithm keeps some unbiased neighbors that it never tries to switch/bias. These "unbiased neighbors" should be the y neighbors with the "highest-cost" in each node's initial partial view, y being a parameter.

This protocol was tested and compared with T-Man, Araneola [27] and GoCast [35], which are other protocols that try to bias the network in any way to achieve better efficiency. Two testing scenarios were used: Cartesian scenario and Planet-Lab scenario. The oracle used measured latency between links (as cost). X-Bot managed to have the lowest latency in a Cartesian scenario while maintaining a reliability of 100%. In the Planet-Lab scenario, X-Bot had the 2nd lowest latency, but T-Man—that achieved the lowest latency—presented a reliability of 16%, while X-Bot maintained a reliability of 100%.

2.5 Addressing Heterogeneity in Unstructured Overlays

All the protocols presented above—either for building and maintaining unstructured overlays or for biasing their topology—assume that the network is homogeneous, i.e., they assume that every node (and many times link) have similar resources. However, in real world scenarios that is not true—specially in the edge [4]—where we can have very powerful machines (e.g., datacenter servers) and very weak machines (i.e., machines with low resources), such as IoT devices, smartphones, etc.

For this reason, in most real-world scenarios, it is not wise to evenly distribute load on the nodes and network links, since they are heterogeneous in nature, which would make some nodes be extremely saturated with work while others would be operating below their capacities. This leads to low performance on services, resulting in high latency (due to some saturated nodes, for example) and poor bandwidth allocation [3]. Therefore, there are some techniques (i.e., protocols/algorithms) that aim to tackle this problem.

There are a lot of ways to tackle resource heterogeneity. Some solutions aim at solving the heterogeneity problem at the service layer by optimizing epidemic broadcast (or other gossip) algorithms for heterogeneous networks. Other solutions tackle the problem at the membership layer by trying to optimize the overlay for heterogeneity so that when some service (like epidemic broadcast) uses it, it benefits from the already optimized overlay.

The main goal of the protocols and algorithms that aim at addressing the resource heterogeneity in unstructured overlays is usually to minimize latency. This is because it is the metric that often has the bigger impact in a real world scenario [4]. Besides that, by optimizing latency we are also indirectly optimizing other metrics—like bandwidth, distance between nodes, etc. For example, if an algorithm strives to achieve low latency, it will probably do so by fairly distributing bandwidth among resources. There are also solutions that—instead of trying to optimize latency—have parameterized optimization

functions that allow us to directly optimize whatever metric/aspect we want (including machine operation cost, for example).

2.5.1 Tackling Heterogeneity at the Service Layer

As mentioned above, there are several solutions that try to optimize services for heterogeneous resources. In this case, we are interested in protocols that adapt epidemic broadcast for heterogeneous settings. Some solutions are presented below.

2.5.1.1 Emergent Structure in Unstructured Epidemic Multicast

The work presented in [3, 30] proposes a probabilistic broadcast protocol (commonly named adaptive gossip) for heterogeneous networks. The protocol works by lazily deferring message transmission according to a configurable policy. It combines two push gossip communication modes (eager and lazy) in an epidemic broadcast protocol.

The proposed gossip algorithm mixes eager and lazy push. In summary, before a node broadcasts a message, it runs a *split* function that—depending on the sender, the neighbors to which the message is being sent, and the message itself—decides to which neighbors the message should be sent via eager push and to which neighbors the message should be sent via lazy push. For example, if node a was going to broadcast message m to nodes $\{b, c, d, e, f\}$, it would first run a *split* function that would return something like $\{eager : \{c, e\},\ pull : \{b, d, f\}\}$; then, node a would send m via eager push to $\{c, e\}$ and via lazy push to $\{b, d, f\}$. It is worth noting that the broadcasting fanout (t) of node a does not change, since the function *split* always returns two sets (x and y), in which $|x| + |y| = |t|$ and $x \cap y = \emptyset$.

