NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
NAME OF THE DEPARTMENT

DIOGO ALEXANDRE DE JESUS FONA

BSc in Computer Science

LARGE-SCALE DECENTRALISED APPLICATION-LEVEL STREAMING

Dissertation Plan
MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
February, 2025

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
NAME OF THE DEPARTMENT

LARGE-SCALE DECENTRALISED APPLICATION-LEVEL STREAMING

DIOGO ALEXANDRE DE JESUS FONA

BSc in Computer Science

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

Examination Committee

Dissertation Plan
MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
February, 2025

Abstract

Decentralized systems are experiencing a resurgence in popularity, driven by the Web 3.0 movement's vision of a more democratic Internet and the increasing demand for scalable solutions beyond traditional cloud infrastructures. Streaming services, a dominant form of modern digital content consumption, rely heavily on centralized infrastructures such as Content Delivery Networks (CDNs). However, the growing need for scalable, cost-efficient, and resilient streaming solutions has renewed interest in decentralized architectures, particularly those based on peer-to-peer (P2P) networks.

This work explores the design and implementation of large-scale decentralized application-level streaming systems, focusing on how data is disseminated from a single source to multiple receivers under real-time constraints. Most widely used streaming solutions today rely on centralized architectures, which require substantial infrastructure investments and create single points of failure. While IP Multicast was introduced decades ago, and could be used as a building block for this type of system, it has not seen widespread deployment across the Internet due to scalability and adoption challenges. As a result, application-layer approaches have emerged as the primary alternative, leveraging end-user devices to distribute content without relying on dedicated infrastructure. We provide an overview of several existing streaming systems, outlining their design and the different approaches used to address the challenges of application level streaming. This serves as a foundation for understanding the strengths and limitations of current solutions. Based on this, we propose a novel peer-to-peer streaming system that will strive to be scalable, take node heterogeneity into account, and support multiple concurrent streams.

To validate the proposed solution, we plan on conducting experimental analysis to benchmark it against existing state of the art solutions, considering performance, resource consumption, and robustness under different conditions.

Keywords: Decentralized Systems, Peer-to-Peer Streaming Systems, Application Level Streaming

Resumo

Os sistemas descentralizados estão a registar um ressurgimento em popularidade, impulsionados pela visão da Web 3.0 de uma Internet mais democrática e da crescente procura de soluções escaláveis para além das infraestruturas de cloud tradicionais. Os serviços de streaming, uma forma dominante do consumo moderno de conteúdos digitais, dependem fortemente de infraestruturas centralizadas, como Rede de fornecimento de conteúdo (CDN). No entanto, a crescente necessidade de soluções de streaming escaláveis, economicamente eficientes e resilientes renovaram o interesse em arquiteturas descentralizadas, particularmente as baseadas em sistemas entre-pares (P2P).

Este trabalho explora o design e a implementação de aplicações de transmissão descentralizadas em larga escala, com foco na forma como os dados são disseminados de uma única fonte para múltiplos receptores sob restrições de tempo real. As soluções de streaming mais utilizadas atualmente dependem de arquiteturas centralizadas, que exigem investimentos substanciais em infraestruturas e criam pontos únicos de falha. Embora o IP Multicast tenha sido introduzido há décadas, e poderia ser utilizado como bloco de construção para este tipo de sistema, este não teve grande implantação na Internet devido a desafios de escalabilidade e adoção. Devido a isto, abordagens de camada de aplicação surgiram como a principal alternativa, utilizando dispositivos dos utilizadores para distribuir conteúdo sem depender de infraestrutura dedicada. Nós fornecemos uma visão geral de vários sistemas de streaming existentes, descrevendo o seu design e as diferentes abordagens utilizadas para abordar os desafios do streaming ao nível da aplicação. Isto serve como um base para compreender os pontos fortes e as limitações das soluções atuais. Tendo os sistemas anteriores em mente, propomos um novo sistema de streaming peer-to-peer que se esforçará por ser escalável, ter em conta a heterogeneidade dos nós e suportar vários fluxos simultâneos.

Para validar a solução proposta, planeamos conduzir análises experimentais para compará-lo com as soluções de última geração existentes, considerando o desempenho, as características consumo e robustez sob diferentes condições.

Palavras-chave: Sistemas Descentralizados, Sistemas de Transmissão Entre-Pares, Transmissão a nivel de Aplicacao

Contents

List of Figures

1 Introduction

Context

Over the past decade, streaming services have emerged as one of the dominant forms of digital content consumption. The scale of this transformation is evident in current internet traffic patterns, with on-demand streaming accounting for approximately 54% of downstream traffic and live streaming contributing an additional 14%[1]. This massive volume of streaming traffic is primarily handled by a few dominant platforms such as YouTube, Netflix, Facebook, Twitch, TikTok among others[1].

The current content delivery model relies on centralized infrastructures, requiring extensive resources through globally distributed Content Delivery Networks (CDNs) and data centers[2][17][51]. The substantial costs and complexity of maintaining such infrastructure have led most content providers to rely on established platforms and their delivery networks, as building and operating private delivery infrastructure at scale would be prohibitively expensive.

IP Multicast[29] is an alternative approach that was proposed decades ago as a potential solution for efficient content distribution. However, it has not achieved widespread deployment across the internet[19]. This limitation, combined with the need for internet-wide multicast solutions, has led to the development of application layer multicast protocols, such as [5][59][10][20], and other decentralized approaches.

Decentralized systems, which gained initial popularity through peer-to-peer systems at the start of the millennium[55], have recently seen renewed interest with the emergence of the Web 3.0[32] movement and its vision of a more democratic Internet. This resurgence, combined with the growing need for systems to scale beyond traditional cloud data centers, has revitalized research into decentralized architectures.

These systems offer an alternative to the centralized model by leveraging unused compute and bandwidth resources at the edge of the Internet[55]. By harnessing the collective power of consumer devices, decentralized systems have the potential to scale efficiently at a lower cost than traditional centralized infrastructures. This approach aligns with broader technological trends, such as edge computing, which aim to move computation and content delivery closer to end users.

The promise of decentralized streaming extends beyond cost efficiency. By distributing content delivery across participating nodes, these systems can potentially create a more resilient streaming ecosystem while reducing dependency on centralized infrastructure providers, making such platforms increasingly accessible for different content creators and hence, more democratic.

Therefore, the main problem this work addresses is the efficient and scalable delivery of real-time streaming content using a decentralized peer-to-peer architecture. This requires overcoming challenges in managing node heterogeneity, ensuring system robustness within a dynamic peer environment, and meeting possibly strict latency demands of streaming applications. Ultimately, this work aims to demonstrate the feasibility and benefits of a decentralized P2P approach as a compelling alternative to traditional centralized streaming infrastructures.

Expected Contributions

The expected contributions of this dissertation are as follows:

- The implementation of existing peer-to-peer streaming protocols to establish a baseline understanding of current approaches.
- The development and implementation of a new decentralized live streaming protocol that will strive to be scalable, take node heterogeneity into account, and support multiple concurrent streams.
- The experimental evaluation of the proposed protocol and existing implementations across different scenarios.

Research Context

This work is part of the ongoing research efforts of the EU project TaRDIS[52][53] (Trustworthy and Resilient Decentralised Intelligence for Edge Systems). TaRDIS aims to simplify the development of correct and efficient heterogeneous swarms by reducing complexity and effort. It accomplishes this through a novel programming paradigm and a comprehensive toolbox designed to support the development and execution of applications in swarm and decentralised distributed systems.

Document Structure

The rest of this document is organized as follows:

Chapter 2 This chapter introduces background concepts and prior work relevant to decentralized livestreaming in peer-to-peer (P2P) systems. It starts with an overview of distributed systems followed by P2P systems, comparing centralized and decentralized approaches. Next, it covers different types of overlay networks. Then, it describes some of

services that can be offered by P2P systems and how they can be implemented over the different overlays. Finally, it presents peer-to-peer livestraming and some of the existing systems that implement it.

Chapter 3 This chapter presents an overview of the proposed work and a rough schedule for its development.

2

Related Work

This chapter reviews related work that provides the foundation for the proposed solution. We begin by introducing peer-to-peer systems as an alternative to centralized systems in Section 2.1. Sections 2.2 and 2.3 offer an overview of centralized and decentralized peer-to-peer systems, along with various overlay networks. In Section 2.4, we explore several peer-to-peer services relevant to our work, such as resource location, which can be used to find data related to streams, and services like multicast and live streaming for data dissemination.

2.1 Peer to Peer systems

A distributed system is a system of computers, referred to as nodes, connected by a communication network that exchange information by sending messages[54].

One of the most prevalent models used in distributed systems is the client-server model. In this model, one node acts as a client that initiates requests for content or services to another node, the server. Systems following this model offer certain advantages, such as being easier to develop, understand, and manage. Furthermore, since the servers providing a particular service are usually administered by a single entity, these systems are relatively straightforward to control. However, this centralization is a double-edged sword. While it simplifies control, it also makes the system susceptible to censorship, as administrators can remove content or selectively deny access.

An alternative to the client-server model is the peer-to-peer (P2P) model. In a P2P system, nodes — referred to as peers — collaborate directly to perform tasks and share resources such as storage, computational power, bandwidth, or data. Peers in these networks are equally privileged participants and can assume the same roles within the system[55][43].

P2P networks typically consist of peers located at the edges of the Internet, often on personal devices such as laptops, desktops, or mobile phones. By pooling their resources, these devices collectively form a distributed system that can scale dynamically with the number of participants. This shared-resource model improves efficiency and

scalability, as the network's capacity grows with the addition of new peers. In contrast, traditional client-server systems are constrained by the server's resources. Scaling a client-server system often involves significant costs, as server owners must invest in better hardware while leaving a considerable amount of edge resources—those available at user devices—untapped.

Peer-to-peer systems can generally be categorized into two types: **centralized** and **decentralized**[55].

2.2 Centralized Peer to Peer systems

In a **centralized peer-to-peer system**, one or more central servers manage critical functions such as maintaining an index to facilitate content discovery or providing a directory service that oversees the network's topology and coordinates connections between peers[55]. While the actual workload is still distributed among the peers, the central component simplifies certain aspects of the system, reducing the complexity of network management. However, this centralization introduces a single point of failure and can become a bottleneck as the system scales. Two popular systems that followed this centralized approach were Napster and early BitTorrent.

Napster One of the earliest and most popular peer-to-peer systems was Napster, a file-sharing program primarily used for distributing MP3 music files. At its peak in February 2001, Napster boasted 26.4 million users [23] and was responsible for up to 60% of the bandwidth usage in certain university networks [49].

Napster functioned by requiring users to connect to a central server using their accounts. Each user could host MP3 files on their personal computer and inform the central server about the files they were sharing. The server itself did not store the files but maintained an index of which files were available on which peers. Users could query the server to locate other peers hosting a desired file. Upon receiving a query, the server would return the addresses of the peers who had the file. The actual file transfer occurred directly between users, bypassing the central server. This design ensured that the bulk of bandwidth usage and storage was distributed across the user base.

