Algorithmic Aspects of Distributed Hash Tables on Cloud, Fog, and Edge Computing Applications: A Survey

Aristeidis Karras[1(✉)], Christos Karras[1], Nikolaos Schizas[1], Spyros Sioutas[1], and Christos Zaroliagis[1,2]

[1] Computer Engineering and Informatics Department, University of Patras, 26504 Patras, Greece
{akarras,c.karras,nschizas,sioutas,zaro}@ceid.upatras.gr
[2] Computer Technology Institute and Press "Diophantus", Patras University Campus, 26504 Patras, Greece

Abstract. In the current era, where data is expanding due to the unforeseen volume, velocity, and variety of data types produced by IoT devices, there is an imperative need to manage such data in remote IoT environments. However, these complexities have been inadequately addressed by conventional data management methods. In such scenarios, Distributed Hash Tables (DHTs) have emerged as an effective solution for efficient data storage and retrieval. Conversely, the dynamizature of IoT data presents its own set of challenges, such as decreased performance, inconsistent data, and increased overhead. To improve the performance of DHTs, we examine their algorithmic properties in cloud, fog, and edge computing environments, taking into account network designs, resource availability, latency requirements, and data proximity. This survey explores the adaptation of algorithmic elements in DHTs for optimal data administration in these cloud computing environments. Moreover, we examine advanced techniques such as effective hashing, adaptive routing, defect tolerance mechanisms, and load balancing. In addition, we address the challenges of managing vast and diverse volumes of IoT data, taking into account the unique features and constraints of cloud, fog, and edge environments. We also conduct contemporary research on security and privacy, focusing on algorithmic and architectural solutions for data integrity, confidentiality, and availability. This work enhances our comprehension of dynamic DHT algorithms and their potential for effective data management across multiple computing paradigms by investigating state-of-the-art research.

Keywords: DHTs · Cloud Computing · Fog Computing · Edge Computing · IoT Systems

1 Introduction

In the information era, where data is generated at an unprecedented scale, speed, and diversity due to the advent of the Internet of Things (IoT) devices, robust

I. Chatzigiannakis and I. Karydis (Eds.): ALGOCLOUD 2023, LNCS 14053, pp. 133–171, 2024.
https://doi.org/10.1007/978-3-031-49361-4_8

data management mechanisms are of paramount importance. Traditional data management techniques have had their limitations exposed due to these evolving complexities. Therefore, Distributed Hash Tables (DHTs) have been highlighted as a viable option for managing data in remote IoT contexts due to their effective, scalable data storage and retrieval capabilities. However, because of the dynamic and diverse nature of IoT data, these typical DHTs encounter their own difficulties, which can result in issues with decreased performance, inconsistent data, and higher overheads.

The demand to overcome these limitations and enhance the capabilities of DHTs is driving a new line of research into the algorithmic properties of DHTs in various computing environments, including cloud, fog, and edge computing. Due to variances in their network designs, resource availability, latency requirements, and closeness to data sources, each of these paradigms calls for certain algorithmic considerations and adaptations. This research provides an in-depth analysis of these algorithmic elements, evaluating how the methodology and underlying algorithms of DHTs can be tailored for optimal data management in the cloud, fog, and peripheral computing contexts. Focusing on the sophisticated algorithmic approaches inherent to DHTs, such as effective hashing, adaptive routing, fault tolerance mechanisms, and load balancing, we provide an analytical perspective on the operation of DHTs in these three distinct contexts. We focus on how these algorithmic choices operate while managing massive and varied data volumes produced by IoT systems, taking into consideration the unique properties and limitations of cloud, fog, and edge settings. Apart from that, we examine state-of-the-art methods in security and privacy research and difficulties linked to DHTs, highlighting algorithmic and architectural solutions that seek to guarantee data integrity, privacy, and availability. Additionaly, this work contributes to a deeper comprehension of the dynamic nature of DHTs algorithms and their potential for effective data management across many computing paradigms. We focus on underpinning the most recent findings in this field, offering insights into the potential advantages and the likely future advances that can arise.

The rest of this survey is organized as follows. Section 2 presents the background and related work in the field of Distributed Hash Tables (DHTs) as well as their algorithmic concepts. In Sect. 3, we present DHTs suitable for Cloud Computing while highlighting decentralized task management, privacy-preserving aggregation, and object storage. Section 4 presents DHTs in Fog and Edge Computing Environments along with data management and use cases, while Sect. 5 presents the Open Problems and Challenges. Ultimately, the survey concludes in Sect. 6 where the findings are presented as well as some possible Future Directions on DHTs.

2 Background and Related Work

The rapid growth in the volume, variety, and velocity of data has necessitated the development of innovative storage and management solutions to meet the demands of modern Big Data Systems. Traditional centralized data storage systems face limitations in scalability, performance, and fault tolerance as data

volumes continue to expand. In response, Distributed Hash Tables (DHTs) have emerged as an essential technology, offering decentralized, scalable, and fault-tolerant mechanisms for distributed data storage and retrieval.

In this section, we overview the background and related work of various DHTs, highlighting their distinctive contributions, optimizations, and applications in the context of Big Data Systems and Big Data Management.

- **Pioneering DHTs:** Chord, Kademlia, and Pastry are considered pioneering DHTs, as they laid the groundwork for many subsequent designs. These early DHTs introduced key concepts such as consistent hashing, logarithmic routing tables, and baseSS routing, which remain relevant and influential in contemporary DHT implementations. By demonstrating the potential of decentralized data storage and retrieval, these DHTs helped shape the future of distributed systems.
- **Churn Resilience and Balanced Performance:** Some DHTs, like Bamboo and Kelips, have been designed to address specific challenges associated with distributed systems. Bamboo, for example, aims to provide efficient and robust routing under high churn rates which is a common issue in peer-to-peer networks. Conversely, Kelips focuses on achieving a balance between maintenance overhead and lookup performance by implementing a soft-state peer-to-peer membership protocol.
- **Latency Optimization and Locality-Awareness:** OneHop and SkipNet are DHTs that target latency optimization and locality awareness. OneHop prioritizes low-latency lookups by maintaining a larger amount of routing information, which results in increased maintenance overhead. SkipNet, on the other hand, employs the skip list data structure to enable efficient, locality-aware routing, leading to performance improvements in specific use cases.
- **High-Performance Object Location and Retrieval:** Coral is a DHT specifically designed for high-performance object location and retrieval, with a specialty similar to the BitTorrent protocol. By optimizing for a particular use case in the IoT context, Coral showcases the potential of DHTs for supporting efficient, distributed file-sharing systems.
- **Enterprise-Grade DHTs:** DHTs such as Apache Cassandra, Riak, and Voldemort have been tailored for enterprise use, underpinning large-scale distributed systems for organizations like LinkedIn and Amazon. These enterprise-grade DHTs often incorporate additional optimizations and features, such as tunable consistency levels, support for complex data types, and seamless integration with other Big Data technologies. This versatility renders them suitable for a broad range of industrial applications.

In summary, the variety of options of Distributed Hash Tables has played a crucial role in advancing Big Data Systems and Big Data Management in the IoT context. The ongoing adaptation and evolution of DHTs have paved the way for a new generation of distributed systems capable of managing the ever-increasing demands of a data-driven society.

2.1 Literature Review

Through our detailed research, we found a significant gap: there is not a single review that fully examines DHT-based strategies from various angles, including the structure and communication of systems, especially in multiple areas like blockchain, fog and cloud computing, IoT, and edge computing. This also extends into Online Social Networks (OSNs), Mobile Ad Hoc Networks (MANETs), and Vehicular Ad Hoc Networks (VANETs). In other words, there is a noticeable lack of deep-diving research that explores how DHT-based applications can be utilized in different technological fields, even with new insights emerging in specific domains. For example, interconnected clouds are explored in [82], fog computing in [35,84,91], and cloud computing in [4,21], each uncovering unique challenges and opportunities for integrating DHT approaches. In light of this, it is crucial to pull together these important, yet scattered, insights to better understand how DHT mechanisms can be applied and integrated into these fast-developing tech areas.

The sole extant study on DHT-based techniques in the blockchain area is [8], which provides a cursory evaluation of the application of DHT-based data storage capability. This work focuses on factors such as privacy, trustworthiness, integrity, data management and load balancing, all of which are thoroughly addressed in this survey. In the IoT context, existing DHT-based surveys primarily focus on features such as routing and lookup processes and mobility management [25]. The DHT-based approaches for scalability and service discovery, which are included in this survey, are, however, not covered in the talks in those surveys.

Neither of the previous studies, including those by [16,72,99] employ a DHT-oriented approach in their classification of Online Social Networks (OSNs). They only cover a brief analysis of DHTs as a potential contender for distributed routing and storage management among other OSN-based technologies. Aspects like spam prevention, service discovery, and data dependence management-all of which are thoroughly discussed in this survey-remain untouched as a result. The only current DHT-based survey in the realm of Mobile Ad Hoc Networks (MANETs), [2], concentrates on routing methods without diving deeper into elements covered by this study, such as data transfer, dynamic topology management, and traffic overhead relocation. Similar to how our survey covers service discovery, scalable routing, security, and privacy, the only DHT-based study [13] in the field of vehicular ad hoc networks (VANETs) to date only looks at the use of DHTs for distributed cluster management. Furthermore, these existing surveys lack a thorough investigation of the challenges, open problems, and research guidelines related to the utilization of DHTs in their respective domains.

In summary, this literature review highlights the absence of comprehensive surveys that encompass DHT-based approaches in various domains, including fog, blockchain, IoT, edge, OSNs, cloud computing, MANETs, and VANETs. It highlights the necessity of a survey that covers the utilization of DHTs in these domains and addresses aspects such as integrity, privacy, trustworthiness, data

management, load balancing, scalability, service discovery, routing, security, and privacy.

2.2 Definition of a DHT

Distributed Hash Tables (DHTs) are an aspect of decentralized systems that map keys to values in a way that allows any participating node in the network to efficiently retrieve the value associated with a given key. Each network node and data item in a DHT is given a distinct key, which is commonly represented by an IP/port combination for nodes and by file names or other distinctive identifiers for data. Assuming that the keys are evenly distributed, this mapping is deterministic, guaranteeing that for every given input, the associated key will stay consistent.

DHTs are frequently used as the base for peer-to-peer (P2P) networks with overlays, where the structure of the DHT defines the underlying network architecture. This approach enables every network participant to easily find the node associated with a certain key. Furthermore, data may be stored in the network and accessed by utilizing the identity of the node linked with the key used to store the data.

Before we dive into the functionality of Distributed Hash Tables (DHTs) in the context described, let us briefly introduce the concepts of Voronoi cells, data points, and Delaunay triangulation to establish a foundation for the forthcoming discussion. In computational geometry, a Voronoi diagram partitions a plane into regions based on the distance to points in a specific subset of the plane. Imagine scattering seeds (data points) across a field. The region of land that is closest to a particular seed—where any point in that region is closer to that seed than to any other—forms a Voronoi cell. Essentially, each seed (data point) owns its own distinct Voronoi cell, comprising all points in a plane that are nearest to it relative to other seeds. This partitioning offers a structured, yet decentralized, method to efficiently locate and access data points in a distributed network. In harmony with this, Delaunay triangulation comes into play by connecting these data points in a way that no point is inside the circumcircle of any triangle in the triangulation, creating a network of neighbors that allows for efficient pathfinding and data retrieval strategies, seamlessly aligning with its application in DHTs.

A DHT can be conceptualized as an area populated by Voronoi cells and data points. Each node is responsible for preserving the data within its Voronoi cell, whose boundaries are determined by its closest neighbors. The Delaunay triangulation pertinent to the node in question includes Voronoi cell neighbors sharing a border with the node in question. From any node in the network, one can ascertain, in sublinear time, the node responsible for a particular key or a specific position in the DHT.