Being the goal of the protocol to achieve the best balance between lazy push and eager push communication modes, the authors in [3] presented 4 different strategies to implement the *split* function: i) deciding lazy/eager push based on a probability; ii) deciding lazy/eager push based on the number of hops that a message has traveled; iii) deciding lazy/eager push based on the radius between the sender and receiver nodes; iv) deciding lazy/eager push based on whether the sending node is one of the super-nodes (i.e., most powerful nodes) or not. The protocol uses monitors in order to extract the metrics necessary for using the strategies above. The authors in [3] suggest that hybrid strategies (mix-ups between the 4 alternatives above) are possible and recommended (depending on the case).

By testing adaptive gossip using a hybrid strategy, the authors in [3] have shown that by scheduling the transmission of payload in a combined eager/lazy push epidemic broadcast protocol, one can reduce the bandwidth while keeping a low latency. In terms of reliability, the tests confirmed that the proposed protocol does not impact reliability (compared to the normal gossip protocol). They also tested inputting noise in the metrics obtained by the monitors, and even in the worst case (when noise was the highest), the worst thing that happened was that the lazy push mode was used all the time or that the

eager push mode was used all the time. Also, note that even though the best results are achieved when all the nodes run the same strategy, correctness is still assured if they do not.

2.5.1.2 Low Latency Probabilistic Broadcast in Wide Area Networks

The work presented in [29] proposes a probabilistic broadcast protocol (i.e., a variant of epidemic broadcast) for heterogeneous networks that reduces the average end-to-end latency by dynamically adapting to the network topology and traffic conditions. It works by ensuring that a virtual backbone-like node structure of faster nodes is created, so that when a message is first sent, the protocol tries to route it first to the faster nodes which will then use all their available bandwidth to quickly and reliably disseminate the message to the remaining nodes.

The protocol is based on epidemic broadcast, but with two differences in the first w rounds of a message being forwarded among nodes. This means that the protocol is slightly different from the classical epidemic broadcast in the first w rounds, and then it adopts an equivalent behavior to it from round $w+1$ until the message is delivered to every node. The two main differences (in the first w rounds) are: i) each node broadcasts to a fanout tm, which is greater or equal to t ($log(n)$, with n being the total number of nodes)—that depends on the node's capacity; ii) each node broadcasts to a non-random set of nodes, which is generated based on the latency as perceived by the sender, their advertised bandwidth, and the sender's bandwidth. This technique results in the more powerful nodes receiving (and therefore sending) more messages, while the less powerful nodes receive (and send) fewer messages. Note that, although the function to extract the non-random set of nodes to broadcast probabilistically chooses more powerful nodes, it also chooses some weak nodes—to make sure that every node has a high probability of receiving the message.

The reported experiments in [29] proved that this approach effectively reduces the latency of the dissemination protocol, not only in heterogeneous environments, but also when the system is subject to high load. Another positive aspect of the protocol is that, by limiting the node selection bias to the early (w) rounds of the gossip procedure, the authors effectively prevent this mechanism from affecting the overall reliability of the broadcast. According to the experiments, this was proved true even when the network is homogeneous.

On a negative note, the authors claim that each node would obtain the estimated (bandwidth) resources of its neighbors through the membership protocol being used. If so, then that means that the membership layer has to be modified to accommodate this operation (to send each neighbor's resources), which results in this protocol not being able to be used with existing membership solutions—unless they are adapted. Adapting the current membership protocol solutions to work with this broadcast protocol would add overhead to the membership protocols. That said, since we would need to adapt the

membership protocol anyway, it would be more effective to solve the whole problem on the membership layer, i.e., make the membership layer responsible for constructing the optimized overlay considering the heterogeneous capacity of nodes, so that the broadcast layer could just receive the (already partially-biased set of) neighbors and simply broadcast to them.