Despite not hosting any files directly, Napster facilitated the exchange of copyrighted material, leading to numerous legal challenges from the music industry. These challenges culminated in Napster being forced to shut down in July 2001. Its closure prompted a significant migration of users to decentralized peer-to-peer systems, which were more resistant to centralized control and shutdowns.

BitTorrent BitTorrent [8] is a peer-to-peer (P2P) file-sharing protocol that was first release in 2001 [6]. The protocol quickly grew in popularity and was estimated to account for approximately 25% of all internet traffic in 2004 [7]. While its prominence has diminished with the rise of social media, video streaming, and cloud storage platforms, BitTorrent

remains significant, currently representing around 4% of upstream internet traffic in 2024 [1].

Unlike Napster, BitTorrent does not provide search functionality. Instead, users must obtain a "torrent" file, a metadata container that includes information such as the list of files, cryptographic checksums, and addresses of one or more trackers. To initiate a download, a peer connects to a tracker to request a list of other peers currently sharing the same torrent file. Once connected, the peer joins the "swarm", the overlay network formed by participants sharing a specific torrent, and begins simultaneously downloading and uploading file fragments.

Early implementations of BitTorrent were centralized, altough not quite to the same degree as Napster, as the protocol relied heavily on tracker servers. A tracker's failure could effectively prevent file sharing by eliminating the mechanism for peer discovery. This limitation has since been address with the introduction of two extensions: Distributed Hash Table (DHT) [18] and Peer Exchange (PEX) [39].

2.3 Decentralized Peer to Peer systems

In contrast to a **centralized peer-to-peer system**, a **decentralized peer-to-peer system** eliminates the need for a central server. Instead, peers organize themselves into an overlay network, collaboratively handling the services previously provided by the centralized component, like content discovery and peer coordination. This distributed responsibility makes the system more robust to failures and often allows for better scalability by distributing the workload more evenly across all participants instead of relying on the capacity of the central component to scale vertically. However this lack of a central component can also introduce extra challenges related to network maintenance and data consistency.

2.3.1 Overlay networks

In decentralized peer-to-peer systems, the absence of a central server forces the peers to directly connect to each other and communicate in order to distribute the workload and provide the services that were previously the responsibility of that central server. The connections maintained between the peers in the system form an overlay network.

An overlay network is a virtual (or logical) network built on top of an existing underlying network (the underlay network). The logical connections between members of the network can span multiple hops on the underlying network. In peer-to-peer systems, overlay networks are commonly implemented at the application layer, using protocols such as UDP or TCP.

One important aspect in a peer-to-peer systems is membership tracking, that is, knowing which peers are currently participating in the system. While that could be done entirely by a central server, when one does not exist this responsibility can be taken over by the overlay network implementation. This can usually be handled in one of two ways:

Full membership tracking: In a system with full membership tracking, each peer keeps track of the entire system's membership. This approach can simplify certain aspects of the system's design and is better suited for systems designed for a relatively small or medium number of participants, such as Narada[12] and Dynamo[16]. However, we will not focus on this type of system, as this design represents a significant limitation when scaling to a large number of participants. The overhead involved in having each peer maintain a complete and consistent membership in the face of churn (the changes in membership caused by the arrival and departure of peers [45]) can be significant, limiting scalability and performance.

Partial membership tracking: In contrast, partial membership tracking allows each peer to maintain information about only a small subset of other peers, with the number of tracked peers typically scaling logarithmically with the system size [44][42][62][35][37]. This reduces the amount of information each peer must store, improving scalability and making the system more resilient to churn. However, the protocol must be designed to avoid network partitions and ensure that the system can maintain connectivity even in the face of peer failures or churn.

When it comes to network topology, overlay networks can usually be categorized in one of two ways: **structured** and **unstructured**[46]. This reflects fundamental aspects related with the network's topology, performance, and the implementation of different distributed services.

2.3.2 Unstructured Overlay Networks

Unstructured overlay networks, also known as random overlays, are a type of overlay that does not impose predetermined restrictions on network topology. Nodes can establish connections and select neighbors flexibly, without adhering to strict organizational rules.

This approach allows for more dynamic network formation, resulting in reduced overhead compared to structured networks. The lack of mandatory topology maintenance makes these networks inherently more resilient to node churn, as connections can be established and broken with minimal disruption.

2.3.2.1 HyParView

HyParView[35] is a membership protocol for reliable gossip-based broadcast that aims to maintain high reliability in the face of high rates of node failure. The protocol maintains an unstructured overlay network, keeping two partial views per peer, an active view and a passive view.

The active view is a small partial view where the links between peers are symmetrical and TCP connections are kept open providing reliable communication and failure detection. This view is maintained using a reactive strategy. When a peer is considered to have failed an attempt is made to replace it with another one from the passive view.

The passive view is a larger partial view but unlinke the active view it is not symmetrical and no TCP connections are maintained. This view is maintained using a cyclic strategy. Periodically each peer generated a list of random peers, called an exchange list, from its passive view and active view. This list is then sent to a random peer in the network, found trough a random walk, that will in turn send back an exchange list of its own. This list is then integrated into the passive view, removing existing nodes if necessary.

2.3.2.2 Freenet

Freenet is a decentralized peer-to-peer network designed to enable the anonymous publication, replication, and retrieval of data while protecting the privacy of both publishers and consumers [13, 14]. It can be thought of as a cooperative distributed filesystem that incorporates location independence and transparent lazy replication.

Freenet does not guarantee permanent file storage, files with fewer requests or lower usage are removed to free up space when required. However, if there is sufficient participation and storage contributions from nodes, the network could achieve enough aggregate capacity to retain most files indefinitely. Each node contributes a designated amount of storage space for use by the network, allowing distributed file management and storage.

Retrieving Data To retrieve a file, an identifier is generated by hashing a descriptive string or the file's content. Unlike distributed hash tables (DHTs), Freenet does not directly map keys to specific nodes. If the requested file is available locally, the node serves it directly. Otherwise, the node consults its routing table for the closest matching key (in lexicographical order) and forwards the request to that node.

If a forwarding attempt would result in a loop, the next closest key in the routing table is tried. Unique request identifiers in the messages are used to avoid and detect loops. Requests propagate through the network up to a pre-configured hop limit. If the request succeeds, the content is returned to the original requester, and the nodes along the path cache the file under the associated key. This caching mechanism improves data availability and reduces latency for future requests.

The routing table is dynamically enriched with information from successfully forwarded requests.

Inserting Data Data insertion follows a similar process. An identifier for the data is first generated, and the insertion request is forwarded to the node in the routing table with the closest matching identifier. This node continues the process, forwarding the request to the next closest match up to a predefined hop limit. If no identifier collisions occur along the path, all nodes involved in the forwarding process store the file, making it accessible to the network.

2.3.3 Structured Overlay Networks

Structured overlay networks are characterized by striving to maintain a network topology. Unlike unstructured networks, these overlays impose specific organizational rules that govern how nodes interconnect and communicate.

The primary advantage of structured networks lies in their ability to provide efficient services such as precise content lookup and message routing. However, this comes at the cost of increased maintenance overhead. Nodes must continuously invest computational resources to maintain a valid network topology in the face of churn.

Distributed Hash Tables (DHTs) are the most common type of structured overlay and are the one we will focus on. In a DHT, nodes operate within a defined identifier space, typically a large numerical range such as a 256-bit hash space. Each peer receives a unique identifier, which may be derived randomly or generated from identifying node information like IP address and port, or public key.

Peers maintain contact information about other network participants, in what is usually called a routing table, typically preserving connections to nodes with identifiers at exponentially increasing distances. The specific distance between nodes is calculated using an implementation-specific metric that compares their unique identifiers.

When routing a message to a specific peer, it is progressively forwarded to the closest known peer at each network hop. Under optimal conditions, this routing strategy allows message delivery in approximately log(N) steps, where N represents the total number of nodes in the network.

Popular examples of DHT implementations include Chord [44], Pastry [42], and Kademlia [37].

2.3.3.1 Chord

Chord[44] is a distributed hash table (DHT) protocol distinguished by its simplicity, provable performance, and correctness. It provides a fundamental operation: mapping a given key to a specific network node within a circular identifier space.

The protocol leverages a hash function (such as SHA-1) to transform node addresses and keys into fixed-length identifiers. Each node's identifier is generated by hashing its IP address, while keys are hashed to determine their corresponding location in the identifier space.

Chord introduces two node relationships: the successor (the node with the lowest identifier following a given node's identifier in the identifier space) and the predecessor (the node with the lowest identifier preceding a given node's identifier in the identifier space).

For a network of N nodes, Chord only requires that each node to correctly maintain its successor in order to function correctly. To enable message routing in approximately log(N) steps and accelerate lookups, each node additionally maintains a "finger table".

This table keeps track of references to nodes at exponentially increasing distances (the i^{th} entry contains the identifer and address information of $successor(n + 2^{i-1})$.)

Each node also keeps information about its predecessor to facilitate network operations like joins and departures.

A key is assigned to the node that is the successor of its identifier in the circular address space. To locate the node responsible for a specific key, a node iteratively traverses the finger tables of other nodes through remote procedure calls, converging on the key's successor.

Node churn and failures can change the key responsibilities attributed to each node, prompting the protocol to notify the upper software layer of such transitions. Since Chord only handles key-to-node mapping any subsequent data transfer must be managed by the overlying application layer.

To maintain state integrity, the protocol periodically executes a stabilization procedure that identifies and corrects potential inconsistencies in the network topology.

2.3.3.2 Kademlia

Kademlia is a peer to peer distributed hash table. The identifier space is a 160 bit string (ex: output of sha1 function), altough implementations can change this (libp2p's kademlia uses 256 bit strings computed from the sha256 function [36]). Nodes and keys are assigned a 160 bit identifer and the distance between two identifiers is defined as the XOR between them ($x \oplus y$, for identifiers x and y).

For an identifier space of m-bit fixed size strings, each peer maintains a list of up to m "k-buckets". The i^{th} k-bucket, for $0 \leq i < m$, contains k, a protocol parameter, entries where each entry has the identifier and contact information of another node whose identifier shares i common prefix bits with current node. These buckets are populated during normal protocol procedures. When a node is contacted by another one it tries to add or update its entry in the corresponding k-bucket. This way there is no need for extra communication specifically to maintan the network topology.

Kademlia's protocol consists of four RPCs:

PING The PING RPC is used as a liveness probe for nodes stored in k-buckets. This allows nodes to evict unresponsive nodes and free space for new ones.

FIND_NODE The FIND_NODE RPC receives as argument a nodes identifier and requests the target node to provide contact information for the k closest nodes, to the argument identifier, that it knows of.