In essence, despite the variability in the specifics of different DHT protocols, the following features are indispensable to their design:

- **Distance Metric:** Central to any DHT is the concept of a distance metric. This requires establishing a mechanism to calculate the distance between two items. Once such a metric is defined, it allows us to articulate what it means to say that a certain node is responsible for the data in its vicinity.
- **Definition of Proximity:** Proximity or closeness is an essential concept in a DHT because it determines which node is responsible for which data, which nodes are its closest neighbours, and how a node should interact with them. Although the concepts of closeness and distance are interconnected, they are in fact distinct. To help clarify this point, we refer to the example of Chord [77], where the distance between two points a and b is defined as the shortest path around the circle in either direction.
- **Midpoint Definition:** This function determines the point in the DHT that is equally distant between any two specified locations.
- **Peer Management Strategy:** The backbone of a DHT's definition is its peer management strategy. Factors such as the size of the peer group, the data they contain, and the frequency at which their status is checked are all components of the peer management strategy. Most trade-off decisions are made at this stage of the DHT configuration process.

Generally, it is not necessary to devise a specific DHT routing scheme. In accordance with the universal routing concept, which is utilized by all DHTs, a message is sent to the closest known node to the destination. The exact routing scheme employed is determined by the protocol. Kademlia, for example, uses concurrent iterative queries for routing, whereas Chord permits both iterative and recursive routing. This illustrates the adaptability of DHT routing mechanisms.

2.3 Terminology

Due to the vast variety of DHTs, different authors have used different names to define congruent DHT components, as certain terminology may only make sense in particular contexts. Due to the fact that this work will discuss numerous DHTs with distinct terminology, we have developed a unified vocabulary:

key - A common hashing technique uses a 160-bit hash produced by a hash algorithm that correlates to a distinct[1] SHA-1[2].

ID - A key that connects to a certain node is the ID. A node's ID and the actual node cannot be distinguished from one another. In this work, nodes, and files are referred to by their IDs and keys, respectively.

[1] One-of-a-kind with a very high probability. Since it is extremely unlikely, formal DHT specifications frequently ignore the hash collision risk. Any of the collision resolution techniques, including chaining and linear probing, may be used to handle this for any file. The only canonical solution to a collision between two nodes, whether it is a node or a file, is to hope it doesn't happen.

[2] Many businesses are discontinuing SHA1 in 2017 because of the study on hash collisions [76] and the availability of hardware available to do SHA hash collisions.

Peer - Added to the network of active users. This section makes the assumption that each peer is made up of a unique set of hardware.

Peerlist - All of a node's peers that it is aware of. The *routing table* is the conventional name for this, however, various DHTs [67,94] overload the terminology. A subset of the entire peer list is any table or list of peers.

Short-hops - The group of peers that, per the DHT's measurement, are "adjacent/nearest" to the node in the keyspace. These are the node's *predecessor(s)* and *successor(s)* in a 1-dimensional ring like Chord [77]. They may also be referred to as "*neighbours*". The part of the peer list that the node is not immediately next to is known as a long-hop, alternatively known as extended connections, shortcuts, or fingers.

Root Node - The node in charge of a certain key is identified as the Root Node.

Successor - The root node's alternate name. The neighbour who will take up a node's tasks if it leaves the network is known as its successor.

n **nodes** - It refers to how many network nodes exist.

Similarly, with slight variations, all DHTs carry out identical processes.

`lookup(key)` - This approach locates the root node of the `key`. The `lookup` function must be used in some fashion in almost all DHT operations.

`put(key, value)` - The root node of the `key` contains the `value`. Unless otherwise stated, `key` is taken to be the hash-key of `value`. The Tapestry is in opposition to this idea.

`get(key)` - Lookup-like, but returns the value stored by `key:put`. This is a minor distinction, as `lookup(key)` may be used to directly query the matching node. However, many systems utilize backup and caching techniques, which keep several copies of the object over the network. We can use `get` if it is a backup or if we do not emphasize which node provides the value mapped with `key`.

`delete(key, value)` - This does not require further explanation as DHTs typically do not focus on the key deletion and allow the application to make that choice. DHTs commonly make the assumption that the key-value pairs they store have a finite lifespan and are automatically deleted after that when they do address the problem.

Each node must have the ability to *join* and take care of itself locally.

`join()`. There are two steps to the join operation. Prior to joining, the connected node must create its peer list. It must initialize a peer list even if it is not required to have a complete peer list when it joins. The connecting node must secondly inform other nodes of its presence.

Maintenance. Typically, maintenance techniques fall into one of two categories: *lazy* or *active*. When doing lazy maintenance, peers are assumed to be in good health until they prove otherwise, at which time they are immediately replaced. Peers are periodically ping-ed and replaced in active maintenance if they can no longer be located. In fact, only neighbors receive proactive care, while everything else suffers slack upkeep.

The overlay peerlist, geometry, fault-tolerance and `lookup` function implementation in DHTs are all examined in this paper. Since the `leave()` procedure is quite straightforward and of little significance, we presume that nodes never depart the network gracefully and instead always fail unexpectedly.

2.4 Chord

Making a new ring-based DHT without being influenced by the Chord prototype, often known as Chord, is challenging [77]. It is highly renowned for its user-friendly routing, rules that make figuring out who owns a key very simple, and large number of variations. Chord has been awarded the 2011 SIGCOMM Test of Time Award for its prominence in Computer Science. Recent studies have revealed that Chord has not been implemented correctly for more than ten years and that many of the invariant qualities Chord claimed to have may occasionally fail [92].

List of Peers and Geometrical Distribution. All messages in Chord travel upstream and wrap around by bouncing from one node to the next with a larger ID. It is a 1-dimensional flexible ring. A unique m-bit key that corresponds to one of the 2 m spots on a ring or ID is generated for each network participant and their associated data. Figure 1 depicts a Chord network example.

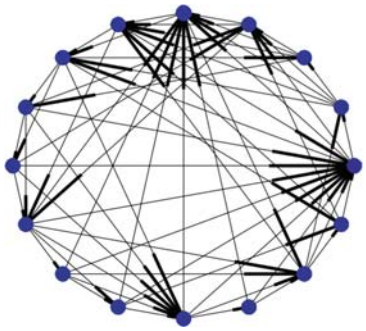

Fig. 1. 16 nodes in a Chord ring. The fingers are seen cutting through the ring (long hop connections).

All information whose keys are upstream from its predecessor's ID and downstream from its own ID is controlled by a network node. A node that controls a key is known as the root or successor of that key. Recursive upstream node queries are used to carry out the lookup and routing operations. In this strategy, searching for a key would take $\mathcal{O}(n)$ time if you just questioned your neighbors.

The *finger table*, a database containing m shortcuts to other peers, is present on each node to expedite lookups. The node that is the successor of the key

$n + 2^{i-1} \mod 2^m$ is the i-th item in the finger table of a node n. The finger that is nearest to the key being looked for is queried by nodes during a lookup, but they do not pass the finger until they get to the root node. The search area for a key is roughly halved with each hop. With an average number of hops of $\frac{1}{2}O(\log_2(n))$, Chord now has a highly scalable lookup time of $\log_2(n)$ for any key [77]. The peer list also contains a list of s fault-tolerant neighbors in each direction in addition to finger tables. This increases the overall size of the peerlist to $log_2(2^m) + 2 \times s = m + 2 \times s$, assuming that each entry is unique.

Join Operations. Within the context of network integration, the act of joining the network entails a series of coordinated operations. This process initiates when a given node, referred to as n, initiates a request to node n', seeking assistance in locating its designated `successor(`n`)`. This successor node is carefully chosen by node n based on relevant data, while the maintenance infrastructure promptly disseminates information to ensure that the existence of node n is acknowledged by other network nodes. During this transitional period, node n performs a crucial role in assuming certain responsibilities that were previously allocated to its predecessor's successor. Under this scenario, node n ensures the continuity and smooth functioning of the network, mitigating any potential disruption caused by the transition.

Robustness. In the spectrum of robust network architectures, the Chord protocol presents a noteworthy approach that ensures data resilience through a sophisticated backup mechanism. By using a strategy wherein nodes back up their data to their immediate predecessors, known as upstream nodes, Chord effectively fortifies the network against potential failures and disruptions. This mechanism becomes crucial due to the fundamental elements of Chord's key management system, wherein a node's closest successor assumes control over its keys upon departure or failure.

The employment of backup procedures to the immediate predecessors, or upstream nodes, serves as a vital safeguard for preserving data integrity within the Chord network. By distributing the responsibility of data storage among these preceding nodes, Chord significantly reduces the risk of material loss, particularly during scenarios where multiple nodes experience simultaneous failures. The presence of a successor list further diminishes the likelihood of data loss, as it allows for seamless reassignment of key control in the face of node departures or failures. It is worth noting that Chord's robustness is not achieved without an associated cost. In terms of message complexity, the backup mechanism necessitates $\mathcal{O}(\lg^2(n))$ messages. This factor indicates the computational overhead incurred by Chord's upkeep cycle, but it also demonstrates the protocol's meticulous attention to ensuring network resilience. By investing in this level of redundancy and backup mechanisms, Chord strives to uphold a high degree of robustness and data availability.

For a comprehensive understanding of Chord's maintenance cycle, including detailed insights into its backup mechanisms and resiliency features, interested

readers can refer to the extensive documentation available in [77] which offers a comprehensive analysis of Chord's design principles, architectural nuances, and contributions to the field of robust network protocols.

Applications. An application of Chord DHT on IoT has been developed in [6]. The application focuses on leveraging the Chord protocol, which is a distributed lookup protocol for decentralized P2P systems, to enable efficient communication and coordination among LoRa wireless sensor nodes. LoRa technology is known for its low-power, long-range capabilities, making it suitable for wireless sensor networks.

2.5 Kademlia

Since a modified version of Kademlia (Mainline DHT) serves as the foundation of the BitTorrent protocol, it is possible that Kademlia [50] is the DHT that is most well-known and often utilized. In order to make nodes able to consume peer list changes with each query, Kademlia was created.

Peerlist and Geometry. Kademlia employs m-bit keys for files and nodes, similar to Chord. The binary tree-based architecture used by Kademlia, on the other hand, uses nodes as the tree's leaves. The XORing of the IDs of any two nodes in the tree yields the distance between them. In contrast to Chord, where distances are not symmetric, the XOR distance metric assumes that they are.

Kademlia nodes use a routing table with m lists, referred to as k-buckets, to keep information about the network. There are up to k nodes for each k-bucket that are between 2^i and 2^{i+1}, where $0 \leq i < m$. A network subtree that is external to the node is represented by each k-bucket. The least recently observed eviction approach, which avoids active nodes, is used to maintain each k-bucket. The sender's details are added to the tail of the relevant k-bucket each time the node gets a message. The information is relocated to the tail if it already exists. The node begins pinging the nodes in the list from the top down if the k-bucket is full. In order to make place for the new node at the tail, a node is deleted from the list as soon as it doesn't respond. If there are no further updates to the k-bucket for a while, the node does a `refresh` on it. A refresh in the k-bucket is a `lookup` of a random key.

Lookup. The `lookup(key)` transmits one message and returns the data for one node in the majority of DHTs. In Kademlia, both of these elements of `lookup` are different: each node that receives a `lookup(key)` returns the k nearest nodes to `key` that it is aware of. The `lookup` is performed simultaneously. The searching node initiates a `lookup(key)` process by sending parallel lookups to the α nodes from the corresponding k-bucket. These *alpha* nodes will each asynchronously provide the k closest nodes it is aware of that are closest to the given `key`. The node will keep sending lookups as they return their findings until no more nodes

are identified. If the destination is a network-stored file, the `lookup` will also be successful if a node claims to have that file.

Joining. An initial contact is made by a joining node before it does a *lookup* on its own ID. New nodes are added to the joining node's peer list and made aware of by other nodes at each stage of the *lookup* method. Last but not least, the joining node does a `refresh` on each k-bucket that is farther away than the node it is aware of.

Fault-Tolerance. By repeatedly executing the `store` command, per file stored on the network is actively republished by nodes once every hour. Two optimizations are employed to prevent network flooding. If a node receives a `store` command on a file it is holding, the timer for that file is first reset, assuming that $k-1$ other nodes also received the command. This indicates that each hour, just one node republishes a file. Second, a republish does not include `lookup`. The `puts` operation of the file in the k closest nodes to the key during `store(data)` operation adds additional fault tolerance. Apart from the operations such as `lookup`, there is minimal active or routine maintenance involved.