2.5.2 Tackling Heterogeneity on the Membership Layer

We have presented solutions that tackle the resource heterogeneity problem on the service layer (through adapting epidemic broadcast to heterogeneous resources). However, that is not the only (nor possibly the best) way to address the resource heterogeneity problem.

Another interesting alternative is to address the challenges imposed by heterogeneity in the membership layer. By doing so, we are providing more flexibility to the services being implemented because the services just have to use the membership layer as before, with no change to the service internal logic. In practice, this means that we could use the same services as before on top of the already optimized overlay. Besides that, optimizing the overlay for heterogeneity (in the membership layer) is often cheaper than addressing the problem in the service layer because the service layer would have to gather data from the other nodes either through the membership protocol or through extra messages exchanged between nodes (in order to have a view of the network, similar to what the work reported in [29] does). Directly changing the overlay also lets us have more precision on the process, by biasing its topology in exactly the way we want—leading to a greater effectiveness in achieving the desired results.

We thereby present two solutions that bias/change the topology of the overlay in order to deal with resource heterogeneity.

2.5.2.1 Unstructured Overlays Based on Super-peers

In an unstructured overlay we can have specific nodes denominated super-peers [2]. These nodes are more powerful than average. They form an overlay between them. Hence, super-peers end up participating in two overlays: the general unstructured overlay (that all nodes participate in) and on an overlay made up only of super-peers.

The super-peer nodes can then be used to propagate more information than the other (non super-peer, i.e., regular) nodes, since they are more powerful. For a broadcast service, we could first disseminate the message between super-peers, and then they would disseminate the message to the regular nodes. This way, we would not saturate the network because of less powerful nodes having to propagate the message to many nodes.

This strategy is particularly useful for search query dissemination in unstructured overlays [2]. Regular processes connect to a super-peer and transmit to it the index of their resources. Search queries are then forwarded to the (closest) super-peer and then disseminated among super-peers to find the owner of the resource being queried.

There are some challenges with using super-peers, however. One huge difficulty is to find out which processes should be super-peers. Since processes/nodes do not have a global view of the network, they do not know if they are more powerful than average or not, i.e., since each node can only see the surrounding nodes, it can be easily fooled if it happens to be in an area with a lot of powerful nodes or a lot of weak nodes. For example, if a node has a lot of weak nodes around it, it may think that it should be a super-peer, when in reality that node is just an average node when considering the whole system. Other (less critical) difficulty that can come up is what to do when a super-peer fails—how can its connected regular nodes find another super-peer in a timely fashion.

2.5.2.2 Biased Layered Tree

Biased Layered Tree [4] is a novel, decentralized membership protocol which takes into account the computational and network capacity available in each node, manually encoded in a numerical value associated with each node and a proximity criteria based on IP prefix commonality. The protocol has the purpose of building a robust hierarchical tree topology that connects and allows managing large numbers of nodes across the cloud and edge.

The protocol assumes that each node contains a "level" value that is manually set. This value indicates whether the node is closer to the cloud or closer to end-devices—and consequently, it encodes whether the node has more or less resources (e.g., a node that has a level 0 is a cloud server and has, therefore, a lot of resources, a node that has a level 4 might be a 5G tower and has moderate resources, and a node that has a level 8 might be a smartphone and has few resources). The algorithm builds a tree–like structure, in which the more powerful nodes are closer to the root and the less powerful nodes are closer to leave positions. This hierarchy is achieved by using the levels. By using this tree–like structure, the nodes with more resources will send/receive more messages and the nodes with less resources will send/receive fewer messages, resulting in a higher efficiency and faster broadcast, minimizing bottlenecks. The authors also try to make sure that nearby devices end up being close to each other in the overlay structure by using the IP address common prefix as a distance criterion, which also helps on minimizing latency. Despite this, the resulting overlay structure is not a pure tree-structure since the protocol uses some redundant links that make the overlay more robust to failures.