FIND_VALUE The FIND_VALUE RPC is similar to the FIND_NODE RPC. It receives a key as argument and requests the target node value associated with that key. If the target node does not have the value it will instead return the k closest nodes to the key.

STORE The STORE RPC is used to store a key-value pair on a given node.

Lookup procedures leverage these RPCs through an iterative approach. When searching for a node or value, the protocol progressively queries the k closest known nodes until no closer candidates are found. These lookups can be executed in parallel, by making parallel requests to a few of the closest known nodes, improving performance and mittigating the effectof slow or unresponsive nodes.

For value storage, Kademlia replicates data across the k closest nodes, found using a lookup operation. This values have to be periodically republished to prevent situations where they would not be found from a query.

2.3.3.3 Pastry

Pastry is a scalable, distributed object location and routing substrate designed for wide-area peer-to-peer applications [42]. Each node in the Pastry network is assigned a unique identifier, called a *nodeId*, which is uniformly distributed across the identifier space. These identifiers are 128 bits long and can be either randomly generated or derived from a cryptographic hash function applied to identifying information, such as the node's IP address or public key.

In Pastry, identifiers are treated as numbers in base 2^b, where b is a system parameter. For instance, if $b = 4$, identifiers are represented as 32-digit hexadecimal numbers. The key routing service provided by Pastry is the ability to efficiently route messages to the node whose identifier is closest to a given target identifier, typically in $O(\log N)$ hops, where N is the number of nodes in the system.

Pastry makes use of a *proximity metric*, that is different from the simple numerical distance between identifiers. This metric measures how close two nodes are in terms of the underlying network topology, which could be based on the number of IP hops or network latency.

Each Pastry node maintains three primary structures to facilitate routing: the *routing table*, *neighborhood set*, and *leaf set*.

Routing Table The *routing table* is a table where each row corresponds to a specific common prefix length with the current node's identifier. Each entry in row r (for $0 \leq r < \lceil \log_{2^b}(N) \rceil - 1$) holds the contact information of a node whose identifier shares the first r digits with the current node's identifier.

Neighborhood Set The *neighborhood set* contains the contact information of M nodes, where M is a system parameter. These nodes are selected based on the *proximity metric*, rather than their numerical identifier distance, and represent the closest nodes to the current node in terms of that metric. The neighborhood set is primarily used to maintain locality properties within the network, rather than for routing purposes.

Leaf Set The *leaf set* consists of L entries, divided into two halves. One half contains nodes whose identifiers are numerically closest and smaller than the current node's identifier, while the other half contains nodes with identifiers numerically closest and larger than the current node's identifier. The leaf set helps to ensure that messages are routed to the node whose identifier is closest to the target identifier, especially when the target lies between the ranges of the leaf set.

Routing Procedure When a node wants to route a message to a target identifier I, it first checks whether I falls within the range of identifiers in its leaf set. If so, the message is forwarded to the node in the leaf set that is closest to I, which may be the current node itself.

If the target identifier I is not within the leaf set range, the message is routed using the *routing table*. The node will look for an entry in the routing table where the identifier shares at least as many common prefix digits with I as the current node's identifier and is numerically closest to I. If no such entry exists, the node will route the message to a node from the union of the routing table, leaf set, and neighborhood set that shares a common prefix with I and is numerically closer to it than the current node.

Joining the Network To join the Pastry network, a new node, say X, contacts a bootstrap node, say B, (obtained out-of-band) and requests B to route a *Join message* to X's identifier. As the message traverses the network, the nodes along the path send their state information back to X, which uses this data to initialize its own routing structures. Node failures are detected through communication timeouts, and affected nodes are replaced using a procedure that varies depending on which data structure (routing table, leaf set, or neighborhood set) the failed node was part of.

2.3.3.4 CAN: Content Addressable Network

The Content Addressable Network[40] (CAN) is a Distributed Hash Table (DHT) that uses a d-dimensional Cartesian coordinate space as its identifier space, unlike other DHTs that employ one-dimensional identifier spaces. In CAN, each node in the network is responsible for managing a "zone" in this d-dimensional space, where each key is mapped to a specific point in the coordinate space using a hash function. The key-value pairs are stored by the node whose zone contains the corresponding key. Nodes maintain contact information for their neighbors in adjacent zones, typically resulting in about $2d$ neighbors if the space is evenly split.

CAN provides basic DHT operations such as *insertion*, *lookup*, and *removal* of key-value pairs. These operations are performed by routing messages through the network to the appropriate node responsible for the target key.

Routing in CAN Routing a message in CAN involves forwarding it to the node whose zone is closest to the destination point P in the coordinate space. If the neighboring nodes

in the direction of P are unreachable (e.g., due to network failures), CAN uses an *expanding ring search*. This method gradually broadens the search area to find a node that is closer to the target point than the current node, allowing the message to continue towards its destination.

Node Joining Process To join the CAN network, a new node must first contact a *bootstrap node* (discovered out-of-band). The joining node randomly selects a point P in the coordinate space and asks the bootstrap node to forward the join request to the node responsible for the zone containing P. The node that owns the target zone will split its zone, assigning a portion to the new node. The new node will be informed of its assigned zone and its neighboring nodes, while the neighbors are also updated about the joining process.

To maintain network consistency, nodes periodically send *refresh messages* to their neighbors. These messages help propagate information about node joins and departures and act as failure detectors—if refresh messages stop being received, it may indicate node failure.

Node Departure When a node leaves the system, it must transfer its key-value pairs to a neighboring node. The node will hand over its zone to a neighboring node such that the union of their zones forms a valid zone. If no such neighbor exists, the node will transfer its zone to the neighbor with the smallest zone, resulting in that neighbor temporarily owning two zones. If a node fails abruptly, its key-value pairs are temporarily lost. Neighboring nodes will execute a *takeover procedure*: one of them will take over the failed node's zone, and the key-value pairs will be refreshed when the publisher re-inserts the data.

Improvements to CAN The basic functioning of CAN, as described above, can be improved in several ways, such as through *multiple realities*, *improved routing metrics* and *caching and replication*. The interested reader can find details on other proposed improvements in [40].

Multiple Realities In the *multiple realities* approach, the coordinate space is replicated across several *realities*. A reality is an independent instance of the coordinate space, and every node is present in each reality, although it occupies a different random point in each one. This technique improves the availability and reliability of data since key-value pairs are replicated across multiple realities. If nodes in one reality become unresponsive, messages can be routed through another reality, thus improving fault tolerance.

Moreover, the replication of key-value pairs across realities allows for more efficient routing. Since nodes in different realities occupy different points in the coordinate space, messages can be routed across large distances by transitioning between realities, further improving the overall routing efficiency.

Improved Routing Metrics By default, CAN routes messages to the node that is closest to the target point, based purely on the coordinate space's distance. However, this metric does not take into account the underlying network topology, such as the *Round Trip Time (RTT)* between nodes. One improvement to the routing process involves using a *progress-to-RTT ratio* as the routing metric. Instead of always routing the message to the node that makes the most progress towards the target point, a node may choose to route the message to the neighbor that provides the best balance between progress and network latency.

Caching and Replication To address the varying demand for key-value pairs in the network, two strategies, *caching* and *replication*, can be employed to reduce load on the nodes responsible for popular keys.

Caching: In the case of frequently requested keys, a node can maintain a local cache of recently accessed key-value pairs. When a request for a cached key arrives, the node can serve the request directly, avoiding the need to forward the request further through the network.

Replication: For key-value pairs experiencing high demand, replication can be used to distribute the load. A node that receives many requests for a particular key may replicate the key-value pair to its neighboring nodes. This increases the number of nodes that can serve the key, improving availability and load distribution. Nodes that hold replicated keys can either serve requests directly or forward them up to the primary node with a certain probability to ensure an even distribution of traffic and prevent overloading any single node.

Together, caching and replication help improve both performance and scalability by reducing bottlenecks and distributing requests more evenly across the network.

2.4 Peer to Peer Services

Figure 2.1 illustrates a four-layer network architecture consisting of the Application layer at the top, followed by the Services layer, then the Overlay Network, and finally the Underlay Network. Once an overlay network (the second layer) has been established, it can be used to provide different services. Moreover, the type of overlay can have a significant impact on the performance and simplicity of these services.

2.4.1 Resource Location

Resource location enables any peer in a system to locate a resource, such as another node or a file, that matches a specific query. This service is fundamental to many peer-to-peer applications, including file-sharing systems like Napster, Gnutella [48], and BitTorrent [18].

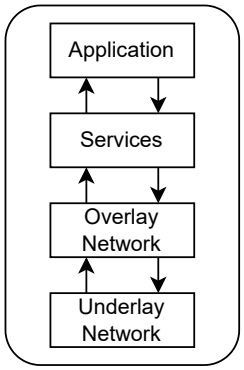

Figure 2.1: Application Layers

In unstructured overlays, resource discovery often relies on flooding techniques. These can be categorized as either *blind*, where nodes do not maintain any state about others, or *informed*, where nodes retain some state to improve search latency and efficiency[47][31]. Altough we only consider blind and informed approaches, [31] describes a few other categories. Unstructured overlays support more expressive query types, while structured overlays, such as Distributed Hash Tables (DHTs), excel at exact key lookups. Although DHTs are traditionally limited to exact matches, research, such as [26], has explored methods to support more complex queries over these structures .

2.4.1.1 Blind Methods in Unstructured Overlays

In blind resource location, nodes do not maintain any state regarding resources to guide the search process. Instead, this type of search relies entirely on flooding techniques to propagate queries across the network, attempting to reach as many nodes as possible in the hope of finding a match. However, simple flooding can quickly become prohibitively expensive in terms of network bandwidth and processing [41]. To address this issue, various approaches have been proposed [55], including:

Breadth-First Search In this method, a query is initially broadcasted to all neighbors with a time-to-live (TTL). Each recipient then decrements the TTL value and forwards the message to its subsequent neighbors. The search process continues recursively and terminates when the TTL reaches zero. If no results are found the query can be repeated with an increased TTL.

Depth-First Search In this method peers query their neighbors sequentially, sorted by their potential relevance to the search. Individual messages are sent with a depth parameter, and the sender waits for responses using a timeout mechanism, after which it will try the next neighbor. This approach allows the network to iterate through potential neighbor paths, giving priority to more promising routes for resource discovery.

Random Walk is a probabilistic approach to resource lookup. In this strategy, queries are forwarded through randomly selected neighbors, introducing an element of randomness to the search process. While this method significantly reduces network communication overhead, it comes with notable trade-offs. The random nature of the search can lead to increased search latency and carries a risk of incomplete result discovery. Despite these limitations, random walk provides a lightweight alternative to more computational or network intensive search techniques.

Gnutella Gnutella was one of the first decentralized decentralized peer-to-peer systems. Like Napster it was file-sharing system, that saw a spike in popularity after Napster's legal challenges. Unlike Napster's centralized architecture, Gnutella's decentralized design made it more resilient to shutdown attempts. This description focuses on Gnutella version 0.4[48].