2.6 CAN

The Content Addressable Network (CAN) [62] runs on a d-dimensional torus, having the entire coordinate space divided among members, in contrast to the other DHTs discussed in this article. The keys that are contained in the "zone" that a node owns are its responsibility. Every key gets hashed into one of the geometric space's points.

Peerlist and Geometry. The utilization of a peer list in the CAN (Content Addressable Network) protocol is characterized by its minimalistic nature, consisting solely of neighboring nodes. To establish an efficient routing scheme, CAN assigns a distinct geometric area within the coordinate space to each participating node. In addition, each node maintains a routing table that contains an extensive list of neighboring nodes in close proximity.

The size of the routing table in CAN is determined by the parameter $\mathcal{O}(d)$, which corresponds to the number of dimensions. This parameter influences the capacity of the routing table, dictating its ability to accommodate node entries. In a populated CAN network, characterized by the presence of at least $2d$ nodes, a lower bound constraint of $\Omega(2d)$ is imposed on the routing table size. This constraint becomes evident upon closer examination of each axis, as it guarantees that each axis is encompassed by at least one node at both ends. Crucially, when the network undergoes fragmentation due to a growing number of nodes joining, maintenance algorithms enable the routing table to dynamically expand and adapt, ensuring it can effectively accommodate the changing network topology.

Lookup. The lookup operation in CAN (Content Addressable Network) involves a structured routing process, in which each node collaborates with its neighbors. Every node maintains a routing table that establishes connections to adjacent nodes or those with which it shares a boundary area. According to the routing method, the lookup message is sent to the neighbor who is closest to the destination in each hop until it reaches the responsible node. This simplistic routing approach requires a minimal storage space of $2 \cdot d$ and achieves an average path length of $\frac{d}{4} \cdot n^{\frac{1}{d}}$ in a spatially uniform distribution across the network's n nodes. In CAN, the overall time required for a lookup operation is bounded by the expression $O(n^{\frac{1}{d}})$ hops, reflecting the scalability of the network.

Interestingly, during the creation of CAN, Kleinberg was concurrently investigating small-world networks, as mentioned in [40]. Kleinberg's research on lattice networks demonstrated analogous characteristics with the introduction of just a single shortcut. In contrast, CAN stands out as a network without any shortcuts. In the event of a lookup failure, a node resorts to selecting the next best available route. But it's crucial to remember that the greedy lookup strategy might not always work if it happens before a node can recover from churn-caused disturbances. Unsuitable candidates are found using an expanding ring search as a fallback method. This approach restarts the greedy forwarding mechanism and ensures the eventual completion of the lookup process.

Joining. The joining procedure involves splitting the geometric space among the nodes. In order to locate the node m now in charge of location P, a member of the node is contacted by node n with position P. The area of node m is divided so that each node is in control of half once node n informs node m of its intention to join. After the new zones are defined, n and m build the routing table for n from its previous neighbors and m. The tables of these nodes are then updated when they are notified of the most recent changes. As a consequence, only $O(d)$ nodes are impacted by the join operation. The original publication by CAN [62], provides further information on this splitting procedure.

Repairing. A node in a DHT alerts its neighbors that it is departing often has little impact on the network in CAN scenarios. A leaving node's (f) zone is simply moved to a neighboring zone of equal size, uniting the two zones. Minor issues arise when there isn't an equally-sized neighbor and this isn't practicable. Whenever this fragmentation happens, the zone belonging to f is assigned to the neighbor with the smallest area, which has to remain while it's been fixed.

Unexpected failures can also be handled very easily. Each node transmits a pulse to its neighbors that include both that of its neighbors and its own. A node starts a `takeover` countdown if it doesn't hear a pulse from f after a certain number of cycles and thinks that f has ended in failure. The node attempts to occupy f's space by sending a `takeover` message to all of f's neighbors after the timer expires. The volume of the node is included in this message. A node that gets a `takeover` message has two choices: it may either reply with a `takeover`

message of its own or if its zone is smaller than the broadcaster's, it can cancel the countdown.

In CAN, when a node fails, the neighboring node with the smallest zone typically assumes control of the zone that belonged to the failed node. This rule produces rapid recoveries that only impact $O(d)$ nodes, but it necessitates the use of a zone reassignment method to eliminate the fragmentation brought on by `takeovers`. In conclusion, fragmentation must be repaired by a maintenance method even when a failing node is discovered very instantly and recovered rapidly. As mentioned earlier in the text, Ratnasamy et al. [62] also present the concept own using landmarks to choose coordinates, rather than a has function. The round-trip time (RTT) to each of the m landmarks is computed by each node. Which yields one of $m!$ permutations. The keyspace is partitioned into $m!$ regions, each corresponding to one of the orderings. A joining node now chooses a random location from the region corresponding to its landmark ordering.

Design Improvements. Ratnasamy et al. identified a number of improvements that could be made to CAN [62]. Some of these improvements have already been explored in Sect. 1. Adding more dimensions to the coordinate space is one way the system has been modified. Increasing d improves fault tolerance and reduces path length. One concept Ratnasamy et al. introduces is the idea of multiple coordinate spaces existing simultaneously, called *realities*. Each object in the DHT exists at a different set of coordinates for each reality simultaneously.

So a node might have coordinates (x_0, y_0, z_0) in one reality, while having coordinates (x_1, y_1, z_1) in another. Independent sets of neighbors for each reality yield different overall topologies and mappings of keys to nodes. Multiple realities increase the cost of maintenance and routing table sizes but provide greater fault tolerance and greater data availability. A final modification involves permitting multiple nodes to share the same zone, meaning that zones are not necessarily required to split during a join operation.

2.7 Pastry

Both Pastry and Tapestry [67,94], which share many similarities, employ a prefix-based routing mechanism first developed by Plaxton et al. (see [61]). In Tapestry and Pastry, every key is encoded as a base $2b$ number (usually in Pastry $b = 4$, producing a hexadecimal that is easily readable). The peer list that results most closely resembles an induced hypercube topology [11], with each node serving as a vertex. The use of a proximity metric is one noteworthy aspect of Pastry. This statistic indicates that IDs close to the node are used by the peer list.

Peerlist. Three elements make up Pastry's peer list: the routing table, a leaf set, and a neighborhood set. The routing table has $2^b - 1$ items per row and $\log_{2^b}(n)$ rows. The peers in the routing table whose first i digits match the sample node ID are found at the ith level. As a consequence, peers without a shared prefix

with the node are found in row 0, followed by peers with a shared prefix in row 1, peers with a shared prefix in row 2, etc. There is one record for each of the $2^b - 1$ potential differences since each ID is a base $2b$ number. Using the system node 05AF as an example, which has a hexadecimal keyspace that spans from 0000 to FFFF and a $b = 4$ value, we can see how this works.

- A suitable peer for the first level 0 entry would be 1322.
- A suitable peer for the tenth[3] entry of level 1 would be 0AF2.
- A suitable peer for the ninth level one entry would be 09AA.
- For the second entrance of level 3, 05F2 would be a suitable peer.

The L nodes with the numerically closest IDs are stored in the leaf set, with half of it being utilized for lower IDs and the other half for bigger IDs. 2^b or 2^{b+1} is a common value for the constant L. The leaf set is utilized for routing when the destination key is close to the ID of the current node. The neighborhood set includes the L closest nodes, as measured by a certain proximity metric. However, this set is not typically used for routing purposes.

Lookup. The `lookup` procedure is a simple example of a recursive process. The `lookup(key)` is complete when the leaf set, which are the nodes closest to the current node, include the `key`. In this situation, the destination will be either the current node or the leaf set.

If the target node cannot be instantly identified, the node consults its routing database to ascertain which node it should connect to next. The length l shared prefix in the lth entry of the node's routing table is searched. The `lookup` is continued using the element from this row that matches at least one further digit of the prefix. The closest ID from the whole peer list is used to begin the `lookup` if this item is absent or has failed. Given that the search space is reduced by $\frac{1}{2^b}$ for each hop along the routing table, lookup is predicted to take $\lceil \log_{2^b} \rceil$.

Joining. In order to join the network, node J sends a `join` message to A, a node that is close by according to the proximity metric. The root of X receives the `join` message., which we'll refer to as *root*, much like a `lookup`. A copy of each node's peer list that got the `join` is sent to J. The routing table's ith row is created by copying the ith node contacted during the `join`, and the leaf set is created by copying the leaf set from the *root*. Because `join` requires that A and J be near to one other, from A's neighborhood set, the neighborhood setting is replicated. This indicates that the neighborhood specified for A would be nearby. Each node in the table receives a copy of the joining node's peer list after it has been created, allowing them to update their routing tables. A `join` has a constant coefficient of $3 * 2^b$ and a cost of $O(log_2^b n)$ messages.

[3] 0 is the 0th level.

Fault Tolerance. Pastry repairs its routing table and leaf set in a less thorough manner. If a leaf set node fails, the node contacts the leaf set node with the lowest or biggest ID, depending on whether the failed node's ID was smaller or bigger. The node replaces the rejected item and returns a copy of its leaf set. When a node fails and the node is present in the routing table, it contacts another node with an item in the same row to identify a replacement. In order to maintain track of its members, the neighborhood set is actively monitored. When a neighborhood set member becomes unresponsive, the node copies the neighborhood set of another entry and makes repairs from that selection.

Proximity Metric. Pastry's goal is to minimize the "distance" messages travel, where distance can be defined by some metric, typically the number of hops. The keyspace nodes closest to the node make up the leaf set. According to the distance measure, the nodes closest to the node make up the neighborhood set. Guarantees routing time is $< \log n$ in typical operation. Guarantees eventual delivery except when half of the leaf nodes fail simultaneously.

2.8 Tapestry

Tapestry [94] is based on the same prefix-based lookup [60] as Pastry [67] and the peer list and lookup operation share many similarities. Tapestry views itself more as a DOLR [14]. This essentially means that it is a distributed key-based lookup system like a DHT [30], but with some subtle differences at the abstract level which manifest as large % implementation changes. The essential difference here is that Tapestry has servers *publish* records/objects on the network, which direct lookups to the server. The assumption here seems to be that the servers, not the responsible node, serve the actual data. DHTs care or don't care on an application-to-application basis whether keys are associated with records or content.

Symphony and Small World Routing. Despite the fact that Chord [77] and Symphony [49] are both $1d$ ring-based DHTs, [40] uses small world network principles to build Symphony, which is a comparable DHT. The term "small world networks" originates from a phenomenon that psychologists observed in the late 1960s. In the experiments of [52], participants were assigned the task of delivering a letter to a specific recipient; in one experiment, the recipient was a stockbroker based in Boston, while in another, it was the wife of a divinity student in Cambridge. The participants were guided by instructions that allowed them to pass the letter only to someone they considered to have a stronger likelihood of personally knowing the intended recipient. For messages to reach their intended recipient in these communications, there were only an average of 5 hops between a theme and a participant.

This inspired research into building a network with links spread randomly yet with quick lookup times. Kleinberg's navigation technique [41] showed that nodes could deliver messages in $\mathcal{O}(\log^2 n)$ hops in a two-dimensional lattice network

by using just their neighbors and a single randomly chosen[4] finger. This means that a $\mathcal{O}(1)$ sized routing table may do a $\mathcal{O}(\log^2 n)$ lookup.

Peerlist. Symphony employs a 1-dimensional ring as opposed to Kleinberg's 2-dimensional lattice[5], which is essentially a 1-dimensional lattice, like Chord. As opposed to employing a keyspace with values ranging from 0 to $2^n - 1$, Symphony distributes keys with m bits to the modular unit range $[0, 1)$. With the help of $\frac{hashkey}{2^m}$, this position was discovered. Although the design is somewhat arbitrary, it simplifies the process of making selections from a random distribution.