The solution (Bias Layered Tree) was tested in a real-world heterogeneous edge network emulation alongside HyParView [18], X-Bot [19], Cyclon [36], and T-Man [11]. Bias Layered Tree provided the lowest latency of all evaluated solutions, due to the structure of the network, that positions nodes with higher capacity at higher points in the tree—which allows to mitigate the queuing effect due to network saturation. It also proved to have a reliability of 100% when no nodes failed.

On the other hand, when more than 25% of the nodes failed, Bias Layered Tree proved to be less reliable than all the other solutions. This happens due to the structured nature

of the overlay, which makes it less robust. Another negative aspect of the solution is the fact that it uses IP address common prefixes as a distance criterion. This is not very reliable, since modern ISPs networks have access to pools of IP addresses that have small common prefixes [8].

2.6 Discussion

Most of the work outlined in the course of this chapter is going to be useful in the development of our overlay and gossip protocol(s). Specifically, we can implement the first iterations of the gossip protocols based on already developed solutions that are optimized for heterogeneity, like the ones explained in Section 2.5.1.1 and 2.5.1.2.

When it comes to the overlay construction and management, we covered many solutions that can be helpful in different areas. On the membership management level, traditional protocols like Cyclon [36] and Scamp [9] can provide important insights on how to manage the overlay—both cyclically (in the case of Cyclon) and reactively (in the case of Scamp). Moreover, the membership management protocols optimized for heterogeneity (like Biased Layered Tree [4] and super-peer overlays [2]) can help us address the problem of heterogeneity by directly managing the overlay. On another note, solutions like T-Man [11] and X-Bot [19] can aid us in the direct manipulation and biasing of neighbors—T-Man in a more aggressive way (due to being faster but can easily break topology proprieties) and X-Bot in a safer manner.

2.7 Summary

In this chapter, we began by explaining the concepts needed to understand the following work. We started by describing the notion of peer-to-peer and what services can be implemented with it, then we delved deeply into broadcast, focusing in epidemic broadcast. After that, we explained the concept of overlays, the different types of overlays available, their advantages and disadvantages, the protocols that build (and maintain) them, and two different algorithms for optimizing unstructured overlays. Having introduced these concepts, we delved in the topic of addressing heterogeneity in unstructured overlays—both on the service and on the membership layer. Finally, we closed the chapter with a discussion on the main challenges that we expect to face in the following work.

In conclusion, the available tools to tackle heterogeneity in unstructured overlays do not suffice our goal—none of them directly adapts the number of neighbors of a node according to its capacity relative to the network. We believe that we can, therefore, contribute to optimizing heterogeneous networks by developing this previously unattempted solution.

In the following chapter, we detail the specifications of the work that is going to be conducted, including the challenges that we expect to face, our plans to experiment and evaluate the produced work, and the schedule to achieve these tasks.

Chapter 3

Future Work

As mentioned in Chapter 1, we plan to address the problem of resource heterogeneity in edge computing by optimizing unstructured overlays. Therefore, in this work, we are making three different contributions: i) the design and implementation of a novel overlay; ii) the design and implementation of a set of support subsystems (i.e., gossip protocols) that effectively exploit the benefits of the proposed overlay; iii) an experimental comparison of our proposal with state of the art decentralized solutions.

In Section 3.1 of this chapter, we start by describing the overlay to be designed and implemented, followed by the challenges that we expect to encounter in this process. We then close this first section by explaining the challenge of building an adequate gossip solution that will get the best out of the proposed overlay. In Section 3.2, we describe our plans to evaluate the implemented solution and compare it with existing ones. Finally, in Section 3.3, we establish the estimated schedule to accomplish the proposed work.

3.1 Proposed Solution

For the reasons mentioned in the beginning of Section 2.5.2 from the previous chapter, we want to address the resource heterogeneity problem in the membership layer—by directly manipulating the overlay. This dissertation focuses on doing so by manipulating the number of neighbors of each node according to its resources. Until now, we have found no solutions that did this. The closest approximation to this idea would be the use of super-peers [2] or a protocol like Biased Layered Tree [4], knowing that both of them do not explicitly change the number of neighbors of each node in order to tackle heterogeneity.