Network Joining Peers enter the Gnutella network by obtaining contact information through "host cache servers", that maintain current network member addresses, or some other out of band method. Upon connecting to an initial contact, the peer sends a PING message that floods through the network until its time-to-live (TTL) expires. Receiving peers forward the PING and respond with a PONG message, which returns via the original PING's reverse path. This mechanism is used for both network discovery and neighbor liveness verification.

Content Discovery Content searches in Gnutella utilize a flooding mechanism with controlled propagation. A peer broadcasts a QUERY message containing a search string, limited by a TTL. Peers hosting matching content respond with a QUERYHIT message, routed back through the original query's reverse path. The QUERYHIT includes matching file details and the serving peer's connection information.

Connection Handling To address connectivity challenges like NAT or firewall restrictions, Gnutella uses the PUSH message. When a serving peer cannot accept direct connections, an interested peer can request that the serving peer establish a connection to it through a PUSH message routed via the QUERYHIT's reverse path, enabling direct file transfer.

2.4.1.2 Informed Methods in Unstructured Overlays

Informed methods for resource location in unstructured overlays leverage stored state in each node to guide the search process, such as indexing the content of neighboring nodes. While these methods can significantly reduce search latency, they may introduce substantial overhead for maintaining the state and risk overloading certain nodes[31].

Some informed resource location techniques are outlined below, however, this is by no means an exhaustive list. Surveys such as [47] and [31] highlight the vast array of methods available for resource discovery in unstructured overlays.

Ant Search Dynamic Query (DQ) [24] is an algorithm employed by Gnutella to estimate the Time-To-Live (TTL) of a query, aiming to reduce flooding while still yielding satisfactory results. Although DQ offers improvements, the issue of freeriders in Gnutella remains significant. A freerider, as defined in [57], is a peer that shares fewer than 100 files. Queries that encounter these peers are less likely to return useful results.

Ant Search [57] addresses this problem by tracking the recent success rates of its neighbors. When forwarding a query, a node sends it only to a configurable fraction of its top neighbors considering their success rates, thus increasing the likelihood of obtaining relevant results while reducing unnecessary traffic.

Local Indices Local Indices [56] is a method where nodes maintain index data for all other nodes within a radius of r hops, a parameter defined system-wide. A system policy dictates the depths at which queries should be processed, allowing a single node to respond on behalf of all other nodes within its radius. This approach significantly reduces the number of nodes that need to process a query, improving efficiency. However, it introduces additional overhead due to the maintenance of the local indices.

Directed BFS Directed BFS (DBFS) [56] improves the basic breadth-first search by having each node maintain statistics about the quality of its neighbors. When forwarding a query, a node selectively forwards it to a subset of its neighbors that it considers to be of the highest quality for that specific query. If neighbors are chosen appropriately, the quality of the results remains largely unaffected, while the overall query overhead is reduced.

2.4.1.3 Structured Overlays

Providing resource location over a DHT is usually implemented by taking advantage of the mapping between nodes and resource identifiers. A resource will first be assigned an (unique) identifier and this identifier is them mapped onto one or more nodes, in an implementation specific way, that will become responsible for that resource.

When dealing with file/data location, this identifier is usually obtained from hashing the file name, file attributes or file content using some hash function. Instead of storing entire content on the responsible peers, a commonly used approach is to maintain pointers or references to the content providers, such as IP and port or identifier. To prevent content loss from sudden peer departures, some systems implement periodic key republishing and optionally content replication across "nearby" peers.

Lookup performance in DHTs is typically much better then unstructured overlays when locating specific content. Unlike unstructured networks that rely on expensive broadcast methods, DHTs can locate specific content in logarithmic O(log N) network hops. This efficiency comes with a significant trade-off in query flexibility. While DHTs excel at exact key matching, they struggle with more complex search strategies, altough some work exists aimed at enabling more complex queries over DHTs[26].

Unstructured overlays, by contrast, support more expressive queries. They can easily implement fuzzy filename matching, attribute-based searches, and more flexible content discovery mechanisms.

BitTorrent's DHT BitTorrent's Distributed Hash Table (DHT) extension [18] provides a decentralized alternative to traditional tracker-based peer discovery, implementing a modified version of the Kademlia protocol previously described in Section 2.3.3.2.

Torrents are uniquely identified by an infohash—a SHA-1 hash derived from a specific section of the torrent file. Both infohashes and peer identifiers occupy the standard 160-bit identifier space characteristic of the original Kademlia design.

Peer discovery leverages the DHT's routing mechanism by querying nodes closest to the torrent's infohash. This process resembles Kademlia's GET_VALUE RPC but with the difference that it can return multiple peer values per key. Peers can announce their participation in a torrent's swarm by informing the nodes nearest to the infohash, similarly to the STORE RPC in the original protocol.

2.4.2 Multicast

IP Multicast[29] is an efficient method for group communication, allowing a sender to broadcast a message to multiple recipients simultaneously. Unlike traditional unicasting, where a message is sent separately to each receiver, IP Multicast enables a sender to transmit the message just once. The network infrastructure, routers and switches, then duplicates and distributes the message to all members of the group, minimizing redundancy and conserving bandwidth.

One of the key advantages of IP Multicast is that receivers do not need to manage group membership directly, this task is handled by the underlying network infrastructure. Group management is facilitated by protocols such as the Internet Group Management Protocol (IGMP) [9] for IPv4 and Multicast Listener Discovery (MLD) [15] for IPv6.

In practice, IP Multicast is widely used for applications like IPTV, where it allows Internet Service Providers (ISPs) to stream television content efficiently to multiple end users within their networks. However, despite its benefits, IP Multicast has not been widely deployed across the global internet[19]. This lack of widespread support makes it challenging to leverage multicast in peer-to-peer, as support for IP Multicast is fractured into "islands" — networks that, while physically connected as part of the global internet, offer multicast functionality only within their own domains. Since native IP Multicast is not available beyond these boundaries, applications have to resort to approaches such as the one described in [59], where tunnels are created between the islands, thereby enabling the dissemination of messages across the otherwise fragmented multicast infrastructure.

To address this limitation, peers in a distributed network can implement application-level multicast. Unlike IP Multicast, which relies on specialized network infrastructure, application-level multicast is managed directly by the end hosts. Although this approach

is less efficient due to the lack of network-level support, it offers greater flexibility and is easier and cheaper to implement for peer-to-peer systems.

In peer-to-peer multicast systems, most implementations rely on constructing one or more multicast trees. These trees can be built over either unstructured or structured overlays, depending on the system design. Each node in the multicast tree may represent a single node or a cluster of nodes.

Some peer-to-peer multicast systems also make use of existing IP Multicast "islands" when available. These systems integrate with the underlying network infrastructure where IP Multicast is supported.

In the following, we describe some existing application-level multicast systems. Comprehensive surveys, such as [58] and [28], showcase the wide variety of approaches and techniques employed in application-level multicast systems.

2.4.2.1 Narada

Narada [12] is an application-layer multicast protocol designed to "construct an overlay structure among participating end systems in a self-organizing and fully distributed manner." Narada dynamically refines this overlay structure as more network information becomes available. The protocol operates by maintaining a mesh network and constructing spanning trees from the mesh.

Each member of the system maintains information about every other member, including its address, the last known sequence number, and a local timestamp indicating when the last sequence number was received. Members periodically exchange refresh messages with their neighbors, sharing information they hold about other members in the network.

Join Procedure: When a node wishes to join the multicast group, it must contact at least one existing member. The method for obtaining the addresses of current members is outside the scope of this protocol. The joining node sends requests to known members, asking to become their neighbor. If it receives a positive response from any member, the node joins the system and begins exchanging refresh messages with its neighbors.

Leave Procedure: When a member leaves the system gracefully, it notifies its neighbors, who propagate the departure information to the rest of the group. In the case of abrupt departures (e.g., node crashes), neighbors detect the absence of refresh messages over time. They then send probe messages to check the node's status. If no response is received, neighbors notify the group that the node has left. To distinguish between updates about departed members and stale information, each member maintains a list of departed nodes, which can be periodically flushed after a sufficient interval.

Partition Repair: If the mesh (i.e., overlay network) becomes partitioned, members in each partition detect the lack of updates from other members. To repair the partition,

members maintain a queue of "stale" members. Periodically, a member will attempt to remove and contact one node from the head of this queue, either establishing it as a neighbor or determining it to be unreachable. If up-to-date information is received about a queued member, it is removed from the queue. The repair parameters are picked such that only a small number of new links are added to the mesh, minimizing disruptions.

Mesh Improvement: Members periodically probe each other and existing links. Based on a *utility* metric, new links can be added, and less efficient links can be removed to improve the mesh.

Delivery Trees: Narada runs a distance-vector protocol over its mesh. Each member maintains the cost and path to every other member, enabling the construction of delivery trees and preventing the count-to-infinity problem. These trees use reverse shortest-path routing.

For instance, consider the network depicted in Figure 2.2 and a message from source *S* received by *C* via neighbor *B*. If *B* is the next hop on the shortest path from *C* to *S*, then *C* forwards the message to its neighbors for which *C* is the next hop to *S*. In this example, *C* forwards the message to *D*. However, *E* does

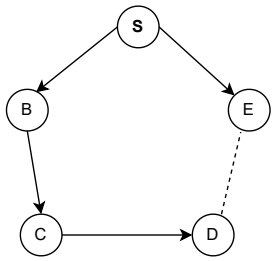

Figure 2.2: Example minimum spanning tree with source S

2.4.2.2 Host Multicast Tree Protocol (HMTP)

The Host Multicast Tree Protocol (HMTP) [59] is an application-layer multicast protocol that leverages existing IP-multicast islands—networks that natively support multicast. These islands are connected to form a tree structure, allowing multicast messages to be distributed across the wider Internet, where native IP multicast support may be unavailable.

Designated Member (DM) and Tree Construction In HMTP, one member from each island is elected as the Designated Member (DM). The DMs communicate with each other over UDP, linking the different islands. Notably, DMs are the only nodes involved in

tree construction and maintenance. To minimize latency, the protocol uses the round-trip member-to-member time as a distance metric during the tree construction process.

Role of the Host Multicast Rendezvous Point (HMRP) The Host Multicast Rendezvous Point (HMRP) is a centralized entity that plays a critical role in the protocol. The HMRP is responsible for registering multicast groups, assisting new members in joining the multicast tree, and keeping track of the root of each multicast group. If the HMRP becomes unavailable, new members cannot join, but existing members can continue to multicast messages without disruption.

Joining the Multicast Network When a DM wishes to join the multicast network, it first queries the HMRP to obtain information about the root of the multicast tree. The DM then selects a parent by requesting the list of children from the root, measuring the latency to each of the children, and recursively querying the child with the lowest latency for its own children. This process continues until the DM finds the best candidate for a parent, either by reaching a leaf node or identifying that the other candidates have higher latencies than the parent. If the chosen parent refuses the connection, the DM continues searching for another suitable parent.