Similar to Chord, nodes are aware of both their immediate predecessor and successor. Similar to Chord, Nodes also maintain a list of $k \geq 1$ fingers, these fingers are chosen at random, unlike Chord. The probability distribution for these fingers is given by the equation $e^{ln(n)+(rand48()-1.0)}$, where `rand48()` is a C function that generates a random float double between "0.0" and "1.0" and "n" is the total number of nodes in the network. Since n is difficult to quantify in P2P networks because of their dynamic nature, each node approximates it based on how close or remote its neighbors are.

In Symphony, it is noteworthy that the links are bidirectional, which leads to each node having a total of $2k$ fingers. Essentially, when a node sends a finger to a peer, the peer responds by generating a finger in return.

Joining and Fault Tolerance. In Symphony, the fault tolerance and joining procedures are quite simple. In order to retrieve the parent node's ID after identifying it, to locate the parent node, a joining node requests help from a member. After integrating itself between its predecessor and successor, the connecting node creates its fingers at random. The usage of successor and predecessor lists handles failures of close neighbors. Failures for fingers are handled in a lazy manner, and when one is discovered, another randomly generated connection is used in its stead.

2.9 ZHT

One of the fundamental principles in DHT design is the recognition that churn - the continual process of nodes joining and leaving the network - is a critical factor necessitating ongoing maintenance. In light of this, nodes are designed to maintain only a minimal subset of the entire network for routing purposes. For the great majority of distributed systems, storing the complete network is not scalable owing to bandwidth restrictions and communication overhead brought on by the frequent joining and departing of nodes.

The bandwidth and memory requirements for every node to store a complete record of the routing table are insignificant in a system that does not account for churn. This would be the case with a high-performance computer cluster or data

[4] Randomly chosen from a specified distribution.
[5] Technically, this is a one-dimensional lattice.

center, where churn is often brought on by hardware failure rather than user attrition. Such a mechanism may be seen in ZHT [45] and Amazon's Dynamo [17]. A "zero-hop hash table," or ZHT, makes use of the predictable lifetime of nodes seen in High-End Computing systems. Nodes are added at the start of a job and deleted at its completion. ZHT can do a `lookup` in $O(1)$ time thanks to this attribute.

Peerlist. ZHT uses a 64-bit ring to operate, giving it a total of $N = 2^{64}$ addresses. ZHT strictly restricts the amount of physical nodes allowed in the network to n. Resulting in n divisions of $\frac{N}{n} = \frac{2^{64}}{n}$ keys. The partitions are distributed equally over the network.

There are i virtual nodes and k physical nodes in the network, and on each of them, ZHT is being used in at least one instance (virtual node). Each instance in the ring is in charge of a certain range of partitions. Since the network experiences minimal or no churn, each node maintains a current list of all other nodes. This list doesn't require frequent updates due to the network's stability. The cost of memory is remarkably low. The footprint of each instance is 10 MB, and each item in the membership table only requires 32 bytes per node. This indicates that there are 0–2 hops involved in routing.

Joining and Fault Tolerance. ZHT employs either a static or dynamic membership. In the case of static membership, once the network is bootstrapped, it does not allow any new nodes to join. In the case of ZHT, dynamic membership allows nodes to join at any moment. A random member is approached and asked for a copy of the peerlist in order to join. The most severely overloaded node can be identified by the joiner. To take over the partitions of that node, the joiner chooses a network address. As per Fault Tolerance, ZHT addresses only scheduled network exits or hardware failures. For data redundancy, nodes create backups with their neighboring nodes.

3 DHTs in Cloud Computing

DHTs are pivotal in orchestrating a symphony of seamless interactions within cloud computing environments, ensuring that resources are utilized efficiently and that data is accessible in a decentralized manner. Figure 2 encapsulates these concepts visually, providing an illustrative overview of the integral role and operational mechanics of DHTs in cloud computing environments.

The DHT-based solutions for cloud computing are summarized in Table 1. This table categorizes various proposed solutions according to certain key attributes, such as the type of solution (e.g., Infrastructure or Application), the Distributed Hash Table (DHT) strategy employed, the kind of nodes involved, the domain (i.e., Centralized, Decentralized, or Distributed), and the primary benefits associated with the solution. These strategies involve various DHT techniques, like Voldemort, Skip Graph, and Chord, and they cater to different layers

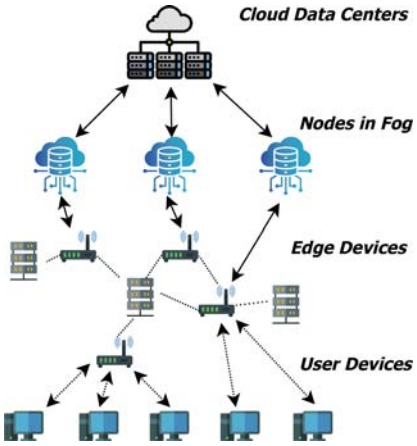

Fig. 2. An Overview of a DHT System in the Context of Cloud Computing.

and aspects of cloud computing, addressing challenges in storage, computation, task scheduling, and data distribution, among others. The varied domains and benefits indicate the versatility and applicability of DHT-based approaches in addressing the multifaceted challenges encountered in cloud computing environments.

Table 1. An overview of DHT-based Solutions for Cloud Computing

Solution	Type	DHT	Nodes	Domain	Benefits
Optimization of Storage Costs [97]	Infrastructure	Voldemort	Servers	Centralized	Cost Reduction
Optimization of Compute Costs [98]	Infrastructure	Voldemort	Virtual Machines	Centralized	Cost Reduction
Cluster Task Scheduling [88]	Application	Chord	Virtual Machines	Decentralized	Task Distribution
m-Cloud [53]	Application	General	Cloud Domains	Decentralized	Resource Integration
Game Object Storage [39]	Application	Chord	Game Objects	Distributed	Game Data Management
IPFS [7]	Application	Kademlia	Data Objects	Decentralized	Distributed File System
Time-sensitive Object Storage [89]	Application	General	Symmetric Keys	Decentralized	Temporal Data Storage
P2P Streaming [27]	Application	General	Video Chunks	Distributed	Media Distribution

3.1 Decentralized Task Management

A container encompasses a complete runtime atmosphere, encompassing an application alongside all its prerequisites and libraries essential for execution. The purpose of containerization is to facilitate software mobility across diverse computing environments through isolation. A container cloud, meanwhile, refers to a platform responsible for managing and scheduling the computational tasks within containers. Kubernetes stands as a prominent instance of such container clouds, widely employed within the industry.

Clustering the containers is how the container cloud orchestrates them. One master node oversees the administration of tasks across all of the nodes in each cluster, which controls how the cluster as a whole is run. However, depending only on a master node might be problematic since it presents a single point of failure and can lead to performance bottlenecks as the cluster gets bigger. This work presents a decentralized task management system based on a Distributed Hash Table (DHT) to address these scalability and reliability challenges [88].

The suggested approach replaces the whole cluster with a node-based decentralized DHT overlay. This overlay is used by nodes to find other nodes and their available resources. A node describes its state using a unique key and delivers its address as the associated value within the DHT to communicate its existence to others. The management protocol, which runs on each node, obtains from the DHT the status information of other cluster nodes. Utilizing this data, it makes informed task scheduling decisions, efficiently distributing jobs across the node itself or other nodes within the cluster.

3.2 Privacy Preserving Aggregation

A Distributed Hash Table (DHT) structure is used by the m-Cloud [53] system to aggregate sensor data while protecting user privacy. A federation of many cloud domains is used in this strategy. Cloud federation is a system concept where several administrative domains communicate information from their individual private clouds with one another [78]. A depiction of the cloud federation system model is presented in Fig. 3, where a larger federated cloud domain consists of four inner private cloud domains. The sensor data must be collected in parts by each cloud domain. To protect the privacy of specific sensor data, the data inside each cloud domain is aggregated before being shared with others.

A Distributed Hash Table (DHT) structure is used by the m-Cloud [53] system to aggregate sensor data while protecting user privacy. A federation of many cloud domains is used in this strategy. Cloud federation is a system concept where several administrative domains communicate information from their individual private clouds with one another [78]. In Fig. 3, a cloud federation system paradigm is shown, where a bigger federated cloud domain is made up of four inner private cloud domains. Some sensor data must be collected by each cloud domain. To protect the confidentiality of specific sensor data, the data inside each cloud domain is aggregated before being shared with others.

With m cloud domains, each sensor divides its data into chunks and shares the ith chunk with the i^{th} cloud domain. The aggregated data bits inside each cloud domain are subjected to the calculation function, and the resulting computation is then forwarded to a single operator. In order to calculate the overall result, the operator then merges the separate cloud results. Each cloud domain is in charge of a certain set of keys and runs a DHT node while collaborating with other domains in a DHT overlay.

To assign sensor data chunks to the cloud nodes, the chunk's key is computed using its hash value, and the responsible DHT cloud node for that key is determined. It should be noted, however, that m-Cloud can only execute aggregated calculations on certain polynomials. All polynomial functions that can be calculated in this method are listed in the paper. $x + x^2 + x^3$ is an example of such a polynomial, with m-Cloud splitting and sharing the x term with one DHT node, x^2 with another, and so on.

Privacy Aspects of Distributed Hash Tables. Privacy-preserving aggregation (PPAgg) is crucial in safeguarding the individual data contributor's privacy while aggregating data from diverse sources, especially in contexts involving sensitive data such as Vehicular Ad Hoc Networks (VANETs), Federated Learning (FL), and Smart Grid Systems. In particular, PPAgg ensures secure aggregation of data and further maintains the confidentiality of each contributor during the process. Key application scenarios include:

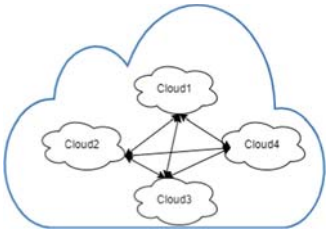

Fig. 3. A depiction of the cloud federation system model.

- VANETs: Utilizing PPAgg to securely aggregate traffic messages into an aggregated ciphertext and subsequently batch-unencrypt them, mitigating privacy, computational, and communication concerns [56,90].
- Federated Learning: Employing PPAgg to aggregate locally trained models into a global model, ensuring participant privacy throughout the aggregation process [36,37,47].
- Smart Grid Systems: Facilitating data privacy, confidentiality, authentication, and integrity during aggregation communication and function queries in fog computing-based smart grid systems [46,57].

– IoT: Enabling efficient aggregation of multidimensional data in IoT applications while preserving data contributor privacy [32,48,58].

Exploring the functionalities of Distributed Hash Tables (DHTs) in decentralized systems, particularly focusing on data storage and retrieval, necessitates a critical examination of the related privacy aspects. Although DHTs are not conventionally recognized as primary privacy-preserving solutions, they do offer scalable and fault-tolerant management of distributed data. However, this comes with inherent privacy challenges stemming from the systems' decentralized and distributed characteristics. Critical points include the potential for nodes to be manipulated and their vulnerability to assorted attacks.

Evaluating PPAgg and DHTs collectively, an essential inquiry emerges: is it plausible to effectively incorporate PPAgg with DHTs to mitigate some inherent privacy concerns? The objective would be to facilitate data aggregation which, in turn, guards individual contributions and ensures the protection of each data point from unauthorized access. Such scrutiny encompasses the implementation of privacy-preserving mechanisms within a DHT environment and the identification of challenges and opportunities therein. This balance between data aggregation and privacy preservation in decentralized networks is pivotal. Consequently, this investigation could lay the groundwork for ensuing research and modifications in the field, steering towards an augmented privacy framework in distributed systems.

Solutions for Privacy Preservation in DHTs. Distributed Hash Tables (DHTs), while widely recognized for their scalability and fault tolerance in managing distributed data, inherently present numerous privacy challenges due to their decentralized and distributed nature. Noteworthy here are the various initiatives aimed at securing both user and data privacy while maintaining the efficiency and functionality of DHTs.