However, so that we can address the heterogeneity problem by manipulating the number of neighbors that each node has, we first have to conquer three obstacles. These

challenges are described below.

3.1.1 Inferring a Node's Relative Capacity

The first challenge is on how to infer the capacity of a node in relation to the other nodes in the network, i.e., how to know how powerful a node is if each node has only a partial view of the full system.

There are a lot of solutions that can be explored in this context. One of them is the dissemination of node resources on the network so that, if the network ends up mildly-stabilizing, the nodes will have a big-enough panoply of node resources that can take into account to know "where" in the resource-scale they are. However, this leads to two problems: i) if there is a high-enough churn, this might not work because nodes will always possess old data about the resources of the network; ii) it is not possible to know exactly when a node has enough information to infer its position in the resource-spectrum of the network, since it cannot know the exact number of nodes in the network at all times. Although not trivial, one possible way to tackle these issues is to infer an approximation to the total number of nodes in the network based on the observed churn variation.

A protocol akin to Vivaldi [5] might also help us in the endeavor of inferring a node's relative capacity (in relation to the other nodes). Vivaldi is a simple, decentralized, lightweight algorithm that assigns synthetic coordinates to hosts such that the distance between the hosts' coordinates accurately predicts the latency between them. As it is decentralized, Vivaldi requires no fixed network infrastructure and/or distinguished hosts to operate. Hence, Vivaldi allows predicting latency to other nodes without having to first contact them. We can use an approach inspired by Vivaldi's operation to map node resources so that we know the resources of other nodes without having to first contact them.

3.1.2 Adjusting the Number of Neighbors of a Node

Another key challenge that we are going to face is the fact that blindly changing the number of neighbors of a node will likely negatively affect the topological proprieties of the overlay. For example, if very resourceful nodes end up with a lot of neighbors and their neighbors do not have many neighbors themselves, we are augmenting clustering, which might have a negative impact on the robustness of the overlay, increasing the possibilities of partitions. One possible solution to tackle the problem of clustering in this example would be for every node to be aware of the neighbors of their neighbors. That way, a node could use the knowledge about the neighbors of its neighbors to restrict some neighboring relationships as to minimize the impact on clustering, and hence, protect global connectivity.

Besides that, we have to find a way of manipulating the neighbors without lowering the reliability of the epidemic broadcast (or other gossip-based solutions) operating on

the service layer. A simple solution to that would be to set a minimum amount of neighbors for each node (e.g., $log(n)$, n being the order of magnitude of the number of nodes expected to be in the system). Although simple and reliable, that might not be the most effective solution (i.e., the one that provides the overall best performance condition for services operating on top of the membership layer).

3.1.3 Building the Gossip Protocol

Lastly, the third challenge that we are faced with is—after having an overlay optimized for heterogeneity—building an adequate epidemic broadcast-like solution that will get the best out of our overlay. Since our goal is to make an efficient heterogeneity-friendly overlay that will be used for message dissemination through gossip, we need to implement a gossip-based solution that will illustrate the benefits of our overlay.

One possible solution would be to build a gossip algorithm inspired on adaptive gossip [3] (explained in Section 2.5.1.1) where the *split* function would depend on the number of neighbors of each node, i.e., the communication mode used would depend on the number of neighbors of the node broadcasting the message. This solution could be a starting point to build the gossip protocol to be used with the devised overlay.

3.2 Evaluation

To be able to show that our solution works as desired, we need to test it and compare it with other alternatives. Therefore, we are going to perform an experimental comparison of our proposal with state of the art decentralized solutions by combining both simulation (in a platform similar to PeerSim [28], to observe the performance of the solution in huge scale) and prototype deployments in realistic distributed infrastructures.