Tree Maintenance and Leaving the Network To maintain the tree, DMs periodically send `REFRESH` messages to their parent (the HMRP if they are the root). In response, the parent sends a `PATH` message that contains the path from the DM to the root of the tree. This ensures that the tree structure remains intact and facilitates updates as the network evolves. If a DM decides to leave the network, it notifies its parent so that it can be removed from the parent's list of children. The DM also informs its children, allowing them to search for a new parent. The process of finding a new parent is similar to the join procedure, except that the child DMs use their current parent's `PATH` to the root and traverse upward in the tree.

Fault Tolerance and Re-Optimization Periodically, every few minutes, DMs re-execute the join procedure to attempt to find a better parent and optimize the multicast tree. This ensures that the network can improve over time by reconfiguring itself based on changing conditions. In the event of a crash, other DMs will notice the absence of `REFRESH` messages and treat the crashed DM as if it had left the network. The `PATH` message is also used for loop detection and correction, ensuring the integrity of the multicast tree structure.

2.4.2.3 Scribe

Scribe [10] is a scalable application-level multicast system designed to support a large number of groups with substantial memberships. It is built on top of Pastry [42], which was briefly described previously in Section 2.3.3.3. Scribe provides best-effort delivery of multicast messages without enforcing message ordering, leaving stronger guarantees

for the user to implement if needed. However, if TCP connections between nodes in the multicast tree remain intact, delivery is both reliable and ordered.

A multicast group in Scribe is identified by a Pastry 120-bit identifier, with the root of the group's multicast tree being the Pastry node closest to that identifier.

Creating a Group To create a group, the owner routes a *CREATE* message to the group's root.

Joining a Group To join a group, a node routes a *JOIN* message to the group's root. Any nodes on the path between the joining node and the root (i.e., those forwarding the message) add the joining node to their list of children for that group. This forwarding process constructs a multicast tree rooted at the group's root. If the *JOIN* message reaches a node already in the group or forwarding messages for the group, the sender of the *JOIN* message is added to the group's children list on the receiving node, and the message is no longer forwarded. Once joined, nodes periodically refresh their membership.

Leaving a Group To leave a group, a node sends a *LEAVE* message to its parent. The parent removes the child from its list of children for the group. If the parent's children list becomes empty and the parent itself is not part of the group, it also sends a *LEAVE* message to its parent.

Sending Messages to the Group To send a message to a multicast group, a node first locates and caches the address of the node responsible for the group. The node then routes a request to the group's root to begin the message dissemination. The root sends the message to all its children, which in turn forward it to their children, and so on, until the message reaches the tree's leaf nodes.

Tree Maintenance To maintain the multicast tree, nodes periodically send heartbeat messages to their children. Children that fail to respond to a heartbeat are considered failed. If a parent (not itself part of the group) no longer has any children for a given group, it leaves the group by sending a *LEAVE* message to its parent.

2.4.2.4 SplitStream

SplitStream [11] is an application-level multicast system designed to address the issue of load imbalance commonly found in traditional tree-based multicast systems. In single-tree multicast systems, interior nodes bear the entire forwarding load while leaf nodes carry none. SplitStream mitigates this problem by constructing multiple multicast trees and distributing (or "striping") the data across them. These trees are structured to ensure that, whenever possible, each node serves as an interior node in only one tree and as a leaf node in all others, thereby balancing the forwarding load among all nodes.

This multi-tree approach not only addresses load imbalance but can also improve robustness if combined with suitable data encoding techniques, such as Multiple Description Coding (MDC). Since each node typically acts as an interior node in only one tree, a node failure should at most result in the loss of a single data stripe for its downstream nodes.

Although SplitStream can be implemented over various tree-based application-level multicast systems, the version discussed here is built on top of Pastry [42] (described in Section 2.3.3.3) and Scribe [10] (described in Section 2.4.2.3).

Multicast Tree Construction SplitStream uses one Scribe multicast tree for each of the k data stripes, where k is a system-defined parameter. A Scribe tree is constructed based on routes from leaf nodes to the group's root. In Pastry, each routing hop should bring a message to a node whose identifier shares an additional digit in common with the destination identifier. To ensure a node acts as an interior node in only one tree, SplitStream assigns unique identifiers to each of the k trees such that they differ in their first digit. As a result, only nodes whose identifiers share the same prefix as a given tree should serve as interior nodes for that tree.

Outdegree Limitation and Push-Down Mechanism Nodes in SplitStream might encounter situations where they exceed their forwarding capacity due to an excessive number of children. To address this, SplitStream introduces a *push-down* mechanism. Instead of accepting new child requests, an overloaded node provides the joining node with a list of its current children. The joining node then attempts to connect to the child in the list with the lowest end-to-end latency. If necessary, this process continues recursively.

However, this mechanism is not guaranteed to succeed. The selected parent may already be a leaf node in the current tree or may have reached its child limit in the tree where it serves as an interior node. In such cases, [11] provides additional details on how to resolve these challenges and ensure a suitable parent is found.

2.4.3 Broadcast

A broadcast service aims to allow any peer to send a message to all other peers in the system. Since broadcast is a special case of multicast, where every node is in the same group, the aproaches previously described in Section 2.4.2 could also be used.

2.4.3.1 Unstructured Overlays

Two different approaches when it comes to implementing a broadcast service over an unstructured overlay are flooding and gossip.

Flooding Flooding is a straightforward method for broadcasting a message across a connected overlay network. In this approach, a peer sends the message to all of its

neighbors, and each neighbor, in turn, forwards the message to its own neighbors. To prevent the message from circulating indefinitely, each message is assigned a unique identifier. A peer will only forward a message if it has not previously encountered that identifier, thus avoiding unnecessary retransmissions. These identifiers can be discarded after a certain time period. This mechanism is similar to how the Gnutella network operates [48], though it does not rely on time-to-live (TTL) values, as the goal is for the message to reach every node in the network. However, this approach has a significant drawback: it can quickly consume large amounts of bandwidth, making it inefficient and difficult to scale for larger networks [41].

Gossip Gossip-based broadcast protocols draw inspiration from the way gossip spreads in social networks, where information rapidly disseminates through a series [46]. In these protocols, a peer seeking to broadcast a message selects f neighbors at random, where f represents the protocol's fanout parameter, and transmits the message to them[33]. These recipient peers subsequently propagate the message through similar random selections, with the ultimate goal of having the message reach the entire network.

Similar to flooding approaches, gossip protocols must implement mechanisms to prevent redundant message forwarding and potential infinite message circulation.

There are several strategies to implement gossip-based broadcasting [34]:

Eager push: Peers immediately forward the complete message to f neighboring nodes upon receiving it.

Pull: Peers conduct periodic inquiries to random neighbors, requesting recently received messages. When a peer identifies a message gap, it directly solicits the missing content from the appropriate neighbor.

Lazy push: Analogous to eager push, but instead of transmitting the full message, peers initially share only a message identifier. Receiving peers must explicitly request the complete message if desired.

These strategies are not mutually exclusive and can be combined to achieve different objectives, such as optimizing message propagation or minimizing bandwidth usage.

2.4.3.2 Structured Overlays

Structured overlays, such as Distributed Hash Tables (DHTs), are primarily designed to enable efficient lookups and message routing compared to unstructured overlays. However, they can also be utilized for broadcasting messages across the network, as explored in studies such as [3], [30], and [27].

One notable approach described in [3] leverages the DHT's ability to route messages in approximately $\log N$ hops within a uniform identifier space. In this method, the broadcasting node sends a tuple $(Message, Limit)$, where $Message$ represents the data to be broadcast, and $Limit$ restricts the reach of the next forwarding step. The node forwards this tuple to one or more nodes at exponentially increasing distances, setting

the limit such that each recipient only forwards the message to nodes within its assigned portion of the identifier space. This prevents redundant message forwarding.

For example, as illustrated in Figure 2.3, if the node at identifier 0 initiates a broadcast, it forwards the message to nodes marked as X, each with a specific limit. These nodes repeat the same procedure, considering only their assigned portions of the identifier space, and continue until no further subdivisions are possible. This structured broadcasting method attempts to be efficient and minimize redundancy.

Concrete descriptions of how this approach is applied in specific DHT protocols, such as Chord and Kademlia, are provided below. This approach however, does not work for all DHTs. For those, different methods might need to be used, such as CAN and the broadcast aproach described in [27].

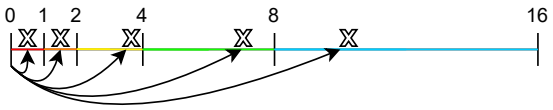

Figure 2.3: DHT Broadcast

Chord In Chord, a broadcast is initiated by a node sending a message to all its neighbors listed in its finger table. If multiple nodes exist for a single finger table entry, the message is sent to the last node in the list for that entry.

For a finger table of size M, the forwarding limit for the node at position $Finger[i]$ (where $1 \leq i \leq M-1$) is defined by the range of identifiers between its own identifier and that of $Finger[i+1]$. This ensures that the node at $Finger[i]$ forwards the message only to nodes whose identifiers fall within this range.

Upon receiving a broadcast message, a node follows the same procedure: it forwards the message to entries in its own finger table, but only to those whose identifiers lie within its designated limit. This structured approach ensures that the broadcast is efficiently propagated across the network and that there is no redundancy[3].

Kademlia In Kademlia, a broadcast is initiated by a node sending a message to peers whose identifiers share an increasing common prefix length (CPL) with its own identifier. For a CPL of i, the node sends the message to one or more peers (if replication is desired) and specifies a limit of $i+1$ in the message.

When a peer receives the message, it continues the process, forwarding the message only to peers whose CPL with itself is at least equal to the received limit. The process terminates when a peer's limit exceeds the CPL shared with any other peer in its routing table.

2.4.4 Peer-to-Peer Livestreaming

Livestreaming is a method of transmitting data in real time from a source to multiple consumers. The amount of data being transferred and the timing constraints vary depending on the application. Examples of livestreaming include IPTV (Internet Protocol Television), stock market data, live sports data, and streaming services such as Twitch, YouTube, and Netflix.

In this work, we focus on peer-to-peer (P2P) livestreaming, specifically the distribution of an arbitrary byte stream from a single source to multiple receivers. Unlike traditional livestreaming approaches that rely on multicast infrastructure within an ISP's network or costly centralized solutions such as Content Delivery Networks (CDNs), P2P livestreaming attempts to make use of the upstream bandwidth of end devices consuming the content. This approach leverages existing resources at the network's edges, reducing dependency on centralized infrastructure and lowering operational costs.

Livestreaming shares similarities with multicast systems, as both aim to efficiently distribute data to multiple consumers. However, livestreaming typically involves stricter timing requirements and a single data producer. Because of this, many multicast systems can be used for livestreaming. Systems such as Narada [12], SplitStream [11], and others explicitly highlight applications like live video conferencing or media streaming.