– **Employing Oblivious Transfer for Query Privacy:** A strategy is introduced to protect the keys involved in DHT queries utilizing oblivious transfer (OT). The objective is to safeguard the queried key from intermediary peers involved in routing the queries to their respective destinations, without compromising spam resistance [5].
– **Adeona and Vanish:** These applications utilize DHTs to introduce innovative privacy-preserving solutions. Specifically, Adeona provides a solution for privacy-preserving laptop tracking, while Vanish aims to safeguard against retrospective assaults on cloud-archived data [23].
– **LaMRD:** LaMRD, a location-aware and privacy-preserving multi-layer resource discovery model for the Internet of Things (IoT), employs DHT technology to facilitate a peer-to-peer communication framework amongst fog nodes. This model ensures crucial security properties and reduced latency as compared to its centralized counterparts [34].
– **VPN0:** VPN0 utilizes a distributed architecture to address the privacy-trust issue inherent to a VPN's centralized authority. This system ensures a dVPN

node manages only pre-approved traffic, without revealing its whitelist or acknowledging the tunneled traffic through the employment of an attestation mechanism and a zero-knowledge proof, amongst other strategies [85].
– **Laribus:** Laribus, designed to detect local man-in-the-middle attacks against SSL/TLS, allows clients to validate the authenticity of a certificate without relying on a central notary service or the collaboration of website proprietors [51].

These studies underscore the feasibility and versatility of DHTs in maintaining data integrity and privacy, highlighting the potential pathways for further research and technological advancements in this domain.

3.3 Object Storage

The solution presented in Online Gaming [39] aims to create an adaptable infrastructure for large-scale peer-to-peer (P2P) online gaming platforms. All of the game's elements, including characters, are hosted on a game server, and players query the server to locate the items with which they interact. This is characteristic of traditional centralized methods. As the number of interacting users increases, the server's workload grows proportionally, making this centralized approach challenging to scale effectively.

To solve this issue, the suggested distributed system spreads user query load across a Distributed Hash Table (DHT) overlay made up of gaming servers. The DHT nodes in this arrangement reflect the game objects themselves. By using hash values as unique identifiers for game objects, these items are uniformly dispersed between game servers, guaranteeing balanced task allocation.

The Inter Planetary File System, frequently referred to as IPFS [7], is a peer-to-peer cloud storage platform. The technology is capable of distinguishing data objects throughout the whole network thanks to its special feature, content-addressing. IPFS does this by representing peers and data items in the Kademlia Distributed Hash Table (DHT).

In the case of small objects, typically around 1KB in size, they are directly stored on the DHT. The key-value pair for these objects consists of the object's hash value as the key and the object's content as the value. On the other hand, bigger items are held on peers, and the DHT is preserved with the reference to those peers. In this instance, the key is left unchanged, but the value is changed to the peer's address who owns the item.

Because IPFS objects are immutable, new versions of an item will always have a different hash value and different content. But in order to allow versioning, the new versions refer to their earlier iterations, resulting in a decentralized object graph. A graph with a single vertex is created by an item in its original form. As new versions of an object are created by making modifications directly to a vertex's children, each vertex serves as an object representation. It is possible to view and access all previous versions of an item by navigating through the object graph.

The accompanying object graph is completely decentralized and spread across several peers holding the object vertices of the network since each object is represented by a DHT node. Through its design, IPFS aims to be flexible enough for use in a wide range of scenarios, such as serving as a personal synchronization folder, a globally mounted file system, the base file system, or as a system for sharing encrypted data among distributed virtual machines.

A solution for time-sensitive access control in cloud storage for data objects is provided by time-sensitive object storage [89]. Under the concept of time-sensitive access control, sensitive data items are only available after a defined release time and are no longer accessible to new users after an expiry period.

To achieve this, the user encrypts their owned data objects using an asymmetric key scheme [38] and stores them on a cloud storage platform. The user additionally chunkifies the symmetric key and distributes it over a distinct DHT-based cloud storage made up of non-colluding nodes to provide time-based access control. This DHT-based cloud storage is distinct from the infrastructure in charge of maintaining customers' encrypted data objects and is solely used to store symmetric key chunks.

The user's symmetric key is encrypted and kept on each node of this DHT-based cloud storage system. These encrypted chunks that have expired are rejected by trustworthy DHT nodes. Even if the information is saved in the cloud, subsequent users won't be able to access it since the key will no longer be in the DHT nodes after the expiration time.

To distribute the symmetric keys of the data objects, a polynomial-based secret sharing algorithm [31] is utilized. The process divides the symmetric keys into parts and distributes them among the DHT nodes. Reconstructing a data object's symmetric key requires retrieving a data object's chunks from the DHT nodes and obtaining a secret timestamp that was supplied by a trustworthy timeserver at the time the relevant data item was published.

To achieve load balancing in Peer-to-Peer (P2P) video streaming systems, a two-tier DHT-based architecture is proposed [27]. Supernodes and user devices make up the first two levels of the architecture. The second tier, which consists of the users' devices, is composed of the supernodes, which are owned by the service provider and are managed by the same administrative domain.

This technique involves each supernode functioning as a service provider in the first tier maintaining a library of video files and streaming them as required to the user nodes in the second layer. The number of movies delivered concurrently on a supernode determines its load. The load is evenly distributed among the supernodes by establishing separate DHT overlays for each group of users watching the same video stream. The service provider's supernode, which initially broadcasts the content, is responsible for creating and managing the DHT overlay of users who are watching a particular stream.

Each stream is segmented into uniform-sized chunks, which are then distributed across the DHT overlay of devices belonging to users who are interested in the stream, aiding in load balancing. Instead of directly retrieving video segments from the content provider's supernode, users interact with their local

DHT overlay. By "local DHT", it is meant that the DHT overlay is specific to the video stream that the user is watching. This approach efficiently distributes the load of the supernode across the DHT overlays of all user devices.

3.4 DHTs for Enhanced Data Management in IoT Within Cloud Computing

Distributed Hash Tables (DHTs) offer a decentralized method for managing data and have become particularly important in the intersection of the Internet of Things (IoT) and cloud computing. Within the IoT domain, a peer-to-peer architecture utilizing DHTs ensures efficient management of elements like node IDs, capabilities, and sensor data [83]. This architecture extends its advantages to cloud services as well, where select peer nodes collaborate to share information, enhancing both scalability and fault-tolerance [96]. DHTs, with their structured approach, have also been adapted for larger cloud systems, exemplified by a cloud model that utilizes a hyperbolic tree structure for its DHT [81]. Further advancements in the field include the development of a digital twin management system for IoT devices using blockchain, highlighting the adaptability of DHT-based solutions [86].

Similarly, the DHT framework is being applied to improve traditional cloud structures, resulting in superior resource management and utilization [88]. The importance of distributing tasks evenly across various nodes in IoT-cloud interfaces has been addressed by introducing efficient load balancing methods, which employ algorithms designed for optimal distribution and resource management [1]. To ensure consistent data management in enterprise contexts, especially with the rise of industry 4.0 applications, DHTs have been incorporated into cloud storage systems. This approach facilitates a unified perspective on enterprise data, made possible by strategic mapping and rule-based systems [59]. These developments underline the essential role of DHTs in modern data management, especially within IoT, emphasizing the ongoing shift towards decentralized, efficient, and fault-resistant cloud computing solutions.

4 DHTs in Fog and Edge Computing

DHTs are turning into an increasingly significant technology in the context of fog and edge computing, considering that they provide an innovative method for developing and deploying applications in decentralized and distributed environments. As the need for real-time data processing and low-latency interactions grows, traditional cloud computing models confront scalability, reaction time, and bandwidth consumption limits. In contrast, fog and edge computing paradigms emphasize on the appropriate distribution of computing and storage resources among edge devices and end-users, which encourages the adoption of DHT-based applications in these environments.

DHTs, originally popularized in peer-to-peer systems, have found renewed significance in fog and edge computing due to their ability to facilitate efficient data storage, retrieval, and resource management in highly distributed

networks. DHTs, as opposed to centralized alternatives that rely on a single powerful data center, spread processing and storage power across several nodes positioned closer to data sources and end-users. This decentralized architecture enhances the system's responsiveness and throughput while simultaneously reducing network congestion and bandwidth demands, thereby contributing to a more efficient and high-performing system.

Routers, base stations, and IoT devices serve as important data sources and processing units in fog and edge computing. DHT-based applications take advantage of these edge devices' processing power, allowing data to be processed and analyzed locally rather of having to go to a distant data center. DHT-based applications increase system performance by relocating computations and data storage closer to the edge, resulting in faster response times, lower latency, and better overall system performance.

Fog computing also introduces the idea of fog nodes, which function as processing hubs between edge devices and cloud data centers. The aggregation, filtering, and preprocessing of data from edge devices is crucial, and these fog nodes, which have more processing and storage capability, are crucial in this process. DHTs provide effective data interchange and job allocation between edge devices and fog nodes by facilitating smooth communication and collaboration between them.

Numerous benefits result from the use of DHT-based applications in fog and edge computing. DHTs offer dynamic scalability, fault tolerance, and load balancing in highly dispersed systems in addition to lowering network traffic and improving resource efficiency. Due to their decentralized architecture, DHTs offer the robust advantage of maintaining continuous data accessibility and responding smoothly to challenges such as node failures, network partitions, and dynamic modifications in network topology.

The uses of DHTs are becoming more varied and significant as fog and edge computing continue to spread across several industries. DHT-based applications are useful in a wide range of use cases, including smart cities, industrial IoT, healthcare systems, and autonomous cars. These applications provide cutting-edge solutions that make use of the close proximity of edge resources to improve effectiveness, dependability, and user experience. They also enable real-time analytics, collaborative data sharing, content distribution, distributed storage systems, and more.

In Table 2, we compare DHT-based applications in fog and edge computing, detailing solution type, DHT, nodes, identifiers, utilization, domain, and references.

Ultimately, applications and infrastructures built on Distributed Hash Table (DHT) technology play a crucial role in fog and edge computing, improving resource utilization, data processing, and application delivery in decentralized and distributed environments. In fog and edge computing systems, DHTs are essential for delivering low-latency interactions, effective resource management, and scalable applications because to their capacity to spread storage and processing power closer to the edge.

Table 2. Comparison among DHT-based Applications in Fog/Edge Computing.

Solution	Type	DHT	Nodes	Identifiers	DHT Utilization	Domain	References
Two-tier Storage	Inf	Chord	Storage Nodes	Content Hash	Object Storage	Decentralized	[74]
Two-tier Overlay	Inf	Chord	Super Peers	Peer IDs	Routing Overlay	Decentralized	[15]
Edge Gaming and AR	App	General	Game Nodes	Asset ID	Hybrid	Distributed	[63]
Access Control Management	App	Chord	Control Nodes	User IDs	Object Storage	Decentralized	[64]
Service Discovery	App	General	Service Nodes	Service Name	Routing Overlay	Decentralized	[70]
Content Distribution	App	Chord	Distribution Nodes	Content IP	Routing Overlay	Distributed	[54]
Cache-based Storage	Inf	Kademlia	Cache Nodes	Content Hash	Object Storage	Decentralized	[12]
Producer-Consumer Buffer	App	General	Buffer Nodes	Content Tag	Object Storage	Decentralized	[75]
Collaborative Computation	App	General	Compute Nodes	Result Hash	Object Storage	Distributed	[73]
Intrusion Detection	App	Kademlia	Monitor Nodes	Packet IP	Object Storage	Decentralized	[71]
Smart Energy Grids	App	General	Energy Nodes	Energy ID	Object Storage	Decentralized	[3,18,65]
Intelligent Transportation Systems	App	General	Vehicle Nodes	Vehicle ID	Routing Overlay	Decentralized	[20,43]
Distributed Healthcare Systems	App	General	Device Nodes	Patient ID	Hybrid[a]	Decentralized	[19,44,66,68]
Resource Discovery (Unreliable Environments)	App	General	Device Nodes	Device URI	Routing Overlay	Distributed	[79]
Resource Discovery (Mobile Environments)	App	General	Mobile Nodes	Resource Tag	Routing Overlay	Decentralized	[24,80]

[a]Combination of Routing Overlay and Object Storage.

4.1 Data Management with DHTs in Fog and Edge Computing

The role of managing substantial data in edge and fog computing environments is crucial, given the dispersed nature of these settings. By focusing on processing and managing data close to the data source, fog and edge computing reduce latency and enhance real-time data interactions [69]. They effectively navigate the limitations related to scalability, response time, and bandwidth, which are often observed in conventional cloud systems.