The experimental work will be partitioned into two phases. In the first phase, we will evaluate the implemented overlay, comparing it with other optimized-for-heterogeneity overlays. In the second phase, we will evaluate our full proposal (i.e., the implemented gossip solution(s) optimized for heterogeneity on top of the previously tested overlay), comparing it with other gossip algorithms optimized for heterogeneity.

3.2.1 Overlay Evaluation

In this phase, we are going to evaluate the devised overlay, comparing it with robust unstructured overlays and other state-of-the-art overlays optimized for heterogeneity, like Biased Layered Tree [4] and overlays based on super-peers [2]. To do so, we are going to evaluate and compare the overlays based on the overlay proprieties explained in Section 2.4 of the previous chapter. These proprieties include: connectivity, accuracy, degree distribution, average path length, and clustering coefficient.

3.2.2 Full Proposal Evaluation

After evaluating and validating the proposed overlay, we can finally evaluate our full proposal (i.e., the devised gossip protocol(s) running on top of the devised overlay) and compare it with other gossip solutions optimized for heterogeneity, like Efficient Epidemic Multicast in Heterogeneous Networks [30] and Low Latency Probabilistic Broadcast in Wide Area Networks [29]. To perform such comparisons, we will use the performance metrics explained in Section 2.3.2.2 of the previous chapter. These metrics include: reliability, relative message redundancy, last delivery hop, and latency.

3.3 Schedule

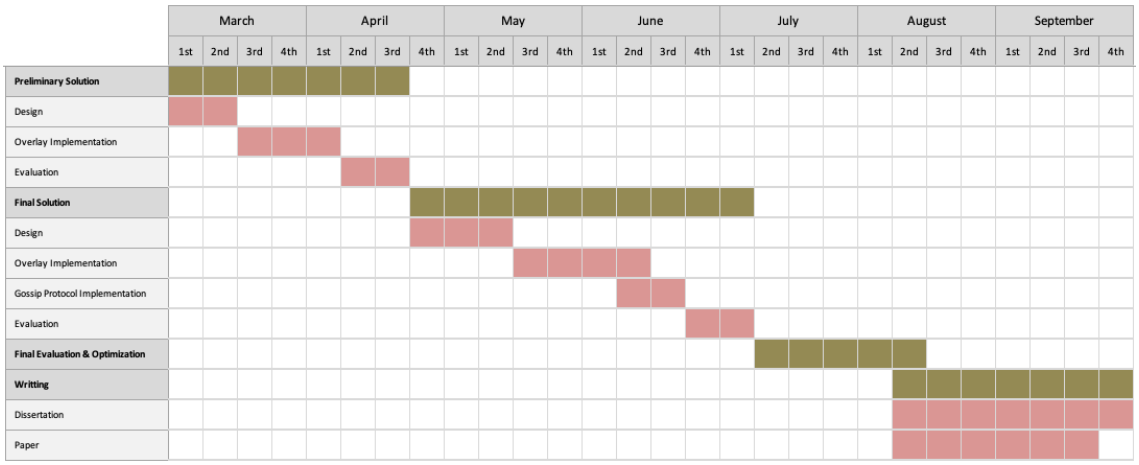


Figure 3.1: Gantt chart with the expected work schedule.

In Figure 3.1, it is presented the expected work schedule. It is divided into four main sections:

- **Preliminary Solution** Consists in the design, implementation and evaluation of a preliminary solution of an overlay;
- **Final Solution** Consists in the design, implementation and evaluation of an overlay solution and the gossip protocol(s) to be used with it;
- **Final Evaluation & Optimization** Consists in the incremental optimization and evaluation of the final solution, as described in Section 3.2;
- **Writing** Consists in the writing of the thesis, as well as an article to be submitted in a conference.

Bibliography

[1] J. Benet. *IPFS - Content Addressed, Versioned, P2P File System*. https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf. 2014. arXiv: 1407.3561 [cs.NI].