When designing a P2P livestreaming system, several common approaches have been proposed in the literature:

Single Distribution Tree: In this approach, a single distribution tree is constructed to stream the content. While conceptually simple, it faces several challenges. The tree must be loop-free, though this can be easier to accomplish if a single node is responsible for its construction. To minimize latency for leaf nodes, the tree should ideally be wider rather than taller. Interior nodes bear all of the forwarding load, which can be problematic if load balancing is a concern. Furthermore, this approach is less robust to churn and node failures, as the departure of a single node can disrupt the stream for its entire subtree. Two examples of systems using this approach are Narada [12], previously described in Section 2.4.2.1, and Plumtree[33] which builds and maintains a single spanning tree, used for broadcast, over a gossip-based overlay network.

Multiple Distribution Trees: This approach builds multiple distribution trees instead of a single one. Properly constructed, these trees improve robustness to churn and failures, as the departure of a single node impacts at most one subtree per tree. If no peer belongs to all subtrees, alternative paths remain for receiving data. In scenarios where Multiple Description Coding (MDC) is used, disruptions may only result in reduced quality rather than a complete stream failure. This approach also helps balance the forwarding load if trees are built so that peers are interior nodes in only one tree while serving as leaves in others, as shown in SplitStream [11]. Examples of systems employing this method include

SplitStream [11], previously described in Section 2.4.2.4, CoopNet [38], outlined in Section 2.4.4.1, and Thicket [20], which, like SplitStream, builds and maintains multiple trees but uses a gossip protocol over an unstructured overlay instead of relying on a Distributed Hash Table (DHT).

Hierarchy of Clusters: In this approach, nodes form clusters, often based on a distance metric such as latency. Each cluster designates a leader. Hierarchical systems like NICE [5] and ZIGZAG [50] create layers of clusters, with the leaders of lower layers forming clusters in higher layers until a single top-layer cluster remains. Data distribution follows paths implicitly defined by this hierarchy. Other systems, such as CliqueStream [4], construct a tree of cluster leaders to stream data to each leader, which then disseminates it within the cluster using methods similar to DONet [61]. Examples of systems using this approach include NICE [5], ZIGZAG [50], and CliqueStream [4].

Push-Pull Meshes: In this approach, peers form a mesh and periodically exchange information about the data they have available. Neighbors can use this information to request missing data. To mitigate the latency associated with pull-only approaches, such as DONet [61], nodes can request certain neighbors to push data as soon as it is received while retaining the pull mechanism as a backup.

Additionally, this approach often incorporates iterative improvement of the mesh. Nodes collect metrics such as latency, bandwidth, or reliability from their interactions with neighbors and use these metrics to select better peers over time. This optimization improves the mesh's performance, allowing the system to adapt to changing network conditions and peer availability. Systems like GridMedia [63] leverage this hybrid push-pull approach along with mesh refinement.

2.4.4.1 CoopNet

CoopNet [38] is a peer-to-peer live streaming system designed to operate as a complement to servers in live streaming content rather than replacing the system entirely with a peer-to-peer approach. It selectively employs peer-to-peer networking to improve the scalability of the service.

CoopNet relies on a central server that serves as both the source of the live stream and the manager of multiple distribution trees, altough in reality this server could be a cluster of servers owned by the source. The stream data is encoded using Multiple Description Coding (MDC)[25], and each description is disseminated across different distribution trees. Peers are only required to forward content if they are actively consuming it, and they are expected to contribute an equal amount of upload and download bandwidth.

Join Procedure To join the system, a peer contacts the server, which provides the necessary address information for its parent nodes in each distribution tree. The peer then attaches itself to these parents to start receiving the stream data.

Leave Procedure When a node leaves the system, it should notify the server, which then reassigns its children to new parents and notifies them. If a node leaves abruptly, all its descendants across the distribution trees will stop receiving data. In such cases, affected nodes contact their parent to determine whether the issue lies with the parent or further upstream. Only the direct children of the failed node need to contact the server to find a new parent.

Tree Management The server handles all tree management and constructs the trees using one of two approaches: randomized or deterministic.

Randomized Tree Management In the randomized approach, the server traverses the tree to identify nodes with available bandwidth. One of these nodes is then randomly selected to serve as the parent of the joining node.

Deterministic Tree Management In this approach the server tries to make each node interior in only one tree, similar to SplitStream[11], as a way to help enforce the constraint that upload and download bandwidth of each node should be roughtly the same. This works by having each node, in the tree where it is interior, have at most N children (where N is the total number of trees) and receive one description per tree that it belongs to.

To insert a new node the server picks the tree with the least amount of interior nodes to be the *fertile tree* and the remaining ones to be *sterile trees*. When inserting the node into the *fertile tree* the server finds the first node that either has capacity for more children or that is the parent of a sterile child. In the first case the joining node becomes a child of the node with capacity and in the second case the sterile child is replaced with the joining node and a new parent is found for that child. When inserting a node into a *sterile tree* the server simply finds the first node with capacity and makes it the parent of the joining node.

2.4.4.2 DONet

DONet [61], or "Data-driven Overlay Network", is a peer-to-peer live media streaming system designed for ease of implementation, efficiency, and robustness. Unlike traditional streaming systems that rely on distribution trees, DONet employs a mesh-based approach. Each peer maintains a cache of known network members and establishes partnerships with a subset of these nodes for data exchange.

Each node in DONet is identified by a unique identifier (e.g., IP address) and maintains a partial view of the system's membership through its *membership cache* (*mCache*).

Join Procedure To join the network, a node first contacts the source. The source selects a random node from its *mCache*, known as the *deputy*, and redirects the joining node to this deputy. This approach distributes the workload among network members and alleviates

pressure on the source. The joining node then requests a list of network members from the deputy, allowing it to establish partnerships and construct its partial membership view.

Membership Maintenance To keep the overlay network and the *mCache*s up-to-date, each node periodically generates a message $(SeqN, Id, NumPartners, TTL)$ that is disseminated using SCAM [22], a gossip protocol. Nodes that receive this message update or create a corresponding *mCache* entry for the node identified by Id.

Departure Procedure When a node leaves the system gracefully, it sends a message similar to the membership maintenance message, informing other nodes of its departure. This allows peers to update their *mCache*s accordingly. If a node fails abruptly, one of its partners detects the failure and broadcasts a similar departure message on behalf of the failed node.

Data Exchange The content being streamed is divided into equally sized chunks, which nodes exchange among their partners. Each node maintains a *Buffer Map* (BM), a bitset representing the chunks it possesses. The BM operates similarly to BitTorrent's [8] bitset but is constrained to 120 bits and includes an offset indicating the first chunk in the bitset due to the continuously growing number of chunks.

Nodes periodically exchange BMs with their partners. Using the received BMs, a scheduling algorithm determines which chunks to download from which partners. Experimental results reported by the authors indicate that stream lag between nodes is generally less than one minute.

Overlay Optimization Because the initial partners of a node are unlikely to be optimal and network conditions can change over time, DONet employs an iterative approach to optimize overlay connections. Each node assigns a score to its partners based on metrics such as bandwidth and chunk availability. Periodically, nodes establish new connections with random nodes from their *mCache* while retaining their highest-scoring partners to discard low performing partners and discover potentially better partners.

2.4.4.3 GridMedia

GridMedia [63, 60] is a peer-to-peer media livestreaming service that employs a push-pull approach over an unstructured overlay. While it shares similarities with DONet [61], its primary distinction lies in the use of a hybrid push-pull method for data exchange, as opposed to DONet's pull-only mechanism.

Join Procedure GridMedia uses a *Rendezvous Point* (RP), a central node that facilitates the bootstrap process for new peers. When joining, a node begins by contacting the RP, which provides a *candidate list* — a set of nodes already participating in the network. The joining node measures the Round-Trip Time (RTT) to each node in the candidate list and

selects its initial neighbors. Some of these neighbors are chosen based on the lowest RTT, while others are picked randomly from the list to ensure diversity.

Membership Maintenance Each node maintains a table containing a partial view of the system's membership. Entries in this table are associated with a Time-To-Live (TTL) value, which allows expired entries to be removed. Nodes periodically share a subset of their membership table with their neighbors, enabling updates or additions to their local view. Additionally, nodes send periodic "alive" messages to their neighbors to signal their presence and maintain connections.

Departure Procedure When a node leaves gracefully, it floods a "quit" message to its neighbors within a limited hop count. Nodes receiving this message promptly remove the departing node from their membership table. If a node leaves abruptly, its absence is detected by the lack of "alive" messages. In such cases, neighbors remove the node from their tables. Over time, as TTL values for the departed node expire in other tables, the entry is gradually purged across the network.

Overlay Optimization To optimize the network, each node calculates an *evaluation index*, which is the sum of packets sent and received between that node and its local peers. If the evaluation index for a given neighbor falls below a predefined threshold, the connection to that neighbor is terminated. The node then selects a new neighbor from its membership table in an attempt to maintain an efficient overlay.

Data Exchange Like DONet [61], GridMedia employs *Buffer Maps* (BM), where the BM represents a variable-length bitset indicating the packets currently available at a node. In pull mode, data exchange occurs in fixed-interval cycles. During each cycle, a node requests specific packets from its neighbors and awaits their responses.

The RP synchronizes the cycles for newly joined nodes. After the initial cycle, the node begins selecting neighbors to request (subscribe) packets from, prioritizing those with higher traffic in the previous interval. Once a node receives a packet it immediatly pushes it to any subscribers. If certain packets are not received during a cycle, the node subsequently pulls these missing packets from alternative neighbors.

2.4.4.4 NICE

NICE [5] is an application-layer multicast protocol designed for low-bandwidth data streaming applications with large receiver sets. It organizes nodes into a hierarchical structure of clusters, where data is transmitted via source-specific trees implicitly defined by this hierarchy. The hierarchy consists of multiple layers, each comprising clusters of nodes. A cluster is a group of interconnected nodes with one designated as the cluster leader. The leader serves is the "center" of the cluster, determined as the node with the lowest maximum end-to-end latency to all other nodes in the cluster. All nodes belong

to a cluster in the lowest layer, and each higher layer is composed of the leaders from the layer below.

Join Procedure To join the system, a node first contacts a *Rendezvous Point* (RP), which is either the leader of the cluster in the highest layer or a node capable of reaching that leader. The RP provides a list of nodes in the highest layer, that are the leaders of the clusters in the layer below. The joining node identifies the closest node from this list and requests the list of members in the cluster for which that node is the leader. This process repeats iteratively until the joining node reaches a cluster in the lowest layer, which it joins.

If the addition of the new node causes a leader change, because it is now the center of the cluster, the old leader removes itself from higher layers, and the new leader joins the next layer. This can cause changes in multiple clusters requiring the system to reconcile.