Distributed Hash Tables (DHTs), initially developed for peer-to-peer systems, are vital for managing diverse data flow and storage in fog and edge computing. DHTs facilitate decentralized processing and storage across nodes, which are often closer to data sources like Internet of Things (IoT) devices and sensors. Their scalability allows them to adjust their size dynamically, ensuring balanced load distribution and fault tolerance, even when the network experiences changes. DHTs support scalable object storage and resource discovery overlays, providing a resilient, scalable, and fault-tolerant data management platform [12,70,71,79].

Implementing DHTs in fog and edge computing offers several key advantages. Firstly, by managing data close to its source, DHTs enhance responsiveness and throughput, reducing latency and promptly serving data requests. Secondly, they alleviate network congestion and moderate bandwidth usage by intelligently routing data and storing it strategically within nearby nodes. Lastly, they bolster overall system performance by improving data interchange and effectively distributing computational tasks [69,95].

DHTs provide consistent data access and adapt to network changes, maintaining operational continuity in fog and edge computing environments. Furthermore, critical components such as routers and fog nodes are integrated into the DHT infrastructure, facilitating a comprehensive and collective approach to data management.

In the healthcare sector, the management of vast amounts of critical data from IoT devices highlights the need for proficient data management in fog

computing [69]. A DHT-enabled, fault-tolerant data management scheme for healthcare IoT has been proposed, advocating a strategy to manage data efficiently and reliably, while ensuring low latency, minimal energy consumption, and cost-effectiveness. DHTs also provide a foundation for enhancing trust and security in data management across edge computing environments through blockchain technologies [95].

In conclusion, combining DHTs with fog and edge computing yields a resilient and scalable data management framework. As edge and fog computing become more integral to modern distributed systems, DHTs will increasingly guide the development of efficient and robust data management strategies in decentralized computing environments.

Utilizing DHTs for Data Management in IoT Within Fog and Edge Computing. Effective data management remains crucial in Internet of Things (IoT) systems, especially when operating within the complex environments shaped by fog and edge computing. Distributed Hash Tables (DHTs), with their fundamental capability to efficiently store and retrieve data across a network of nodes, become notably vital in ensuring smooth and scalable data management across a broad range of devices.

In this context, the utility of DHTs can be distilled into a few key points, showcasing their merit in addressing the unique challenges posed by IoT frameworks operating in fog and edge computing environments:

- **Consistent Data Availability:** DHTs ensure that data is always accessible, even if some nodes face downtime or failure, thus providing a steady data flow vital for the smooth operation of IoT systems.
- **Quick Data Access:** The structured nature of DHTs enables rapid and efficient data access, a necessity for maintaining the real-time responses that are fundamental in distributed IoT systems.
- **Flexible Scalability:** DHTs can easily adapt to varying loads, ensuring the data management layer continually meets the dynamic requirements of the network without sacrificing performance.
- **Even Load Distribution:** Through their inherent load-balancing capabilities, DHTs can evenly distribute loads across the network, preventing potential bottlenecks and ensuring smooth data and computational flow across the IoT network.
- **Data Security:** DHTs can be configured to bolster data security, ensuring that data interactions and transfers are both safe and reliable within the network.
- **Relevant Data Provision:** DHTs enable context-aware data management, ensuring that pertinent data is prioritized and made accessible as per the specific demands and operational contexts.
- **Efficient Network Management:** As IoT networks evolve, DHTs provide mechanisms to efficiently manage network topology, ensuring that changes in node statuses are addressed with minimal disruption to data consistency and access.

In summary, DHTs stand out as crucial enablers for robust and reliable data management and processing within IoT frameworks that are situated in fog and edge computing environments. They ensure not only reliable data availability, efficient data access, and scalable operations but also facilitate secure and context-aware data interactions and efficient network management, thereby enhancing the overall operational efficiency of IoT systems.

4.2 Use-Cases of DHT-Based Applications in Fog and Edge Computing

DHT-based applications in fog and edge computing exhibit remarkable versatility, finding utility across an expanding array of use cases within various domains. Beyond the previously mentioned examples, the following scenarios highlight the breadth of applications leveraging DHTs:

- Smart Energy Grids: DHTs enable efficient management and coordination of distributed energy resources in smart grids [3,18,65]. By utilizing DHT-based applications, energy producers, consumers, and grid operators can securely exchange information, optimize energy distribution, and enable demand response mechanisms. This empowers the grid with enhanced reliability, energy efficiency, and integration of renewable energy sources.
- Intelligent Transportation Systems: DHTs are essential to manage the enormous volumes of real-time data in these systems. In order to manage traffic flows effectively, cut down on travel times, and improve road safety, they enable dynamic routing, traffic prediction, and congestion management [20,43]. DHT-based systems can also facilitate cooperative communication between vehicles. Through this feature, vehicles can share information and collaborate in real-time, thereby enhancing situational awareness and potentially contributing to safer and more efficient roadways.
- Distributed Healthcare Systems: In distributed healthcare systems, DHTs provide a strong platform for organizing and securely sharing medical data as presented in [19,44,66,68]. Healthcare practitioners may securely access patient information, provide remote consultations, and enable dispersed care teams to make decisions together by utilizing DHT-based apps. The real-time monitoring and analysis of health data from wearable devices is also supported by DHTs, enabling prompt interventions and individualized medical treatment.
- Edge-based AI: DHTs are being used increasingly to implement edge-based artificial intelligence (AI) models. By using the computing capabilities of edge devices, DHT-based applications enable the deployment of AI models closer to the data source. As a result, real-time and context-aware AI inference at the edge is made possible, reducing the delay associated with relaying data to centralized cloud servers for processing. Applications span from object identification and video analytics in security systems to speech recognition and natural language processing in smart assistants.

– Edge Gaming and Augmented Reality: DHT-based applications support interactive gaming experiences and augmented reality (AR) applications at the network edge [63]. By distributing game assets, synchronization data, and AR overlays across edge devices, DHTs enable low-latency interactions, multiplayer capabilities, and seamless AR experiences. This enhances user engagement, reduces network dependency, and enables collaborative gaming and AR applications in various settings.

5 Open Problems and Challenges

– Typically, decisions regarding the performance of surveyed solutions are typically made based on the average value of the aggregated data. For example, when determining whether to be a regular peer or a super peer in the two-tier DHTs [15,80]. However, it's important to note that the aggregated average value may not always be the optimal choice, particularly in situations where the standard deviation is significantly low or high. This can lead to decisions that are affected by noise. For instance, in the case of two-tier architectures, it could result in the majority of nodes being selected as peers or super peers. Exploring alternative methods of aggregating performance metrics, such as using the median, for edge and fog computing solutions is a future research direction that demands further study.
– One of the major issues with DHT-based systems is the expense of stabilizing lookup tables when nodes fail or move between online and offline states. According to various study citations [54,79], DHT nodes (such as edge and fog servers) are configured with their DHT lookup tables by a central registry server, which regularly updates these tables based on the dynamic behavior of the nodes. However, it's crucial to research and includes [29], an efficient and decentralized DHT stabilizing technique, into the edge and fog computing ecosystems. This would aid in addressing the distributed lookup table stabilization's cost and scalability issues.
– Running distinct requests for each value inside the desired range is a standard way to handle range queries in DHT overlays. With this method, a range query with k values may be answered in a system with n DHT nodes with a message complexity of $\mathcal{O}(k \times logn)$. The range query may result in linear message complexity if k is greater than n. It is critical to create effective DHT-based approaches for answering range requests in order to improve system performance in edge and fog computing ecosystems. Further research should be conducted to explore and study these techniques, as they have the potential to significantly benefit system performance in scenarios involving edge and fog computing.

- Infrastructure-oriented resource optimization solutions, such as [97,98], often lean towards centralization by requiring almost the entire system behavior to be predefined. For example, they could rely on a specified time-based distribution of data items throughout the whole system. Furthermore, because they are not designed to respond to dynamic system changes such as abrupt changes in the distribution of data items, these solutions frequently have a static nature. There have been several attempts to advance toward completely decentralized, dynamic solutions, such as [29], which function based on the local perspective of individual nodes and react to system dynamics by predicting future behaviour based on current and previous states. A distributed system adds some mistakes when decisions are made entirely based on the local view of nodes. An intriguing research topic would be error margin minimization in decentralized and dynamic systems to calculate operational trade-off points. This investigation would help understand the limitations and trade-offs associated with decentralized decision-making in dynamic environments.
- Integrating DHTs in modern solutions such as autonomous vehicles, requires the use of technologies, such as the Cloud, which function as gateways or intermediaries between the DHT and other systems on the network. Cars for example can submit service requests through these gateways, which transmit signals on a regular basis to alert them of their presence. If the gateway vehicles are uniformly disseminated throughout the VANET (Vehicular Ad Hoc Network), the connection to the DHT is frequently strong. However, this may lead to conditions of congested channels. It is imperative to conduct research on the selection of entry-level vehicles in order to comprehend the trade-offs between performance and cost.
- Due to the dispersion of the VANET, numerous DHTs could exist outside of the vehicles, such as in the cloud or near edge layers. For service discovery, messages must be moved from one DHT to another on occasion. The service discovery messages can be sent and received without cutting the DHTs off altogether. The development of a suitable communication mechanism to keep DHTs connected at both the vehicle and cloud/edge levels is currently ongoing. Before DHT-assisted VANET applications can be effectively deployed, this issue has to be solved. Finding a reliable and efficient communication protocol that ensures seamless connectivity among the DHTs in different layers is crucial for the success of VANET applications [28].

6 Conclusions and Future Work

In conclusion, this comparative survey has explored six prominent DHT algorithms, namely Chord, Kademlia, CAN, Pastry, Symphony, and ZHT, which offer efficient and scalable solutions for creating peer-to-peer overlay networks.

These algorithms enable decentralized retrieval and storage of data and materials, eliminating the need for centralized network management. In contrast to first-generation peer-to-peer applications that rely on centralized indexes, these DHT algorithms facilitate content searching without centralized control. As the demand for P2P applications continues to grow, it becomes essential to develop systems capable of accommodating expansive network growth and dynamic network topologies. The innovative peer protocols discussed in this survey open up opportunities for applying peer-to-peer technologies beyond traditional file-sharing domains.

Future work should focus on advancing the capabilities of distributed hash tables and their integration with cloud and fog/edge computing paradigms. One promising avenue is to investigate techniques for efficient data storage and replication strategies in distributed hash tables operating within cloud and fog/edge environments. Additionally, future research directions can be directed towards developing robust and scalable algorithms for DHT stabilization, considering the dynamic nature of cloud and fog/edge networks. Furthermore, addressing the challenge of seamlessly connecting multiple DHTs across different layers, such as vehicles, cloud, and edge, remains an open problem. The practical deployment of DHT-assisted applications in VANETs and other distributed systems could be facilitated by the development of communication protocols that guarantee stable and efficient connections between DHTs in different environments. In the context of cloud and fog/edge computing, continued exploration and innovation in these areas have the potential to significantly improve the performance and capabilities of DHT-based systems.

Ultimately, the findings of our work are shown in Table 3, as presented in Appendix 6 where each DHT is presented along with key information. This table delineates key operational parameters and inner aspects of several DHT models, such as Chord and WiCHORD, among others, offering insights into their respective efficiencies and methodological variances concerning key management and node operations. Serving as a comprehensive reference, it thus facilitates an in-depth understanding and comparative assessment of the various DHT models, enabling astute decision-making in their application and deployment.

Appendix

In the following section, all DHT-based algorithms and protocols are summarized. The table contains the name of each DHT solution, the size of each routing table, the lookup and delete key operations, the join and leave operations, as well as some comments per each method. This table serves as a concluding remark for our survey, as it presents all information gathered in one easy-to-read place.

Table 3. A comparison and overview of several DHTs.