[2] B. Beverly Yang and H. Garcia-Molina. «Designing a super-peer network.» In: *Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405)*. 2003, pp. 49–60. DOI: 10.1109/ICDE.2003.1260781.

[3] N. Carvalho et al. «Emergent Structure in Unstructured Epidemic Multicast.» In: *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. 2007, pp. 481–490. DOI: 10.1109/DSN.2007.40.

[4] P. A. Costa, P. Fouto, and J. Leitao. «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.

[5] F. Dabek et al. «Vivaldi: A Decentralized Network Coordinate System.» In: *SIGCOMM Comput. Commun. Rev.* 34.4 (Aug. 2004), 15–26. ISSN: 0146-4833. DOI: 10.1145/1030194.1015471. URL: https://doi.org/10.1145/1030194.1015471.

[6] C. Diot et al. «Deployment issues for the IP multicast service and architecture.» In: *IEEE Network* 14.1 (2000), pp. 78–88. DOI: 10.1109/65.819174.

[7] P. T. Eugster et al. «Epidemic information dissemination in distributed systems.» In: *Computer* 37.5 (2004), pp. 60–67. DOI: 10.1109/MC.2004.1297243.

[8] M. J. Freedman et al. «Geographic locality of IP prefixes.» In: *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. 2005, pp. 13–13.

[9] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. «Scamp: Peer-to-Peer Lightweight Membership Service for Large-Scale Group Communication.» In: *Networked Group Communication*. Ed. by J. Crowcroft and M. Hofmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 44–55. ISBN: 978-3-540-45546-2.

[10] R. M. (IBM). *How cloud, fog, and mist computing can work together*. https://developer.ibm.com/technologies/iot/articles/how-cloud-fog-and-mist-computing-can-work-together. 2018.

[11] M. Jelasity and O. Babaoglu. «T-Man: Gossip-Based Overlay Topology Management.» In: *Engineering Self-Organising Systems*. Ed. by S. A. Brueckner et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–15. ISBN: 978-3-540-33352-4.

[12] M. F. Kaashoek, A. S. Tanenbaum, and S. F. Hummel. «An Efficient Reliable Broadcast Protocol.» In: *SIGOPS Oper. Syst. Rev.* 23.4 (Oct. 1989), 5–19. ISSN: 0163-5980. DOI: 10.1145/70730.70732. URL: https://doi.org/10.1145/70730.70732.

[13] A. Kermarrec, L. Massoulie, and A. J. Ganesh. «Probabilistic reliable dissemination in large-scale systems.» In: *IEEE Transactions on Parallel and Distributed Systems* 14.3 (2003), pp. 248–258. DOI: 10.1109/TPDS.2003.1189583.

[14] P. Labs. *Filecoin: A Decentralized Storage Network*. https://filecoin.io/filecoin.pdf. 2017.

[15] P. Labs. *libp2p*. https://libp2p.io.

[16] P. Labs. *Protocol Labs Website*. https://protocol.ai/about. 2021.

[17] J. Leitao, J. Pereira, and L. Rodrigues. «Epidemic Broadcast Trees.» In: *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*. 2007, pp. 301–310. DOI: 10.1109/SRDS.2007.27.

[18] J. Leitao, J. Pereira, and L. Rodrigues. «HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast.» In: *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. 2007, pp. 419–429. DOI: 10.1109/DSN.2007.56.

[19] 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.

[20] J. Leitão. *PhD thesis, Topology Management for Unstructured Overlay Networks*. Tech. rep. Technical University of Lisbon, 2009.

[21] J. Leitão, J. Pereira, and L. Rodrigues. «Gossip-Based Broadcast.» In: Oct. 2010, pp. 831–860. DOI: 10.1007/978-0-387-09751-0_29.

[22] J. Leitão et al. *Towards Enabling Novel Edge-Enabled Applications*. 2019. arXiv: 1805.06989 [cs.DC].