Cluster Maintenance Each node periodically sends heartbeat messages to every other cluster member. These messages include the sender's estimated distance to the cluster leader, and, in the case of the leader's heartbeat, the message also contains the full cluster membership. This mechanism helps nodes detect failures through missing heartbeats.

Clusters must maintain a size within a predefined range. If a cluster grows too large or too small, the leader initiates a split or merge operation to maintain the system's invariants.

Departure Procedure When a node leaves gracefully, it sends a REMOVE message to its neighbors. If a node departs unexpectedly, its departure is detected via missing heartbeat messages. If the departing node is a leader, the remaining cluster members exchange heartbeats that include their estimates of the most suitable new leader based on distance. This process continues until a new leader is chosen. However, this process can fail, [5] provides additional details on handling such scenarios.

Data Exchange Data transmission occurs over source-specific trees. A node that wants to send data transmits it to every member of all the clusters it belongs to. Upon receiving the data, a node forwards it to other clusters it belongs to, provided the node that it received the message from is not already a member of those clusters.

2.4.5 Discussion

The previous section described different approaches to peer-to-peer livestreaming systems. Tree-based methods, while conceptually straightforward and easier to construct with centralized coordination, inherently suffer from fragility. A single node failure can disrupt entire subtrees, or even the entire tree building process if it uses centralized coordination, making the approach less resilient in dynamic network environments. Mesh based approaches, while more robust to churn and failures, can be more introduce more redundancy and overhead and may require iterative optimization to achieve acceptable performance.

3

Future Work

3.1 Problem Description

Traditional live streaming systems have predominantly relied on centralized architectures, where content distributors use dedicated streaming servers or content delivery networks (CDNs) to deliver media to end users. While these approaches have demonstrated effectiveness, they come with high barriers to entry and centralize control within the platform.

In contrast, decentralized systems, initially popularized by the peer-to-peer paradigm at the start of the 21st century, have recently regained attention with the rise of the Web 3.0 movement. This movement advocates for a more decentralized and democratic Internet. Decentralized live streaming systems promise greater scalability, lower barriers to entry, and a shift of control away from centralized entities.

This dissertation addresses the challenge of designing and implementing a decentralized, peer-to-peer live streaming system capable of distributing multiple concurrent streams while maintaining acceptable quality of service. The proposed system avoids reliance on IP multicast—a technology that, despite its benefits, has not achieved widespread adoption across the Internet. Instead, the focus is on developing an application-layer solution that achieves high efficiency and scalability.

The requirements and constraints of our problem can be summarized as follows:

- The system should support multiple concurrent streams, each carrying arbitrary byte data.
- Nodes should be able to efficiently locate and join specific streams using unique identifiers.
- The system operates under soft real-time constraints, allowing for delays of up to one minute, and should ensure consistent and reliable delivery of stream content.
- The system should not rely on any central component and peers must self-organize into efficient distribution structures, adapt to network dynamics, and maintain these

structures as peers join and leave the system (churn). The system must also handle heterogeneous peer capabilities, as nodes may have varying upload and download capacities, processing power, and network conditions.

- The system should scale effectively with the number of participating peers.

3.2 Proposed Solution

The proposed solution consists of two components:

Stream Discovery Overlay Network The first component is an overlay network designed to efficiently locate and join streams. This network will be built using an existing Distributed Hash Table (DHT), altough exactly which one is still not decided, and maintains a mapping between stream identifiers and the peers currently participating in each stream. By leveraging the DHT, the system enables efficient and scalable stream discovery, allowing peers to quickly find and join active streams.

Content Distribution Overlay Network The second component is an overlay network dedicated to the distribution of content for a single stream. This network is based on a mesh structure, where each peer connects to a limited number of other peers, forming a decentralized mesh. Each peer within the mesh is assigned a score calculated using relevant metrics such as available bandwidth, proximity to the source, and uptime within the stream. These scores are used to select neighbors, ensuring that higher-quality peers are positioned closer to the source.

Content transmission within the network is performed using a hybrid push/pull strategy, similar to the approach outlined in [63, 60]. This strategy ensures efficient data dissemination while adapting to varying network conditions.

3.3 Evaluation

To evaluate the proposed solution, we plan to implement a few existing peer-to-peer streaming systems and compare their performance with our approach. These implementations will be developed using the Babel framework [21]. The evaluation will focus on relevant performance metrics such as:

- **Average Stream Delay:** The average time it takes for data to travel from the source to the peers.
- **Data Redundancy:** The amount of duplicated data transmitted within the network.
- **Data Delivery Rate:** The percentage of data successfully delivered to peers.
- **Robustness to Churn:** The system's ability to maintain performance despite frequent peer arrivals and departures.

3.4 Work Plan

The work plan is shown in Figure 3.1 and is divided into the following main stages:

Lookup Overlay This stage consists of the design, implementation, and testing of a DHT-based overlay network for effective stream discovery.

Dissemination Overlay This stage consists of the design, implementation and testing of an overlay network for content distribution. This stage is the main focus of this work and may extend beyond the time frame shown in the work plan if the early results of the evaluation stage show unexpected results that require changes to the proposed solution.

Experimental Evaluation This stage consists of the implementation of some existing solutions to serve as a baseline for comparision as well as the evaluation setup that will be used to compare the proposed solution with the existing ones. After that some scenarios informed by use cases will be used to obtain the experimental results.

Writing This stage consists of writing the dissertation and a paper for submission at a national or international venue.

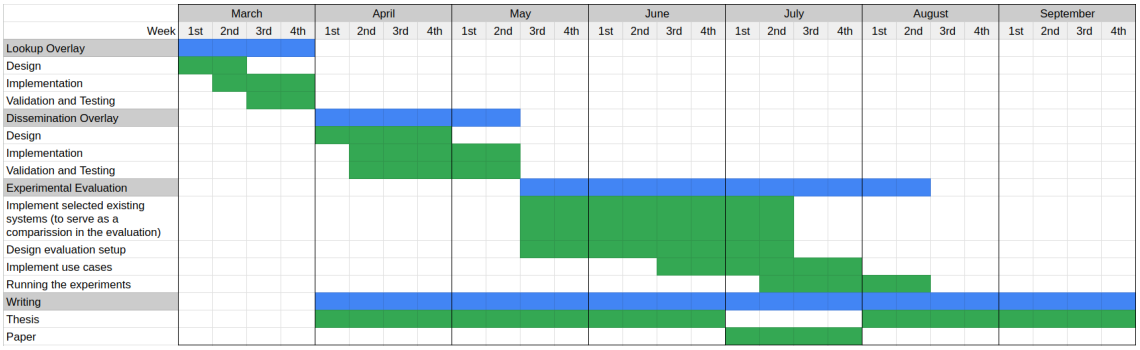

Figure 3.1: Work Plan

Bibliography

[1] *2024 Global Internet Phenomena Report*. URL: https://www.sandvine.com/phenomena (visited on 2024-12-07) (cit. on pp. 1, 6).

[2] V. K. Adhikari et al. "Unreeling netflix: Understanding and improving multi-cdn movie delivery". In: *2012 Proceedings IEEE Infocom*. IEEE. 2012, pp. 1620–1628 (cit. on p. 1).

[3] S. El-Ansary et al. "Efficient broadcast in structured P2P networks". In: *Peer-to-Peer Systems II: Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22, 2003. Revised Papers 2*. Springer. 2003, pp. 304–314 (cit. on pp. 24, 25).

[4] S. Asaduzzaman, Y. Qiao, and G. Bochmann. "CliqueStream: An Efficient and Fault-Resilient Live Streaming Network on a Clustered Peer-to-Peer Overlay". en. In: *2008 Eighth International Conference on Peer-to-Peer Computing*. Aachen, Germany: IEEE, 2008-09, pp. 269–278. ISBN: 978-0-7695-3318-6. DOI: 10.1109/P2P.2008.35. URL: http://ieeexplore.ieee.org/document/4627289/ (cit. on p. 27).

[5] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. "Scalable application layer multicast". en. In: *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*. Pittsburgh Pennsylvania USA: ACM, 2002-08, pp. 205–217. ISBN: 978-1-58113-570-1. DOI: 10.1145/633025.633045. URL: https://dl.acm.org/doi/10.1145/633025.633045 (cit. on pp. 1, 27, 30, 31).

[6] *BitTorrent - a new P2P app*. URL: https://web.archive.org/web/20080129085545/http://finance.groups.yahoo.com/group/decentralization/message/3160 (visited on 2024-12-07) (cit. on p. 5).

[7] *BitTorrent: The "one third of all Internet traffic" Myth*. URL: https://torrentfreak.com/bittorrent-the-one-third-of-all-internet-traffic-myth/ (visited on 2024-12-07) (cit. on p. 5).

[8] *BitTorrent.org*. URL: http://www.bittorrent.org/index.html (visited on 2024-12-07) (cit. on pp. 5, 29).

[9] B. Cain et al. *Internet Group Management Protocol, Version 3*. RFC 3376. 2002-10. DOI: 10.17487/RFC3376. URL: https://www.rfc-editor.org/info/rfc3376 (cit. on p. 18).

[10] M. Castro et al. "SCRIBE: A large-scale and decentralized application-level multicast infrastructure". In: *IEEE Journal on Selected Areas in communications* 20.8 (2002), pp. 1489–1499 (cit. on pp. 1, 21, 23).

[11] M. Castro et al. "Splitstream: High-bandwidth multicast in cooperative environments". In: *ACM SIGOPS operating systems review* 37.5 (2003), pp. 298–313 (cit. on pp. 22, 23, 26–28).

[12] Y.-h. Chu et al. "A case for end system multicast". en. In: *IEEE Journal on Selected Areas in Communications* 20.8 (2002-10), pp. 1456–1471. ISSN: 0733-8716. DOI: 10.1109/JSAC.2002.803066 (cit. on pp. 7, 19, 26).

[13] I. Clarke et al. "Freenet: A distributed anonymous information storage and retrieval system". In: *Designing privacy enhancing technologies: international workshop on design issues in anonymity and unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings*. Springer. 2001, pp. 46–66 (cit. on p. 8).

[14] I. Clarke et al. "Protecting free expression online with freenet". In: *IEEE Internet Computing* 6.1 (2002), pp. 40–49 (cit. on p. 8).

[15] L. Costa and R. Vida. *Multicast Listener Discovery Version 2 (MLDv2) for IPv6*. RFC 3810. 2004-06. DOI: 10.17487/RFC3810. URL: https://www.rfc-editor.org/info/rfc3810 (cit. on p. 18).

[16] G. DeCandia et al. "Dynamo: Amazon's highly available key-value store". In: *ACM SIGOPS operating systems review* 41.6 (2007), pp. 205–220 (cit. on p. 7).

[17] J. Deng et al. "Internet scale user-generated live video streaming: The Twitch case". In: *Passive and Active Measurement: 18th International Conference, PAM 2017, Sydney, NSW, Australia, March 30-31, 2017, Proceedings 18*. Springer. 2017, pp. 60–71 (cit. on p. 1).