DHTs	Size of a routing table	Lookup, Insert, Delete Key	Join/Leave	Comments
Chord [77]	$\mathcal{O}(\log n)$, maximum $m+2s$	$\mathcal{O}(\log n)$, avg $(\frac{1}{2}\log n)$	$< \mathcal{O}(\log n)$ total messages	s is neighbors in 1 direction , m = keysize in bits
WiCHORD [6]	$\mathcal{O}(m)$	Lookup: $\mathcal{O}(\log n)$, Insert/Delete:-	$\mathcal{O}(\log n)$	m = keysize in bits. Suitable for Cloud and Edge applications in IoT
ZHT [45]	$\mathcal{O}(n)$	$O(1)$	$\mathcal{O}(n)$	Expects a very low churn
CAN [62]	$\Omega(2d)$	$\mathcal{O}(n^{\frac{1}{d}})$, average $\frac{d}{4} \cdot n^{\frac{1}{d}}$	Affects $\mathcal{O}(d)$ nodes	d is the number of dimensions
Koorde [33]	$\mathcal{O}(\log_{2^{k+1}} n)$	$\mathcal{O}(\log_{2^{k+1}} n)$	$\mathcal{O}(\log_{2^{k+1}} n)$	Requires $\mathcal{O}(k \log n)$ state at each node, where k is the number of bits in a node identifier
Mainline DHT	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	Kademlia-based
Pastry [67]	$\mathcal{O}(\log_\beta n)$	$\mathcal{O}(\lceil \log_{2^\beta} \rceil)$	$\mathcal{O}(\log_\beta n)$	NodeIDs are base β numbers
Kademlia [50]	$\mathcal{O}(\log n)$, maximum $m \cdot k$	$(\lceil \log n \rceil) + c$	$\mathcal{O}(\log(n))$	This is without considering optimization
Riak [42]	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	Based on Amazon's Dynamo, uses consistent hashing
Symphony [49]	$2k + 2$	average $\mathcal{O}(\frac{1}{k}\log^2 n)$	$\mathcal{O}(\log^2 n)$ messages, constant < 1	$k \geq 1$, fingers are chosen at random
VHash	$\Omega(3d+1) + \mathcal{O}((3d+1)^2)$	$\mathcal{O}(\sqrt[d]{n})$ hops	$3d + 1$	hops are based least latency, approximates regions
Apache Cassandra [10]	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	Uses consistent hashing, based on Chord
Voldemort [9]	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	Dynamo-like, used in LinkedIn and other services
MapChain [87]	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	Hybrid of Chord and Kademlia, high resilience
TomP2P [55]	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	Similar to Kademlia, uses consistent hashing
LPRS-Chord [93]	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	Latency-aware, efficient routing, based on Chord
CoralCDN [22]	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	Fast object location and retrieval, used with BitTorrent
Kelips [26]	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	Soft-state membership protocol, balances overhead and lookup performance
Bamboo	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	Efficient and robust routing under high churn rates
OneHop	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	Low-latency lookup, increased maintenance overhead
Plaxton-based DHTs, Pastry [67], Tapestry [94]	$\mathcal{O}(\log_\beta n)$	$\mathcal{O}(\log_\beta n)$	$\mathcal{O}(\log_\beta n)$	NodeIDs are base β numbers

References

1. Abed, M.M., Younis, M.F.: Developing load balancing for IoT-cloud computing based on advanced firefly and weighted round robin algorithms. Baghdad Sci. J. **16**(1), 130–139 (2019)
2. Abid, S.A., Othman, M., Shah, N.: A survey on DHT-based routing for large-scale mobile ad hoc networks. ACM Comput. Surv. (CSUR) **47**(2), 1–46 (2014)
3. Alladi, T., Chamola, V., Rodrigues, J.J.P.C., Kozlov, S.A.: Blockchain in smart grids: a review on different use cases. Sensors **19**(22) (2019). https://doi.org/10.3390/s19224862. https://www.mdpi.com/1424-8220/19/22/4862
4. Androutsellis-Theotokis, S., Spinellis, D.: A survey of peer-to-peer content distribution technologies. ACM Comput. Surv. (CSUR) **36**(4), 335–371 (2004)
5. Backes, M., Goldberg, I., Kate, A., Toft, T.: Adding query privacy to robust DHTs. In: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, pp. 30–31 (2012)
6. Balatsouras, C.P., Karras, A., Karras, C., Tsolis, D., Sioutas, S.: WiCHORD: a chord protocol application on P2P LoRa wireless sensor networks. In: 2022 13th International Conference on Information, Intelligence, Systems & Applications (IISA), pp. 1–8 (2022). https://doi.org/10.1109/IISA56318.2022.9904339
7. Benet, J.: IPFS - content addressed, versioned, P2P file system. CoRR abs/1407.3561 (2014). arxiv.org/abs/1407.3561
8. Berdik, D., Otoum, S., Schmidt, N., Porter, D., Jararweh, Y.: A survey on blockchain for information systems management and security. Inf. Process. Manag. **58**(1), 102397 (2021)
9. Bonvin, N., Papaioannou, T.G., Aberer, K.: A self-organized, fault-tolerant and scalable replication scheme for cloud storage. In: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, pp. 205–216. Association for Computing Machinery, New York, NY, USA (2010). https://doi.org/10.1145/1807128.1807162
10. Cassandra, A.: Apache Cassandra. **13** (2014). http://www.planetcassandraorg/what-is-apache-cassandra
11. Condie, T., Kacholia, V., Sank, S., Hellerstein, J.M., Maniatis, P.: Induced churn as shelter from routing-table poisoning. In: NDSS (2006)
12. Confais, B., Lebre, A., Parrein, B.: An object store service for a fog/edge computing infrastructure based on IPFS and a scale-out NAS. In: 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), pp. 41–50. IEEE (2017)
13. Cooper, C., Franklin, D., Ros, M., Safaei, F., Abolhasan, M.: A comparative survey of VANET clustering techniques. IEEE Commun. Surv. Tutorials **19**(1), 657–681 (2016)
14. Dabek, F., Zhao, B., Druschel, P., Kubiatowicz, J., Stoica, I.: Towards a common API for structured peer-to-peer overlays. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735, pp. 33–44. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45172-3_3
15. D'Angelo, M., Caporuscio, M.: SA-Chord: a self-adaptive P2P overlay network. In: 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W), pp. 118–123. IEEE (2018)
16. De Salve, A., Mori, P., Ricci, L.: A survey on privacy in decentralized online social networks. Comput. Sci. Rev. **27**, 154–176 (2018)
17. DeCandia, G., et al.: Dynamo: Amazon's highly available key-value store. In: ACM SIGOPS Operating Systems Review, vol. 41, pp. 205–220. ACM (2007)

18. Demertzis, F.F., Karopoulos, G., Xenakis, C., Colarieti, A.: Self-organised key management for the smart grid. In: Papavassiliou, S., Ruehrup, S. (eds.) ADHOC-NOW 2015. LNCS, vol. 9143, pp. 303–316. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19662-6_21
19. Egala, B.S., Pradhan, A.K., Dey, P., Badarla, V., Mohanty, S.P.: Fortified-chain 2.0: intelligent blockchain for decentralized smart healthcare system. IEEE Internet Things J., 1 (2023). https://doi.org/10.1109/JIOT.2023.3247452
20. El-Salakawy, G., Abu El-Kheir, M.: Blockchain-based data management in vehicular networks. In: 2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES), pp. 146–151 (2020). https://doi.org/10.1109/NILES50944.2020.9257890
21. Fersi, G., Louati, W., Ben Jemaa, M.: Distributed hash table-based routing and data management in wireless sensor networks: a survey. Wireless Netw. **19**, 219–236 (2013)
22. Freedman, M.J., Freudenthal, E., Mazieres, D.: Democratizing content publication with coral. In: NSDI, vol. 4, p. 18 (2004)
23. Geambasu, R., Falkner, J., Gardner, P., Kohno, T., Krishnamurthy, A., Levy, H.M.: Experiences building security applications on DHTs. Technical report, UW-CSE-09-09-01 (2009)
24. Gedeon, J., Meurisch, C., Bhat, D., Stein, M., Wang, L., Mühlhäuser, M.: Router-based brokering for surrogate discovery in edge computing. In: 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), pp. 145–150. IEEE (2017)
25. Ghaleb, S.M., Subramaniam, S., Zukarnain, Z.A., Muhammed, A.: Mobility management for IoT: a survey. EURASIP J. Wirel. Commun. Netw. **2016**, 1–25 (2016)
26. Gupta, I., Birman, K., Linga, P., Demers, A., van Renesse, R.: Kelips: building an efficient and stable P2P DHT through increased memory and background overhead. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735, pp. 160–169. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45172-3_15
27. Gupta, R.K., Hada, R., Sudhir, S.: 2-tiered cloud based content delivery network architecture: an efficient load balancing approach for video streaming. In: 2017 International Conference on Signal Processing and Communication (ICSPC), pp. 431–435 (2017). https://doi.org/10.1109/CSPC.2017.8305885
28. Hassanzadeh-Nazarabadi, Y., Boshrooyeh, S.T., Otoum, S., Ucar, S., Özkasap, Ö.: DHT-based communications survey: architectures and use cases. CoRR abs/2109.10787 (2021). arxiv.org/abs/2109.10787
29. Hassanzadeh-Nazarabadi, Y., Küpçü, A., Özkasap, Ö.: Interlaced: fully decentralized churn stabilization for skip graph-based DHTs. J. Parallel Distrib. Comput. **149**, 13–28 (2021)
30. Hildrum, K., Kubiatowicz, J.D., Rao, S., Zhao, B.Y.: Distributed object location in a dynamic network. Theory Comput. Syst. **37**(3), 405–440 (2004)
31. Ito, M., Saito, A., Nishizeki, T.: Secret sharing scheme realizing general access structure. Electron. Commun. Japan (Part III: Fundam. Electron. Sci.) **72**(9), 56–64 (1989)
32. Jastaniah, K., Zhang, N., Mustafa, M.A.: Efficient privacy-friendly and flexible IoT data aggregation with user-centric access control. arXiv preprint arXiv:2203.00465 (2022)
33. Kaashoek, M.F., Karger, D.R.: Koorde: a simple degree-optimal distributed hash table. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735, pp. 98–107. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45172-3_9

34. Kamel, M.B., Ligeti, P., Reich, C.: Lamred: location-aware and decentralized multi-layer resource discovery for IoT. Acta Cybernet. **25**(2), 319–349 (2021)
35. Karagiannis, V.: Compute node communication in the fog: survey and research challenges. In: Proceedings of the Workshop on Fog Computing and the IoT, pp. 36–40 (2019)
36. Karras, A., Karras, C., Giotopoulos, K.C., Tsolis, D., Oikonomou, K., Sioutas, S.: Peer to peer federated learning: towards decentralized machine learning on edge devices. In: 2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM), pp. 1–9 (2022). https://doi.org/10.1109/SEEDA-CECNSM57760.2022.9932980
37. Karras, A., Karras, C., Giotopoulos, K.C., Tsolis, D., Oikonomou, K., Sioutas, S.: Federated edge intelligence and edge caching mechanisms. Information **14**(7) (2023). https://doi.org/10.3390/info14070414. https://www.mdpi.com/2078-2489/14/7/414
38. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. CRC Press (2020)
39. Kavalionak, H., Carlini, E., Ricci, L., Montresor, A., Coppola, M.: Integrating peer-to-peer and cloud computing for massively multiuser online games. Peer-to-Peer Network. Appl. **8**, 301–319 (2015)
40. Kleinberg, J.: The small-world phenomenon: an algorithmic perspective. In: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, pp. 163–170. ACM (2000)
41. Kleinberg, J.M.: Navigation in a small world. Nature **406**(6798), 845 (2000)
42. Klophaus, R.: Riak core: building distributed applications without shared state. In: ACM SIGPLAN Commercial Users of Functional Programming, CUFP 2010. Association for Computing Machinery, New York, NY, USA (2010). https://doi.org/10.1145/1900160.1900176
43. Kuhn, E., Mordinyi, R., Goiss, H.D., Bessler, S., Tomic, S.: A P2P network of space containers for efficient management of spatial-temporal data in intelligent transportation scenarios. In: 2009 Eighth International Symposium on Parallel and Distributed Computing, pp. 218–225 (2009). https://doi.org/10.1109/ISPDC.2009.27
44. Kumar, R., Marchang, N., Tripathi, R.: Distributed off-chain storage of patient diagnostic reports in healthcare system using IPFS and blockchain. In: 2020 International Conference on COMmunication Systems & NETworkS (COMSNETS), pp. 1–5 (2020). https://doi.org/10.1109/COMSNETS48256.2020.9027313
45. Li, T., et al. ZHT: a light-weight reliable persistent dynamic scalable zero-hop distributed hash table. In: 2013 IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS), pp. 775–787. IEEE (2013)
46. Liu, J.N., Weng, J., Yang, A., Chen, Y., Lin, X.: Enabling efficient and privacy-preserving aggregation communication and function query for fog computing-based smart grid. IEEE Trans. Smart Grid **11**(1), 247–257 (2019)
47. Liu, Z., Guo, J., Yang, W., Fan, J., Lam, K.Y., Zhao, J.: Privacy-preserving aggregation in federated learning: a survey. IEEE Trans. Big Data (2022)
48. Loukil, F., Ghedira-Guegan, C., Boukadi, K., Benharkat, A.N.: Privacy-preserving IoT data aggregation based on blockchain and homomorphic encryption. Sensors **21**(7), 2452 (2021)
49. Manku, G.S., Bawa, M., Raghavan, P., et al.: Symphony: distributed hashing in a small world. In: USENIX Symposium on Internet Technologies and Systems, p. 10 (2003)