[23] J. Liang and K. Nahrstedt. *RandPeer: Membership management for QoS sensitive peer to peer applications*. Tech. rep. 2005.

[24] Q. Lv et al. «Search and Replication in Unstructured Peer-to-Peer Networks.» In: *Proceedings of the 16th International Conference on Supercomputing*. ICS '02. New York, NY, USA: Association for Computing Machinery, 2002, 84–95. ISBN: 1581134835. DOI: 10.1145/514191.514206. URL: https://doi.org/10.1145/514191.514206.

[25] R. Mahmud, R. Kotagiri, and R. Buyya. «Fog Computing: A Taxonomy, Survey and Future Directions.» In: *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*. Ed. by B. Di Martino et al. Singapore: Springer Singapore, 2018, pp. 103–130. ISBN: 978-981-10-5861-5. DOI: 10.1007/978-981-10-5861-5_5. URL: https://doi.org/10.1007/978-981-10-5861-5_5.

[26] 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.

[27] R. Melamed and I. Keidar. «Araneola: a scalable reliable multicast system for dynamic environments.» In: *Third IEEE International Symposium on Network Computing and Applications, 2004. (NCA 2004). Proceedings.* 2004, pp. 5–14. DOI: 10.1109/NCA.2004.1347755.

[28] A. Montresor and M. Jelasity. «PeerSim: A scalable P2P simulator.» In: *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*. 2009, pp. 99–100. DOI: 10.1109/P2P.2009.5284506.

[29] J. Pereira et al. «Low latency probabilistic broadcast in wide area networks.» In: *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004.* 2004, pp. 299–308. DOI: 10.1109/RELDIS.2004.1353030.

[30] J. Pereira, R. Oliveira, and L. Rodrigues. «Efficient Epidemic Multicast in Heterogeneous Networks.» In: *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*. Ed. by R. Meersman, Z. Tari, and P. Herrero. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1520–1529. ISBN: 978-3-540-48276-5.

[31] K. Petersen et al. «Bayou: Replicated Database Services for World-Wide Applications.» In: *Proceedings of the 7th Workshop on ACM SIGOPS European Workshop: Systems Support for Worldwide Applications*. EW 7. New York, NY, USA: Association for Computing Machinery, 1996, 275–280. ISBN: 9781450373395. DOI: 10.1145/504450.504497. URL: https://doi.org/10.1145/504450.504497.

[32] R. Rodrigues and P. Druschel. «Peer-to-peer systems.» In: *Communications of the ACM* 53.10 (2010), pp. 72–82.

[33] W. Shi et al. «Edge Computing: Vision and Challenges.» In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. DOI: 10.1109/JIOT.2016.2579198.

[34] I. Stoica et al. «Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications.» In: *SIGCOMM Comput. Commun. Rev.* 31.4 (Aug. 2001), 149–160. ISSN: 0146-4833. DOI: 10.1145/964723.383071. URL: https://doi.org/10.1145/964723.383071.

[35] C. Tang, R. N. Chang, and C. Ward. «GoCast: gossip-enhanced overlay multicast for fast and dependable group communication.» In: *2005 International Conference on Dependable Systems and Networks (DSN'05)*. 2005, pp. 140–149. DOI: 10.1109/DSN.2005.52.

[36] S. Voulgaris, D. Gavidia, and M. van Steen. «CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays.» In: *Journal of Network and Systems Management* 13.2 (2005), pp. 197–217. ISSN: 1573-7705. DOI: 10.1007/s10922-005-4441-x. URL: https://doi.org/10.1007/s10922-005-4441-x.

[37] S. Wilkinson et al. *Storj A Peer-to-Peer Cloud Storage Network*. 2014.

[38] B Yang and H. Garcia-Molina. «Comparing Hybrid Peer-to-Peer Systems.» In: *27th International Conference on Very Large Data Bases (VLDB 2001)*. This is a shortened version; see the extended version for full details. 2001. URL: http://ilpubs.stanford.edu:8090/727/.