[18] *DHT Protocol*. URL: http://www.bittorrent.org/beps/bep_0005.html (visited on 2024-12-07) (cit. on pp. 6, 14, 18).

[19] C. Diot et al. "Deployment issues for the IP multicast service and architecture". In: *IEEE network* 14.1 (2000), pp. 78–88 (cit. on pp. 1, 18).

[20] M. Ferreira, J. Leitao, and L. Rodrigues. "Thicket: A Protocol for Building and Maintaining Multiple Trees in a P2P Overlay". en. In: *2010 29th IEEE Symposium on Reliable Distributed Systems*. New Delhi, Punjab India: IEEE, 2010-10, pp. 293–302. ISBN: 978-0-7695-4250-8. DOI: 10.1109/SRDS.2010.19. URL: http://ieeexplore.ieee.org/document/5623402/ (cit. on pp. 1, 27).

[21] P. Fouto et al. "Babel: A framework for developing performant and dependable distributed protocols". In: *2022 41st International Symposium on Reliable Distributed Systems (SRDS)*. IEEE. 2022, pp. 146–155 (cit. on p. 33).

[22] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. "Peer-to-peer membership management for gossip-based protocols". In: *IEEE transactions on computers* 52.2 (2003), pp. 139–149 (cit. on p. 29).

[23] *GLOBAL NAPSTER USAGE PLUMMETS, BUT NEW FILE-SHARING ALTERNATIVES GAINING GROUND, REPORTS JUPITER MEDIA METRIX*. URL: https://web.archive.org/web/20080413104420/http://www.comscore.com/press/release.asp?id=249 (visited on 2024-11-26) (cit. on p. 5).

[24] *Gnutella Dynamic Query Protocol v0.1*. URL: https://web.archive.org/web/20041214161017/http://www.limewire.com/developer/dynamic_query.html (visited on 2024-12-18) (cit. on p. 17).

[25] V. K. Goyal. "Multiple description coding: Compression meets the network". In: *IEEE Signal processing magazine* 18.5 (2002), pp. 74–93 (cit. on p. 27).

[26] M. Harren et al. "Complex queries in DHT-based peer-to-peer networks". In: *Peer-to-Peer Systems: First InternationalWorkshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers 1*. Springer. 2002, pp. 242–250 (cit. on pp. 15, 17).

[27] L. Henrio, F. Huet, and J. Rochas. "An optimal broadcast algorithm for content-addressable networks". In: *Principles of Distributed Systems: 17th International Conference, OPODIS 2013, Nice, France, December 16-18, 2013. Proceedings 17*. Springer. 2013, pp. 176–190 (cit. on pp. 24, 25).

[28] M. Hosseini et al. "A survey of application-layer multicast protocols". In: *IEEE Communications Surveys & Tutorials* 9.3 (2007), pp. 58–74 (cit. on p. 19).

[29] *Host extensions for IP multicasting*. RFC 988. 1986-07. DOI: 10.17487/RFC0988. URL: https://www.rfc-editor.org/info/rfc988 (cit. on pp. 1, 18).

[30] K. Huang and D. Zhang. "DHT-based lightweight broadcast algorithms in large-scale computing infrastructures". In: *Future Generation Computer Systems* 26.3 (2010), pp. 291–303 (cit. on p. 24).

[31] E. Khatibi and M. Sharifi. "Resource discovery mechanisms in pure unstructured peer-to-peer systems: a comprehensive survey". In: *Peer-to-Peer Networking and Applications* 14 (2021), pp. 729–746 (cit. on pp. 15, 16).

[32] G. Korpal and D. Scott. "Decentralization and web3 technologies". In: *Authorea Preprints* (2022) (cit. on p. 1).

[33] J. Leitao, J. Pereira, and L. Rodrigues. "Epidemic broadcast trees". In: *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*. IEEE. 2007, pp. 301–310 (cit. on pp. 24, 26).

[34] J. Leitao, J. Pereira, and L. Rodrigues. "Epidemic broadcast trees". In: *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*. IEEE. 2007, pp. 301–310 (cit. on p. 24).

[35] 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)*. IEEE. 2007, pp. 419–429 (cit. on p. 7).

[36] *libp2p Kademlia DHT specification*. URL: https://github.com/libp2p/specs/blob/master/kad-dht/README.md#distance (visited on 2024-12-11) (cit. on p. 10).

[37] P. Maymounkov and D. Mazieres. "Kademlia: A peer-to-peer information system based on the xor metric". In: *International workshop on peer-to-peer systems*. Springer. 2002, pp. 53–65 (cit. on pp. 7, 9).

[38] V. N. Padmanabhan, H. J. Wang, and P. A. Chou. "Resilient peer-to-peer streaming". In: *11th IEEE International Conference on Network Protocols, 2003. Proceedings.* IEEE. 2003, pp. 16–27 (cit. on p. 27).

[39] *Peer Exchange (PEX)*. URL: http://www.bittorrent.org/beps/bep_0011.html (visited on 2024-12-07) (cit. on p. 6).

[40] S. Ratnasamy et al. "A scalable content-addressable network". In: *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. 2001, pp. 161–172 (cit. on pp. 12, 13).

[41] J. Ritter. *Why gnutella can't scale. no, really*. 2001 (cit. on pp. 15, 24).

[42] A. Rowstron and P. Druschel. "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems". In: *Middleware 2001: IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12–16, 2001 Proceedings 2*. Springer. 2001, pp. 329–350 (cit. on pp. 7, 9, 11, 21, 23).

[43] R. Steinmetz. *Peertopeer Systems and Applications*. en. Ed. by R. Steinmetz and K. Wehrle. Lecture notes in computer science. Berlin, Germany: Springer, 2005-01 (cit. on p. 4).

[44] I. Stoica et al. "Chord: A scalable peer-to-peer lookup service for internet applications". In: *ACM SIGCOMM computer communication review* 31.4 (2001), pp. 149–160 (cit. on pp. 7, 9).

[45] D. Stutzbach and R. Rejaie. "Understanding churn in peer-to-peer networks". In: *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. 2006, pp. 189–202 (cit. on p. 7).

[46] S. Tarkoma. *Overlay Networks: Toward Information Networking*. 1st. USA: Auerbach Publications, 2010. ISBN: 143981371X (cit. on pp. 7, 24).

[47] S. M. Thampi et al. "Survey of search and replication schemes in unstructured p2p networks". In: *arXiv preprint arXiv:1008.1629* (2010) (cit. on pp. 15, 16).

[48] *The Gnutella Protocol Specification v0.4*. URL: https://web.archive.org/web/20010603073444/http://www.clip2.com/GnutellaProtocol04.pdf (visited on 2024-12-04) (cit. on pp. 14, 16, 24).

[49] *The Napster Nightmare*. URL: https://web.archive.org/web/20111019152028/http://www.isp-planet.com/politics/napster.html (visited on 2024-11-26) (cit. on p. 5).

[50] D. Tran, K. Hua, and T. Do. "ZIGZAG: an efficient peer-to-peer scheme for media streaming". In: *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*. San Francisco, CA: IEEE, 2003, 1283–1292 vol.2. ISBN: 978-0-7803-7752-3. DOI: 10.1109/INFCOM.2003.1208964. URL: https://ieeexplore.ieee.org/document/1208964/ (cit. on p. 27).

[51] *Trustworthy and Resilient Decentralised Intelligence for Edge Systems*. URL: https://docs.aws.amazon.com/whitepapers/latest/amazon-cloudfront-media/appendix-a-amazon-interactive-video-service.html (visited on 2025-01-11) (cit. on p. 1).

[52] *Trustworthy and Resilient Decentralised Intelligence for Edge Systems*. URL: https://cordis.europa.eu/project/id/101093006 (visited on 2025-01-10) (cit. on p. 2).

[53] *Trustworthy and Resilient Decentralised Intelligence for Edge Systems*. URL: https://project-tardis.eu/ (visited on 2025-01-10) (cit. on p. 2).

[54] M. Van Steen and A. S. Tanenbaum. *Distributed Systems*. North Charleston, SC: Createspace Independent Publishing Platform, 2017-02 (cit. on p. 4).

[55] Q. H. Vu, M. Lupu, and B. C. Ooi. *Peer-to-Peer Computing: Principles and Applications*. en. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. ISBN: 978-3-642-03513-5. DOI: 10.1007/978-3-642-03514-2. URL: https://link.springer.com/10.1007/978-3-642-03514-2 (cit. on pp. 1, 4, 5, 15).

[56] B. Yang and H. Garcia-Molina. "Improving search in peer-to-peer networks". In: *Proceedings 22nd International Conference on Distributed Computing Systems*. IEEE. 2002, pp. 5–14 (cit. on p. 17).

[57] K.-H. Yang, C.-J. Wu, and J.-M. Ho. "Antsearch: An ant search algorithm in unstructured peer-to-peer networks". In: *IEICE Transactions on Communications* 89.9 (2006), pp. 2300–2308 (cit. on p. 17).

[58] C. K. Yeo, B.-S. Lee, and M. H. Er. "A survey of application level multicast techniques". In: *Computer Communications* 27.15 (2004), pp. 1547–1568 (cit. on p. 19).

[59] B. Zhang, S. Jamin, and L. Zhang. "Host multicast: a framework for delivering multicast to end users". en. In: *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*. Vol. 3. New York, NY, USA: IEEE, 2002, pp. 1366–1375. ISBN: 978-0-7803-7476-8. DOI: 10.1109/INFCOM.2002.1019387. URL: http://ieeexplore.ieee.org/document/1019387/ (cit. on pp. 1, 18, 20).

[60] M. Zhang et al. "Large-scale live media streaming over peer-to-peer networks through global internet". en. In: *Proceedings of the ACM workshop on Advances in peer-to-peer multimedia streaming*. Hilton Singapore: ACM, 2005-11, pp. 21–28. ISBN: 978-1-59593-248-8. DOI: 10.1145/1099384.1099388. URL: https://dl.acm.org/doi/10.1145/1099384.1099388 (cit. on pp. 29, 33).

[61] X. Zhang et al. "Coolstreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming". In: *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* Vol. 3. Miami, FL, USA: IEEE, 2005, pp. 2102–2111. ISBN: 978-0-7803-8968-7. DOI: 10.1109/INFCOM.2005.1498486. URL: http://ieeexplore.ieee.org/document/1498486/ (cit. on pp. 27–30).

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

[63] L. Zhao et al. "Gridmedia: A Practical Peer-to-Peer Based Live Video Streaming System". In: *2005 IEEE 7th Workshop on Multimedia Signal Processing*. Shanghai: IEEE, 2005-10, pp. 1–4. ISBN: 978-0-7803-9288-5. DOI: 10.1109/MMSP.2005.248686. URL: http://ieeexplore.ieee.org/document/4013958/ (cit. on pp. 27, 29, 33).