50. Maymounkov, P., Mazières, D.: Kademlia: a peer-to-peer information system based on the XOR metric. In: Druschel, P., Kaashoek, F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 53–65. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45748-8_5
51. Micheloni, A., Fuchs, K.P., Herrmann, D., Federrath, H.: Laribus: privacy-preserving detection of fake SSL certificates with a social P2P notary network. In: 2013 International Conference on Availability, Reliability and Security, pp. 1–10. IEEE (2013)
52. Milgram, S.: The small world problem. Psychol. Today **2**(1), 60–67 (1967)
53. Nakagawa, I., et al.: DHT extension of m-cloud - scalable and distributed privacy preserving statistical computation on public cloud. In: 2015 IEEE 39th Annual Computer Software and Applications Conference, vol. 3, pp. 682–683 (2015). https://doi.org/10.1109/COMPSAC.2015.94
54. Nakayama, T., Asaka, T.: Peer-to-peer bidirectional streaming using mobile edge computing. In: 2017 Fifth International Symposium on Computing and Networking (CANDAR), pp. 263–266. IEEE (2017)
55. Name, A.: TomP2P, a P2P-based key-value pair storage library (2012). https://tomp2p.net/. Accessed: insert-the-date-you-accessed-the-website
56. Napoli, A., et al.: Enabling router bypass and saving cost using point-to-multipoint transceivers for traffic aggregation. In: Optical Fiber Communication Conference, pp. W3F–5. Optica Publishing Group (2022)
57. Orda, L.D., Jensen, T.V., Gehrke, O., Bindner, H.W.: Efficient routing for overlay networks in a smart grid context. In: SMARTGREENS, pp. 131–136 (2019)
58. Peng, C., Luo, M., Wang, H., Khan, M.K., He, D.: An efficient privacy-preserving aggregation scheme for multidimensional data in IoT. IEEE Internet Things J. **9**(1), 589–600 (2021)
59. Petrasch, R., Hentschke, R.: Cloud storage hub: data management for IoT and industry 4.0 applications: towards a consistent enterprise information management system. In: 2016 Management and Innovation Technology International Conference (MITicon), pp. MIT-108–MIT-111 (2016). https://doi.org/10.1109/MITICON.2016.8025236
60. Plaxton, C.G., Rajaraman, R., Richa, A.W.: Accessing nearby copies of replicated objects in a distributed environment. In: Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA 1997, pp. 311–320. ACM, New York, NY, USA (1997). https://doi.org/10.1145/258492.258523
61. Plaxton, C.G., Rajaraman, R., Richa, A.W.: Accessing nearby copies of replicated objects in a distributed environment. Theory Comput. Syst. **32**(3), 241–280 (1999)
62. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network (2001)
63. Ren, P., Liu, L., Qiao, X., Chen, J.: Distributed edge system orchestration for web-based mobile augmented reality services. IEEE Trans. Serv. Comput. **16**(3), 1778–1792 (2023). https://doi.org/10.1109/TSC.2022.3190375
64. Riabi, I., Saidane, L.A., Ayed, H.K.B.: A proposal of a distributed access control over fog computing: the its use case. In: 2017 International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN), pp. 1–7. IEEE (2017)
65. Rodrigues, A.S., Rizzetti, T.A., Canha, L.N., Milbradt, R.G., Appel, S.F., Duarte, Y.S.: Implementing a distributed firewall using a DHT network applied to smart grids. In: 2016 51st International Universities Power Engineering Conference (UPEC), pp. 1–5 (2016). https://doi.org/10.1109/UPEC.2016.8113985

66. Roehrs, A., da Costa, C.A., da Rosa Righi, R.: OmniPHR: a distributed architecture model to integrate personal health records. J. Biomed. Inform. **71**, 70–81 (2017)
67. Rowstron, A., Druschel, P.: Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45518-3_18
68. Saberi, M.A., Adda, M., Mcheick, H.: Break-glass conceptual model for distributed EHR management system based on blockchain, IPFS and ABAC. Procedia Comput. Sci. **198**, 185–192 (2022). https://doi.org/10.1016/j.procs.2021.12.227. https://www.sciencedirect.com/science/article/pii/S1877050921024662. 12th International Conference on Emerging Ubiquitous Systems and Pervasive Networks/11th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare
69. Saeed, W., Ahmad, Z., Jehangiri, A.I., Mohamed, N., Umar, A.I., Ahmad, J.: A fault tolerant data management scheme for healthcare internet of things in fog computing. KSII Trans. Internet Inf. Syst. (TIIS) **15**(1), 35–57 (2021)
70. Santos, J., Wauters, T., Volckaert, B., De Turck, F.: Towards dynamic fog resource provisioning for smart city applications. In: 2018 14th International Conference on Network and Service Management (CNSM), pp. 290–294. IEEE (2018)
71. Sharma, R., Chan, C.A., Leckie, C.: Evaluation of centralised vs distributed collaborative intrusion detection systems in multi-access edge computing. In: 2020 IFIP Networking Conference (Networking), pp. 343–351. IEEE (2020)
72. Siddula, M., Li, L., Li, Y.: An empirical study on the privacy preservation of online social networks. IEEE Access **6**, 19912–19922 (2018)
73. Simić, M., Stojkov, M., Sladić, G., Milosavljević, B.: Edge computing system for large-scale distributed sensing systems. In: ICIST, pp. 36–39 (2018)
74. Sonbol, K., Özkasap, Ö., Al-Oqily, I., Aloqaily, M.: EdgeKV: decentralized, scalable, and consistent storage for the edge. J. Parallel Distrib. Comput. **144**, 28–40 (2020)
75. Song, J., Gu, T., Ge, Y., Mohapatra, P.: Smart contract-based computing resources trading in edge computing. In: 2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications, pp. 1–7. IEEE (2020)
76. Stevens, M.M.J., et al.: Attacks on hash functions and applications. Mathematical Institute, Faculty of Science, Leiden University (2012)
77. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup service for internet applications. SIGCOMM Comput. Commun. Rev. **31**, 149–160 (2001). https://doi.org/10.1145/964723.383071
78. Tanenbaum, A.S., Steen, M.: Distributed systems [sl] (2007)
79. Tanganelli, G., Vallati, C., Mingozzi, E.: Edge-centric distributed discovery and access in the internet of things. IEEE Internet Things J. **5**(1), 425–438 (2017)
80. Tanganelli, G., Vallati, C., Mingozzi, E.: A fog-based distributed look-up service for intelligent transportation systems. In: 2017 IEEE 18th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), pp. 1–6. IEEE (2017)
81. Tiendrebeogo, T.: A cloud computing system using virtual hyperbolic coordinates for services distribution. In: Zhang, Y., Peng, L., Youn, C.-H. (eds.) CloudComp 2015. LNICST, vol. 167, pp. 269–279. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-38904-2_28

82. Toosi, A.N., Calheiros, R.N., Buyya, R.: Interconnected cloud computing environments: challenges, taxonomy, and survey. ACM Comput. Surv. (CSUR) **47**(1), 1–47 (2014)
83. Tracey, D., Sreenan, C.: Using a DHT in a peer to peer architecture for the internet of things. In: 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), pp. 560–565 (2019). https://doi.org/10.1109/WF-IoT.2019.8767261
84. Varshney, P., Simmhan, Y.: Demystifying fog computing: characterizing architectures, applications and abstractions. In: 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), pp. 115–124. IEEE (2017)
85. Varvello, M., Azurmendi, I.Q., Nappa, A., Papadopoulos, P., Pestana, G., Livshits, B.: VPN0: a privacy-preserving decentralized virtual private network. arXiv preprint arXiv:1910.00159 (2019)
86. Wang, C., Cai, Z., Li, Y.: Sustainable blockchain-based digital twin management architecture for IoT devices. IEEE Internet Things J. **10**(8), 6535–6548 (2023). https://doi.org/10.1109/JIOT.2022.3153653
87. Wu, T., Yeoh, P.L., Jourjon, G., Thilakarathna, K.: MapChain: a DHT-based dual-blockchain data structure for large-scale IoT systems. In: 2021 IEEE 7th World Forum on Internet of Things (WF-IoT), pp. 177–182 (2021). https://doi.org/10.1109/WF-IoT51360.2021.9595910
88. Xie, X.L., Wang, Q., Wang, P.: Design of smart container cloud based on DHT. In: 2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), pp. 2971–2975 (2017). https://doi.org/10.1109/FSKD.2017.8393255
89. Xiong, J., Li, F., Ma, J., Liu, X., Yao, Z., Chen, P.S.: A full lifecycle privacy protection scheme for sensitive data in cloud computing. Peer-to-Peer Network. Appl. **8**, 1025–1037 (2015)
90. Yang, Y., Zhang, L., Zhao, Y., Choo, K.K.R., Zhang, Y.: Privacy-preserving aggregation-authentication scheme for safety warning system in fog-cloud based VANET. IEEE Trans. Inf. Forensics Secur. **17**, 317–331 (2022)
91. Yousefpour, A., et al.: All one needs to know about fog computing and related edge computing paradigms: a complete survey. J. Syst. Architect. **98**, 289–330 (2019)
92. Zave, P.: Using lightweight modeling to understand chord. ACM SIGCOMM Comput. Commun. Rev. **42**(2), 49–57 (2012)
93. Zhang, H., Goel, A., Govindan, R.: Incrementally improving lookup latency in distributed hash table systems. SIGMETRICS Perform. Eval. Rev. **31**(1), 114–125 (2003). https://doi.org/10.1145/885651.781042
94. Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.D.: Tapestry: a resilient global-scale overlay for service deployment. IEEE J. Sel. Areas Commun. **22**(1), 41–53 (2004)
95. Zhaofeng, M., Xiaochang, W., Jain, D.K., Khan, H., Hongmin, G., Zhen, W.: A blockchain-based trusted data management scheme in edge computing. IEEE Trans. Industr. Inf. **16**(3), 2013–2021 (2020). https://doi.org/10.1109/TII.2019.2933482
96. Zhelev, R., Georgiev, V.: A DHT-based scalable and fault-tolerant cloud information service. UBICOMM **2011**, 67 (2011)
97. Zhou, J., Fan, J., Jia, J.: A cost-efficient resource provisioning algorithm for DHT-based cloud storage systems. Concurrency Comput. Pract. Exp. **28**(18), 4485–4506 (2016)

98. Zhou, J., He, W.: A novel resource provisioning model for DHT-based cloud storage systems. In: Hsu, C.-H., Shi, X., Salapura, V. (eds.) NPC 2014. LNCS, vol. 8707, pp. 257–268. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44917-2_22
99. Zuo, X., Iamnitchi, A.: A survey of socially aware peer-to-peer systems. ACM Comput. Surv. (CSUR) **49**(1), 1–28 (2016)